UNIVERSIDAD CAMILO JOSÉ CELA
FACULTAD DE EDUCACIÓN

Tesis Doctoral

Scratch como herramienta para la enseñanza de la programación en la Educación Primaria.

Análisis de usabilidad en la escuela pública de la Comunidad de Madrid

Autor: David Alonso Urbano

Directores:
Dr. Max Walter Römer Pieretti
Dr. Rafael Conde Melguizo

Madrid, 2017

UNIVERSIDAD CAMILO JOSÉ CELA
FACULTAD DE EDUCACIÓN

Tesis Doctoral

Scratch como herramienta para la enseñanza de la programación en la Educación Primaria.

Análisis de usabilidad en la escuela pública de la Comunidad de Madrid

Autor: David Alonso Urbano

Directores:
Dr. Max Walter Römer Pieretti
Dr. Rafael Conde Melguizo

Madrid, 2017

—

A Patricia,
que ilumina mis días

—

Agradecimientos

A mis directores, Max y Rafael, por ser mis guías en este proceso.

Gracias a Patricia, a mis padres Julio e Isabel, y a mi hermano Jaime, por su cariño, y su apoyo incondicional.

Gracias a Pablo, Jorge, Sagrario, Rubén, Óscar, Sergio, Noelia, Miguel, Silvia y Andrés. Sin vuestra ayuda, esta tesis no habría sido posible.

Gracias a Ignacio y a Jose, socios, amigos y compañeros en mil batallas.

Y por supuesto, muchas gracias a todos los participantes del estudio.

ÍNDICE GENERAL

PARTE INTRODUCTORIA

PARTE TEÓRICA

PARTE EMPÍRICA

ÍNDICE
DE TABLAS

ÍNDICE
DE FIGURAS

ÍNDICE DE ABREVIATURAS

ACM	*Association for Computing Machinery*
ADC	Sumar con reserva o "*addwithcarry*"
ADD	Sumar sin reserva
AMF	Análisis de medios y fines
ANSI	Instituto Nacional Estadounidense de Estándares
API	*Application Programming Interface*
ASA	Agencia de Estándares Estadounidense
ASCII	*American Standard Code for Information Interchange*
BUP	Bachillerato Unificado Polivalente
CASE	*Computer Aided Software Engineering*
CM	Comunidad de Madrid
COU	Curso de Orientación Universitaria
CPU	Unidad Central de Procesos
DeSeCo	Definición y Selección de Competencias clave
EATP	Enseñanzas y Actividades Técnicos Profesionales
EGB	Educación General Básica
ESO	Educación Secundaria Obligatoria
ESP	*Empirical Studies of Programmers*
GUI	*Graphical User Interface*
IOS	i*Phone Operative Sistem*
FAQ	*Frequently Asked Questions*
FORTRAN	*The IBM Mathematical Formula Translating S*ystem
HCI	*Human Computer Interface*
ISO	*Organization for Standardization*
LGE	Ley General de Educación
LOCE	Ley Orgánica de Calidad en la Educación
LOE	Ley Orgánica de Educación
LOGSE	Ley Orgánica General del Sistema Educativo

LOMCE	Ley Orgánica de la Mejora en la Calidad Educativa
MOOC	*Massive Open Online Course*
PC	*Personal Computer*
PBL	*Project-based learning*
PC	Personal Computer
PISA	*Programme for International Student Assessment*
PROLOG	*Programmation Logique*
RAE	Real Academia Española
RPC	*Remote Procedure Calls*
RPG	*Role Play Game*
SIGCHI	*Special Interest Group on Computer-Human Interaction*
SDK	*Software Development Kit*
SLO	El instituto nacional para el desarrollo curricular
SQL	*Standard Query Language*
TIC	Tecnología de la Información y la Comunicación
TICD	Tratamiento de la Información y Competencia Digital
TPI	Teoría del Procesamiento de la Información

—

PARTE INTRODUCTORIA

—

1. Introducción

Los grandes cambios científicos y tecnológicos que se han producido en los últimos años han llevado a nuevas formas de producción, distribución, administración y relación que han transformado nuestra realidad. De una sociedad basada en la industria, hemos pasado a una basada en los servicios donde la información juega un papel fundamental (Castells, 2006). Estamos inmersos en la sociedad de la información donde la comunicación se puede establecer en cualquier lugar y en cualquier momento, gracias a la existencia de la tecnología móvil, tanto de manera sincrónica (chat, videollamadas, etc.) como asincrónica (foros, email, etc.).

La escuela no es ajena a estos cambios. Por ello, diversos organismos internacionales (Ananiadou y Claro, 2009) se han ocupado de ofrecer recomendaciones, y realizar informes para que los distintos países tengan en cuenta la situación actual, y adapten sus políticas en este sentido.

Haciéndose eco de estas recomendaciones, diferentes agentes de la comunidad educativa han solicitado una respuesta concreta para ofrecer a los estudiantes formación en el ámbito tecnológico, con el objetivo de prepararles para afrontar las demandas que el mercado laboral y la sociedad del conocimiento impone. Por un lado, se aboga por que estas temáticas se aborden tempranamente, y que los estudiantes, desde la infancia, aprendan a integrar la tecnología en su día a día. Por otro, se busca que a estas disciplinas se les confiera el mismo rango de importancia que a cualquier otra. El caso que nos ocupa, de las nuevas competencias que se les va a exigir a las generaciones del futuro, surge, de forma destacada, el manejo de los lenguajes de programación, y la adquisición del denominado pensamiento computacional (Wing, 2006), que viene a añadirse a la ya conocida Competencia Digital.

A raíz de diversas iniciativas de colectivos académicos y científicos, muchos países han introducido, o están en proceso de introducir, la programación en el currículum de las enseñanzas obligatorias (Balanskat y Engelhardt, 2015). La programación parece haber pasado de ser una disciplina especializada con un cuerpo de conocimiento propio que conlleva un aprendizaje largo y complejo, a percibirse como una herramienta o habilidad fácilmente transmisible y de carácter transversal. Esto es debido en parte a que su contenido científico y técnico se ha simplificado en favor de la usabilidad, con la aparición de herramientas cuyo diseño visual hace más asequible los conceptos relacionados con la programación.

Entre estas herramientas, una de las escogidas de forma preferente por las instituciones para la enseñanza de la programación es Scratch. Un ejemplo de esto

lo encontramos en países como Estados Unidos (Tucker et al., 2011) y Reino Unido (UK Department of Education, 2013), y en el ámbito nacional, señalamos el Decreto 89/2014 (Decreto Comunidad de Madrid, 2015, p. 90), donde se menciona de forma explícita Scratch como el instrumento a utilizar en la asignatura de libre configuración llamada “Tecnología y recursos digitales para la mejora del aprendizaje”.

Scratch es un proyecto del MIT (*Massachusetts Institute of Technology*), concretamente del grupo de investigación Lifelong Kindergarten del laboratorio de medios del MIT. Está diseñado para enseñar a programar mediante la creación de juegos e historias interactivas, en la horquilla de edades entre los 8 y los 16 años. Se trata de la herramienta de aprendizaje de programación más extendida en el mundo, utilizándose en más de 150 países (MIT, 2013).

El diseño de la sintaxis visual de Scratch está basada en los juegos de construcción, como Lego. Este diseño permite presentar, de una forma asequible, la lógica de la programación. El usuario construye programas agrupando bloques gráficos como si fueran piezas de un rompecabezas. Estos bloques representan las estructuras de programación, y las acciones que se pueden realizar dentro del programa (mover un objeto, reproducir un sonido, etc.). Cada bloque tiene una forma diferente, y hay ciertas piezas que se pueden unir entre ellas, y otras no. Encajando las piezas donde la unión es posible se construyen estructuras de programación sintácticamente correctas.

El sistema de bloques permite programar animaciones y elementos interactivos sin necesidad de escribir código textual. Esto se consigue mediante una interfaz gráfica con un diseño HCI (*Human Computer Interaction*) clásico, mediante pantalla, teclado y ratón, y manipulación de objetos gráficos. Este diseño facilita que cualquier individuo que sepa manejar un Sistema Operativo con interfaz gráfica (Windows, Mac OS, GNU-Linux Ubuntu, etc.) sea capaz de comprender la lógica del software y empezar a programar.

Los proyectos creados con Scratch se pueden compartir a través de la plataforma web que el MIT ha creado para tal efecto, dando la posibilidad al resto de usuarios tanto de probar el resultado del programa, como de acceder al código fuente (en este caso, los bloques) que lo han hecho posible. Esto, junto con la gran cantidad de documentación disponible, ha propiciado que exista una enorme comunidad en torno a Scratch, y que sea tan popular.

1.1. Pertinencia de esta tesis

Aprender a programar se presenta a priori como una tarea difícil (Caspersen y Bennedsen, 2007). La forma de abordar la enseñanza de la programación es un tema que lleva preocupando desde hace tiempo a la comunidad científica y educativa. Por este motivo se han realizado un número significativo de estudios al respecto (Cherubini, Venolia, DeLine y Ko, 2007; Myers, Burnett, Wiedenbeck y Ko, 2007; Norcio, 1982; Patel, Fogarty, Landay y Harrison, 2008; Rosson y Carroll, 1996; Soloway, Ehrlich y Bonar, 1982), sin embargo la preocupación no se ha disipado, y sigue siendo un tema de total vigencia.

Como hemos dicho antes, la ley educativa de la Comunidad de Madrid establece Scratch como la herramienta a utilizar para el proceso de enseñanza-aprendizaje de la programación (Decreto Comunidad de Madrid, 2015, p. 90). De esta mención expresa, surge la presente tesis, preguntándonos si es esta herramienta la adecuada para la enseñanza de la programación en el contexto de la Comunidad de Madrid.

Para responder a esta pregunta primero debemos acotar qué significa adecuado, porque según lo entendamos, podremos abordar el estudio desde diferentes perspectivas. Dado que se pretende establecer esta adecuación en términos científicos, el baremo que utilicemos debe ser medible. Por este motivo se escoge la usabilidad para comprobar la adecuación de Scratch como herramienta para el proceso de enseñanza-aprendizaje de la programación.

La usabilidad es un término que hace referencia a la facilidad de uso de una aplicación o producto interactivo (Baeza Yates y Rivera Loaiza, 2002). Como se verá en el capítulo 5.1, tiene un carácter empírico (Hassan-Montero y Ortega-Santamaría, 2009), por lo que podremos utilizarla como instrumento científico para medir la adecuación de Scratch.

En capítulo 4.4.4 se describen los factores que influyen en el aprendizaje de la programación. Entre estos factores se menciona por un lado la dificultad intrínseca asociada la programación (sintaxis complicadas, entornos de desarrollo poco asequibles, etc.), y por otro la falta de adecuación de los medios de enseñanza que se utilizan.

Al unir esta definición con lo expuesto en el párrafo anterior, podemos decir que, si el medio utilizado para enseñar a programar es usable, el aprendizaje se facilitará, o al menos se eliminará uno de los factores que lo obstaculizan. Igualmente, si se

consigue paliar la dificultad intrínseca que acarrea la programación, se estará allanando el camino para su aprendizaje y su uso.

Como se verá en el capítulo 5.3, los estudios científicos de usabilidad aplicados a los lenguajes y a las interfaces de programación tienen una larga tradición, y se realizan, entre otros motivos, con el propósito de romper las barreras que se encuentra un principiante al aproximarse a la programación.

Cualquier producto, cuando se concibe, está pensado para satisfacer las necesidades de un público específico, que comparte unas características homogéneas que influyen decisivamente en su forma de relacionarse con la tecnología. Es decir, estos productos serán usables si lo son para esta audiencia objetiva, lo cual no significa que necesariamente deban serlo para el resto de la población (Hassan-Montero y Ortega-Santamaría, 2009). Por consiguiente, una vez decidido que se va a utilizar la usabilidad como criterio para medir la adecuación de Scratch, se debe circunscribir el ámbito del estudio. Así, la presente investigación se acota a la Escuela Primaria de la Comunidad de Madrid, por ser el contexto en el que se aplica el Decreto 89/2014 que da origen a esta tesis, y por otras razones que se describen en la siguiente sección.

Por la relativamente reciente implantación de esta ley, y porque la impartición de la asignatura de libre configuración “Tecnología y recursos digitales para la mejora del aprendizaje” no se está haciendo de forma masiva en toda la Comunidad de Madrid, no existen estudios de referencia que hayan evaluado la adecuación de Scratch desde la perspectiva de la usabilidad. Sin embargo, un estudio de estas características se antoja necesario: si esta herramienta es la adecuada, su elección estará propiciando la consecución de los objetivos que plantea esta asignatura, es decir, que los estudiantes aprendan los fundamentos de la programación, y de esta manera adquieran una de las competencias consideradas como clave por la Unión Europea para los ciudadanos del Siglo XXI. Si la herramienta no es usable, aprender a utilizarla eclipsará el verdadero objetivo que marca la ley, que no es otro que aprender a programar.

De esta manera se justifica la pertinencia de la realización del estudio que aborda esta tesis, y su actualidad.

1.2. Estructura y contenido de la tesis

La pregunta de investigación que se plantea es la siguiente: ¿Es Scratch, desde el punto de vista de la usabilidad, una herramienta adecuada para aprender a programar, en el contexto de la Educación Primaria de la Comunidad de Madrid?

Los conceptos que sostienen esta pregunta, y dónde se abordan dentro de la tesis, son los siguientes:

- Scratch: el proyecto de Scratch y sus características se describen en el capítulo 6.
- Lenguaje de programación: en el capítulo 3 se explica qué es un lenguaje de programación, se describen sus distintas tipologías, y se explica las reglas sintácticas propias de un lenguaje estructurado como el que utiliza Scratch.
- Aprender: en el capítulo 4.1 se hace un recorrido por las diferentes teorías de aprendizaje.
- Aprender a programar: en el capítulo 4.4 se enlazan las teorías de aprendizaje con el aprendizaje concreto de la programación. Asimismo, se describen las capacidades a desarrollar en este proceso, centrando el foco en dos aspectos fundamentales: la capacidad para resolver problemas (capítulo 4.2), y la capacidad para plantear esa solución en los términos que el ordenador entiende, es decir, el denominado pensamiento computacional (capítulo 4.3). También se abordarán las posibles técnicas, metodologías y herramientas a utilizar en este proceso (capítulos 4.4.1, 4.4.2 y 4.5 respectivamente).
- Usabilidad: en el capítulo 5 se explica lo que es usabilidad, y dado su carácter empírico, se describen los posibles mecanismos para su medición, tanto en ámbitos generales, como en los específicos relacionados con la programación.
- Educación Primaria en la Comunidad de Madrid: todo lo relacionado con este contexto se describe en el capítulo 2.

La hipótesis de investigación es que Scratch es una herramienta que facilita el aprendizaje de la programación desde el punto de vista de la usabilidad. De validarse esta hipótesis, habría una evidencia más, que reforzaría la idea de que Scratch es la herramienta que se debe utilizar en esta asignatura de libre configuración que propone la Comunidad de Madrid.

De esta manera el objetivo general de la investigación sería evaluar la herramienta Scratch desde el punto de vista de su usabilidad para determinar la adecuación de

su uso para el aprendizaje de la programación. Los objetivos de la investigación se pormenorizan en el capítulo 8.2.

Para llevar a cabo esta investigación se escoge la metodología descriptiva (Hernández Sampieri, Fernández Collado y Baptista Lucio, 2004), de muestra estructural (Ibáñez, 2003), mediante la técnica cuantitativa de la encuesta (Beltrán, 2000).

En un estudio cuantitativo descriptivo se definen una serie de cuestiones para recabar información, y se miden los datos recogidos sobre cada una de ellas, para así describir lo que se investiga (Hernández Sampieri et al., 2004, p. 95). En este caso lo que se investiga es la usabilidad de Scratch como indicador de la adecuación de su uso para aprender a programar.

Con respecto a la muestra, al ser el objeto de esta investigación la evaluación de una tecnología, las decisiones muestrales no buscan la representación estadística, ni tienen como objetivo que los datos recogidos sean extrapolables estadísticamente a la población general. En lugar de la inferencia, este estudio pretende comprender mejor el significado del fenómeno estudiado, y el de sus conceptos asociados (Valles Martínez, 2003, p. 92). Los detalles de la población y muestra se encuentran en el capítulo 9.2, y en el 10.3.4 se pormenoriza lo relativo a la muestra con la que se ha realizado el estudio.

Para cumplir con los objetivos de la investigación, el análisis de Scratch se realizó para su versión en castellano, y se planificó en tres fases:

- análisis de experto
- análisis heurístico
- análisis de usuario.

Los detalles de estos instrumentos de análisis, y la justificación de su validez en un estudio de usabilidad, se detallan en el capítulo 5.2. Los análisis de experto, heurístico y de usuario llevados a cabo, se describen en los capítulos 10.1, 10.2, y 0 respectivamente.

En el análisis de experto se determinaron las normas de usabilidad que se debían tomar como referentes en las siguientes fases del estudio. En el análisis heurístico, de entre las normas escogidas, se definieron y evaluaron las pautas necesarias para comprobar si Scratch es una herramienta usable. Estas pautas son los indicadores

que permiten verificar el cumplimiento de los objetivos establecidos en la investigación (ver capítulo 8.2). La medición de estos indicadores se realizó a través del cuestionario, cuyas preguntas surgieron de las pautas procedentes del análisis heurístico, y de los estándares de aprendizaje de la programación presentados en el marco teórico (ver capítulo 10.3.1). Las variables que operacionalizan los indicadores son las posibles respuestas a las que da opción el cuestionario. Todos los detalles de su elaboración se encuentran en el capítulo 10.3.1.

El análisis de los datos recogidos por el cuestionario se ha hecho en términos de números absolutos. Es decir, se contabiliza cuántos participantes han elegido cada una de las opciones, y posteriormente se calculan los porcentajes en función de las respuestas que ha habido a esa pregunta, no en función del total de participantes, considerando fuera del estudio a efectos estadísticos a los que no han contestado. A partir de estos datos se describe la realidad que muestran.

En la planificación de la investigación se previó la posibilidad de que hubiera ciertos aspectos externos a la herramienta que pudieran tener influencia en las respuestas recogidas, concretamente la disponibilidad de ordenador en casa, la formación recibida en el manejo de la herramienta, y la posible práctica autónoma con ella fuera del aula. Por este motivo se decidió estudiar las tendencias señaladas por los datos recogidos, para ver si merecería la pena hacer un análisis más profundo de esta posible relación. Esto se concretó en un análisis bivariable que permitiera observar las tendencias, y se subordinó la decisión de hacer un estudio correlacional a los resultados de dicho análisis. En el capítulo 10.3.6 se explica cómo se resolvió este tema.

A partir de la discusión de los resultados obtenidos en el análisis de datos del cuestionario, se establecen las conclusiones, se comprueba la consecución de los objetivos planteados, y se responde a la pregunta de investigación. Todo ello se aborda en detalle en el capítulo 12.

1.3. Principales aportaciones de esta tesis

Esta tesis aborda un tema de total actualidad, como es la enseñanza de los lenguajes de programación en la Escuela Primaria. Asimismo, ofrece un estudio diseñado para validar uno de los aspectos de las vigentes políticas educativas, y verificar que están orientadas en la dirección correcta.

Esta tesis, por un lado, ofrece una reflexión sobre el proceso de enseñanza-aprendizaje de la programación. Por otro lado, proporciona un instrumento que

permite evaluar desde la perspectiva de la usabilidad, la adecuación de una determinada herramienta para dicho proceso de enseñanza-aprendizaje.

La herramienta que se somete a evaluación es Scratch. En las conclusiones se verá que Scratch es una herramienta usable, aunque presenta aspectos susceptibles de mejora en este ámbito.

También se concluye que saber utilizar Scratch no es lo mismo que saber programar. Como se explica en el capítulo 4, saber programar tiene dos componentes: saber resolver un problema en los términos que el ordenador comprende (pensamiento computacional) y saber expresar esa solución con la sintaxis adecuada. Con respecto a la sintaxis, en esta investigación se han encontrado indicadores que señalan que la lógica de bloques de Scratch se entiende. Pero que se sepa qué piezas se pueden combinar, no significa que se conozca cuáles se deben utilizar, y cómo se deben agrupar para elaborar un programa que dé respuesta a un problema. Es decir, con los datos de esta investigación no podemos afirmar ni negar que aprendiendo a manejar Scratch se fomente la adquisición del pensamiento computacional.

Asimismo, la investigación aporta que en el proceso de enseñanza-aprendizaje de la programación a través de Scratch, el papel del docente se revela como fundamental.

Por último, esta tesis ofrece en el capítulo 4.5 a modo de resumen, y en los anexos de forma extendida, una presentación de posibles herramientas alternativas a utilizar en lugar de Scratch, si en el estudio se confirmara que no es adecuada, o si se quisiera comparar con otras para así determinar su idoneidad. La presentación de estas herramientas es meramente descriptiva, dejando su análisis para futuras investigaciones.

—

PARTE TEÓRICA

—

2. Marco legislativo

Para entender por qué Scratch y la enseñanza de los lenguajes de programación están ahora incluidos en la ley de la Comunidad de Madrid, primero se hará una revisión del contexto europeo, y de cómo la programación se ha incluido a nivel curricular en los espacios educativos de los distintos países. Segundo, se hará una breve revisión de la evolución de las distintas leyes educativas, desde la LGE (Ley General de Educación) de 1970, hasta la LOMCE (Ley Orgánica de la Mejora en la Calidad Educativa) de 2014, para comprobar cómo la tecnología y los lenguajes de programación se han ido incorporando al plano legislativo educativo.

Dado que uno de los objetivos de este capítulo es ubicar la tecnología dentro de la legislación educativa, se antoja necesario presentar previamente una aproximación del concepto de las TIC (Tecnologías de la Información y la Comunicación).

Marques (2000) hace un análisis de las TIC para entender el alcance educativo de las mismas, y resalta algunas cuestiones relacionadas con la terminología empleada. El factor tecnológico, como el conjunto de conocimientos científicos puestas al beneficio de las actividades humanas; el factor informativo como el conjunto de datos compartidos a nivel colectivo; y el comunicativo, como el conjunto de mensajes transmitidos entre personas.

Por lo tanto, según esta aproximación terminológica se determina que las TIC, no solo son meras herramientas por donde circula el conocimiento, sino que además, su alcance tiene valor a nivel social.

Cobo Romaní (2011), por otro lado, define las TIC como:

> Tecnologías de la Información y la Comunicación (TIC): Dispositivos tecnológicos (hardware y software) que permiten editar, producir, almacenar, intercambiar y transmitir datos entre diferentes sistemas de información que cuentan con protocolos comunes. Estas aplicaciones, que integran medios de informática, telecomunicaciones y redes, posibilitan tanto la comunicación y colaboración interpersonal (persona a persona) como la multidireccional (uno a muchos o muchos a muchos). Estas herramientas desempeñan un papel sustantivo en la generación, intercambio, difusión, gestión y acceso al conocimiento. (p. 312)

Según se entiende en esta apreciación, el componente social de las TIC es de vital relevancia en el ámbito educativo, puesto que los espacios físicos tradicionales donde se desarrollaba el intercambio social es sustituido por el ciberespacio. Obviamente este hecho brinda una oportunidad para la creación de un nuevo

espacio social en el que se redefinan los parámetros de las relaciones sociales tradicionales.

Otro aspecto a destacar en esta definición del autor es que si los medios por donde se vehicula la información han cambiado, también habrá cambiado el proceso cognitivo mediante el cual se adquiere ese conocimiento.

En consecuencia, tanto en el aspecto social como cognitivo serán vitales para entender las principales aportaciones de las TIC al ámbito de la educación (Marquès Graells, 2000).

2.1. La enseñanza de la programación en el sistema educativo europeo

Según el informe *Computing our future* (Balanskat y Engelhardt, 2015), entre los países de la Unión Europea, la enseñanza de la programación ya forma parte del plan de estudios (a nivel nacional, regional o local) en 16 de ellos: Austria, Bulgaria, Dinamarca, Estonia, Francia, Hungría, Irlanda, Israel, Lituania, Malta, España, Polonia, Portugal, Eslovaquia y el Reino Unido (Inglaterra). A esta lista, hay una última incorporación, la de Finlandia, que ha integrado la programación dentro del currículo básico para el curso 2016-17.

Es de especial relevancia como Francia y Polonia han avanzado en la integración de la programación en el plan de estudios entre 2014 y 2015. Este último está estableciendo un nuevo currículo de ciencias de la computación para todos los niveles escolares. En Francia, los nuevos planes de estudio para la escuela (primaria y secundaria inferior) se han hecho públicos en septiembre / octubre de 2015 y se han puesto en práctica en septiembre de 2016.

Bélgica, los Países Bajos y Noruega, para el curso 2016/2017, no han integrado plenamente la programación en sus planes de estudios. Noruega, para este año escolar, decidió, a modo de prueba, incluir la programación como asignatura opcional en las escuelas secundarias inferiores. Muchas escuelas ya han comenzado a enseñar programas a los estudiantes como parte de la materia opcional "Tecnología en la práctica" (secundaria inferior), o como parte de las matemáticas y ciencias naturales. Los Países Bajos no incluyen la programación en sus planes de estudios. Hay una asignatura de informática en la educación secundaria, pero no es obligatoria, y las escuelas pueden elegir si impartirla o no. Incluso dentro de las escuelas que se decanten por impartirla, los estudiantes pueden decidir cursarla o no. Si bien no hay planes inmediatos para integrar la programación como tema obligatorio, esta cuestión aún está pendiente de debate.

El Instituto Nacional para el Desarrollo Curricular (SLO) está trabajando en el desarrollo de metas y un currículo posible sobre las competencias digitales, incluyendo la programación / pensamiento computacional. En Bélgica, en otoño de 2015 se inició el debate sobre la reforma curricular en general, y sobre la integración de las competencias digitales y la programación, tanto en la educación de niños como de adultos.

En la Tabla 1 se muestra, a modo de resumen, el panorama de integración de los lenguajes de programación en el currículo de 19 países de la Unión Europea, incluido España, y su año de implantación (Balanskat y Engelhardt, 2015).

Tabla 1
Integración de los lenguajes de programación en el currículo de 19 países de la Unión Europea, incluido España, y año de implantación

	NACIONAL	REGIONAL	A NIVEL DE ESCUELA	AÑO DE IMPLANTACIÓN
AUSTRIA	X			
BÉLGICA		X		
BULGARIA	X			
REPÚBLICA CHECA			X	
DINAMARCA	X			2014
ESTONIA	X		X	
FINLANDIA	X	X	X	2016
FRANCIA	X			2016
HUNGRÍA	X			1995
IRLANDA	X		X	2014
ISRAEL	X			1976
LITUANIA	X		X	1986
MALTA	X			1997
POLONIA	X			1985
PORTUGAL	X			2012
ESLOVAQUIA	X		X	1990
ESPAÑA	X	X		2015
REINO UNIDO (INGLATERRA)	X			2014

Fuente: (Balanskat y Engelhardt, 2015, p. 37)

En la *Figura 1* se muestra un resumen de la situación en 2015 del nivel de integración de las Ciencias de la Computación en el currículo escolar de 18 países de la UE, y el nivel educativo en el que se imparte esta disciplina (FECYT, Google y Everis, 2016).

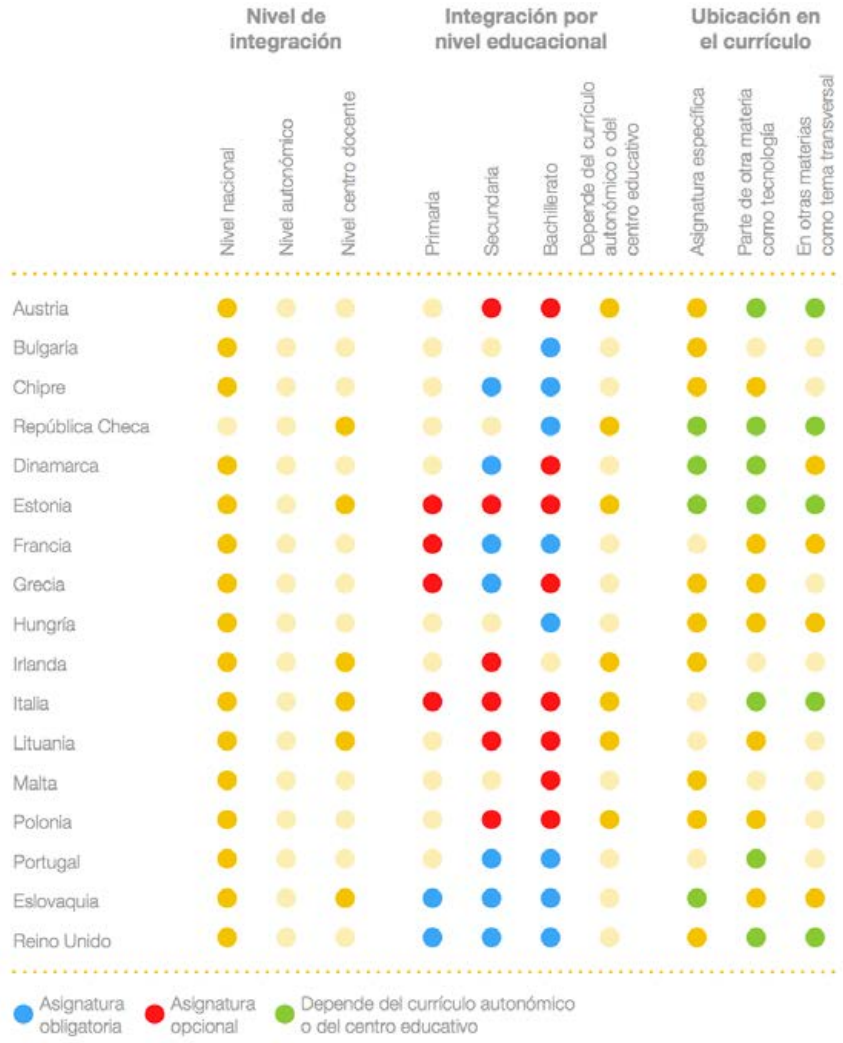


Figura 1. Resumen de la situación en 2015 del nivel de integración de las Ciencias de la Computación en el currículo escolar de 18 países de la UE, y el nivel educativo en el que se imparte esta disciplina.

Fuente: (FECYT et al., 2016, p. 15)

Según un estudio realizado por Balanskat y Engelhardt (2015, p. 26), en general, los países de la Unión Europea integran la programación en sus planes de estudios por diferentes razones. La mayoría de los países tiene como objetivo desarrollar las habilidades de pensamiento lógico de sus estudiantes (15 países), y las capacidades para la resolución de problemas (14 países). Más de la mitad de los países, es decir, 11, se centran en el desarrollo de las competencias clave establecidas por el marco europeo, y las competencias de programación. El interés creciente de cada vez más estudiantes para estudiar ciencias de la Informática es también un motivo para 11 países. El objetivo de fomentar la empleabilidad en el sector es clave para solo 8 países.

Polonia está en el proceso de introducir la informática y la programación a todos los estudiantes a partir de 12 años.

Por último, merece la pena destacar el currículo nacional del Reino Unido, que incluye la enseñanza de los lenguajes de programación en todas las etapas educativas a partir de los 5 años. Por su relevancia, en los anexos se incluyen los detalles de este currículo, y el denominado *Computing Progression Pathways* (Dorling y Walker, 2015), que podríamos traducir como Itinerarios para la Progresión en la Informática, donde se describen los resultados de aprendizaje correspondientes a las áreas de Informática, Tecnología de la Información y Alfabetización Digital en la evolución escolar de un estudiante.

2.2. Evolución de las políticas educativas en España en relación con las TIC

Este apartado trata de hacer un recorrido por la evolución de la legislación educativa española en las últimas décadas, sobre la inclusión de las TIC en el currículum, y de cómo ha cambiado este planteamiento a nivel pedagógico.

En un primer momento se mencionará brevemente la LGE (Ley General de Educación), que dará paso a la reforma de 1990 promovida por la LOGSE (Ley Orgánica General del Sistema Educativo), un texto que intentó reformar o "modernizar" el sistema educativo español, equiparándolo al contexto europeo. Según Revuelta Guerrero y Escolano Benito (2003), fue una reforma que a nivel pedagógico e incluso social supuso un punto y aparte porque, aunque la ley de 1970 seguía siendo funcional y progresista con la lógica tardofranquista, no se ajustaba a los cambios que se estaban dando e iban a darse a nivel científico-tecnológico y social. Por este motivo, aunque antes de la LOGSE, se empiezan a realizar cambios importantes para el tema que nos ocupa, fundamentalmente se van a analizar los textos a partir de este momento.

En esta reseña se valorarán aquellas disposiciones legales que han tenido una vigencia suficiente para propiciar cambios considerables en el sistema educativo, obviando aquellas que no han supuesto a nivel curricular cambios considerables con respecto a sus predecesoras, o bien, no han llegado a aplicarse por motivos políticos, como es el caso de la LOCE (Ley Orgánica de Calidad en la Educación).

En este recorrido, se ha colocado el texto de la LOE (Ley Orgánica de Educación) como ley "bisagra" entre dos momentos dentro de la concepción de las TICS en el sistema educativo español, en el intento de equiparar el sistema educativo con el panorama tecnológico fuera de las aulas. Por ejemplo, en la LOE se configuró una

modalidad integrada de Bachillerato, el Científico Tecnológico, en el que además de una redefinición del currículo en esta área de conocimiento, se empezó a gestar la necesidad de materias que serían transversales y que serían también una base científico-tecnológica para todos los alumnos de este nivel.

Al final de este recorrido, veremos cómo la Comisión Europea establece la Competencia Digital como una de las competencias clave a desarrollar en el currículo actual, y así lo recoge la LOMCE (Ley Orgánica para la Mejora de la Calidad Educativa), que es la ley vigente en el momento en el que se desarrolla esta tesis.

Cabe aclarar también que en este capítulo se atenderá básicamente a los aspectos curriculares, pues son donde podemos ver cómo se ha ido gestando un interés creciente por las tecnologías, y particularmente por los lenguajes de programación.

2.2.1. Antecedentes legislativos: LGE, LOGSE y LOCE

2.2.1.1. La Ley General de Educación (LGE)

En el momento que estaba vigente la Ley 14/1970 de Educación General, hay una intención por parte de la comunidad educativa por introducir e integrar el ordenador a nivel curricular, principalmente por parte de los docentes (Rodríguez Mondejar, 2000), bajo las directrices del Proyecto Atenea (Escudero, 1991). De hecho, a partir del Bachillerato Unificado Polivalente (BUP) y el Curso de Orientación Universitaria (COU) se empieza a impartir *I*nformática dentro de las Enseñanzas y Actividades Técnicos Profesionales (EATP).

Sin duda, este fue el antecedente de una reforma que progresivamente se sucedería en las siguientes décadas, y en las que las TIC cobran un papel fundamental en el Diseño Curricular Base (CNIIE, 1991).

2.2.1.2. La Ley de Ordenación General del Sistema Educativo (LOGSE)

En este punto de la evolución legislativa, la LOGSE ("Ley Orgánica 14/1970, del 6 de agosto, General de Educación y Financiamiento de la Reforma Educativa. Boletín Oficial del Estado. núm. 187, pp. 12525 a 12546.," 1970) fue un momento crucial para que poco a poco se fuera pergeñando un espacio para el desarrollo de competencias y destrezas sobre las tecnologías de la información y comunicación, tanto desde el punto de vista del conocimiento de la propia tecnología, en cuanto a

herramientas y principios teóricos para su funcionamiento, como al manejo de datos para un fin concreto.

Antes de iniciar un análisis más detallado de la importancia de esta reforma, se hace necesario enunciar las principales características a nivel curricular.

Principalmente, hay una diferencia en la nomenclatura y en la duración de la enseñanza obligatoria. A partir de este momento la educación obligatoria finalizaría con el curso 4º de la ESO con edad de escolarización obligatoria de hasta los 16 años de edad. De tal manera que la Educación Primaria pasa a tener en este momento seis años de duración (desde los 6 hasta los 12 años de edad), la Educación Secundaria Obligatoria (ESO) tiene una duración de 4 años (desde los 12 hasta los 16 años de edad) y se introduce también la etapa del Bachillerato que dura dos años y que no tiene edad de permanencia como en el caso anterior.

La importancia de la nueva planificación del Bachillerato es la inclusión de cuatro modalidades de Bachillerato, entre las que destacamos la modalidad Tecnológica.

A nivel metodológico se produce un salto cualitativo puesto que en la etapa de Educación Primaria y la ESO, el proceso de enseñanza-aprendizaje se adapta a los ritmos de aprendizaje según las particularidades de cada estudiante. De hecho, se prevé una adaptación curricular y atención a la diversidad para conseguir una mejor integración de todos los estudiantes con necesidades especiales.

En Bachillerato, en este aspecto, pretende capacitar al alumno para desarrollar la competencia de “aprender a aprender”, es decir, una mayor autonomía para el aprendizaje, tanto a nivel individual como en equipo.

El lanzamiento del nuevo texto-marco legislativo coincide con el momento en el que el Ministerio lanza el “Programa de Nuevas Tecnologías de la Información y la Comunicación” para el empleo de los ordenadores en los centros educativos. Este programa se concentraba en los siguientes aspectos:

- Reflexión de cómo incluir las tecnologías a nivel didáctico.
- Inclusión de las TIC en las diferentes áreas de conocimiento y comprobar y explorar las posibilidades interdisciplinares de las mismas.
- Potenciar el aprendizaje autónomo, tanto en un contexto individual como grupal.
- Incluir en el aprendizaje de estas herramientas didácticas a alumnos con necesidades educativas especiales (CNIIE, 1991).

A nivel de contenidos curriculares es reseñable dentro de la Educación Primaria, en la elaboración del Decreto 1006/1991, del 14 de junio, por el que se establecen las

enseñanzas mínimas correspondientes a esta etapa educativa, que dentro del área de Conocimiento del Medio Natural, Social y Cultural, se plantea como objetivo el identificar la necesidad de la tecnología en la vida humana (Real Decreto 1006/1991, 14 de junio), al mismo tiempo que en los bloques de contenido de este área, 7 y 9, "Máquinas y aparatos" y "Medios de comunicación y transporte" respectivamente, se tocan tangencialmente conceptos, procedimientos y actitudes, relacionados con los cambios tecnológicos y comunicacionales en el entorno.

En cuanto a la etapa educativa de la ESO, cabe destacar el impacto a nivel curricular de la inclusión de "Tecnología" como asignatura independiente y obligatoria en los tres primeros años de etapa, convirtiéndose en optativa en el último año (Real Decreto 1007/1991 14 de junio).

Dentro de las diferentes áreas se comprueba cómo se aboga por el carácter interdisciplinar de las TIC, por ejemplo, en el área de Ciencias de la Naturaleza se determina como conocimiento a adquirir el emplear diferentes fuentes de información, para evaluar el contenido de las mismas y una valoración crítica, o también, en el área de Geografía e Historia se plantea el empleo de las herramientas TIC "como recurso didáctico y fuente de conocimiento e información" (Escandell Bermúdez, Rodríguez Martín y Cardona Hernández, 2005, p. 227).

En el Real Decreto 1345/1991 del 6 de septiembre por el que se establece el currículo de la ESO, se especifica que mientras que en la Educación Primaria, la tecnología está integrada como un punto de interacción de los estudiantes con el entorno, en la ESO, deriva en el área de conocimiento específica de la Tecnología, ya que se requiere de esquemas de conocimiento más complejos para profundizar en destrezas y actitudes más analíticas para comprender el impacto a nivel social y científico del desarrollo tecnológico.

Asimismo, en los objetivos generales de etapa, se destaca la necesidad de utilizar las TIC como un instrumento clave en el proceso de enseñanza-aprendizaje. El texto también pone de manifiesto una idea primordial sobre el alcance cognitivo y actitudinal en el desarrollo del "espíritu tecnológico", puesto que su inclusión en esta etapa determina la integración de métodos y procesos de detección de un problema y el desarrollo de una solución.

Según López Curiel (2014), tras la publicación del Real Decreto 3473/2000 del 29 de diciembre que modifica el currículo del área del citado texto de 1991, se recogen todos los numerosos cambios acontecidos en ese periodo y que sobre todo sitúa a Internet como potente medio para localizar, generar y valorar información de diferente índole.

En cuanto a los contenidos, como explica López Curiel (2014, pp. 29-30), en primero y segundo, los contenidos se centrarán en desarrollar contenidos procedimentales sobre el manejo de Internet para la búsqueda de información y como medio de comunicación, así como la introducción de software de creación de gráficos o los componentes básicos de un ordenador.

En tercero y cuarto de la ESO se van a empezar a abordar contenidos más complejos como los lenguajes de programación y sus posibles aplicaciones, una introducción a la robótica, el funcionamiento de la máquina a diferentes niveles, y la organización de la información a través de los gestores de las bases de datos o redes informáticas.

Aunque el decreto promulgado en el 1991 y su posterior modificación en el decreto del 2000 demuestran una preocupación incipiente y real sobre la alfabetización digital y sobre todo, de la revisión de los contenidos curriculares para adaptarlos a los vertiginosos cambios en el desarrollo de estas herramientas digitales. No obstante, su inclusión queda relegada al ámbito de la asignatura de Tecnología e en cursar como optativa la asignatura de Informática en el segundo ciclo de la ESO (3º y 4º). A pesar de ellos, en la aparición que vemos en los textos es anecdótica y se entienden éstas como meras herramientas secundarias o complementarias que por sus características motivacionales son un buen método para incentivar la comprensión de ciertos contenidos de otras áreas de conocimiento, pero que obvian lo que a juicio de Moreno Herrero (2005) debe ser los marcos de referencia para determinar su fundamentación educativa: funcionalidad, cuyos criterios son el empleo de estos recursos que contribuyen a mejorar la organización pedagógica y para el desarrollo del proceso de enseñanza-aprendizaje o para facilitar nuevas aplicaciones y usos con los mismos; posibilidades didácticas, entendidas éstas como las posibles aplicaciones para fomentar la innovación docente de acuerdo a unas necesidades concretas, y para fomentar el aprendizaje significativo, las relaciones entre iguales, el conocimiento de diferentes lenguajes o el aprendizaje autónomo y colaborativo; y por último, sobre los aspectos técnicos, cuyos criterios radican en la versatilidad en su uso, el adecuado manejo, la posibilidad de trabajar en entornos multitarea, multiusuario o en redes interconectadas (Moreno Herrero, 2005, p. 393).

2.2.2. Leyes vigentes: LOE y LOMCE

Tras el avance de la LOGSE en el área de conocimiento sobre la que versa esta investigación, se han sucedido dos reformas: la LOE (Ley Orgánica de la Educación) y la LOMCE (Ley Orgánica para la Mejora de la Calidad Educativa). Ambas leyes se agrupan en este capítulo porque la LOMCE, en el momento en el

que se empieza a redactar la presente tesis, está en proceso de implantación, y la LOE no está derogada en su totalidad.

2.2.2.1. Antecedentes europeos

Este apartado sirve como antesala al resto del capítulo, señalando algunos referentes europeos que dan lugar a las vigentes leyes educativas españolas.

“La educación tiene que adaptarse en todo momento a los cambios de la sociedad, sin dejar de transmitir por ello el saber adquirido, los principios y los frutos de la experiencia”. (Delors y Century, 1996).

La abundancia de información, a la que tenemos acceso gracias a las tecnologías, nos muestra hechos y sucesos que debemos interpretar y analizar para que sean convertidos en conocimiento. Este conocimiento es la base sobre la que se asienten "*sociedades de conocimiento auténticas, que sean fuente de desarrollo humano y sostenible*" (Bindé, 2005).

El Informe Delors (1996) acuña el concepto de educación para toda la vida indicando que "*la escuela debe inculcar el gusto y el placer de aprender, la capacidad de aprender a aprender*" donde cada uno pueda responsabilizarse de sí mismo y realizar su proyecto personal. Según este informe, la educación para toda la vida se fundamenta en los siguientes principios:

- Aprender a vivir juntos
- Aprender a conocer
- Aprender a hacer
- Aprender a ser

Las competencias clave, según la red EURYDICE (2002), son las que se consideran indispensables para una participación satisfactoria en la sociedad a lo largo de la vida. La necesidad de memorización por parte de las personas ya no es un pilar básico educativo debido, entre otras causas, a la inmediatez del acceso a la información; en cambio necesitamos instrumentos que permitan "*seleccionar, procesar y aplicar el conocimiento requerido con el fin de hacer frente a los modelos cambiantes de empleo, ocio y familia. Esto explica la tendencia creciente en la enseñanza por desarrollar competencias en vez de enseñar conocimientos de hechos*" (EURYDICE, 2002).

El Proyecto DeSeCo (Rychen y Salganik, 2000) clasifica las competencias clave en tres categorías:

- Usar herramientas de manera interactiva. Implica tanto la destreza técnica como su uso para alcanzar metas más amplias de intercambio entre la persona y su ambiente.
- Interactuar con grupos heterogéneos. Necesitamos saber tratar a las personas en una sociedad cada vez más plural y diversa.
- Actuar de forma autónoma.

La Recomendación 2006/962/CE sobre las competencias clave para el aprendizaje permanente del Parlamento Europeo (2006), define ocho competencias clave:

- Comunicación en la lengua materna.
- Comunicación en lenguas extranjeras.
- Competencia matemática y las competencias básicas en ciencia y tecnología.
- Competencia digital.
- Aprender a aprender.
- Competencias sociales y cívicas.
- Sentido de la iniciativa y el espíritu de empresa.
- Conciencia y la expresión culturales.

2.2.2.2. Ley Orgánica de la Educación (LOE)

En este apartado se realizará una reseña de lo que suponen la LOE y la LOMCE en cuanto a los principios metodológicos, los cambios fundamentales con respecto a la anterior ley, y también con respecto a los contenidos curriculares y la implementación de las TIC en las diferentes áreas de conocimiento que nos interesan.

Comenzaremos analizando la Ley Orgánica de Educación 2/2006 (LOE), y los cambios acontecidos con respecto a los textos precedentes.

Uno de los ejes rectores del currículo en esta ley es el trabajo por competencias básicas que deberán adquirirse al final de la etapa. A partir de la LOE el currículo se entiende como el conjunto de objetivos, competencias básicas, contenidos, métodos pedagógicos y criterios de evaluación de cada una de las enseñanzas reguladas en la presente Ley (LOE 2/2006, del 3 de mayo). La LOE establece que las competencias básicas se deben alcanzar en la ESO siendo prescriptivo, para las administraciones educativas, una evaluación de diagnóstico de las competencias básicas alcanzadas por los alumnos al final del segundo ciclo de la Educación Primaria. Es decir, la etapa de Educación Primaria tiene un carácter global e

integrador y que supone una preparación para el trabajo por competencias que se va a desarrollar en la siguiente etapa educativa.

El estado español se hizo eco de las recomendaciones europeas al incluir las competencias. Ya en el preámbulo de la LOE se indica la necesidad de "*una educación completa, que abarque los conocimientos y las competencias básicas que resultan necesarias en la sociedad actual*". Existen ocho competencias básicas a través de las cuales se logra el desarrollo personal del alumno. Estas competencias son las que marcan los objetivos educativos, que a su vez determinarán los contenidos que se impartirán en el aula.

En diferentes decretos desarrollados a partir de la nueva ley se entiende cómo son incluidas las competencias. En el Real Decreto 1513/2006 del 7 de diciembre, y en el Real Decreto 1631/2006 del 29 de diciembre por el que se establecen las enseñanzas mínimas de la Educación Primaria y Secundaria respectivamente, se indica como las competencias básicas se incorporan por primera vez a las enseñanzas mínimas indicando que su logro deberá capacitar a los alumnos y alumnas para su realización personal, el ejercicio de la ciudadanía activa, la incorporación a la vida adulta de manera satisfactoria y el desarrollo de un aprendizaje permanente a lo largo de la vida.

En el anexo primero del Real Decreto 1513/2006 se menciona que la relación entre las áreas y materias con las competencias básicas no es unívoca, indicando que cada una de las áreas contribuye al desarrollo de diferentes competencias y, a su vez, cada una de las competencias básicas se alcanzará como consecuencia del trabajo en varias áreas o materias (Anexo I, Real Decreto 1513/2006, 7 de diciembre). Esto sucede de igual forma en el Real Decreto 1631/2006 relacionado con el currículo de Secundaria puesto que en cada una de las materias a impartir y de las que se fijan los contenidos, se da cuenta de la contribución que tiene cada una de ellas para la adquisición de las diferentes competencias.

Una de estas competencias es la llamada competencia TICD (Tratamiento de la Información y Competencia Digital), que es el espacio que ha reservado la normativa para la inclusión de las llamadas nuevas tecnologías (aunque dado que son tecnología del presente, ya no se pueden llamar nuevas) en la educación española, y por tanto donde tendrían cabida los lenguajes de programación. También como se ha expuesto no hay una relación unívoca entre un área de conocimiento y una competencia determinada. Es decir, en el caso concreto de la competencia TICD, son varias las áreas de conocimiento que la desarrollan.

Según López (2006), la competencia TICD:

> Es el conjunto de habilidades para buscar, obtener, procesar y comunicar información y transformarla en conocimiento. Incluye aspectos diferentes que van desde el acceso y selección de la información hasta el uso y la transmisión de ésta en distintos soportes, incluyendo la utilización de las tecnologías de la información y la comunicación como un elemento esencial para informarse y comunicarse (p.11).

Al mismo tiempo que ésta se relaciona con la obtención crítica de información y requiere el dominio básico de lenguajes específicos y pautas de programación de los mismos.

Dentro de esta definición y de cómo se entiende esta competencia para el desarrollo integral de un individuo en periodo de formación, básicamente, esta competencia versa sobre la capacidad para analizar información que "otro" construye, dando menos espacio para la creación de contenidos, es decir, se entiende el desarrollo de la capacidad crítica para valorar estos mensajes y para poder comunicarse, pero se tiene poco en cuenta al docente como agente principal de creación de contenidos.

En cuanto a la configuración del plan de estudios de las diferentes etapas, llama la atención el cambio que se produce en la asignatura de "Tecnología", donde quizás, se aborda de una manera más incisiva la competencia de la que se ha hablado previamente. En el artículo 4 del Real Decreto 1631/2006 se establece que esta materia, que pasa a llamarse "Tecnologías", ya no es de carácter troncal en los tres primeros cursos de la ESO, y por lo tanto era competencia de las comunidades autónomas decidir cuándo se impartiría, y el desarrollo de los contenidos fijados en los anexos de esta disposición. En este mismo artículo se comprueba que la organización curricular para el último curso de la ESO, tanto Tecnología como Informática pasaban a ser ofertadas como optativas, dejando también a criterio de las administraciones autonómicas los términos en los que se ofertaban las asignaturas y la especificación de los contenidos de acuerdo a la realidad del centro educativo.

Por lo tanto, si se hace una comparativa entre esta ley y la anterior, se infiere que la carga lectiva de esta asignatura, muy importante para el desarrollo de la competencia digital, se reduce, aunque se mantiene intacta la optatividad de Tecnología e Informática.

En los contenidos en la etapa de Educación Primaria, vemos como en el área de Conocimiento del Medio Natural, Social y Cultural, los contenidos relacionados con el manejo y uso del ordenador y en concreto de Internet, como herramienta con diferentes aplicaciones y posibilidad van avanzando a medida que los alumnos

tienen mayor edad dentro del bloque “Objetos, máquinas y tecnologías”. No obstante, los mismos son poco desarrollados con respecto a la ley anterior, puesto que solo se hace referencia a la búsqueda guiada de información en la Red y en la edición de textos.

En el caso de la ESO, ya no se especifican los contenidos por cursos como en la LOGSE, sino que como se ha comentado, las comunidades pueden distribuirlos en los cursos que consideren.

De tal manera que Informática queda configurada con cuatro bloques de contenidos principales (anexo II, Real Decreto 1631/2006, 29 de diciembre):

- Sistemas operativos y seguridad informática.
- Multimedia.
- Publicación y difusión de contenidos.
- Internet y redes sociales.

En esta disposición se comprueba la importancia curricular que tiene la Red, y sin embargo, la materia adolece de poseer una referencia sólida a los diferentes lenguajes de programación y al desarrollo del pensamiento computacional.

Por otro lado, en la asignatura de Tecnología, los bloques de contenido relacionados con las TIC y el desarrollo tecnológico son las siguientes:

Cursos 1º a 3º:

- Hardware y sistemas operativos (Bloque 2).
- Tecnologías de la comunicación. Internet (Bloque 8).

En los mismos volvemos a ver que la importancia didáctica recae sobre Internet, su manejo, el desarrollo de la capacidad crítica para emplearlo como herramienta, fuente de información y de transmisión de conocimiento, al mismo tiempo que se hace hincapié sobre el ordenador como “máquina” con una serie de componentes tecnológicos que hay que considerar y conocer para comprender su funcionamiento.

Curso 4º:

- Tecnologías de la comunicación (Bloque 3).
- Control y robótica (Bloque 4).

En este último sí que se reseña la necesidad de conocer el funcionamiento de un robot y sus distintos componentes como actuadores, para poder programar su funcionamiento. Sin duda muy importante, pero sin diferir mucho de la ley que precede a este texto.

2.2.2.3. Ley Orgánica para la Mejora de la Calidad Educativa (LOMCE)

Al igual que en la ley anterior, la LOMCE continúa con las recomendaciones promovidas por Europa para la adquisición de competencias clave (LOMCE 8/2013, 9 de diciembre). En el Real Decreto 126/2014 del 28 de febrero por el que se establecen las enseñanzas mínimas de la etapa de Educación Primaria, se indica que las competencias clave son un conjunto de conceptos, destrezas y valores que el alumnado pone en marcha al aplicar de forma integrada los contenidos propios de cada enseñanza y etapa educativa.

A diferencia de la anterior ley, la LOMCE establece siete competencias básicas denominadas de forma diferentes:

- Competencia en comunicación lingüística
- Competencia matemática y las competencias básicas en ciencia y tecnología.
- Competencia digital.
- Aprender a aprender.
- Competencias sociales y cívicas.
- Sentido de la iniciativa y el espíritu de empresa.
- Conciencia y la expresión culturales.

La LOMCE supone una modificación parcial sobre algunos puntos clave de la LOE. Este texto legal, cuyas aplicaciones se están realizando paulatinamente en los últimos años, ha incitado a numerosos debates sobre la viabilidad del proyecto. Una de las principales críticas es la visión mercantilista de la educación, que si bien se entiende como motor de la sociedad, y como principal resorte de la misma, en este caso, y según algunos autores (Bayona Aznar, 2013) supone solamente cumplir con los índices de éxito de los informes PISA.

Tal como se anuncia en el preámbulo de la propia ley:

> La educación es el motor que promueve la competitividad de la economía y el nivel de prosperidad de un país. El nivel educativo de un país determina su capacidad de competir con éxito en la arena internacional y de afrontar los desafíos que se planteen en el futuro (LOMCE, 8/2013, de 9 de diciembre, Preámbulo, párr. 5).

Además de configurar el sistema educativo para dar respuesta a las evaluaciones externas europeas, cuya prueba objetiva son los informes PISA citados anteriormente, ha supuesto que en esta ley se incentive la formación profesional para evitar el fracaso escolar. Para lo cual, este proyecto, buscando evitar las altas tasas de abandono educativo, propone anticipar la elección de itinerarios en 3º de la

ESO, de tal manera que se puede optar por un itinerario profesional (Enseñanzas aplicadas) o académico (Enseñanzas Académicas) en este momento.

En el preámbulo de esta ley se menciona que la reforma se realiza para responder a las nuevas exigencias que demanda la sociedad del conocimiento, bien sea por el desarrollo de la creatividad como principal estandarte, o por el manejo de las tecnologías de la información y la comunicación en un mundo cada vez más interconectado, y que requiere de habilidades orientadas para estos fines.

En el Real Decreto 126/2014, de 28 de febrero, por el que se establece el currículo básico de la Educación Primaria, vemos cómo desaparece una asignatura crucial para el desarrollo de la creatividad como es la Educación Artística, que pasa a ser una asignatura optativa entre las específicas que cada Administración educativa o centro docente podrá o no ofrecer. En esta línea, la ley pretende racionalizar la oferta educativa eliminando parte de los itinerarios por áreas de conocimiento. Por ejemplo, el Bachillerato cuenta con tres modalidades, la que antes era la modalidad de Ciencias y Tecnología pasa a denominarse a Ciencias. De esta manera, las tres modalidades existentes son Ciencias, Humanidades y Ciencias Sociales, y Artes.

Como se ha comentado previamente, la etapa de Educación Secundaria es la que sufre más cambios en la configuración u organización de las asignaturas. Si en la LOE éstas se dividían en comunes y optativas, en la nueva configuración se establecen en troncales, específicas y de libre configuración autonómica. Además, el primer ciclo cuenta con tres cursos, mientras que el cuarto pasa a tener un carácter propedéutico según los itinerarios profesional o académico.

Cabe aclarar que dentro de las asignaturas específicas y de libre configuración del primer ciclo, se ofrecen un total de ocho, pero es la administración de las comunidades autónomas la que decide si las materias son ofertadas o no. Entre estas asignaturas está Tecnología y no aparece Informática entre ellas. En general, según la distribución lectiva, tienen mucha mayor carga la lengua castellana y extranjera, y matemáticas, es decir, aquellas áreas más instrumentales, y que más interesa fomentar para mejorar los resultados de cara a los informes PISA.

En el último curso de esta etapa comprobamos que la tecnología como tal adopta un carácter eminentemente orientado a las diferentes ramas profesionales, puesto que se oferta en el itinerario de Enseñanzas Aplicadas o profesional, de entre tres asignaturas. Aunque según cita el art. 14 del Real Decreto 1105/2014 del 26 de diciembre por el que se establecen las enseñanzas mínimas de la ESO y Bachillerato, queda a disposición del gobierno autonómico fijar los contenidos mismos y la optatividad de las asignaturas. En el anexo I, en la materia de Tecnología, vemos que en el bloque primero de contenidos se incluye la

introducción a conceptos básicos y lenguajes de programación, así como en el bloque de contenidos relacionado con robótica, se incluyen los lenguajes básicos de programación para controlar de forma autónoma los robots.

Hay otra asignatura que merece especial atención en la LOMCE: Tecnologías de la información y Comunicación, que es común en ambos itinerarios. Al igual que en los anteriores casos mencionados su determinación queda a merced de la administración educativa autonómica.

Si hablamos específicamente de los lenguajes de programación, la LOMCE y las leyes de las comunidades autónomas (las que deciden incluirla en su currículo), los sitúan de la siguiente manera:

- A nivel nacional, los lenguajes de programación forman parte de la asignatura optativa “Tecnologías de la Información y la Comunicación” en Bachillerato.
- A nivel autonómico, cuatro Comunidades Autónomas desarrollan contenidos adicionales relacionados con lenguajes de programación en sus currículos:
    - Cataluña: en segundo ciclo de la ESO, la programación forma parte de una asignatura optativa denominada “Informática”, cuyo itinerario formativo está focalizado en el diseño de aplicaciones para dispositivos móviles.
    - Comunidad Foral de Navarra: en 4º y 5º de Primaria se han incluido contenidos obligatorios de programación, no como asignatura separada, sino transversalmente en al área de matemáticas.
    - Comunidad de Madrid: en 1º, 2º y 3º de la ESO, los lenguajes de programación son el núcleo central de la asignatura obligatoria “Tecnología, Programación y Robótica”. En Educación Primaria existe una asignatura optativa de programación denominada “Tecnología y recursos digitales para la mejora del aprendizaje”. Se entrará más en detalle sobre esta comunidad en el siguiente apartado.
    - También cabe señalar el caso de la Comunidad Valenciana que, aunque no incluye explícitamente contenidos de programación en el currículum de la asignatura optativa “Informática” de primer ciclo de la ESO, tiene un cuerpo de profesores con una titulación específica en dicha materia (al menos un profesor de este tipo por centro), lo que viene promoviendo que, de facto, se estén impartiendo contenidos relacionados con los lenguajes de programación en la mencionada asignatura de “Informática”.

2.3. Leyes educativas actuales de la Comunidad de Madrid en relación con la enseñanza de la programación

En el caso de la Comunidad de Madrid, a raíz de la nueva configuración curricular promulgada por la LOMCE, se ha planteado una nueva asignatura denominada Tecnología, Programación y Robótica, dentro de las asignaturas de libre configuración autonómica. La impartición de esta asignatura se ha planteado para el curso escolar 2015-2016 para el primer y tercer curso de la ESO y para el año 2016-2017 para el segundo y cuarto.

En la Orden 48/2015 del 14 de mayo por el que se establece el currículo dentro de la Comunidad de Madrid, vemos formulados los contenidos de esta asignatura, y de forma directa, vemos que la misma se articula en torno a cinco ejes, uno de los cuales es “la programación y el pensamiento computacional” (Orden 48/2015, del 14 de mayo).

Además, otro dato a resaltar, es que en el primer curso de la ESO se enuncia como uno de los contenidos principales el aprendizaje de herramientas de programación por bloques. Un hito bastante importante en la enseñanza de habilidades de pensamiento computacional y para la introducción del estudiante en conceptos de programación, y como evidencia esta legislación, se plantea la importancia de la transversalidad de la programación y su aplicación práctica en ámbitos como la robótica.

En el caso de la Educación Primaria, dentro de la ley educativa, en el Decreto 89/2014 por el que se regula el contenido mínimo del currículo en esta etapa, existe una asignatura de libre configuración: “Tecnología y recursos digitales para la mejora del aprendizaje”. Los contenidos de la misma están basados en principios didácticos que se han visto también en leyes anteriores, como por ejemplo el empleo de la Red de una forma responsable, crítica y para tomar conciencia de Internet como fuente de información. Además de otros contenidos instrumentales que insisten en el conocimiento de determinadas herramientas, también encontramos un avance importante, ya que se incluye por primera vez contenidos dirigidos a introducir en el alumnado en diferentes lenguajes de programación, y además se nombra para ello una herramienta concreta, Scratch, dentro de los descriptores de esta asignatura, en el apartado denominado “Fundamentos de programación. Creación de pequeños programas informáticos (Scratch)” (Decreto Comunidad de Madrid, 2015, p. 90).

El punto fuerte de esta “innovación” dentro de la ley es que además de expresar una vía para introducir al alumnado en conceptos y procedimientos de programación, también se expresa una aplicación interesante de esta herramienta, y es la creación de juegos y aventuras gráficas (Decreto 89/2014, del 24 de julio).

3. Los lenguajes de programación

Como parte del marco teórico que sustenta la investigación, se presentan los lenguajes de programación. En este capítulo se hacer un recorrido por todo lo relacionado con este ámbito.

Primero, se comienza con el sistema binario, como base de la informática, y por ende los lenguajes de programación.

En segundo, lugar se mostrará una clasificación de los lenguajes de programación, donde se introducirán ciertos conceptos que se utilizarán a lo largo de toda la tesis.

Por último, se explicarán los fundamentos de la sintaxis de los lenguajes de programación.

3.1. El sistema binario como base de la informática

Para comprender el funcionamiento de un lenguaje de programación hay que remontarse a los principios básicos de la informática, es decir, al sistema binario. Toda información que recorre los circuitos de un ordenador se basa en secuencias de ceros y unos, o lo que es lo mismo, impulsos eléctricos o la ausencia de estos.

A Pingala, matemático hindú del siglo III a. C., se le atribuye el descubrimiento del concepto del número cero, así como la primera definición de un sistema binario de numeración (Van Nooten, 1993). A lo largo de la historia se han encontrado distintos vestigios del uso de los sistemas binarios, aunque hubo que esperar hasta el siglo XV para que se desarrollaran los conceptos teóricos en los que se basa la informática actual.

El aristócrata, científico y filósofo Francis Bacon, en 1605 planteó la teoría de un sistema que permitía traducir y asociar las letras del alfabeto, a secuencias de ceros y unos (López, 1989).

Gottfried Leibniz publica en 1703 el artículo *Explication de l'Arithmétique Binaire*, donde documenta y sienta las bases del sistema binario vigente hoy en día.

En su libro *The Laws of Thought* el matemático británico George Boole (1854), presenta lo que hoy en día se conoce como Álgebra de Boole, un sistema de lógica binaria fundamental en la informática actual, en el que se basan los circuitos digitales, y por tanto la estructura de los ordenadores.

Los estudios realizados por estos y otros científicos han sentado las bases del código binario, a través del cual, números, letras, operaciones matemáticas complejas… pueden ser representadas con cadenas de ceros y unos. Podemos entender cómo esto es posible partiendo de las bases que tenemos asentadas y que nos son cercanas (números decimales, alfabeto latino, etc.) y su relación con el sistema binario.

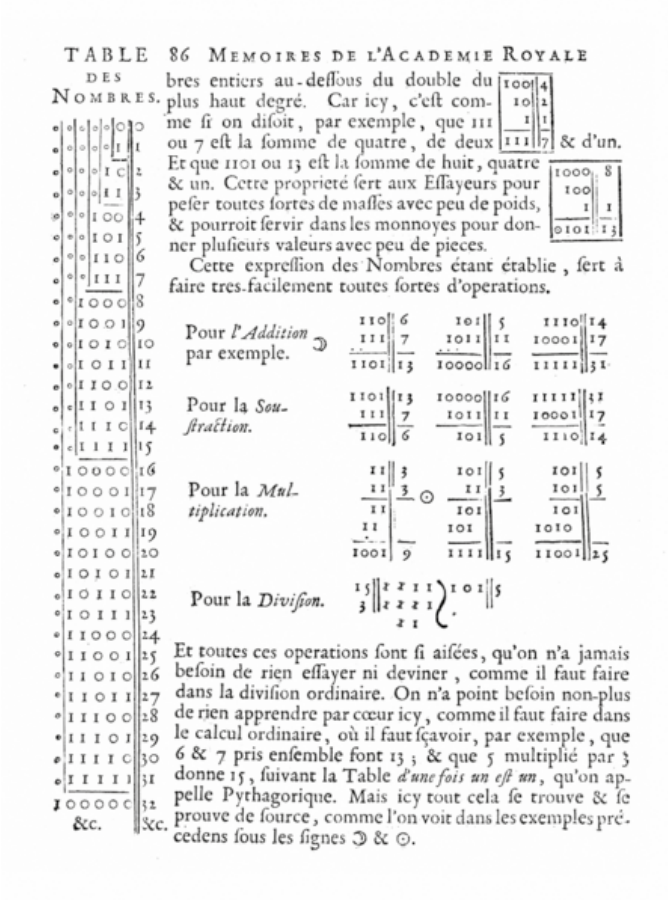


TABLE DES NOMBRES.

86 MEMOIRES DE L'ACADEMIE ROYALE

bres entiers au-deſſous du double du plus haut degré. Car icy, c'eſt comme ſi on diſoit, par exemple, que 111 ou 7 eſt la ſomme de quatre, de deux & d'un. Et que 1101 ou 13 eſt la ſomme de huit, quatre & un. Cette proprieté ſert aux Eſſayeurs pour peſer toutes ſortes de maſſes avec peu de poids, & pourroit ſervir dans les monnoyes pour donner pluſieurs valeurs avec peu de pieces.

Cette expreſſion des Nombres étant établie, ſert à faire tres-facilement toutes ſortes d'operations.

Pour *l'Addition* par exemple.

Pour la *Souſtraction.*

Pour la *Multiplication.*

Pour la *Diviſion.*

Et toutes ces operations ſont ſi aiſées, qu'on n'a jamais beſoin de rien eſſayer ni deviner, comme il faut faire dans la diviſion ordinaire. On n'a point beſoin non-plus de rien apprendre par cœur icy, comme il faut faire dans le calcul ordinaire, où il faut ſçavoir, par exemple, que 6 & 7 pris enſemble font 13; & que 5 multiplié par 3 donne 15, ſuivant la Table *d'une fois un eſt un*, qu'on appelle Pythagorique. Mais icy tout cela ſe trouve & ſe prouve de ſource, comme l'on voit dans les exemples précedens ſous les ſignes ☽ & ⊙.

&c. &c.

Figura 2. Página del artículo Explication de l'Arithmétique Binaire de Leibniz.

Fuente: (Leibniz, 1703)

3.1.1. Sistema binario – sistema decimal

Los sistemas numéricos actuales, en su mayoría, son sistemas ponderados, es decir, en una secuencia de dígitos cada posición tiene un peso asociado (Angulo Usategui, García Zubía y Angulo Martínez, 2007, p. 38). En el sistema decimal que utilizamos cada día disponemos de diez dígitos (los números entre el 0 y el 9), y

combinándolos entre ellos podemos construir cualquier número. La posición que ocupa cada dígito tiene un peso distinto, por lo que si cogemos dos números cualquiera, (por ejemplo, el 1 y el 9) obtendremos una cifra distinta en función de la disposición de cada uno de ellos (19 o 91). Por ser un sistema decimal, el peso de cada posición se obtendrá partiendo del 1 y multiplicando por 10 el peso de la posición anterior. Esto es, comenzando por la derecha, la primera cifra tiene un peso de 1, la segunda de 10, la tercera de 100, y así sucesivamente.

Peso:	100	10	1
Dígito:	2	3	5

235 = 2x100 + 3x10 + 5x1

El sistema binario, de la misma manera que el decimal, es un sistema de numeración posicional ponderado. En este caso, el hecho de ser binario hace que el peso de la posición de cada dígito se obtenga partiendo del 1 y multiplicando por 2 el peso de la posición anterior.

De esta forma podemos entender que el número binario 11010010 se corresponde con el número 210 en decimal. Se puede representar de la siguiente manera:

Peso:	128	64	32	16	8	4	2	1
Dígito:	1	1	0	1	0	0	1	0

Teniendo en cuenta los pesos, se suman los números 128, 64, 16 y 2, de tal manera que:

$11010010_2 = 128 + 64 + 16 + 2 = 210_{10}$

3.1.2. El alfabeto latino y el sistema binario

De la misma manera que entendemos que cualquier número en decimal tiene su equivalencia en binario, podemos entender también que es posible extrapolar estos mismos principios a las letras y símbolos del alfabeto latino. El código ASCII sienta las bases de esta conversión (Gorn, Bemer y Green, 1963).

ASCII (*American Standard Code for Information Interchange* — Código Estándar Estadounidense para el Intercambio de Información), es un código de caracteres que establece valores numéricos para un total de 95 caracteres imprimibles, junto con otros 32 caracteres no imprimibles, tales como el retorno de carro y otros caracteres de control que se usan a nivel interno en informática.

En la Tabla 2 se puede ver una muestra de algunos de los 95 caracteres imprimibles antes mencionados, y la representación numérica, tanto en el sistema binario como en el decimal, que establece el código ASCII.

Tabla 2
Ejemplos de símbolos imprimibles en Código ASCII, y sus equivalencias en el sistema binario y decimal.

Binario	Decimal	Símbolo	Binario	Decimal	Símbolo	Binario	Decimal	Símbolo
0010 0000	32	espacio ( )	0100 0000	64	@	0110 0000	96	`
0010 0001	33	!	0100 0001	65	A	0110 0001	97	a
0010 0010	34	"	0100 0010	66	B	0110 0010	98	b
0010 0011	35	#	0100 0011	67	C	0110 0011	99	c

Fuente: Elaboración propia.

La ASA (Agencia de Estándares Estadounidense), que posteriormente pasaría a ANSI (Instituto Nacional Estadounidense de Estándares) hace público el código ASCII en 1963. Desarrollado inicialmente en el ámbito de la telegrafía, hoy en día es utilizado en prácticamente todos los ordenadores.

Como se ha mencionado antes, toda información que recorre los circuitos de un ordenador se basa en secuencias de ceros y unos, o lo que es lo mismo, impulsos eléctricos o la ausencia de estos. Entendiendo que podemos convertir cualquier palabra, número u operación matemática a código binario, y con esta base construir tareas más complejas, se podría concluir que una forma de programar un ordenador es introducir la secuencia adecuada de ceros y unos para el propósito que se tenga. No es difícil imaginar la dificultad y las limitaciones que puede entrañar el hecho de programar un ordenador a base de ceros y unos, es decir, en el denominado "lenguaje máquina" (Quero Catalinas, 2002, p. 128) que el ordenador entiende. Es por esto que los lenguajes de programación han ido evolucionando, buscando una mayor funcionalidad, versatilidad y cercanía al lenguaje humano.

3.1.3. Evolución de los lenguajes de programación: del código binario a los lenguajes de alto nivel

El desarrollo de los ordenadores y los lenguajes de programación han ido de la mano, aunque a ritmos diferente. Los ordenadores han evolucionado siguiendo la premisa de obtener máquinas cada vez más pequeñas, más rápidas y con mayor capacidad de procesamiento (Bergin, 2007).

La primera generación de computadoras (1951 al 1958) utilizaba tarjetas perforadas para los procesos de lectura y escritura. Estas tarjetas contenían información en forma de perforaciones según el código binario. La segunda y tercera generación de ordenadores (1959 – 1964 y 1965 – 1971 respectivamente) consiguieron máquinas más pequeñas, más rápidas y más eficientes energéticamente con inventos como el transistor en 1958 y con la aparición posterior de los circuitos integrados (Asensi Artiga, 1993). Sin embargo, el mecanismo de programación en los computadores de 2ª y 3ª generación seguían estando basados en tarjetas perforadas.

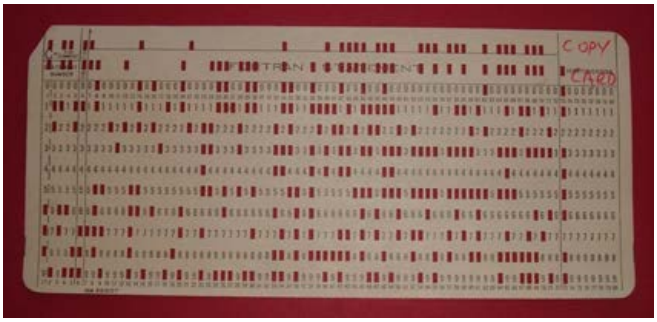


Figura 3. Tarjeta perforada utilizada en los ordenadores de principios de los 70.

Fuente: (Reinhold, 2006)

La cuarta generación de computadoras (1971 hasta la fecha) supuso la mayor revolución en el ámbito de la informática, tanto a nivel de hardware, como de software y la forma de programarlo. Luis Arroyo (1991) caracteriza esta cuarta generación por la sustitución en la memoria de los núcleos de ferrita por los chips de silicio como material de base, la microminiaturización de los componentes y circuitos integrados, y la aparición de la pantalla como interfaz con el usuario. Todos estos factores posibilitaron la aparición del PC (sigla en inglés de *personal computer*), es decir, ordenadores diseñados para ser utilizados por una sola persona.

La abundancia del silicio, su fácil extracción, y la producción en masa de microprocesadores basados en este material, “abarató los costes de producción” (Patterson y Hennessy, 2013, p. 23), e hizo que el ordenador personal fuera

asequible para el público en general, y utilizado de igual manera en el hogar y en contextos industriales. Los PCs también fueron concebidos para poder ser utilizados por usuarios sin conocimientos técnicos, en contraposición con las computadoras de la 1ª generación que precisaban de conocimientos electrónicos concretos.

La forma en la que el ordenador personal se extendió a toda la población, condicionó también la evolución de los lenguajes de programación. Ordenadores con mayor capacidad de proceso precisaban de un software más complejo, que a su vez necesitaba ser desarrollado con lenguajes de programación más sofisticados. Y la generalización del uso del ordenador por cada vez más personas, hizo que creciera de igual manera la demanda de aprender a programarlos, y con ella la búsqueda de formas programación cada vez más simples y más asequibles.

3.2. Clasificación de los lenguajes de programación

Los lenguajes de programación tienen varios criterios de clasificación (Forouzan, 2003; Quero Catalinas, 2002). Como se ha visto anteriormente en la evolución de los lenguajes de programación, una forma de clasificarlos es por su grado de evolución y ***su cercanía al lenguaje natural***. Según este criterio tenemos:

- Lenguajes de Bajo nivel
- Lenguajes de Alto nivel

Más adelante se profundizará en estos dos conceptos, pero se puede afirmar que cuanto más cercano esté el lenguaje de programación al lenguaje natural, de mayor nivel se considerará. Según sea el proceso de traducción desde lo que escribe el programador hasta lo que entiende la máquina, o lo que es lo mismo, el proceso por el que pasa un lenguaje de alto nivel hasta ser traducido a bajo nivel da lugar a una segunda clasificación que podríamos denominar tipos de lenguajes ***según la manera de traducir y ejecutar el código:***

- Lenguajes Ensamblados
- Lenguajes Compilados
- Lenguajes Interpretados
- Lenguajes Preprocesados

Los lenguajes de programación de alto nivel son los más extendidos y utilizados. Es por esto que se pueden establecer con este tipo de lenguajes una tercera

clasificación ***en función del estilo de programación y la forma de plantear la resolución del problema***:

- Lenguajes Procedurales.
- Lenguajes Declarativos.
- Lenguajes Funcionales o Aplicativos
- Lenguajes Especiales
- Lenguajes Orientados a Objetos.

Podemos hacer una cuarta clasificación ***en función del ámbito de aplicación*** y el tipo de software para el que se usan:

- Lenguajes de aplicación científica.
- Lenguajes de proceso de datos y gestión de información.
- Lenguajes orientados a la inteligencia artificial.
- Lenguajes para programación de sistemas.
- Lenguajes para aplicaciones Web.

Por último, podemos clasificar a los lenguajes por su **estilo de codificación**:

- Textuales
- Visuales

3.2.1. Según su cercanía al lenguaje natural

La evolución de los lenguajes de programación se ha basado en buscar la forma de transmitir gran cantidad de información en un formato reducido, y en que además esta forma de transmisión sea cada vez más cercana al lenguaje natural. En este camino se han ido desarrollando diversos sistemas de traducción para acercar el sistema binario que “entiende” el ordenador al lenguaje natural que “entiende” el ser humano.

De esta manera, un lenguaje de programación se considera de bajo nivel cuanto más cerca esté del lenguaje de la máquina, y de alto nivel cuanto más cerca esté del lenguaje natural. El hecho de que un lenguaje se considere de bajo nivel no significa que dicho lenguaje sea inferior en prestaciones a otro de alto nivel, se refiere a la proximidad entre el software y el hardware.

Bajo nivel.

- ***1ª generación: código máquina***

El sistema binario, que como en apartados anteriores se ha mencionado, también es conocido como *lenguaje máquina*, se encuentra en el más bajo de los niveles. El fabricante del hardware establece los bloques de ceros y unos que reconocerá la CPU (unidad central de procesos), y la habilitará para poder llevar a cabo una serie de operaciones.

Dado que el código máquina solo utiliza el sistema binario como forma de comunicación, su organización está por completo supeditada a la circuitería de la computadora, y por tanto alejada de cualquier forma de expresión humana. Es decir, escribir la solución de un problema en código máquina se hace de forma poco intuitiva, y obliga al programador a dominar la arquitectura del ordenador sobre el que programa. Sin embargo, el código máquina pone a disposición del programador todos los recursos del ordenador, consiguiendo una gran eficiencia en el resultado, si se mide esta eficiencia en cuanto a los recursos de memoria utilizados, y en cuanto al tiempo de ejecución.

- ***2ª generación: ensamblador***

Como se ha visto en capítulos anteriores, se pueden fijar mecanismos de traducción a través de los cuales una secuencia de ceros y unos puede simbolizar números, letras, símbolos… Con este mismo principio, y buscando esa cercanía con el lenguaje humano, se crea el lenguaje de ensamblador, en el que se asocian conjuntos de palabras de fácil memorización con funciones del código máquina. Un programa llamado *Ensamblador* se encarga de realizar esta interpretación. Algunos ejemplos de instrucciones de ensamblador son ADD (sumar sin reserva), ADC (sumar con reserva o "*addwithcarry*"), *M* (producto o “*multiply*”), etc. (Godfrey, 1991, p. 20).

El uso del Ensamblador se considera como la primera tentativa de acercamiento entre el lenguaje máquina y el lenguaje humano (Regan, 2008), dado que se consiguen representaciones y direccionamientos simbólicos de las operaciones. Además se introduce el concepto de comentario (líneas de texto dentro del programa que el ordenador ignora, y que sirven al programador para incluir sus anotaciones) con lo que el código es mucho más legible.

Sin embargo, las dificultades que presentaba el código máquina se mantienen. Para realizar un programa sigue siendo necesario un conocimiento profundo de la arquitectura física de la máquina sobre la que se trabaja, dada la total dependencia

entre el *hardware* y el *software.* Esto unido a que el número de instrucciones es reducido, y su posibilidad de combinación poco flexible, hace que la tarea del programador sea aún difícil y tediosa. De igual manera, se mantiene la ventaja de poder explotar al máximo los recursos del ordenador y de conseguir programas muy eficientes.

Para paliar de alguna manera las restricciones del lenguaje ensamblador, algunos fabricantes, como IBM, crearon los denominados macroensambladores (Godfrey, 1991, p. 6). Estos macroensambladores manejan un lenguaje que asocia cada instrucción a varias secuencias o acciones del código máquina, a diferencia de la relación que uno a uno que presenta el ensamblador tradicional. Este principio de asociar varias instrucciones a una es en el que se sustentan los lenguajes de programación de más alto nivel.

El lenguaje de ensamblador se sigue utilizando hoy en día para pulir y mejorar programas desarrollados con otros lenguajes de alto nivel, y dar instrucciones concretas en aquellos aspectos donde se necesite un mayor control del *hardware*.

Alto nivel.

- ***3ª generación***

Como se ha mencionado, la evolución de los lenguajes ha estado basada en desarrollar instrucciones que agruparán varias acciones que, con un lenguaje de bajo nivel, había que detallar una por una. Estas instrucciones de alto nivel serían trasladadas al ordenador a través de Compiladores o de Intérpretes (Ortega de la Puente, Alfonseca Moreno, de la Cruz Echeandía y Pulido, 2006, pp. 319 - 322). A muy grandes rasgos, las principales características de uno y otro se muestran a continuación:

- Un Compilador lee y analiza por completo el código programado, traduciéndolo globalmente para generar un ejecutable para un tipo de ordenador específico, no siendo trasladable dicho ejecutable a otras plataformas. El ejecutable generado y el código del programa se convierten en entes independientes el uno del otro. C, PASCAL, COBOL o BASIC se encontrarían dentro de esta categoría.
- Un Intérprete lee línea por línea el código, traduciéndolo y ejecutándolo en el momento. No se genera un ejecutable, por tanto el código deberá estar presente en cada ejecución, y podrá ser utilizado en cualquier plataforma que tenga ese intérprete. JavaScript es un ejemplo de este tipo de lenguajes.

Los lenguajes de 3ª generación amplían el nivel de abstracción del diseño del programa con respecto a generaciones anteriores, y suelen ser lenguajes procedurales o imperativos, es decir, lenguajes que estructuran el programa en una secuencia de acciones que se ejecutan una detrás de otra en el orden adecuado. En apartados posteriores se amplía el concepto de lenguaje imperativo.

- ***4ª generación: 4GL***

James Martin (1982) es el primer autor en acuñar el término 4GL para referirse a los lenguajes de 4ª generación.

No existe un consenso absoluto con respecto a las especificaciones de los 4GL, pero diversos autores (Solano Mata, Yong Morales y Camacho Brenes, 2007) los describen como lenguajes no procedurales que buscan aún más cercanía con el lenguaje humano (principalmente el inglés), y tiempos de desarrollo más cortos. Se utilizan dentro de un *software* marco previamente desarrollado donde el usuario/programador, en lugar de programar el proceso a realizar, describe con la sintaxis de ese *software* el resultado que quiere obtener.

Estos mismos autores clasifican los lenguajes de 4ª generación en diferentes categorías:

- Generadores de informes a partir de datos. Por ejemplo, SQL entraría dentro de esta categoría.
- Generadores de formularios. En este apartado se podrían clasificar entornos como .NET donde que habilita un marco de trabajo el que el programador define las ventanas que tendrá el programa que se pretende realizar, y diseña la interacción que tendrá el usuario final.
- Entornos de 4ª generación, como las herramientas CASE para el análisis y diseño de sistemas, o las herramientas de generación de código de forma semiautomática.
- Administración de datos. Programas como SPSS (IBM), utilizados en investigación y análisis de datos, serían un ejemplo de un generador de informes.
- Generadores de aplicaciones. Son entornos de desarrollo que incorporan funcionalidades predefinidas con un propósito concreto. Por ejemplo, dentro de esta categoría estarían programas como Unity o Unreal Engine, concebidos para desarrollar videojuegos, que incorporan dentro de ellos utilidades relacionadas con este fin: manejo de gráficos 2D y 3D, implementación de físicas, etc.

3.2.2. Según la manera de traducir y ejecutar el código

En esta clasificación se incluyen conceptos descritos previamente en apartados anteriores, por lo que en este capítulo simplemente se presentarán brevemente. Según la manera de traducir el código realizado por el programador a algo comprensible por la máquina, y la forma de ejecutar el resultado, podemos clasificar los lenguajes en las siguientes categorías:

Lenguajes Ensamblados: Constan de una serie de instrucciones asociadas a códigos en lenguaje máquina, en una relación de uno a uno (cada instrucción está asociada a un código). Un programa denominado Ensamblador se encarga de llevar a cabo la traslación del lenguaje ensamblado al código máquina.

Lenguajes Compilados: Lenguajes por lo general de alto nivel que son procesados en bloque, generando a través del compilador un ejecutable independiente del código fuente, pero dependiente de la plataforma para la que es generado.

Lenguajes Interpretados: Lenguajes en los que las instrucciones son analizadas y ejecutadas de forma procedural por el programa intérprete. Código e intérprete deben ir de la mano y estar presentes en cada plataforma en la que se quiera ejecutar el programa.

Lenguajes Preprocesados: son lenguajes previamente traducidos a un nivel intermedio, y posteriormente ejecutados en la plataforma de destino donde se encuentra instalado un *software* que actúa de base para el programa. Un ejemplo de este tipo de lenguajes es Java. Para ejecutar una aplicación desarrollada en Java primero se necesita hacer un proceso de compilación previo (lo que hemos denominado antes como traducción a un nivel intermedio de más bajo nivel). Además de esto, el ordenador final donde vaya a funcionar la aplicación deberá tener instalado un programa concreto denominado Máquina Virtual de Java. La Máquina Virtual de Java se encargará del último nivel de traducción, el que haga referencia al hardware específico y al sistema operativo. La ventaja de estos lenguajes radica en su portabilidad (un mismo código puede servir para cualquier ordenador y cualquier sistema operativo, dado que de esto se encarga la máquina virtual). Esto no sucede por ejemplo con los lenguajes compilados, donde el compilador realizará la traducción y generará un archivo ejecutable para la máquina concreta.

3.2.3. Según el estilo de programación y la forma de plantear la resolución de un problema

Lenguajes procedurales o imperativos: Establecen la resolución de un problema en una serie tareas secuenciales, que a su vez se dividen en partes más pequeñas o subtareas. Estos lenguajes cuentan además con estructuras de control condicionales e iterativas. Las estructuras condicionales permiten que dentro de los pasos que llevan a la resolución del problema se puedan tomar un camino u otro en función de si se cumple una determinada condición. Las estructuras iterativas permiten repetir una o varias instrucciones un número determinado de veces en función también de una condición. Otros tipos de lenguajes como los declarativos, orientados a objetos, etc. incorporan dentro de ellos todos los conceptos que manejan los lenguajes procedurales. Ejemplos de lenguajes de este tipo son: C, COBOL PASCAL, etc.

Lenguajes declarativos: Están basados en las matemáticas y en la lógica, y se encuentran alejados del lenguaje natural. Al utilizar estos lenguajes, el programador no tiene que especificar al detalle el proceso de cómo llevar a cabo cada tarea, sino más bien qué tareas tiene que hacer el programa. Para entender este tipo de lenguajes se puede buscar el símil de utilizar una calculadora: el usuario de ésta sabe qué tipo de operaciones puede hacer (sumas, multiplicaciones, raíces cuadradas…) y el orden en el que las quiere hacer, pero no tiene que especificar cómo se realiza cada una de esas operaciones. Las instrucciones que se le da a una hoja de cálculo podrían entrar en la categoría de lenguajes declarativos.

Lenguajes funcionales o aplicativos: Están diseñados para resolver problemas en ámbitos específicos, y suelen ser generadores de aplicaciones, principalmente de gestión, que permitan automatizar ciertos procesos. Podemos entender lo que es un lenguaje de programación funcional sabiendo que las hojas de cálculo en general, y concretamente la sintaxis que se utiliza para especificar que una celda realice una determinada operación, se consideran lenguajes de programación funcional. Este tipo de lenguajes han sido utilizados sobre todo en entornos de aprendizaje y académicos, como por ejemplo *Mathematica* (Wolfram, 2017), aunque también se pueden encontrar ejemplos de este tipo de lenguajes en el entorno empresarial, por ejemplo *F#* (2012) o *XSLT* (W3C, 1999).

Lenguajes Especiales: son lenguajes que no se utilizan para la resolución de problemas, ya que están circunscritos a un ámbito específico. Por ejemplo, dentro de esta categoría están el lenguaje de SQL, destinado a extraer información concreta de una base de datos a través de consultas, o el lenguaje de HTML, utilizado para la maquetación de contenidos Web.

Orientados a objetos: El uso de este estilo de programación se basa en acercar el diseño del programa a la vida real, y a generar códigos fáciles de modificar y corregir, de escalar, y de reutilizar en otros programas.

En este paradigma, además de determinar las acciones a realizar y el orden de éstas como se hace en la programación imperativa, también se establecen las entidades que las van a llevar a cabo. Estas entidades son elementos relevantes dentro del programa que el paradigma de la programación orientada a objetos define de forma abstracta en forma de lo que se denomina *Clase*. Esta definición de Clase viene dada por la identificación sus atributos o propiedades característicos, y por la asignación de las acciones (métodos) que podrá hacer.
Las Clases son una especie de molde con el que se podrán crear Objetos, es decir, representaciones concretas de esa clase, con valores asignados a las distintas propiedades o atributos. En la ejecución del programa se crearán tantos objetos como se precise, y todos ellos tendrán el comportamiento que de forma genérica se define en su clase asociada.

Lenguajes como C++ o Java siguen el paradigma de la Programación Orientada a Objetos.

3.2.4. Según su campo de aplicación

Lenguajes de aplicación científica: Lenguajes que manejan sintaxis propias de operaciones y algoritmos matemáticos. Ejemplos de lenguajes que se podrían incluir en este contexto son FORTAN y PASCAL**.**
Lenguajes de proceso de datos y de gestión de información: Lenguajes utilizados tanto en el manejo de ficheros (apertura, lectura / escritura, cierre, etc.), como en la gestión de sistemas de Bases de Datos (consulta, inserción, borrado, modificación de datos). SQL es un ejemplo de lenguajes de este tipo.

Lenguajes orientados a la inteligencia artificial: Lenguajes utilizados con el propósito de programar sistemas expertos, videojuegos, robótica, y en general cualquier desarrollo que involucre comportamientos autónomos del programa, determinados por una inteligencia artificial. LISP y PROLOG son lenguajes que habitualmente se categorizan en este apartado, aunque realmente cualquier lenguaje de alto nivel es apto para el desarrollo de IA.

Lenguajes para programación de sistemas: Lenguajes utilizados para la codificación de *software* (sistemas operativos, compiladores, intérpretes…) que actúe como interfaz entre el usuario y el *hardware*. Aunque en un origen se utilizaba Ensamblador para realizar este tipo de programas, hoy en día existen lenguajes de

alto nivel que son mucho más aptos. Por ejemplo Unix, (base de otros sistemas operativos como las distribuciones de Linux, Android, o sistemas IOS de Apple) está íntegramente programado con el lenguaje C (Ritchie y Thompson, 1978).

3.2.5. Según su estilo de codificación

Lenguajes textuales: Lenguajes en el que el programador debe codificar el programa a través de texto siguiendo la sintaxis del lenguaje de programación utilizado. La mayoría de lenguajes de aplicación profesional (Java, C++, C#, etc.) se encuentran dentro de esta categoría.

```
1 using System;
2
3 namespace TextualProgrammingLanguageExample
4 {
5     class MainClass
6     {
7         public static void Main (string[] args)
8         {
9             Console.WriteLine ("Hello World!");
10        }
11    }
12 }
```

Figura 4. Ejemplo de código escrito con C#, un lenguaje textual.

Fuente: elaboración propia.

Lenguajes visuales: el programador no tiene que escribir código, en vez de esto arrastra y suelta figuras, colocándolas en el lugar y en el orden correcto.

Dentro de los lenguajes visuales podemos encontrar una tipología en el que las figuras tienen forma de bloques que encajan entre sí, como si de un juego de construcción se tratara. Estos bloques representan las estructuras de programación, y las acciones que se pueden realizar dentro del programa Cada bloque tiene una forma diferente, y hay ciertas piezas que se pueden unir entre ellas, y otras no. Encajando las piezas donde la unión es posible se construyen estructuras de programación sintácticamente correctas. Scratch o Blockly son un ejemplo de este tipo de lenguajes.

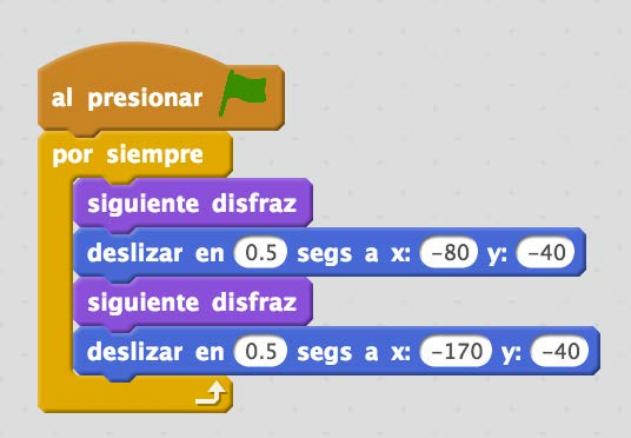


Figura 5. Ejemplo de código desarrollado con Scratch, un lenguaje visual.

Fuente: elaboración propia.

También podemos encontrar lenguajes visuales en el que las figuras son flechas e iconos fácilmente reconocibles. Un ejemplo de este tipo de lenguajes es Lightbot, en el que a través de la colocación de las flechas se le puede dar las instrucciones a un pequeño robot autómata para que recorra un camino hasta llegar a la meta. Otros ejemplos de este tipo de lenguajes son *ScratchJr*, o *The Foos*. Este tipo de lenguajes son aptos incluso para niñas y niños que están en las primeras etapas de aprendizaje de la lectoescritura.

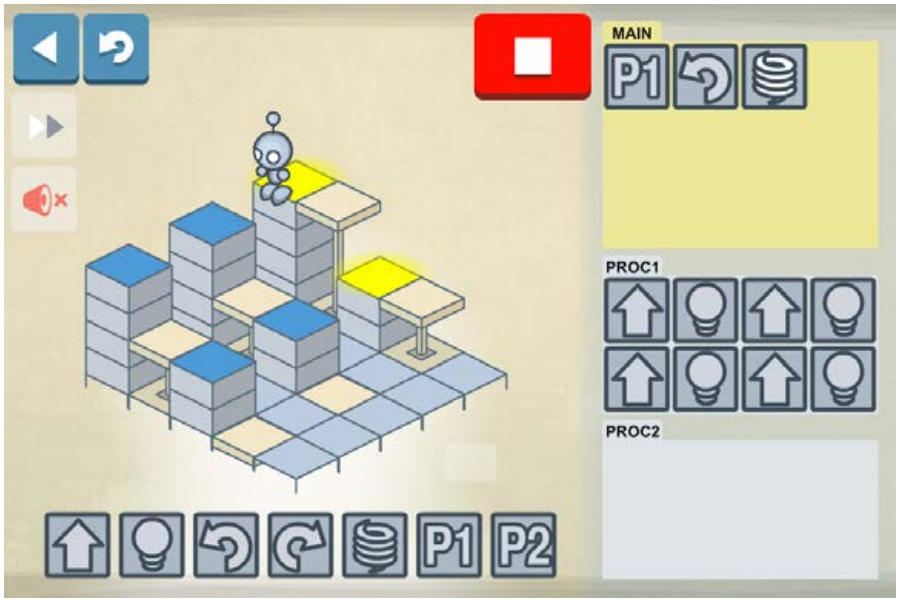


Figura 6. Ejemplo de programa elaborado con Lightbot, a través de flechas e iconos.

Fuente: (Lightbot Inc, 2015).

3.3. Sintaxis de los lenguajes de programación

Aprender a escribir en cualquier lenguaje (castellano, inglés, árabe, chino…) implica por un lado conocer la sintaxis y la simbología que utiliza (el abecedario), y por otro saber cómo colocar estos símbolos para que tengan significado y expresen el concepto deseado (semántica).

Como se ha mencionado en el capítulo 3.2, existen diferentes paradigmas de programación, y cada uno tiene sus propios principios y reglas. Estos principios determinarán cómo es la sintaxis de los lenguajes de programación que se enmarquen dentro de cada paradigma.

De todos los paradigmas existentes, este apartado se va a centrar en la programación imperativa o estructurada. El motivo de esta elección es múltiple. Por ejemplo, es bastante común que en los cursos de introducción a la programación se utilice como instrumento de aprendizaje un lenguaje de programación estructurado. Asimismo, es un paradigma que está presente en los lenguajes de programación de uso profesional más populares, (por ejemplo, C++ o Java, siguen un paradigma híbrido donde se junta programación orientada a objetos y programación imperativa). Pero la razón fundamental para centrarse en este modelo es que Scratch, que es el objeto de estudio de esta tesis, sigue este tipo de paradigma (2016c).

El modelo de programación imperativa o estructurada se basa expresar en forma de programa la solución a un problema, a través del método algorítmico y heurístico. Aunque este tema se desarrolla en profundidad más adelante, en el capítulo 4.2, podemos adelantar que un algoritmo es la especificación de una serie de tareas a realizar para resolver un problema, así como el orden en el que deben realizarse.

Knuth (1997) identifica los rasgos más importantes de un algoritmo:

- Tiene una entrada (definición y acotación del problema).
- Tiene una salida (solución al problema).
- Es finito, tiene un número limitado de tareas para resolver el problema.
- Es definible, cada tarea a realizar se puede expresar de forma clara, concisa y sin ambigüedades.
- Es eficiente, midiendo la eficiencia en términos relativos al tiempo que se tarda en resolver el problema con este método, el tiempo en el que tarda en expresar la solución, los recursos que se consumen en el proceso, entre otros.

Por tanto, programar según el paradigma imperativo es básicamente conocer la forma de plantear el algoritmo que resuelve un problema en los términos que un ordenador puede entender, y esto es independiente del lenguaje de programación. En una segunda fase, el lenguaje de programación establecerá la sintaxis con la que traducir ese algoritmo. Es decir, cuando un aprendiz de programador se inicia en esta ciencia, se encuentra la dificultad doble de adquirir por un lado este pensamiento algorítmico, y por otro, la sintaxis propia del lenguaje en particular.

Desde esta perspectiva, un lenguaje de programación constituye el conjunto de símbolos, expresiones y reglas de sintaxis que permiten expresar en los términos adecuados un algoritmo. Pero los lenguajes de programación no son la única herramienta para representar un algoritmo, también se puede expresar a través del lenguaje natural, de diagramas de flujo, o de pseudocódigo. Todas estas formas de expresión servirían para cumplir el principio de definibilidad del que hablaba Knuth (1997).

La representación de un mismo algoritmo varía según el lenguaje o método de expresión que se utilice, incluso puede cambiar si cambia el programador, a pesar de haber utilizado el mismo instrumento. Pseudocódigo, diagrama de programación, diagrama de flujo… todos tendrán su propia forma de expresión, lo que se denomina notación algorítmica.

Estas notaciones algorítmicas, a pesar de su idiosincrasia particular, comparten ciertos conceptos. Diversos autores (Brennan y Resnick, 2012; Martínez Morales y Rosquete De Mora, 2009; Ramírez, 2015; Rodríguez Sala, 2003) han realizado propuestas de estandarización de la notación algorítmica. Este apartado de la tesis reúne algunas de las conclusiones obtenidas en estas investigaciones, y pretende mostrar una referencia de los conceptos comunes que están presentes en cualquier lenguaje que siga el paradigma de la programación imperativa, y que se deberán conocer si se pretende programar según este modelo.

3.3.1. Principios básicos de la computación en los que se basa la notación algorítmica

La idea fundamental de programa nace del *Entscheidungsproblem*, traducido en castellano como el problema de decisión, que David Hilbert y Wilhelm Ackermann plantean en 1928 cuestionando si la matemática es completa, consistente y decible (Hilbert y Ackermann, 1975). O dicho de otra manera, Hilbert y Ackermann se preguntaban si existe una forma de calcular cualquier función matemática sobre los números naturales y de expresar ese cálculo mediante un algoritmo que puede calcular la función. Si ese algoritmo existiera, cualquier problema que estuviera bien

definido se podría resolver con tan solo ejecutar dicho algoritmo. A continuación se muestra un ejemplo:

> Formula lógica sobre naturales: P(n) es cierta o falsa. Por ejemplo, esPar(n)
> Función matemática sobre naturales: f : N -> N
> P(n) es cierta/falsa sii f(n) = 0/1
> f(n) = 0 si n%2 == 0
> f(n) = 1 en otro caso

Esta cuestión sirvió de punto de partida a estudiosos de la lógica de la época, como Alan Turing y Alonzo Church. Ambos crearon un modelo formal de cómputo, con un lenguaje y una gramática propios, con el propósito de poder formular los cálculos que llevan a la resolución de una función matemática, y así poder responder a la pregunta que plantea el *Entscheidungsproblem*. La propuesta de modelo de Turing fue la denominada Máquina de Turing (Turing, 1937), y en el caso de Church, el llamado Cálculo Lambda (Church, 1932).

Turing y Church, llegaron por caminos diferentes a las mismas conclusiones:

- Una función es computable si existe una función de Cálculo Lambda que la computa (Church, 1936b).
- Una función es computable si existe una Máquina de Turing que la computa (Turing, 1937).
- Las funciones que son computables por el Cálculo Lambda también lo son por la Máquina de Turing. Cualquier función que pueda ser formalmente expresada a través del Cálculo Lambda tiene su equivalente en la Máquina de Turing (Turing y Copeland, 2004, p. 44). La tesis de Church-Turing plantea la hipótesis de que “todo lo necesario para expresar formalmente un algoritmo se puede encontrar en el modelo de la Máquina de Turing” (Turing y Copeland, 2004, p. 577), estableciendo así la equivalencia entre algoritmo y función Turing-computable (computable por una máquina de Turing).
- Existen funciones que no son computables por la máquina de Turing, y por ende, por ninguno de sus equivalentes (Turing, 1937).

De esta forma ambos dan una respuesta negativa al *Entscheidungsproblem* (Church, 1936a; Turing, 1937), dando lugar a la tesis de Church-Turing (Turing y Copeland, 2004), que propone que no existe el algoritmo que pueda aseverar si la proposición planteada por una función es cierta o no. La tesis de Church-Turing no es un teorema, solo es una afirmación. Que existan funciones cuyo cálculo no pueda expresarse formalmente a través de la máquina de Turing no significa que no exista otro sistema formal que sí pueda expresarlo. En cualquier caso, la tesis de

Church-Turing tiene una amplia aceptación, y ha sido respaldada por otros estudiosos de la materia como Gödel que afirmó que cualquier sistema formal que contenga los axiomas de la aritmética es inconsistente o incompleto. Si es consistente, dicha consistencia no podrá ser demostrada dentro del propio sistema formal, por lo tanto el sistema formal es incompleto (Gödel, 2016).

Un lenguaje es Turing-completo si para todos los algoritmos que computan funciones matemáticas escritos en ese lenguaje, existe una máquina de Turing-equivalente al algoritmo (Turing y Copeland, 2004).

El teorema del programa estructurado de Böhm y Jacopini (1966), y el teorema de Dijkstra (1968), basados en planteamientos anteriores como la tesis de Church-Turing, establecen que todo algoritmo puede describirse utilizando solamente tres tipos de instrucciones:

- Secuencia de acciones (expresiones aritméticas, asignación, entrada/salida de datos sin límite de almacenamiento, etc.) llevadas a cabo en orden, una detrás de otra.
- Sentencias condicionales, que establecen caminos alternativos para el algoritmo, según se responda a una pregunta o condición (SI se cumple la condición EJECUTA la acción 1, SI NO se cumple la condición EJECUTA la acción 2)
- Bucles, que establecen la realización de una determinada tarea mientras se cumpla una determinada condición. La tarea se dejará de realizar cuando la condición deje de cumplirse.

Todas estas teorías, a pesar haberse planteado en la primera mitad del siglo XX, tienen plena vigencia (Wolfram, 2002). Estos tres tipos de instrucciones están presentes en cualquier lenguaje que sigue el paradigma de programación imperativa.

Brennan y Resnick (2012), consideran también conceptos computacionales básicos las secuencias de acciones, las sentencias condicionales y bucles referenciados en los teoremas de programa estructurado y de Dijkstra, y añaden a esta lista los conceptos de datos, operadores, paralelismo y eventos. Para el propósito de esta tesis, es de especial relevancia el testimonio de estos autores, pues son parte del equipo de investigación precursor de la herramienta Scratch. Por este motivo su propuesta se va a tomar como referencia para realizar una descripción general de la notación algorítmica en el paradigma imperativo. Datos, operadores, secuencia de acciones, sentencias condicionales e iteraciones, son conceptos que está presente en cualquier lenguaje de programación estructurado. Paralelismo y eventos están presentes en Scratch, y por eso se incluyen en este estudio.

Todos los conceptos mencionados se desarrollan a continuación:

- **Datos**

Dato es la expresión general que describe los objetos con los cuales opera un algoritmo (Joyanes Aguilar, 2008, p. 4). Estos datos pueden ser numéricos, lógicos (verdadero o falso), caracteres, cadenas de caracteres, etc.

En este contexto es habitual asociar el concepto de dato, a los de variable y constante. Una variable es un objeto que puede cambiar su valor en el transcurso de un algoritmo, mientras que una constante permanecerá con su valor inicial sin poder modificarlo. Trasladando estos elementos al entorno de un lenguaje de programación, tanto una variable como una constante son espacios que se reservan en el sistema de almacenaje de datos del ordenador (por lo general, en la memoria principal) a los que se les asigna un nombre simbólico que permitirá referenciar el contenido del espacio reservado. El dato contenido en ese espacio podrá cambiar o no en función de si se trata de una variable o una constante (Joyanes Aguilar, 2008, p. 5).

También dentro de esta categoría podemos incluir a las listas de datos, a las que en ciertos lenguajes de programación se las denomina *arrays* o matrices. Una *array* es un objeto contenedor de datos que contiene un número fijo de valores de un solo tipo.

- **Operadores**

Son elementos dentro del algoritmo que permiten manipular los datos, y obtener nuevos valores a partir de ellos.

Los operadores se pueden clasificar según el tipo de operando (dato) que manipula y el resultado que obtiene. De esta manera tendremos:

- Operadores aritméticos, que operan con valores numéricos (suma, resta, multiplicación, etc.) y obtiene un valor numérico (Felleisen, Findler, Flatt y Krishnamurthi, 2001, p. 21).
- Operadores relacionales, que se utilizan para establecer la relación existente entre dos valores cualesquiera (mayor que, igual a, menor que, etc.), y obtienen un resultado lógico (verdadero o falso) (Felleisen et al., 2001, p. 40).
- Operadores lógicos, que basan sus principios en el álgebra booleana (Boole y Corcoran, 2003), operan con valores lógicos (AND, OR, NOT, etc.), y obtienen un resultado lógico. Los operandos de un operador lógico pueden ser el resultado de una operación relacional (Felleisen et al., 2001, p. 40).

Con datos, variables, operadores y funciones (concepto que veremos a continuación) se pueden construir **expresiones.** Una expresión es una

representación de un valor que surge como resultado de la combinación de varios elementos (datos, operadores, etc.), interpretados según las normas de precedencia y asociación que determine el lenguaje de programación. Un ejemplo de expresión sería *x = (5 + 2) x 3*. En este ejemplo, x representa el valor de 21, como resultado de sumar primero 5 + 2 (los paréntesis marcan la precedencia de esta operación), y después multiplicarlo por 3. En este caso el símbolo = serviría para asignar el valor de la operación a la variable *x*. Se podría considerar que el símbolo = es un operador de asignación, añadiendo así una nueva categoría a la lista de tipos de operadores definidos anteriormente.

- **Secuencia de acciones**

Una acción elemental es aquella que el ordenador puede llevar a cabo, y que involucra los conceptos de datos y operadores explicados anteriormente. Una acción puede suponer recoger un dato (acción de entrada), utilizar los operadores con los datos disponibles para obtener un resultado (acciones aritmético-lógicas), o mostrar el resultado obtenido (acciones de salida).

En un algoritmo tiene igual importancia expresar las secuencias de acciones a realizar, como el orden correcto en el que se tienen que ejecutar.

En ocasiones un algoritmo complejo se puede descomponer en algoritmos más sencillos. A estos algoritmos más simples se les puede asignar un identificador, y en la expresión del algoritmo general considerarlos como una acción más, representada por ese identificador. Las funciones y los procedimientos serían la equivalencia de estos subalgoritmos en el contexto de la programación, donde una porción de código dentro de todo el programa representa la solución algorítmica a una parte del problema. Esa porción de código se invocará a través de su identificador dentro del programa general cuando sea necesario. La diferencia entre función y procedimiento es que una función es una rutina que devuelve un valor como resultado de la consecución del algoritmo, y un procedimiento no devuelve nada (McConnell, 2004, p. 181). Al dato que devuelve una función se le denomina valor de retorno. Funciones y procedimientos pueden recibir datos de entrada de los que partir (lo que se conoce como argumentos o parámetros). El envío de los parámetros de entrada, y la recepción del valor de retorno en el caso de las funciones, se realizan desde el programa principal, aunque pueden existir funciones que no reciban argumentos de entrada.

A esta acción de dividir un programa en subprogramas se la denomina Modularización (Parnas, Clements y Weiss, 1985), y es una práctica que se recomienda porque, entre otras razones, reduce la complejidad del programa, evita que el código se duplique innecesariamente, y favorece la legibilidad del código, así como su portabilidad (McConnell, 2004).

- **Sentencia condicional**

Una sentencia condicional se utiliza cuando en un algoritmo se quiere establecer que una acción o secuencia de acciones solo se ejecuten si se cumple una determinada condición.

Las sentencias condicionales pueden ser de varios tipos:

- **Sentencia condicional simple:** Sólo si la *Condición* (expresión lógica) es verdadera se ejecutará la *Secuencia de acciones*. Una sentencia condicional simple tendría la siguiente forma:

```
Si Condición Entonces
      Secuencia de acciones A;
Fin Si
```

- **Sentencia condicional doble o alternativa:** Se establece una secuencia de acciones tanto para cuando se cumple la condición, como para cuando no se cumple. Una sentencia condicional doble tendría la siguiente forma:

```
Si Condición Entonces
      Secuencia de acciones A;
Si No
      Secuencia de acciones B;
Fin Si
```

En la condición alternativa se puede plantear una segunda condición, quedando la estructura de la siguiente forma:

```
Si Condición1 Entonces
      Secuencia de acciones A;
Si No y Si Condición2 Entonces
      Secuencia de acciones B;
Si No
      Secuencia de acciones C;
Fin Si
```

La última opción Si No quedaría como la opción a realizar si no se han complicado ninguna de las condiciones anteriores. Una casuística concreta de esta opción sería la sentencia condicional múltiple que se describe a continuación.

- **Sentencia condicional múltiple:** Cuando la secuencia de acciones a realizar depende del valor discreto de un determinado dato, se puede establecer una sentencia condicional múltiple, que tendría la siguiente forma:

```
Seleccionar Valor discreto
      En el caso del que el valor sea Valor 1:
            Realizar Secuencia de acciones 1;
      En el caso del que el valor sea Valor 2:
            Realizar Secuencia de acciones 2;
      …
      En el caso del que el valor sea Valor n:
            Realizar Acción n;
      [Por defecto:    Realizar Secuencia de acciones X;]
Fin Caso
```

En este tipo de sentencias, opcionalmente, se puede establecer una acción por defecto si el valor discreto no coincide con ninguno de los casos planteados.

- **Estructura iterativa o bucle**

La composición iterativa se utiliza cuando la ejecución de una acción o secuencia de acciones debe repetirse varias veces. Una condición determinará cuándo deben continuar las iteraciones. La condición que condiciona la continuidad del bucle puede definirse antes o después de definir la secuencia de acciones. Si la condición está al principio de la estructura iterativa, tendrá la siguiente forma:

```
Mientras se cumpla la Condición Realizar
      Secuencia de acciones;
Fin Mientras
```

Si la condición se encuentra al final de la estructura iterativa, tendrá la siguiente forma:

```
Realizar
      Secuencia de acciones;
Mientras se cumpla la Condición
```

La secuencia de acciones incluida en una estructura iterativa en la que se define la condición al principio, puede no llegar a ejecutarse nunca si el contexto anterior a la estructura iterativa establece que la condición no se cumpla antes de iniciar el bucle. Por el contrario, una estructura iterativa en la que se define la condición al final garantiza que la secuencia de acciones se ejecute al menos una vez. Un bucle en el que la condición de permanencia no cambia es un bucle infinito.

Existe una estructura iterativa adicional que tiene una estructura indicada en los casos en los que la repetición de la secuencia de acciones se quiere llevar a cabo un número concreto de veces. Esta estructura tiene la siguiente forma:

```
Desde que contador = 0 Mientras que contador < n Hacer
      Secuencia de acciones;
      Incrementar en uno el contador;
Fin Desde
```

En este caso la secuencia de acciones se repetirá *n* veces, y un dato variable, representado en el ejemplo como *contador*, se incrementará en uno en cada iteración, llevando de esta manera el control del número de vueltas completadas en cada momento. Esta estructura admite distintas variantes, por ejemplo, comenzar con el contador igualado al total de vueltas y decrementar su valor en uno por cada vuelta.

Esta sentencia en realidad es un caso particular de las anteriores formas de iteración, y se podría expresar también de la misma manera:

```
i = 0;
Mientras i < n Hacer
      Secuencia de acciones;
      i = i + 1;
Fin Mientras
```

Un bucle puede contener dentro de él otro bucle. A este tipo de estructuras se les denominan bucles anidados.

Otra forma de iteración se puede llevar a cabo a través de la recursión o recursividad, donde “en la definición de la función matemática se utiliza la propia función que se está definiendo” (Rojas, 2015, p. 8). Este concepto de recursividad trasladado a la programación implica el uso de funciones. El algoritmo que incluye esa función se llama a sí mismo para resolver el problema, haciendo las transformaciones necesarias para llegar al caso base y terminar (o no, si lo que se quiere realizar es un bucle infinito). Un ejemplo clásico para entender el concepto de recursión es el algoritmo que calcula el factorial de un número:

```
Función Factorial(numero)
      VAR resultado: dato de tipo entero;
       Si (numero < 2) Entonces
            resultado = 1;
      Si No
            resultado = n * Factorial(numero - 1);
      Fin Si
      DEVUELVE resultado;
Fin de la función;
```

Según el principio establecido por la tesis de Church-Turing (Turing y Copeland, 2004), una función matemática expresada a través del Cálculo Lambda tiene su equivalente en la máquina de Turing. Ese mismo principio se aplica en el caso de la recursividad en el paradigma de programación imperativa: cualquier problema que se pueda resolver a través de un algoritmo recursivo, puede resolverse también utilizando bucles (Insa y Silva, 2015). En otros paradigmas, como el funcional o lógico, los bucles ni siquiera existen, y toda iteración debe realizarse a través de métodos recursivos. Sin embargo, no son totalmente equivalentes, y según la circunstancia y el tipo de problema a resolver, a veces es preferible utilizar la recursión, y a veces es mejor utilizar los bucles. Una de las diferencias más significativas se encuentra en el rendimiento que ofrecen ambos mecanismos. Diversos estudios (Harrison y Khoshnevisan, 1992; Liu y Stoller, 1999; McCarthy, 1962), demuestran que ciertos compiladores generan un programa más eficiente si se realizan a través de bucles, y defienden la transformación de los programas a este modelo. Por el contrario, también existen estudios (Gustavson, 1997; Myreen y Gordon, 2009) que argumentan que los algoritmos recursivos suelen ser más

comprensibles e intuitivos, más fáciles de depurar, más fáciles de reutilizar, y que incluso consiguen un mejor rendimiento en determinados contextos (Elmroth y Gustavson, 2000).

- **Paralelismo**

El paralelismo es un modelo de programación que amplía el concepto de modularización: además de dividir el algoritmo en tareas más pequeñas, se trata de ejecutar estas tareas de forma simultánea (Almasi y Gottlieb, 1988). El paralelismo se fundamenta en dos principios: sincronización y comunicación. Por poner un ejemplo para entender este concepto, si un equipo de personas se dividen el trabajo para llevar a cabo un proyecto, y todos trabajan en paralelo, para que su trabajo confluya en un objetivo común debe haber comunicación entre los miembros de ese equipo. Esta comunicación permitirá que se sincronicen en las pequeñas tareas, y que éstas se hagan en el orden adecuado cuando el resultado de una acción es necesario para comenzar otra. Este ejemplo se puede trasladar al ámbito de la programación, donde se deberán establecer los mecanismos necesarios para que las funciones que se ejecuten en paralelo estén comunicadas, accedan a los recursos comunes de forma controlada, estén sincronizadas, etc.

Barney (2015) enumera las principales ventajas de utilizar paralelismo a la hora de programar:

- Ahorro de tiempo: en teoría, utilizar más recursos para realizar una tarea reducirá el tiempo que tarde ésta en realizarse.
- Posibilidad de resolver problemas mayores y más complejos: muchos problemas son de tal envergadura, o tan difíciles de resolver, que no es posible solventarlos con un solo equipo. Un único recurso de cálculo solo puede hacer una tarea a la vez, múltiples recursos podrán hacer varias tareas. Un problema inabarcable es posible que pueda descomponerse en problemas más pequeños que sí puedan afrontarse en paralelo.
- Proporcionar concurrencia: posibilidad de hacer varias cosas al mismo tiempo.
- Aprovechamiento de recursos remotos: el paralelismo permite utilizar recursos de computación disponibles en una red, o incluso en Internet si los recursos locales son insuficientes.
- Ahorro de costes: utilizar múltiples recursos de computación "baratos" trabajando en conjunto, puede ahorrar costes frente a lo que supondría adquirir un “superordenador”.

- Superación de restricciones de memoria: los equipos individuales tienen recursos de memoria finitos. Para problemas grandes, el uso de las memorias de múltiples ordenadores puede superar este obstáculo.
- Aprovechar mejor los recursos disponibles: los dispositivos actuales, incluidos los ordenadores portátiles, los dispositivos móviles, etc., presentan una arquitectura con múltiples procesadores/núcleos. El software paralelo está específicamente diseñado para este tipo de hardware. En la mayoría de los casos, los programas en serie se ejecutan en estos dispositivos desaprovechando parte de su potencia de cálculo.

- **Evento**

Etzion y Niblett (2010) proporcionan dos definiciones de evento. Por un lado, un evento es una ocurrencia dentro de un sistema o dominio particular, es algo que ha ocurrido, o que está sucediendo en ese dominio. Por otro lado, la palabra evento también se usa para definir una entidad de programación que representa tal ocurrencia en un sistema informático.

En un número significativo de compiladores, los eventos aparecen en forma de excepciones cuyo papel es interrumpir el flujo regular de la ejecución del programa y provocar que se lleve a cabo un proceso alternativo. Por ejemplo, si un programa trata de dividir entre cero, se lanza un evento de excepción que permita al usuario terminar el programa con un mensaje de error, o realizar una corrección para posteriormente continuar con el proceso de cálculo. Pero los eventos no están únicamente asociados a detectar o señalar errores, de hecho su uso más habitual va por otros derroteros en los entornos de desarrollo actuales. Por ejemplo, los eventos están presentes en entornos gráficos de programación, como Visual Studio o Java AWT, donde los componentes de la interfaz de usuario (botones, listas desplegables, ventanas, etc.) están diseñados para responder a eventos que realice el usuario, como presionar una tecla, o hacer clic con el ratón.

3.3.2. Otras formas de representar un algoritmo: pseudocódigo y diagramas de flujo

3.3.2.1. Diagramas de flujo

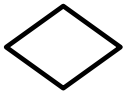
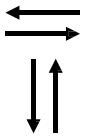
Goldstine y Von Neumann (1948) son posiblemente los primeros en utilizar un diagrama de flujo para plantear un programa informático. Corrado y Jacopini (1966) son también referentes en este ámbito.

Un diagrama de flujo permite crear una representación gráfica del algoritmo. Para hacer esta representación se utilizan símbolos, flechas, figuras geométricas, etc., con un significado definido, y unas reglas de combinación. A través de estas formas se podrá reflejar los distintos pasos que hay que recorrer para llegar de un planteamiento inicial a una solución final, atravesando diferentes caminos, en los términos que plantea el algoritmo que lo resuelve. Así, un diagrama de flujo comienza con un punto de inicio, y las flechas identifican el flujo del camino, hasta llegar a un punto final.

Una posible simbología estandarizada de un diagrama de flujo está catalogada por la *International Organization for Standardization* (ISO) en el documento *International Standard 1028 "Information Processing – Flowchart Symbols"*, y por la *American National Standard*, en el documento *Flowchart Symbols and their Usage in Information Processing, ANSI X3.5-1970* (Chapin, 1970, 1979).

A continuación se enumeran algunos los elementos básicos que componen esta simbología, y su significado (Chapin, 1970; Peña Marí, 2005):

Tabla 3
Algunos de los principales símbolos para la elaboración de un diagrama de flujo

Símbolo	Nombre / Significado	Símbolo	Nombre / Significado
	Proceso – acción o secuencia de acciones.		Conector - conecta las secciones del diagrama de flujo, de modo que el diagrama puede mantener un flujo lineal.
	E/S - Entrada y salida de datos.		Terminal – indica el principio y el fin del algoritmo
	Decisión – sentencia condicional. Si una de las flechas que parten del rombo vuelve a un punto anterior del diagrama, representará un bucle.		Flechas – indican la dirección hacia donde fluye o progresa el programa

Fuente: (Peña Marí, 2005)

Por lo general, para construir un diagrama de flujo se siguen las siguientes reglas:

- Todos los símbolos del diagrama de flujo están conectados por flechas.
- Los diagramas de flujo se dibujan de modo que el flujo generalmente va de arriba abajo, aunque en ocasiones también pueden ir de izquierda a derecha.
- El comienzo y el final del flujo se indica usando el símbolo de terminal.

A continuación se muestra un ejemplo de diagrama de flujo:

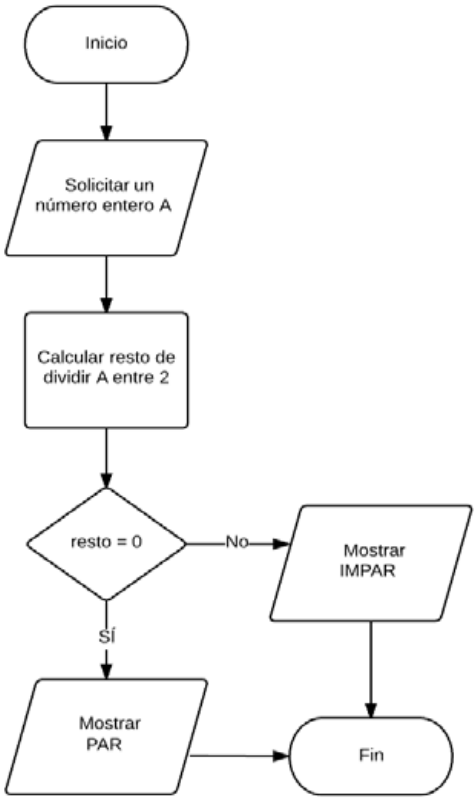


Figura 7. Diagrama de flujo de un algoritmo que determina si un número entero es par o impar.

Fuente: elaboración propia.

Es habitual en los cursos de programación que los estudiantes empiecen a escribir código sin haber asimilado o adquirido conocimientos de base previos. Un diagrama de flujo pude ser un instrumento más asequible para representar un algoritmo en estas etapas iniciales del aprendizaje (Crews y Ziegler, 1998).

3.3.2.2. Pseudocódigo

El Pseudocódigo es un instrumento que permite realizar una descripción informal de alto nivel del principio de funcionamiento de un algoritmo (Zobel, 2014). El pseudocódigo utiliza estructuras que se podrían encontrar en un lenguaje de programación profesional, solo que en el caso del pseudocódigo están diseñadas para que las entienda un ser humano, y no un compilador o un intérprete. En ese proceso de simplificación, se suelen omitir elementos que no son vitales para entender el algoritmo, y no se siguen unas reglas de sintaxis estrictas, por eso no hay una estandarización del pseudocódigo como tal, aunque sí existen notaciones

técnicas, utilizadas con el mismo propósito que el pseudocódigo con las estructuras más definidas, como la notación Backus–Naur (Knuth, 1964).

El pseudocódigo se suele utilizar en la enseñanza de la programación, porque un algoritmo descrito con esta herramienta es más fácil de comprender para una persona que aún no conoce las especificaciones de un lenguaje de programación convencional.

A continuación se pone un ejemplo de pseudocódigo, del mismo algoritmo que se representa en un diagrama de flujo en la *Figura 7* donde se determina si un número introducido por teclado es par o impar (en este ejemplo, % representa el operador que obtiene el resto de una división entre dos números enteros):

```
INICIO
      a: Número Entero
      resto: Número Entero
      SACAR POR PANTALLA: “Introduce un número”
      RECOGER POR TECLADO: a
      resto = a % 2
      Si resto es igual a 0 entonces
            SACAR POR PANTALLA: “El número es par”
      Si no, entonces
            SACAR POR PANTALLA: “El número es impar”
FIN
```

Por completar este recorrido por la sintaxis, y las distintas maneras de representar un algoritmo, a continuación en la *Figura 8* se representa este mismo ejemplo de número par o impar, esta vez utilizando Scratch.

Figura 8: Algoritmo que determina si un número entero es par o impar, representado con Scratch.

Fuente: elaboración propia con la herramienta Scratch (MIT, 2013).

4. El proceso de enseñanza-aprendizaje de la programación

Carsten Schulte (2016) revisa el papel de la programación en la educación, y llega a las siguientes conclusiones sobre lo que supone aprender a programar:

- La programación es una forma de interacción con la computadora, relacionada con la pérdida de la manipulación directa y la necesidad de utilizar una notación abstracta.
- La programación supone afrontar la complejidad de la solución de un problema de una forma específica mediante el uso de representaciones propias dentro del programa, también denominadas subprogramas o módulos.
- Programar tiene que ver con la automatización de procesos.
- Como a menudo no existe un camino directo claro entre el problema y la solución, la programación incluye un proceso creativo en esa búsqueda de soluciones.
- Es frecuente que la retroalimentación inmediata esté a menudo ausente. Es decir, en el momento en el que se escribe una línea de código no se tiene la absoluta certeza de que dicha línea vaya a cumplir con el propósito con el que está siendo redactada.
- Los programas se pueden escribir para el uso personal, o para que sean utilizados por otros. Un programa, por tanto, se dirige no solo al ordenador como lector, sino también (y probablemente aún más) a un lector humano. En conclusión, la programación incluye aspectos de un proceso de comunicación.

Henry M. Walker, citado por Ben-Ari (2016) explica que el término programación es utilizado por los informáticos para describir todo el proceso de resolución de problemas. El propio Ben-Ari propone una definición que se centra en el programa como elemento tecnológico: “La programación es la especificación formal de un cálculo que puede ser ejecutado por una computadora.” (“*Programming is the formal specification of a computation that can be executed by a computer.”)* (Ben-Ari, 2016, p. 44)

El autor explica que esta definición debe interpretarse en el sentido más amplio posible, dado que los programas se escriben no solo con lo que tradicionalmente se conciben como lenguajes de programación, como pueden ser C++ o Java, sino también con entornos visuales educativos como Alice o Scratch. Incluso especificar un cálculo en una hoja de cálculo puede entrar dentro del concepto de esta definición de programación. Aunque una hoja de cálculo (excluyendo las macros) no presenta todas las estructuras clásicas de programación (bucles, etc.), las fórmulas

deben ser expresadas usando un formalismo con una sintaxis y semántica precisas. Incluso la redacción de un documento se puede considerar programación. Desde esta perspectiva se podría considerar LaTeX y HTML / CSS como lenguajes de programación donde las propiedades de un documento son formalmente especificadas (LaTeX incluye incluso elementos de recursión, variables, sentencias condicionales, etc.).

Eckerdal y Berglund (2005) tratan de acotar la definición de programación. Para estos autores saber programar significa:

- Entender algunos lenguajes de programación, y utilizarlos para escribir códigos de programas.
- Adquirir una forma de pensar que se alinea con los lenguajes de programación. Dicho de otra manera, aprender cómo se debe pensar para poder expresarse con la lógica y las reglas que tiene un lenguaje de programación.
- Comprender mejor los programas informáticos que están presentes en nuestra vida cotidiana (por ejemplo, entender cómo la máquina de refrescos es capaz de devolver el cambio exacto).
- Adquirir una forma de pensar que permite resolver problemas.
- Adquirir habilidades que pueden ser utilizadas más allá del ámbito de la programación.

Cada una de las definiciones son complementarias entre sí, y todas en su conjunto sirven de punto de partida para este capítulo, que pretende transitar por todo lo que significa aprender a programar.

Si estamos hablando de aprender a programar, primero debemos entender qué significa aprender. En el capítulo 4.1 se hace un breve recorrido por las que se consideran principales teorías de aprendizaje (conductismo, cognitivismo y constructivismo), y se incluye una reseña al construccionismo y al conectivismo, que aun no considerándose teorías clásicas, enuncian ciertos principios que se vinculan directamente con el aprendizaje de la programación, y por tanto deben estar presentes en este apartado.

Como se ha visto en la introducción de este capítulo, una parte intrínseca a la definición de programación, es la resolución de problemas. En el capítulo 4.2 se desarrolla la parte relativa a los posibles planteamientos y mecanismos para resolver problemas, presentando las bases teóricas existentes al respecto.

Para programar no basta con saber resolver un problema, hace falta plantear esa solución en los términos que el ordenador entiende. El capítulo 4.3 se centra en la

descripción del llamado pensamiento computacional, es decir, en la parte de las definiciones de programación referentes a la forma en la que se debe pensar para plantear un problema en términos que el ordenador pueda entender.

Por último, el capítulo 4.4 reúne todo lo expuesto en apartados anteriores, y analiza cómo se aprende y cómo se enseña a programar. Para ello aborda algunas técnicas y metodologías que se pueden aplicar en la enseñanza y el aprendizaje de la programación, presenta una lista de posibles herramientas que se pueden utilizar con este propósito, enumera estándares de aprendizaje que pueden servir de indicadores para medir el nivel de conocimiento adquirido relativo a la programación, y presenta los factores que influyen y pueden dificultar este aprendizaje. Se completará el capítulo mostrando algunos ejemplos de iniciativas que promueven la enseñanza de los lenguajes de programación fuera del ámbito reglado, tanto a nivel nacional como mundial, que permitirán hacerse una idea de la dimensión que está adquiriendo este asunto.

4.1. Teorías del aprendizaje

Las teorías de aprendizaje son marcos conceptuales en los cuales el conocimiento es obtenido, procesado y retenido (Chaudhary, 2013).

Este capítulo pretende sentar las bases que sustentan cualquier aprendizaje. Entendiendo cómo se aprende en general, podremos entender los procesos asociados al aprendizaje concreto de la programación (ver capítulo 4.4). Esto nos permitirá analizar la herramienta Scratch como instrumento de enseñanza y aprendizaje.

Comúnmente se consideran conductismo, cognitivismo y constructivismo como las tres teorías principales de aprendizaje. Por ello en este capítulo se describen brevemente. También se incluyen el construccionismo y el conectivismo porque, como se ha mencionado en la introducción del capítulo 4, aun no considerándose teorías clásicas (de hecho, algunos autores ni siquiera las consideran entes propios dentro de las teorías de aprendizaje), describen ciertos principios que se vinculan directamente con el aprendizaje de la programación, y por este motivo se decide su presencia en este apartado.

4.1.1. Conductismo

El primer documento en sentar las bases del paradigma psicológico conductista aparece en 1913, publicado por John B. Watson, “La psicología tal como la ve un

conductista" (Watson, 1913), en el que resume los principios del pensamiento conductista:

> Todas las escuelas de psicología, excepto la del conductismo, afirman que la "conciencia" es el tema de la psicología. El conductismo, por el contrario, sostiene que el tema de la psicología humana es el comportamiento o las actividades del ser humano. El conductismo afirma que la "conciencia" no es un concepto definible ni utilizable; Que es simplemente otra palabra para el "alma" de los tiempos más antiguos. La vieja psicología está así dominada por una sutil filosofía religiosa (p.1).

La razón de que el conductismo pretenda este cambio de objeto de estudio se basa en que para Watson la conciencia no es susceptible de ser definida ni medida, y por tanto no puede ser objeto de estudio científico. Se plantea pues un cambio en el planteamiento de la psicología, hasta entonces centrada en la conciencia del individuo, para trasladar el foco de atención a la conducta (*behaviour*) del mismo, el cual es observable y medible. De aquí procede la expresión "caja negra" para referirse a los procesos no medibles, y por tanto despreciables, que tienen lugar en la mente del sujeto (Good y Brophy, 1990).

Los seguidores del conductismo tradicional, herederos de la tradición empirista que concibe como fuente principal del conocimiento a la experiencia (Ertmer y Newby, 1993), concibieron el aprendizaje como un proceso de formación de conexiones entre estímulos y respuestas (Bransford, Brown y Cocking, 2000), o lo que es lo mismo, a los cambios de la conducta observable (en forma o frecuencia) a partir de un estímulo (Ertmer y Newby, 1993).

Los primeros experimentos (y probablemente los más famosos) en conductismo se efectuaron sobre animales, tales como la rata blanca (Watson, 1903) (que le valió el doctorado al propio Watson), los célebres perros de Pavlov y las palomas de Skinner. En un principio, estos experimentos querían extrapolar el comportamiento y aprendizaje animal al comportamiento y aprendizaje humanos, solo prestando atención exclusivamente a la conducta observable, o más concretamente, a los cambios en esta conducta que evidencian el aprendizaje (Campos, 2001)

Para la psicología conductista existen dos medios por los cuales se modifican las conexiones ya establecidas o se crean nuevas conexiones entre estímulos y respuestas: el condicionamiento clásico o respondiente (Skinner, Ardila y Barrera, 1977) y el condicionamiento operante.

4.1.1.1. Condicionamiento clásico

El condicionamiento clásico se refiere al aprendizaje por detección de regularidades o relaciones entre estímulos y entre estímulos y respuestas o conductas. Las

respuestas a las que aplica el condicionamiento clásico son de tipo involuntario, innato y vinculado a la supervivencia de la especie (Skinner et al., 1977; Trianes Torres y Gallardo Cruz, 2004). Por ello se trata de un aprendizaje común con la escala animal.

Cuando un estímulo suscita de forma automática una respuesta, estamos hablando de un estímulo incondicionado que desencadena una respuesta incondicionada. Si, mediante contigüidad o asociación el sujeto relaciona un estímulo que no produce la respuesta incondicionada (denominado estímulo neutro) con el estímulo incondicionado, el estímulo neutro se convierte en condicionado y es capaz de suscitar por sí solo, en ausencia del estímulo incondicionado, la respuesta que pasa a denominarse condicionada (Trianes Torres y Gallardo Cruz, 2004).

En su famoso experimento, Pavlov estudió que un perro, ante la presentación de un plato con comida (estímulo incondicionado), comenzaba a salivar (respuesta incondicionada). Al presentar el plato de comida con posterioridad al sonido de una campana (estímulo neutro), el perro asociaba el estímulo condicionado y el neutro, hasta tal punto que la campana por sí sola (estímulo condicionado) producía la salivación (respuesta condicionada) (Mergel, 1998).

Pavlov enunció varios principios asociados al proceso de condicionamiento:

- Los condicionamientos negativos, en los que la respuesta condicionada es la ansiedad, requieren menor repetición en la asociación del estímulo neutro y el estímulo incondicionado (por ejemplo, basta un accidente para desencadenar ansiedad a partir del estímulo, inicialmente neutro, de subir a un avión) (Trianes Torres y Gallardo Cruz, 2004)
- Existe generalización de estímulos: un estímulo neutro similar a un estímulo condicionado puede producir la respuesta condicionada. (Mergel, 1998). En un experimento diseñado por Watson y Rayner, en el que un bebé de nueve meses llamado Albert generaba un condicionamiento entre un sonido estridente (estímulo incondicionado), miedo y llanto (respuesta incondicionada) y una rata blanca (estímulo neutro). Una vez consolidado el aprendizaje (con solo mostrar la rata, el niño respondía con miedo y llanto), la respuesta condicionada también se producía al mostrar al niño un conejo, un perro, e incluso la cabeza (con el pelo blanco) del propio Watson sin que se hubiera utilizado ninguno de ellos como estímulos para el condicionamiento inicial, sino que desencadenaban la respuesta por su similitud con el estímulo condicionado (Pérez Álvarez, 1995).

- Extinción: si deja de ejercitarse la respuesta condicionada, el condicionamiento desaparece y el sujeto vuelve al estado en el que solo el estímulo incondicionado suscita la respuesta incondicionada (Mergel, 1998).
- Una vez extinguida, la respuesta condicionada se puede recuperar mediante un nuevo proceso de asociación de los mismos estímulos. Este proceso será más corto que el condicionamiento inicial (Mergel, 1998).
- Condicionamiento de orden superior: una vez adquirido un condicionamiento, es posible asociar un nuevo estímulo neutro al estímulo condicionado, tal y como se hizo en un principio con el estímulo incondicionado, y crear un nuevo condicionamiento (Mergel, 1998).

4.1.1.2. Condicionamiento operante

El condicionamiento operante ha tenido mucha más relevancia en su aplicación a la educación, especialmente porque se aplica no ya a respuestas innatas sino a respuestas voluntarias.

Parte de la base de que las respuestas de los sujetos tienen influencia (“operan”) sobre el ambiente, y no solo son una respuesta a este. El esquema es el siguiente:

Dado un estímulo inicial, llamado discriminativo, que no provoca respuesta pero que es susceptible de provocar una conducta, si a raíz de dicha conducta se produce una consecuencia gratificante para el sujeto, entonces ante el estímulo discriminativo es más probable que se produzca dicha conducta que si la consecuencia es desagradable.

Las consecuencias que provocan un aumento en la frecuencia de una determinada conducta se denominan refuerzos. Los refuerzos tienden a aumentar la frecuencia de determinadas conductas, ya sean éstos positivos (en forma de recompensa) o negativos (en forma de cese de una estimulación aversiva).

Por otro lado, las consecuencias que disminuyen la frecuencia de una conducta se denominan castigos. Al igual que en el caso de los refuerzos, los castigos pueden ser positivos (en los que se presentan al sujeto consecuencias aversivas) o negativos (en los que se priva al sujeto de estímulos gratificantes). El castigo se emplea con enorme frecuencia en educación, aun inconscientemente, pero su uso ha de ser meticulosamente estudiado para garantizar su éxito: en muchas ocasiones es mucho más eficaz un refuerzo negativo (Trianes Torres y Gallardo Cruz, 2004).

Las conductas que no son reforzadas ni castigadas (es decir, que no producen refuerzo) tienden asimismo a extinguirse (Mergel, 1998).

El estudio del condicionamiento operante, del que su mayor exponente es W. F. Skinner, tiene base en los estudios de E. Thorndike, que lo llevaron a enunciar su famosa Ley del Efecto. Esta ley afirma que cuando la conexión entre un estímulo y la conducta es recompensada, este se refuerza, y cuando es castigado, se debilita. En un primer enunciado, Thorndike afirmaba que la fuerza de la recompensa y del castigo son iguales y opuestas. Más tarde, una serie de experimentos obligaron a revisar esta ley, pues la influencia del castigo en el debilitamiento de una conexión parecía ser muy inferior a la de la recompensa en el fortalecimiento de la misma (Mergel, 1998). Nótese que el significado del término “efecto” guarda una enorme similitud con el de “refuerzo” en el condicionamiento operante de Skinner.

Para un conductista, en definitiva, un aprendizaje es eficaz cuando la asociación estímulo-respuesta es fuerte, es decir, cuando siempre que se presenta cierto estímulo el sujeto ejecuta cierta conducta (Campos, 2001).

4.1.2. Cognitivismo o cognoscitivismo

La teoría conductista no satisfacía completamente a muchos investigadores, precisamente por la idea de la “caja negra”. Como reacción, en los años 60 se produjo el auge de una nueva teoría, llamada teoría cognitiva, más centrada en lo que sucede en esa caja negra conductista. Al pretender dar un sentido a cómo la mente procesa los datos que percibe por los sentidos, esta teoría se asimila al enfoque del procesamiento de la información, hasta el punto de que ambas expresiones son frecuentemente intercambiables (Trianes Torres y Gallardo Cruz, 2004).

Good y Brophy (1990), expresan claramente el papel del cognitivismo en relación con el conductismo en tanto no pretende desacreditarlo, sino complementarlo en muchos aspectos:

> Los teóricos del cognoscitivismo reconocen que una buena cantidad de aprendizaje involucra las asociaciones que se establecen mediante la proximidad con otras personas y la repetición. También reconocen la importancia del refuerzo, pero resaltan su papel como elemento retroalimentador para corrección de respuestas y sobre su función como un motivador. Sin embargo, inclusive aceptando tales conceptos conductistas, los teóricos del cognoscitivismo ven el proceso de aprendizaje como la adquisición o reorganización de las estructuras cognitivas a través de las cuales las personas procesan y almacenan la información. (p. 187)

Así pues, los estudiosos de la Teoría del Procesamiento de la Información (TPI) centran su estudio en cómo la mente recibe los estímulos (la información), cómo los reorganiza y cómo crea representaciones mentales a partir de ellos (Campos, 2001; Shannon, 2001). En el desarrollo de esta teoría ha jugado un papel muy importante el auge de la tecnología, en particular de la informática. Esto es debido a que, si bien no conocemos el funcionamiento en términos pormenorizados del cerebro humano, sí se pueden establecer analogías entre este funcionamiento y el de los ordenadores, que sí es bien conocido. Esta reflexión la lleva a cabo la cibernética (Delval, 2002; Wiener, 1948).

La TPI se centra en el papel que juega la memoria en el aprendizaje. Desde un punto de vista inicial, se puede considerar la memoria como un ente único, habida cuenta de que hay personas con mayor capacidad para almacenar datos que otras (Cowan, 2008). Pero en 1958 Donald E. Broadbent desarrolló un modelo de memoria, llamado *modelo modal* (Baddeley, 1992), basado no en un único elemento, sino en tres compartimentos independientes que interactúan entre sí. Este modelo múltiple ha sido apoyado en sus líneas generales por varias investigaciones (Cowan, 1988), y tiene las siguientes características:

- Un primer compartimento de memoria sensorial, en el cual una cantidad ilimitada de información procedente de los sentidos se almacena durante una duración estimada de entre 0.25 y 2 segundos. Es importante notar que los estímulos se almacenan en esta etapa independientemente de que el sujeto esté centrando su atención en ellos o no. Esto significa que la inmensa mayoría de la información que pasa por este primer compartimento es descartada casi inmediatamente (Craik y Lockhart, 1972).

- El segundo compartimento es el de la memoria a corto plazo. Cuando el sujeto presta atención a un estímulo concreto, pasa del almacén sensorial al almacén de memoria a corto plazo. Este difiere sustancialmente del anterior en que su capacidad es limitada: se estima que entre siete y nueve ítems de información simultáneamente (Ericsson y Kintsch, 1995), si bien el límite de capacidad no está claramente delimitado, y de hecho ni siquiera se puede asegurar que el límite sea fijo (Cowan, 1988). Al contrario, el tiempo durante el cual la información reside en la memoria a corto plazo se amplía, pasando a ser normalmente entre 5 y 20 segundos, pero siempre menos que 30 (Craik y Lockhart, 1972). Posteriores trabajos han mostrado que el modelo inicial de Broadbent es insuficiente para explicar el funcionamiento de la memoria a corto plazo, y así Miller, Galanter y Pribram (1986), citados por Cowan (2008) introducen la idea de la “memoria operativa” (*working*

memory), en lugar de la memoria a corto plazo, como la parte de la memoria que se utiliza para planificar y llevar a cabo acciones. No es un almacén estático, sino que posee mecanismos para procesar la información que temporalmente reside en la memoria a corto plazo tradicional. Baddeley (1992) considera la memoria operativa como un conjunto formado a su vez por dos elementos: el ejecutivo central, encargado de controlar y regular todo el sistema de la memoria operativa, y dos procesos de almacenamiento, el bucle fonológico, que trata la información verbal, y la agenda viso-espacial, que trata la información visual.

- El tercer compartimento del modelo modal es la memoria a largo plazo, que organiza una cantidad (de la que no se conoce límite) de datos durante un periodo de tiempo que puede alargarse durante muchos años, e incluso se considera la posibilidad de que nunca haya pérdida de información en este repositorio, sino que la accesibilidad es la que se pueda ver afectada con el tiempo (Craik y Lockhart, 1972).

Según el paradigma cognitivo, aprender “se centra en incorporar a la estructura de memoria nuevos aprendizajes y ser capaz de recuperarlos y usarlos cuando se necesita” (Galvis Panqueva, 1992).

Dentro del campo de la educación desde el paradigma cognitivo, uno de los investigadores más importante es Jerome S. Bruner, para el cual el aprendizaje se basa en “los procesos mediante los cuales simplificamos la interacción con la realidad a partir de la agrupación de objetos, sucesos o conceptos (por ejemplo, el perro y el gato son animales)”. A esos procesos, Bruner los denomina *categorización* (Esteban Guilar, 2009). Así, el alumno genera sus propias categorías en la medida en la que interactúa con su ambiente, lo que convierte el aprendizaje en una actividad en la que el sujeto participa dinámicamente, al contrario de lo que se considera el aprendizaje bajo la perspectiva conductista, que ignora la influencia del ambiente en el sujeto y solamente estudia la relación entre estímulos y respuestas (Delval, 2002). Una de las ideas de Bruner, el currículo en espiral, que consiste en que el alumno debe pasar sucesivamente por los mismos conceptos pero en cada etapa cubriéndolos más en profundidad (Esteban Guilar, 2009) ha sido adoptada por la educación española para diseñar el currículo de Matemáticas a lo largo de la etapa de Educación Secundaria Obligatoria ("Ley Orgánica 2/2006, de 3 de mayo, de Educación. Boletín Oficial del Estado. núm. 106, pp. 17158 a 17207," 2006).

Bruner es el autor de la teoría del aprendizaje por descubrimiento, que se entiende como “actividad autorreguladora de resolución de problemas, que requiere la comprobación de hipótesis como centro lógico del acto de descubrimiento” (Barrón Ruiz, 1993). Para Bruner, es imprescindible que el sujeto tenga la experiencia personal de descubrir una información para que este la aprenda de forma significativa (Baro Cálciz, 2011). Los principios del aprendizaje por descubrimiento son los siguientes (Barrón Ruiz, 1993):

1. El ser humano es naturalmente capaz de descubrir conocimiento. Esta premisa significa que el ser humano dispone de la capacidad de autorregular su comportamiento, lo que le da pie a desarrollar experiencias de aprendizaje por descubrimiento.

2. A través del descubrimiento, el ser humano construye un nuevo objeto mental. Estos objetos o construcciones son nuevos para el sujeto, aunque no lo sean para la sociedad.

3. Identificar un problema es el punto de partida del aprendizaje por descubrimiento. Sin el interés que suscita la existencia de una barrera que hay que superar, es difícil que se desencadene un proceso de indagación y descubrimiento.

4. El aprendizaje por descubrimiento es un proceso de resolución significativa de problemas. Ante un problema, el sujeto emprende un proceso de comprobación de teorías en la búsqueda de la comprensión del problema.

5. La comprobación de conjeturas es el centro lógico del acto de descubrimiento. No basta con concebir hipótesis que expliquen fenómenos, sino que es fundamental comprobar que esas hipótesis son correctas.

6. “Por descubrimiento” implica que el sujeto debe autorregular su actividad, y ésta debe ser creativa. La mera aplicación de algoritmos proporcionados al sujeto o la reproducción de conocimiento ya adquirido no puede calificarse por descubrimiento.

7. La construcción de hipótesis implica necesariamente la producción de errores. Al contrario que en otras teorías educativas como las conductistas, el error no es algo indeseable en el aprendizaje por descubrimiento, por dos razones. La primera, el error es algo intrínseco a la propia teoría. La segunda, el error suscita la construcción de nuevas hipótesis.

8. La influencia del ambiente sociocultural forma parte del aprendizaje por descubrimiento. El entorno sociocultural del sujeto orienta de algún modo toda actividad de aprendizaje por descubrimiento. Existen numerosas experiencias de aprendizaje por descubrimiento protagonizadas por colectivos, en las que los individuos se benefician de un entorno de colaboración e intercambio de pensamientos y razonamientos.

9. El grado de descubrimiento será mayor cuanto menos predeterminado esté el proceso de solución del problema. Si el conocimiento viene determinado por indicaciones externas o por recursos internos disponibles por el sujeto, es decir, la resolución no es más que la reproducción de un proceso ya a su alcance, la experiencia de descubrimiento será nula.

10. El aprendizaje por descubrimiento puede ser guiado pedagógicamente en el aula. Habida cuenta de que el descubrimiento se fundamenta en la forma en que se estructuran los datos disponibles frente a un problema en busca de su solución, y que ese proceso de invención y descubrimiento sigue ciertas pautas dentro de la originalidad que implica, se puede educar al alumno en las actitudes que favorecen un comportamiento creativo, tanto por parte del profesor como mediante el contacto con los demás alumnos.

La Teoría del aprendizaje por descubrimiento ha tenido buena acogida, por el contraste que supone respecto a la educación tradicional (incluidos los enfoques conductistas del aprendizaje), en la que se prima la transmisión de conocimientos del profesor al alumno, el cual tiene un papel eminentemente pasivo a lo largo del proceso de aprendizaje. Sin embargo, también ha sido criticada por sus limitaciones, como el hecho de presuponer que el descubrimiento es una consecuencia lógica del razonamiento a partir de datos empíricos (Gil Pérez, 1983).

Por otro lado, observándolo con un enfoque más práctico, el aprendizaje por descubrimiento dificulta la adquisición de todos los conocimientos de que constan los currículos de los programas de estudios actuales, invalidándolo como única estrategia de aprendizaje. Asimismo, es muy difícil aplicar una metodología basada en el descubrimiento en grupos grandes, en los que es necesario atender a la diversidad (Bara Soro y Sánchez Manzano, 2001).

4.1.3. Constructivismo

Al contrario que en el caso del conductismo y del cognitivismo, que exploran el aprendizaje desde perspectivas muy diferentes y con resultados dispares, el

constructivismo no se aleja excesivamente del cognitivismo, sino que se considera más bien una rama de este. La diferencia fundamental radica en que, si bien el aprendizaje es en ambos paradigmas una actividad cognitiva, en lugar de procesar la realidad (o la información que sobre la realidad nos ofrecen los sentidos), para un constructivista el sujeto "produce su propia y única realidad". Si bien la realidad es percibida por el sujeto, este no adquiere el conocimiento de ella, no representa la realidad en su interior, sino que crea una interpretación personal del mundo a partir de sus experiencias. Esto es muy importante, pues el aprendizaje no puede ser eficaz sin un contexto significativo que ayude a asimilar los conocimientos: "el aprendizaje siempre toma lugar en un contexto y el contexto forma un vínculo inexorable con el conocimiento inmerso en él" (Ertmer y Newby, 1993). La actividad del alumno se centra, pues, no en interiorizar y acumular datos, sino de integrar los nuevos conocimientos dentro de la realidad que el alumno posee (Chadwick, 1999), formando estructuras cognitivas que se desarrollan continuamente. La transformación de las estructuras ya existentes viene inducida por las actividades en las que el aprendiz reconoce un propósito (Rico, 1995).

En este paradigma surge, como reacción al aprendizaje por descubrimiento de Bruner, el aprendizaje significativo brindado por David Ausubel (1976). Ausubel desarrolla en los años 70 una teoría que complementa a la de Bruner, pues considera que, aunque el aprendizaje no puede reducirse a la memorización de datos, el descubrimiento no es factible como única fuente de aprendizaje. Así, su teoría distingue dos dimensiones en el proceso de aprendizaje (Díaz Barriga Arceo, Hernández Rojas y García González, 2002):

- La primera es la relativa a la forma en que se adquiere el conocimiento. Nos encontramos con aprendizajes por repetición y por descubrimiento.

- La segunda es relativa a la forma en que el conocimiento es asimilado en la estructura de conocimiento del alumno, es decir, en la realidad cognitiva que ha construido. El tipo de aprendizaje respecto a esta dimensión sería el aprendizaje significativo

Ahora bien, aprendizaje significativo, aprendizaje por repetición, y aprendizaje por descubrimiento no serían categorías cerradas, sino que se podría organizar estas situaciones de aprendizaje situándolas en un plano cartesiano en el que los ejes son las dos dimensiones del proceso de aprendizaje (Chadwick, 1999; Díaz Barriga Arceo et al., 2002), como podemos ver en la *Figura 9*.

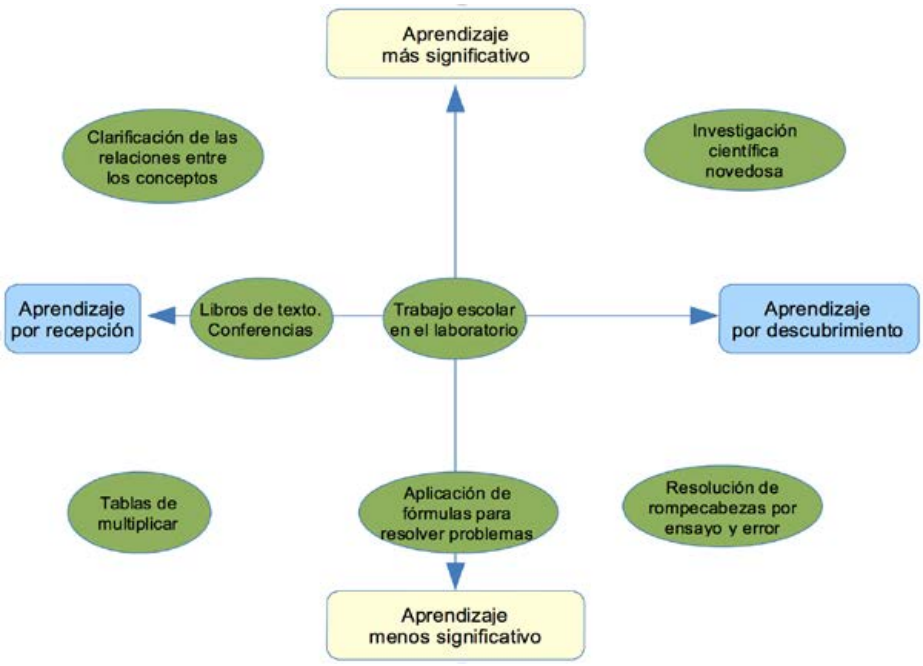


Figura 9. Representación de la organización del aprendizaje significativo, el aprendizaje por repetición, y el aprendizaje por descubrimiento.

Fuente: (Chadwick, 1999; Díaz Barriga Arceo et al., 2002)

Según Ausubel (1976), el estudiante organiza objetos, hechos y conceptos y sus interrelaciones de un modo jerárquico. De este modo, los hechos y proposiciones subordinados, es decir, aquéllos que están más bien contenidos en otros que son continentes, se integran dentro de los supraordinados, que son ideas más inclusivas y generales. En algunas ocasiones introduciremos contenidos en este esquema de forma que se incluya dentro de otros conocimientos ya existentes, en otras formará un conjunto con ellos, y en otras se aprenderán conceptos coordinados, es decir, pertenecientes al mismo nivel jerárquico. Es fundamental tener en cuenta las relaciones entre los conocimientos para que éstos encuentren un lugar natural en el esquema mental del estudiante. Cuando se encuentra ese lugar y se integra el conocimiento en la estructura jerárquica cognitiva del alumno, incluyendo las conexiones que lo relacionan con los conocimientos supraordinados, subordinados y coordinados de la misma, es cuando podemos hablar de aprendizaje significativo (Díaz Barriga Arceo et al., 2002).

Naturalmente, la capacidad de asimilar los conocimientos por parte de un estudiante de bachillerato no es en absoluto parecida a la que posee un estudiante de infantil. Consciente de ello, Jean Piaget (1972) uno de los principales autores cognitivo-constructivista, crea su teoría del desarrollo cognitivo, la cual sigue en vigor.

Hasta mitad del siglo XX, se consideraba que la mente del niño recién nacido está en blanco (tabula rasa) en la que se van inscribiendo gradualmente las experiencias vividas. Son pues agentes inactivos en su propio aprendizaje. Además, el pensamiento general de la época afirmaba que sin lenguaje no se puede desarrollar pensamiento abstracto. Aunque autores como Leibniz ya habían hecho críticas al empirismo, no es hasta las últimas décadas del siglo XX donde ha sido posible demostrar que los bebés son seres dotados de un número de capacidades mediante las cuales exploran activamente su entorno y desarrollan sus estructuras cognitivas (Bransford et al., 2000).

Piaget se separa de la teoría de la tabula rasa. Concluye que el desarrollo cognitivo atraviesa una serie de etapas, en las que toman parte diferentes esquemas cognitivos. Las edades que las delimitan no son barreras infranqueables, sino medias experimentales (Piaget y Fernández Buey, 1972):

- Etapa sensoriomotora. 0-2 años. Los niños no piensan mediante conceptos, no tienen una representación interna de los acontecimientos externos a él. No comprenden la continuidad de la existencia de los objetos (o personas) fuera del alcance de sus sentidos. Esa comprensión se va adquiriendo hacia los 9-10 meses. El niño comienza a comprender las relaciones causales, empezando por aquéllas en las que la causa es él mismo. La enorme importancia que juegan los sentidos en esta etapa se pone de manifiesto en los experimentos por Hatwell (1985), en los que los niños ciegos de nacimiento arrastran un desfase considerable (entre 3 y 4 años) con niños videntes y ciegos tardíos en el desarrollo de las capacidades más generales asociadas a esta etapa.

- Etapa preoperacional. 2-7 años. El niño interactúa con el mundo mediante el uso de palabras e imágenes mentales. Una de las características de esta etapa es que el niño cree que todas las demás personas ven el mundo de la misma manera que él, luego son incapaces de ponerse en el lugar de otros. Este egocentrismo también provoca una limitación a la hora de trabajar en equipo y para la conversación con otros niños. También interpretan el mundo de una forma animista: las cosas inanimadas están dotadas de pensamiento e intencionalidad. En esta etapa desarrollan el pensamiento simbólico, imitan objetos de conducta, utilizan el juego simbólico, dibujan, etc. En definitiva, adquieren pensamiento abstracto. Otro aspecto importante en esta etapa es el de la conservación, que es la capacidad de entender que aunque la forma cambie, la cantidad no cambia cuando lo hace la forma. Por ejemplo:

  - Si una fila de fichas está dispuesta con éstas más separadas, entonces posee más fichas. Esto se debe a la incapacidad del niño para entender la reversibilidad y a que el niño se centra solo en un aspecto del estímulo.

  - La distancia entre dos puntos A y B no es igual que la distancia entre B y A, especialmente si el camino posee pendiente.

- Etapa de las operaciones concretas. 7-12 años. Comienza a desarrollar el razonamiento lógico, el cual solo son capaces de aplicar a problemas concretos o reales. Los objetos que no es capaz de percibir con los sentidos (bien porque no lo han hecho, bien porque son objetos imaginados) escapan a dicha capacidad de razonamiento. El niño es capaz de ordenar los objetos en conjuntos abstractos. Lo más importante, aparecen las operaciones: reuniones y disociaciones de clases, encadenamiento de relaciones A<B<C, correspondencias (esto da lugar a la idea de los números como entes abstractos). Estas operaciones, sin embargo, solo se aplican a objetos o eventos del presente inmediato (Piaget, 2008) y no a proposiciones verbales, es decir, hipótesis abstractas sobre las que cabría aplicar relaciones lógicas. Por esta razón se denominan concretas.

- Etapa de operaciones formales. Comienza a los 12 años. Aparece la capacidad de razonar no usando solamente objetos, sino hipótesis a partir de las cuales se pueden extraer consecuencias independientemente de la verdad o falsedad de las premisas. Estos razonamientos se efectúan en virtud de las nuevas operaciones formales adquiridas por el niño: implicaciones, conjunciones, disyunciones, incompatibilidades, etc. Se asiste, pues, al nacimiento del pensamiento lógico abstracto.

Estas etapas, si bien son comunes a todos los niños, no son compartimentos cerrados de capacidades. Un niño puede pertenecer a una cierta etapa en lo relativo a una determinada capacidad cognitiva, y en otra etapa distinta para una capacidad distinta. Además, aunque a través de fenomenología muy mal conocida, parece que estas etapas del desarrollo cognitivo, especialmente en los aspectos de adquisición de estructuras mentales, se relacionan con la maduración del sistema nervioso del niño, el cual termina a los 15-16 años (Piaget y Fernández Buey, 1972).

Asimismo, las diferencias entre entornos socioculturales diferentes y las diferencias intrínsecas de cada individuo provocan que la velocidad de desarrollo varíe de un niño a otro, como se pone de manifiesto en múltiples estudios:

- En Irán, aparecen notables diferencias entre niños de la capital y niños analfabetos de poblados (Piaget, 2008).

- Según Lynn (2010), existe un salto cualitativo en las capacidades matemáticas (evaluadas en el informe PISA) de los niños del norte de Italia respecto a sus compatriotas del sur.

Aunque en los ejemplos expuestos no es clara la existencia del factor social, sí son significativa la presencia de estas diferencias. En cualquier caso, aunque los límites de edades para cada etapa sean flexibles, su orden nunca lo es (Piaget, 2008).

Analizando el estudio de Piaget se puede concluir que es entre los 7 y los 11 años (desde el primer al quinto curso de la Enseñanza Primaria Española) donde se adquiere la capacidad de realizar operaciones lógicas, y a partir de los 11-12 años (quinto y sexto curso de Primaria, y desde primero de la Educación Secundaria) donde se desarrolla la capacidad de abstracción. En el cuarto curso de la Educación Secundaria Obligatoria (15-16 años) y en primer curso de Bachillerato (16-17 años), las capacidades cognitivas del alumno medio han alcanzado el nivel de uso de operaciones formales, el mayor nivel de abstracción que se espera de ellos.

4.1.4. Construccionismo

El construccionismo es desarrollado por Papert (1980) tomando los principios constructivistas y asumiendo las teorías de autores como Wallon (1980), que afirman que el hombre es genéticamente social y que la construcción del conocimiento solo es posible desde la interacción social del sujeto.

En el aprendizaje construccionista, los estudiantes construyen modelos mentales para entender el mundo que les rodea, y trasladan esos modelos del ámbito cognitivo al ámbito real, a través de la manipulación y la construcción de artefactos. El construccionismo defiende el aprendizaje centrado en el estudiante, que utiliza la información que ya tiene para adquirir más conocimiento (Alesandrini y Larson, 2002).

Los estudiantes aprenden a través de la participación en el aprendizaje basado en proyectos donde hacen conexiones entre diferentes ideas y áreas de conocimiento facilitadas por el profesor a través de la mentorización, en lugar de usar charlas magistrales, o tutoriales paso a paso (Alesandrini y Larson, 2002). Es decir, el profesor construccionista es un mediador y un facilitador que ayuda a los estudiantes a alcanzar sus objetivos, y no es un instructor o un conferenciante (Harel y Papert, 1991).

Además, el construccionismo sostiene que el aprendizaje puede ocurrir más efectivamente cuando las personas son activas en la fabricación de objetos tangibles en el mundo real. En este sentido, el construccionismo está conectado con el aprendizaje experiencial y se basa en la teoría epistemológica de Jean Piaget del constructivismo (Cakir, 2008). En definitiva, el aprendizaje construccionista implica que los estudiantes saquen sus propias conclusiones a través de la experimentación creativa y la interacción social (Harel y Papert, 1991).

Para desarrollar la teoría construccionista, Papert, junto con Daniel G. Bobrow, Wally Feurzeig, y Cynthia Solomon, crean el entorno de programación Logo, que fue concebido como un instrumento de aprendizaje y de desarrollo de la creatividad.

Logo es una adaptación multiparadigma del lenguaje de programación LISP (que sigue el paradigma de programación funcional) (Harvey, 1997). Posteriormente se han desarrollado versiones de Logo con otros lenguajes de programación como Python, e incluso hay intérpretes de Logo en español, como MSWLogo (Harvey, 2002; Sacristan, 2002) o Logo Writer (Cavallo, 2011). Dando las instrucciones apropiadas través del lenguaje de programación, el usuario de Logo puede hacer moverse a un pequeño robot representado por un triángulo (lo que el entorno llama “tortuga”) que en su recorrido va trazando líneas por la pantalla.

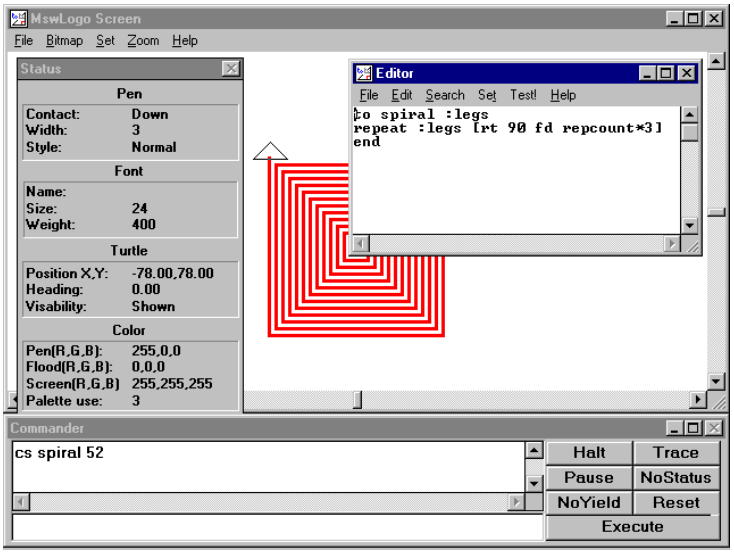


Figura 10. Ejemplo de programa realizado con MswLogo.

Fuente: (Harvey, 2002)

Clements (1986) clasifica Logo como un espacio de aprendizaje extensible (los programas a realizar no están predefinidos inicialmente) pensado para desarrollar

las capacidades metacognitivas a través de la interacción y la sistematización de procedimientos.

Papert (1980) se basó en los principios constructivistas de Piaget (2008) para el diseño de Logo, y también en las aportaciones de autores como Vygotsky (1986) que defienden que los procesos mentales están condicionados por el lenguaje y las herramientas que se utilizan para aprender.

Para Papert (1980), el usuario de Logo aprende reflexionando sobre cómo tiene que afrontar la tarea que tiene que realizar, haciéndola, y aprendiendo de los fallos cometidos en el proceso de consecución. Para esto último, Logo cuenta con un entorno de depuración que muestra los errores cometidos, y orienta sobre cómo corregirlos. Este proceso de aprendizaje a través del ensayo y error, eje central en Logo, bebe también de las teorías conductistas (si el usuario ha escrito el código correctamente, tendrá la satisfacción de ver a la tortuga moverse, sino, verá el mensaje de error en el depurador), y asume que el aprendiz, al identificar sus errores y comprender el porqué de éstos, reorganiza sus esquemas cognitivos.

Diversos autores han analizado Logo en profundidad, destacando por su completitud el trabajo de Maddux y Johnson (1997). También se pueden citar como referentes las investigaciones sobre la efectividad de esta herramienta en ámbitos como el estudio de la geometría (Battista y Clements, 1988; Clements y Meredith, 1993; Palumbo, 1990), en la comprensión oral y la adquisición de vocabulario (Robinson, Gilley y Uhlig, 1988), en la aceleración del paso de la etapa preoperacional a la operacional concreta y formal propia de la teoría de Piaget (Hines, 1983), y en el aprendizaje basado en reglas para la resolución de problemas a través de métodos algorítmicos (Gorman y Bourne, 1983). Igualmente existen autores (Hamada, 1987; Olive, 1991) que en sus estudios concluyen que esas mejoras se centran en ámbitos muy concretos, pero que cuando Logo se traslada a contextos más generales, se muestra ineficaz como instrumento de aprendizaje.

Los detractores de Logo marcaron la tendencia, y los lenguajes de programación desaparecieron de las aulas a principios de la década de 1990. Sin embargo, el trabajo de Papert sirvió de base e inspiración para el trabajo de otros autores, (por ejemplo, Michael Resnick, uno de los padres de Scratch), que ha supuesto el resurgir actual de la programación en las etapas escolares más tempranas.

4.1.5. Conectivismo

El conectivismo es una teoría desarrollada por Siemens (2005) y Downes (2005) para dar respuesta a las carencias que encontraron en las teorías de aprendizaje

clásicas en el contexto de la sociedad actual donde la presencia de los entornos digitales está tan presente (Castells, 2006), y donde el conocimiento crece exponencialmente (Siemens, 2005).

De forma sintética, se podría decir que el conectivismo concibe el aprendizaje como un proceso en el que el sujeto establece una red de conexiones entre diferentes fuentes de información especializadas. El mantenimiento y la actualización de esas fuentes es la columna vertebral del aprendizaje desde la perspectiva conectivista. Las conexiones que permiten aprender más tienen mayor importancia que el estado actual del conocimiento del sujeto (Siemens, 2005), o dicho de otra manera, la capacidad para que ese sujeto pueda aprender lo que necesite en el futuro es más importante que el conocimiento del que dispone actualmente.

En palabras de Siemens (2005), traducido por Leal (2007, p. 6), los principios en los que se basa el conectivismo son los siguientes:

> "*Principios del conectivismo:*
>
> - *El aprendizaje y el conocimiento dependen de la diversidad de opiniones.*
> - *El aprendizaje es un proceso de conectar nodos o fuentes de información especializados.*
> - *El aprendizaje puede residir en dispositivos no humanos.*
> - *La capacidad de saber más es más crítica que aquello que se sabe en un momento dado.*
> - *La alimentación y mantenimiento de las conexiones es necesaria para facilitar el aprendizaje continuo.*
> - *La habilidad de ver conexiones entre áreas, ideas y concepto s es una habilidad clave.*
> - *La actualización (conocimiento preciso y actual) es la intención de todas las actividades conectivistas de aprendizaje.*
> - *La toma de decisiones es, en sí misma, un proceso de aprendizaje. El acto de escoger qué aprender y el significado de la información que se recibe, es visto a través del lente de una realidad cambiante. Una decisión correcta hoy, puede estar equivocada mañana debido a alteraciones en el entorno informativo que afecta la decisión."*

Hay autores como Verhagen (2006) que no consideran el conectivismo como una teoría de aprendizaje, dado que éstas versan sobre cómo aprenden las personas, mientras que el conectivismo se centra en el qué y el porqué de ese aprendizaje. Otros autores como Kerr (2007) afirman que las teorías de aprendizaje tradicionales son suficientes, y que ya resuelven los retos para los que supuestamente el conectivismo está planteado.

4.1.6. Resumen de las teorías de aprendizaje

Como conclusión a este capítulo se recogen y resumen los conceptos más significativos de cada teoría:

Tabla 4
Resumen de los conceptos clave de las teorías de aprendizaje descritas en el capítulo 4.1.

	PRINCIPALES TEORÍAS DE APRENDIZAJE			**OTRAS TEORÍAS**	
	CONDUCTISMO	**COGNITIVISMO**	**CONSTRUCTIVISMO**	**CONSTRUCCIONISMO**	**CONECTIVISMO**
Forma en la que ocurre el aprendizaje	Provocado por estímulo / respuesta	Recepción activa Organizadores previos Organizadores secuenciales	El estudiante construye sus esquemas de conocimiento a través de la interacción con la realidad / sociedad	Traslación de los conceptos mentales a la realidad tangible	Conocimiento distribuido en una red Acceso a la información a través de la tecnológica Reconocimiento e interpretación de patrones
Factores que determinan el aprendizaje	El tipo de recompensa o castigo El tipo de estímulo	Atención y motivación Asociación con conocimientos previos Estructuración de los nuevos conocimientos	El alumno dispone de las herramientas que le permiten construir el conocimiento	Manipulación Contexto social	Cantidad y variedad de la información disponible Creación de conexiones Se prima la adquisición de la capacidad para aprender conocimientos futuros
Papel que juega la memoria	Se repiten las experiencias premiadas, se evita las castigadas	Codificación, almacenamiento, y recuperación de la información	Memoria en permanente "construcción" Uso flexible de conocimientos previos Dependiente del contexto	Aprender haciendo, se recuerda mejor lo que se experimenta	Patrones de adaptación Representante del estado actual que existente en las redes
Tipos de aprendizajes que ejemplifican esta teoría	Basado en tareas	Resolución de problemas Procesamiento de la información	Aprendizaje social	Aprendizaje basado en proyectos Aprendizaje social	El aprendizaje complejo, el núcleo de un rápido cambio, diversas fuentes de conocimiento

	PRINCIPALES TEORÍAS DE APRENDIZAJE			OTRAS TEORÍAS	
	CONDUCTISMO	COGNITIVISMO	CONSTRUCTIVISMO	CONSTRUCCIONISMO	CONECTIVISMO
Cómo se relacionan el docente y el alumno	El estudiante tiene una actitud pasiva El docente recrea situaciones y las dirige	El estudiante es el protagonista El docente es un mediador y un guía	Estudiante activo, construye su propio aprendizaje Docente facilitador, se relaciona con el estudiante como un igual	El estudiante es el protagonista El docente es un mediador y un guía	Estudiante activo, valora sus necesidades de aprendizaje, observa, emula y actualiza sus conocimientos. El docente es un líder motivador que enseña a evaluar y a organizar la información

Fuente: elaboración propia

4.2. Aprender a resolver problemas

Uno de los temas que las distintas teorías de aprendizaje abordan, es la forma en la que se puede adquirir la capacidad para resolver problemas. Además, como hemos visto en el inicio del capítulo 4, resolver problemas es parte intrínseca de lo que significa programar. Este capítulo se centra en este aspecto, y presenta las diferentes formas en las que se puede abordar la resolución de un problema.

Monereo (2000) pone de manifiesto la confusión que existe en la terminología aplicada a la actividad de aprendizaje, pues muy frecuentemente se intercambian expresiones tales como capacidad y habilidad, procedimientos y métodos, etc. Él propone las siguientes definiciones:

- Capacidad: disposición de tipo genético que se pueden desarrollar, dando lugar a las habilidades.
- Habilidad: el uso de una capacidad en un determinado contexto o conducta. Puede ser tanto inconsciente (una persona con la capacidad de oír puede ser más hábil que otra en distinguir las notas que emite un piano) como consciente (una persona puede tener mayor habilidad manual que otra a la hora de construir un armario). Las habilidades se manifiestan mediante procedimientos.
- Estrategia: se diferencia de la habilidad en que siempre es consciente. Más adelante en este mismo capítulo se desarrolla el concepto de aprendizaje de estrategias, clave para Monereo.
- Un procedimiento, destreza, técnica o método es un conjunto de acciones ordenadas que tienen como finalidad la consecución de una meta.

Son varios los autores (Bransford y Stein, 1986; Pressley, Willoughby, Woloshyn y Wood, 1990; Valls, 1993) que defienden que existen dos formas de categorizar procedimientos: en cuanto al tipo de reglas que sustentan el procedimiento y en cuanto al tipo de objetivo que persiguen.

Según el primer criterio, tenemos procedimientos algorítmicos y heurísticos:

- Los procedimientos algorítmicos o técnicas se componen de pasos estrictos que siempre se aplican de la misma forma, y que garantizan la consecución de la meta.
- Los procedimientos heurísticos o estrategias no tienen reglas fijas y no siempre se obtiene un resultado óptimo a partir de su aplicación. Por ejemplo, descomponer un problema en problemas más pequeños y manipulables.

Según el segundo criterio, tenemos procedimientos disciplinares e interdisciplinares:

- Los procedimientos disciplinares son los propios de cada disciplina. Por ejemplo, la división de polinomios por la regla de Ruffini.
- Los procedimientos interdisciplinares o de aprendizaje. Por ejemplo, el subrayado o la capacidad para resumir un texto.

Por último, un método es un conjunto de técnicas que tienen un contexto común. Por ejemplo, existen distintos métodos de mecanografía, compuestos por un número de técnicas congruentes en su forma.

La RAE (2014) en una de sus acepciones, define problema como “Planteamiento de una situación cuya respuesta desconocida debe obtenerse a través de métodos científicos”. Uniendo esta definición con los planteamientos de Monereo previamente expuestos, un problema trata de trazar una estrategia y determinar una serie de métodos con el objetivo de encontrar una solución a una determinada situación.

Moursund (2003) en sus estudios encuentra una serie elementos que habitualmente están presentes en cualquier problema:

- Un punto de partida.
- Un objetivo final.
- Un conjunto de medios o instrumentos a disposición de quien vaya a tratar de resolver el problema, así como las condiciones y restricciones en las que estos medios se pueden o no utilizar.

- La experiencia, la habilidad y la motivación que caracteriza a la persona o personas que se van a encargar de afrontar el problema. A este contexto Moursund lo denomina dominio.

También es común que la resolución de un problema no se haga en un único paso, sino que se descomponga en etapas con objetivos más pequeños que, en su resolución conjunta, supongan la consecución del objetivo final. En cada una de estas etapas se llevarán a cabo una serie de acciones de tipo conductual y/o cognoscitivo, que modificarán el estado de dicha etapa y la harán evolucionar a la siguiente (Schunk, 1997).

Cada tipo de problema suele tener una serie de métodos concretos, propios del ámbito de la situación a resolver. Por ejemplo, un problema de matemáticas se resuelve utilizando métodos matemáticos. No obstante, son varios los autores, como Schunk (1997) o Woolfolk (1999), que afirman que se pueden utilizar con éxito procedimientos y estrategias de carácter general para la resolución de problemas, independientemente el ámbito de éstos.

López García (2009) enumera algunas de estas estrategias generales, que se recogen y desarrollan a continuación:

- ***Ensayo y error***: Está basado en probar a realizar diferentes acciones hasta conseguir una solución, aunque puede que esta solución nunca se llegue a conseguir, o que conseguirla suponga invertir una enorme cantidad de tiempo. Este tipo de estrategias son adecuadas cuando el número de acciones posibles es pequeño y se puede intentar llevar a cabo todas ellas, comenzando con la que se considere con más posibilidades de ser exitosa.

- ***Iluminación:*** Supone llegar hasta la solución de un problema de forma repentina. Wallas (1921) descompone el proceso que lleva a la iluminación, al que denomina proceso creativo, en cuatro fases (Paniagua Arís, 2001):
    - Preparación, fase en la que se contextualiza el problema, se define el objetivo, y se reúne toda la información necesaria para afrontar el proceso.
    - Incubación, fase en la que se reflexiona y se madura la información recopilada en el paso anterior.
    - Iluminación, fase en la que surgen las ideas que posibilitarán conseguir el objetivo que solucionará el problema.
    - Verificación, fase en la que se comprueba que la solución alcanzada es válida, en base a unos mecanismos de evaluación que habitualmente se establecen en la fase de preparación.

- ***Heurística:*** Consiste en utilizar una serie de métodos empíricos para hacer posible un hallazgo, entendiendo método empírico como aquél que tiene una serie de indicadores asociados para medir si ha cumplido con su propósito. Son diversos los autores que proponen métodos concretos para resolver un problema a través de una estrategia heurística. En este capítulo se abordan los modelos de Bransford y Stein, y del matemático húngaro George Polya.

  Bransford y Stein (1984) proponen el modelo conocido como "IDEAL", llamado así por las iniciales de las etapas que lo componen:
  - Identificar el problema.
  - Definir y presentar el problema.
  - Explorar las estrategias viables.
  - Avanzar en las estrategias.
  - Lograr la solución y volver para evaluar los efectos de las actividades.

  El modelo de Polya (1957) también propone una serie de etapas, y además plantea las preguntas que se deben poder responder antes de avanzar a la siguiente fase. A continuación se describen los procesos mentales que se atraviesan al resolver un problema siguiendo la estrategia heurística propuesta por Polya, y un ejemplo de las preguntas que se podrían plantear para poder progresar en el proceso (Boscán Mieles y Klever Montero, 2012):
  - Entender el problema: ¿Qué es lo que se debe resolver? ¿Cuáles son las condiciones y los datos de los que se dispone?
  - Trazar un plan: ¿Se conocen referentes, o problemas similares a este? Si la respuesta es sí, ¿se conocen los métodos que se utilizaron para resolverlos? O en su defecto, ¿se conocen mecanismos que pudieran ser aplicables a la resolución de este problema?
  - Ejecutar el plan: llevar a cabo cada uno de los pasos de la estrategia escogida. ¿Es cada paso correcto, y sirve para avanzar hacia el siguiente?
  - Revisar el resultado: ¿es el resultado satisfactorio? ¿Cumple con los objetivos y con las expectativas propuestas?

  Son varios los autores (Wilson, Fernández y Hadaway, 1993) que defienden que el modelo de Polya no tiene por qué ser lineal, habiendo margen para alterar el orden de los pasos. Para ilustrar esta flexibilidad, a continuación se expone un ejemplo llevado al ámbito de la programación. En la manera de plantear una solución a un problema, un programador puede decidir que necesita entenderlo mejor y regresar a la etapa de análisis de requisitos; o si cuando ha trazado un camino y trata de seguirlo, no encuentra la manera de recorrerlo, concluye que la siguiente tarea será diseñar un nuevo plan, o darle otro enfoque (Guzdial, 2000; Wilson et al., 1993).

- ***Algoritmia:*** Consiste en definir una lista de procedimientos a seguir en un orden muy concreto, que por lo general garantizan llegar a una resolución correcta del problema.

  La RAE (2014) define algoritmo como “conjunto ordenado y finito de operaciones que permite hallar la solución de un problema”.

  Hay una tendencia generalizada a asociar algoritmo con algo exclusivamente tecnológico, pero no es así. Podemos entender el concepto de algoritmo si tratamos de desligarlo de la componente informática, y nos quedamos con la parte de la definición que se refiere a la lista de pasos que tengo que seguir para resolver un planteamiento. En esta línea se expone a continuación un ejemplo de algoritmo aplicado a una situación de la vida cotidiana:

  Por ejemplo: “¿qué se necesitaría desde un principio para hacer una tortilla?”.

  Esta sería una posible respuesta a la propuesta realizada:

  Algoritmo para hacer una tortilla:
  1. Entro en la cocina.
  2. Saco un huevo de la nevera.
  3. Bato el huevo.
  4. Cojo una sartén y echo un poco de aceite.
  5. Enciendo el fuego y coloco la sartén.
  6. Cuajo la tortilla.
  7. Me como la tortilla.

  Aunque los pasos básicos a seguir están planteados en la forma y en el orden correcto, se pueden encontrar ciertas fisuras que pueden hacer que el algoritmo falle en un punto, y no le permita continuar con los siguientes. Por ejemplo, ¿qué sucede si…?
  El algoritmo será adecuado en la medida en la que contemple todos los caminos que permitan llegar desde el planteamiento inicial hasta la meta en el contexto definido. Siguiendo con el ejemplo anterior, el algoritmo se mejoraría de la siguiente manera:

  Algoritmo para hacer una tortilla 2.0:
  1. Entro en la cocina.
  2. Compruebo si hay huevos en el frigorífico.
  3. Si hay huevos en el frigorífico, saco un huevo y salto al paso 8
  4. Si hay huevos en el frigorífico, voy a comprarlos.
  5. Para comprar los huevos, cojo dinero.

6. Si tengo dinero suficiente, compro los huevos y voy al paso 8
7. Si no tengo dinero suficiente, no se puede hacer la tortilla (fin).
8. Compruebo si puedo encender el fuego de la cocina.
9. Si puedo encender el fuego, enciendo el fuego y voy al paso 11.
10. Si puedo encender el fuego, no se puede hacer la tortilla (fin).
11. Busco una sartén apropiada para hacer una tortilla.
12. Si no encuentro una sartén, no se puede hacer la tortilla (fin).
13. Etc.

Quizás parezca excesivo que el objetivo sea definir absolutamente todos los supuestos, y es posible que sea improbable que algunos sucedan, pero cuantos más escenarios se contemplen y se les busque solución, más completo será el algoritmo.

- **Análisis de medios y fines (AMF):** Modelo planteado por Newell y Simon (1972) que consiste en comparar la situación inicial en la que se plantea el problema, con la solución que se quiere alcanzar, para reconocer las diferencias entre una y otra, y buscar la manera de reducirlas. Posteriormente se trata de identificar los procesos y operaciones intermedias por las que habrá que pasar para alcanzar la solución. Newell y Simon proponen para esto el modelo de procesamiento de información, descrito en el siguiente apartado.

- **Modelo de procesamiento de información:** Continuando con la teoría de análisis de medios y fines, Newell y Simon (1972) realizan un planteamiento que se basa en definir varios estados para un problema: estado inicial, estado final, y estados intermedios hacia la solución. Asimismo, aportan tres conceptos clave donde se aplican estos estados:
    - Sistema de procesamiento de la información: es la persona o personas que se enfrentan a un problema. (Newell y Simon, 1972, p. 19).
    - Entorno de la tarea: se refiere a la concreción del problema y su contexto, incluyendo la descripción tanto del estado inicial del problema como del final (la meta u objetivo a conseguir). Aquí también se incluirían las restricciones propias del problema, y las limitaciones de la persona que va a resolverlo (Voss, 1989).
    - Espacio del problema: es la representación interna del entorno de la tarea utilizado por la persona o personas que solucionarán el problema (Newell y Simon, 1972, p. 789). El espacio del problema también se refiere al conocimiento de la persona sobre lo que es relevante para la interpretación del problema y la búsqueda de la solución.

Los estados intermedios hacia la solución requieren de la realización de procesos en cada uno de ellos. Newell y Simon afirman que un conjunto

relativamente pequeño de procesos elementales bastaría para procesar toda la información (Newell y Simon, 1972, p. 29).

A continuación, se muestra una gráfica que ilustra el modelo planteado por Newell y Simmons basado en la investigación de Rahikainen (2002, p. 25).

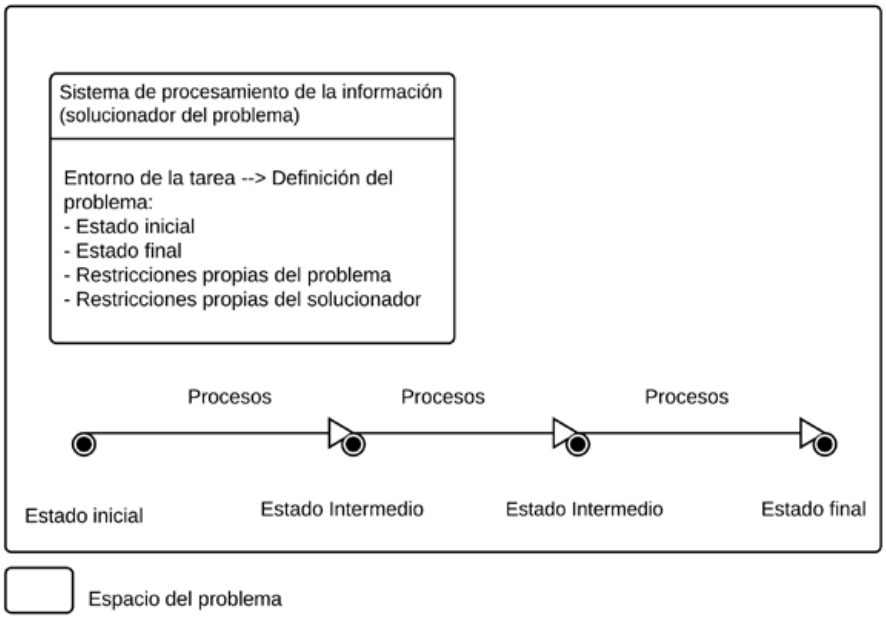


Figura 11. Representación del modelo de procesamiento de información de Newell y Simon.

Fuente: elaboración propia.

- ***Razonamiento analógico*:** Razonar por analogía significa entender un concepto apoyándose en el conocimiento que ya se tiene de otros conceptos parecidos (Cubillo y González Labra, 1998). Aplicado a la resolución de problemas, significa comparar la situación a solucionar con otras similares que se consiguieron resolver, buscando las similitudes y evaluando si son aplicables al nuevo problema. Son varios los autores que relacionan conceptualmente razonamiento analógico con la inteligencia y la capacidad heurística (Allen y Butler, 1996; Blum, Facundo Abal, Lozzia, Picón Janeiro y Attorresi, 2011; Resnick y Glaser, 1975).

- ***Lluvia de ideas:*** Se atribuye la creación de esta teoría a Alex F. Osborn (1953), quien describe el concepto de *Brainstorming* en su libro *Applied Imagination*. La lluvia o tormenta de ideas (traducción literal de *Brainstorming*) consiste en proponer soluciones factibles a un problema.

Otros autores (Howard, Dekoninck y Culley, 2010; Jones, 1980), han desarrollado en sus estudios las bases del Brainstorming, o lo han enlazado con otras teorías, como la analogía para la generación de nuevas ideas (Wilson, Rosen, Nelson y Yen, 2010).

Mayer (1992), a partir del trabajo de Osborn, propone estructurar el modelo en las siguientes etapas:

- Acotar el problema, lo que Osborn denomina *Fact-finding* (buscar los hechos). En esta fase también se reúnen y analizan los datos relevantes al problema.
- Proponer gran cantidad de posibles soluciones, lo que Osborn denomina *Idea-finding* (encontrar la idea). En esta fase es fundamental que no se emitan juicios ni se valoren las posibles soluciones hasta que no estén todas formuladas.
- Establecer los criterios para valorar las soluciones propuestas.
- Utilizar estos criterios para escoger la solución óptima, lo que Osborn denomina *Solution-finding* (encontrar la solución).

- ***Sistemas de producción:*** Este concepto es introducido por Anderson (1990) en su teoría de procesamiento de información. Schunk (1997), a partir de los argumentos de Anderson, defiende que para resolver un problema se utiliza lo que denominan sistemas de producción. Un sistema de producción fija un conjunto de reglas donde se establecen unas condiciones que, si se cumplen, desencadenan acciones asociadas. Utilizando la terminología de Schunk (1997, p. 247), un sistema de producción está formado por:
  - Antecedentes, que incluye la definición del objetivo a lograr y los enunciados de prueba.
  - Proposiciones condicionales, que son las condiciones establecidas.
  - Consecuencias, que son las acciones a realizar si se cumple una condición.

  El recorrido por esta secuencia de condiciones y acciones lleva a la solución del problema, atravesando distintos estados donde cada condición determina el camino a seguir para llegar al siguiente estado.

- ***Pensamiento lateral***: Este concepto ha sido popularizado por Edward De Bono (1986). El pensamiento lateral difiere de otros tipos de pensamientos verticales o lógicos. El pensamiento vertical rechaza otros caminos para escoger el camino a seguir en la resolución de un problema. Por el contrario, el pensamiento lateral no selecciona un camino concreto, sino que intenta seguir todos los caminos, y en base a la experiencia acumulada en estos recorridos, trata de definir uno

nuevo. Dicho con las propias palabras de De Bono (1986, p. 47), “el pensamiento vertical es selectivo, el pensamiento lateral es creador”.

De Bono propone una categorización de los problemas a efectos de su solución por el pensamiento lateral (De Bono, 1986, p. 67):

- Problemas donde la información es escasa, o donde se necesitan técnicas más eficaces para su manejo.
- Problemas donde no se precisan datos adicionales a los ya disponibles, sino una reorganización de la información disponible.
- Problemas donde hay ausencia de problema, es decir, el problema no es evidente, no está acotado del todo, o no plantea una situación en la que necesariamente haya que realizar acciones de mejora, aunque sí se contempla que es posible mejorar.

Esta catalogación también es un indicador de en qué casos puede ser útil escoger el pensamiento lateral como estrategia de resolución de problemas.

Más allá del aprendizaje de procedimientos para la resolución de problemas, todo profesor pretende estimular la capacidad de análisis (qué procedimiento utilizar en cada caso) y la reflexión acerca de los procedimientos mismos, cuáles son sus límites y por qué funcionan.

Según Monereo *et al.* (2000), una estrategia de aprendizaje es un “*proceso de toma de decisiones (conscientes e intencionales) en el cual el alumno elige y recupera, de manera coordinada, los conocimientos que necesita para cumplimentar una determinada demanda u objetivo, dependiendo de las características de la situación educativa en que se produce la acción”*. (p.27)

Un estudiante utiliza estrategias de aprendizaje cuando se adapta a las distintas situaciones en que se encuentra en el transcurso de una actividad con fin educativo, ya sean situaciones internas, estrictamente pertenecientes a la propia actividad, o externas, relacionadas con el entorno en el que dicha actividad se desarrolla.

Para favorecer el aprendizaje estratégico, o lo que es lo mismo, fomentar el desarrollo de estrategias de aprendizaje en los alumnos, Monereo *et al.* (2000) establecen tres grandes objetivos que deben estar presentes en el Diseño Curricular:

- Mejorar el conocimiento declarativo, según Gagné, citado por Monereo (2000), aquéllos susceptibles de ser transmitidos mediante el lenguaje verbal) y el procedimental (el conjunto de procedimientos que posee el

alumno), que incluye tanto los procedimientos disciplinares como los interdisciplinares.

- Hacer al alumno consciente sobre las operaciones y decisiones mentales que realiza cuando aprende un contenido o resuelve una tarea.

- Favorecer en el alumno la reflexión sobre las condiciones en que se efectúa una determinada actividad o se aprenden determinados contenidos, con el objetivo de transferir en lo posible las estrategias empleadas anteriormente a situaciones nuevas en las que se pueden reconocer condiciones similares.

Estos objetivos se pueden resumir en la frase empleada innumerables veces en educación “aprender a aprender”.

Para Monereo, existen cinco pautas que deberían tenerse en cuenta en toda actuación pedagógica, para que ésta fomente el aprendizaje estratégico:

- Las actividades que se presenten al alumno deben ser lo suficientemente complejas como para que este necesite planificar su actuación, supervise el proceso y evalúe el resultado al concluirlas.

- Para objetivos concretos, tratar de no utilizar técnicas de estudio simples, pues fomentan un aprendizaje mecánico y no significativo. El alumno debe ser consciente de los diferentes procedimientos de aprendizaje a su disposición y debe ser capaz de distinguir los menos útiles a la situación en que se encuentre.

- El aula debe ser un lugar en el que se fomente la reflexión, la duda y el intercambio de ideas sobre cómo se puede aprender sobre un tema.

- Las estrategias de aprendizaje deben ser transferibles a otras tareas y materias y, si es posible, a otros contextos.

- Las estrategias de aprendizaje que se enseñen han de estar acordes al contexto y las circunstancias de los alumnos.

Esta última pauta es especialmente relevante en esta investigación. Los alumnos en la actualidad viven inmersos en un universo de tecnología e información. La vertiginosa revolución tecnológica que caracteriza estos tiempos queda patente en la Ley de Moore, que afirma que el número de transistores por centímetro en circuitos integrados, y por tanto la potencia de los microchips, se duplica cada 18

meses (Monereo et al., 2000; Tuya, 2006). Esto hace que la tecnología crezca exponencialmente, y genera el efecto palpable en la sociedad que plasmara Peter Eio, presidente de Lego Systems hasta 2001, en su frase: “Por primera vez en la historia de la humanidad, una nueva generación está capacitada para utilizar la tecnología mejor que sus padres.”

La brecha intergeneracional se ha hecho realmente patente en la actualidad, y en virtud de la Ley de Moore, que se sigue cumpliendo a pesar de los estudios que vaticinaron su colapso (Winner, 2001), esta brecha se hará cada vez mayor.

Dispositivos como *smartphones* y *tablets*, juguetes electrónicos, o robots (Alonso de Castro, 2013; Sánchez-Prieto, Olmos-Migueláñez y García-Peñalvo, 2016), tienen como público niñas y niños cada vez más pequeños. Los jóvenes de hoy en día portan teléfonos móviles con conexión a internet que utilizan de múltiples formas: mostrar vídeos a sus amigos y grabar los suyos propios, escuchar música, jugar a videojuegos, chatear con sus amigos, etc. En casa tienen ordenadores y videoconsolas.

Así pues, como afirma Sánchez Rosal (2012) “*es pertinente ofrecerle al aprendiz espacios educativos análogos con su contexto social-cultural que le permita asimilar la información propuesta por los medios electrónicos*” (p.27).

Sartori, citado por Balardini (2002), habla del nuevo *homo videns*, los jóvenes *videoformados*, “cuyo saber se ubica fundamentalmente en la esfera de lo perceptivo concreto y su centro de atención primordial se encuentra en la imagen” (p.51). Para el homo videns la pantalla (tanto televisión como videoconsolas, móviles, ordenadores) es un medio no solo de entretenimiento, sino que es un elemento fundamental en su mundo social. Para estos autores la tecnología debe traspasar la barrera de lo puramente lúdico y la enseñanza debe ser permeable a lo que esta ofrece. Esto supone nuevos desafíos para los docentes (Tardif, 2012), que incluyen, por ejemplo, cómo definir actividades y contenidos apropiados para el desarrollo de niñas y niños de diferentes edades (Sánchez-Prieto et al., 2016).

Pero que la tecnología esté tan presente en el contexto niños y adolescentes no es el único motivo para que sean parte de su formación desde las primeras etapas educativas. El mundo actual está impulsado por el software (Manovich, 2013), por lo que la sociedad de hoy en día demanda profesionales cualificados para el sector empresarial de las TI (Tecnologías de la Información). Por este motivo la alfabetización digital se está incluyendo desde las etapas educativas más tempranas del desarrollo (Astrachan, Hambrusch, Peckham y Settle, 2009; Bers, Flannery, Kazakoff y Sullivan, 2014; Cejka, Rogers y Portsmore, 2006)

combinándola con otras competencias claves como la lectura, la escritura y las matemáticas.

Se puede considerar a la alfabetización digital como un primer nivel en el que se adquiere la capacidad de utilizar la tecnología en el ámbito personal (Sartori, 1998). Dominar la tecnología sería el siguiente nivel, y supondría poder utilizarla de forma independiente, y evolucionar en su conocimiento según la propia tecnología evolucione a lo largo de la vida profesional (Council, 1999).

Uno de los enfoques, cada vez con más presencia en la enseñanza de la alfabetización digital ha sido fomentar el aprendizaje de la programación (DiSessa, 2001; Prensky, 2008; Rushkoff, 2012; Vee, 2013), incluyendo el uso activo del pensamiento algorítmico propio de la programación para resolver problemas. Resnick *et al.* (2009) van un paso más allá, equiparando el estadio más elevado del dominio de la tecnología con saber programar.

Este enfoque establece que las capacidades que se presentan como las más efectivas en la creación de un programa informático van asociadas a una manera de razonar, que es útil no solo en el ámbito de las actividades cognitivas utilizadas en la programación, sino también en otros ámbitos (García-Peñalvo, Rees, Jormanainen, Tuul y Reimann, 2016). Rushkoff (2010) añade un punto de crítica social al hecho de aprender a programar con su célebre frase "*Program or be programmed*" ("Programar o ser programado") con la que resume la idea de que saber programar proporciona perspectiva, espíritu crítico y en definitiva, libertad, en un mundo cada vez más digital, lleno de aplicaciones cuyas reglas han sido programadas por otras personas.

Esto significa que existe una manera específica de pensar y organizar las ideas, que es apropiada para las habilidades computacionales debido a que promueve el análisis y la interrelación de ideas para la organización lógica y la representación de los procedimientos. Esas capacidades se pueden entrenar y mejorar desde las primeras etapas de aprendizaje, realizando las actividades apropiadas. Es decir, se trataría de fomentar un tipo de pensamiento específico, el llamado pensamiento computacional (Zapata-Ros, 2015).

4.3. Pensamiento computacional

Recuperando la definición de programación que se plantea al inicio del capítulo 4, podemos decir que programar es básicamente plantear la solución a un problema en los términos que el ordenador entiende, y describir esa solución con la sintaxis de un lenguaje. Este capítulo hace un recorrido por esa forma de pensar que se

debe tener para poder plantear la solución a un problema en los términos que el ordenador entiende, el denominado pensamiento computacional.

Wing (2006) populariza el término pensamiento computacional, al que relaciona con solucionar problemas, diseñar sistemas y comprender los principios de un comportamiento, apoyándose en los pilares en los que se sustenta la informática, y utilizando una amplia gama de herramientas mentales propias de esta ciencia.

Si tenemos en cuenta que la informática es el estudio de la computación, es decir, qué puede ser procesado y cómo procesarlo, el pensamiento computacional tiene las siguientes características (García-Peñalvo et al., 2016; Wing, 2006):

- El pensamiento computacional significa conceptualizar, no programar. La informática no solo codifica. Pensar como un ingeniero o ingeniera de software significa algo más que ser capaz de programar, requiere pensar a múltiples niveles de abstracción.
- El pensamiento computacional precisa de habilidades fundamentales, no de capacidad para memorizar. Una habilidad fundamental es algo que todo ser humano debe conocer para funcionar en la sociedad moderna.
- El pensamiento computacional trata sobre la forma de pensar de los seres humanos, no de los ordenadores. El pensamiento computacional se refiere a la forma en la que los seres humanos resuelven los problemas, pero no está tratando de hacer que los humanos piensen como ordenadores. Para resolver problemas, con o sin ordenadores, se necesita inteligencia, imaginación, creatividad, e incluso emoción. Así lo defienden diversos autores (De Bono, 1986; Polya y Zugazagoitia, 1965). Estas características son propias del ser humano, no de los ordenadores.
- El pensamiento computacional complementa y combina el pensamiento matemático y el pensamiento propio de la ingeniería. La informática se basa intrínsecamente en el pensamiento matemático, dado que, al igual que todas las ciencias, sus fundamentos formales descansan en las matemáticas. La informática se basa asimismo de forma intrínseca en el pensamiento propio de la ingeniería, dado que busca construir sistemas que interactúan con el mundo real.
- El pensamiento computacional se basa en ideas, no en herramientas.
- El pensamiento computacional es para todos, en todas partes. El pensamiento computacional será una realidad cuando esté tan integrado que desaparezca como una filosofía específica.

Wing (2011) revisa este tema y proporciona esta nueva definición: "El pensamiento computacional es el proceso de pensamiento involucrado en la formulación de los problemas y sus soluciones para que dichas soluciones estén representadas de

forma que puedan ser efectivamente realizadas por un agente de procesamiento de la información".

Autores como Aho (2006, p. 33) consideran al pensamiento computacional como el proceso mental que permite la formulación de las soluciones a un problema en los términos que plantea la algoritmia, es decir, en una secuencia de acciones ordenadas. Una parte importante en este proceso es encontrar modelos apropiados de cálculo con el que formular el problema y derivar sus soluciones.

Nunes (2011) concibe el pensamiento computacional como un proceso cognitivo que se utiliza como forma de encontrar algoritmos para resolver problemas, de tal manera que ese proceso, propio de la ciencia de la computación pueda ser aplicado a otras ciencias.

La Sociedad Internacional de la Tecnología en la Educación (ISTE) y la Asociación de Profesores de Informática (CSTA), han desarrollado una definición operativa de pensamiento computacional, que describe las características presentes en este tipo de pensamiento a la hora de resolver un problema (ISTE y CSTA, 2011). Esta definición pretende crear un marco de trabajo y un vocabulario común con el que los profesionales de la educación puedan trabajar (Moreno León, 2013). Según esta definición operativa, el pensamiento computacional incluye las siguientes características para resolver un problema:

- Plantear un problema de manera que el ordenador y otras herramientas puedan ayudar a solucionarlo.
- Organizar la información de forma lógica para analizarla.
- Representar de forma abstracta la información por medio de modelos y las simulaciones.
- Automatizar la solución del problema a través del pensamiento algorítmico
- Identificar, analizar e implementar posibles soluciones con el objetivo de lograr la combinación más efectiva y eficiente de pasos y recursos.
- Generalizar y transferir este proceso de resolución de problemas para que se pueda inferir a otro tipo de problemas.

Furber (2012) define el pensamiento computacional como el proceso de asociar elementos propios de la computación a otros ámbitos de la vida cotidiana, utilizando herramientas y técnicas propias de la informática para comprender el funcionamiento de las cosas.

Grover y Pea (2012, p. 29) destacan la falta de consenso internacional sobre una definición de pensamiento computacional, y enumeran una serie de elementos que servirían para evaluar el aprendizaje y el desarrollo del pensamiento computacional:

- Abstracciones y generalizaciones de patrones (incluyendo modelos y simulaciones).
- Tratamiento sistemático de la información.
- Sistemas de símbolos y representaciones.
- Nociones algorítmicas de flujo de control.
- Descomposición estructurada de problemas (modularización).
- Pensamiento iterativo, recursivo y paralelo.
- Lógica condicional.
- Eficiencia y limitaciones de rendimiento.
- Depuración y detección sistemática de errores.

Siguiendo con la perspectiva de Grover y Pea (2012), y tratando de identificar el elemento que mayor peso tiene el en pensamiento computacional, estos autores se decantan por la abstracción, de la que dicen: "*es definir patrones, generalizar a partir de casos concretos; y es la clave para lidiar con la complejidad*" (2012, p. 39).

Brennan y Resnick (2012) proponen tres dimensiones claves para establecer el marco de referencia del pensamiento computacional:

- Conceptos computacionales, que son los conceptos que los estudiantes emplean cuando codifican: datos, operadores, secuencias de acciones, sentencias condicionales, bucles, eventos y paralelismo.
- Prácticas computacionales, que son mecanismos de resolución de problemas que ocurren en el proceso de codificación: experimentación e iteración, pruebas y depuración, reutilización y mezcla, y abstracción y modularización.
- Perspectivas computacionales, que son la comprensión de los estudiantes de sí mismos, sus relaciones con los demás, y el mundo digital a su alrededor: expresar, conectar y cuestionar.

Lee (Grover y Pea, 2013), en una línea similar a la de Furber, concibe el pensamiento computacional como una habilidad que se desarrolla utilizando el ordenador como instrumento para representar, estudiar y resolver los problemas del mundo real, consiguiendo de esta manera comprender cada vez mejor lo que un ordenador es capaz de hacer, y cómo darle instrucciones para que lo haga.

Riley y Hunt (Riley y Hunt, 2014) afirman que la mejor aproximación que se puede hacer al término de pensamiento computacional es definirlo como la forma en la que los científicos informáticos piensan y razonan.

García-Peñalvo (García-Peñalvo, 2016) define el pensamiento computacional como la aplicación de alto nivel de abstracción y un enfoque algorítmico para resolver cualquier tipo de problema.

Cada vez son más los países que incluyen dentro de su currículo educativo la informática y el desarrollo del pensamiento computacional (Balanskat y Engelhardt, 2015). Esta inclusión se realiza para estructurar y formular objetivos educativos más avanzados y más útiles. No se trata únicamente de proporcionar a la sociedad un mayor número de profesionales que dominen las TI, de quienes hay una escasez constante en el mercado de trabajo, se trata también de que cualquier persona tenga unas capacidades mínimas, dado que cada vez más profesiones en diferentes disciplinas requieren de una comprensión de la informática. No solo nos tenemos que ceñir al ámbito profesional, la informática también está presente en infinidad de situaciones de la vida cotidiana. En definitiva, la informática reúne las condiciones para convertirse en un sujeto de pleno derecho, con vínculos más profundos con otros temas.

Se está cambiando el foco, de conocer y usar formas específicas de tecnología, a entender los principios básicos de la informática como un campo en sí mismo, que encapsula aspectos de ciencia, tecnología, matemáticas, etc., dentro de él. El desarrollo del pensamiento computacional permite a los estudiantes dominar las habilidades involucradas en la resolución de muchos tipos diferentes de problemas que surgen de la propia naturaleza del procesamiento de la información eficaz, y esa eficacia se relaciona generalmente con que el proceso esté automatizado. Asimismo, automatización en este contexto se asocia a abstracción: no se trata de encontrar la solución a un problema concreto, sino hallar un método que solucione cualquier problema que presente premisas similares.

Los principales conceptos y enfoques del pensamiento computacional, entendiendo este según se ha descrito hasta ahora como una competencia que incluye un conjunto más amplio de herramientas mentales, se ilustra en la siguiente tabla, que es una adaptación de la original publicada en www.barefootcas.org.uk (2014).

Tabla 5
Conceptos y enfoques del pensamiento computacional

PENSAMIENTO COMPUTACIONAL: CONCEPTOS Y ENFOQUES			
CONCEPTOS	LÓGICA Predicción y análisis	REFLEXIONANDO Experimentando y jugando	ENFOQUES
	ALGORITMOS Pasos y reglas	CREANDO Diseñando y haciendo	
	DESCOMPOSICIÓN Desglose en partes	DEPURANDO Encontrado y arreglando errores	
	PATRONES Detección y uso de similitudes	PERSEVERANDO Siguiendo hacia delante	
	ABSTRACCIÓN Eliminación del detalle innecesario	COLABORANDO Trabajando juntos	
	EVALUACIÓN Emitiendo un juicio		

Nota: esta tabla es una adaptación de la ilustración original de www.barefootcas.org.uk (2014)

El grupo de investigación *Google for Education* (2016) aborda el concepto de pensamiento computacional, al que define como un proceso para resolver problemas, que incorpora una serie de características y disposiciones. Así mismo defienden que el pensamiento computacional es imprescindible para la programación de aplicaciones informáticas, pero que las capacidades que implica son aplicables a otros ámbitos como las matemáticas, la ciencia y las humanidades.

Para *Google for Education* (2016), adquirir pensamiento computacional conlleva desarrollar una serie de capacidades:

- Capacidad para expresar un problema de forma que sea trasladable al ordenador, o a otra herramienta, para que pueda ser resuelto.
- Capacidad para recopilar datos y organizarlos de forma lógica.
- Capacidad para abstraer y representar los datos recogidos en forma de modelos o simulaciones.
- Capacidad para desarrollar algoritmos que automaticen el proceso de resolución de un problema.
- Capacidad para analizar las posibles soluciones a un problema, detectar sus debilidades, e idear e implementar nuevas soluciones más efectivas, entendiendo que una solución es más efectiva si consume menos recursos, o si resuelve el problema en menos pasos.
- Capacidad para generalizar la solución de un problema concreto y trasladarla a un contexto más amplio.

4.4. Cómo se aprende y cómo se enseña a programar

La enseñanza y el aprendizaje de la programación parece ser un tema que históricamente ha sido motivo de preocupación a nivel mundial, a tenor del testimonio de varios autores (Bergin y Reilly, 2005a, 2005b; Boyle, Carter y Clark, 2002; Fincher et al., 2006; Gomes y Mendes, 2007a, 2007b; Jenkins, 2002).

En este capítulo se abordan algunas de las técnicas que se utilizan para enseñar y aprender programar, y cómo las teorías de aprendizaje descritas en el capítulo 4.1 se aplican en este contexto.

4.4.1. Técnicas para aprender a programar

Dale y Nyland (1960) realizaron varios estudios sobre las distintas técnicas de aprendizaje que puede utilizar un estudiante para aprender, y cuáles son las más efectivas en términos de lo que se retiene o recuerda con cada una de ellas. Estos autores aunaron el resultado de sus estudios y lo representaron gráficamente en forma de pirámide, en la que se ve reflejada la eficacia de cada método. A esta representación la llamaron Cono de Dale (1960).

A continuación, se muestra una posible interpretación del Cono de Dale (James, 2009).

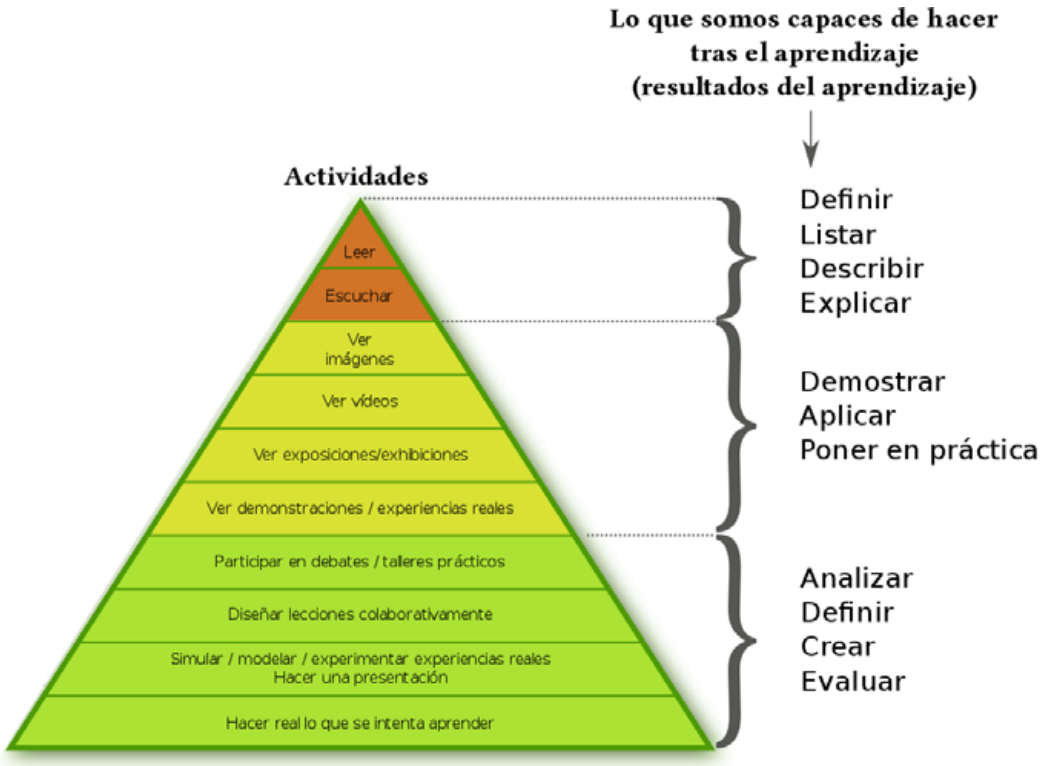


Figura 12. Posible interpretación del Cono de Dale.

Fuente: (Psychoslave, 2016)

Cada estrato de la pirámide refleja una técnica o modo de aprender. En esta adaptación del cono se ha incluido actividades más cercanas a la realidad actual (por ejemplo, ver vídeos) que no estaban en el original. Las descripciones de la columna de la derecha representan los resultados de aprendizaje, es decir, lo que el estudiante va a poder hacer después de realizar esas actividades. Al hacer esta representación, Dale quiso remarcar que las actividades de la base de la pirámide

son más eficaces como instrumento de aprendizaje, descendiendo esta eficacia según la actividad se sitúa más cerca de la punta. Es decir, según Dale se aprende mucho mejor aquello que se experimenta (base de la pirámide), que si simplemente se lee sobre ello (punta de la pirámide).

Trasladando la teoría de Dale al aprendizaje de la programación, queda claro que lo más efectivo para aprender a programar es practicar, aunque previamente se necesitará haber adquirido una serie de conceptos, tal vez mediante docencia, lectura de libros, visionado de tutoriales, estudio de ejemplos, etc.

Enlazando con la teoría de Dale, Alarcón (Alarcón, 2010a) plantea una metodología que se puede aplicar a aprender a programar, reflejada en la *Figura 13* como un algoritmo expresado a través de un diagrama de flujo:

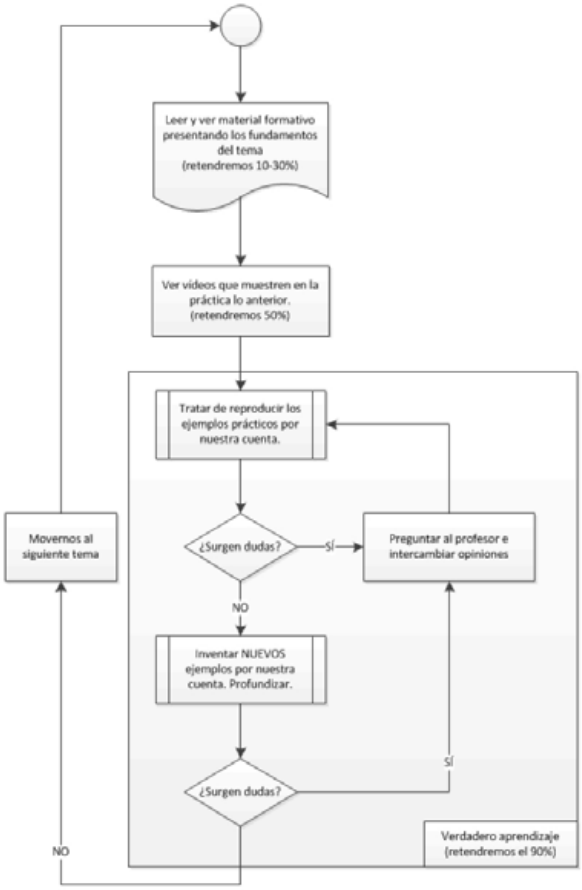


Figura 13. Secuencia de aprendizaje en materias TIC.

Fuente: (Alarcón, 2010b)

La formación que generalmente se puede impartir en una clase de programación cubre la zona superior del cono de Dale. La parte correspondiente a la base de la pirámide, la que Dale plantea como más eficaz, y que tiene que ver con la práctica de la programación, no siempre tiene la presencia que debería en este tipo de aprendizajes por falta de recursos (no se dispone de ordenadores en el aula, la ratio de estudiantes es tan grande que impide que el profesor pueda dar un *feedback* personalizado, etc.). En el capítulo 4.4.4 se abordarán con mayor profundidad las dificultades que un estudiante se puede encontrar en el aprendizaje de la programación.

Las técnicas que están de la base del cono de Dale son las que se encuentran dentro del recuadro grande de la *Figura 13*, y por su propia naturaleza deben ser iterativas: practicar y resolver las dudas, antes de avanzar hacia conceptos más complejos. La responsabilidad de realizar esta iteración recae fundamentalmente sobre el estudiante, siendo el docente un orientador, un apoyo y un instrumento de resolución de dudas. Normalmente la falta de tiempo suele provocar que no se itere suficientemente este proceso. Por eso un curso de iniciación a la programación sirve para introducir al estudiante en el lenguaje, pero no se puede considerar que a su finalización se esté realmente capacitado, y se domine. El dominio vendrá con la práctica y el tiempo dedicado a profundizar en la materia. En el capítulo 4.4.3 se abordarán con más detalle los indicadores que pueden servir para determinar el grado de conocimiento adquirido.

James (2008) habla de la mejor forma de iniciarse en un lenguaje de programación, y lo equipara con cualquier otro proceso de aprendizaje, donde un instructor guía al estudiante con pequeños pasos al principio, y le plantea proyectos de complejidad creciente, hasta tener conocimiento suficiente como para escribir una aplicación completa sin ayuda.

Efectivamente el aprendizaje de la programación no es diferente a otros procesos de aprendizaje, y todas las teorías vistas en el capítulo 4.1 se aplican en mayor o menor medida en la didáctica de esta disciplina.

La enseñanza de la programación habitualmente se centra en la dificultad de las tareas y en su característica motivadora (García-Peñalvo et al., 2016). Esto significa vincular el aprendizaje con la respuesta a un estímulo, siguiendo las teorías conductista tradicionales (Zapata-Ros, 2015). Ciertos principios del modelo conductista también están presentes en la *gamificación*[1] (Koster, 2013) o ludificación de los procesos de aprendizaje propios de multitud de herramientas como Scratch[2]

[1] La palabra gamificación proviene del término inglés Game (juego) y se refiere a trasladar aspectos
[2] https://scratch.mit.edu/

o Alice[3], donde se aprende a programar realizando videojuegos, o *Code Combat*[4], que en sí mismo es un videojuego donde se debe dar instrucciones al personaje a través de la sintaxis del lenguaje de JavaScript, para que se mueva, ataque a los enemigos, etc. Es decir, el aprendiz de programador recibe la recompensa de ver su videojuego funcionando si escribe un código semántica y sintácticamente correcto. La atención a los factores motivacionales se desarrolla con más profundidad en el cognitivismo.

Las teorías cognitivistas también forman parte de la enseñanza de la programación. Todo profesor de este ámbito, además de explicar los procesos para la resolución de problemas concretos, tratará de fomentar la capacidad de análisis en el estudiante, para que identifique los procedimientos que debe utilizar en cada caso, y reflexione sobre por qué los utiliza y cómo funcionan. Por poner un símil de esto, cuando tratamos de colgar un cuadro, buscamos en la caja de herramientas, y de entre todas ellas escogemos un martillo y un clavo, por ser los que más se ajustan a este menester. El clavo se pondrá en la pared, y se golpeará con el martillo. Es decir, de acuerdo al objetivo, hay unas herramientas que son apropiadas, y hay una forma correcta de utilizarlas conjuntamente. Igualmente, para construir un algoritmo, deberemos escoger entre los distintos instrumentos que ofrece la programación (secuencias de acciones, sentencias condicionales, bucles, etc.) según su adecuación al problema a resolver, y se tendrán que combinar de la forma más apropiada.

Siguiendo con el cognitivismo, la Teoría del Procesamiento de la Información (Campos, 2001; Shannon, 2001) se hace patente en el aprendizaje de la programación cuando un estudiante aprende a realizar un algoritmo sencillo (por ejemplo, identificar si un número es divisible entre otro) y lo relaciona en la resolución de algoritmos más complejos (por ejemplo, calcular el máximo común divisor de dos números).

En la enseñanza de la programación también se puede identificar el aprendizaje por repetición (De Bono, 1986), por ejemplo, cuando el profesor muestra el código de ese algoritmo, y el estudiante lo replica en su ordenador, aún sin reflexionar sobre las reglas que sigue dicho algoritmo. En este proceso el estudiante descubre los formalismos sintácticos del lenguaje de programación, y aprende de los errores que ha cometido en el proceso.

Complementariamente a este proceso, se encuentra el aprendizaje significativo del constructivismo (Ausubel et al., 1976), donde el estudiante reflexiona y analiza los

[3] http://www.alice.org/index.php
[4] https://codecombat.com/

conceptos aprendidos, integrándolos en el conjunto de sus conocimientos, y crea una realidad propia. De esta manera dos estudiantes pueden codificar dos soluciones completamente diferentes a un mismo problema, y ser ambas válidas.

La teoría de desarrollo de Piaget (2013) ha estado muy presente en el diseño de aplicaciones para enseñar a programar a niños y niñas en edades tempranas, entendiendo que la capacidad de abstracción, necesaria para elaborar un algoritmo, no se empieza a desarrollar hasta la etapa preoperacional, y que el razonamiento lógico, indispensable en la construcción de un programa informático, no se adquiera hasta la etapa operacional. Por este motivo, la mayoría de herramientas informáticas concebidas con este propósito están recomendadas para estudiantes con 7 años o más, aunque existen entornos aptos para edades a partir de 3 años de menor ambición conceptual.

Se ha visto también en esta tesis cómo las teorías construccionistas de Papert (1980) confluyen en el entorno de programación de Logo, enraizado en el constructivismo de Piaget (2013), y en el enfoque educativo de "aprender haciendo", que parte de la idea de que el estudiante construye activamente el conocimiento conectando sus esquemas cognitivos previos y el mundo real a través de la interacción.

Autores como Bruner (1991) o Wallon (1980) destacan la importancia del factor social en el aprendizaje, y herramientas como Scratch recogen el testigo de sus testimonios, haciéndolos presentes en sus dinámicas de aprendizaje, donde los programas realizados se comparten con el resto de la comunidad, posibilitando reutilizar y mezclar el código de otros con el código propio para construir una nueva solución.

Por último, podemos encontrar la aplicación de las teorías conectivistas de Siemens (2005) y Downes (2005) por ejemplo en la enseñanza de la programación que se centra sobre todo en desarrollar el pensamiento computacional del estudiante, para después profundizar en la sintaxis de un lenguaje concreto. De esta manera, cuando el estudiante afronte posteriormente el aprendizaje de un nuevo lenguaje de programación, le será relativamente sencillo, porque la forma de plantear algoritmos será la misma, y solo cambiará la forma de expresarlos. En definitiva, siendo consciente del ritmo al que evoluciona la tecnología, y de que aparecerán nuevos y más eficientes lenguajes de programación, se prima la capacidad para aprender en el futuro frente a lo que se aprende en el presente.

Si tuviéramos que condensar todo lo expuesto hasta ahora, y resumir en pocas palabras qué características definen al aprendizaje efectivo de la programación, un posible acercamiento podría ser el siguiente:

- El estudiante está motivado (Amabile y Pillemer, 2012; Campos, 2001; Garris, Ahlers y Driskell, 2002)
- El estudiante aprende haciendo, y hace para aprender (Cakir, 2008; Harel y Papert, 1991).
- El estudiante aprende a colaborar, y colabora para aprender (Collazos et al., 2007; Dillenbourg, 1999)
- El estudiante aprende a aprender (Downes, 2005; Siemens, 2005)

4.4.2. Metodologías para enseñar a programar

De las muchas metodologías que se pueden aplicar a la enseñanza y el aprendizaje de la programación, se destacan a continuación tres de ellas, por incluir los aspectos del aprendizaje efectivo que se señalan en la conclusión del capítulo 4.4.1 por su relevancia actual, por los estudios que avalan su eficacia en el aprendizaje (Hamdan, McKnight, McKnight y Arfstrom, 2013; Lye y Koh, 2014; Werner, Campe y Denner, 2012; Williams, 2001; Williams y Kessler, 2002; Yarbro, Arfstrom, McKnight y McKnight, 2014), y por ser combinables y complementarias entre ellas:

- ***Project-based learning (PBL):*** también conocido en castellano Aprendizaje Basado en Proyectos (ABP), es un método de enseñanza en el que los estudiantes adquieren conocimientos y habilidades trabajando durante un período de tiempo para investigar y responder a una pregunta, problema o desafío auténtico, atractivo y complejo (Blumenfeld et al., 1991). En esta metodología se plantea a los estudiantes retos motivadores, como por ejemplo, programar un pequeño videojuego, que se va construyendo poco a poco a base de pequeños ejercicios que forman las partes del gran proyecto.

- ***Pair Programming***: que podríamos traducir como Programación en Parejas, es un método de aprendizaje colaborativo en el cual los estudiantes programan de dos en dos en lugar de individualmente. Para llevarlo a la práctica, lo habitual es que uno en la pareja adopte el rol de “conductor” (*driver*), siendo el que controla el ratón y el teclado y se encarga de escribir el código, mientras que el otro toma el papel de “navegante” (*navigator*), y observa el problema en toda su amplitud, ofrece sugerencias, señala errores, hace preguntas, etc. (Williams, 2001; Williams y Kessler, 2002).

  Existen diversos estudios (McDowell, Werner, Bullock y Fernald, 2002; McDowell, Werner, Bullock y Fernald, 2006; Werner y Denning, 2009) que avalan este enfoque, e indican la significativa mejora de las competencias

adquiridas por parte de los estudiantes que trabajaron en parejas frente a los que trabajaron de forma individual.

- ***Flipped classroom:*** *que se podría traducir como* "aula invertida". Este término, al igual que el de *"flipped learning"* (aprendizaje inverso, o del revés) fue acuñado por Bergmann y Sams (2012) para definir un enfoque pedagógico que combina la enseñanza directa y el aprendizaje constructivista, y que opta por llevar fuera del aula la realización de ciertas tareas y procesos de aprendizaje, para dedicar el tiempo de docencia presencial en facilitar y potenciar otros métodos de adquisición y práctica de conocimientos.

Se denomina aula invertida, porque en lugar de seguir los métodos tradicionales, en los que primero se explica la materia en clase, y después se mandan deberes para consolidar lo aprendido en casa, esta metodología propone hacerlo al revés: que los estudiantes primero accedan desde casa a los contenidos, y posteriormente hagan las tareas en clase. De esta manera el tiempo en el aula queda disponible para que los profesores puedan involucrar a los estudiantes en debates, prácticas de laboratorio, proyectos de resolución de problemas, desafíos, etc. (Tourón y Santiago, 2015, p. 196). La tecnología disponible (campus virtual, archivos en la nube, etc.) y la facilidad para acceder a contenidos on-line ha permitido que las técnicas de *flipped classroom* sean aplicables en prácticamente cualquier contexto de enseñanza. En el caso de la programación, al trasladar la exposición de ciertos contenidos a un sistema en línea, se gana tiempo en el aula para realizar un trabajo práctico guiado, indispensable para aprender a programar.

Según Goodwin y Miller (2013) la eficacia de la metodología *flipped classroom,* por su relativamente reciente aparición, no ha tenido tiempo de ser suficientemente demostrada, aunque ya se han documentado algunas investigaciones donde los resultados son prometedores (Hamdan et al., 2013; Yarbro et al., 2014).

No podemos concluir este capítulo sin hacer una reseña al MOOC (acrónimo en inglés de *Massive Open Online Course)* que podríamos traducir como Curso Online Masivo Abierto. Según Siemens (2012), el término MOOC fue utilizado por primera vez en 2008 por Dave Cormier y Bryan Alexander, y hace referencia a los cursos *on-line* dirigidos a un amplio número de participantes a través de Internet según el principio de educación abierta y masiva que defienden las teorías conectivistas. Algunos de los ejemplos de plataformas de éxito que ofrecen MOOCs son Udemy[5], edX[6], Coursera[7] y FutureLearn[8]. Otro exponente es Khan Academy[9] del que se ha

[5] https://www.udemy.com/
[6] https://www.edx.org/

llegado a decir que está reinventando las reglas de la educación (Noer, 2012). Todas estas plataformas incluyen entre su oferta de MOOCs una enorme cantidad de cursos para aprender a programar.

Los MOOCs y la tecnología Big Data[10], además permiten recopilar información sobre los hábitos de los estudiantes y sobre su forma de aprender, lo que abre las puertas a múltiples estudios de investigación sobre cómo mejorar las herramientas educativas y la forma de enseñar.

4.4.3. Indicadores para evaluar el conocimiento adquirido sobre programación

Por su relevancia con él ámbito de esta tesis, se describen a continuación los indicadores incluidos dentro de la ley educativa de la Comunidad de Madrid, en el Decreto 89/2014 (Decreto Comunidad de Madrid, 2015, p. 90). En este texto se describen los contenidos, criterios de evaluación y estándares de aprendizaje evaluables para toda la etapa, incluidos en la asignatura de libre configuración autonómica "Tecnología y recursos digitales para la mejora del aprendizaje". En el aspecto que a esta tesis le compete, se tienen en cuenta los puntos relativos al contenido descrito como "Fundamentos de programación. Creación de pequeños programas informáticos (Scratch)" (Decreto Comunidad de Madrid, 2015, p. 90).

Para este contenido, el decreto establece dos criterios de evaluación con sus correspondientes estándares de aprendizaje:

1. Conocer los fundamentos de la programación.
   1.1. Utiliza objetos, variables y listas para el desarrollo de sus programas.
   1.2. Interpreta los resultados esperados de pequeños bloques de programas.
   1.3. Evalúa los resultados del programa.
   1.4. Depura un programa para que el funcionamiento se adecue al previsto.

2. Programar juegos sencillos, animaciones e historias interactivas.
   2.1. Selecciona los elementos gráficos y los sonidos que formarán su programa.

[7] https://www.coursera.org/
[8] https://www.futurelearn.com/
[9] https://es.khanacademy.org/
[10] *Big Data* es un término que se aplica a toda aquella información que por su volumen no puede ser procesada por herramientas tradicionales, y a los métodos usados para encontrar patrones dentro de esa información (Barranco, 2012).

2.2. Determina las acciones individuales que necesita el funcionamiento del programa.
2.3. Determina el orden y el sentido de los movimientos (arriba, abajo, derecha,
2.4. izquierda) y los giros para conseguir el resultado deseado
2.5. Determina las interacciones entre los diferentes elementos de su programa.

Como complemento a este capítulo, y para poder contextualizar mejor los estándares de aprendizaje que incluye la ley de la Comunidad de Madrid, en los anexos se incluyen otros dos referentes internacionales, los definidos en el Reino Unido y en Estados Unidos, dos países pioneros con una larga tradición en la promoción de la enseñanza de los lenguajes de programación desde las primeras etapas escolares. Del Reino Unido, se presentan las competencias definidas en el *Computing Progression Pathways* (Dorling y Walker, 2015), y de Estados Unidos se muestran las elaboradas por la CSTA (*Computer Science Teachers Association*), en referencia a los estándares de computación (Seehorn y Pirmann, 2016).

4.4.4. Factores que influyen en el aprendizaje de la programación

Caspersen y Bennedsen (2007, p. 111) afirman que aprender a programar es algo considerado “notoriamente difícil”, y que a pesar de que hay una trayectoria de más de cuarenta años en la historia de la humanidad enseñando y aprendiendo a programar, la enseñanza de la programación sigue siendo un desafío importante. Mead *et al.* (2006, p. 183) comentan: "en los últimos veinticinco años, los estudios nacionales e internacionales han proporcionado indicadores empíricos que demuestran que el aprendizaje para programar es realmente un reto para la mayoría de los estudiantes".

Gomes y Mendes (2007b) recopilan las dificultades con las que se encuentran sus estudiantes de programación en este aprendizaje. Tomando como base este estudio, el de otros autores (Byrne y Lyons, 2001; Davies, 1993; Gomes, Carmo, Bigotte y Mendes, 2006), y la propia experiencia del autor de esta tesis de más de diez años como profesor de programación con estudiantes de distintas edades, a continuación se describen las principales características de la problemática asociada a enseñar y aprender a programar:

1. **Los métodos de enseñanza no son siempre los más adecuados** para las necesidades de muchos estudiantes, por diferentes motivos:

2. **La enseñanza no es personalizada.** No todos los estudiantes avanzan al mismo ritmo, y sin embargo la clase que se imparte es la misma para todas y todos. No siempre es posible ofrecer la atención personalizada y la supervisión que sería deseable. La retroalimentación inmediata durante la resolución de problemas y la explicación detallada de los aspectos de más difícil comprensión probablemente podría paliar este problema, pero en la mayoría de las ocasiones las limitaciones de recursos y tiempo no lo permiten.

3. **Las estrategias docentes de los profesores no siempre van a la par de los estilos de aprendizaje de todos los estudiantes**. Diferentes estudiantes tienen diferentes formas de aprender. Hay estudiantes que requieren de mucha asistencia, o de partir de ejemplos previos en lugar de un “lienzo en blanco” para comenzar un nuevo programa. Otros, sin embargo, necesitan tiempo para llegar a conclusiones por ellos mismos, y precisan de más autonomía. El profesor debe entender la naturaleza de sus estudiantes y elegir las estrategias más apropiadas según el caso (Jenkins, 2002).

4. **La programación implica varios conceptos dinámicos que muchas veces se enseñan a través de medios estáticos** (presentaciones proyectadas, explicaciones verbales, diagramas, dibujos de pizarra, textos, etc.). Muchos estudiantes no entienden la dinámica del programa explicada a través de este tipo de materiales.

5. **Los profesores están más concentrados en enseñar un lenguaje de programación y sus detalles sintácticos, en vez de promover la resolución de problemas usando un lenguaje de programación**. Esto también se aplica a los libros de texto que introducen al aprendizaje de la programación. En general, este tipo de libros dedican la mayor parte de su contenido a presentar conocimientos sobre un lenguaje en particular y se centran explícitamente en los aspectos declarativos y de sintaxis del lenguaje, pero no tratan con suficiente profundidad la forma en que el conocimiento se utiliza o se aplica. Como han señalado varios autores y, en particular, Davies (1993), el conocimiento del lenguaje es solo una parte de todo lo que supone aprender a programar. El lenguaje es una herramienta para expresar ideas y algoritmos. Sin embargo, es habitual que en la enseñanza de la programación se enseñe a los estudiantes gran cantidad de detalles sintácticos del lenguaje, antes de que estos hayan comprendido ciertos conceptos importantes de programación.

6. **Los métodos de estudio de los estudiantes no son siempre los más adecuados para aprender a programar.** Los estudiantes usan metodologías de estudio incorrectas. Muchos estudiantes, para resolver problemas de otras

disciplinas, memorizan fórmulas o procedimientos. Pero en ocasiones memorizan algo que no terminan de comprender, o no dimensionan el tipo de problemas donde lo pueden aplicar. Como se ha mencionado en el capítulo del pensamiento computacional, programar no tiene que ver con memorizar. Su proceso de aprendizaje debe ser esencialmente práctico y muy intensivo (a programar se aprende programando).

7. **Las habilidades, conocimientos y aptitudes del estudiante no siempre son las más adecuadas. A continuación se describen algunas carencias detectadas:**
   - Los estudiantes no saben cómo resolver problemas. La falta de habilidades genéricas que tienen los muchos estudiantes para resolver problemas les sitúa en un punto de partida muy complicado para aprender a programar. Es decir, los estudiantes no saben cómo crear algoritmos, principalmente porque no saben cómo resolver problemas. Como se ha descrito en el capítulo 2.2.2, la resolución de problemas precisa de habilidades múltiples que los estudiantes a menudo no tienen:
     - Comprensión del problema - Muchas veces los estudiantes tratan de resolver un problema sin entenderlo por completo. A veces esto sucede porque el estudiante tiene dificultades para interpretar la declaración del problema, y otros simplemente porque están demasiado ansiosos por empezar a escribir código y no leen e interpretan correctamente la descripción del problema.
     - Relación del conocimiento - Muchos estudiantes no establecen analogías correctas con problemas pasados, y no transfieren conocimientos previos a los nuevos problemas. Tienden a agrupar los problemas que tienen las mismas características superficiales en lugar del mismo principio. En consecuencia, muchas veces los estudiantes basan sus soluciones en problemas no relacionados, lo que conduce a soluciones incorrectas.
     - Reflexión sobre el problema y la solución - Los estudiantes tienden a escribir una respuesta antes de pensar cuidadosamente sobre ella. Muchas veces las pruebas se realizan superficialmente y se satisfacen solo porque el programa funciona con un conjunto de datos, sin realizar pruebas más extensas.
     - Falta de persistencia - Los estudiantes suelen renunciar a resolver un problema si no encuentran rápidamente una posible solución, aunque esta diste de ser la óptima o la más adecuada. Por lo general, resolver problemas de programación exige esfuerzo y persistencia. Sin embargo, cuando se enfrentan a cualquier dificultad, muchos estudiantes prefieren pedir la solución a un colega o simplemente renunciar, en lugar de seguir

tratando de resolver el problema. Esto es especialmente importante, ya que el aprendizaje es más eficaz cuando los estudiantes llegan por sí mismos a la solución, en lugar de que simplemente la lean o alguien se la cuente (James, 2009).

8. **Muchos estudiantes no tienen suficientes conocimientos de matemáticas y de lógica.** (Gomes et al., 2006) realizaron algunos experimentos explorando las relaciones entre las competencias matemáticas de resolución de problemas y la falta de habilidades de programación mostradas por un grupo de estudiantes que no fueron aprobados en su curso de programación inicial. Esta experiencia se llevó a cabo durante el segundo semestre de 2005/2006 y los autores concluyeron que los estudiantes involucrados tenían profundas dificultades en varias áreas, como el cálculo básico y la teoría de números o simples conceptos geométricos y trigonométricos. Los autores también reportan dificultades para transformar un problema textual en una fórmula matemática que lo resuelva. También se identificaron las limitaciones en el nivel de abstracción y el razonamiento lógico. El conocimiento matemático es muy importante para la programación y es posible encontrar estudios (Byrne y Lyons, 2001), por ejemplo que evidencian la relación entre las habilidades de programación y la experiencia en matemáticas.

9. **Los estudiantes no asimilan conceptos básicos, no saben cómo funcionan las estructuras de programación comunes, o tienen una concepción errónea sobre ellos.** También es habitual que los estudiantes muestren dificultades para detectar errores simples de programación, tanto lógicos como sintácticos.

10. **La actitud del estudiante no siempre es las más adecuadas. Los estudiantes no trabajan lo suficiente para adquirir las competencias de programación necesarias.** Muchos estudiantes, si no ven resultados con relativa rapidez, se frustran y pierden fuelle en el transcurso del estudio. En la programación, como en otras ciencias, los conocimientos avanzados requieren de la comprensión de los conocimientos básicos, si se flaquea al principio, el aprendizaje cada vez se pondrá más cuesta arriba. Este problema no se circunscribe solo al ámbito de la programación.

11. **Los estudiantes no tienen la motivación suficiente como para enfrentarse a lo que supone este tipo de aprendizaje.** Y si no tienen una motivación intrínseca difícilmente tendrán éxito (Ng y Bereiter, 1991). Asimismo, el período en el que se afronta este aprendizaje coincide en ocasiones con la adolescencia o la transición a la madurez, etapas vitales que acarrean cambios importantes. El aprendizaje de la programación es complicado de por sí, y si además se produce

en un período de cierta inestabilidad, esto solo puede contribuir a un aumento de la dificultad (Jenkins, 2002).

12. **La naturaleza intrínseca de la programación también supone un problema para los estudiantes:**
    - La programación exige un alto nivel de abstracción, y es lo que quizás más cuesta adquirir a los estudiantes. El aprendiz de programación tiene que tener capacidades de generalización, transferencia y pensamiento crítico, entre otros. Por norma general, en las primeras fases del aprendizaje es cuando empiezan a surgir las dificultades, en el momento en el que se espera que los estudiantes entiendan y apliquen ciertos conceptos abstractos de programación, como por ejemplo las sentencias condicionales, para resolver problemas.

    - La sintaxis de la mayoría de los lenguajes profesionales de programación no facilita su aprendizaje, por su complejidad. Son extensos y tienen muchos detalles sintácticos difíciles de memorizar. Esa complejidad requiere que los estudiantes tengan que concentrarse simultáneamente en la construcción del algoritmo y en las reglas sintácticas. Este aspecto es de crucial relevancia para lo que compete a esta tesis, pues la usabilidad del lenguaje de programación se revela como un factor clave para el éxito o el fracaso en su aprendizaje.

Davies (1993) distingue entre conocimientos de programación (por ejemplo, en un lenguaje de paradigma imperativo, ser capaz de indicar cómo funciona un bucle) y las estrategias de programación (la forma en que se usa y aplica el conocimiento, por ejemplo, usar la sentencia "*for*" adecuadamente en un programa). Aprender a programar, asimismo, está relacionado con otros tipos de conocimientos sobre ordenadores, sobre un lenguaje o lenguajes de programación, sobre herramientas y recursos de programación, y sobre teorías y métodos formales.

Analizando estos estudios, se puede concluir que para afrontar el aprendizaje de la programación existen una serie de necesidades a cubrir, que se pueden clasificar como requisitos técnicos o materiales, y requisitos personales.

a) Requisitos técnicos y materiales:

Para abordar los conceptos básicos de la programación, bastaría con una hoja de papel y un lápiz, aunque según se profundice en el aprendizaje, se necesitará de un ordenador con un entorno de programación, es decir, con el software necesario para poder escribir el código de programación y que este sea interpretado y ejecutado por la máquina. Este entorno dependerá del lenguaje en el que decidamos

programar. A esta lista se podría añadir una conexión a Internet como fuente de recursos bibliográficos, y lugar de consulta y resolución de dudas.

b) Requisitos personales:

Según lo desarrollado hasta ahora sobre lo que supone afrontar el aprendizaje de la programación, se puede concluir que existen una serie de capacidades básicas que el estudiante debe tener, o en su defecto, adquirir, para poder programar. Una propuesta de estas características se expone a continuación:

- **Pensamiento computacional:** Cuando se trata de buscar una solución a un problema, el programador debe enfrentarse a la situación como un ordenador lo haría, sin presuponer que hay un conocimiento previo, y precisando hasta el más mínimo de los detalles.

- **Capacidad de abstracción:** En el contexto de la informática la abstracción consiste en aislar un elemento de su contexto o del resto de elementos que lo acompaña, de este modo en programación el término se refiere al énfasis en el ¿qué hace? más que en el ¿cómo lo hace? Programar es básicamente encontrar la forma de solucionar un problema. Pero la solución planteada debe ser global y válida para todas las variantes posibles (por ejemplo, si se trata de sumar números, el programa propuesto deberá ser capaz de sumar cualesquiera que estos sean). El programador debe tener la capacidad de abstraerse y de contemplar los problemas desde todas sus perspectivas. Un mismo problema puede presentar varias soluciones, cuantos más puntos de vista se contemplen, mayor será la posibilidad de encontrar la mejor resolución.

- **Pensamiento algorítmico:** A la hora de pensar una solución a implementar mediante programación, se debe plantear de forma estructurada (determinar la secuencia lógica de acciones para llegar de la situación inicial de la que partimos, a la situación final). Ese orden y esa estructuración ayudará a no cometer errores ni olvidos que afecten a la funcionalidad del programa final.

- **Conocimientos matemáticos:** Si bien no es absolutamente necesario, si es bueno tener nociones matemáticas sobre operaciones aritméticas y ciertos conocimientos de álgebra booleana. Por este motivo en las carreras universitarias la asignatura de Programación suele ir acompañada de otras como Matemática Discreta, Álgebra, Cálculo, etc.

- **Motivación:** Gomes, Santos y Mendes (2012) en su investigación encuentran una fuerte correlación entre los estudiantes que tienen una buena percepción

de sí mismos sobre sus capacidades para aprender a programar, y los que habitualmente rinden muy bien académicamente en las disciplinas de programación. Un resultado similar también se da en los estudiantes con un alto grado de motivación. En definitiva, para aprender a programar es fundamental partir de una posición personal en la que se va a estar seguro del éxito, y tener el grado de motivación suficiente para afrontar las dificultades que puedan surgir en el aprendizaje. Un último apunte desde la perspectiva de la creatividad: tal y como nos recuerda Amabile y Pillemer (2012, p. 7) *"*el estado del sujeto intrínsecamente motivado es conducente a la creatividad, mientras que el extrínsecamente motivado va en detrimento*"*.

- **Otros conocimientos de base:** Para empezar a programar no se necesitan conocimientos de base más allá de las capacidades antes mencionadas, que permitan identificar un problema y plantear una solución de forma organizada. Más adelante, una vez se dominen las cuestiones básicas de la programación, quizás sea necesario aprender sobre software, archivos, sistemas operativos y otras cuestiones relacionadas con la informática. Pero para aprender a programar, esto no es lo primero.

¿Cualquier persona puede aprender a programar?

El autor de esta tesis no ha encontrado estudios relevantes que ayuden a responder a esta pregunta. Si extrapolamos de una forma muy general las teorías innatistas a este contexto (Chomsky, 2014) podríamos considerar que existe una predisposición natural que hace que resulte más sencillo o más complicado aprender a programar, de la misma manera que se puede tener un talento artístico que nos permita adentrarnos en el mundo del dibujo con más o menos éxito. En cualquier caso, con la suficiente dedicación aprender a programar es potencialmente asequible para cualquier persona con unas mínimas aptitudes. Así lo defiende por ejemplo la Hora del Código (Code.org, 2015), o autores como Resnick, Maloney, Monroy-Hernández, Rusk, Eastmond, Brennan, Millner, Rosenbaum, Silver y Silverman (Resnick et al., 2009).

En este mismo apartado se ha visto que se precisan una serie de capacidades básicas para afrontar el aprendizaje de un lenguaje de programación (pensamiento computacional, una base matemática, etc.), aunque no se dice en qué medida se necesitan. El autor de esta tesis no ha encontrado herramientas estandarizadas y con amplia aceptación, que permitan cuantificar exactamente los niveles de conocimiento necesarios en estos aspectos. Esto plantea una posible línea de investigación, en la que se desarrollen instrumentos que permitan medir los requisitos mínimos para aprender a programar.

4.5. Herramientas que se pueden utilizar en el proceso de enseñanza-aprendizaje de la programación

En este capítulo se muestra una recopilación de posibles recursos que se pueden utilizar para la enseñanza y el aprendizaje de la programación. Cuando hablamos de recursos nos referimos a entornos de programación que, por su diseño y sus contenidos, pueden ser una buena opción para adentrarse en el mundo de la programación.

El propósito de esta recopilación es, por un lado, ofrecer una visión del estado del arte de este tipo de recursos, para poder entender el contexto el que se sitúa Scratch. Por otro lado, pretende servir de fuente para posibles vías de investigación futuras que pretendan evaluar alternativas a Scratch. En los anexos se realiza un análisis detallado de cada recurso, mientras que en este capítulo solo se muestra un resumen sus principales características.

La variedad de recursos de este tipo es inmensa, y esta recopilación no pretende ser absolutista, ni hacer una descripción exhaustiva. Además, es susceptible de someterse a constante revisión.

Para escoger las herramientas a incluir en este análisis se han tomado como referentes estudios previos en la materia (Adam y Mowers, 2013; Alonso Urbano y Hueso Rivas, 2014; Cronin, 2014; Kharbach, 2014), así como índices de popularidad de las herramientas de programación (TIOBE, 2017).

De cada recurso se proporciona la siguiente información:

- **Herramienta:** nombre comercial de la herramienta
- **Página web:** sitio web de referencia, donde se puede encontrar información de la herramienta, y los enlaces de descarga o compra.
- **Versión en 2017 / Año:** a fecha de 2017, versión que existe disponible y año en el que fue publicada dicha versión.
- **Plataformas:** tipos de dispositivos y plataformas para las que la herramienta está disponible
- **Tipo de software / Modelo de negocio:** el tipo de software distingue si es software libre (su código fuente está disponible para copiarlo, cambiarlo, reutilizarlo, distribuirlo, etc.) o privativo (el código fuente no está disponible, y la capacidad para usar el programa, modificarlo o distribuirlo está restringida). El modelo de negocio se refiere a si la herramienta es gratuita o de pago.

- **Edad Recomendada:** edades para las que el fabricante / desarrollador recomienda su uso**.**
- **Característica diferenciadora:** elementos que confieren a la herramienta un valor diferencial.

A continuación se muestra la tabla resumen con las herramientas categorizadas según lo descrito anteriormente.

Tabla 6
Posibles herramientas que se pueden utilizar en la enseñanza y aprendizaje de la programación.

Herramienta	**Página web**	**Versión en 2017 / Año**	**Plataformas**	**Tipo de software /Modelo de negocio**	**Edad Recomen-dada**	**Característica diferenciadora**
Adventure Maker	http://www.adventuremaker.com/	adenture_maker_v4_7_1, del año 2008	Windows	Privativo y gratuito, pero tiene versiones de pago (sin uso comercial (69$) y de uso comercial (139$)	No especificada	Ofrece un posible entorno profesional para desarrollar videojuegos
AgentSheets	http://www.agentsheets.com/	AgentSheets 4.0, del año 2016	JVM (Java Virtual Machine), Mac 10.8+, Windows 7+	Privativo, 48 - 67€	5+	Ofrece herramientas para la creación tanto de juegos, como de simulaciones científicas. Contaba con el apoyo de la NASA.
Alice	www.alice.org/index.php	Alice 3.3, 22 de agosto del año 2016	Windows (Vista o superior), Mac OS (10.6 o superior), Linux (Ubuntu, Red Hat)	Libre y gratuito	8+	Entornos 3D muy atractivos. Posibilidad de utilizar los lenguajes Java, C++ y C#
Baltie	http://www.sgpsys.com/en/	SGP Baltie 3 y SPG Baltie 4 C# Pro, versiones del año 2017	Windows	Privativo y de pago (12$ - 189$)	6 - 13	Herramienta que requiere de pocos conocimientos de base para poder utilizarse
Blockly	https://developers.google.com/blockly/	Web, última actualización el 23 de junio del año 2016	Chrome, Firefox, Safari, Opera, IE Android, iOS	Privativo y gratuito	4 - 8	Se puede exportar el programa por bloques a otros lenguajes, como JavaScript. Crear bloques de código personalizados

Herramienta	Página web	Versión en 2017 / Año	Plataformas	Tipo de software /Modelo de negocio	Edad Recomendada	Característica diferenciadora
Caleiduino	http://www.caleiduino.com/	Año 2016	Plataforma propia, pero los drivers necesarios pueden descargarse para Windows y Mac	Libre y gratuito	No especificada	Totalmente configurable. Libre distribución
Cargo-Bot	https://twolivesleft.com/CargoBot/	Cargo-Bot 1.0.1, el día 8 de mayo del año 2012	iOS 5.0 o superior: iPhone, iPad	Privativo y gratuito.	4+	Es el primer juego del App Store desarrollado enteramente usando Codea, una app para la creación rápida de juegos y simulaciones
CodeCombat	https://codecombat.com/	Web, del año 2017	Web	Privativo y gratuito	No especificada	Su función multijugador. Posibilidad de aprender del código directamente. Sistema de aprendizaje por niveles de conocimiento
Code-a-Pilar	http://www.fisher-price.com/en_US/brands/think-and-learn/index.html	Code-a-pillar 1.2.1 para Android, del 22 de diciembre del año 2016 Code-a-pillar 1.2.2 para iOS, del 23 de diciembre del año 2016	iOS	Privativo y de pago (50$ - 70$)	3 - 6	Herramienta que funciona tanto como juguete físico como en forma de aplicación.
Codeable Crafts	https://www.codeablecrafts.com/	App Store: 26 de febrero de 2017 Play Store: 8 de julio de 2015	Android e iOS	Privativo y gratuito	6 - 8	Además de ser una herramienta para programar, incluye una gran variedad de recursos para potenciar el talento artístico y la creatividad.

Herramienta	Página web	Versión en 2017 / Año	Plataformas	Tipo de software /Modelo de negocio	Edad Recomendada	Característica diferenciadora
Codin Game	https://www.codingame.com/start	Web, año 2017	Web	Privativo y gratuito	16+	Permite la utilización de más de 20 lenguajes de programación diferentes
Cody&Roby	http://codeweek.it/cody-roby-en/	_	Web	Privativo y gratuito	No especificada	Se trata de un juego de mesa, no requiere de elementos externos
Construct2	https://www.scirra.com/construct2	25 de octubre del año 2016	Windows	Privativo. Versión gratuita con capacidades limitadas, y versión de pago (329$)	12+	Permite crear videojuegos mediante eventos. Entorno totalmente visual, basado en arrastrar y soltar.
Daisy the Dinosaur	http://daisythedinosaur.com/	2 de noviembre del año 2016	iOS 8.0 o superior	Privativo y gratuito	6 – 8	Adecuado para que niñas y niños de edades muy tempranas se inicien en la programación
Desktop Dungeons	http://www.desktopdungeons.net/	Desktop Dungeons de enero del año 2016	Windows, MAC OS, Ubuntu, Android e iOS	Privativo y de pago (10€ - 23€)	12+	Videojuego de partidas rápidas, con una estética que homenajea a los videojuegos clásicos. Uso del lenguaje C#
E-Slate	http://e-slate.cti.gr/	Versión del año 2000	Windows	Libre y gratuito	6+	Requiere un dispositivo con muy pocos recursos para poder funcionar. Posibilita realizar simulaciones
Guido van Robot	https://sourceforge.net/projects/gvr/files/	Versión de GvR del año 2007	Windows XP o superior, Mac OS 10.3 o superior	Privativo y gratuito	12+	Herramienta pensada para ser utilizada en un entorno educativo, con la asistencia de un docente
Hackety Hack	http://www.hacketty.com/	No especificada	Windows, MAC OS, Linux	Privativo y gratuito	13+	Lecciones de corta duración incluidas dentro de la herramienta

Herramienta	Página web	Versión en 2017 / Año	Plataformas	Tipo de software /Modelo de negocio	Edad Recomen-dada	Característica diferenciadora
Hopscotch: Coding for Kids	https://www.gethopscotch.com/	Hopscotch: Coding for Kids 3.15.0, del 7 de febrero del año 2017	iPhone e iPad	Privativo. Dos versiones, una gratuita (temporal por 7 días) y una por suscripción (cada mes, cada 3 meses, 6 meses, o 1 año), entre 6€ - 75€	4 - 14	Posibilidad de crear aplicaciones y juegos propios, y compartirlos con el resto de usuarios desde una plataforma móvil, sin necesidad de tocar código directamente
Human Resources Machine	https://tomorrowcorporation.com/humanresourcemachine	Android: https://play.google.com/store/apps/details?id=com.tomorrowcorporation.humanresourcemachine&hl=es	Windows, MAC OS X, Wii U, Android	Privativo y de pago (5,49€ - 9,99€)	7+	Es un videojuego. Gamificación del proceso de aprendizaje: enseña a programar mientras se juega a él.
Infinifactory	http://www.zachtronics.com/infinifactory/	24 de julio del año 2015	Windows, MAC OS X, Linux y PlayStation 4	Privativo y de pago (23,99€)	12+	Ofrece la posibilidad de aprender a pensar y organizar diferentes formas de resolver un problema mientras se juega y se es partícipe de una historia
Karel	http://www.olimpiadadeinformatica.org.mx/omi/omi/material/Karel_el_Robot.aspx	Karel.exe, del 23 de agosto del año 2012	Windows, MAC OS y Linux	Privativo y gratuito	11 - 16	Ofrece acceso al código en varios lenguajes de programación
KidsRuby	http://kidsruby.com/	Versión 1.4, del año 2014	Windows, MAC OS X, Raspberry Pi y Debian Linux	Privativo y gratuito	12+	Ofrece mucha libertad creativa. Orientada a una enseñanza tutorada
Kodable	https://www.kodable.com/	7 de febrero del año 2017	Web. También cuenta con aplicación para iOS	Privativo y de pago (a partir de 29$)	6 - 8	Cuenta con muchos ejercicios actualizados regularmente y posee diferentes planes de enseñanza

Herramienta	Página web	Versión en 2017 / Año	Plataformas	Tipo de software /Modelo de negocio	Edad Recomen-dada	Característica diferenciadora
Kodu	https://www.kodugamelab.com/	Kodu.exe, Versión 1.4.164.0 del día 12 de junio del año 2016	Windows XP, Vista, 7 y 8	Privativo y gratuito	6 - 11	Posibilita crear escenarios en 3D, crear personajes, y añadir Inteligencia Artificial a los elementos creados.
Laby	http://laby.thr.pm/	Versión de Laby 1.2.4 del 28 de noviembre de 2014	iOS y Android	Privativo y gratuito	3+	Es un videojuego. Se aprende mientras se juega.
Learn to Program BASIC	http://www.basic256.org/index_en	BASIC256Portable_1.99.99.67.paf.exe, del 9 de septiembre del año 2016	Windows	Libre y gratuito	15+	Traducido a varios idiomas, un lenguaje sencillo de aprender
Lego Mindstorm	https://www.lego.com/en-us/mindstorms/?domainredir=mindstorms.lego.com	Android: versión 1.0.71, del 25 de enero del año 2017 iOS: versión del 9 de diciembre del año 2016	Windows (Vista o superior), MAC (10.6 o superior), Linux, Android (4.2 o superior) e iOS (8 o superior)	Privativo y de pago (400$)	4 - 14	Vincula la programación a la robótica. Combina software, hardware y piezas de LEGO.
Light-bot	https://lightbot.com/	Lightbot versión 1.6.5 (para Android), 23 de enero del año 2016	iOS, Android, Windows, MAC y Kindle	Privativo y de pago (2´99$ - 4,99$)	4+	Tiene versiones diferentes en relación a la edad de los usuarios. Posibilidad de uso en diferentes plataformas
Logo	http://neoparaiso.com/logo/winlogo.html	Versión del año 2015	Windows, Web	Privativo y gratuito	15+	Sencillo de utilizar. Traducido a varios idiomas.
Made with Code	https://www.madewithcode.com/	Versión del año 2017Web	Web	Privativo y gratuito	8+	Apoyado por Google. Accesible desde cualquier dispositivo a través de la web.
Mama	http://www.eytam.com/mama	Mama 2.0, versión del año 2015	Windows (XP o superior)	Privativo y gratuito	8+	Programación arrastrando y soltando. Acceso al código generado.

Herramienta	Página web	Versión en 2017 / Año	Plataformas	Tipo de software /Modelo de negocio	Edad Recomendada	Característica diferenciadora
Minecraft	https://minecraft.net/es-es/	Versión descargable de la página del año 2017	Windows, Mac, Linux (compatibles con Java), Windows Phone, Android, iOS, Xbox 360, Xbox One, PlayStation3, PlayStation4, PlayStation Vita, Raspberry Pi, Wii U	Privativo y de pago (23,95€)	7+	Es un videojuego. Posibilidad de crear mundos únicos y mecanismos complejos a partir de materiales obtenidos en el propio juego.
MIT App Inventor	http://appinventor.mit.edu/	Versión de la web del año 2017	Chrome, Firefox, Safari, Opera, IE, Android e iOS	Libre y gratuito	12+	Puede utilizarse en casi cualquier navegador web de forma gratuita
Project Spark	https://www.microsoft.com/es-es/store/p/project-spark/9wzdncrfhw95#	Versión actualizada de Project Spark antes del cierre de sus servidores el 12 de agosto del año 2016	Xbox One y Windows 10	Privativo y gratuito	7+	Gran variedad de contenidos y recursos propios
Python Turtle	http://pythonturtle.org/	Python Turtle versión 0.1.2009.8.2.1, del año 2015	Windows, MAC OS	Privativo y gratuito	12+	Sencillez del lenguaje
RoboMind	http://www.robomind.net/es/	Windows: RoboMind 6.0.1 MAC OS: RoboMind 5.3 Linux: 6.0.1	Windows (XP o superior), MAC OS (X o superior) y Linux	Privativo y de pago (7 – 14$ al año)	6 – 18	Se adapta tanto a usuarios novatos como a expertos.
RPG Maker	http://www.rpgmakerweb.com/	RPG Maker 2017	Windows y MAC	Privativo, existen versiones gratuitas (30 días) y de pago (80$)	No especificada	Permite crear videojuegos simplemente arrastrando y soltando. Permite acceder al código generado, y modificarlo.

Herramienta	**Página web**	**Versión en 2017 / Año**	**Plataformas**	**Tipo de software /Modelo de negocio**	**Edad Recomen-dada**	**Característica diferenciadora**
Scratch	https://scratch.mit.edu/	Versión web del año 2017	Web, pero también cuenta con una versión offline: Windows, MAC, Linux, Android	Privativo y gratuito	8 - 16	Posibilidad de crear y compartir proyectos propios con otros usuarios, utilizando la web del fabricante.
Snap	http://snap.berkeley.edu/	Versión de la web de Snap del año 2017	Web, aunque tiene una versión *offline*	Privativo y gratuito	8 – 16	Ampliación del lenguaje de Scratch.
Stagecast Creator	http://acypher.com/creator/	Stagecast Creator versión 2.1, del año 2016	Windows	Privativo, no está a la venta	8+	Utiliza un lenguaje de programación natural. Sencillo de utilizar para hacer pequeñas animaciones y juegos
Star Wars: Building a Galaxy with Code	https://code.org/starwars	Versión de la web del año 2015	Web, también cuenta con una versión offline (Windows y MAC OS X)	Privativo y gratuito	8 - 16	La herramienta está traducida a varios idiomas. Posibilidad de usar código o bloques con programación predefinida
Stencyl	http://www.stencyl.com/	Febrero del año 2017	MAC, Windows y Linux	Privativo y gratuito	21+	Posibilidad de crear juegos en 2D de una calidad aceptable para ordenador, web y dispositivos móviles
The Code Academy	https://www.codecademy.com/	Versión actualizada de la web en el año 2017	Web	Privativa, gratuita, aunque tiene una versión de pago que da acceso a contenido adicional (20$)	No especificada	Posibilidad de utilizar diferentes lenguajes de programación. Combinable con otras herramientas, para las que ofrece tutoriales

Herramienta	Página web	Versión en 2017 / Año	Plataformas	Tipo de software /Modelo de negocio	Edad Recomen-dada	Característica diferenciadora
Tickle	https://tickleapp.com/	Versión 4.2.1 de Tickle, del 31 de diciembre del año 2016	iOS	Privativo y de pago. La herramienta es gratuita y puede usarse sin necesidad de utilizar los robots, pero queda muy limitada	6 – 10	Posibilidad de ver los resultados de un programa físicamente través de robots
Tynker	https://www.tynker.com/	23 de febrero del año 2017	iOS, Android y a través de la web	Privativo y de pago. Hay varios packs (4 meses, 1 año o de por vida), el precio depende del pack de enseñanzas (8$ - 16$), además de un servidor de Minecraft	8 - 14	Herramienta para crear videojuegos. Ofrece diferentes perfiles de uso para adaptarse a las necesidades de los usuarios
Unity 5	https://store.unity.com/es	Versión de Unity 5.5.2, del 24 de febrero de 2017	Windows, MAC OS X y Linux	Privativo. Existe una versión gratuita con opciones limitadas, y varias versiones de pago (35$ - 125$ por puesto al mes), orientadas a empresas y profesionales del sector	18+	Motor para realizar videojuegos de gran potencia. Utilizada en el ámbito profesional.
Unreal Engine 4	https://www.unrealengine.com/what-is-unreal-engine-4	Unreal Engine 4.15, del 15 de febrero del año 2017	Windows y MAC	Privativo y gratuito, pero a partir de los 3000$ recaudados por ventas de los proyectos desarrollados con la herramienta, la empresa se queda con un 5% de cada venta	18+	Motor para realizar videojuegos de gran potencia. Utilizada en el ámbito profesional. Acceso gratuito a todas las prestaciones de la herramienta.

Herramienta	Página web	Versión en 2017 / Año	Plataformas	Tipo de software /Modelo de negocio	Edad Recomen-dada	Característica diferenciadora
W	https://www.vttoth.com/CMS/projects/49-w-a-simple-programming-language	Versión de W del año 2016	Windows	Privativo y gratuito	No especificado	Herramienta de fácil aprendizaje, requiere de muy poca potencia del dispositivo en el que vaya a utilizarse.
Wimi5	http://wimi5.com/es/	Versión online, actualizadas para el año 2017	Web, pero se está trabajando en poder utilizarlos desde plataformas móviles como Android e iOS	Privativo y gratuito, pero un 30% de las ganancias totales por las ventas de los proyectos creados en la plataforma se los lleva la empresa autora	No especificada	Herramienta con una gran comunidad, de fácil acceso y que da la posibilidad de desarrollar proyectos tanto a empresas como desarrolladores independientes
Zero Engine	http://zero.digipen.edu/	Versión del año 2016	Windows	Privativo. De uso exclusivo para los estudiantes de Digipen. Podríamos decir que su precio es la matrícula de Digipen.	18+	Exclusividad de la herramienta, está enfocado a las necesidades de los estudiantes de DigiPen

Fuente: elaboración propia

4.6. Iniciativas en el ámbito no reglado que promueven el aprendizaje de la programación

Para terminar de comprender la relevancia del tema que se investiga en esta tesis, y completar el marco social en la que se encuadra, a continuación se describen algunas iniciativas que se han llevado a cabo para aprender a programar en el ámbito no reglado, tanto a nivel mundial, como nacional. En este apartado se dan algunas pinceladas de estas experiencias, y se proporcionan sus respectivas páginas web para una posible ampliación de la información.

4.6.1. Iniciativas mundiales

En EEUU se lanza en el 2013 ***Code.org***[11], un programa sin ánimo de lucro dedicado a expandir el acceso a la programación, y a abogar por su inclusión en la educación formal. Fue fundada por los hermanos Hadi y Ali Partovi. Esta organización se encarga de impartir lecciones gratuitas de programación, a través de su sitio web. Asimismo, Code.org son los promotores de **La Hora del Código**[12], un movimiento que llega a decenas de millones de estudiantes en más de 180 países, y que imparte lecciones de programación sobre distintos lenguajes, y con distintos niveles de dificultad, de una hora de duración.

Google también ha sido promotor de una iniciativa destacada, ***Cs First*** [13]. Su plataforma está orientada a padres, profesores y estudiantes que deseen aprender sobre programación. Además, aboga por romper la brecha de género con el propósito de incentivar la participación de niñas y mujeres en este ámbito, con la plataforma de ***Made witch Code*** [14].

A un nivel más modesto encontramos varias organizaciones sin ánimo de lucro que tienen el objetivo de llevar la alfabetización digital de una manera divertida y atractiva a todo tipo de público. Algunas de ellas son:

Khan Academy [15] es una organización educativa sin fines lucrativos, creada en el 2006 con el objetivo de proporcionar una educación gratuita de calidad para todo el mundo. Entre sus contenidos relacionados con la programación podemos encontrar la Computación: programas de computadoras, Ciencias de la Computación, la Hora del Código y la Animación Digital.

[11] https://code.org/
[12] https://hourofcode.com/es
[13] https://www.cs-first.com/en/home
[14] https://www.madewithcode.com/
[15] https://es.khanacademy.org/

Code Club [16] y ***Coder Dojo*** [17]son espacios que fomentan el asociacionismo de personas interesadas en la programación. A día de hoy cuenta con más de 100 países colaboradores.

Partiendo de este movimiento global sobre la enseñanza de la programación nace en Europa ***EU Code Week*** [18] y ***All You Need Is Code*** [19], que promueven el desarrollo del pensamiento computacional para todos los niveles educativos.

Para compartir experiencias relacionadas con el aprendizaje de la programación, entre diferentes países europeos, existen varios espacios de encuentro, entre los que cabe destacar ***Future Classroom Lab*** [20] en Bruselas, o ***Computational Thinking & Coding @ Schools*** [21] en Viena.

4.6.2. Iniciativas en España

España ha seguido la estela de este movimiento mundial, y ha sido pionera en promover el aprendizaje de los lenguajes de programación entre niños y niñas en edades muy tempranas. Tenemos un ejemplo en el proyecto **KPL (*Kid's Programming Language*) - Diviértete programando** [22], una experiencia que se llevó a cabo en León como actividad extraescolar, y se enmarcó dentro del Plan Nacional de Investigación Científica, Desarrollo e Innovación Tecnológica 2008-2011 con el objetivo de iniciar a niños y adolescentes en la disciplina de la programación, favoreciendo simultáneamente el aprendizaje transversal de otras materias escolares como matemáticas, lengua, inglés, etc.

En Google España nos encontramos con **Genios** [23], una iniciativa que forma parte del Programa de Apoyo a la Infancia, de la ONG Ayuda en Acción, y que tiene como fin promover una integración social y tecnológica, enseñando nociones básicas de programación y código a estudiantes de Primaria de más de treinta centros escolares.

Otra iniciativa destacada es **Programamos** [24], una asociación sin ánimo de lucro formada por un equipo de educadores e investigadores que pretende promover el aprendizaje de los lenguajes de programación en todas las etapas educativas. Sus

[16] https://www.codeclubworld.org/
[17] https://coderdojo.com/
[18] http://codeweek.eu/
[19] http://www.allyouneediscode.eu/
[20] http://fcl.eun.org/
[21] https://www.ocg.at/de/ct-c-at-schools
[22] http://www.aletic.es/archivos/tic/1246352824.pdf
[23] http://www.genios.org/
[24] https://programamos.es/,

aportaciones en el ámbito científico son realmente notables, por ejemplo son los creadores de **Dr. Scratch** [25], una herramienta que evalúa la calidad del código de programas hechos con Scratch, y organizadores de multitud de actividades que tienen como objetivo promover el desarrollo del pensamiento computacional desde pequeños a mayores.

También se pueden encontrar entidades privadas interesadas en la enseñanza de la programación. Podemos destacar **Diwo** [26]**,** una iniciativa de la empresa BQ, fabricante de *smartphones*, tabletas, libros electrónicos e impresoras 3D, que tiene el objetivo de promover la inclusión de la programación en la educación. En su web podemos encontrar recursos sobre robótica, programación, Scratch, incluso contenidos sobre la asignatura de Educación Secundaria: Tecnología, Programación y Robótica. Otra iniciativa en el ámbito privado es ***Girls in Lab*** [27], que tiene como fin inspirar, educar e involucrar a las niñas en la tecnología y la programación.

En España son muchos los proyectos que se están llevando a cabo a nivel regional o local sobre la enseñanza de la programación. A continuación se exponen tres ejemplos en Madrid, Cataluña, y Aragón.

En Madrid existen experiencias muy interesantes en espacios educativos como Cosmocaixa, y el Museo Nacional de Ciencias y Tecnología. Un ejemplo es **Educaixa** [28], dentro del espacio de Cosmocaixa, en el que se realiza la actividad de "Programa tu universo", en colaboración con la Fundación Everis y Udigital, con el fin de impulsar la programación entre estudiantes, familias y docentes. Otro ejemplo, esta vez en el Museo Nacional de Ciencia y Tecnología en Alcobendas, donde Google y la Fundación Española para la Ciencia y la Tecnología (**FECYT**) [29] ofrecen talleres y programas de formación con el objetivo de que los estudiantes practiquen el uso creativo de las tecnologías, y aprendan nociones básicas de programación, fomentando el espíritu innovador y emprendedor. Por último, en esta comunidad cabe destacar la plataforma ***eMadrid*** [30], donde se presentan investigaciones en el campo científico tecnológico.

Otra de las comunidades que merece la pena referenciar es Cataluña. En el Parque Tecnológico de la Universidad de Gerona encontramos a un grupo interdisciplinario de investigadores que forman la plataforma ***UdiGitalEdu*** [31], dedicada al diseño y desarrollo de experiencias tecnológicas. Otro ejemplo originado en esta comunidad

[25] http://www.drscratch.org/
[26] http://diwo.bq.com/
[27] http://girlsinlab.com/
[28] http://agenda.obrasocial.lacaixa.es/-/programa-tu-universo-con-scratch
[29] http://www.muncyt.es
[30] http://www.emadridnet.org/
[31] http://udigital.udg.edu/

es ***Inventors4Change*** [32], una plataforma de experiencias colaborativas entre niños y niñas, de diferentes países y con otras realidades, conectados a través del arte y la tecnología. Otro espacio destacado lo encontramos en Barcelona, que desde 2012 lleva a cabo ***Mschools*** [33], una iniciativa educativa impulsada por ***Mobile World Capital Barcelona*** [34] que abarca múltiples facetas, y que incluye un programa donde se enseña programación y ayuda a estudiantes y a docentes a integrar las tecnologías digitales en el aula de una manera eficaz.

En Aragón cabe destacar el proyecto ***Etopia_kids*** [35] de la Fundación Zaragoza Ciudad del Conocimiento, en colaboración con la Obra Social de Ibercaja y el Ayuntamiento de Zaragoza, que lanzan en 2013 este programa para desarrollar actividades con tecnologías creativas de código abierto dirigido a niños y niñas de edades comprendidas entre los 6 y los 14 años.

[32] http://www.inventors4change.org/
[33] http://mschools.mobileworldcapital.com/
[34] http://mobileworldcapital.com/es/
[35] https://www.etopiakids.es/

5. La usabilidad como factor en el aprendizaje

En capítulo 4.4.4 se describen los factores que influyen en el aprendizaje de la programación. Entre estos factores se ha mencionado por un lado la dificultad intrínseca asociada la programación (sintaxis complicadas, entornos de desarrollo poco asequibles, etc.), y por otro la falta de adecuación de los medios de enseñanza que se utilizan.

La usabilidad es un término que hace referencia a la facilidad de uso de una aplicación o producto interactivo (Baeza Yates y Rivera Loaiza, 2002). Al unir esta definición con lo expuesto en el párrafo anterior, podemos decir que si el medio utilizado para enseñar a programar es usable, el aprendizaje se facilitará, o al menos se eliminará uno de los factores que lo obstaculizan. Igualmente, si se consigue paliar la dificultad intrínseca que acarrea la programación, se estará allanando el camino para su aprendizaje y su uso.

Como se verá en el capítulo 5.3, los estudios científicos de usabilidad aplicados a los lenguajes y a las interfaces de programación tienen una larga tradición, y se realizan, entre otros motivos, con el propósito de romper las barreras que se encuentra un principiante al aproximarse a la programación.

El capítulo 5 hace un recorrido por el concepto de usabilidad, comenzando con una definición operativa del mismo, que mostrará su carácter empírico. Después se explicarán los mecanismos que se pueden utilizar para medir la usabilidad. Por último, en el mencionado capítulo 5.3, se presentará un estado del arte de los estudios de usabilidad aplicados a la programación.

5.1. Definición operativa de usabilidad

Hassan-Montero y Ortega-Santamaría (2009, p. 9) resaltan la dimensión empírica del concepto de usabilidad, por ser una característica que puede ser medida y evaluada. Así estos autores, citando a Nielsen (2003), califican la usabilidad como un atributo de calidad definido formalmente a través de distintos componentes o variables que permiten esta medición y evaluación. Nielsen (2003), autor de referencia en este tema, enumera estos componentes de calidad:

- **Facilidad de Aprendizaje** (*Learnability*): ¿Cómo de fácil resulta para los usuarios llevar a cabo tareas básicas la primera vez que se enfrentan al diseño?
- **Eficiencia** (*Efficiency*): Cuando los usuarios han aprendido el funcionamiento del diseño, ¿cuánto tardan en realizar las tareas?

- **Cualidad de ser recordado** (*Memorability*): Cuando los usuarios vuelven a usar el diseño después de un tiempo sin hacerlo, ¿cuánto tardan en volver a usarlo eficazmente?
- **Errores** (*Errors*): ¿Cuántos errores comete el usuario al realizar una tarea?, ¿cómo de graves son las consecuencias de esos errores?, ¿cómo de rápido puede el usuario corregir esos errores?
- **Satisfacción** (*Satisfaction*): ¿Cómo de agradable es para el usuario el uso del diseño?

Nielsen (2003) también destaca el papel de la usabilidad a la hora de evaluar la utilidad de algo, y proporciona las siguientes definiciones:

- Algo tiene utilidad (*utility*) si ofrece las características que necesitan de ello.
- Algo es usable (*usability*) si es fácil y agradable utilizar las características que ofrece
- Algo será completamente útil (*useful*) si aúna ambas características: usabilidad (*usability*) + utilidad (*utility*).

El concepto de usabilidad está teniendo una relevancia capital en el diseño de software, enfocándose éste en crear aplicaciones y productos que le sean completamente útiles al usuario, según la definición que acabamos de ver. Las herramientas digitales concebidas para la enseñanza y el aprendizaje no son una excepción. Los diseñadores de este tipo de aplicaciones se encuentran con el reto de desarrollar un software que tenga utilidad para estudiante. Para ello, además de las directrices de usabilidad y de diseño centrado en el usuario (Costabile, 2001), se precisan métodos de diseño centrados en el estudiante (Quintana, Carra, Krajcik y Soloway, 2001) para que los entornos de aprendizaje sean accesibles de una forma educativamente productiva.

Los sistemas eficaces deben incluir funciones tan avanzadas como sea necesario, pero su interfaz debe ocultar su complejidad, proporcionando una fácil interacción para captar el interés de los estudiantes (Ardito et al., 2004). Una interfaz mal diseñada se convierte en una barrera para el aprendizaje efectivo (Kruse, 2003). Por tanto, la usabilidad de un sistema se presenta a priori como un factor clave para facilitar el aprendizaje o dificultarlo, en función de si es o no adecuada.

5.2. Mecanismos para evaluar la usabilidad de una aplicación o producto

En este punto merece la pena recalcar la dimensión empírica del concepto de usabilidad, lo que significa que puede ser medida y evaluada a través de diferentes componentes o variables (Hassan-Montero y Ortega-Santamaría, 2009). A la hora

de establecer un análisis de usabilidad, una posibilidad es utilizar dos tipos de análisis complementarios entre sí: los análisis heurísticos y los análisis de usuarios (Hassan Montero y Martín Fernández, 2004):

- La evaluación heurística es un método de inspección en el que expertos en la materia evalúan de forma independiente el producto, fundamentándose en reconocidos principios de usabilidad, como las normativas internacionales (Hassan Montero y Martín Fernández, 2004, p. 339; Nielsen, 1994).
- La evaluación con usuarios es un método que combina diferentes técnicas de evaluación de usabilidad, y que consiste en llevar a cabo una serie de pruebas en un entorno controlado, en las que se observa cómo un grupo de usuarios, reales o potenciales, llevan a cabo las tareas que se les encomiendan, para posteriormente realizar un análisis de los problemas con los que se han encontrado durante la prueba (Hassan Montero y Martín Fernández, 2004; Henry, Law y Barnicle, 2001).

Esta perspectiva tradicional de análisis de la usabilidad se fundamenta en el encaje entre las capacidades funcionales del individuo, y las posibilidades que ofrece el diseño de los productos y/o el entorno. La forma de evaluar este encaje entre el diseño de “lo técnico” y las capacidades de “lo funcional” es perfectamente posible con los análisis heurísticos y de usuarios utilizados habitualmente. Por este motivo estos tipos de análisis son los escogidos para medir la usabilidad de Scratch.

Entre otras, a continuación se describen algunas normas que pueden ser referentes a la hora de hacer un análisis heurístico:

- *Nielsen Norman Group Guidelines*. Se trata de los principios de evaluación heurística más populares a lo largo del mundo. A pesar de poder definir una gran lista de directrices en profundidad, pueden agruparse en estos principios básicos (Nielsen, 1994):
    - Que el sistema siempre informe al usuario sobre lo que está ocurriendo.
    - El sistema debe hablar el lenguaje del usuario
    - Control total del usuario, incluso en los momentos en que se equivoca
    - Diseño estándar de las diferentes posibilidades
    - Prevención de errores mediante la confirmación de actos importantes
    - Eficiencia de uso para el usuario experto
    - Diseño minimalista sin información irrelevante para el usuario
    - Sistema de ayuda adaptado al usuario y su lenguaje
    - Incluir ayuda y documentación

- *Australia Government Usability Checklist (AGIMO, 2008).* Se trata de un documento elaborado por el Gobierno de Australia que define 43 preguntas que podemos hacernos sobre un software o una página web para evaluar su usabilidad y accesibilidad. Es importante hacer notar que en estas directrices la accesibilidad se considera un elemento de primera importancia. Aún no hemos abordado este aspecto en nuestra investigación, aunque planificamos hacerlo en siguientes fases.

  Las 43 preguntas se agrupan en 6 áreas:

  - Arquitectura y navegación. Recoge 13 preguntas sobre la forma de navegación del usuario por la arquitectura del sistema.
  - Layout y diseño. 7 preguntas que evalúan el diseño de la interfaz.
  - Contenidos. 4 preguntas acerca del contenido, evaluando especialmente el lenguaje.
  - Formularios. Dos preguntas sobre formulario. Es una directriz dirigida especialmente al análisis de páginas web que precisan la entrada de información por parte del usuario.
  - Plataforma e implementación. Se trata de 9 preguntas de carácter más técnico sobre la resolución, correcto funcionamiento de los elementos, etc.
  - Accesibilidad. 8 preguntas que evalúan la accesibilidad del sistema para personas con diversidades funcionales o necesidades especiales.

- *MIT Usability Guidelines* (MIT y IST, 2011). El *Massachusetts Institute of Technology* proporciona a los evaluadores una herramienta de 10 líneas de análisis heurístico que recogen 62 directrices:

  - Navegación. 5 directrices que evalúan la forma de navegar del usuario.
  - Funcionalidad. 4 directrices para evaluar cómo se adapta el sistema a los diferentes tipos de usuario.
  - Control. 5 directrices para evaluar la capacidad de control del sistema por parte del usuario.
  - Lenguaje y contenido. 7 directrices que se centran en la información que ofrece el sistema.
  - Ayuda y guías. 2 directrices para evaluar el acceso a la ayuda si es necesario.
  - Accesibilidad. 14 directrices basadas en las recomendaciones de la W3C (W3C, 2008)
  - Consistencia. 2 directrices que evalúan la coherencia del sistema en sus diferentes niveles y profundidades.

- Error. 8 directrices para la prevención de errores y correcciones si son necesarias.
- Arquitectura y claridad visual. 8 directrices para evaluar la interfaz del sistema y su diseño.

- *HHS Guidelines.* Estas directrices fueron diseñadas por el Departamento de Salud y Servicios Sociales del Gobierno de Estados Unidos (HHS). Son 18 líneas de evaluación heurística que agrupan 209 directrices (Shneiderman y Leavitt, 2006). A su vez, cada directriz está evaluada con respecto a dos parámetros:
  - Importancia relativa. 16 expertos externos evaluaron cada directriz en una escala de 1 a 5 con respecto a la pregunta "¿Cómo es de importante esta directriz para el éxito de una web?
  - Solidez de los datos. 8 evaluadores externos expertos en revisión por pares evaluaron en una escala de 1 a 5 la solidez de los datos que podía ofrecer cada directriz (HHS, 2004).

  Las 18 líneas de evaluación que agrupan las directrices son:
  - Proceso de diseño y evaluación. 11 directrices sobre el proceso de diseño previo al desarrollo, como los objetivos.
  - Optimizar la UX. 16 directrices sobre la experiencia del usuario.
  - Accesibilidad. 13 directrices sobre accesibilidad fundamentadas en la legislación estadounidense, conocida como Section 508 (U.S.Congress, 1998).
  - Hardware y Software. 5 directrices sobre aspectos técnicos como buscadores, resolución, etc.
  - Home page. 9 directrices para evaluar la página de inicio de la web o el sistema.
  - Layout. 13 directrices para evaluar el esquema de diseño.
  - Navegación. 12 directrices que evalúan mapas, enlaces, menús, etc.
  - Paginado. 5 directrices para evaluar el uso adecuado del scrolling.
  - Encabezados, títulos y etiquetas. 8 directrices que evalúan estos elementos.
  - Enlaces. 14 directrices muy centradas en diseño web, más que en sistema de software, que evalúan el diseño de enlaces para la navegación.
  - Texto. 11 directrices para evaluar el texto centrándose en su apariencia: contraste, tamaño, etc.

  - Listas. 9 directrices de nuevo muy centradas en el diseño web sobre el uso de listas para ordenar la información y el acceso a la misma.
  - Controles. 25 directrices sobre los aspectos técnicos de botones, widgets, etc. y su respuesta ante la interacción del usuario.
  - Gráficos, imágenes y multimedia. 16 directrices para evaluar los elementos gráficos.
  - Contenido web. 11 directrices que evalúan el texto, en este caso desde el punto de vista del contenido y uso del lenguaje.
  - Organización del contenido. 9 directrices sobre la agrupación de elementos y las categorías utilizadas desde el punto de vista de la comprensión del usuario.
  - Búsqueda. 9 directrices sobre las posibilidades de búsqueda de información por parte del usuario sin tener que recorrer la totalidad del sistema.
  - Test de usabilidad. 13 recomendaciones acerca de cómo evaluar todos los puntos anteriores con usuarios.

- *Nokia Usability Guidelines for Games.* Las directrices para el desarrollo de software interactivo de Nokia (2003) se centran sobre todo en los aspectos de usabilidad que deben tener los videojuegos. Estas se agrupan en 10 líneas*:*

  - Pantalla de inicio y menú principal. 4 directrices para un apropiado acceso al software.
  - Controles. 4 directrices sobre el control del software de forma natural o esperada por el usuario.
  - Pausa y guardar partida. 3 directrices sobre cómo pausar y guardar en el caso de un jugador, dos jugadores o multijugador.
  - Retroalimentación. 5 directrices para evaluar la información ofrecida al usuario.
  - Desafío. 6 directrices sobre retos y objetivos en el juego.
  - Ruido. 4 directrices para evitar el exceso de información o la información redundante.
  - Gráficos. 4 directrices para facilitar la comprensión de los elementos gráficos.
  - Ayuda. 4 directrices sobre InGame Help.
  - Puntuación. 5 directrices para diseñar un high score adecuado que motive al usuario y no le desmoralice.
  - Reinicio. 2 directrices para reiniciar las partidas en los juegos individuales o multijugador.

5.3. Estado del arte de los estudios de usabilidad aplicados a la programación

Una definición sobre programación expuesta desde la perspectiva del ser humano dice que "la programación es el proceso de transformar un plan mental en uno que sea compatible con la computadora" (Hoc y Nguyen-Xuan, 1990). El lenguaje de programación es la forma en la que se expresa esta transformación, y cuanto menor sea ésta, más fácil será la tarea de programar (Green, 1989).

Los ámbitos de la programación y la interacción hombre-ordenador, comúnmente conocida como HCI (*Human Computer Interface*) están ligados intrínsecamente y comparten áreas comunes, como por ejemplo el análisis de los requisitos que debe cumplir un programa, y la evaluación del programa desarrollado. De la misma manera, los desarrolladores de software son también usuarios de lenguajes de programación, de entornos de desarrollo y de herramientas de gestión. En definitiva, los principios y métodos de HCI son aplicables a lenguajes y herramientas de programación de la misma manera que lo son a cualquier otro software.

Los primeros ESPs (*Empirical Studies of Programmers*) o estudios empíricos sobre programadores, a los que también se les ha llamado estudios de psicología de la programación, se remontan a varias décadas atrás (Weinberg, 1971). Desde entonces podemos encontrar a lo largo de la historia varios ejemplos de estudios HCI que han centrado su foco en la actividad de la programación (Cherubini et al., 2007; Myers et al., 2007; Norcio, 1982; Patel et al., 2008; Rosson y Carroll, 1996; Soloway et al., 1982), y aunque se han realizado avances significativos en este terreno, sigue siendo un tema que preocupa a la comunidad científica, porque aún no se ha conseguido que programar deje de ser considerado como una tarea difícil (Caspersen y Bennedsen, 2007).

Por el ámbito que compete a esta tesis, este apartado va a realizar un breve recorrido por el estado del arte de los estudios HCI de referencia que investigan la usabilidad en los lenguajes de programación, y en la interfaz de los entornos que se utilizan para programar.

Si hablamos de estos últimos, podemos referirnos a un entorno de programación utilizando el término API (*Application Programming Interface*) o Interfaz de programación de aplicaciones. El Instituto de Ingeniería de Software de la universidad de *Carnegie Mellon University* define API como "tecnología que facilita el intercambio de mensajes o datos entre dos o más aplicaciones de software diferentes" (Carnegie Mellon Software Engineering Institute, 2008). Además, clasifican las APIs en las siguientes categorías:

- APIs de llamadas de procedimiento remoto (RPC – *Remote Procedure Calls*)
- APIs de lenguaje de consulta estándar (SQL – *Standard Query Language*)
- APIs para transferencia de archivos y entrega de mensajes

Esta definición puede ser útil para algunas categorizaciones y taxonomías académicas, pero no refleja el uso común del término. Para este propósito podemos tomar como referencia la definición que De Souza, Redmiles, Cheng, Millen y Patterson (2004) proporcionan sobre el concepto de API: "cualquier interfaz bien definida que defina el servicio que un componente, módulo o aplicación proporciona a otros elementos de software". Según esta definición, API sería la interfaz para un modelo de objeto que ha codificado un miembro de un equipo de desarrollo, una librería de programación, o un kit de desarrollo de software (SDK – *Software Development Kit*). Evidentemente, hay diferencias significativas entre estos tipos de APIs, pero en esencia su propósito y el uso que se les da es el mismo: todas ellas proporcionan una interfaz de usuario que sirve para desarrollar aplicaciones.

Las APIs también han sido objeto de interés en los ESPs (Rosson y Carroll, 1996), pero es en estos últimos años donde la usabilidad en las APIs ha llamado más la atención. En parte, esto se debe al aumento en número y en capacidades de estas APIs. Como muestra de esta afirmación tenemos el crecimiento exponencial que ha experimentado en los últimos años la red de desarrollo de Microsoft, comercialmente denominada MSDN (Microsoft, 2017a), tanto en volumen de herramientas como en la potencia y versatilidad que ofrecen. A la par de este crecimiento también se ha producido un auge del uso de APIs en todos los ámbitos (empresarial, universitario, etc.). Es poco común en los tiempos actuales encontrar contextos en los que se desarrolle sin utilizar una API, y ciertos ámbitos, como el desarrollo de interfaces de usuario, la programación gráfica, el desarrollo de servicios web, etc., se puede decir que son prácticamente imposibles de abordar sin el uso una API.

Uno de los primeros estudios formales sobre usabilidad en las APIs fue el de McLellan, Roesler, Tempest y Spinuzzi (1998). Asimismo, son referencia en este ámbito los trabajos de Bloch (2005) sobre la API de Java[36], y el de Cwalina y Abrams (2008) sobre el entorno .NET[37]. Bloch señala el diseño de las API como un factor clave en el éxito o el fracaso de las empresas (Bloch, 2005).

[36] Java es un lenguaje de programación y una plataforma informática comercializada por Sun Microsystems desde 1995.
[37] .NET es un entorno de desarrollo multiplataforma de código abierto, comercializado por Microsoft

Siguiendo esta misma línea de investigación podemos encontrar los trabajos de Pugh (2006) sobre el diseño de interfaces, y de Tulach (2008) que recoge el testigo de Bloch (2005), y desde su experiencia como programador de Java aborda el diseño práctico de APIs. Cada uno de estos textos describe el diseño de las APIs como un arte donde la usabilidad juega un papel primordial.

A mediados de la década de 2000, Steven Clarke también comienza a realizar estudios de usabilidad en las API (Clarke, 2004; Clarke y Becker, 2003), y realiza una labor encomiable presentando informes sobre las APIs de uso generalizado, como la de Microsoft. Su trabajo también es digno de mención porque ha servido de inspiración directa a un número significativo de trabajos de los investigadores de Carnegie Mellon, dirigidos por Brad Myers y Jeff Stylos (Beaton, Jeong, Xie, Stylos y Myers, 2008; Ellis, Stylos y Myers, 2007; Stylos y Clarke, 2007; Stylos, Clarke y Myers, 2006; Stylos, Faulring, Yang y Myers, 2009; Stylos y Myers, 2007). Estos estudios han podido cuantificar el impacto que tiene el diseño de las APIs, en el usuario, y cómo un mal diseño puede tener como resultado que se tarde entre 3 y 10 veces más en hacer una tarea (Ellis et al., 2007; Stylos y Clarke, 2007).

Mientras que Clarke fue pionero en el análisis de usabilidad de la API de Microsoft, el equipo de investigación de De Souza es considerado como uno de los primeros en ofrecer un análisis formal de las API desde una perspectiva organizacional (de Souza et al., 2004).

Si nos centramos específicamente en la usabilidad del lenguaje de programación, podemos encontrar relativamente pocos estudios que proporcionen una orientación clara a los diseñadores a la hora de crear mecanismos que faciliten la tarea a los programadores. Kaijanaho (2015) ha realizado recientemente una prospección sobre este tema, y encontró 141 estudios, realizados entre 1950 y 2012, muchos de los cuales eran de utilidad marginal. Sólo 65 estudios se podían considerar relevantes, de los cuáles 22 fueron experimentos controlados aleatorios (Kaijanaho, 2015, p. 185).

En estos estudios sobre lenguajes de programación y usabilidad se han utilizado métodos HCI convencionales, y también se han creado ex profeso otros completamente nuevos.

Como ejemplo de métodos convencionales se puede considerar el uso de ensayos controlados aleatorios (Kaijanaho, 2015). En su investigación, Kaijanaho (2015) examina la usabilidad en el uso de las distintas variantes de declaraciones de sentencias condicionales.

Otro ejemplo del uso de métodos HCI convencionales en un estudio lo vemos presente en el llevado a cabo por Altadmri y Brown (2015), que realizan un seguimiento longitudinal del comportamiento del programador para proporcionar una visión de la usabilidad del lenguaje. En su estudio Altadmri y Brown (2015) parten de los datos recopilados durante un año en el proyecto Blackbox (Brown, Kölling, McCall y Utting, 2014) sobre el comportamiento de más de 250.000 estudiantes de programación de todo el mundo, y analizan la proliferación de los errores que se cometen al programar, la frecuencia con la que estos se producen, y el tiempo que se tarda en arreglarlos, mostrando cómo estos factores se relacionan entre sí.

Como ejemplos de métodos específicamente diseñados para evaluar un lenguaje de programación podemos incluir el “marco de las dimensiones cognitivas” (Green, 1989) que trata de proporcionar un vocabulario estandarizado que se pueda incorporar en el diseño de lenguajes y entornos de programación, para que éstos puedan ser considerados como usables. Este método se ha utilizado para evaluar tanto lenguajes de programación textuales como visuales. Steven Clarke es uno de los investigadores destacados que ha hecho uso del marco de las dimensiones cognitivas en sus estudios (Clarke, 2004; Clarke y Becker, 2003).

Otro ejemplo de método específico es el de "programación natural" (Myers, Pane y Ko, 2004), que trata de entender cómo las personas conciben ciertos conceptos, y cómo los expresan. Para comprender mejor la dimensión de la programación natural en este contexto, se puede citar la investigación de Pane y Mayers (2000), en la que estudian cómo personas con escasa o ninguna experiencia en programación entienden y verbalizan expresiones lógicas que incluyen sentencias AND y OR (por ejemplo, en una lista de figuras selecciona las que sean triángulos y además sean rojos), y cómo debería ser un lenguaje de programación para que construir este tipo de expresiones sea una tarea asequible.

En general, de una índole o de otra, los estudios que se realizan en este terreno pretenden crear mecanismos que permitan medir la usabilidad de los lenguajes y entornos de programación estudiando cuánto facilitan el aprendizaje, cuál es su efectividad, qué productividad que se consigue con su uso, y cuál es la propensión a cometer errores al utilizarlos. La ACM (*The Association for Computing Machinery*), principal referente en este campo de investigación, periódicamente organiza el congreso *ACM CHI Conference on Human Factors in Computing Systems* (ACM, 2017), donde se exponen los principales avances en el ámbito de la usabilidad y la programación. El resultado de sus contribuciones se ha utilizado como fuente para la elaboración de este capítulo (Daughtry, Farooq, Stylos y Myers, 2009; Myers et al., 2016).

6. Scratch

Scratch es la herramienta que se somete a estudio en esta tesis, porque es el programa elegido para la enseñanza de la programación en la Comunidad de Madrid, como se ha visto en el 2.3. Este capítulo tiene como objetivo presentar sus características más destacadas, así como los principios de su diseño. El capítulo concluye haciendo una breve reseña a la aplicación de Scratch en el contexto educativo, tanto en el ámbito reglado como en el no formal.

6.1. Introducción a la herramienta Scratch

Scratch es un proyecto del MIT (*Massachusetts Institute of Technology*), concretamente del grupo de investigación *Lifelong Kindergarten* del laboratorio de medios del MIT en colaboración con el grupo de Yasmin Kafai en UCLA. Está diseñado para enseñar a programar, mediante la creación de videojuegos, historias animadas, y presentaciones interactivas, a estudiantes en la horquilla de edades entre los 8 y los 16 años. Scratch es posiblemente la herramienta de aprendizaje de programación más extendida en el mundo, utilizándose en más de 150 países (MIT, 2013).

Scratch sigue la estela de otros entornos de programación y lenguajes que le han precedido, dirigidos a programadores principiantes (Guzdial, 2004; Kelleher y Pausch, 2005).

El entorno de Scratch permite crear fácilmente animaciones y elementos interactivos. Esto se consigue mediante una interfaz gráfica con un diseño HCI (*Human Computer Interaction*) clásico mediante pantalla, teclado y ratón. Este diseño facilita que cualquier persona que sepa manejar un Sistema Operativo moderno (Windows, Mac OS, GNU-Linux Ubuntu, etc.) sea capaz de comprender la lógica del software y empezar a utilizarlo.

Scratch tiene una interfaz gráfica que permite a los usuarios crear y manipular imágenes en 2D, añadir música y sonidos, crear animaciones, y dotar a cualquier objeto de interactividad. Un proyecto Scratch consiste en una escena fija (fondo) y una serie de objetos movibles. Cada objeto contiene su propio conjunto de imágenes, sonidos, variables y secuencias de comandos. Esta organización permite exportar el proyecto fácilmente, y también importar de forma sencilla elementos de otros proyectos.

La pantalla de Scratch se divide en cuatro áreas, como se puede ver en la *Figura 14*. En la parte izquierda de la herramienta está el escenario. El cuadro azul de la

esquina superior derecha del escenario es un botón que permite que la escena se muestre a pantalla completa. Debajo de la escena hay un área que muestra las miniaturas de todos los *sprites*[38] del proyecto. Al hacer clic en una de estas miniaturas se selecciona el *sprite* correspondiente. El panel central tiene tres pestañas: programas, disfraces y sonidos. Seleccionando cada una de esas pestañas cambiará el panel de la derecha, donde el usuario podrá ver y modificar los disfraces (imágenes), los sonidos, o el comportamiento (el programa) del *sprite* elegido. Seleccionando la pestaña de "Programas" en el panel central, aparecerán los bloques de comandos que permiten crear el programa. Los bloques de comandos se dividen en diez categorías: Movimiento, Apariencia, Sonido, Lápiz, Datos, Eventos, Control, Sensores, Operadores, y Más Bloques. Cada categoría está asociada a un color.

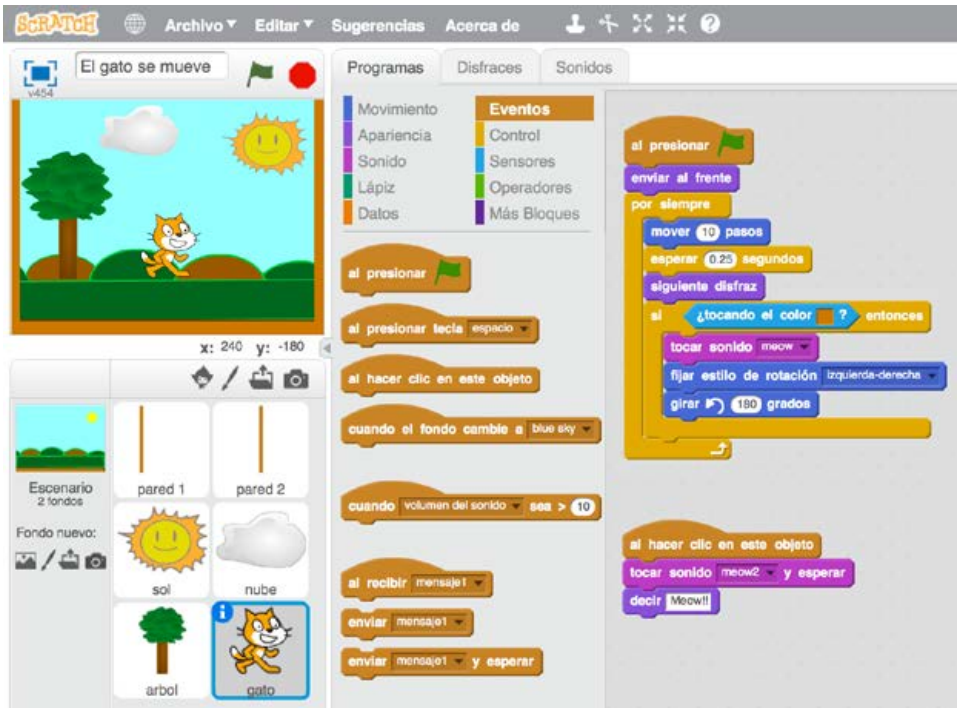


Figura 14. Interfaz de Scratch.

Fuente: elaboración propia con la herramienta Scratch (MIT, 2013)

La forma en la que el usuario de Scratch puede insertar código de programación es a través de los bloques de comandos que se arrastran y sueltan dentro de los espacios habilitados para ello (pestaña "Programas" de personajes, escenarios, etc. Ver *Figura 14*). El diseño de estos bloques está basado en los juegos de construcción, como LEGO. Este diseño permite presentar, de una forma asequible,

[38] En informática gráfica, el término *sprite* hace referencia a una imagen o mapa de bits bidimensional que se integra en una escena más grande.

la lógica de la programación. El usuario construye programas agrupando bloques gráficos como si fueran piezas de un rompecabezas. Estos bloques representan las estructuras de programación, y las acciones que se pueden realizar dentro del programa (mover un objeto, reproducir un sonido, etc.). Cada bloque tiene una forma diferente, y hay ciertas piezas que se pueden unir entre ellas, y otras no. Encajando las piezas donde la unión es posible se construyen “pilas” de bloques que configuran estructuras de programación sintácticamente correctas. Este método de programación de Scratch, como la de otros entornos por bloques similares, busca simplificar la programación eliminando la capacidad de cometer errores de sintaxis.

Se puede ejecutar un bloque individual o una pila de bloques haciendo clic con el ratón sobre ellos, siendo sus efectos visibles de inmediato. Además de con el ratón, hay otra forma de lanzar una secuencia de acciones representada por una pila de bloques. Hay ciertos bloques dentro de la categoría “Eventos” donde las piezas tienen forma de sombrero. Esta forma no es casual, son piezas pensadas para ponerlas en la primera posición de una pila de bloques. Esa pila se activará en respuesta a algún evento en tiempo de ejecución, como por ejemplo, que se inicie el programa, se presione una tecla determinada, o se haga clic con el ratón en una imagen. Varias pilas pueden ejecutarse al mismo tiempo de modo que un usuario puede crear paralelismo en su programa, en ocasiones sin darse cuenta.

El vocabulario de Scratch incluye más de 100 comandos con distintas utilidades: posicionamiento absoluto usando coordenadas cartesianas, movimiento relativo, transformación de imagen (rotación, escalado y efectos), animación de *sprites* (intercambio entre disfraces), reproducción de sonidos y música, o incluso creación de dibujos en tiempo de ejecución a través de un lápiz programable. Dado que muchos de estos comandos toman números como parámetros, el usuario Scratch tiene un contexto significativo para mejorar su comprensión de los números. Por ejemplo, usar un argumento negativo con el comando Mover hace que el *sprite* se mueva hacia atrás.

Scratch tiene capacidad para realizar operaciones aritméticas, de comparación y booleanas. Hay bloques que sirven para detectar cuando un *sprite* está tocando el borde o un color en particular. Igualmente existen bloques que informan de la ubicación del ratón, o de la tecla que se ha pulsado.

Scratch tiene varias estructuras de control, incluyendo sentencias condicionales (si, si-si no) y bucles (repetir, por siempre, repetir hasta que).
Además, Scratch permite crear variables y listas. Las variables pueden ser de dos tipos: visibles para todos los objetos, o solo para el objeto seleccionado. A

continuación, en la *Figura 15*, se puede ver un ejemplo de cómo se pueden crear variables o listas en Scratch, y cómo se les asigna un nombre a esas variables.

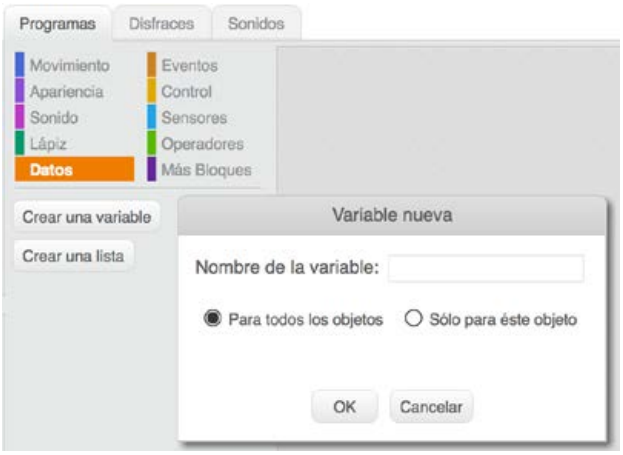


Figura 15. Interfaz para crear variables en Scratch.

Fuente: elaboración propia con la herramienta Scratch (MIT, 2013).

Las variables visibles por todos los objetos se pueden utilizar como mecanismo de comunicación entre diferentes *sprites.* Otra forma de comunicación asíncrona se realiza a través de la recepción y envío de mensajes mediante los bloques “enviar” y “al recibir”. Estos bloques se podrán poner en el mismo objeto, o en objetos distintos. Los mensajes son cadenas de texto que se pueden crear a través de estos bloques. Si el mensaje que se envía es igual que el que se recibe, se desencadenará la secuencia de acciones relacionada, esto es, la pila de bloques anexada al bloque “al recibir”. En la *Figura 16* se ve cómo son estas piezas, u cómo se puede crear un nuevo mensaje. Todos los mensajes creados estarán disponibles en la lista desplegable de cada una de estas piezas.

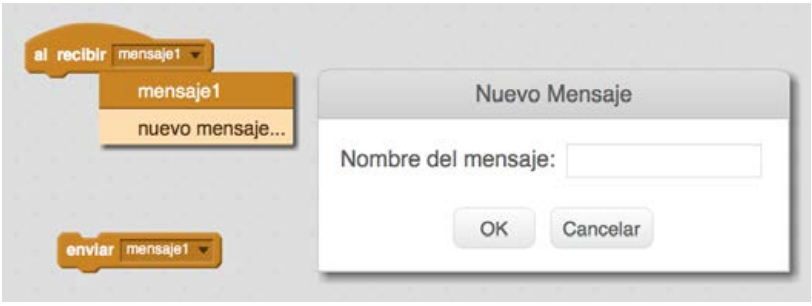


Figura 16. Interfaz para enviar y recibir mensajes en Scratch.

Fuente: elaboración propia con la herramienta Scratch (MIT, 2013).

Scratch también posibilita la creación de bloques definidos por el usuario a través de la categoría “Más Bloques”. Cuando se crea un nuevo bloque realmente se está definiendo una función. A ese bloque personalizado se le añadirán los parámetros que recibe la función, y opcionalmente etiquetas de texto para conectar esos parámetros, cuya utilidad es hacer que el bloque en su conjunto tenga cierto sentido (ver *Figura 17*). Al poder crear funciones, se puede también implementar recursividad. Esto es un ejemplo de que, a pesar de ser un entorno concebido para un público infantil y juvenil, Scratch tiene versatilidad para codificar programas complejos.

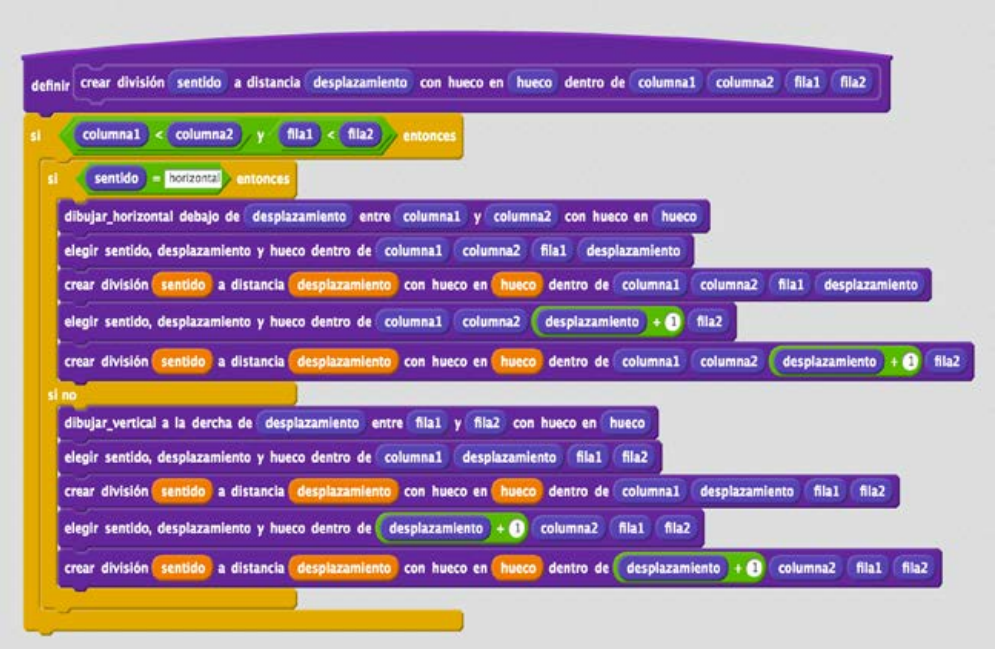


Figura 17. Ejemplo de bloque personalizado de Scratch que implementa una función recursiva para crear un laberinto a base de trazar líneas que van dividiendo el espacio.

Fuente: elaboración propia con la herramienta Scratch (MIT, 2013) a partir de una idea original de Juan Félix Mateo.

Al ejecutar un programa, las pilas de bloques se iluminan cuando se están llevando a cabo las acciones que hay definidas en ellas. Esto permite hacer un seguimiento del recorrido que hace el código en su ejecución. El hecho de tener los paneles separados posibilita que se puedan ver a la vez lo que sucede en el escenario y el código que se está ejecutando.

Con respecto al diseño de la interfaz, por último, merece la pena reseñar por su relevancia con esta tesis, la presencia de un panel de ayuda en el lateral derecho que incluye tutoriales y explicaciones muy completas sobre cómo empezar a

manejar Scratch, cómo hacer proyectos concretos (historias, juegos, etc.), o para qué sirven y cómo se utilizan los bloques.

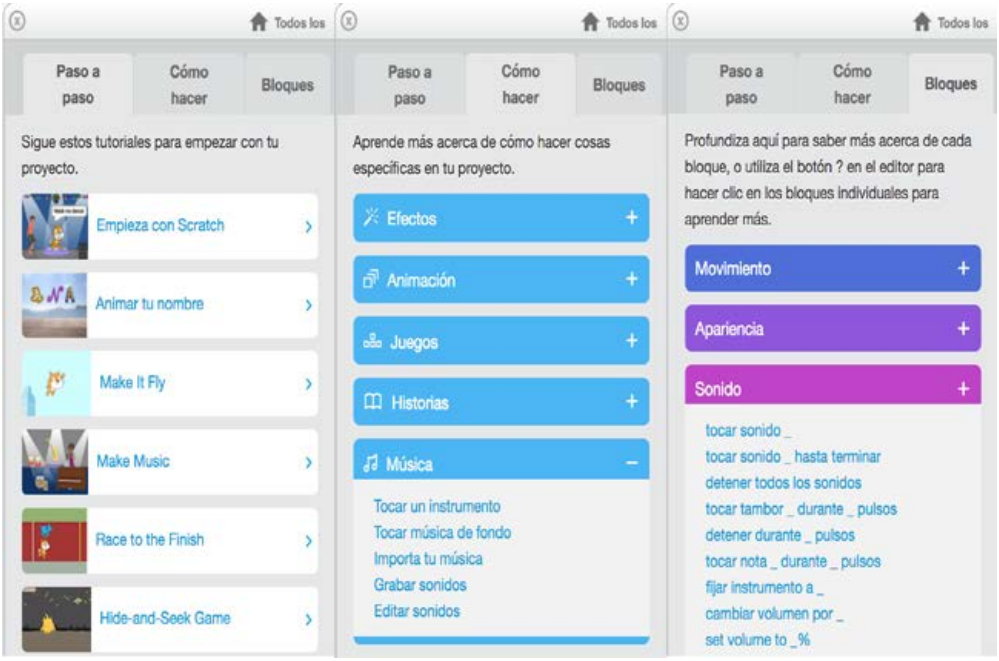


Figura 18. Visión de las tres pestañas del panel de ayuda de Scratch.

Fuente: elaboración propia a partir de la herramienta Scratch (MIT, 2013).

Scratch tiene también una gran componente social. Los proyectos creados con Scratch se pueden compartir a través de la plataforma web que el MIT ha creado para tal efecto, dando así la posibilidad al resto de usuarios tanto de probarlos, como de acceder al código fuente (en este caso, los bloques), para poder mezclarlo con código propio en nuevos programas. Esto, junto con la gran cantidad de documentación disponible, ha propiciado que exista una enorme comunidad en torno a Scratch, y que sea tan popular. En marzo de 2017, Scratch ocupa el puesto 20 en el índice TIOBE[39], que mide la popularidad de los lenguajes de programación basándose en los resultados de los motores de búsqueda.

[39] http://www.tiobe.com/

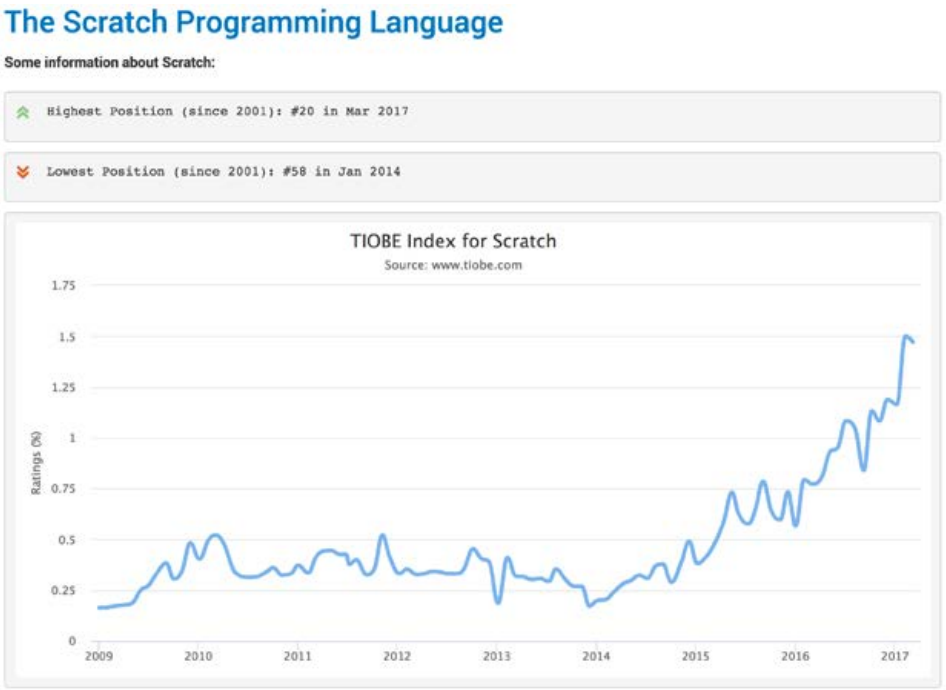


Figura 19. Gráfica de la evolución del índice TIOBE[40] de la herramienta Scratch.

Fuente: (TIOBE, 2017)

6.2. Principios clave del diseño de Scratch

En palabras de Resnick *et al.* (2009) los tres principios clave del diseño de Scratch son: "*Make it more tinkerable, more meaningful, and more social than other programming environments*", que podríamos traducirlo como "hacerlo más experimentable, más significativo y más social que otros entornos de programación". El diseño de la interfaz de usuario de Scratch surgió de un deseo de hacer sus conceptos clave tan tangibles y manifiestos como fuera posible (Maloney, Peppler, Kafai, Resnick y Rusk, 2008).

Resnick y Rosenbaum (2013) nos aclaran lo que quieren decir con el concepto *tinkerable.* En inglés, *to tinker* significa remendar o reparar. En el contexto de Scratch *tinkerable* se refiere a las posibilidades que ofrece la herramienta de explorar, probar algo, ver que no funciona y arreglarlo, y así llegar a nuevas ideas, en un proceso constante. Con significativo, o traduciéndolo literalmente *meaningful,*

[40] El índice de comunidad de programación TIOBE es un indicador de la popularidad de los lenguajes de programación. Las calificaciones se basan en el número de programadores cualificados que se estima que hay en todo el mundo, el número de cursos y proveedores de terceros, y el número de búsquedas en los principales motores. El índice TIOBE no indica cuál es el mejor lenguaje de programación ni con cuál se han escrito más líneas de código (TIOBE, 2017).

lleno de significado, los autores buscan reflejar que Scratch se concibió para ser más intuitivo, y más fácil de comprender y utilizar que cualquier otro entorno. El hecho de que sea más sociable también confiere a la herramienta una cierta componente de accesibilidad: hay multitud de proyectos compartidos, y son accesibles simplemente teniendo un ordenador y una conexión a internet. Esto, y la posibilidad de acceder al código de otros para mezclarlo con el propio multiplica las posibilidades de acceso al conocimiento (también se podría argumentar que un exceso de información perjudica más que beneficia, pues dificulta encontrar lo que se busca, o que no toda la información disponible es fiable, pero esto sería otro debate).

Resnick *et al.* (2009) también explican que en Scratch se trataron de seguir los principios de "*low floor*" y "*high ceiling*" que Papert enunció sobre las cualidades que debía tener el diseño de un entorno de programación. A la aportación de Papert, Resnick *et al.* (2009) además añaden que también debe ser "*wide walls*".

Al referirse a *low floor* (de suelo bajo), los autores quieren hacer una metáfora sobre bajar los requisitos para acceder al entorno, es decir, el diseño debe posibilitar que un usuario que no tenga experiencia previa con la informática o la programación no tenga problemas para empezar a utilizar Scratch, y vea resultados con relativa rapidez. Cuando hablan de *high ceiling* (de techo alto), quieren decir que a pesar de la sencillez a la que se refiere *low floor*, el sistema debe ser suficientemente potente como para implementar estructuras complejas y programas sofisticados. Con *wide walls* (de paredes amplias) los autores se refieren a la versatilidad que debe ofrecer la herramienta, y no solo servir para aprender a programar, sino también para poder trabajar muchas otras áreas (matemáticas, ciencias, arte, música, etc.).

6.3. Estudios aplicados a Scratch

Scratch ha sido evaluado en varios contextos. A continuación se describen brevemente algunos ejemplos significativos, llevados a cabo tanto dentro del ámbito de la educación reglada como fuera de ella.

Meerbaum-Salant *et al.* (2013) diseñaron un plan de formación de Scratch de dos horas de duración, y observaron su implantación en dos aulas de noveno grado (adolescentes con 14 – 15 años). Un análisis de las calificaciones que estos estudiantes obtuvieron en un pre-test y post-test de conceptos sobre Informática, mostró una mejora significativa después de usar Scratch, a pesar de que estos exámenes incluían conceptos abstractos como inicialización, variables y concurrencia.

Maloney et al. (2008) describen su experiencia utilizando Scratch en un centro extraescolar urbano analizando los programas creados en este contexto. En este estudio se pedía escoger a los participantes entre varios softwares de diseño, y Scratch se reveló como el favorito. También se vio que durante las primeras semanas de uso de esta herramienta, alrededor del 20% de los proyectos incluían solo manipulación de medios pero no código, y aproximadamente la mitad de los programas restantes utilizaban bucles e interacción con el usuario. Sin embargo, a partir del primer trimestre los programas examinados ya incluían sentencias condicionales y de sincronización. Es decir, con el paso del tiempo los participantes fueron capaces de desarrollar programas cada vez más complejos.

También existe diversos estudios que centran su foco en el concepto de *wide walls* al que se referían Resnick *et al.* (2009). Por ejemplo, Burke (2012) explica en las conclusiones de su estudio que Scratch, además de utilizarse para enseñar a programar a niñas y niños en edades tempranas, también puede servir para que ejerciten su habilidad lectoescritora. Otros ejemplos los encontramos en Fessakis et al. (2013) o Khan et al. (2011), que en sus investigaciones encontraron evidencias de la mejora en la adquisición de conceptos matemáticos tras haber utilizado Scratch. También podemos citar el estudio de Peppler y Kafai (2007) donde destacan que Scratch es un excelente medio para potenciar y estimular las capacidades artísticas y creativas de los estudiantes.

Por último decir que, como se ha visto en el capítulo 5.3, los estudios de usabilidad aplicados a lenguajes o a APIs de programación son habituales, y tratan de encontrar mecanismos para hacerlos más asequible en su uso. El autor de esta tesis no ha encontrado estudios relevantes donde se haga un análisis de usabilidad de la herramienta de Scratch.

—

PARTE EMPÍRICA

—

7. Antecedentes

Cada vez es más difícil abstraerse de la influencia que la tecnología está teniendo en nuestras vidas. Si observamos la progresión de los últimos años, todo apunta a que su presencia en el día a día del ciudadano del siglo XXI sea cada vez mayor. Este incremento va a conllevar también la necesidad de un mayor nivel de conocimiento de las Tecnologías de la Información[41] (Conde Melguizo y Rozalén, 2012). La evolución natural sugiere tomar el camino que nos lleve a desarrollar la capacidad de entenderlas a un nivel más profundo.

Los lenguajes de programación son el nivel más básico de comunicación con un componente tecnológico. Por tanto, entender sus fundamentos abre una vía para una mejor comprensión del funcionamiento de la tecnología. Es por esto que se hace plausible que, de la misma manera que el lenguaje verbal, el lenguaje matemático, o el lenguaje del arte están incluidos dentro de la educación elemental, los lenguajes de programación adquieran el mismo rango de importancia. Según se expone en capítulo 2.1, esta es la tendencia en Europa, y ya son mayoría los países que incluyen a nivel curricular la enseñanza de los lenguajes de programación, cada vez en etapas más tempranas. España no ha sido una excepción, y como se expone en el capítulo 2.3, esta materia ya está presente en las distintas etapas educativas.

Según lo expuesto en el capítulo 4.4.4, aprender a programar se presenta a priori como una tarea difícil (Caspersen y Bennedsen, 2007). La forma de abordar la enseñanza de la programación es un tema que lleva preocupando desde hace tiempo a la comunidad científica y educativa. Por este motivo se han realizado un número significativo de estudios al respecto (Cherubini et al., 2007; Myers et al., 2007; Norcio, 1982; Patel et al., 2008; Rosson y Carroll, 1996; Soloway et al., 1982), sin embargo la preocupación no se ha disipado, y sigue siendo un tema de total vigencia.

Como se refleja en el capítulo 5, cuando se utiliza una herramienta tecnológica como medio para la enseñanza de una materia, la usabilidad se revela como un factor clave. Una interfaz mal diseñada se convierte en una barrera para el aprendizaje efectivo (Kruse, 2003). Por tanto es pertinente analizar la usabilidad de dicha herramienta, y utilizar ese análisis para valorar su adecuación como instrumento de aprendizaje.

[41] Tecnologías de la Información, refiriéndose estas como ordenadores y dispositivos de telecomunicación utilizados para almacenar, manipular y transmitir datos (FOLDOC, 2017).

8. Planteamiento del problema

Como se ha visto en el capítulo 2.3, en la Comunidad de Madrid, ámbito en el que se desarrolla esta tesis, la enseñanza de la programación se incluye en el currículo de Primaria en la asignatura de libre configuración autonómica “Tecnología y recursos digitales para la mejora del aprendizaje”, según se refleja en el Decreto 89/2014 (Decreto Comunidad de Madrid, 2015, p. 90). Dentro de los contenidos de esta asignatura, hay uno concreto denominado “Fundamentos de programación. Creación de pequeños programas informáticos (Scratch)” (Decreto Comunidad de Madrid, 2015, p. 90). Es decir, esta asignatura señala de forma explícita la herramienta Scratch como instrumento para enseñar los fundamentos de la programación.

Como se ha explicado en los antecedentes, la usabilidad, como concepto empírico (Hassan Montero y Ortega Santamaría, 2009, p. 9), puede servir como medio para valorar si Scratch es la herramienta óptima para el propósito por el cual se escoge. Si la herramienta no es usable, aprender a utilizarla eclipsará el verdadero objetivo que marca la ley, es decir, aprender a programar.

8.1. Pregunta de investigación

La pregunta de investigación que planteamos es la siguiente: ¿Es Scratch, desde el punto de vista de la usabilidad, una herramienta adecuada para aprender a programar, en el contexto de la Educación Primaria de la Comunidad de Madrid?

Los conceptos que sostienen esta pregunta son los siguientes:

- Scratch: el proyecto de Scratch y sus características se describen en el capítulo 6.
- Lenguaje de programación: en el capítulo 3 se explica qué es un lenguaje de programación, se describen sus distintas tipologías, y se explica las reglas sintácticas propias de un lenguaje estructurado como el que utiliza Scratch.
- Aprender: en el capítulo 4.1 se hace un recorrido por las diferentes teorías de aprendizaje.
- Aprender a programar: en el capítulo 4.4 se enlazan las teorías de aprendizaje con el aprendizaje concreto de la programación. Asimismo, se describen las capacidades a desarrollar en este proceso (por ejemplo, el pensamiento computacional), y las posibles técnicas, metodologías y herramientas a utilizar.

- Usabilidad: en el capítulo 5 se explica lo que es usabilidad, y dado su carácter empírico, se describen los posibles mecanismos para su medición, tanto en ámbitos generales, como en los específicos relacionados con la programación.
- Educación Primaria en la Comunidad de Madrid: todo lo relacionado con este contexto se describe en el capítulo 2.

La hipótesis de investigación es que Scratch es una herramienta que facilita el aprendizaje de la programación desde el punto de vista de la usabilidad. De validarse esta hipótesis, habría una evidencia más, que reforzaría la idea de que Scratch es la herramienta que se debe utilizar en esta nueva asignatura de libre configuración que propone la Comunidad de Madrid. Aunque la investigación se encamine a contrastar la hipótesis, se espera que en su desarrollo aparezca información adicional que sea relevante para los objetivos y la pregunta de investigación

8.2. Objetivos de la investigación

A continuación se enuncian los objetivos generales y específicos de la parte empírica.

8.2.1. General

- **Objetivo General (OG):** Evaluar la herramienta Scratch desde el punto de vista de la usabilidad para determinar la adecuación de su uso para el aprendizaje de la programación, en el contexto de la Educación Primaria en la Comunidad de Madrid.

8.2.2. Específicos

- **Objetivo Específico 1 (OE1):** Evaluar la usabilidad de la herramienta Scratch
- **Objetivo Específico 2 (OE2):** Estudiar si la usabilidad de Scratch facilita o dificulta el aprendizaje de la programación.
- **Objetivo Específico 3 (OE3):** Diseñar y validar un instrumento que permita evaluar la adecuación de una herramienta de aprendizaje de la programación desde el punto de vista de su usabilidad.

9. Metodología

La presente investigación se encuadra dentro de la metodología descriptiva (Hernández Sampieri et al., 2004), de muestra estructural (Ibáñez, 2003), mediante la técnica cuantitativa de la encuesta (Beltrán, 2000).

En un estudio cuantitativo descriptivo se definen una serie de cuestiones para recabar información, y se miden los datos recogidos sobre cada una de ellas, para así describir lo que se investiga (Hernández Sampieri et al., 2004, p. 95). En este caso lo que se investiga es la usabilidad de Scratch como indicador de la adecuación de su uso para aprender a programar. La usabilidad como concepto empírico proporciona las variables necesarias que sustentan un estudio descriptivo en la investigación que nos atañe (Hernández Sampieri et al., 2004, p. 104).

9.1. Instrumentos y procedimientos de recogida y análisis de datos

Para cumplir con los objetivos de la investigación, el análisis de Scratch se realizó para su versión en castellano, y se planificó en tres fases: análisis de experto, análisis heurístico y análisis de usuario. Los detalles de estos instrumentos de análisis, y la justificación de su validez en un estudio de usabilidad, se detallan en el marco teórico, en el capítulo 5.2. Análisis de experto, heurístico y de usuario se desarrollan con detalle en los capítulos 10.1, 10.2, y 10.3 respectivamente. En este apartado se hace una breve descripción de los mismos, a efectos de especificar los instrumentos y procedimientos utilizados.

En el análisis de experto se determinaron las normas de usabilidad que se debían tomar como referentes en las siguientes fases del estudio. En el análisis heurístico, de entre las normas escogidas, se definieron y evaluaron las pautas necesarias para comprobar si Scratch es una herramienta usable. Estas pautas debían ser los indicadores que permitieran verificar el cumplimiento de los objetivos establecidos en la investigación (ver capítulo 8.2). La medición de estos indicadores se realizó a través del cuestionario, cuyas preguntas surgieron de las pautas procedentes del análisis heurístico, y de los estándares de aprendizaje de la programación presentados en el marco teórico (ver capítulo 10.3.1). Las variables que operacionalizaron los indicadores fueron las posibles respuestas a las que daba opción el cuestionario. Estas respuestas podían ser de tres tipos:

- Respuestas abiertas
- Respuestas binarias (sí / no)
- Respuestas múltiples, siguiendo una Escala de Likert (Wuensch, 2005) que mida el grado de autonomía de los participantes para realizar las tareas sobre las que cuestiona el formulario.

En la planificación de la investigación se previó la posibilidad de que hubiera ciertos aspectos externos a la herramienta que pudieran tener influencia en las respuestas recogidas: la disponibilidad de ordenador en casa, la formación recibida en el manejo de la herramienta, y la posible práctica autónoma con ella fuera del aula. En esta fase se decidió que inicialmente no se iba a hacer un estudio correlacional (Hernández Sampieri et al., 2004, p. 97), solo se iba a estudiar las tendencias señaladas por los datos recogidos, para ver si merecería la pena hacerlo. La razón se debe a que existen precedentes en otras investigaciones que apuntan a que pueden existir factores externos al diseño del programa que condicionen el análisis de usabilidad (Aránega Pardo, 2009; Holzinger, Searle y Wernbacher, 2011; Sonderegger y Sauer, 2010). Sin embargo, en la fase de planificación no se puede saber la pertinencia de hacer un estudio de este tipo, esta decisión solo es posible tomarla estudiando los datos recogidos. Por tanto, se decidió incluir en la investigación un análisis bivariable para observar las tendencias, y se subordinó la decisión de hacer un estudio correlacional a los resultados de este análisis. En el capítulo 10.3.6 se explica cómo se resolvió este tema.

9.2. Población y muestra

Con respecto a la muestra, al ser el objeto de esta investigación la evaluación de una tecnología, las decisiones muestrales no buscan la representación estadística, ni tienen como objetivo que los datos recogidos sean extrapolables estadísticamente a la población general. En lugar de la inferencia, este estudio pretende comprender mejor el significado del fenómeno estudiado, y el de sus conceptos asociados (Valles Martínez, 2003, p. 92).

En este tipo de investigación estamos hablando de un muestreo teórico (Glaser y Strauss, 1967; Valles Martínez, 2002, p. 93) o un muestreo estructural (Ibáñez, 2003, p. 89). La selección y agrupamiento de participantes en la investigación no responde a criterios estadísticos, sino estructurales. La muestra de población necesaria para realizar, por ejemplo, una encuesta, es un conjunto extraído de criterios probabilísticos, pero los participantes en una muestra estructural se

seleccionan en función de un conjunto tipológico, y están cercados por una frontera que les define (Ibáñez, 2003, p. 73). La frontera o límite que define a los individuos del interior del conjunto muestral frente al exterior es artificial, ya que ha sido trazada por el investigador y responde a su fundamentación teórica y sus intereses en la investigación (Ibáñez, 2003, p. 74). Por ello, este muestreo es también conocido como muestreo teórico, pues consiste en la selección estratégica de casos, siguiendo las pautas de un esquema conceptual previo del que se deriva una tipología (Glaser y Strauss, 1967; Valles Martínez, 2002, p. 93). Los detalles de la muestra con la que se ha realizado el estudio se desarrollan en el capítulo 10.3.4.

10. Análisis y resultados

10.1. Análisis de experto

La primera fase del estudio consistió en reunir a un grupo de expertos en los ámbitos relativos a esta investigación. Los expertos que se consideraron como aptos para la consecución de los objetivos, pertenecen a los siguientes colectivos:

- Grupo de investigación TSIC[42] (Sistemas Telemáticos para la Sociedad de la Información y el Conocimiento) de la Universidad Politécnica de Madrid.
- Equipo de Dirección del grado en Diseño Multimedia y Gráfico[43] de ESNE y la Universidad Camilo José Cela.
- Profesorado de Educación Primaria, expertos en TIC, y con experiencia docente con la herramienta Scratch.

Los investigadores del grupo TSIC fueron escogidos por su amplia experiencia en llevar la tecnología a espacios sociales donde a priori pueden surgir dificultades (niños, tercera edad, entornos de subdesarrollo, etc.). El equipo de Dirección del grado en Diseño Multimedia y Gráfico fue escogido por ser expertos en usabilidad, y por haber formado parte del comité que configuró el plan de estudios del Máster Oficial Universitario en Experiencia de Usuario para el Diseño de Productos y Servicios Digitales[44], de la mano de ESNE y Telefónica I+D. Igualmente se contó con profesores de Educación Primaria que utilizan Scratch en el aula, dado que el estudio se iba a desarrollar en ese ámbito.

De esta primera fase no se pretendía extraer conclusiones sobre el software, sino el correcto diseño y planificación de los siguientes pasos a seguir. Del análisis de experto se concluyó que la manera correcta de evaluar las limitaciones de Scratch por su diseño HCI tradicional persona-pantalla-teclado-ratón, sería someter al programa a la evaluación en función de las principales normas de análisis heurístico, la mayoría de ellas relacionadas con el diseño web, precisamente por ser web la principal interfaz de Scratch, más allá del editor *offline* del que también dispone esta herramienta. Concretamente las normas que los expertos consultados determinaron como las indicadas para el análisis heurístico son las siguientes:

[42] http://www.etsist.upm.es/investigacion/grinv/TSIC
[43] http://www.esne.es/oferta-academica/grados/grado-en-diseno-multimedia-y-grafico/
[44] http://www.esne.es/oferta-academica/posgrados/master-en-experiencia-de-usuario-para-el-diseno-de-productos-y-servicios-digitales/

- *Nielsen Norman Group Guidelines* (Nielsen, 1994). Esta norma se escogió por ser el principal referente en usabilidad.

- *HHS Guidelines.* (Shneiderman y Leavitt, 2006). Esta norma se escogió por la amplitud de áreas que abarcaba, lo cual posibilitaba el análisis de distintos aspectos de la herramienta Scratch. Otro motivo para escoger esta guía es la información que se asocia a cada pauta, para la que la norma refleja su "Importancia Relativa" *("Relative Importance")* y su "Medida de Confianza" *("Strength of Evidence").* La Importancia Relativa es una valoración emitida por un nutrido grupo de expertos encuestados, que valoran la repercusión que esa pauta tiene en la usabilidad de una aplicación. Este dato tiene un valor entre 1 y 5, asignándose 5 a las pautas consideradas como más importantes, y 1 a las que menos. El mismo baremo se utiliza en la Medida de Confianza, (valor de 5 para las que más confianza ofrecen, y 1 a las que menos), basándose esta calificación en las evidencias existentes en investigaciones, estudios, resultados de experimentos, etc. sobre dicha pauta. En la *Figura 20* se ve un ejemplo de cómo aparecen las pautas en la web de las *HHS Guidelines.*

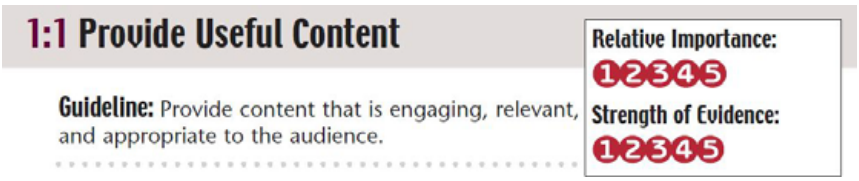


Figura 20. Captura de la norma 1.1 de la *HHS Guidelines*.

Fuente: (Shneiderman y Leavitt, 2006)

- *MIT Usability Guidelines* (MIT y IST, 2011). Estas pautas se utilizan fundamentalmente en la evaluación del diseño web, y no todas son aplicables en el análisis de una herramienta como Scratch. Se escogieron por el interés que puede tener someter Scratch, un software desarrollado por el MIT, a las directrices de usabilidad de la web IST (*Information Services & Technology*) del propio MIT (2011).

- *Nokia Usability Guidelines for Games* (Nokia, 2003). La normativa de Nokia ofrece una guía de usabilidad en el diseño de la interfaz de los videojuegos. A pesar de no tratarse de una norma tan profunda como las anteriores, al ser Scratch un sistema que permite la creación de videojuegos, y no existir otros

referentes de mayor importancia en este campo concreto, se decidió incluirla en el análisis heurístico.

- *Australian Government Usability Checklist* (AGIMO, 2008). Esta guía se escogió por estar entre las ocho normas más utilizadas en estudios de usabilidad, y por ser sencilla de entender.

10.2. Análisis heurístico

En este apartado se describen, dentro de las normas escogidas, las pautas concretas que, por su relevancia con el estudio, se eligieron para llevar a cabo el análisis heurístico. Igualmente, se indica si a tenor de la opinión de los expertos consultados, y situándose en el mejor de los casos, la pauta a priori se cumple o no se cumple, o si no hay evidencias que permitan afirmar con seguridad su cumplimiento. Cuando decimos "situándose en el mejor de los casos", nos referimos a que cuando se afirma que una pauta se cumple, es porque lo hace de forma general, aunque tal vez un análisis más exhaustivo pudiera dar otro resultado. A estas pautas se les va a dar el beneficio de la duda, para posteriormente verificar la afirmación de su cumplimiento en el análisis de usuario. No debemos olvidar que el análisis heurístico tiene como objetivo generar las preguntas del cuestionario del análisis de usuario.

10.2.1. Nielsen Norman Group Guidelines

Se estima que puede haber cerca de 2400 pautas en el total de publicaciones de Nielsen, recopiladas en la web del *Nielsen Norman Group* (2017). De entre todas ellas, las que probablemente sean las 10 más conocidas son las que se utilizan en el presente estudio (Nielsen, 1994). En este apartado se describen como una interpretación de las originales escritas en inglés, tomando como referencia la revisión que hace de este tema Carreras (2012).

Cada pauta se identifica con el prefijo NNGG y una numeración, para poder hacer referencia a ella en apartados posteriores de esta tesis. Esta numeración coincide con la establecida por Carreras (2012) en su investigación.

NNGG1. Visibilidad del estado del sistema. Esta pauta no se cumple. El sistema no muestra mensajes sobre su estado, aunque por la forma en la que

está concebido Scratch, estos mensajes supuestamente no parecen necesarios.

NNGG2. Utilizar el lenguaje de los usuarios. Esta pauta sí se cumple. Además de que Scratch está traducido a varios idiomas, incluido el castellano, la interfaz utiliza algunos términos cercanos al público infantil y juvenil al que va dirigido.

NNGG3. Control y libertad para el usuario. Esta pauta se cumple parcialmente. Scratch restringe las acciones que puede hacer el usuario de forma deliberada para minimizar la posibilidad de que se cometan errores (por ejemplo, no se pueden encajar dos piezas cualesquiera). Sin embargo, dentro de estas restricciones, las acciones que permite el programa son incontables, si tenemos por ejemplo en cuenta los resultados que se pueden conseguir combinando piezas.

NNGG4. Consistencia y estándares. Esta pauta sí se cumple. Los distintos elementos de Scratch se distribuyen en el espacio de trabajo de forma similar a como lo hacen otros programas (por ejemplo, menú principal en la parte superior, dentro de menú la opción “Archivo” que permite crear nuevo proyecto, etc.).

NNGG5. Prevención de errores. Esta pauta sí se cumple. Como se menciona en la pauta NNGG3, el diseño de Scratch está pensado para minimizar la posibilidad de cometer errores que dejen el programa en un estado inconsistente o inestable. Los errores que se previenen son fundamentalmente sintácticos (solo se pueden combinar las piezas para las que desde el punto de vista de la programación tenga sentido hacerlo), pero no semánticos, es decir, el usuario puede codificar un programa incorrecto, entendiendo incorrecto como que no hace lo que debería hacer. Cabe mencionar que si el usuario se equivoca, y quiere deshacer una acción, el programa no ofrece una opción sencilla para volver a un estado anterior.

NNGG6. Minimizar la carga de la memoria del usuario. Esta pauta sí se cumple. Scratch utiliza distintas técnicas para que el usuario no tenga que memorizar demasiada información. Por ejemplo, las piezas tienen una forma, un color, y un texto descriptivo dentro de ellas que permite identificar la acción que realiza con un simple golpe de vista.

NNGG7. Flexibilidad y eficiencia de uso. Esta pauta se cumple parcialmente. Con respecto a la flexibilidad, las posibilidades de combinación de las piezas, hacen que se puedan realizar un sinfín de tareas diferentes. Con respecto a la eficiencia, es posible ver inmediatamente los efectos de casi cualquier acción de Scratch (por ejemplo, al colocar una pieza, o una pila de ellas en el área de "Programa", y hacer clic sobre ella), con lo que se pueden hacer pruebas intermedias para evitar llegar a un código terminado que no cumpla con el propósito para el que fue diseñado. Sin embargo, hay aspectos del diseño de Scratch que hacen que se tarde más de lo deseable en hacer ciertas tareas sencillas, por ejemplo, sacar una pieza de una pila (al arrastrar una pieza fuera de una pila, se arrastran también todas las piezas con las que esté enganchadas).

NNGG8. Diálogos estéticos y diseño minimalista. Esta pauta sí se cumple. En los textos que se muestran se usan palabras y conceptos muy sencillos, y en un número limitado, para no confundir al usuario.

NNGG9. Ayudar a los usuarios a reconocer, diagnosticar y recuperarse de los errores. Esta pauta no se cumple. Como se ha mencionado en el análisis de la pauta NNGG5, Scratch busca minimizar los errores sintácticos cometidos al no dejar combinarse entre ellas ciertas piezas, pero no dejar que se combinen no es en sí mismo un mensaje de error. Tampoco hay mensajes que explique por qué esas piezas no se pueden combinar. Por otro lado, aunque los errores sintácticos sean marginales, los errores semánticos sí son posibles. La herramienta no ofrece un camino fácil para enmendar estos errores, ni existen mensajes de error al respecto, aunque esto en cierto modo es lógico, no hay forma de que Scratch sepa lo que el usuario tenía en mente realizar. Precisamente un lenguaje de programación tiene ese objetivo: servir de instrumento para comunicar al ordenador lo que se quiere hacer.

NNGG10. Ayuda y documentación. Esta pauta sí se cumple, existe tanto un sistema de ayuda, como una guía para realizar proyectos tipo, que no solo incluye texto, sino también imágenes, vídeos, etc., que ilustran las acciones a realizar. La ayuda está indexada por categorías, para facilitar la búsqueda de información. Además, en la parte superior de la interfaz, hay un botón con el símbolo de interrogación que, pulsándolo primero, y después haciendo clic sobre una pieza, muestra inmediatamente su utilidad. Esto lo podemos ver en la *Figura 21*, en la que se ha buscado ayuda sobre la pieza "mover 10 pasos". Cabe mencionar que el sistema de ayuda podría ser más completo si incluyera

un motor de búsqueda, o si para cada pieza no solo se incluyera un ejemplo y se mostraran más posibilidades de uso.

Figura 21. Botón de ayuda de Scratch.

Fuente. Elaboración propia con la herramienta Scratch (MIT, 2013)

10.2.2. HHS Guidelines

Los expertos consultados consideraron que no era necesario analizar las 209 directrices que incluye las *HHS Guidelines* (Shneiderman y Leavitt, 2006), dado que muchas de ellas están enfocadas al estudio de la web tradicional, y no aportan una información relevante con respecto a la usabilidad de Scratch. Por ejemplo, quedaron excluidas del análisis las pautas relativas al hardware y software, y a la apariencia del texto (títulos, encabezados, etc.).

Se consideraron aplicables al análisis de Scratch las pautas relacionadas con los procesos de diseño y evaluación, de optimización de la experiencia del usuario, y las utilizadas en los test de usabilidad. En la elección final, también se tuvo en cuenta la Importancia Relativa y la Medida de Confianza de cada pauta.

En este apartado las pautas elegidas se enuncian como una descripción de las originales escritas en inglés. Cada pauta se identifica con el prefijo HHS y una numeración. Esto se hace para poder referenciarlas en capítulos posteriores de la tesis. La numeración de cada pauta, según se establece en este capítulo, corresponde con la numeración original; por ejemplo, la que aquí se identifica como pauta HHS1.1 se corresponde con la pauta 1.1 documentada en *HHS Guidelines* (Shneiderman y Leavitt, 2006).

PROCESOS DE DISEÑO Y EVALUACIÓN:

HHS1.1. Proporciona contenido útil al usuario. Esta pauta sí se cumple, en general toda la información que aparece en el entorno de Scratch es relevante.

HHS1.2. Utiliza recursos para entender las necesidades del usuario. Esta pauta sí se cumple. Scratch tiene un equipo de investigadores detrás que evalúan constantemente los hábitos de los usuarios. El diseño de la herramienta está en constante evolución.

HHS1.3. Entiende las necesidades de los usuarios, y se adapta a sus expectativas. Esta pauta sí se cumple, la aplicación se ajusta a los requisitos de navegación, contenido, organización, etc., del tipo de usuarios que tiene (fundamentalmente público infantil y juvenil).

HHS1.4. Involucra a los usuarios en el diseño del sistema. Esta pauta sí se cumple. Scratch tiene espacios habilitados para hacer sugerencias.

HHS1.5. Tiene claros cuáles son los objetivos principales de la aplicación. Esta pauta sí se cumple. Todo el diseño de la interfaz de Scratch está pensado para ser una herramienta de programación con la que realizar videojuegos, historias interactivas, etc., de forma sencilla.

HHS1.6. El diseño da prioridad al rendimiento y a la facilidad para hacer las tareas, frente a cuestiones estéticas. Esta pauta sí se cumple. La interfaz de Scratch no contiene elementos cuyo único propósito sea adornar o decorar la pantalla. Cada decisión de diseño (por ejemplo, categorías de bloques con diferentes colores) tiene una utilidad práctica.

HHS1.7. Es posible encontrarla fácilmente en el Top 30 de búsquedas web. Esta pauta sí se cumple, al buscar Scratch en un buscador de referencia como Google, la aplicación aparece en primera posición.

HHS1.8. La aplicación es funcional y cumple con el propósito para el que fue diseñada. Esta pauta se cumple parcialmente. Scratch es una herramienta que sirve para crear videojuegos e historias interactivas. Su utilidad como herramienta de aprendizaje de la programación es lo que se cuestiona en esta tesis.

OPTIMIZACIÓN DE LA EXPERIENCIA DEL USUARIO:

HHS2.1. No hay pop-ups en las aplicaciones. Esta pauta sí se cumple, no se abren ventanas ni se generan procesos sin la acción previa del usuario.

HHS2.2. Para realizar determinadas tareas se utilizan secuencias de acciones de forma parecida a como se hacen en programas de índole similar. Esta pauta sí se cumple. Por ejemplo, los mecanismos para crear un nuevo proyecto, guardarlo, descargarlo, etc., siguen las estructuras clásicas de este tipo de aplicaciones.

HHS2.3. La aplicación está optimizada para reducir los tiempos de carga. Esta pauta sí se cumple, los tiempos de espera de carga son los habituales de cualquier página web.

HHS2.4. No precisa que el usuario memorice cómo llevar a cabo determinadas funciones dentro del programa. Esta pauta se cumple parcialmente. Hay ciertos procesos que se tienen que hacer en un orden determinado (por ejemplo, para crear un objeto que emite un sonido, se debe añadir primero un nuevo objeto, después añadir un sonido, después ir a la sección Programas de ese objeto, arrastrar el bloque que sirve para nuestro propósito, etc.), y este orden se debe memorizar. Decimos que se cumple en parte porque la mayoría de los procesos se explican en tutoriales incluidos dentro de Scratch, con lo que a priori no sería necesario memorizar nada si se utiliza esta ayuda que ofrece el programa. En cualquier caso, disponer de un lugar donde buscar información no garantiza resolver todas las dudas, porque para resolverlas, primero hay que saber qué se debe buscar (Hirsch Jr, 2000).

HHS2.5. Minimiza el tiempo de carga de la aplicación. Esta pauta sí se cumple, y es complementaria a la pauta HHS2.3.

HHS2.6. Avisa al usuario cuando está a punto de abandonar la aplicación. Esta pauta sí se cumple. Si el usuario abandona la aplicación sin haberla guardado, aparece un mensaje de alerta.

HHS2.7. Reproduce la información de una forma usable (útil, clara, accesible). Esta pauta sí se cumple. El lenguaje que utiliza es sencillo, y está adaptado al público para el que está destinado, fundamentalmente infantil y juvenil.

HHS2.8. El formato de texto está diseñado para que se pueda imprimir. Esta pauta no se cumple, la herramienta no está pensada para que se pueda imprimir lo que hay en la pantalla. Las pautas HHS2.8, HHS2.11 y HHS2.14 son complementarias.

HHS2.9. La aplicación informa al usuario cuando tiene que esperar para poder continuar. Esta pauta no se cumple, la aplicación no emite ningún tipo de mensaje a este respecto.

HHS2.10. Informa al usuario del tiempo que tiene que esperar para descargar un archivo. Esta pauta no se cumple, cuando se descarga un archivo de la página (un proyecto, un objeto, etc.), la información del tiempo de descarga la ofrece el navegador, pero no la aplicación de Scratch.

HHS2.11. Permite al usuario que pueda imprimir el entorno. Esta pauta no se cumple, la aplicación no tiene ningún instrumento que permita imprimir lo que hay en la pantalla. Las pautas HHS2.8, HHS2.11 y HHS2.14 son complementarias.

HHS2.12. No obliga al usuario a tener que simultanear la realización de acciones dentro de la aplicación, y la lectura de un texto. Esta pauta sí se cumple, no hay ninguna acción dentro de la aplicación que requiera hacer dos tareas a la vez.

HHS2.13. Existe documentación para explicar las funciones de la aplicación al usuario. Esta pauta sí se cumple. Scratch dispone de una ayuda muy completa que explica cada proceso que se puede realizar dentro de la aplicación.

HHS2.14. Permite la impresión. Esta pauta no se cumple. Las pautas HHS2.8, HHS2.11 y HHS2.14 son complementarias.

HHS2.15. Los usuarios pueden solicitar ayuda adicional. Esta pauta no se cumple, no existen mecanismos para solicitar ayuda adicional más allá que la que está incluida dentro de la aplicación.

HHS18. TEST DE USABILIDAD

HHS18.2. Los usuarios pueden contribuir con sus comentarios a la mejora del programa. Esta pauta sí se cumple. Existen mecanismos de comunicación con el equipo de desarrollo que se pueden utilizar para dar sugerencias de mejora**.**

HHS18.3. Se realizan estudios de usabilidad para realizar cambios en la aplicación. No existen evidencias que puedan justificar el cumplimiento de esta pauta. Sin embargo, se incluye como un elemento más para justificar la pertinencia de un estudio como el que se realiza esta tesis.

HHS18.4. Priorización de la usabilidad en la realización de tareas dentro de la aplicación. Esta pauta sí se cumple. Cada tarea que se puede hacer dentro de la aplicación prioriza la usabilidad frente a otros criterios.

HHS18.5. Distingue entre la frecuencia y la gravedad de los errores que se producen en la aplicación. Esta pauta no se cumple, no existen mecanismos dentro de Scratch para identificar los errores.

10.2.3. MIT Usability Guidelines

De las 62 directrices recogidas en las *MIT Usability Guidelines* (MIT y IST, 2011), solo se escogieron las que los expertos consultados consideraron directamente aplicables al análisis de usabilidad de Scratch. Por este motivo se quedaron fuera, por ejemplo, las pautas relativas a la navegación, propias de un análisis de web tradicional. También se excluyeron las pautas vinculadas a la ayuda del programa, o a los mensajes de error que emite, por ser muy similares a las descritas en otras normas, y no aportar nada nuevo ni más completo con respecto a éstas.

En este apartado las pautas a analizar se enuncian como una traducción de las originales escritas en inglés, y están agrupadas por las categorías que se describen en el capítulo 5.2. Asimismo, se identifica cada pauta con el prefijo MIT y una numeración, para de esta manera poder referenciarlas en apartados posteriores de la tesis.

- **Funcionalidad**

  MIT1. La aplicación se adapta tanto a principiantes como a expertos. Esta pauta sí se cumple, a través de Scratch se pueden realizar tanto tareas muy sencillas como muy complejas.

MIT2. Las funciones están claramente etiquetadas. Está pauta sí se cumple. Cada sección tiene su etiqueta correspondiente.

- **Control de usuario**

MIT3. El sitio refleja el flujo de trabajo del usuario. Esta pauta se cumple parcialmente. Aunque sí proporciona información en ciertos procesos (por ejemplo, se ilumina el código que se está ejecutando), también se pueden encontrar casos en los que Scratch es opaco (por ejemplo, no incorpora un sistema que indique porqué algo falla, o que señale dónde se produce un error).

MIT4. El usuario puede deshacer cualquier operación. Esta pauta se cumple parcialmente. El usuario puede deshacer la última operación realizada, pero no de forma sencilla. Los botones "Deshacer / Rehacer", y la funcionalidad asociada tradicionalmente al CTRL+Z solo existen cuando se modifica un disfraz, pero no cuando se coloca una pieza del programa. En este caso solo se permite recuperar la última pieza o bloque de piezas borradas mediante la opción "Recuperar borrado" del menú "Editar".

MIT5. Existe un punto de salida claro en cada página. Esta pauta sí se cumple, la transición y la forma de salir de cada pantalla está claramente identificada.

- **Lenguaje y contenido**

MIT6. La información relacionada con tareas importantes se muestra de forma destacada. Esta pauta sí se cumple. Por ejemplo, si se realiza una acción que provoca que se abandone la página sin guardar el proyecto, se muestra un mensaje al respecto, y no se continúa hasta que el usuario confirme su deseo de salir sin guardar.

MIT7. No se incluye información poco relevante o que se utilice en raras ocasiones. Esta pauta sí se cumple. El diseño de Scratch es minimalista, y la interfaz no incluye contenido que no sea necesario.

MIT8. El lenguaje es simple, sin jerga. Esta pauta se cumple parcialmente. En general el lenguaje que se utilizan dentro de Scratch podría considerarse que está adaptado al público infantil y juvenil (por ejemplo, el término "Disfraces" para referirse a los objetos gráficos que se pueden incluir en un

proyecto), sin embargo también utiliza terminología técnica (variables, objetos, sensores, etc.).

- **Consistencia**

  MIT9. La misma palabra o frase se usa de forma constante para describir un elemento. Esta pauta se cumple parcialmente. En general a cada elemento se le denomina de la misma manera en todos los apartados donde aparece, pero se pueden encontrar casos donde esto no es así. En la versión en castellano, la que se evalúa en la presente investigación, es frecuente encontrar palabras que no está traducidas. Por ejemplo, en las ilustraciones de la ayuda, las piezas aparecen algunas ve con su nombre en inglés, y no en el idioma escogido por el usuario (ver *Figura 21*).

- **Claridad arquitectónica y visual**

  MIT10. El diseño del sitio es directo y conciso. Esta pauta sí se cumple. El diseño de Scratch se puede calificar como eminentemente funcional. No se incluyen elementos exclusivamente decorativos.

  MIT11. La redundancia en el diseño del sitio solo se produce si esto beneficia a la productividad del usuario. Esta pauta sí se cumple, existe muy poca redundancia, y en los casos en los que existe tiene un propósito (por ejemplo, el nombre de cada objeto incluido en pantalla aparece tanto debajo de la escena como en su disfraz, para que no haya dudas de que uno se corresponde con el otro).

  MIT12. Existe espacio en blanco suficiente; la página no es demasiado densa. Esta pauta se cumple parcialmente. La interfaz no está sobrecargada de información, y deja espacio para elaborar los programas, aunque si el proyecto es muy complejo e involucra un gran número de piezas, la navegación por el espacio del programa se vuelve complicada, y se hace difícil localizar apartados concretos dentro del código.

10.2.4. Nokia Usability Guidelines for Games

Aunque Scratch es una herramienta que sirve para crear videojuegos, no es exactamente un videojuego. Por este motivo no todas las directrices incluidas en las *Nokia Usability Guidelines for Games* (Nokia, 2003) son aplicables en el análisis de su usabilidad. Por ejemplo, por no tratarse de elementos incluidos en Scratch, se

han excluido pautas relativas al diseño de pantallas para juegos multijugador, o las relacionadas con la creación de personajes controlados por inteligencia artificial. También se han quedado fuera ciertas pautas que se abordan con mayor nivel de detalle en otras normas contempladas en el análisis heurístico, como por ejemplo algunas directrices relativas a la ayuda que ofrece el programa.

En este apartado las pautas a analizar se enuncian como una interpretación de las originales escritas en inglés, traduciendo el término "*game"* (juego o videojuego) como "programa informático" o simplemente "programa", para ampliar el contexto de su aplicación. Las pautas se presentan agrupadas por las categorías que se describen en el capítulo 5.2. Asimismo, cada pauta se identifica con el prefijo NOKIA y una numeración para poder referenciarlas en apartados posteriores del estudio.

- **Pantalla de inicio y menú principal**

  NOKIA1. Utiliza una pantalla de inicio con una secuencia que invite al uso del programa, pero que no desvele demasiada información. Esta pauta se cumple parcialmente, no existe pantalla alguna de introducción, pero cada vez que se inicia un nuevo proyecto, el icónico gato de Scratch aparece en él, listo para que se le dote de vida.

- **Controles**

  NOKIA2. El programa está concebido para que el usuario no tenga que pulsar dos teclas a la vez, dado que en teclados pequeños esto puede suponer una complicación. Esta pauta sí se cumple, el programa puede usarse casi en su totalidad con el ratón, y en las ocasiones en las que el teclado es necesario, no es habitual tener que pulsar obligatoriamente una combinación de teclas de forma simultánea.

- **Pausa y guardar partida**

  NOKIA3. El programa ofrece la posibilidad de pausar o guardar en cualquier momento. Esta pauta sí se cumple, existe la posibilidad de guardar el proyecto y recuperarlo más tarde sin impedimentos.

- **Retroalimentación**

  NOKIA4. El programa proporciona un *feedback* claro sobre los elementos esenciales del programa. Esta pauta sí se cumple. Por ejemplo, se utiliza una simbología claramente reconocible para señalar cuándo el código

programado está en ejecución (la bandera verde se ilumina y el octógono rojo se apaga) o cuando está parado (el octógono rojo se ilumina, y la bandera verde se apaga). Igualmente se iluminan las piezas concretas que se están ejecutando en un momento dado, aunque tal vez este *feedback* se muestre durante un tiempo insuficiente, y es posible que se necesitara alguna indicación adicional.

- **Desafío**

  NOKIA5. Aprender a utilizar el programa lleva un minuto, y dominarlo toda una vida. Esta pauta se cumple parcialmente. El diseño de Scratch invita a la exploración, y es relativamente sencillo descubrir cómo funcionan los rudimentos básicos (arrastrar piezas al área indicada, combinarlas, etc.). Pero manejar lo básico no significa saber programar, y si no se tiene experiencia previa con otros lenguajes o entornos de programación, es posible que iniciarse en el manejo de Scratch de forma autónoma se aventure complicado, y mucho más dominarlo.

  NOKIA6. El programa ofrece recompensas al usuario para mantenerlo motivado. Esta pauta se cumple parcialmente. La mayor recompensa es poder ver funcionando el proyecto imaginado, y en esa línea Scratch permite ver de forma inmediata el resultado de un programa, aunque esté desarrollado solo en parte. Más allá de esto, Scratch no ofrece recompensas adicionales.

- **Ruido**

  NOKIA7. El programa permite que el usuario pueda ajustar el volumen de sonidos y música. Esta pauta no se cumple. Sólo será posible ajustar el volumen de sonidos y música a través de los controles del sistema operativo del dispositivo donde se esté utilizando Scratch, dado que dentro del programa no existe esta opción.

- **Ayuda**

  NOKIA8. El programa incluye consejos y sugerencias, que aparecen con el uso de ciertos elementos del programa. Esta pauta se cumple parcialmente, no se muestran consejos ni sugerencias durante el manejo del programa, aunque sí existe una ayuda con ejemplos donde se incluyen tutoriales que indican paso a paso cómo realizar ciertas tareas.

- **Puntuación**

  NOKIA9. El programa proporciona un medio para mostrar los logros conseguidos con la comunidad de usuarios. Esta norma sí se cumple, la web de Scratch permite compartir los proyectos realizados.

- **Reinicio**

  NOKIA10. El programa proporciona una forma sencilla de reinicio. Esta pauta se cumple parcialmente**.** Puede iniciarse un nuevo proyecto en cualquier momento, sin embargo no es posible realizar tareas como volver al estado original de un proyecto cargado después de haberlo modificado.

10.2.5. Australian Government Usability Checklist

La *Australian Government Usability Checklist* (AGIMO, 2008) contiene directrices que mayoritariamente están centradas en evaluar la usabilidad de páginas web. La interfaz de Scratch es web (se ejecuta en un navegador), pero no es una página web al uso. Por este motivo, dentro de esta norma se han excluido del análisis heurístico las pautas relacionadas con la estructuración propia de una página web tradicional que no están presentes en Scratch. Por poner un ejemplo, se han quedado fuera todas las pautas relativas a la creación de formularios.

En este apartado las pautas a analizar se enuncian como una traducción de las originales escritas en inglés, y se presentan agrupadas por las mismas categorías que se describen en el capítulo 5.2. Asimismo, cada pauta se identifica con el prefijo AGUC y una numeración para poder referenciarlas en apartados posteriores de la tesis.

- **Arquitectura y Navegación**

  AGUC1. ¿La estructura se ajusta a su finalidad? Esta pauta sí se cumple, los diferentes espacios de trabajo de Scratch están dispuestos según la función que deben desempeñar. Por ejemplo, el programa está concebido para que las acciones se lleven a cabo con cierto orden, y en ese sentido modificar la apariencia de un objeto (lo que en Scratch se denomina Disfraz) y su comportamiento (Programa) son tareas que se realizan de forma independiente (cambiar uno no tiene por qué afectar al otro), y nunca al mismo tiempo. Por este motivo, los paneles de Programas y Disfraces no se pueden visualizar a la vez. Sin embargo, sí es útil que el escenario esté siempre visible, independientemente del panel seleccionado (Programas, Disfraces o

Sonidos), de esta manera cuando se modifique un programa o un disfraz, se podrá ver instantáneamente el resultado de este cambio en el escenario.

AGUC2. ¿Está claro el esquema de navegación? Esta pauta se cumple parcialmente. No todos los elementos de Scratch donde se puede interactuar con el ratón están claramente representados. En general se pueden identificar dónde hay posibilidad de interacción por el contexto (objetos con forma de botón, elemento de un menú, etc.), pero el icono del ratón no cambia cuando está sobre un objeto sobre el que se puede hacer clic, algo que sin embargo es habitual en las interfaces web.

AGUC3. ¿Es posible encontrar lo que se busca? Esta pauta sí se cumple, cada zona del espacio de trabajo está claramente etiquetada.

AGUC4. ¿Hay un número razonable de elementos de navegación? Esta pauta sí se cumple, el número de elementos de navegación existentes es equivalente a las funcionalidades que ofrece Scratch. No existen menús saturados de opciones que puedan abrumar a un usuario novato.

AGUC5. ¿Hay alguna opción para buscar información? Esta pauta no se cumple, no existe un buscador de ayuda por palabras clave. En el área de trabajo de Scratch, desarrollada con Flash, tampoco está disponible la opción de búsqueda propia de los navegadores.

- **Diseño y maquetación**

AGUC6. ¿Es correcto el diseño? No existen evidencias para responder con rotundidad a esta pregunta. En cualquier caso, los expertos consultados coinciden en que el diseño a priori parece correcto, aunque es susceptible de mejoras.

AGUC7. ¿Se hace un uso correcto de la alineación y agrupación de los elementos? Esta pauta sí se cumple, la distribución de los elementos en el espacio de trabajo es armónica.

AGUC8. ¿Se hace un uso correcto del contraste? Esta pauta sí se cumple, los colores escogidos para los textos contrastan claramente con los utilizados en los fondos.

AGUC9. ¿Está el sitio ordenado? Esta pauta sí se cumple, los elementos del espacio de trabajo están correctamente organizados.

- **Plataforma e implementación**

AGUC10. ¿La aplicación carga en un tiempo razonable (entre 3 y 10 segundos)? Esta pauta sí se cumple, Scratch es una herramienta ligera que carga rápidamente.

AGUC11. ¿Funcionan todos los links? Esta pauta sí se cumple, todos los links funcionan correctamente.

AGUC12. ¿Funciona en cualquier navegador? ¿Funciona en cualquier dispositivo? Esta pauta no se cumple, al estar desarrollado en Flash, Scratch no se puede utilizar dispositivos que no soporten esta tecnología, como por ejemplo *tablets* y *smartphones* con sistemas operativos iOS y Android.

AGUC13. ¿Funciona en cualquier resolución? Esta pauta no se cumple, Scratch no tiene un diseño adaptativo, y en resoluciones pequeñas (anchura por debajo de los 800 píxeles) no se ve óptimamente.

- **Accesibilidad**

AGUC14. ¿Para cada elemento no textual se ofrece un texto equivalente? Esta pauta sí se cumple. Salvo contadas excepciones, los botones incorporan un *tooltip* (descripción emergente) que informa sobre su utilidad.

AGUC15. ¿La información que se transmite a través de un código de colores, también está disponible si se obvia el color? Esta pauta sí se cumple. Scratch utiliza distintos colores para identificar la categoría de cada pieza (movimiento, apariencia, sonido, etc.). Por ejemplo, una pieza de color morado se corresponde con una acción que permite cambiar la apariencia de un objeto. Esto también se puede saber por la ubicación de dicha pieza dentro de una categoría, por el texto que incluye, o a través de su menú contextual (al pulsar con el botón derecho sobre una pieza, aparece un enlace a la ayuda de esa pieza concreta).

10.3. Análisis de usuario

En este tipo de estudios es habitual que como complemento al análisis heurístico se realice un análisis de usuario, basado en la aplicación de varias técnicas para recoger la experiencia de los usuarios que utilizan una herramienta o prototipo (Hassan Montero y Martín Fernández, 2004; Henry et al., 2001; Nielsen, 1994). La mayor parte de estas técnicas reproducen las metodologías clásicas de la ciencia social: cuestionarios, entrevistas, grupos, etnografía, etc. (Conde Melguizo, 2013a, 2013b).

La tercera fase del estudio se realizó con un análisis de usuario a través de un cuestionario.

10.3.1. Elaboración del cuestionario

Recordemos que esta investigación trata de averiguar si desde el punto de vista de la usabilidad Scratch es una herramienta que facilita o dificulta el aprendizaje de la programación. Por tanto, para elaborar el cuestionario fue necesario tener en cuenta por un lado los aspectos de usabilidad de la herramienta, y por otro el propósito con el que se utiliza, es decir, enseñar a programar.

Los indicadores para medir la usabilidad de la herramienta se obtuvieron del análisis heurístico y de las normas que se utilizaron en este como referencia. Los indicadores para medir si un estudiante aprende a programar con la herramienta se basan por un lado en los principios clásicos de la computación, y por otro en los criterios de evaluación establecidos en el decreto donde se regula la enseñanza de la programación en la Educación Primaria (Decreto Comunidad de Madrid, 2015). Ambos aspectos se desarrollan a continuación.

10.3.1.1. Principios de la computación que se han tenido en cuenta para el análisis de usuario

Como se ha mencionado en apartados anteriores, Scratch sigue el paradigma de programación imperativa (MIT, 2016c), en la que para plantear la solución a un problema se utiliza un algoritmo, siendo este un conjunto ordenado y finito de operaciones que llevan a dicha solución (Real Academia Española, 2014). Según se explica en capítulo 3.3.1, el teorema de Dijkstra (1968), demuestra que todo algoritmo puede describirse utilizando solamente tres tipos de instrucciones:

secuencia de acciones, sentencias condicionales, y bucles. En este mismo capítulo también se explica como Brennan y Resnick (2012), además de los elementos anteriores, conciben como elementos de computación básicos a **datos, operadores, paralelismo y eventos**.

Por este motivo, en la elaboración del cuestionario se incluyeron preguntas destinadas a evaluar la comprensión de los mencionados conceptos computacionales básicos, conceptos que, por otro lado, están presentes de forma directa o indirecta en los estándares de aprendizaje incluidos en el decreto que regula el desarrollo de la Enseñanza Primaria en la Comunidad de Madrid, como se muestra a continuación.

10.3.1.2. Criterios de evaluación recogidos en la ley de la Comunidad de Madrid utilizados para el análisis de usuario

En el Decreto 89/2014 (Decreto Comunidad de Madrid, 2015, p. 90) se describen los contenidos, criterios de evaluación y estándares de aprendizaje evaluables para toda la etapa, incluidos en la asignatura de libre configuración autonómica "Tecnología y recursos digitales para la mejora del aprendizaje". En el aspecto que a este texto le compete, para la elaboración del formulario se tienen en cuenta los puntos relativos al contenido descrito como "Fundamentos de programación. Creación de pequeños programas informáticos (Scratch)" (Decreto Comunidad de Madrid, 2015, p. 90).

Para este contenido, el decreto establece dos criterios de evaluación con sus correspondientes estándares de aprendizaje:

1. Conocer los fundamentos de la programación.
   1.1. Utiliza objetos, variables y listas para el desarrollo de sus programas.
   1.2. Interpreta los resultados esperados de pequeños bloques de programas.
   1.3. Evalúa los resultados del programa.
   1.4. Depura un programa para que el funcionamiento se adecue al previsto.

2. Programar juegos sencillos, animaciones e historias interactivas.
   2.1. Selecciona los elementos gráficos y los sonidos que formarán su programa.

2.2. Determina las acciones individuales que necesita el funcionamiento del programa.

2.3. Determina el orden y el sentido de los movimientos (arriba, abajo, derecha, izquierda) y los giros para conseguir el resultado deseado.

2.4. Determina las interacciones entre los diferentes elementos de su programa.

Analizando los estándares de aprendizaje que se acaban de enunciar, se puede comprobar cómo los conceptos incluidos en el capítulo 10.3.1.1 se mencionan de forma directa (por ejemplo, objetos y variables, en el punto 1.1), o de forma indirecta.

10.3.2. Preguntas del cuestionario y justificación de su presencia

Una vez definidos los principios que dan origen a cada pregunta, según se ha explicado en el capítulo 9.1, éstas se plantean para que las respuestas posibles sean:

- Respuestas abiertas.
- Respuestas con dos opciones (por ejemplo, sí o no).
- Respuestas con más de dos opciones. Las preguntas que se responden de esta forma están ordenadas en una variable ordinal, de tal manera que las dos primeras respuestas corresponden a la capacidad del participante para realizar la tarea de forma autónoma, y las dos últimas a que han necesitado ayuda o no han podido realizar la tarea.

El cuestionario definitivo tiene la siguiente forma:

Pregunta 1. ¿Qué es lo primero en que te fijas al abrir Scratch?

- *Posibles respuestas a la Pregunta 1:* Respuesta abierta.
- *Justificación de la Pregunta 1:* Esta pregunta se realiza para evaluar los aspectos que más llaman la atención a los participantes del estudio. Surge de las pautas del análisis heurístico relacionadas con la apariencia de la interfaz, y concretamente con NNGG1, NNGG8, HHS1.1, MIT7, NOKIA1, AGUC5.

Pregunta 2. Cuando has utilizado Scratch, ¿has podido realizar estas acciones?

- Elegir un fondo
- Elegir un personaje
- Editar el personaje (hacerlo más grande o más pequeño)
- Hacer que el personaje se mueva
- Hacer que se sumen puntos
- Añadir sonidos al juego

- *Posibles respuestas a la Pregunta 2:*
  - Sí, yo solo y a la primera
  - Sí, yo solo, pero después de probar un rato
  - Sí, pero me han tenido que ayudar
  - No

- *Justificación de la Pregunta 2:* Con esta pregunta se pretende evaluar la dificultad que encuentran los participantes para realizar las acciones básicas que se requieren para crear un videojuego sencillo, o una historia interactiva. Por un lado, se cuestiona sobre cómo añadir un nuevo elemento, y por otro, sobre cómo modificarlo. Hacer que el personaje se mueva y hacer que se sumen puntos son acciones que requieren programación, con lo que con estas preguntas evalúan la comprensión de varios conceptos computacionales (ver capítulo 10.3.1.1). Hacer que el personaje se mueva requiere del uso de eventos, hacer que se sumen puntos del uso de variables, y ambas del uso de secuencia de acciones. La Pregunta 2 en su conjunto también nos sirve para evaluar los estándares de aprendizaje determinados por la Comunidad de Madrid (ver capítulo 10.3.1.2). Por último, esta pregunta también surge del análisis heurístico, de las pautas relacionadas con el manejo de la interfaz, concretamente de NNGG3, NNGG9, HHS1.6, HHS1.8, HHS2.4, MIT2, MIT6, NOKIA4, NOKIA5, AGUC1, AGUC2 y AGUC3.

Pregunta 3. ¿Has podido guardar tu juego para poder seguir editándolo más tarde o en otra clase?

- *Posibles respuestas a la Pregunta 3:* Sí / No
- *Justificación de la Pregunta 3:* Esta pregunta pretende evaluar si el usuario es capaz de salir del sistema y puede guardar el trabajo realizado para recuperarlo posteriormente. Las pautas del análisis heurístico que dan origen a esta pregunta son NNGG4, HHS2.2, MIT5, NOKIA3, NOKIA10.

Pregunta 4. ¿Has utilizado elementos (dibujos, sonidos, etc.) creados por ti?

- *Posibles respuestas a la Pregunta 4:*
  - Sí, dibujos y sonidos
  - Sí, pero solo dibujos
  - Sí, pero solo sonidos
  - No, he utilizado solo dibujos y sonidos de Scratch

- *Justificación de la Pregunta 4:* Esta pregunta pretende evaluar la capacidad de los participantes para realizar tareas algo más complejas, y no solo utilizar lo que Scratch trae por defecto. Las pautas del análisis heurístico que dan origen a esta pregunta son NNGG7, HHS1.8 y MIT1.

Pregunta 5. ¿Has podido jugar a tu juego?

- *Posibles respuestas a la Pregunta 5:* Sí / No
- *Justificación de la Pregunta 5:* Esta pregunta busca evaluar si el participante ha podido realizar con Scratch la tarea encomendada. Pretende también ser un indicador para evaluar los estándares de aprendizaje determinados por la CM (ver capítulo 10.3.1.2). Por último, las pautas del análisis heurístico que dan origen a esta pregunta son HHS1.8 y NOKIA6.

Pregunta 6. ¿Te ha resultado divertido jugar a tu juego? (Esta pregunta solo se realiza si se ha respondido Sí a la Pregunta 5)

- *Posibles respuestas a la Pregunta 6:* Sí / No
- *Justificación de la Pregunta 6:* Esta pregunta da continuidad a la Pregunta 5, y pretende indagar en la forma en la que Scratch es una herramienta motivadora para sus usuarios. Las pautas del análisis heurístico que dan origen a esta pregunta son HHS1.3 y NOKIA6.

Pregunta 7. ¿Qué es lo que más te ha gustado? (Esta pregunta solo se realiza si se ha respondido Sí a la Pregunta 5)

- *Posibles respuestas a la Pregunta 7:* Respuesta abierta.
- *Justificación de la Pregunta 7:* Esta pregunta busca complementar a la *Pregunta 6*, y recabar más información al respecto. Igualmente surge de la pauta NOKIA6, y la HHS1.3.

Pregunta 8. ¿Y lo que menos te ha gustado de tu juego? (Esta pregunta solo se realiza si se ha respondido Sí a la Pregunta 5)

- *Posibles respuestas a la Pregunta 8:* Respuesta abierta.
- *Justificación de la Pregunta 8:* Al igual que la Pregunta 7, esta pregunta busca complementar a la *Pregunta 6*, y recabar más información al

respecto, en este caso haciendo hincapié en los aspectos desmotivadores que pueden surgir del uso de Scratch. De la misma manera que las dos preguntas anteriores, la Pregunta 8 surge de la pauta NOKIA6 y la HHS1.3.

Pregunta 9. Cuando vas a crear un programa, utilizas bloques que se combinan de distintas maneras: los puedes poner antes, o después, o dentro de otros bloques... ¿Cuánto has tardado en descubrir cómo se combinan los bloques?

- *Posibles respuestas a la Pregunta 9:*
  - A la primera, tras uno o dos intentos los he combinado solo
  - Después de probar un rato
  - Lo he entendido, pero alguna vez me ha tenido que volver a ayudar el profesor
  - El profesor me ha tenido que ayudar todo el rato

- *Justificación de la Pregunta 9:* Esta pregunta pretende evaluar lo intuitivo que es para los participantes el uso del sistema de programación por bloques que incorpora Scratch. Las pautas del análisis heurístico que dan origen a esta pregunta son NNGG6, MIT10 y AGUC1.

Pregunta 10. Dime si los siguientes bloques se pueden combinar:

- Por siempre – Apuntar hacia
- Repetir – ¿tecla presionada?
- No – ir a x: y:
- Si entonces – ¿tocando?

Figura 22. Imagen que en el formulario acompaña a la Pregunta 10.

Fuente: elaboración propia

- *Posibles respuestas a la Pregunta 10:* Sí / No
- *Justificación de la Pregunta 10:* Esta pregunta pretende confirmar la veracidad de las respuestas de la Pregunta 9, y verificar que efectivamente la lógica de programación de los bloques se entiende.

También en esta pregunta se busca evaluar la comprensión de algunos conceptos computacionales (ver capítulo 10.3.1.1). Las piezas “por siempre” y “repetir” sirven para llevar a cabo bucles. Las piezas “si entonces” y “no” se utilizan en la construcción de sentencias condicionales.

Pregunta 11. ¿En algún momento algo no funcionaba cómo querías?

- *Posibles respuestas a la Pregunta 11:*
  - Todo ha ido bien de principio a fin
  - He tenido que corregir alguna cosa

- *Justificación de la Pregunta 11:* Las pautas del análisis heurístico que dan origen a esta pregunta son NNGG5, NNGG7, HHS1.8 , MIT3 y NOKIA5.

Pregunta 12. ¿Qué es lo que has tenido que corregir? (Esta pregunta solo se realiza si se ha respondido que ha tenido que corregir alguna cosa en la Pregunta 11)

- *Posibles respuestas a la Pregunta 12:* Respuesta abierta.
- *Justificación de la Pregunta 12:* Esta pregunta trata de complementar a la Pregunta 11, y recabar información sobre los elementos de Scratch que los participantes tienen dificultades para manejar. Las pautas del análisis heurístico que dan origen a esta pregunta son NNGG5 y NNGG7.

Pregunta 13. Cuando algo no funcionaba como tú querías… (Esta pregunta solo se realiza si se ha respondido que ha tenido que corregir alguna cosa en la Pregunta 11).

- *Posibles respuestas a la Pregunta 13:*
  - Lo he resuelto yo solo
  - He utilizado la ayuda del programa
  - He pedido ayuda al profesor y hemos utilizado la ayuda del programa
  - He pedido ayuda al profesor y me lo ha resuelto

- *Justificación de la Pregunta 13:* Esta pregunta busca evaluar en qué medida el usuario de Scratch puede ser autónomo para aprender a utilizar el programa, y solucionar los problemas que surjan en su manejo. Las pautas del análisis heurístico que dan origen a esta pregunta son NNGG5, NNGG7, NNGG10, HHS2.7, HHS2.13, MIT4 y AGUC3.

Pregunta 14. ¿Al final has conseguido que el programa funcione como tú querías, o has dejado algún problema sin resolver? (Esta pregunta solo se

realiza si se ha respondido que ha tenido que corregir alguna cosa en la Pregunta 11).

- *Posibles respuestas a la Pregunta 14:*
  - Sí, funciona perfectamente
  - He resuelto más de la mitad de los problemas
  - He resuelto menos de la mitad de los problemas
  - No, el programa no funciona como yo quería

- *Justificación de la Pregunta 14:* Esta pregunta complementa a la Pregunta 13 y pretende determinar en qué medida se resuelven los problemas que surgen. Las pautas del análisis heurístico que dan origen a esta pregunta son NNGG5, NNGG7, HHS2.7, HHS2.13, MIT4, NOKIA6 y AGUC3.

Pregunta 15. ¿Había alguna palabra en Scratch que no entendías qué significaba?

- *Posibles respuestas a la Pregunta 15:*
  - Sí, hay palabras que no entendía qué significaban
  - He entendido todas las palabras a la primera

- *Justificación de la Pregunta 15:* Esta pregunta está enfocada a la evaluación del lenguaje que se utiliza en Scratch. Las pautas del análisis heurístico que dan origen a esta pregunta son NNGG2, MIT2, MIT8 y MIT9.

Pregunta 16. ¿Qué palabras no entendías? (Esta pregunta solo se realiza si se ha respondido que hay palabras que no se entienden en la Pregunta 15).

- *Posibles respuestas a la Pregunta 16:* Respuesta abierta.
- *Justificación de la Pregunta 16:* Esta pregunta busca complementar a la Pregunta 15, y trata de recabar información sobre las palabras que no se entienden, y el porqué de esta falta de comprensión.

Pregunta 17. Después de utilizar Scratch, ¿sabrías decirme cuáles de estas palabras están en el programa?

- Evento
- Variable
- Objeto
- Interfaz
- Operador
- Sensor
- Mensaje

- *Posibles respuestas a la Pregunta 17:* Sí / No
- *Justificación de la Pregunta 17:* Por un lado esta pregunta está enfocada a la evaluación del lenguaje utilizado en Scratch, y surge de las pautas del análisis heurístico NNGG2, MIT2, MIT8 y MIT9. Por otro lado, también se busca evaluar la comprensión de los conceptos computacionales básicos (ver capítulo 10.3.1.1).

Pregunta 18. ¿Has podido crear el juego que tenías pensado?

- *Posibles respuestas a la Pregunta 18:*
  - Si, ha quedado como yo quería
  - Se parece mucho a lo que quería hacer, pero faltarían cosas
  - Es diferente, porque no he podido hacer lo que quería
  - No he podido hacer un juego completo

- *Justificación de la Pregunta 18:* Esta pregunta pretende evaluar la consecución de los objetivos de la práctica que llevan a cabo los participantes cuando responden al cuestionario. Además se relaciona con uno de los estándares de aprendizaje indicados por la Comunidad de Madrid, que determina la capacidad que debe tener el estudiante para evaluar los resultados del programa realizado (ver capítulo 10.3.1.2). Por último, las pautas del análisis heurístico que dan lugar a esta pregunta son NNGG3, NNGG7, HHS1.3, HHS1.8 y NOKIA6.

Pregunta 19. ¿Qué hubieses añadido a tu juego para que fuera como lo habías pensado? (Esta pregunta solo se realiza si se ha respondido que el juego no ha quedado exactamente como se esperaba en la Pregunta 18, es decir, cualquier opción excepto la primera).

- *Posibles respuestas a la Pregunta 19:* Respuesta abierta.
- *Justificación de la Pregunta 19:* Esta pregunta complementa a la Pregunta 18, y busca profundizar en lo que los participantes echan de menos en sus proyectos, para averiguar si es algo que no se puede hacer con Scratch, o si siendo posible no han encontrado la forma de hacerlo.

Pregunta 20. ¿En qué curso estás?

- *Posibles respuestas a la Pregunta 20:*
  - 4º de primaria
  - 5º de primaria
  - 6º de primaria
  - Otro

- *Justificación de la Pregunta 20:* Esta es una pregunta de control realizada para confirmar que los participantes que responden al cuestionario cumplen los requisitos de la muestra estructural definida para el análisis.

Pregunta 21. Eres:

- *Posibles respuestas a la Pregunta 21:*
  - Mujer
  - Hombre

- *Justificación de la Pregunta 21:* Esta es una pregunta de control, para evaluar una posible influencia del género de los participantes en los resultados.

Pregunta 22. Escribe tu edad

- *Posibles respuestas a la Pregunta 22:* Respuesta abierta
- *Justificación de la Pregunta 22:* Esta es una pregunta de control realizada para confirmar que los participantes que responden al cuestionario cumplen los requisitos de la muestra estructural definida para el análisis.

Pregunta 23. ¿Cuántas clases has dado de Scratch antes de hacer esta encuesta? Si no lo sabes, puedes preguntar a tu profesora o profesor.

- *Posibles respuestas a la Pregunta 23:* Respuesta abierta
- *Justificación de la Pregunta 23:* Esta pregunta se realiza para valorar en qué medida el número de clases recibidas influye en las respuestas de los participantes.

Pregunta 24. Además de las clases, ¿has practicado en casa?

- *Posibles respuestas a la Pregunta 24:* Sí / No
- *Justificación de la Pregunta 24:* Esta pregunta se realiza para valorar en qué medida el haber practicado en casa influye en las respuestas de los participantes.

Pregunta 25. Antes de las clases de Scratch, ¿qué experiencia tenías en utilizar un ordenador?

- *Posibles respuestas a la Pregunta 25:*
  - Tengo un ordenador en casa y lo utilizo habitualmente (todos o casi todos los días)
  - Tengo un ordenador en casa, pero lo utilizo poco (una o dos veces por semana o menos)
  - Tengo un ordenador en casa, pero lo utilizo poco (una o dos veces por semana o menos)

- En mi casa no hay ordenador, pero utilizo el del colegio, o el de un familiar o amigo, de vez en cuando (una o dos veces por semana o menos)
- No suelo utilizar ningún ordenador

- *Justificación de la Pregunta 25:* Esta pregunta se realiza para valorar en qué medida el haber practicado en casa influye en las respuestas de los participantes.

Pregunta 26. Ahora que hemos terminado, te propongo un reto, ¿sabrías abrir un nuevo juego, elegir un personaje y hacer que se mueva, en unos minutos, y sin que nadie te ayude?

- *Posibles respuestas a la Pregunta 26:*
  - Si, sabría hacerlo solo
  - Si, pero a lo mejor me tendrían que ayudar
  - Me tendría que ayudar el profesor
  - No sabría hacerlo

- *Justificación de la Pregunta 26:* Esta pregunta se realiza a modo de conclusión, para poder evaluar el control que los usuarios creen que tienen del programa. Por último, esta pregunta también surge del análisis heurístico, de las pautas relacionadas con el manejo de la interfaz, y concretamente de NNGG3, NNGG9, HHS1.6, HHS1.8, HHS2.4, MIT2, MIT6, NOKIA4, NOKIA5, AGUC1, AGUC2 y AGUC3.

Pregunta 27. Por último, si pudieras cambiar el aspecto de Scratch, ¿qué modificarías? (Por ejemplo, añadir algo que eches de menos, mover a otro lugar algún botón, etc.)

- *Posibles respuestas a la Pregunta 27:* Respuesta abierta.
- *Justificación de la Pregunta 27:* Esta pregunta se realiza para recoger la opinión de los participantes sobre los aspectos de usabilidad de Scratch que perciben como mejorables.

10.3.3. Validación del formulario

El formulario, una vez elaborado, fue validado por expertos en Scratch, y por profesores de Educación Primaria. Esta validación tiene un doble objetivo, por un lado adecuar el lenguaje de las preguntas para que sea asequible a estudiantes de 5º y 6º de Primaria, y por otro lado, encontrar la manera de solventar la deseabilidad social, que al trabajar con niñas y niños se potencia (Lemos, 2006), y así conseguir

formular las preguntas de tal manera que los encuestados respondan lo que verdaderamente piensan al respecto, y no lo que creen que deben responder.

Así mismo, una primera versión del formulario fue presentada como comunicación en el 1er Congreso Internacional en Arte, Diseño y Desarrollo de Videojuegos en la Industria Creativa (Alonso Urbano y Conde Melguizo, 2015), y sometida al Comité Científico de dicho congreso.

10.3.4. Decisión Muestral

Al tratarse de un estudio de muestra estructural, los participantes del análisis de usuario debían reunir unas características comunes que delimitaran la frontera que los define (Ibáñez, 2003, p. 73). El estudio se ha realizado en la Comunidad de Madrid, con niñas y niños de 5º y 6º de primaria, en colegios públicos donde se ha decidido impartir la asignatura de "Tecnología y recursos digitales para la mejora del aprendizaje", o donde los profesores utilizan Scratch como herramienta transversal para el desarrollo de las competencias que marca el currículo. Es decir, todos los estudiantes encuestados comparten las siguientes características:

- Tienen una edad comprendida entre los 10 y los 12 años.
- Estudian en un colegio público de la Comunidad de Madrid.
- Han utilizado Scratch antes de realizar la encuesta.

La justificación de que estas fueran las características que definen a la muestra estructural se describen a continuación.

La edad de los participantes viene determinada por los estudios de Piaget (2013) sobre el desarrollo cognitivo, que se detallan en el capítulo 4.1.3. Es a partir de los 10 años donde los estudiantes desarrollan la capacidad de abstracción, imprescindible en el pensamiento computacional, para plantear la solución a un problema en forma de algoritmo, y en definitiva para poder programar según el paradigma imperativo que caracteriza a Scratch (MIT, 2016c). Siguiendo las teorías de Piaget, se determinó que para evaluar la usabilidad de Scratch en el aprendizaje de la programación dentro del contexto de la Escuela Primaria, los participantes debían tener una edad entre los 10 y los 12 años.

Que los colegios escogidos pertenecieran a la Comunidad de Madrid tiene su origen en la ley educativa de esta comunidad. De hecho, esta ley es la que origina la presente tesis. Como se ha visto en el capítulo 2.3, las comunidades autónomas tienen competencia legislativa para desarrollar el currículo a partir de la ley educativa estatal. La Comunidad de Madrid, en el Decreto 89/2014 (Decreto

Comunidad de Madrid, 2015, p. 90) designa a Scratch como la herramienta a utilizar en la asignatura de libre configuración autonómica “Tecnología y recursos digitales para la mejora del aprendizaje”. La mayoría de las comunidades autónomas del territorio español no incluyen en su currículo de Primaria la enseñanza de la programación, y exceptuando Madrid, ninguna menciona de forma literal a Scratch. Por este motivo, el análisis de usuario se hace en colegios de la Comunidad de Madrid.

El hecho de que fueran colegios públicos viene motivado por uno de los objetivos secundarios del estudio: poder analizar el factor de disponer o no de un ordenador en casa, para comprobar su influencia en el uso de Scratch. El informe de la OECD (2015, p. 198) refleja que en 2012, el 2,1% de los estudiantes españoles no disponían de ordenador en casa. Este mismo informe dictamina que los estudiantes que no disponen de un ordenador en sus hogares se encuadran dentro de un grupo de población con un nivel socioeconómico bajo (OECD, 2015, p. 135). También existen estudios en los se afirma que las familias con un poder adquisitivo más bajo suelen escoger la escuela pública para la escolarización de sus hijos (Fernández Enguita, 2008). En definitiva, a tenor de estos datos, para cumplir con uno de los objetivos del estudio había más probabilidades de encontrar participantes que no tuvieran ordenador en casa en colegios públicos, que en colegios privados o concertados.

10.3.5. Descripción del trabajo de campo

El estudio se llevó a cabo en los siguientes colegios:

- CEIP Miguel de Cervantes, en el municipio de Leganés.
- CEIP Río Bidasoa, en el municipio de Móstoles.
- CEIP Lope de Vega, en el municipio de Madrid, en el distrito de Carabanchel.

De los colegios que se prestaron a participar en el estudio, estos tres reunían las características definidas para la muestra estructural (ver capítulo 10.3.4).

Como se ha explicado en el capítulo 10.3.3, el cuestionario tuvo un proceso de validación previo al análisis de usuario. Como último paso de esta validación, antes de pasar el formulario a todos los colegios, se hizo una *prueba piloto* con 16 estudiantes del CEIP Lope de Vega a finales del curso 2015-16. Con ello se pretendía verificar que efectivamente las preguntas del cuestionario eran comprendidas por los participantes del estudio, y que se adaptaban a los objetivos del mismo. Las respuestas a este pre-test se analizaron con el profesor que llevó a cabo la encuesta, y fruto de este análisis se modificaron varias preguntas. A continuación se describe un ejemplo de estos cambios.

La Pregunta 10, originalmente estaba formulada de la siguiente manera: “Para saber si has comprendido cómo funcionan los bloques de Scratch, ponme un ejemplo de dos bloques que no pueden ponerse juntos”. En el pre-test se vio que la mayoría de los alumnos no contestaron a esta pregunta, pero no porque no manejaran la lógica de bloques de Scratch, sino porque no recordaban el nombre exacto de las piezas. Así lo corroboró el profesor que llevó a cabo la prueba piloto, que reconoció que incluso a él le costaría responder sin tener el programa delante. Esto vino a confirmar que la pregunta estaba mal planteada, con el agravante de que uno de los rasgos que definen la usabilidad de una herramienta es que no obligan al usuario a memorizar grandes cantidades de información (ver por ejemplo la pauta NNGG6). Así la pregunta se cambió para que mostrara una imagen con 4 ejemplos de parejas de piezas, y preguntara por cuáles se podían combinar y cuáles no, como se puede ver en la *Figura 23*.

Figura 23. Pregunta 10 según aparece en el formulario definitivo utilizado en el análisis de usuario.

Fuente: elaboración propia.

El estudio final con los tres colegios se llevó a cabo en los inicios del curso académico 2016-17. Antes de responder al cuestionario se planificó realizar un proyecto con Scratch. Este proyecto podía ser el marcado por el tutorial “Hazlo Volar” de La Hora del Código (MIT, 2016a), u otro de similares características (distintos objetos en escena con comportamiento asociado, personajes con movimiento, y variables donde acumular puntos en función de algún logro). Se escogió este proyecto por incluir en su realización todos los conceptos incluidos en el cuestionario.

En base a la experiencia del pre-test, para la realización del proyecto de Scratch se estimó una hora, y para responder al cuestionario 20 minutos. Por tanto, los profesores involucrados dedicaron dos sesiones de clase consecutivas a la realización de la actividad completa.

Por último, cabe señalar que los tutores legales de cada participante rellenaron un formulario de autorización (ver anexos), y se tomaron las medidas obligadas para el cumplimiento de la ley de protección de datos.

10.3.6. Resultados del análisis de usuario

Como se ha explicado en el capítulo 10.3.1, las preguntas son de tres tipos: preguntas abiertas, preguntas binarias (sí o no), y preguntas con respuestas múltiples. Después de estudiar las respuestas abiertas, se vio que estas tenían poco valor, y que no permitían extraer conclusiones válidas. Por este motivo se decidió no analizarlas en detalle.

Asimismo, se hicieron una serie de preguntas de control (edad, curso, género), habituales en este tipo de cuestionarios. Se comprobó que las respuestas a estas preguntas no contenían datos extremos (por ejemplo, que solo hubieran respondido hombres). Entre los estudiantes que participaron en la investigación, un 57,3% son hombres, y un 42,7% mujeres. De los 103 encuestados, 32 cursaban 5º de Primaria, y 72 cursaban 6º. Todos ellos tenían una edad entre 10 y 12 años.

Para el resto de respuestas, el análisis se ha hecho en términos de números absolutos. Es decir, se contabiliza cuántos participantes han elegido cada una de las opciones, y posteriormente se calculan los porcentajes en función de las respuestas que ha habido a esa pregunta, no en función del total de participantes, considerando fuera del estudio a efectos estadísticos a los que no han contestado.

Como se ha explicado en el capítulo 10.3.2, las preguntas que tienen más de dos respuestas están ordenadas en una variable ordinal de tal manera que las dos primeras respuestas corresponden a la capacidad del participante para realizar la tarea de forma autónoma, y las dos últimas a que han necesitado ayuda o no han podido realizar la tarea. Por ejemplo, en la Pregunta 3, cuando se pregunta sobre si se ha podido cambiar un fondo, la primera respuesta es “Sí, yo solo y a la primera”. Es decir, no solo ha podido cambiar el fondo, sino que además no ha tenido ninguna duda sobre cómo hacerlo. La segunda respuesta es “Sí yo solo, pero después de probar un rato”. Una respuesta así indica que la tarea presenta alguna dificultad, pero aun así el participante lo ha podido hacer sin ayuda. La tercera respuesta es “Sí, pero me han tenido que ayudar”. Esta respuesta está indicando que el participante no ha podido resolver de forma autónoma la tarea. La última respuesta de “No”, indica que la tarea no se ha podido completar, ni siquiera con ayuda. Desde este punto de vista es desde el que se van a interpretar todas las respuestas.

Se interpreta de esta manera porque estamos midiendo la usabilidad de Scratch. Si la herramienta es usable, el participante tiene que ser capaz de utilizarla con la mínima asistencia posible (Nielsen, 2003; Onrubia, 2005).

El análisis de cada respuesta se realiza de la siguiente manera. Primero se hace un análisis univariable en el que se estudian los datos agregados, sin cruzarlos, simplemente describiendo las respuestas. Posteriormente se realiza un análisis bivariable en el que se busca la posible relación de las respuestas obtenidas con las siguientes variables:

- Disponer o no de ordenador en casa
- Número de clases recibidas antes de realizar el cuestionario
- Si ha habido o no práctica autónoma con Scratch más allá de las clases recibidas

Para cada pregunta se hace una descripción de la realidad hacia la que apuntan las respuestas de los participantes, y si a la vista de los datos obtenidos, las pautas de usabilidad que originaron las preguntas indican su cumplimiento.

10.3.6.1. Cuando has utilizado Scratch, ¿has podido realizar estas acciones?

10.3.6.1.1. Análisis univariable: elegir un fondo

Como se puede ver en la Tabla 7 y en la *Figura 24* cerca del 90% de los encuestados pueden realizar esta tarea de forma autónoma, casi un 68% además lo hacen a la primera.

Tabla 7
Respuestas obtenidas a la pregunta de si se ha podido elegir un fondo

Opciones	Nº de respuestas	Porcentaje de respuestas
Sí, yo solo y a la primera	70	67,96%
Sí, yo solo, pero después de probar un rato	22	21,36%
Sí, pero me han tenido que ayudar	8	7,77%
No	3	2,91%
TOTAL	103	100%

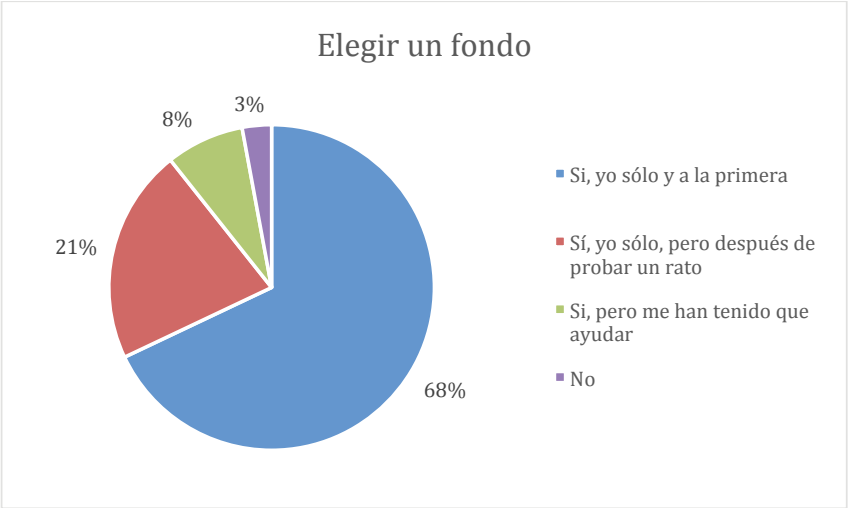


Figura 24. Gráfica de las respuestas obtenidas a la pregunta relativa a si se ha podido elegir un fondo.

Fuente: Elaboración propia.

10.3.6.1.2. Análisis bivariable: elegir un fondo y número de clases recibidas

Al buscar si existe relación con poder o no elegir un fondo, y el número de clases recibidas, por los resultados que se muestran en la Tabla 8 y en la *Figura 25* se puede ver que esta relación no existe, los porcentajes en cada respuesta de los que han recibido más clases que la media y los que han recibido menos son prácticamente iguales.

Tabla 8
Respuestas obtenidas a la pregunta de si se ha podido elegir un fondo, y su relación con el número de clases recibidas previamente.

	Sí, yo solo y a la primera		**Sí, yo solo, pero después de probar un rato**		**Si, pero me han tenido que ayudar**		**No**		**TOTAL**	
Clases recibidas	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**
Más clases que la media (Mayor)	29	60,42%	13	27,08%	4	8,33%	2	4,17%	48	100%
Menos clases que la media (Menor)	41	74,55%	9	16,36%	4	7,27%	1	1,82%	55	100%
TOTAL	70	67,96%	22	21,36%	8	7,77%	3	2,91%	103	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas, y % representa el porcentaje que ese número supone sobre el total de respuestas.

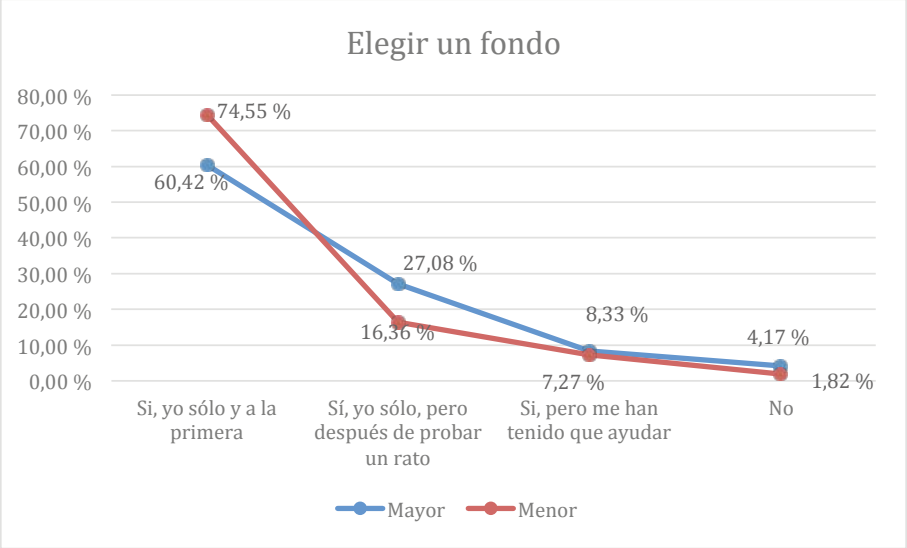


Figura 25. Gráfica de las respuestas obtenidas a la pregunta relativa a si se ha podido elegir un fondo, y su relación con el número de clases recibidas previamente.

Fuente: Elaboración propia.

10.3.6.1.3. Análisis bivariable: elegir un fondo y práctica autónoma

Como se puede ver en la en la Tabla 9 y en la *Figura 26* el porcentaje de respuestas por los alumnos que sí han tenido práctica autónoma y los que no la han tenido es prácticamente el mismo. No se aprecian diferencias significativas en este aspecto.

Tabla 9
Respuestas obtenidas a la pregunta de si se ha podido realizar la acción "elegir un fondo", y su relación con la práctica autónoma.

	Si, yo solo y a la primera		**Sí, yo solo, pero después de probar un rato**		**Si, pero me han tenido que ayudar**		**No**		**TOTAL**	
Práctica autónoma	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**
Sí	48	69,57%	14	20,29%	6	8,70%	1	1,45%	69	100%
No	21	63,64%	8	24,24%	2	6,06%	2	6,06%	33	100%
TOTAL	69	67,65%	22	21,57%	8	7,84%	3	2,94%	102	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla.

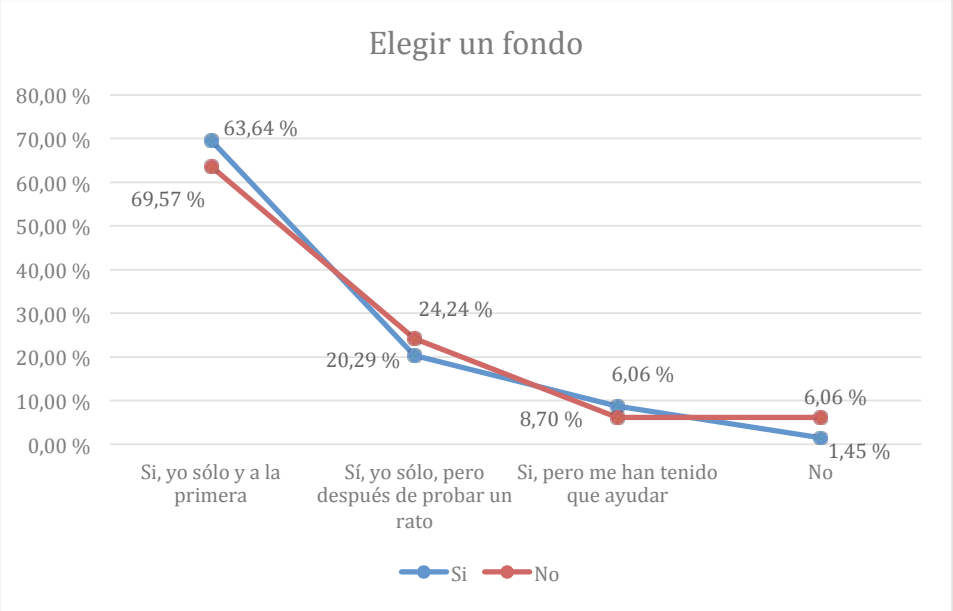


Figura 26. Gráfica de las respuestas obtenidas a la pregunta relativa a si se ha podido realizar la acción de elegir un fondo, y su relación con la práctica autónoma.

Fuente: Elaboración propia

10.3.6.1.4. Análisis bivariable: elegir un fondo y hábitos de uso del ordenador

En este análisis lo más destacado, por lo que se puede ver en la Tabla 10 y la *Figura 27*, es que la mayoría de los participantes que no pudieron incluir un fondo, no suelen utilizar un ordenador.

Tabla 10
Respuestas obtenidas a la pregunta de si se ha podido elegir un fondo, y su relación con los hábitos de uso del ordenador.

	Sí, yo solo y a la primera		**Sí, yo solo, pero después de probar un rato**		**Sí, pero me han tenido que ayudar**		**No**		**TOTAL**	
Hábitos de uso del ordenador	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**
Tengo un ordenador en casa y lo utilizo habitualmente (todos o casi todos los días)	27	77,14%	7	20%	1	2,86%	0	0%	35	100%
Tengo un ordenador en casa, pero lo utilizo poco (una o dos veces por semana o menos)	32	64%	10	20%	7	14%	1	2%	50	100%
En mi casa no hay ordenador, pero utilizo el del colegio, o el de un familiar o amigo, habitualmente (todos o casi todos los días)	4	66,67%	2	33,33%	0	0%	0	0%	6	100%
En mi casa no hay ordenador, pero utilizo el del colegio, o el de un familiar o amigo, de vez en cuando (una o dos veces por semana o menos)	2	66,67%	1	33,33%	0	0%	0	0%	3	100%
No suelo utilizar ningún ordenador	5	55,56%	2	22,22%	0	0%	2	22,22%	9	100%
TOTAL	70	67,96%	22	21,36%	8	7,77%	3	2,91%	103	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla

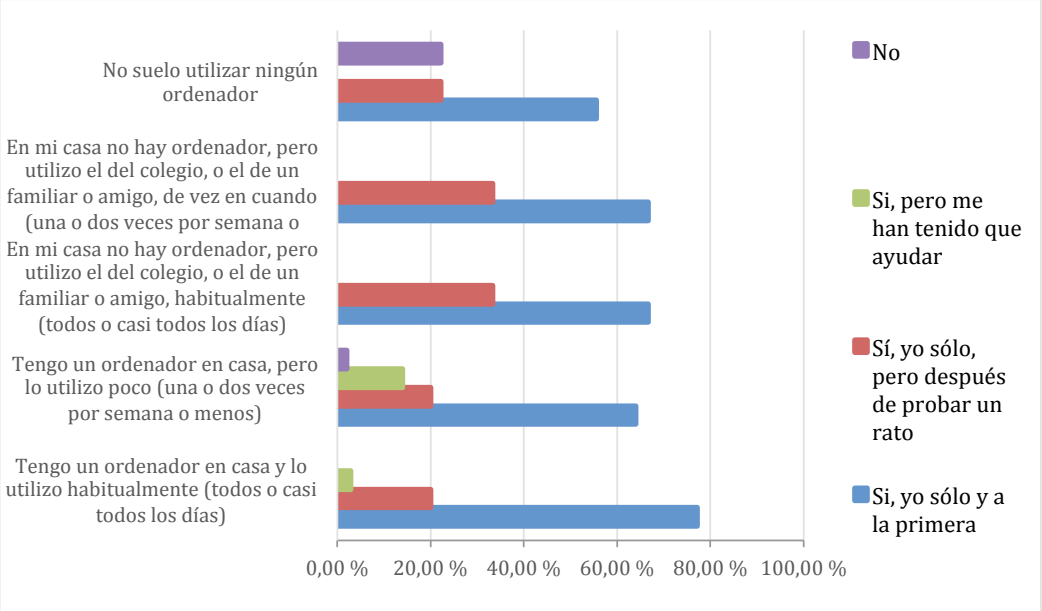


Figura 27. Gráfica de las respuestas obtenidas a la pregunta relativa a si se ha podido elegir un fondo, y su relación con los hábitos de uso del ordenador.

Fuente: Elaboración propia.

10.3.6.1.5. Análisis univariable: elegir un personaje

Según podemos apreciar en la Tabla 11 y en la *Figura 28*, los participantes en prácticamente su totalidad pudieron realizar la tarea sin problemas, solo el 11,65% han necesitado ayuda, y apenas una persona no pudo completar la tarea.

Tabla 11
Respuestas obtenidas a la pregunta de si se ha podido elegir un personaje.

Opciones	**Nº de respuestas**	**Porcentaje de respue**
Sí, yo solo y a la primera	74	71,84%
Sí, yo solo, pero después de probar un rato	16	15,53%
Sí, pero me han tenido que ayudar	12	11,65%
No	1	0,97%
TOTAL	103	100%

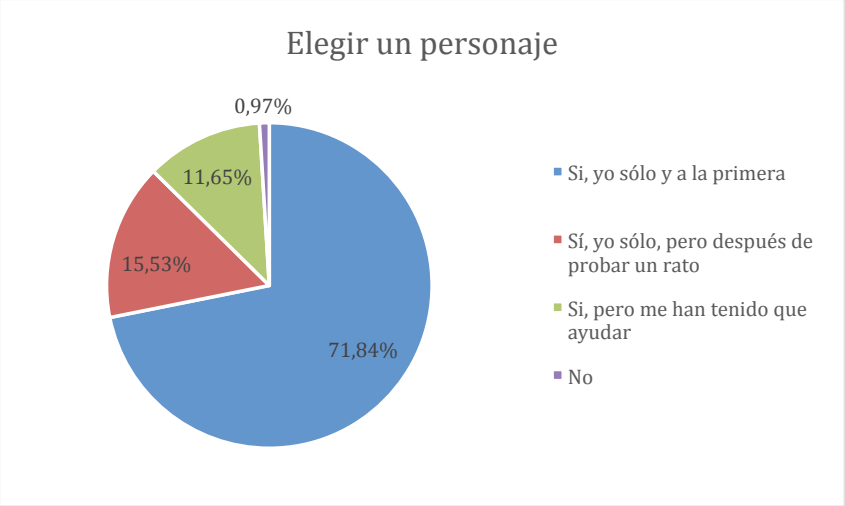


Figura 28. Gráfica de las respuestas obtenidas a la pregunta relativa a si se ha podido elegir un personaje.

Fuente: Elaboración propia.

10.3.6.1.6. Análisis bivariable: elegir un personaje y clases recibidas

Como se puede apreciar en la Tabla 12 y en la *Figura 29*, la relación entre poder elegir un personaje y el número de clases recibidas no es relevante, los porcentajes de respuestas entre los que han recibido más y menos clases que la media es muy similar.

Tabla 12
Respuestas obtenidas a la pregunta de si se ha podido elegir un personaje, y su relación con el número de clases recibidas previamente.

	Sí, yo solo y a la primera		**Sí, yo solo, pero después de probar un rato**		**Sí, pero me han tenido que ayudar**		**No**		**TOTAL**	
Clases recibidas	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**
Más clases que la media (Mayor)	31	64,58%	10	20,83%	6	12,50%	1	2,08%	48	100%
Menos clases que la media (Menor)	43	78,18%	6	10,91%	6	10,91%	0	0%	55	100%
TOTAL	74	71,84%	16	15,53%	12	11,65%	1	0,98%	103	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla.

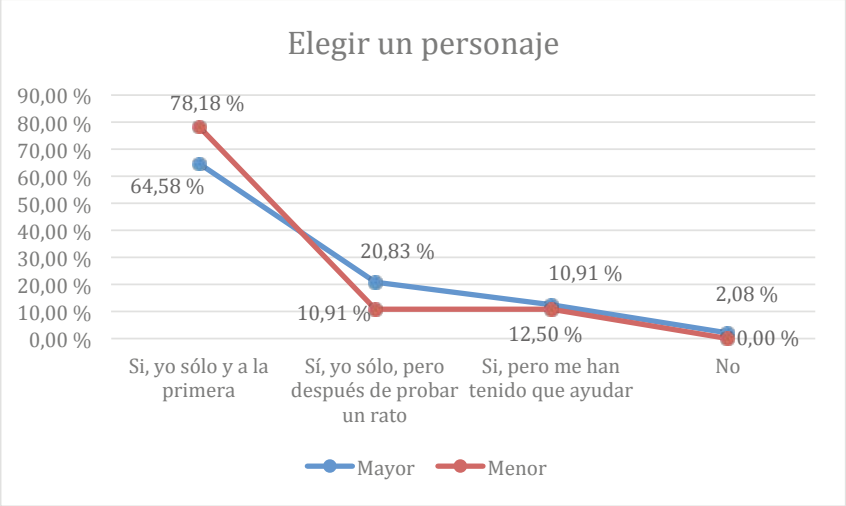


Figura 29. Gráfica de las respuestas obtenidas a la pregunta relativa a si se ha podido elegir un personaje, y su relación con el número de clases recibidas previamente.

Fuente: Elaboración propia.

10.3.6.1.7. Análisis bivariable: elegir un personaje y práctica autónoma

En la Tabla 13 y en la *Figura 30* se puede apreciar que la relación entre saber cómo elegir un personaje, y la práctica autónoma con Scratch es inapreciable.

Tabla 13
Respuestas obtenidas a la pregunta de si se ha podido realizar la acción "elegir un personaje", y su relación con la práctica autónoma.

	Sí, yo solo y a la primera		**Sí, yo solo, pero después de probar un rato**		**Si, pero me han tenido que ayudar**		**No**		**TOTAL**	
Práctica autónoma	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**
Sí	54	78,26%	9	13,04%	6	8,70%	0	0%	69	100%
No	19	57,58%	7	21,21%	6	18,18%	1	3,03%	33	100%
TOTAL	73	71,57%	16	15,69%	12	11,76%	1	0,98%	102	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla.

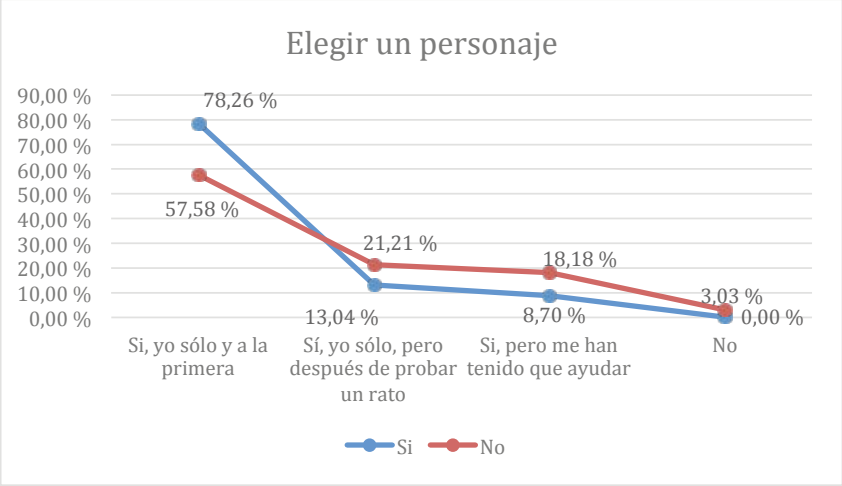


Figura 30. Gráfica de las respuestas obtenidas a la pregunta relativa a si se ha podido realizar la acción de "elegir un personaje", y su relación con la práctica autónoma.

Fuente: Elaboración propia.

10.3.6.1.8. Análisis bivariable: elegir un personaje y hábitos de uso del ordenador

Como se puede apreciar en la Tabla 14 y en la *Figura 31*, hay una mayor cantidad de participantes que han necesitado ayuda o que no han podido hacer la tarea entre los que menos utilizan el ordenador, aunque este dato tiene una importancia relativa, dado que la gran mayoría de los participantes realizaron esta tarea sin problemas.

Tabla 14
Respuestas obtenidas a la pregunta de si se ha podido elegir un personaje, y su relación con los hábitos de uso del ordenador.

	Sí, yo solo y a la primera		**Sí, yo solo, pero después de probar un rato**		**Sí, pero me han tenido que ayudar**		**No**		**TOTAL**	
Hábitos de uso del ordenador	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**
Tengo un ordenador en casa y lo utilizo habitualmente (todos o casi todos los días)	28	80%	6	17,14%	1	2,86%	0	0%	35	100%
Tengo un ordenador en casa, pero lo utilizo poco (una o dos veces por semana o menos)	37	74%	8	16%	5	10%	0	0%	50	100%

Hábitos de uso del ordenador	Sí, yo solo y a la primera		Sí, yo solo, pero después de probar un rato		Sí, pero me han tenido que ayudar		No		TOTAL	
	Nº	%	Nº	%	Nº	%	Nº	%	Nº	%
En mi casa no hay ordenador, pero utilizo el del colegio, o el de un familiar o amigo, habitualmente (todos o casi todos los días)	4	66,67%	0	0%	2	33,33%	0	0%	6	100%
En mi casa no hay ordenador, pero utilizo el del colegio, o el de un familiar o amigo, de vez en cuando (una o dos veces por semana o menos)	3	100%	0	0%	0	0%	0	0%	3	100%
No suelo utilizar ningún ordenador	2	22,22%	2	22,22%	4	44,44%	1	11,11%	9	100%
TOTAL	74	71,84%	16	15,53%	12	11,65%	1	0,97%	103	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla.

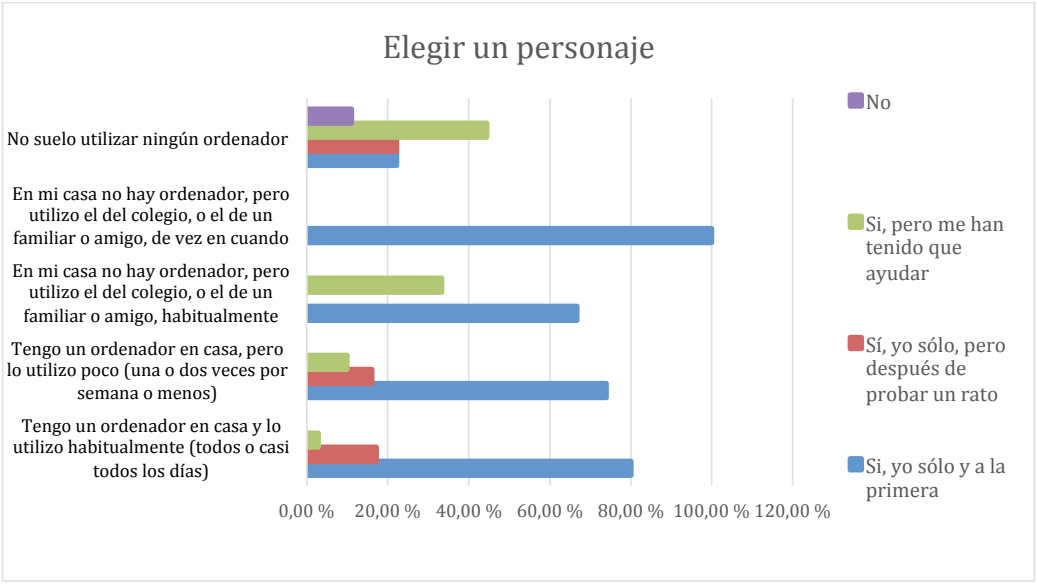


Figura 31. Gráfica de las respuestas obtenidas a la pregunta relativa a si se ha podido elegir un personaje, y su relación con los hábitos de uso del ordenador.

Fuente: Elaboración propia.

10.3.6.1.9. Análisis univariable: editar el personaje

Según se refleja en la Tabla 15 y en la *Figura 32*, siguen siendo mayoría los que pueden realizar la tarea, pero hay más personas que han necesitado ayuda que en tareas anteriores.

Tabla 15
Respuestas obtenidas a la pregunta de si se ha podido editar el personaje.

Opciones	**Nº de respuestas**	**Porcentajes de respuestas**
Sí, yo solo y a la primera	43	41,75%
Sí, yo solo, pero después de probar un rato	33	32,04%
Sí, pero me han tenido que ayudar	26	25,24%
No	1	0,97%
TOTAL	103	100%

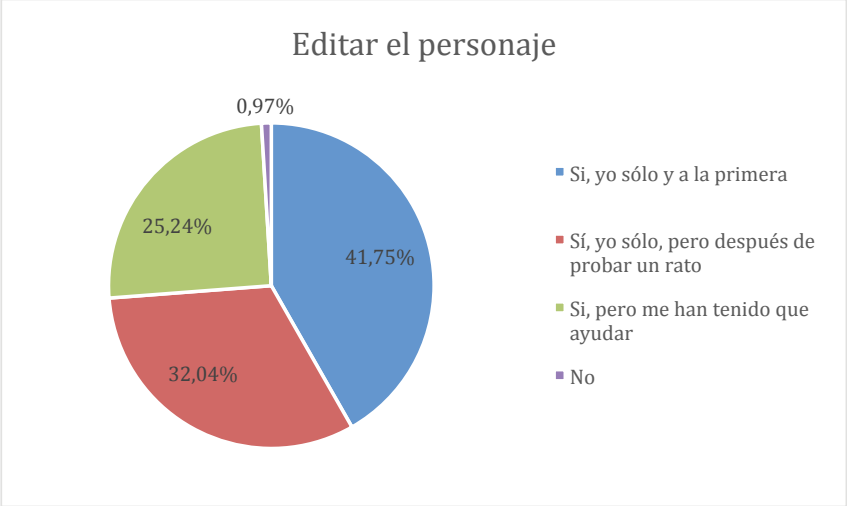


Figura 32. Gráfica de las respuestas obtenidas a la pregunta relativa a si se ha podido editar el personaje.

Fuente: Elaboración propia.

10.3.6.1.10. Análisis bivariable: editar el personaje y clases recibidas

Como se puede apreciar en la Tabla 16 y en la *Figura 33*, no se puede afirmar que exista relación existente entre que los participantes puedan editar el personaje, y el número de clases recibidas, los porcentajes de respuestas entre los que han recibido más y menos clases que la media es muy similar.

Tabla 16
Respuestas obtenidas a la pregunta de si se ha podido realizar la acción de [Editar el personaje (hacerlo más grande o más pequeño)], y su relación con el número de clases recibidas previamente.

	Sí, yo solo y a la primera		**Sí, yo solo, pero después de probar un rato**		**Sí, pero me han tenido que ayudar**		**No**		**TOTAL**	
Clases recibidas	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**
Más clases que la media (Mayor)	21	43,75%	13	27,08%	13	27,08%	1	2,08%	48	100%
Menos clases que la media (Menor)	22	40%	20	36,36%	13	23,64%	0	0%	55	100%
TOTAL	43	41,75%	33	32,04%	26	25,24%	1	0,97%	103	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla.

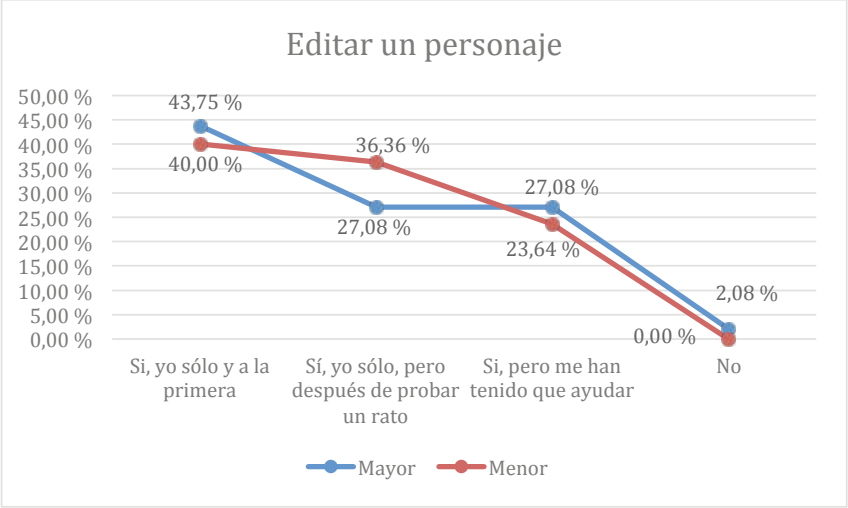


Figura 33. Gráfica de las respuestas obtenidas a la pregunta relativa a si se ha podido editar un personaje, y su relación con el número de clases recibidas previamente.

Fuente: Elaboración propia.

10.3.6.1.11. Análisis bivariable: editar el personaje y práctica autónoma

Como se puede apreciar en la Tabla 17 y en la Figura 34, la relación entre la capacidad para editar el personaje, y la práctica autónoma con Scratch que los participantes hayan podido tener fuera del aula no es significativa, aunque hay una

leve tendencia a que los participantes que menos usan el ordenador encuentren más problemas para realizar esta tarea.

Tabla 17
Respuestas obtenidas a la pregunta de si se ha podido realizar la acción "Editar el personaje (hacerlo más grande o más pequeño)", y su relación con *la práctica autónoma*.

Práctica autónoma	Sí, yo solo y a la primera		Sí, yo solo, pero después de probar un rato		Si, pero me han tenido que ayudar		No		TOTAL	
	Nº	%	Nº	%	Nº	%	Nº	%	Nº	%
Sí	27	39,13%	27	39,13%	15	21,74%	0	0%	69	100%
No	15	45,45%	6	18,18%	11	33,33%	1	3,03%	33	100%
TOTAL	42	41,18%	33	32,35%	26	25,49%	1	0,98%	102	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla.

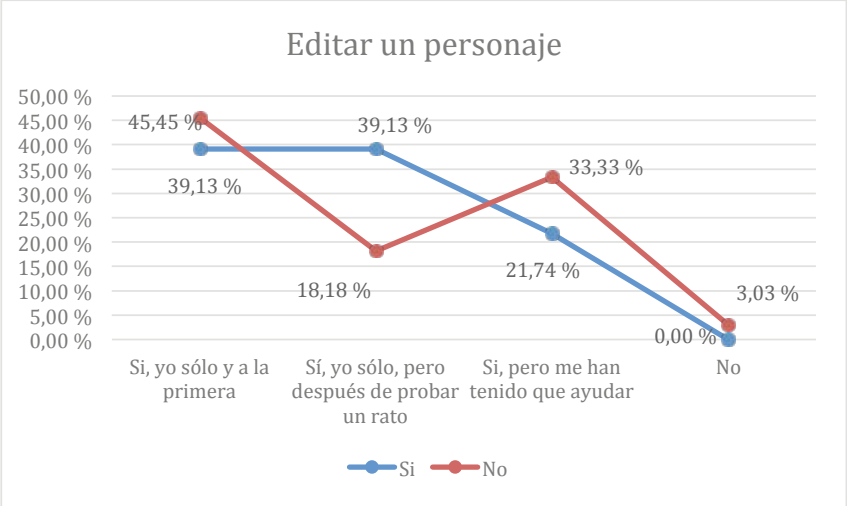


Figura 34. Gráfica de las respuestas obtenidas a la pregunta relativa a si se ha podido realizar la acción de "editar un personaje", y su relación con la práctica autónoma

Fuente: Elaboración propia.

10.3.6.1.12. Análisis bivariable: editar el personaje y hábitos de uso del ordenador

Como se ve reflejado en Tabla 18 y en la *Figura 35* no hay resultados concluyentes que puedan indicar una relación entre saber editar el personaje y los hábitos de uso del ordenador.

Tabla 18
Respuestas obtenidas a la pregunta de si se ha podido realizar la acción [Editar el personaje (hacerlo más grande o más pequeño)]", y su relación con los hábitos de uso del ordenador.

	Sí, yo solo y a la primera		**Sí, yo solo, pero después de probar un rato**		**Sí, pero me han tenido que ayudar**		**No**		**TOTAL**	
Hábitos de uso del ordenador	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**
Tengo un ordenador en casa y lo utilizo habitualmente (todos o casi todos los días)	16	45,71%	11	31,43%	8	22,86%	0	0%	35	100%
Tengo un ordenador en casa, pero lo utilizo poco (una o dos veces por semana o menos)	21	42%	17	34%	12	24%	0	0%	50	100%
En mi casa no hay ordenador, pero utilizo el del colegio, o el de un familiar o amigo, habitualmente (todos o casi todos los días)	2	33,33%	2	33,33%	2	33,33%	0	0%	6	100%
En mi casa no hay ordenador, pero utilizo el del colegio, o el de un familiar o amigo, de vez en cuando (una o dos veces por semana o menos)	0	0%	1	33,33%	1	33,33%	1	33,33%	3	100%
No suelo utilizar ningún ordenador	4	44,44%	2	22,22%	3	33,33%	0	0%	9	100%
TOTAL	43	41,75%	33	32,04%	26	25,24%	1	0,97%	103	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla.

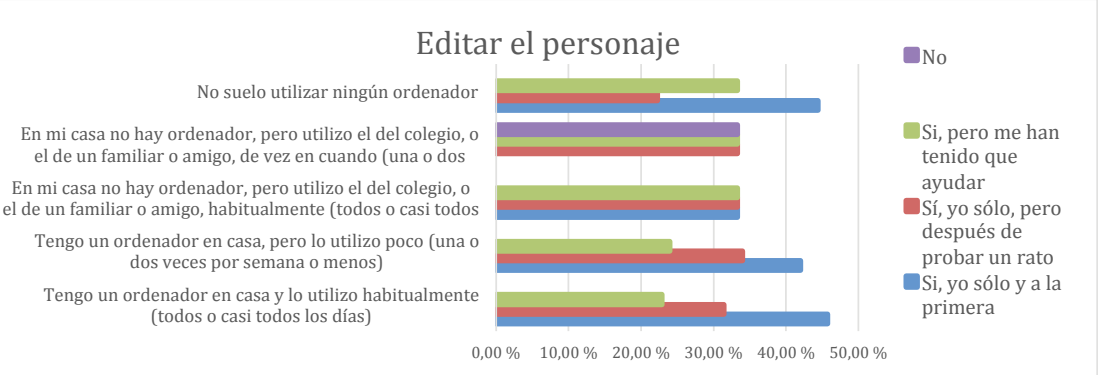


Figura 35. Gráfica de las respuestas obtenidas a la pregunta relativa a si se ha podido editar el personaje, y su relación con los hábitos de uso del ordenador.

Fuente: Elaboración propia.

10.3.6.1.13. Análisis univariable: mover el personaje

En la Tabla 19 y en la *Figura 36* se puede apreciar que la mayoría puede realizar la tarea de mover al personaje, aunque de nuevo el porcentaje de participantes que necesitan ayuda es mayor que en ocasiones anteriores, siendo aún minoría

Tabla 19
Respuestas obtenidas a la pregunta de si se ha podido mover el personaje.

Opciones	Nº de respuestas	Porcentaje de respuestas
Sí, yo solo y a la primera	37	35,92%
Sí, yo solo, pero después de probar un rato	27	26,21%
Sí, pero me han tenido que ayudar	38	36,89%
No	1	0,97%
TOTAL	103	100%

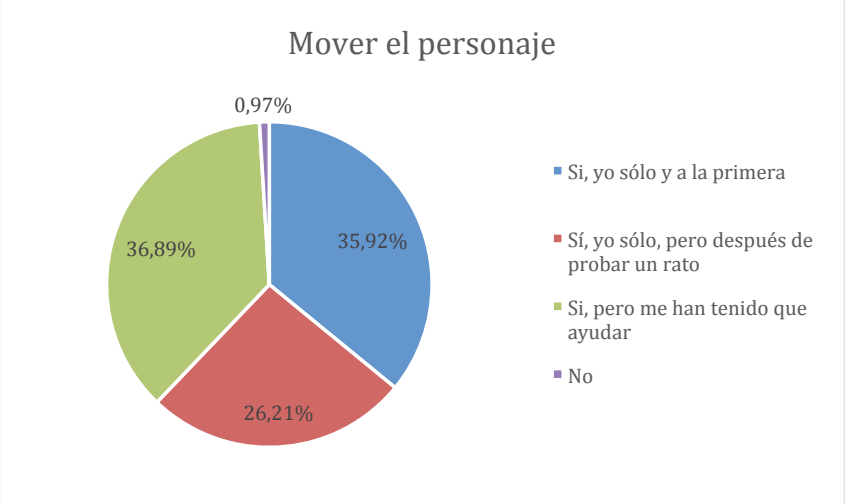


Figura 36 Gráfica de las respuestas obtenidas a la pregunta relativa a si se ha podido hacer que el personaje se mueva.

Fuente: Elaboración propia.

10.3.6.1.14. Análisis bivariable: mover el personaje y clases recibidas

En la Tabla 20 y en la *Figura 37* se puede ver que no es relevante el número de clases recibidas respecto a mover el personaje, siendo levemente mayor el porcentaje de participantes que han necesitado ayuda y han recibido menos clases que la media, que los que han recibido más, e igualmente han necesitado ayuda.

Tabla 20

Respuestas obtenidas a la pregunta de si se ha podido mover el personaje, y su relación con el número de clases recibidas previamente.

	Sí, yo solo y a la primera		**Sí, yo solo, pero después de probar un rato**		**Sí, pero me han tenido que ayudar**		**No**		**TOTAL**	
Clases recibidas	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**
Más clases que la media (Mayor)	19	39,58%	14	29,17%	14	29,17%	1	2,08%	48	100%
Menos clases que la media (Menor)	18	32,73%	13	23,64%	24	43,64%	0	0%	55	100%
TOTAL	37	35,92%	27	26,21%	38	36,89%	1	0,97%	103	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla.

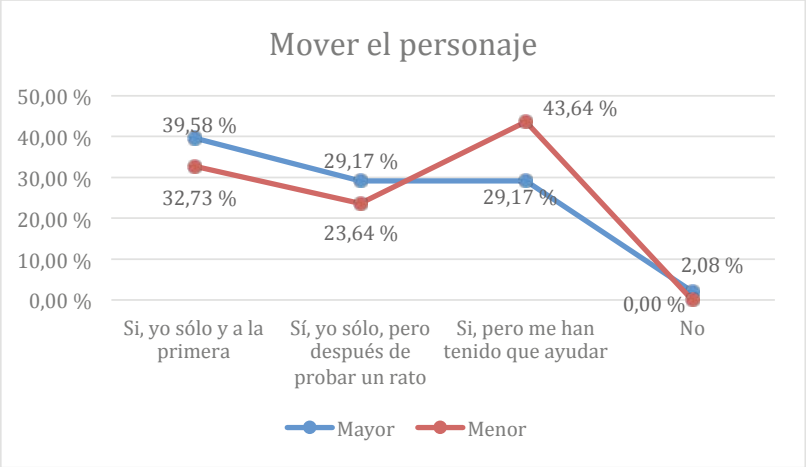


Figura 37. Gráfica de las respuestas obtenidas a la pregunta relativa a si se ha podido hacer que el personaje se mueva, y su relación con el número de clases recibidas previamente.

Fuente: Elaboración propia.

10.3.6.1.15. Análisis bivariable: mover el personaje y práctica autónoma

En la Tabla 21 y en la *Figura 38* se puede ver que la relación entre mover el personaje y la práctica autónoma no es significativa.

Tabla 21
Respuestas obtenidas a la pregunta de si se ha podido mover el personaje y su relación con la práctica autónoma.

	Sí, yo solo y a la primera		Sí, yo solo, pero después de probar un rato		Si, pero me han tenido que ayudar		No		TOTAL	
Práctica autónoma	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**
Sí	23	33,33%	20	28,99%	26	37,68%	0	0%	69	100%
No	13	39,39%	7	21,21%	12	36,36%	1	3,03%	33	100%
TOTAL	36	35,29%	27	26,47%	38	37,25%	1	0,98%	102	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla.

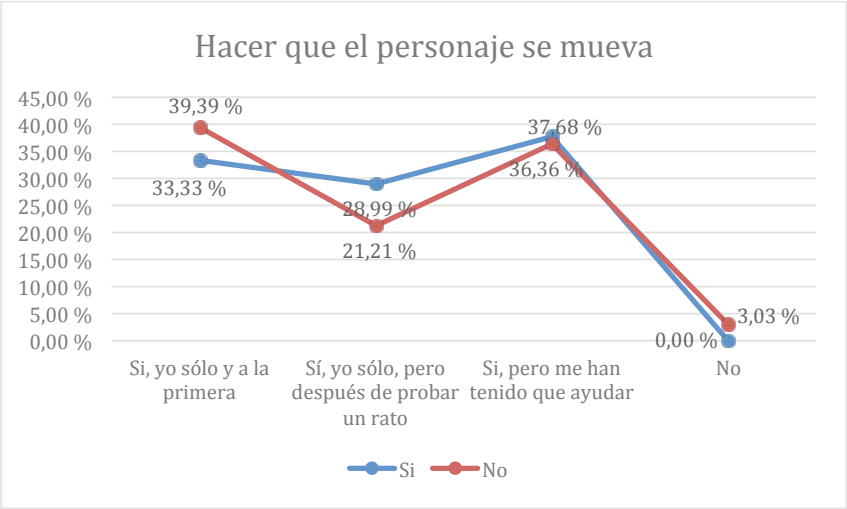


Figura 38. Gráfica de las respuestas obtenidas a la pregunta relativa a si se ha podido mover el personaje y su relación con la práctica autónoma.

Fuente: Elaboración propia.

10.3.6.1.16. Análisis bivariable: mover el personaje y hábitos de uso del ordenador

En la Tabla 22 y en la *Figura 39* se puede ver que todos los participantes que no pudieron hacer la tarea se encuadran en el grupo que no suelen utilizar el ordenador. Sin embargo, entre los que no usan un ordenador habitualmente, o lo usan poco, hay un alto porcentaje que pueden realizar la tarea a la primera, con lo que tampoco se pueden sacar conclusiones sobre una posible relación entre estas dos variables.

Tabla 22
Respuestas obtenidas a la pregunta de si se ha podido mover el personaje.

Hábitos de uso del ordenador	Sí, yo solo y a la primera		Sí, yo solo, pero después de probar un rato		Sí, pero me han tenido que ayudar		No		TOTAL	
	Nº	%	Nº	%	Nº	%	Nº	%	Nº	%
Tengo un ordenador en casa y lo utilizo habitualmente (todos o casi todos los días)	10	28,57%	12	34,29%	13	37,14%	0	0%	35	100%
Tengo un ordenador en casa, pero lo utilizo poco (una o dos veces por semana o menos)	18	36%	13	26%	19	38%	0	0%	50	100%
En mi casa no hay ordenador, pero utilizo el del colegio, o el de un familiar o amigo, habitualmente (todos o casi todos los días)	2	33,33%	2	33,33%	2	33,33%	0	0%	6	100%
En mi casa no hay ordenador, pero utilizo el del colegio, o el de un familiar o amigo, de vez en cuando (una o dos veces por semana o menos)	1	33,33%	0	0%	2	66,67%	0	0%	3	100%
No suelo utilizar ningún ordenador	6	66,67%	0	0%	2	22,22%	1	11,11%	9	100%
TOTAL	37	35,92%	27	26,21%	38	36,89%	1	0,97%	103	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla.

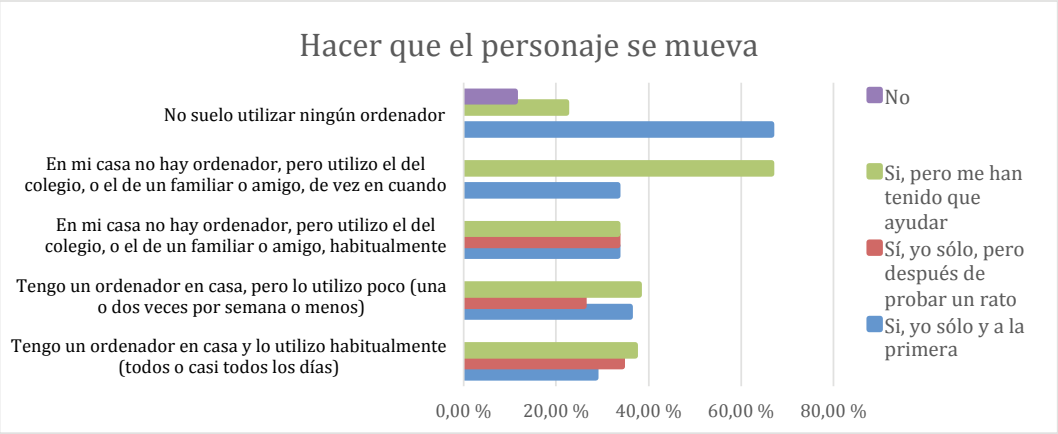


Figura 39. Gráfica de las respuestas obtenidas a la pregunta relativa a si ha podido mover el personaje, y su relación con los hábitos de uso del ordenador.

Fuente: Elaboración propia.

10.3.6.1.17. Análisis univariable: sumar puntos

Según se ve en la Tabla 23 y en la *Figura 40*, sumar puntos es una tarea que una mayoría no puede hacer, o que necesita ayuda para hacerla. Cabe mencionar que esta tarea requiere de programación, y de la utilización del concepto de variable en el programa realizado.

Tabla 23
Respuestas obtenidas a la pregunta de si se ha podido sumar puntos.

Opciones	Nº de respuestas	Porcentaje de respuestas
Si, yo solo y a la primera	6	5,83%
Sí, yo solo, pero después de probar un rato	22	21,36%
Si, pero me han tenido que ayudar	38	36,89%
No	37	35,92%
TOTAL	103	100%

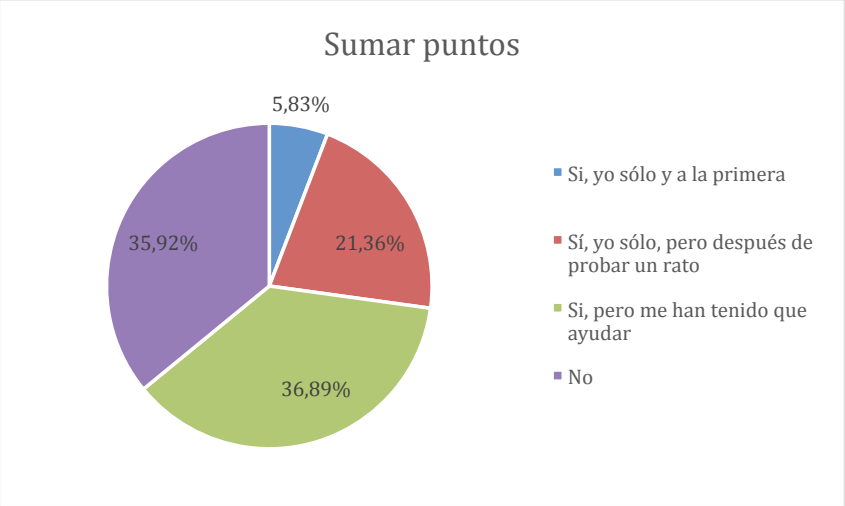


Figura 40. Gráfica de las respuestas obtenidas a la pregunta relativa a si se ha podido sumar puntos.

Fuente: Elaboración propia.

10.3.6.1.18. Análisis bivariable: sumar puntos y clases recibidas

En la Tabla 24 y en la *Figura 41* se aprecia que no hay una tendencia clara que indique la relación entre poder sumar puntos y el número de clases recibidas.

Tabla 24
Respuestas obtenidas a la pregunta de si se ha podido sumar puntos, y su relación con el número de clases recibidas previamente.

Clases recibidas	**Sí, yo solo y a la primera**		**Sí, yo solo, pero después de probar un rato**		**Sí, pero me han tenido que ayudar**		**No**		**TOTAL**	
	Nº	**%**	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**
Más clases que la media (Mayor)	4	8,33%	9	18,75%	13	27,08%	22	45,83%	48	100%
Menos clases que la media (Menor)	2	3,64%	13	23,64%	25	45,45%	15	27,27%	55	100%
TOTAL	6	5,83%	22	21,36%	38	36,89%	37	35,92%	103	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla.

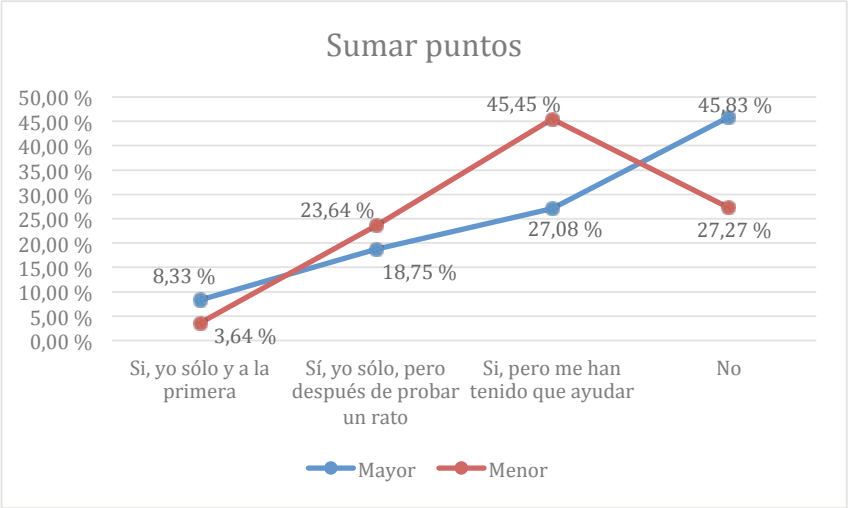


Figura 41. Gráfica de las respuestas obtenidas a la pregunta relativa a si se ha podido sumar puntos, y su relación con el número de clases recibidas previamente.

Fuente: Elaboración propia.

10.3.6.1.19. Análisis bivariable: sumar puntos y práctica autónoma

De la misma manera que en el análisis anterior, lo que refleja la Tabla 25 y la *Figura 42* es una dispersión de los datos, con lo que no se puede decir que haya una

tendencia clara que indique que puede haber relación entre la práctica autónoma, y saber cómo hacer que el programa sume puntos.

Tabla 25
Respuestas obtenidas a la pregunta de si se ha podido realizar la acción "sumar puntos", y su relación con la práctica autónoma.

	Sí, yo solo y a la primera		**Sí, yo solo, pero después de probar un rato**		**Sí, pero me han tenido que ayudar**		**No**		**TOTAL**	
Práctica autónoma	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**
Sí	3	4,35%	12	17,39%	30	43,48%	24	34,78%	69	100%
No	2	6,06%	10	30,30%	8	24,24%	13	39,39%	33	100%
TOTAL	5	4,90%	22	21,57%	38	37,25%	37	36,27%	102	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla.

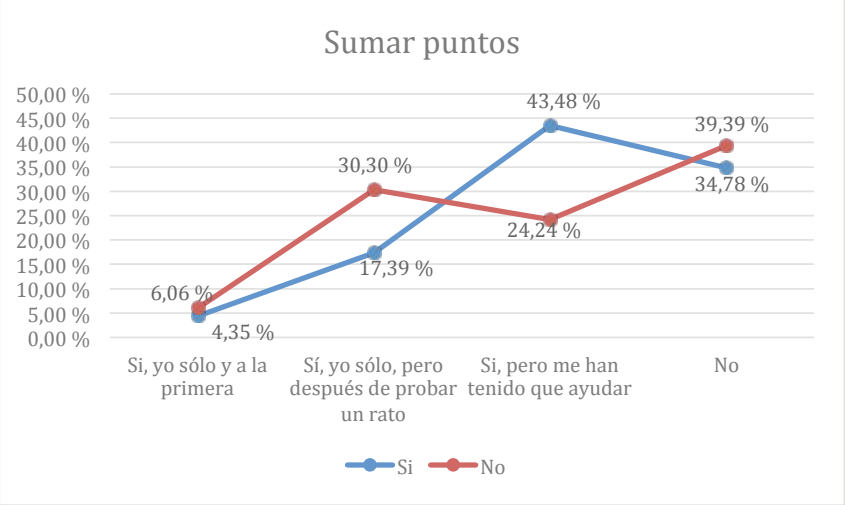


Figura 42. Gráfica de las respuestas obtenidas a la pregunta relativa a si se ha podido realizar la acción de "sumar puntos", y su relación con la práctica autónoma.

Fuente: Elaboración propia.

10.3.6.1.20. Análisis bivariable: sumar puntos y hábitos de uso del ordenador

En la Tabla 26 y en la *Figura 43* se ve que los que no pueden hacer la tarea también se encuentran entre los que tienen ordenador en casa y lo utilizan habitualmente, por lo que no se puede decir que exista relación entre estas dos variables.

Tabla 26
Respuestas obtenidas a la pregunta de si se ha podido sumar puntos, y su relación con los hábitos de uso del ordenador.

	Sí, yo solo y a la primera		**Sí, yo solo, pero después de probar un rato**		**Sí, pero me han tenido que ayudar**		**No**		**TOTAL**	
Hábitos de uso del ordenador	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**
Tengo un ordenador en casa y lo utilizo habitualmente (todos o casi todos los días)	3	8,57%	5	14,29%	17	48,57%	10	28,57%	35	100%
Tengo un ordenador en casa, pero lo utilizo poco (una o dos veces por semana o menos)	2	4%	12	24%	16	32%	20	40%	50	100%
En mi casa no hay ordenador, pero utilizo el del colegio, o el de un familiar o amigo, habitualmente (todos o casi todos los días)	0	0%	3	50%	3	50%	0	0%	6	100%
En mi casa no hay ordenador, pero utilizo el del colegio, o el de un familiar o amigo, de vez en cuando (una o dos veces por semana o menos)	0	0%	1	33,33%	1	33,33%	1	33,33%	3	100%
No suelo utilizar ningún ordenador	1	11,11 %	1	11,11%	1	11,11%	6	66,67%	9	100%
TOTAL	6	5,83%	22	21,36%	38	36,89%	37	35,92%	103	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla.

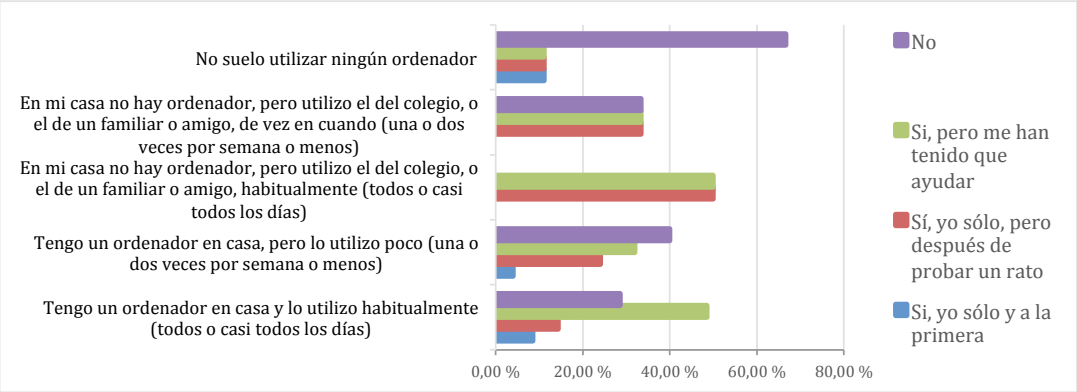


Figura 43. Gráfica de las respuestas obtenidas a la pregunta relativa a si se ha podido sumar puntos, y su relación con los hábitos de uso del ordenador.

Fuente: Elaboración propia.

10.3.6.1.21. Análisis univariable: incluir sonidos

En la Tabla 27 y en la *Figura 44* se ve que el porcentaje de respuestas está muy repartido. Siguen siendo mayoría los que pueden hacer la tarea, aunque sea con ayuda, que los que no pueden hacerla.

Tabla 27
Respuestas obtenidas a la pregunta de si se han incluido sonidos en el juego.

Opciones	**Nº de respuestas**	**Porcentaje de respuest**
Sí, yo solo y a la primera	32	31,07%
Sí, yo solo, pero después de probar un rato	24	23,30%
Sí, pero me han tenido que ayudar	26	25,24%
No	21	20,39%
TOTAL	103	100%

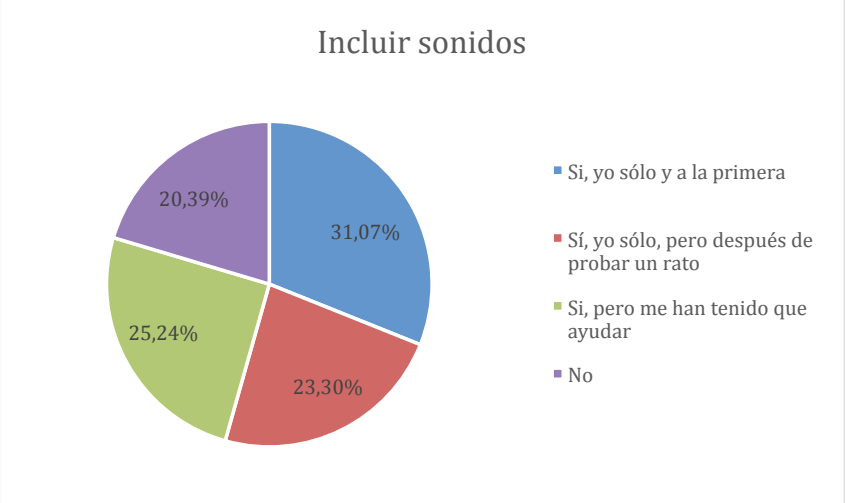


Figura 44. Gráfica de las respuestas obtenidas a la pregunta relativa a si se ha podido incluir sonidos.

Fuente: Elaboración propia.

10.3.6.1.22. Análisis bivariable: incluir sonidos y clases recibidas

En la Tabla 28 y en la *Figura 45* se ve que los que han necesitado ayuda, o no han podido realizar la tarea se encuentran mayoritariamente entre los que han recibido menos clases que la media.

Tabla 28
Respuestas obtenidas a la pregunta de si se han incluido sonidos en el juego, y su relación con el número de clases recibidas previamente.

Clases recibidas	**Sí, yo solo y a la primera**		**Sí, yo solo, pero después de probar un rato**		**Sí, pero me han tenido que ayudar**		**No**		**TOTAL**	
	Nº	**%**	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**
Más clases que la media (Mayor)	21	43,75%	14	29,17%	6	12,50%	7	14,58%	48	100%
Menos clases que la media (Menor)	11	20%	10	18,18%	20	36,36%	14	25,45%	55	100%
TOTAL	32	31,07%	24	23,30%	26	25,24%	21	20,39%	103	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla.

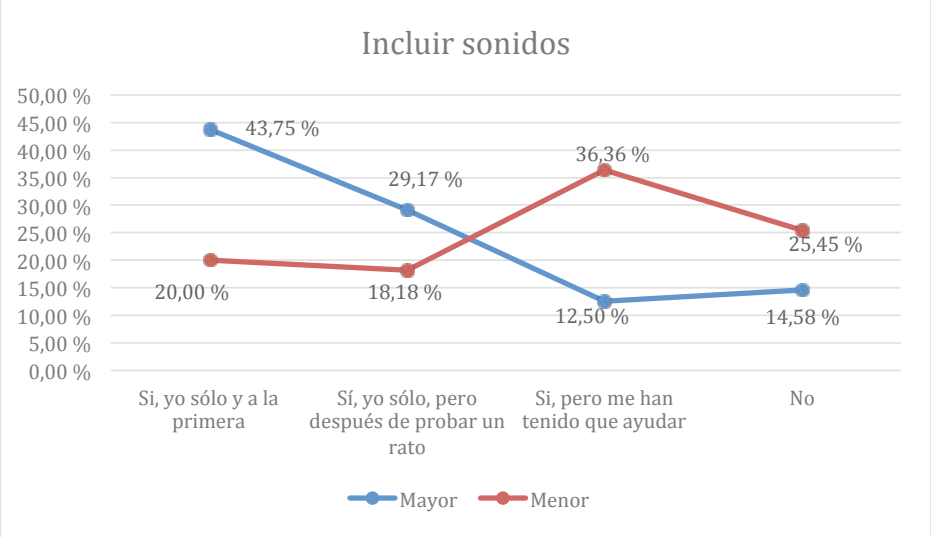


Figura 45. Gráfica de las respuestas obtenidas a la pregunta relativa a si se han incluido sonidos en el juego, y su relación con el número de clases recibidas previamente.

Fuente: Elaboración propia.

10.3.6.1.23. Análisis bivariable: incluir sonidos y práctica autónoma

En la Tabla 29 y en la *Figura 46* se aprecia cierta tendencia que señala que los participantes que han necesitado ayuda son los que no han tenido práctica autónoma.

Tabla 29
Respuestas obtenidas a la pregunta de si se han incluido sonidos en el juego, y su relación con la práctica autónoma.

	Sí, yo solo y a la primera		Sí, yo solo, pero después de probar un rato		Sí, pero me han tenido que ayudar		No		TOTAL	
Práctica autónoma	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**
Sí	22	31,88%	19	27,54%	13	18,84%	15	21,74%	69	100%
No	9	27,27%	5	15,15%	13	39,39%	6	18,18%	33	100%
TOTAL	31	30,39%	24	23,53%	26	25,49%	21	20,59%	102	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla.

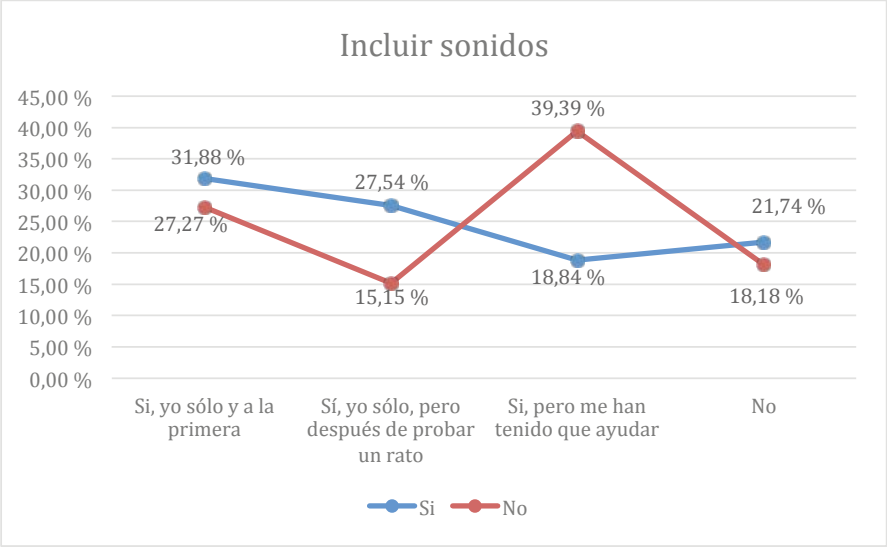


Figura 46. Gráfica de las respuestas obtenidas a la pregunta relativa a si se han incluido sonidos en el juego, y su relación con la práctica autónoma.

Fuente: Elaboración propia.

10.3.6.1.24. Análisis bivariable: incluir sonidos y hábitos de uso del ordenador

En la Tabla 30 y en la *Figura 47* no se aprecian datos relevantes que apunten a que pueda haber relación entre los hábitos de uso del ordenador, y saber incluir sonidos en el juego.

Tabla 30
Respuestas obtenidas a la pregunta de si han incluido sonidos en el juego, y su relación con los hábitos de uso del ordenador.

	Sí, yo solo y a la primera		Sí, yo solo, pero después de probar un rato		Sí, pero me han tenido que ayudar		No		TOTAL	
Hábitos de uso del ordenador	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**
Tengo un ordenador en casa y lo utilizo habitualmente (todos o casi todos los días)	13	37,14%	6	17,14%	10	28,57%	6	17,14%	35	100%
Tengo un ordenador en casa, pero lo utilizo poco (una o dos veces por semana o menos)	12	24%	13	26%	14	28%	11	22%	50	100%
En mi casa no hay ordenador, pero utilizo el del colegio, o el de un familiar o amigo, habitualmente (todos o casi todos los días)	1	16,67%	3	50%	1	16,67%	1	16,67%	6	100%
En mi casa no hay ordenador, pero utilizo el del colegio, o el de un familiar o amigo, de vez en cuando (una o dos veces por semana o menos)	2	66,67%	0	0%	0	0%	1	33,33%	3	100%
No suelo utilizar ningún ordenador	4	44,44%	2	22,22%	1	11,11%	2	22,22%	9	100%
TOTAL	32	31,07%	24	23,30%	26	25,24%	21	20,39%	103	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla.

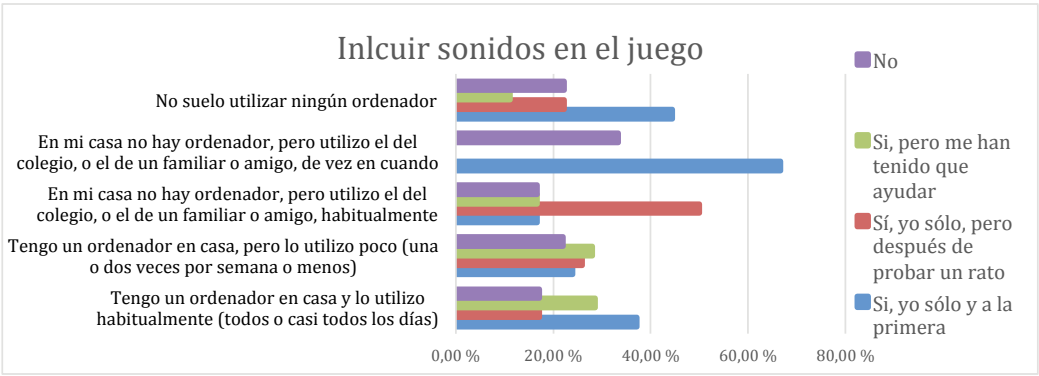


Figura 47. Gráfica de las respuestas obtenidas a la pregunta relativa a si se han incluido sonidos en el juego, y su relación con los hábitos de uso del ordenador.

Fuente: Elaboración propia.

10.3.6.1.25. Conclusiones globales a la Pregunta 2

La Pregunta 2 cuestiona sobre distintas acciones dentro del programa. Lo primero que se observa es que en la mayoría de las acciones los participantes pueden hacer las tareas de forma autónoma, más del 50% en todos los casos eligen las dos primeras respuestas. También se observa que según van aumentando la dificultad (editar un personaje, hacerlo mover, etc.) necesitan probar un rato, o incluso la ayuda del profesor. Cuando se trata de mover el personaje la opción de “me han tenido que ayudar” tiene un número mayoritario de respuestas. Mover al personaje es la única de las acciones de la Pregunta 2, donde se requiere programación.

La Pregunta 2 está relacionada con las pautas NNGG3, NNGG9, que tienen que ver con el control del usuario sobre el programa, y su capacidad para resolver los problemas que pueden surgir. Según los datos recogidos, la usabilidad es adecuada cuando se trata de añadir nuevos elementos al programa. Las dificultades aparecen cuando se requiere de programación para hacer una tarea, como es el caso de mover un personaje o sumar puntos (en este último, se precisa el uso de variables).

Las pautas HHS1.6, HHS1.8, MIT2, MIT6, NOKIA4, NOKIA5, AGUC1, AGUC2 y AGUC3, relacionadas con la Pregunta 2, evalúan que el diseño de la herramienta se adecúa al propósito de esta, y la hace funcional. Según las respuestas parece que efectivamente esto es así, dado que apenas el 1% de los encuestados no pueden realizar las tareas encomendadas. En cualquier caso, no hay evidencias categóricas, y el cumplimiento de estas mismas pautas se evalúa también en otras preguntas.

Para medir el cumplimiento de la pauta HHS2.4 a través de las respuestas a la Pregunta 2, que evalúa que el usuario, para hacer determinadas tareas, no tiene que memorizar su proceso, nos podemos fijar en el análisis bivariable de las distintas tareas, y el número de clases recibidas. En estas gráficas la línea azul representa las respuestas de los participantes que han recibido un mayor número de clases que la media, y la línea naranja las respuestas de los que han recibido menos clases que la media. Como se pueden observar en todas estas gráficas, ambas líneas están prácticamente superpuestas, lo que indica que en las respuestas a la Pregunta 2 no ha influido el número de clases recibidas. Es decir, para poder hacer estas tareas no se han tenido que memorizar procesos o recordar lo que se enseñó en clases pasadas, con lo que se cumpliría la pauta HHS2.4. Sólo hay una diferencia algo más acentuada en el porcentaje de respuestas de las personas que han tenido que pedir ayuda para mover el personaje, la mayoría de

ellas en el grupo de participantes que menos clases han recibido, pero sin llegar a ser una diferencia determinante.

10.3.6.2. ¿Has podido guardar tu juego para seguir editándolo más tarde o en otra clase?

10.3.6.2.1. Análisis univariable: guardar el juego

En la Tabla 31 y en la *Figura 48* se ve que la mayoría de los participantes pueden guardar el juego sin problemas, con lo que todo apunta a que la manera de hacerlo está claramente identificada en Scratch.

Tabla 31
Respuestas obtenidas a la pregunta si se ha podido guardar el juego para seguir editándolo más tarde o en otra clase.

Opciones	Nº de respuestas	Porcentajes de respuestas
Sí	98	95,15%
No	5	4,85%
Total	103	100%

Figura 48. Gráfica de las respuestas obtenidas a la pregunta relativa a si se ha podido guardar el juego para seguir editándolo más tarde o en otra clase.

Fuente: Elaboración propia

10.3.6.2.2. Análisis bivariable: guardar el juego y clases recibidas

En la Tabla 32 y en la *Figura 49* no se ven diferencias significativas que apunten a la relación de estas dos variables.

Tabla 32
Respuestas obtenidas a la pregunta de si se ha podido guardar el juego para seguir editándolo más tarde o en otra clase, y su relación con el número de clases recibidas previamente.

Clases recibidas	**Sí**		**No**		**TOTAL**	
	Nº	**%**	**Nº**	**%**	**Nº**	**%**
Más clases que la media (Mayor)	45	93,75%	3	6,25%	48	100%
Menos clases que la media (Menor)	53	96,36%	2	3,64%	55	100%
TOTAL	98	95,15%	5	4,85%	103	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla.

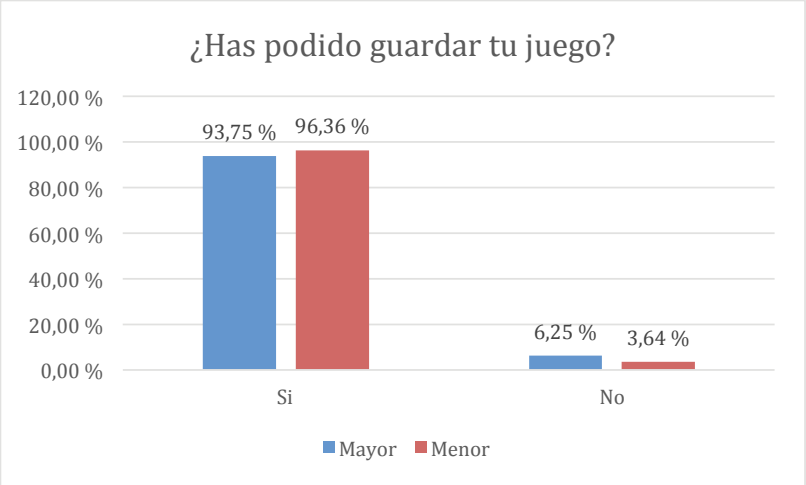


Figura 49 Gráfica de las respuestas obtenidas a la pregunta relativa a si se ha podido guardar el juego para seguir editándolo más tarde o en otra clase, y su relación con el número de clases recibidas previamente.

Fuente: Elaboración propia.

10.3.6.2.3. Análisis bivariable: guardar el juego y práctica autónoma

En la Tabla 33 y en la *Figura 50* se ve que los pocos participantes que no pudieron guardar el juego se encuadran entre los que menos práctica autónoma han tenido.

Tabla 33
Respuestas obtenidas a la pregunta de si se ha podido guardar el juego para seguir editándolo más tarde en otra clase, y su relación con la práctica autónoma.

	Sí		No		TOTAL	
Práctica autónoma	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**
Sí	68	98,55%	1	1,45%	69	100%
No	29	87,88%	4	12,12%	33	100%
TOTAL	97	95,10%	5	4,90%	102	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla.

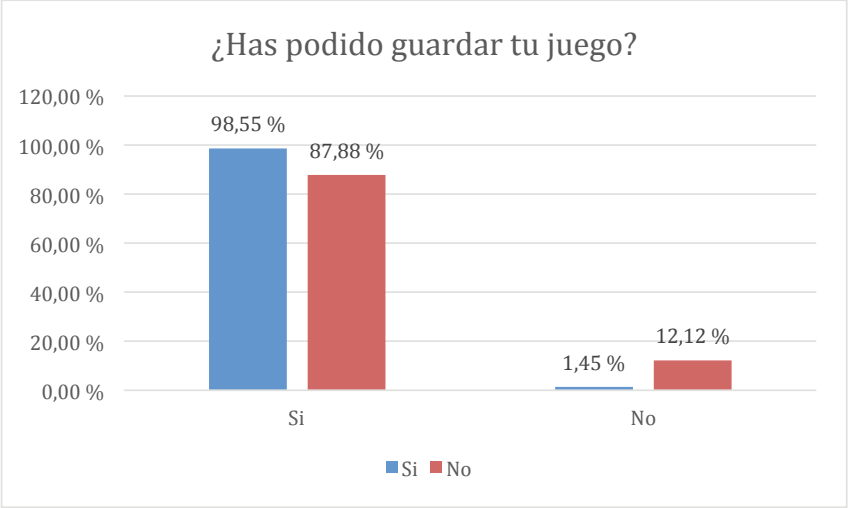


Figura 50. Gráfica de las respuestas obtenidas a la pregunta relativa a si se ha podido guardar el juego para seguir editándolo más tarde en otra clase, y su relación con la práctica autónoma.

Fuente: Elaboración propia.

10.3.6.2.4. Análisis bivariable: guardar el juego y hábitos de uso del ordenador

En la Tabla 34 y en la *Figura 51* se ve que todos los que utilizan el ordenador habitualmente han podido realizar esta tarea sin problemas. En cualquier caso, el número de personas que no han podido hacer la tarea es pequeño, con lo que no se pueden extraer conclusiones categóricas.

Tabla 34
Respuestas obtenidas a la pregunta de si se ha podido guardar el juego para seguir editándolo más tarde o en otra clase, y su relación con los hábitos de uso del ordenador.

	Sí		**No**		**TOTAL**	
Hábitos de uso del ordenador	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**
Tengo un ordenador en casa y lo utilizo habitualmente (todos o casi todos los días)	35	100%	0	0%	35	100%
Tengo un ordenador en casa, pero lo utilizo poco (una o dos veces por semana o menos)	47	94%	3	6%	50	100%
En mi casa no hay ordenador, pero utilizo el del colegio, o el de un familiar o amigo, habitualmente (todos o casi todos los días)	5	83,33%	1	16,67%	6	100%
En mi casa no hay ordenador, pero utilizo el del colegio, o el de un familiar o amigo, de vez en cuando (una o dos veces por semana o menos)	3	100%	0	0%	3	100%
No suelo utilizar ningún ordenador	8	88,89%	1	11,11%	9	100%
TOTAL	98	95,15%	5	4,85%	103	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla.

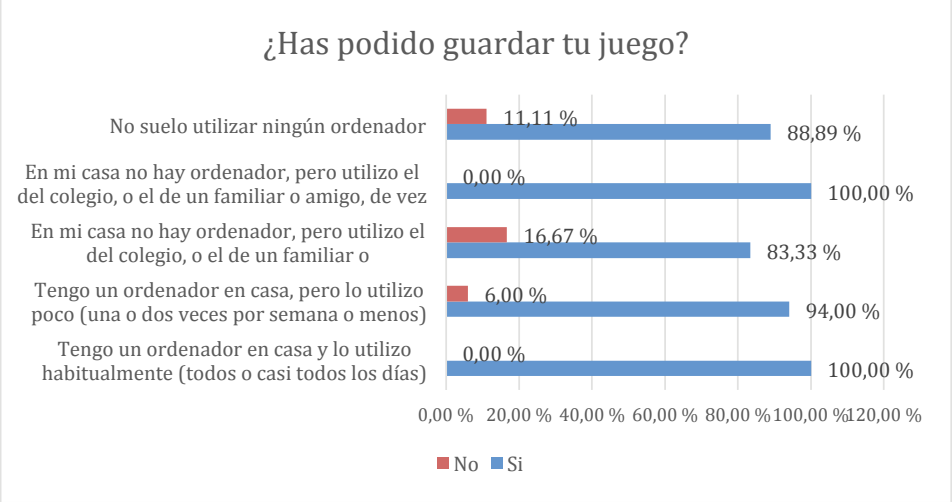


Figura 51. Gráfica de las respuestas obtenidas a la pregunta relativa a si se ha podido guardar el juego para seguir editándolo más tarde o en otra clase, y su relación con los hábitos de uso del ordenador.

Fuente: Elaboración propia.

10.3.6.2.5. Conclusiones a la pregunta "¿Has podido guardar tu juego para poder seguir editándolo más tarde o en otra clase?"

La Pregunta 3 está relacionada con las pautas NNGG4, HHS2.2, MIT5, NOKIA3 y NOKIA10, que evalúan la posibilidad de guardar el proyecto para poder recuperarlo posteriormente. A tenor de los resultados (solo un 4,85% no pueden hacerlo), se confirma que guardar el proyecto no genera problemas de usabilidad. Esto tiene que ver con lo intuitivo de la acción. En usabilidad algo es intuitivo cuando no se necesitan explicaciones para poder hacerlo (Nielsen, 2000). No significa que sea algo que exija un movimiento natural, más bien suele estar relacionado con las tecnologías que se han utilizado previamente (Aguilar Gil y Conde Melguizo, 2017). Si un proceso se asemeja a otro que ya hemos realizado antes, nos resultará intuitivo hacerlo. En Scratch, si el usuario está registrado en el sistema, el programa se guarda automáticamente. Si no lo está, la forma de guardar el proyecto se realiza a través del menú "Archivo" y la opción "Descargar a tu computadora", muy similar a como se hace en otros programas.

10.3.6.3. ¿Has utilizado elementos (dibujos, sonidos, etc.) creados por ti?

10.3.6.3.1. Análisis univariable: utilizar elementos creados por el participante

En la Tabla 35 y en la *Figura 52* se puede ver que la práctica totalidad de los encuestados pudieron elaborar elementos propios, siendo mayoría los que solo crearon dibujos.

Tabla 35
Respuestas obtenidas a la pregunta de si se ha podido utilizar elementos (dibujos, sonido, etc.) más allá de los existentes en Scratch creados por el participante.

Opciones	Nº de respuestas	Porcentaje de respuestas
Si, dibujos y sonidos	30	29,41%
Si, pero solo dibujos	57	55,88%
Si, pero solo sonidos	3	2,94%
No, yo no he creado nada. Sólo he utilizado dibujos y sonidos que ya estaban en Scratch	12	11,76%
Total	102	100%

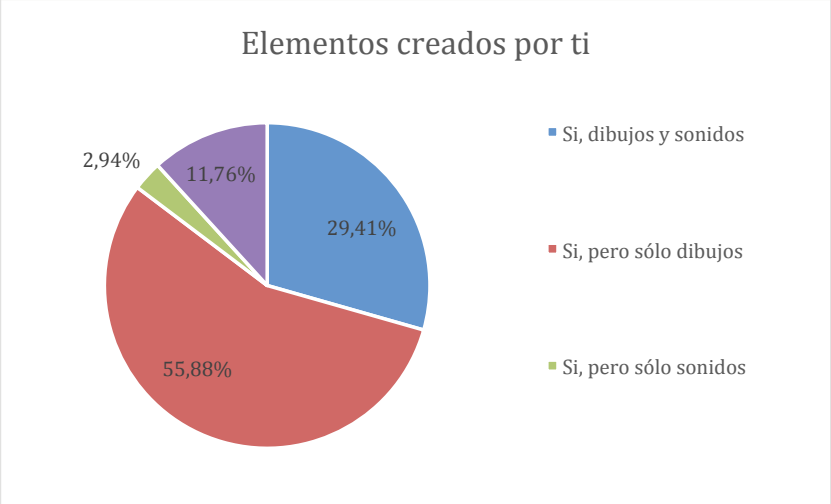


Figura 52. Gráfica de las respuestas obtenidas a la pregunta relativa a si se ha podido crear elementos (dibujos, sonido, etc.) más allá de los existentes en Scratch creados por el participante.

Fuente: Elaboración propia.

10.3.6.3.2. Análisis bivariable: utilizar elementos creados por el participante y clases recibidas

En la Tabla 36 y en la *Figura 53* se puede haber que no hay diferencias apreciables entre los que recibieron más clases que la media, y los que recibieron menos (ambas líneas van prácticamente en paralelo).

Tabla 36
Respuestas obtenidas a la pregunta de si se ha podido crear elementos (dibujos, sonidos, etc.), más allá de los existentes en Scratch, creados por el participante, *y su relación con el número de clases recibidas previamente*.

Clases recibidas	**Sí, dibujos y sonidos**		**Sí, pero solo dibujos**		**Sí, pero solo sonidos**		**Sólo he utilizado dibujos y sonidos que ya estaban en Scratch**		**TOTAL**	
	Nº	**%**	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**
Más clases que la media (Mayor)	15	31,25%	24	50%	2	4,17%	7	14,58%	48	100%
Menos clases que la media (Menor)	15	27,78%	33	61,11%	1	1,85%	5	9,26%	54	100%
TOTAL	30	29,41%	57	55,88%	3	2,94%	12	11,76%	102	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla.

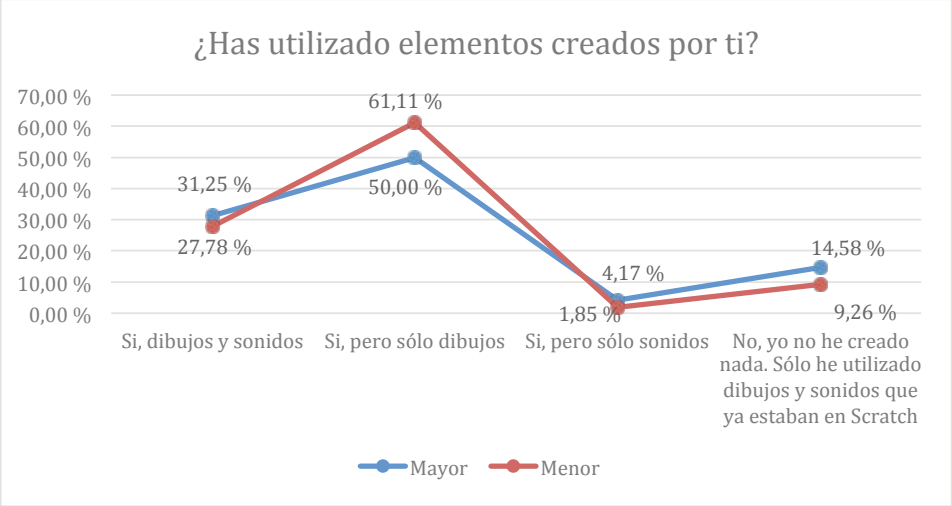


Figura 53. Gráfica de las respuestas obtenidas a la pregunta relativa a si se ha podido crear elementos (dibujos, sonidos, etc.), más allá de los existentes en Scratch, creados por el participante, y su relación con el número de clases recibidas previamente.

Fuente: Elaboración propia.

10.3.6.3.3. Análisis bivariable: utilizar elementos creados por el participante y práctica autónoma

En la Tabla 37 y en la *Figura 54* se ve que no hay diferencias de relevancia entre los que sí han tenido práctica autónoma y los que no la han tenido, con respecto a poder crear dibujos y sonidos propios.

Tabla 37
Respuestas obtenidas a la pregunta de si se ha podido utilizar elementos (dibujos, sonidos, etc.), más allá de los existentes en Scratch, creados por el participante, y su relación con la práctica autónoma.

	Sí, dibujos y sonidos		**Sí, pero solo dibujos**		**Sí, pero solo sonidos**		**No, yo no he creado nada. Sólo he utilizado dibujos y sonidos que ya estaban en Scratch**		**TOTAL**	
Práctica autónoma	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**
Sí	18	26,47%	38	55,88%	3	4,41%	9	13,24%	68	100%
No	11	33,33%	19	57,58%	0	0%	3	9,09%	33	100%
TOTAL	29	28,71%	57	56,44%	3	2,97%	12	11,88%	101	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla.

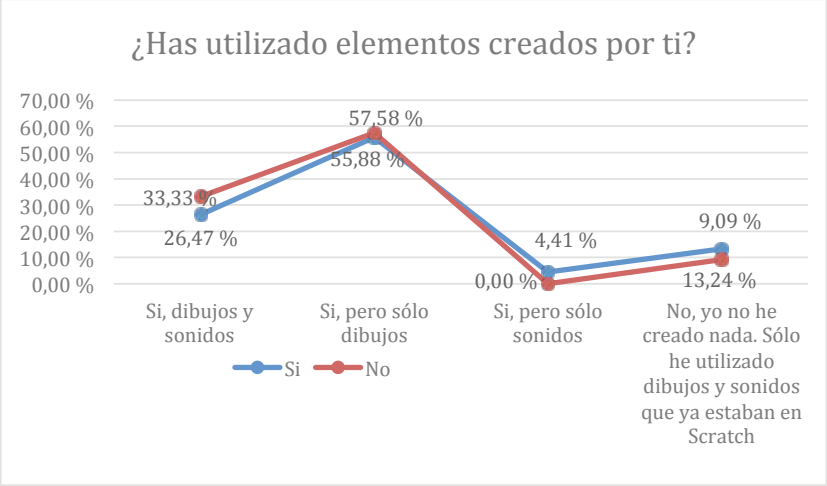


Figura 54. Gráfica de las respuestas obtenidas a la pregunta relativa a si se han utilizado elementos, más allá de los existentes en Scratch, creados por el participante, y su relación con la práctica autónoma.

Fuente: Elaboración propia.

10.3.6.3.4. Análisis bivariable: utilizar elementos creados por el participante y hábitos de uso del ordenador

En la Tabla 38 y en la *Figura 55* no se puede apreciar relación entre poder crear objetos propios, y el hábito de uso del ordenador.

Tabla 38
Respuestas obtenidas a la pregunta de si se han utilizado elementos (dibujos, sonidos, etc.), más allá de los existentes en Scratch, creados por el participante, *y su relación con los hábitos de uso del ordenador*.

	Si, dibujos y sonidos		**Si, pero solo dibujos**		**Si, pero solo sonidos**		**No, yo no he creado nada. Sólo he utilizado dibujos y sonidos que ya estaban en Scratch**		**TOTAL**	
Hábitos de uso del ordenador	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**
Tengo un ordenador en casa y lo utilizo habitualmente (todos o casi todos los días)	12	35,29%	17	50%	2	5,88%	3	8,82%	34	100%
Tengo un ordenador en casa, pero lo utilizo poco (una o dos veces por semana o menos)	11	22%	32	64%	1	2%	6	12%	50	100%

	Si, dibujos y sonidos		Si, pero solo dibujos		Si, pero solo sonidos		No, yo no he creado nada. Sólo he utilizado dibujos y sonidos que ya estaban en Scratch		TOTAL	
Hábitos de uso del ordenador	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**
En mi casa no hay ordenador, pero utilizo el del colegio, o el de un familiar o amigo, habitualmente (todos o casi todos los días)	4	66,67%	2	33,33%	0	0%	0	0%	6	100%
En mi casa no hay ordenador, pero utilizo el del colegio, o el de un familiar o amigo, de vez en cuando (una o dos veces por semana o menos)	1	33,33%	2	66,67%	0	0%	0	0%	3	100%
No suelo utilizar ningún ordenador	2	22,22%	4	44,44%	0	0%	3	33,33%	9	100%
TOTAL	30	29,41%	57	55,88%	3	2,94%	12	11,76%	102	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla.

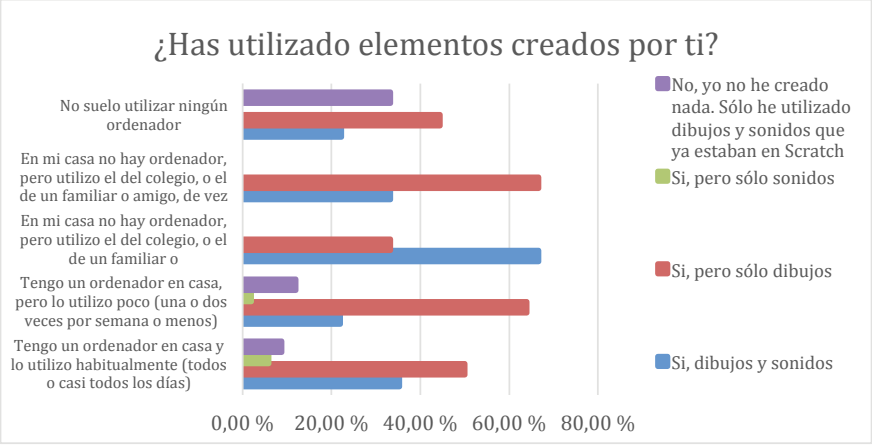


Figura 55. Gráfica de las respuestas obtenidas a la pregunta relativa a si se han utilizado los elementos *(dibujos, sonidos, etc.)*, más allá de los existentes en Scratch, creados por el participante, y su relación con los hábitos de uso del ordenador.

Fuente: Elaboración propia.

10.3.6.3.5. Conclusiones a la pregunta "¿Has utilizado elementos (dibujos, sonidos, etc.) creados por ti?

Según las respuestas a la Pregunta 4, la mayoría de los encuestados sí han podido crear sus propios elementos. Un 29,41% utilizan tanto dibujos como sonidos, un 55,88% solo dibujos, y un 2,94% solo sonidos. Hay un 11,76% que no crea ningún elemento, y solo utiliza los presentes en Scratch. De estos datos se pueden sacar varias conclusiones. Por un lado, parece ser que lo visual prevalece sobre lo auditivo. Por otro lado, la pauta HHS1.8 asociadas a la Pregunta 4, se cumple: la mayoría de los participantes pueden crear nuevos elementos, lo que significa que el programa sirve para este propósito. Las pautas NNGG7 y MIT1, que evalúan la flexibilidad del programa, y que se adapte a usuarios expertos y novatos, también se cumple, las personas que no saben crear nuevos dibujos o sonidos, pueden continuar con el desarrollo de su proyecto porque Scratch se los proporciona. Entre los que solo han utilizado elementos de Scratch no se distingue quiénes lo han hecho porque no han sabido, y quiénes por elección propia. Los docentes involucrados en el estudio, con los que se contrastaron las respuestas, afirmaron que se animó a los participantes a que crearan sus propios elementos, con lo que probablemente la mayoría de ese 11,76% que no lo hizo fue por no saber cómo.

10.3.6.4. ¿Has podido jugar a tu juego?

10.3.6.4.1. Análisis univariable: jugar al juego

En la Tabla 39 y la *Figura 56* se puede ver que un 86,41% pueden jugar a su juego, y solo unos pocos se quedan sin poder hacerlo.

Tabla 39
Respuestas obtenidas a la pregunta de si se ha podido jugar al juego.

Opciones	Nº de respuestas	Porcentajes de respuestas
SI	89	86,41%
NO	14	13,59%
TOTAL	103	100%

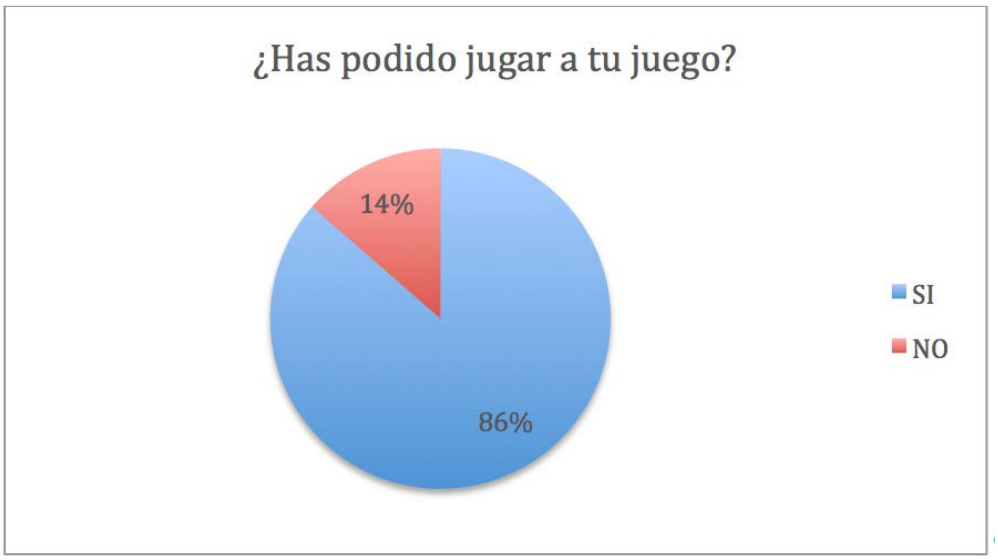


Figura 56. Gráfica de las respuestas obtenidas a la pregunta relativa a si se ha podido jugar al juego, y su relación con el número de clases recibidas previamente.

Fuente: Elaboración propia.

10.3.6.4.2. Análisis bivariable: jugar al juego y clases recibidas

En la Tabla 40 y en la *Figura 57* se ve que, contrariamente a lo que pudiera pensarse en un primer momento, la mayoría de los participantes que no pudieron jugar a su juego habían recibido más clases que la media.

Tabla 40
Respuestas obtenidas a la pregunta de si se ha podido jugar al juego, y su relación con el número de clases recibidas previamente.

Clases recibidas	**Sí**		**No**		**TOTAL**	
	Nº	**%**	**Nº**	**%**	**Nº**	**%**
Más clases que la media (Mayor)	36	75%	12	25%	48	100%
Menos clases que la media (Menor)	53	96,36%	2	3,64%	55	100%
TOTAL	89	86,41%	14	13,59%	103	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla.

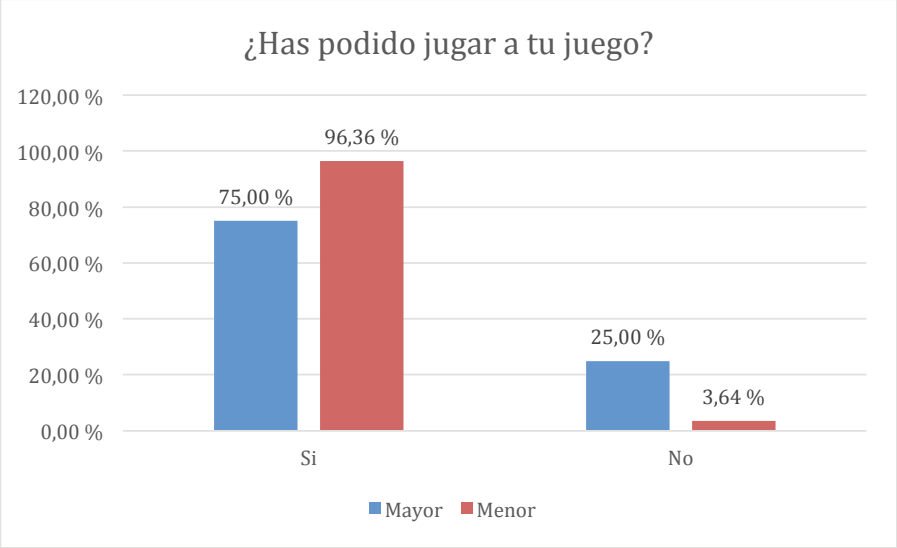


Figura 57. Gráfica de las respuestas obtenidas a la pregunta relativa a si se ha podido jugar al juego, y su relación con el número de clases recibidas previamente.

Fuente: Elaboración propia.

10.3.6.4.3. Análisis bivariable: jugar al juego y práctica autónoma

En la Tabla 41 y en la *Figura 58* se ve que hay una ligera tendencia a que los que no pudieron jugar al juego se agrupen dentro de los que no tuvieron práctica autónoma con Scratch.

Tabla 41
Respuestas obtenidas a la pregunta de si se ha podido jugar al juego, y su relación con la práctica autónoma.

Práctica autónoma	**Sí**		**No**		**TOTAL**	
	Nº	**%**	**Nº**	**%**	**Nº**	**%**
Sí	63	91,30%	6	8,70%	69	100%
No	25	75,76%	8	24,24%	33	100%
TOTAL	88	86,27%	14	13,73%	102	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla.

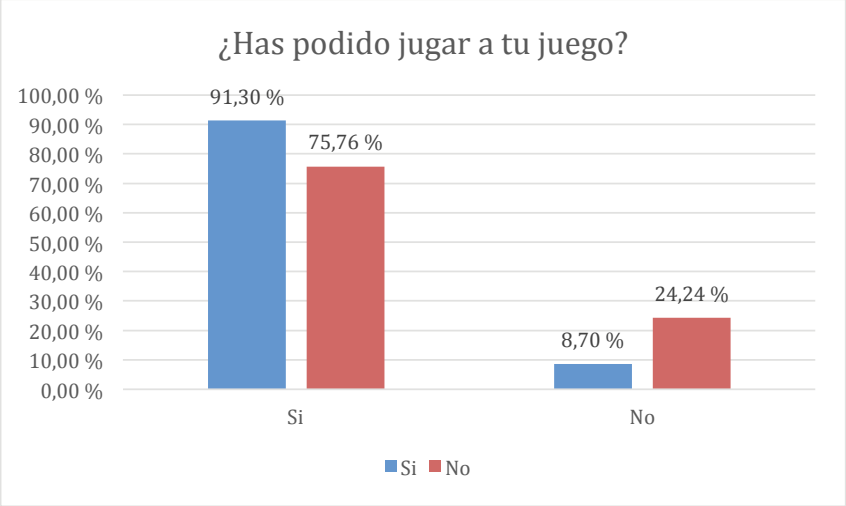


Figura 58. Gráfica de las respuestas obtenidas a la pregunta relativa a si se ha podido jugar al juego, y su relación con la práctica autónoma.

Fuente: Elaboración propia.

10.3.6.4.4. Análisis bivariable: jugar al juego y hábitos de uso del ordenador

En la Tabla 42 y en la *Figura 59* se puede ver que de los pocos que no han podido jugar al juego, la mayoría se encuentran entre los que menos utilizan el ordenador.

Tabla 42
Respuestas obtenidas a la pregunta de si ha podido jugar al juego, elegir un personaje y hacer que se mueva unos minutos sin que nadie le ayude y su relación con los hábitos de uso del ordenador.

	Sí		**No**		**TOTAL**	
Hábitos de uso del ordenador	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**
Tengo un ordenador en casa y lo utilizo habitualmente (todos o casi todos los días)	34	97,14%	1	2,86%	35	100%
Tengo un ordenador en casa, pero lo utilizo poco (una o dos veces por semana o menos)	44	88%	6	12%	50	100%
En mi casa no hay ordenador, pero utilizo el del colegio, o el de un familiar o amigo, habitualmente (todos o casi todos los días)	4	66,67%	2	33,33%	6	100%

	Sí		No		TOTAL	
Hábitos de uso del ordenador	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**
En mi casa no hay ordenador, pero utilizo el del colegio, o el de un familiar o amigo, de vez en cuando (una o dos veces por semana o menos)	3	100%	0	0%	3	100%
No suelo utilizar ningún ordenador	4	44,44%	5	55,56%	9	100%
TOTAL	89	86,41%	14	13,59%	103	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla.

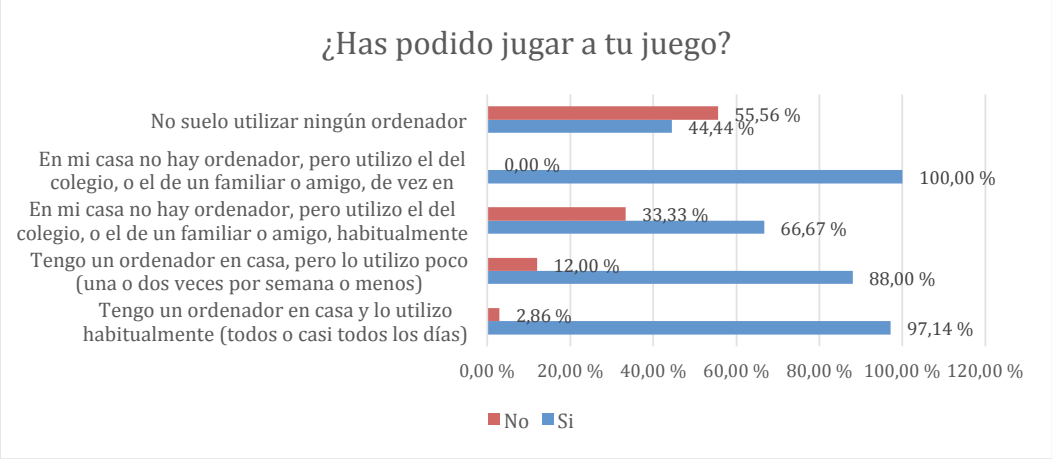


Figura 59. Gráfica de las respuestas obtenidas a la pregunta relativa a si se ha podido jugar al juego, y su relación con los hábitos de uso del ordenador.

Fuente: Elaboración propia.

10.3.6.4.5. Conclusiones a la pregunta "¿Has podido jugar a tu juego?"

La Pregunta 5 está relacionada con la pauta HHS1.8, que cuestiona la funcionalidad y utilidad del programa. La mayoría de los participantes han podido jugar a su juego, con lo que esta pauta se cumple si nos referimos a la utilidad de Scratch para poder crear juegos que se puedan jugar. Esta pregunta también está relacionada la NOKIA6, que se relaciona con las recompensas y motivaciones que ofrece el programa. Un 86% han podido jugar al juego, con lo que parece ser que esta pauta también se cumple, si entendemos como recompensa el poder jugar a tu propia creación.

10.3.6.5. ¿Te ha resultado divertido jugar a tu juego?

10.3.6.5.1. Análisis univariable: si ha resultado divertido el juego

En la Tabla 43 y en la *Figura 60* se ve que una práctica totalidad de los participantes han disfrutado al jugar a su juego.

Tabla 43
Respuestas obtenidas a la pregunta de si ha resultado divertido al participante jugar al juego.

Opciones	Nº de respuestas	Porcentajes de respuestas
SI	85	95,51%
NO	4	4,49%
TOTAL	89	100%

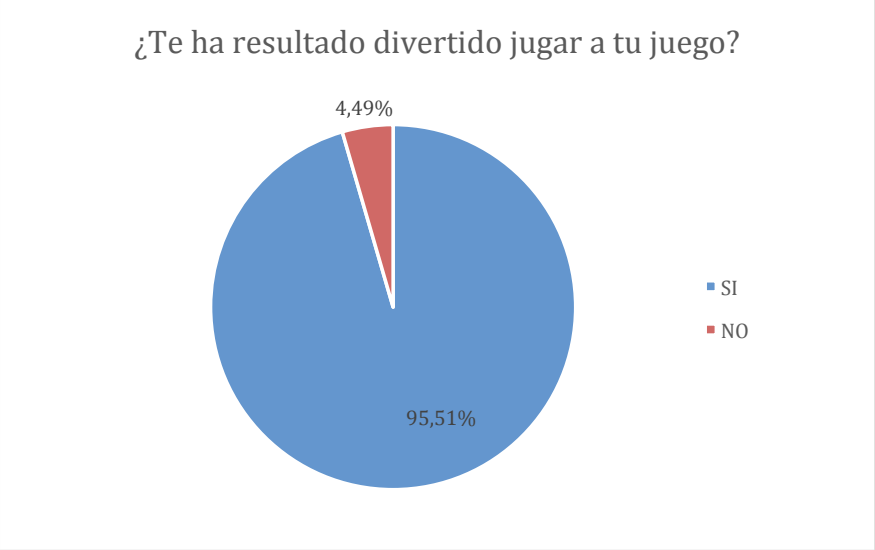


Figura 60. Gráfica de las respuestas obtenidas a la pregunta relativa a si ha resultado divertido al participante jugar al juego.

Fuente: Elaboración propia.

10.3.6.5.2. Análisis bivariable: si ha resultado divertido el juego y clases recibidas

En la Tabla 44 y en la *Figura 61* se ve que el 4,49% que no ha disfrutado de su juego se encuentra íntegramente encuadrado entre los participantes que han recibido menos clases que la media.

Tabla 44

Respuestas obtenidas a la pregunta de si ha resultado divertido al participante jugar al juego, y su relación con el número de clases recibidas previamente.

Clases recibidas	Sí		No		TOTAL	
	Nº	%	Nº	%	Nº	%
Más clases que la media (Mayor)	36	100%	0	0%	36	100%
Menos clases que la media (Menor)	49	92,45%	4	7,55%	53	100%
TOTAL	85	95,51%	4	4,49%	89	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla.

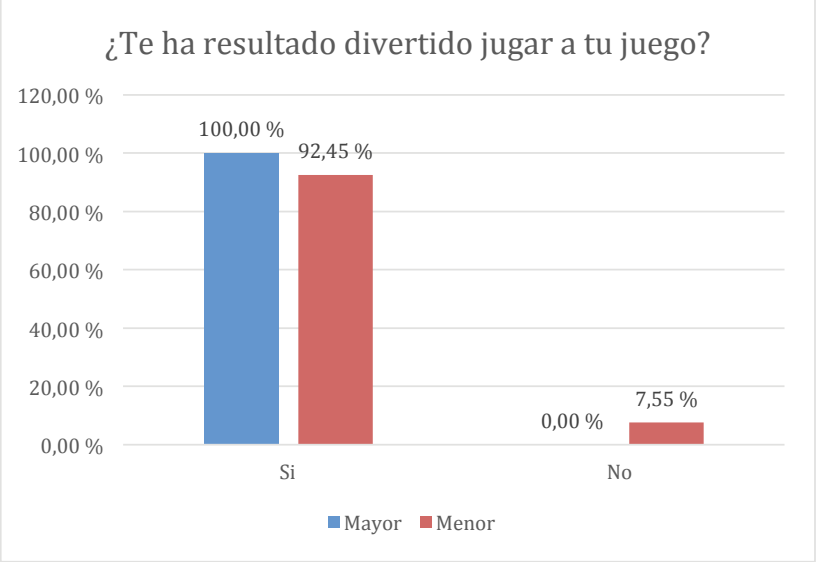


Figura 61. Gráfica de las respuestas obtenidas a la pregunta relativa a si ha resultado divertido al participante jugar al juego, y su relación con el número de clases recibidas previamente.

Fuente: Elaboración propia.

10.3.6.5.3. Análisis bivariable: si ha resultado divertido el juego y práctica autónoma

En la Tabla 45 y en la *Figura 62*, de los 4 participantes que respondieron que no les resultó divertido jugar al juego, 3 no han tenido práctica autónoma, y 1 sí. En cualquier caso, el número de respuestas de las personas que no han disfrutado del juego es muy pequeño, por lo que no se puede establecer una relación clara entre estas dos variables.

Tabla 45
Respuestas obtenidas a la pregunta de si ha resultado divertido al participante jugar al juego, y su relación con la práctica autónoma.

	Sí		No		TOTAL	
Práctica autónoma	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**
Sí	62	98,41%	1	1,59%	63	100%
No	22	88%	3	12%	25	100%
TOTAL	84	95,45%	4	4,55%	88	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla.

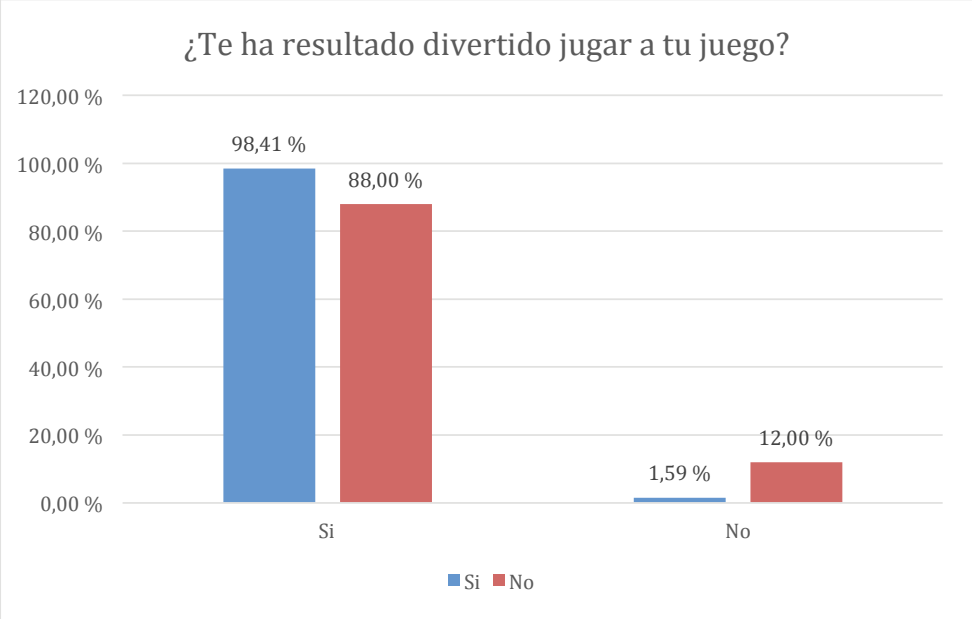


Figura 62. Gráfica de las respuestas obtenidas a la pregunta relativa a si ha resultado divertido al participante jugar al juego, y su relación con la práctica autónoma.

Fuente: Elaboración propia.

10.3.6.5.4. Análisis bivariable: si ha resultado divertido el juego y hábitos de uso del ordenador

Lo más relevante que se puede ver en la Tabla 46 y en la *Figura 63* es que el 100% de los participantes que menos utilizan el ordenador (poco o nada) han disfrutado de su juego.

Tabla 46
Respuestas obtenidas a la pregunta de si ha resultado divertido al participante jugar al juego, y su relación con los hábitos de uso del ordenador.

	Sí		No		TOTAL	
Hábitos de uso del ordenador	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**
Tengo un ordenador en casa y lo utilizo habitualmente (todos o casi todos los días)	33	97,06%	1	2,94%	34	100%
Tengo un ordenador en casa, pero lo utilizo poco (una o dos veces por semana o menos)	41	93,18%	3	6,82%	44	100%
En mi casa no hay ordenador, pero utilizo el del colegio, o el de un familiar o amigo, habitualmente (todos o casi todos los días)	4	100%	0	0%	4	100%
En mi casa no hay ordenador, pero utilizo el del colegio, o el de un familiar o amigo, de vez en cuando (una o dos veces por semana o menos)	3	100%	0	0%	3	100%
No suelo utilizar ningún ordenador	4	100%	0	0%	4	100%
TOTAL	85	95,51%	4	4,49%	89	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla.

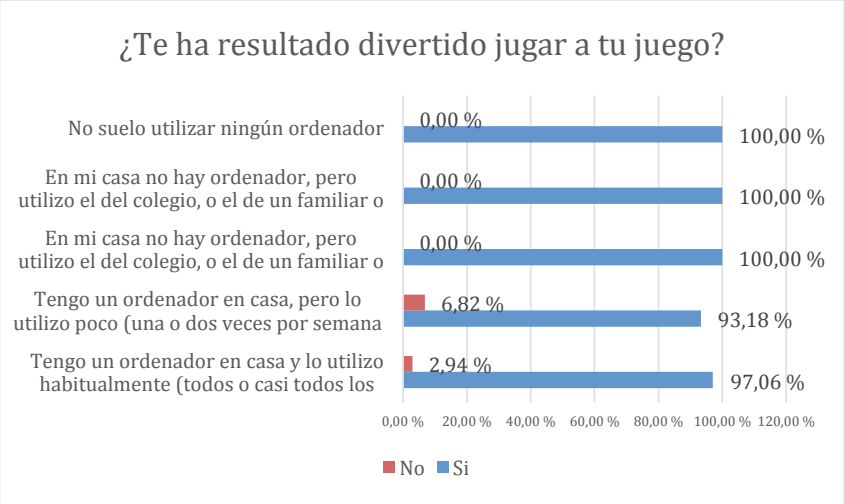


Figura 63. Gráfica de las respuestas obtenidas a la pregunta relativa a si ha resultado divertido al participante jugar al juego, y su relación con los hábitos de uso del ordenador.

Fuente: Elaboración propia.

10.3.6.5.5. Conclusiones a la pregunta "¿Te ha resultado divertido jugar a tu juego?"

La Pregunta 6 solo se realiza a las personas que sí pudieron crear el juego (es decir, respondieron Sí a la Pregunta 5). Esta pregunta está vinculada con las pautas HHS1.3 y NOKIA6, relacionadas con si la herramienta se adapta a las expectativas de los usuarios, y si es motivante. A la vista de los resultados, parece que ambas pautas se cumplen, la mayoría de los participantes disfrutan jugando a su juego.

10.3.6.6. ¿Cuánto has tardado en descubrir cómo se combinan los bloques?

10.3.6.6.1. Análisis univariable: tiempo en descubrir cómo se combinan los bloques

En la Tabla 47 y en la *Figura 64* se refleja que a priori la mayoría entiende la lógica de bloques que utiliza Scratch para programar

Tabla 47
Respuestas obtenidas a la pregunta de cuánto se ha tardado en descubrir cómo se combinan los bloques.

Opciones	Nº de respuestas	Porcentajes de respuestas
A la primera, tras uno o dos intentos	26	25,49%
Después de probar un rato	46	45,10%
Lo he entendido, pero alguna vez me ha tenido que volver a ayudar el profesor	28	27,45%
El profesor me ha tenido que ayudar todo el rato	2	1,96%
TOTAL	102	100%

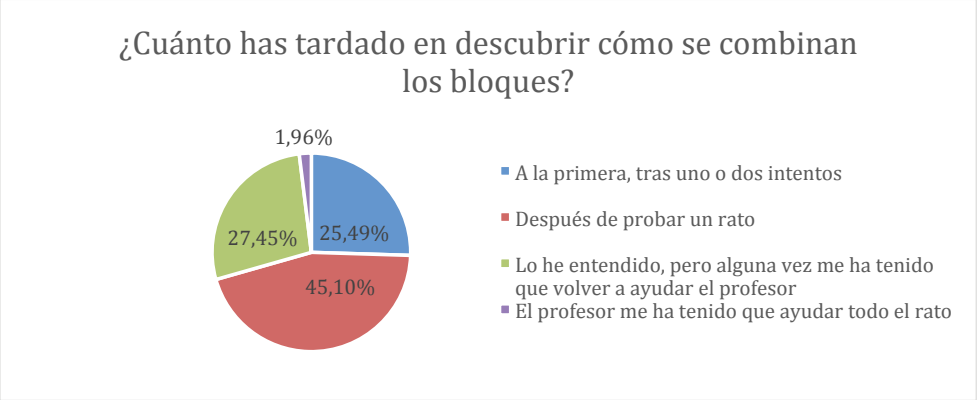


Figura 64. Gráfica de las respuestas obtenidas a la pregunta relativa a cuánto se ha tardado en descubrir cómo se combinan los bloques.

Fuente: Elaboración propia.

10.3.6.6.2. Análisis bivariable: tiempo en descubrir cómo se combinan los bloques y clases recibidas

Analizando la Tabla 48 y en la *Figura 65* no se encuentran evidencias que apunten a la relación entre saber combinar bloques, y el número de clases recibidas.

Tabla 48
Respuestas obtenidas a la pregunta de cuánto se ha tardado en descubrir cómo se combinan los bloques, y su relación con el número de clases recibidas previamente.

	A la primera, tras uno o dos intentos		Después de probar un rato		Lo he entendido, pero alguna vez me ha tenido que volver a ayudar el profesor		El profesor me ha tenido que ayudar todo el rato		TOTAL	
Clases recibidas	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**
Más clases que la media (Mayor)	11	22,92%	19	39,58%	17	35,42%	1	2,08%	48	100%
Menos clases que la media (Menor)	15	27,78%	27	50%	11	20,37%	1	1,85%	54	100%
TOTAL	26	25,49%	46	45,10%	28	27,45%	2	1,96%	102	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla.

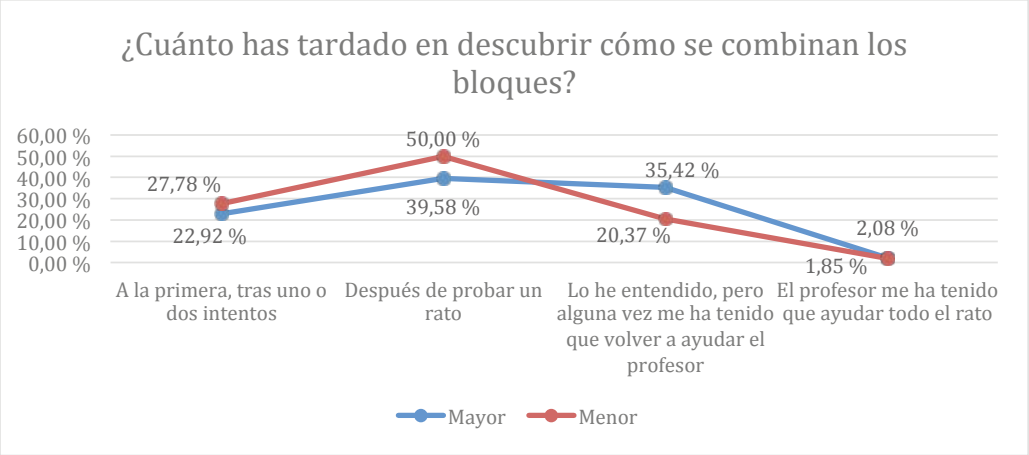


Figura 65. Gráfica de las respuestas obtenidas a la pregunta relativa a cuánto se ha tardado en descubrir cómo se combinan los bloques, y su relación con el número de clases recibidas previamente.

Fuente: Elaboración propia.

10.3.6.6.3. Análisis bivariable: tiempo en descubrir cómo se combinan los bloques y práctica autónoma

A partir de los datos que se ven en la Tabla 49 y se reflejan en la *Figura 66*, no se puede afirmar que exista relación entre la práctica autónoma con Scratch y conocer la lógica de bloques del programa.

Tabla 49
Respuestas obtenidas a la pregunta de cuánto se ha tardado en descubrir cómo se combinan los bloques, y su relación con la práctica autónoma.

	A la primera, tras uno o dos intentos		**Después de probar un rato**		**Lo he entendido, pero alguna vez me ha tenido que volver a ayudar el profesor**		**El profesor me ha tenido que ayudar todo el rato**		**TOTAL**	
Práctica autónoma	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**
Sí	20	28,99%	33	47,83%	16	23,19%	0	0%	69	100%
No	6	18,75%	12	37,50%	12	37,50%	2	6,25%	32	100%
TOTAL	26	25,74%	45	44,55%	28	27,72%	2	1,98%	101	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla.

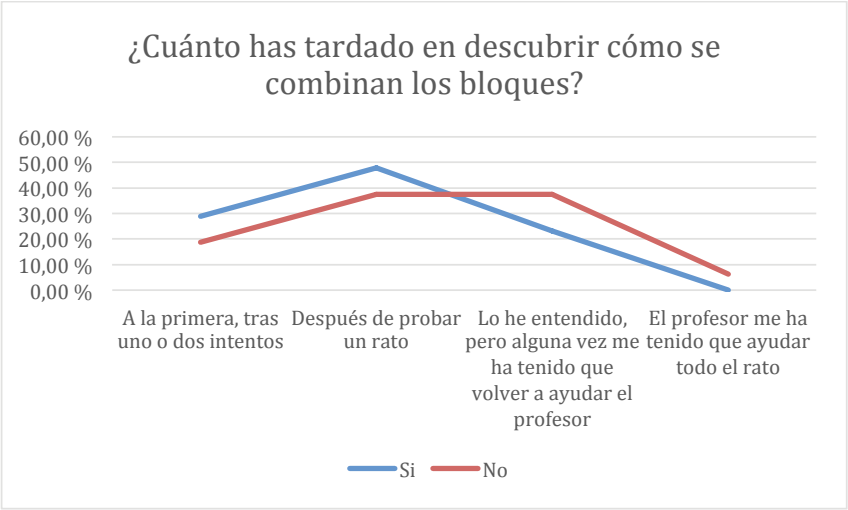


Figura 66. Gráfica de las respuestas obtenidas a la pregunta relativa a cuánto se ha tardado en descubrir cómo se combinan los bloques.

Fuente: Elaboración propia.

10.3.6.6.4. Análisis bivariable: tiempo en descubrir cómo se combinan los bloques y hábitos de uso del ordenador

Al igual que en análisis bivariable anterior, por lo que se aprecia en Tabla 50 y en la *Figura 67*, hay una leve tendencia a que los participantes que más ayuda han necesitado sean los que menos utilizan el ordenador.

Tabla 50
Respuestas obtenidas a la pregunta de cuánto se ha tardado en descubrir cómo se combinan los bloques, y su relación con los hábitos de uso del ordenador.

	A la primera, tras uno o dos intentos		**Después de probar un rato**		**Lo he entendido, pero alguna vez me ha tenido que volver a ayudar el profesor**		**El profesor me ha tenido que ayudar todo el rato**		**TOTAL**	
Hábitos de uso del ordenador	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**
Tengo un ordenador en casa y lo utilizo habitualmente (todos o casi todos los días)	9	25,71%	21	60%	4	11,43%	1	2,86%	35	100%
Tengo un ordenador en casa, pero lo utilizo poco (una o dos veces por semana o menos)	13	26,53%	17	34,69%	19	38,78%	0	0%	49	100%
En mi casa no hay ordenador, pero utilizo el del colegio, o el de un familiar o amigo, habitualmente (todos o casi todos los días)	0	0%	4	66,67%	2	33,33%	0	0%	6	100%
En mi casa no hay ordenador, pero utilizo el del colegio, o el de un familiar o amigo, de vez en cuando (una o dos veces por semana o menos)	2	66,67%	0	0%	1	33,33%	0	0%	3	100%
No suelo utilizar ningún ordenador.	2	22,22%	4	44,44%	2	22,22%	1	11,11%	9	100%
TOTAL	26	25,49%	46	45,10%	28	27,45%	2	1,96%	102	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla.

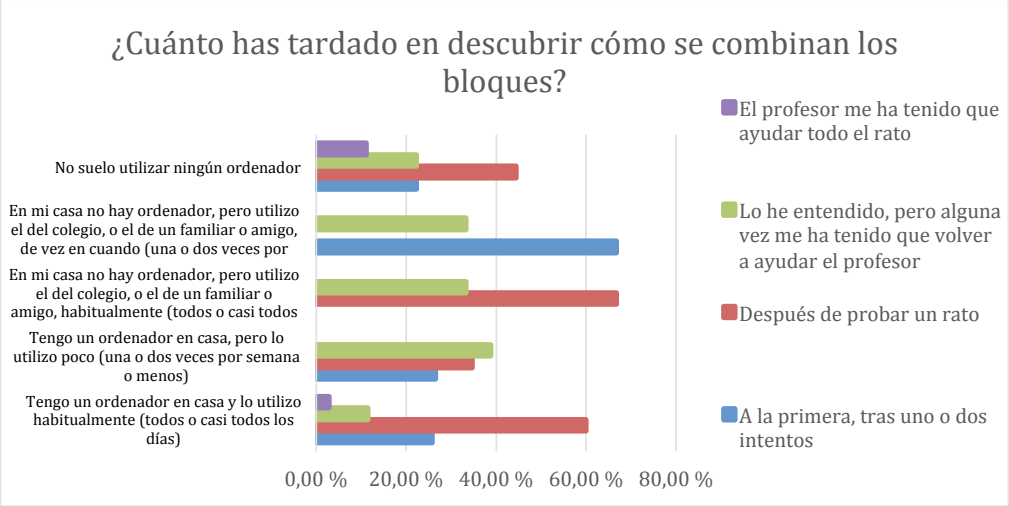


Figura 67. Gráfica de las respuestas obtenidas a la pregunta relativa a cuánto se ha tardado en descubrir cómo se combinan los bloques, y su relación con los hábitos de uso del ordenador.

Fuente: Elaboración propia.

10.3.6.6.5. Conclusiones a la pregunta "¿Cuánto has tardado en descubrir cómo se combinan los bloques?"

La Pregunta 9 pretende evaluar lo intuitivo que es el sistema de programación por bloques que incorpora Scratch, y por las respuestas, parece que sí lo es, dado que no llega al 2% los participantes que realmente experimentan problemas. Las pautas del análisis heurístico que dan origen a esta pregunta son la NNGG6, relacionada con minimizar lo que el usuario tiene que memorizar, la MIT10, relacionada con el diseño sencillo del programa, y la AGUC1, que cuestiona si la estructura se ajusta a su finalidad. Según las respuestas a la Pregunta 9, parece ser que estas tres pautas se cumplen, y que el sistema de combinación de bloques es intuitivo. En cualquier caso, la Pregunta 10 se utiliza, entre otras cosas, para confirmar que efectivamente los participantes han sido capaces de comprender esta lógica de bloques.

10.3.6.7. ¿Sabrías decirme si se pueden encajar las siguientes piezas?

10.3.6.7.1. Análisis univariable: respuestas a si se pueden encajar las siguientes piezas [1. por siempre - apuntar hacia]

Esta pregunta se debía responder con un Sí, dado que estas piezas sí se pueden combinar. Como se puede apreciar en la Tabla 51 y en la *Figura 68*, el 97,62% de los participantes acierta.

Tabla 51
Respuestas obtenidas a la pregunta de si se ha podido combinar los bloques [1. por siempre - apuntar hacia]

Opciones	Nº de respuestas	Porcentaje de respuestas
Sí	82	97,62%
No	2	2,38%
TOTAL	84	100%

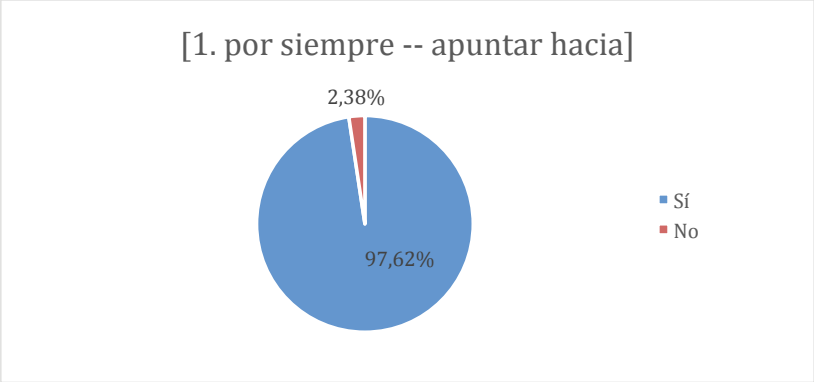


Figura 68. Gráfica de las respuestas obtenidas a la pregunta relativa a si se ha podido combinar los bloques [1. por siempre - apuntar hacia]

Fuente: Elaboración propia.

10.3.6.7.2. Análisis bivariable: respuestas a si se pueden encajar las siguientes piezas [1. por siempre - apuntar hacia] y clases recibidas

En la Tabla 52 y en la *Figura 69* se puede ver que todos los que no respondieron correctamente a esta pregunta, recibieron menos clases que la media, pero al ser tan pocos los participantes que han fallado, no se puede establecer una relación clara entre estas variables.

Tabla 52
Respuestas obtenidas a la pregunta de si se ha podido combinar los bloques [1. por siempre - apuntar hacia], y su relación con el número de clases recibidas previamente.

	Sí		No		TOTAL	
Clases recibidas	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**
Más clases que la media (Mayor)	38	100%	0	0%	38	100%
Menos clases que la media (Menor)	44	95,65%	2	4,35%	46	100%
TOTAL	82	97,62%	2	2,38%	84	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla.

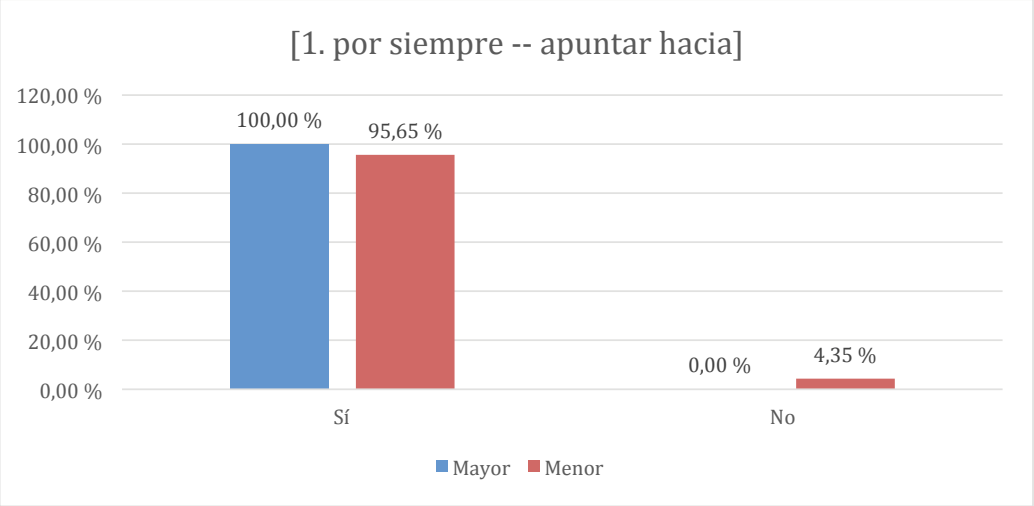


Figura 69. Gráfica de las respuestas obtenidas a la pregunta relativa a si se ha podido combinar los bloques [1. por siempre - apuntar hacia], y su relación con el número de clases recibidas previamente.

Fuente: Elaboración propia.

10.3.6.7.3. Análisis bivariable: respuestas a si se pueden encajar las siguientes piezas [1. por siempre - apuntar hacia] y práctica autónoma

En la Tabla 53 y en la *Figura 70* se puede apreciar que no hay indicios significativos de relación entre haber respondido correctamente a esta pregunta, y la práctica autónoma

Tabla 53
Respuestas obtenidas a la pregunta de si ha podido combinar los bloques [1. por siempre - apuntar hacia], y su relación con la práctica autónoma.

	Sí		No		TOTAL	
Práctica autónoma	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**
Sí	56	98,25%	1	1,75%	57	100%
No	25	96,15%	1	3,85%	26	100%
TOTAL	81	97,59%	2	2,41%	83	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla.

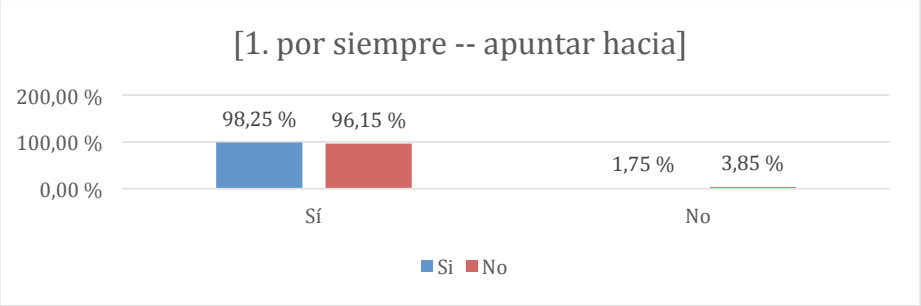


Figura 70. Gráfica de las respuestas obtenidas a la pregunta relativa a combinar bloques [1. por siempre - apuntar hacia], y su relación con la práctica autónoma.

Fuente: Elaboración propia.

10.3.6.7.4. Análisis bivariable: respuestas a si se pueden encajar las siguientes piezas [1. por siempre - apuntar hacia] y hábitos de uso del ordenador

En la Tabla 54 y en la *Figura 71* se ve reflejado que prácticamente la totalidad de los participantes que fallaron al responder esta pregunta se encuentran entre los estudiantes que no utilizan el ordenador en casa. En cualquier caso, es una evidencia muy débil como para poder establecer una relación clara entre no haber acertado, y los hábitos de uso del ordenador.

Tabla 54
Respuestas obtenidas a la pregunta de si se ha podido combinar los bloques [1. por siempre - apuntar hacia], y su relación con los hábitos de uso del ordenador.

	Sí		**No**		**TOTAL**	
Hábitos de uso del ordenador	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**
Tengo un ordenador en casa y lo utilizo habitualmente (todos o casi todos los días)	32	100%	0	0%	32	100%
Tengo un ordenador en casa, pero lo utilizo poco (una o dos veces por semana o menos)	41	97,62%	1	2,38%	42	100%
En mi casa no hay ordenador, pero utilizo el del colegio, o el de un familiar o amigo, habitualmente (todos o casi todos los días)	2	100%	0	0%	2	100%
En mi casa no hay ordenador, pero utilizo el del colegio, o el de un familiar o amigo, de vez en cuando (una o dos veces por semana o menos)	2	100%	0	0%	2	100%
No suelo utilizar ningún ordenador	5	83,33%	1	16,67%	6	100%
TOTAL	82	97,62%	2	2,38%	84	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla.

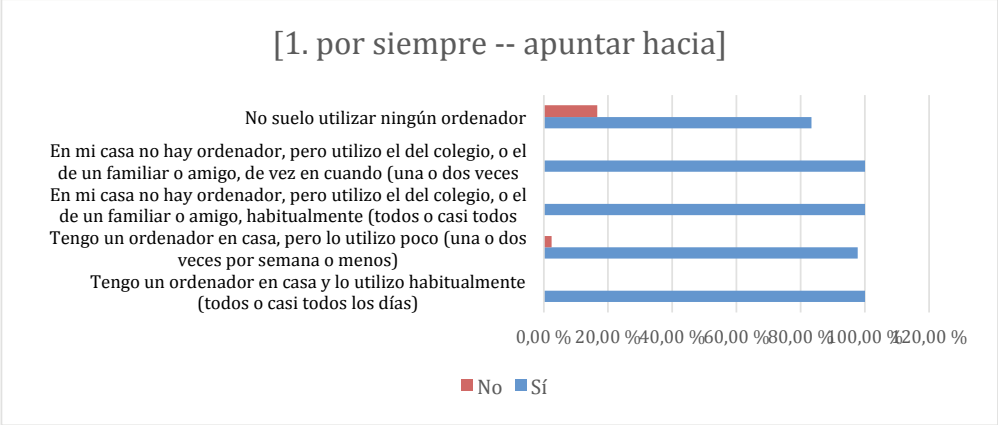


Figura 71. Gráfica de las respuestas obtenidas a la pregunta relativa a si se ha podido combinar los bloques [1. por siempre - apuntar hacia], y su relación con los hábitos de uso del ordenador.

Fuente: Elaboración propia.

10.3.6.7.5. Análisis univariable: respuestas a si se pueden encajar las siguientes piezas [2. repetir - ¿tecla presionada?]

Las piezas por las que se pregunta no se pueden combinar. Por lo que se puede apreciar en la Tabla 55 y en la *Figura 72*, un 89,29% aciertan la respuesta.

Tabla 55
Respuestas obtenidas a la pregunta de si se pueden encajar las siguientes piezas [2. repetir - ¿tecla presionada?]

Opciones	Nº de respuestas	Porcentajes de respuestas
Sí	9	10,71%
No	75	89,29%
TOTAL	84	100%

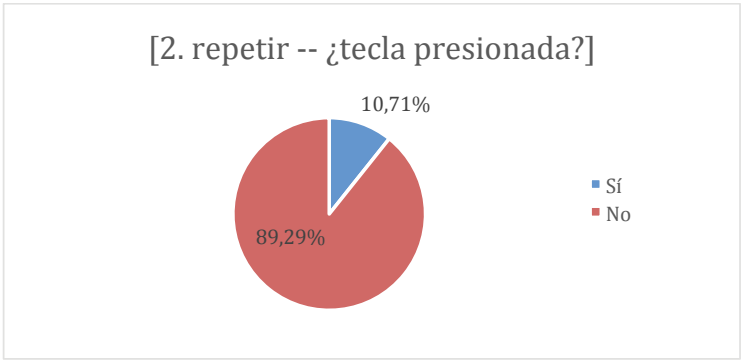


Figura 72. Gráfica de las respuestas obtenidas a la pregunta relativa a si se pueden encajar las siguientes piezas [2. repetir - ¿tecla presionada?]

Fuente: Elaboración propia.

10.3.6.7.6. Análisis bivariable: respuestas a si se pueden encajar las siguientes piezas [2. repetir - ¿tecla presionada?], y clases recibidas

En la Tabla 56 y en la *Figura 73* no se aprecia relación entre haber acertado o fallado esta pregunta, y el número de clases recibidas, los porcentajes en uno y otro caso son muy similares.

Tabla 56
Respuestas obtenidas a la pregunta de si se pueden encajar las siguientes piezas [2. repetir - ¿tecla presionada?], y su relación con el número de clases recibidas previamente.

	Sí		No		TOTAL	
Clases recibidas	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**
Más clases que la media (Mayor)	5	13,16%	33	86,84%	38	100%
Menos clases que la media (Menor)	4	8,70%	42	91,30%	46	100%
TOTAL	9	10,71%	75	89,29%	84	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla.

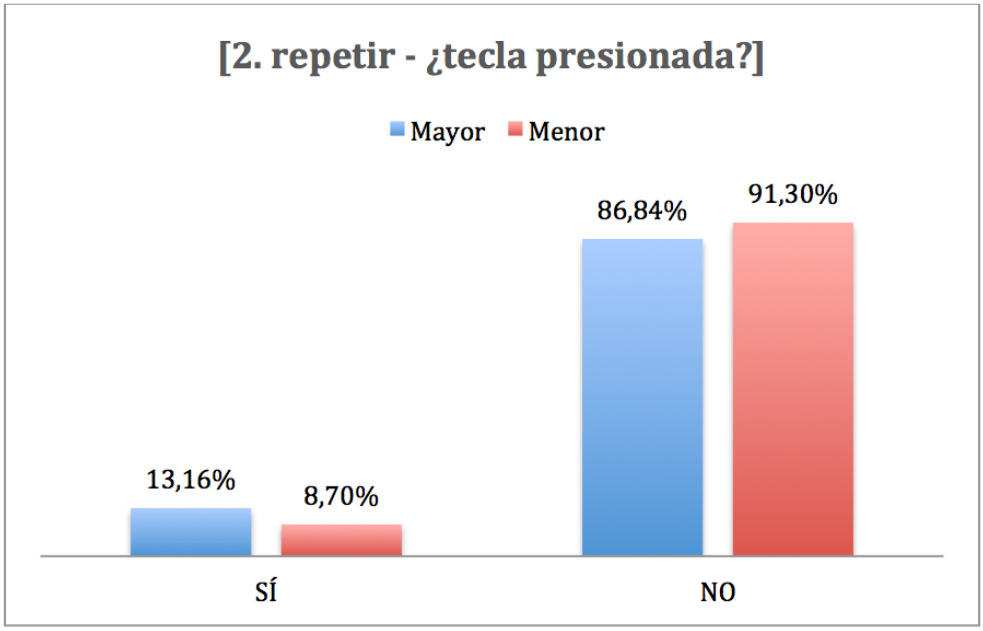


Figura 73. Gráfica de las respuestas obtenidas a la pregunta relativa a si se pueden encajar las siguientes piezas [2. repetir - ¿tecla presionada*?], y su relación con el número de clases recibidas previamente*

Fuente: Elaboración propia.

10.3.6.7.7. Análisis bivariable: respuestas a si se pueden encajar las siguientes piezas [2. repetir - ¿tecla presionada?] y práctica autónoma

En la Tabla 57 y en la *Figura 74* reflejan que no hay más personas que fallaron la respuesta entre los que no tuvieron práctica autónoma. En cualquier caso, son muy pocos los que no aciertan, por lo que no se puede afirmar con rotundidad esta tendencia.

Tabla 57

Respuestas obtenidas a la pregunta de si se pueden encajar las siguientes piezas[2. repetir - ¿tecla presionada?], y su relación con la práctica autónoma.

	Sí		No		TOTAL	
Práctica autónoma	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**
Sí	5	8,77%	52	91,23%	57	100%
No	4	15,38%	22	84,62%	26	100%
TOTAL	9	10,84%	74	89,16%	83	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla.

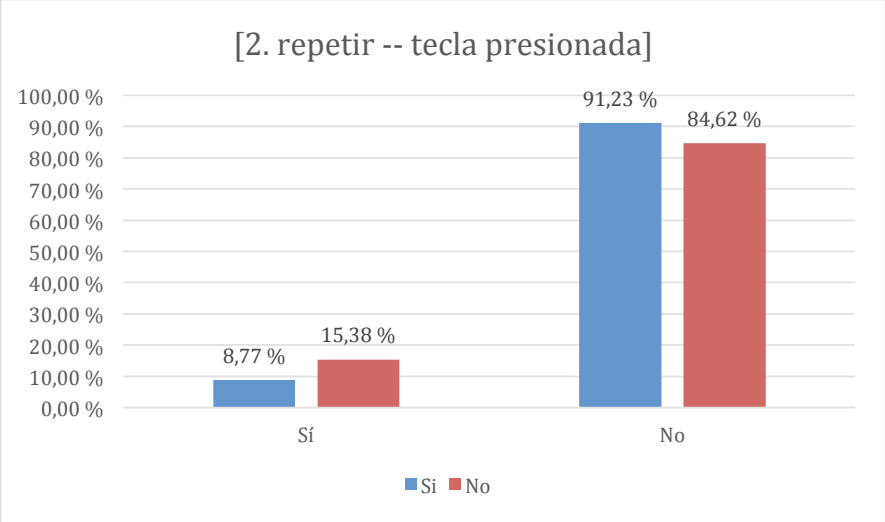


Figura 74. Gráfica de las respuestas obtenidas a la pregunta relativa a si se pueden encajar las siguientes piezas [2. repetir - ¿tecla presionada?], y su relación con la práctica autónoma.

Fuente: Elaboración propia.

10.3.6.7.8. Análisis bivariable: respuestas a si se pueden encajar las siguientes piezas [2. repetir - ¿tecla presionada?] y hábitos de uso del ordenador

En la Tabla 58 y en la *Figura 75* se puede ver que los participantes que no acertaron a la pregunta no se encuadran en un grupo concreto, los hay que utilizan el ordenador con asiduidad, y los hay que no lo utilizan, repartidos al 50% entre uno y otro grupo. Por tanto, no se aprecian indicios de relación con la variable que mide los hábitos de uso.

Tabla 58
Respuestas obtenidas a la pregunta de si se pueden encajar las siguientes piezas [2. repetir - ¿tecla presionada?], y su relación con los hábitos de uso del ordenador.

	Sí		**No**		**TOTAL**	
Hábitos de uso del ordenador	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**
Tengo un ordenador en casa y lo utilizo habitualmente (todos o casi todos los días)	2	6,25%	30	93,75%	32	100%
Tengo un ordenador en casa, pero lo utilizo poco (una o dos veces por semana o menos)	6	14,29%	36	85,71%	42	100%
En mi casa no hay ordenador, pero utilizo el del colegio, o el de un familiar o amigo, habitualmente (todos o casi todos los días)	0	0%	2	100%	2	100%
En mi casa no hay ordenador, pero utilizo el del colegio, o el de un familiar o amigo, de vez en cuando (una o dos veces por semana o menos)	0	0%	2	100%	2	100%
No suelo utilizar ningún ordenador	1	16,67%	5	83,33%	6	100%
TOTAL	9	10,71%	75	89,29%	84	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla.

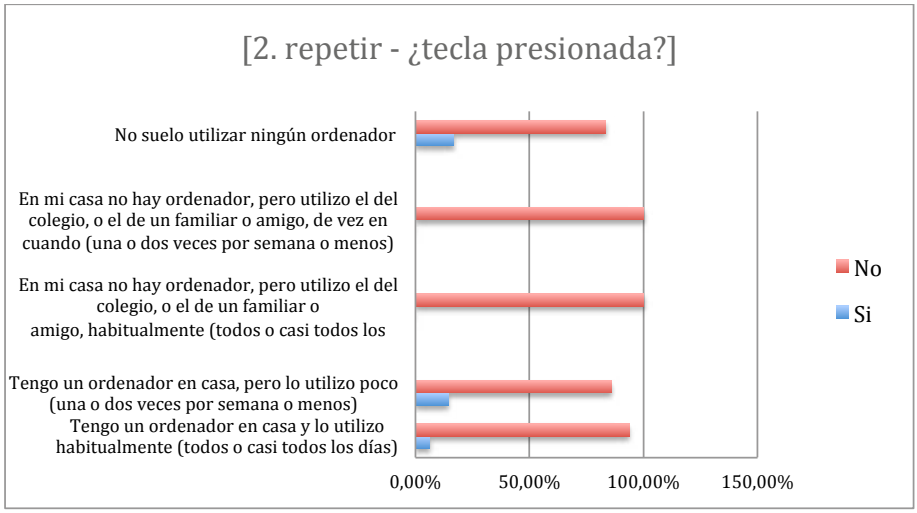


Figura 75. Gráfica de las respuestas obtenidas a la pregunta relativa a si se pueden encajar las siguientes piezas [2. repetir - ¿tecla presionada?] y su relación con los hábitos de uso del ordenador.

Fuente: Elaboración propia.

10.3.6.7.9. Análisis univariable: respuestas a si se pueden encajar las siguientes piezas [3. no - ir a x: y:]

La respuesta correcta a esta pregunta era No, dado que las piezas por la que cuestiona esta pregunta no se pueden encajar. En este caso un 69,05% responde correctamente, como se puede ver en la Tabla 59 y en la *Figura 76*, un porcentaje menor que el encontrado en preguntas anteriores.

Tabla 59
Respuestas obtenidas a la pregunta de si se pueden encajar las siguientes piezas [3. no - ir a x: y:]

Opciones	Nº de respuestas	Porcentaje de respuestas
Sí	26	30,95%
No	58	69,05%
TOTAL	84	100%

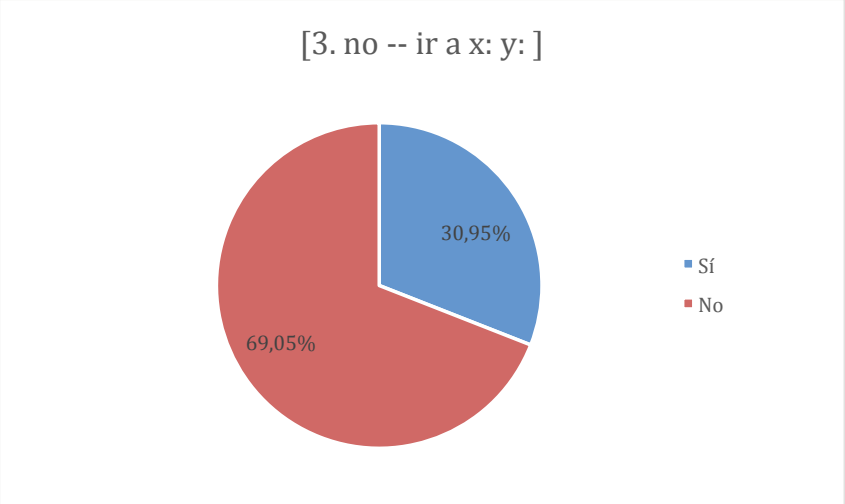


Figura 76. Gráfica de las respuestas obtenidas a la pregunta relativa a si se pueden encajar las siguientes piezas [3. no - ir a x: y:]

Fuente: Elaboración propia.

10.3.6.7.10. Análisis bivariable: respuestas a si se pueden encajar las siguientes piezas [3. no - ir a x: y:], y clases recibidas

Según se ve en la Tabla 60 y en la *Figura 77*, los porcentajes entre los que han respondido correctamente y los que no lo han hecho, en relación al número de clases recibidas, está muy equilibrado, con lo que no se aprecia relación entre estas dos variables.

Tabla 60
Respuestas obtenidas a la pregunta de si se pueden encajar las siguientes piezas [3. no - ir a x: y:], *y su relación con el número de clases recibidas previamente*

	Sí		**No**		**TOTAL**	
Clases recibidas	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**
Más clases que la media (Mayor)	13	34,21%	25	65,79%	38	100%
Menos clases que la media (Menor)	13	28,26%	33	71,74%	46	100%
TOTAL	26	30,95%	58	69,05%	84	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla.

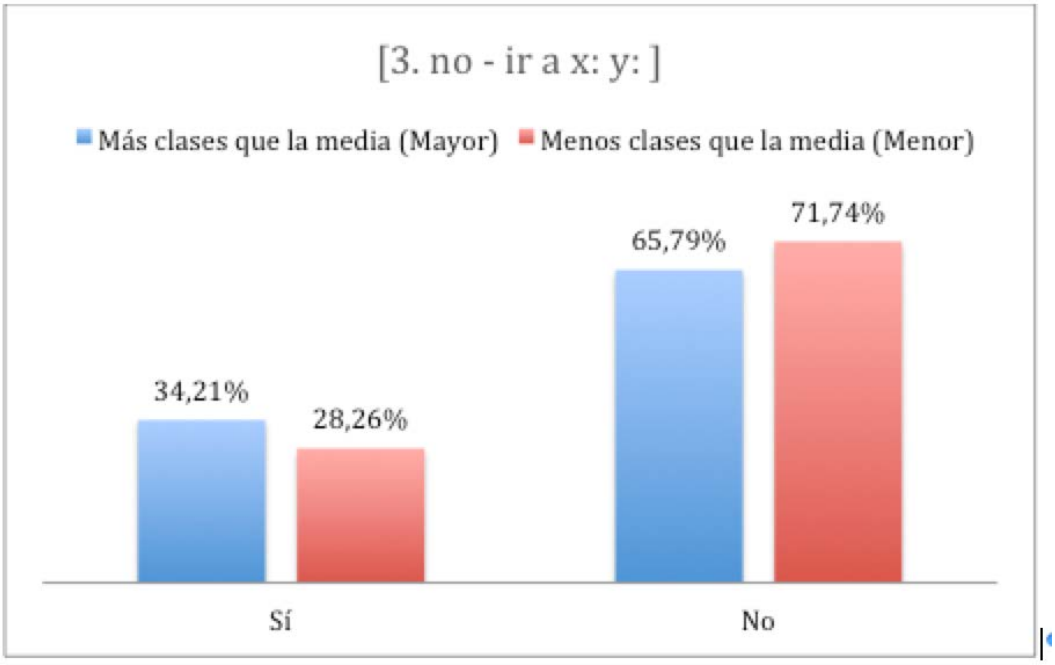


Figura 77. Gráfica de las respuestas obtenidas a la pregunta relativa a si se pueden encajar las siguientes piezas [3. no - ir a x: y:], y su relación con el número de clases recibidas previamente.

Fuente: Elaboración propia.

10.3.6.7.11. Análisis bivariable: respuestas a si se pueden encajar las siguientes piezas [3. no - ir a x: y:] y práctica autónoma

En la Tabla 61 y en la *Figura 78* se puede ver que entre los que respondieron correctamente, hay mayoría entre los que sí tuvieron práctica autónoma.

Tabla 61
Respuestas obtenidas a la pregunta de si se pueden encajar las siguientes piezas [3. no - ir a x: y:], y su relación con la práctica autónoma.

	Sí		No		TOTAL	
Práctica autónoma	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**
Sí	14	24,56%	43	75,44%	57	100%
No	12	46,15%	14	53,85%	26	100%
TOTAL	26	31,33%	57	68,67%	83	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla.

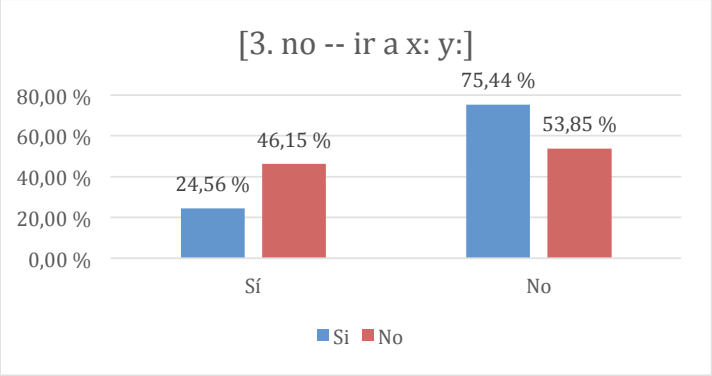


Figura 78. Gráfica de las respuestas obtenidas a la pregunta relativa a si se pueden encajar las siguientes piezas [3. no - ir a x: y:], y su relación con la práctica autónoma.

Fuente: Elaboración propia.

10.3.6.7.12. Análisis bivariable: respuestas a si se pueden encajar las siguientes piezas [3. no - ir a x: y:] y hábitos de uso del ordenador

Por los datos expuestos en la Tabla 62, y su representación en la *Figura 79*, se puede ver que no hay indicios que apunten a la relación entre estas dos variables.

Tabla 62
Respuestas obtenidas a la pregunta de si se puede encajar las siguientes piezas [3. no - ir a x: y:], *y su relación con los hábitos de uso del ordenador*.

	Sí		No		TOTAL	
Hábitos de uso del ordenador	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**
Tengo un ordenador en casa y lo utilizo habitualmente (todos o casi todos los días)	10	31,25%	22	68,75%	32	100%
Tengo un ordenador en casa, pero lo utilizo poco (una o dos veces por semana o menos)	12	28,57%	30	71,43%	42	100%
En mi casa no hay ordenador, pero utilizo el del colegio, o el de un familiar o amigo, habitualmente (todos o casi todos los días)	1	50%	1	50%	2	100%
En mi casa no hay ordenador, pero utilizo el del colegio, o el de un familiar o amigo, de vez en cuando (una o dos veces por semana o menos)	1	50%	1	50%	2	100%
No suelo utilizar ningún ordenador	2	33,33%	4	66,67%	6	100%
TOTAL	26	30,95%	58	69,05%	84	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla.

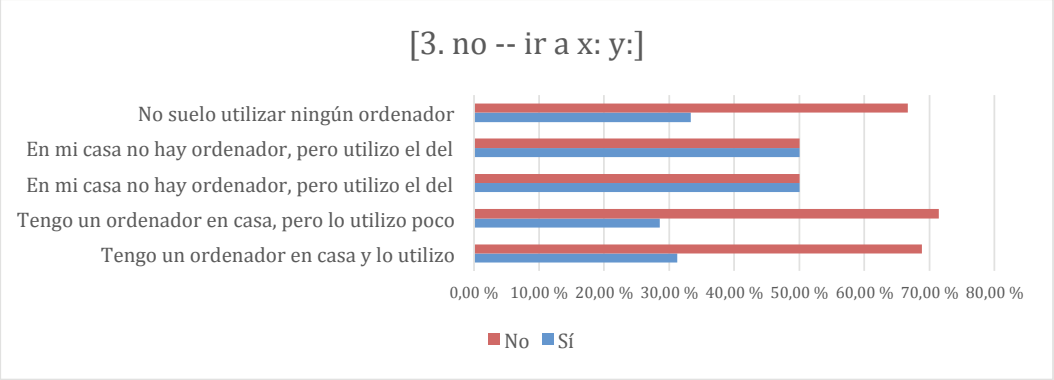


Figura 79. Gráfica de las respuestas obtenidas a la pregunta relativa a si se pueden encajar las siguientes piezas [3. no - ir a x: y:], y su relación con los hábitos de uso del ordenador.

Fuente: Elaboración propia.

10.3.6.7.13. Análisis univariable: respuestas a si se pueden encajar las siguientes piezas [4. si entonces - ¿tocando?]

En esta pregunta, la respuesta correcta era sí, dado que las piezas mencionadas sí se pueden combinar. En la Tabla 63 y en la *Figura 80* se puede apreciar que el 84,15% responde correctamente.

Tabla 63
Respuestas obtenidas a la pregunta de si se pueden encajar las siguientes piezas [4. si entonces - ¿tocando?]

Opciones	Nº de respuestas	Porcentaje de respuestas
Sí	69	84,15%
No	13	15,85%
TOTAL	82	100%

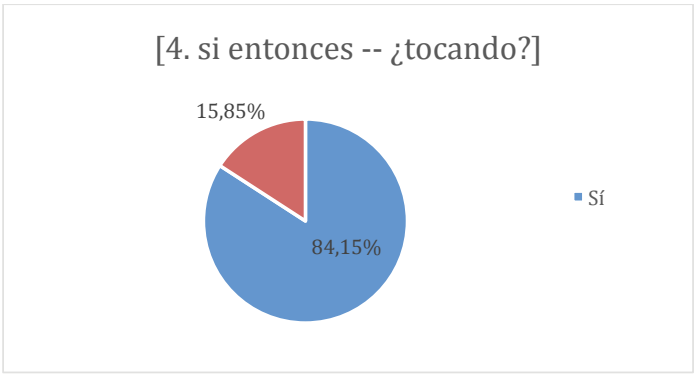


Figura 80. Gráfica de las respuestas obtenidas a la pregunta relativa a si se pueden encajar las siguientes piezas [4. si entonces - ¿tocando?]

Fuente: Elaboración propia.

10.3.6.7.14. Análisis bivariable: respuestas a si se pueden encajar las siguientes piezas [4. si entonces - ¿tocando?] y clases recibidas

En la Tabla 64 y en la *Figura 81* se puede ver que los porcentajes entre los que han respondido tanto sí como no, y la relación con el número de clases recibidas, es prácticamente el mismo. Por tanto, no se puede afirmar que exista relación entre estas dos variables.

Tabla 64
Respuestas obtenidas a la pregunta de si se pueden encajar las siguientes piezas [4. si entonces - ¿tocando?], *y su relación con el número de clases recibidas previamente*.

Clases recibidas	**Sí**		**No**		**TOTAL**	
	Nº	**%**	**Nº**	**%**	**Nº**	**%**
Más clases que la media (Mayor)	32	84,21%	6	15,79%	38	100%
Menos clases que la media (Menor)	37	84,09%	7	15,91%	44	100%
TOTAL	69	84,15%	13	15,85%	82	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla.

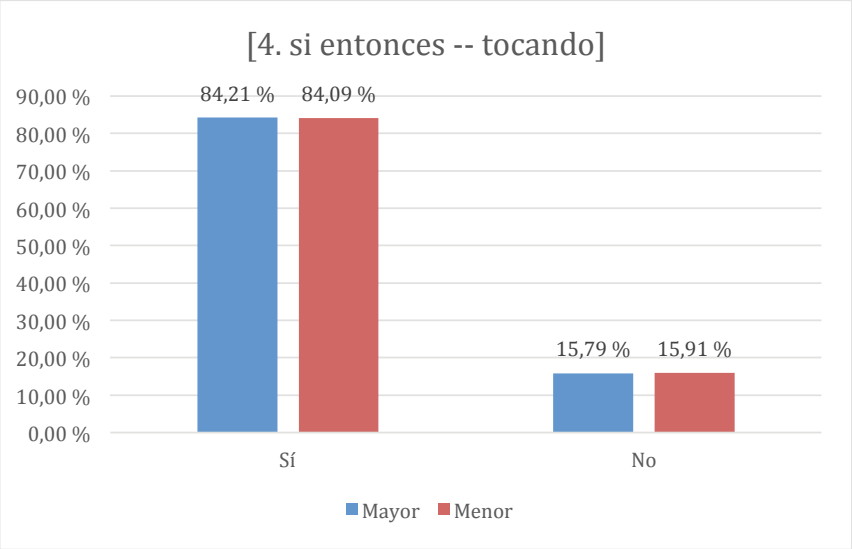


Figura 81. Gráfica de las respuestas obtenidas a la pregunta relativa a si se pueden encajar las siguientes piezas [4. si entonces - ¿tocando?], y su relación con el número de clases recibidas previamente.

Fuente: Elaboración propia.

10.3.6.7.15. Análisis bivariable: respuestas a si se pueden encajar las siguientes piezas [4. si entonces - ¿tocando?] y práctica autónoma

En los datos mostrados por la Tabla 65 y la *Figura 82*, los participantes que sí tuvieron práctica autónoma, son mayoría entre los que aciertan, y minoría entre los que fallan.

Tabla 65
Respuestas obtenidas a la pregunta de si se pueden encajar las siguientes piezas [4. si entonces - ¿tocando?], *y su relación con la práctica autónoma*.

	Sí		No		TOTAL	
Práctica autónoma	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**
Sí	50	89,29%	6	10,71%	56	100%
No	18	72%	7	28%	25	100%
TOTAL	68	83,95%	13	16,05%	81	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla.

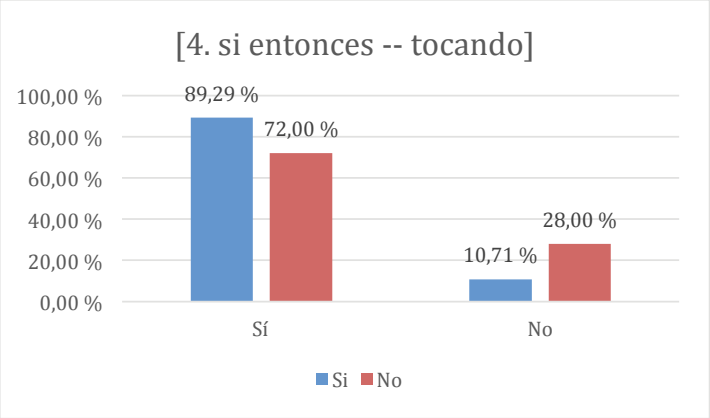


Figura 82. Gráfica de las respuestas obtenidas a la pregunta relativa a si se pueden encajar las siguientes piezas [4. si entonces - ¿tocando?], y su relación con la práctica autónoma.

Fuente: Elaboración propia.

10.3.6.7.16. Análisis bivariable: respuestas a si se pueden encajar las siguientes piezas [4. si entonces - ¿tocando? y hábitos de uso del ordenador

En la Tabla 66 y en la *Figura 83* se puede ver que hay un mayor porcentaje de estudiantes que han fallado en la respuesta, dentro del grupo de los que no utilizan el ordenador, o que lo utilizan poco. En cualquier caso, este dato no es significativo, pues de los 82 participantes que han respondido a la pregunta, solo 8 reúnen estas características.

Tabla 66
Respuestas obtenidas a la pregunta de si se pueden encajar las siguientes piezas [4. si entonces - ¿tocando?], y su relación con los hábitos de uso del ordenador.

	Sí		No		TOTAL	
Hábitos de uso del ordenador	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**
Tengo un ordenador en casa y lo utilizo habitualmente (todos o casi todos los días)	24	80%	6	20%	30	100%
Tengo un ordenador en casa, pero lo utilizo poco (una o dos veces por semana o menos)	39	92,86%	3	7,14%	42	100%
En mi casa no hay ordenador, pero utilizo el del colegio, o el de un familiar o amigo, habitualmente (todos o casi todos los días)	2	100%	0	0%	2	100%
En mi casa no hay ordenador, pero utilizo el del colegio, o el de un familiar o amigo, de vez en cuando (una o dos veces por semana o menos)	1	50%	1	50%	2	100%
No suelo utilizar ningún ordenador	3	50%	3	50%	6	100%
TOTAL	69	84,15%	13	15,85%	82	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla.

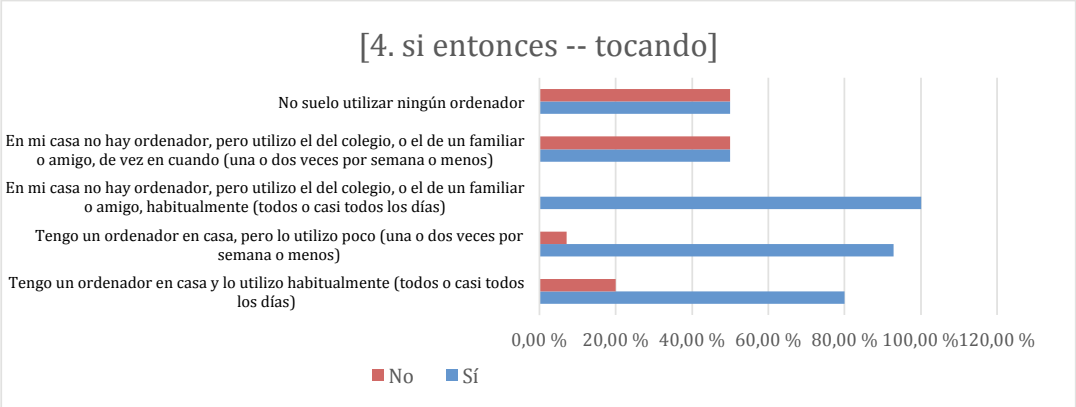


Figura 83. Gráfica de las respuestas obtenidas a la pregunta relativa a si se pueden encajar las siguientes piezas [4. si entonces - ¿tocando?], y su relación con los hábitos de uso del ordenador.

Fuente: Elaboración propia.

10.3.6.7.17. Conclusiones globales a la pregunta "¿Sabrías decirme si se pueden encajar las siguientes piezas? "

En general una gran mayoría de los participantes han respondido correctamente a la Pregunta 10, que cuestionaba sobre ciertos bloques que, en según qué casos, se podían o no combinar, según la estructura de programación que representara. A tenor de los datos, parece que esta lógica se comprende. Además, se refuerza lo que ya apuntaba la Pregunta 9, donde también se concluía que en general los participantes parecían entender la lógica de combinación de las distintas piezas de Scratch con las que se elaboran los programas.

Asimismo, esta pregunta está entre las que tratan de evaluar la comprensión de ciertos conceptos computacionales básicos, en este caso bucles y sentencias condicionales. De momento las conclusiones no son determinantes, pero se puede afirmar la mayoría de los participantes han sabido identificar las piezas que se pueden y que no se pueden combinar con el bloque "si entonces", que representa una sentencia condicional, y con "repetir", que representa un bucle.

10.3.6.8. ¿En algún momento algo no funcionaba como querías?

10.3.6.8.1. Análisis univariable: si en algún momento algo no funcionaba

En esta pregunta, enfocada a evaluar los posibles errores que los participantes hayan podido encontrar en el desarrollo de su juego, podemos comprobar que una gran mayoría han tenido que corregir alguna cosa, y solo un 17,48% han conseguido completarlo todo a la primera. Así lo reflejan tanto la Tabla 67 como la *Figura 84*.

Tabla 67
Respuestas obtenidas a la pregunta de si en algún momento algo no funcionaba.

Opciones	Nº de respuestas	Porcentajes de respuestas
Todo ha ido bien de principio a fin	18	17,48%
He tenido que corregir algunas cosas	85	82,52%
TOTAL	103	100%

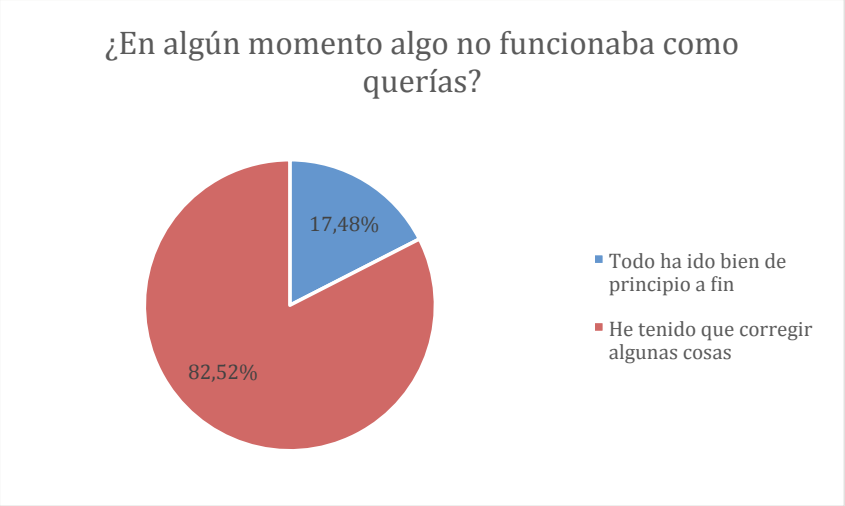


Figura 84. Gráfica de las respuestas obtenidas a la pregunta relativa a si en algún momento algo no funcionaba.

Fuente: Elaboración propia.

10.3.6.8.2. Análisis bivariable: si en algún momento algo no funcionaba y clases recibidas

En la Tabla 68 y en la *Figura 85* se puede ver que no hay relación entre el hecho de que algo no funcionara como el participante quería, y el número de clases recibidas.

Tabla 68
Respuestas obtenidas a la pregunta de si en algún momento algo no funcionaba, y su relación con el número de clases recibidas previamente.

Clases recibidas	Todo ha ido bien de principio a fin		He tenido que corregir algunas cosas		TOTAL	
	Nº	%	Nº	%	Nº	%
Más clases que la media (Mayor)	9	18,75%	39	81,25%	48	100%
Menos clases que la media (Menor)	9	16,36%	46	83,64%	55	100%
TOTAL	18	17,48%	85	82,52%	103	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla.

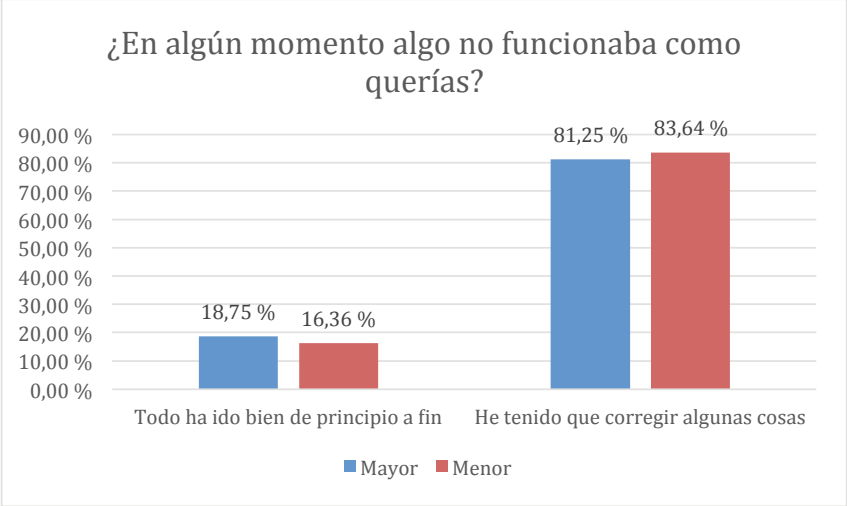


Figura 85. Gráfica de las respuestas obtenidas a la pregunta relativa a si en algún momento algo no funcionaba, y su relación con el número de clases recibidas previamente.

Fuente: Elaboración propia.

10.3.6.8.3. Análisis bivariable: si en algún momento algo no funcionaba y práctica autónoma

En la Tabla 69 y en la *Figura 86* se puede ver que no hay relación entre el hecho de que algo no funcionara como el participante quería, y la práctica autónoma que el alumno haya podido tener con Scratch previamente al desarrollo de la actividad.

Tabla 69
Respuestas obtenidas a la pregunta de si en algún momento algo no funcionaba, y su relación con la práctica autónoma.

Práctica autónoma	**Todo ha ido bien de principio a fin**		**He tenido que corregir algunas cosas**		**TOTAL**	
	Nº	**%**	**Nº**	**%**	**Nº**	**%**
Sí	12	17,39%	57	82,61%	69	100%
No	6	18,18%	27	81,82%	33	100%
TOTAL	18	17,65%	84	82,35%	102	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla.

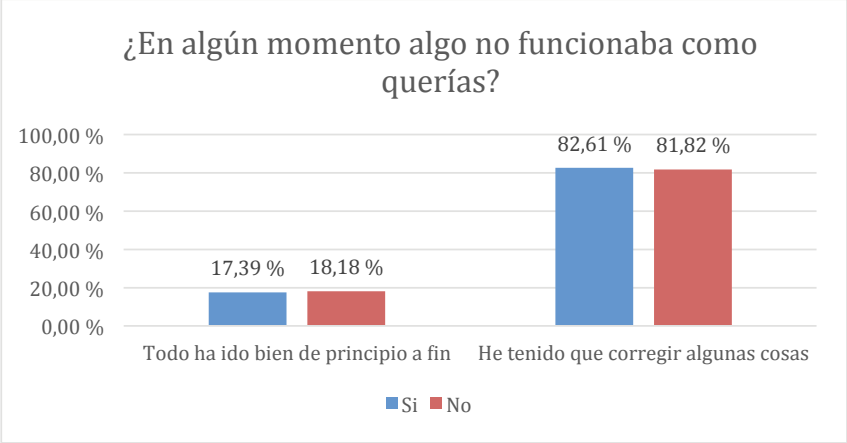


Figura 86. Gráfica de las respuestas obtenidas a la pregunta relativa a si en algún momento algo no funcionaba, y su relación con la práctica autónoma.

Fuente: Elaboración propia.

10.3.6.8.4. Análisis bivariable: si en algún momento algo no funcionaba y hábitos de uso del ordenador

Como en ocasiones anteriores, en los datos que muestran la Tabla 70 y la *Figura 87* no se aprecian indicios de relación de las respuestas obtenidas y los hábitos de uso del ordenador por parte de los participantes. La tendencia mayoritaria a tener que corregir alguna cosa se mantiene, independientemente de que se disponga de ordenador.

Tabla 70
Respuestas obtenidas a la pregunta de si en algún momento algo no funcionaba, y su relación con los hábitos de uso del ordenador.

	Todo ha ido bien de principio a fin		**He tenido que corregir algunas cosas**		**TOTAL**	
Hábitos de uso del ordenador	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**
Tengo un ordenador en casa y lo utilizo habitualmente (todos o casi todos los días)	3	8,57%	32	91,43%	35	100%
Tengo un ordenador en casa, pero lo utilizo poco (una o dos veces por semana o menos)	8	16%	42	84%	50	100%
En mi casa no hay ordenador, pero utilizo el del colegio, o el de un familiar o amigo, habitualmente (todos o casi todos los días)	4	66,67%	2	33,33%	6	100%

	Todo ha ido bien de principio a fin		He tenido que corregir algunas cosas		TOTAL	
Hábitos de uso del ordenador	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**
En mi casa no hay ordenador, pero utilizo el del colegio, o el de un familiar o amigo, de vez en cuando (una o dos veces por semana o menos)	1	33,33%	2	66,67%	3	100%
No suelo utilizar ningún ordenador	2	22,22%	7	77,78%	9	100%
TOTAL	18	17,48%	85	82,52%	103	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla.

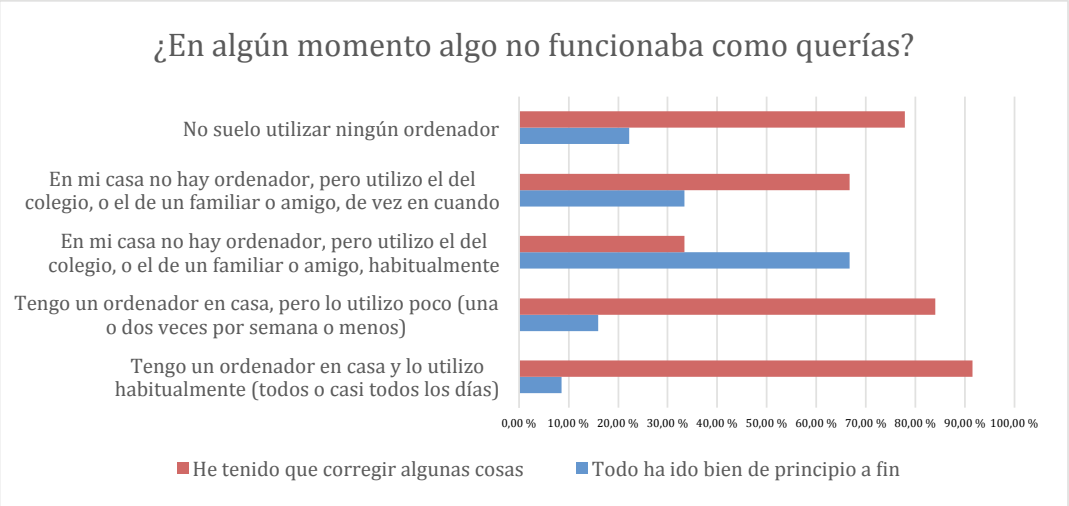


Figura 87. Gráfica de las respuestas obtenidas a la pregunta relativa a si en algún momento algo no funcionaba, y su relación con los hábitos de uso del ordenador.

Fuente: Elaboración propia.

10.3.6.8.5. Conclusiones globales a la pregunta "¿En algún momento algo no funcionaba como tú querías? "

La Pregunta 11 pretende evaluar en qué medida los participantes se han encontrado con problemas en la realización de su juego, y se relaciona con las pautas NNGG5, NNGG7, HHS1.8 , MIT3 y NOKIA5.

La pauta NNGG5 evalúa los mecanismos de la prevención de errores que tiene el sistema, y la NNGG7 sobre los procesos que tienen que memorizar los usuarios. La mayoría de los encuestados, (un 82%) han tenido que corregir algo. De momento esto solo apunta a que es frecuente que los procesos no se resuelvan a la primera.

En preguntas posteriores se podrá tener más información sobre cómo se solucionan los problemas que surgen.

La pauta HHS1.8 evalúa si la aplicación cumple con su propósito. De nuevo los datos no arrojan indicios determinantes a este respecto.

La pauta MIT3 cuestiona si el sistema refleja el flujo de trabajo del usuario. Ya en el análisis heurístico se arrojaban dudas sobre su cumplimiento, y si a esto le añadimos que la mayoría de los participantes han encontrado problemas en la realización de sus tareas, podemos concluir que parece ser que efectivamente hay aspectos que mejorar con respecto a la información sobre el flujo de trabajo que ofrece Scratch.

Con respecto a la pauta NOKIA5, que evalúa el reto que supone el manejo de la herramienta, parece que efectivamente se cumple, los datos señalan que Scratch no es un programa que se domine rápidamente.

10.3.6.9. Cuando algo no funcionaba como tú querías…

10.3.6.9.1. Análisis univariable: comportamiento del participante cuando algo no funcionaba

Esta pregunta pretende medir el grado de ayuda que necesitaron los participantes cuando surgieron problemas. Según los datos de la Tabla 71 y en la *Figura 88*, son mayoría los que no han podido resolver el problema de forma autónoma, y han tenido que recurrir a la ayuda del profesor.

Tabla 71
Respuestas obtenidas a la pregunta relativa al comportamiento del participante cuando algo no funcionaba.

Opciones	Nº de respuestas	Porcentajes de respuestas
Lo he resuelto yo solo sin ninguna ayuda	19	23,17%
He utilizado la ayuda del programa	11	13,41%
He pedido ayuda al profesor y hemos utilizado la ayuda del programa	26	31,71%
He pedido ayuda al profesor y me lo ha resuelto	26	31,71%
TOTAL	82	100%

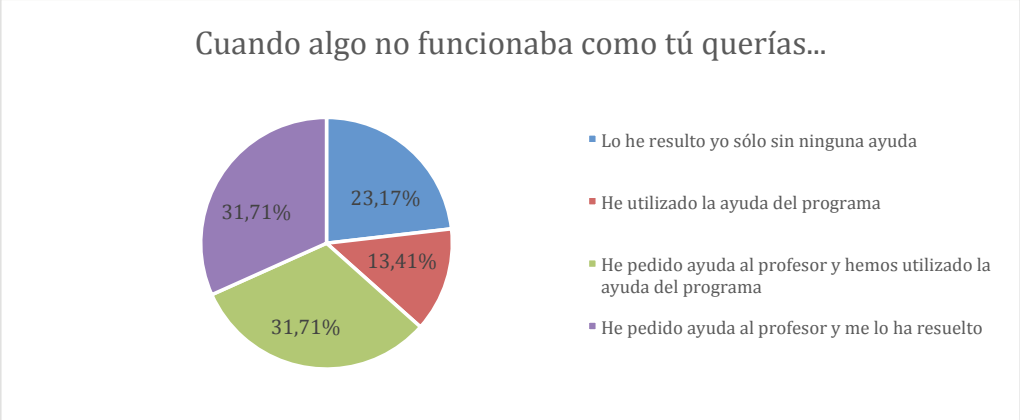


Figura 88. Gráfica de las respuestas obtenidas a la pregunta relativa al comportamiento del participante cuando algo no funcionaba.

Fuente: Elaboración propia.

10.3.6.9.2. Análisis bivariable: comportamiento del participante cuando algo no funcionaba y clases recibidas

En la Tabla 72 y en la *Figura 89* no se pueden apreciar indicios significativos de que haya relación entre el número de clases recibidas y el grado de ayuda que precisan los participantes para resolver un problema, porque hay dispersión de los datos.

Tabla 72
Respuestas obtenidas a la pregunta del comportamiento del participante cuando algo no funcionaba, y su relación con el número de clases recibidas previamente.

	Lo he resuelto yo solo sin ninguna ayuda		**He utilizado la ayuda del programa**		**He pedido ayuda al profesor y hemos utilizado la ayuda del programa**		**He pedido ayuda al profesor y me lo ha resuelto**		**TOTAL**	
Clases recibidas	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**
Más clases que la media (Mayor)	8	21,62%	5	13,51%	8	21,62%	16	43,24%	37	100%
Menos clases que la media (Menor)	11	24,44%	6	13,33%	18	40%	10	22,22%	45	100%
TOTAL	19	23,17%	11	13,41%	26	31,71%	26	31,71%	82	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla.

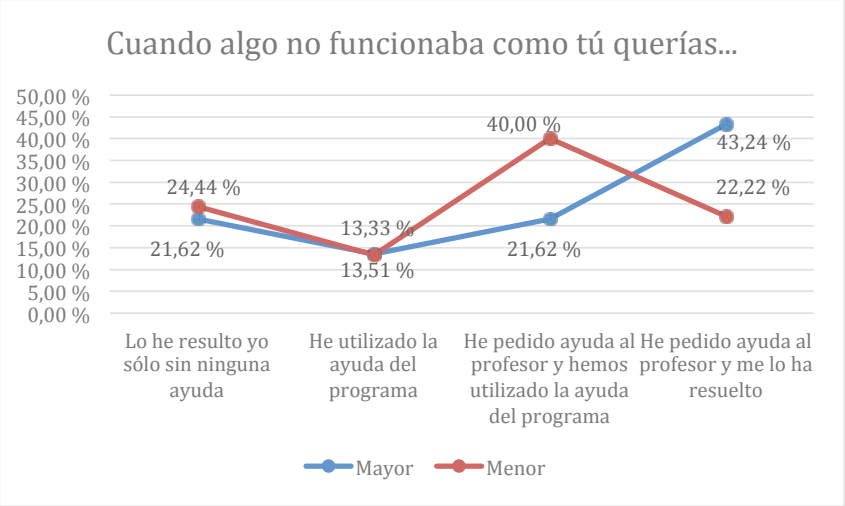


Figura 89. Gráfica de las respuestas obtenidas a la pregunta relativa al comportamiento del participante cuando algo que no funcionaba, y su relación con el número de clases recibidas previamente.

Fuente: Elaboración propia.

10.3.6.9.3. Análisis bivariable: comportamiento del participante cuando algo no funcionaba, y práctica autónoma

En la Tabla 73 y en la *Figura 90* no muestran una relación clara entre comportamiento del participante cuando algo no funcionaba, y la práctica autónoma. Hay cierta tendencia a que los que tienen práctica autónoma tienen más facilidad para resolver el problema.

Tabla 73

Respuestas obtenidas a la pregunta relativa al comportamiento del participante cuando algo no funcionaba, y su relación con la práctica autónoma.

	Lo he resuelto yo solo sin ninguna ayuda		**He utilizado la ayuda del programa**		**He pedido ayuda al profesor y hemos utilizado la ayuda del programa**		**He pedido ayuda al profesor y me lo ha resuelto**		**TOTAL**	
Práctica autónoma	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**
Sí	16	29,09%	8	14,55%	16	29,09%	15	27,27%	55	100%
No	3	11,54%	3	11,54%	10	38,46%	10	38,46%	26	100%
TOTAL	19	23,46%	11	13,58%	26	32,10%	25	30,86%	81	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla.

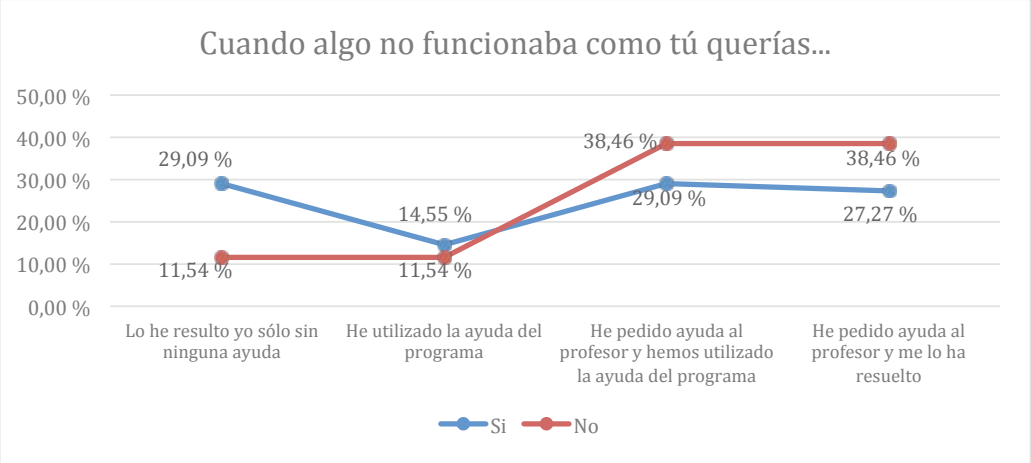


Figura 90. Gráfica de las respuestas obtenidas a la pregunta relativa al comportamiento del participante cuando algo no funcionaba, y su relación con la práctica autónoma.

Fuente: Elaboración propia.

10.3.6.9.4. Análisis bivariable: comportamiento del participante cuando algo no funcionaba, y hábitos de uso del ordenador

En la Tabla 74 y en la *Figura 91* se ve que hay cierta tendencia a que los que resuelven solos sin ninguna ayuda se concentren entre los que más utilizan el ordenador.

Tabla 74
Respuestas obtenidas a la pregunta de cuando algo no funciona, y su relación con los hábitos de uso del ordenador.

	Lo he resuelto yo solo sin ninguna ayuda		**He utilizado la ayuda del programa**		**He pedido ayuda al profesor y hemos utilizado la ayuda del programa**		**He pedido ayuda al profesor y me lo ha resuelto**		**TOTAL**	
Hábitos de uso del ordenador	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**
Tengo un ordenador en casa y lo utilizo habitualmente (todos o casi todos los días)	10	32,26%	3	9,68%	10	32,26%	8	25,81%	31	100%
Tengo un ordenador en casa, pero lo utilizo poco (una o dos veces por semana o menos)	8	19,51%	7	17,07%	11	26,83%	15	36,59%	41	100%
En mi casa no hay ordenador, pero utilizo el del colegio, o el de un familiar o amigo, habitualmente (todos o casi todos los días)	1	50%	0	0%	1	50%	0	0%	2	100%

Hábitos de uso del ordenador	Lo he resuelto yo solo sin ninguna ayuda		He utilizado la ayuda del programa		He pedido ayuda al profesor y hemos utilizado la ayuda del programa		He pedido ayuda al profesor y me lo ha resuelto		TOTAL	
	Nº	%	Nº	%	Nº	%	Nº	%	Nº	%
En mi casa no hay ordenador, pero utilizo el del colegio, o el de un familiar o amigo, de vez en cuando (una o dos veces por semana o menos)	0	0%	1	50%	1	50%	0	0%	2	100%
No suelo utilizar ningún ordenador.	0	0%	0	0%	3	50%	3	50%	6	100%
TOTAL	19	23,17%	11	13,41%	26	31,71%	26	31,71%	82	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla.

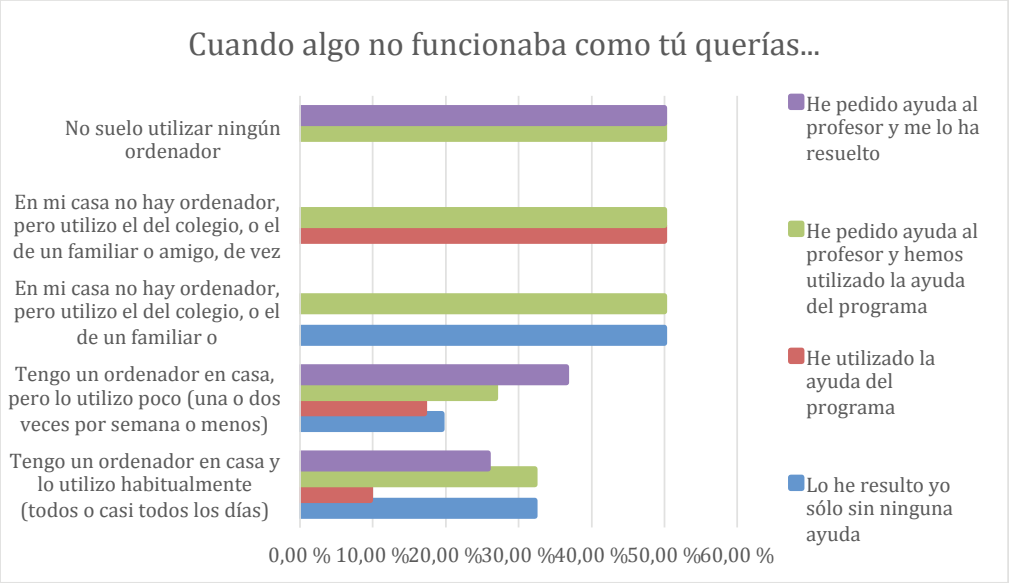


Figura 91. Gráfica de las respuestas obtenidas a la pregunta relativa a cuando algo no funcionaba como se quería, y su relación con los hábitos de uso del ordenador.
Fuente: Elaboración propia.

10.3.6.10. ¿Al final has conseguido que el programa funcione como tú querías o has dejado algún problema sin resolver?

10.3.6.10.1. Análisis univariable: si se ha conseguido que el programa funcione o si se ha dejado algún problema sin resolver

Los datos que muestran la Tabla 75 y la *Figura 92* indican que la mayoría (un 64,29%) han conseguido que el programa funcione como ellos querían. No hay ningún participante de los que respondieron a la pregunta, al que no le funcionara el programa.

Tabla 75
Respuestas obtenidas a la pregunta de si al final se ha conseguido que el programa funcione o si se ha dejado algún problema sin resolver.

Opciones	Nº de respuestas	Porcentaje de respuestas
Sí, funciona perfectamente	54	64,29%
He resuelto más de la mitad de los problemas	25	29,76%
He resuelto menos de la mitad de los problemas	5	5,95%
No funciona	0	0%
TOTAL	84	100%

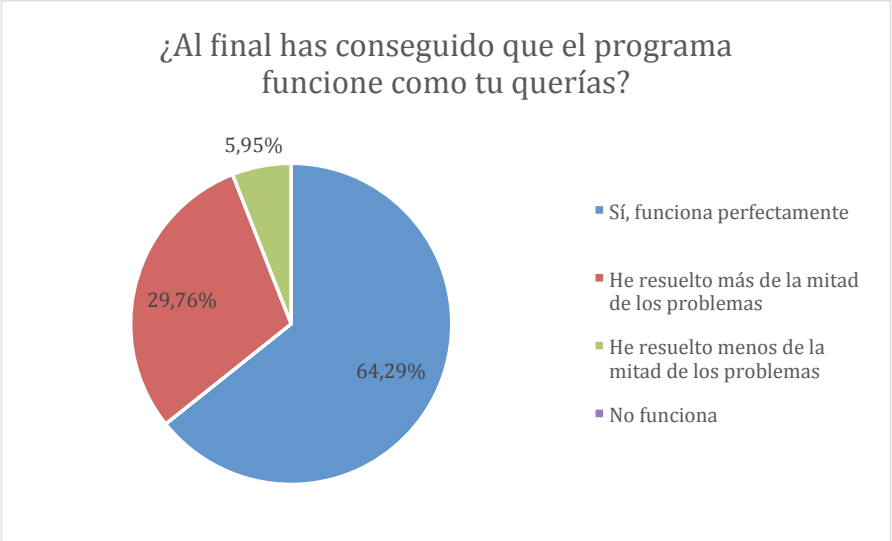


Figura 92. Gráfica de las respuestas obtenidas a la pregunta relativa a si al final se ha conseguido que el programa funcione o si se ha dejado algún problema sin resolver.

Fuente: Elaboración propia.

10.3.6.10.2. Análisis bivariable: si se ha conseguido que el programa funcione o si se ha dejado algún problema sin resolver, y clases recibidas

En la Tabla 76 y en la *Figura 93* se muestra que los porcentajes de los que han recibido más clases que la media, y los que han recibido menos, van prácticamente en paralelo. No se aprecia relación entre estas dos variables.

Tabla 76
Respuestas obtenidas a la pregunta de si se ha conseguido que el programa funcione o si se ha dejado algún problema sin resolver, y su relación con el número de clases recibidas previamente.

	Sí, funciona perfectamente		**He resuelto más de la mitad de los problemas**		**He resuelto menos de la mitad de los problemas**		**No funciona**		**TOTAL**	
Clases recibida	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**
Más clases que la media (Mayor)	24	63,16%	10	26,32%	4	10,53%	0	0%	38	100%
Menos clases que la media (Menor)	30	65,22%	15	32,61%	1	2,17%	0	0%	46	100%
TOTAL	54	64,29%	25	29,76%	5	5,95%	0	0%	84	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla.

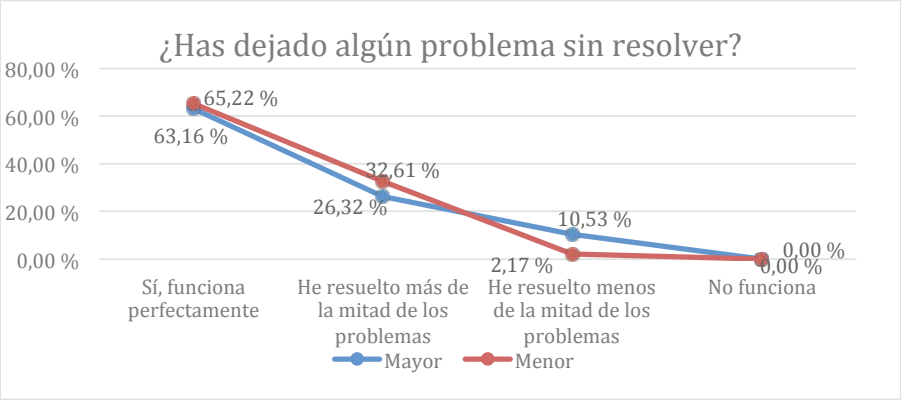


Figura 93. Gráfica de las respuestas obtenidas a la pregunta relativa a si ha conseguido que el programa funcione o si se ha dejado algún problema sin resolver, y su relación con el número de clases recibidas previamente.

Fuente: Elaboración propia.

10.3.6.10.3. Análisis bivariable: si se ha conseguido que el programa funcione o si se ha dejado algún problema sin resolver y práctica autónoma

En la Tabla 77 y en la *Figura 94* podemos apreciar algo similar a lo anterior, los porcentajes asociados a los participantes que sí han tenido práctica autónoma, y los que no la han tenido, son muy similares. En la gráfica se puede ver claramente estos comportamientos paralelos. Por tanto, no se puede afirmar que exista relación entre estas dos variables.

Tabla 77

Respuestas obtenidas a la pregunta de si se ha conseguido que el programa funcione o si se ha dejado algún problema sin resolver, y su relación con la práctica autónoma.

	Lo he resuelto yo solo sin ninguna ayuda		**He utilizado la ayuda del programa**		**He pedido ayuda al profesor y hemos utilizado la ayuda del programa**		**He pedido ayuda al profesor y me lo ha resuelto**		**TOTAL**	
Práctica autónoma	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**
Sí	39	68,42%	18	31,58%	0	0%	0	0%	57	100%
No	14	53,85%	7	26,92%	5	19,23%	0	0%	26	100%
TOTAL	53	63,86%	25	30,12%	5	6,02%	0	0%	83	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla

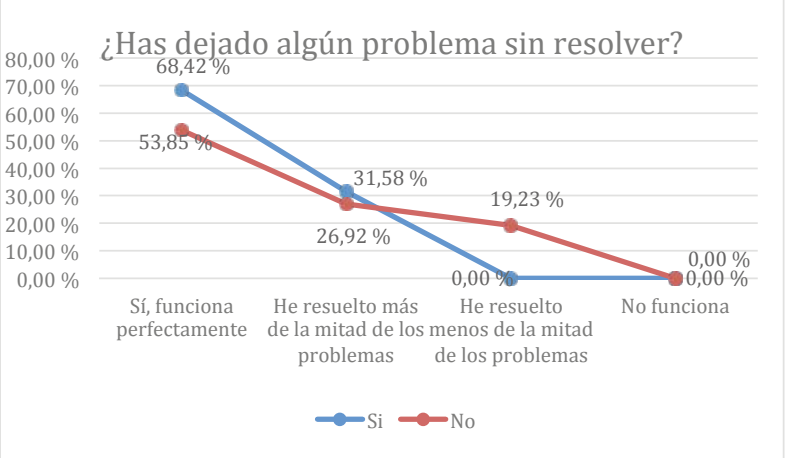


Figura 94. Gráfica de las respuestas obtenidas a la pregunta relativa a si ha conseguido que el programa funcione o si se ha dejado algún problema sin resolver, y su relación con la práctica autónoma.

Fuente: Elaboración propia.

10.3.6.10.4. Análisis bivariable: si se ha conseguido que el programa funcione o si se ha dejado algún problema sin resolver y hábitos de uso del ordenador

De lo que se puede observar en la Tabla 78 y en la *Figura 95*, lo más destacable es que hay cierta tendencia, aunque no determinante, a que los participantes que han resuelto menos problemas sean también los que menos utilizan el ordenador.

Tabla 78
Respuestas obtenidas a la pregunta de si se ha conseguido que el programa funcione o si se ha dejado algún problema sin resolver, y su relación con los hábitos de uso del ordenador.

	Sí, funciona perfectamente		**He resuelto más de la mitad de los problemas**		**He resuelto menos de la mitad de los problemas**		**No funciona**		**TOTAL**	
Hábitos de uso del ordenador	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**
Tengo un ordenador en casa y lo utilizo habitualmente (todos o casi todos los días)	22	68,75%	9	28,13%	1	3,13%	0	0%	32	100%
Tengo un ordenador en casa, pero lo utilizo poco (una o dos veces por semana o menos)	27	64,29%	13	30,95%	2	4,76%	0	0%	42	100%
En mi casa no hay ordenador, pero utilizo el del colegio, o el de un familiar o amigo, habitualmente (todos o casi todos los días)	1	50%	0	0%	1	50%	0	0%	2	100%
En mi casa no hay ordenador, pero utilizo el del colegio, o el de un familiar o amigo, de vez en cuando (una o dos veces por semana o menos)	1	50%	1	50%	0	0%	0	0%	2	100%
No suelo utilizar ningún ordenador.	3	50%	2	33,33%	1	16,67%	0	0%	6	100%
TOTAL	54	64,29%	25	29,76%	5	5,95%	0	0%	84	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla.

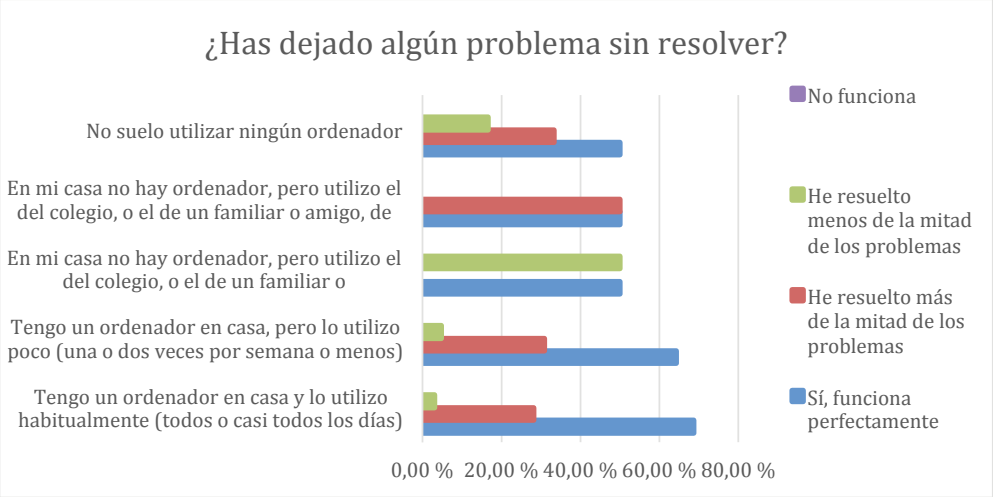


Figura 95. Gráfica de las respuestas obtenidas a la pregunta relativa a si se ha conseguido que el programa funcione o si se ha dejado algún problema sin resolver, y su relación con los hábitos de uso del ordenador.

Fuente: Elaboración propia.

10.3.6.10.5. Conclusiones a la pregunta "¿Al final has conseguido que el programa funcione como tú querías o has dejado algún problema sin resolver?"

La Pregunta 14 solo se realizó a los participantes que previamente habían respondido que habían tenido que corregir alguna cosa en la Pregunta 11. Esta pregunta complementa a la Pregunta 13 y pretende determinar en qué medida se resuelven los problemas que surgen. Las pautas del análisis heurístico que dan origen a esta pregunta son NNGG5, NNGG7, HHS2.7, HHS2.13, MIT4, NOKIA6 y AGUC3.

Las pautas NNGG5 y NNGG7 están relacionadas con la prevención de errores y la eficiencia de uso respectivamente. No hay ningún encuestado al que no le funcionara el programa, y solo un 5,95% resolvió menos de la mitad de sus problemas, con lo que estas dos pautas se cumplen.

Las pautas HHS2.7 y HHS2.13 tienen que ver con la sencillez de los mensajes que muestra el programa, y la documentación que ofrece, y la pauta AGUC3 con la facilidad para encontrar lo que se busca. Los datos recogidos en esta pregunta no son definitivos con respecto a estas pautas, que también se evalúan en otras preguntas, pero el hecho de que la mayoría de los participantes resolvieran sus problemas es un argumento más para indicar que se cumplen.

La pauta MIT4 tiene que ver con la capacidad del usuario para deshacer una acción. Una vez más, que la mayoría puedan resolver sus problemas apunta a que esta pauta se cumple.

La pauta NOKIA6 tiene que ver con las recompensas que se le ofrece al usuario. Para que algo sea divertido, la dificultad que suponga llevarlo a cabo debe estar equilibrada. Si es muy fácil, no supondrá un reto y será aburrido. Si es muy difícil, frustrará al usuario (Koster, 2013). Por las respuestas a esta pregunta, vemos que el manejo de Scratch no muestra signos de resultar frustrante, si para llegar a esta conclusión tomamos el escaso porcentaje de participantes (5,95%) que resuelven menos de la mitad de sus problemas.

Por último, indicar que en el análisis bivariable no se ha encontrado una relación evidente entre la pregunta y las variables de número de clases recibidas, hábitos de uso del ordenador, y práctica autónoma

10.3.6.11. ¿Había algunas palabras en Scratch que no entendías lo que significaban?

10.3.6.11.1. Análisis univariable: palabras presentes en Scratch que no se han entendido

Esta pregunta está encaminada a analizar el lenguaje que utiliza Scratch. Los datos que se muestran en la Tabla 79 y que están representados en la *Figura 96* indican que una mayoría (61,17%) han respondido que entienden todas las palabras. Para verificar que esto es así se realizan otras preguntas en el cuestionario.

Tabla 79
Respuestas obtenidas a la pregunta relativa a si había palabras presentes en Scratch que no se entendía su significado.

Opciones	Nº de respuestas	Porcentaje de respuestas
He entendido todas las palabras a la primera	63	61,17%
Si, hay palabras que no entendía para que servían	40	38,83%
TOTAL	103	100%

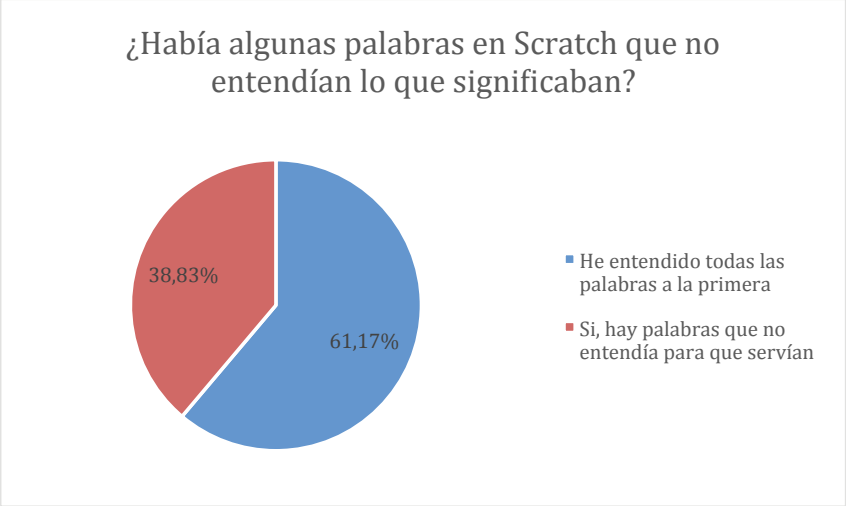


Figura 96. Gráfica de las respuestas obtenidas a la pregunta relativa a si había palabras presentes en Scratch que no se entendía su significado.

Fuente: Elaboración propia.

10.3.6.11.2. Análisis bivariable: palabras presentes en Scratch que no se han entendido, y clases recibidas

Según lo que se puede apreciar en la Tabla 80 y en la *Figura 97*, no parece haber relación entre las respuestas a esta pregunta, y el número de clases recibidas. En ambas respuestas, los porcentajes entre los que han recibido más y menos clases que la media son muy similares.

Tabla 80
Respuestas obtenidas a la pregunta de si había algunas palabras en Scratch que no se entendía su significado, y su relación con el número de clases recibidas previamente.

Clases recibidas	He entendido todas las palabras a la primera		Si, hay palabras que no entendía para que servían		TOTAL	
	Nº	%	Nº	%	Nº	%
Más clases que la media (Mayor)	30	62,50%	18	37,50%	48	100%
Menos clases que la media (Menor)	33	60%	22	40%	55	100%
TOTAL	63	61,17%	40	38,83%	103	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla.

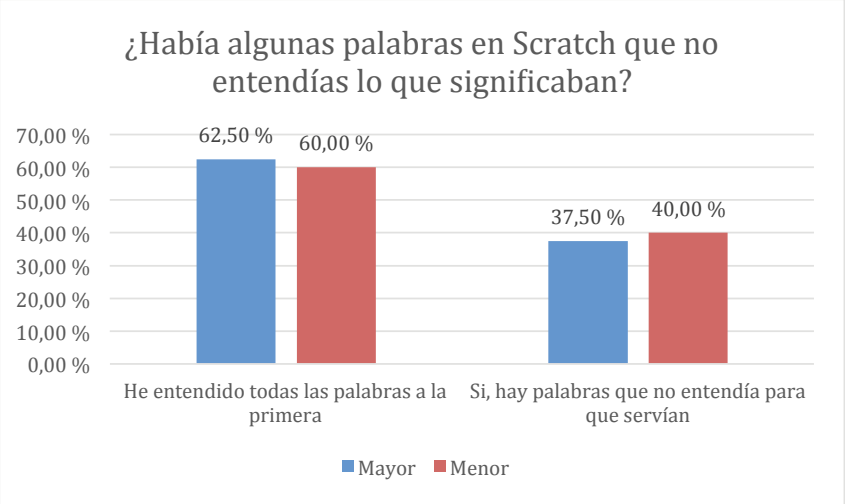


Figura 97. Gráfica de las respuestas obtenidas a la pregunta relativa a si había palabras presentes en Scratch que no se entendía su significado, y su relación con el número de clases recibidas previamente.

Fuente: Elaboración propia.

10.3.6.11.3. Análisis bivariable: palabras presentes en Scratch que no se han entendido y práctica autónoma

Al igual que en análisis bivariable anterior, los datos que muestran la Tabla 81 y la *Figura 98* indican que no hay una relación clara entre el grado de comprensión de las palabras utilizadas en Scratch, y la práctica autónoma.

Tabla 81
Respuestas obtenidas a la pregunta de si había palabras presentes en Scratch que no se entendía su significado, y su relación con la práctica autónoma.

	He entendido todas las palabras a la primera		**Si, hay palabras que no entendía para que servían**		**TOTAL**	
Práctica autónoma	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**
Sí	41	59,42%	28	40,58%	69	100%
No	21	63,64%	12	36,36%	33	100%
TOTAL	62	60,78%	40	39,22%	102	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla.

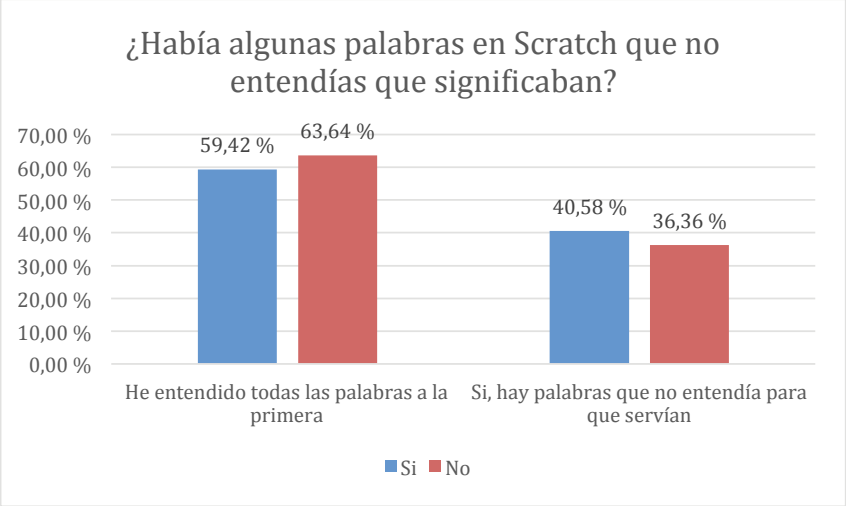


Figura 98. Gráfica de las respuestas obtenidas a la pregunta relativa a si había palabras presentes en Scratch que no se entendía su significado, y su relación con la práctica autónoma.

Fuente: Elaboración propia.

10.3.6.11.4. Análisis bivariable: palabras presentes en Scratch que no se han entendido y hábitos de uso del ordenador

Por los datos que vemos en Tabla 82 y en la *Figura 99*, no hay relación clara entre las palabras presentes en Scratch que no se han entendido, y los hábitos de uso del ordenador de los participantes. Las respuestas están repartidas entre los que más y los que menos usan el ordenador.

Tabla 82
Respuestas obtenidas a la pregunta de si había palabras presentes en Scratch que no se entendía su significado, y su relación con los hábitos de uso del ordenador.

	Sí		**No**		**TOTAL**	
Hábitos de uso del ordenador	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**
Tengo un ordenador en casa y lo utilizo habitualmente (todos o casi todos los días)	20	57,14%	15	42,86%	35	100%
Tengo un ordenador en casa, pero lo utilizo poco (una o dos veces por semana o menos)	33	66%	17	34%	50	100%
En mi casa no hay ordenador, pero utilizo el del colegio, o el de un familiar o amigo, habitualmente (todos o casi todos los días)	4	66,67%	2	33,33%	6	100%

Hábitos de uso del ordenador	Sí		No		TOTAL	
	Nº	%	Nº	%	Nº	%
En mi casa no hay ordenador, pero utilizo el del colegio, o el de un familiar o amigo, de vez en cuando (una o dos veces por semana o menos)	3	100%	0	0%	3	100%
No suelo utilizar ningún ordenador	3	33,33%	6	66,67%	9	100%
TOTAL	63	61,17%	40	38,83%	103	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla.

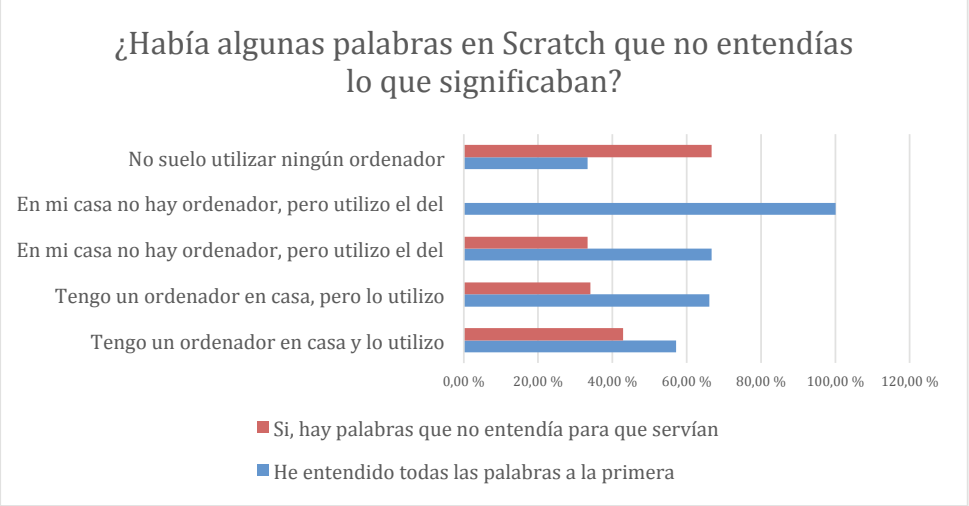


Figura 99. Gráfica de las respuestas obtenidas a la pregunta relativa a si había palabras presentes en Scratch que no se entendía su significado, y su relación con los hábitos de uso del ordenador.

Fuente: Elaboración propia.

10.3.6.11.5. Conclusiones a la pregunta "¿Había algunas palabras en Scratch que no entendías lo que significaban?"

La Pregunta 15 está enfocada a la evaluación del lenguaje que se utiliza en Scratch. Las pautas del análisis heurístico que dan origen a esta pregunta son NNGG2, que evalúa si se utiliza el lenguaje de los usuarios, y MIT2, MIT8 y MIT9, relacionadas con la sencillez de las palabras utilizadas, que no se incluya jerga, y que la misma terminología se utilice de forma constante en todo el programa. Las respuestas a esta pregunta no son definitivas para afirmar o negar el cumplimento de estas pautas, por eso se evalúan también en otras preguntas, pero las respuestas obtenidas dan un primer apunte a que sí se cumplen, dada la mayoría que indica

que lo entienden todo a la primera. Aunque en posteriores preguntas se demuestre que esto no es así, que tengan la percepción de que lo entienden todo es significativo.

10.3.6.12. ¿Sabrías decirme cuáles de estas palabras están en el programa?

10.3.6.12.1. Análisis univariable: si la palabra "evento" está presente en Scratch

La palabra evento sí está en el programa, es una de las categorías de bloques. Al analizar los datos mostrados en la Tabla 83 y en la *Figura 100* se ve que aproximadamente la mitad de los encuestados sí identifican "evento" como una palabra presente en Scratch, mientras que la otra mitad no.

Tabla 83
Respuestas obtenidas a la pregunta de si la palabra "evento" está presente en Scratch.

Opciones	Nº de respuestas	Porcentaje de respuestas
Sí	53	52,48%
No	48	47,52%
TOTAL	101	100%

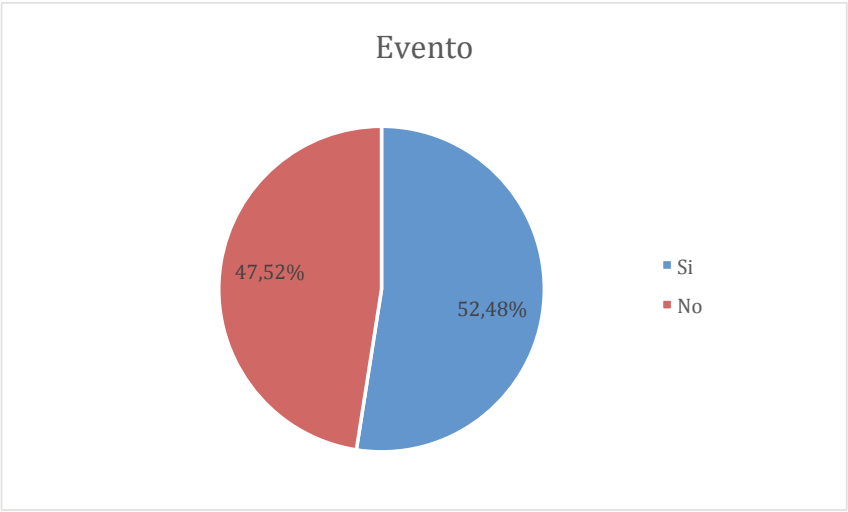


Figura 100. Gráfica de las respuestas obtenidas a la pregunta relativa a si está la palabra "evento" está presente en Scratch.

Fuente: Elaboración propia.

10.3.6.12.2. Análisis bivariable: si la palabra "evento" está presente en Scratch y clases recibidas

Por lo que se ve en la Tabla 84 y en la *Figura 98* la mayoría de los que han respondido correctamente se encuentran entre los que más clases han recibido.

Tabla 84
Respuestas obtenidas a la pregunta de si la palabra "evento" está presente en Scratch, y su relación con el número de clases recibidas previamente.

Clases recibidas	Sí		No		TOTAL	
	Nº	%	Nº	%	Nº	%
Más clases que la media (Mayor)	33	68,75%	15	31,25%	48	100%
Menos clases que la media (Menor)	20	37,74%	33	62,26%	53	100%
TOTAL	53	52,48%	48	47,52%	101	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla.

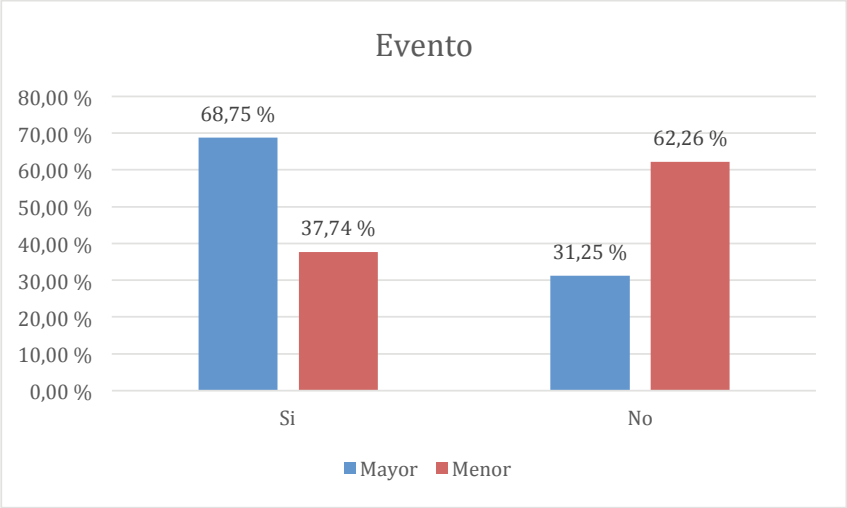


Figura 101. Gráfica de las respuestas obtenidas a la pregunta relativa a si la palabra "evento" está presente en Scratch, y su relación con el número de clases recibidas previamente.

Fuente: Elaboración propia.

10.3.6.12.3. Análisis bivariable: si la palabra "evento" está presente en Scratch y práctica autónoma

Según lo mostrado en la Tabla 85 y en la *Figura 99*, los porcentajes de los que aciertan y fallan están repartidos de una forma más o menos equilibrada entre los

que sí tienen práctica autónoma y los que no, existiendo una pequeña mayoría entre los que aciertan, y que sin embargo no han tenido práctica autónoma. En definitiva, no hay una relación clara entre estas dos variables.

Tabla 85
Respuestas obtenidas a la pregunta de si la palabra "evento" está presente en Scratch, y su relación con la práctica autónoma.

Práctica autónoma	**He entendido todas las palabras a la primera**		**Si, hay palabras que no entendía para que servían**		**TOTAL**	
	Nº	**%**	**Nº**	**%**	**Nº**	**%**
Sí	32	47,76%	35	52,24%	67	100%
No	20	60,61%	13	39,39%	33	100%
TOTAL	52	52%	48	48%	100	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla

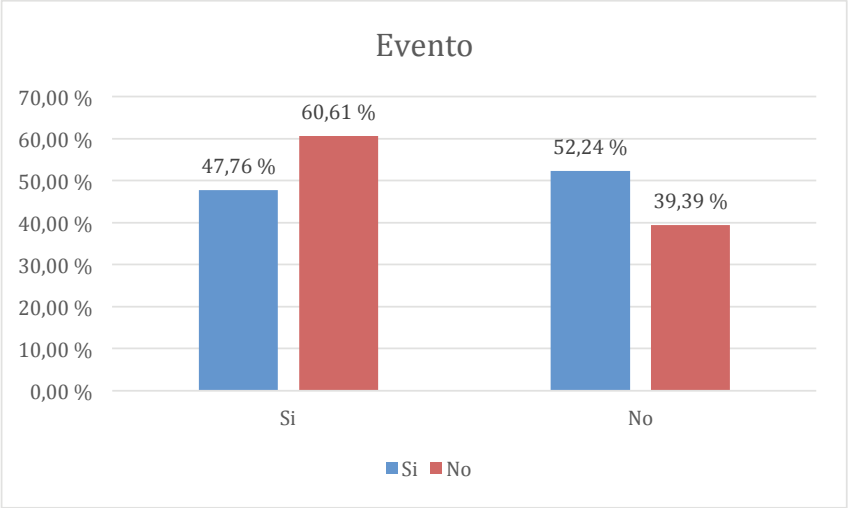


Figura 102. Gráfica de las respuestas obtenidas a la pregunta relativa a si la palabra "evento" está presente en Scratch, y su relación con la práctica autónoma.

Fuente: Elaboración propia.

10.3.6.12.4. Análisis bivariable: si la palabra "evento" está presente en Scratch y hábitos de uso del ordenador

No hay indicios claros de que exista una relación entre ser capaz de identificar la palabra "evento" dentro de Scratch, y los hábitos de uso del ordenador, según lo que se puede apreciar en la Tabla 86 y en la *Figura 103*.

Tabla 86
Respuestas obtenidas a la pregunta de si la palabra "evento" está presente en Scratch, y su relación con los hábitos de uso del ordenador.

	Sí		No		TOTAL	
Hábitos de uso del ordenador	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**
Tengo un ordenador en casa y lo utilizo habitualmente (todos o casi todos los días)	13	37,14%	22	62,86%	35	100%
Tengo un ordenador en casa, pero lo utilizo poco (una o dos veces por semana o menos)	27	56,25%	21	43,75%	48	100%
En mi casa no hay ordenador, pero utilizo el del colegio, o el de un familiar o amigo, habitualmente (todos o casi todos los días)	4	66,67%	2	33,33%	6	100%
En mi casa no hay ordenador, pero utilizo el del colegio, o el de un familiar o amigo, de vez en cuando (una o dos veces por semana o menos)	2	66,67%	1	33,33%	3	100%
No suelo utilizar ningún ordenador	7	77,78%	2	22,22%	9	100%
TOTAL	53	52,48%	48	47,52%	101	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla.

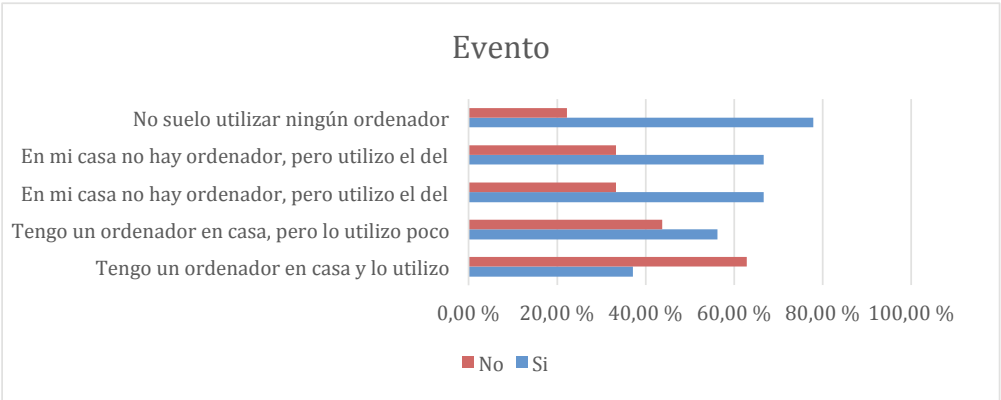


Figura 103. Gráfica de las respuestas obtenidas a la pregunta de si la palabra "evento" está presente en Scratch, y su relación con los hábitos de uso del ordenador.

Fuente: Elaboración propia.

10.3.6.12.5. Análisis univariable: si la palabra "variable" está presente en Scratch.

La palabra "variable" sí está presente en Scratch, y era un elemento que se debía utilizar para realizar la tarea propuesta por el docente en la actividad previa a responder el cuestionario. Por lo que se ve en la Tabla 87 y en la *Figura 104*, el 72,55% responde correctamente.

Tabla 87
Respuestas obtenidas a la pregunta de si la palabra "variable" está presente en Scratch.

Opciones	Nº de respuestas	Porcentaje de respuestas
Sí	74	72,55%
No	28	27,45%
TOTAL	102	100%

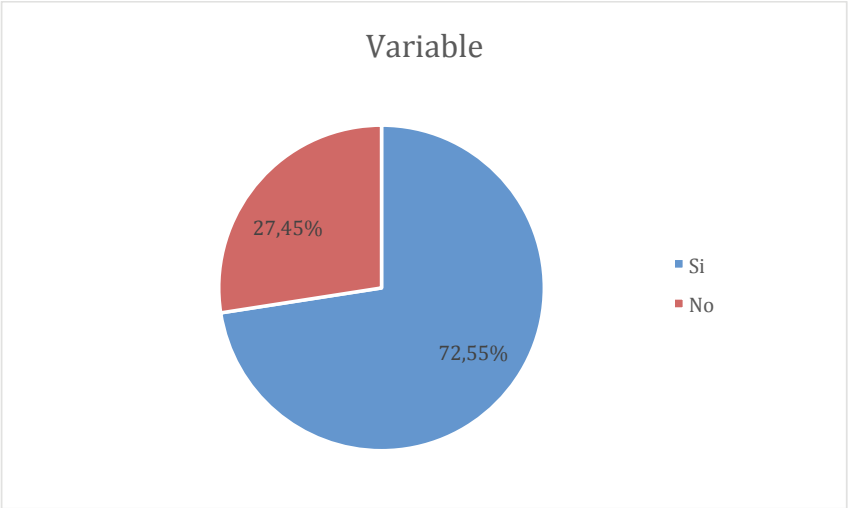


Figura 104. Gráfica de las respuestas obtenidas a la pregunta relativa a si la palabra "variable" está presente en Scratch.

Fuente: Elaboración propia.

10.3.6.12.6. Análisis bivariable: si la palabra "variable" está presente en Scratch y clases recibidas

Por lo que se ve en la Tabla 88 y en la *Figura 105*, son mayoría los que sí aciertan la pregunta, y sin embargo han recibido menos clases que la media.

Tabla 88
Respuestas obtenidas a la pregunta de si a si la palabra "variable" está presente en Scratch, y su relación con el número de clases recibidas previamente.

	Sí		No		TOTAL	
Clases recibidas	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**
Más clases que la media (Mayor)	27	57,45%	20	42,55%	47	100%
Menos clases que la media (Menor)	47	85,45%	8	14,55%	55	100%
TOTAL	74	72,55%	28	27,45%	102	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla.

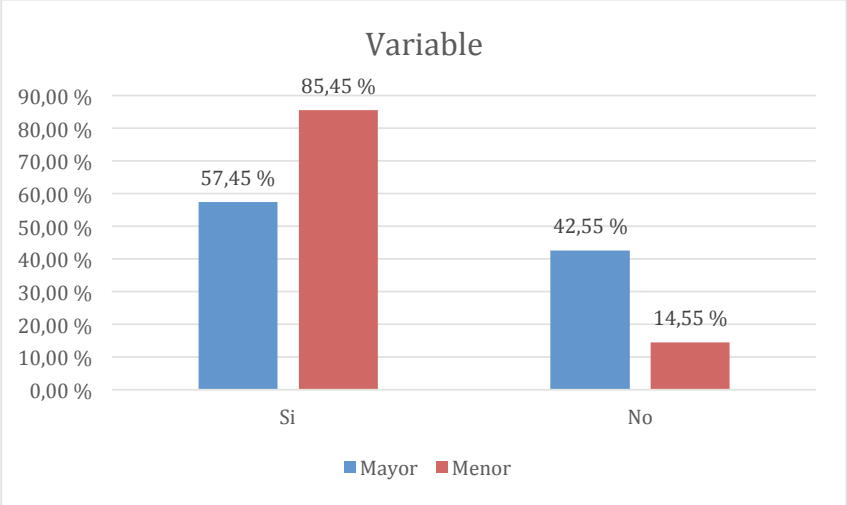


Figura 105. Gráfica de las respuestas obtenidas a la pregunta relativa a si la palabra "variable" está presente en Scratch, y su relación con el número de clases recibidas previamente.

Fuente: Elaboración propia.

10.3.6.12.7. Análisis bivariable: si la palabra "variable" está presente en Scratch y práctica autónoma

La relación entre identificar si la palabra "variable" está presente en Scratch, y la práctica autónoma no está clara, a tenor de los datos mostrados en la Tabla 89 y en la *Figura 106*. Tanto en el caso de los que sí han tenido práctica autónoma, como en los que no, los porcentajes entre los que aciertan y los que fallan son muy similares. Por este motivo no hay evidencias que permitan relacionar estas dos variables.

Tabla 89
Respuestas obtenidas a la pregunta de si la palabra "variable" está presente en Scratch, y su relación con la práctica autónoma.

Práctica autónoma	He entendido todas las palabras a la primera		Si, hay palabras que no entendía para que servían		TOTAL	
	Nº	%	Nº	%	Nº	%
Sí	54	78,26%	15	21,74%	69	100%
No	20	62,50%	12	37,50%	32	100%
TOTAL	74	73,27%	27	26,73%	101	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla

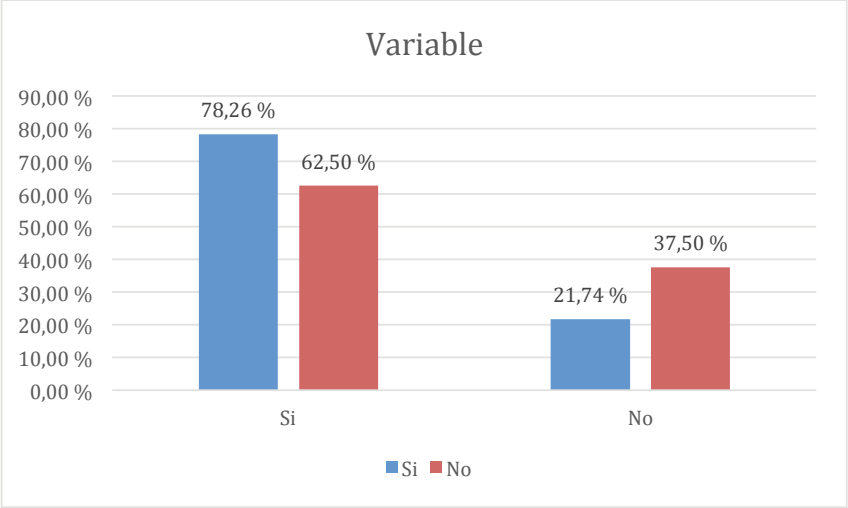


Figura 106. Gráfica de las respuestas obtenidas a la pregunta relativa a si la palabra "variable" está presente en Scratch, y su relación con la práctica autónoma.

Fuente: Elaboración propia.

10.3.6.12.8. Análisis bivariable: si la palabra "variable" está presente en Scratch y hábitos de uso del ordenador

Por lo que se aprecia en la Tabla 90 y en la *Figura 107*, tampoco hay una relación clara entre los hábitos de uso del ordenador, y el haber identificado la palabra "variable" dentro de Scratch.

Tabla 90
Respuestas obtenidas a la pregunta de si la palabra "variable "está presente en Scratch, y su relación con los hábitos de uso del ordenador.

	Sí		No		TOTAL	
Hábitos de uso del ordenador	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**
Tengo un ordenador en casa y lo utilizo habitualmente (todos o casi todos los días)	26	74,29%	9	25,71%	35	100%
Tengo un ordenador en casa, pero lo utilizo poco (una o dos veces por semana o menos)	40	81,63%	9	18,37%	49	100%
En mi casa no hay ordenador, pero utilizo el del colegio, o el de un familiar o amigo, habitualmente (todos o casi todos los días)	2	33,33%	4	66,67%	6	100%
En mi casa no hay ordenador, pero utilizo el del colegio, o el de un familiar o amigo, de vez en cuando (una o dos veces por semana o menos)	2	66,67%	1	33,33%	3	100%
No suelo utilizar ningún ordenador	4	44,44%	5	55,56%	9	100%
TOTAL	74	72,55%	28	27,45%	102	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla.

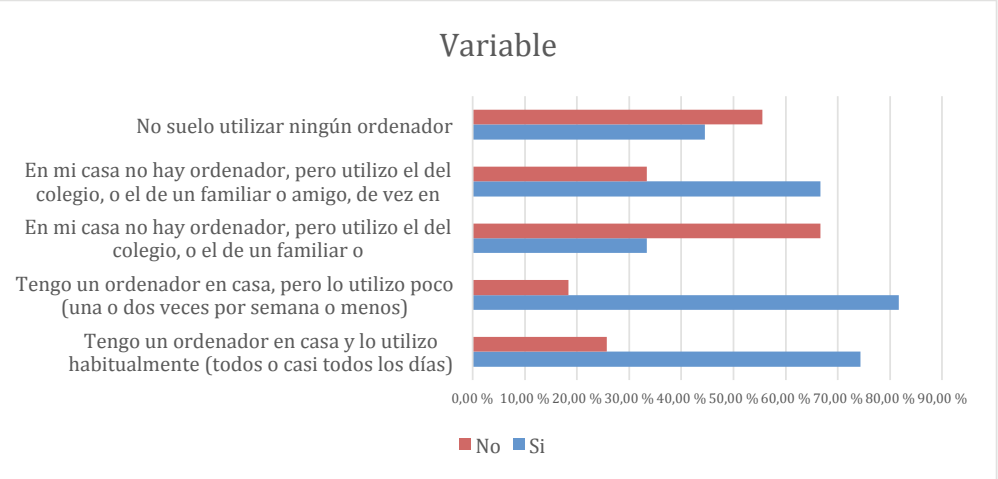


Figura 107. Gráfica de las respuestas obtenidas a la pregunta de si la palabra "variable" está presente en Scratch, y su relación con los hábitos de uso del ordenador.

Fuente: Elaboración propia.

10.3.6.12.9. Análisis univariable: si la palabra "objeto" está presente en Scratch

La palabra "objeto" sí está presente, como etiqueta del contenedor de los elementos presentes en la escena. Por lo que se ve en la Tabla 91 y en la *Figura 108*, un 89,22% reconoce la palabra "objeto" dentro de Scratch.

Tabla 91
Respuestas obtenidas a la pregunta de si la palabra "objeto" está presente en Scratch.

Opciones	Nº de respuestas	Porcentaje de respuestas
Sí	91	89,22%
No	11	10,78%
TOTAL	102	100%

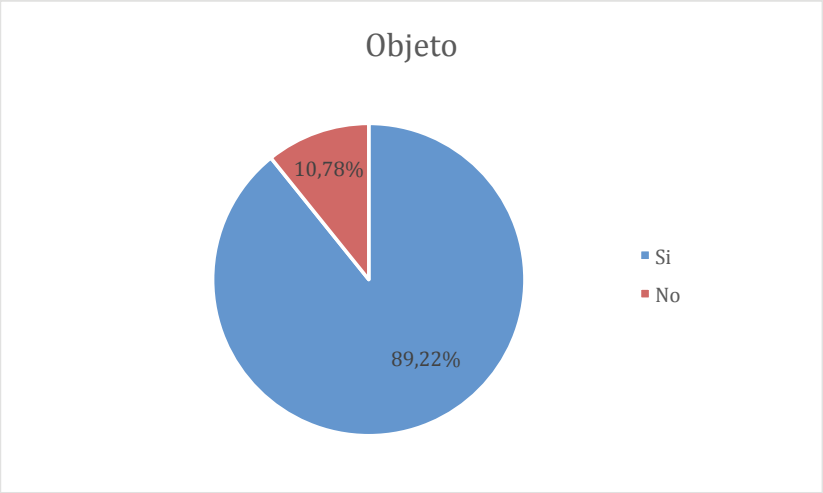


Figura 108. Gráfica de las respuestas obtenidas a la pregunta relativa a si la palabra "objeto" está presente en Scratch.

Fuente: Elaboración propia.

10.3.6.12.10. Análisis bivariable: si la palabra "objeto" está presente en Scratch y clases recibidas

Por lo que reflejan la Tabla 92 y la *Figura 109*, y dada la similitud de los porcentajes en uno y otro caso, no se puede establecer una relación clara entre haber sido capaz reconocer la palabra "objeto" dentro de Scratch, y el número de clases recibidas.

Tabla 92

Respuestas obtenidas a la pregunta de si la palabra "objeto" está presente en Scratch, y su relación con *el número de clases recibidas previamente*.

Clases recibidas	Sí		No		TOTAL	
	Nº	%	Nº	%	Nº	%
Más clases que la media (Mayor)	45	93,75%	3	6,25%	48	100%
Menos clases que la media (Menor)	46	85,19%	8	14,81%	54	100%
TOTAL	91	89,22%	11	10,78%	102	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla.

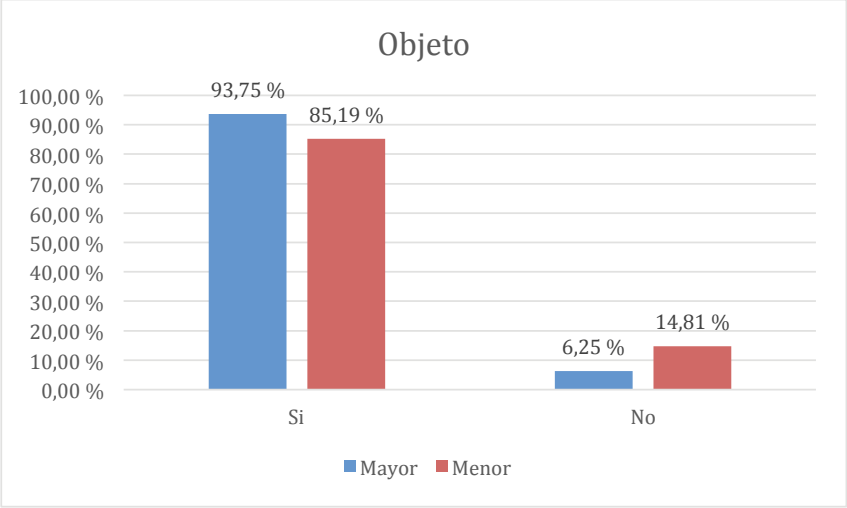


Figura 109. Gráfica de las respuestas obtenidas a la pregunta relativa a si la palabra "objeto" está presente en Scratch, y su relación con el número de clases recibidas previamente.

Fuente: Elaboración propia.

10.3.6.12.11. Análisis bivariable: si la palabra "objeto" está presente en Scratch y práctica autónoma

Según lo que se muestra en Tabla 93 y en la *Figura 110*, no hay una relación evidente entre haber sido capaz reconocer la palabra "objeto" dentro de Scratch, y la práctica autónoma. De nuevo los porcentajes en uno y otro caso son muy similares.

Tabla 93
Respuestas obtenidas a la pregunta de si la palabra "objeto" está presente en Scratch, y su relación con la práctica autónoma.

Práctica autónoma	**Sí**		**No**		**TOTAL**	
	Nº	**%**	**Nº**	**%**	**Nº**	**%**
Sí	62	91,18%	6	8,82%	68	100%
No	28	84,85%	5	15,15%	33	100%
TOTAL	90	89,11%	11	10,89%	101	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla

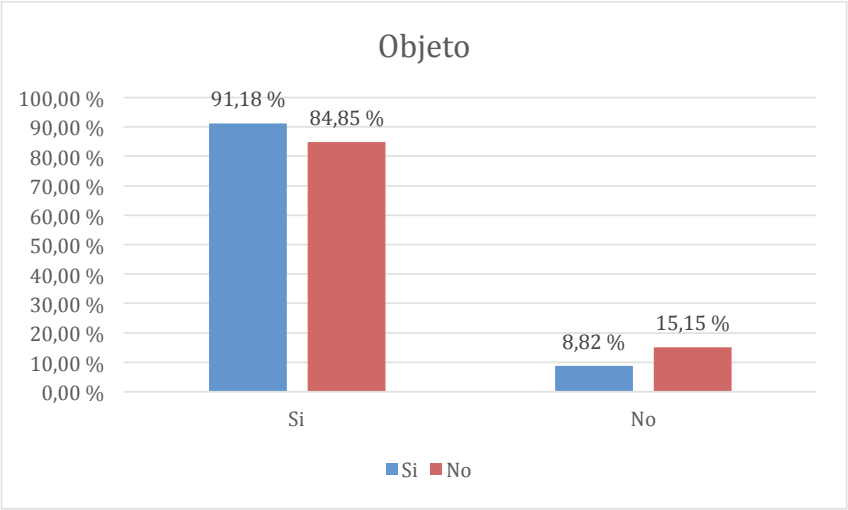


Figura 110. Gráfica de las respuestas obtenidas a la pregunta relativa a si la palabra "objeto" está presente en Scratch, y su relación con la práctica autónoma.

Fuente: Elaboración propia.

10.3.6.12.12. Análisis bivariable: si la palabra "objeto" está presente en Scratch y hábitos de uso del ordenador

Según los datos de la Tabla 94 y de la *Figura 111*, los pocos encuestados que responden incorrectamente a la pregunta, están repartidos entre prácticamente todos los casos de hábitos de uso del ordenador, con lo que no se puede establecer una relación clara entre estas dos variables.

Tabla 94
Respuestas obtenidas a la pregunta de si la palabra "objeto" está presente en Scratch, y su relación con los hábitos de uso del ordenador.

	Sí		No		TOTAL	
Hábitos de uso del ordenador	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**
Tengo un ordenador en casa y lo utilizo habitualmente (todos o casi todos los días)	32	91,43%	3	8,57%	35	100%
Tengo un ordenador en casa, pero lo utilizo poco (una o dos veces por semana o menos)	42	85,71%	7	14,29%	49	100%
En mi casa no hay ordenador, pero utilizo el del colegio, o el de un familiar o amigo, habitualmente (todos o casi todos los días)	6	100%	0	0%	6	100%
En mi casa no hay ordenador, pero utilizo el del colegio, o el de un familiar o amigo, de vez en cuando (una o dos veces por semana o menos)	3	100%	0	0%	3	100%
No suelo utilizar ningún ordenador	8	88,89%	1	11,11%	9	100%
TOTAL	91	89,22%	11	10,78%	102	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla.

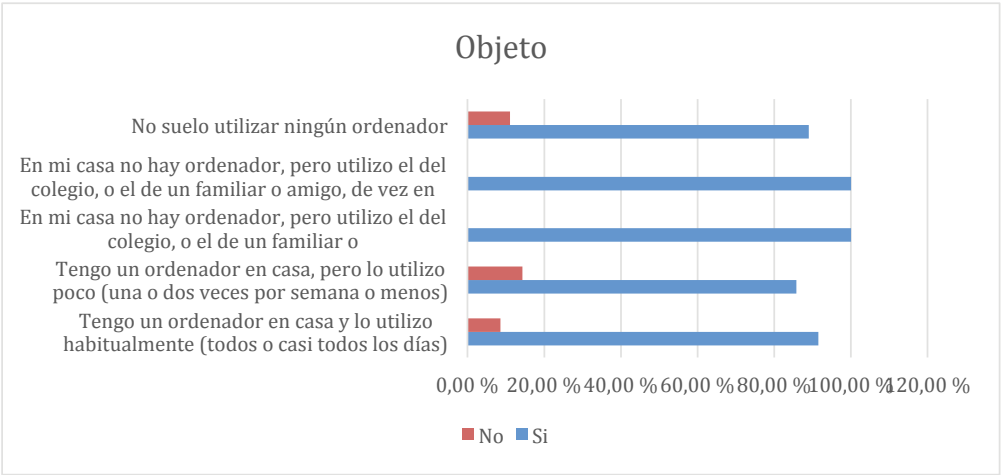


Figura 111. Gráfica de las respuestas obtenidas a la pregunta de si la palabra "objeto" está presente en Scratch, y su relación con los hábitos de uso del ordenador.

Fuente: Elaboración propia.

10.3.6.12.13. Análisis univariable: si la palabra "interfaz" está presente en Scratch y clases recibidas

La palabra "interfaz" no está presente de forma directa en Scratch. Según se muestra en la Tabla 95 y en la *Figura 112*, la mayoría responden correctamente a esta pregunta.

Tabla 95
Respuestas obtenidas a la pregunta de si la palabra "interfaz" está presente en Scratch.

Opciones	Nº de respuestas	Porcentajes de respuestas
Sí	19	19%
No	81	81%
TOTAL	100	100%

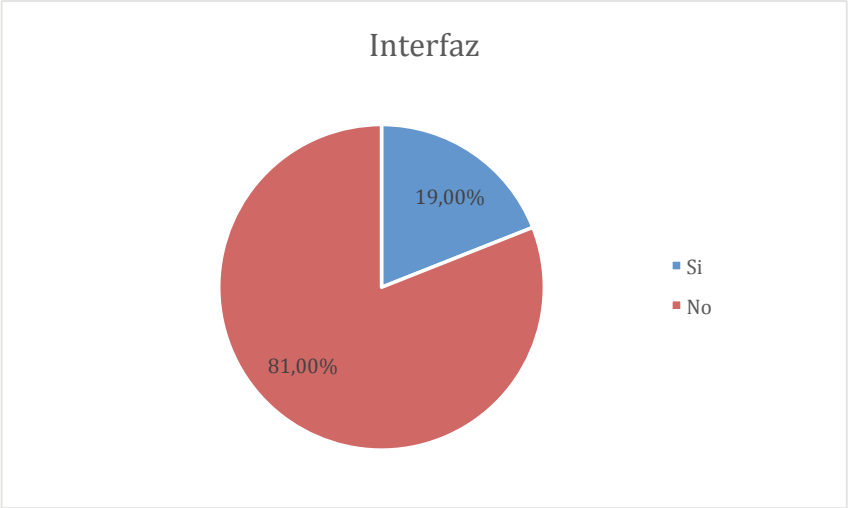


Figura 112. Gráfica de las respuestas obtenidas a la pregunta relativa a si la palabra "interfaz" está presente en Scratch.

Fuente: Elaboración propia.

10.3.6.12.14. Análisis bivariable: si la palabra "interfaz" está presente en Scratch y clases recibidas

Según lo que se aprecia en la Tabla 96 y en la *Figura 113*, dada la similitud de los porcentajes en uno y otro caso, no se puede establecer una relación clara entre haber sido capaz reconocer que la palabra "interfaz" no se encuentra dentro de Scratch, y el número de clases recibidas.

Tabla 96
Respuestas obtenidas a la pregunta de si la palabra "interfaz" está presente en Scratch, y su relación con el número de clases recibidas previamente.

Clases recibidas	Sí		No		TOTAL	
	Nº	%	Nº	%	Nº	%
Más clases que la media (Mayor)	13	27,66%	34	72,34%	47	100%
Menos clases que la media (Menor)	6	11,32%	47	88,68%	53	100%
TOTAL	19	19%	81	81%	100	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla.

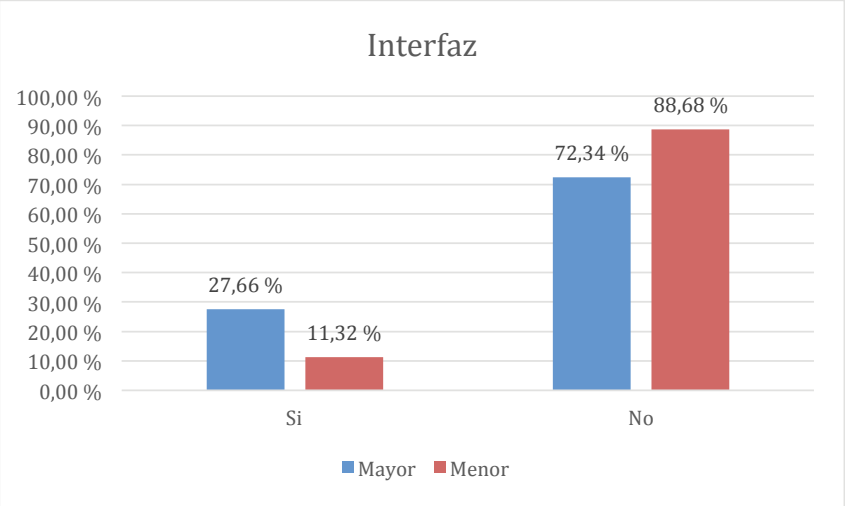


Figura 113. Gráfica de las respuestas obtenidas a la pregunta relativa a si la palabra "interfaz" está presente en Scratch, y su relación con el número de clases recibidas previamente.

Fuente: Elaboración propia.

10.3.6.12.15. Análisis bivariable: si la palabra "interfaz" está presente en Scratch y práctica autónoma

A tenor de los datos mostrados en la Tabla 97 y en la *Figura 114*, no hay una relación evidente entre haber sido capaz reconocer la ausencia de la palabra "interfaz" dentro de Scratch, y la práctica autónoma. De nuevo los porcentajes en uno y otro caso son muy similares.

Tabla 97
Respuestas obtenidas a la pregunta de si la palabra "interfaz" está presente en Scratch, y su relación con la práctica autónoma.

Práctica autónoma	Sí		No		TOTAL	
	Nº	%	Nº	%	Nº	%
Sí	13	19,40%	54	80,60%	67	100%
No	5	15,63%	27	84,38%	32	100%
TOTAL	18	18,18%	81	81,82%	99	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla

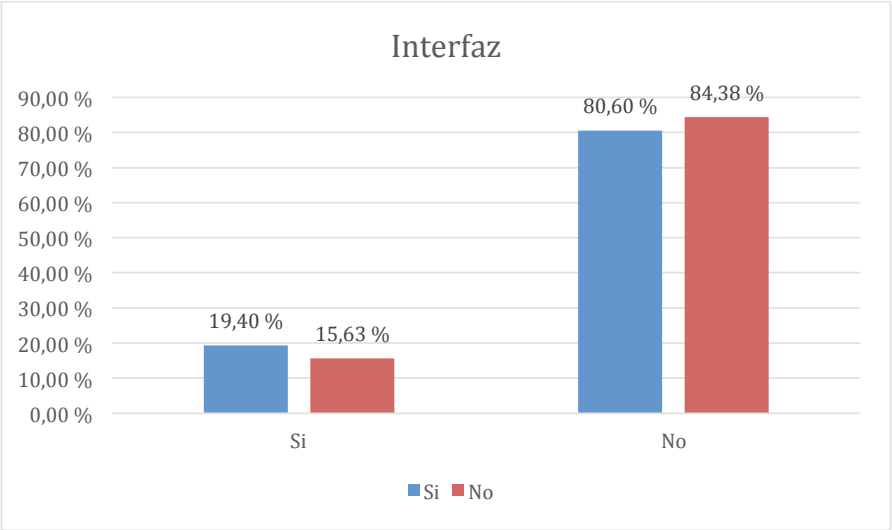


Figura 114. Gráfica de las respuestas obtenidas a la pregunta relativa a si la palabra "interfaz" está presente en Scratch, y su relación con la práctica autónoma.

Fuente: Elaboración propia.

10.3.6.12.16. Análisis bivariable: si la palabra "interfaz" está presente en Scratch y hábitos de uso del ordenador

Según lo que muestra la Tabla 98 y la *Figura 115*, el escaso porcentaje de participantes que responden incorrectamente a la pregunta, está repartido entre varios casos de hábitos de uso del ordenador, habiendo una ligera inclinación hacia los que más utilizan el ordenador. En cualquier caso, no son datos concluyentes que permitan establecer una relación clara entre estas dos variables.

Tabla 98
Respuestas obtenidas a la pregunta de si la palabra "interfaz" está presente en Scratch, y su relación con los hábitos de uso del ordenador

	Sí		No		TOTAL	
Hábitos de uso del ordenador	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**
Tengo un ordenador en casa y lo utilizo habitualmente (todos o casi todos los días)	8	22,86%	27	77,14%	35	100%
Tengo un ordenador en casa, pero lo utilizo poco (una o dos veces por semana o menos)	9	19,15%	38	80,85%	47	100%
En mi casa no hay ordenador, pero utilizo el del colegio, o el de un familiar o amigo, habitualmente (todos o casi todos los días)	0	0%	6	100%	6	100%
En mi casa no hay ordenador, pero utilizo el del colegio, o el de un familiar o amigo, de vez en cuando (una o dos veces por semana o menos)	0	0%	3	100%	3	100%
No suelo utilizar ningún ordenador	2	22,22%	7	77,78%	9	100%
TOTAL	19	19%	81	81%	100	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla.

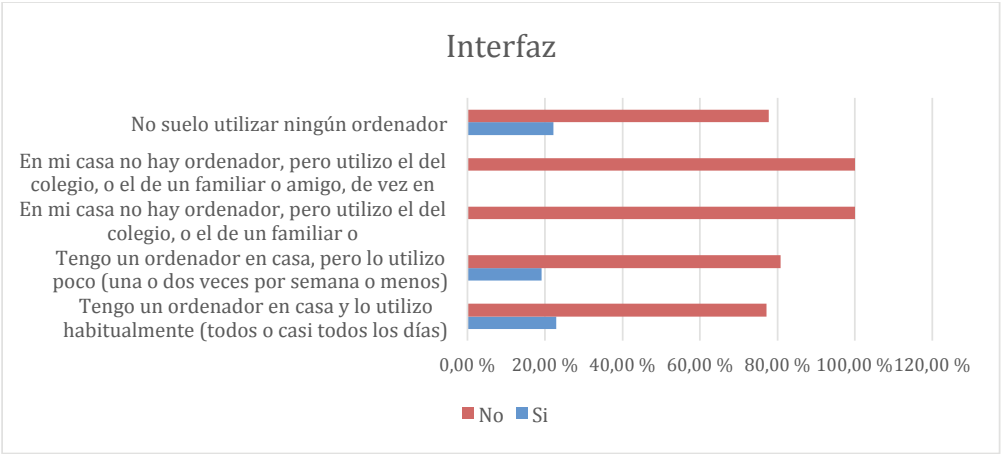


Figura 115. Gráfica de las respuestas obtenidas a la pregunta de si la palabra "interfaz" está presente en Scratch, y su relación con los hábitos de uso del ordenador.

Fuente: Elaboración propia.

10.3.6.12.17. Análisis univariable: si la palabra "operador" está presente en Scratch

La palabra "operador" sí está presente en Scratch como una de las categorías de bloques de programación. Según muestra la Tabla 99 y en la *Figura 116*, un 63% responde correctamente a esta pregunta.

Tabla 99
Respuestas obtenidas a la pregunta de si la palabra "operador" está presente en Scratch.

Opciones	Nº de respuestas	Porcentaje de respuestas
Sí	63	63%
No	37	37%
TOTAL	100	100%

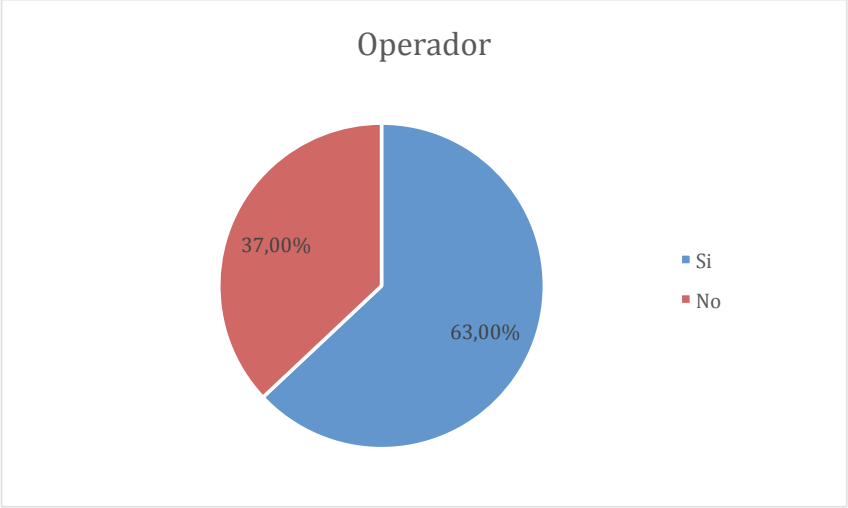


Figura 116. Gráfica de las respuestas obtenidas a la pregunta relativa a si la palabra "operador" está presente en Scratch.

Fuente: Elaboración propia.

10.3.6.12.18. Análisis bivariable: si la palabra "operador" está presente en Scratch y clases recibidas

Por lo que podemos ver en la Tabla 100 y en la *Figura 117*, hay una mayoría entre los que han recibido menos clases que la media, pero que sin embargo responden correctamente a la pregunta.

Tabla 100
Respuestas obtenidas a la pregunta de si la palabra "operador" está presente en Scratch, y su relación con el número de clases recibidas previamente.

Clases recibidas	Sí		No		TOTAL	
	Nº	%	Nº	%	Nº	%
Más clases que la media (Mayor)	20	43,48%	26	56,52%	46	100%
Menos clases que la media (Menor)	43	79,63%	11	20,37%	54	100%
TOTAL	63	63%	37	37%	100	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla.

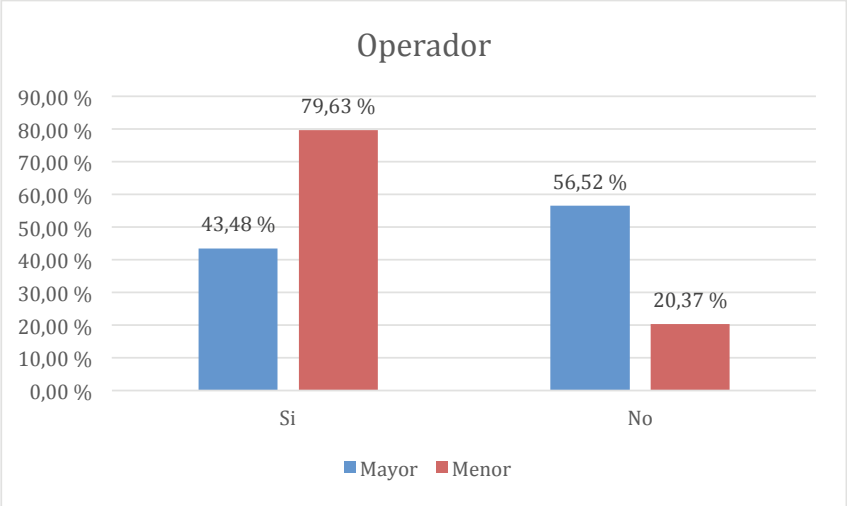


Figura 117. Gráfica de las respuestas obtenidas a la pregunta relativa a si la palabra "operador" está presente en Scratch.

Fuente: Elaboración propia.

10.3.6.12.19. Análisis bivariable: si la palabra "operador" está presente en Scratch y práctica autónoma

Según muestra la Tabla 101 y en la *Figura 118*, por la similitud de los porcentajes en uno y otro caso, no se puede establecer una relación clara entre haber sido capaz reconocer la presencia de la palabra "operador" dentro de Scratch, y la práctica autónoma.

Tabla 101
Respuestas obtenidas a la pregunta de si la palabra "operador" está presente en Scratch, y su relación con la práctica autónoma.

Práctica autónoma	Sí		No		TOTAL	
	Nº	%	Nº	%	Nº	%
Sí	45	65,22%	24	34,78%	69	100%
No	17	56,67%	13	43,33%	30	100%
TOTAL	62	62,63%	37	37,37%	99	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla

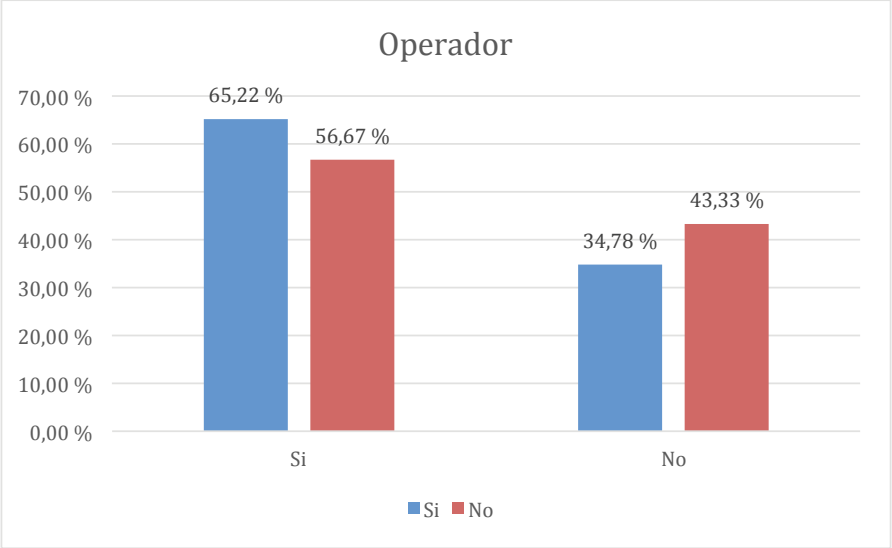


Figura 118. Gráfica de las respuestas obtenidas a la pregunta relativa a si la palabra "operador" está presente en Scratch, y su relación con la práctica autónoma.

Fuente: Elaboración propia.

10.3.6.12.20. Análisis bivariable: si la palabra "operador" está presente en Scratch y hábitos de uso del ordenador

Según los datos que se muestran en la Tabla 102 y en la *Figura 119*, los participantes que respondieron correcta e incorrectamente a esta pregunta están repartidos entre todos los casos de hábitos de uso del ordenador, y no se muestra una tendencia clara que relacione estas dos variables.

Tabla 102
Respuestas obtenidas a la pregunta de si la palabra "interfaz" está presente en Scratch, y su relación con los hábitos de uso del ordenador

	Sí		No		TOTAL	
Hábitos de uso del ordenador	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**
Tengo un ordenador en casa y lo utilizo habitualmente (todos o casi todos los días)	21	61,76 %	13	38,24 %	34	100%
Tengo un ordenador en casa, pero lo utilizo poco (una o dos veces por semana o menos)	36	73,47 %	13	26,53 %	49	100%
En mi casa no hay ordenador, pero utilizo el del colegio, o el de un familiar o amigo, habitualmente (todos o casi todos los días)	1	16,67 %	5	83,33 %	6	100%
En mi casa no hay ordenador, pero utilizo el del colegio, o el de un familiar o amigo, de vez en cuando (una o dos veces por semana o menos)	1	33,33 %	2	66,67 %	3	100%
No suelo utilizar ningún ordenador	4	50,00 %	4	50,00 %	8	100%
TOTAL	63	63,00 %	37	37,00 %	100	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla

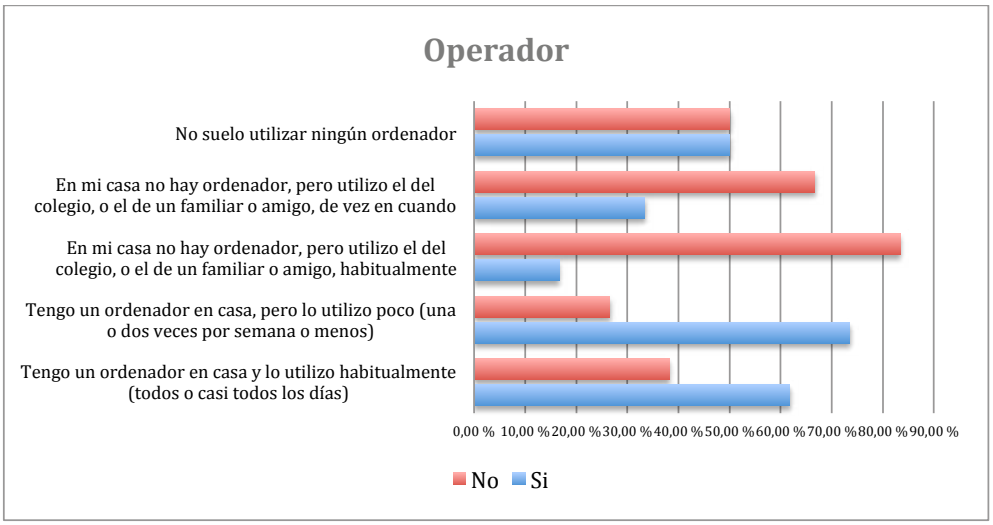


Figura 119. Gráfica de las respuestas obtenidas a la pregunta de si la palabra "interfaz" está presente en Scratch, y su relación con los hábitos de uso del ordenador.

Fuente: Elaboración propia.

10.3.6.12.21. Análisis univariable: si la palabra "sensor" está presente en Scratch

Sensores es una categoría de bloques de programación, por lo tanto la respuesta correcta a esta pregunta es sí. En la Tabla 103 y en la *Figura 120* se muestra que un 85,44% responden adecuadamente.

Tabla 103
Respuestas obtenidas a la pregunta de si está la palabra "sensor" está presente en Scratch.

Opciones	Nº de respuestas	Porcentaje de respuestas
Sí	88	85,44%
No	15	14,56%
TOTAL	103	100%

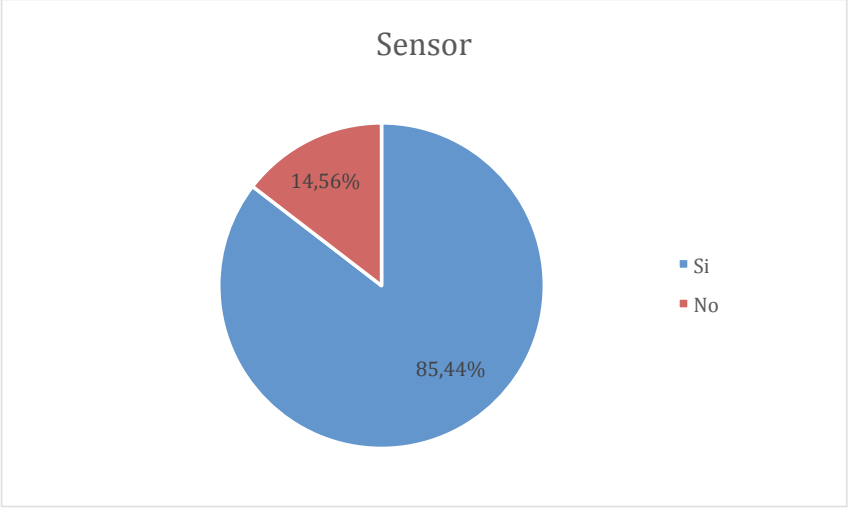


Figura 120. Gráfica de las respuestas obtenidas a la pregunta relativa a si la palabra "sensor" está presente en Scratch.

Fuente: Elaboración propia.

10.3.6.12.22. Análisis bivariable: si la palabra "sensor" está presente en Scratch y clases recibidas

En la Tabla 104 y en la *Figura 121*, por la similitud de los porcentajes en uno y otro caso, no se puede establecer una relación clara entre haber sido capaz reconocer la presencia de la palabra "sensor" dentro de Scratch, y el número de clases recibidas.

Tabla 104
Respuestas obtenidas a la pregunta de si la palabra "sensor" está presente en Scratch, y su relación con el número de clases recibidas previamente.

Clases recibidas	Sí		No		TOTAL	
	Nº	%	Nº	%	Nº	%
Más clases que la media (Mayor)	39	81,25%	9	18,75%	48	100%
Menos clases que la media (Menor)	49	89,09%	6	10,91%	55	100%
TOTAL	88	85,44%	15	14,56%	103	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla.

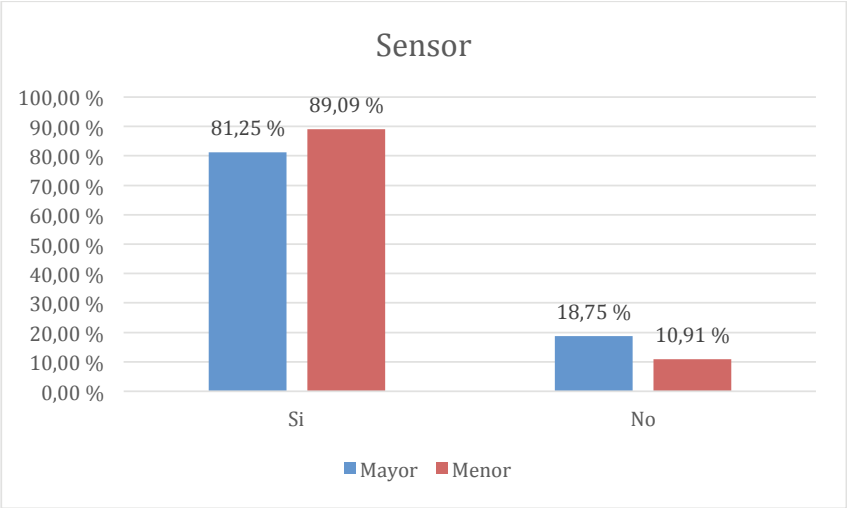


Figura 121. Gráfica de las respuestas obtenidas a la pregunta relativa a si la palabra "sensor" está presente en Scratch, y su relación con el número de clases recibidas previamente.

Fuente: Elaboración propia.

10.3.6.12.23. Análisis bivariable: si la palabra "sensor" está presente en Scratch y práctica autónoma

Como se puede ver en la Tabla 105 y en la *Figura 122*, los porcentajes son muy parecidos entre los que han tenido práctica autónoma, y los que no. Es decir, no parece haber relación entre haber sido capaz de detectar la palabra "sensor" dentro de Scratch, y la práctica autónoma.

Tabla 105
Respuestas obtenidas a la pregunta de si la palabra "sensor" está presente en Scratch, y su relación con la práctica autónoma.

Práctica autónoma	Sí		No		TOTAL	
	Nº	%	Nº	%	Nº	%
Sí	60	86,96%	9	13,04%	69	100%
No	28	84,85%	5	15,15%	33	100%
TOTAL	88	86,27%	14	13,73%	102	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla

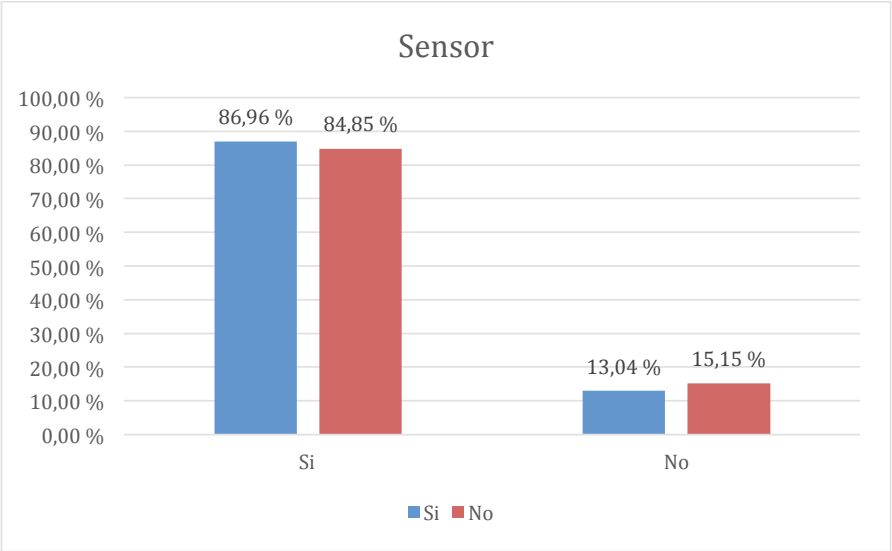


Figura 122. Gráfica de las respuestas obtenidas a la pregunta relativa a si la palabra "sensor" está presente en Scratch, y su relación con la práctica autónoma. Fuente: Elaboración propia.

10.3.6.12.24. Análisis bivariable: si la palabra "sensor" está presente en Scratch y hábitos de uso del ordenador

Sólo un 15% respondieron de forma incorrecta a esta pregunta, y entre ellos, la mayoría (un 55,56%) coincide con que no utilizan ningún ordenador. Así se ve reflejado en la Tabla 106 en la *Figura 123*.

Tabla 106
Respuestas obtenidas a la pregunta de si la palabra "sensor" está presente en Scratch, y su relación con los hábitos de uso del ordenador.

	Sí		No		TOTAL	
Hábitos de uso del ordenador	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**
Tengo un ordenador en casa y lo utilizo habitualmente (todos o casi todos los días)	30	85,71%	5	14,29%	35	100%
Tengo un ordenador en casa, pero lo utilizo poco (una o dos veces por semana o menos)	46	92%	4	8%	50	100%
En mi casa no hay ordenador, pero utilizo el del colegio, o el de un familiar o amigo, habitualmente (todos o casi todos los días)	5	83,33%	1	16,67%	6	100%
En mi casa no hay ordenador, pero utilizo el del colegio, o el de un familiar o amigo, de vez en cuando (una o dos veces por semana o menos)	3	100%	0	0%	3	100%
No suelo utilizar ningún ordenador	4	44,44%	5	55,56%	9	100%
TOTAL	88	85,44%	15	14,56%	103	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla.

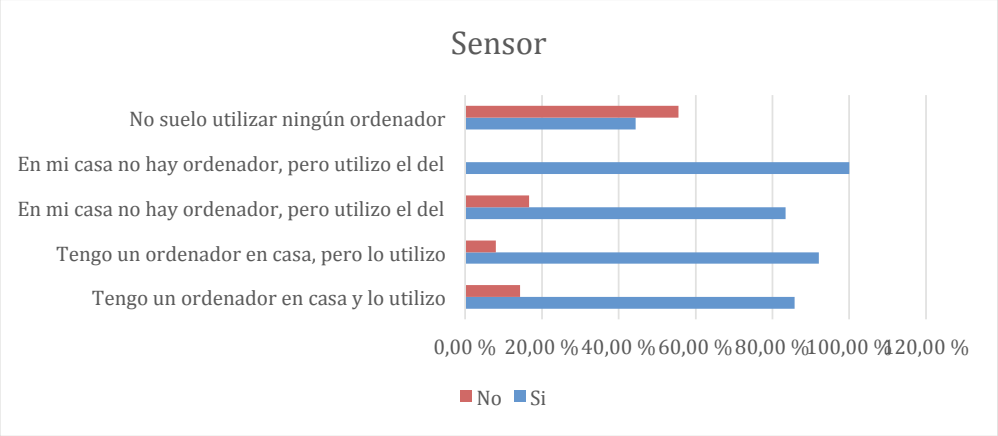


Figura 123. Gráfica de las respuestas obtenidas a la pregunta de si la palabra "sensor" está presente en Scratch, y su relación con los hábitos de uso del ordenador. Fuente: Elaboración propia.

10.3.6.12.25. Análisis univariable: si la palabra "mensaje" está presente en Scratch

La palabra "mensaje" está presente en Scratch dentro de las piezas categorizadas como eventos. La palabra como tal aparece de forma testimonial, pero el concepto de mensaje sí está muy presente, y es el elemento que permite implementar el concepto computacional de paralelismo. En definitiva, la respuesta adecuada a esta pregunta es que sí está presente. Según se muestra en la Tabla 107 y en la *Figura 124*, un 80% responden correctamente.

Tabla 107
Respuestas obtenidas a la pregunta de si la palabra "mensaje" está presente en Scratch.

Opciones	Nº de respuestas	Porcentajes de respuestas
Sí	80	80%
No	20	20%
TOTAL	100	100%

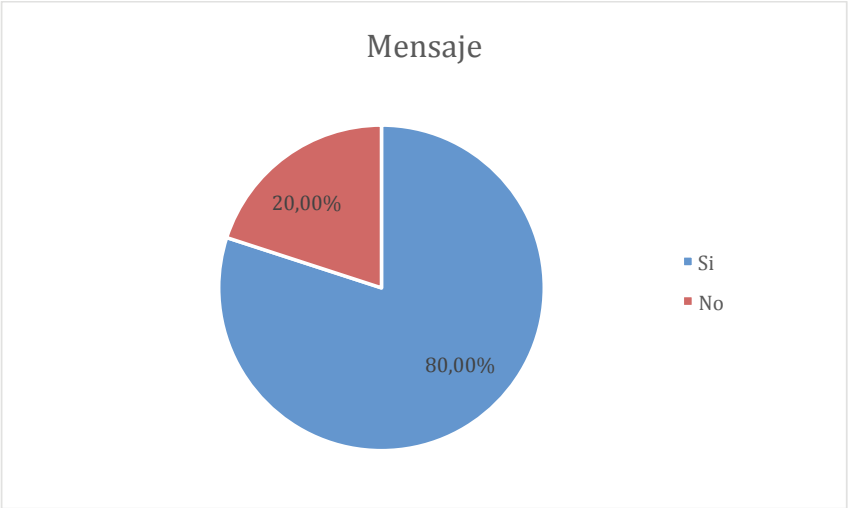


Figura 124. Gráfica de las respuestas obtenidas a la pregunta relativa a si la palabra "mensaje" está presente en Scratch.

Fuente: Elaboración propia.

10.3.6.12.26. Análisis bivariable: si la palabra "mensaje" está presente en Scratch y clases recibidas

Por la similitud de los valores mostrados en la Tabla 108 y en la *Figura 125*, tanto entre las respuestas de los que han recibido más clases que la media, como en los

que menos, no se puede identificar de forma clara la relación entre estas dos variables.

Tabla 108
Respuestas obtenidas a la pregunta de si la palabra "mensaje" está presente en Scratch, y su relación con el número de clases recibidas previamente.

	Sí		No		TOTAL	
Clases recibidas	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**
Más clases que la media (Mayor)	40	85,11%	7	14,89%	47	100%
Menos clases que la media (Menor)	40	75,47%	13	24,53%	53	100%
TOTAL	80	80%	20	20%	100	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla.

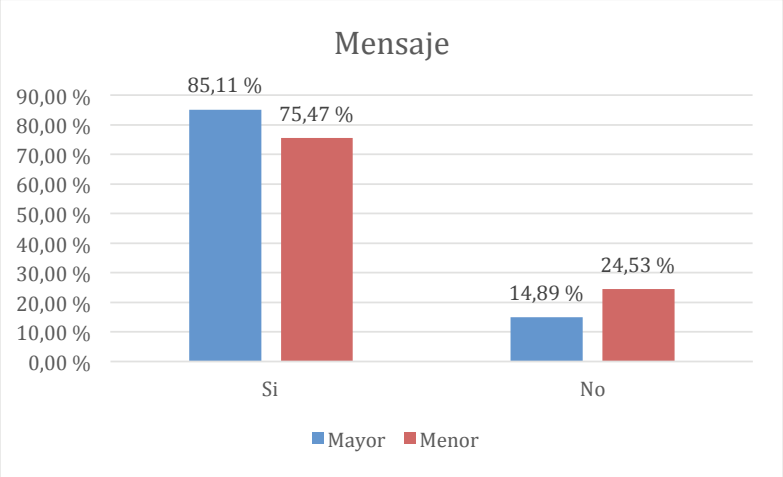


Figura 125. Gráfica de las respuestas obtenidas a la pregunta relativa a si la palabra "mensaje" está presente en Scratch, y su relación con el número de clases recibidas previamente.

Fuente: Elaboración propia.

10.3.6.12.27. Análisis bivariable: si la palabra "mensaje" está presente en Scratch y práctica autónoma

Al igual que en análisis bivariable anterior, la similitud de los porcentajes mostrados en la Tabla 109 y en la *Figura 126*, entre las respuestas de los que sí han tenido práctica autónoma y los que no, indica que a priori no parece haber relación entre estas dos variables.

Tabla 109
Respuestas obtenidas a la pregunta de si la palabra "mensaje" está presente en Scratch, y su relación con la práctica autónoma.

Práctica autónoma	**Sí**		**No**		**TOTAL**	
	Nº	**%**	**Nº**	**%**	**Nº**	**%**
Sí	52	78,79%	14	21,21%	66	100%
No	27	81,82%	6	18,18%	33	100%
TOTAL	79	79,80%	20	20,20%	99	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla

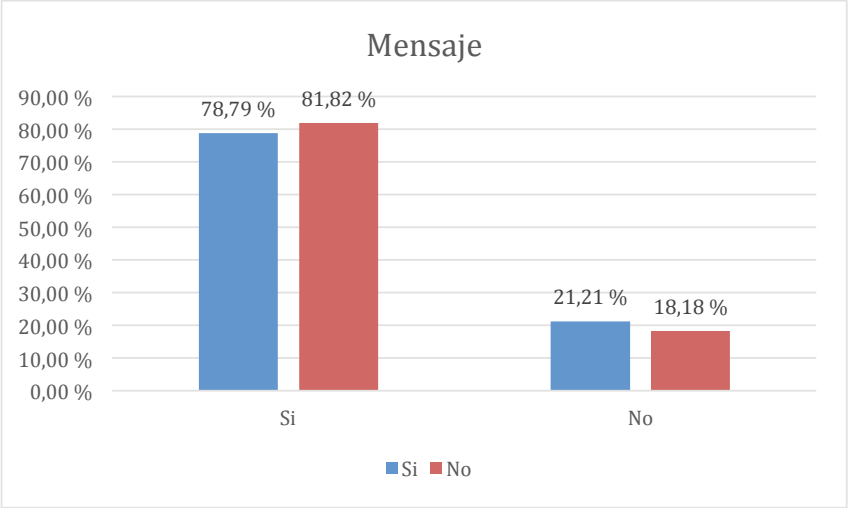


Figura 126. Gráfica de las respuestas obtenidas a la pregunta relativa a si la palabra "mensaje" está presente en Scratch, y su relación con la práctica autónoma.

Fuente: Elaboración propia.

10.3.6.12.28. Análisis bivariable: si la palabra "mensaje" está presente en Scratch y hábitos de uso del ordenador

Tampoco parece haber relación entre la capacidad para identificar la presencia de los mensajes en Scratch, y los hábitos de uso del ordenador, como se puede apreciar en la Tabla 110 y en la *Figura 127*. Los participantes que respondieron a cada una de las opciones están distribuidos de forma equilibrada entre prácticamente todos los grupos de hábitos de uso.

Tabla 110
Respuestas obtenidas a la pregunta de si está la palabra "mensaje"" en el programa, y su relación con los hábitos de uso del ordenador.

	Sí		No		TOTAL	
Hábitos de uso del ordenador	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**
Tengo un ordenador en casa y lo utilizo habitualmente (todos o casi todos los días)	26	74,29%	9	25,71%	35	100%
Tengo un ordenador en casa, pero lo utilizo poco (una o dos veces por semana o menos)	38	80,85%	9	19,15%	47	100%
En mi casa no hay ordenador, pero utilizo el del colegio, o el de un familiar o amigo, habitualmente (todos o casi todos los días)	6	100%	0	0%	6	100%
En mi casa no hay ordenador, pero utilizo el del colegio, o el de un familiar o amigo, de vez en cuando (una o dos veces por semana o menos)	2	66,67%	1	33,33%	3	100%
No suelo utilizar ningún ordenador	8	88,89%	1	11,11%	9	100%
TOTAL	80	80%	20	20%	100	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla.

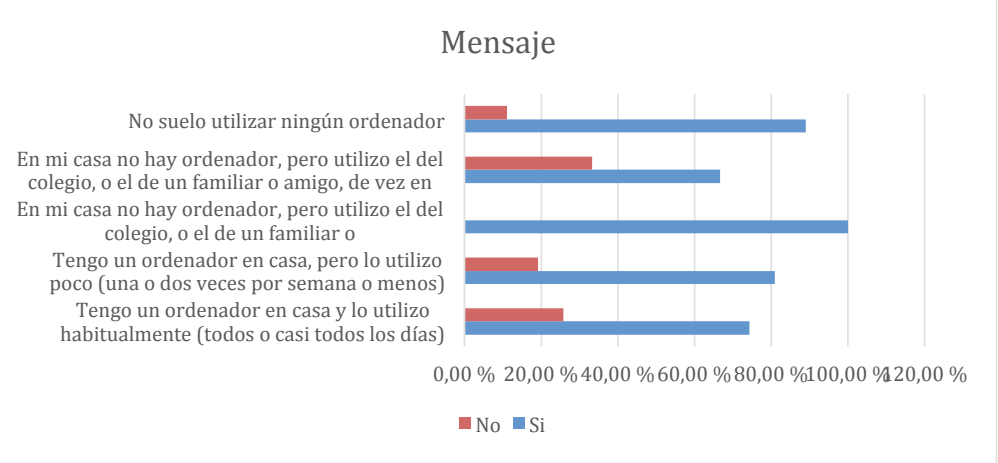


Figura 127. Gráfica de las respuestas obtenidas a la pregunta de si la palabra "mensaje" está presente en Scratch, y su relación con los hábitos de uso del ordenador.

Fuente: Elaboración propia.

10.3.6.12.29. Conclusiones a la pregunta "¿Sabrías decirme cuáles de estas palabras están en el programa?

La Pregunta 17, por un lado está enfocada a la evaluación del lenguaje utilizado en Scratch, y surge de las pautas del análisis heurístico NNGG2, MIT2, MIT8 y MIT9. Por otro lado, es una de las preguntas que busca evaluar la comprensión de los conceptos computacionales básicos (ver capítulo 10.3.1.1).

Con respecto a estos últimos, parece ser que los participantes reconocen la presencia en Scratch de estos conceptos relacionados con la programación y la informática. En cualquier caso, esta pregunta no puede considerarse definitiva como para poder afirmar que dichos conceptos se comprenden y se saben utilizar. Para esto existen otras preguntas que complementan a esta, como la que cuestiona sobre los bloques que se pueden o no encajar.

Las pautas que dan origen a la pregunta son las mismas que las de la Pregunta 16, y como se ha mencionado, pretenden evaluar el lenguaje utilizado en Scratch. Las respuestas a la Pregunta 17 vienen a confirmar lo que ya se apuntaba en la Pregunta 16, la mayoría de los encuestados identifican correctamente las palabras que se encuentran dentro de la aplicación de Scratch, lo que indicaría el cumplimiento de las pautas evaluadas.

10.3.6.13. ¿Has podido crear el juego que tenías pensado?

10.3.6.13.1. Análisis univariable: si ha podido crear el juego que se tenía pensado

Según los datos mostrados en la Tabla 111 y en la *Figura 128*, un 53,40% han podido hacer el juego completo, y un 32,04% algo que se acerca mucho a lo que querían hacer. Es decir, son minoría los que no han podido crear el juego que tenían pensado.

Tabla 111
Respuestas obtenidas a la pregunta de si se ha podido crear el juego que se tenía pensado.

Opciones	Nº de respuestas	Porcentaje de respuestas
Si, ha quedado como yo quería	55	53,40%
Se parece mucho a lo que quería hacer, pero faltarían cosas	33	32,04%
Es diferente, porque no he podido hacer lo que quería	4	3,88%
No he podido hacer un juego completo	11	10,68%
TOTAL	103	100%

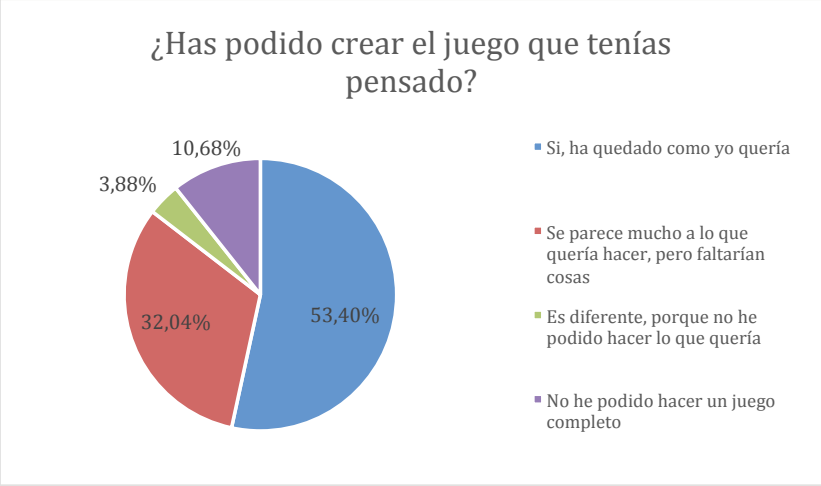


Figura 128. Gráfica de las respuestas obtenidas a la pregunta relativa a si se ha podido crear el juego que se tenía pensado.

Fuente: Elaboración propia.

10.3.6.13.2. Análisis bivariable: si ha podido crear el juego que se tenía pensado y clases recibidas

En la Tabla 112 y en la *Figura 129* se ve un paralelismo entre las respuestas de los que han recibido más clases que la media y los que menos. Por tanto, no se puede establecer una relación clara entre haber podido crear un juego, y haber recibido más clases.

Tabla 112
Respuestas obtenidas a la pregunta de si se ha podido crear el juego que se tenía pensado, y su relación con el número de clases recibidas previamente.

	Sí, ha quedado como yo quería		**Se parece mucho a lo que quería hacer, pero faltarían cosas**		**Es diferente, porque no he podido hacer lo que quería**		**No he podido hacer un juego completo**		**TOTAL**	
Clases recibidas	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**
Más clases que la media (Mayor)	26	54,17%	15	31,25%	1	2,08%	6	12,50%	48	100%
Menos clases que la media (Menor)	29	52,73%	18	32,73%	3	5,45%	5	9,09%	55	100%
TOTAL	55	53,40%	33	32,04%	4	3,88%	11	10,68%	103	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla.

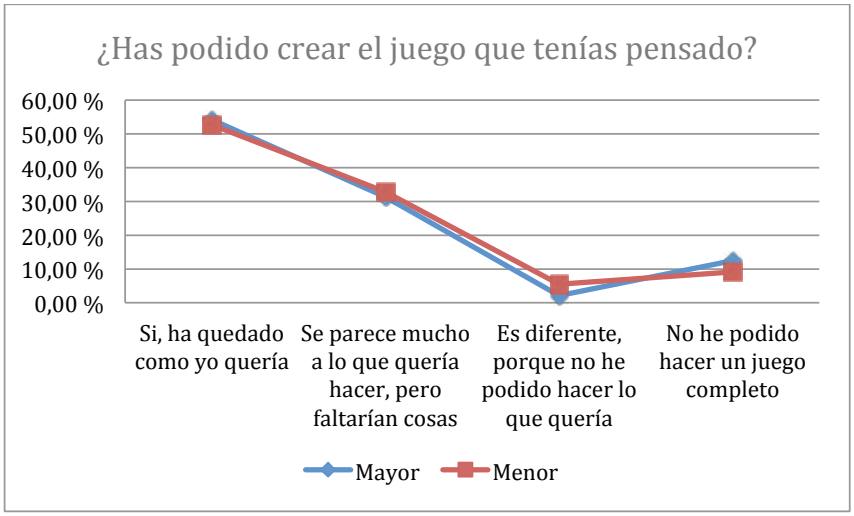


Figura 129. Gráfica de las respuestas obtenidas a la pregunta relativa a si se ha podido crear el juego que se tenía pensado, y su relación con el número de clases recibidas previamente.

Fuente: Elaboración propia.

10.3.6.13.3. Análisis bivariable: si ha podido crear el juego que se tenía pensado y práctica autónoma

Los datos de la Tabla 113 y de la *Figura 130* muestran una dispersión de los datos que indica que no hay una relación clara entre haber podido crear el juego, y la práctica autónoma.

Tabla 113
Respuestas obtenidas a la pregunta de si se ha podido crear el juego que se tenía pensado, y su relación con la práctica autónoma.

	Sí, ha quedado como yo quería		**Se parece mucho a lo que quería hacer, pero faltarían cosas**		**Es diferente, porque no he podido hacer lo que quería**		**No he podido hacer un juego completo**		**TOTAL**	
Práctica autónoma	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**
Sí	40	57,97%	25	36,23%	1	1,45%	3	4,35%	69	100%
No	14	42,42%	8	24,24%	3	9,09%	8	24,24%	33	100%
TOTAL	54	52,94%	33	32,35%	4	3,92%	11	10,78%	102	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla

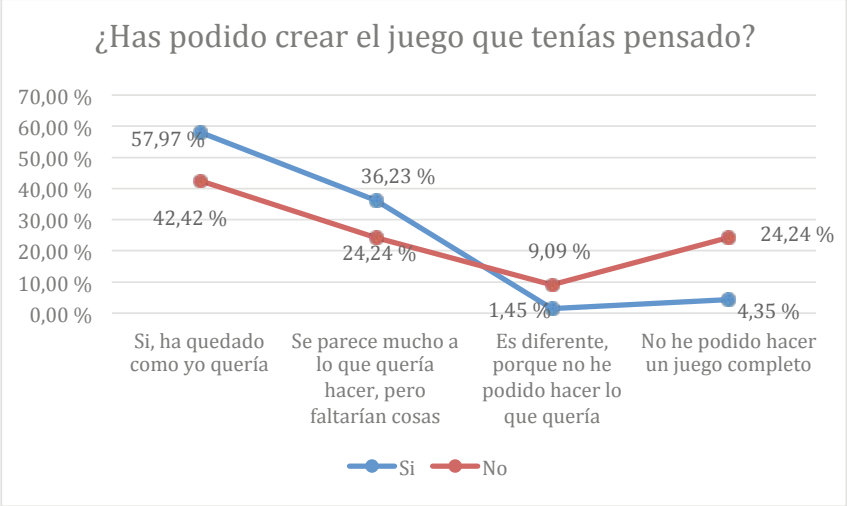


Figura 130. Gráfica de las respuestas obtenidas a la pregunta relativa a si ha podido crear el juego que tenía pensado, y su relación con la práctica autónoma.

Fuente: Elaboración propia.

10.3.6.13.4. Análisis bivariable: si ha podido crear el juego que se tenía pensado y hábitos de uso del ordenador

Lo más destacable de lo que mostrado en la Tabla 114 y en la *Figura 131* es que hay una cierta tendencia a que los participantes que no han podido hacer un juego completo, sean mayoría los que no tienen ordenador en casa.

Tabla 114
Respuestas obtenidas a la pregunta de si se ha podido crear el juego que se tenía pensado, y su relación con los hábitos de uso del ordenador.

	Sí, ha quedado como yo quería		**Se parece mucho a lo que quería hacer, pero faltarían cosas**		**Es diferente, porque no he podido hacer lo que quería**		**No he podido hacer un juego completo**		**TOTAL**	
Hábitos de uso del ordenador	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**
Tengo un ordenador en casa y lo utilizo habitualmente (todos o casi todos los días)	19	54,29%	14	40%	1	2,86%	1	2,86%	35	100%
Tengo un ordenador en casa, pero lo utilizo poco (una o dos veces por semana o menos)	25	50%	19	38%	2	4%	4	8%	50	100%

	Sí, ha quedado como yo quería		Se parece mucho a lo que quería hacer, pero faltarían cosas		Es diferente, porque no he podido hacer lo que quería		No he podido hacer un juego completo		TOTAL	
Hábitos de uso del ordenador	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**
En mi casa no hay ordenador, pero utilizo el del colegio, o el de un familiar o amigo, habitualmente (todos o casi todos los días)	2	33,33%	0	0%	1	16,67%	3	50%	6	100%
En mi casa no hay ordenador, pero utilizo el del colegio, o el de un familiar o amigo, de vez en cuando (una o dos veces por semana o menos)	3	100%	0	0%	0	0%	0	0%	3	100%
No suelo utilizar ningún ordenador.	6	66,67%	0	0%	0	0%	3	33,33%	9	100%
TOTAL	55	53,40%	33	32,04%	4	3,88%	11	10,68%	103	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla.

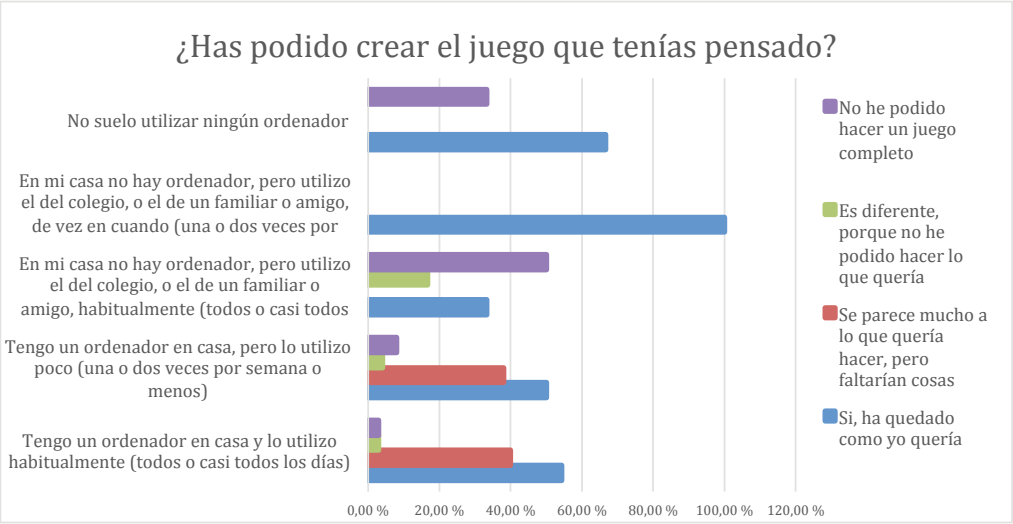


Figura 131. Gráfica de las respuestas obtenidas a la pregunta relativa a si se ha podido crear el juego que se tenía pensado, y su relación con los hábitos de uso del ordenador.

Fuente: Elaboración propia.

10.3.6.13.5. Conclusiones a la pregunta "¿Has podido crear el juego que tenías pensado?"

La Pregunta 18 pretende evaluar la consecución de los objetivos de la práctica que llevan a cabo los participantes cuando responden al cuestionario. Además se relaciona con uno de los estándares de aprendizaje indicados por la Comunidad de Madrid, que determina la capacidad que debe tener el estudiante para evaluar los resultados del programa realizado (ver capítulo 10.3.1.2). Los resultados indican que la mayoría de los participantes han podido realizar el juego que querían, o algo muy parecido.

Asimismo, esta pregunta procede de las pautas del análisis heurístico NNGG3, NNGG7, HHS1.3, HHS1.8 y NOKIA6.

Las pautas NNGG3 y NNGG7 evalúan el control del usuario sobre el programa, y la eficiencia de su uso, respectivamente. Un 10,88% no han podido realizar ningún tipo de juego, y un 3,88% no han podido hacer lo que querían. Es decir, son una minoría los que no han tenido el control sobre el programa y no han podido cumplir con sus objetivos. A la vista de estos datos, podríamos decir que las pautas NNGG3 y NNGG7 se cumplen.

Las pautas HHS1.3, HHS1.8 evalúan que el programa se adapta a las expectativas del usuario, y sirve para el propósito para el que fue concebida. Dado que la mayoría de los encuestados consiguen llevar a cabo lo que pretendían, podríamos decir que estas pautas se cumplen.

La pauta NOKIA6 tiene que ver con la motivación que ofrece el programa al usuario. Como hemos mencionado en otras ocasiones, poder ver en funcionamiento el videojuego que uno mismo ha ideado e implementado es algo realmente motivante, y según las respuestas recogidas, una gran mayoría ha podido hacerlo.

10.3.6.14. ¿Sabrías abrir un nuevo juego, elegir un personaje y hacer que se mueva, en unos minutos, y sin que nadie te ayude?

10.3.6.14.1. Análisis univariable: abrir un nuevo juego

En la Tabla 115 y en la *Figura 132* se puede ver solo un encuestado manifiesta que no va a poder crear un nuevo juego. También se ve que en torno un 70% afirma que lo podría hacer sin ayuda.

Tabla 115
Respuestas obtenidas a la pregunta de si se ha podido abrir un nuevo juego, elegir un personaje y hacer que se mueva sin ayuda

Opciones	Nº de respuestas	Porcentaje de respuestas
Si, sabría hacerlo solo	71	68,93%
Si, pero a lo mejor me tendrían que ayudar	23	22,33%
Me tendría que ayudar el profesor	8	7,77%
No	1	0,97%
Total	103	100%

Figura 132. Gráfica de las respuestas relativas a la pregunta de si se ha podido abrir un nuevo juego, elegir un personaje y hacer que se mueva sin ayuda

Fuente: Elaboración propia.

10.3.6.14.2. Análisis bivariable: abrir un nuevo juego y clases recibidas

La Tabla 116 y la *Figura 133* muestra una dispersión de los datos, por lo que no parece haber relación entre el número de clases recibidas y la respuesta que los participantes han dado a esta pregunta.

Tabla 116
Respuestas obtenidas a la pregunta de si se ha podido abrir un nuevo juego, elegir un personaje y hacer que se mueva sin ayuda y su relación con el número de clases recibidas previamente.

	Sí, sabría hacerlo solo		**Sí, pero a lo mejor me tendrían que ayudar**		**Me tendría que ayudar el profesor**		**No sabría hacerlo**		**TOTAL**	
Clases recibidas	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**
Más clases que la media (Mayor)	37	77,08%	4	8,33%	6	12,50%	1	2,08%	48	100%
Menos clases que la media (Menor)	34	61,82%	19	34,55%	2	3,64%	0	0%	55	100%
TOTAL	71	68,93%	23	22,33%	8	7,77%	1	0,97%	103	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla.

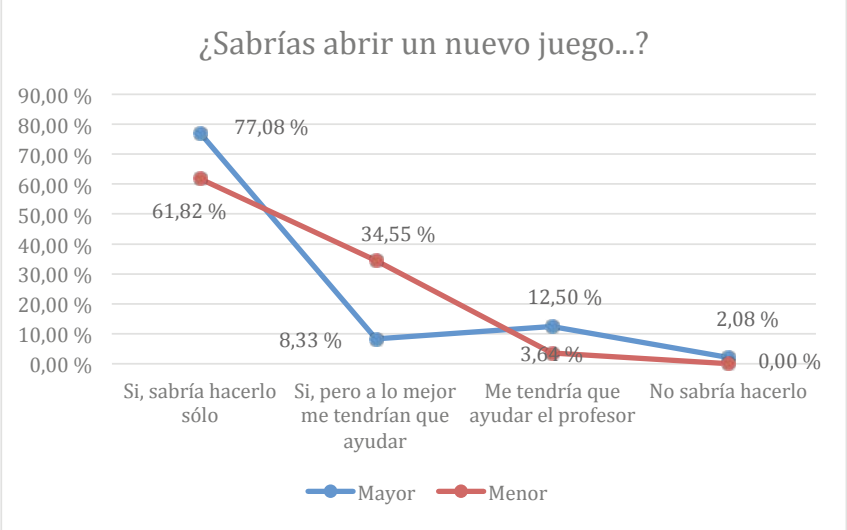


Figura 133. Gráfica de las respuestas obtenidas a la pregunta relativa a si sabría crear un nuevo juego sin ayuda, y su relación con el número de clases recibidas previamente.

Fuente: Elaboración propia.

10.3.6.14.3. Análisis bivariable: abrir un nuevo juego y práctica autónoma

La Tabla 117 y en la *Figura 134* muestran un paralelismo entre los alumnos que han tenido práctica autónoma y los que no la han tenido, con lo que no podemos afirmar que exista relación entre estas dos variables.

Tabla 117
Respuestas obtenidas a la pregunta de si sabría abrir un nuevo juego, elegir un personaje y hacer que se mueva unos minutos sin que nadie ayude, y su relación con la práctica autónoma.

	Si, sabría hacerlo solo		**Si, pero a lo mejor me tendrían que ayudar**		**Me tendría que ayudar el profesor**		**No sabría hacerlo**		**TOTAL**	
Práctica autónoma	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**
Sí	51	73,91%	14	20,29%	3	4,35%	1	1,45%	69	100%
No	19	57,58%	9	27,27%	5	15,15%	0	0%	33	100%
TOTAL	70	68,63%	23	22,55%	8	7,84%	1	0,98%	102	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla.

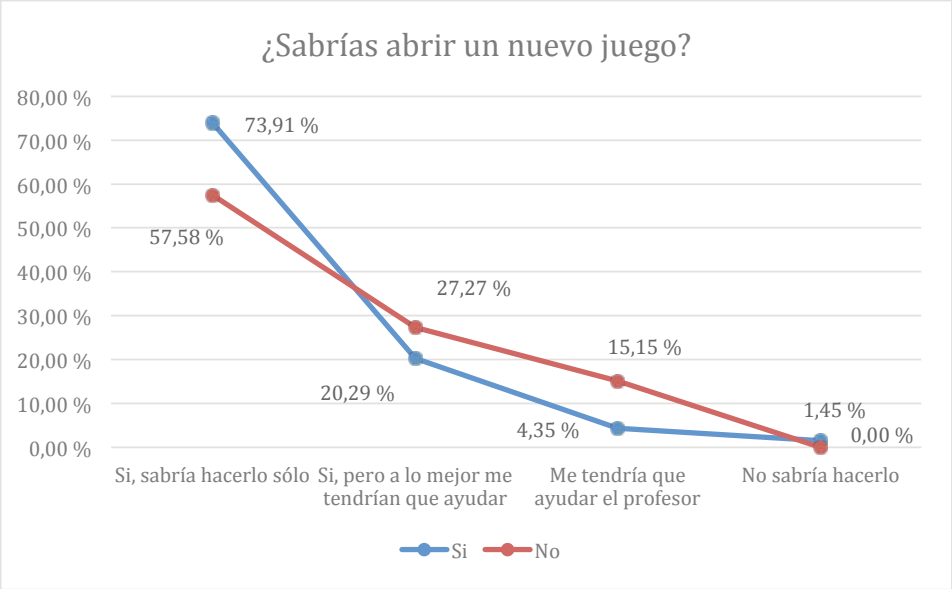


Figura 134. Gráfica de las respuestas obtenidas a la pregunta relativa a si sabría abrir un nuevo juego, elegir un personaje y hacer que se mueva unos minutos sin que nadie ayude, y su relación con la práctica autónoma.

Fuente: Elaboración propia.

10.3.6.14.4. Análisis bivariable: abrir un nuevo juego y hábitos de uso del ordenador

En la Tabla 118 y en la *Figura 135* no hay datos concluyentes que indiquen la relación entre abrir un nuevo juego, y los hábitos de uso del ordenador.

Tabla 118
Respuestas obtenidas a la pregunta de si sabría abrir un nuevo juego, elegir un personaje y hacer que se mueva unos minutos sin que nadie le ayude, y su relación con los hábitos de uso del ordenador.

	Sí, sabría hacerlo solo		**Sí, pero a lo mejor me tendrían que ayudar**		**Me tendría que ayudar el profesor**		**No sabría hacerlo**		**TOTAL**	
Hábitos de uso del ordenador	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**	**Nº**	**%**
Tengo un ordenador en casa y lo utilizo habitualmente (todos o casi todos los días)	26	74,29%	7	20%	2	5,71%	0	0%	35	100%
Tengo un ordenador en casa, pero lo utilizo poco (una o dos veces por semana o menos)	30	60%	16	32%	3	6%	1	2%	50	100%
En mi casa no hay ordenador, pero utilizo el del colegio, o el de un familiar o amigo, habitualmente (todos o casi todos los días)	4	66,67%	0	0%	2	33,33%	0	0%	6	100%
En mi casa no hay ordenador, pero utilizo el del colegio, o el de un familiar o amigo, de vez en cuando (una o dos veces por semana o menos)	3	100%	0	0%	0	0%	0	0%	3	100%
No suelo utilizar ningún ordenador.	8	88,89%	0	0%	1	11,11%	0	0%	9	100%
TOTAL	71	68,93%	23	22,33%	8	7,77%	1	0,97%	103	100%

Nota: en el encabezado de la tabla, Nº representa el número de respuestas recibidas en cada caso, y % representa el porcentaje que ese número supone sobre el total de respuestas para cada fila de la tabla.

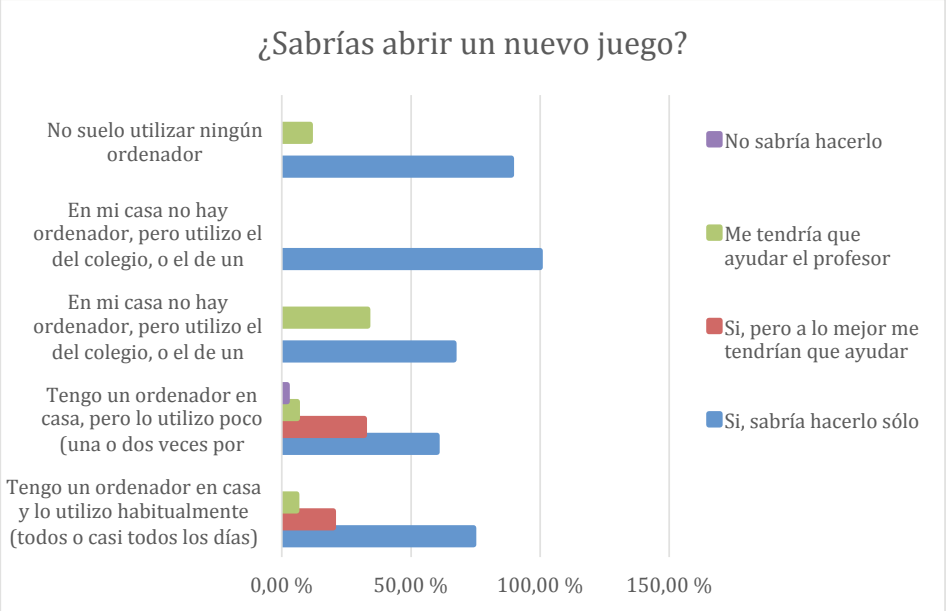


Figura 135. Gráfica de las respuestas obtenidas a la pregunta relativa a si sabría abrir un nuevo juego, y su relación con los hábitos de uso del ordenador.

Fuente: Elaboración propia.

10.3.6.14.5. Conclusiones a la pregunta "¿Sabrías abrir un nuevo juego, elegir un personaje, y hacer que se mueva?"

La Pregunta 26 se realiza a modo de conclusión, para poder evaluar la percepción que los encuestados tienen sobre cuánto dominan el programa. Cerca de un 70% afirman que podrían hacerlo sin problemas, mientras que el resto reconocen que podrían necesitar algo de ayuda.

Esta pregunta también surge del análisis heurístico, de las pautas relacionadas con el manejo de la interfaz, y concretamente de NNGG3, NNGG9, HHS1.6, HHS1.8, HHS2.4, MIT2, MIT6, NOKIA4, NOKIA5, AGUC1, AGUC2 y AGUC3.

Las pautas NNGG3 y NNGG9 evalúan el control del usuario sobre el programa y su capacidad para resolver los errores. La mayoría de los encuestados perciben que no necesitarían ayuda en la creación de un nuevo juego, y que podrían resolver los problemas, luego podríamos decir que estas pautas se cumplen.

Las pautas HHS1.6, HHS1.8, y HHS2.4 evalúan la eficiencia y la funcionalidad del programa. Por las respuestas recogidas, podríamos decir que esta pauta se cumple dado que la mayoría de los participantes manifiestan confianza en su autonomía para manejar Scratch.

La pauta NOKIA4 evalúa la suficiencia del *feedback* que da el programa. Que una amplia mayoría de participantes consideren que podrían hacer un juego solos, puede ser un indicador (aunque no el único) de que esta pauta se cumple. La pauta NOKIA5 evalúa la sencillez para iniciarse en el programa. Igualmente podríamos decir que se cumple, dado que solo un participante comenta que no podría hacerlo.

Las pautas AGUC1, AGUC2 y AGUC3 evalúan la arquitectura de navegación del programa, y que este se ajuste a su finalidad. Como en la evaluación de pautas anteriores, podríamos afirmar que las pautas se cumplen fijándonos en esa mayoría que cree que podría afrontar el reto propuesto.

11. Discusión

Como se explica en el capítulo 10.3.6, a la hora de analizar los datos recogidos en el cuestionario, se hizo inicialmente un análisis univariable, estudiando los datos agregados de las respuestas, para posteriormente hacer un análisis bivariable con factores externos al diseño del programa, que podrían tener influencia en las respuestas (número de clases recibidas, práctica autónoma con Scratch, y hábitos de uso del ordenador). La discusión se va a realizar en este mismo orden.

11.1. Análisis univariable

La primera batería de preguntas se corresponde con las acciones que se pueden hacer con Scratch: elegir un fondo, elegir un personaje, mover un personaje, etc. Lo primero que se observa es que la mayoría de los encuestados pueden hacer las acciones de forma independiente. En algunas acciones concretas, necesitan probar un rato (la segunda opción), pero en general, más del 50% en todos los casos, eligen las opciones que pueden hacer por sí solos. Es cierto que según va aumentando la dificultad de las tareas (hacer que el personaje se mueva, sumar puntos) aumentan también las respuestas a la segunda opción de las preguntas, incluso se precisa la ayuda del profesor. Estas acciones que denominamos más complejas son las que requieren de lógica de programación, y en estos casos alrededor de la tercera parte de los niños, precisan ayuda del docente.

Merece la pena fijar la atención en las respuestas a la pregunta relativa a sumar puntos (0). Para realizar esta acción en el videojuego se requiere crear una variable, y programar unas determinadas secuencias de acciones que incluyen sentencias condicionales. Solo la cuarta parte de los participantes ha optado por respuestas autónomas. Un 5,83% responde que podría hacerlo solo y a la primera. Un 21,36% ha podido solo después de probar un rato, pero en este caso a las tres cuartas partes de los participantes, o les han tenido que ayudar o no han sido capaces de hacerlo (es muy significativo que el 36,89% no han podido).

Por lo tanto, observamos que las acciones que guardan más relación con el entorno gráfico, es decir, abrir un documento, abrir una imagen, pegarla en un fondo, etc., los participantes las manejan sin problemas. Estas acciones guardan similitud con lo que se puede realizar con otros programas informáticos, como por ejemplo Microsoft Word, es decir, con lo que podríamos denominar el uso cotidiano que se puede hacer con un ordenador. En estos casos, la usabilidad es transparente, realizar estas acciones es algo natural para los encuestados. En el momento que se necesita añadir lógica de programación (por ejemplo, mover el personaje), empiezan a surgir las dificultades, aunque todavía la mayoría lo pueden hacer. Sin embargo,

en tareas como crear una variable o utilizar una sentencia condicional, propias de la acción de sumar puntos, el papel del profesor es fundamental. De hecho, la opción "sí, pero me han tenido que ayudar", es la que mayor volumen de respuestas aglutina, con un 36,89%. En cierto modo, estas son las respuestas que se preveía tener: a mayor dificultad, mayor necesidad de ayuda. Según la tarea se aproxima a la dificultad intrínseca de la programación, el software por sí mismo no soluciona el problema. Para que el estudiante llegue a esa solución debe existir una formación previa por parte del docente.

Después de sumar puntos, se pregunta por incluir sonidos. Tal vez esta pregunta debería ir antes, por guardar más relación con las acciones de añadir elementos que no requieren de programación. En este caso se vuelve a observar que más de un 50% de los participantes puede hacerlo de forma autónoma.

En el caso de la pregunta que cuestiona sobre si se ha podido guardar el juego, un abrumador 95% lo realiza sin dificultades. De nuevo nos encontramos con una opción relacionada con el uso habitual del ordenador que no genera problemas de usabilidad. Este tipo de acciones se relacionan con el concepto de intuitivo, entendiendo que algo resulta intuitivo cuando se sabe hacer sin haber tenido un proceso previo de aprendizaje. No significa que se exija un movimiento natural, sino que son acciones parecidas a las que hemos llevado a cabo previamente con otras tecnologías (Conde Melguizo, Muñoz Muñoz y González González, 2008). Es decir, si estamos familiarizados con tecnologías similares que hemos utilizado antes, podremos manejar de forma más o menos intuitiva la tecnología presente, según el grado de similitud. En contraposición, las acciones con las que no estemos familiarizados, requerirán de una adecuada usabilidad para que puedan ser llevadas a cabo sin problemas.

En conclusión, a tenor de las preguntas analizadas hasta ahora, las acciones que tienen que ver con el manejo de otras tecnologías (guardar el juego, etc.) se realizan sin problemas, porque a una mayoría le resulta intuitivas. Las acciones específicas de Scratch (añadir un personaje, añadir un sonido, etc.) que por lo general no tienen que ver con tecnologías que se puedan haber usado previamente, se resuelven correctamente por la adecuación de la usabilidad que presenta la herramienta. Las acciones que requieren de programación no se resuelven simplemente con la usabilidad del programa, ni a través de la intuición, en ese momento el profesor se presenta como un factor determinante para que la tarea se pueda llevar a su consecución.

Cuando se pregunta a los participantes si han incluido elementos creados por ellos mismos, solo un 10% responden que no, que solo ha utilizado elementos propios de Scratch. Más de la mitad han utilizado dibujos y sonidos, y solo sonidos un 2,94%.

Parece lógico pensar que a los participantes les ha resultado más intuitivo y natural incluir dibujos en una interfaz en lo que lo primero que te encuentras es el dibujo del gato, icono del programa. Esto plantea otra posible línea de investigación a futuro: analizar la predominancia de lo visual frente a lo sonoro en este tipo de entornos.

Hay dos preguntas que evalúan, ya no tanto el uso de Scratch, sino la experiencia que proporciona al usuario, y el grado de satisfacción que genera su uso. La primera de estas preguntas es si han podido jugar al juego que han creado. La segunda, si jugar al juego les ha resultado divertido. Un 86,41% han podido jugar a su juego, y a un 95% le ha resultado divertido. Son porcentajes que permiten afirmar el cumplimiento de las pautas de usabilidad relativas a la satisfacción con el uso programa, y a los elementos motivadores que este incluye.

La siguiente pregunta evalúa si los participantes comprenden la lógica de combinación de bloques, y les pide que digan cuánto han tardado en descubrirla. Son mayoría los que responden que han comprendido dicha lógica de forma autónoma: un 25,49% responden que tras uno o dos intentos, y un 45,10% que después de probar un rato. En el otro extremo, encontramos un tercio de los participantes que han necesitado la ayuda del profesor.

Para complementar las conclusiones que se puedan sacar de estas respuestas, a continuación se analiza una serie de preguntas donde se presenta a los participantes una serie de bloques, agrupados por parejas, algunos de los cuales se pueden combinar, y otros no. En esta pregunta se les pide que identifiquen si es posible dicha combinación. De esta manera confirmarían lo que respondieron en la pregunta anterior, es decir, que efectivamente comprenden la lógica de bloques. A continuación se recupera la gráfica que ilustraba la pregunta del cuestionario:

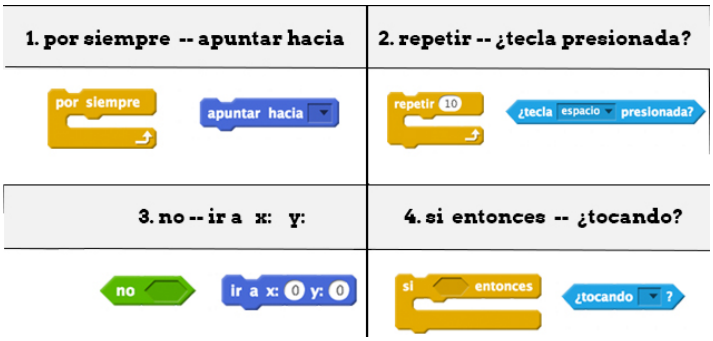


Figura 136. Imagen que en el formulario acompaña a la Pregunta 10.

Fuente: elaboración propia

- Con respecto a la pareja 1 [por siempre – apuntar hacia], que sí se pueden combinar, un 97,62% responde correctamente.
- Con respecto a la pareja 2. [repetir - ¿tecla presionada?], que no se pueden combinar, un 89,29% responden correctamente.
- Con respecto a la pareja 3 [no – ir a x: y:], que no se pueden combinar, un 69,05% responden correctamente.
- Con respecto a la pareja 4 [si entonces - ¿tocando?], que sí se pueden combinar, un 84% responden correctamente.

Según estos datos, podemos concluir que la lógica de bloques se comprende. Esta comprensión tiene mucho que ver con lo intuitivo que resulta unir piezas, que se encajan como si se tratara de un rompecabezas. Con esta técnica Scratch resuelve los problemas que se presentan en el aprendizaje de un lenguaje de programación textual, derivados de la rigidez y de la dificultad intrínseca a la sintaxis de dicho lenguaje. En cualquier caso, entender la lógica de bloques no es un factor determinante que permita afirmar que los conceptos de programación se comprenden. De hecho, las respuestas a las primeras preguntas analizadas en este capítulo apuntaban hacia lo contrario.

Dentro del cuestionario hay un bloque de preguntas que evalúan la capacidad de los participantes para resolver los problemas que puedan surgir en el uso de Scratch. Un 82,52% de los participantes se han encontrado en algo en algún momento que no funcionaba como ellos querían, mientras que para un 17,48% todo ha ido bien.

Cuando se pregunta a ese 82,52% cómo resolvieron los problemas que se encontraron, un 23,17% lo resuelven solos, y un 13,41% recurren a la ayuda del programa. Sin embargo, la opción mayoritaria es en la que se pide ayuda al profesor, en alguna de las dos opciones planteadas. Un 31,71% responde “he pedido ayuda al profesor y hemos utilizado la ayuda del programa”, y un 31,71% “he pedido ayuda al profesor y me lo ha resuelto”. Es decir, más del 60% de los participantes han pedido ayuda al profesor. Estos datos vienen a reforzar la idea planteada anteriormente sobre el papel primordial del docente. Cuando la interacción es sencilla, predomina la autonomía del niño. Cuando se precisa un aprendizaje de elementos más complejos de computación, el docente se vuelve imprescindible.

En la línea de evaluar la capacidad de los participantes para resolver los problemas acaecidos, la siguiente pregunta les cuestiona si han conseguido hacer el programa que se proponían. A un 64,29% les funciona perfectamente, y un 29,76% han resuelto más de la mitad de los problemas. Tan solo un 5,95% resuelven menos de la mitad, y ningún participante escoge la opción de “No funciona”. Es decir, Scratch

se presenta como una herramienta funcional, y en general los participantes se muestran satisfechos con su uso.

Otro aspecto importante a evaluar dentro de la usabilidad, es la utilización de un lenguaje adecuado dentro del programa, con el que el usuario esté familiarizado. Con este propósito se plantean varias preguntas. Por ejemplo, una de ellas dice "¿Había algunas palabras en Scratch que no sabías lo que significaba?" El 61,17% responde que ha entendido todas a la primera, y el 38,83% manifiesta que había algunas palabras que no entendía para qué servían. Es bastante relevante que casi el 40% haya encontrado palabras cuyo significado desconocía, aunque sigan siendo mayoría los que dicen que sí han entendido todo.

Tratando de ahondar en esta cuestión, el formulario incluye una pregunta donde se le pide a los participantes que identifiquen si determinadas palabras están presentes en Scratch. Además, de la evaluación del lenguaje, todas las palabras por las que cuestiona esta pregunta (excepto objeto) tienen que ver con conceptos computacionales. A continuación se enumeran las respuestas obtenidas:

- Palabra "evento", que sí está presente. El 52,48% responde correctamente.
- Palabra "variable", que sí está presente. El 72,55% responde correctamente.
- Palabra "interfaz", que no está presente. El 81%, responde correctamente.
- Palabra "operador", que sí está presente. El 63%, responde correctamente.
- Palabra "objeto", que sí está presente. El 90%, responde correctamente.
- Palabra "sensor", que sí está presente. El 85,44%, responde correctamente.
- Palabra "mensaje", que sí está presente. El 80%, responde correctamente.

En definitiva, a la vista de los datos obtenidos en el análisis univariable, podemos decir que los participantes han aprendido un procedimiento, y han conseguido hacer un programa. La mayoría han podido crear el juego que tenían pensado, o algo muy parecido. También han podido jugar a su juego, y les ha resultado divertido hacerlo. Pero que esto signifique que también hayan asimilado conceptos relacionados con la programación, y que en el futuro sean capaces de llevar a cabo otros desarrollos, es más que cuestionable. No hay resultados concluyentes al respecto, y habría que seguir profundizando en este tema, para ver si estos mismos participantes son capaces de enfrentarse a otros programas diferentes. Los resultados evidencian algo que en cierto modo ya se esperaba: no es lo mismo saber manejar Scratch, que saber programar.

11.2. Análisis bivariable

Como se ha explicado en el capítulo 10.3.6, además del análisis univariable, se cruzan los datos con otras tres variables, para tratar de averiguar si más allá del

diseño del programa, que es lo que se evalúa en un estudio de usabilidad, existen elementos externos que condicionan las respuestas del cuestionario. Las tres variables que se estudian son las siguientes:

- *Número de clases recibidas previamente a la realización de la actividad*. Partimos de la hipótesis de que, a mayor número de clases, mayor será el aprendizaje.
- *Práctica autónoma*. En teoría, si se ha practicado fuera del aula, se tendrá más experiencia en el manejo de la herramienta, se podrán hacer tareas más complejas, etc.
- *Hábitos de uso del ordenador*. Se plantea la hipótesis de que, si se dispone de ordenador y se utiliza habitualmente, se estará más familiarizado con los procesos de interacción informática.

11.2.1. Número de clases recibidas

Con respecto al volumen de clases recibidas, se calculó la media de clases que habían recibido los participantes, y se les dividió en dos grupos: los que habían recibido más clases que la media, y los que habían recibido menos. Una vez hecho esto, se volvió a analizar todas las preguntas, observando los resultados diferenciados de los que están por encima de la media y los que están por debajo.

En las primeras preguntas, que evalúan la capacidad de los encuestados para hacer ciertas tareas (elegir un fondo, insertar un sonido, mover un personaje, hacer que se sumen puntos), el número de clases recibidas no es relevante. En los gráficos presentados en el capítulo 10.3.6.1, las líneas que representan los que más y menos clases han recibido son prácticamente paralelas en casi todos los casos, solo hay una pequeña diferencia en el gráfico de incluir sonidos, donde los estudiantes que menos clases han recibido son los que más ayuda del profesor han necesitado. En cualquier caso, es un dato aislado, y en general no parece haber demasiada influencia del número de clases recibidas a la hora de saber realizar las acciones planteadas.

Pasamos a las preguntas que miden la satisfacción respecto al software (si se ha podido jugar al juego, y si ha resultado divertido). De nuevo no parece que haya influencia del número de clases recibidas, la tendencia es prácticamente la misma. De hecho, las relaciones que aparecen serían contrarias a la hipótesis, por ejemplo, entre los que no han podido jugar al juego, son mayoría los que han recibido más clases que la media. En cualquier caso, los que no han podido jugar son un 4,49%, por lo que esta relación no puede considerarse significativa. En cuanto a si se han divertido, contestan afirmativamente el 100% de los que han recibido más clases. En definitiva, la relación entre el número de clases y la satisfacción con el software no puede establecerse.

Esta tónica se mantiene en el resto de preguntas sobre la adecuación del lenguaje, y sobre la resolución de problemas: en prácticamente ningún caso el número de clases genera diferencias significativas. Existen casos curiosos, como en la pregunta que se les pide identificar si ciertas palabras se encuentran en Scratch (variable, operador, etc.), donde paradójicamente se observa que los que han recibido menos clases aciertan más. El número de clases en este trabajo de campo no ha sido relevante.

Observando los resultados podemos concluir que el número de clases recibidas no resulta determinante en las respuestas. Esta afirmación la hacemos con la reserva de que se ha hecho esta medición en base a la media de clases recibidas por todos los participantes, y que en algunos casos, la diferencia entre los que han recibido más clases que la media, y los que han recibido menos, puede ser de dos o tres clases.

11.2.2. Práctica autónoma

Cabe decir que cuando se pregunta al participante si ha tenido práctica autónoma con Scratch, no sabemos exactamente en qué ha consistido esta práctica. Esto es una limitación que tiene la investigación.

Con respecto a las preguntas sobre las acciones y el uso del programa, se observan en casi todos los casos resultados prácticamente paralelos entre los que sí han tenido esta práctica, y los que no. Por tanto, no se puede afirmar que haya una relación clara entre haber practicado en casa y el uso del programa. Esta misma tendencia se puede trasladar a todas las preguntas del cuestionario. Solo hay dos casos que se salen de esta norma:

El primer caso es que hay una cuarta parte de los que no han tenido práctica autónoma, que tampoco han podido jugar al juego. Aquí se podría decir que practicar en casa ha influido en que hayan terminado el trabajo, pero por el volumen de respuestas (solo 8 estudiantes están en este caso) no son datos concluyentes.

El segundo caso se sitúa en la pregunta que cuestiona sobre la manera de resolver algo cuando no funcionaba como se quería. Los participantes que han tenido práctica autónoma son mayoría entre los que resuelven los problemas sin ayuda, mientras los que no han practicado fuera del aula, tienden a responder que han necesitado la ayuda del programa, del profesor, o de ambos. En cualquier caso, no hay diferencias muy exacerbadas, y la tendencia no es definitiva. Lo que sí es claro es que más del 62% de los encuestados manifiestan la necesidad de recurrir al

profesor, porque de forma autónoma no han podido resolver todos los problemas que se les han presentado.

11.2.3. Hábitos de uso del ordenador

El último análisis, en el que se cruzan las respuestas de los encuestados según si disponen de ordenador, es tal vez el más interesante. Para este análisis partimos de la hipótesis de que aquel que utiliza el ordenador habitualmente, le será más sencillo adaptarse a la utilización de Scratch.

Esta hipótesis se sustentaría en las respuestas obtenidas a las preguntas relacionadas con acciones que se pueden encontrar en otros programas (abrir un nuevo proyecto, guardarlo, etc.), que una amplia mayoría de los encuestados realizan sin problemas. Sin embargo, en las tareas que tienen que ver con programación, el porcentaje de los que han tenido éxito se reduce de forma drástica. Por eso, este análisis bivariable busca comprobar si disponer de ordenador solo influye en las tareas relacionadas con el uso cotidiano del ordenador, o por el contrario influye en todo el proceso de aprendizaje.

El 90% de los encuestados disponen de ordenador en casa, y son muy pocos los que no lo tienen. Esto de nuevo plantea una limitación dentro de la investigación.

Al cruzar las distintas respuestas con los alumnos que no disponen de ordenador, o no lo utilizan, se ven diferencias, pero estas no apuntan a ninguna tendencia clara. Por ejemplo, en las preguntas relacionadas con el uso de Scratch, en general, los encuestados que no pueden hacer una tarea son los que no usan habitualmente el ordenador. Sin embargo, no es así en todas las preguntas, por ejemplo, hay una mayoría que no utilizan nunca el ordenador, que sin embargo hacen que el personaje se mueva a la primera, y sin ayuda.

Otra pregunta donde se aprecian diferencias es en la de “¿Has podido guardar tu juego?”. El 100% de los que disponen de ordenador, han podido guardar el juego, mientras que los que no han podido guardarlo coinciden con los que habitualmente no usan el ordenador.

Con respecto a las preguntas relacionadas con la satisfacción con el uso del programa, los pocos que responden que no han podido jugar al juego coinciden en su mayoría con participantes que no disponen de un ordenador en casa. Por otro lado, cuando se les pregunta si les ha resultado divertido, el 100% de los que no tienen ordenador contestan que sí. Parece ser que Scratch le resulta más motivante a los usuarios que menos utilizan el ordenador, tal vez por la novedad que para ellos supone.

También destacan las preguntas sobre las posibles combinaciones de bloques. La mayoría de los encuestados aciertan al identificar los bloques que se pueden combinar, pero hay varios casos (por ejemplo, por siempre – apuntar hacia), donde las personas que fallan, coinciden con las que no utilizan ningún ordenador.

Respecto a las preguntas relacionadas con el lenguaje que utiliza Scratch, la situación es similar. En la pregunta que cuestiona por la presencia de ciertas palabras en Scratch (variable, objeto, interfaz, etc.), aunque no en todos los casos, hay una cierta tendencia a que los encuestados que fallan, sean los mismos que los que no disponen de ordenador, o no lo utilizan.

En definitiva, hay indicios que apuntan a que los encuestados que no disponen de ordenador, tienen más dificultades para acercarse a Scratch. Respecto a los que sí disponen de él, no hay diferencias reseñables, salvo en las tareas que tienen que ver con el uso habitual del ordenador, donde no encuentran ningún problema.

—

PARTE FINAL

—

12. Conclusiones

Tomando como base el marco teórico, los resultados obtenidos en el estudio empírico, la discusión donde se analizan estos resultados, y la información adicional incluida en los anexos, se pueden sacar las siguientes conclusiones:

1. Scratch es una herramienta usable. En el análisis heurístico que se hizo de Scratch, planteado en el capítulo 10.2, se comprobó el cumplimiento de las pautas fundamentales incluidas en las normas de referencia. Estas pautas dieron lugar a las preguntas del formulario utilizado en el análisis de usuario, detallado en el capítulo 10.3. El cumplimiento de cada pauta se evaluó de nuevo a través los resultados obtenidos de la encuesta. Como se puede ver en el capítulo 10.3.6, el cumplimiento de cada pauta quedó confirmado en el análisis de usuario. Por tanto, se cumple el Objetivo Específico 1: evaluar la usabilidad de la herramienta de Scratch, y se concluye que es usable.

2. Aunque Scratch es una herramienta usable, presenta aspectos susceptibles de mejora en este ámbito. No es un objetivo de la investigación señalar estos aspectos, por eso no se va a realizar una descripción exhaustiva de los mismos, y solo se enumerarán algunos elementos reseñables, surgidos de la investigación, donde la usabilidad de Scratch podría mejorarse:

    a. Scratch está desarrollado con tecnología Adobe Flash [45], que no se puede ejecutar en *smartphones* ni *tablets.* Esto crea una barrera de aprendizaje para los estudiantes que no dispongan de ordenador.
    b. El *feedback* que proporciona Scratch es escaso. Cuando algo no funciona, la herramienta no explica por qué. No existen instrumentos de depuración del programa.
    c. El seguimiento de la ejecución del programa es complicado. Cuando se ejecuta un programa, se ilumina la pila de piezas completa que está en ejecución, en lugar de ir bloque por bloque. En este aspecto el usuario tiene una información parcial de lo que está sucediendo en cada momento.
    d. Scratch no incluye mecanismos sencillos para deshacer acciones. No existe un botón deshacer, ni un atajo de teclado al respecto. Si se quiere quitar una pieza de una pila de ellas, al arrastrarla con el ratón se quitarán también las piezas conectadas.
    e. La ayuda que incluye el programa es lineal, y carece de un buscador por temáticas, o por palabras clave. Se incluye un solo ejemplo de

[45] http://www.adobe.com/es/products/flashplayer.html

uso por cada pieza, y sería deseable tener más. No toda la ayuda está traducida del inglés al idioma del usuario.

f. Scratch podría incluir sugerencias y consejos integrados en el manejo de la herramienta, orientados a los usuarios noveles.
g. La interfaz presenta problemas para albergar programas de grandes dimensiones, que incluyan muchas piezas de código. No es posible tener una visión completa del desarrollo, y se debe hacer un uso constante de las barras de *scroll*.

3. Saber utilizar Scratch no es lo mismo que saber programar. Los participantes del estudio mostraron facilidad para todo lo que tenía que ver con el manejo de la herramienta, pero cuando se trataba de realizar tareas que incluían programación surgían las dificultades. Como se ha presentado en el capítulo 4, saber programar tiene dos componentes: saber resolver un problema en los términos que el ordenador comprende (pensamiento computacional) y saber expresar esa solución con la sintaxis adecuada. Con respecto a la sintaxis, en esta investigación se han encontrado indicadores que señalan que la lógica de bloques de Scratch se entiende. Pero que se sepa qué piezas se pueden combinar, no significa que se conozca cuáles se deben utilizar, y cómo se deben agrupar para elaborar un programa que dé respuesta a un problema. Es decir, con los datos de esta investigación no podemos afirmar ni negar que aprendiendo a manejar Scratch se fomente la adquisición del pensamiento computacional.

4. Scratch es una herramienta que no dificulta el aprendizaje de la programación. Por los datos obtenidos en la investigación no podemos afirmar que lo facilite, pero sí que no lo dificulta. Según las respuestas del cuestionario, el aprendizaje de los rudimentos básicos de la herramienta es sencillo, y no entorpece otros propósitos. Podemos afirmar por tanto que se cumple el Objetivo Específico 2**:** Evaluar si la usabilidad de Scratch facilita o dificulta el aprendizaje de la programación.

5. En el proceso de enseñanza-aprendizaje de la programación a través de Scratch, el papel del docente se revela como fundamental, a tenor de las respuestas recibidas. Según estas, cuando se trata de hacer tareas que involucren programación, la herramienta se antoja difícil de manejar sin la presencia del docente. Por tanto, una de las conclusiones que se pueden sacar de la presente investigación, es que se debe tener en cuenta la formación que debe recibir el docente, para que pueda enseñar a programar con Scratch.

6. Los dos puntos anteriores vienen a reflejar que Scratch facilita en cierto modo la comprensión de la sintaxis de la programación, pero no queda claro que facilite la semántica de las construcciones de los programas. De hecho, el que sea necesaria la supervisión del docente en el proceso de enseñanza-aprendizaje, indica lo contrario.

7. El formulario que surge del análisis de experto y heurístico, es un instrumento válido para evaluar la usabilidad de Scratch en el contexto de la Educación Primaria de la Comunidad de Madrid. Por tanto, podemos afirmar que se cumple el Objetivo Específico 3: Diseñar y validar un instrumento que permita evaluar la adecuación de una herramienta de aprendizaje de la programación, desde el punto de vista de su usabilidad.

8. En la investigación se ha detectado que puede haber indicios que señalen que los niños y niñas que no disponen de ordenador en casa pueden encontrar más dificultades a la hora de aprender a programar con Scratch. En cualquier caso, no se han encontrado evidencias estadísticas que sustenten esta afirmación. Serían necesarias otras investigaciones más específicas para confirmar esta hipótesis.

En definitiva, se ha confirmado que Scratch es una herramienta usable, y que no dificulta el aprendizaje de la programación. Para determinar si es la herramienta ideal en el contexto de la asignatura de “Tecnología y recursos digitales para la mejora del aprendizaje” sería necesario compararla con otras herramientas. Uno de los valores que aporta esta tesis, es que incluye en los anexos un pre-análisis de 50 posibles alternativas a Scratch, y propone un instrumento para su evaluación.

Con este último punto podemos afirmar que se ha cumplido el Objetivo General: Evaluar la herramienta Scratch desde el punto de vista de su usabilidad para determinar la adecuación de su uso para el aprendizaje de la programación.

13. Limitaciones

La propia complejidad del sistema educativo español supuso una de las primeras limitaciones de esta investigación En el momento en el que se comienza el estudio, la LOMCE estaba en periodo de implantación. La ley ya decía que se podía impartir la asignatura “Tecnología y recursos digitales para la mejora del aprendizaje”, con carácter optativo, pero no había demasiada información sobre cómo llevarla a cabo. Resultó difícil encontrar colegios que estuvieran interesados en participar en el estudio, y que además cumplieran con los requisitos que debía tener la muestra.

Los procesos administrativos tampoco fueron sencillos, los padres y/o madres de cada estudiante tuvieron que firmar un formulario de consentimiento para que estos pudieran participar en el estudio (ver anexos).

Por último, cabe recordar que este estudio no se realiza en el contexto de una tesis doctoral, y no de un proyecto financiado, por lo que los recursos con los que se contaba eran limitados.

14. Propuestas a futuro

A continuación se enuncian algunas propuestas de investigación que se podrían realizar en un futuro, a partir de la investigación presente:

- Completar el análisis de datos analizados, cruzándolos con la variable de género.
- Realizar el mismo estudio, esta vez con otras herramientas de las propuestas en el capítulo 4.5
- Realizar un estudio encaminado a proponer un nuevo diseño de Scratch, que subsane las deficiencias de usabilidad que tenga
- Replicar el estudio en colegios privados y concertados de la Comunidad de Madrid
- Realizar el estudio en otras comunidades autónomas, como Navarra, donde también se utiliza Scratch en las aulas, y comparar los datos con los obtenidos en Madrid
- Traducir el cuestionario al inglés, y realizar el estudio a nivel internacional
- Elaborar una herramienta online, orientada a docentes de Primaria, que permita rellenar el cuestionario, y ver los resultados en el momento, cumpliendo los obligados protocolos de protección de datos.

BIBLIOGRAFÍA

15. Referencias bibliográficas

Abadie, B. (2014). Hobby Consolas: Análisis de Project Spark XBox One. Obtenido en http://www.hobbyconsolas.com/reviews/analisis-project-spark-xbox-one-91988

ACM. (2017). The ACM CHI Conference on Human Factors in Computing Systems. Obtenido en https://chi2017.acm.org/

Adam, A. y Mowers, H. (2013). 7 Apps for Teaching Children Coding Skills. Obtenido en https://www.edutopia.org/blog/7-apps-teaching-children-coding-anna-adam

AGIMO. (2008). User Profiling and Testing Toolkit. Obtenido en https://www.finance.gov.au/archive/user-profiling-and-testing-toolkit/usability-checklist.html

Aguilar Gil, M. y Conde Melguizo, R. (2017). La Accesibilidad a la Administración Electrónica en España de las Personas con Discapacidad Motora. *Administração Pública e Gestão Social, 9*(1).

Aho, A. V. (2012). Computation and Computational Thinking. *The Computer Journal, 55*(7), 832-835. doi: 10.1093/comjnl/bxs074

Alarcón, J. M. (2010a). La mejor forma de aprender a programar. Obtenido en http://geeks.ms/jalarcon/2010/09/24/la-mejor-forma-de-aprender-a-programar/

Alarcón, J. M. (2010b). Secuencia de aprendizaje en materias TIC. Obtenido en http://www.jasoft.org/blog/content/binary/WindowsLiveWriter/Lamenorformadeaprenderaprogramar_12420/SecuenciaAprendizajeTIC_4.png

Alesandrini, K. y Larson, L. (2002). Teachers bridge to constructivism. *The Clearing House, 75*(3), 118-121.

Allen, B. A. y Butler, L. (1996). The effects of music and movement opportunity on the analogical reasoning performance of African American and White school children: A preliminary study. *Journal of Black Psychology, 22*(3), 316-328.

Almasi, G. S. y Gottlieb, A. (1988). Highly parallel computing.

Alonso de Castro, M. G. (2013). Educational projects based on mobile learning. *Teoría de la Educación; Educación y Cultura en la Sociedad de la Información, 15*(1), 10.

Alonso Urbano, D. y Conde Melguizo, R. (2015). Proyecto HIPO. Investigación y desarrollo de herramientas de interacción gestual para aprender a programar como contenido del currículo de primaria: análisis de Scratch *Videojuegos: desarrollo e industria creativa* (pp. 101-113). Madrid: Editorial ESNE.

Alonso Urbano, D. y Hueso Rivas, S. (2014). Base didáctica en la enseñanza de lenguajes de programación para niños: herramientas, interfaces y mecanismos visuales *Tecnología y narrativa audiovisual* (pp. 263-275). Madrid: Fragua.

Altadmri, A. y Brown, N. C. (2015). *37 million compilations: Investigating novice programming mistakes in large-scale student data.* Paper presentado en Proceedings of the 46th ACM Technical Symposium on Computer Science Education.

Amabile, T. M. y Pillemer, J. (2012). Perspectives on the social psychology of creativity. *The Journal of Creative Behavior, 46*(1), 3-15.

American Standards Association. (1963). American standard code for information interchange. *ASA X3, 4*.

Ananiadou, K. y Claro, M. (2009). 21st century skills and competences for new millennium learners in OECD countries.

Anderson, J. R. (1990). *The adaptive control of thought*. Hillsdale, NJ: Erlbaum.

Angulo Usategui, J. M., García Zubía, J. y Angulo Martínez, I. (2007). *Sistemas digitales y tecnología de computadores* (2ª ed.). Madrid: Editorial Paraninfo.

Aránega Pardo, C. (2009). Usabilidad y satisfacción de las personas que trabajan con Tecnologías de la Información y de la Comunicación. *No Solo Usabilidad*(8).

Ardito, C., De Marsico, M., Lanzilotti, R., Levialdi, S., Roselli, T., Rossano, V. y Tersigni, M. (2004). *Usability of e-learning tools.* Paper presentado en Proceedings of the working conference on Advanced visual interfaces.

Arroyo, L. (1991). *200 años de informática*. Madrid: Espasa Calpe.

Asensi Artiga, V. (1993). Evolución histórica de las Tecnologías de la Información y su aplicación en el proceso documental. *Revista general de información y documentación, 3*(2), 131.

Astrachan, O., Hambrusch, S., Peckham, J. y Settle, A. (2009). *The present and future of computational thinking.* Paper presentado en ACM SIGCSE Bulletin.

Ausubel, D. P., Novak, J. y Hanesian, H. (1976). Significado y aprendizaje significativo. *Psicología educativa: un punto de vista cognoscitivo. Mexico: Editorial Trillas*, 55-107.

Baddeley, A. (1992). Working Memory and Conscious Awareness. *Theories of memory*, 11-20.

Baeza Yates, R. y Rivera Loaiza, C. (2002). Ubicuidad y Usabilidad en la Web. *Centro de Investigación de la Web Departamento de Ciencias de la Computación, Universidad de Chile*.

Balanskat, A. y Engelhardt, K. (2015). *Computing our future: Computer programming and coding-Priorities, school curricula and initiatives across Europe*. Belgium: European Schoolnet.

Balardini, S. (2002). Jóvenes, tecnología, participación y consumo. *Proyecto Ju*.

Bara Soro, P. M. y Sánchez Manzano, E. (2001). *Estrategias metacognitivas y de aprendizaje: estudio empírico sobre el efecto de la aplicación de un programa*

metacognitivo, y el dominio de las estrategias de aprendizaje en estudiantes de ESO, BUP y Universidad. Universidad Complutense de Madrid.

Barefootcas.org. (2014). The Computational Thinker: Concepts & Approaches. Obtenido en http://barefootcas.org.uk/wp-content/uploads/2014/06/Barefoot-CT-Poster-for-website.jpg

Barney, B. (2015). *Introduction to Parallel Computing*. Livermore, CA: Lawrence Livermore National Laboratory.

Baro Cálciz, A. (2011). Metodologías activas y aprendizaje por descubrimiento. *Revista digital innovacion y experiencias educativas*.

Barranco, R. (2012). ¿Qué es Big Data? Obtenido en https://www.ibm.com/developerworks/ssa/local/im/que-es-big-data/

Barrón Ruiz, A. (1993). Aprendizaje por descubrimiento: principios y aplicaciones inadecuadas. *Enseñanza de las Ciencias, 11*(1), 003-011.

Battista, M. T. y Clements, D. H. (1988). A case for a Logo-based elementary school geometry curriculum. *Arithmetic Teacher, 36*(3), 11-17.

Bayona Aznar, B. (2013). *Los ejes de la LOMCE.* Paper presentado en Forum Aragón: revista digital de FEAE-Aragón sobre organización y gestión educativa.

Beaton, J., Jeong, S. Y., Xie, Y., Stylos, J. y Myers, B. A. (2008). *Usability challenges for enterprise service-oriented architecture APIs.* Paper presentado en Visual Languages and Human-Centric Computing, 2008. VL/HCC 2008. IEEE Symposium on.

Beltrán, M. (2000). Cinco vías de acceso a la realidad social. *M. García, J. Ibáñez y F. Alvira (Comp.), El análisis de la realidad social. Métodos y técnicas de investigación*, 189-222.

Ben-Ari, M. M. (2016). In defense of programming. *ACM Inroads, 7*(1), 44-46.

Benesse Corporation. (2017). Codeable Crafts. Obtenido en https://www.codeablecrafts.com/

Bergin, S. y Reilly, R. (2005a). *The influence of motivation and comfort-level on learning to program.* Paper presentado en Proceedings of the PPIG.

Bergin, S. y Reilly, R. (2005b). *Programming: factors that influence success.* Paper presentado en ACM SIGCSE Bulletin.

Bergin, T. J. (2007). A History of the History of Programming Languages. *COMMUNICATIONS OF THE ACM, 50*(5), 69.

Bergmann, J. y Sams, A. (2012). *Flip your classroom: Reach every student in every class every day*. Alexandria, VA International Society for Technology in Education.

Bers, M. U., Flannery, L., Kazakoff, E. R. y Sullivan, A. (2014). Computational thinking and tinkering: Exploration of an early childhood robotics curriculum. *Computers & Education, 72*, 145-157.

Bindé, J. (2005). Hacia las sociedades del conocimiento: informe mundial de la UNESCO. Francia: Ediciones Unesco.

Bloch, J. (2005). How to Write a Good API and Why it Matters. *Keynote Address for LCSD workshop at OOPSLA*.

Blum, G. D., Facundo Abal, J., Lozzia, G. S., Picón Janeiro, J. C. y Attorresi, H. F. (2011). Analogías de figuras: Teoría y construcción de ítemes. *Interdisciplinaria, 28*(1), 131-144.

Blumenfeld, P. C., Soloway, E., Marx, R. W., Krajcik, J. S., Guzdial, M. y Palincsar, A. (1991). Motivating project-based learning: Sustaining the doing, supporting the learning. *Educational psychologist, 26*(3-4), 369-398.

Bobrow, D., Feurzeig, W. y Papert, S. (2007). WinLogo. Obtenido en http://guindo.pntic.mec.es/crangil/winlogo.htm

Bogliolo, A. (2014a). Code Week Obtenido en http://codeweek.it/category/codyroby/page/2/

Bogliolo, A. (2014b). Codeweek: Cody&Roby. Obtenido en http://codeweek.it/cody-roby/kit-fai-da-te/

Boole, G. (1854). The Laws of Thought (1854).

Boole, G. y Corcoran, J. (2003). *An Investigation of the Laws of Thought*. Ireland: Dover.

Boscán Mieles, M. M. y Klever Montero, K. L. (2012). Metodología basada en el método heurístico de polya para el aprendizaje de la resolución de problemas matemáticos. *Escenarios, 10*(2), 7-19.

Boyle, R., Carter, J. y Clark, M. (2002). What makes them succeed? Entry, progression and graduation in Computer Science. *Journal of Further and Higher Education, 26*(1), 3-18.

Bransford, J. D., Brown, A. L. y Cocking, R. R. (2000). *How people learn*. Washington, DC: National Academy Press.

Bransford, J. D. y Stein, B. S. (1984). The ideal problem solver. A guide for improving thinking, learning, and creativity. *A Series of Books in Psychology, New York: Freeman, 1984, 1*.

Bransford, J. D. y Stein, B. S. (1986). *Solución ideal de problemas: guía para mejor pensar, aprender y crear.* New York, NY: Labor.

Brennan, K. y Resnick, M. (2012). *New frameworks for studying and assessing the development of computational thinking.* Paper presentado en Proceedings of the 2012 annual meeting of the American Educational Research Association, Vancouver, Canada.

Brown, N. C. C., Kölling, M., McCall, D. y Utting, I. (2014). *Blackbox: a large scale repository of novice programmers' activity.* Paper presentado en Proceedings of the 45th ACM technical symposium on Computer science education.

Bruner, J. (1991). *Actos de significado*. Madrid: Alianza.

Burke, Q. (2012). The markings of a new pencil: Introducing programming-as-writing in the middle school classroom. *Journal of Media Literacy Education, 4*(2), 121-135.

Byrne, P. y Lyons, G. (2001). *The effect of student attributes on success in programming.* Paper presentado en ACM SIGCSE Bulletin.

Cakir, M. (2008). Constructivist approaches to learning in science and their implications for science pedagogy: A literature review. *International journal of environmental & science education, 3*(4), 193-206.

Campbell, C. (2014). Polygon: As Project Spark beta begins players seek diverging creative paths. Obtenido en http://www.polygon.com/2014/3/4/5467530/as-project-spark-beta-begins-players-seek-diverging-creative-paths

Campos, Y. (2001). *Paradigmas psicoeducativos*. México, DF: DGENAMDF.

Carnegie Mellon Software Engineering Institute. (2008). *Software Technology Roadmap: Application Programming Interface*. Pittsburgh, PA: Carnegie Mellon Press.

Carneige Mellon University. (2017). Pbworks: Alice. Obtenido en http://alice3.pbworks.com/

Carreras Montoto, O. (2012). Web Usability Guidelines – Directrices de usabilidad web. Obtenido en https://olgacarreras.blogspot.com.es/2012/03/web-usability-guidelinesdirectrices-de.html - hhs

Caspersen, M. E. y Bennedsen, J. (2007). *Instructional design of a programming course: a learning theoretic approach.* Paper presentado en Proceedings of the third international workshop on Computing education research.

Castells, M. (2006). Informacionalismo, redes y sociedad red: una propuesta teórica. En M. Castells (Ed.), *La sociedad red: una visión global* (pp. 27-75). Madrid: Alianza Editorial.

Cavallo, E. E. (2011). Micromundos: La herencia de Logowriter. Obtenido en https://educavallologo.wordpress.com/tag/logo-writer/

Cejas, S. (2016). Vida Extra: Project Spark desaparece de XBox One. Obtenido en https://www.vidaextra.com/aventura-plataformas/project-spark-desaparece-de-xbox-one-y-windows-10-y-sus-servidores-cerraran-en-agosto

Cejka, E., Rogers, C. y Portsmore, M. (2006). Kindergarten robotics: Using robotics to motivate math, science, and engineering literacy in elementary school. *International Journal of Engineering Education, 22*(4), 711.

Chadwick, C. B. (1999). La psicología del aprendizaje desde el enfoque constructivista. *Rev. latinoam. psicol, 31*(3), 463-475.

Chapin, N. (1970). Flowcharting with the ANSI standard: A tutorial. *ACM Computing Surveys (CSUR), 2*(2), 119-146.

Chapin, N. (1979). Full report of the Flowchart Committee on ANS Standard X3. 5-1970. *ACM SIGPLAN Notices, 14*(3), 16-27.

Chaudhary, A. (2013). Theheories of Teaching. *International Journal for Research in Education (IJRE), 2*(3).

Cherubini, M., Venolia, G., DeLine, R. y Ko, A. J. (2007). *Let's go to the whiteboard: how and why software developers use drawings.* Paper presentado en Proceedings of the SIGCHI conference on Human factors in computing systems.

Chomsky, N. (2014). *Aspects of the Theory of Syntax* (Vol. 11). Cambridge, MA MIT press.

Church, A. (1932). A set of postulates for the foundation of logic. *Annals of mathematics*, 346-366.

Church, A. (1936a). A note on the Entscheidungsproblem. *The journal of symbolic logic, 1*(01), 40-41.

Church, A. (1936b). An unsolvable problem of elementary number theory. *American journal of mathematics, 58*(2), 345-363.

Clarke, S. (2004). Measuring API usability. *Doctor Dobbs Journal, 29*(5), S1-S5.

Clarke, S. y Becker, C. (2003). *Using the cognitive dimensions framework to evaluate the usability of a class library.* Paper presentado en Proceedings of the First Joint Conference of EASE PPIG (PPIG 15).

Clements, D. H. (1986). Effects of Logo and CAI environments on cognition and creativity. *Journal of Educational Psychology, 78*(4), 309.

Clements, D. H. y Meredith, J. S. (1993). Research on Logo: Effects and efficacy. *Journal of Computing in Childhood Education, 4*(4), 263-290.

CNIIE. (1991). El sistema educativo español.

Cobo Romaní, J. C. (2011). El concepto de tecnologías de la información. Benchmarking sobre las definiciones de las TIC en la sociedad del conocimiento. *Zer-Revista de Estudios de Comunicación, 14*(27), 295-318.

Code Academy. (2017). Code Academy. Obtenido en https://www.codecademy.com/

Code Studio. (2015). Star Wars: Building a Galaxy with Code. Obtenido en https://code.org/starwars

Code.org. (2015). Hour of Code. Obtenido en https://hourofcode.com/es

Collazos, C. A., et al. (2007). Evaluating collaborative learning processes using system-based measurement. *Educational Technology & Society, 10*(3), 257-274.

Computer Technology Institue. (2000). E-slate. Obtenido en http://e-slate.cti.gr/

Conde Melguizo, R. (2013a). *La autonomía de la persona con discapacidad como ciudadano: estudio de caso de análisis de la accesibilidad de la administración electrónica para las personas con discapacidad motora*. Paper presentado en XI Congreso AECPA (Asociación Española de Ciencias Políticas y de la Administración), Sevilla.

Conde Melguizo, R. (2013b). *La autonomía de la persona con discapacidad como ciudadano: estudio de caso de análisis de la accesibilidad de la administración electrónica para las personas con discapacidad motora*. Paper presentado en XI Congreso FES (Federación Española de Sociología), Madrid.

Conde Melguizo, R., Muñoz Muñoz , A. y González González, M. L. (2008). *Gobiernat-e: propuesta de participación ciudadana a través de las nuevas tecnologías. Sociología e ingeniería trabajando juntas.* Paper presentado en Sociedad, consumo y sostenibilidad: actas y textos elaborados a partir del" XIII Congreso de Sociología en Castilla-La Mancha".

Conde Melguizo, R. y Rozalén, R. (2012). *Manifiesto 4.0: el necesario papel de la sociología en el equilibrio de la sociedad digital.* Paper presentado en Nuevos tiempos, nuevos retos, nuevas sociologías.

Corrado, B. y Jacopini, G. (1966). Flow diagrams, turing machines and languages with only two formation rules. *Communications of the ACM, 9*(5), 366-371.

Costabile, M. F. (2001). Usability in the software life cycle. *Handbook of software engineering and knowledge engineering, 1*, 179-192.

Council, N. R. (1999). *Being fluent with information technology*. Washington, DC: National Academies Press.

Cowan, N. (1988). Evolving conceptions of memory storage, selective attention, and their mutual constraints within the human information-processing system. *Psychological bulletin, 104*(2), 163.

Cowan, N. (2008). What are the differences between long-term, short-term, and working memory? *Progress in brain research, 169*, 323-338.

Craik, F. I. y Lockhart, R. S. (1972). Levels of processing: A framework for memory research. *Journal of verbal learning and verbal behavior, 11*(6), 671-684.

Crews, T. y Ziegler, U. (1998). *The flowchart interpreter for introductory programming courses.* Paper presentado en Frontiers in Education Conference, 1998. FIE'98. 28th Annual.

Cronin, A. (2014). Teach Coding in the Classroom: Resources from ISTE '14. Obtenido en https://www.edutopia.org/blog/teach-coding-classroom-resources-iste-14

Cubillo, J. C. y González Labra, M. J. (1998). El razonamiento analógico como solución de problemas. *Introducción a la psicología del pensamiento. Madrid: Trotta.*, 409-451.

Cwalina, K. y Abrams, B. (2008). *Framework design guidelines: conventions, idioms, and patterns for reusable. net libraries*. Upper Saddle River, NJ: Pearson Education.

Cypher, A. (2011). Stagecast Creator. Obtenido en http://acypher.com/creator/

Dale, E. y Nyland, B. (1960). Cone of learning. *Educational Media*.

Daughtry, J. M., Farooq, U., Stylos, J. y Myers, B. A. (2009). *API usability: CHI'2009 special interest group meeting.* Paper presentado en CHI'09 Extended Abstracts on Human Factors in Computing Systems.

Davies, S. P. (1993). Expertise and display-based strategies in computer programming. *PEOPLE AND COMPUTERS*, 411-411.

De Bono, E. (1986). *El pensamiento lateral: Manual de creatividad*. Barcelona

Mexico: Editorial Paidós.

de Souza, C. R., Redmiles, D., Cheng, L.-T., Millen, D. y Patterson, J. (2004). *Sometimes you need to see through walls: a field study of application programming interfaces.* Paper presentado en Proceedings of the 2004 ACM conference on Computer supported cooperative work.

Decreto 89/2014, de 24 de julio, del Consejo de Gobierno, por el que se establece para la Comunidad de Madrid el Currículo de la Educación Primaria. (2015).

Delors, J. y Century, I. C. o. E. f. t. T.-f. (1996). *La Educación encierra un Tesoro: Informe a la UNESCO de la Comisión Internacional sobre la Educación para el Siglo XXI. Compendio*. Madrid: Santillana.

Delval, J. (2002). *El desarrollo humano*. Madrid: Siglo XXI.

Descubre Arduino. (2016). Obtenido en http://descubrearduino.com/caleiduino-un-caleidoscopio-digital-sonoro-e-interactivo/

Díaz, A. (2015a). Game Industria. Obtenido en http://gamedustria.com/entrevistamos-los-creadores-de-wimi5-plataforma-online-de-videojuegos-en-html-5/

Díaz, A. (2015b). Gameindustria: Wimi5. Obtenido en http://gamedustria.com/entrevistamos-los-creadores-de-wimi5-plataforma-online-de-videojuegos-en-html-5/

Díaz Barriga Arceo, F., Hernández Rojas, G. y García González, E. L. (2002). *Estrategias docentes para un aprendizaje significativo: una interpretación constructivista*. Madrid: McGraw Hill.

Díaz, M. (2014). Hipertextual: Made with code. Obtenido en https://hipertextual.com/archivo/2014/06/made-with-code/

Digipen Institue of Technology. (2016). Zero Digipen. Obtenido en http://zero.digipen.edu/

Dijkstra, E. W. (1968). Letters to the editor: go to statement considered harmful. *Communications of the ACM, 11*(3), 147-148.

Dillenbourg, P. (1999). What do you mean by collaborative learning. *Collaborative-learning: Cognitive and computational approaches, 1*, 1-15.

DiSessa, A. A. (2001). *Changing minds: Computers, learning, and literacy*. Cambridge, MA: Mit Press.

Dorling, M. y Walker, M. (2015). Computing at School: Computing Progression Pathways Obtenido en http://community.computingatschool.org.uk/resources/1946

Downes, S. (2005). An introduction to connective knowledge. En T. Hug (Ed.), *Media, Knowledge & Education* (pp. 77-103). Austria: Innsbruck University Press.

Eckerdal, A. y Berglund, A. (2005). What Does It Take to Learn'Programming Thinking'?

Elkner, J. (2004). Guido van Robot. Obtenido en http://gvr.sourceforge.net/screen_shots/

Ellis, B., Stylos, J. y Myers, B. (2007). *The factory pattern in API design: A usability evaluation.* Paper presentado en Proceedings of the 29th international conference on Software Engineering.

Elmroth, E. y Gustavson, F. G. (2000). Applying recursion to serial and parallel QR factorization leads to better performance. *IBM Journal of Research and Development, 44*(4), 605-624.

Epic Games. (2017a). Unreal Engine. Obtenido en https://www.unrealengine.com/what-is-unreal-engine-4

Epic Games. (2017b). Unreal Engine 4. Obtenido en https://www.unrealengine.com/what-is-unreal-engine-4

Epic Games, I. (2017). Unreal Engine. Obtenido en https://www.unrealengine.com/

Ericsson, K. A. y Kintsch, W. (1995). Long-term working memory. *Psychological review, 102*(2), 211.

Ertmer, P. A. y Newby, T. J. (1993). Behaviorism, cognitivism, constructivism: Comparing critical features from an instructional design perspective. *Performance improvement quarterly, 6*(4), 50-72.

Escandell Bermúdez, M. O., Rodríguez Martín, A. y Cardona Hernández, G. (2005). Convergencia Europea y profesorado. Hacia un nuevo perfil para el aprendizaje flexible. *Revista electrónica interuniversitaria de formación del profesorado, 20*(8-5), 17-21.

Escudero, J. (1991). Proyecto Atenea. *Informe de evaluación. Madrid: Ministerio de Educación*.

Espeso, P. (2015). Xataka: Cómo iniciar a un niño en la programación desde cero con Scratch. Obtenido en https://www.xataka.com/aplicaciones/como-iniciar-a-un-nino-en-la-programacion-desde-cero-con-scratch

Esteban Guilar, M. (2009). Las ideas de Bruner:" De la revolución cognitiva" a la" revolución cultural. *Educere, 13*(044).

Etzion, O. y Niblett, P. (2010). *Event processing in action*. Stamford, CT: Manning Publications Co.

Europeo, P. y Europea, C. d. l. U. (2006). Recomendación 2006/962/CE del Parlamento Europeo y del Consejo, de 18 de diciembre de 2006, sobre las competencias clave para el aprendizaje permanente [Diario Oficial L 394 de 30.12. 2006].

EURYDICE. (2002). *Key competencies: A developing concept in general compulsory education* (Vol. 5). Brussels: European Unit.

F# Software Foundation. (2012). F#. Obtenido en http://fsharp.org/

Fablab. (2016). Obtenido en http://fablab.ua.es/noticias/caleiduino/

FECYT, Google y Everis. (2016). Educación en Ciencias de la Computación en España 2015. Obtenido en https://www.fecyt.es/es/system/files/publications/attachments/2016/05/everis_informegoogle_210x297_rgb_8apres.pdf

Felleisen, M., Findler, R. B., Flatt, M. y Krishnamurthi, S. (2001). *How to design programs: An introduction to computing and programming*. Cambridge, MA: MIT press.

Fernández Enguita, M. (2008). Escuela pública y privada en España: la segregación rampante. *RASE: Revista de la Asociación de Sociología de la Educación, 1*(2), 42-69.

Fessakis, G., Gouli, E. y Mavroudi, E. (2013). Problem solving by 5–6 years old kindergarten children in a computer programming environment: A case study. *Computers & Education, 63*, 87-97.

Fincher, S., et al. (2006). *Predictors of success in a first programming course.* Paper presentado en Proceedings of the 8th Australasian Conference on Computing Education-Volume 52.

Fisher-Price. (2016). Fisher-Price: Code-a-pillar. Obtenido en http://www.fisher-price.com/en_US/brands/think-and-learn/index.html

FOLDOC. (2017). The Free Online Dictionary of Computing. Obtenido en http://foldoc.org

Forouzan, B. A. (2003). *Introducción a la ciencia de la computación: de la manipulación de datos a la teoría de la computación*. Madrid: Cengage Learning Editores.

Fox, J.-A. (2015). App Education: Collaborative art with Sphero and Tickle app. Obtenido en http://www.appeducation.com/2015/09/06/collaborative-art-with-sphero-and-tickle-app/

Furber, S. (2012). Shut down or restart? The way forward for computing in UK schools. *The Royal Society, London*.

Galvis Panqueva, Á. H. (1992). *Ingeniería de software educativo*. Barcelona: Ediciones Unidas.

Gamificación S.L. (2013). Qué es la Gamificación. Obtenido en http://www.gamificacion.com/que-es-la-gamificacion

García-Peñalvo, F. J. (2016). What Computational Thinking Is. *Journal of Information Technology Research, 9*(3).

García-Peñalvo, F. J., Rees, A., Jormanainen, I., Tuul, M. y Reimann, D. (2016). *An overview of the most relevant literature on coding and computational thinking with emphasis on the relevant issues for teachers*. Belgium: TACCLE3 Consortium.

Garris, R., Ahlers, R. y Driskell, J. E. (2002). Games, motivation, and learning: A research and practice model. *Simulation & gaming, 33*(4), 441-467.

Gil Pérez, D. (1983). Tres paradigmas básicos en la enseñanza de las ciencias. *Enseñanza de las Ciencias, 1*(1), 026-033.

Gillete, J. (2010). Hackety Hack [Aplicación Informática]. Obtenido de https://hackety-hack.com

Glaser, B. G. y Strauss, A. L. (1967). *The discovery of grounded theory: strategies for qualitative theory*. Chicago, IL: Aldine Publishing Co.

Gödel, K. (2016). *Consistency of the Continuum Hypothesis.(AM-3)* (Vol. 3). Princeton, NJ: Princeton University Press.

Godfrey, J. F. (1991). Lenguaje ensamblador para microcomputadoras IBM. *Editorial Prentice-Hall*.

Goldstine, H. H. y Von Neumann, J. (1948). *Planning and coding of problems for an electronic computing instrument* (Vol. 2). Princeton, NJ: Institute for Advanced Study Princeton, N. J.

Gomes, A., Carmo, L., Bigotte, E. y Mendes, A. (2006). *Mathematics and programming problem solving.* Paper presentado en 3rd E-Learning Conference–Computer Science Education.

Gomes, A. y Mendes, A. J. (2007a). *An environment to improve programming education.* Paper presentado en Proceedings of the 2007 international conference on Computer systems and technologies.

Gomes, A. y Mendes, A. J. (2007b). *Learning to program-difficulties and solutions.* Paper presentado en International Conference on Engineering Education–ICEE.

Gomes, A., Santos, A. y Mendes, A. J. (2012). *A study on students' behaviours and attitudes towards learning to program.* Paper presentado en Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education.

González, J. M. (2016). Caleiduino. Obtenido en http://www.caleiduino.com/

Good, T. L. y Brophy, J. E. (1990). *Educational psychology: A realistic approach*. Massachusetts, MA: Longman/Addison Wesley Longman.

Goodwin, B. y Miller, K. (2013). Evidence on flipped classrooms is still coming in. *Educational Leadership, 70*(6), 78-80.

Google Company. (2016). Made with Code. Obtenido en https://www.madewithcode.com/projects/

Google Company. (2017a). Blockly Games. Obtenido en https://blockly-games.appspot.com/

Google Company. (2017b). Blockly Web (Version Web) [Aplicación Informática]. Obtenido de https://developers.google.com/blockly/

Google Company. (2017c). Google Developers: Blockly. Obtenido en https://developers.google.com/blockly/

Google For Education. (2016). Exploring Computational Thinking. Obtenido en https://edu.google.com/resources/programs/exploring-computational-thinking/ - !ct-overview

Gorman, H. y Bourne, L. E. (1983). Learning to think by learning LOGO: Rule learning in third-grade computer programmers. *Bulletin of the Psychonomic Society, 21*(3), 165-167.

Gorn, S., Bemer, R. W. y Green, J. (1963). American standard code for information interchange. *Communications of the ACM, 6*(8), 422-426.

Green, T. R. (1989). Cognitive dimensions of notations. *People and computers V*, 443-460.

Group, L. K. (2013). Scratch MIT Edu. Obtenido en https://scratch.mit.edu/scratch2download/

Grover, S. y Pea, R. (2013). Computational Thinking in K–12. *Educational Researcher, 42*(1), 38-43. doi: 10.3102/0013189X12463051

Gustavson, F. G. (1997). Recursion leads to automatic variable blocking for dense linear-algebra algorithms. *IBM Journal of Research and Development, 41*(6), 737-755.

Guzdial, M. (2000). Soporte tecnológico para el aprendizaje basado en proyectos. En C. Dede (Ed.), *Aprendiendo con tecnología*. Argentina Paidos.

Guzdial, M. (2004). Programming environments for novices. *Computer science education research, 2004*, 127-154.

Hamada, R. M. (1987). The relationship between learning Logo and proficiency in mathematics. *Dissertation Abstracts International, 47*, 2510-A.

Hamdan, N., McKnight, P., McKnight, K. y Arfstrom, K. M. (2013). The flipped learning model: A white paper based on the literature review titled a review of flipped learning: Flipped Learning Network.

Harel, I. y Papert, S. (1991). *Constructionism*. New York, NY: Ablex Publishing.

Harrison, P. G. y Khoshnevisan, H. (1992). A new approach to recursion removal. *Theoretical Computer Science, 93*(1), 91-113.

Harvey, B. (1997). *Computer Science Logo Style: Symbolic Computing* (Vol. 1). Cambridge, MA MIT press.

Harvey, B. (2002). Welcome to MSWLogo. Obtenido en http://www.softronix.com/logo.html

Hassan Montero, Y. y Martín Fernández, F. J. (2004). Propuesta de adaptación de la metología de diseño centrado en el usuario para el desarrollo de sitios web accesibles. *Revista española de documentación científica, 27*(3), 330-344.

Hassan Montero, Y. y Ortega Santamaría, S. (2009). *Informe APEI sobre usabilidad* (Vol. 3). Gijón: APEI, Asociación Profesional de Especialistas en Información.

Hassan-Montero, Y. y Ortega-Santamaría, S. (2009). *Informe APEI sobre usabilidad* (Vol. 3). Gijón: Asociación Profesional de Especialistas en Información.

Hatwell, Y. (1985). *Piagetian Reasoning and the Blind*. New York, NY.: ERIC.

Henry, S. L., Law, C. y Barnicle, K. (2001). *Adapting the design process to address more customers in more situations.* Paper presentado en UPA (Usability Professionals' Association) 2001 Conference.

Hernández Sampieri, R., Fernández Collado, C. y Baptista Lucio, P. (2004). *Metodología de la investigación*. Mexico, DF: McGraw Hill.

Hilbert, D. y Ackermann, W. (1975). *Elementos de Lógica Teórica* (2ª ed.). Madrid: Ed. Tecnos

Hines, S. N. (1983). Preschoolers+ Computers= ABC: Computer Programming Abilities of Five-Year-Old Children. *Educational computer, 3*(4), 10-12.

Hirsch Jr, E. (2000). You can always look it up—or can you. *American Educator, 24*(1), 4-9.

Hoc, J.-M. y Nguyen-Xuan, A. (1990). Language semantics, mental models and analogy. *Psychology of programming, 10*, 139-156.

Holzinger, A., Searle, G. y Wernbacher, M. (2011). The effect of previous exposure to technology on acceptance and its importance in usability and accessibility engineering. *Universal Access in the Information Society, 10*(3), 245-260.

Hopscotch Technologies. (2016a). Daisy the Dinosaur [Aplicación Informática]. Obtenido de https://itunes.apple.com/us/app/daisy-the-dinosaur/id490514278

Hopscotch Technologies (2016b). Hopscotch: Coding for Kids. Obtenido en https://itunes.apple.com/us/app/hopscotch-coding-for-kids/id617098629?mt=8&ign-mpt=uo%3D4

Howard, T. J., Dekoninck, E. A. y Culley, S. J. (2010). The use of creative stimuli at early stages of industrial product innovation. *Research in Engineering design, 21*(4), 263-274.

Ibáñez, J. (2003). Perspectivas de la investigación social: el diseño en las tres perspectivas y como se realiza mediante grupos de discusión. *El análisis de la realidad social*, 57-98.

IBM. (2017). IBM SPSS. Obtenido en https://www.ibm.com/analytics/es/es/technology/spss/

Ictinpractice. (2015). Ictinpractice.

Insa, D. y Silva, J. (2015). Automatic transformation of iterative loops into recursive methods. *Information and Software Technology, 58*, 95-109.

Is Game Time. (2012). Obtenido en https://isgametime.wordpress.com/2012/03/31/crear-mods-para-minecraft/

ISTE y CSTA. (2011). Operational Definition of Computational Thinking for K-12 Education.

James, J. (2009). Strategies for learning programming languages. Obtenido en http://www.techrepublic.com/blog/programming-and-development/strategies-for-learning-programming-languages-what-works-and-what-doesnt/1159

Jenkins, T. (2002). *On the difficulty of learning to program.* Paper presentado en Proceedings of the 3rd Annual Conference of the LTSN Centre for Information and Computer Sciences.

Jones, J. C. (1980). *Design Methods: Seeds of Human Futures* (J. W. Sons Ed.). Great Britain: Chichester.

Joyanes Aguilar, L. (2008). *Fundamentos de la Programación: Algoritmos, estructuras de datos y objetos* (4ª ed.). Madrid: Mc-Graw Hill.

Kahn, K., Sendova, E., Sacristán, A. I. y Noss, R. (2011). Young students exploring cardinality by constructing infinite processes. *Technology, Knowledge and Learning, 16*(1), 3-34.

Kaijanaho, A.-J. (2015). Evidence-based programming language design: a philosophical and methodological exploration. *Jyväskylä studies in computing 222.*

Kelleher, C. y Pausch, R. (2005). Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Computing Surveys (CSUR), 37*(2), 83-137.

Kerr, B. (2007). Which radical discontinuity. Obtenido en http://billkerr2.blogspot.com.es/2007/02/which-radical-discontinuity.html

Kharbach, M. (2014). Educators Technology: Teaching coding in class - 17 apps to try. Obtenido en http://www.educatorstechnology.com/2014/11/teaching-coding-in-class-17-apps-to-try.html

Kitchen, R. (2015). RoboMind. Obtenido en http://www.robomind.net/es/

Knuth, D. (1997). *The Art of Computer Programming: Fundamental Algorithms* (3ª ed. Vol. 1). Redwood City, CA: Addison-Wesley.

Knuth, D. E. (1964). Backus normal form vs. backus naur form. *Communications of the ACM, 7*(12), 735-736.

Koster, R. (2013). *Theory of fun for game design*: O'Reilly Media, Inc.

Kruse, K. (2003). Effective user interface design. *Learning Circuits*.

Larsen, I. y Reneau, J. M. (2010). Basic256. Obtenido en http://www.basic256.org/basiclinks

Lee, I. (2016). Reclaiming the roots of CT. *CSTA Voice: The Voice of K–12 Computer Science Education and Its Educators, 12*(1), 3-5.

LEGO Group. (2017). LEGO Mindstorms. Obtenido en https://www.lego.com/en-us/mindstorms/downloads/download-software

Leibniz, G. W. (1703). Page of the Article "Explication de l'Arithmétique Binaire". Obtenido en https://upload.wikimedia.org/wikipedia/commons/a/ac/Leibniz_binary_system_1703.png

Lemos, V. (2006). La deseabilidad social en la evaluación de la personalidad infantil. *Suma Psicológica, 13*(1), 7-14.

Ley Orgánica 2/2006, de 3 de mayo, de Educación. Boletín Oficial del Estado. núm. 106, pp. 17158 a 17207 (2006).

Ley Orgánica 8/2013, de 9 de diciembre, para la Mejora de la Calidad Educativa. Boletín Oficial del Estado. núm. 295, pp. 97858 a 97921 (2013).

Ley Orgánica 14/1970, del 6 de agosto, General de Educación y Financiamiento de la Reforma Educativa. Boletín Oficial del Estado. núm. 187, pp. 12525 a 12546. (1970).

Lightbot Inc. (2015). Lightbot : Programming Puzzles [Aplicación Informática]. Obtenido de https://play.google.com/store/apps/details?id=com.lightbot.lightbot

Lightbot Inc. (2016). Lightbot [Aplicación Informática]. Obtenido de https://itunes.apple.com/us/app/lightbot-programming-puzzles/id924624572

Listforge. (2017). Minecraft Schematics. Obtenido en http://www.minecraft-schematics.com/category/redstone/

Liu, Y. A. y Stoller, S. D. (1999). From recursion to iteration: what are the optimizations? *ACM Sigplan Notices, 34*(11), 73-82.

López Curiel, R. (2014). *Las TIC en el aula de Tecnología. Guía para su aplicación a la metodología de proyectos*. Raleigh, NC: Lulu.com.

López García, J. C. (2009). *Algoritmos y Programación: Guía para docentes*. Cali, Colombia: Fundación Gabriel Piedrahita Uribe.

López, J. (2006). *Las competencias básicas del currículo en la LOE.* Paper presentado en V Congreso Internacional “Educación y Sociedad”. Granada, España. .

López, W. (1989). HISTORIA DE LA COMPUTACIÓN.

Lye, S. Y. y Koh, J. H. L. (2014). Review on teaching and learning of computational thinking through programming: What is next for K-12? *Computers in Human Behavior, 41*, 51-61.

Lynn, R. (2010). In Italy, north–south differences in IQ predict differences in income, education, infant mortality, stature, and literacy. *Intelligence, 38*(1), 93-100.

Maddux, C. D. y Johnson, D. L. (1997). Logo: A retrospective. *Computers in the Schools, 14*(1-2), 1-8.

Maloney, J. H., Peppler, K., Kafai, Y., Resnick, M. y Rusk, N. (2008). *Programming by choice: urban youth learning programming with scratch* (Vol. 40). New York, NY: ACM.

Manovich, L. (2013). *Software takes command* (Vol. 5). Great Britain: A&C Black.

Maocho, F. (2016). Felix Maocho Wordpress: El caleidoscopio de Arduino. Obtenido en https://felixmaocho.wordpress.com/2016/06/25/caleiduino-el-caleidoscopio-de-arduino/

Marquès Graells, P. (2000). Las TIC y sus aportaciones a la sociedad. *Departamento de pedagogía aplicada, facultad*.

Martin, J. (1982). *Application development without programmers*. Upper Saddle River, NJ: Prentice Hall PTR.

Martínez Morales, A. A. y Rosquete De Mora, D. H. (2009). NASPI: Una notación algorítmica estándar para programación. *Télématique: Revista Electrónica de Estudios Telemáticos, 8*(3), 55-74.

Mayer, R. E. (1992). *Thinking, problem solving, cognition*. New York, NY: WH Freeman/Times Books/Henry Holt & Co.

McCarthy, J. (1962). *Towards a Mathematical Science of Computation.* Paper presentado en IFIP Congress.

McConnell, S. (2004). *Code complete* (2º ed.). Redmond, WA: Pearson Education.

McDowell, C., Werner, L., Bullock, H. y Fernald, J. (2002). The effects of pair-programming on performance in an introductory programming course. *ACM SIGCSE Bulletin, 34*(1), 38-42.

McDowell, C., Werner, L., Bullock, H. E. y Fernald, J. (2006). Pair programming improves student retention, confidence, and program quality. *Communications of the ACM, 49*(8), 90-95.

McLellan, S. G., Roesler, A. W., Tempest, J. T. y Spinuzzi, C. I. (1998). Building more usable APIs. *IEEE software, 15*(3), 78-86.

Mead, J., Gray, S., Hamer, J., James, R., Sorva, J., Clair, C. S. y Thomas, L. (2006). A cognitive approach to identifying measurable milestones for programming skill acquisition. *ACM SIGCSE Bulletin, 38*(4), 182-194.

Meerbaum-Salant, O., Armoni, M. y Ben-Ari, M. (2013). Learning computer science concepts with scratch. *Computer Science Education, 23*(3), 239-264.

Mergel, B. (1998). Diseño instruccional y teoría del aprendizaje. *Universidad de Saskatchewan, Canadá, 16*.

Microsoft. (2017a). MSDN: Learn to Develop with Microsoft Developer Network. Obtenido en https://msdn.microsoft.com/

Microsoft. (2017b). .NET. Obtenido en https://www.microsoft.com/net

Microsoft Research. (2016). Kodu Game Lab. Obtenido en https://www.kodugamelab.com/about/

Miller, G. A., Galanter, E. y Pribram, K. H. (1986). *Plans and the structure of behavior*. Brewster, NY: Adams Bannister Cox.

Minecrafteo Corp. (2017). Minecrafteo. Obtenido en http://www.minecrafteo.com/

MIT. (2013). Scratch (Version Web) [Aplicación Informática]. Obtenido de https://scratch.mit.edu/

MIT. (2016a). La Hora del Código: Hazlo Volar. Obtenido en https://scratch.mit.edu/go

MIT. (2016b). MIT App Inventor (Version Web) [Aplicación Informática]. Obtenido de https://appinventor.mit.edu

MIT. (2016c). Scratch Wiki. Obtenido en https://wiki.scratch.mit.edu/wiki/Programming

MIT y IST. (2011). Usability Guidelines. Obtenido en http://web.archive.org/web/20110511060055/http://ist.mit.edu/services/consulting/usability/guidelines

Mojang. (2017). Minecraft. Obtenido en https://minecraft.net/es-es/?ref=m

Möneg, J. (2017). Snap. Obtenido en http://snap.berkeley.edu/

Monereo, C., Castelló, M., Clariana, M., Palma, M. y Pérez, M. (2000). Análisis de los factores que intervienen en la enseñanza-aprendizaje de estrategias en el aula. *Estrategias de enseñanza y aprendizaje.(75-97). Barcelona, España: Grao*.

Moreno Herrero, I. (2005). Medios y recursos didácticos en el aula. En P. Sánchez Delgado (Ed.), *Enseñar y aprender* (pp. 169-184). Salamanca: Ediciones Témpora.

Moreno León, J. (2013). ¿Qué es el pensamiento computacional? Obtenido en http://programamos.es/que-es-el-pensamiento-computacional/

Moursund, D. G. (2003). Project-based learning using information technology.

Myers, B., Stefik, A., Hanenberg, S., Kaijanaho, A.-J., Burnett, M., Turbak, F. y Wadler, P. (2016). *Usability of Programming Languages: Special Interest Group (SIG) Meeting at CHI 2016.* Paper presentado en Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems.

Myers, B. A., Burnett, M. M., Wiedenbeck, S. y Ko, A. J. (2007). *End user software engineering: CHI 2007 special interest group meeting.* Paper presentado en CHI'07 extended abstracts on human factors in computing systems.

Myers, B. A., Pane, J. F. y Ko, A. (2004). Natural programming languages and environments. *Communications of the ACM, 47*(9), 47-52.

Myreen, M. O. y Gordon, M. J. C. (2009). Transforming programs into recursive functions. *Electronic Notes in Theoretical Computer Science, 240*, 185-200.

Neutron Fuel. (2017). Tynker. Obtenido en https://www.tynker.com/kids-experience/

Neutron Fuel, I. (2017a). iTunes Apple. Obtenido en https://itunes.apple.com/us/app/tynker-learn-programming-visual/id805869467?ls=1&mt=8

Neutron Fuel, I. (2017b). Tynker. Obtenido en https://www.tynker.com/

Newell, A. y Simon, H. A. (1972). *Human problem solving* (Vol. 104): Prentice-Hall Englewood Cliffs, NJ.

Ng, E. y Bereiter, C. (1991). Three levels of goal orientation in learning. *Journal of the Learning Sciences, 1*(3-4), 243-271.

Nielsen, J. (1994). *Usability inspection methods.* Paper presentado en Conference companion on Human factors in computing systems.

Nielsen, J. (2000). *Usabilidad: Diseño de sitios web*. Madrid: Prentice Hall.

Nielsen, J. (2003). Usability 101: Introduction to usability.

Nielsen Norman Group. (2017). Evidence-Based User Experience Research, Training, and Consulting

. Obtenido en https://www.nngroup.com/reports/

Noer, M. (2012). One man, one computer, 10 million students: How Khan Academy is reinventing education. *Forbes*.

Nokia. (2003). Forum Nokia Usability Guidelines for Games. Obtenido en http://www.forum.nokia.com/main/technical_services/usability/games_usability_guidelines.html

Norcio, A. (1982). *Indentation, documentation and programmer comprehension.* Paper presentado en Proceedings of the 1982 conference on Human factors in computing systems.

Nunes, D. J. (2011). Ciência da computaçao na educaçao básica. *Jornal da Ciência, 9*(09).

OECD. (2015). Students, Computers and Learning: Making the Connection, PISA, OECD Publishing. Obtenido en http://dx.doi.org/10.1787/9789264239555-en

Olive, J. (1991). Logo programming and geometric understanding: An in-depth study. *Journal for Research in Mathematics Education*, 90-111.

Onrubia, J. (2005). Aprender y enseñar en entornos virtuales: actividad conjunta, ayuda pedagógica y construcción del conocimiento. *Revista de educación a distancia*.

Oracle. (2016). ¿Qué es la tecnología Java y para qué la necesito? . Obtenido en https://www.java.com/es/download/faq/whatis_java.xml

Ortega de la Puente, A., Alfonseca Moreno, M., de la Cruz Echeandía, M. y Pulido, E. (2006). *Compiladores e intérpretes: teoría y práctica*. Madrid: Pearson Educación.

Osborn, A. F. (1953). *Applied Imagination: Principles and Procedures of Creative Thinking*. New York, NY: Scribners.

Palumbo, D. B. (1990). Programming language/problem-solving research: A review of relevant issues. *Review of educational research, 60*(1), 65-89.

Pane, J. F. y Myers, B. A. (2000). *Tabular and textual methods for selecting objects from a group.* Paper presentado en Visual Languages, 2000. Proceedings. 2000 IEEE International Symposium on.

Paniagua Arís, E. (2001). La creatividad y las tecnologías de la información y las comunicaciones. *Anales de documentación, 4*, 179-191.

Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. New York, NY: Basic Books, Inc.

Parnas, D. L., Clements, P. C. y Weiss, D. M. (1985). The modular structure of complex systems. *IEEE Transactions on software Engineering*(3), 259-266.

Patel, K., Fogarty, J., Landay, J. A. y Harrison, B. (2008). *Investigating statistical machine learning as a tool for software development.* Paper presentado en Proceedings of the SIGCHI Conference on Human Factors in Computing Systems.

Patterson, D. A. y Hennessy, J. L. (2013). *Computer organization and design: the hardware/software interface*. San Francisco, CA: Newnes.

Pattis, R. E. (2012). Karel. Obtenido en http://saber-1206pcam12.blogspot.com.es/

Pattis, R. E. (2013). Karel. Obtenido en https://nclab.com/page/32/?option=com_docman&task=cat_view&gid=29&limit=20&limitstart=0&order=name&dir=ASC&Itemid=39

Pau, R. (2011). Geek IS the new Chic: Alice in a technocamp wonderland. Obtenido en https://reenapau.wordpress.com/2011/12/30/alice-in-a-technocamp-wonderland/

Peña Marí, R. (2005). *Diseño de programas: formalismo y abstracción* (3º ed.). Madrid: Pearson Educación.

Peppler, K. A. y Kafai, Y. B. (2007). *Collaboration, computation, and creativity: media arts practices in urban youth culture.* Paper presentado en Proceedings of the 8th iternational conference on Computer supported collaborative learning.

Pérez Álvarez, M. (1995). Fracaso del conductismo Watsoniano y éxito del punto de vista conductista. *Acta Comportamentalia: Revista Latina de Análisis del Comportamiento, 3*(3).

Piaget, J. (2008). Intellectual evolution from adolescence to adulthood. *Human Development, 51*(1), 40-47.

Piaget, J. (2013). *The construction of reality in the child* (Vol. 82). Great Britain: Routledge.

Piaget, J. y Fernández Buey, F. (1972). *Psicología y Pedagogía*. Barcelona: Ariel.

Polya, G. (1957). *How to Solve it: A New Aspects of Mathematical Methods*: Prentice University Press.

Polya, G. y Zugazagoitia, J. (1965). *Cómo plantear y resolver problemas*. México, DF: Trillas.

Prensky, M. (2008). Programming is the new literacy. *Edutopia magazine*.

Pressley, M., Willoughby, T., Woloshyn, V. E. y Wood, E. (1990). Elaborative interrogation facilitates adult learning of factual paragraphs. *Journal of Educational Psychology, 82*(3), 513.

Progopedia. (2015). Baltie 3. Obtenido en http://progopedia.com/language/baltie/

Psychoslave. (2016). Posible interpretación del Cono de Dale. Obtenido en https://commons.wikimedia.org/wiki/File:Triangulo_del_aprendizaje.svg

Pugh, K. (2006). *Interface-oriented design*. Dallas, TX: Pragmatic Bookshelf.

QFC Design. (2016). Desktop Dungeons. Obtenido en http://store.steampowered.com/app/226620

Quero Catalinas, E. (2002). *Sistemas operativos y lenguajes de programación*. Madrid: Editorial Paraninfo.

Quintana, C., Carra, A., Krajcik, J. y Soloway, E. (2001). Learner-centered design: Reflections and new directions.

Rachum, R. (2015). Python Turtle (Version 0.1.2009.8.2.1) [Aplicación Informática].

Rahikainen, R. (2002). *Learning through cognitive and collaborative problem-solving processes in technological product development*. Finland: Tampere University Press.

Ramírez, E. (2015). Alpha: una notación algorítmica basada en pseudocódigo. *Télématique: Revista Electrónica de Estudios Telemáticos, 14*(1), 97-121.

Real Academia Española. (2014). *Diccionario de la lengua española (23ª ed.)*. Obtenido en http://dle.rae.es/

Real Decreto 1006/1991, del 14 de junio, por el que se establecen las enseñanzas mínimas correspondientes a la Educación Primaria. Boletín Oficial del Estado, núm. 152, pp. 21191 a 21193 (1991a).

Real Decreto 1007/1991, de 14 de junio, por el que se establecen las enseñanzas mínimas correspondientes a la Educación Secundaria Obligatoria. Boletín Oficial del Estado. núm.152, pp. 21193 a 21195 (1991b).

Real Decreto 1345/1991 del 6 de septiembre por el que se establece el currículo de la Educación Secundaria Obligatoria. Boletín Oficial del Estado. núm. 220, pp. 30228 a 30231 (1991c).

Real Decreto 3473/2000, de 29 de diciembre, por el que se modifica el Real Decreto 1007/1991, de 14 de junio, por el que se establecen las enseñanzas mínimas correspondientes a la educación secundaria obligatoria. Boletín Oficial del Estado. núm. 14, pp. 1810 a 1858 (2000).

Real Decreto 1513/2006, de 7 de diciembre, por el que se establecen las enseñanzas mínimas de la Educación primaria. Boletín Oficial del Estado. núm. 29. pp. 43053 a 43102 (2006a).

Real Decreto 1631/2006, del 29 de diciembre, por el que se establecen las enseñanzas mínimas de la Educación Secundaria. Boletín Oficial del Estado. núm. 5. pp. 677 a 773 (2006b).

Regan, G. (2008). *A brief history of computing*. Germany: Springer Science & Business Media.

Reinhold, A. (2006). IBM 1130 program punch card. Obtenido en https://upload.wikimedia.org/wikipedia/commons/d/d8/IBM1130CopyCard.agr.jpg

Repenning, A. (2014). AgentSheets. Obtenido en http://www.agentsheets.com/education/index.html

Resnick, L. B. y Glaser, R. (1975). Problem Solving and Intelligence. En L. B. Resnick (Ed.), *The nature of intelligence* (pp. 205-230). Hillsdale, MI: Lawrence Erlbaum Associates.

Resnick, M., et al. (2009). Scratch: programming for all. *Communications of the ACM, 52*(11), 60-67.

Resnick, M. y Rosenbaum, E. (2013). Designing for tinkerability. En M. Honey y D. E. Kanter (Eds.), *Design, make, play: Growing the next generation of STEM innovators* (pp. 163-181). Great Britain: Routledge.

Revuelta Guerrero, R. C. y Escolano Benito, A. (2003). *La educación en la España contemporánea. Políticas educativas, escolarización y culturas pedagógicas*. Madrid: JSTOR.

Rico, L. (1995). Errores y dificultades en el aprendizaje de las matemáticas.

Riley, D. D. y Hunt, K. A. (2014). *Computational thinking for the modern problem solver*. Boca Raton, FL: CRC Press.

Ritchie, D. M. y Thompson, K. (1978). The UNIX time-sharing system. *The Bell System Technical Journal, 57*(6), 1905-1929.

Robinson, M. A., Gilley, W. F. y Uhlig, G. E. (1988). The effects of guided discovery Logo on SAT performance of first grade students. *Education, 109*(2).

Rodríguez Mondejar, F. (2000). Las actitudes del profesorado hacia la informática. *Pixel-Bit. Revista de Medios y Educación*(15), 91-103.

Rodríguez Sala, J. J. (2003). *Introducción a la programación. Teoría y práctica: teoría y práctica*. Alicante: Editorial Club Universitario.

Rodríguez, T. (2012). Genbeadev. Obtenido en https://www.genbetadev.com/herramientas/google-blockly-un-lenguaje-visual-para-aprender-a-programar

Rojas, R. (2015). A tutorial introduction to the lambda calculus. *arXiv preprint arXiv:1503.09060*.

Rosson, M. B. y Carroll, J. M. (1996). The reuse of uses in Smalltalk programming. *ACM Transactions on Computer-Human Interaction (TOCHI), 3*(3), 219-253.

Rushkoff, D. (2010). *Program or be programmed: Ten commands for a digital age*. Nueva York, NY: Or Books.

Rushkoff, D. (2012). Code literacy: A 21st-century requirement. *Edutopia, 2016*.

Rychen, D. y Salganik, L. (2000). Definition and Selection of Competencies: Theoretical and Conceptual Foundations (DeSeCo) Background Paper Neuchâtel: DeSeCo Secretariat: Paris: OECD Programe" Development of Students' Key Competencies in Basic School (grades 58) and also aims to develop scientific, learning–to-learn and communication in the mother tongue competences. Education and Training.

Sacristan, A. I. (2002). MSWLogo 6.5b Versión en Español.

Saines, G., Erickson, S. y Winter, N. (2017). CodeCombat. Obtenido en https://codecombat.com

Sánchez Rosal, A. A. (2012). Incorporacion de las TICs en el aprendizaje de la matemática en el sector universitario. *Revista de Educación Matemática, 27*(3).

Sánchez-Prieto, J. C., Olmos-Migueláñez, S. y García-Peñalvo, F. J. (2016). Informal tools in formal contexts: Development of a model to assess the acceptance of mobile technologies among teachers. *Computers in Human Behavior, 55*, 519-528.

Sartori, G. (1998). La opinión teledirigida. Videopolítica. *Claves de razón práctica*(79).

Schunk, D. H. (1997). *Teorías del aprendizaje*. Upper Saddle River, NJ: Pearson Educación.

Scirra. (2017). Construct 2. Obtenido en https://www.scirra.com/construct2

Seehorn, D. y Pirmann, T. (2016). *CSTA K-12 Computer Science Standards*. New York, NY: The Association for Computing Machinery, Inc.

Shannon, C. E. (2001). A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review, 5*(1), 3-55.

Shapes Robotics. (2016a). Eytam. Obtenido en http://www.eytam.com/home

Shapes Robotics. (2016b). Eytam: Mama documentation. Obtenido en http://www.eytam.com/mama/doc

Shapes Robotics, G. o. L. (2016). Eytam. Obtenido en http://www.eytam.com/mama/instructor

Shneiderman, B. y Leavitt, M. O. (2006). Research-based web design and usability guidelines. *Washington DC, Department of Health and Human Services*.

Siemens, G. (2005). Connectivism: A learning theory for the digital age. *International Journal of Instructional Technology and Distance Learning*.

Siemens, G. (2012). What is the theory that underpins our moocs? *Elearnspace*.

Siemens, G. y Leal Fonseca, D. E. (2007). Conectivismo: Una teoría de aprendizaje para la era digital.

Skinner, B. F., Ardila, R. y Barrera, F. (1977). *Sobre el conductismo*: Fontanella.

Solano Mata, A. A., Yong Morales, G. A. y Camacho Brenes, A. S. (2007). Introducción a los Lenguajes de Cuarta Generación (4GL).

Soloway, E., Ehrlich, K. y Bonar, J. (1982). *Tapping into tacit programming knowledge.* Paper presentado en Proceedings of the 1982 conference on Human factors in computing systems.

Sonderegger, A. y Sauer, J. (2010). The influence of design aesthetics in usability testing: Effects on user performance and perceived usability. *Applied ergonomics, 41*(3), 403-410.

Stencyl LLC. (2017). Stencyl. Obtenido en http://www.stencyl.com/

Studio, C. (2015). Star Wars: Building a Galaxy with Code. Obtenido en https://studio.code.org/download/starwars

Stylos, J. y Clarke, S. (2007). *Usability implications of requiring parameters in objects' constructors.* Paper presentado en Software Engineering, 2007. ICSE 2007. 29th International Conference on.

Stylos, J., Clarke, S. y Myers, B. (2006). *Comparing API design choices with usability studies: A case study and future directions.* Paper presentado en Proceedings of the 18th PPIG Workshop.

Stylos, J., Faulring, A., Yang, Z. y Myers, B. A. (2009). *Improving API documentation using API usage information.* Paper presentado en Visual Languages and Human-Centric Computing, 2009. VL/HCC 2009. IEEE Symposium on.

Stylos, J. y Myers, B. (2007). *Mapping the space of API design decisions.* Paper presentado en Visual Languages and Human-Centric Computing, 2007. VL/HCC 2007. IEEE Symposium on.

SurfScore. (2016). Kodable. Obtenido en https://www.kodable.com/

Tardif, M. (2012). El oficio docente en la actualidad. Perspectivas internacionales y desafíos a futuro. *POLÍTICAS DOCENTES*.

Taringa. (2016). Minecraft. Obtenido en http://k38.kn3.net/taringa/1/1/7/2/7/4/54/serverminecraft/E5B.jpg?7309

Team Dakota. (2015). Project Spark [Aplicación Informática]. Obtenido en http://www.projectspark.org/

The Adventure Maker Team. (2008). AdventureMaker. Obtenido en http://www.adventuremaker.com/

The Hybrid Group. (2014). KidsRuby (Version 1.4) [Aplicación Informática].

Thompson, C. (2011). How Khan Academy is changing the rules of education. *Wired Magazine, 126*, 1-5.

THR Games. (2015). Laby. Obtenido en http://laby.thr.pm/

Tickle Labs. (2016). Tickle. Obtenido en https://tickleapp.com/

TIOBE. (2017). TIOBE - The software quality company. Obtenido en https://www.tiobe.com/

Tomorrow Corporation. (2017). Human Resources Machine [Aplicación Informática]. Obtenido de https://tomorrowcorporation.com/humanresourcemachine

Toth, V. T. (2016). VToth. Obtenido en https://www.vttoth.com/CMS/projects/49-w-a-simple-programming-language

Tourón, J. y Santiago, R. (2015). El modelo Flipped Learning y el desarrollo del talento en la escuela: Flilpped Learning model and the development of talent at school. *Revista de Educación, 368*, 176-208.

Trianes Torres, M. V. y Gallardo Cruz, J. A. (2004). Psicología de la educación y del desarrollo en contextos escolares.

Tucker, A., et al. (2011). *CSTA K-12 Computer Science Standards*.

Tulach, J. (2008). *Practical API design: Confessions of a java framework architect*. New York, NY: Apress.

Turing, A. M. (1937). On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London mathematical society, 2*(1), 230-265.

Turing, A. M. y Copeland, B. J. (2004). *The essential Turing*. Great Britain: Oxford University Press.

Tuya, M. (2006). Todo lo que quiso saber sobre la Ley de Moore y nunca se atrevió a preguntar. Obtenido en https://www.baquia.com/emprendedores/todo-lo-que-quiso-saber-sobre-la-ley-de-moore-y-nunca-se-atrevio-a-preguntar

U.S.Congress. (1998). Section 508 of the Rehabilitation Act. Obtenido en http://www.section508.gov/section508-laws

UK Department of Education. (2013). National curriculum in England: computing programmes of study. Obtenido en https://www.gov.uk/government/publications/national-curriculum-in-england-computing-programmes-of-study

Unity Technologies. (2017). Unity. Obtenido en https://unity3d.com/es/unity

Valles Martínez, M. S. (2002). Ventajas y desafíos del uso de programas informáticos (eg ATLAS. ti y MAXqda) en el análisis cuantitativo: una reflexión metodológica desde la grounded theory y el contexto de la investigación social española: Fundación Centro de Estudios Andaluces.

Valles Martínez, M. S. (2003). *Técnicas cualitativas de investigación social. Reflexión metodológica y práctica profesional.* Madrid: Síntesis.

Valls, E. (1993). *Los procedimientos: aprendizaje, enseñanza y evaluación*. Barcelona: Horsori Editorial.

Van Nooten, B. (1993). Binary numbers in Indian antiquity. *Journal of Indian philosophy, 21*(1), 31-50.

Van Oosten, J. (2012). 3D GEP: Introduction to Unity. Obtenido en https://www.3dgep.com/introduction-to-unity-3-5/

Vandal. (2014). Project Spark. Obtenido en http://www.vandal.net/juegos/xbone/project-spark/21302

Varela, R. (2015). Vandal. Obtenido en http://www.vandal.net/noticia/1350667215/rpg-maker-mv-muestra-sus-novedades-en-video/

Vee, A. (2013). Understanding computer programming as a literacy. *Literacy in Composition Studies, 1*(2), 42-64.

Verhagen, P. (2006). Connectivism: A new learning theory.

Viana, R. (2012). Cargo-Bot. Obtenido en https://twolivesleft.com/CargoBot/

Villalobos, D. (2014). FayerWayer. Obtenido en https://www.fayerwayer.com/2014/06/google-lanza-iniciativa-made-with-code-que-invita-a-jovenes-mujeres-a-codificar/

Voss, J. F. (1989). Problem solving and the educational process. *Foundations for a psychology of education*, 251-294.

Vygotsky, L. S. (1986). *Thought and language (rev. ed.)*. Cambridge, MA: MIT Press.

W3C. (1999). XSL Transformations (XSLT) Version 1.0. Obtenido en https://www.w3.org/TR/xslt

W3C. (2008). Web content accessibility guidelines (WCAG) 2.0.

Wallas, G. (1921). The art of thought. . Nueva York: Harcourt, Brace & World, Inc.

Wallon, H. y Palacios González, J. (1980). *Psicología del niño: una comprensión dialéctica del desarrollo infantil*. Madrid: Pablo del Rio, Editor.

Watson, J. B. (1903). *Animal education: An experimental study on the psychical development of the white rat, correlated with the growth of its nervous system* (Vol. 4). Chicago, IL: University of Chicago Press.

Watson, J. B. (1913). Psychology as the behaviorist views it. *Psychological review, 20*(2), 158.

Weinberg, G. M. (1971). *The psychology of computer programming* (Vol. 932633420). New York, NY: Van Nostrand Reinhold.

Werner, L., Campe, S. y Denner, J. (2012). *Children learning computer science concepts via Alice game-programming.* Paper presentado en Proceedings of the 43rd ACM technical symposium on Computer Science Education.

Werner, L. y Denning, J. (2009). Pair programming in middle school: What does it look like? *Journal of Research on Technology in Education, 42*(1), 29-49.

Wiener, N. (1948). *Cybernetics*. New York, NY: JSTOR.

Williams, L. (2001). *Integrating pair programming into a software development process.* Paper presentado en Software Engineering Education and Training, 2001. Proceedings. 14th Conference on.

Williams, L. y Kessler, R. (2002). *Pair programming illuminated*. Massachusetts, MA: Addison-Wesley Longman Publishing Co., Inc.

Wilson, J. O., Rosen, D., Nelson, B. A. y Yen, J. (2010). The effects of biological examples in idea generation. *Design Studies, 31*(2), 169-186.

Wilson, J. W., Fernández, M. L. y Hadaway, N. (1993). Mathematical problem solving. *Research ideas for the classroom: High school mathematics*, 57-58.

Wimi5. (2016). Wimi5. Obtenido en http://wimi5.com/es/

Wing, J. M. (2006). Computational thinking. *Communications of the ACM, 49*(3), 33-35.

Wing, J. M. (2011). Research Notebook: Computational Thinking – What and Why? *The Link. The Carnegie Mellon University School of Computer Science*.

Winner, L. (2001). Dos visiones de la civilización tecnológica. *Del texto Ciencia, tecnología, sociedad y cultura*, 55-65.

Wolfram. (2017). Mathematica de Wolfram. Obtenido en https://www.wolfram.com/mathematica/

Wolfram, S. (2002). *A New Kind of Science*. Champaign, IL: Wolfram Media Inc.

Woolfolk, A. E. (1999). *Psicología Educativa*. México: Prentice Hall.

Wuensch, K. L. (2005). What is a Likert Scale? and How Do You Pronounce'Likert?'. *East Carolina University, 4*.

Yarbro, J., Arfstrom, K. M., McKnight, K. y McKnight, P. (2014). Extension of a review of flipped learning: Flipped Learning Network.

Zachtronics. (2015). Infinifactory. Obtenido en http://store.steampowered.com/app/300570/

Zapata-Ros, M. (2015). Pensamiento computacional: Una nueva alfabetización digital. *Revista de Educación a Distancia*(46).

Zobel, J. (2014). *Writing for computer science* (3º ed. Vol. 8). Germany: Springer.

Zona Outdoor. (2017). LEGO Mindstorms. Obtenido en http://www.blauden.com/robot-lego-mindstorms-ev3

ANEXOS

16. ANEXO 1. Respuestas abiertas en el formulario

Tabla 119
Respuestas de los participantes a la pregunta "¿Qué es lo primero en lo que te fijas al abrir Scratch?"

¿Qué es lo primero en lo que te fijas al abrir Scratch?
en el gato del principio y en todas las opciones que tienen para manejar el muñeco
en el gato y en que voy a hacer
En las imágenes y los diferentes floques.
en el muñeco que es el gato
el gato como si estuviera corriendo
Que el circuito vaya bien
En el personaje, porque pienso en cual poner y despues pondré el código para que se mueva.
en los bloques de la izquerda
en el gato y en en las cosas que puedo hacer con él
en el muñeco
en el gato
el gato
en los colorines de cada tipo de indicacion, te atraen para ver que son y probar con ellos y asi puedes diferenciar los tipos de indicaciones.
los personajes que voy a usar
En lo primero que me figo es en el gato, por que es lo primero que se ve.
Luego la ventana donde ponemos los controles y las ordenes.
el gato y todos sus controles
todos sus controles
a ver si el gato va en 90 grados
En el gato
en el gatro
el gatooo ^..^ U
En lo primero que me fijo al entrar en Scratch es que hoy un gato y en la iazquierda te sale como un panel de contron que tiene distintas cosas para controlar y elegir.
En lo primero que me fijo es en el gatito que hay en medio del recuadro, y en los programas que salen arriba a la izquierda.
El gato y en como estaba distribuido el programa
en el gato que es el muñeco que aparece lo primero
miro los proyectos y luyego lla in greso
me gusta los proyestos
Me llama la atención los proyectos.
En poner mi usuario para ver mis proyectos.
los muñecos y los proyectos que han hecho los demás.
A mi me gustasn los muñecos.
Los proyectos y usuarios.

¿Qué es lo primero en lo que te fijas al abrir Scratch?
me fijo en la palabra ingresar.
Me fijo en los proyectos destacados.
Los proyectos.
Los muñequitos.
Me fijo en los proyectos de otras personas.
me fijo en el pez
los muñecosy los proyectos que han hecho.
ingresar en es scratch.
me fijo en los proyectos de otras personas
que no me gusta
que tengas el usb metido
el gato
En el icono
En el gato y como funcionaba.
En el gatito.
el personaje
en el gato
en el gato
en el gatito
lo primero que vi fue el gatito
Cuantos tipos de bloques de programacion hay.
el gato
que no parece sre muy dificil pero luego se va complicando
En el muñeco
en el personaje
en el muñeco
en que es un video juego para aprender a hacer un video juego.
En el escenario
en las acciones que podia hacer el robot
que habia cosas muy raras.
que es un juego divertido con muchas maneras de jugar
en el gato naranja
en lo primero que me fijé fué en todas la obciones que habia para combinar y para hacer mil cosas con la figura que eligieras
En las opciones que tiene para realizar el juego.
en todas las opciones que hay
me he fijado en los tres moñecos
para poner mi cuenta
personajes,actualizaciones,juegos,tutoriales
Para poner el usuario y la contraseña
para poner mi cuenta
que podías crear programacíones , que iba a tener muchísima imaginacíon y que podía pasarmelo bien

¿Qué es lo primero en lo que te fijas al abrir Scratch?
Que podías hacer programas y hacer tus estudios
Que tenia pinta de tener mucha imaginación,y tambien en los muñecos que hacian mucha gracia
en mi cuenta de usuario y en los muñecos
los muñecos y mi cuenta
en mi cuenta de usuario y como se ingresa
Muñecos y juegos creados por otra gente
me he fijado en los tres muñequitos que aparecen
en lo primero que me fijo es en los muñecos.
En los personajes y en los juegos mas destacados.
los moñecos
En el gato.
En todas las cosas divertidas que hay.
En un gato u muchos apartados de colorines
Del gato naranja y de los bloques de programación.
En el gato que te aparece en la pantalla.
en el gatito
en los programas que tiene el gato.
En el gato que hay y muchas instrucciones que te dan.
En un gato
En un gato.
Que tenia que programar si quisiera que el gato se moviera.
En los scratch de los demás.
Que se pueden hacer muchas cosas.
En el gato
Me fijo en el gato naranja.
Cuando abro scratch en lo primero que me fijo es el el gato.
Me fijo en que siempre sale el gato en la pantalla principal.
un gato en un fondo blanco.

Tabla 120
Respuestas de los participantes a la pregunta “¿Qué es lo que más te ha gustado?”

¿Qué es lo que más te ha gustado?
lo real que es
que puedes hacer lo que te propongas
El gato que te sale nada más empezar
que puedes montar tu un juego
poider crear lo q ati te da la gana y con eso pasartelo bien
Que lo he hecho yo
Jugarlo. Además me lleno de satisfacción cuando lo juego porque lo he hecho yo y sin ayuda.
la libertad de crear mi propio juego
pensar quelo estoy pasando bien con algo que he creado yo

¿Qué es lo que más te ha gustado?
poder hacer que el muñeco dispare a cosas y que haga lo que tu quieras
que te lias y tienes que descuvrir en que as fallado
poder interactuar con muñequitos porque te ayuda a estimularte y a pensar.
hacer el juego
Pues me a gustado de todo, el juego iba genial!
que hay muchos controles y cada control sirbe para algo muy interesante.
que con unos pocos controles puedes hacer mucho
el haber sabido que lo he hecho yo
hacerlo con muchas ganas de hacer mas
q lo has creado tu y si tenia algun fallo lo podias modificar
con scratch hacer juegos y cosas para los pequeños
Lo que mas me ha gustado ha sido poder crear un tipo de juego tu mismo, no es lo normal que en un colegio se hagan estas cosas tan chulas como esta.
Lo que mas me ha gustado ha sido poder jugar con mi juego , y con los juegos de los demas.
Hacerlo yo, que aprendes mucho haciendolo y que como tienes tus propias ideas y es a tu gusto
cuando tu lo manejabas y te ha salido bien el juego y era divertido
De el profe me felicitara por mis proyectos.
crear juegos para que los demas los juegen y se diviertan.
Las conversaciones.
Los Proyectos de los demás jugadores.
los movimientos y cuando hablan
El procedimiento del juego.
Que podemos aprender mas cosas de las que sabemos.
El movimiento, los personajes, el fondo...
eldibujo y la creacion
jugar con scratch
Lo que mas me ha gustado de mi proyecto es el movimiento y los personajes.
nada
poder jugar con scratch
nada
las diversas opciones del programa
Que puedes hacer lo que querias.
Aprender a programar.
cuando terminas el juego
poder hacer que el muñeco se mueva
todo en general
crearlo
mober los personages
Todo
las diferentes progamaciones que se podian utilizar
crear personajes
Crear juegos
crear mi propio juego con mis normas y mis diseños

¿Qué es lo que más te ha gustado?
cuando hacias una historia
jugar a el.
Jugar con el juego
cuando creamos nuestrom juego,eso estaba batante bien
cuando creavamos el juego.
jugar con el scrach
que todo el mundo podia jugar
ue el muñeco que estaba haciendo todos los movimientos hacia todo lo que le decia ytodos los fondos que habia en el juego
A mi que al hacerlo en grupos pudimos aprender mas y el juego salio mucho mejor.
que las dos emos podido aprender y que las dos emos participado
jugar con los juegos de los demas
que lo e hecho yo
Mis programaciones
Que podíamos jugar,crear y divertirno
aprender SCRACH
Que puedas hacer tus propios controles
los dibujos
poder crear lo que yo quiera y los elementos que quiera
Los personajes.
cuando hago un dibujo con algun color y muebo al monieco si tocas elcolor que as puesto pierdes
Me a gustado la cantidad de cosas que se pueden hacer.
Poder crear mis propias personas.
Que es muy creativo hacerlo.
Que lo podía hacer yo y de el tema que quisiese.
El sencillo proceso de programación.
Cuando se movían los personajes.
hacer que dispare
que puedes empezar a programar de una manera sencilla y facil.
Que lo he podido hacer yo y he podido aprender realizandolo.
divertirme un poco
Divertirme jugando y crear mi propio juego.
Que cuándo te pillaban te quitaban vidas y me rei mucho
Que puedas hacer muchas cosas.
Poder jugar tú mismo.
Poder jugar a los juegos que yo he creado y compartirlos para que las gente los pueda probar.
Ver como programando, puedo hacer muchas cosas.
Lo que más me ha gustado de mi juego es que lo he creado yo.
Lo que más me ha gustado de mi juego a sido que lo he creado yo misma y eso requiere mucho esfuerzo.
Jugar con mi juego.

Tabla 121
Respuestas de los participantes a la pregunta "¿Y lo que menos te ha gustado de tu juego?"

¿Y lo que menos te ha gustado de tu juego?
que habeces se quedda trabado
que te pone de los nervios
que es un poco lioso
pues que a veces lo poinia bien lo cambiaba y lo ponia otra vez como antes y funcionaba
que a veces no me salga
Cuando ha tenido algún fallo.
el repetir lo que me salia mal
no poder poner sonido
cuando lo programas y no te sale lo que quieres hacer o no encuentras el boton que quieres
nada
la verdad esque esta todo bastante bien
que no funcionaba muy bien y lo tenia que volver hacer y me volvia loco
Al rato te aburres
que hay controles que no se entienden.
que hay controles que no se entienden y que si ago un juego de que si toca este color aga tarara no lo hace
que habia cosas mal hechas
no entenderlo
q cuando algo no me salia me cabreaba y no quieria terminarlo o no queria seguir
Lo que menos me ha gustado ha sido que me liara mucho y en varios casos pedir ayuda.
Lo que menos me ha gustado de mi juego ha sido tener que hacerlo sola.
Tener que estar modificandolo todo el rato
que algunas veces se paraba y no funcionaba o que algunas cordenadas estaban mal y no salia bien
De que no supiera como hacerlos.
de momento nada
Nada.
Los personajes de tu juego.
que se bugea mucho
La dificultad de hacerlo.
El procedimiento, por que es muy largo.
Me ha gustado todo de mi juego.
perder
dibujar
me ha gustado todo de mi juego.
todo
que habia pocas clases
todo

¿Y lo que menos te ha gustado de tu juego?
la dificultad de programar
NADA
La simpleza.
que es corto
¿Y lo que menos te ha gustado de tu juego?
me gusto todo
me ha gustado todo
jugar mucho rato con el
sinceramente nada
No se explicaban muy bién en lo s bloques.
los graficos
programar
Que cuando no sabia como hacer algo y estaba en mi casa, me pasaba una hora hasta poder hacerlo, porque no estaba ni con profesor ni con instrucciones
los graficos
es mucho lio
crearlo.
Cuando habia fallos y yo pensaba que ya estaba listo
lo que menos me a gustado a sido que iba de uno en uno
que era soso porque no sabiamos hacer gran cosa
hacerlo
que no podiamosjugar todos a la vez
que a veces no me salian las cosas
Que no tuvimos mucho tiempo para hacer el juego.
que nos a costado bastante hacer el juego
que no tenia sonido
que tienen bugs
Que no podía crear cosas porque no sabía crear muchas cosas
Que me lo pasaba muy bien
el juego era muy lento
Que es muy difícil de hacer (para mi)
Me fallaban cosas
que hubiese muchos elementos y me lio
Que hubiese muchos personajes
me ha gustado todo
Todo me a gustado en general.
Los gráficos.
Me ha gustado todo.
Que se tarda mucho y es muy complicado.
Los bloques de variables.
Cuando no sabia programarlo bien.
la nave , no me salio muy bien

¿Y lo que menos te ha gustado de tu juego?
que no respondia a algunos de los botones.
Que tardas un poco en terminarlo.
Que a veces las cosas se me movían de sitio
Que a veces, al programarlo no me salía bien.
Que si te quitaban vidas morias y tenias que empezar de nuevo.
Que muchas veces me salía mal.
Nada.
Que a veces es muy difícil programarlo.
Que algunas veces, quiero hacer muchas cosas pero algunas no me salen.
Lo que menos me ha gustado es que he tardado mucho tiempo en hacerlo y a veces me aburría.
Lo que menos me ha gustado es que hay veces que tardaba mucho en hacerlo.
Ha sido que se me perdiera el juego después de que lo hubiera terminado.

Tabla 122
Respuestas de los participantes a la pregunta "¿Por qué no has podido jugar a tu juego?"

¿Por qué no has podido jugar a tu juego?
no se
porque no lo terminé
por k nose en trar
por que no esta terminada
Porque no no he creado un juego.
por que no los e terminado
Porque no sabía.
por
por que no me ha dado tiempo
por que no había tiempo
por no me a dado tiempo
No,por que no he creado ningún juego con movimiento
por que aun no lo he terminado
porque en el tienpo de programacion estaba mas pendiente de que la programacion me saliese bien

Tabla 123
Respuestas de los participantes a la pregunta “¿Qué es lo que has tenido que corregir?”

¿Qué es lo que has tenido que corregir?
que no me iba que un personaje no hablaba y es que no recibia una señal
La posición de los bloques.
el movimiento
si lo q he dicho lo q no me gustaba que no me iba al principio y luego si
Que tuviera q moverse
Bastantes cosas, ya ni me acuerdo.
el "más que,igual que,menos que"
al girar se ponia el muñeco al reves etc.
pues a veces me falta o me sobra un boton, y no funciona como es debido
la banderita verde
ponia esperar por ejemplo 5 segundos para que saliese algo envede mandar señal para que saliera automaticamente.
me habia salido mal porque no ponia lo que tenia que poner
Los pasos que daba el muñeco, a donde iba, algunas cosas que no hacian caso... Cosas así por el estilo.
los movimientos y lo de enviar un mensaje a alguien.
que habia muchos nombres cuando pones lo de enviar a....
lo de las distancias
alguna vez he tenido que corregir la forma en la que he puesto los bloques
cuando el gato no decia hola
cosas que no quedaban bien, si era aburrido meter algo mas guay y la ortografia en el cuento interactivo
Lo que he tenido que corregir ha sido cosas como poner bien los bloques, que si hay una combersación tenia que dejar mas segundos para que diera tiempo a leerlo...
Lo que he tenído que corragir ha sido algun movimiento.
Los pasos porque ponia pocos o muchos y se me iva y los segundos para hablar
pues que las cordenadas no estaban bien y el gato no lo hacia
muchas cosas
Cosas
los comandos
Los movimientos de los personajes.
cosas
los movimientos de direccion
El movimiento y la rapidez.
Algun movimiento.
El escenario y algun movimiento.
cuando se mobia se movia mal
faltas de ortografía
algunos comandos
Cosas que no funcionavan como queria.
Un problema con los fondos.

¿Qué es lo que has tenido que corregir?
errores
cambiar el personaje
bastantes cosas
que la pelota se moviese
mober los personages
que un muñeco andara por que le daba y no funcionaba
Los movimientos
que algo no salia como queria
algunos errores.
Lo que más he tenido que que corregir era pasar de escenario en el juego y que salga todo bien
los movimientos del robot
algunos bloques porque los ponía mal.
los errores
los sensores del robot
el movimiento sobre todo.
Algunos fallos de ortografia y fallos en algun bloque.
la ortografia y poner cada personaje en su sitio
el personaje no se movia como queria,no hacian lo que programe
fallos
Muchos bloques
Un poco de todo cuando me equivocaba lo corregida etc...
ha los personages
muchas cosas: el movimiento, los personajes...
algunos proyectos que no se habían guardado
Los bloques de comando.
lo que me resultaba dificil
la programacion.
Que los personajes no me hacían caso al decirle lo que hacer.
Pues los bloques que juntaba muchos bloques y me liaba.
Las faltas.
Faltas .
Lo de enviar mensaje a ...y el cambio de fondos.
Los puntos de los contadores y las variables.
Los bloques y el sonido porque no transcurría cuando yo quería.
Las palabras,que no tenian tilde
los bloques de esperar.
Enviarle las señales a el gato y el cambio de fondo.
Poner los números bien colocados para que funcione bien,ir a instrucciones y cambiar de sentido a los personajes
He tenido que poner o quitar bloques y poner menos o más segundos para que los personajes hablaran uno después de otro y se pudiese leer bien.
Poner los bloques ordenados.
Algunos bloques.
Los movimientos y la programación.
Cuando dos personajes hacía diálogo, tenía que poner más segundos o menos...
He tenido que corregir cuando ponía bloques que no hacían falta o que no estaban en su sitio.
He tenido que corregir cuando no funcionaba porque no había puesto bien los bloques o no los encontraba.
Las faltas.

Tabla 124
Respuestas de los participantes a la pregunta “¿Qué palabras no entendías?”

¿Qué palabras no entendías?
no me acuerdo
variable la que mas pero una vez que me explicaron ya bien
Ahora mismo no me acuerdo. :-O
los operadores
los operadore
variable
no me acuerdo
No me acuerdo muy bien
sensor
no me acuerdo
Algunas
algunas
Lo que significaba Scratch
fijar x a 0
No me acuerdo de las palabras ahora mismo.
no entenido ninbunfallo
ahora mismo no me ha cuerdo
entendia todo
no me acuerdo
No me acuerdo.
apuntar hacia arriba
apuntar hacia arriba
algunas.
"no"
algunas
si pero no muchas
Ahora mismo no me acuerdo;)
no me acuerdo
Las palabras que no sabia
cambiar x por 10
las que estavan en ingles.
untintled
Entendia todas las palabras.
Ahora no se...
Ahora las entiendo mas o menos
No me acuerdo.
Ir a x.
Operadores, una variable...
Lápiz y variables.

Tabla 125
Respuestas de los participantes a la pregunta "¿Qué hubieses añadido a tu juego para que sea como lo habías pensado?"

¿Qué hubieses añadido a tu juego para que sea como lo habías pensado?
que tubiese mas cosas para entretener porque es divertido pero corto
que este un poco más ordenado
Que el muñeco tendria una pistola y disparase
más personajes y más fondos
mas realismo y que el muñeco desapareca cuando lo tocas
Bastantes cosas, es que todavia estabamos probando scratch y había cosas que no sabia como funcionaba.
Hubiese añadido más controles, más objetos...
que no tenga tantos controles
que te tenga menos botones
que el maacador vaya de dos en dos
algun bloque mas
Le habria añadido mas objetos que no salen en el scratch.
Sonidos y que la gentu huviera podido dibujar lo que quisiera
pues mas cosas para que sea mas divertido
Que sea mejor
algo diver
Personajes y Sonidos.
cambiar el color a los personajes
yO QUE SE
pos muy bonito
me lo habria pasado mejor
nada
mas fondos, mejores graficos
mejores graficos
me hubiera gustado teneer mas tiempo para terminar mi juego y tener mejores graficos me hubierab gustado
mas realismo.
Música y que algunas cosas no estubieran pixeladas
mejores personajes y mas variables
mas realismo
Pues algun bloque mas y asi hubiera sido un pelin mas divertido.
pues algun personaje y bloque mas y asi seria mejor
poniendo coches atracos sangre motosbi icicletas el patin electrico
añadir algo que lo hiciera mas divertido
bloque para crear cosas nuevas
poner mas cosas bicis el patin electrico
Muchas cosas
Pues muchas cosas
pues mas testo de los personages
un oso y una oveja

algo que entendiese
le abria puesto rallos laser a los personajes.
Creo que nada porque se acerca bastante a lo que queria.
Que cuantos mas puntos consiga mas naves conseguia y mas bichos vendrian
dibujos mios.
Algunas cosas que no se hacer pero mas adelante voy a poder hacer.
Alguna animación que no era posible hacerlo.
Más movimientos.

17. ANEXO 2. Autorización para participar en el estudio

Título del Proyecto: **Scratch como herramienta para la enseñanza de la programación en la Educación Primaria. Análisis de usabilidad en la escuela pública de la Comunidad de Madrid**

La legislación vigente establece que la participación de toda persona en un proyecto de investigación y/o experimentación requerirá una previa y suficiente información sobre el mismo y la prestación del correspondiente consentimiento. Establece igualmente el ordenamiento jurídico que cuando el sujeto sea menor de edad la autorización será prestada por los padres, quien ejerza la patria potestad o, en su caso, el representante legal del menor después de haber escuchado a éste si tiene, al menos, doce años cumplidos. A tal efecto, a continuación se detallan los objetivos y características del proyecto de investigación arriba referenciado, como requisito previo a la obtención del consentimiento que habilita para la colaboración voluntaria en el proyecto:

1) OBJETIVOS:
   · Analizar la herramienta de Scratch para saber en qué medida su diseño facilita o dificulta el aprendizaje de los lenguajes de programación.
2) DESCRIPCIÓN DEL ESTUDIO
   · A las alumnas y alumnos que han trabajado en el aula con la herramienta de Scratch se les pide rellenar una única vez un cuestionario que pretende evaluar por un lado lo que han aprendido, y por otro las dificultades que se han encontrado. Se estima que el cuestionario se tarda en rellenar entre 15 y 20 minutos.
3) POSIBLES BENEFICIOS
   · Determinar si Scratch es la mejor herramienta para enseñar a programar a niñas y niños de la Escuela Primaria, o es preferible utilizar otras alternativas.
4) POSIBLES INCOMODIDADES Y/O RIESGOS DERIVADOS DEL ESTUDIO
   · No se prevé ninguna incomodidad o riesgo. La encuesta es anónima, y no incluye preguntas de carácter personal, sólo preguntas de control como son la edad y el género de la alumna o alumno.
5) PREGUNTAS E INFORMACIÓN:
   A continuación se describen los apartados de los que consta la encuesta:
   1. Preguntas sobre las acciones que permite realizar el programa
   2. Preguntas sobre los bloques que se utilizan para programar
   3. Preguntas sobre cómo se corrigen los errores
   4. Preguntas sobre las palabras técnicas del programa
   5. Preguntas sobre el resultado de los proyectos realizados con el programa
   6. Preguntas sobre las posibles mejoras a incluir en el programa
   7. Preguntas de control para la persona que responde el cuestionario:
      a. Nombre del colegio
      b. Ciudad en la que está el colegio
      c. Curso
      d. Edad
      e. Género
      f. Experiencia previa con el programa y con el uso de ordenadores

6) PROTECCIÓN DE DATOS: Este proyecto requiere la utilización y manejo de datos de carácter personal que, en todo caso, serán tratados con las exigencias requeridas por la legislación de protección de datos vigente garantizando la confidencialidad de los mismos.

La participación en este proyecto de investigación es voluntaria y el sujeto puede retirarse del mismo en cualquier momento sin que se le pueda exigir ningún tipo de explicación ni prestación.

Y para que conste por escrito a efectos de información de las personas participantes y de sus representantes legales, se formula y entrega la presente hoja informativa.

En Madrid, a __ de _____de _____. Fdo. David Alonso, Investigador principal

HOJA DE INFORMACIÓN SOBRE PARTICIPACIÓN EN PROYECTO DE INVESTIGACIÓN Y/O EXPERIMENTACIÓN

CONSENTIMIENTO INFORMADO

D./Dª[46].....

en calidad de[47] ……..

He/hemos leído la hoja de información que se me/nos ha entregado**,** copia de la cual figura en el reverso de este documento, y la he/hemos comprendido en todos sus términos.

He/hemos sido suficientemente informado/s y he/hemos podido hacer preguntas sobre los objetivos y metodología aplicados en el proyecto de investigación "*Scratch como herramienta para la enseñanza de la programación en la Educación Primaria. Análisis de usabilidad en la escuela pública de la Comunidad de Madrid"* y para el que se ha pedido la colaboración de mi/nuestro…… (hijo, pupilo o representado cuyo nombre y apellidos es [48]

……

Comprendo/comprendemos que la participación es voluntaria y que el menor en cuya representación actúo/actuamos puede retirarse del mismo

- Cuando quiera.
- Sin tener que dar explicaciones y exponer mis motivos.
- Sin ningún tipo de repercusión negativa.

Por todo lo cual, PRESTO/PRESTAMOS EL CONSENTIMIENTO para la participación en el proyecto de investigación al que este documento hace referencia.

En ..... a …… de ..... de .....

Fdo. ……..

[46] Los padres, si ambos ejercen la patria potestad, deben firmar conjuntamente este consentimiento informado.
[47] Padres, tutor o representante legal del menor.
[48] Nombre completo del menor

18. ANEXO 3. Formulario que se utilizó en el estudio

Análisis de Scratch

Este formulario está pensado para ser rellenado por estudiantes de Primaria después de realizar alguna actividad o juego con Scratch, independientemente del número de clases que hayan recibido, o del tipo de actividad o juego realizado antes de completar el formulario. Pueden rellenarlo solos o con asistencia del profesor si es necesario.

*Obligatorio

¿Qué es lo primero en lo que te fijas al abrir Scratch?

Cuando has utilizado Scracth, ¿has podido realizar estas acciones? *

	Sí, yo sólo y a la primera	Sí, yo sólo, pero después de probar un rato	Sí, pero me han tenido que ayudar	No
Elegir un fondo	○	○	○	○
Elegir un personaje	○	○	○	○
Editar el personaje (hacerlo más grande o más pequeño)	○	○	○	○
Hacer que el personaje se mueva	○	○	○	○
Hacer que se sumen puntos	○	○	○	○
Incluir sonidos al juego	○	○	○	○

¿Has podido guardar tu juego para seguir editandolo más tarde o en otra clase?

- Sí
- No

¿Has utilizado elementos (dibujos, sonidos, etc.) creados por ti?

- Sí, dibujos y sonidos
- Sí, pero sólo dibujos
- Sí, pero sólo sonidos
- No, yo no he creado nada. Sólo he utilizado dibujos y sonidos que ya estaban en Scratch

¿Has podido jugar a tu juego?

- Sí
- No

No has podido jugar a tu juego

¿Por qué no has podido jugar a tu juego?

Preguntas sobre los bloques

Cuando vas a crear un programa, utilizas bloques que se combinan de distintas maneras: los puedes poner antes, o después, o dentro de otros bloques... ¿Cuánto has tardado en descubrir cómo se combinan los bloques?

- A la primera, tras uno o dos intentos
- Después de probar un rato
- Lo he entendido, pero alguna vez me ha tenido que volver a ayudar el profesor
- El profesor me ha tenido que ayudar todo el rato

Dime si los siguientes bloques se pueden combinar:

	Sí	No
1.por siempre -- apuntar hacia	○	○
2. repetir -- ¿tecla presionada?	○	○
3. no -- ir a x: y:	○	○
4. si entonces -- ¿tocando?	○	○

¿En algún momento algo no funcionaba como querías?

- Todo ha ido bien de principio a fin
- He tenido que corregir algunas cosas

Corregir

¿Qué es lo que has tenido que corregir?

Cuando algo no funcionaba como tú querías...

- ○ Lo he resulto yo sólo sin ninguna ayuda
- ○ He utilizado la ayuda del programa
- ○ He pedido ayuda al profesor y hemos utilizado la ayuda del programa
- ○ He pedido ayuda al profesor y me lo ha resuelto

¿Al final has conseguido que el programa funcione como tú querías, o has dejado algún problema sin resolver?

- ○ Sí, funciona perfectamente
- ○ He resuelto más de la mitad de los problemas
- ○ He resuelto menos de la mitad de los problemas
- ○ No, el programa no funciona como yo quería

Lenguaje

¿Había algunas palabras en Scratch que no entendías lo que significaban?

- ○ Si, hay palabras que no entendía para que servían
- ○ He entendido todas las palabras a la primera

Palabras no comprendidas

¿Qué palabras no entendías?

Lista de palabras

Después de utilizar Scratch, ¿sabrías decirme cuales de estas palabras están en el programa?

Scratch debe estar cerrado para responder a esta pregunta

	Si	No
Evento	○	○
Variable	○	○
Objeto	○	○
Interfaz	○	○
Operador	○	○
Sensor	○	○
Mensaje	○	○

Juego

¿Has podido crear el juego que tenías pensado?

- ○ Si, ha quedado como yo quería
- ○ Se parece mucho a lo que quería hacer, pero faltarían cosas
- ◉ Es diferente, porque no he podido hacer lo que quería
- ○ No he podido hacer un juego completo

Mejoras

¿Qué hubieses añadido a tu juego para que fuera como lo habías pensado?

Últimas preguntas

¿En qué curso estás? *

- ○ 3º de primaria
- ○ 4º de primaria
- ○ 5º de primaria
- ○ 6º de primaria
- ○ Otro:

Eres *

- ○ Mujer
- ○ Hombre

Escribe tu edad *

¿Cuántas clases has dado de Scratch antes de hacer esta encuesta? Si no lo sabes, puedes preguntar a tu profesora o profesor *

¿Cuántas clases has dado de Scratch antes de hacer esta encuesta? Si no lo sabes, puedes preguntar a tu profesora o profesor *

Además de las clases, ¿has practicado en casa?

- Sí
- No

Antes de las clases de Scratch, ¿qué experiencia tenías en utilizar un ordenador?

- Tengo un ordenador en casa y lo utilizo habitualmente (todos o casi todos los días)
- Tengo un ordenador en casa, pero lo utilizo poco (una o dos veces por semana o menos)
- En mi casa no hay ordenador, pero utilizo el del colegio, o el de un familiar o amigo, habitualmente (todos o casi todos los días)
- En mi casa no hay ordenador, pero utilizo el del colegio, o el de un familiar o amigo, de vez en cuando (una o dos veces por semana o menos)
- No suelo utilizar ningún ordenador

Ahora que hemos terminado, te propongo un reto, ¿sabrías abrir un nuevo juego, elegir un personaje y hacer que se mueva en unos minutos sin que nadie te ayude?

- Si, sabría hacerlo sólo
- Si, pero a lo mejor me tendrían que ayudar
- Me tendría que ayudar el profesor
- No sabría hacerlo

Por último, si pudieras cambiar el aspecto de Scratch, ¿qué modificarías?

Por ejemplo, añadir algo que eches de menos, mover a otro lugar algún botón, etc.

19. ANEXO 4. Análisis de herramientas que se pueden utilizar en el proceso de enseñanza-aprendizaje de la programación

En este anexo se muestra una recopilación de posibles recursos que se pueden utilizar para la enseñanza y el aprendizaje de la programación. Cuando hablamos de recursos nos referimos a entornos de programación que, por su diseño y sus contenidos, pueden ser una buena opción para iniciarse en el mundo de la programación.

El propósito de esta recopilación es, por un lado, ofrecer una visión del estado del arte de este tipo de recursos, para poder entender el contexto el que se sitúa Scratch. Por otro lado, pretende servir de fuente para posibles vías de investigación futuras que pretendan evaluar alternativas a Scratch. En los anexos se realiza una descripción más detallada de cada recurso, mientras que en este capítulo solo se muestra un resumen sus principales características.

La variedad de recursos de este tipo es inmensa, y esta recopilación no pretende ser absolutista, ni hacer un análisis exhaustivo. Además, es susceptible de someterse a constante revisión.

Para escoger las herramientas a incluir en este análisis se han tomado como referentes estudios previos en la materia (Adam y Mowers, 2013; Alonso Urbano y Hueso Rivas, 2014; Cronin, 2014; Kharbach, 2014), así como índices de popularidad de las herramientas de programación (TIOBE, 2017), y se ha llevado a cabo en colaboración con el investigador Andrés Conde.

De cada recurso se proporciona la siguiente información:

- **Herramienta:** nombre comercial de la herramienta
- **Página web:** sitio web de referencia, donde se puede encontrar información de la herramienta, y los enlaces de descarga o compra.
- **Versión en 2017 / Año:** a fecha de 2017, versión que existe disponible y año en el que fue publicada dicha versión.
- **Plataformas:** tipos de dispositivos y plataformas para las que la herramienta está disponible
- **Tipo de software / Modelo de negocio:** el tipo de software distingue si es software libre (su código fuente está disponible para copiarlo, cambiarlo, reutilizarlo, distribuirlo, etc.) o privativo (el código fuente no está disponible, y

la capacidad para usar el programa, modificarlo o distribuirlo está restringida). El modelo de negocio se refiere a si la herramienta es gratuita o de pago.

- **Edad Recomendada:** edades para las que el fabricante / desarrollador recomienda su uso**.**
- **Característica diferenciadora:** elementos que confieren a la herramienta un valor diferencial.

Así mismo, de cada recurso se proporciona una descripción de la interfaz, capturas de pantalla de la misma, y un análisis de sus principales características,

1. Adventure Maker

Tabla 126
Resumen de características de la herramienta *Adventure Maker*

Nombre de la herramienta	*Adventure Maker*
Página web	http://www.adventuremaker.com/
URL de descarga	http://www.adventuremaker.com/downloads.htm
Fecha de creación / aparición en el mercado	Año 1999
Última versión en 2017 / año de esa versión	adenture_maker_v4_7_1, del año 2008 (versión gratuita).
Plataformas en las que se puede instalar	Windows
Tipo de software	Privativo
Precio / modelo de negocio	Gratuito, pero tiene versiones de pago: sin uso comercial (69$) y de uso comercial (139$).
Edades para las que está indicado	No están especificadas.
Aspectos educativos que trabaja	Aprendizaje de la programación, desarrollo de la creatividad.
Característica más destacada / valor diferencial	Cuando la herramienta estaba en desarrollo, abarcar un cierto mercado emergente para desarrolladores de videojuegos, que podían utilizar su motor gráfico.

Se trata de una herramienta que posee un motor gráfico, tanto para 2D como 3D, y cuya potencia es apropiada para la creación de apartados gráficos o la creación de videojuegos propios, pudiendo aportar comportamientos diferentes a los objetos introducidos en el motor. En la *Figura 137* podemos ver diferentes recursos de diferentes tipos, que pueden ser subidos a la herramienta y usarse para la creación de proyectos.

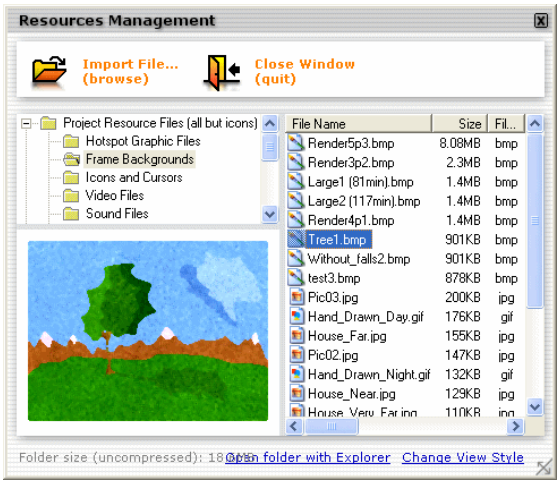


Figura 137. Recursos en *Adventure Maker*.

Fuente: (The Adventure Maker Team, 2008)

Adventure Maker posee un lenguaje de programación propio. El motor le da al desarrollador las herramientas que necesita, a través de varias ventanas, separadas por categorías, tales como imágenes, sonido, comportamientos, etc. En la Figura 138 puede verse diferentes propiedades que la herramienta ofrece al usuario para personalizarla a su gusto (generales y avanzadas).

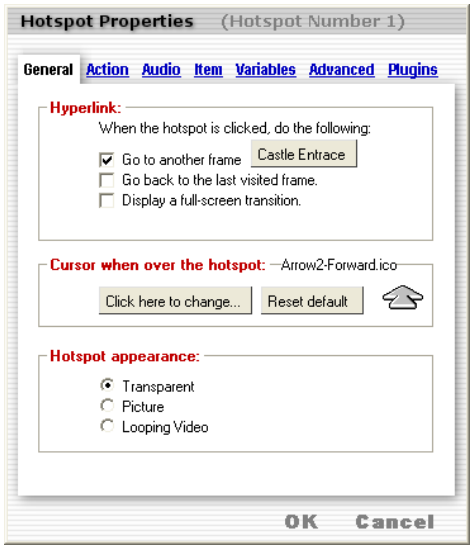


Figura 138: Propiedades de *Adventure Maker*.

Fuente: (The Adventure Maker Team, 2008)

Una de las principales ventajas de la herramienta es que ofrece un tutorial al usuario al ejecutarla, debido a su lenguaje propio, lo que facilita su enseñanza y uso. Además, puede exportarse hacia iPhone e iPod Touch, además de la web (HTML)

para poder probar lo que se cree en la herramienta. Tampoco requiere de conocimientos previos de programación o de algún lenguaje que la requiera.

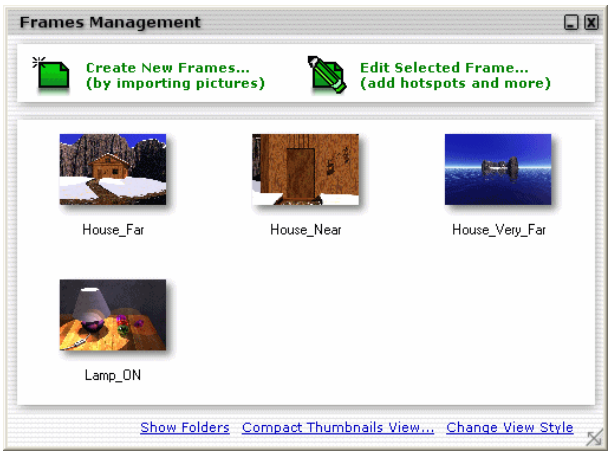


Figura 139. *Frames* para crear un nivel en *Adventure Maker*.

Fuente: (Repenning, 2014)

Sin embargo, a nivel de diseño, presenta mucha información desordenada y poco intuitiva, sin dejar claro la forma de interacción con la herramienta. Asimismo, los tutoriales se basan en bloques de texto poco dinámicos, y en ocasiones pueden generar dudas en el usuario, que no puede resolverlas dentro de la propia herramienta.

Esta herramienta dejó de desarrollarse en el año 2008.

2. Agent Sheets

Tabla 127
Resumen de características de la herramienta *Agent Sheets*

Nombre de la herramienta	*Agent Sheets*
Página web	http://www.agentsheets.com/
URL de descarga	AgentCubes: http://www.agentsheets.com/agentcubes/download/index.html AgentSheets trial: http://www.agentsheets.com/products/trial/index.html
Fecha de creación / aparición en el mercado	Año 1996 / 19 de Mayo del año 2014
Última versión en 2017 / año de esa versión	AgentCubes 2.5 Lite, del año 2016 (versión gratuita) AgentSheets4.0, del año 2016 (versión de pago, gratuita por durante tres días)
Plataformas en las que se puede instalar	JVM (Java Virtual Machine), Mac 10.8+, Windows 7+
Tipo de software	Privativo
Precio / modelo de negocio	La anterior versión era gratuita. La actual versión se vende desde los 48-67€
Edades para las que está indicado	Desde los 5 años
Aspectos educativos que trabaja	Enseñanza de principios básicos de la programación y lenguajes de programación
Característica más destacada / valor diferencial	Contaba con el apoyo de la NASA, además de ofrecer las herramientas para la creación no sólo de juegos, sino también simulaciones científicas

Agent Sheets se trata de una herramienta que permite la creación de videojuegos, y simulaciones de carácter científico. Utiliza su propio lenguaje de programación, a través del cual se puede trabajar sobre varios elementos: animaciones, sonido, texto, reacción a diferentes eventos a través del ratón o el teclado, etc. Además, posee un soporte web. En la figura superior pueden verse diferentes imágenes para la creación de una parte del proyecto que se está realizando.

La web desde donde se puede acceder a la herramienta cuenta con cuatro pestañas: *Agent Sheets* y *Agent Cubes*, *Education* (educación) y *About Us* (sobre nosotros). Desde la propia web se puede acceder a diferentes proyectos de otros usuarios, o incluso a la compra y venta de los mismos. En los menús de compra de la herramienta se muestra el precio correspondiente según la versión escogida, aunque será posible adquirir gratuitamente la última versión durante 3 días.

La pestaña *Education* (ver Figura 140) contiene una sección para docentes donde se pueden compartir y descargar recursos pedagógicos.

Figura 140. Ventana *Education* en la web de *Agent Sheets*.

Fuente: (The Adventure Maker Team, 2008)

La pestaña *About Us* esquematiza cómo surgió el desarrollo y la idea de esta herramienta.

Según los propios desarrolladores, esta aplicación está recomendada para niños y niñas a partir de los 5 años.

Enumerando las principales propiedades de la versión gratuita, podemos sacar la siguiente información respecto al uso de la herramienta:

Al hacer clic en *New Agent*, la herramienta creará un perfil desde el cual los usuarios podrán personalizar un avatar, que podrá ser visto en los bloques de código. Estos bloques utilizan un sistema de programación muy similar al de la herramienta Blockly, que veremos más adelante. Los bloques se utilizarán para construir el programa dentro de un cuadro de trabajo. Hay distintas categorías de bloques, como por ejemplo *Actions* (acciones) y *Conditions* (condiciones).

Dentro del menú *Actions*, encontramos acciones básicas, como mover o cambiar tanto mensajes de texto como sonidos.

Dentro del cuadro de trabajo, la opción *New Rule* (nueva regla) añadirá distintas opciones a los métodos elegidos. La opción *New Method* (nuevo método) creará uno nuevo, además del seleccionado. *Test* (Prueba) compilará el programa y

permitirá comprobar los resultados. En la *Figura 141* se puede ver el espacio de trabajo que permite crear métodos propios.

Figura 141. New Method en Agent Sheets.

Fuente: (The Adventure Maker Team, 2008)

3. Alice

Tabla 128
Resumen de características de la herramienta Alice

Nombre de la herramienta	Alice
Página web	www.alice.org/index.php
URL de descarga	http://www.alice.org/index.php?page=downloads/download_alice 3.1
Fecha de creación / aparición en el mercado	Año 1998 / año 2008 (alfa), año 2009 (beta)
Última versión en 2017 / año de esa versión	Alice 3.3, 22 de agosto del año 2016
Plataformas en las que se puede instalar	Windows (Vista o superior), Mac OS (10.6 o superior), Linux (Ubuntu, Red Hat)
Tipo de software	Libre
Precio / modelo de negocio	Gratuito
Edades para las que está indicado	Desde los 8 años
Aspectos educativos que trabaja	Conocimientos básicos de la programación, programación orientada a objetos
Característica más destacada / valor diferencial	Capacidad del uso del 3D y los lenguajes Java, C++ y C#

Alice contiene un amplio set de opciones, orientadas al aprendizaje de la programación. Sin embargo, tiene un precio bastante elevado si se compara con otras de su misma categoría, algunas incluso más completas, y aun así gratuitas.

El sistema para programar que ofrece es sencillo de utilizar. Este hecho, unido a que la interfaz proporciona asistencia constante al usuario, convierte a Alice en una posible herramienta para iniciarse en el mundo de la programación, y más concretamente, en el paradigma de la programación orientada a objetos, a través del lenguaje Java.

La versión 3.1, del 3 de julio de 2014, presenta unas enormes mejoras con respecto a versiones anteriores, mucho más básicas y con menos funciones. Estas mejoras hicieron que compañías de la dimensión de *Electronic Arts* se fijara en esta herramienta, y la utilizara para crear algunos de sus juegos, como *Los Sims.*

Una de las mejoras de esta versión fue la inclusión de un soporte de asistencia al usuario, que incluye un FAQ (preguntas y dudas comunes), como puede verse en la *Figura 142.*

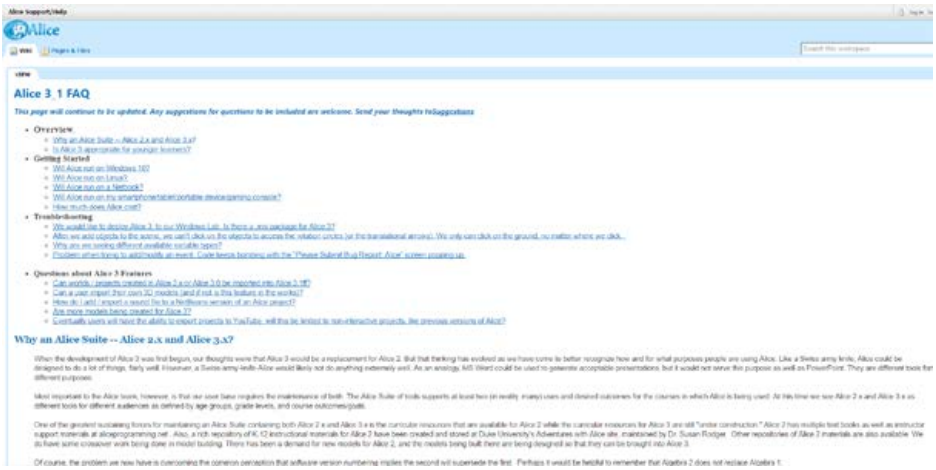


Figura 142. FAQ de la web de la herramienta Alice.

Fuente: (Carneige Mellon University, 2017)

Al iniciar la herramienta se ofrece al usuario abrir un proyecto en modo básico, o en modo avanzado, con una escena ya creada u otra en blanco, como se puede apreciar en las *Figura 143* y *Figura 144.*

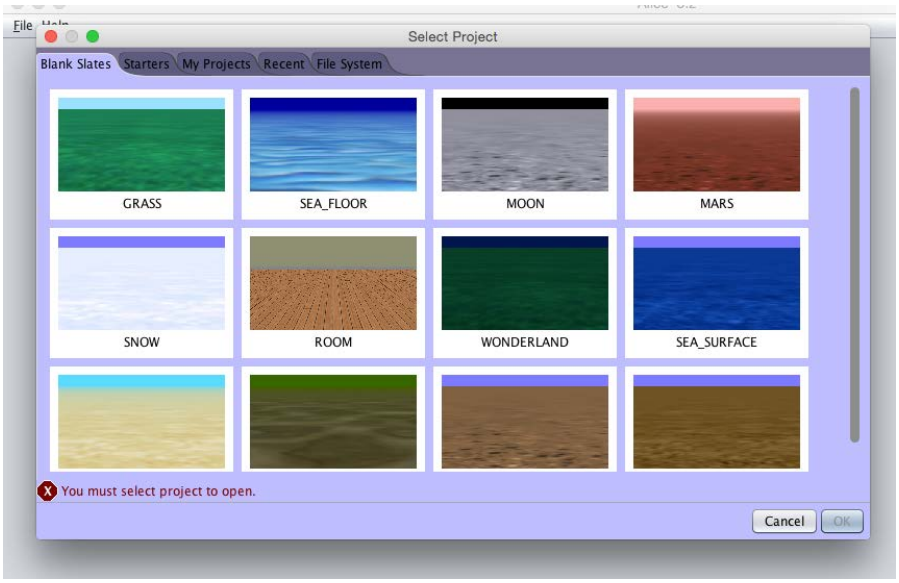


Figura 143. Proyecto en blanco en Alice 3.2.

Fuente: (Carneige Mellon University, 2017)

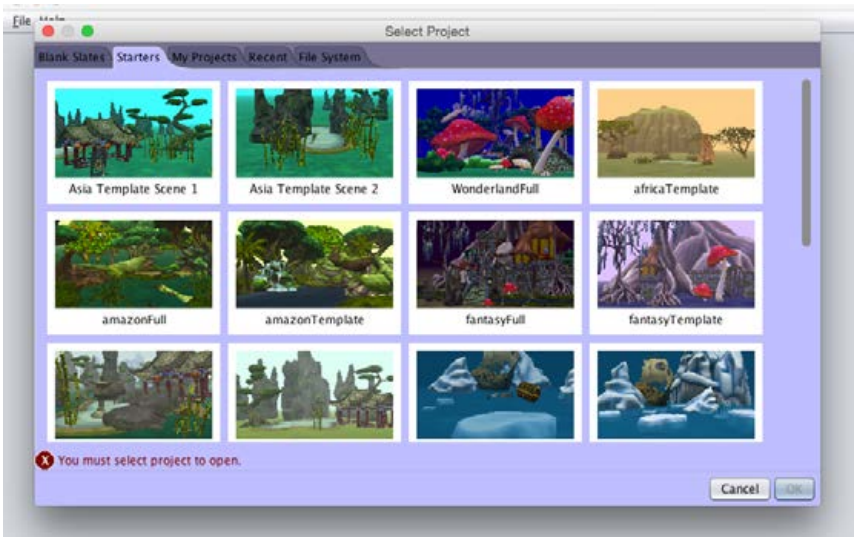


Figura 144. Proyecto avanzado como base en Alice 3.2.

Fuente: (Carneige Mellon University, 2017)

El usuario puede elegir entre varias opciones de escena, todas con un estilo *cartoon* (dibujos animados), y con colores llamativos. En la *Figura 145*, se muestra un ejemplo de una escena con montañas y nieve.

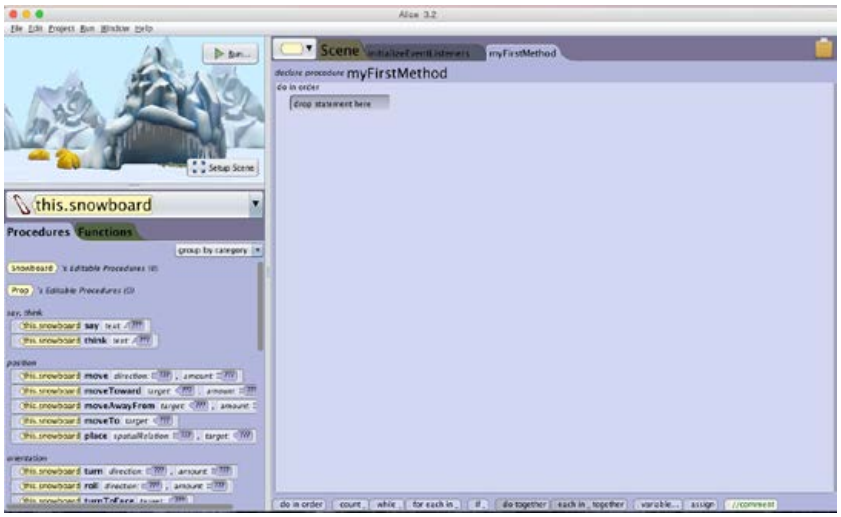


Figura 145. Escena nevada en Alice 3.2.

Fuente: (Carneige Mellon University, 2017)

Cuando se crea una nueva escena, el usuario puede añadir a ella diferentes objetos y *props* [49], y modificar su comportamiento. En general, su manejo se asemeja al de otros *engines* (motores) profesionales utilizados para crear videojuegos. En la *Figura 146* puede observarse los comportamientos, funciones y propiedades que la clase "*Prop*" contiene.

Figura 146. *Props* de una escena en Alice 3.2.

Fuente: (Carneige Mellon University, 2017)

En la *Figura 147* se muestra uno de los métodos creados para el proyecto ("*myFirstMethod*") y en el que se define su comportamiento.

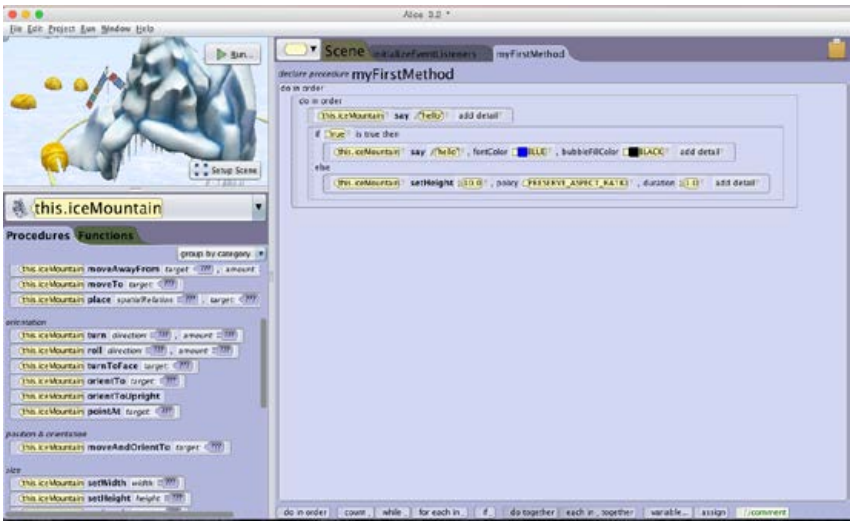


Figura 147. Escena nevada con métodos en Alice 3.2.

Fuente: (Carneige Mellon University, 2017)

[49] El término *prop* en el contexto de los videojuegos se refiere a la utilería o atrezo que tiene una escena (mesas, sillas, barriles, etc.)

En la *Figura 148* se pueden ver los diferentes métodos, funciones y propiedades de la clase "*Scene*".

Figura 148. Escena nevada con métodos 2 en Alice 3.2.

Fuente: (Carneige Mellon University, 2017)

En la pestaña *Help* (ayuda), se puede enviar el proyecto a una base de datos online en la cual se representan los elementos de nuestro trabajo, con un máximo de capacidad.

El programa consta con dos marcos de trabajo:

- El primero, situado en el extremo superior izquierdo, es un espacio donde se muestra la escena, en la cual podemos añadir y manipular elementos. Cada elemento abrirá sus respectivos comportamientos en el marco de trabajo de programación. El marco central es sobre el que se trabaja el código.

- El segundo marco, justo debajo anterior, ofrece procesos y funciones a elegir, para montar el código en el marco central. Esta forma de estructurar el código (seleccionar el código, arrastrarlo y soltarlo en el lugar adecuado) resulta más o menos intuitiva. Sin embargo la ausencia de una ayuda que indique los pasos a seguir hace imprescindible la presencia de un docente que explique y tutele los ejercicios a realizar.

Si en el marco central, movemos la pestaña *My First Method* (en la que se construye el código), a *Scene* (escena), se puede observar a la izquierda, donde antes se encontraba el código seleccionable, la jerarquía de las clases existentes.

En este marco, si se selecciona cualquiera de las clases, se podrá modificar procesos, funciones y propiedades.

Por último, en su marco visual, encontramos el botón *Run* (ejecutar), el cual compila el programa y permite visualizar el proyecto realizado, e interactuar con él.

El valor didáctico de esta aplicación es superior al de otras herramientas, por la variedad de recursos que ofrece siendo gratuita. Además, es una herramienta que puede resultar útil a profesionales que no tienen muchos conocimientos de programación, para realizar prototipos, o incluso pequeñas aplicaciones relacionadas con la animación o los videojuegos.

4. Baltie

Tabla 129
Resumen de características de la herramienta Baltie

Nombre de la herramienta	Baltie
Página web	http://www.sgpsys.com/en/
URL de descarga	http://www.sgpsys.com/en/download.asp
Fecha de creación / aparición en el mercado	Año 1996
Última versión en 2017 / año de esa versión	SGP Baltie 3 y SPG Baltie 4 C# Pro , versiones del año 2017
Plataformas en las que se puede instalar	Windows
Tipo de software	Privativo
Precio / modelo de negocio	Herramienta de pago. Existen diferentes versiones, cada una con un precio diferente según la versión, número de ordenadores y el tiempo que vaya a utilizarse: SGP Baltie 3, entre los 12$ - 139$ por año, 500$ de forma ilimitada SGP Baltie 4 C# Pro, entre los 25$ - 189$ por año, 800$ de forma ilimitada
Edades para las que está indicado	SGP Baltie 3 para edades entre los 6 – 16 años SPG Baltie 4 C# Pro para edades desde los 13 años
Aspectos educativos que trabaja	SGP Baltie 3, fundamentos básicos sobre la programación SPG Baltie 4 C# Pro, programación orientada a objetos
Característica más destacada / valor diferencial	Puede usarse como un método de enseñanza en escuelas y es una herramienta que requiere muy poco para poder usarse

Baltie es una herramienta orientada a la enseñanza de conceptos básicos sobre programación. Cuenta con diferentes versiones, con distinta orientación en las estructuras de programación que utilizan, así como en la forma de presentar el código o los utensilios necesarios para la creación del mismo. Nos centraremos principalmente en dos versiones: SGP Baltie 3 y SPG Baltie 4 C# Pro.

Baltie 3 propone un mecanismo para programar basado en la selección y la colocación de símbolos. Cada símbolo contiene un código y unas funciones propias, que puede interconectarse con otros arrastrándolo desde la parte superior hacia la parte inferior de la interfaz. Estas conexiones crearán los diferentes comportamientos que constituirán el programa que se quiera codificar. Esta forma de programación puede ser una buena opción para iniciarse en el ámbito de la programación y del pensamiento algorítmico.

Un botón con forma de triángulo verde, en la parte superior, permite ejecutar el programa y ver los resultados. En la *Figura 149* se puede ver los diferentes métodos que Baltie ofrece a los usuarios al comenzar a usar la herramienta. En la Figura 150 se puede observar una serie de símbolos conectados entre sí para realizar una función (parte inferior).

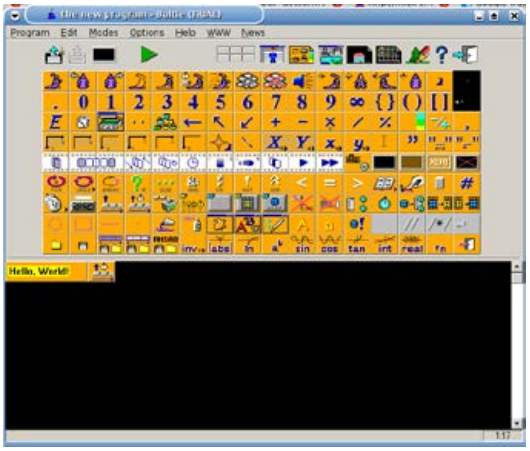


Figura 149. Métodos en Baltie 4.

Fuente: (Carneige Mellon University, 2017)

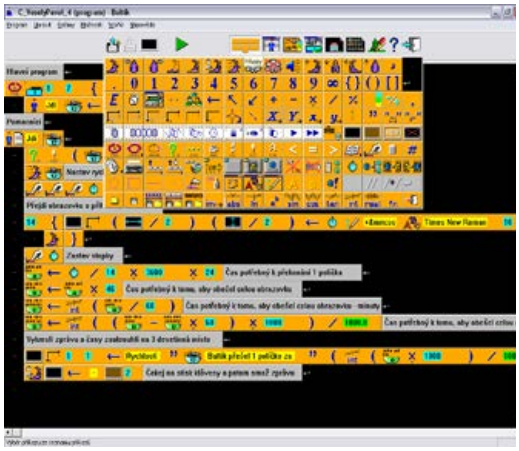


Figura 150. Métodos y conexiones de Baltie 4.

Fuente: (Carneige Mellon University, 2017)

Baltie 4 posee algunas funciones adicionales a las de su predecesor, como por ejemplo, la posibilidad de crear ejecutables (.exe), de utilizar objetos 3D, o la de usar como lenguaje de programación C#, a cuyo código se puede acceder. Es decir, sigue existiendo la posibilidad de programar arrastrando y soltando símbolos, solo que en esta versión se puede acceder al código subyacente. C# es un lenguaje que sigue el paradigma de la programación orientada a objetos. En la *Figura 151* se puede ver un ejemplo de la interfaz donde se accede al código.

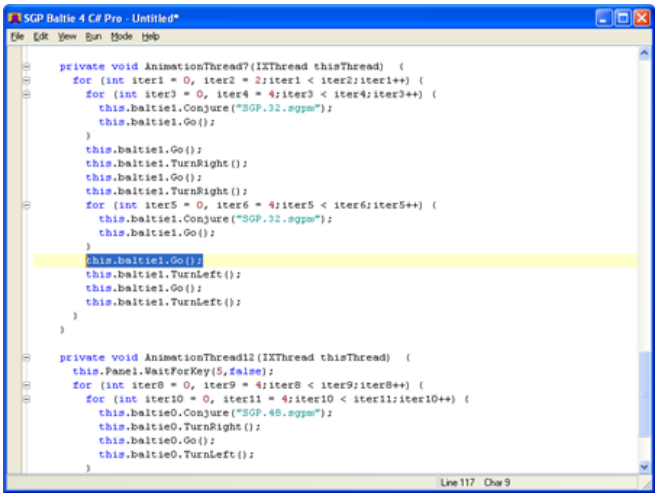


Figura 151. Código en Blatie 4.

Fuente: (Carneige Mellon University, 2017)

En la *Figura 152* se muestra una de las novedades que incluye Baltie 4, su modo 3D. En la parte derecha de la figura puede verse el código generado a través de los símbolos que están en la parte inferior.

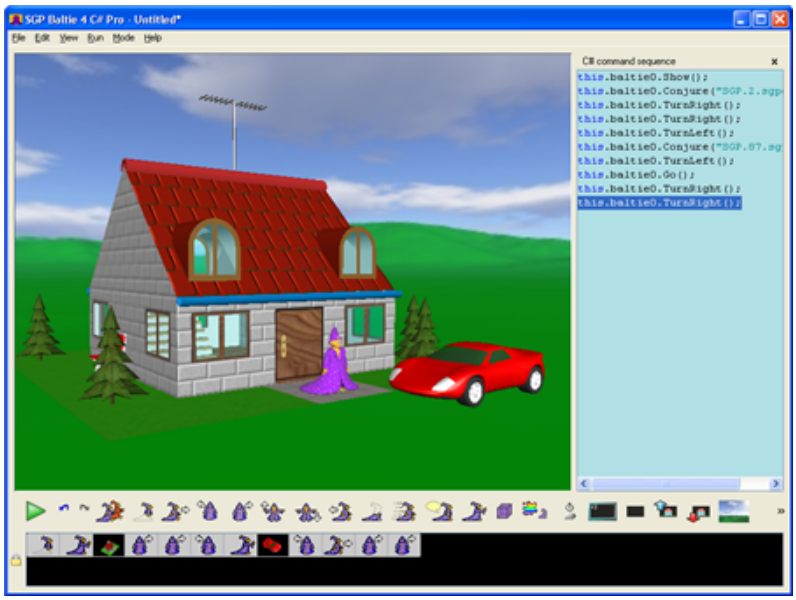


Figura 152. Modo 3D de Baltie 4.

Fuente: (Progopedia, 2015)

Baltie 3, por el contrario, sólo permite la creación de espacios 2D, pero ambos mantienen la posibilidad de cargar imágenes propias, o modificar las ya existentes, como puede verse en la *Figura 153*.

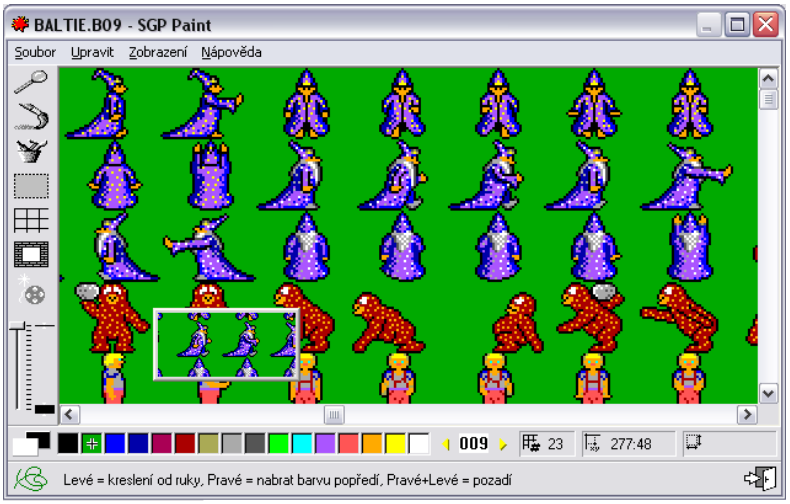


Figura 153. Imágenes creadas en el modo "Paint" de Baltie 3.

Fuente: (Progopedia, 2015)

Una de las mayores ventajas de Baltie respecto a otras herramientas, es la abundancia de competiciones y torneos a los que los usuarios pueden apuntarse, para desarrollar y mostrar sus habilidades. El tiempo que lleva en el mercado, y su uso extendido en colegios e institutos, sobre todo en zonas de Europa del Este (Polonia, Eslovenia y República Checa), favorece la interconexión entre competiciones y competidores. Otro aspecto a destacar de esta herramienta es que no precisa de máquinas potentes para poder ejecutarse.

Como inconveniente, se puede decir que los avances que Baltie ha tenido en cuanto a prestaciones desde sus orígenes, no ha estado a la altura del de otras herramientas de la competencia, que ofrecen más versatilidad y potencia. Su interfaz tampoco ha evolucionado a nivel de diseño, lo que le confiere cierta imagen de herramienta del pasado.

Por último, decir que su precio es elevado con respecto a lo que ofrece, y sobre todo al compararlo, como decíamos antes, con herramientas de la competencia. Tampoco posee una versión de prueba por tiempo limitado.

5. Blockly

Tabla 130
Resumen de características de la herramienta Blockly

Nombre de la herramienta	Blockly
Página web **URL de descarga**	https://developers.google.com/blockly/ https://blockly-games.appspot.com/
Fecha de creación / aparición en el mercado	Verano del año 2011 / Mayo de año 2012
Última versión en 2017 / año de esa versión	Versiones de la web, última actualización el 23 de junio del año 2016
Plataformas en las que se puede instalar	A través de la web: Chrome, Firefox, Safari, Opera, IE Android, iOS
Tipo de software	Privativo
Precio / modelo de negocio	Gratuito
Edades para las que está indicado	Edades entre los 4 y los 8 años
Aspectos educativos que trabaja	Conocimientos básicos sobre la programación
Característica más destacada / valor diferencial	La posibilidad de exportación del lenguaje de la herramienta a otros lenguajes de programación, utilizando código, además de bloques con programación predefinida

Se trata de una aplicación para introducir a niños y niñas de corta edad (de entre 4 y 8 años) a la programación, y desarrollar el sentido de la espacialidad. El mecanismo de programación que utiliza Blockly se basa en la selección de bloques de diferentes colores, cada uno de los cuáles tiene asociado un comportamiento. Los usuarios pueden colocar estos boques en un orden para desarrollar el programa que quieran hacer.

Como sistema de aprendizaje, Blockly propone una especie de tour por la herramienta, que consiste en completar una serie de retos, en forma de minijuegos didácticos de dificultad creciente. Este recorrido se muestra en la *Figura 154.*

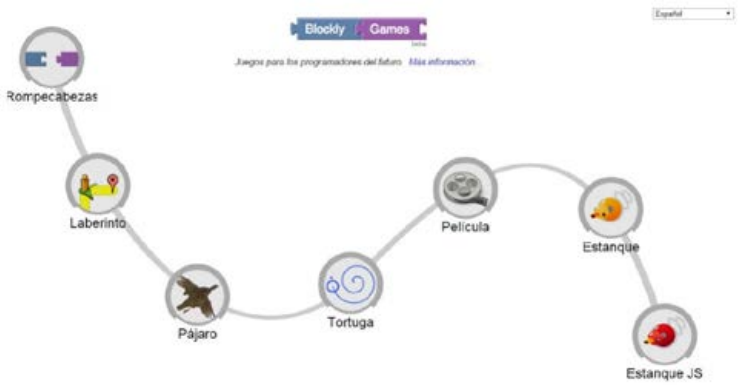


Figura 154. Minijuegos en Blockly.

Fuente: *(Google Company, 2017b)*

A continuación se van a describir cada uno de estos minijuegos:

Rompecabezas (*Figura 155*):
Este juego consiste en asociar animales con sus características. Es decir, aparece una foto, y se solicita al jugador que diga cuál es su nombre, cuántas patas tiene, etc. De esta manera se trabaja el concepto computacional de variable, o trasladándolo al paradigma de la programación orientada a objetos, el concepto de propiedad.

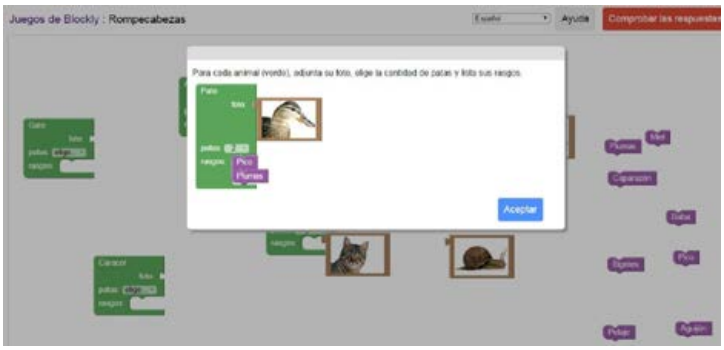


Figura 155. Minijuego "Rompecabezas" en Blockly.

Fuente: *(Google Company, 2017b)*

Laberinto (*Figura 156* y *Figura 157*):
Este juego busca trabajar los conceptos de bucles y sentencias condicionales. Se trata, con los bloques de programación adecuados, construir un programa que consiga sacar de un laberinto a un personaje. El programa da la opción de cambiar la temática del entorno del ejercicio (por ejemplo, el personaje puede ser un astronauta en el espacio, o a un panda en un bosque de bambús). A medida que se avanza, los ejercicios adquieren una complejidad más elevada.

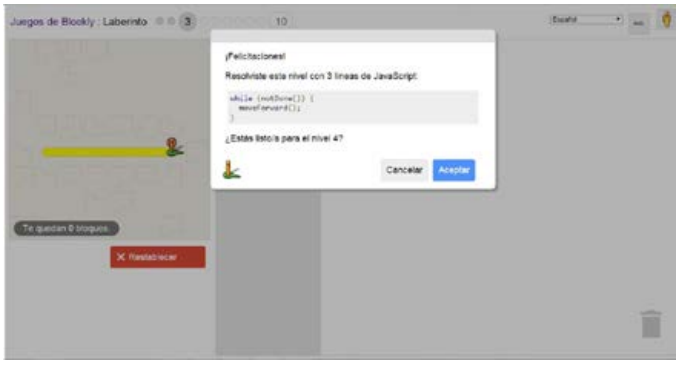


Figura 156. Minijuego "Laberinto" en Blockly.

Fuente: *(Google Company, 2017b)*

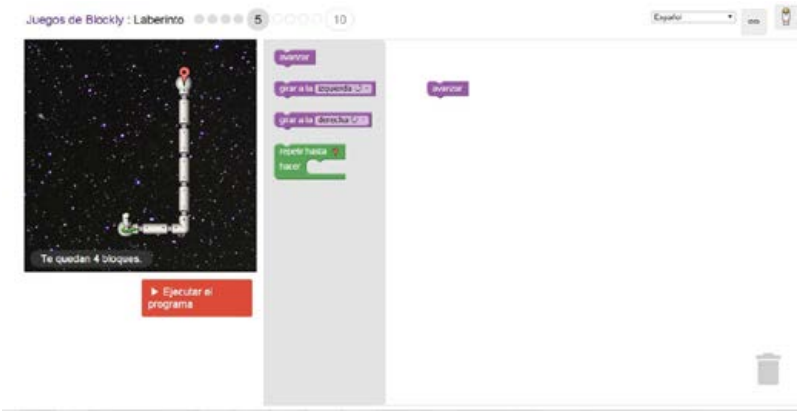


Figura 157. Minijuego "Laberinto" 5 en Blockly.

Fuente: *(Google Company, 2017b)*

Pájaro (*Figura 158*):
Ejercicio muy similar al del laberinto, pero usando esta vez los 360º (en el caso del laberinto, solo se utilizaba arriba, abajo, izquierda y derecha).

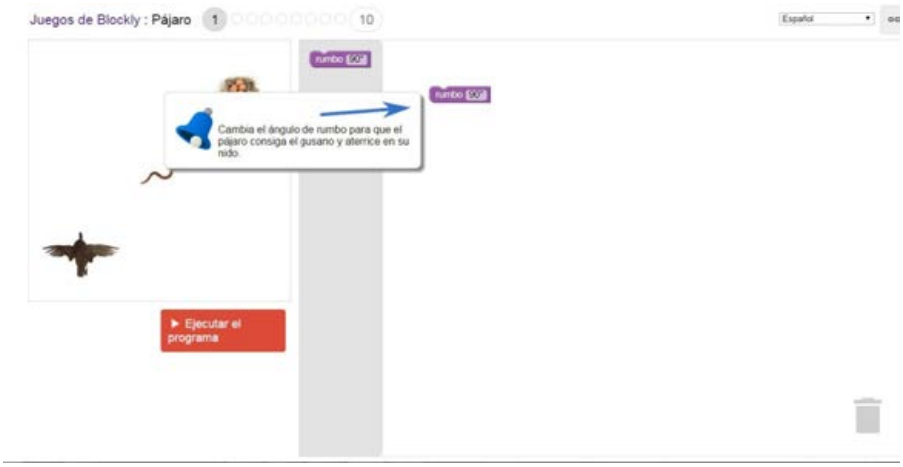


Figura 158. Minijuego "Pájaro" en Blockly.

Fuente: *(Google Company, 2017b)*

Tortuga (Figura 159):
Profundiza en el mismo apartado que el ejercicio anterior, pero se da mayor libertad al usuario a la hora de seleccionar los parámetros.

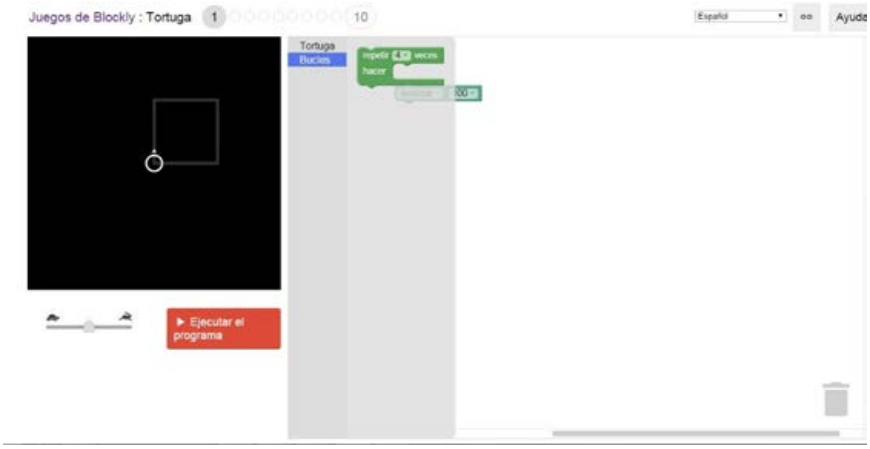


Figura 159: Minijuego “Tortuga” en Blockly.

Fuente: *(Google Company, 2017b)*

Película (*Figura 160):*
Consiste en adaptar una serie de parámetros (tamaño, colores y medidas) dentro del eje cartesiano, para intentar replicar una figura mostrada.

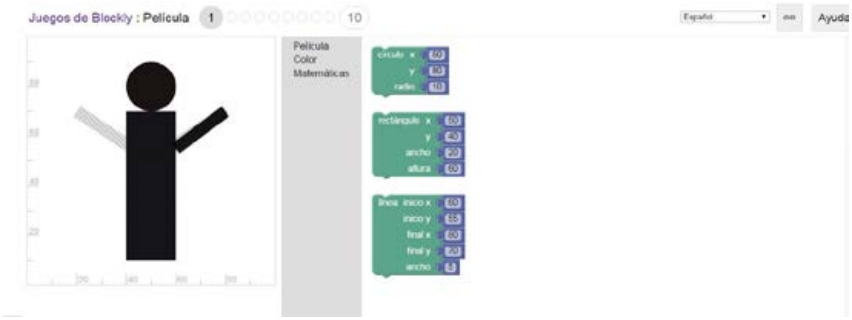


Figura 160. Minijuego “Película” en Blockly.

Fuente: *(Google Company, 2017b)*

Estanque (*Figura 161*) y *Estanque JS* (JavaScript, *Figura 162*):
Se trata de utilizar los bloques de programación, para conseguir que el avatar de un pato dispare a otro pato. Modificando ciertos parámetros, se puede cambiar la trayectoria, la fuerza del disparo e, incluso, (en niveles muy avanzados) el movimiento. El Estanque JS (JavaScript) es exactamente el mismo tipo de ejercicio, solo que para su consecución se precisa hacer el programa en código textual.

Figura 161. Minijuego "Estanque" en Blockly.

Fuente: *(Google Company, 2017b)*

Figura 162. Minijuego "Estanque JS" en Blockly.

Fuente: *(Google Company, 2017b)*

Aunque los primeros ejercicios son sencillos, y asequibles para niños y niñas de edades muy tempranas, los últimos tienen una complejidad que precisa de mayor capacidad de abstracción, y por ende, de usuarios de más edad.

En cualquier caso, es una herramienta de alto valor didáctico, y su diseño es muy sencillo, y a la vez atractivo (posee una interfaz limpia, y utiliza las paletas de color propias y características de Google).

Una de las ventajas fundamentales que tiene Blockly es la posibilidad de desarrollar programas tanto con el sistema de bloques, como a través de JavaScript, es decir, por código textual. De hecho, se puede hacer un programa por bloques, y ver el equivalente en JavaScript. De esta manera se facilita el aprendizaje de ambos métodos, y la transición de uno al otro.

Como inconveniente cabe decir que no ofrece más formación que los minijuegos mencionados, con lo que puede quedarse algo escasa en ese sentido.

6. Caleiduino

Tabla 131
Resumen de características de la herramienta Caleiduino

Nombre de la herramienta	Caleiduino
Página web	http://www.caleiduino.com/
URL de descarga	http://www.caleiduino.com/software/
Fecha de creación / aparición en el mercado	Año 2016
Última versión en 2017 / año de esa versión	Año 2016
Plataformas en las que se puede instalar	Plataforma propia, pero los drivers necesarios pueden descargarse para Windows y Mac
Tipo de software	Libre
Precio / modelo de negocio	Acceder a todo el contenido es gratuito, pero existen packs para comprar todas las piezas para su elaboración (extremadamente limitadas), además de una guía para crearlo con materiales caseros. Otras piezas deben comprarse en páginas externas, sus precios varían.
Edades para las que está indicado	No está especificado
Aspectos educativos que trabaja	Electrónica, programación estructurada, informática básica
Característica más destacada / valor diferencial	La gran capacidad de modificación que te permite y su libre distribución

Caleiduino (*Figura 163*) se trata de un juguete digital basado en un caleidoscopio tradicional. Su uso es muy similar al de Arduino, es decir, se dispone de una placa electrónica sobre la que se conectan una serie de componentes, y de esta manera se crean aplicaciones interactivas. Caleiduino se autodefine como una herramienta para el aprendizaje de la electrónica y la programación de una forma lúdica.

El aparato está formado por una placa PBC conectada a una plataforma Arduino nano 3.0, una pantalla TFT 1.8′, un acelerómetro analógico de 3 ejes GY-61, un piezoeléctrico, un interruptor y una batería de 9V. Al ser un caleidoscopio, también posee 3 espejos en forma de prisma triangular. El aparato está cubierto con una carcasa de metacrilato que cubre la parte externa y le da su forma final. La finalidad de estos componentes generar gráficos y sonidos a partir del movimiento del usuario, gracias al acelerómetro anteriormente mencionado (Maocho, 2016). En la *Figura 164* se muestran los diferentes pasos a seguir para el montaje del caleidoscopio.

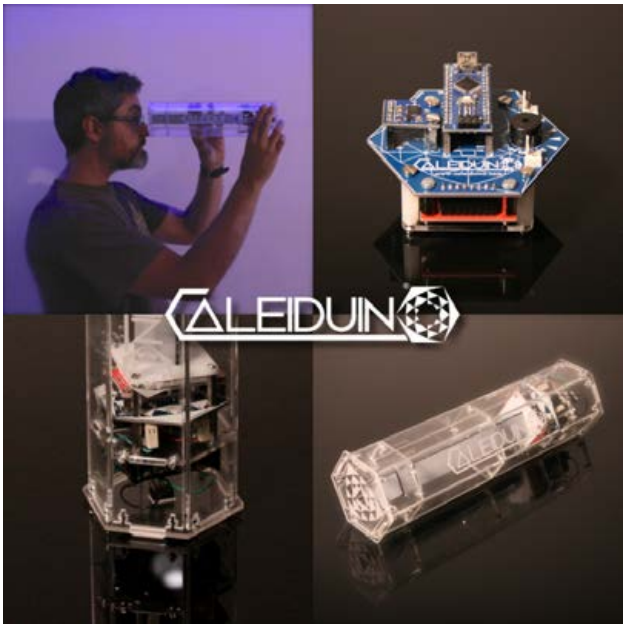


Figura 163. Logo y diferentes piezas que componen Caleiduino.

Fuente:(González, 2016)

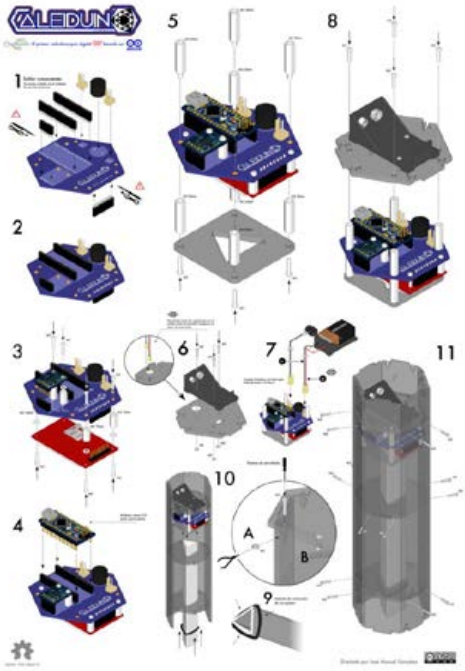


Figura 164. Piezas y montaje de Caleiduino.

Fuente:(González, 2016)

Una de las mayores virtudes de Caleiduino es que es *open source* (software libre). Desde su página web se puede acceder al código base, sobre el que se soporta el funcionamiento de Caleiduino, y modificarlo a voluntad. En la *Figura 165* se muestra la página desde la que se puede descargar el software de Caleiduino.

Figura 165. Software de Caleiduino.

Fuente:*(*González, 2016)

En la web del fabricante existen vídeos y tutoriales sobre los procesos necesarios para el montaje de Caleiduino. Sin embargo, muchas de las herramientas que se utilizan en estos tutoriales, deben comprarse en webs externas. Los precios de esas herramientas varían y requieren ciertos conocimientos sobre electrónica y montaje. Si se quiere modificar el código base de Caleduino, se debe conocer el lenguaje de programación C++. Es decir, no es una herramienta accesible para todos los usuarios.

En la *Figura 166* se muestra el código (que puede ser copiado desde la propia web) necesario para la creación de Caleiduino.

```
Este ejemplo de CALEIDUINO se basa en las librerias de Adafruit
GFX
y ST_7735, que hacen posible conectar Arduino con una pantalla TFT
de 1.8'. Gracias a Arduino y a Adafruit por desarrollar el
hardware
y el software que hacen posible el proyecto CALEIDUINO. Este
sketch
de codigo es abierto y de dominio publico. Esta a disposicion de
cualquiera que desee crear su propio CALEIDUINO.

¡¡Animate y crea tu propio caleidoscopio digital sonoro!!

(CC)Jose Manuel Gonzalez 2016

************************************************************/

// Conexiones a la pantalla TFT
// #define sclk 13 // Una esta linea si prefieres Opcion 1 (lento)
// #define mosi 11 // Una esta linea si prefieres Opcion 1 (lento)
#define cs  8
#define dc  9
#define rst 10 // Se puede conectar tambien al PIN RESET
// (no con placa CALEIDUINO)
```

Figura 166. Muestra del código base de Caleiduino.

Fuente:*(*González, 2016)

7. Cargo-Bot

Tabla 132
Resumen de características de la herramienta Cargo-Bot

Nombre de la herramienta	Cargo-Bot
Página web	https://twolivesleft.com/CargoBot/
URL de descarga	https://itunes.apple.com/us/app/cargo-bot/id519690804?ls=1&mt=8
Fecha de creación / aparición en el mercado	Año 2012
Última versión en 2017 / año de esa versión	Cargo-Bot 1.0.1, el día 8 de mayo del año 2012
Plataformas en las que se puede instalar	iOS 5.0 o superior: iPhone, iPad
Tipo de software	Privativo
Precio / modelo de negocio	Gratis
Edades para las que está indicado	Desde los 4 años
Aspectos educativos que trabaja	Resolución de problemas, espacialidad, conceptos básicos sobre la programación
Característica más destacada / valor diferencial	Es el primer juego de la App Store desarrollado enteramente usando Codea, una app para la creación rápida de juegos y simulaciones

Cargo-Bot (*Figura 167)* se trata de un pequeño juego de puzles para dispositivos móviles y *tablets* en el que los jugadores controlan los brazos de un robot, a través de diferentes instrucciones. Se trata de utilizar esas instrucciones en un orden concreto, para recrear una estructura que aparecerá en pantalla, hecha a base de cajas apiladas. Las instrucciones que el usuario programe moverán un brazo mecánico que colocará las cajas en el orden establecido.

Figura 167. Pantalla inicial de Cargo-Bot.

Fuente: (Viana, 2012)

Cuando los jugadores hayan completado la figura, ganarán puntos y “estrellas” en función del tiempo transcurrido, y del tipo de solución que hayan encontrado (la más fácil, la más eficiente, la más corta, etc.). El juego posee varios niveles de dificultad, como se muestran en la *Figura 168*, que se van desbloqueando cuando el jugador adquiere un cierto número de puntos o puzles resueltos.

Figura 168. Niveles de dificultad del juego Cargo-Bot.

Fuente: (Viana, 2012)

Uno de los puntos interesantes de este juego es la utilización de *loops* (bucles) y sentencias condicionales (por ejemplo, el uso de una orden exclusivamente si se tiene recogida una caja de un color concreto).

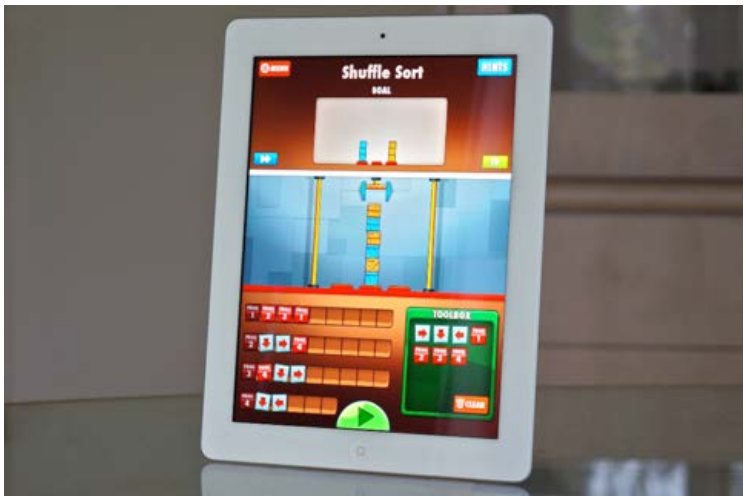


Figura 169. Un nivel en el juego Cargo-Bot.

Fuente: (Viana, 2012)

El hecho de no poder acceder al código asociado a las instrucciones que controlan el brazo, es algo que evita que sea una herramienta más completa, pero ello no quita su valor didáctico. Su cualidad más destacada es su capacidad de combinar el carácter lúdico asociado a su enfoque de juego, con su capacidad para enseñar conceptos comunes de programación de una forma simple. Además, es una herramienta gratuita

8. Code Combat

Tabla 133
Resumen de características de la herramienta Code Combat.

Nombre de la herramienta	Code Combat
Página web	https://codecombat.com/
URL de descarga	https://codecombat.com/
Fecha de creación / aparición en el mercado	Febrero de 2013
Última versión en 2017 / año de esa versión	Versión web del año 2017
Plataformas en las que se puede instalar	A través de la web
Tipo de software	Libre
Precio / modelo de negocio	Gratuito, aunque tiene versión de pago que amplía los niveles y da diferentes recompensas.
Edades para las que está indicado	No hay edades especificadas, pero hay una serie de opciones que aconsejan unas ciertas edades para niveles concretos
Aspectos educativos que trabaja	Sintaxis programación, conceptos básicos de la programación
Característica más destacada / valor diferencial	Su función multijugador, la capacidad de aprender del código directamente en varios lenguajes un sistema de aprendizaje por niveles de conocimiento

Code Combat (*Figura 170)* es una herramienta para aprender programación en forma de un videojuego RPG, con el uso de los lenguajes JavaScript, Lua, CoffeScript, Python y Clojure.

Figura 170. Logo de *Code Combat*.

Fuente: (Saines, Erickson y Winter, 2017)

En la parte izquierda de su interfaz se observa una gran ventana donde se muestra al personaje que se está controlando, su vida (parte baja central), el escenario o el nivel y la zona del mapa en la que se encuentra (parte superior central), los objetivos a cumplir (parte superior izquierda), y una pequeña barra de herramientas donde se podrá poner en pausa o en movimiento al personaje y su recorrido por el nivel, así como alejarse o acercarse para ver mejor la situación en la que se encuentra (barra colocada sobre la vida del personaje). También cuenta con un espacio para acceder al menú del juego, y la opción de registrarse si el usuario aún no lo ha hecho. En la siguiente imagen se muestra uno de los primeros niveles del juego, donde se pueden observar sus elementos principales.

Figura 171. Un nivel del juego *Code Combat*.

Fuente: (Saines et al., 2017)

A la derecha de esta ventana, encontramos el espacio de trabajo, donde se lleva a cabo el aprendizaje. En la parte superior puede verse el código que podrá modificarse para que el personaje realice diferentes acciones. En la parte inferior tenemos las ayudas para superar el nivel, y los diferentes comandos que se pueden

usar, que dependerán de la dificultad del nivel y de las “habilidades” que desbloquee el personaje.

Los niveles se desarrollan cumpliendo distintos objetivos, moviendo al personaje hasta unos puntos marcados (ayudados por una pequeña línea de trayectoria) y realizando acciones según se llega hasta el punto siguiente.

Al finalizar un nivel, dependiendo de los objetivos y sub-objetivos que se hayan cumplido, el jugador gana experiencia y gemas, que puede usar para comprar objetos, equipar mejor a su personaje, y desbloquear habilidades. El jugador es llevado al mapa de la zona donde puede seleccionar los niveles. Cada nivel se puede realizar varias veces, o se puede avanzar hacia uno nuevo. En la *Figura 172* se pueden ver algunos de los niveles del juego, cuántas horas de aprendizaje aporta al usuario, los diferentes conceptos que se enseñan en cada uno, etc.

Figura 172. Mundos y niveles en *Code Combat*.

Fuente: (Viana, 2012)

Se trata de un software totalmente gratuito, pensado para que cada usuario pueda regular su ritmo de aprendizaje. Sus principales ventajas son, por un lado, que es posible utilizar diferentes lenguajes, y por otro, que los niveles están ordenados según el concepto de programación que trabajan. Además de esto, tiene un modo multijugador (*Figura 173*) que le confiere una componente social.

Como inconveniente se podría decir que todo lo que se programa está enfocado a completar cada nivel del juego, y no hay posibilidad de crear código fuera de este contexto.

Figura 173. Modo multijugador de *Code Combat*.

Fuente: (Viana, 2012)

9. Code-a-pillar

Tabla 134
Resumen de características de la herramienta Code-a-pillar

Nombre de la herramienta	Code-a-pillar
Página web	http://www.fisher-price.com/en_US/brands/think-and-learn/index.html
URL de descarga	http://www.fisher-price.com/en_US/brands/think-and-learn/index.html
Fecha de creación / aparición en el mercado	Año 2016
Última versión en 2017 / año de esa versión	Code-a-pillar 1.2.1 para Android, del 22 de diciembre del año 2016 Code-a-pillar 1.2.2 para iOS, del 23 de diciembre del año 2016
Plataformas en las que se puede instalar	Es un juguete, no se puede instalar, pero sí que tiene un par de aplicaciones anexas para Android e iOS
Tipo de software	Privativo
Precio / modelo de negocio	El juguete vale entre 50$ - 70$ (según el lugar de compra)
Edades para las que está indicado	Edades entre 3-6 años
Aspectos educativos que trabaja	Espacialidad, resolución de problemas, estructuración
Característica más destacada / valor diferencial	Herramienta que funciona tanto de juguete como en forma de app

Code-a-pillar es un juguete indicado para niños y niñas de entre 3 y 6 años de edad. Consiste en un robot con forma de oruga, divido por secciones, cada una de las cuales contiene una serie de acciones. Según se construya la oruga, y se coloquen estas secciones, se determinará el orden de ejecución de las acciones, comenzando

por la cabeza. En la *Figura 174* se puede ver cómo es el montaje de las diferentes piezas que componen el juguete.

Figura 174. Juguete Code-a-pillar.

Fuente: (Fisher-Price, 2016)

Una vez que todas las piezas están juntas (hasta un máximo de 8), el usuario puede comenzar el movimiento de la oruga pulsando sobre un botón colocado en la cabeza. La idea es que, gracias a unas pequeñas plataformas con sensores (una como inicio y otra como final), los usuarios consigan hacer llegar al juguete hasta la plataforma final, poniendo por delante los obstáculos que deseen. Es un juguete que además de moverse puede emitir sonidos e iluminarse.

Adjunto al juguete, también existen unos videojuegos gratuitos, disponibles para diferentes plataformas móviles, en los que los jugadores tendrán que resolver una serie de puzles para que la oruga llegue hasta el final, utilizando prácticamente las mismas piezas que en el juguete real. En la *Figura 175* se muestra uno de los niveles de la aplicación de Code-a-pillar.

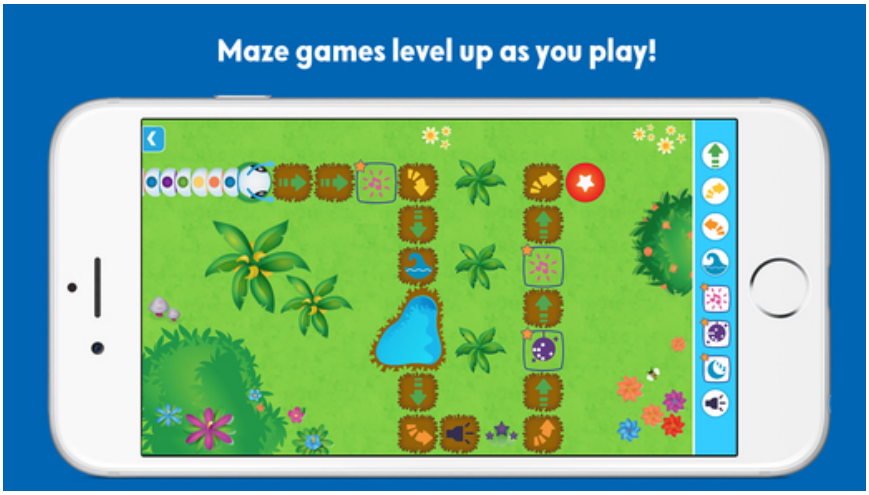


Figura 175. App Code-a-pillar.

Fuente: (Fisher-Price, 2016)

Se trata por tanto de un juguete muy vistoso, aunque no demasiado novedoso dado que existen algunas propuestas parecidas, en su mayoría orientadas a un público más adulto (por ejemplo, Lego Mindstorms, que veremos más adelante). Con su uso sólo se aprenden conceptos básicos de programación, lo cual parece lógico, teniendo en cuenta las edades para las que está recomendado. En cualquier caso, como decíamos, existen otras herramientas y juegos parecidos que sí trabajan competencias de programación más avanzadas. Para algunos usuarios, su relativamente elevado precio también puede ser un inconveniente.

10. Codeable Crafts

Tabla 135
Resumen de características de la herramienta *Codeable Crafts*

Nombre de la herramienta	Codeable Crafts
Página web	https://www.codeablecrafts.com/
URL de descarga	App Store: https://itunes.apple.com/us/app/codeable-crafts/id1050901522?mt=8 Play Store: https://play.google.com/store/apps/details?id=jp.co.benesse.ccs
Fecha de creación / aparición en el mercado	2015
Última versión en 2017 / año de esa versión	App Store: 26 de febrero de 2017 Play Store: 8 de julio de 2015
Plataformas en las que se puede instalar	Android e iOS
Tipo de software	Privativo
Precio / modelo de negocio	Gratuito
Edades para las que está indicado	Edades entre los 6-8 años
Aspectos educativos que trabaja	Conceptos básicos de la programación, estructuración
Característica más destacada / valor diferencial	Personalización y modelo de negocio gratuito

Codeable Crafts es una aplicación diseñada para la enseñanza de conceptos básicos de la programación, a niños y niñas entre 6 y 8 años. La aplicación da la oportunidad a los usuarios de crear y personalizar una serie de personajes a partir de unas bases. En la *Figura 176* se muestra la pantalla principal de la aplicación, desde la que se puede acceder al resto de la herramienta.

Figura 176. Una escena en *Codeable Crafts*.

Fuente: (Benesse Corporation, 2017)

En la *Figura 177* se muestran una serie de personajes creados con la herramienta que se pueden editar, modificar y borrar.

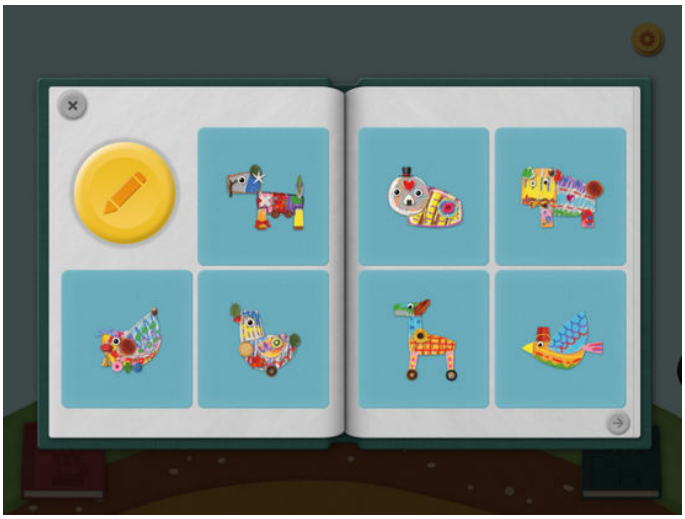


Figura 177: Personajes ya creados en *Codeable Craft*.

Fuente: (Benesse Corporation, 2017)

Una vez creados los personajes, se ofrecen una serie de elementos para componer una escena, a la que se añadirán los personajes. El comportamiento de estos personajes se programa a través de bloques, uniéndolos entre sí. Los bloques están separados por categorías, dependiendo del tipo al que pertenezcan (de conexión, de acción, etc.), y algunos bloques permiten ser modificados para llevar a cabo

ciertas acciones un número determinado de veces antes de finalizar. La herramienta también permite compartir fotos y personajes entre otros usuarios.

Figura 178. Creación de una escena en *Codeable Crafts*.

Fuente: (Benesse Corporation, 2017)

Lo más destacado de *Codeable Crafts* es todo lo relativo a su diseño, y al ámbito artístico de personajes y escenarios. Con respecto a lo que ofrece para aprender a programar, guarda cierta similitud con otras herramientas, como Scratch, aunque posiblemente tenga menos potencial que estas.

11. Codin Game

Tabla 136
Resumen de características de la herramienta *Codin Game*

Nombre de la herramienta	Codin Game
Página web	https://www.codingame.com/start
URL de descarga	https://www.codingame.com/training
Fecha de creación / aparición en el mercado	4 de abril del año 2012
Última versión en 2017 / año de esa versión	Versión de la web
Plataformas en las que se puede instalar	Funciona a través de navegador, cualquiera puede acceder
Tipo de software	Privativo
Precio / modelo de negocio	Gratuito
Edades para las que está indicado	Desde los 16 años en adelante
Aspectos educativos que trabaja	Conceptos básicos sobre programación y programación orientada a objetos, estructuración y resolución de problemas
Característica más destacada / valor diferencial	Permite la utilización de más de 20 lenguajes de programación diferentes

Codin Game (*Figura 179)* se trata de una herramienta web diseñada para crear videojuegos. Cuenta con una enorme comunidad de usuarios, y está pensado para jugadores ya habituados a la programación.

El programa requerirá a los usuarios que se registren, para poder acceder tanto a los ejercicios propuestos, como a la ayuda de la comunidad. Asimismo, el registro da acceso al ranking multijugador, donde los distintos usuarios podrán competir contra jugadores de todo el mundo, y así aumentar su puntuación.

Una vez registrados, los usuarios podrán acceder a diferentes ejercicios, categorizados por dificultades. Al seleccionar uno de ellos, se accede a la parte esencial de la herramienta, donde se codifica el programa que resuelve el ejercicio propuesto. La interfaz que se verá en ese momento está dividida en 4 apartados:

En el área superior izquierda se encuentra una ventana de previsualización, en la que se podrá ver el resultado del juego. También se podrá acceder a ciertas opciones, como los ajustes gráficos, la velocidad a la que se desarrolla la escena, ponerla en pausa, avanzar o ir al paso anterior, incluso compartir el juego online. Bajo el área de previsualización, se podrán ver los objetivos del juego y sus reglas.

En la parte superior derecha se encuentra el área de programación, donde se escribe el código. También se podrá elegir en qué lenguaje se quiere programar. Por último, el área inferior izquierda contiene un espacio de depuración de errores, donde se mostrará información de lo que sucede a nivel interno en la ejecución del código, pudiéndose ver los *bugs* (errores) y los eventos concretos que suceden.

Figura 179. Logo y nivel de juego en *Codin Game*.

Fuente: (Saines et al., 2017)

En la parte inferior derecha observaremos las diferentes tareas u objetivos a cumplir. También se encontrará el botón para resolver el ejercicio, y comprobar si es funcional en todos los casos. En la *Figura 180* se puede ver un juego creado en la herramienta, su previsualización (a la izquierda), el código utilizado (a la derecha) y algunos de los comandos (parte inferior)

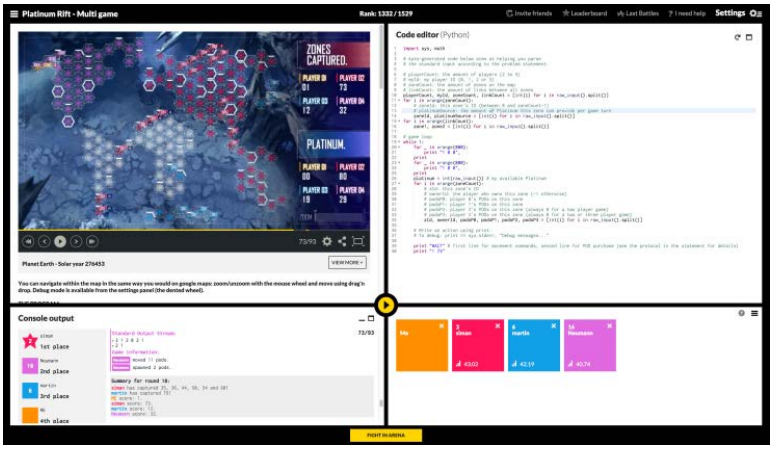


Figura 180. Juego "*Platinum Rift*" de *Codin Game*.

Fuente: (Saines et al., 2017)

Las principales ventajas de *Codin Game* es que, además de ser gratuito, es multiplataforma (se accede a la herramienta desde un navegador web), y puede

utilizarse con más de 20 lenguajes de programación diferentes. Además, tiene una fuerte componente motivacional, dado que convierte aprender a programar en un juego y una competición.

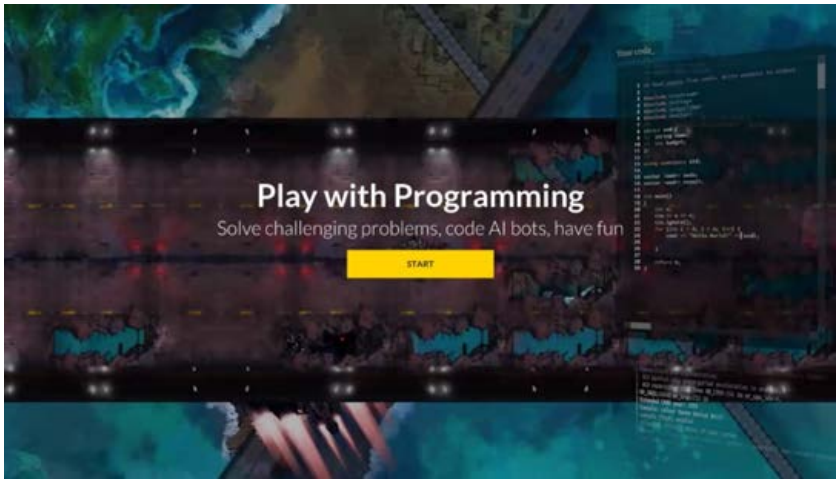


Figura 181. Pantalla inicial de la web.

Fuente: (Saines et al., 2017)

Codin Game cuenta con una comunidad muy activa, y existen multitud de espacios en Internet donde acceder a información relacionada con esta herramienta.

Como inconveniente, podemos decir que alguien que nunca se haya enfrentado a un código de programación probablemente verá esta herramienta como inasumible. Como decíamos, está enfocada principalmente para un público que ya tiene cierta experiencia en este aspecto.

12. Cody&Roby

Tabla 137
Resumen de características de la herramienta Cody&Roby

Nombre de la herramienta	Cody&Roby
Página web	http://codeweek.it/cody-roby-en/
URL de descarga	http://codeweek.it/cody-roby-en/diy-starter-kit/
Fecha de creación / aparición en el mercado	Año 2014
Última versión en 2017 / año de esa versión	Se trata de un juego de mesa, no se ha modificado desde su lanzamiento
Plataformas en las que se puede instalar	Funciona a través de navegador, cualquiera puede acceder
Tipo de software	Libre
Precio / modelo de negocio	Gratuito
Edades para las que está indicado	No especificada
Aspectos educativos que trabaja	Estructuración y resolución de problemas
Característica más destacada / valor diferencial	Se trata de un juego de mesa, no requiere de elementos externos, electricidad, etc.

Se trata de un juego de mesa que se puede descargar desde la página web de sus creadores. El archivo que se descarga contiene diferentes cartas, personajes y un tablero, todos imprimibles a papel. El juego está pensado esencialmente para ser jugado por dos jugadores.

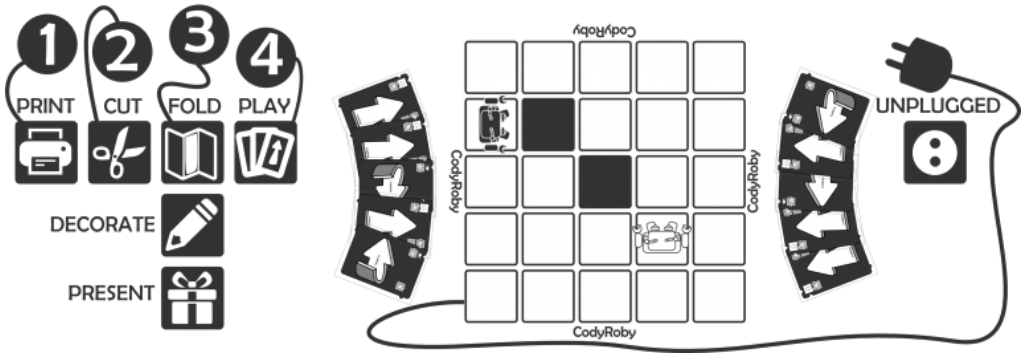
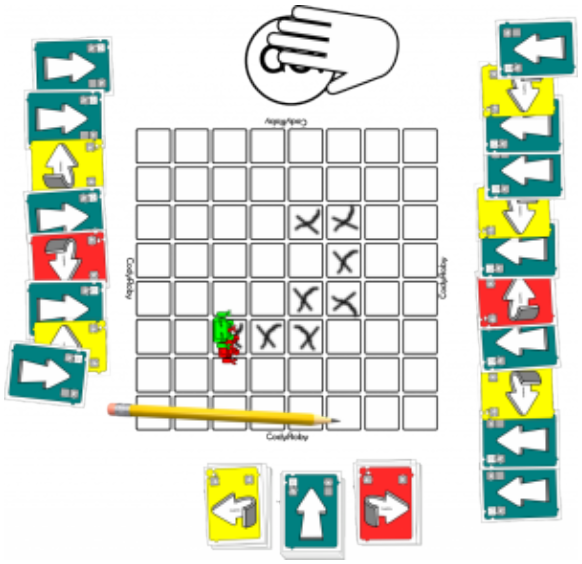


Figura 182. Pasos para montar y comenzar el juego *Cody&Roby*.

Fuente: (Bogliolo, 2014b)

Las cartas base del juego son 3: Avanzar, girar a derecha y girar izquierda. Con estas reglas básicas, la propia web presenta varios juegos a los usuarios:

- *Carrera* (*Figura 183*) es un juego en el que las cartas de dirección anteriormente mencionadas, sirven para indicar la trayectoria que seguirán los personajes controlados por cada jugador. Los jugadores deben hallar la forma más rápida de llegar hasta una meta, establecida por consenso por ambos jugadores. Una vez que uno de los jugadores crea haber hallado la respuesta, deberá pulsar con el dedo (de forma simbólica) en el otro lado del tablero, donde estará colocado el botón GO! (comenzar). Una vez comprobada que la secuencia es correcta, se determinará un ganador. Si la respuesta es incorrecta, se continuará el juego hasta que uno de los jugadores encuentre la solución. También se puede ganar al adversario, aunque este haya encontrado una solución, si la propuesta propia usa menos cartas para llegar hasta el final.



- *Figura 183*. Juego “Carrera” en *Cody&Roby*.
- Fuente: (Bogliolo, 2014b)

- *Turista*, es en realidad el mismo juego que “Carrera”, pero está más orientado al ámbito infantil, los personajes se cambian por los jugadores y el tablero pasa a ser el suelo. El camino a seguir (tablero) pasa a ser (idealmente) piezas de gomaespuma que se pueden encajar entre sí. La función de las cartas es la misma.

- *Duelo*, en el que uno de los jugadores trata de llegar hasta el final del tablero, y el otro a intentar cazar al jugador contrario. El juego se desarrolla por turnos.

Cada uno de estos juegos admite variantes, en las que dentro del tablero se incluyen muros que no se pueden atravesar, trampas, etc., que añaden un punto de dificultad en la búsqueda del camino hacia la meta.

Esta herramienta puede servir para hacer entender a niños y niñas de temprana edad, conceptos sencillos sobre estructuración y resolución de problema, y sobre pensamiento algorítmico. También es un juego apto para adultos. En cualquier caso, los conceptos de programación que trabaja son escaso.

Una versión comercial más desarrollada de este juego, con distintos tableros prediseñados, y mecánicas muy similares, es *RoboRally* [50].

13. Construct 2

Tabla 138
Resumen de características de la herramienta Construct 2

Nombre de la herramienta	Construct 2
Página web	https://www.scirra.com/construct2
URL de descarga	https://www.scirra.com/construct2/releases/r239
Fecha de creación / aparición en el mercado	4 de febrero del año 2011
Última versión en 2017 / año de esa versión	25 de octubre del año 2016
Plataformas en las que se puede instalar	Windows
Tipo de software	Libre / Privativo
Precio / modelo de negocio	Hay una versión gratuita, una personal (75€) y una licencia para negocio (329€)
Edades para las que está indicado	Mayores de 12 años
Aspectos educativos que trabaja	Competencias digitales, conceptos básicos de programación, programación orientada a objetos
Característica más destacada / valor diferencial	Creación de videojuegos mediante eventos y de manera muy visual

[50] https://boardgamegeek.com/boardgame/18/roborally

Construct 2 *(Figura 184)* es un motor de videojuegos 2D basado en HTML5, desarrollado por Scirra y orientado para estudiantes sin experiencia en programación.

Figura 184. Ventana inicial de Construct 2.

Fuente: (Scirra, 2017)

Aunque podemos considerarla como una herramienta de programación por bloques, también permite utilizar los lenguajes de HTML5, C++ y JavaScript. Cuando se finaliza un proyecto y se exporta, se tiene acceso al código fuente, programado en alguno de estos lenguajes según el tipo de exportación que se haya elegido. Ese código se podrá modificar y recompilar en un SDK[51] apropiado.

La interfaz de Construct 2 se divide en ventanas con distintas opciones, siendo *Layout* y Eventos las principales:

- *Layout*, es el espacio donde se agregan y modifican los elementos gráficos del juego (personajes, escenarios, etc.), y se manipulan las propiedades de los distintos objetos y eventos incluidos en el juego. En la *Figura 186* se muestran algunos de los objetos a incluir en el *Layout*, (cajas de texto, partículas, botones, controles de juego, etc.), cada uno con un comportamiento asociado.
- *Eventos*, permite capturar sucesos y responder a ellos con una acción ("cuando ocurra _, ejecuta _"). Estos eventos se combinarán con los objetos dispuestos en el *Layout*, (por ejemplo, "cuando ocurra que se ha pulsado el botón 1, el objeto disparo aparece en pantalla"). En la *Figura 185* se muestra un proyecto en desarrollo con Construct 2, donde podemos ver una

51 SDK – Software Development Kit (Kit de desarrollo de software)

previsualización del juego (parte central), los objetos colocados en la escena (parte inferior derecha) y algunos eventos.

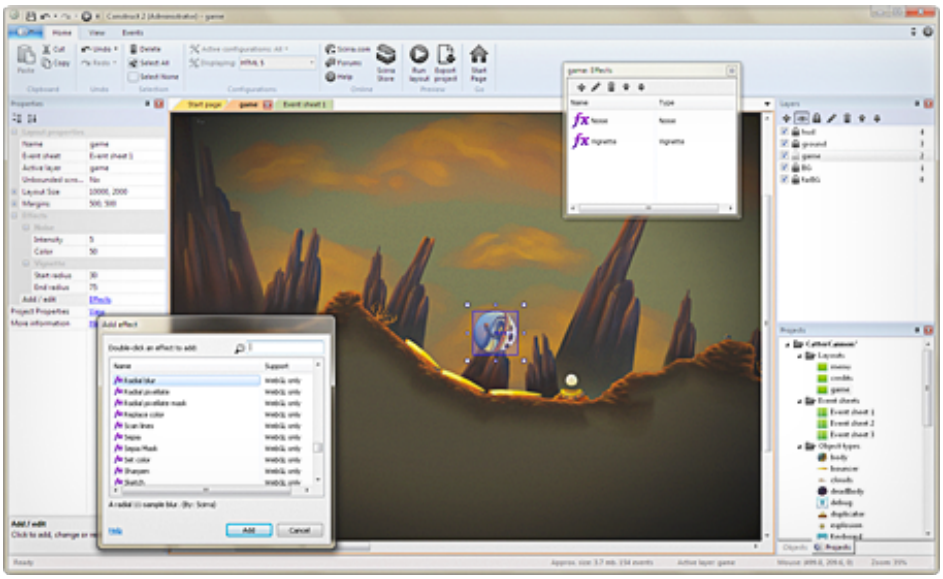


Figura 185. Proyecto en desarrollo en Construct 2.

Fuente: (Scirra, 2017)

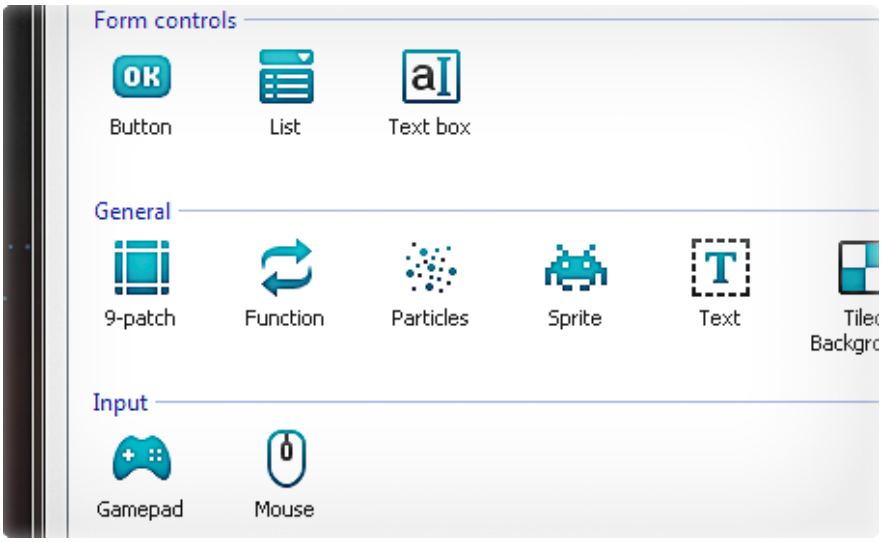


Figura 186. Opciones de desarrollo en Construct 2.

Fuente: (Scirra, 2017)

Por último, mencionar que tiene una comunidad de usuarios muy activa, y dispone una gran cantidad de documentación y tutoriales.

14. Daisy the Dinosaur

Tabla 139
Resumen de características de la herramienta *Daisy the Dinosaur*

Nombre de la herramienta	Daisy the Dinosaur
Página web	http://daisythedinosaur.com/
URL de descarga	https://itunes.apple.com/us/app/daisy-the-dinosaur/id490514278
Fecha de creación / aparición en el mercado	2012
Última versión en 2017 / año de esa versión	2 de noviembre del año 2016
Plataformas en las que se puede instalar	iOS 8.0 o superior
Tipo de software	Privativo
Precio / modelo de negocio	Gratuito
Edades para las que está indicado	Edades entre los 6-8 años
Aspectos educativos que trabaja	Matemáticas, estructuración, secuencias, resolución de problemas
Característica más destacada / valor diferencial	Acercamiento de la programación a los niños de forma simple y valorando la eficiencia

Daisy the Dinosaur (*Figura 187)* es una aplicación gratuita, cuya finalidad es la enseñanza de conceptos básicos de programación a niños de edades entre los 6 y 7 años.

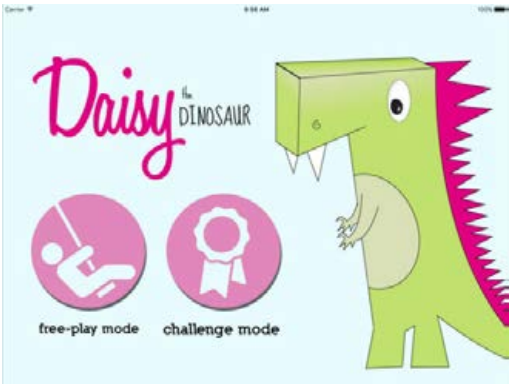


Figura 187. Pantalla inicial de *Daisy the Dinosaur*.

Fuente: (Hopscotch Technologies, 2016a)

Al iniciar la aplicación, esta ofrece a los usuarios dos modos: *Challenge Mode* (Modo Reto) y *Free Play Mode* (Modo de Juego Libre). El primero se desarrolla en un entorno más controlado, y el segundo permite al jugador experimentar, después

de seguir un pequeño tutorial en el que aprende los rudimentos básicos de la herramienta.

El juego ofrece varios puzles para resolver. Cabe destacar que la herramienta no cuenta con ningún método para guardar el desarrollo, con lo que los puzles se deberán resolver en el momento.

La interfaz del juego está pensada para un público infantil, y cuenta con símbolos y texto. En la parte superior izquierda de la pantalla se encuentra la sección *commands* (comandos), donde se agrupan los bloques que representan las posibles acciones con las que se construirá el programa que resolverá el puzle. En función del reto, se deberá elaborar un código, a base de arrastrar y soltar en la zona central denominada *program* (programa), los bloques adecuados en un orden concreto. De esta manera conseguiremos controlar a Daisy, el dinosaurio protagonista del juego, y lo haremos avanzar por el escenario. En la parte inferior se encuentra el espacio denominado *stage* (escenario) donde se verá el resultado del código programado. Entre los comandos a elegir podremos encontrar acciones como avanzar, girar, saltar, así como bucles y sentencias condicionales. Cuando arrastramos un comando a la zona del programa, lo podremos modificar y adaptar a través de una ventana emergente. Una vez completada la secuencia de comandos, se pulsará el botón *Play* (jugar) para ver los resultados.

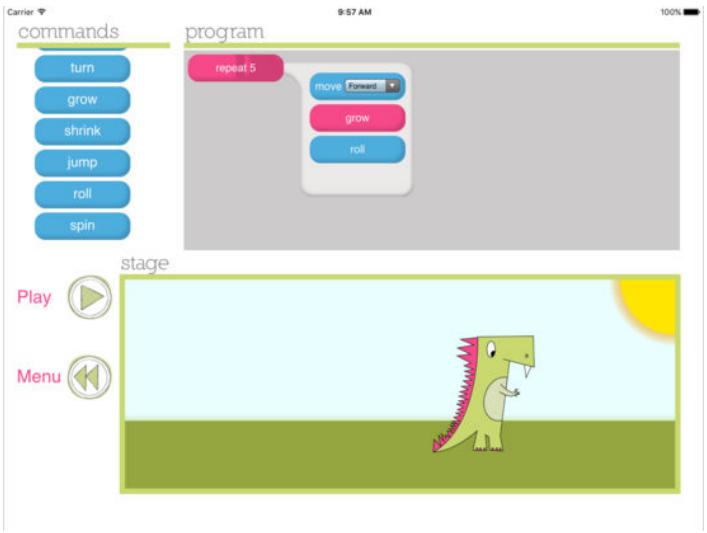


Figura 188. Un nivel en Daisy the Dinosaur.

Fuente: (Hopscotch Technologies, 2016a)

Por último, decir que aunque los comandos están organizados por un sistema de colores, es probable que algunos jugadores, sobre todo los más pequeños, tengan problemas para identificar todas las posibilidades que ofrece.

15. Desktop Dungeons

Tabla 140
Resumen de características de la herramienta *Desktop Dungeons*

Nombre de la herramienta	Desktop Dungeons
Página web	http://www.desktopdungeons.net/
URL de descarga	Hay varias plataformas: Steam: http://store.steampowered.com/app/226620 Android: https://play.google.com/store/apps/details?id=com.QCFDesign.DesktopDungeonsMobile Itunes: https://itunes.apple.com/us/app/desktop-dungeons/id979917622
Fecha de creación / aparición en el mercado	7 de noviembre del año 2013
Última versión en 2017 / año de esa versión	Desktop Dungeons de enero del año 2016
Plataformas en las que se puede instalar	Windows, MAC OS, Ubuntu, Android e iOS
Tipo de software	Privativo
Precio / modelo de negocio	Tiene precios variados según la versión y la plataforma: Edición normal: 10-15€ Edición especial: 23€
Edades para las que está indicado	Mayores de 12 años
Aspectos educativos que trabaja	Estrategia, planificación, estructuración y resolución de problemas
Característica más destacada / valor diferencial	Partidas completas rápidas y una estética que rememora a los RPG antiguos. Uso del lenguaje C#

Desktop Dungeons (*Figura 189*) es un RPG [52] en 2D de combate por turnos, con un sistema de partidas rápidas y cortas, donde el jugador debe acabar con todos los enemigos de una mazmorra creada de forma procedural. Este juego utiliza el lenguaje de programación C# y está creado con el motor Unity.

El juego comienza con un menú principal donde el jugador puede escoger el personaje (clase y raza) y tipo de partida que quiere jugar.

Dentro de la partida, el personaje se mueve y ataca mediante clics del ratón, o con las teclas “WASD”. También dispone de un menú *in-game* a la derecha con las características del personaje, y la posibilidad de utilizar hechizos que el jugador vaya obteniendo mientras avanza por el escenario de juego. En la *Figura 190* puede verse una de las escenas del juego, el Reino, en el que los jugadores pueden comprar objetos para sus personajes y pueden ver diferentes desafíos diarios con los que podrán ganar oro, la moneda del juego.

[52] RPG, *Role Play Game*, Juego de Rol

Figura 189. Logotipo de *Desktop Dungeons*.

Fuente: (QFC Design, 2016)

Figura 190. Vista del reino en Desktop Dungeons.

Fuente: (QFC Design, 2016)

Dentro de la partida, además de los anteriormente mencionados hechizos, se pueden encontrar objetos, bonificadores y avatares.

El diseño del juego es muy simple y directo, e invita a ser jugado varias veces, debido a su diseño procedural (es el sistema del juego el que genera un escenario cada vez, con lo que cada partida supone un nuevo desafío).

En la *Figura 191* se podrá apreciar una partida en *Desktop Dungeons*, con la ventana de juego en la parte central, los objetos y características del personaje (arriba a la derecha) y los objetos que se pueden comprar durante el nivel (abajo a la derecha).

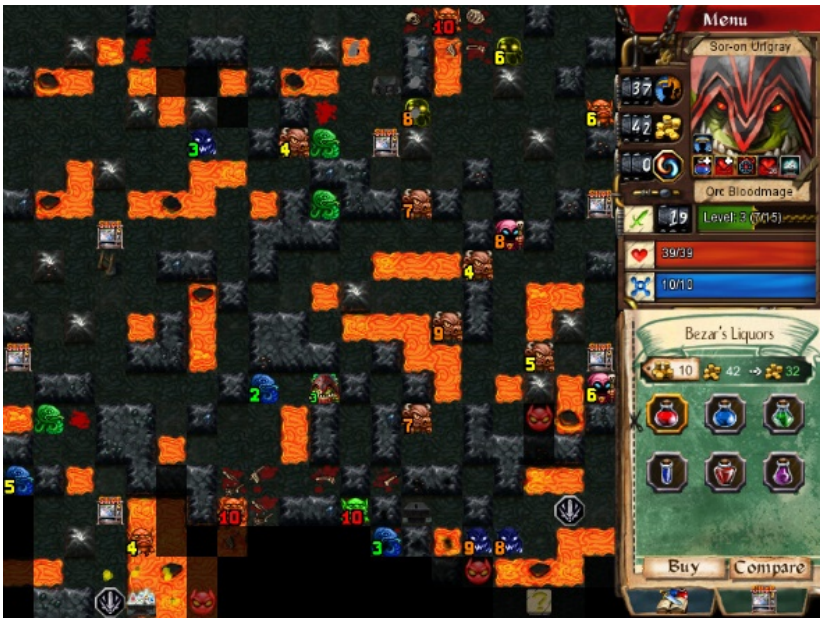


Figura 191. Una partida en *Desktop Dungeons*.

Fuente: (QFC Design, 2016)

Aunque sigue existiendo una versión *Alpha* gratuita en la web del fabricante, el juego tiene una vocación comercial. Cabe decir también que cuenta con una amplia comunidad de jugadores, foros y espacios para el soporte técnico y la resolución de dudas.

No es una aplicación sencilla de manejar, y se tarda cierto tiempo en aprender cómo funciona. Tampoco se tiene acceso al código generado, y la programación que en él se realiza no se puede trasladar a otros ámbitos, por lo que tiene más valor como juego que como herramienta de aprendizaje de la programación.

16. E-Slate

Tabla 141
Resumen de características de la herramienta E-Slate

Nombre de la herramienta	E-Slate
Página web	http://e-slate.cti.gr/
URL de descarga	http://e-slate.cti.gr/download/EN/setup.exe
Fecha de creación / aparición en el mercado	Año 1993
Última versión en 2017 / año de esa versión	Versión del año 2000
Plataformas en las que se puede instalar	Windows
Tipo de software	Libre
Precio / modelo de negocio	Gratuito
Edades para las que está indicado	Depende de la aplicación dentro de la herramienta
Aspectos educativos que trabaja	Conceptos básicos sobre la programación, matemáticas, geografía, historia
Característica más destacada / valor diferencial	Requiere muy poco para funcionar y es capaz de realizar simulaciones

E-Slate es un entorno de aprendizaje exploratorio. Esta herramienta ofrece un entorno para la creación de software dinámico, a través de diferentes opciones de programación.

Figura 192. Buscador en un mapa desde E-Slate.

Fuente: (Computer Technology Institue, 2000)

Los proyectos creados pueden ser compartidos con otros usuarios en la red. Igualmente se puede descargar el proyecto de otros, acceder a su código y modificarlo. El programa inicialmente cuenta con varias aplicaciones:

- La primera aplicación emula la física de una pelota. Dicha pelota se puede lanzar con un muelle, hacer que recorra una distancia, o subir una cuesta, etc. Todo esto se consigue modificando el ángulo de lanzamiento, la masa, la aceleración o la velocidad de la pelota, pudiendo verse cuál sería su trayectoria con relación a las características dadas.
- En la segunda aplicación observamos un entorno espacial en el que se puede emular la rotación de la luna alrededor de la tierra, pudiendo modificarse para esta emulación todos varios parámetros (velocidad, etc.).
- La tercera es una aplicación de geografía donde se puede observar de forma interactiva la superficie y la población de los diferentes países del mundo, además de marcar sus coordenadas, calcular distancias, etc.

Esta herramienta es útil en la enseñanza de diversas materias (por ejemplo, la tercera aplicación se puede utilizar en una clase de geografía). En el ámbito de la programación, puede servir para ayudar a comprender ciertos conceptos, como el de variable, pero más allá de esto es algo limitada. Además, la última versión es del año 2000, con lo que existen alternativas más modernas y completas.

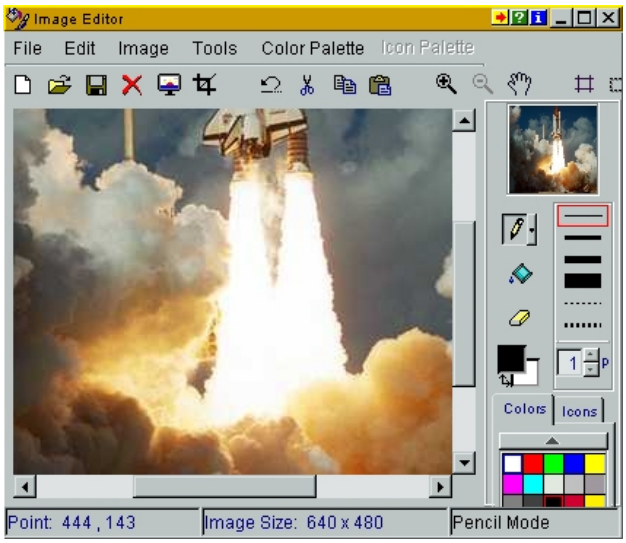


Figura 193. Editor de imágenes en E-Slate.

Fuente: (Computer Technology Institue, 2000)

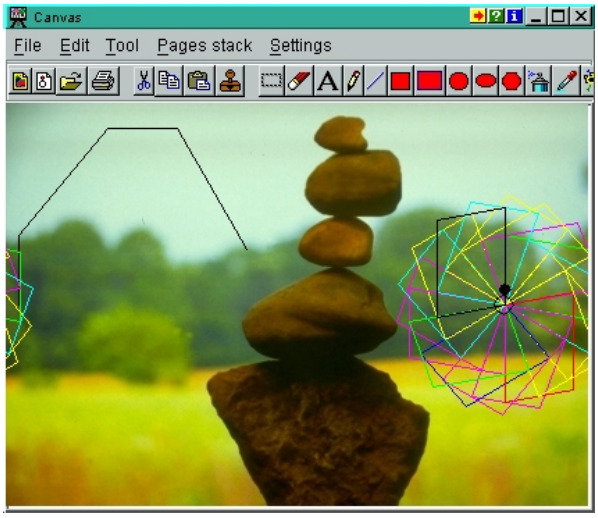


Figura 194. *Canvas* en la herramienta E-Slate.

Fuente: (Computer Technology Institue, 2000)

17. Guido van Robot

Tabla 142
Resumen de características de la herramienta Guido van Robot

Nombre de la herramienta	Guido van Robot
Página web	https://sourceforge.net/projects/gvr/files/
URL de descarga	http://gvr.sourceforge.net/download/
Fecha de creación / aparición en el mercado	Año 2004
Última versión en 2017 / año de esa versión	Versión de GvR del año 2007
Plataformas en las que se puede instalar	Windows XP o superior, Mac OS 10.3 o superior
Tipo de software	Privativo
Precio / modelo de negocio	Gratuito
Edades para las que está indicado	Mayores de 12 años
Aspectos educativos que trabaja	Conceptos básicos de la programación
Característica más destacada / valor diferencial	Es una herramienta que está pensada para ser enseñada en un entorno educativo y con la asistencia de un profesor

Guido van Robot es una herramienta para enseñar programación, que usa el lenguaje de Python, y guarda cierta similitud con Karel. Fue creada con las bases del compilador de pyKarel, y utiliza la misma tecnología.

La herramienta consta de dos cuadros de trabajo: el de la izquierda, más ancho, que controla la interfaz visual y el área de compilación, y el derecho, más estrecho, destinado a la escritura y edición del código.

Cuando se inicia la aplicación, en la parte izquierda, se puede observar un tablero compuesto por puntos que se podrán unir a través de líneas. En la *Figura 195* se muestra un ejercicio realizado con Guido van Robot, en el que se propone al usuario un laberinto, creado con paredes intransitables (en rojo), del que el personaje (triángulo), a través de las instrucciones programadas (derecha) deberá conseguir salir .

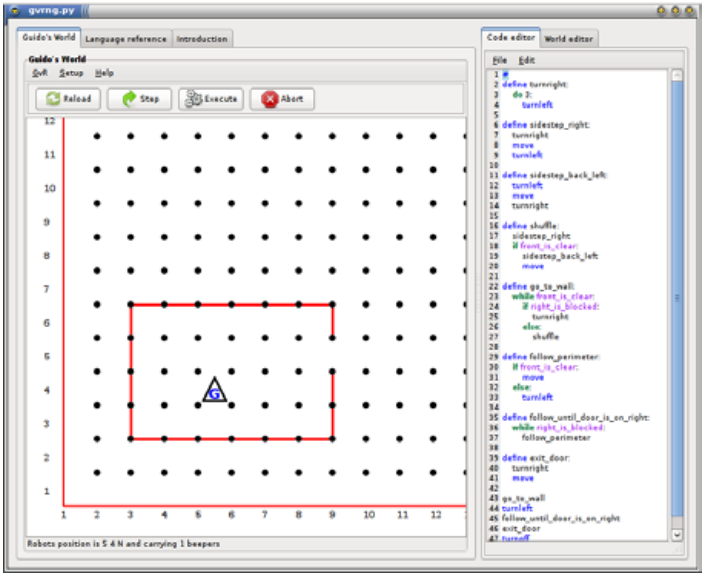


Figura 195. Inicio de la herramienta Guido van Robot.

Fuente: (Elkner, 2004)

El usuario podrá generar sus propios laberintos, o hacer que el sistema los cree. También puede acceder a un apartado denominado *Lessons* (Lecciones) en el que se proponen 18 ejercicios para que, de forma tutorada, pueda desarrollar sus capacidades dentro del programa. Además del enunciado, cada ejercicio cuenta con una explicación sobre cómo hacerlo. También cabe mencionar la existencia de una sección llamada *Lenguage preference* (preferencias de lenguaje) donde se podrá encontrar ayuda sobre el lenguaje de programación.

En la *Figura 196* se muestra otro ejemplo de ejercicio, donde el usuario programa el movimiento del personaje, para que en su recorrido pinte una serie de dibujos.

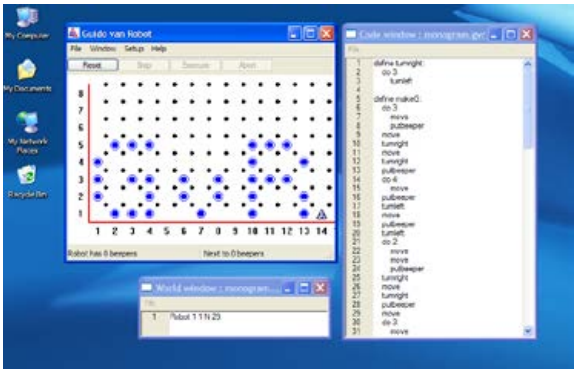


Figura 196. Uno de los ejercicios de Guido van Robot.

Fuente: (Computer Technology Institue, 2000)

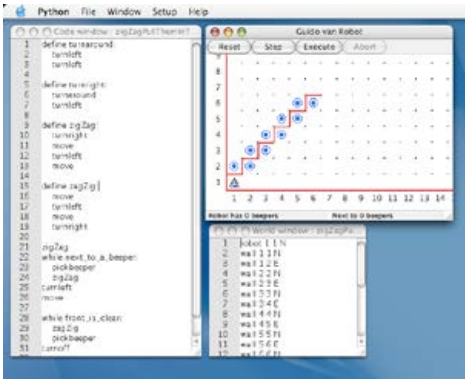


Figura 197. Un ejercicio de Guido van Robot en su versión para MAC OS.

Fuente: (Computer Technology Institue, 2000)

Para hacernos una idea del valor didáctico de esta herramienta, a continuación, se expone una pequeña sinopsis de 2 lecciones, dentro de las 18 mencionadas:

LESSON 5: "*Para mantener los costes de fabricación bajos, la fábrica sólo construyó para Guido engranajes de avanzar y girar a la izquierda. He leído en el manual de instrucciones que Guido tiene la capacidad de aprender a hacer otras cosas. Por ejemplo, si Guido gira a la izquierda tres veces, se girará a la derecha. Pero como programador de robots necesitamos decirle a Guido cómo hacer esto*." En este ejemplo se trabaja el pensamiento algorítmico, y la resolución ordenada de problemas a partir de los recursos de los que se dispone.

LESSON 6: "Ya conoces la sentencia *if*. Se utiliza para tomar una decisión. A veces hay una decisión más complicada que hacer. A Guido le gusta el pastel de manzana, pero su mamá no siempre se lo hace. Ella le hace galletas todo el tiempo, sin embargo, él quiere hacer una declaración como esta: "Mamá, me gustaría comer un poco de pastel de manzana, pero si no hay, entonces me gustaría una galleta." Este ejemplo pretende trabajar la sentencia de programación *if / else.*

18. Hackety Hack

Tabla 143
Resumen de características de la herramienta Hackety Hack

Nombre de la herramienta	Hackety Hack
Página web	La página web oficial está caída: http://www.hackety.com/
URL de descarga	La página web oficial está caída: http://www.hackety.com/downloads/latest/windows
Fecha de creación / aparición en el mercado	25 de diciembre del año 2010
Última versión en 2017 / año de esa versión	No especificada, la página web está caída y los enlaces existentes está en páginas externas
Plataformas en las que se puede instalar	Windows, MAC OS, Linux
Tipo de software	Privativo
Precio / modelo de negocio	Gratuito
Edades para las que está indicado	Mayores de 13 años
Aspectos educativos que trabaja	Conceptos básicos sobre la programación, programación orientada a objetos, matemáticas, resolución de problemas, estructuración
Característica más destacada / valor diferencial	Lecciones de corta duración muy tutoradas a través de la herramienta

Hackety Hack (*Figura 198*) es una herramienta diseñada para la enseñanza de la programación a través del lenguaje Ruby.

Figura 198. Pantalla inicial Hackety Hack.

Fuente: (Gillete, 2010)

Esta herramienta, a diferencia con otras anteriormente analizadas, da rienda suelta al usuario, y le deja investigar y realizar sus propias creaciones. Por una parte esto

supone una ventaja y le confiere valor, pero por otra, hace que se requiera de mayor nivel de conocimiento de su manejo.

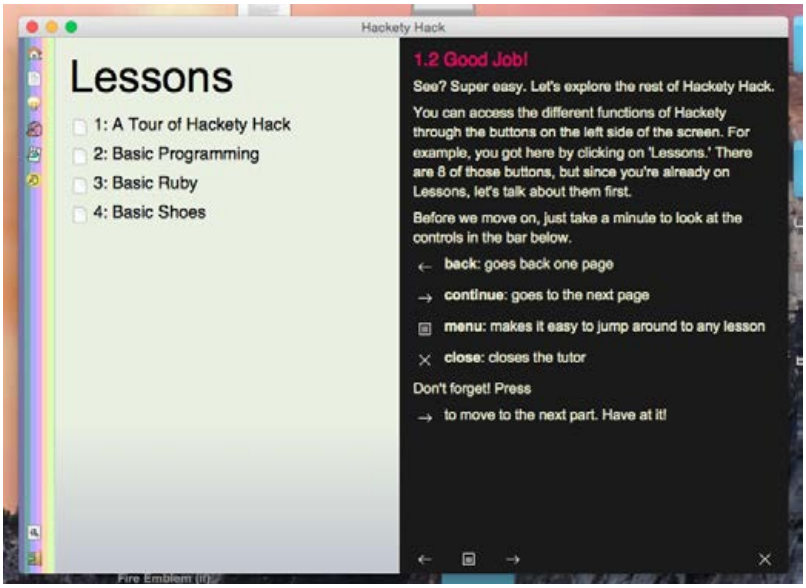


Figura 199. Apartado Lessons en Hackety Hack.

Fuente: (Gillete, 2010)

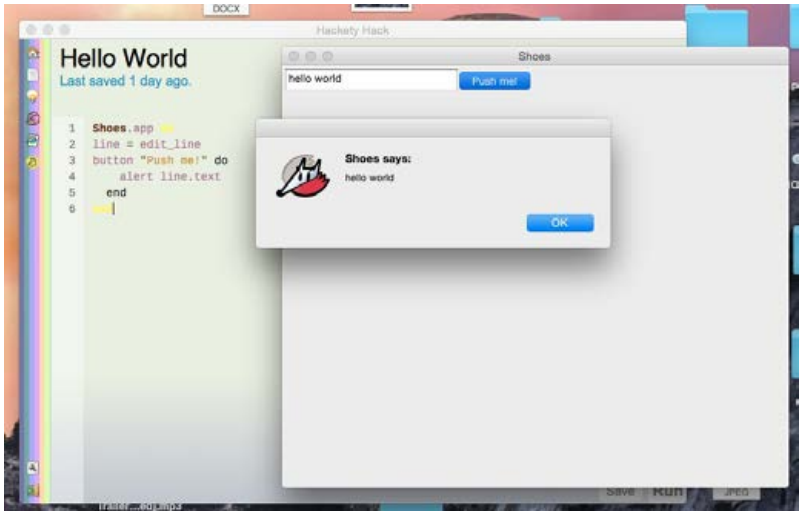


Figura 200. Un ejercicio en Hackety Hack.

Fuente: (Gillete, 2010)

En marzo de 2017 el sitio web oficial está caído, lo que hace suponer que la herramienta ha dejado de desarrollarse o recibir apoyo. Por este motivo, el análisis que a continuación se expone, se hace en base a la información recopilada previamente a esta fecha.

En la *Figura 201* se muestran los códigos que se pueden utilizar en la herramienta para pintar en diferentes colores.

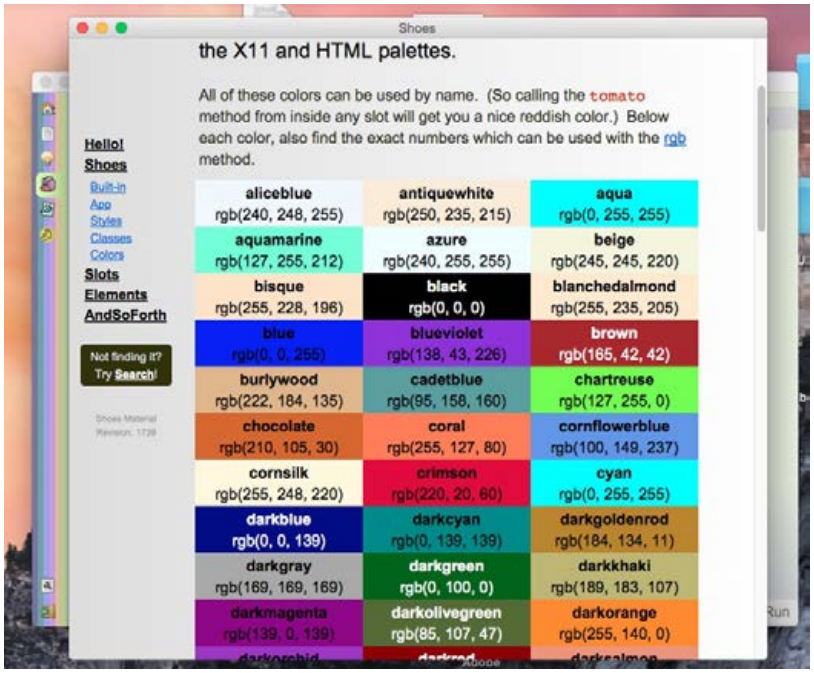


Figura 201. Paletas de colores y su códigos en Hackety Hack.

Fuente: (Gillete, 2010)

Hackety Hack dispone de una interfaz llamativa, llena de colores. Una de los primeros aspectos que llama la atención es la sección *Lessons* (Lecciones), que introduce al usuario en el código que se va a utilizar, y presenta los aspectos más significativos de la herramienta. Una vez hecho esto, la herramienta propone un pequeño ejercicio a resolver.

En la parte izquierda se podrán observar y elegir las distintas lecciones, así como una sección donde se recoge, en un esquema visual, todas las librerías de programación a disposición del usuario. También se informa sobre el lugar del que proviene cada una de ellas, si son invocadas en el código. Por último, decir que existe un pequeño apartado, denominado *Cheat Sheet* (hoja para “hacer trampas”), con atajos a las variables y funciones, como se muestra en la *Figura 202*.

Esta herramienta también contiene secciones que permiten adentrarse en el paradigma de la programación orientada a objetos, de una forma muy parecida a la que lo hace la herramienta *Turtle* (Tortuga), pero cabe decir que esta parte de la materia requiere de unos conocimientos más avanzados por parte del usuario, que sin la ayuda de un docente, será muy difícil. En general, la dificultad intrínseca de su

manejo sugiere la necesidad de la presencia de profesorado especializado, en apoyo del material que la propia herramienta ofrece (lecciones, ejercicios, etc.).

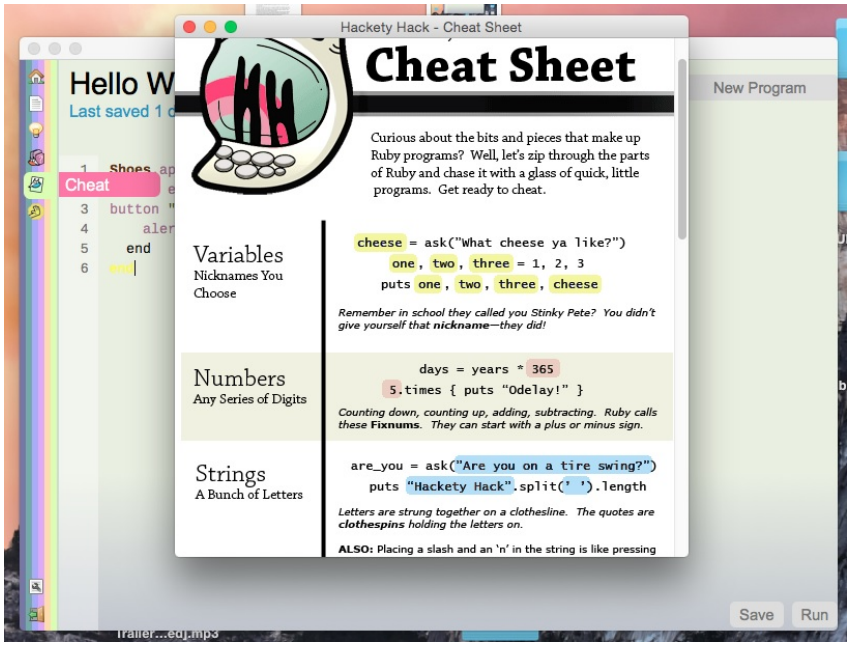


Figura 202. Apartado *Cheat Sheets* en Hackety Hack.

Fuente: (Gillete, 2010)

19. Hopscotch: Coding for Kids

Tabla 144
Resumen de características de la herramienta Hopscotch: Coding for Kids

Nombre de la herramienta	Hopscotch: *Coding for Kids*
Página web	https://www.gethopscotch.com/
URL de descarga	https://itunes.apple.com/us/app/hopscotch-coding-for-kids/id617098629?mt=8&ign-mpt=uo%3D4
Fecha de creación / aparición en el mercado	2013
Última versión en 2017 / año de esa versión	Hopscotch: Coding for Kids 3.15.0, del 7 de febrero del año 2017
Plataformas en las que se puede instalar	iPhone e iPad
Tipo de software	Privativo
Precio / modelo de negocio	Cuenta con dos versiones, una gratuita (temporal por 7 días) y una por suscripción (cada mes, cada 3 meses, 6 meses o 1 año), entre 6€ - 75€
Edades para las que está indicado	Edades entre los 4 - 14 años
Aspectos educativos que trabaja	Conceptos básicos de programación, conceptos básicos de la programación orientada a objetos, matemáticas, creatividad
Característica más destacada / valor diferencial	Poder crear tus propias aplicaciones y juegos, y compartirlos con el resto de usuarios desde una plataforma móvil, sin necesidad de tocar código directamente

Hopscotch: *Coding for Kids* es una aplicación para iOS pensada para la enseñanza de los conceptos básicos de la programación a niños y adolescentes, utilizando bloques con codificación predefinida, para la creación de proyectos y aplicaciones.

Con esta herramienta la programación se lleva a cabo a través de paneles, a los que se les puede ir añadiendo componentes con diferentes opciones, y de esta manera realizar lo que el usuario quiera conseguir.

En un proyecto nuevo, en la parte baja, los usuarios podrán añadir un panel base desde el que comenzar, dándole un nombre si así lo desean. Una vez creado, el usuario podrá colocar este panel en cualquier lugar de la pantalla. Cuando el panel esté situado en el lugar apropiado, se podrá pulsar sobre el botón *add code* (agregar código) para añadir otros paneles. Posteriormente, dentro del panel base, como se muestra en la *Figura 204*, en la parte baja aparecerán una serie de pestañas diferentes, con símbolos ordenados por colores, cada una de las cuales contiene paneles con comportamientos diferentes, así como objetos, como por ejemplo,

comenzar juego, dibujar desde un punto, seguir a un objeto, agregar un personaje, agregar un texto, agregar una forma, etc.

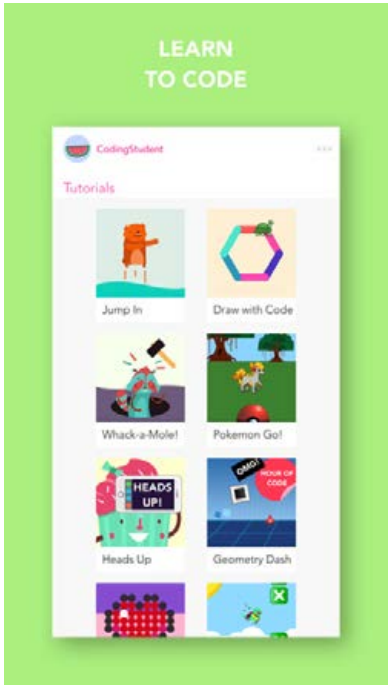


Figura 203. Pantalla de tutoriales.

Fuente: (Hopscotch Technologies, 2016b)

Figura 204. Creando un proyecto desde cero en Hopscotch.

Fuente: (Hopscotch Technologies, 2016b)

Es una herramienta muy interesante, con una interfaz intuitiva y sencilla. Además, ofrece una rápida visualización de resultados del código programado, así como diversas opciones de configuración, y comportamientos. Los usuarios podrán implementar casi todas las funcionalidades que deseen dentro de su programa, sin necesidad de utilizar código textual.

Hopscotch cuenta también con una significativa cantidad de tutoriales y ayudas dentro del programa, como se muestran en la *Figura 203,* además de una serie de videos colgados en la red que explican algunos conceptos de programación, e indican cómo crear ciertos tipos de proyecto desde cero. Sin embargo, la versión gratuita es temporal (7 días) y no cuenta con todo el contenido que mencionamos. Es necesaria una suscripción de pago para poder acceder a los tutoriales avanzados, y para poder subir a la herramienta nuestras propias imágenes y *assets* (elementos). La opción de pago también permite el acceso a las actualizaciones, y los futuros contenidos que el fabricante agregue a la herramienta. De la misma manera, la suscripción permite descargar proyectos de otras personas, y modificarlos a voluntad.

Hopscotch permite la exportación a HTML5 de los proyectos, pero no cuenta con soporte para que estos puedan ser creados en un entorno web.

En general podemos decir que es una herramienta con vocación didáctica. Se aprende a utilizarla en poco tiempo, y se puede usar como instrumento educativo de otras materias. Su precio es relativamente barato, teniendo en cuenta las prestaciones que brinda. Sin embargo, no ofrece ningún tipo de descuentos para docentes, que deben crearse una cuenta individual, en lugar de poder compartir una dentro del colegio. Otro inconveniente potencial, viene derivado de la ausencia de una opción para el control parental, si se desea que esta herramienta no sea utilizada sin supervisión.

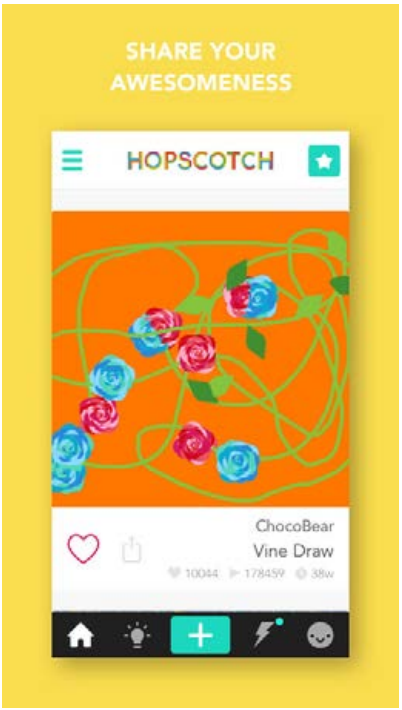


Figura 205. Una publicación en Hopscotch.

Fuente: (Hopscotch Technologies, 2016b)

20. Human Resource Machine

Tabla 145
Resumen de características de la herramienta Human Resource Machine

Nombre de la herramienta	Human Resource Machine
Página web	https://tomorrowcorporation.com/humanresourcemachine
URL de descarga	PC: http://store.steampowered.com/app/375820/ Android: https://play.google.com/store/apps/details?id=com.tomorrowcorporation.humanresourcemachine&hl=es
Fecha de creación / aparición en el mercado	15 de octubre del año 2015
Última versión en 2017 / año de esa versión	Depende de la plataforma: Android: https://play.google.com/store/apps/details?id=com.tomorrowcorporation.humanresourcemachine&hl=es
Plataformas en las que se puede instalar	Windows, MAC OS X, Wii U, Android
Tipo de software	Privativo
Precio / modelo de negocio	Depende de la plataforma:5,49€ - 9,99€
Edades para las que está indicado	7 años o superior
Aspectos educativos que trabaja	Resolución de problemas, estructuración, conceptos básicos de la programación
Característica más destacada / valor diferencial	Es un videojuego, la enseñanza está tratada a través de él

Human Resource Machine (*Figura 206)* es un videojuego de puzles por niveles desarrollado por "*Tomorrow Corporation*", en el que se controla a un oficinista dentro de una gran empresa de paquetería.

La mecánica principal consiste en utilizar diferentes instrucciones para llevar una serie de paquetes de una bandeja *IN* (dentro) a una bandeja *OUT* (fuera), con unas determinadas condiciones y retos en cada nivel.

El jugador posee una lista de funciones, que pueden arrastrar a un espacio a la derecha de la pantalla, y colocarlas según el orden en el que quiera utilizarlas, tratando de alcanzar su objetivo. En la parte baja de la pantalla se puede ver la opción de ejecutar el programa y así ver cómo funciona, acelerar su proceso, pausarlo, empezar de nuevo o retroceder. Todo lo descrito en este párrafo se puede observar en la *Figura 207.*

Figura 206. Pantalla de inicio de *Human Resource Machine*.

Fuente:(Tomorrow Corporation, 2017)

Figura 207. Un nivel de *Human Resource Machine*.

Fuente:(Tomorrow Corporation, 2017)

El juego cuenta con más de 40 niveles (*Figura 208),* con una interfaz intuitiva, simple y directa, utilizando texto y un código de colores para representar cada función dentro del juego. En cada nivel se premia la eficiencia del jugador, es decir, que la lista de funciones que haya programado, no sólo resuelva la tarea encomendada, sino que además lo haga de forma óptima (con menor número de funciones, y con un menor tiempo de ejecución).

No se trata realmente de una herramienta para aprender programación, sólo permite entrever una serie de conceptos básicos de este ámbito, como bucles o sentencias condicionales. Tampoco se tiene acceso al código generado, y no se puede extrapolarse a otros ámbitos.

Figura 208. Niveles dentro de *Human Resource Machine*.

Fuente:(Tomorrow Corporation, 2017)

En la *Figura 209* se muestra un nivel del juego completado, las estadísticas del jugador respecto al nivel, y su eficiencia en completarlo.

Figura 209. Nivel completo en *Human Resource Machine*.

Fuente:(Tomorrow Corporation, 2017)

21. Infinifactory

Tabla 146
Resumen de características de la herramienta Infinifactory

Nombre de la herramienta	Infinifactory
Página web	http://www.zachtronics.com/infinifactory/
URL de descarga	http://store.steampowered.com/app/300570/
Fecha de creación / aparición en el mercado	30 de junio del año 2015
Última versión en 2017 / año de esa versión	Depende de la plataforma, pero la página oficial tiene un último comunicado del 24 de julio del año 2015
Plataformas en las que se puede instalar	Windows, MAC OS X, Linux y PlayStation 4
Tipo de software	Privativo
Precio / modelo de negocio	Varía según la plataforma, en Steam: 23,99€
Edades para las que está indicado	Mayores de 12 años
Aspectos educativos que trabaja	Capacidades matemáticas y sociales
Característica más destacada / valor diferencial	Aprender a pensar y organizar diferentes formas de resolver un problema a la vez que juegas y eres participe de una historia

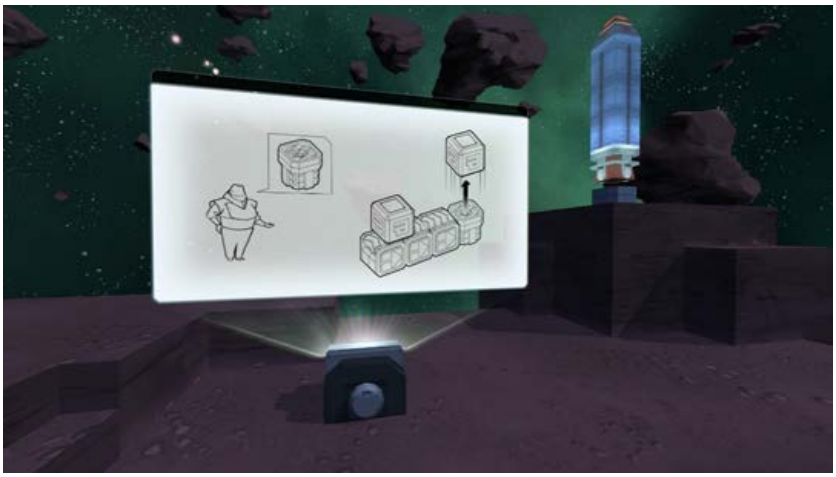


Figura 210. Uno de los tutoriales de Infinifactory.

Fuente: (Zachtronics, 2015)

Infinifactory es un videojuego de puzles en 3D, en el que los jugadores se ponen en la piel de un humano que ha sido abducido por extraterrestres, y es obligado a construir ciertos elementos con diferentes propósitos. Los elementos que el jugador debe construir se forman a base de objetos como botones, ensambladores, imanes,

etc. Cuando se consigue resolver un determinado número de puzles, se avanza a la siguiente zona.

Cuando el jugador entra en el juego y recibe las primeras instrucciones, puede observar que en la parte inferior izquierda tienen una serie de espacios destinados a la selección de los bloques que se podrán utilizar para resolver cada puzle. Asimismo, en la parte inferior derecha hay un reloj que indica al jugador el tiempo que se está tardando en resolverlo. Todo lo mencionado en este párrafo se puede observar en la *Figura 211*.

Figura 211. Un nivel en Infinifactory.

Fuente: (Zachtronics, 2015)

Una de las mecánicas de Infinifactory ofrece la posibilidad de pausar el tiempo, y hacerlo retroceder o avanzar a voluntad del jugador, para ver si su construcción funciona correctamente. El usuario puede avanzar más deprisa si cree tener una solución, o ver en qué punto se ha equivocado, o si falta alguna pieza para completar el puzle.

Otra de sus mecánicas principales es la construcción, similar a la que utiliza el videojuego Minecraft, pero esta no permitirá colocar cualquier bloque en cualquier lugar. De esta forma se reta a los jugadores a utilizar su creatividad y sus capacidades para resolver problemas.

Al terminar cada nivel, el juego hace una serie de preguntas al jugador para medir sus capacidades de resolución durante el nivel: si ha conseguido llegar hasta el final

satisfactoriamente, cuánto tiempo ha tardado, si le ha resultado muy difícil, y si el puzle propuesto le ha parecido interesante.

Figura 212. Un nivel en Infinifactory, visto sin interfaz.

Fuente: (Zachtronics, 2015)

Este juego está disponible en *Steam* [53], y como otros productos de esta plataforma, se ofrece al usuario la posibilidad de acceder al *Steam Workshop*, donde puede crear sus propios puzles y retos, siempre y cuando esté comprobado que el puzle se puede resolver (el sistema tiene mecanismos que hacen esta comprobación).

Si hablamos de su utilidad para aprender a programar, podemos decir que es una herramienta que favorece el pensamiento algorítmico (aunque hay puzles que se pueden resolver por ensayo y error), pero no se maneja ningún lenguaje de programación, con lo que resulta en este aspecto algo limitado.

[53] http://store.steampowered.com/?l=spanish

22. Karel

Tabla 147
Resumen de características de la herramienta Karel

Nombre de la herramienta	Karel
Página web	http://www.olimpiadadeinformatica.org.mx/omi/omi/material/Karel_el_Robot.aspx
URL de descarga	http://www.olimpiadadeinformatica.org.mx/omi/omi/material/Karel_el_Robot.aspx
Fecha de creación / aparición en el mercado	31 de Diciembre del año 2000
Última versión en 2017 / año de esa versión	Karel.exe, del 23 de agosto del año 2012
Plataformas en las que se puede instalar	Windows, MAC OS y Linux (aunque algunos necesitan de pequeños programas intermediarios)
Tipo de software	Privativo
Precio / modelo de negocio	Gratuito
Edades para las que está indicado	Edades comprendidas entre los 11 – 16 años
Aspectos educativos que trabaja	Estructuración, conceptos básicos sobre la programación y utilización de un lenguaje específico, y propio de un lenguaje de programación complejo.
Característica más destacada / valor diferencial	Posee acceso al código en varios lenguajes de programación

Karel es una herramienta de enseñanza de la programación, creada por Richard E. Pattis (2013), como extensión de su libro "*Karel el Robot: Una agradable introducción al arte de la programación*". Pattis pretendía con su libro y con la herramienta Karel enseñar a sus estudiantes a pensar de manera ordenada y eficiente. Karel funciona en la mayoría de sistemas basados en Unix (por ejemplo Linux), y además es utilizado oficialmente en una de las pruebas en la Olimpiada Mexicana de Informática [54]. En esta herramienta pueden usarse los lenguajes de programación Pascal, Java y Python (en el que se basó la herramienta Guido van Robot). Una de las características más reseñables de Karel es que se puede elegir cuál de estos lenguajes se quiere utilizar, y es posible cambiar de uno a otro en cualquier momento del programa.

El programa presenta una interfaz muy parecida a una herramienta de programación común, pero de una manera simplificada. En la parte superior podemos observar las opciones principales, como la distribución del nivel, opciones, inicializar el programa, ejecutar, etc.

[54] http://www.olimpiadadeinformatica.org.mx/

En la *Figura 213* se muestra un nivel de estilo “laberinto”, en el que el jugador debe programar las ordenes adecuadas, a través de código textual, para que su avatar consiga recoger todos los diamantes, y así pasar a la siguiente escena. La interfaz cuenta con un espacio para escribir el código, y un sistema de detección y corrección de errores, tanto sintácticos (no se ha escrito algo correctamente) como semánticos (con las instrucciones que se le da al robot, se topa con un obstáculo, o una pared, que no le permite avanzar).

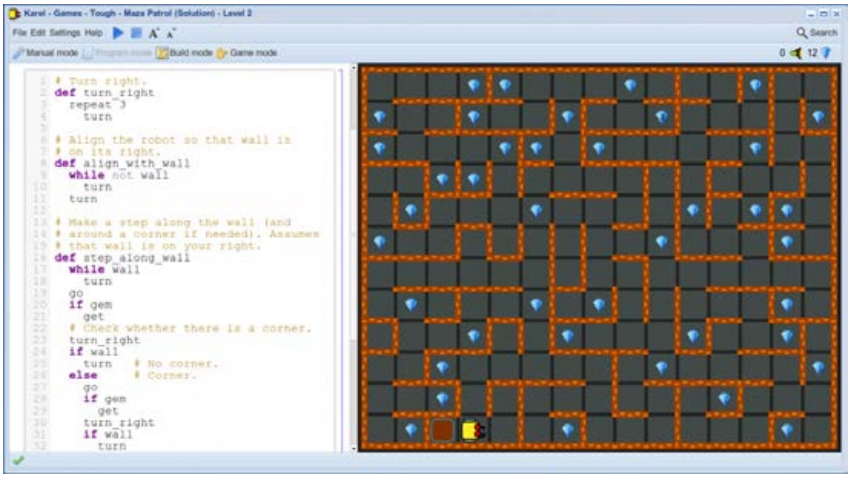


Figura 213. Un ejercicio sobre un laberinto en Karel.

Fuente: (Pattis, 2013)

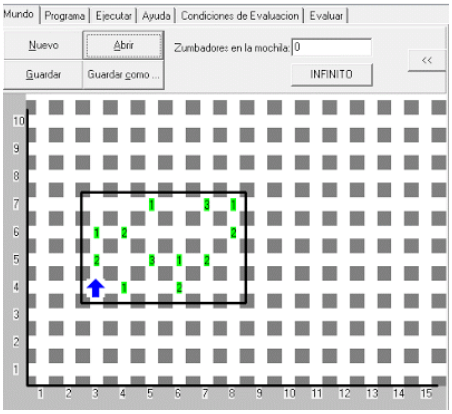


Figura 214. Un ejercicio de Karel.

Fuente: (Pattis, 2013)

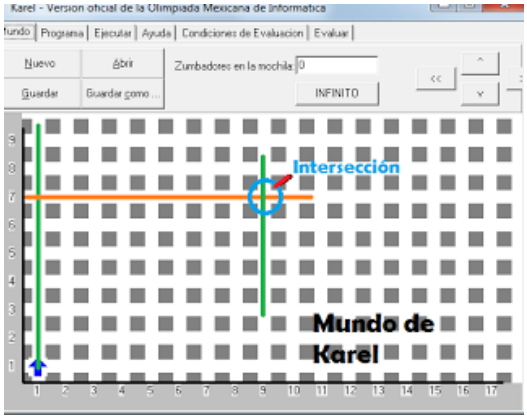


Figura 215. Otro ejercicio en Karel

Fuente: (Pattis, 2013)

Tanto la herramienta como el lenguaje de programación son muy sencillos de utilizar, y es posible manejar ambos con cierta soltura en relativamente poco tiempo, y con un uso mínimo de la ayuda.

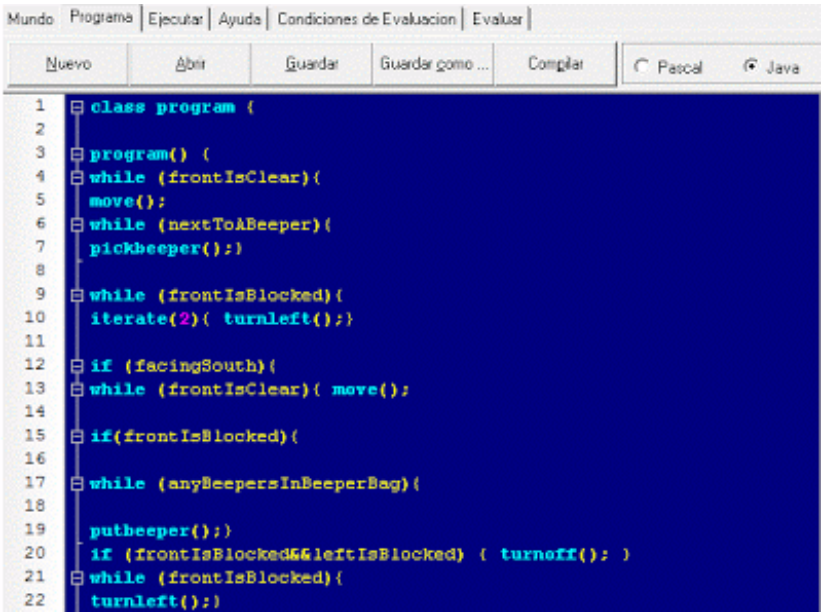


Figura 216. Un ejemplo del lenguaje de programación utilizado en Karel.

Fuente: (Pattis, 2013)

Aunque esta herramienta puede servir para asentar las bases de la programación del paradigma imperativo, las posibilidades de programar comportamientos complejos son escasas. Otra limitación está en los modos de juego que ofrece, que básicamente se limitan a laberintos en los que se deben recoger objetos.

A partir de la herramienta de Karel se ha desarrollado Guido van Robot, que mejora ciertos aspectos del original, y está traducido a varios idiomas. Otra mejora con respecto a Karel, es que Guido van Robot cuenta con una gran comunidad de usuarios, y una amplia variedad de documentación y tutoriales.

23. KidsRuby

Tabla 148
Resumen de características de la herramienta KidsRuby

Nombre de la herramienta	KidsRuby
Página web	http://kidsruby.com/
URL de descarga	http://kidsruby.com/download.html
Fecha de creación / aparición en el mercado	Año 2010 / 2011
Última versión en 2017 / año de esa versión	Depende de la plataforma, pero menos en dispositivos Raspberry Pi (versión 1.3) todos son la versión 1.4, del año 2014
Plataformas en las que se puede instalar	Windows, Mac OS X, Raspberry Pi y Debian Linux
Tipo de software	Privativo
Precio / modelo de negocio	Gratuito
Edades para las que está indicado	Mayores de 12 años
Aspectos educativos que trabaja	Conceptos básicos de la programación, resolución de problemas, programación orientada objetos
Característica más destacada / valor diferencial	Da mucha libertad creativa, está orientada a una enseñanza tutorada

Se trata de una herramienta para la enseñanza de programación muy similar a la anteriormente analizada Hackety Hack. En su página web [55] podemos encontrar una presentación de ambas herramientas.

KidsRuby está disponible para los sistemas Windows, Mac OSX, Raspberry Pi y Debian Linux Desde la web se pueden descargar los instaladores, pero cabe mencionar que tanto las versiones para Windows como para Mac presentan errores de instalación en las últimas versiones de estos sistemas operativos. Igualmente es necesario mencionar que KidsRuby necesita de bastantes recursos para su correcto funcionamiento, con lo que puede presentar problemas si se ejecuta en máquinas poco potentes.

[55] http://kidsruby.com/

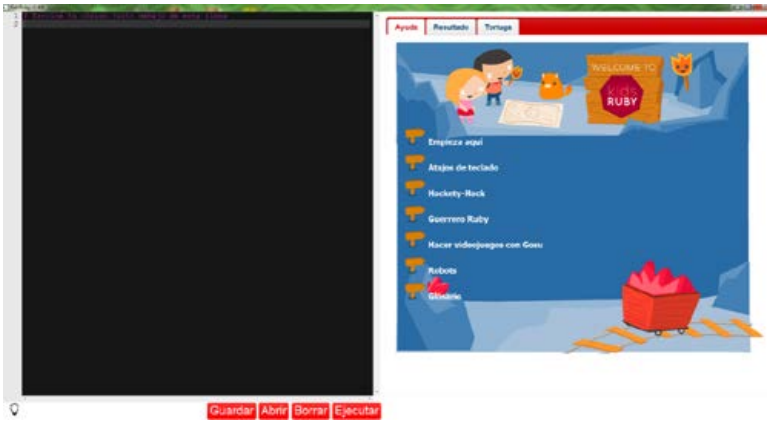


Figura 217. Pantalla inicial de KidsRuby.

Fuente: (The Hybrid Group, 2014)

Desde la sección de descargas de la web se puede acceder a ejercicios y problemas. También existe una sección Contribute (contribuir) donde los usuarios pueden acceder a documentación y ejercicios, así como compartir los propios con la comunidad.

En la interfaz de la herramienta podemos observar dos cuadros de trabajo distintos: uno para editar el código, y otro a la derecha, de manera auxiliar, con menús de ayuda y recordatorios. En la *Figura 218* se puede ver un ejemplo de un ejercicio resuelto a través de un programa elaborado con código textual.

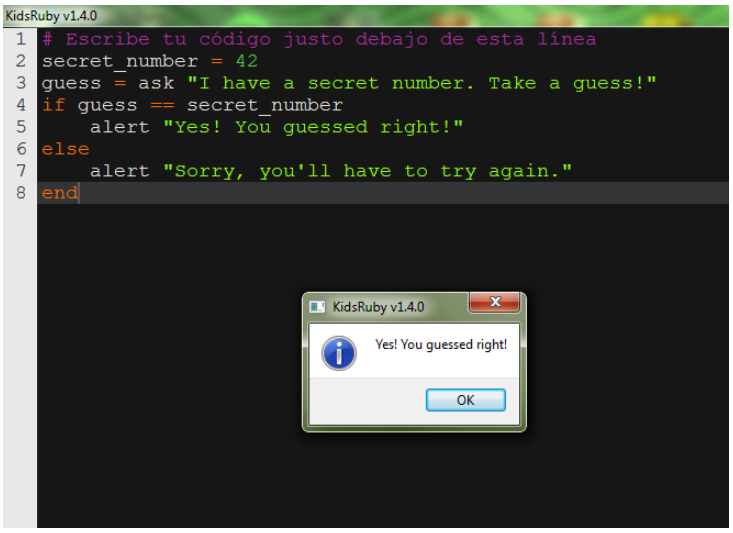


Figura 218. Ejercicio resuelto en KidsRuby.

Fuente: *(Pattis, 2012)*

En la *Figura 219* se puede ver otro ejemplo de un ejercicio, junto con la ventana donde se visualiza el resultado de su ejecución.

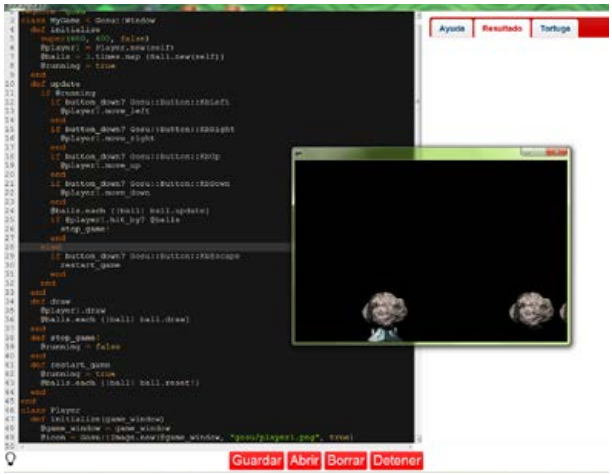


Figura 219. Previsualización de un código en KidsRuby.

Fuente: *(Pattis, 2012)*

En la sección “Ayuda” (*Figura 220* y *Figura 221*) podremos encontrar algo llamativo: Hackety-Hack y KidsRuby tienen en esta sección prácticamente los mismos contenidos. Esto es un hecho que evidencia el parecido entre ambas herramientas, solo que Hackety-Hack presenta un mayor número de utilidades, y en general se puede decir que es más completa. Investigando sobre el asunto, podemos destacar que el mismo creador de Hackety Hack, también es colaborador y coordinador en esta aplicación, al ser un compilador de Ruby.

Figura 220. Ventana “Ayuda” en KidsRuby.

Fuente: (Pattis, 2012)

Figura 221. Ejercicios de HacketyHack dentro de KidsRuby.

Fuente: (Pattis, 2012)

La herramienta incluye dentro de ella un tutorial para realizar un pequeño videojuego, y aprender aspectos relacionados con la programación orientada a objetos.

Además de estos extras, recoge un glosario *(Figura 222*) con los conceptos más importantes de programación, como por ejemplo, qué es una función o una variable.

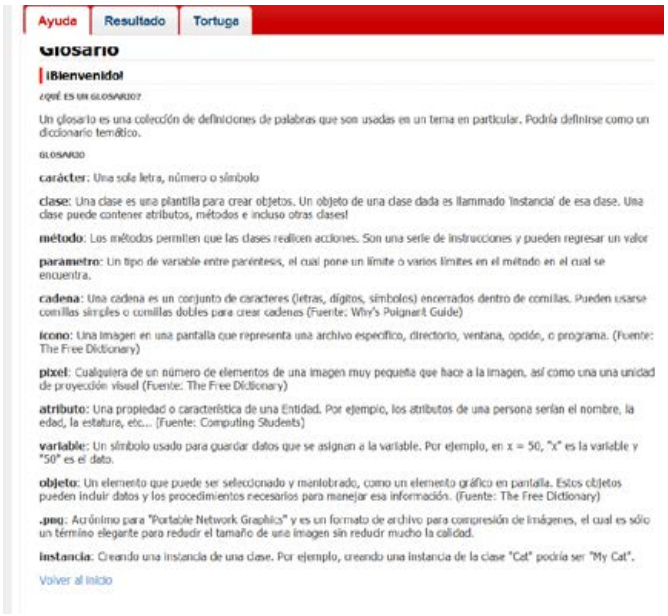


Figura 222. Glosario dentro de KidsRuby.

Fuente: (Pattis, 2012)

24. Kodable

Tabla 149
Resumen de características de la herramienta Kodable

Nombre de la herramienta	Kodable
Página web	https://www.kodable.com/
URL de descarga	https://www.kodable.com/pricing
Fecha de creación / aparición en el mercado	Año 2013
Última versión en 2017 / año de esa versión	Depende de la plataforma, pero en su mayoría son las versiones del 7 de febrero del año 2017
Plataformas en las que se puede instalar	Según la página web, puedes pedirla en cualquier plataforma
Tipo de software	Privativo
Precio / modelo de negocio	Existe una versión gratuita temporal (8 semanas) y luego 4 packs (Kick Start, School, District y Personal), cada uno con sus propios precios y enseñanzas, que van desde los 29$ hasta cifras más altas, pero que son acordadas con las escuelas una vez se quiera introducir la herramienta
Edades para las que está indicado	Edades entre los 6 – 8 años
Aspectos educativos que trabaja	Conceptos básicos de la programación, estructuración, resolución de problemas
Característica más destacada / valor diferencial	Puede ser utilizada en cualquier dispositivo, cuenta con muchos ejercicios actualizados regularmente y posee diferentes planes de enseñanza

Kodable (*Figura 223*) es una herramienta creada para la enseñanza de conceptos básicos de la programación a niños entre los 6 y los 8 años de edad, a través del lenguaje JavaScript.

Figura 223. Logo de Kodable.

Fuente: (SurfScore, 2016)

Kodable basa sus enseñanzas en la realización de ejercicios con forma de juego, que contienen puzles que los usuarios deben resolver. En estos puzles, el jugador deberá hacer que un pequeño personaje, a través de las órdenes que se le indiquen, siga por un camino determinado, y recoja ciertos objetos hasta llegar al final, donde se completará el nivel y se avanzará al siguiente. Las órdenes se darán a través de una serie de símbolos, que representan unas instrucciones concretas preprogramadas.

Con respecto a su interfaz, en la parte izquierda superior podemos encontrar el espacio de trabajo, donde los usuarios, arrastrando los símbolos adecuados en el orden correcto, crean el programa que resolverá el puzle. En la parte derecha, estarán los símbolos de los que se dispone en ese momento, pudiendo utilizarse tantos del mismo tipo como se quiera. Finalmente, en la parte baja, está el área de visualización del puzle, el personaje, el camino a seguir, los objetos a conseguir, etc.

Los primeros niveles están orientados a los más pequeños, y en ellos se enseñan conceptos básicos, tanto generales del juego, como la forma de modificar la dirección y el sentido del personaje, como propios de la programación, como bucles y sentencias condicionales (estos últimos tendrán su propio espacio de trabajo). Según se avance por los distintos niveles, la complejidad de los puzles aumentará, así como el número de símbolos o instrucciones disponibles.

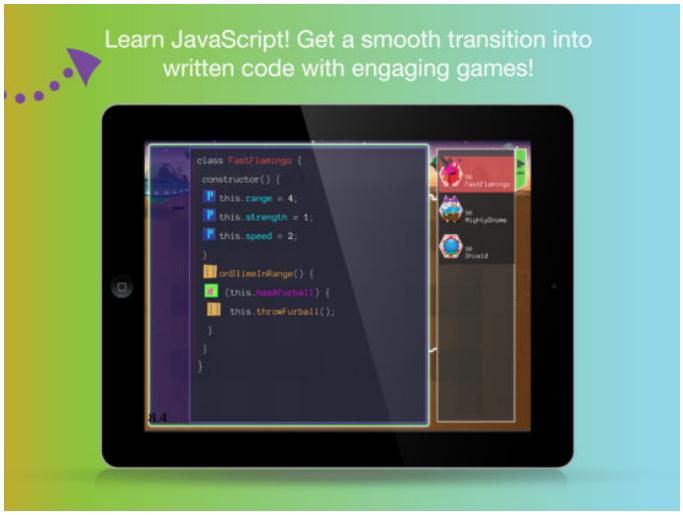


Figura 224. Un ejercicio en Kodable utilizando código.

Fuente: (SurfScore, 2016)

La herramienta cuenta con varios packs de enseñanza, adaptados a usuarios de distintas edades, y en el que se trabajan diferentes conceptos de programación.

Kodable está pensada para ser utilizada en una clase con un tutor especializado en la materia. En su página web se pueden encontrar una gran variedad de recursos para docentes. También cabe destacar que la herramienta cuenta con recursos orientados al ámbito familiar, para que madres y padres puedan llevar el control del avance de sus hijos. La herramienta proporciona datos para hacer este seguimiento: horas de trabajo, temario a realizar, puzles completados correctamente, etc.

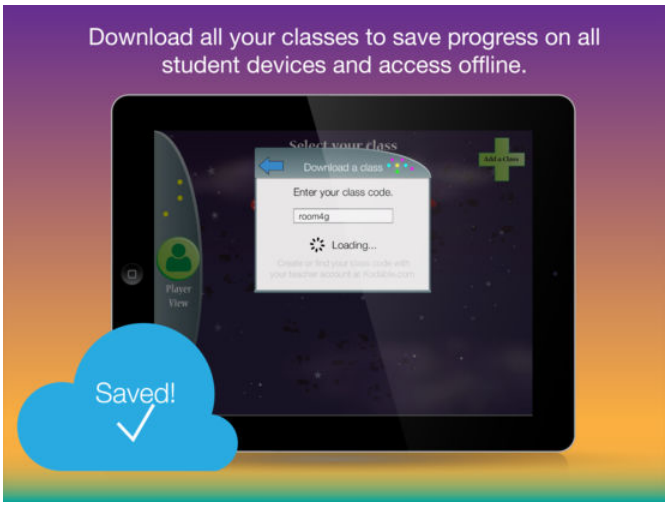


Figura 225. Descarga de una clase de enseñanzas en Kodable.

Fuente: (SurfScore, 2016)

La herramienta se actualiza regularmente, y se añaden frecuentemente nuevos ejercicios y documentación.

Por último mencionar que Kodable cuenta con una amplia comunidad de usuarios. La empresa responsable de Kodable promueve una serie de eventos, como competiciones entre escuelas, para mantener activa a su comunidad. También ofrece espacios virtuales para compartir experiencias, ejercicios, tutoriales, etc.

25. Kodu

Tabla 150
Resumen de características de la herramienta Kodu

Nombre de la herramienta	Kodu
Página web	https://www.kodugamelab.com/
URL de descarga	https://www.microsoft.com/en-us/download/details.aspx?id=10056
Fecha de creación / aparición en el mercado	30 de junio del año 2009
Última versión en 2017 / año de esa versión	Kodu.exe, Versión 1.4.164.0 del día 12 de junio del año 2016
Plataformas en las que se puede instalar	Windows XP, Vista, 7 y 8
Tipo de software	Privativo
Precio / modelo de negocio	Gratuito
Edades para las que está indicado	Niños con edades entre los 6 – 11 años
Aspectos educativos que trabaja	Conceptos básicos de la programación, creatividad, estructuración
Característica más destacada / valor diferencial	Creación de escenarios en 3D y la adición de personajes e IA (inteligencia Artificial)

Kodu (*Figura 226*) es una herramienta de programación visual, concebida principalmente para el desarrollo de videojuegos. Su interfaz está diseñada para ser accesible a un público infantil. La interacción con esta interfaz se puede hacer con un teclado, un ratón, o con el controlador de Xbox.

Figura 226. Menú principal de Kodu.

Fuente: (Microsoft Research, 2016)

Cuando se crea un nuevo proyecto en Kodu, se muestra un escenario genérico vacío, y una barra de herramientas en la parte baja, donde se encuentran los objetos disponibles para incluir en dicho escenario..

Cuando se accede a cada objeto, en la parte superior izquierda, a modo de tutorial, se muestran las diferentes funciones, y posibles configuraciones de las que dispone: cambiar su tamaño, altura, rotación, etc., y, la más interesante, la opción “programar”, en la que se pueden controlar el comportamiento de cada objeto (su movimiento, los eventos a los que responde, etc.). También existen opciones para modificar su gráfica.

Igualmente se puede controlar la iluminación y la cámara del mundo creado. Existe un botón “jugar”, que permitirá probar lo que se haya creado hasta ese momento

En la *Figura 227* se muestran algunas de las opciones que ofrece la herramienta a la hora de editar las características de un objeto.

Figura 227. Opciones diferentes para un objeto en Kodu.

Fuente: (Microsoft Research, 2016)

Todos los mundos creados pueden ser guardados, y compartidos online con el resto de usuarios.

Una de sus principales ventajas de Kodu, es que cualquier persona puede comenzar a utilizarlo sin necesidad de tener formación previa en el ámbito de la programación

Como desventaja se puede decir que no permite visualizar el código generado de manera directa. El lenguaje de programación utilizado es uno propio de la herramienta, con lo que su aprendizaje no se puede extrapolar a otros ámbitos.

26. Laby

Tabla 151
Resumen de características de la herramienta Laby

Nombre de la herramienta	Laby
Página web	http://laby.thr.pm/
URL de descarga	https://play.google.com/store/apps/details?id=pm.thr.laby
Fecha de creación / aparición en el mercado	Año 2014
Última versión en 2017 / año de esa versión	Versión de Laby 1.2.4 del 28 de noviembre de 2014
Plataformas en las que se puede instalar	iOS y Android
Tipo de software	Privativo
Precio / modelo de negocio	Gratuito
Edades para las que está indicado	Desde los 3 años
Aspectos educativos que trabaja	Matemáticas, cálculo
Característica más destacada / valor diferencial	Herramienta gratuita

Todas las versiones de Laby *(Figura 228*) mantienen el mismo formato y contenido, tanto para PC, dispositivos móviles, o navegadores web. En la web del fabricante se proporciona el enlace a cada una de estas versiones.

Figura 228. Pantalla principal de la web de Laby.

Fuente: *(THR Games, 2015)*

El hecho de que tenga un diseño común para todos los dispositivos, puede dar lugar a cierta confusión en determinadas acciones. Por ejemplo, si dentro del navegador pulsamos en el botón backpage (volver a la página anterior), saldremos de la aplicación, no al último lugar donde hemos estado dentro de ella (para esto último existe un botón concreto).

Laby tiene dos modos de juego: normal (jugaremos contra el ordenador) y multijugador (jugaremos contra otros usuarios). Para el modo multijugador se requiere tener una cuenta de Facebook, y conceder ciertos permisos a la aplicación para que interactúe con la red social.

En una modalidad u otra el juego consiste en lo siguiente: se ofrece al jugador un tablero, en el que unas casillas contendrán números u operaciones aritméticas, y otras estarán vacías. Inicialmente se propondrá un número por el que empezar, y un resultado al que llegar. Partiendo de ese número inicial tendremos que movernos por el tablero, como si se tratara de un laberinto, para llegar al resultado. Para avanzar de una casilla a otra, en la casilla vacía intermedia, se deberá poner o el número o la operación que combinada con la casilla origen, da como resultado la casilla destino. Se dispone como máximo de 5 minutos para resolver el puzle. Una vez resuelto, se avanzará al siguiente, de más dificultad.

Figura 229. Ejemplo de un nivel en la web de Laby.

Fuente: *(THR Games, 2015)*

En la *Figura 230* se muestra la pantalla que aparece después de completar un nivel, donde aparece el número de estrellas ganadas. El máximo número de estrellas se obtendrá si se ha conseguido resolver el puzle con el menor número de pasos posibles, y en el menor tiempo. El nivel se podrá repetir tantas veces como se quiera, para conseguir todas las estrellas. Los logros se pueden compartir en Facebook.

Figura 230. Nivel completado en la web de Laby.

Fuente: *(THR Games, 2015)*

Es una herramienta útil para realizar ejercicios de cálculo mental, y trabajar el pensamiento algorítmico. Sin embargo trabaja muy pocos conceptos del ámbito de la programación. También decir que existe poca documentación relacionada con la aplicación, y que no dispone de espacio para la comunidad de usuarios.

27. Learn to program with BASIC

Tabla 152
Resumen de características de la herramienta BASIC

Nombre de la herramienta	BASIC 256
Página web	http://www.basic256.org/index_en
URL de descarga	https://sourceforge.net/projects/basic256prtbl/
Fecha de creación / aparición en el mercado	30 de enero del año 2014
Última versión en 2017 / año de esa versión	BASIC256Portable_1.99.99.67.paf.exe, del 9 de septiembre del año 2016
Plataformas en las que se puede instalar	Windows
Tipo de software	Libre
Precio / modelo de negocio	Gratuito
Edades para las que está indicado	Mayores de 15 años
Aspectos educativos que trabaja	Estructuración, creatividad, conceptos básicos de la programación
Característica más destacada / valor diferencial	Traducido a varios idiomas, un lenguaje sencillo de aprender

Se trata de una herramienta específicamente concebida para la enseñanza de la programación.

Su interfaz está estructurada de la siguiente manera. Al igual que otras aplicaciones de escritorio, dispone de una barra de herramientas, en la parte superior, desde la que se accede a las opciones, abrir o crear un nuevo proyecto, guardar, ejecutar, pausar o depurar el mismo, o acceder a la ayuda.

Bajo la barra de herramientas se encuentra el área de codificación, que ocupa la mayor parte de la pantalla. Este espacio está habilitado para escribir en él el código de programación.

A la derecha del área de codificación se muestran dos ventanas, una arriba, denominada *Text Output* (salida de texto), y otra abajo, denominada *Graphics Output* (salida de gráficos). En *Text Output* se muestran los mensajes que el sistema emite al ejecutar el código, como por ejemplo, si se produce un error de compilación. En *Graphics Output* se podrá ver el resultado del código programado.

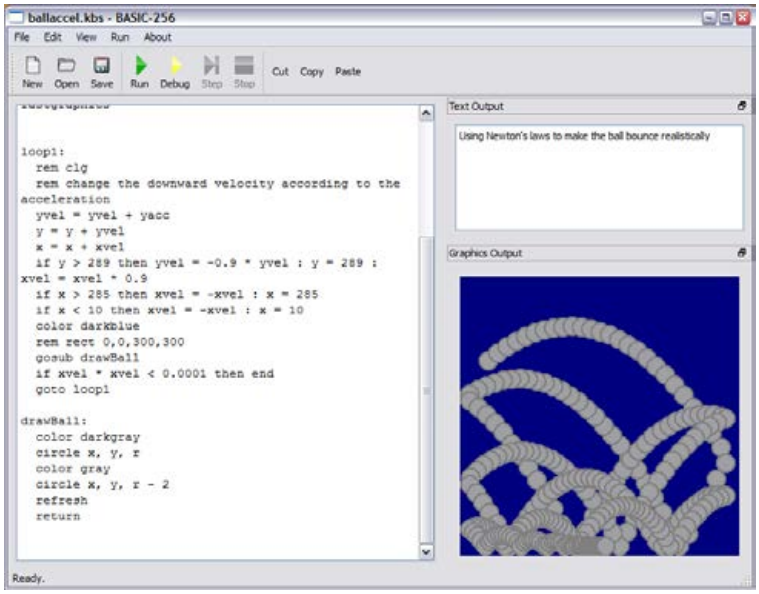


Figura 231. Ejemplo de ejercicio realizado con BASIC 256.

Fuente: (Larsen y Reneau, 2010)

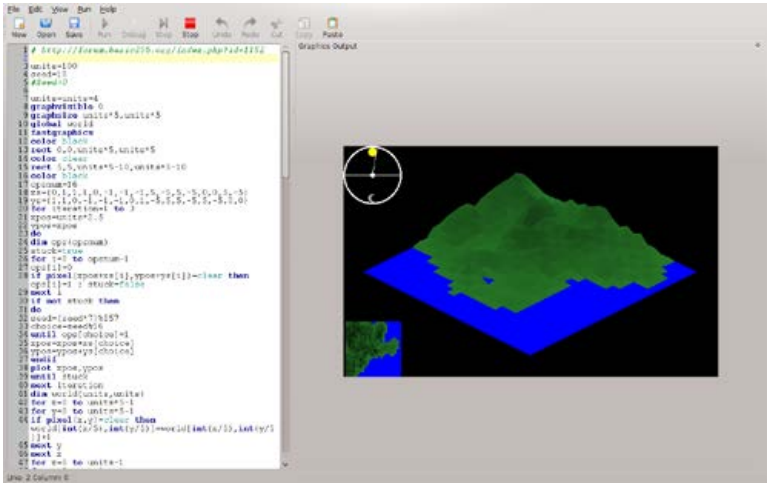


Figura 232. Ejemplo de ejercicio realizado con BASIC donde se genera un terreno.

Fuente: (Larsen y Reneau, 2010)

La interfaz de esta herramienta está traducida a varios idiomas. Además, cuenta con gran cantidad de tutoriales para aprender a utilizarla, separados por temáticas.

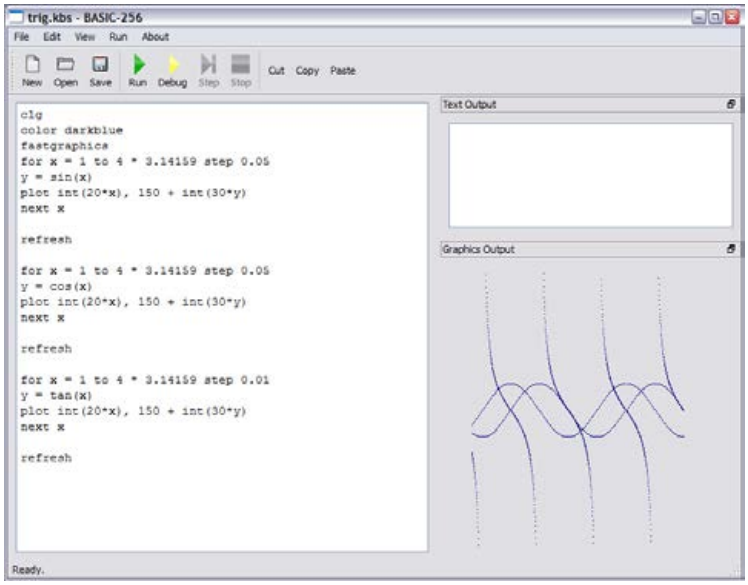


Figura 233. Ejemplo de ejercicio realizado con BASIC donde se experimenta con formas y colores.

Fuente: (Larsen y Reneau, 2010)

Como inconveniente, podemos decir que el lenguaje de programación que utiliza no tiene apenas aplicación en el ámbito profesional, dado que no permite hacer tareas excesivamente complejas.

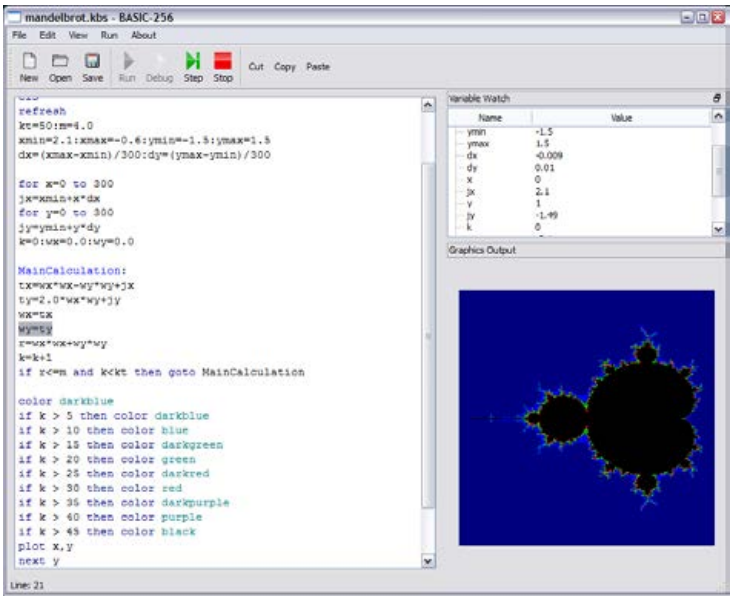


Figura 234. Ejemplo de ejercicio realizado con BASIC donde se representan figuras geométricas

Fuente: (Larsen y Reneau, 2010)

28. Lego Mindstorms

Tabla 153
Resumen de características de la herramienta Lego Mindstorms

Nombre de la herramienta	Lego Mindstorms
Página web	https://www.lego.com/en-us/mindstorms/?domainredir=mindstorms.lego.com
URL de descarga	https://www.lego.com/en-us/mindstorms/downloads
Fecha de creación / aparición en el mercado	Septiembre del año 1998
Última versión en 2017 / año de esa versión	Para Android: versión 1.0.71, del 25 de enero del año 2017 Para iOS: versión del 9 de diciembre del año 2016
Plataformas en las que se puede instalar	Windows (Vista o superior), MAC (10.6 o superior), Linux, Android(4.2 o superior) e iOS(8 o superior)
Tipo de software	Libre
Precio / modelo de negocio	Depende del producto, el hardware contiene el software para el desarrollo, a partir de los 400$
Edades para las que está indicado	Edades entre los 4 – 14 años
Aspectos educativos que trabaja	Estructuración, resolución de problemas, estrategia, geometría, utilización de un lenguaje específico
Característica más destacada / valor diferencial	Comprobación de resultados de forma "palpable" a través de los robots creados

Lego Mindstorms (*Figura 235*) se trata de una serie de robots, vendidos como juguetes, dentro de la línea de productos que ofrece la compañía LEGO. Estos robots están pensados para que niños y niñas puedan aprender las bases de la robótica, la programación y la inteligencia artificial. Lego Mindstorms, utiliza un software basado en la GUI[56] de Robolab.

Lego Minstorms cuenta con sus propias herramientas de hardware y software, pero existe la posibilidad de utilizar kits de desarrollo externos, y lenguajes de programación como Java, BrickOS, Not Quite C y Physical Etoys.

Un robot de Lego Minstorms está compuesto por piezas de construcción de LEGO, motores, sensores, engranajes, ruedas, ejes, etc.

[56] GUI, *Graphical User Interface* (Interfaz Gráfica de Usuario)

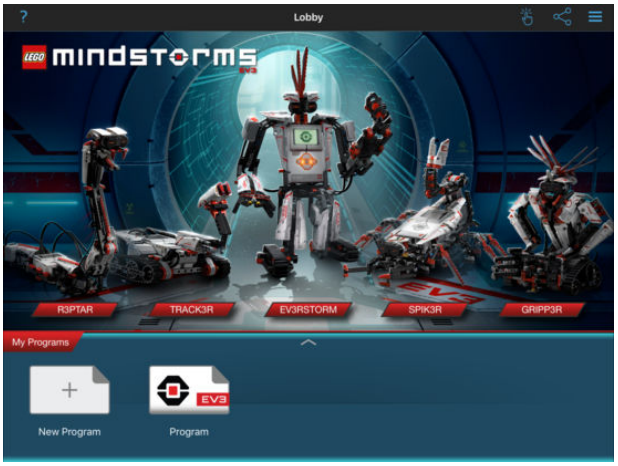


Figura 235. Lobby y robots de Lego Mindstorms.

Fuente: (LEGO Group, 2017)

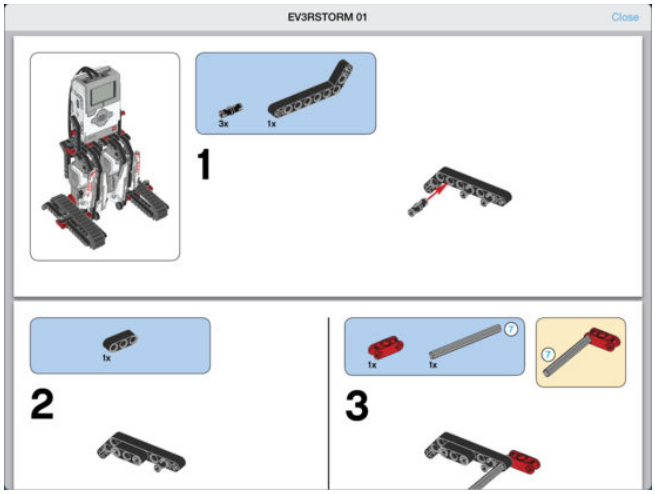


Figura 236. Instrucciones de montaje de uno de los robots de Lego Mindstorms.

Fuente: (LEGO Group, 2017)

Para el estudio del software que programa al robot se ha elegido la versión EV3. Al abrir este programa, nos encontramos con un espacio de trabajo como el que se ve en la *Figura 237.*

La programación de los robots se realiza a través de un sistema de bloques, muy similar al de herramientas como Scratch. Cada uno de esos bloques estará asociado a una función. Asimismo, los bloques contienen dentro de ellos una serie de parámetros que se podrán modificar (dependiendo del bloque, estos parámetros modifican la velocidad, el tiempo de ejecución, el sonido que se produce, el evento al que están asociados, etc.). Para hacer que el robot tenga el comportamiento

deseado, se deberán colocar los bloques, convenientemente configurados, en el área de programación, en el orden adecuado.

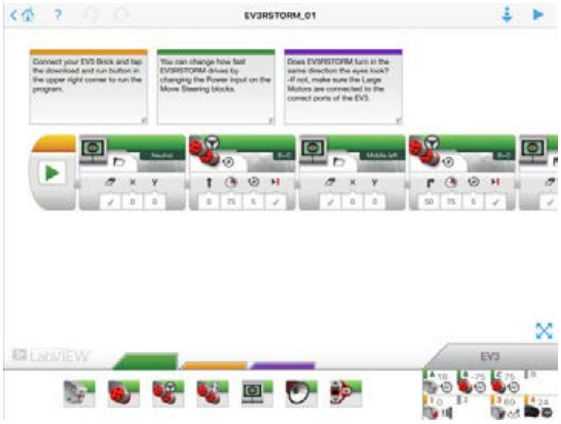


Figura 237. Inicio de la herramienta Lego Mindstorms.

Fuente: (LEGO Group, 2017)

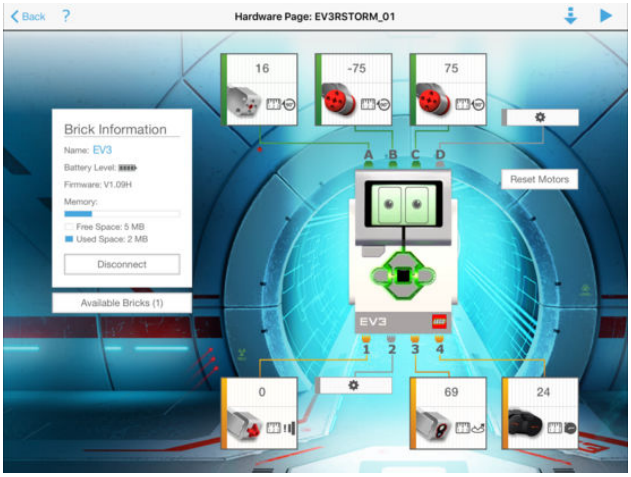


Figura 238. Bloque EV3 y sus diferentes puertos de salida en Lego Mindstorms.

Fuente: (LEGO Group, 2017)

El programa permite crear proyectos desde cero, o utilizar las plantillas que incorpora la herramienta, con las opciones más comunes (por ejemplo, hacer que el robot se mueva, y que cambie de dirección si se encuentra un obstáculo). Las plantillas son realmente útiles para los usuarios noveles, y evita el “miedo al lienzo en blanco” que puede experimentar un aprendiz de programador.

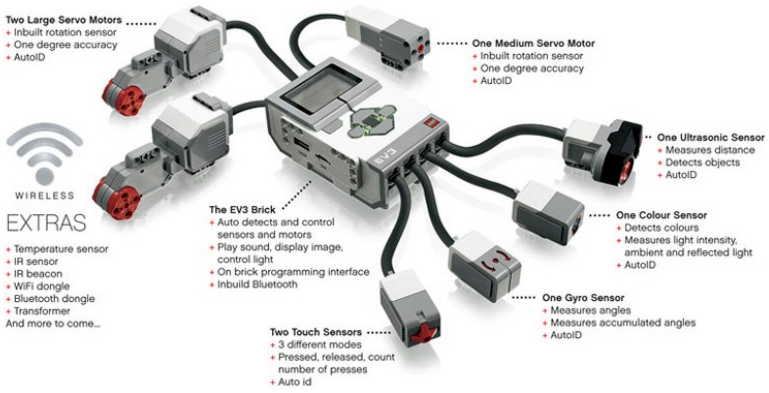


Figura 239. Bloque EV3, salidas y pequeña explicación de cada una.

Fuente: (Zona Outdoor, 2017)

Lego Mindstorms también cuenta con varias guías de trabajo, que pueden descargarse directamente desde la web, así como tutoriales en forma de vídeos de Youtube. La web del fabricante cuenta con foros, y espacios para su comunidad de usuarios.

29. Lightbot

Tabla 154
Resumen de características de la herramienta Hopscotch: Coding for Kids

Nombre de la herramienta	Lightbot
Página web	https://lightbot.com/
URL de descarga	https://lightbot.com/
Fecha de creación / aparición en el mercado	27 de octubre del año 2014 (en la versión adulta de la herramienta, para MAC)
Última versión en 2017 / año de esa versión	Lightbot versión 1.6.5 (para Android), 23 de enero del año 2016
Plataformas en las que se puede instalar	iOS, Android, Windows, MAC y Kindle
Tipo de software	Privativo
Precio / modelo de negocio	Depende de la plataforma, de 2´99$ - 4,99$
Edades para las que está indicado	Existen 2 versiones: Lightbot Jr. a partir de los 4 años Lightbot: 9 – 11 años
Aspectos educativos que trabaja	Espacialidad, estructuración, conceptos básicos de la programación
Característica más destacada / valor diferencial	Tiene versiones diferentes en relación a la edad de los usuarios, existe en muchas plataformas diferentes

Ligthbox es un videojuego en el que se trata de hacer llegar a un pequeño robot hasta el final de un mapa. El jugador deberá programar las instrucciones que seguirá el robot para llegar a una meta, atravesando un camino que albergará una serie de retos. Estas instrucciones le harán avanzar hacia un lugar, encender plataformas, activar objetos, girar, saltar, etc. Cada instrucción está representada por un símbolo. La programación, por tanto, consiste en arrastrar y soltar en el área de codificación los símbolos adecuados en el orden correcto para la consecución de los objetivos.

Cuando se complete un nivel, se avanzará al siguiente, que presentará mayor dificultad.

La interfaz del juego está dividido en varias partes, según se muestra en la *Figura 240*:

El área de mayor tamaño es la del escenario que el robot debe transitar. Debajo de este, se encuentran las instrucciones (símbolos) que se podrán utilizar en el nivel en el que se esté. A la derecha está el área de codificación, donde se arrastran los símbolos que componen el programa. Algunas acciones permiten crear nuevas zonas de trabajo (por ejemplo, esto es lo que se debe hacer para crear bucles o condiciones).

En la parte superior encontramos algunas opciones, como pausar el juego, tener acceso a la ayuda, un botón para reiniciar el nivel y otro para volver atrás, al menú de niveles. También, en el lado izquierdo, se encuentran las opciones de sonido, y la velocidad a la que se desarrolla la acción.

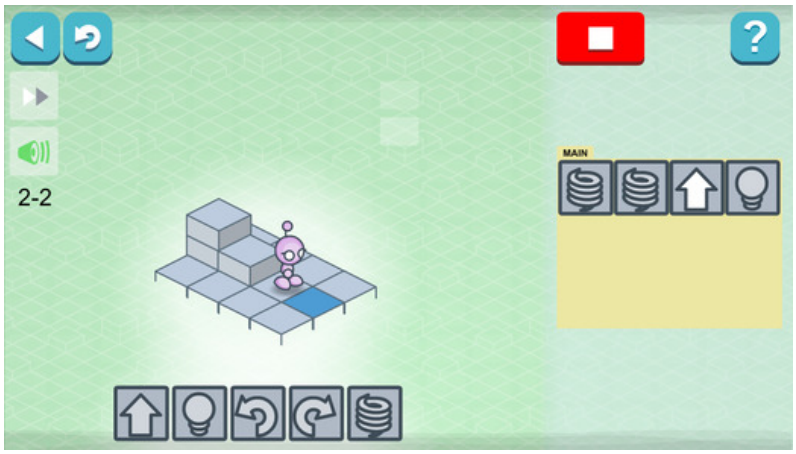


Figura 240. Un nivel de Lightbot.

Fuente: (Lightbot Inc, 2016)

Lightbot es una herramienta eminente lúdica, que permite trabajar el pensamiento algorítmico, y sobre todo está indicado para niñas y niños de edades muy tempranas. Como herramienta de programación tiene escaso valor.

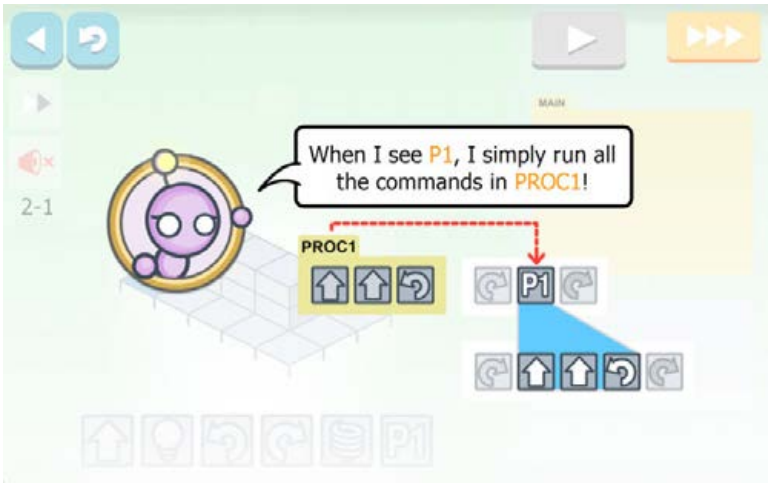


Figura 241. Un tutorial en Lightbot.

Fuente: (Lightbot Inc, 2016)

30. Learn to program with Logo

Tabla 155
Resumen de características de la herramienta WinLogo

Nombre de la herramienta	WinLogo
Página web	http://neoparaiso.com/logo/winlogo.html
URL de descarga	http://neoparaiso.com/logo/winlogo.html
Fecha de creación / aparición en el mercado	Año 1967
Última versión en 2017 / año de esa versión	Versión del año 2015
Plataformas en las que se puede instalar	Windows, pero también puede usarse a través de la web
Tipo de software	Privativo
Precio / modelo de negocio	Gratuito
Edades para las que está indicado	A partir de los 15 años
Aspectos educativos que trabaja	Matemáticas, geometría, estructuración, conceptos básicos sobre la programación
Característica más destacada / valor diferencial	Muy fácil aprendizaje, traducido a varios idiomas. Herramienta gratuita

Logo es una herramienta de programación, diseñada con fines didácticos, y basado en el lenguaje Lisp. Los detalles sobre su origen se pueden consultar en el capítulo 4.1.4.

Logo trabaja todos los conceptos básicos de programación (variables, bucles, sentencias condicionales, etc.), y soporta el manejo de listas, archivos, y entrada/salida.

En esta herramienta el usuario dará una serie de instrucciones a una tortuga virtual, para que recorra un espacio en pantalla. En su trayectoria la tortuga dibujará un rastro, y según su recorrido se podrán crear distintas figuras.

Logo cuenta con distintas versiones y entornos de programación. En la *Figura 242* se puede observar la interfaz de uno de estos entornos, con la tortuga en la ventana central.

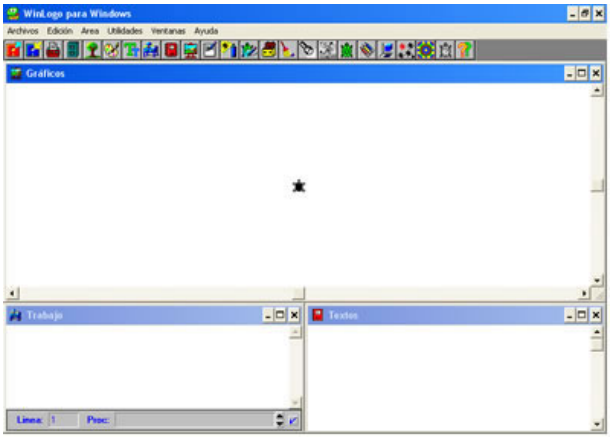


Figura 242. Interfaz de WinLogo.

Fuente: *(Bobrow, Feurzeig y Papert, 2007)*

Aunque el aspecto de la interfaz varía según la versión, podemos encontrar en todas ellas apartados similares:

- Espacio de trabajo, donde los usuarios podrán escribir y modificar su código.
- Espacio de visualización, donde se podrá comprobar lo que hace la tortuga como respuesta a las instrucciones programadas.
- Ventana de texto, donde se muestra información sobre la ejecución del programa (errores de compilación, etc.).

Figura 243. Barra de herramientas de WinLogo.

Fuente: *(Bobrow et al., 2007)*

Logo está traducido a varios idiomas, tanto el entorno, como sus instrucciones de uso y tutoriales. Además, existen varias versiones con diferentes interfaces, con lo que los usuarios pueden elegir la que prefieran o les resulte más cómoda.

31. Made with Code

Tabla 156
Resumen de características de la herramienta Made with Code

Nombre de la herramienta	Made with Code
Página web	https://www.madewithcode.com/
URL de descarga	https://www.madewithcode.com/
Fecha de creación / aparición en el mercado	Se creó en el año 2010, pero salió al mercado el 19 de julio de 2014
Última versión en 2017 / año de esa versión	Versión del año 2017
Plataformas en las que se puede instalar	Funciona a través de la web: Chrome, Firefox, Safari, Opera, IE, Android, iOS
Tipo de software	Privativo
Precio / modelo de negocio	Gratuito
Edades para las que está indicado	A partir de los 8 años
Aspectos educativos que trabaja	Conceptos básicos de la programación, diseño, arte, animación
Característica más destacada / valor diferencial	Apoyado por Google y su posibilidad de ser usado a través de la web

Google, en una investigación que llevó a cabo, llegó a la conclusión, de que había más hombres que mujeres en profesiones relacionadas con la programación. Con la intención de incentivar un cambio en esta tendencia, impulsó la creación del proyecto *Made with Code*.

Made with Code consta de una aplicación web, orientada a un público infantil y juvenil, eminentemente femenino. En ella se incluyen una serie de juegos que requerirán de la codificación de un programa para resolverlos. El motor de programación es el mismo que el de Blockly, también desarrollado por Google. Para poder entender cómo *Made with Code* enseña a programar, a continuación se describen algunos de estos juegos:

- Led Dress (*Figura 244*):

Se trata de diseñar un vestido animado con luces y colores, a través de instrucciones programadas.

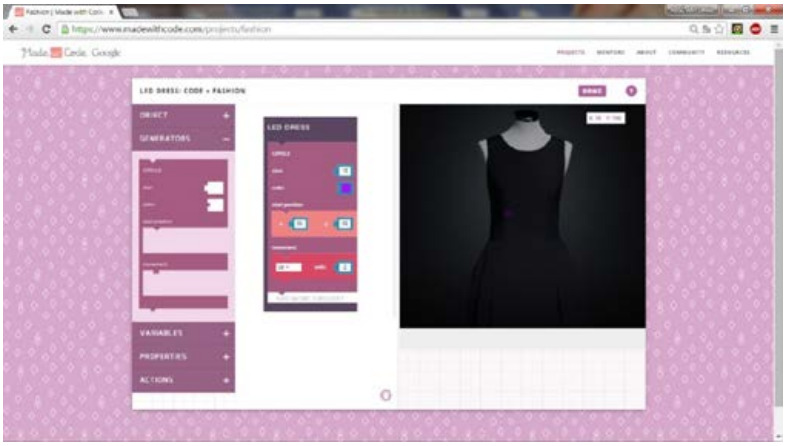


Figura 244. Juego "Led Dress" en la página web de Made with Code.

Fuente: (Google Company, 2016)

- Yeti (*Figura 245)*:

En este minijuego el usuario es guiado por un proceso que crea un monstruo, le cambia de apariencia, y le hace bailar.

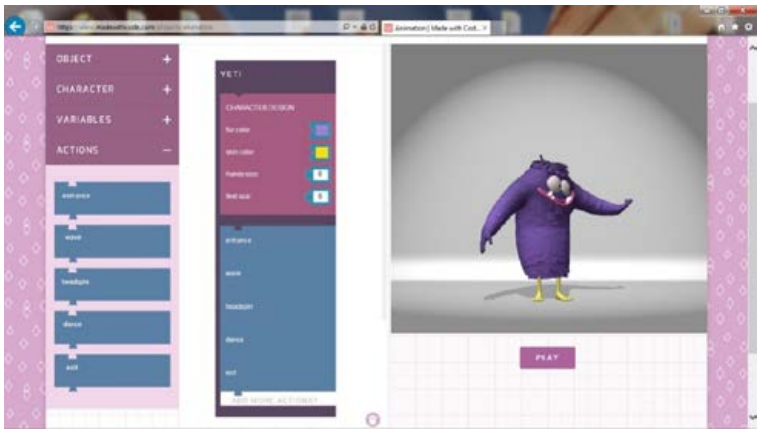


Figura 245. Juego "Yeti" en la página web de Made with Code.

Fuente: (Google Company, 2016)

- Kaleidoscope (*Figura 246*):

Se trata de crear un caleidoscopio con una temática a elegir, configurando ciertos parámetros como el tamaño de los dibujos, y la velocidad a la que se mueven.

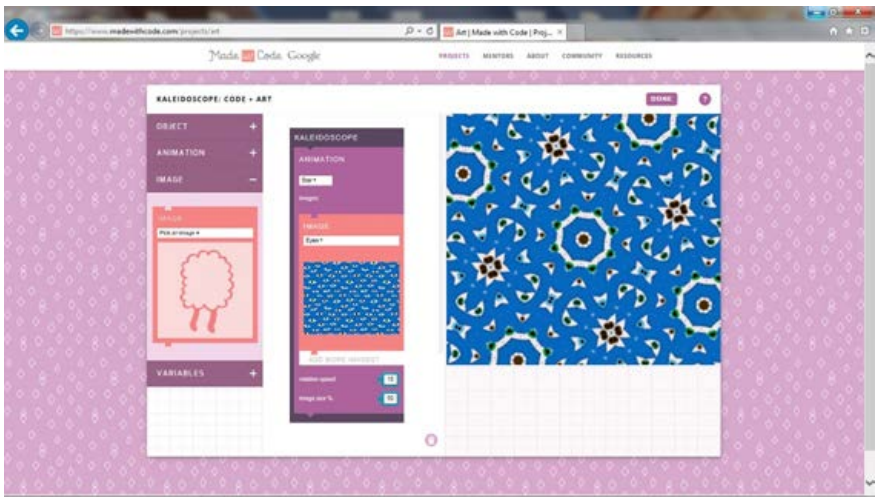


Figura 246. Juego "Kaleidoscope" en la página web de Made with Code.

Fuente: (Google Company, 2016)

- Music Mixer *(Figura 247):*

Se trata de crear una escena a base de música y sonidos, que generarán unos dibujos animados según se configuren.

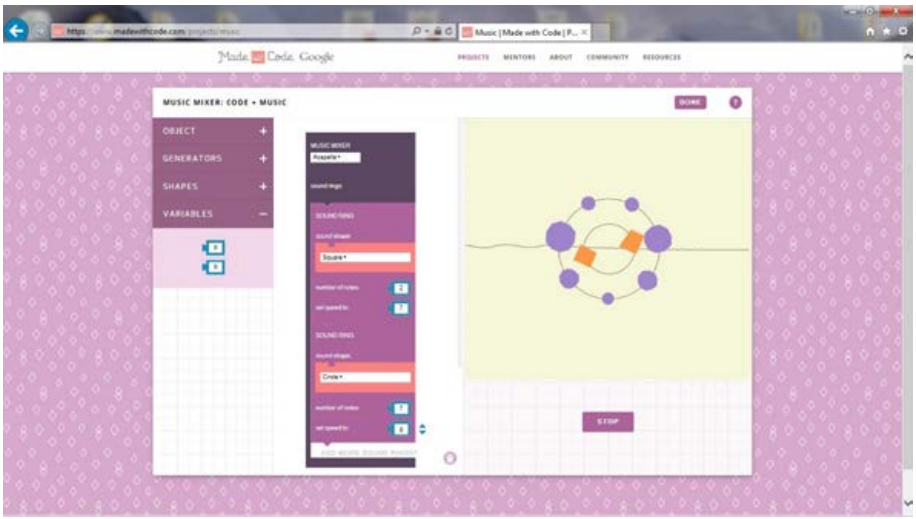


Figura 247. Juego "Music Mixer" en la página web de Made with Code.

Fuente: (Google Company, 2016)

- Beats (*Figura 248):*

Similar al anterior, en este caso la escena se construye con la reproducción de notas musicales, interpretadas por distintos instrumentos.

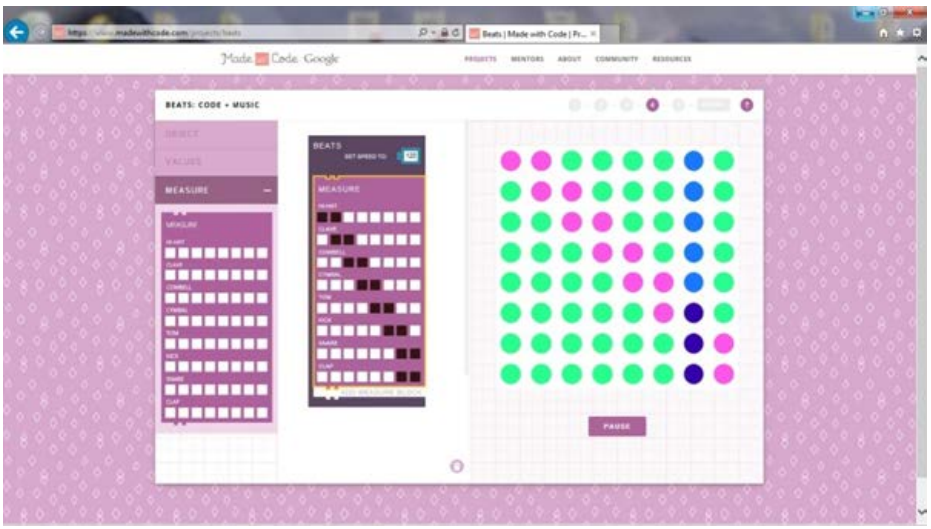


Figura 248. Juego "Beats" en la página web de Made with Code.

Fuente: (Google Company, 2016)

- Avatar *(Figura 249):*

Este juego consiste en colocar una serie de figuras geométricas dentro de los ejes cartesianos, para componer un dibujo. Esto se realizará mediante un código programado que modificará la posición, el tamaño y el color de las figuras.

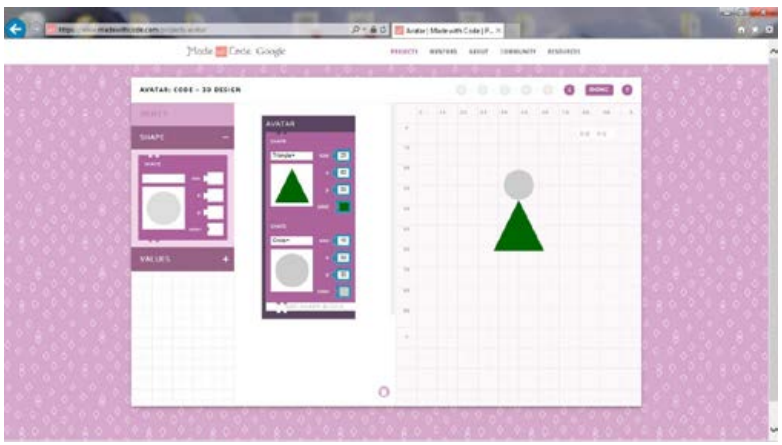


Figura 249. Juego "Avatar" en la página web de Made with Code.

Fuente: (Google Company, 2016)

- Garden Robot (*Figura 250*):

Consiste en una serie de minijuegos en los que el usuario tendrá que utilizar dos tipos de comando: *move right* (mover a la derecha) y *add water to pot* (añadir agua a la jarra) para ir rellenando unas macetas con agua, y conseguir que crezcan sus flores. Un robot ejemplificará visualmente nuestras acciones. Consta de un total de 5 niveles, en los que habrá que calcular las cantidades de agua. A partir del nivel 3, será posible utilizar el comando *do while*, que permite la construcción de bucles.

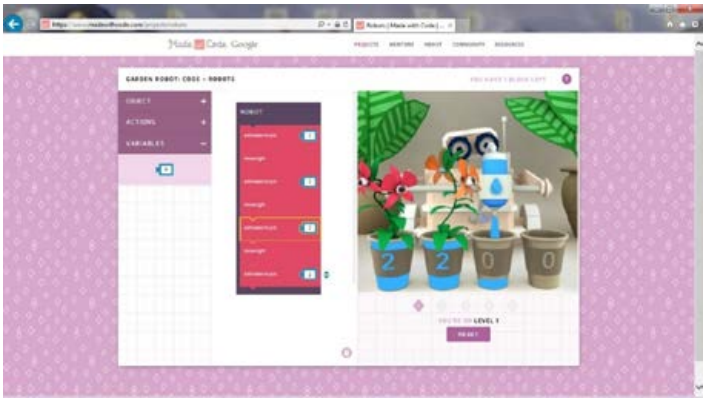


Figura 250. Juego "Garden Robot" en la página web de Made with Code.

Fuente: (Google Company, 2016)

- Accesorizer *(Figura 251):*

Consiste en añadir accesorios a una imagen seleccionada, y controlar a través de la programación sus parámetros, para que se ajusten según el resultado deseado.

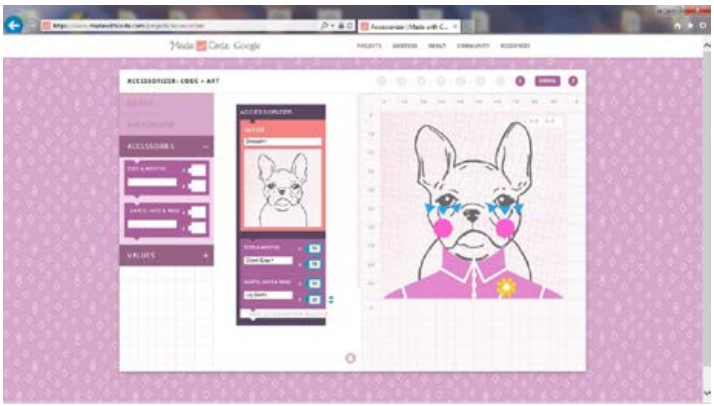


Figura 251. Juego "Accessorizer" en la página web de Made with Code.

Fuente: (Google Company, 2016)

- Dance *(Figura 252):*

Se trata de animar a una bailarina, a través de sentencias de programación.

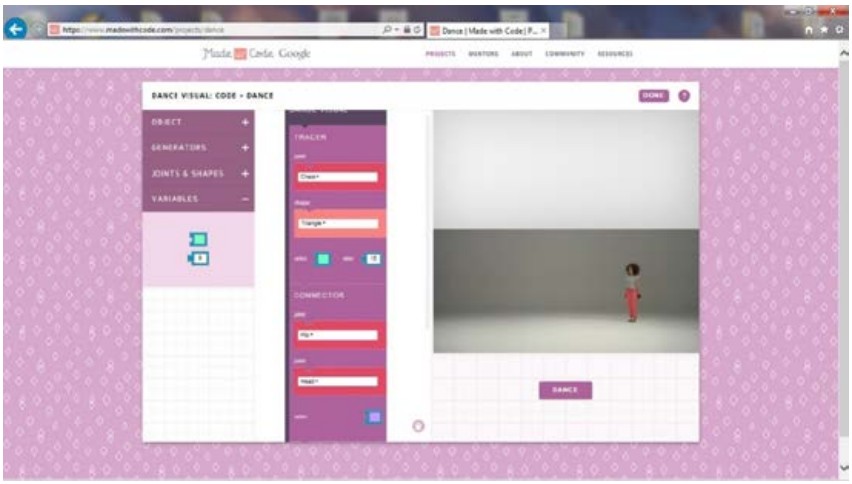


Figura 252. Juego "Dance" en la página web de Made with Code.

Fuente: (Google Company, 2016)

Made with Code tiene un espacio donde se puede compartir los proyectos, o acceder los realizados por otros usuarios para modificarlos. También dispone de abundante documentación.

Entre estos juegos, de los que hablaremos más adelante, hay que destacar un punto negativo. La ayuda que ofrece la página es excesiva, llegando incluso a ser pesada, dificultando la concentración del usuario en los ejercicios.

32. Mama

Tabla 157
Resumen de características de la herramienta Mama

Nombre de la herramienta	Mama
Página web	http://www.eytam.com/mama
URL de descarga	http://www.eytam.com/mama
Fecha de creación / aparición en el mercado	22 de febrero del año 2010 (una versión estable de la herramienta)
Última versión en 2017 / año de esa versión	Mama 2.0, versión del año 2015
Plataformas en las que se puede instalar	Windows (XP o superior)
Tipo de software	Privativo
Precio / modelo de negocio	Gratuito
Edades para las que está indicado	Desde los 8 años
Aspectos educativos que trabaja	Conceptos básicos de la programación y la programación orientada a objetos, creatividad y planificación.
Característica más destacada / valor diferencial	Permite el acceso al código y es una herramienta gratuita y accesible

Mama es una herramienta construida sobre las bases de Alice. Permite crear videojuegos e historias interactivas, combinando elementos gráficos 2D y 3D, con JavaScript, un lenguaje que sigue el paradigma de la programación orientado a objetos.

La interfaz de Mama presenta el aspecto que se muestra en la *Figura 253*, y cuyos elementos se describen a continuación:

Situada en la parte superior, está la barra de herramientas básicas del programa, donde se encontrarán botones de uso común, como *Play* (jugar) o Deshacer/Rehacer, Insertar Personaje, Insertar Imágenes, exportar a YouTube, crear ejecutable, Insertar objeto 3D, o acceder a la Ayuda del programa.

En la zona superior izquierda, bajo la barra de herramientas, se muestra un esquema, con la lista de todos los objetos incluidos en el escenario el escenario.

En la zona inferior izquierda, debajo de la anterior, se encuentra a que podríamos llamar Área de Detalles, donde se muestra la información relativa al objeto que se seleccione. Esta información se muestra organizada en tres categorías: propiedades, métodos y funciones.

En el marco central, en un área de mayor extensión, se encuentra el espacio de visualización de la escena, donde se agregarán los distintos elementos que la compongan. En este espacio también se puede acceder a las opciones de movimiento, rotación y angulación de los mismos.

A la derecha se encuentra el denominado *Events Area* (Área de Eventos), donde se determina qué acción se ejecuta como respuesta a un suceso dentro del programa.

En la parte inferior se encuentra el área de codificación, donde se escribe el programa a desarrollar, arrastrando las instrucciones disponibles en un orden concreto para obtener los resultados esperados.

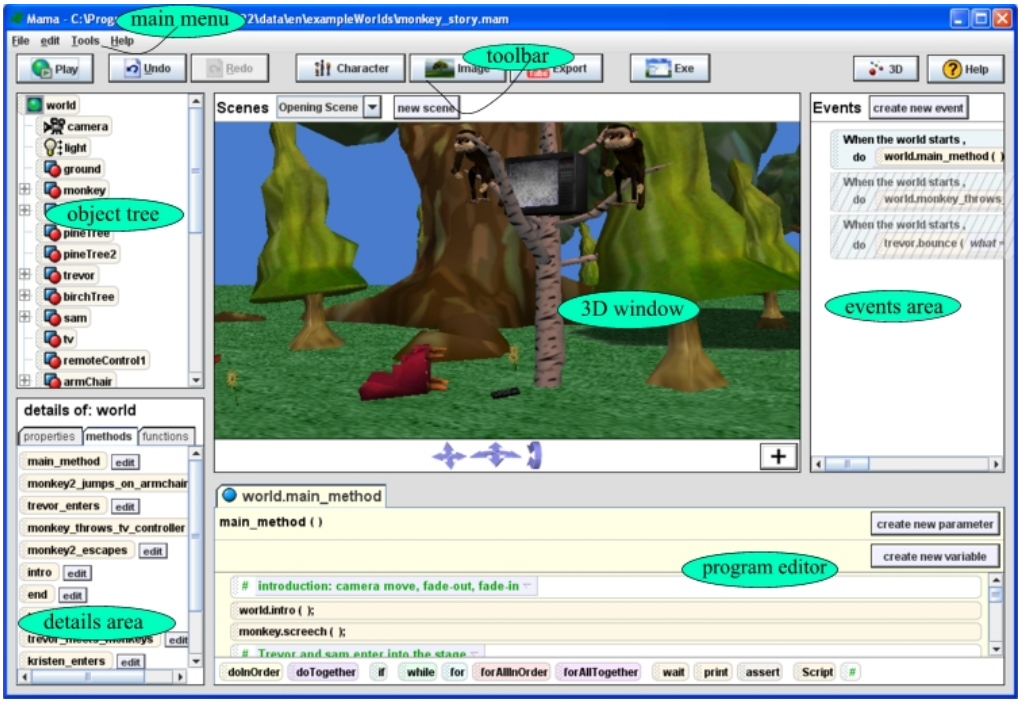


Figura 253. Interfaz de Mama

Fuente: (Shapes Robotics, 2016b)

La herramienta está traducida al inglés y al hebreo. También cuenta con varios tutoriales, y documentación de ayuda en su página web que explica cómo realizar ciertos procesos, como por ejemplo, cómo crear una animación, y publicarla en YouTube,

Para comprobar la similitud Mama con la herramienta de Alice, a continuación se incluye una imagen de la interfaz de esta última.

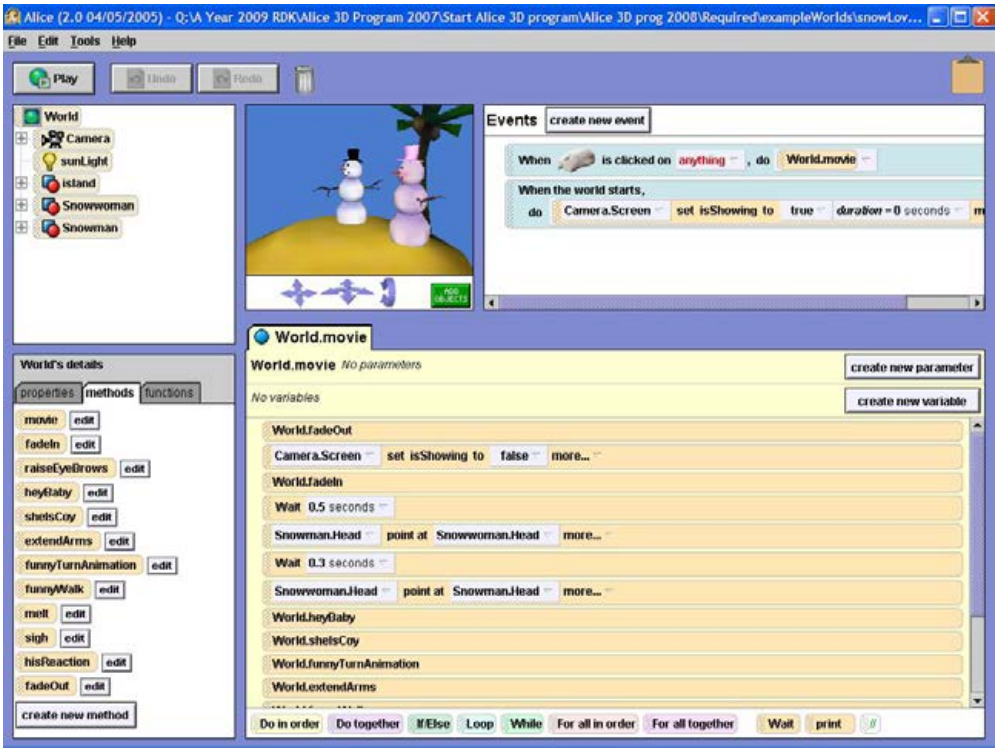


Figura 254. Un proyecto en Alice 2.0.

Fuente: (Pau, 2011)

33. Minecraft

Tabla 158
Resumen de características de la herramienta Minecraft

Nombre de la herramienta	Minecraft
Página web	https://minecraft.net/es-es/
URL de descarga	https://minecraft.net/es-es/store/?ref=m
Fecha de creación / aparición en el mercado	18 de noviembre del año 2011
Última versión en 2017 / año de esa versión	Versión descargable de la página del año 2017
Plataformas en las que se puede instalar	Windows, Mac, Linux(compatibles con Java), Windows Phone, Android, iOS, Xbox 360, Xbox One, PlayStation 3, PlayStation 4, PlayStation Vita, Raspberry Pi, Wii U
Tipo de software	Privativo
Precio / modelo de negocio	Pago único de 23,95 €
Edades para las que está indicado	Desde los 7 años
Aspectos educativos que trabaja	Creatividad, competencias sociales y digitales
Característica más destacada / valor diferencial	La posibilidad de crear mundos únicos y mecanismos complejos a partir de materiales obtenidos en el propio juego de manera "simple"

Minecraft (*Figura 255*) es un videojuego con gráfica 3D basada en vóxeles[57], programado con el lenguaje JavaScript. Se trata de controlar a un personaje que intentará sobrevivir en un mundo abierto ante los ataques de otros jugadores, o de monstruos controlados por el sistema, consiguiendo recursos y diferentes materiales, que utilizará para construir herramientas y edificaciones. Los recursos obtenidos se podrán almacenar en un inventario, o refinar en un área de trabajo como paso previo a su almacenaje o utilización.

Fundamentalmente se trata de un videojuego, y dentro de él lo único que podríamos relacionar con la programación son los mods (modificadores) y los redstone (piedra roja).

Redstone es el material que funciona como combustible o energía, y es un elemento indispensable para crear artefactos o construcciones. Para crear estos artefactos o estas construcciones, el jugador puede combinar recursos, en una cantidad y un orden concreto. Por ejemplo, combinando hierro y carbón, se consigue acero. Combinando acero y madera, se elabora un martillo.

[57] Un vóxel es la representación tridimensional de un pixel. Básicamente podríamos decir que es una figura geométrica tridimensional con forma de cubo.

Figura 255. Imagen promocional de Minecraft en su web.

Fuente: *(Mojang, 2017)*

Con un martillo, se pueden hacer construcciones complejas, etc. Existen un sinfín de combinaciones, y con ellas se trabajar el pensamiento algorítmico. Podemos ver un ejemplo de estas construcciones en *la Figura 256.*

En la *Figura 257* podemos ver un esquema donde se explica la lógica de combinación de materiales y objetos, que se debe seguir para construir determinados artefactos. Por ejemplo, la *Figura 256* muestra unas puertas automáticas, que se abren cuando se transmite energía a través del terreno, al pasar por una placa de presión que activa un *switch* (intercambiador). Para que esto sea posible, se ha tenido que seguir el esquema de la *Figura 257*, que guarda cierta similitud con los mismos principios de la programación de circuitos.

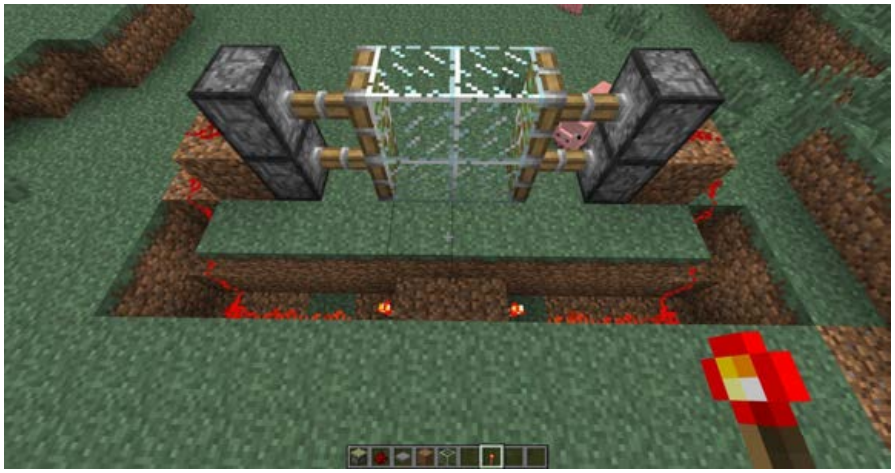


Figura 256. Un ejemplo de una construcción utilizando *redstone* en Minecraft.

Fuente: *(Taringa, 2016)*

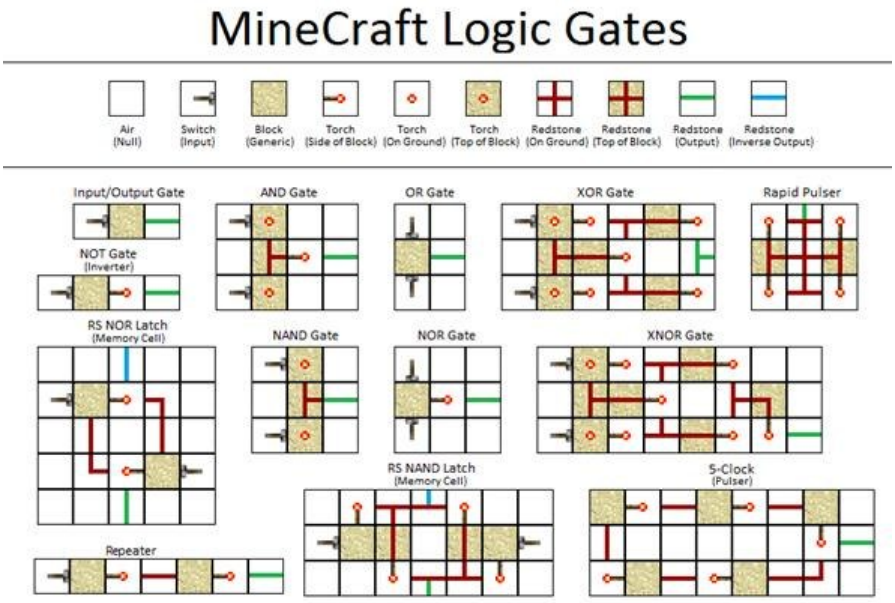


Figura 257. Reglas de uso del material redstone en Minecraft.

Fuente: *(Mojang, 2017)*

Los *mods* son *pluggins* que se instalan como extras al programa, y que ofrecen un plus de jugabilidad, al añadir funcionalidades y nuevas posibilidades de combinación de elementos. Si se conoce el lenguaje de JavaScript, se pueden crear nuevos *mods*.

En el aspecto creativo, existen multitud de ejemplos de jugadores que han creado dentro del juego construcciones realmente complejas.
A pesar de todas las posibilidades que ofrece, Minecraft no posee una guía oficial para la creación de inventos, ni para el desarrollo de *mods.* La mayoría de los recursos y ayuda existentes han sido generados por su amplia comunidad de usuarios.

34. MIT App Inventor

Tabla 159
Resumen de características de la herramienta MIT App Inventor

Nombre de la herramienta	MIT App Inventor
Página web	http://appinventor.mit.edu/
URL de descarga	http://appinventor.mit.edu/explore/get-started.html
Fecha de creación / aparición en el mercado	Año 2009 / 12 de julio del año 2010
Última versión en 2017 / año de esa versión	Versión de la web del año 2017
Plataformas en las que se puede instalar	Chrome, Firefox, Safari, Opera, IE, Android e iOS
Tipo de software	Libre
Precio / modelo de negocio	Gratuito
Edades para las que está indicado	Mayores de 12 años
Aspectos educativos que trabaja	Concepto básicos de la programación
Característica más destacada / valor diferencial	Puede utilizarse en casi cualquier operador web de forma gratuita

MIT App Inventor (*Figura 258*) es una herramienta creada por el MIT en colaboración con Google, que permite desarrollar aplicaciones a través de un sistema de programación basado en bloques con comportamiento predefinido.

Esta herramienta requiere de una cuenta de Google para poder utilizarse.

En su interfaz podemos encontrar diversos apartados. Uno de ellos, *Projects* (Proyectos), da acceso a iniciar un nuevo proyecto, continuar con uno existente, o compartirlo con otros usuarios. Para comenzar un nuevo proyecto, se deberá pulsar en *Start New Project* (comenzar nuevo proyecto) (*Figura 259*).

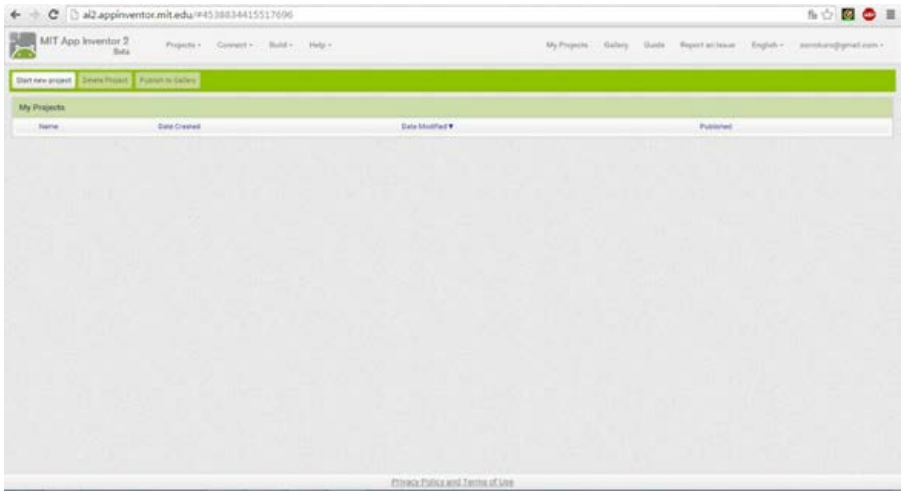


Figura 258. Página inicial de App Inventor.

Fuente: *(MIT, 2016b)*

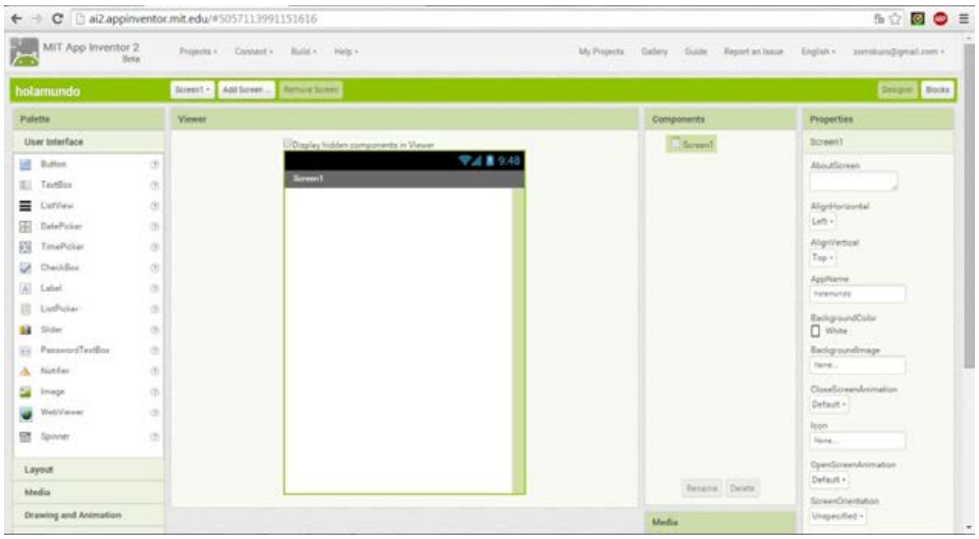


Figura 259. Nuevo proyecto en App Inventor.

Fuente: *(MIT, 2016b)*

Cundo se crea un proyecto, se trabaja sobre un espacio central que simula la pantalla de un teléfono móvil, en un panel denominado *Designer* (diseño). Sobre este espacio se añadirán los elementos de interfaz (botones, imágenes, cuadros de texto, etc.) que se quiera tener en la aplicación que se está creando. También se pueden agregar elementos multimedia (vídeos, animaciones, sonidos, etc.).

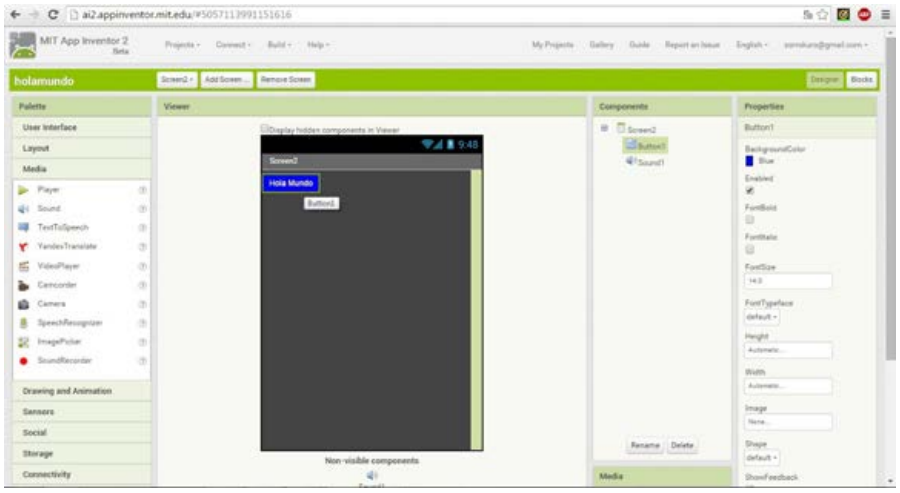


Figura 260. Previsualización de un proyecto en App Inventor.

Fuente: *(MIT, 2016b)*

Cada elemento que se agregue, dispondrá de una serie de propiedades que permitirán configurarlo. Dichas propiedades aparecerán en un panel a la derecha de la pantalla, al seleccionar un determinado elemento.

Dentro de la herramienta hay otro panel, denominado *Blocks* (*Figura 261).* El panel *Blocks* y el panel *Designer* se muestran de forma alternativa (cuando se selecciona uno, desaparece el otro). Será en este panel *Blocks* donde se codifique el programa que implementará la funcionalidad de la aplicación que se está creando, a través de un sistema de programación por bloques, similar al que utiliza Blockly.

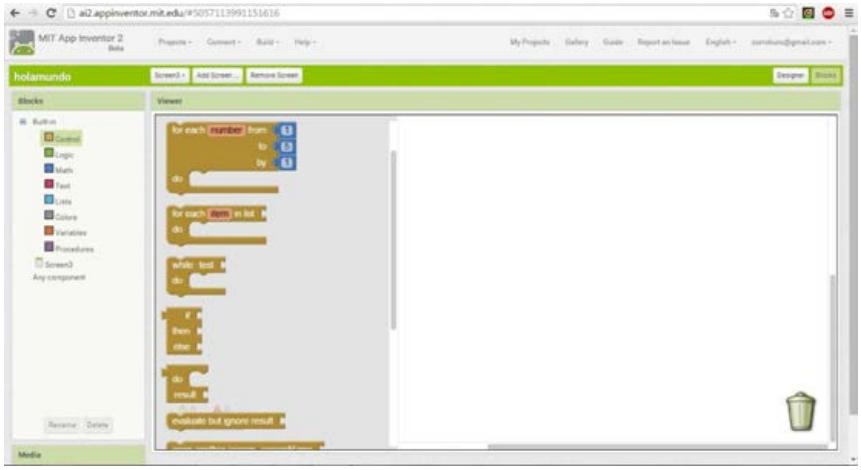


Figura 261. Pestaña “Blocks” en App Inventor.

Fuente:*(MIT, 2016b)*

MIT App Inventor permite compartir las aplicaciones realizadas con otros usuarios, así como acceder a ellas, valorar su calidad, modificarlas, etc. También cabe mencionar que está traducida a diversos idiomas.

35. Project Spark

Tabla 160
Resumen de características de la herramienta Project Spark

Nombre de la herramienta	Project Spark
Página web	https://www.microsoft.com/es-es/store/p/project-spark/9wzdncrfhw95#
URL de descarga	https://www.microsoft.com/es-es/store/p/project-spark/9wzdncrfhw95
Fecha de creación / aparición en el mercado	5 de octubre del año 2015
Última versión en 2017 / año de esa versión	Versión actualizada de Project Spark antes del cierre de sus servidores el 12 de agosto del año 2016
Plataformas en las que se puede instalar	Xbox One y Windows 10
Tipo de software	Privativo
Precio / modelo de negocio	Gratuito, pero el juego tenía micropagos dentro del mismo
Edades para las que está indicado	Mayores de 7 años
Aspectos educativos que trabaja	Creatividad, estructuración, conceptos básicos de la programación
Característica más destacada / valor diferencial	La gran variedad y personalización que permitía, además de la gran cantidad de contenido emergente que podía crearse

Project Spark es una herramienta que combina el aspecto lúdico, con la funcionalidad de un motor de desarrollo de videojuegos.

Figura 262. Creando un nivel desde cero en Project Spark.

Fuente: *(Team Dakota, 2015)*

Los videojuegos que Project Spark posibilita crear, son almacenados en su entorno online, permitiendo a toda su comunidad de usuarios ejercer tanto de creadores como de jugadores. En la *Figura 262* se puede ver la creación de un nivel, utilizando las diferentes opciones que proporciona la herramienta.

Project Spark tiene dos modos de uso:

- *Play Mode* (Modo Jugar): esta modalidad a su vez se subdivide en otras tres:
    - *Community Games* (Juegos de la Comunidad), da opción a jugar a niveles ya creados por otros usuarios. El sistema clasifica estos niveles por categorías, según su temática, dificultad, etc.
    - *Episodic Adventures* (Aventuras Episódicas), permite jugar a juegos donde se narra una historia relacionada con el universo del juego.
    - *Crossroads (Caminos cruzados),* es un modo de juego híbrido entre *Create Mode*, que se verá a continuación, y los descritos anteriormente.

- *Create Mode* (Modo Crear), es el modo que estaría relacionado con el ámbito de la programación, y el que se analizará en este apartado con más nivel de detalle. *Create Mode* da la opción de crear un nuevo proyecto, o de continuar con un nivel/juego creado anteriormente.

Cuando se crea un nuevo proyecto, se puede elegir entre una de las siguientes opciones:

Start from scratch *(Comenzar desde un boceto)*, es el modo más personalizable. Permite a los usuarios empezar un proyecto desde cero, dando la posibilidad de escoger biomas, colocar props, e insertar los llamados *brains* (cerebros) que confieren inteligencia artificial a los objetos, según la configuración que el usuario decida.

Figura 263. Modificando un nivel con la herramienta "Pintura" en Project Spark.

Fuente: *(Team Dakota, 2015)*

Learn to Create (Aprender a Crear), tiene la misma interfaz que *Start From Scratch*, solo que se muestra cada acción a realizar paso a paso mediante tutoriales.

Figura 264. Colocando y editando objetos en un proyecto en Project Spark.

Fuente: *(Team Dakota, 2015)*

Project Spark hace uso del *Kodu Visual Programming Toolkit,* una de las herramientas analizadas anteriormente, para todo lo que tiene que ver con programación.

Otra herramienta de programación dentro de Project Spark la encontramos en el denominado *Brain System* (Sistema de Cerebros), un panel que permite configurar el comportamiento de los mencionados *brains*. Cuando se crea un nuevo *brain*, en

el *Brain System* aparecen dos parámetros etiquetados como *when y do* (cuando y hacer). Al seleccionar cada uno de ellos, se despliega una ruleta con las opciones que se pueden asociar a cada uno. A *when* se asociarán eventos que pueden suceder, y a *do* las acciones a realizar en respuesta a dicho evento. Por ejemplo, si un personaje del juego se topa con una pared (*when*), cambia el sentido de su trayectoria (*do*).

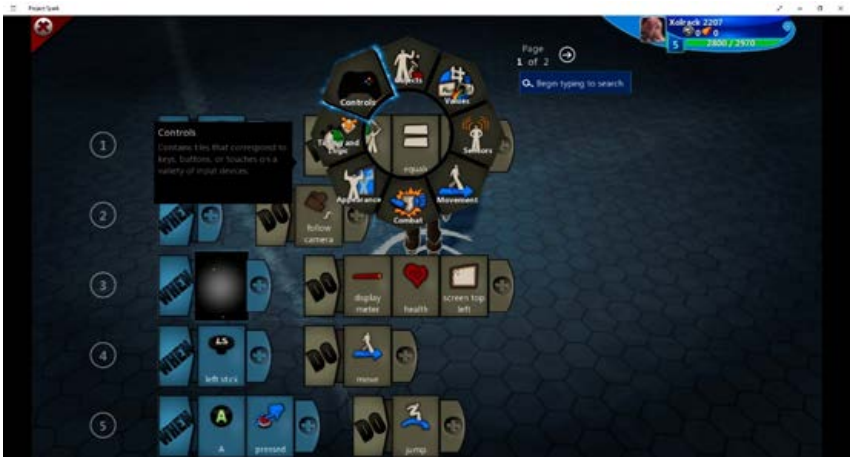


Figura 265. Programando los controles de un personaje en Project Spark.

Fuente: *(Team Dakota, 2015)*

El manejo del juego se puede hacer tanto con un mando de consola, como con teclado y ratón. Si se utiliza desde un *smartphone* o *tablet*, el dedo del usuario cumple con la función del ratón.

Project Spark está diseñado para que tanto usuarios noveles como expertos puedan crear proyectos de distintos niveles de complejidad, aunque una personalización total requerirá del uso de código textual, y estará fuera del alcance de aquellos que no tengan conocimientos avanzados de programación.

Uno de sus puntos débiles está en la imposibilidad de exportar un proyecto fuera de la herramienta, es decir, los proyectos creados sólo se pueden utilizar dentro del entorno de Project Spark. Otro aspecto negativo es que no se pueden incluir elementos gráficos propios (imágenes, modelos 3D, texturas, etc.), cosa que sí se puede hacer en un motor de videojuegos.

36. Python Turtle

Tabla 161
Resumen de características de la herramienta Python Turtle

Nombre de la herramienta	Python Turtle
Página web	http://pythonturtle.org/
URL de descarga	http://pythonturtle.org/
Fecha de creación / aparición en el mercado	Año 1986 / Año 2009
Última versión en 2017 / año de esa versión	Python Turtle versión 0.1.2009.8.2.1, del año 2015
Plataformas en las que se puede instalar	Windows, MAC OS
Tipo de software	Privativo
Precio / modelo de negocio	Gratuito
Edades para las que está indicado	Mayores de 12 años
Aspectos educativos que trabaja	Conceptos básicos sobre la programación, estructuración
Característica más destacada / valor diferencial	Sencillez del lenguaje

PythonTurtle es una herramienta con la que los usuarios podrán programar el comportamiento de una pequeña tortuga, que en su recorrido, dejará un rastro de colores, con el que dibujará diversas figuras. Con esta descripción, podríamos estar hablando de Logo. De hecho PythonTurtle está inspirado en él, con la salvedad de que el lenguaje con el que se programa es Python.

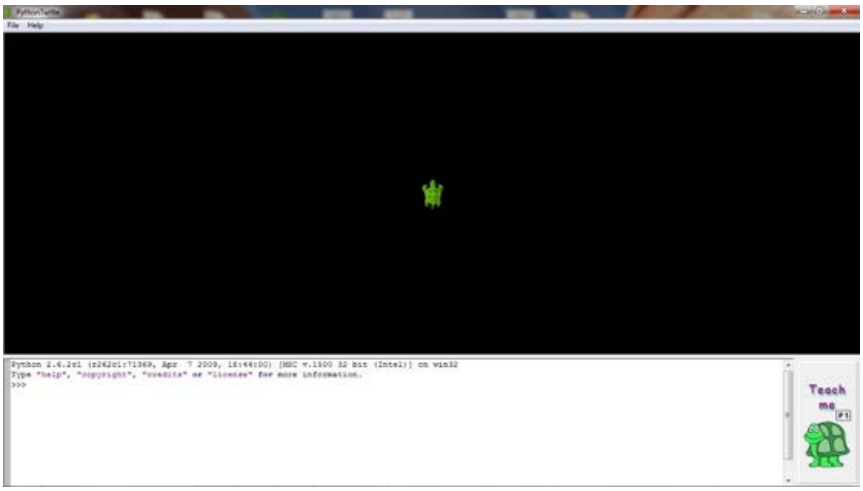


Figura 266. Pantalla inicial de Python Turtle.

Fuente: *(Rachum, 2015)*

Hay que destacar que la información disponible sobre esta herramienta es escasa. El contenido de su página web se limita a una breve descripción, y a mostrar los enlaces de descarga de los instaladores para las distintas plataformas.

Dentro de la herramienta existen una serie de ejercicios, que se explican paso a paso. Si se pulsa la tecla F1, se accederá a este tutorial.

Además de la sintaxis de Python, el programa utiliza comandos propios, como *turn* (girar), que hace girar a la tortuga en el sentido de las agujas del reloj. Por ejemplo, *turn* (270) hará que gire hacia la izquierda. Para hacer esta misma acción también se podría poner el comando *turn*(-90).

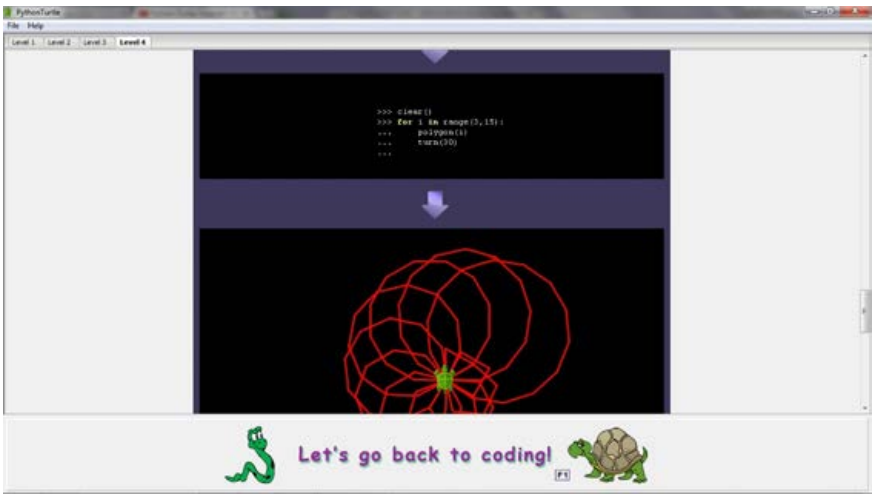


Figura 267. Un ejemplo de código para girar el puntero en Python Turtle.

Fuente: *(Rachum, 2015)*

El entorno de programación dificulta la tarea resolución de errores. Por ejemplo, si se produce un error de compilación, para corregirlo, habrá que cerrar y abrir de nuevo el programa. Además, una vez compilado el código, no se podrá modificar, y todo lo que se añada como nuevo, se hará a partir de la última posición compilada.

En la *Figura 268* se muestra una captura de uno de los ejercicios guiados propuestos por la herramienta, junto con el código que lo soluciona.

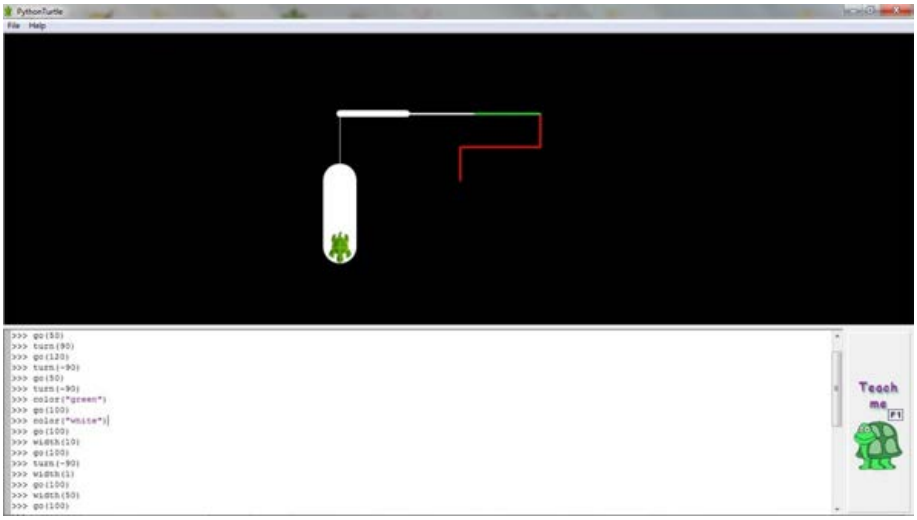


Figura 268. Nivel 1 de Python Turtle completado con código.

Fuente: *(Rachum, 2015)*

37. RoboMind

Tabla 162
Resumen de características de la herramienta RoboMind

Nombre de la herramienta	RoboMind
Página web	http://www.robomind.net/es/
URL de descarga	http://www.robomind.net/en/download.html
Fecha de creación / aparición en el mercado	Año 2005
Última versión en 2017 / año de esa versión	Depende de la plataforma, del año 2016. Windows: RoboMind 6.0.1 MAC OS: RoboMind 5.3 Linux: 6.0.1
Plataformas en las que se puede instalar	Windows (XP o superior), MAC OS (X o superior) y Linux
Tipo de software	Privativo
Precio / modelo de negocio	Pago anual (7 – 14$ al año), aunque cuenta con varias posibilidades para estudiantes y cuentas personales
Edades para las que está indicado	Edades entre los 6 – 18 años
Aspectos educativos que trabaja	Competencias digitales, conceptos básicos de programación
Característica más destacada / valor diferencial	Se trata de una herramienta muy accesible, barata y dividida por dificultades según las necesidades del usuario

RoboMind es una herramienta para la enseñanza de conceptos básicos de la programación y la automatización de procesos, dirigido a estudiantes sin formación previa en esta rama.

RoboMind utiliza un lenguaje de programación similar al de la herramienta WinLogo, que también se analiza en este capítulo. Asimismo, RoboMind guarda cierta gran similitud con *Star Wars: Building a Galaxy with Code*, para web, otra herramienta que también será analizada más adelante en este documento.

RoboMind contiene gran cantidad de ejercicios de dificultades diversas. La mecánica general de estos ejercicios consiste en mover a un pequeño robot por un escenario, realizando diferentes caminos hasta llegar a un lugar determinado.

Cada escenario plantea retos concretos. Para resolverlos, el robot, mientras recorre el camino hacia la meta, tendrá que realizar diversas acciones (pintar el recorrido por donde pasa, escribir algo en el escenario, mover un objeto de sitio, etc.).

La interfaz del programa tiene tres zonas claramente diferenciadas (*Figura 269* y *Figura 270*): el área de programación (en la parte de la izquierda), el área de visualización (en la parte derecha) y la barra de funciones (en la parte baja).

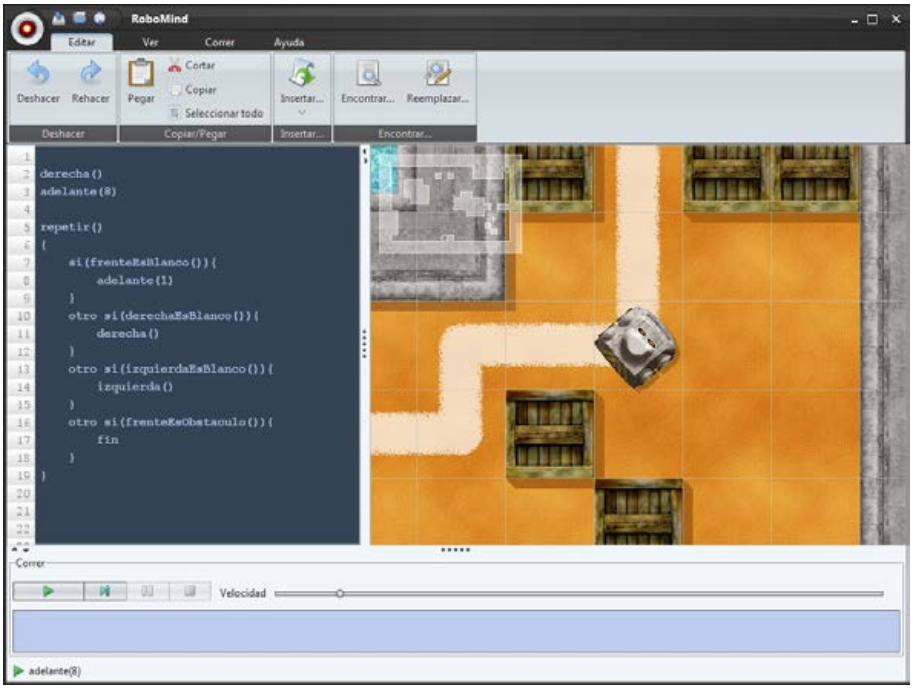


Figura 269. Visualización de un nivel en RoboMind.

Fuente: *(Kitchen, 2015)*

En el área de la programación se codificará las acciones a realizar por el robot.

La barra de funciones contiene el botón *Play* (Jugar), *Stop* (Pausar) y *Reset* (Reiniciar), además de una barra que controla la velocidad de ejecución del programa.

En el área de visualización, podremos observar el resultado del programa creado, una vez se pulse el botón *Play*.

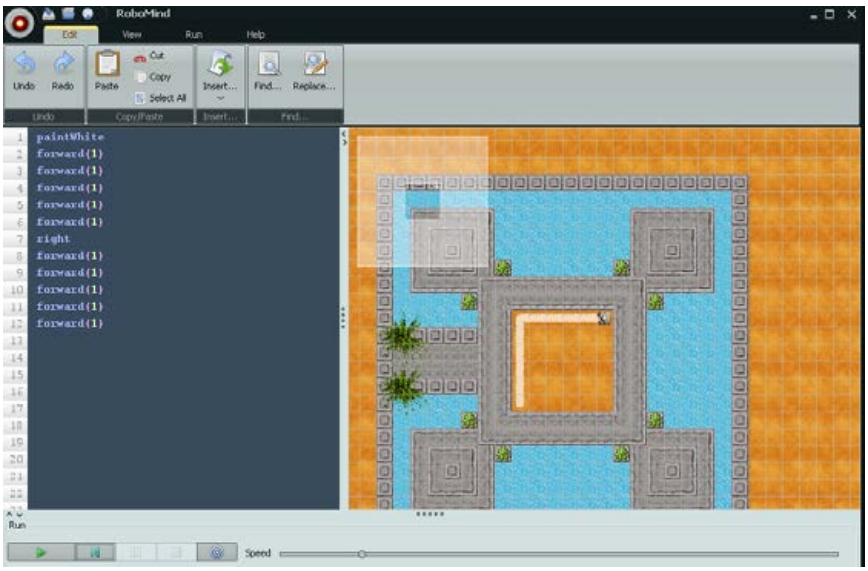


Figura 270. Un nivel en RoboMind.

Fuente: (Kitchen, 2015)

Una de sus mayores ventajas es que está traducido a 24 idiomas. Además, RoboMind se puede conectar con los robots de Lego Mindstorms, y utilizar su entorno para programar el comportamiento de robots físicos.

Como inconvenientes, se puede decir que los aspectos de programación se limitan a hacer mover el robot. También, que el lenguaje de programación que utiliza es propio, con lo que su aprendizaje no es extrapolable a otros entornos.

38. RPG Maker

Tabla 163
Resumen de características de la herramienta RPG Maker

Nombre de la herramienta	RPG Maker
Página web	http://www.rpgmakerweb.com/
URL de descarga	http://www.rpgmakerweb.com/
Fecha de creación / aparición en el mercado	Agosto del año 2015
Última versión en 2017 / año de esa versión	Versión actualizada del RPG Maker desde su fecha de lanzamiento hasta el día de hoy
Plataformas en las que se puede instalar	Windows y MAC
Tipo de software	Privativo
Precio / modelo de negocio	Existen versiones gratuitas (30 días) y de pago (80$)
Edades para las que está indicado	No está especificada
Aspectos educativos que trabaja	Conceptos básicos de la programación, creatividad
Característica más destacada / valor diferencial	Posibilidad de crear videojuegos propios y acceso al código

RPG Maker es una herramienta de desarrollo, especialmente indicada para implementar juegos de rol.

Todas sus versiones incluyen un amplio set de recursos, como gráficos para la creación de mapas y personajes, música, efectos de sonido, etc. Además, cuenta con una enorme comunidad de usuarios, que comparten material de todo tipo, apto para ser utilizado dentro de RPG Maker. También existe en la red gran cantidad de tutoriales y guías de aprendizaje de esta herramienta.

La versión que se analiza en este apartado es RPG Maker MV, publicada en agosto de 2015. Dicha versión trajo consigo varias novedades, entre las que destaca su capacidad para exportar los proyectos creados a plataformas como MAC, iOS o Android.

Cuando se inicia un nuevo proyecto, el programa cargará varios archivos predeterminados (escenarios, personajes, etc.), que se mostrarán organizados por categorías en un panel situado a la izquierda da la interfaz, bajo la barra de herramientas. Debajo de este panel, también a la izquierda, se encuentra un esquema donde se muestran los distintos escenarios que contiene nuestro proyecto. En la parte derecha se muestra un personaje, que también se podrá editar. En la parte central se muestra el escenario que se esté creando en ese momento.

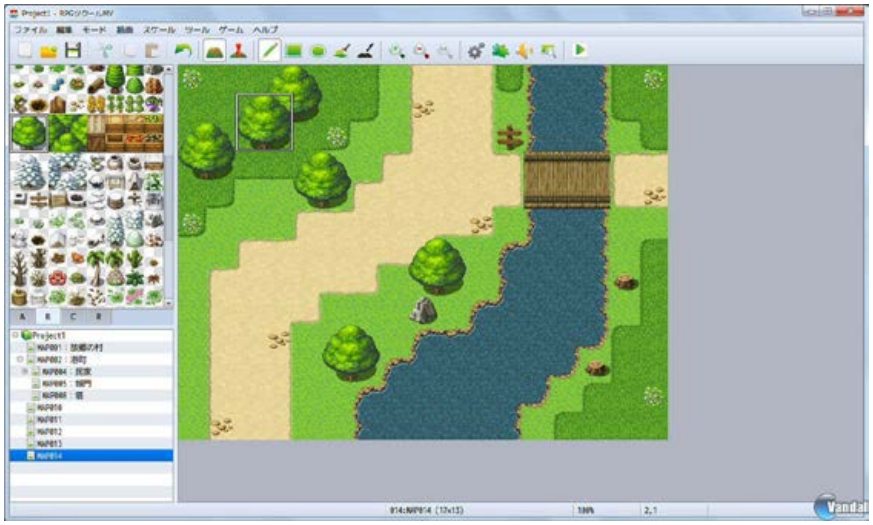


Figura 271. Creación de un escenario y colocación de sus elementos en RPG Maker V.

Fuente: (Varela, 2015)

Sobre el escenario se arrastrarán los objetos que queremos que contenga. Es posible hacer clic con el botón derecho del ratón sobre un objeto ya colocado, para configurar su comportamiento (por ejemplo, definir una respuesta ante un determinado evento).

En la *Figura 272* se muestra el panel que se despliega al seleccionar la opción Tools (Herramientas) en la barra de herramientas. A través de este panel se podrá acceder a la base de datos del programa, para modificar los objetos almacenados (su representación gráfica, descripción, tipo, animaciones, etc.), o para crear objetos nuevos.

Figura 272. Panel *Tools* de la herramienta RPG Maker

Fuente: (Varela, 2015)

Si se pulsa sobre un personaje, se podrá programar sus comportamientos, sus diálogos, y los eventos a los que responde, a través de bloques de código.

Los proyectos finalizados se pueden exportar a diversas plataformas, por lo que esta herramienta es muy utilizada en el ámbito profesional, para la creación de juegos de rol. Si se quiere crear juegos de otro género, RPG Maker no es la opción más adecuada.

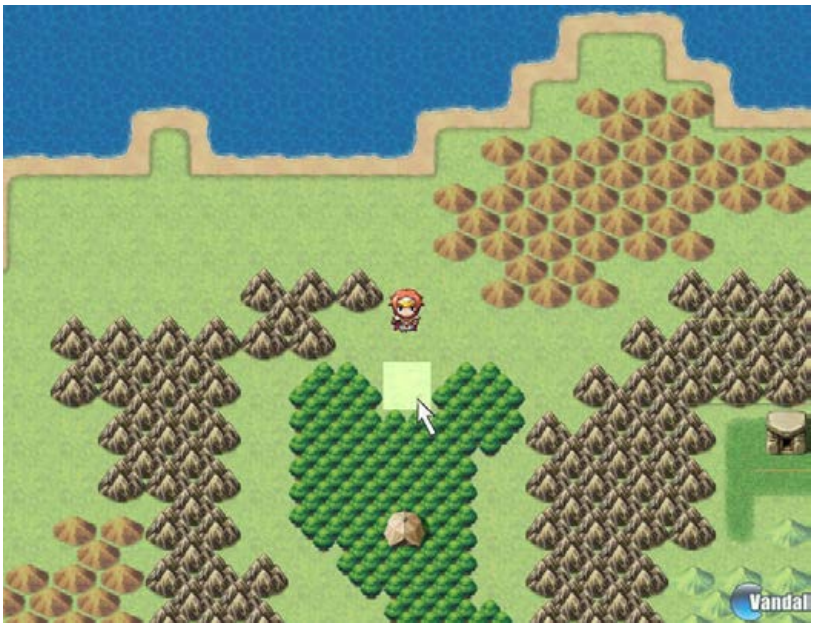


Figura 273. Proyecto terminado en RPG Maker V.

Fuente: (Varela, 2015)

39. Scratch

Tabla 164
Resumen de características de la herramienta Scratch

Nombre de la herramienta	Scratch
Página web	https://scratch.mit.edu/
URL de descarga	https://scratch.mit.edu/scratch2download/
Fecha de creación / aparición en el mercado	Año 2007 – 2013 (versión 2.0)
Última versión en 2017 / año de esa versión	Versión web del día 6 de marzo del año 2017
Plataformas en las que se puede instalar	Funciona a través de la web, pero también cuenta con una versión *offline* (sin necesidad de conexión) en varias plataformas: Windows, MAC, Linux, Android
Tipo de software	Privativo
Precio / modelo de negocio	Gratuito
Edades para las que está indicado	Edades desde 8 – 16 años
Aspectos educativos que trabaja	Conceptos de la programación orientada a objetos, conceptos básicos de la programación orientada a objetos, estructuración, resolución de problemas, creatividad
Característica más destacada / valor diferencial	La posibilidad de crear y compartir proyectos propios con otros usuarios, utilizando la web como único requisito

Esta herramienta es ampliamente analizada en el capítulo 6 de esta tesis. A continuación se reproduce un resumen de lo que se expone en el mencionado capítulo.

Scratch es un proyecto del MIT (*Massachusetts Institute of Technology*), concretamente del grupo de investigación *Lifelong Kindergarten* del laboratorio de medios del MIT en colaboración con el grupo de Yasmin Kafai en UCLA. Está diseñado para enseñar a programar, mediante la creación de videojuegos, historias animadas, y presentaciones interactivas, a estudiantes en la horquilla de edades entre los 8 y los 16 años. Scratch es posiblemente la herramienta de aprendizaje de programación más extendida en el mundo, utilizándose en más de 150 países (MIT, 2013).

Scratch sigue la estela de otros entornos de programación y lenguajes que le han precedido, dirigidos a programadores principiantes (Guzdial, 2004; Kelleher y Pausch, 2005).

El entorno de Scratch permite crear fácilmente animaciones y elementos interactivos. Esto se consigue mediante una interfaz gráfica con un diseño HCI (*Human Computer Interaction*) clásico mediante pantalla, teclado y ratón. Este diseño facilita que cualquier persona que sepa manejar un Sistema Operativo

moderno (Windows, Mac OS, GNU-Linux Ubuntu, etc.) sea capaz de comprender la lógica del software y empezar a utilizarlo.

Scratch tiene una interfaz gráfica que permite a los usuarios crear y manipular imágenes en 2D, añadir música y sonidos, crear animaciones, y dotar a cualquier objeto de interactividad. Un proyecto Scratch consiste en una escena fija (fondo) y una serie de objetos movibles. Cada objeto contiene su propio conjunto de imágenes, sonidos, variables y secuencias de comandos. Esta organización permite exportar el proyecto fácilmente, y también importar de forma sencilla elementos de otros proyectos.

La pantalla de Scratch se divide en cuatro áreas, como se puede ver en la *Figura 275*. En la parte izquierda de la herramienta está el escenario. El cuadro azul de la esquina superior derecha del escenario es un botón que permite que la escena se muestre a pantalla completa. Debajo de la escena hay un área que muestra las miniaturas de todos los *sprites*[58] del proyecto. Al hacer clic en una de estas miniaturas se selecciona el *sprite* correspondiente. El panel central tiene tres pestañas: programas, disfraces y sonidos. Seleccionando cada una de esas pestañas cambiará el panel de la derecha, donde el usuario podrá ver y modificar los disfraces (imágenes), los sonidos, o el comportamiento (el programa) del *sprite* elegido. Seleccionando la pestaña de “Programas” en el panel central, aparecerán los bloques de comandos que permiten crear el programa. Los bloques de comandos se dividen en diez categorías: Movimiento, Apariencia, Sonido, Lápiz, Datos, Eventos, Control, Sensores, Operadores, y Más Bloques. Cada categoría está asociada a un color.

Figura 274. Página principal de Scratch en la web.

Fuente: (MIT, 2013)

[58] En informática gráfica, el término *sprite* hace referencia a una imagen o mapa de bits bidimensional que se integra en una escena más grande.

A continuación se muestran una serie de capturas de pantalla de la interfaz de Scratch.

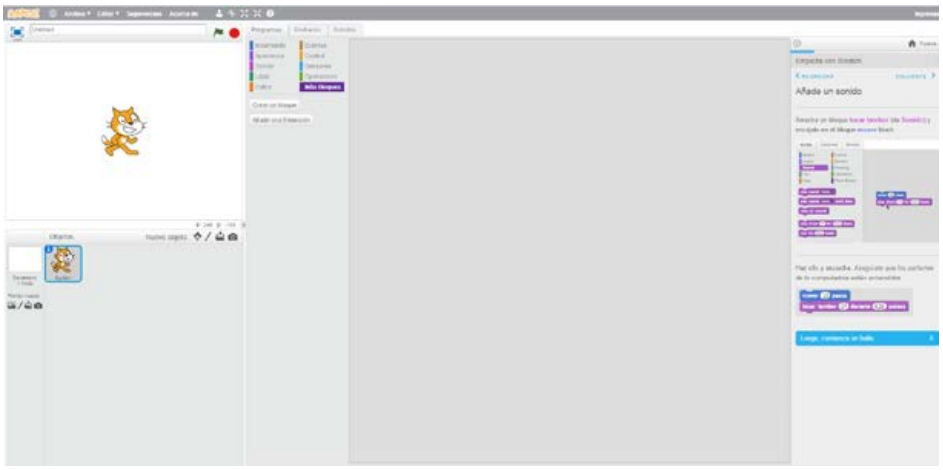


Figura 275. Pestaña inicial al comenzar a utilizar Scratch.

Fuente: (MIT, 2013)

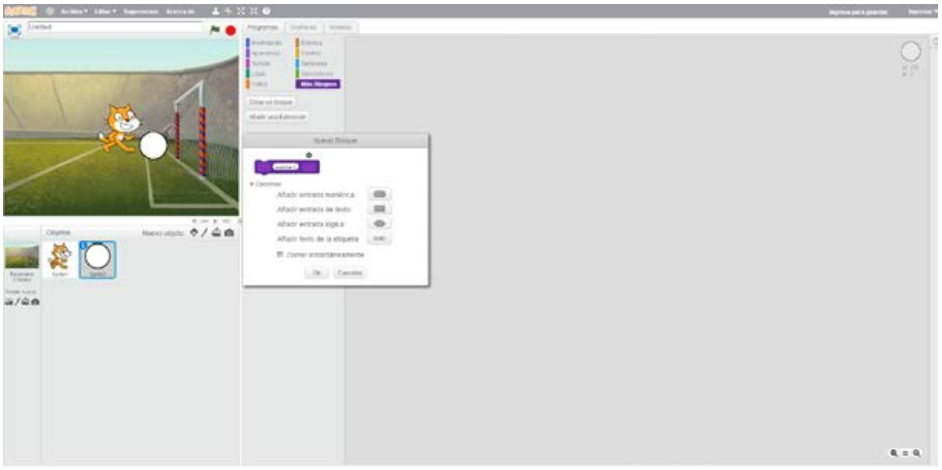


Figura 276. Creación de bloques con sus propias variables en Scratch.

Fuente: (MIT, 2013)

40. Snap

Tabla 165
Resumen de características de la herramienta Snap

Nombre de la herramienta	Snap
Página web	http://snap.berkeley.edu/
URL de descarga	http://snap.berkeley.edu/snapsource/snap.html
Fecha de creación / aparición en el mercado	Año 2011
Última versión en 2017 / año de esa versión	Versión de la web de Snap del año 2017
Plataformas en las que se puede instalar	Funciona a través de la web, aunque tiene una versión *offline*
Tipo de software	Privativo
Precio / modelo de negocio	Gratuito
Edades para las que está indicado	Desde los 8 – 16 años
Aspectos educativos que trabaja	Conceptos básicos de la programación y la programación orientada a objetos, creatividad, estructuración
Característica más destacada / valor diferencial	Posee muchas opciones, permite la exportación a Scratch y viceversa, es una herramienta gratuita

Snap es un lenguaje de programación basado en Scratch, y al igual que este, fue concebido como una herramienta para la enseñanza de la programación.

Figura 277. Logo de la herramienta Snap.

Fuente: (Möneg, 2017)

Su interfaz es muy similar a la de Scratch, como se puede ver en las diferentes capturas de pantalla de la herramienta que se muestran a continuación.

En la zona izquierda de la pantalla se encuentra los bloques de programación, categorizados según su utilidad. Cada una de esas categorías se encuentra en un panel diferente, a los que se accede pulsando sobre la pestaña que los identifica.

En la parte central de la pantalla se encuentra el área de codificación, donde se arrastrarán los bloques de programación que compondrán el programa que se quiera desarrollar.

En la parte de la derecha se encuentra el área de visualización, donde se podrá agregar los elementos gráficos, y comprobar los resultados del programa creado. Bajo esta área se encuentran los objetos que componen la escena, así como los botones para crear nuevas escenas.

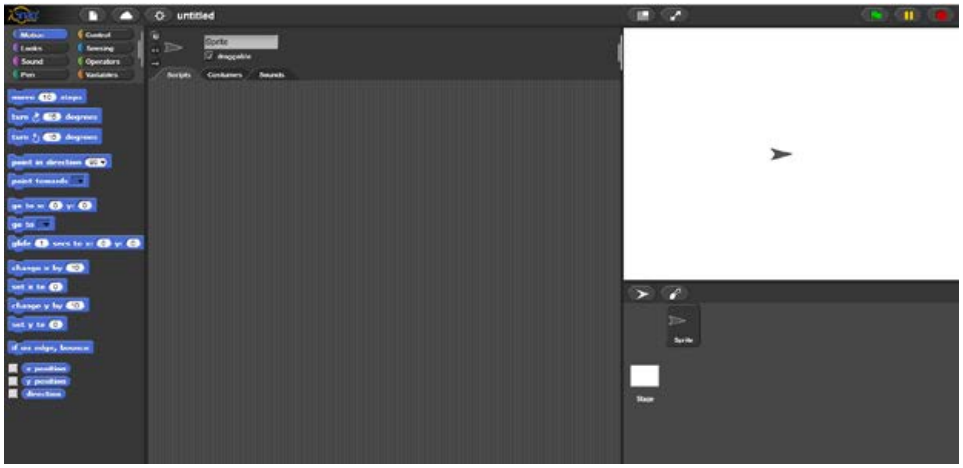


Figura 278. Interfaz de Snap en un proyecto vacío.

Fuente: (Möneg, 2017)

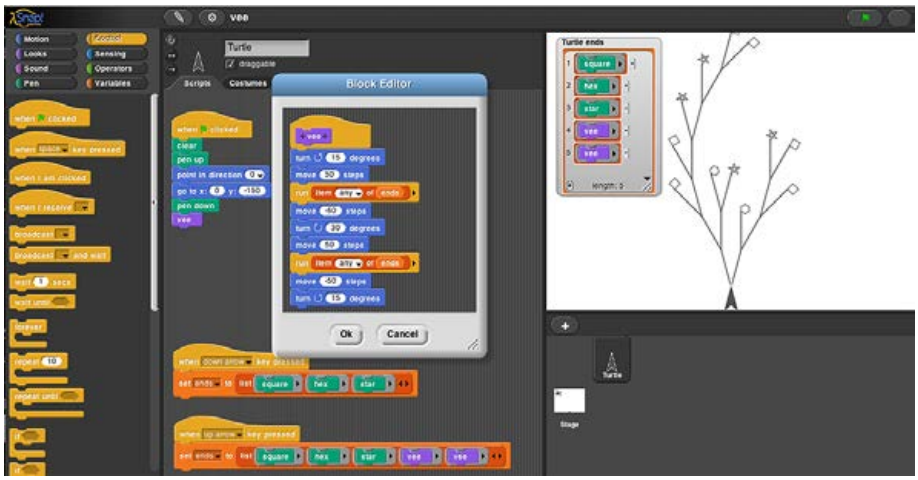


Figura 279. Creación de un programa en Snap utilizando la herramienta de creación de bloques.

Fuente: (Möneg, 2017)

Al igual que Scratch, los proyectos de Snap se pueden compartir, editar y remezclar.

Una de las mayores diferencias entre Snap y Scratch, es que Snap dispone de un número mayor de comandos y bloques de programación (*Figura 280*). De hecho, Snap se creó para suplir las carencias que tenía Scratch en este sentido. También ofrece la posibilidad de crear bloques personalizados.

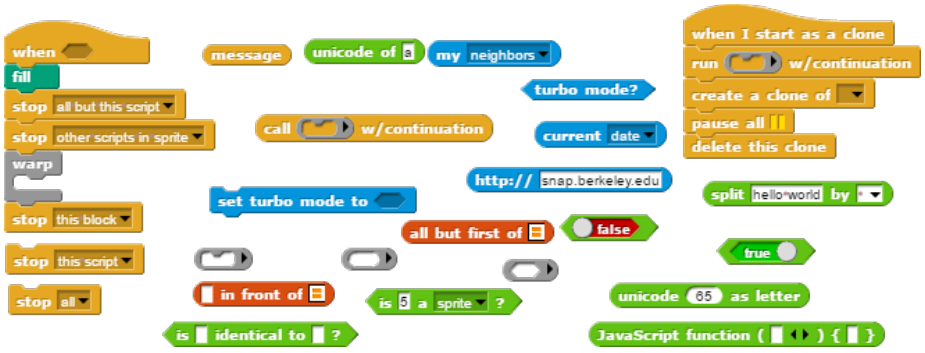


Figura 280. Muestra de algunos de los bloques que Snap tiene en comparación a Scratch

Fuente: (Möneg, 2017)

41. Stagecast Creator

Tabla 166
Resumen de características de la herramienta Stagecast Creator

Nombre de la herramienta	Stagecast Creator
Página web	http://acypher.com/creator/
URL de descarga	http://acypher.com/creator/
Fecha de creación / aparición en el mercado	31 de octubre del año 1996
Última versión en 2017 / año de esa versión	Stagecast Creator versión 2.1, del año 2016
Plataformas en las que se puede instalar	Windows
Tipo de software	Privativo
Precio / modelo de negocio	No se vende actualmente
Edades para las que está indicado	Desde los 8 años
Aspectos educativos que trabaja	Conceptos básicos de la programación
Característica más destacada / valor diferencial	Utiliza un lenguaje muy fácil y es sencillo hacer pequeñas animaciones y juegos

Stagercast Creator es un motor de desarrollo de videojuegos 2D, que permite aplicar diferentes comportamientos a objetos gráficos. Estos objetos pueden ser importados y editados dentro de la herramienta, o creados desde cero. Tanto para crear un nuevo objeto, como para modificarlo, se dispondrá de un editor, que se representa en la *Figura 281.* En dicha imagen puede verse como se crea y anima una imagen, a través de una barra de tiempo, y la creación de distintos fotogramas:

Figura 281. Edición de los sprites de una imagen en Stagecast Creator.

Fuente: (Cypher, 2011)

El comportamiento de los objetos se codifica con el lenguaje de programación Java. En la *Figura 282* se muestra un proyecto creado con Stagecast Creator.

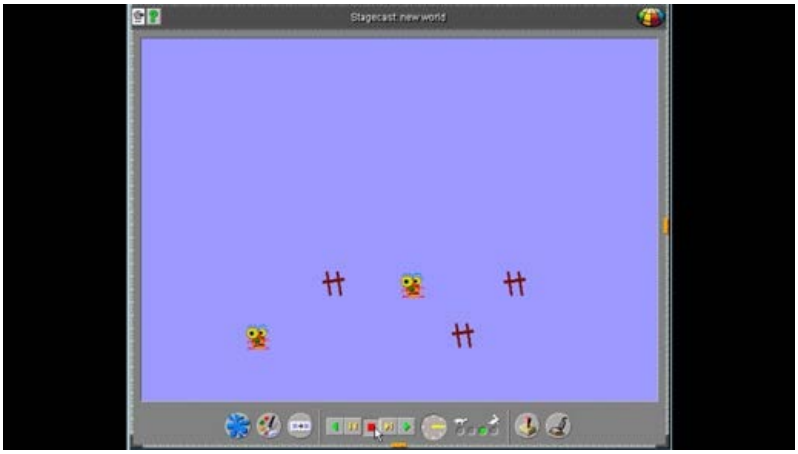


Figura 282. Un pequeño proyecto en Stagecast Creator.

Fuente: (Cypher, 2011)

42. Star Wars: Building a Galaxy with Code

Tabla 167
Resumen de características de la herramienta Hopscotch: Coding for Kids

Nombre de la herramienta	Star Wars: Building a Galaxy with Code
Página web	https://code.org/starwars
URL de descarga	https://studio.code.org/download/starwars
Fecha de creación / aparición en el mercado	13 de julio del año 2015
Última versión en 2017 / año de esa versión	Versión de la web del año 2015
Plataformas en las que se puede instalar	Funciona a través de la web, en la gran mayoría de servidores. También cuenta con una versión offline (Windows y MAC OS X), traducida en muchos idiomas
Tipo de software	Privativo
Precio / modelo de negocio	Gratuito
Edades para las que está indicado	Edades entre los 8 – 16 años
Aspectos educativos que trabaja	Conceptos básicos de la programación, estructuración, resolución de problemas
Característica más destacada / valor diferencial	La herramienta está traducido a muchísimos idiomas, es gratuita y muy fácil de entender. Posibilidad de usar código o bloques con programación predefinida

Star Wars: *Building a Galaxy with Code* es una aplicación web, que tiene como finalidad la enseñanza de la programación a personas sin experiencia en la misma, a través de minijuegos ambientados en el universo de Star Wars.

Figura 283. Una de las desarrolladoras explicando cómo y porqué se desarrolló Star Wars: *Building a Galaxy with Code*.

Fuente: (Code Studio, 2015)

Cada uno de estos minijuegos incluye un tutorial donde se muestra la idea principal del proyecto y sus características, además de mostrarnos el espacio de trabajo, la

forma de trabajar y los objetivos a cumplir. El juego consta de 15 niveles en los que los jugadores deberán resolver unos puzles. Al completar un nivel se tendrá acceso al siguiente. Un ejemplo de este juego lo tenemos en el primer nivel, donde se dispone de un robot, que se deberá programar para que recorra el escenario recogiendo ciertos objetos.

El lenguaje de programación que se utiliza es una versión adapta de JavaScript. La sintaxis es la propia de este lenguaje, solo que en Star Wars: *Building a Galaxy with Code,* el código a utilizar se encuentra dentro de bloques. Al igual que en herramientas de programación por bloques, como Scratch, algunas piezas o bloques se pueden combinar entre sí, y otras no, dependiendo de si dicha combinación da como resultado una pila de código sintácticamente correcta. También se puede optar por escribir este mismo código de forma textual. En la *Figura 284* se muestran algunos de los bloques anteriormente mencionados, ordenados y apilados entre sí para definir un comportamiento.

Figura 284. Un ejemplo del espacio de trabajo en uno de los puzles de desarrolló Star Wars: *Building a Galaxy with Code*.

Fuente: (Code Studio, 2015)

En la esquina superior izquierda de la interfaz se encuentra una ventana donde se podrá ver el resultado del código programado. Debajo, puede verse el botón de ejecutar y resetear el puzle y, bajo este botón, los objetivos e instrucciones a seguir para completar el nivel. A la derecha de la ventana de previsualización, se encuentra un espacio denominado *Toolbox* (caja de herramientas), donde están los

bloques de código de los que se dispondrá para dar órdenes al personaje. Finalmente, a la derecha se puede ver el espacio de codificación, donde se apilarán los bloques, o se escribirá el código que generará el programa necesario para completar el nivel.

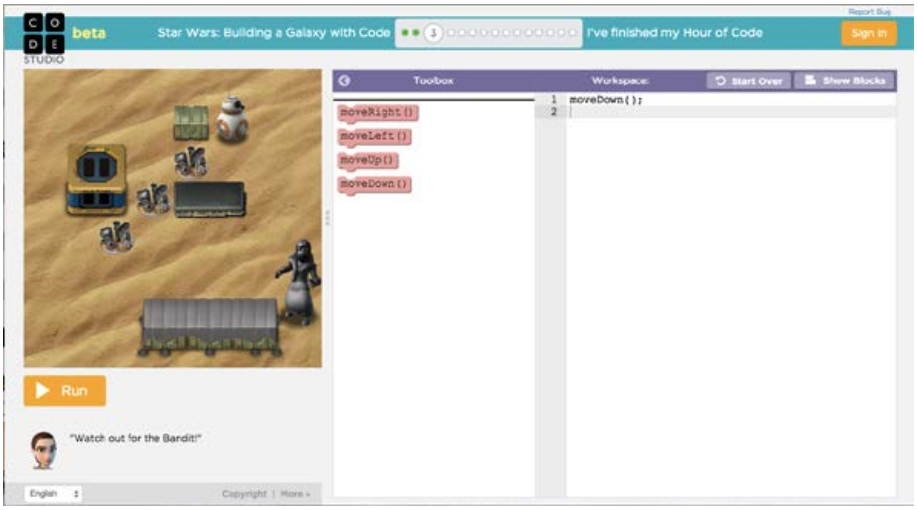


Figura 285. Nivel 3 de Star Wars: *Building a Galaxy with Code*.

Fuente: (Code Studio, 2015)

Cuanto más se avanza en el juego, más información sobre el lenguaje se va obteniendo, y aparecen más tutoriales que muestran cómo afrontar los puzles siguientes.

Una gran ventaja de esta herramienta es la posibilidad de utilizar tanto un sistema de programación por bloques, como uno textual.

Comparándolo con otras herramientas, como Scratch, podemos decir Star Wars: *Building a Galaxy with Code,* centra sus esfuerzos en la enseñanza de la programación, en detrimento del desarrollo de otras capacidades creativas.

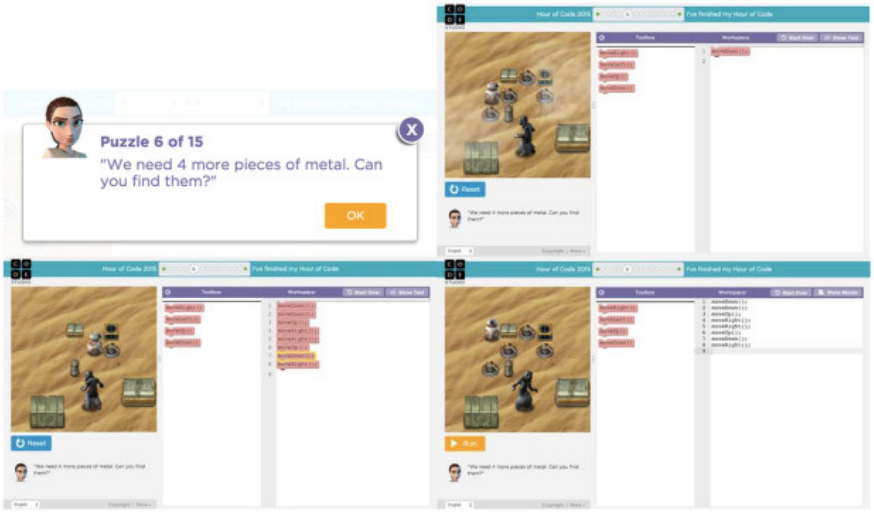


Figura 286. Ejemplos para completar el nivel 6 en Star Wars: *Building a Galaxy with Code*.

Fuente: (Code Studio, 2015)

43. Stencyl

Tabla 168
Resumen de características de la herramienta Stencyl

Nombre de la herramienta	Stencyl
Página web	http://www.stencyl.com/
URL de descarga	http://www.stencyl.com/download/
Fecha de creación / aparición en el mercado	31 de mayo del año 2011
Última versión en 2017 / año de esa versión	Depende de la plataforma, pero están actualizadas desde febrero del año 2017
Plataformas en las que se puede instalar	MAC, Windows y Linux
Tipo de software	Privativo
Precio / modelo de negocio	Gratuito
Edades para las que está indicado	Está indicada tanto para profesionales como público principal (desde los 21 años) pero también tiene un *toolkit* (caja de herramientas) para educación de la escuela Primaria
Aspectos educativos que trabaja	Conceptos básicos de la programación orientada a objetos, creatividad
Característica más destacada / valor diferencial	La posibilidad de hacer juegos propios en 2D de una calidad aceptable para ordenador, web y dispositivos móviles

Stencyl es un motor 2D concebido para la creación de videojuegos. Para programar en Stencyl, se puede escoger entre un sistema de programación por bloques propio de la herramienta, similar a Scratch, o entre un lenguaje de programación textual

denominado Haxe. Los proyectos realizados con Stencyl se pueden exportar a diferentes plataformas: para web (vía Adobe Flash Player, o bien a través de HTML5, que está en fase experimental), para Windows, Mac y Linux, y para dispositivos móviles iOS y Android.

En la *Figura 287* puede verse un ejemplo del sistema de programación por bloques que incorpora Stencyl.

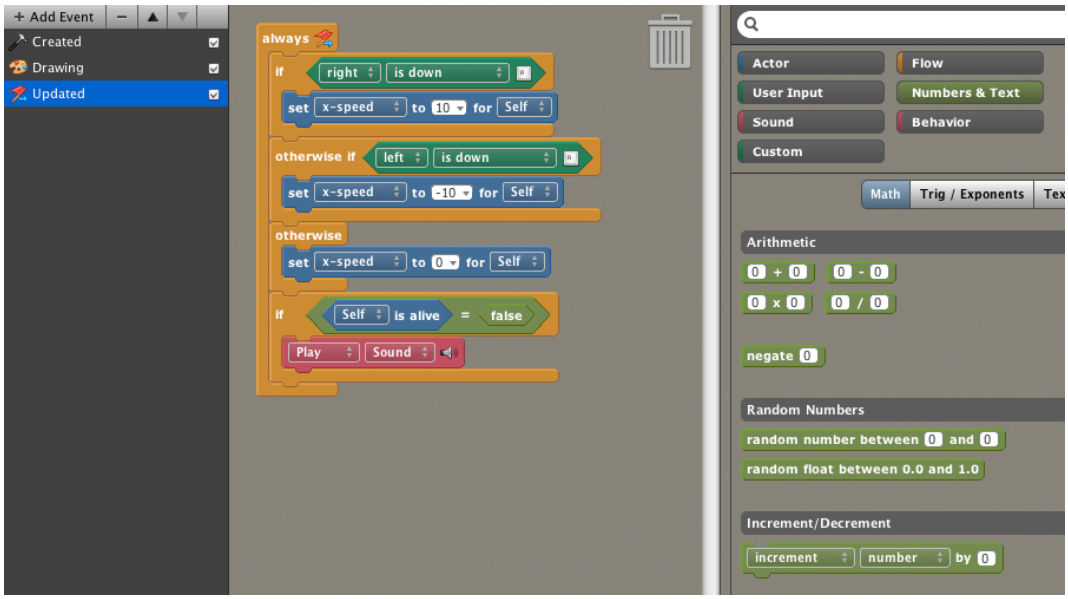


Figura 287. Ejemplo del sistema de programación por bloques que incorpora Stencyl.

Fuente: (Stencyl LLC, 2017)

La herramienta cuenta con un espacio de codificación, y con una serie de editores para trabajar con los elementos gráficos. Estos espacios, a los que se puede acceder a través del menú situado a la izquierda de la pantalla, se dividen en 4 categorías:

- *Resources* (Recursos):
    - *Actor types* (Tipos de actor): permite crear actores o elementos que vayan a tener un cierto comportamiento.
    - *Backgrounds* (Fondos): permite crear de fondos que servirán como base gráfica para el estilo visual del proyecto.
    - *Fonts* (Fuentes): permite crear de fuentes de texto.
    - *Scenes* (Escenas) (*Figura 289*): permite crear de escenas.

- *Sounds* (Sonidos): permite crear de sonidos.
- TileSets (Colección de tiles[59]): permite crear tiles, y animarlos.

- *Logic* (Lógica):
  - *Actor Behaviors* (Comportamientos del actor): área donde se programa el comportamiento de los distintos elementos que componen el proyecto. Esta programación puede hacerse de dos maneras: modo *Design* (Diseño), para programar a través de un sistema por bloques, o modo Code (Código), para programar textualmente a través del lenguaje Haxe
  - *Scene Behaviors* (Comportamientos de la escena): espacio de trabajo para programar el comportamiento de la escena. También permite escoger entre los modos *Code* o *Design*.
  - *Code*: espacio para la inclusión de código, aplicado a cualquier elemento.

- *Resource packs* (Packs de recursos): colección de elementos como personajes, escenas, animaciones, etc.

- *Extensions* (extensiones): extensiones/add-ons (añadidos) que se pueden incluir al proyecto, como publicidad, controles de mandos de consulta, etc.

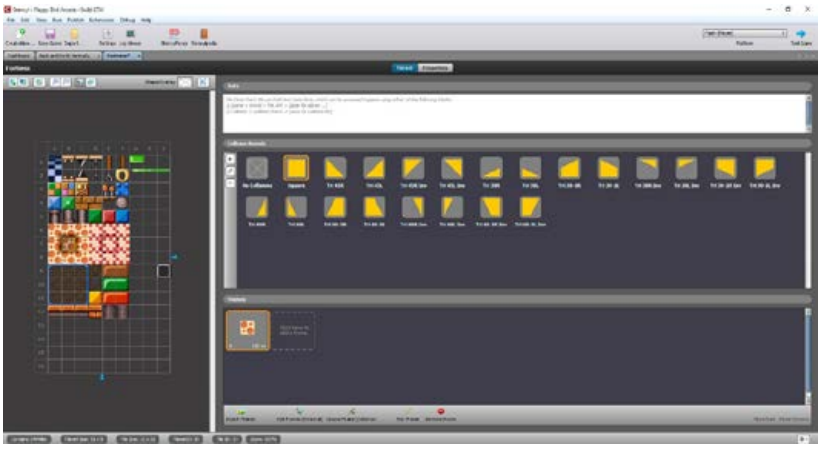


Figura 288. Apartado de gráficos en Stencyl.

Fuente: (Stencyl LLC, 2017)

[59] Un *tile,* que podríamos traducir como baldosa, es un elemento gráfico que se utiliza para componer escenarios.

En la página web de Stencyl se puede descargar un completo manual para iniciarse en el manejo de la herramienta. Asimismo, cuenta con una serie de videotutoriales que guían paso a paso en la realización de ciertos procesos. Dentro del programa también se puede acceder a una ayuda.

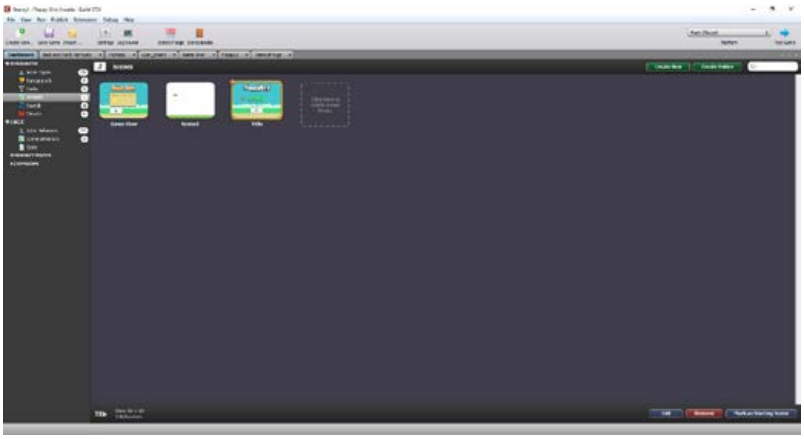


Figura 289. Orden en el que se muestran diferentes escenas en un proyecto en Stencyl.

Fuente: (Stencyl LLC, 2017)

44. The Code Academy

Tabla 169
Resumen de características de la herramienta The Code Academy

Nombre de la herramienta	Code Academy
Página web	https://www.codecademy.com/
URL de descarga	https://www.codecademy.com/learn/all
Fecha de creación / aparición en el mercado	Año 2011
Última versión en 2017 / año de esa versión	Versión actualizada de la web en el año 2017
Plataformas en las que se puede instalar	Funciona a través de la web
Tipo de software	Privativo
Precio / modelo de negocio	La herramienta es gratuita, aunque tiene una versión de pago que da acceso a contenido adicional (20$)
Edades para las que está indicado	No está especificada
Aspectos educativos que trabaja	Conceptos básicos de la programación, programación orientada a objetos, estructuración, creatividad
Característica más destacada / valor diferencial	Cuenta con muchísimos lenguajes de programación y otras herramientas que poder aprender

The Code Academy (*Figura 290*) es una plataforma online gratuita (aunque tiene una modalidad de pago que permite el acceso a ciertos extras), creada para la enseñanza de la programación en diferentes lenguajes, a través de tutoriales. Cuenta con una gran comunidad de usuarios, y actualiza sus contenidos de manera continua.

Figura 290. Logo de *The* Code Academy.

Fuente: (Code Academy, 2017)

Lo primero que un usuario de *The Code Academy* debe hacer después de registrarse, es elegir con qué lenguaje de programación quiere trabajar. Esta decisión no es definitiva, puede cambiar de lenguaje siempre que quiera. En la pantalla donde se hace esta elección (*Figura 291*), en cada lenguaje que se haya utilizado, aparecerá un indicador del nivel de conocimiento adquirido. Una vez elegido el lenguaje, aparecerá una lista de ejercicios enfocados a su aprendizaje.

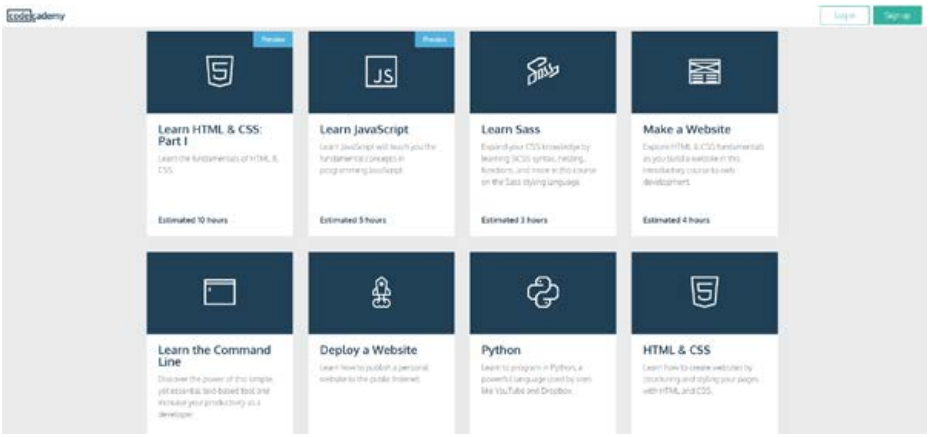


Figura 291. Parte de las herramientas que se pueden aprender en Code Academy.

Fuente: (Code Academy, 2017)

Cuando se accede a un ejercicio, se pueden distinguir varios apartados:

- A la izquierda, se muestran las instrucciones y pasos a seguir para resolver los ejercicios, o recibir ayuda si fuera necesario.
- A la derecha de las Instrucciones, se encuentra el área de trabajo, donde se escribe el código para resolver el ejercicio.

- A la derecha del área de trabajo, está el área de previsualización, donde se puede ver el resultado de lo que se ha programado.

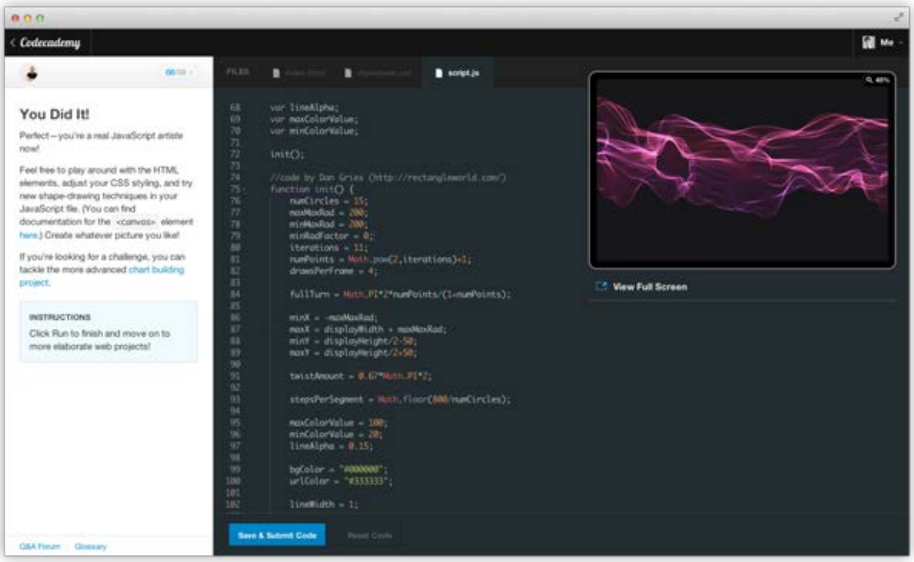


Figura 292. Un ejemplo de un ejercicio resuelto en *The Code Academy*.

Fuente: (Code Academy, 2017)

Debajo de los apartados anteriormente mencionados, hay un espacio donde se muestran los mensajes del sistema, y los errores de sintaxis cometidos. Este espacio también cuenta con una serie de botones, que permiten guardar el código, mostrar la solución al ejercicio, resetearlo, o acceder a foros, ayudas y a un glosario de la herramienta.

The Code Academy está fundamentalmente orientado para un público adulto. Si se trata de enseñar a niños y niñas a programar, *The Code Academy* quizás no sea la mejor opción.

45. Tickle

Tabla 170
Resumen de características de la herramienta Tickle

Nombre de la herramienta	Tickle
Página web	https://tickleapp.com/
URL de descarga	https://itunes.apple.com/app/tickle-program-drones-robots/id1063639403?&mt=8&utm_source=tickleapp.com&utm_medium=web&utm_campaign=tickleapp3
Fecha de creación / aparición en el mercado	Año 2015
Última versión en 2017 / año de esa versión	Versión 4.2.1 de Tickle, del 31 de diciembre del año 2016
Plataformas en las que se puede instalar	iOS
Tipo de software	Privativo
Precio / modelo de negocio	De pago, la herramienta es gratuita y puede usarse sin necesidad de usarla en los robots (que es su principal punto de interés), pero queda muy limitada
Edades para las que está indicado	Edades entre los 6 – 10 años
Aspectos educativos que trabaja	Estructuración, planificación, resolución de problemas, geometría
Característica más destacada / valor diferencial	Posibilidad de ver los resultados de un programa de un modo físico a través de los robots

Tickle (*Figura 293)* es una aplicación para dispositivos móviles, creada para la enseñanza de programación, a través de una serie de robots, o través de la propia aplicación. Utiliza un sistema de programación por bloques, similar al de Scratch.

Figura 293. Imagen promocional de Tickle.

Fuente: (Tickle Labs, 2016)

En el sistema de programación de Tickle, los bloques tienen unos colores asignados, que los clasifican según su utilidad: eventos, controles, operadores, movimientos, etc.

Una vez seleccionado el robot que queremos programar (cada robot tiene opciones y características propias), se accederá al espacio de trabajo. Este espacio está dividido en dos apartados:

- En la parte izquierda, se encuentra el set de bloques disponible para el robot elegido.

- En la parte derecha, se encuentra el espacio de codificación, al que se arrastrarán los bloques para componer el programa.

Cuando se complete el programa, se podrá pulsar sobre el botón *Play*(Comenzar) o *Stop* (Parar), para comprobar el resultado del código programado.

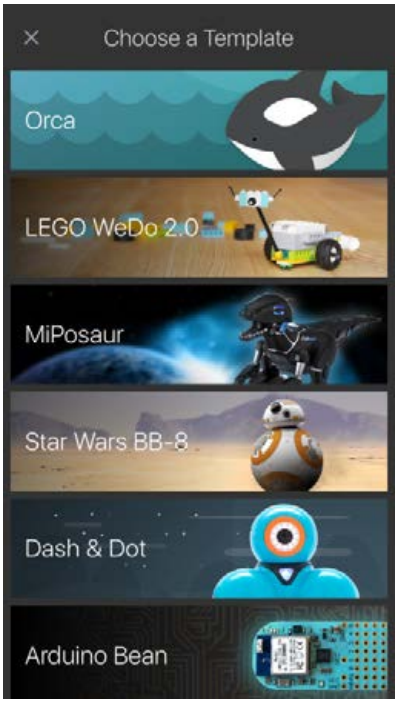


Figura 294. Opciones para la programación de cada uno de los robots el modo Orca en Tickle.

Fuente: (Tickle Labs, 2016)

La herramienta cuenta también con algunos botones para acciones concretas (colocados en la parte inferior derecha), como realizar una parada de emergencia (por ejemplo, para drones voladores, si están en una situación de peligro), volver al paso anterior, o avanzar al siguiente paso.

La herramienta también tiene un modo "Sin Robots", en el que los usuarios pueden programar el comportamiento del avatar de una orca, para que se mueva o haga distintas acciones dentro de una escena. Las opciones de programación en este modo son más básicas, y los resultados menos llamativos que con los robots.

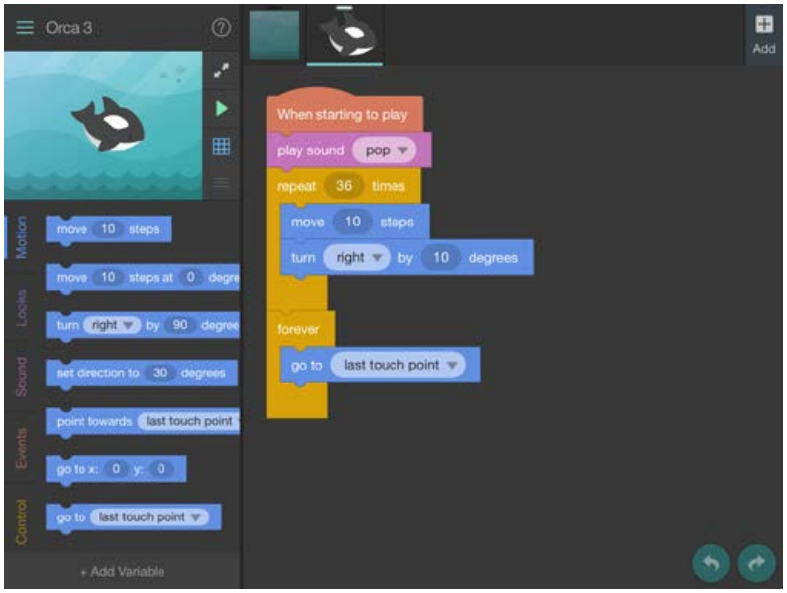


Figura 295. Modo Orca y ejemplo de un proyecto en Tickle.

Fuente: (Fox, 2015)

El lenguaje de programación por bloques que utiliza, no es extrapolable a otros contextos fuera de esta aplicación.

46. Tynker

Tabla 171
Resumen de características de la herramienta Tynker

Nombre de la herramienta	Tynker
Página web	https://www.tynker.com/
URL de descarga	https://www.tynker.com/parents/
Fecha de creación / aparición en el mercado	Julio del año 2014
Última versión en 2017 / año de esa versión	Depende de la plataforma, pero están actualizadas desde el 23 de febrero del año 2017
Plataformas en las que se puede instalar	iOS, Android y a través de la web
Tipo de software	Privativo
Precio / modelo de negocio	De pago, hay varios packs (4 meses, 1 año o de por vida), el precio depende del pack de enseñanzas que se quieran aprender (entre los 8$ - 16$), además de un servidor de Minecraft
Edades para las que está indicado	Edades entre los 8 – 14 años
Aspectos educativos que trabaja	Uso de lenguajes específicos, conceptos básicos de la programación, matemáticas, resolución de ejercicios
Característica más destacada / valor diferencial	La posibilidad de crear juegos propios, es una herramienta con varios niveles de dificultad adaptables a las necesidades de los usuarios

Tynker es una herramienta que permite realizar videojuegos y aplicaciones. Permite elegir entre un sistema de programación por bloques, similar al de Scratch, o utilizar los lenguajes textuales Python o JavaScript.

Tynker permite crear proyectos desde cero, o bien realizar ejercicios y problemas, que se resuelven a través de un programa codificado. Estos modos de aprendizaje se pueden realizar desde tres vías: a través de una aplicación para *tablet*, en el universo de Minecraft, o con proyectos incluidos en La Hora del Código.

La parte en la que centraremos este análisis será de la de la aplicación para *tablet*. Cuando se crea un nuevo proyecto en esta aplicación, aparece un área de trabajo denominado *Tynker WorkShop* (Taller de Tynker). Este espacio de trabajo cuenta con tres áreas diferenciadas: área de bloques (a la izquierda), área de codificación (en el centro), y área de visualización de resultados (a la derecha).

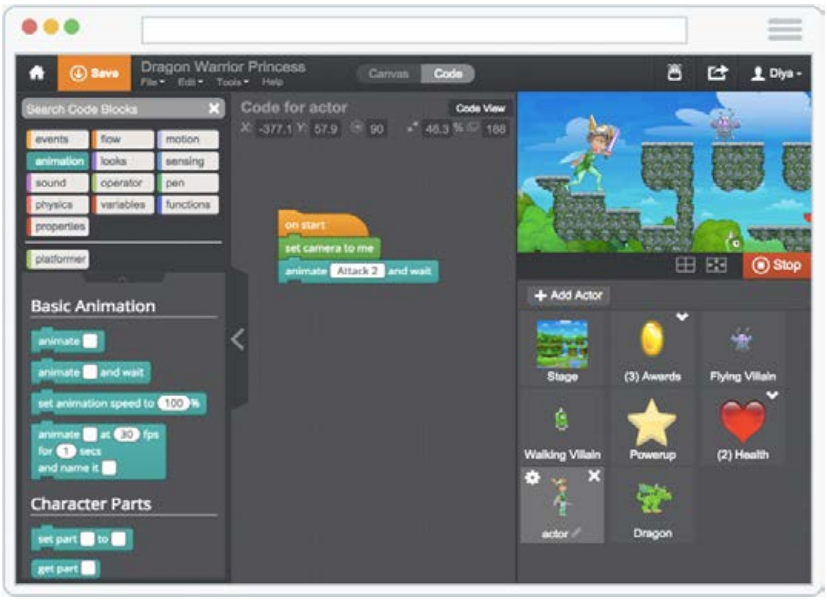


Figura 296. Un proyecto de ejemplo (*Dragon Warrior Princess*) realizado en la aplicación de Tynker.

Fuente: (Neutron Fuel, 2017)

Al igual que en herramientas similares, los bloques de programación se encuentran clasificados por un código de colores, según su utilidad. Estos bloques se arrastrarán al área de codificación para crear el programa. En el área de visualización se podrá comprobar el resultado del programa realizado. La novedad de Tynker es que el área de codificación es configurable, podremos elegir entre programar por bloques, o utilizar código textual a través de JavaScript. Podemos hacer esta transición en cualquier momento, y el código que esté desarrollado hasta ese momento se traducirá al otro sistema, siempre que sea sintácticamente correcto.

Unos botones situados en la parte superior, permiten intercambiar la vista del espacio central, entre una área de codificación (botón *Code*), o una área para la edición de elementos gráficos (botón *Canvas*)

En la *Figura 297* se muestra un proyecto realizado con la aplicación de Tynker, donde un tutorial (parte izquierda), incluido en la herramienta, explica paso por paso el proceso para su elaboración.

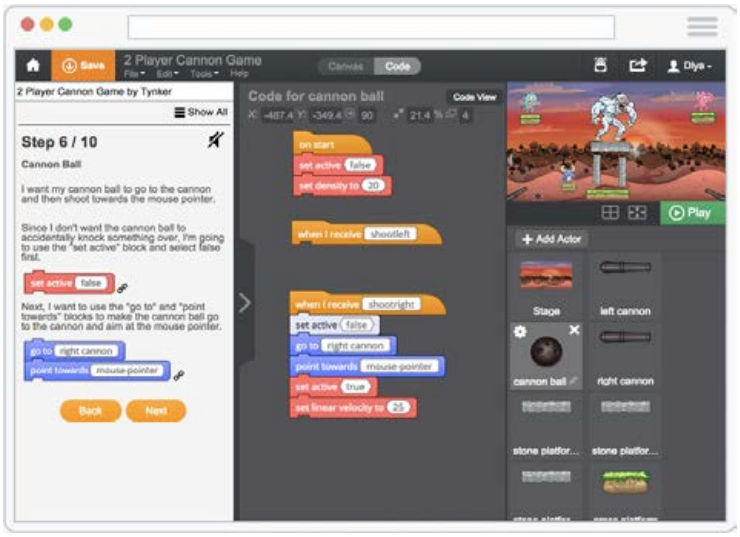


Figura 297. Creación de un juego a través de un tutorial, en la aplicación de Tynker.

Fuente: *(Neutron Fuel, 2017)*

Tynker da la opción de compartir el proyecto realizado con la comunidad de usuarios. Dentro de esa comunidad, algunos usuarios se declaran como *testers* (evaluadores). El programa tiene una opción que permite enviar vía email el enlace del juego a uno de esos *testers*, y así obtener un *feedback* del proyecto.

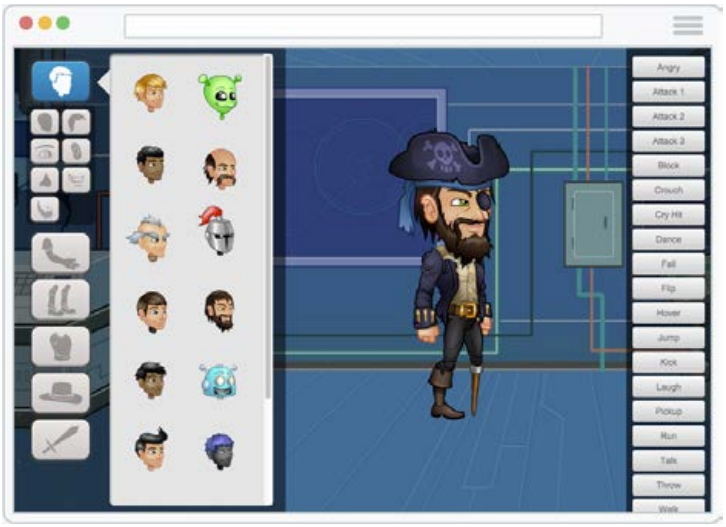


Figura 298. Editor de personajes de Tynker, en un ejercicio incluido en La Hora del Código.

Fuente: *(Neutron Fuel, 2017)*

47. Unity 5

Tabla 172
Resumen de características de la herramienta Unity 5

Nombre de la herramienta	Unity 5
Página web	https://store.unity.com/es
URL de descarga	https://unity3d.com/es/unity
Fecha de creación / aparición en el mercado	Versión 5.3 el día 8 de Septiembre del año 2015
Última versión en 2017 / año de esa versión	Versión de Unity 5.5.2, del 24 de febrero de 2017
Plataformas en las que se puede instalar	Windows, MAC OS X y Linux
Tipo de software	Libre
Precio / modelo de negocio	Existen varias versiones, pero hay una gratuita (con características recortadas) y el resto son de pago (desde los 35$ - 125$ por puesto al mes), más orientado a empresas y profesionales del sector
Edades para las que está indicado	Desde los 18 años, es una herramienta pensada para el mundo profesional
Aspectos educativos que trabaja	Programación, programación orientada a objetos, estructuración, planificación, creatividad
Característica más destacada / valor diferencial	Una gran potencia de motor respecto al resto de herramientas analizadas y que permite un gran acabado y personalización completa

Unity 5 (*Figura 299*) es uno de los motores de desarrollo de videojuegos 2D y 3D más extendidos en el ámbito profesional, debido a su versatilidad, y a sus opciones de exportación (prácticamente para cualquier plataforma).

Figura 299. Logo de Unity.

Fuente: *(Van Oosten, 2012)*

La herramienta posee una interfaz dividida en varios apartados, cada una con diferentes funciones. La herramienta permite mover estos apartados, así como mostrarlos u ocultarlos, según se desee configurar el espacio de trabajo. En la *Figura 300* se muestra la configuración estándar, donde aparecen las secciones principales del programa.

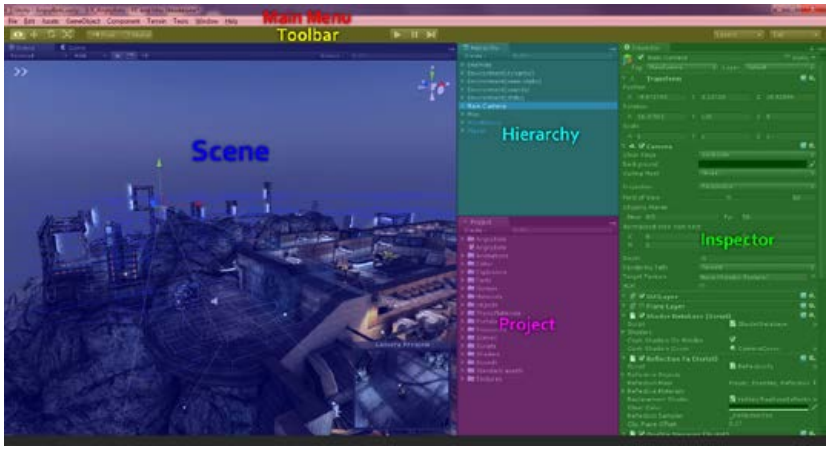


Figura 300. Apartados e interfaz de Unity 5.

Fuente: *(Van Oosten, 2012)*

En la parte superior se encuentra la barra de herramientas, que contiene opciones como mover un objeto (arrastrándolo por el escenario), rotar, escalar, pausar o poner el programa en marcha. En este apartado también se encuentran los distintos menús que dan a acceso a herramientas específicas dentro del programa (por ejemplo, *Mechanim*, la utilidad de Unity para crear máquinas de estados, o *Animate,* para generar animaciones).

Uno de los apartados es *Scene,* donde se compone la escena del juego. Este espacio tiene dos modos de visualización: modo edición (la opción por defecto), donde se podrán agregar y modificar los objetos de la escena, y modo *Game* (Juego), que permitirá previsualizar los resultados del proyecto. Si se pulsa en el botón *Play* (Jugar), que se encuentra en la barra de herramientas, en la ventana *Game* se visualizará el juego en ejecución. Si no se ha pulsado *Play*, o si el juego se ha pausado, en la ventana *Game* se observará una vista estática del proyecto.

Otro de los apartados es *Hierarchy* (Jerarquía), donde se muestra un esquema de todos los elementos que hay en el escenario. El nombre *Hierarchy* procede de la posibilidad que ofrece este espacio para establecer un orden jerárquico entre los objetos del escenario (por ejemplo, dentro de un objeto personaje, habrá un objeto brazo).

Otro de las áreas de trabajo principales es el *Inspector* (Inspector). Cuando se selecciona un objeto en *Hierarchy,* automáticamente aparecerán sus propiedades en el *Inspector.* Estas propiedades estarán separadas por categorías, y dependerán del tipo de objeto que hayamos seleccionado. Por ejemplo, cualquier objeto tendrá unas propiedades X, Y, Z, que determinarán su posición en el eje de coordenadas del escenario. Los objetos 3D, tendrán además una propiedad que definirá el material del que están hechos, otra que determinará su comportamiento con respecto a la física, etc. Un objeto de tipo *Light* (luz), tendrá propiedades como el color de la luz que emite, intensidad, si proyecta sombra, etc.

Por último, merece la pena mencionar los apartados de *Project* (Proyecto), donde aparecerán los elementos (modelos 3D, imágenes, scripts, etc.) que se han importado al proyecto, y *Console* (Consola), donde se mostrarán los mensajes del sistema (por ejemplo, si se producen errores en la ejecución del código).

En la Figura 301 se observa una escena, mostrada desde cuatro puntos de vista diferentes.

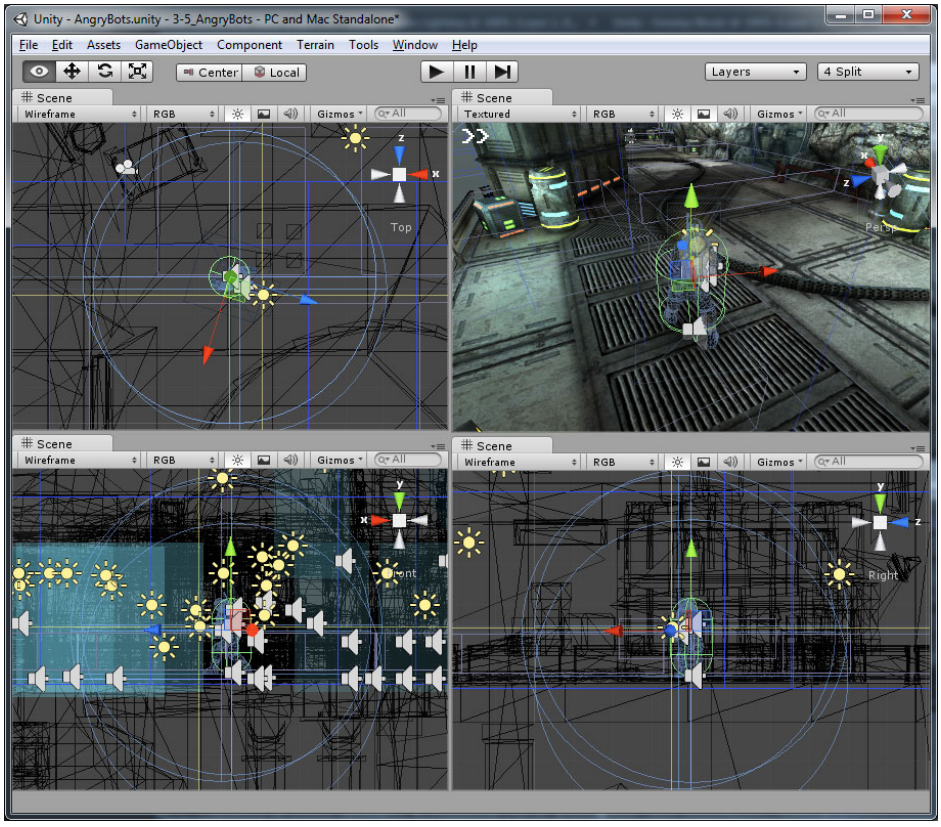


Figura 301. Distintas perspectivas del visor de Unity 5 en un proyecto.

Fuente: *(Van Oosten, 2012)*

El área de codificación se encuentra fuera de la herramienta de Unity. Se pueden utilizar kits de desarrollo como Visual Studio, o Mono Developer. Los principales lenguajes de programación que admite Unity son son C# y JavaScript. Ambos lenguajes siguen el paradigma de la programación orientada a objetos. La forma de unir la parte gráfica con la programación, es a través de la herencia. Las clases programadas se agregan como una propiedad más de un objeto, dentro del *Inspector.*

Las posibilidades que ofrece esta herramienta son inmensas, pero igual de inmensa es su API de programación. Esto significa que se necesita de amplios conocimientos para poder dominarla en todas sus facetas. Para poder adquirir esos conocimientos, el fabricante ofrece abundante documentación y videotutoriales. En cualquier caso,

aunque puede ser utilizada en el ámbito educativo, es una herramienta enfocada al desarrollo profesional.

48. Unreal Engine 4

Tabla 173
Resumen de características de la herramienta Hopscotch: Coding for Kids

Nombre de la herramienta	Unreal Engine 4
Página web	https://www.unrealengine.com/what-is-unreal-engine-4
URL de descarga	https://www.unrealengine.com/download
Fecha de creación / aparición en el mercado	Mayo del año 2012
Última versión en 2017 / año de esa versión	Unreal Engine 4.15, del 15 de febrero del año 2017
Plataformas en las que se puede instalar	Windows y MAC
Tipo de software	Libre
Precio / modelo de negocio	Gratuito, pero a partir de los 3000$ invertidos por ventas de los proyectos desarrollados en la herramienta, la empresa se queda con un 5% de cada venta
Edades para las que está indicado	Desde los 18 años, está pensada para profesionales del sector
Aspectos educativos que trabaja	Programación, programación orientada a objetos, estructuración, planificación, matemáticas, creatividad
Característica más destacada / valor diferencial	Una herramienta con un motor muy potente y un precio muy razonable

Unreal Engine 4 (*Figura 302*) es un motor de desarrollo de videojuegos, principalmente 3D, de gran potencia, utilizado en el ámbito profesional. Utiliza C++ como lenguaje de programación.

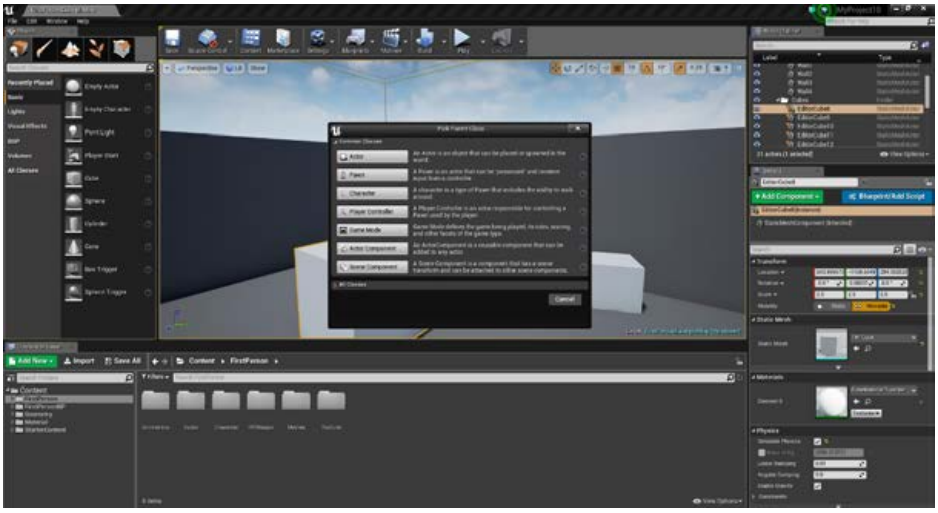


Figura 302. Pantalla inicial e interfaz de Unreal Engine 4.

Fuente: (Epic Games, 2017a)

La interfaz de *Unreal* tiene distintos apartados, cada uno de ellos con una funcionalidad. Dentro de los editores de código y materiales se pueden utilizar el clic derecho y atajos de teclado para sacar funciones dentro del mismo programa, lo que facilita y reduce el tiempo necesario de desarrollo para la creación de proyectos.

Además de C++, *Unreal* ofrece la posibilidad de programar ciertas tareas y comportamientos a través de un sistema de *blueprints* (planos). Los *blueprints* son una especie de cajas con un comportamiento predefinido, y una serie de parámetros. Estas cajas se podrán interconectar entre sí, formando la estructura de un programa. El sistema tiene una serie de *blueprints* predefinidas. A su vez, un *blueprint* se puede crear o editar con C++.

Bien a través de código, o bien a través de *blueprints* , el comportamiento de los distintos elementos que componen una escena en Unreal, se definen principalmente a través de tres apartados:

- *Viewport* (ventana de visualización), donde se puede seleccionar a qué objeto se le quiere aplicar un determinado comportamiento, así como editarlo y ajustarlo visualmente.
- *Construction Script* (Construcción del código), en este apartado se codifica el comportamiento del objeto.
- *Event Graph* (Gráfico de eventos), en este apartado se definirán los eventos a los que responderá el objeto, y las acciones con las que responderá a ese suceso.

Otro elemento reseñable de la interfaz de Unreal es su editor de materiales. También con un sistema de *blueprints,* se puede configurar cualquier tipo de material. Por ejemplo, un material puede estar compuesto por un *bluprint* que define su color de base en a partir de unos parámetros RGB[60], otro *blueprint* que defina su textura, otro que determine cómo se comporta cuando la luz incide en él, etc. La combinación de todos esos *blueprints* dará como resultado el material deseado. Las posibilidades de combinación son innumerables, y los resultados de la composición del material se muestran en tiempo real, según se van agregando los *blueprints*. Los materiales, una vez creados, se aplican a los objetos del proyecto.

Unreal destaca también por su motor de físicas, y por su motor de iluminación. Los resultados que se pueden conseguir son fotorealistas.

Por último decir que, al igual que Unity, Unreal permite exportar los proyectos a prácticamente todas las plataformas.

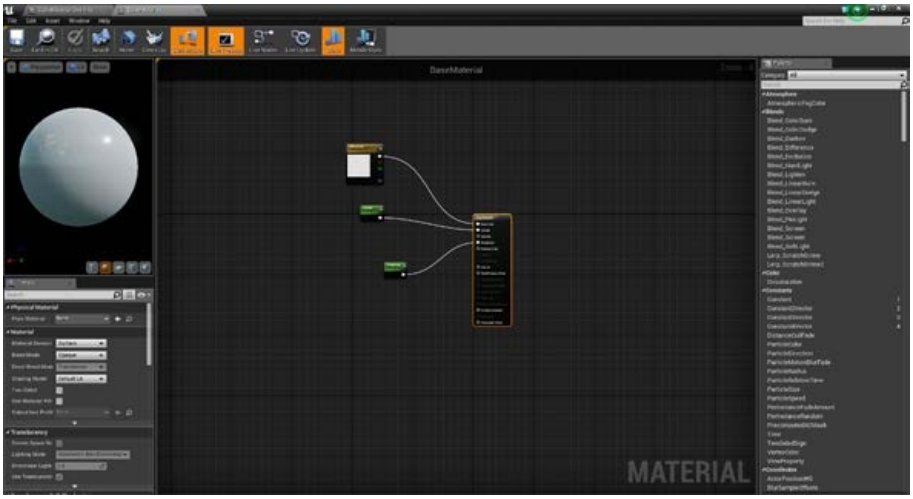


Figura 303. Editor de materiales en Unreal Engine.

Fuente: (Epic Games, 2017a)

Unreal es una herramienta compleja de manejar. A pesar de que existe una gran cantidad de documentación y tutoriales sobre su uso, no es una herramienta que pueda ser utilizada por cualquiera, y su propósito es más profesional que educativo.

[60] RGB, Red, Green, Blue. RGB es un sistema de identificación de colores, basado en la mezcla de rojos, verdes y azules.

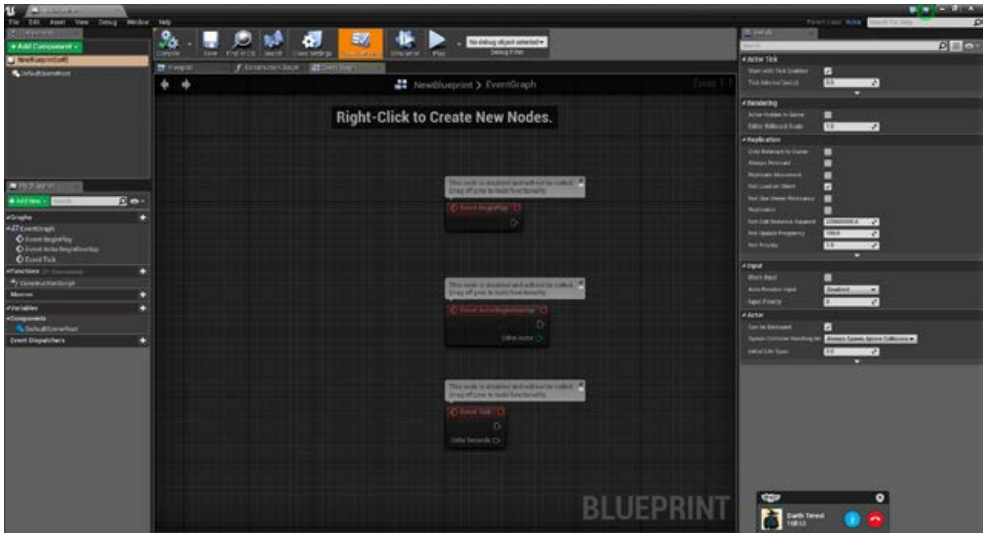


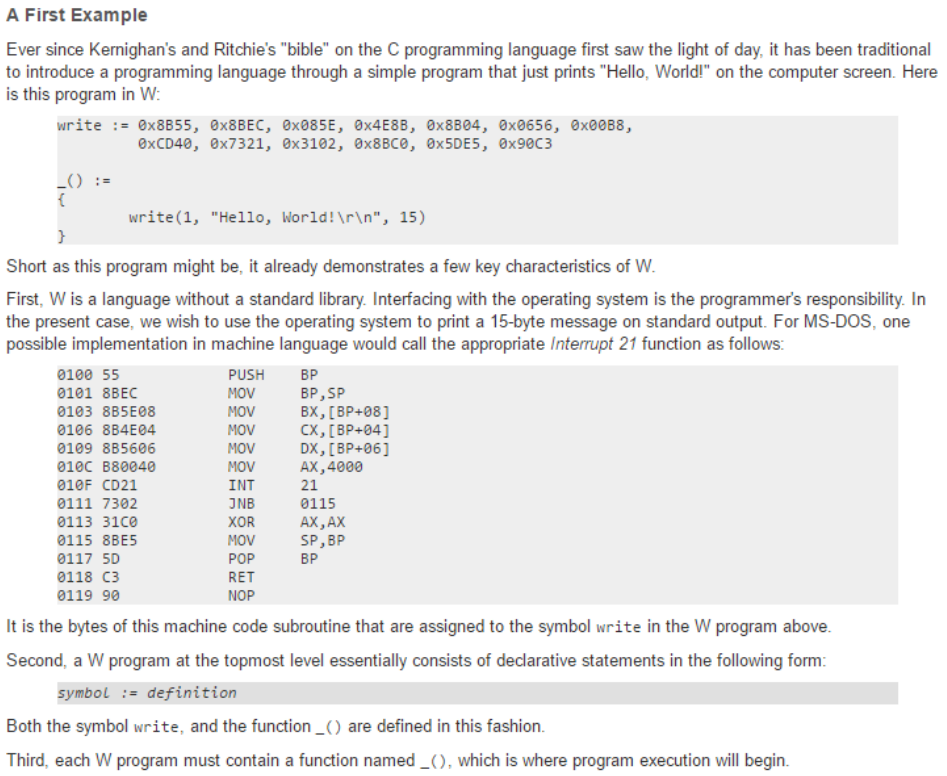
Figura 304. Editor de eventos en Unreal Engine.

Fuente: (Epic Games, 2017a)

49. W

Tabla 174
Resumen de características de la herramienta W

Nombre de la herramienta	W
Página web	https://www.vttoth.com/CMS/projects/49-w-a-simple-programming-language
URL de descarga	https://www.vttoth.com/CMS/projects/49-w-a-simple-programming-language
Fecha de creación / aparición en el mercado	20 de febrero del año 2001
Última versión en 2017 / año de esa versión	Versión de W del año 2016
Plataformas en las que se puede instalar	Windows
Tipo de software	Privativo
Precio / modelo de negocio	Gratuito
Edades para las que está indicado	No está especificado
Aspectos educativos que trabaja	Conceptos básicos de la programación, uso de un lenguaje específico
Característica más destacada / valor diferencial	Se trata de una herramienta de fácil aprendizaje, requiere de muy poca potencia para poder usarse



A First Example

Ever since Kernighan's and Ritchie's "bible" on the C programming language first saw the light of day, it has been traditional to introduce a programming language through a simple program that just prints "Hello, World!" on the computer screen. Here is this program in W:

```
write := 0x8B55, 0x8BEC, 0x085E, 0x4E8B, 0x8B04, 0x0656, 0x00B8,
         0xCD40, 0x7321, 0x3102, 0x8BC0, 0x5DE5, 0x90C3

_() :=
{
        write(1, "Hello, World!\r\n", 15)
}
```

Short as this program might be, it already demonstrates a few key characteristics of W.

First, W is a language without a standard library. Interfacing with the operating system is the programmer's responsibility. In the present case, we wish to use the operating system to print a 15-byte message on standard output. For MS-DOS, one possible implementation in machine language would call the appropriate *Interrupt 21* function as follows:

```
0100 55            PUSH    BP
0101 8BEC          MOV     BP,SP
0103 8B5E08        MOV     BX,[BP+08]
0106 8B4E04        MOV     CX,[BP+04]
0109 8B5606        MOV     DX,[BP+06]
010C B80040        MOV     AX,4000
010F CD21          INT     21
0111 7302          JNB     0115
0113 31C0          XOR     AX,AX
0115 8BE5          MOV     SP,BP
0117 5D            POP     BP
0118 C3            RET
0119 90            NOP
```

It is the bytes of this machine code subroutine that are assigned to the symbol `write` in the W program above.

Second, a W program at the topmost level essentially consists of declarative statements in the following form:

```
symbol := definition
```

Both the symbol `write`, and the function `_()` are defined in this fashion.

Third, each W program must contain a function named `_()`, which is where program execution will begin.

Figura 305. Un ejemplo de código desarrollado con W.

Fuente: *(Toth, 2016)*

W es un lenguaje de programación textual, creado sobre la base del lenguaje C, tratando de hacerlo más sencillo de aprender y de utilizar. W se incluye dentro de esta recopilación como ejemplo de uno de los intentos de hacer más asequible un lenguaje de uso profesional.

Su principal característica es que no cuenta con librerías estándar, y la interacción con el sistema operativo sobre el que se ejecute el código debe ser desarrollada por el programador. El resto de sus características, así como los detalles de su sintaxis, se pueden ver en su página web. En la *Figura 305* se muestra un ejemplo del código desarrollado con W.

50. Wimi5

Tabla 175
Resumen de características de la herramienta Wimi5

Nombre de la herramienta	Wimi5
Página web	http://wimi5.com/es/
URL de descarga	http://wimi5.com/date-de-alta/
Fecha de creación / aparición en el mercado	Año 2015
Última versión en 2017 / año de esa versión	Versión de la web y online, actualizadas para el año 2017
Plataformas en las que se puede instalar	Funciona a través de la web, pero se están trabajando en poder utilizarlos desde plataformas móviles como Android e iOS
Tipo de software	Libre
Precio / modelo de negocio	Gratuito, pero un 30% de las ganancias totales por las ventas de los proyectos creados en la plataforma se los lleva la empresa autora
Edades para las que está indicado	No especificada, pero está orientada al ámbito profesional de desarrolladores de videojuegos
Aspectos educativos que trabaja	Creatividad, estructuración y conceptos básicos de la programación y programación orientada a objetos
Característica más destacada / valor diferencial	Herramienta con una gran comunidad, de fácil acceso y que da la posibilidad de desarrollar sus proyectos tanto a empresas como desarrolladores independientes

Wimi5 (*Figura 306*) es una plataforma para la creación de videojuegos y aplicaciones, a través del lenguaje de programación HTML5. La herramienta cuenta con un editor que ayuda a la creación de proyectos, utilizando plantillas de código, que ya incluyen ciertas funcionalidades, y evitan que el usuario tenga que comenzar a desarrollar desde cero.

Figura 306. Logo de Wimi5.

Fuente: (Díaz, 2015b)

Al utilizar HTML5, los proyectos desarrollados se pueden publicar para web (la página del fabricante incluye un espacio para que sus usuarios compartan sus creaciones), o bien se pueden exportar a múltiples plataformas, a través de herramientas de terceros, como Ludei[61] o Phonegap[62]. Otra característica destacada de Wimi5, es que incluye utilidades para la monetización de los desarrollos (posibilidad de colocar publicidad dentro del juego, etc.).

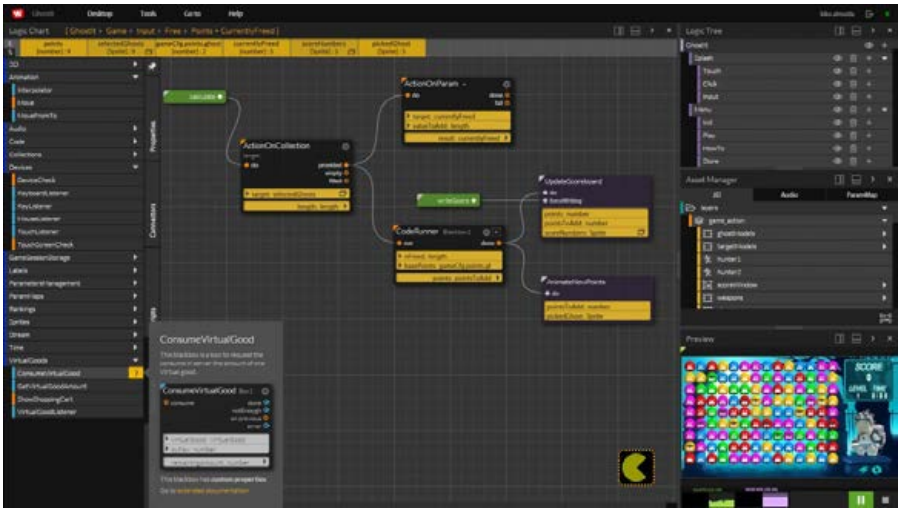


Figura 307. Ejemplo de creación de un proyecto en Wimi5 a través de nodos.

Fuente: (Díaz, 2015b)

Como se puede ver en la *Figura 307*, la interfaz de Wimi5 presenta un aspecto similar a la de otras herramientas como *Unreal Engine*, aunque sus prestaciones no son comparables.

La programación de los comportamientos también se realiza a través de un sistema de *blueprints* (cajas con un comportamiento y unos parámetros predefinidos, que se pueden conectar entre sí para construir un programa). Dentro de la interfaz existe un espacio específicamente diseñado para crear estas *blueprints*, y conectarlas. Asimismo, la interfaz cuenta con un área para la selección, edición y organización de los objetos por jerarquías, y una ventana donde se visualiza el resultado del desarrollo.

La web del fabricante cuenta con una cantidad considerable de tutoriales, y de plantillas de juegos. También hay una amplia comunidad en torno a esta herramienta, así como foros donde se pueden consultar dudas.

[61] https://www.ludei.com/
[62] http://phonegap.com/

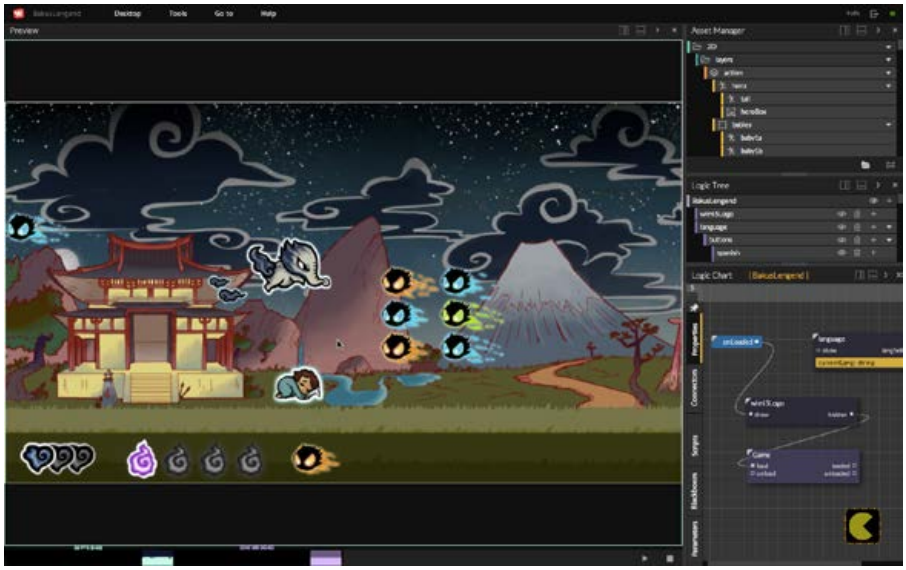


Figura 308. Previsualización de un proyecto creado en Wimi5.

Fuente: (Díaz, 2015b)

51. Zero Engine

Tabla 176
Resumen de características de la herramienta Zero Engine

Nombre de la herramienta	Zero Engine
Página web	http://zero.digipen.edu/
URL de descarga	http://zero.digipen.edu/Builds/ZeroLauncherSetup.exe
Fecha de creación / aparición en el mercado	Año 2012
Última versión en 2017 / año de esa versión	Versión del año 2016
Plataformas en las que se puede instalar	Windows
Tipo de software	Privativo
Precio / modelo de negocio	No cuesta dinero, pero sólo los alumnos de Digipen pueden usarlos, por lo que podríamos decir que su precio sería la matrícula
Edades para las que está indicado	Desde los 18 años
Aspectos educativos que trabaja	Creatividad, conceptos básicos de la programación
Característica más destacada / valor diferencial	Exclusividad de la herramienta, está enfocado a las necesidades de los alumnos de DigiPen

Zero Engine es un motor de simulación, de uso exclusivo por parte de los estudiantes de *DigiPen Institute of Technology*, una escuela que enseña a hacer

videojuegos, con sede central en Washington, y dos delegaciones en Singapur y Bilbao.

Figura 309. Creación de animaciones en Zero Engine.

Fuente: (Digipen Institue of Technology, 2016)

Las características de este motor son las siguientes:

- Un editor propio, ZED (Zero Editor)
- Una arquitectura de componentes avanzada.
- Soporte para juegos 2D y 3D.
- Motor personalizable con restricciones basadas en físicas.
- Modelo realista de vuelo.
- Luz dinámica en 3D.
- Composición de *shaders* (efectos de iluminación) dinámicos
- Uso del lenguaje de programación: Zilch (derivado de C++).

Al ser una herramienta de uso exclusivo, no se ha podido probar, y las características que aquí se refieren son las indicadas por el fabricante.

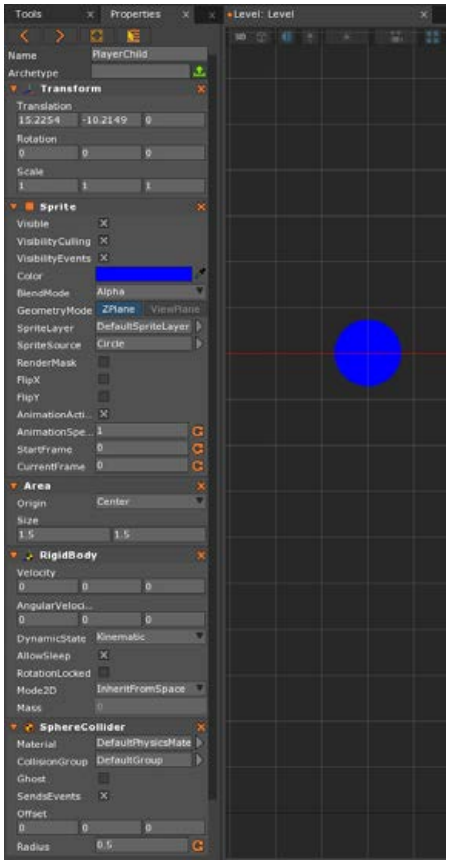


Figura 310. Herramientas de posición y características del espacio en Zero Engine.

Fuente: (Digipen Institue of Technology, 2016)

20. ANEXO 5. Indicadores para evaluar el conocimiento adquirido sobre programación: Referentes internacionales.

En este anexo se describen dos estándares definidos por organismos de prestigio internacional que han marcado la pauta en este terreno, y que pueden servir como referencia para evaluar el conocimiento adquirido sobre programación.

20.1. Indicadores en el Reino Unido

El primero de ellos es el propuestos por la asociación inglesa CAS (*Computing at School*), uno de los países donde desde hace años se lleva promoviendo la programación en las aulas.

El denominado *Computing Progression Pathways* (Dorling y Walker, 2015), que podríamos traducir como Itinerarios para la Progresión en la Informática, describe los resultados de aprendizaje correspondientes a las áreas de Informática, Tecnología de la Información y Alfabetización Digital del programa de estudio del currículo nacional del Reino Unido. El documento se ha elaborado en el contexto la organización CAS (*Computing at School*), que agrupa a una amplia comunidad educativa con el interés común de promover la enseñanza de la programación. Este texto es un referente para el ámbito de esta tesis dado que el Reino Unido ha sido uno de los países pioneros en implantar la enseñanza de la programación a lo largo de todas las etapas educativas obligatorias (UK Department of Education, 2013). El objetivo del *Computing Progression Pathways* es definir la progresión que un estudiante debe seguir desde el "*Year* 1" (5-6 años, equivalente al último curso de Educación Infantil en España) hasta el "*Year* 9" (13-14 años, equivalente a 2º ESO), y los logros que debe alcanzar en el ámbito de la informática.

Computing Progression Pathways define 6 itinerarios a seguir (Dorling y Walker, 2015):

- Algoritmos ("*Algorithms*")
- Programación y Desarrollo ("*Programming & Development*")
- Datos y Representación de Datos ("*Data & Data Representation*")
- Hardware y Procesamiento ("*Hardware & Processing*")
- Comunicación y Redes ("*Communication & Networking*")
- Tecnologías de la Información ("*Information Technology*")

En el contexto educativo británico, el pensamiento computacional se entiende como una capacidad transversal a desarrollar en todos los itinerarios. Por este motivo, en cada resultado de aprendizaje que presenta el documento se indica qué aspecto del pensamiento computacional se está desarrollando. Los aspectos o conceptos relacionados con el pensamiento computacional que determina este texto son los siguientes*:*

- Abstracción ("*Abstraction*" = AB)
- Descomposición ("*Decomposition*" = DE)
- Pensamiento Algorítmico ("*Algorithmic Thinking*" = AL)
- Evaluación ("*Evaluation*" = EV)
- Generalización ("*Generalization*" = GE)

A continuación se muestran los 6 itinerarios de *Computing Progression Pathways*, y los resultados de aprendizaje de cada uno clasificados según la edad de aplicación,

Tabla 177
Computing Progression Pathways, traducción de la tabla original elaborada por *Computing at School*.

	Algoritmos	**Programación y Desarrollo**	**Datos y Representación de Datos**	**Hardware y Procesamiento**	**Comunicación y Redes**	**Tecnologías de la Información**
5 – 6 años	• Comprende qué es un algoritmo y es capaz de expresar algoritmos simples lineales (sin ramas) de manera simbólica (AL) • Comprende que los ordenadores necesitan instruccione s precisas (AL) • Muestra cuidado y precisión para evitar errores (AL)	• Sabe que los usuarios pueden desarrollar sus propios programas, y lo muestra creando un programa simple en un entorno que no requiere texto (p. ej. robots programables) (AL) • Ejecuta, comprueba y modifica programas (AL) • Comprende que los programas se ejecutan siguiendo instrucciones • precisas • (AL)	• Reconoce que el contenido digital puede ser representado de muchas formas (AB) (GE) • Distingue entre algunas de estas formas y puede explicar sus diferencias a la hora de comunicar información (AB)	• Comprende que los ordenadores no tienen inteligencia, y que no hacen nada al menos que un programa sea ejecutado (AL) • Reconocen que todo el software ejecutado en los dispositivos digitales está programado (AL) (AB) (GE)	• Obtiene contenido de Internet utilizando un navegador web (AL) • Comprende la importancia de comunicarse de manera segura y respetuosa on-line, y la necesidad de mantener en privado la información personal (EV) • Sabe qué hacer cuando le afecta/preocupa algún contenido o es contactado on-line (AL)	• Utiliza software bajo el control del profesor para crear, editar y almacenar contenido digital, utilizando nombres apropiados para carpetas y archivos (AB) (GE) (DE) • Comprende que la gente interactúa con computadoras • Comparte su conocimiento de la tecnología en la escuela • Conoce usos comunes de las tecnologías de la información más allá del aula (GE) • Habla acerca de su trabajo y hace cambios para mejorarlo (EV)

	Algoritmos	Programación y Desarrollo	Datos y Representación de Datos	Hardware y Procesamiento	Comunicación y Redes	Tecnologías de la Información
6 – 7 años	• Comprende que los algoritmos son implementados en los dispositivos digitales como programas informáticos (AL) • Diseña algoritmos simples utilizando bucles y condiciones (p. ej. sentencias "If") (AL) • Utiliza el razonamiento lógico para predecir resultados (AL) • Detecta y corrige errores (p. ej. depuración) en algoritmos (AL)	• Utilizar operadores aritméticos, sentencias condicionales "If", y bucles en sus programas (AL) • Utiliza el razonamiento lógico para predecir el comportamiento de los programas (AL) • Detecta y corrige errores simples semánticos (p. ej. depuración) en los programas (AL)	• Reconoce diferentes tipos de datos: texto, número (AB) (GE) • Aprecia que los programas pueden funcionar con diferentes tipos de datos (GE) • Reconoce que los datos pueden ser estructurados en tablas para hacerlos más útiles (AB) (DE)	• Reconoce que existe un amplio abanico de dispositivos digitales que pueden ser considerados como un ordenador (AB) (GE) • Reconoce y puede usar una variedad de dispositivos que soportan "inputs" y "outputs" • Comprende cómo los programas especifican las funciones de un ordenador, dentro de su propósito general (AB)	• Navega la red y puede llevar a cabo búsquedas web sencillas para recopilar contenido digital (AL) (EV) • Evidencia un uso de los ordenadores seguro y responsable, conociendo diversas vías para informar de contenidos o contactos on-line inaceptables.	• Utiliza la tecnología con una independencia creciente, organizando contenido digital con arreglo a un objetivo (AB) • Muestra conciencia sobre la calidad del contenido digital recopilado (EV) • Utiliza una amplia variedad de software para manipular y presentar contenido digital: datos e información (AL) • Comparte sus experiencias con la tecnología en el aula y más allá de la misma (GE) (EV) • Habla sobre su trabajo y hace mejoras a sus soluciones basadas en el feedback recibido (EV)

	Algoritmos	**Programación y Desarrollo**	**Datos y Representación de Datos**	**Hardware y Procesamiento**	**Comunicación y Redes**	**Tecnologías de la Información**
7 – 8 años	• Diseña soluciones (algoritmos) que usan repeticiones y condicionales compuestos (p. ej. "If/else") (AL) • Utiliza diagramas para expresar soluciones (AB) • Utiliza el razonamiento lógico para predecir resultados ("outputs"), mostrando conciencia de su afectación por las entradas ("inputs") (AL)	• Crea/escribe programas que implementan algoritmos para lograr metas fijadas (AL) • Declara y asigna variables (AB) • Usa bucles indefinidos ("post-tested loops", p. ej. "repeat until") y secuencias de sentencias condicionales compuestas (p. ej. "If/else") (AL)	• diferencia entre datos e información (AB) • Sabe por qué ordenar datos en un archivo plano puede mejorar la búsqueda de información (EV) • Utiliza filtros o puede realizar búsquedas unicriteriales de información (AL)	• Sabe que los ordenadores recogen datos de varios dispositivos de "input", incluidos sensores y software de aplicaciones (AB) • Comprende la diferencia entre hardware y software de aplicaciones, y sus roles dentro del sistema informático (AB)	• Comprende la diferencia entre Internet y un servicio de internet (p.e la "World Wide Web") (AB) • Muestra conciencia de, y utiliza, una variedad de servicios de internet (p. ej. VOIP220) • Reconoce qué es un comportamiento aceptable e inaceptable cuando se utilizan tecnologías y servicios on-line.	• Recopila, organiza y presenta datos e información en forma de contenido digital (AB) • Crea contenido digital para logra una meta determinada, a través de la combinación de distintos paquetes de software y servicios de internet para comunicar con una audiencia más amplia (p. ej. "blogging") (AL) • Hace mejoras apropiadas a sus soluciones basadas en el feedback recibido, y puede comentar acerca del éxito de dicha solución (EV)

	Algoritmos	Programación y Desarrollo	Datos y Representación de Datos	Hardware y Procesamiento	Comunicación y Redes	Tecnologías de la Información
8 – 9 años	• Muestra conciencia de qué tareas son completadas mejor por personas o por ordenadores (EV) • Diseña soluciones a través de la descomposición de un problema, y crea una sub-solución para cada una de estas partes (DE) (AL) (AB) • Reconoce que existen diferentes soluciones para el mismo problema (AL) (AB)	• Comprende la diferencia, y los usos apropiados, entre las sentencias condicionales simples (“If”) y las compuestas (“If/else”) (AL) • Utiliza variables y operadores relacionales221 dentro de un bucle para determinar su terminación (AL) (GE) • Diseña, escribe y depura programas modulares utilizando procedimientos y funciones (AL) (DE) • Sabe que un procedimiento o función puede ser utilizado para ocultar el detalle de una sub-solución (AL) (DE) (AB) (GE)	• Realiza búsquedas más complejas de información (p. ej. utilizando operadores Booleanos y operadores relacionales) (AL) (GE) (EV) • Analiza y evalúa datos e información, y reconoce que una pobre calidad de los datos conduce a resultados no confiables y a conclusiones inexactas (AL) (EV)	• Comprende por qué y cuándo se utilizan los ordenadores (EV) • Comprende las funciones principales de un sistema operativo (DE) (AB) • Conoce la diferencia entre redes físicas, sin cables y móviles (AB)	• Comprende cómo usar de manera efectiva los motores de búsqueda, y sabe cómo se seleccionan los resultados de búsqueda; incluyendo que los buscadores utilizan programas de “rastreo web”222 (AB) (GE) (EV) • Selecciona, combina y usa distintos servicios de internet (EV) • Demuestra un uso responsable de las tecnologías y servicios on-line, y conoce diversas vías para informar de sus inquietudes y preocupaciones.	• Hace juicios sobre contenidos digitales, con el fin de evaluarlos y proponer mejoras dirigidas a audiencias determinadas (EV) (GE) • Identifica la audiencia cuando diseña y crea contenido digital (EV) • Comprende el potencial de las tecnologías de la información para el trabajo colaborativo cuando los ordenadores están conectados en red (GE) • Utiliza criterios para evaluar la calidad de las soluciones; puede identificar mejoras haciendo retoques sobre una solución actual, y futuras soluciones (EV)

	Algoritmos	**Programación y Desarrollo**	**Datos y Representación de Datos**	**Hardware y Procesamiento**	**Comunicación y Redes**	**Tecnologías de la Información**
9 – 10 años	• Comprende que la iteración es la repetición de un proceso, como un bucle (AL) • Reconoce que existen diferentes algoritmos para el mismo problema (AL) (GE) • Representa soluciones utilizando una notación estructurada (AL) (AB) • Puede identificar similitudes y diferencias entre situaciones, y utilizarlo para resolver problemas análogos (reconocimiento de patrones) (GE)	• Comprende que la programación funciona de puente entre las soluciones algorítmicas y las computadoras (AB) • Tiene experiencia práctica en lenguajes textuales de programación de alto nivel223, utilizando librerías estandarizadas de código cuando programa (AB) (AL) • Utiliza un amplio abanico de operadores y expresiones (p. ej. operadores booleanos224), y los aplica en el contexto del control de los programas (AL) • Selecciona apropiadamente entre los distintos tipos de datos (AL) (AB)	• Sabe que los ordenadores y dispositivos digitales utilizan el sistema binario para representar todos los datos (AB) • Comprende cómo los patrones de bits representan números e imágenes (AB) • Sabe que los ordenadores transfieren los datos en sistema binario (AB) • Comprende la relación entre el sistema binario y el tamaño de los archivos (no comprimidos) (AB) • Define diferentes tipos de datos: números reales y booleanos (AB) • Consulta datos en una tabla utilizando un lenguaje de consulta225 (“query language”) habitual (AB)	• Reconoce y comprende la función de las principales partes internas que forman la arquitectura básica de un ordenador (AB) • Comprende los conceptos que hay detrás de los ciclos de ejecución y procesamiento (AB) (AL) • Sabe que hay un amplio abanico de sistemas operativos y de software de aplicaciones para un mismo hardware (AB)	• Comprende cómo los motores de búsqueda clasifican y ordenan los resultados de la misma (AL) • Comprende cómo construir una página web estática utilizando HTML226 y CSS227 (AL) (AB) • Comprende la transmisión de datos entre dispositivos digitales a través de las redes, incluyendo Internet (p. ej. direcciones IP y conmutación de paquetes228) (AL) (AB)	• Evalúa la adecuación de distintos dispositivos digitales, servicios de internet y software de aplicaciones para logras metas determinadas (EV) • Reconoce los aspectos éticos en torno a la aplicación de las tecnologías de la información más allá de la escuela. • Diseña criterios para evaluar de manera crítica la calidad de las soluciones; utiliza dichos criterios para identificar mejoras y puede realizar retoques apropiados a la solución (EV)

	Algoritmos	Programación y Desarrollo	Datos y Representación de Datos	Hardware y Procesamiento	Comunicación y Redes	Tecnologías de la Información
10 – 11 años	• Comprende que una solución recursiva a un problema aplica repetidamente la misma solución a casos particulares del mismo (AL) (GE) • Reconoce que algunos problemas comparten las mismas características y utilizan el mismo algoritmo para ser resueltos (AL) (GE) • Entiende la noción de rendimiento de un algoritmo, y aprecia que algunos algoritmos tienen distinto nivel de rendimiento aplicados en una misma tarea (AL) (EV)	• Utiliza sentencias condicionales anidadas (AL) • Aprecia la necesidad de, y escribe, funciones personalizadas a través del uso de parámetros (AL) (AB) • Conoce la diferencia entre, y usa apropiadamente, procedimientos y funciones (AL) (AB) • Comprende y utiliza la negación en los operadores (AL) • Utiliza y manipula estructuras de datos unidimensionales (AB) • Detecta y corrige errores sintácticos (AL)	• Comprende cómo los números, las imágenes, los sonidos, o los conjuntos de caracteres pueden usar los mismos patrones de bits (AB) (GE) • Realiza operaciones simples utilizando patrones de bits (p. ej. adición binaria) (AB) (AL) • Comprende la relación entre la resolución y la profundidad de color de una imagen digital, incluido su efecto sobre el tamaño del archivo (AB) • Distingue entre los datos usados en un programa particular (una variable) y la estructura de almacenamiento para esos datos (AB)	• Comprende la arquitectura Von Neumann229 en relación con los ciclos de ejecución, incluyendo cómo los datos son almacenados en la memoria (AB) (GE) • Comprende las operaciones y funciones básicas de la memoria localizada y direccionable (AB)	• Conoce los nombres del hardware (p. ej. “hubs”230, “routers”231, “switches”232) y de los protocolos (p. ej. SMTP, iMAP, POP, FTP, TCP/ IP) asociados con las redes de sistemas informáticos. • Utiliza las tecnologías y servicios online de manera segura, y sabe cómo identificar e informar de conductas inapropiadas (AL)	• Justifica la elección de, y combina y usa de manera independiente, múltiples dispositivos digitales, servicios de internet y software de aplicaciones para lograr unas metas determinadas (EV) • Evalúa la fiabilidad del contenido digital, y toma en consideración la usabilidad de los distintos aspectos del diseño visual a la hora de crear artefactos digitales para una audiencia conocida (EV) • Identifica y explica cómo el uso de la tecnología puede impactar sobre la sociedad. • Diseña criterios para los usuarios para evaluar la calidad de las soluciones, utiliza el feedback de los usuarios para identificar mejoras, y puede realizar ajustes adecuados a la solución (EV)

	Algoritmos	Programación y Desarrollo	Datos y Representación de Datos	Hardware y Procesamiento	Comunicación y Redes	Tecnologías de la Información
11 – 12 años	• Reconoce que el diseño de un algoritmo es distinto de su expresión particular en un lenguaje de programación (que dependerá de los comandos y sintaxis de programación disponibles) (AL) (AB) • Evalúa la efectividad de algoritmos y modelos aplicados a problemas similares (AL) (AB) (GE) • Reconoce qué información puede ser filtrada e ignorada para generalizar la soluciones a un problema (AL) (AB) (GE) • Utiliza el razonamiento lógico para explicar cómo funciona un algoritmo (AL) (AB) (DE) • Representa algoritmos utilizando un lenguaje formal-estructurado (AL) (DE) (AB)	• Aprecia el efecto del alcance de una variable (p. ej. una variable local no puede ser accedida desde el exterior de su función) (AB) (AL) • Comprende y aplica la transferencia de parámetros (AB) (GE) (DE) • Comprende la diferencia entre, y usa, los bucles indefinidos233 “pre-tested” (p. ej. “while loop”), y “post-tested” (p. ej. “until loop”) (AL) • Aplica una aproximación modular a la detección y corrección de errores (AB) (DE) (GE)	• Conoce la relación entre la representación de los datos y la calidad de los datos (AB) • Comprende la relación entre el sistema binario y los circuitos eléctricos, incluyendo la lógica Booleana (AB) • Comprende cómo, y por qué, los valores son datos escritos o codificados en muchos lenguajes diferentes, cuando son manipulados durante los programas (AB)	• Sabe que los procesadores tienen conjuntos de instrucciones, y que éstos se relacionan con las instrucciones de bajo nivel que son llevadas a cabo por el ordenador (AB) (AL) (GE) •	• Conoce el propósito del hardware y los protocolos asociados con las redes de sistemas informáticos (AB) (AL) • Comprende el modelo cliente-servidor, incluido cómo las páginas web dinámicas utilizan secuencias de comandos del lado del servidor, y procesan y almacenan datos introducidos por el usuario (AL) (AB) (DE) • Reconoce que la persistencia y perdurabilidad de los datos en Internet requiere de una protección cuidados de la identidad y la privacidad on-line	• Emprende proyectos creativos que recogen, analizan, y evalúan datos para cubrir las necesidades de un determinado grupo de usuarios (AL) (DE) (EV) • Diseña de manera efectiva artefactos digitales para una audiencia amplia y remota (AL) (DE) • Considera las propiedades de los medios cuando los incorpora a sus artefactos digitales (AB) • Documenta el feedback de los usuarios, las mejoras identificadas y los ajustes realizados a la solución (AB) • Explica y justifica cómo el uso de la tecnología impacta en la sociedad, desde una perspectiva tanto social como económica, política, legal, ética y moral (EV)

	Algoritmos	Programación y Desarrollo	Datos y Representación de Datos	Hardware y Procesamiento	Comunicación y Redes	Tecnologías de la Información
12 – 13 años	• Diseña la solución a un problema que depende de soluciones a casos particulares dentro del mismo problema (recursividad) (AL) (DE) (AB) (GE) • Comprende que algunos problemas no pueden ser resueltos computacionalmente (AB) (GE)	• Diseña y escribe programas modulares anidados que refuerzan la reusabilidad utilizando sub-rutinas siempre que sea posible (AL) (AB) (GE) (DE) • Comprende la diferencia entre un bucle "while" ("repetir mientras") y un bucle "for" que usa un contador (AL) (AB) • Comprende y utiliza estructuras de datos bidimensionales (AB) (DE)	• Realiza operaciones utilizando patrones de bits (p.ej. conversión del sistema binario al hexadecimal, sustracción binaria, etc.) (AB) (AL) (GE) • Comprende y puede explicar la necesidad de comprimir los datos, y puede ejecutar métodos simples de compresión (AL) (AB) • Conoce qué es un base de datos relacional234 ("relational database"), y comprende los beneficios de almacenar datos en múltiples tablas (AB) (GE) (DE)	• Tiene experiencia práctica con algún lenguaje de programación de bajo nivel (AB) (AL) (DE) (GE) • Comprende y puede explicar la Ley de Moore (GE) • Comprende y puede explicar la capacidad multitarea de los ordenadores (AB) (AL) (DE)	• Comprende el hardware asociado con las redes de sistemas informáticos, incluyendo WANs y LANs; comprende su propósito y cómo funcionan, incluyendo las direcciones MAC (AB) (AL) (DE) (GE) •	• Comprende los aspectos éticos en torno a la aplicación de las tecnologías de la información, y la existencia de marcos legales que rigen su uso (p. ej. "Data Protection Act", "Computer Misuse Act", "Copyright", etc.) (EV) •

Fuente: (Dorling y Walker, 2015)

20.2. Indicadores en Estados Unidos

Otro texto de referencia que contiene indicadores que permiten evaluar el grado de conocimiento sobre programación es el llevado a cabo por la CSTA (*Computer Science Teachers Association*), sobre estándares de computación (Seehorn y Pirmann, 2016).

Los estándares que este documento refleja no se relacionan directamente con ninguna materia concreta, porque en el currículo estadounidense no existe una asignatura obligatoria de Programación como tal. Estas capacidades están pensadas para ser desarrolladas de forma transversal a lo largo de las distintas etapas educativas, en un itinerario que primero aborde la relación personal del estudiante con la informática, después la informática en la sociedad, y por último la informática en el mundo. El siguiente gráfico muestra este itinerario según lo concibe la CSTA:

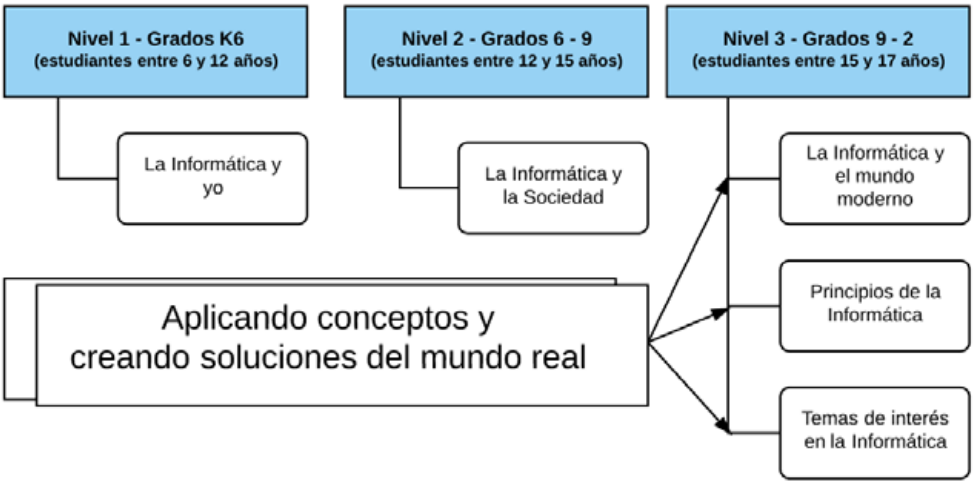


Figura 311. Estructura organizativa de los estándares de aprendizaje asociados a la Informática, según la CSTA.

Fuente: adaptación del original (Seehorn y Pirmann, 2016, p. 13).

La CSTA aborda estos tres niveles distinguiendo cinco aspectos complementarios, en los que clasifica los resultados de aprendizaje. Estos cinco aspectos, o como ellos los denominan, *Strands* (hilos), son:

1. Pensamiento Computacional ("*Computational Thinking*")
2. Prácticas de Informática y Programación ("*Computing Practice and Programming*")
3. Colaboración ("*Collaboration*")
4. Dispositivos Informáticos y de Comunicación ("*Computer and Communication Devices*")
5. Impactos Éticos, Globales y sobre la Comunidad ("*Community, Global and Ethical Impacts"*)

Por el ámbito que compete a esta tesis, en este capítulo sólo se van a mostrar los dos primeros: Pensamiento Computacional, y Prácticas de Informática y Programación.

- **Pensamiento Computacional**

Grados K1 – K2: estudiantes de entre 6 y 9 años (Seehorn y Pirmann, 2016, p. 18). En el ámbito del Pensamiento Computacional, el estudiante será capaz de:

- Utilizar recursos tecnológicos (p. ej. puzles, programas de pensamiento lógico) para resolver problemas apropiados para su edad.
- Utilizar herramientas de escritura, cámaras digitales, y herramientas de dibujo para ilustrar pensamientos, ideas e historias de forma secuencial.
- Entender cómo clasificar y ordenar información en un sentido útil, como, por ejemplo, ordenar a otros estudiantes por edad, sin necesidad de utilizar un ordenador.

- Demostrar cómo los 0s y 1s se pueden utilizar para representar información, como por ejemplo imágenes digitales en blanco y negro .
- Reconocer que el software se crea para controlar las operaciones que realiza el ordenador de manera secuencial.

Grados K3 – K6 – estudiantes entre 9 y 12 años (Seehorn y Pirmann, 2016, p. 18).
En el ámbito del Pensamiento Computacional, el estudiante será capaz de:

- Comprender y utilizar los pasos básicos de la solución algorítmica de problemas (p. ej. planteamiento y exploración del problema, examen de casos de la muestra, diseño, implementación y evaluación).
- Desarrollar una comprensión sencilla de un algoritmo (p. ej. un algoritmo de búsqueda, un algoritmo de clasificación, o una secuencia de eventos), a través de ejercicios sin ordenador.
- Demostrar como una cadena de bits puede ser utilizada para representar información alfanumérica.
- Describir cómo se puede utilizar una simulación para resolver un problema.
- Hacer una lista de sub-problemas a considerar a la hora de afrontar un problema de magnitud mayor.
- Comprender las conexiones entre las ciencias de la computación y otras áreas.

Grados K6 – K9 – estudiantes entre 12 y 15 años (Seehorn y Pirmann, 2016, p. 20).
En el ámbito del Pensamiento Computacional, el estudiante será capaz de:

- Utilizar los pasos básicos de la resolución algorítmica de problemas para diseñar soluciones eficaces a los mismos.
- Describir el proceso de paralelización en la medida que se relaciona con la solución de problemas.
- Definir un algoritmo como una secuencia de instrucciones que puede ser procesada por un ordenador.
- Evaluar de qué manera diferentes algoritmos pueden ser usados para resolver el mismo problema.
- Representar algoritmos de búsqueda y algoritmos de ordenación.
- Describir y analizar una secuencia de instrucciones que está siendo seguida-ejecutada (p. ej. describir el comportamiento de un personaje en un videojuego como objeto controlado por reglas y algoritmos)
- Representar datos en una amplia variedad de formas, incluyendo texto, sonidos, imágenes y números.
- Utilizar representaciones visuales para dar cuenta del estado de un problema o estructura de datos (p. ej. gráficos, cuadros, diagramas de red, diagramas de flujo)
- Interactuar con modelos y simulaciones relativas a contenidos específicos (p. ej. ecosistemas, epidemias, dinámica molecular), para apoyar su aprendizaje e investigación.
- Evaluar qué tipo de problemas pueden ser resueltos utilizando modelos y simulaciones.
- Analizar el grado en el que un modelo computacional representa con exactitud el mundo real.
- Utilizar la abstracción para descomponer un problema en sub-problemas.

- Comprender la noción de jerarquía y abstracción en la computación, incluyendo lenguajes de alto nivel, traducción, conjuntos de instrucciones, y circuitos lógicos.
- Examinar las conexiones entre elementos de las matemáticas y de las ciencias de la computación, incluyendo números binarios, lógicas, variables y funciones.
- Proporcionar ejemplos de aplicaciones interdisciplinares del pensamiento computacional.

Grados K9 – K12 – estudiantes entre 15 y 17 años (Seehorn y Pirmann, 2016, p. 23).

En el ámbito del Pensamiento Computacional, el estudiante será capaz de:

- Utilizar funciones y parámetros predefinidos, clases y métodos para dividir un problema complejo en partes más simples.
- Describir el proceso de desarrollo del software utilizado para resolver los problemas en el mismo (p. ej. diseño, codificación, evaluación y verificación)
- Explicar cómo la secuenciación, la selección, la iteración y la recursividad forman los bloques de construcción de los algoritmos.
- Comparar distintas técnicas para analizar colecciones masivas de datos.
- Describir las relaciones entre las representaciones binarias y hexadecimales.
- Analizar la representación (e intercambio/traducción) entre distintas formas de información digital.
- Describir cómo varios tipos de datos son almacenados en un sistema informático.
- Utilizar la modelización y la simulación para representar y comprender fenómenos naturales.
- Discutir/debatir el valor de la abstracción para gestionar la complejidad de los problemas.
- Describir el concepto de procesamiento en paralelo como una estrategia para resolver problemas de magnitud superior
- Describir cómo la computación comparte características con el arte y la música; en la traducción de una intención humana a un objeto o artefacto.

- **Prácticas de la Informática y la Programación**

Grados K1 – K2 – estudiantes de entre 6 y 9 años (Seehorn y Pirmann, 2016, p. 18).

En el ámbito de las Prácticas de la Informática y la programación, el estudiante será capaz de:

- Utilizar recursos tecnológico-digitales para llevar a cabo investigaciones apropiadas a su edad.
- Utilizar recursos digitales-multimedia adecuados a su etapa evolutiva (p. ej. libros educativos y software educativo), para apoyar su aprendizaje a través del currículum.

- Crear productos digitales-multimedia adecuados a su etapa evolutiva con el apoyo de sus profesores, familia y compañeros de aula.
- Construir un conjunto de sentencias/afirmaciones que sirvan para acometer una tarea sencilla (p. ej. una receta de cocina)
- Identificar trabajos que utilizan la computación y la tecnología.
- Reunir y organizar información utilizando herramientas de mapeo conceptual.

Grados K3 – K6 – estudiantes entre 9 y 12 años (Seehorn y Pirmann, 2016, p. 21).
En el ámbito de las Prácticas de la Informática y la programación, el estudiante será capaz de:

- Utilizar recursos tecnológico-digitales (p. ej. calculadoras, instrumentos de recogida de datos, dispositivos móviles, videos, software educativo, y herramientas web) para la resolución de problemas y el aprendizaje auto-dirigido.
- Utilizar herramientas y periféricos que apoyen y promuevan la productividad personal, remedien los déficits de habilidades, y faciliten el aprendizaje.
- Utilizar herramientas tecnológico-digitales (p. ej. presentaciones multimedia, herramientas web, cámaras digitales, y escáneres) para realizar escritura individual y colaborativa, así como actividades de comunicación y publicación.
- Reunir y manipular datos utilizando una amplia variedad de herramientas digitales.
- Construir/escribir un programa como conjunto de instrucciones “paso por paso” que debe ser llevado a cabo (p. ej. cómo hacer un sándwich de mantequilla y mermelada)
- Implementar soluciones a problemas utilizando lenguajes de programación visuales “por bloques”.
- Utilizar dispositivos computacionales-digitales para acceder a información remota, y comunicarse con otros para apoyar el aprendizaje directo e independiente; y perseguir los intereses personales.
- Navegar a través de páginas web utilizando hipervínculos, y llevar a cabo búsquedas simples utilizando motores de búsqueda.
- Identificar un amplio rango de trabajos que requieren conocimientos de computación.
- Reunir y manipular datos utilizando una amplia variedad de herramientas digitales.

Grados K6 – K9 – estudiantes entre 12 y 15 años
En el ámbito de las Prácticas de la Informática y la programación, el estudiante será capaz de:

- Seleccionar las herramientas y recursos tecnológico-digitales apropiados para acometer una amplia variedad de tareas y solucionar problemas.
- Utilizar una amplia variedad de herramientas y periféricos multimedia-digitales para apoyar la productividad personal y el aprendizaje a lo largo del currículum.

- Diseñar, desarrollar, publicar y presentar productos (p. ej. páginas web, aplicaciones móviles, animaciones) utilizando recursos tecnológico-digitales, que demuestren y comuniquen conceptos curriculares.
- Demostrar la comprensión de algoritmos y sus aplicaciones prácticas.
- Implementar soluciones a problemas utilizando un lenguaje informático de programación, incluyendo: bucles, sentencias condicionales, expresiones lógicas, variables y funciones.
- Demostrar buenas prácticas en lo relativo a la seguridad de la información personal; utilizando contraseñas, encriptación y transacciones seguras.
- Identificar carreras interdisciplinares que son potenciadas por las ciencias de la computación.
- Demostrar una actitud positiva a la resolución de problemas abiertos y a la programación (p. ej. sentirse cómodo con la complejidad, persistencia, adaptabilidad, paciencia, propensión a la experimentación, creatividad, aceptación de retos)
- Recoger y analizar datos que se emitan como resultado de múltiples ejecuciones de un programa informático.

Grados K9 – K12 – estudiantes entre 15 y 17 años (Seehorn y Pirmann, 2016, p. 23).

En el ámbito de las Prácticas de la Informática y la programación, el estudiante será capaz de:

- Crear y organizar páginas web a través del uso de una amplia variedad de herramientas de diseño y programación web.
- Utilizar dispositivos móviles y emuladores para diseñar, desarrollar, e implementar aplicaciones móviles
- Utilizar varios métodos de depuración y testeo para asegurar la corrección de los programas informáticos.
- Aplicar técnicas de análisis, diseño e implementación para resolver problemas (p. ej. usar uno o más modelos cíclicos-iterativos de software)
- Usar APIs (“*Application Program Interfaces*”) y librerías para facilitar soluciones de programación.
- Seleccionar formatos de archivo adecuados para diversos tipos y usos de datos.
- Describir una variedad de lenguajes de programación disponibles para resolver problemas y desarrollar sistemas.
- Explicar el proceso de ejecución de un programa.
- Explicar los principios de la seguridad informática a través del estudio de la encriptación, la criptografía, y las técnicas de autenticación.
- Explorar una variedad de carreras profesionales en las cuales es central la computación.
- Describir cómo las funciones matemáticas y estadísticas, así como los conjuntos y la lógica, son utilizados en la computación.
- Describir técnicas para localizar y recoger conjuntos de datos a pequeña y gran escala.