Universidad de La Laguna

# Investigación e innovación docente en el ámbito de la informática y la comunicación: Programación visual

*Teaching research and innovation on computers and communication scope: Visual programming*

Eduardo Manuel Segredo González

*Trabajo de Fin de Máster*

Área de Lenguajes y Sistemas Informáticos

Facultad de Educación

Universidad de La Laguna

La Laguna, 12 de junio de 2015

Dra. Dª. **Coromoto León Hernández**, con N.I.F. 78.605.216-W profesora Titular de Universidad adscrita al Departamento de Ingeniería Informatica y de Sistemas de la Universidad de La Laguna

**C E R T I F I C A**

Que la presente memoria titulada:

"*Investigación e innovación docente en el ámbito de la informática y la comunicación: Programación Visual*"

ha sido realizada bajo su dirección por Dr. D. **Eduardo Manuel Segredo González**, con N.I.F. 78.564.242-Z.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna, a 12 de junio de 2015.

# Agradecimientos

En primer lugar, me gustaría darle las gracias a Coro, mi directora de Trabajo de Fin de Máster, por todo el tiempo empleado y por su constante apoyo.

Dedico este trabajo a mis padres, a mi hermana y a mi novia por todo su cariño y paciencia, y en general, a toda mi familia por haber estado presente siempre que lo he necesitado.

También querría dedicar este trabajo a mis amigos y compañeros de clase por todos esos buenos momentos que han hecho mucho más llevaderas las cosas.

Por último, me gustaría agradecer al Insituto de Educación Secundaria Domingo Pérez Minik, y en especial, a Betty y al resto de compañeros y compañeras del Departamento de Informática de dicho centro, toda la dedicación y confianza depositadas sin las que no hubiera conseguido llevar a cabo este trabajo.

*Eduardo Manuel Segredo González*

**Resumen**

*Uno de los objetivos del presente trabajo ha sido realizar una recopilación bibliográfica del estado del arte del pensamiento computacional y los lenguajes de programación visual en el ámbito educativo. Se han analizado los lenguajes de programación visual más ampliamente utilizados y se ha seleccionado Scratch para estudiar su influencia en la evolución del pensamiento computacional de alumnado perteneciente a los niveles de primaria y secundaria. En este trabajo se presenta el estudio que desde un inicio se pretendía realizar durante el período de prácticas externas en un centro educativo. Cabe mencionar que llevarlo a cabo por completo no fue viable, debido a restricciones de tiempo y de disponibilidad del alumnado. No obstante, ha sido posible obtener un conjunto de datos inicial de algunos grupos de estudiantes gracias al uso de un instrumento de medición del pensamiento computacional. Este conjunto de datos inicial es el punto de partida para realizar el estudio completo presentado en este trabajo en un futuro cercano.*

***Palabras clave:*** *Pensamiento computacional; Pensamiento algorítmico; Scratch; Lenguajes de programación visual; Educación;*

**Abstract**

*One of the main goals of this work has been to carry out an study about the state of the art concerning computational thinking and visual programming languages in the educational field. The most frequently used visual programming languages have been investigated, and Scratch has been selected in order to study its influence in the computational thinking evolution of K-12 students. The possibility of performing the whole study presented herein during the training time in a high school was non-existent due to restrictions on time and availability of students. We thus present the design of the study we expected to carry out. Nevertheless, it has been possible to obtain a set of initial data coming from several groups of K-12 students by applying a tool that measures their computational thinking. This set of initial data is the starting point to carry out the whole study presented in this work in the near future.*

***Keywords:*** *Computational thinking; Algorithmic thinking; Scratch; Visual programming languages; Education;*

# Índice general

# Índice de figuras

# Índice de tablas

# Lista de acrónimos

| | |
|---|---|
| CEIP | Centro de Educación Infantil y Primaria |
| CSTA | Computer Science Teachers Association |
| ESO | Educación Secundaria Obligatoria |
| FPB | Formación Profesional Básica |
| ICOT | ISTE Classroom Observation Tool |
| IES | Instituto de Educación Secundaria |
| ISTE | International Society for Technology in Education |
| K-12 | Kindergarten-12 |
| LOGSE | Ley Orgánica General del Sistema Educativo |
| MIT | Massachusetts Institute of Technology |
| MOF | Meta-Object Facility |
| NRC | National Research Council |
| OCL | Object Constraint Language |
| PBL | Problem-based Learning |
| PDC | Programa de Diversificación Curricular |
| TIC | Tecnologías de la Información y la Comunicación |
| VPL | Visual Programming Language |
| WOS | Web Of Science |

# Capítulo 1

# Introducción

En este primer capítulo se introducen los conceptos que enmarcan el presente trabajo. Del mismo modo, también se presentan los objetivos que se pretenden alcanzar en el mismo. El capítulo se organiza tal y como se indica a continuación. En la Sección 1.1 se realiza una revisión de la literatura actual sobre el pensamiento computacional y se profundiza en el concepto del pensamiento algorítmico, que es una de las dimensiones del pensamiento computacional. Además, se dan las características del aprendizaje basado en problemas. A continuación, en la Sección 1.2, se presentan los lenguajes de programación visual, así como una taxonomía de los mismos atendiendo a diferentes criterios. Algunas de las características de los lenguajes de programación visual más conocidos y ampliamente utilizados en el ámbito de la educación se describen en la Sección 1.3. Por último, en la Sección 1.4, se comparten los objetivos a abordar en este trabajo.

## 1.1. Pensamiento computacional

En la actualidad la mayoría de los cursos de informática se encuentran enfocados, o bien a la enseñanza del uso de aplicaciones de oficina o *suites ofimáticas*, o bien a la enseñanza de de aplicaciones concretas para un campo de aplicación determinado. En menor medida, también se pueden encontrar cursos en los que se estudian los diferentes componentes hardware de un ordenador o los diferentes sistemas operativos existentes. Sin embargo, la computación no sólo está formada por el hardware y el software que puede encontrarse presente en la vida cotidiana, el cual permite realizar diversas tareas como crear y/o editar archivos de texto, hojas de cálculo o presentaciones, entre otras. La informática es el conjunto de conocimientos científicos y técnicos que hacen posible el tratamiento automático de la información y que permiten implementarlo por medio de los ordenadores. Al igual que todas las ciencias, los fundamentos formales de las ciencias de la computación se basan en las matemáticas.

Se hace necesario introducir cursos en los que se estudien los conceptos claves de las ciencias de la computación, tales como la *abstracción*, los *algoritmos* o la *simulación*, entre otros. Estos cursos deberían ser de carácter general, y no únicamente para ingenieros o especialistas. Es indudable que las matemáticas y la lengua son imprescindibles para iniciar cualquier tipo de estudios, y desde nuestro punto de vista, lo mismo debería de ocurrir con las ciencias de la computación. De hecho, los conceptos de las ciencias de la computación están presentes en casi todas las profesiones, como pueden ser la medicina, el derecho, los negocios, la política, en cualquier otro tipo de ciencia o ingeniería, e incluso, en las artes. Por ello, se está impulsando activamente un nuevo enfoque de enseñanza en todos los niveles educativos que incluya el *Pensamiento Computacional* (del inglés, *Computational Thinking*).

El pensamiento computacional podría describirse como los procesos de pensamiento implicados en la formulación de problemas y representación de sus soluciones, de manera que dichas soluciones puedan ser ejecutadas por un agente de procesamiento de información (ya sea un humano, un ordenador o combinaciones de ambos). Este término se hizo famoso gracias a [Wing, 2006], que indica que el pensamiento computacional se trata de un procedimiento que permite *"la resolución de problemas, el diseño de sistemas y la comprensión de la conducta humana haciendo uso de conceptos fundamentales de la informática. El pensamiento computacional incluye una serie de herramientas mentales que reflejan la amplitud del campo de la informática"*. En su artículo, [Wing, 2006] también remarca que el pensamiento computacional *"representa un conjunto de actitudes y habilidades universalmente aplicables que todos, y no sólo aquellos que se dedican a las ciencias de la computación, estarían ansiosos de aprender y utilizar"*.

Desde entonces, el pensamiento computacional ha atraído su atención en el contexto de la educación primaria y secundaria, principalmente en países anglosajones, como Estados Unidos[1]. No obstante, aún no existe consenso alguno sobre la definición del concepto de pensamiento computacional, habiendo múltiples variantes [Barr and Stephenson, 2011, Brennan and Resnick, 2012, Grover and Pea, 2013]. Por ejemplo, la Sociedad Internacional para la Tecnología en Educación (del inglés, *International Society for Technology in Education (ISTE)*), junto con la Asociación de Profesores de Ciencias de la Computación (del inglés, *Computer Science Teachers Association (CSTA)*), ve el pensamiento computacional como un proceso de solución de problemas que incluye, pero no se limita a las siguientes dimensiones [CSTA, 2011]:

- Formular problemas de manera que permitan usar ordenadores y otras herramientas para solucionarlos.
- Organizar datos de manera lógica y analizarlos.
- Representar datos mediante abstracciones, como modelos y simulaciones.

[1] En el caso de Estados Unidos, *Kindergarten-12 (K-12)* es el término utilizado para referirse al conjunto de cursos que comprenden los niveles de educación primaria y secundaria, cuyo alumnado tiene edades comprendidas entre los 4–6 años hasta los 17–19.

- Automatizar soluciones mediante pensamiento algorítmico, es decir, mediante una serie de pasos ordenados que permitan alcanzar dichas soluciones (véase la Sección 1.1.1).
- Identificar, analizar e implementar posibles soluciones con el objeto de encontrar la combinación de pasos y recursos más eficiente y efectiva.
- Generalizar y transferir ese proceso de solución de problemas a una gran diversidad de estos.

Por otro lado, el Consejo Nacional de Investigación (del inglés, *National Research Council (NRC)*) de Estados Unidos recomienda las matemáticas y el pensamiento computacional como unas de las ocho principales prácticas en el ámbito de las ciencias y la ingeniería [National Research Council, 2012]. En este sentido, las matemáticas y el pensamiento computacional abarcan el uso de herramientas informáticas que permiten representar variables físicas y las relaciones entre ellas. Las dos definiciones anteriores consideran que los estudiantes hacen uso del pensamiento computacional incluso cuando no están utilizando algún tipo de herramienta informática. Contrariamente, la programación si que implica que los estudiantes hagan uso del pensamiento computacional a través de la construcción de artefactos [Resnick et al., 2009, Kafai and Burke, 2013]. De este modo, estas dos definiciones genéricas podrían no ser adecuadas si nos referimos al pensamiento computacional logrado mediante el uso de la programación.

Dado que en este trabajo se pretende analizar el pensamiento computacional a través del uso de la programación, la definición propuesta por [Brennan and Resnick, 2012], teniendo en cuenta el lenguaje de programación *Scratch*, es más apropiada. Scratch es un lenguaje de programación visual (del inglés, *Visual Programming Language (VPL)*) ampliamente utilizado, sobre todo en primaria y secundaria [Kafai et al., 2011, Tangney et al., 2010, Theodorou and Kordaki, 2010, Baytak and Land, 2011]. Scratch comparte características de los lenguajes de programación visual modernos, los cuales son fáciles de aprender, aparte de que proporcionan retroalimentación visual de los programas desarrollados en forma de objetos animados, permitiendo al alumnado la creación de medios interactivos, como por ejemplo, animaciones y juegos. Otro añadido es que Scratch es probablemente el lenguaje de programación visual más apropiado para desarrollar el pensamiento computacional a través de la programación en el ámbito de la educación primaria y secundaria. Considerando Scratch, en [Brennan and Resnick, 2012] se propusieron tres dimensiones para el pensamiento computacional: los conceptos computacionales, las prácticas computacionales y las perspectivas computacionales. La Tabla 1.1 muestra una descripción y ejemplos para cada una de estas tres dimensiones.

Estas dimensiones permiten comprender cómo los estudiantes de primaria y secundaria abordan la programación y también si se encuentran familiarizados con el conocimiento del lenguaje de programación Scratch y su principal precursor, el lenguaje de programación Logo [Logo.media.mit.edu, 2015], cuya primera versión fue creada en 1967 por Seymour Papert. Todo lo anterior incluye el conocimiento sintáctico, semántico y esquemático (conceptos computacionales), así como el conocimiento estratégico (prácticas computaciona-

Tabla 1.1: Dimensiones del pensamiento computacional

| Dimensión | Descripción | Ejemplos |
|---|---|---|
| Conceptos | Conceptos que el programador utiliza | Variables, Bucles, etc. |
| Prácticas | Prácticas de resolución de problemas que surgen durante la programación | Ser incremental e iterativo<br>Testeo y depuración<br>Reutilización<br>Abstracción<br>Modularización |
| Perspectivas | Conocimientos que los alumnos y alumnas tienen sobre si mismos, las relaciones con los iguales, y el mundo tecnológico que les rodea | Expresarse y cuestionarse sobre la tecnología |

les). Teniendo en cuenta las palabras de Harold Abelson en 1982, *"Logo le da nombre a una filosofía de la educación y a una familia de lenguajes de programación en continua evolución que da soporte a dicha filosofía"*. Los entornos de programación Logo que se han venido desarrollando durante las últimas décadas se basan en la filosofía construccionista, la cual se basa a su vez en la teoría constructivista de la educación, y por lo tanto, se han diseñado para dar soporte al aprendizaje construccionista. El construccionismo ve el conocimiento como algo que se crea en la mente de los individuos que aprenden debido a la interacción con otros individuos y con el mundo que les rodea. De hecho, esto último coincide con la dimensión de las perspectivas computacionales propuestas por [Brennan and Resnick, 2012] para el pensamiento computacional (véase la Tabla 1.1). El principal impulsor de las teorías constructivistas fue Jean Piaget, psicólogo suizo y maestro de Seymour Papert, fundador del construccionismo que ha dedicado gran parte de su vida al estudio y documentación de los procesos de aprendizaje en niños y niñas.

### 1.1.1. El pensamiento algorítmico

Tal como se mostró en la sección anterior, en el marco operativo del pensamiento computacional dado por el ISTE y la CSTA, el pensamiento algorítmico es una de las dimensiones del pensamiento computacional. Un algoritmo es un *"método consistente en instrucciones exactamente definidas para resolver un problema determinado"* [Futschek, 2006]. Otros autores, en el contexto de la computación, lo definen como *un número finito de pasos que convierte los datos de un problema (entrada) en una solución (salida)* [Cormen et al., 2009].

El pensamiento algorítmico se considera uno de los conceptos clave que permite a las personas tener fluidez en el uso de las *Tecnologías de la Información y la Comunicación (TIC)*. De este modo, el NRC [National Research Council, 1999] lo describe como un conjunto de conceptos que incluye, entre otros, los siguientes:

- Descomposición funcional.
- Repetición: iteración y/o recursión.

- Organización de datos básicos: registro, matriz, lista, etc.
- Generalización y parametrización.
- Algoritmos vs. programas.
- Diseño desde arriba hacia abajo y refinamiento.

El pensamiento algorítmico es, de alguna manera, un conjunto de habilidades que están conectadas a la construcción y comprensión de algoritmos. Según [Futschek, 2006], este pensamiento incluye las siguientes capacidades o competencias:

1. Analizar problemas dados.
2. Especificar un problema de manera precisa.
3. Encontrar las acciones básicas que son adecuadas para resolver el problema dado.
4. Construir un algoritmo correcto para resolver un problema determinado utilizando las acciones básicas.
5. Pensar en todos los posibles casos tanto especiales como normales de un problema.
6. Mejorar la eficiencia de un algoritmo.

*"El pensamiento algorítmico posee una fuerte componente creativa: la construcción de nuevos algoritmos que resuelvan problemas dados. Si alguien quiere hacer esto, necesita pensar algorítmicamente"* [Futschek, 2006]. Es importante mencionar en este punto que el pensamiento algorítmico no es un mero componente del pensamiento computacional, sino que es una dimensión compleja que se entrelaza con otros componentes. De hecho, las primeras tres capacidades propuestas por [Futschek, 2006] para el pensamiento algorítmico se podrían enmarcar claramente en la dimensión de formulación de problemas de la definición de pensamiento computacional dada por el ISTE y por la CSTA.

Con la finalidad de que el alumnado sea capaz de desarrollar tanto la habilidad para la resolución de problemas, como el pensamiento algorítmico, en este trabajo se propone la utilización de una plantilla para analizar problemas que atiende a las cuatro primeras competencias propuestas por [Futschek, 2006] nombradas en líneas anteriores. El desarrollo de las dos últimas competencias (pensar en casos extremos y mejorar la eficiencia de un algoritmo) no se incluyen en dicha plantilla por tratarse de habilidades que, quizás, sean más adecuadas para niveles más avanzados a primaria y secundaria. Para rellenar esta plantilla, a partir del enunciado de un problema determinado, se pide a los estudiantes que lleven a cabo un análisis de cinco aspectos sobre el problema: la formulación del problema, los datos disponibles, las posibles restricciones, los resultados que se esperan y los procesos o procedimientos involucrados. Este análisis, previo al trabajo con un entorno de programación determinado, como puede ser Scratch, implica el desarrollo de una actividad cognitiva que incluye habilidades como la planificación, la abstracción o la comprensión lingüística, entre otras, todas ellas identificadas en los estudios sobre pensamiento computacional realizados por [Wing, 2006] y por el ISTE y la CSTA [CSTA, 2011]. Esta plantilla se muestra en la Sección A.1 del Anexo A.

Tabla 1.2: Diferencias entre el aprendizaje tradicional y el basado en problemas

| **Aprendizaje tradicional** | **Aprendizaje basado en problemas** |
|---|---|
| El profesorado transmite la información al alumnado | El alumnado toma la responsabilidad de aprender y crear alianzas entre alumnado y profesorado |
| El profesorado organiza el contenido en exposiciones de acuerdo a su disciplina | Los profesores diseñan su curso basado en problemas abiertos |
| El alumnado es visto como receptores pasivos de información | El profesorado busca mejorar la iniciativa de los estudiantes y motivarlos. Ven a los estudiantes como sujetos capaces de aprender solos |
| Las exposiciones del profesor son basadas en comunicación unidireccional | El alumnado trabaja en equipos para resolver problemas, adquieren y aplican el conocimiento en una variedad de contextos |
| El aprendizaje es individual y de competencia | Los alumnos interaccionan y aprenden en un ambiente colaborativo |

### 1.1.2. El aprendizaje basado en problemas

Teniendo en cuenta la definición de [Triantafyllou and Timcenk, 2013], el aprendizaje basado en problemas (del inglés, Problem-based Learning (PBL)) se trata de una estrategia *"centrada en el estudiante, en la que este aprende a través de la experiencia en la resolución de problemas"*. Este método de aprendizaje fue propuesto a finales de la década de los sesenta por Howard Barrows y sus colegas de la Escuela de Medicina de la Universidad McMaster de Ontario, Canadá.

El PBL es un método docente en el que el alumnado, de manera autónoma, tomando protagonismo en su propio aprendizaje, aunque guiado por el profesorado, debe encontrar la respuesta a una pregunta o solución a un problema, de forma que al conseguir resolverlo correctamente suponga que los estudiantes hayan tenido que buscar, entender e integrar y aplicar los conceptos básicos del contenido del problema, así como los conceptos relacionados. De este modo, los estudiantes consiguen elaborar un diagnóstico de las necesidades de aprendizaje, construir su propio conocimiento sobre la materia y trabajar de manera cooperativa.

En sentido estricto, el PBL no requiere que se incluya la solución de la situación o problema presentado. Al inicio de una materia, el estudiante no tiene suficientes conocimientos y habilidades que le permitan resolver el problema de forma efectiva. El objetivo es que el alumnado sea capaz de descubrir qué necesita conocer para avanzar en la resolución de la cuestión planteada (diagnóstico de necesidades de aprendizaje). A lo largo del proceso educativo, a medida que el estudiante progresa, se espera que sea competente en planificar y llevar a cabo intervenciones que le permitan, finalmente, la resolución satisfactoria del problema (construcción del conocimiento), todo ello, trabajando de manera cooperativa.

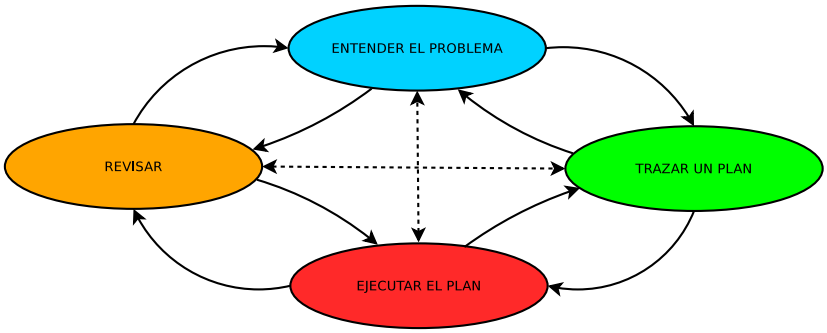



Figura 1.1: Metodología de resolución de problemas propuesta por George Pólya

La estrategia de PBL fue diseñada con varios objetivos, como por ejemplo, ayudar a los estudiantes a desarrollar *"conocimientos flexibles, habilidades para resolver problemas de manera eficaz, el aprendizaje auto-dirigido, habilidades de colaboración eficaces y motivación intrínseca"* [Hmelo-Silver, 2004]. En un ambiente de aprendizaje en el cual se implementa el PBL, el papel del profesorado se ve modificado profundamente respecto a entornos educativos más tradicionales. En este caso, el profesorado se convierte en un guía u orientador que acompaña al alumnado en el proceso de aprendizaje. Por lo tanto, *"el papel del docente es el de guiar y desafiar en el proceso de aprendizaje, en lugar de ser un simple comunicador o transmisor de conocimientos"* [Triantafyllou and Timcenk, 2013]. La Tabla 1.2 muestra las principales diferencias entre los métodos de aprendizaje tradicionales y las metodologías PBL.

Cuando se aplica la estrategia PBL el alumnado debe *"considerarse como un agente activo que participa en la construcción del conocimiento mediante actividades construccionistas, en las que se desarrollan conceptos en diferentes áreas gracias el desarrollo de aplicaciones informáticas"* [Triantafyllou and Timcenk, 2013]. Existen diferentes estrategias para llevar a cabo la resolución de problemas. Por ejemplo, [Morales and Landa, 2004] establecen que el desarrollo del PBL ocurre en ocho pasos o fases:

1. **Leer y analizar el escenario problema**. Se busca que el alumnado entienda el enunciado y lo que se le demanda.

2. **Realizar una tormenta de ideas**. Supone que el alumnado tome conciencia de la situación a la que se enfrenta.

3. **Hacer una lista con aquello que se conoce**. Implica que los estudiantes recurran a aquellos conocimientos de los que ya disponen, a los detalles del problema que conocen y que podrán utilizar para su posterior resolución.

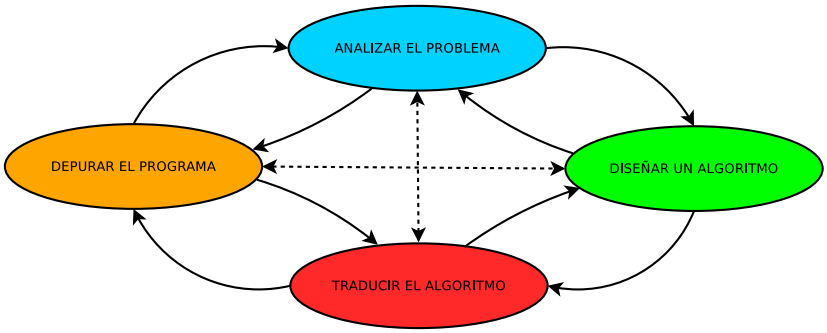



Figura 1.2: Estrategia típica para el desarrollo de programas

4. **Hacer una lista con aquello que no se conoce**. Este paso pretende hacer consciente lo que no se sabe y que necesitarán para resolver el problema, incluso es deseable que puedan formular preguntas que orienten la resolución del problema.
5. **Hacer una lista con aquello que necesita hacerse para resolver el problema**. El alumnado debe plantearse las acciones a seguir para llevar a cabo la resolución.
6. **Definir el problema**. Se trata concretamente el problema que va a resolver y en el que se va a centrar.
7. **Obtener información**. En este punto se espera que los estudiantes se distribuyan las tareas de búsqueda de información.
8. **Presentar resultados**. En este paso se espera que el alumnado que haya trabajado en grupo estudie y comprenda, a la vez que comparta la información obtenida en el paso 7, y por último, que elabore dicha información de manera conjunta para poder resolver la situación planteada.

En el contexto de la informática, la metodología propuesta por [Polya, 1957] (Figura 1.1) es una de las más utilizadas, aparte de ser más simple que la expuesta con anterioridad. Esta metodología consta de cuatro pasos que coinciden, en muchos casos, con los pasos presentes en una estrategia típica de desarrollo de software (Figura 1.2), de ahí que sea apropiada para la resolución de problemas en el ámbito informático.

## 1.2. Lenguajes de programación visual

En el ámbito de las ciencias de la computación, un lenguaje de programación visual permite a los usuarios la creación de programas mediante la manipulación gráfica de los elementos que los constituyen, en vez de utilizar un editor de textos, o incluso, un entorno de

programación visual o gráfico[2]. Esto significa que los programas creados con este tipo de lenguajes tienen asociada una representación visual parcial o total, tal y como pueden ser gráficos, dibujos, animaciones o iconos, entre otros. La programación, por tanto, se lleva a cabo a través del uso de expresiones visuales, disposiciones espaciales de texto y símbolos gráficos, utilizados o bien como elementos de la sintaxis, o bien como notación secundaria. Por ejemplo, algunos lenguajes de programación visual, conocidos como lenguajes de programación de flujo de datos o lenguajes de programación gráfica [Bragg and Driskill, 1994], se basan en la idea del uso de *"cajas y flechas"*, donde las cajas u otro tipo de objetos, considerados entidades, se conectan entre sí mediante flechas, líneas o arcos que representan relaciones entre entidades.

Los lenguajes de programación tradicionales, como Java o C++, usan una representación muy parecida a la manera de *"pensar"* de los ordenadores [Smith et al., 2000]. Por otro lado, los lenguajes de programación visual utilizan una representación que es más cercana al lenguaje humano. Por lo general, los lenguajes de programación visual son menos potentes que los lenguajes tradicionales, dado que suelen ser lenguajes específicos del dominio. Por ejemplo, Scratch se utiliza para crear historias y juegos animados. No obstante, es mejor utilizar lenguajes de programación visual que lenguajes tradicionales para promover de manera más sencilla las tres dimensiones del pensamiento computacional entre el alumnado perteneciente a cursos de primaria y secundaria. Esto se debe, principalmente, a que la sintaxis del lenguaje se simplifica y a que las sentencias y palabras reservadas se parecen mucho más al lenguaje hablado. El alumnado, por lo general, sólo necesita escoger, arrastrar y soltar los elementos gráficos asociados a los elementos que intervienen en su programa. Un ejemplo de esto, se puede observar en la parte derecha de la Figura 1.3, la cual muestra la interfaz que proporciona Scratch para el desarrollo de programas.

Los lenguajes de programación visual ayudan a reducir la carga cognitiva del alumnado, lo que *"les permite concentrarse en la lógica y estructuras involucradas en la programación, en vez de centrarse en la mecánica de escribir programas"* [Kelleher and Pausch, 2005]. De este modo, los lenguajes de programación visual permiten que el alumnado pueda adquirir los conceptos computacionales de una manera mucho más sencilla que aprendiendo complejas sintaxis de programación. Por otro lado, los lenguajes de programación visual también facilitan al alumnado la adquisición de la dimensión de las prácticas computacionales debido a que el resultado final de la programación es visible en forma de objetos animados. Esta visualización hace que las prácticas computacionales, como testear y depurar, sean menos exigentes para el alumnado desde el punto de vista cognitivo. Como consecuencia, los estudiantes de primaria y secundaria son capaces de adquirir de una forma más fácil prácticas de resolución de problemas. Por último, este tipo de lenguajes ha pasado a formar parte de la *"tecnología como compañera en los procesos de aprendizaje"* [Jonassen et al., 2008], y posiblemente, ayude al alumnado de primaria y secundaria a hacer uso de las prácticas computacionales con el objetivo de mejorar su capacidad ge-

[2]No debe confundirse un lenguaje de programación visual, como puede ser Scratch, con un entorno de programación visual o gráfico, como puede ser Microsoft Visual Studio. A pesar de que los lenguajes de programación asociados a dicho entorno se denominan Visual C#, Visual Basic, etc., no se tratan de lenguajes de programación visual.

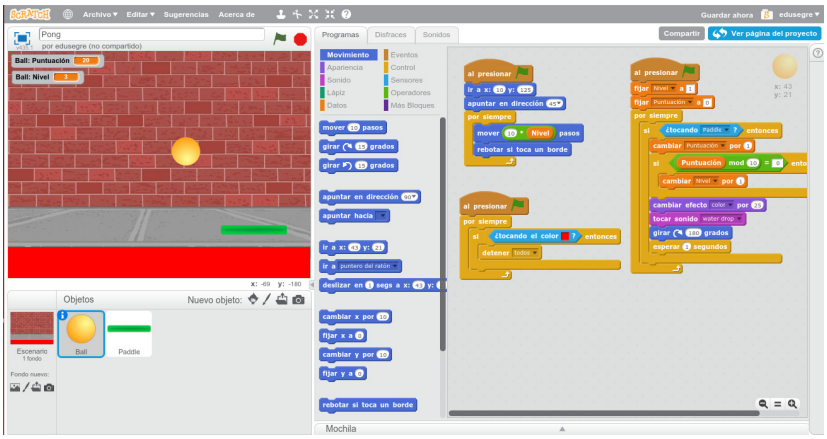



Figura 1.3: Ejemplo de lenguaje de programación visual: Scratch

neral de resolución de problemas [Ratcliff and Anderson, 2011, Lin and Liu, 2012]. Este tipo de lenguajes de programación también hacen que los estudiantes se involucren en mayor medida en la construcción de productos multimedia, habilitando de este modo la programación como medio para que los mismos puedan expresar sus ideas. Esto podría moldear la perspectiva computacional que tiene el alumnado acerca del mundo de la tecnología, o lo que es lo mismo, desarrollar su alfabetización digital para crear, compartir y reusar recursos digitales [Mills, 2010, Hague and Payton, 2011, Ng, 2012], dejando de ser simples consumidores pasivos de tecnología [Resnick et al., 2009]. Es por ello que los lenguajes de programación visual están tomando cada vez más relevancia en el contexto de la educación primaria y secundaria, dado que permiten llevar a cabo dichas experiencias de alfabetización digital [Mills, 2010].

Los lenguajes de programación visual pueden clasificarse de acuerdo a diferentes taxonomías. Considerando el sistema de clasificación propuesto por [Burnett and Baker, 1994][3], los lenguajes de programación visual pueden agruparse, o bien según el paradigma de programación, o bien según la representación visual o gráfica utilizada. Teniendo en cuenta el paradigma de programación, algunas de las categorías se enumeran a continuación:

- **Lenguajes concurrentes**, como por ejemplo A-BITS, un lenguaje de programación visual propuesto por [Ajiro and Tsuchida, 2005], y que permite desarrollar programas concurrentes y asíncronos que requieran de muchas operaciones a nivel de bit.
- **Lenguajes funcionales**, como por ejemplo DEAL [Erwig, 1994], un lenguaje de programación visual que permite llevar a cabo la especificación de algoritmos.

[3]En este enlace se mantiene actualizada gran parte de la bibliografía existente sobre investigación en lenguajes de programación visual haciendo uso de este sistema de clasificación.

- **Lenguajes imperativos**, como por ejemplo ALVIS [Hundhausen et al., 2004], un lenguaje que permite al alumnado principiante de informática la construcción de visualizaciones de los algoritmos que estén siendo objeto de estudio, así como la presentación de estas visualizaciones al resto de compañeros y al profesorado.
- **Lenguajes orientados a objetos**, como por ejemplo MOSL, un lenguaje de programación visual propuesto por [Amelunxen et al., 2006], orientado a la ingeniería dirigida por modelos y que se basa en diferentes estándares como *Meta-Object Facility (MOF)* u *Object Constraint Language (OCL)*, entre otros.

Por otro lado, teniendo en cuenta la representación gráfica utilizada, los lenguajes de programación visual pueden clasificarse en:

- **Lenguajes basados en diagramas**, como el lenguaje VISIONARY propuesto por [Benzi et al., 1999], un lenguaje de programación visual diseñado para realizar consultas en bases de datos relacionales.
- **Lenguajes basados en iconos**, como por ejemplo el lenguaje de programación visual OMEGA [Pinet and Lbath, 2000], que permite llevar a cabo el modelado de sistemas geográficos distribuidos.
- **Lenguajes basados en secuencias de dibujos estáticos**, como por ejemplo Script Cards [Howland et al., 2006], un lenguaje de programación visual destinado a niños y niñas que permite crear historias interactivas sin la necesidad de aprender complejos lenguajes de scripting.
- **Lenguajes basados en sonidos o en el habla**, como por ejemplo Spoken Java [Begel and Graham, 2005], una variante idéntica a Java desde el punto de vista semántico que permite a los desarrolladores de software expresar el código de sus programas articulando palabras del mismo modo que si estuvieran leyendo un fragmento de texto en voz alta.

## 1.3. Lenguajes de programación visual en la educación

En el contexto educativo, varios son los lenguajes de programación visual que han surgido. En las siguientes secciones se enumeran algunos de los más extendidos, mencionando sus características más importantes.

### 1.3.1. Alice

*Alice* [Alice.org, 2015] es un innovador entorno de programación (véase la Figura 1.4) que permite, de un modo sencillo, la creación de animaciones en tres dimensiones para contar historias, jugar a juegos interactivos o compartir vídeos en la Web. Por otro lado, Alice es una herramienta educativa libre diseñada como primera toma de contacto con la programación orientada a objetos. Permite al alumnado el aprendizaje de los conceptos de

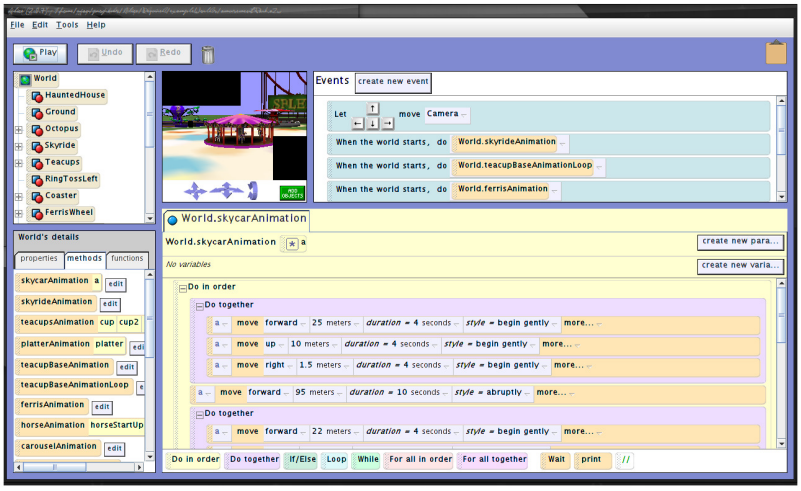



Figura 1.4: Interfaz de usuario de Alice

programación básicos mediante la creación de historias animadas y sencillos videojuegos. En Alice, diferentes objetos en tres dimensiones (personas, animales o vehículos, entre otros) se posicionan en un mundo virtual y los estudiantes crean un programa para animar a todos esos objetos. Alice se desarrolló, primeramente, en la Universidad de Virginia y, posteriormente, en la Universidad Carnegie Mellon, por el grupo de investigación liderado por Randy Pausch [User Interface Group, 1995]. Existen dos versiones principales, Alice 2 y Alice 3. La primera versión va destinada para aprender habilidades relacionadas con el pensamiento lógico y computacional, así como con los principios básicos de la programación, mientras que la segunda versión pone especial énfasis en los conceptos de la orientación a objetos y en una transición completa al lenguaje de programación Java. A continuación, se enumeran algunas de las ventajas de aprender a programar mediante el uso de Alice:

1. La mayoría de lenguajes de programación se diseñan con el objetivo de facilitar la producción de código, introduciendo de este modo, una complejidad adicional. No obstante, Alice se diseñó para enseñar a programar sin que sea necesario aprender complejas sintaxis o semánticas de programación como las que presentan lenguajes como C++. Los usuarios pueden colocar objetos disponibles en la galería de Alice dentro del mundo virtual que han imaginado, para a continuación, programar el comportamiento y la animación de dichos objetos arrastrando y soltando elementos que representan estructuras lógicas del lenguaje de programación. Adicionalmente, los usuarios pueden manipular la cámara y la iluminación del mundo virtual.

2. Alice permite visualizar de manera inmediata las escenas producidas, permitiendo al alumnado comprender de manera fácil la relación entre las sentencias del programa y el comportamiento de los objetos utilizados en la animación. Mediante la manipulación de dichos objetos en el mundo virtual, los estudiantes adquieren experiencia en el uso de los constructos que suelen aprenderse en un curso de iniciación a la programación.

3. Alice viene integrado con su propio entorno de desarrollo, lo que permite que no sea necesario aprender ningún tipo de sintaxis del lenguaje. No obstante, es importante mencionar que Alice se basa en el paradigma de programación orientado a objetos, así como en el modelo de programación orientado a eventos.

4. Alice se diseñó para hacer atractivo el estudio de la programación a grupos de alumnado que normalmente no la utilizan, como alumnado de primaria y secundaria. Alice fomenta la creación de historias, a diferencia de la gran mayoría de lenguajes de programación, que han sido diseñados para la computación. Por último, es importante resaltar que Alice suele utilizarse en cursos de iniciación a la programación, tanto en colegios como en universidades.

Una variante de Alice es *Looking Glass* [Lookingglass.wustl.edu, 2015], desarrollada por la Washington University in St. Louis, y que proporciona algunas novedades respecto a Alice, como un conjunto de animaciones de gran detalle que permite a los usuarios la programación de interacciones sociales entre personajes, una galería de personajes y paisajes en tres dimensiones o la posibilidad de reutilizar animaciones publicadas por otros usuarios para llevar a cabo proyectos más complejos.

### 1.3.2. App Inventor

*Massachusetts Institute of Technology (MIT) App Inventor* [Appinventor.mit.edu, 2015] es una herramienta de programación visual basada en bloques que permite a todo el mundo, incluso a novatos, empezar a programar y construir aplicaciones completamente funcionales para dispositivos Android. Los nuevos usuarios de App Inventor pueden construir y ejecutar su primera aplicación en menos de una hora, y programar aplicaciones más complejas en mucho menos tiempo que haciendo uso de los tradicionales lenguajes basados en texto. Inicialmente desarrollado por el profesor Hal Abelson y un equipo de la división de educación de Google, App Inventor se ejecuta como un servicio Web administrado por personal del *Center for Mobile Learning* del MIT, en el que colaboran el *Computer Science and Artificial Intelligence Laboratory (CSAIL)* del MIT y el *MIT Media Lab*. Además, App Inventor da soporte a una comunidad mundial de aproximadamente unos 3 millones de usuarios ubicados en más de 195 países. Los más de 100.000 usuarios semanales que se encuentran en activo han construido más de 7 millones de aplicaciones para Android. App Inventor, así como los proyectos que han influenciado su desarrollo, principalmente los lenguajes Logo y StarLogo, se basan en las teorías de aprendizaje construccionistas, con la programación como vehículo que lleva a grandes ideas mediante el aprendizaje activo.

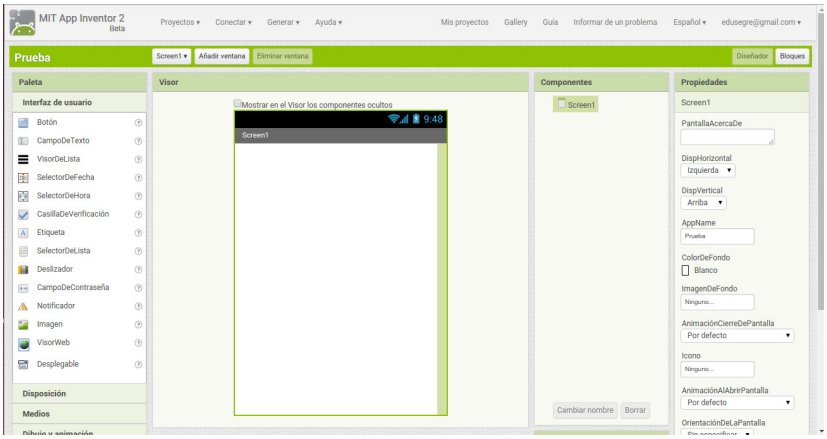



Figura 1.5: Interfaz de usuario de MIT App Inventor – Vista de diseñador

La interfaz de usuario que proporciona App Inventor se encuentra dividida en diferentes secciones y proporciona dos vistas principales. A la primera vista se accede pulsando sobre el botón *Diseñador* (Figura 1.5) que se encuentra en la barra de color verde en la parte superior derecha de la interfaz de App Inventor. En esta vista, a la izquierda de la interfaz puede encontrarse la paleta de componentes, que contiene diferentes componentes como botones, etiquetas o campos de texto, entre otros muchos, los cuales permiten diseñar la interfaz de las aplicaciones. Para añadir componentes a la interfaz de una aplicación simplemente se deben arrastrar los componentes desde la paleta al visor que se encuentra en la parte central de la interfaz de App Inventor. A la derecha de la interfaz se encuentran la lista de componentes y las propiedades de los mismos. Para ver las propiedades de un componente determinado y cambiar sus valores basta con seleccionar un componente de la lista correspondiente.

A la segunda vista se accede pulsando sobre el botón *Bloques* (Figura 1.6) que se encuentra justo al lado del botón Diseñador. A la izquierda de esta vista, se encuentra la paleta de bloques. En la parte superior se encuentran los bloques de propósito general y en la parte inferior los bloques específicos de cada componente que se haya añadido al visor de la vista de diseñador. Por otro lado, en la parte derecha de esta vista se encuentra el visor de bloques. Arrastrando bloques desde la paleta de bloques al visor de bloques se crean los programas.

### 1.3.3. StarLogo

*StarLogo* [Starlogo.mit.edu, 2015] es un lenguaje de simulación basado en agentes desarrollado por Mitchel Resnick, Eric Klopfer y otros compañeros en el *MIT Media Lab*. Este

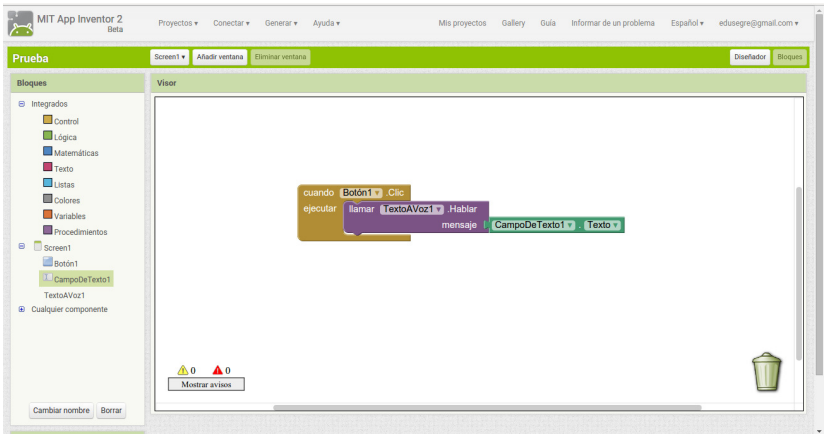



Figura 1.6: Interfaz de usuario de MIT App Inventor – Vista de bloques

lenguaje diseñado para el ámbito educativo, permite al alumnado modelar el comportamiento de sistemas descentralizados, como por ejemplo, bandadas de pájaros, atascos de tráfico o colonias de hormigas, entre otros. StarLogo es una extensión del lenguaje de programación *Logo*. Logo permite la creación de animaciones y dibujos gracias al uso de órdenes o comandos que se le dan a una *tortuga* gráfica. StarLogo extiende esta idea permitiendo a los estudiantes controlar miles de tortugas gráficas en paralelo. Además, StarLogo hace que el mundo de las tortugas se torne computacionalmente activo, es decir, permite escribir programas para las diferentes zonas que conforman ese mundo, y tanto estas zonas como las tortugas pueden interactuar entre sí. Por ejemplo, las tortugas pueden programarse para que *olfateen* el mundo a su alrededor y cambien su comportamiento en función de lo que sean capaces de percibir en las diferentes zonas que lo componen. StarLogo es un lenguaje especialmente adecuado para modelar complejos sistemas descentralizados, los cuales nunca antes habían sido accesibles a gente sin avanzados conocimientos matemáticos y de programación.

La primera versión de StarLogo se podía ejecutar en un ordenador paralelo denominado *Connection Machine 2*. Una versión posterior, conocida como *MacStarLogo*, ya podía ejecutarse en ordenadores Macintosh. La versión actual de StarLogo se encuentra desarrollada en Java y es por ello que funciona en la gran mayoría de ordenadores. También se encuentra una versión denominada *OpenStarLogo* cuyo código fuente se encuentra disponible en línea, aunque la licencia bajo la que se ha liberado no es una licencia de código abierto debido a las restricciones de uso comercial del mismo. En los programas StarLogo se puede hacer uso de tres tipos principales de *actores*:

- **Tortugas**. Los principales habitantes del mundo StarLogo son criaturas gráficas conocidas como *tortugas*. Las tortugas pueden utilizarse para representar casi cualquier

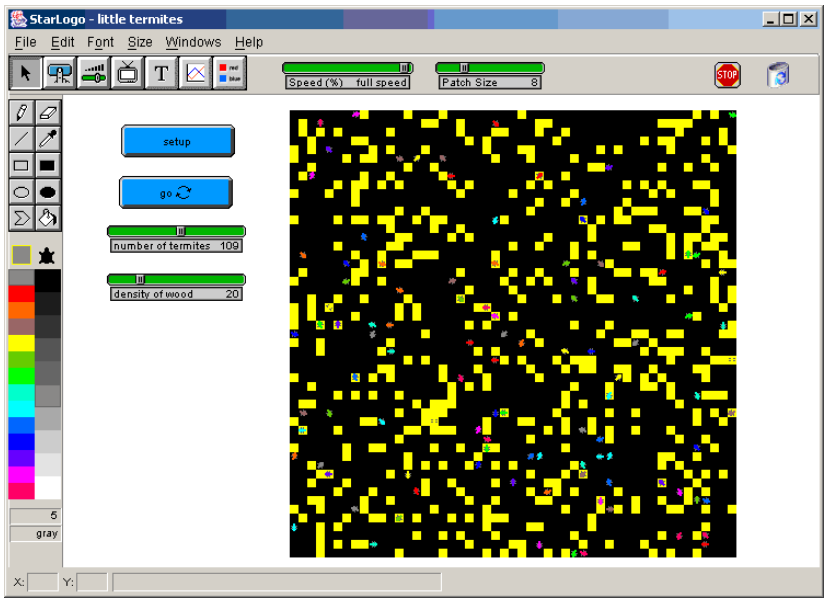



Figura 1.7: Ventana principal de StarLogo

tipo de objeto: una hormiga en una colonia, un coche en un atasco de tráfico o un anticuerpo en un sistema inmune, por ejemplo. Cada tortuga tiene una posición determinada, un título, un color y un lápiz para dibujar. En StarLogo, a diferencia de otros lenguajes como Logo, se pueden controlar las acciones e interacciones de miles de tortugas en paralelo.

- **Zonas**. Las zonas son trozos del mundo en el que viven las tortugas. Las zonas no son objetos meramente pasivos sino que también, al igual que sucede con las tortugas, pueden ejecutar órdenes de StarLogo, así como interaccionar con tortugas y otras zonas. Las zonas se organizan en una malla, correspondiéndose cada zona con un cuadrado del área de gráficos presente en la interfaz de StarLogo.
- **Observador**. El observador ve lo que ocurre en el mundo StarLogo desde una perspectiva aérea. El observador puede crear nuevas tortugas y monitorizar la actividad de las tortugas y zonas existentes.

La interfaz que proporciona StarLogo se encuentra compuesta por diferentes ventanas. La ventana principal de StarLogo, que se muestra en la Figura 1.7, se divide en diferentes secciones. El área de gráficos, que inicialmente es de color negro, es donde las tortugas se mueven y dibujan. Las tortugas se mueven sobre la malla de zonas, y se puede mover a una tortuga directamente arrastrándola con el ratón. El área blanca a la izquierda del área de

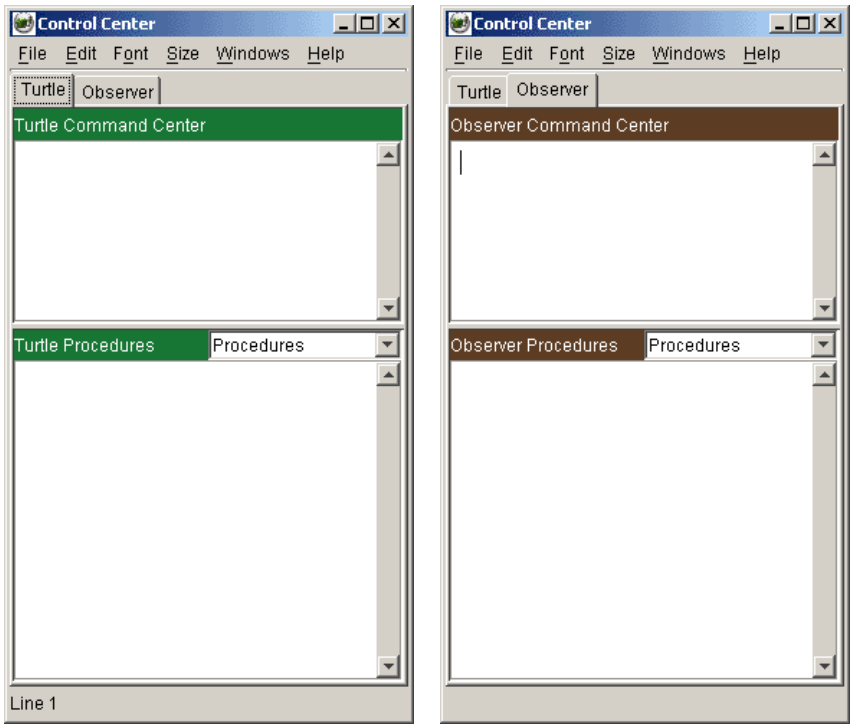



Figura 1.8: Ventanas de control de StarLogo

gráficos se denomina interfaz. En la interfaz se pueden añadir botones, monitores y barras de deslizamiento que permiten interactuar directamente con los programas desarrollados en StarLogo. Para crear y visualizar este tipo de elementos se debe utilizar la barra de herramientas principal, situada en la parte superior de la ventana de StarLogo. Otras barras que se pueden encontrar son las de dibujo y color, situadas en el margen izquierdo de la ventana de StarLogo, y que permiten seleccionar el color, así como diferentes herramientas con las que dibujar.

El área de órdenes de las ventanas de control es donde se escriben las órdenes StarLogo. Por otro lado, en el área de procedimientos, el usuario puede escribir sus propios procedimientos StarLogo. Se puede distinguir entre órdenes específicas para las tortugas y órdenes que sólo pueden ser ejecutadas por el observador, tal y como se muestra en la Figura 1.8.

Otras ventanas disponibles en StarLogo son la ventana de información, que muestra notas de explicación, comentarios e instrucciones; la ventana de salida, que se usa para imprimir información generada por StarLogo (por ejemplo, el comando *print* imprime información

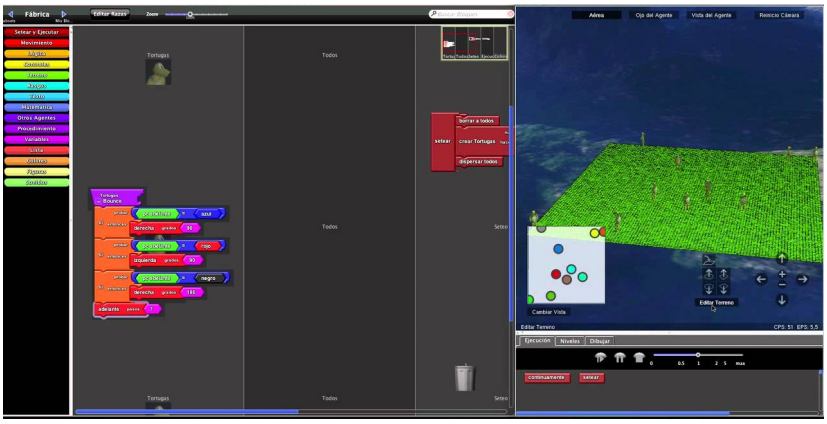


Figura 1.9: Interfaz de usuario de StarLogo TNG

en esta ventana); y una ventana que permite generar gráficos en tiempo real a medida que un programa StarLogo se ejecuta.

StarLogo *The Next Generation (TNG)* [Starlogo.tng.mit.edu, 2015], cuya primera versión se lanzó en julio de 2008, proporciona características adicionales y novedosas respecto a StarLogo, como un mundo tridimensional basado en OpenGL o un lenguaje de programación visual basado en bloques que permite incrementar la facilitad de uso y aprendizaje del lenguaje. Los estudiantes utilizan los bloques para crear sus programas, los cuales se visualizan en el mundo tridimensional. La Figura 1.9 muestra una captura de pantalla de la interfaz proporcionada por esta versión de StarLogo.

La última versión de StarLogo, conocida como *StarLogo Nova* [Slnova.org, 2015], se lanzó en versión beta en el verano de 2014 y ha sido desarrollada por el *MIT Scheller Teacher Education Program* bajo la dirección de Eric Klopfer. StarLogo Nova es una evolución de StarLogo TNG, y como tal, toma de este último el lenguaje de programación visual basado en bloques y el motor de visualización en tres dimensiones para llevarlos a un navegador Web (Figura 1.10). StarLogo Nova introduce varias novedades respecto a StarLogo TNG, de las cuales las más importantes se enumeran a continuación:

- Crear, editar y ejecutar juegos y simulaciones directamente en el navegador, sin necesidad de instalar ningún software en los equipos.
- Colaborar en la creación de proyectos con otros usuarios.
- Incorporar sonidos y modelos tridimensionales en formato Collada propios en los proyectos.
- Organizar el código de una manera más clara.

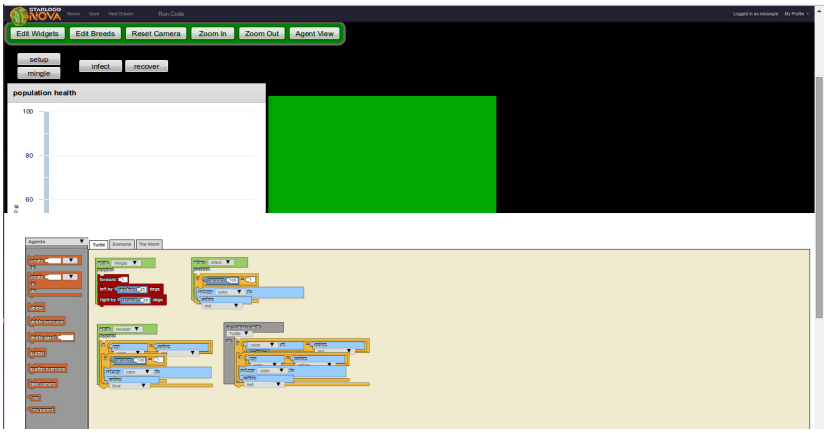



Figura 1.10: Interfaz Web proporcionada por StarLogo Nova

- Programar las interacciones entre agentes de manera más simple, gracias a la incorporación de bloques de detección de colisiones.
- Personalizar los agentes con características diseñadas por los usuarios (como energía, salud, o vidas, entre otros).
- Trabajar con miles de agentes, incluso en ordenadores antiguos de bajas prestaciones.

## 1.4. Objetivos

En el presente trabajo, se pretenden alcanzar los siguientes objetivos:

- Llevar a cabo un estudio bibliográfico sobre el pensamiento computacional, así como un análisis de los lenguajes de programación visual más ampliamente utilizados en el contexto educativo.
- Analizar la literatura existente en busca de trabajos previos que aborden el pensamiento computacional a través de Scratch, especialmente en los niveles de primaria y secundaria.
- Diseñar un estudio que permita conocer el impacto de utilizar un lenguaje de programación visual como Scratch para desarrollar competencias propias del pensamiento computacional.
- Recopilar datos previos aplicando instrumentos de medición del pensamiento computacional que permitan establecer un punto de partida para aplicar el estudio diseñado.

# Capítulo 2

# Scratch y el pensamiento computacional

Este capítulo comienza con una breve introducción a Scratch en la Sección 2.1, exponiendo sus principales características, los recursos que existen para poder aprender a explotar todo su potencial, los componentes más importantes de la interfaz gráfica que proporciona su entorno de programación, y por último, los tipos de bloques disponibles. A continuación, y para finalizar el capítulo, en la Sección 2.2 se lleva a cabo un breve resumen de la bibliografía existente sobre el desarrollo del pensamiento computacional a través del uso de Scratch.

## 2.1. Introducción a Scratch

Scratch es un lenguaje de programación y un entorno gráfico gratuito ampliamente utilizado por estudiantes, docentes y padres para crear medios interactivos, como por ejemplo, historias, juegos y animaciones. Es desarrollado y mantenido por el grupo *Lifelong Kindergarten* del Media Lab del *Massachusetts Institute of Technology (MIT)*, cuyo investigador principal es Mitchel Resnick. Scratch va dirigido, principalmente, a niños de entre 8 y 16 años, y a educadores. No obstante, también suelen utilizarlo personas de diferentes edades, niños más pequeños junto a sus padres, e incluso, universidades como Harvard en sus cursos cero de iniciación a la programación.

El término *scratching* en el ámbito de la informática se refiere a la reutilización de código que puede ser beneficioso y efectivo para otros propósitos diferentes a los que originaron su escritura. Este código debe poder combinarse, compartirse y adaptarse fácilmente a nuevos escenarios, una de las características clave de Scratch. Esta característica, denominada remezcla (del inglés, *remixing*), proporciona a los usuarios de Scratch un modo de acceder a proyectos ya existentes, y así establecer un punto de partida para construir los suyos propios. Por todo lo anterior, el nombre de Scratch surge de la técnica de *scratching* en

una mesa de mezclas, llevando a cabo un símil entre la mezcla de sonidos y la mezcla de proyectos realizados con Scratch.

Este lenguaje se encuentra fuertemente influenciado por los lenguajes Logo y StarLogo, así como por los Lego Mindstorms. Además, cuenta con diferentes evoluciones, como puede ser Snap [Snap.berkeley.edu, 2015], que permite la definición de funciones de orden superior, arrays multidimensionales y sprites orientados a objetos pudiendo utilizar, incluso, el concepto de herencia. Otra variante es Catrobat [Catrobat.org, 2015], un lenguaje de programación visual para móviles y tabletas inspirado en Scratch. La aplicación Pocket Code permite crear, descargar y compartir proyectos desarrollados en Catrobat.

Aunque no es su objetivo principal, Scratch proporciona los elementos necesarios para iniciarse en el mundo de la programación y puede utilizarse en multitud de aplicaciones educativas de propósito construccionista, como la impartición de clases mediante presentaciones animadas o la enseñanza de música y arte interactivo, así como en aplicaciones científicas, como pueden ser simulaciones y visualizaciones de experimentos.

Scratch permite que los usuarios utilicen el paradigma de la programación dirigida por eventos gracias al uso de objetos móviles programables (del inglés, *sprites*). Estos objetos se pueden diseñar en Scratch haciendo uso de un editor gráfico que forma parte de su entorno de programación. No obstante, también pueden importarse desde fuentes externas, como por ejemplo, una cámara Web. En la actualidad, Scratch 2.0 se encuentra disponible como una aplicación Web y como una aplicación de escritorio para los sistemas operativos Windows, Mac OS X y Linux.

Cabe mencionar que Scratch no es sólo un lenguaje de programación visual. También es una comunidad en línea donde se pueden compartir todos los proyectos realizados con esta herramienta. De hecho, es el entorno de programación que más resultados ha producido en el ámbito escolar, con más de 9.753.566 proyectos compartidos en su comunidad a fecha de la escritura de estas líneas.

Existen numerosos recursos para aprender Scratch, entre los que es importante resaltar los siguientes:

- Para recién iniciados, existe una guía paso a paso en Scratch, un tutorial que permite aprender lo básico del lenguaje y su entorno de programación.
- Por otro lado, también se encuentra disponible una guía de inicio.
- Una amplia variedad de videotutoriales.
- Las tarjetas de Scratch, que brindan una forma entretenida y divertida de aprender nuevas funcionalidades y características. La Figura 2.1 muestra una de estas tarjetas.
- Una wiki, hecha por scratchers para scratchers.
- Por último, en la página de ayuda de Scratch, se puede encontrar una visión general de todos los recursos disponibles para aprender Scratch.

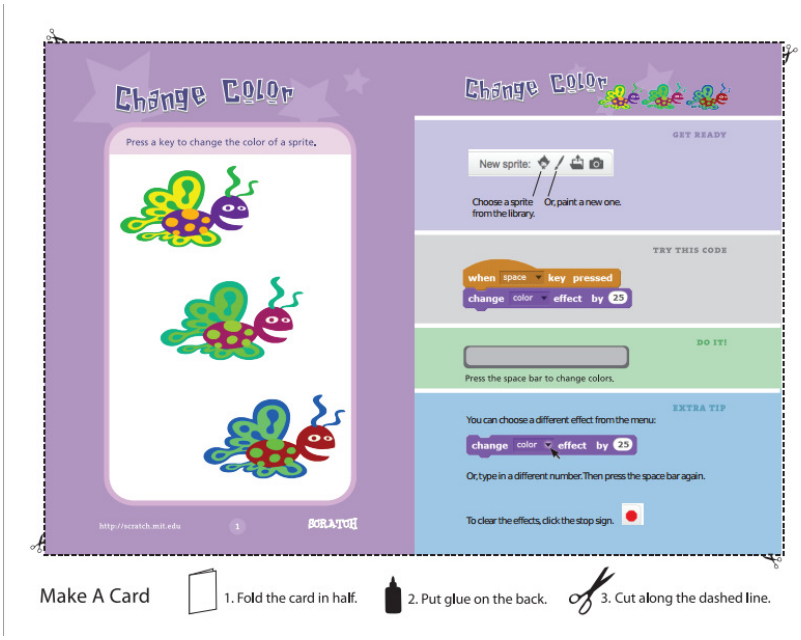



Figura 2.1: Ejemplo de tarjeta de Scratch

Además de todos los recursos anteriores, también es importante mencionar la comunidad en línea ScratchEd, lugar de encuentro de docentes que usan Scratch donde comparten historias, intercambian recursos, hacen preguntas, conocen nueva gente e informan de las últimas novedades sobre eventos relacionados con el mundo Scratch. Esta comunidad se lanzó en julio de 2009, y desde entonces, más de 8.000 educadores de todo el mundo se han unido a ella.

Scratch se lanzó oficialmente en mayo de 2007 e inicialmente tuvo amplia acogida entre quienes venían trabajando con alguna de las versiones de Logo. Pero, en un periodo corto de tiempo, su audiencia se amplió y consiguió cautivar a docentes de todo el planeta que comenzaron a usarlo en sus clases. A partir de mayo de 2013, hubo un cambio radical en el sitio Web de Scratch, que desde esa fecha, desplegó su versión 2.0, la cual funciona completamente en línea. Hasta ese momento, en ese sitio Web los usuarios podían subir los proyectos que habían elaborado con la versión 1.4 de la herramienta, que se tenía que descargar e instalar en el ordenador. Además, también se podían crear galerías con esos proyectos, consultar los proyectos de otros usuarios, o ver proyectos destacados por la comunidad de usuarios, entre otras opciones. A partir del lanzamiento de la versión 2.0, ya se puede crear, editar y visualizar los proyectos directamente en un navegador Web, sin necesidad de descargar e instalar ningún programa en el ordenador.

Las principales diferencias entre Scratch 2.0 y Scratch 1.4 se enumeran a continuación:

- **Crear y editar proyectos en línea**. Esto promueve la reutilización del código.
- **Crear nuevos bloques** a modo de procedimientos, los cuales pueden recibir parámetros. Con esta opción, se puede implementar en los proyectos de Scratch, por ejemplo, la recursividad.
- Existen ahora tres nuevos bloques que permiten **clonar objetos dinámicamente**, mediante programación.
- **La mochila permite copiar y mover con facilidad objetos, disfraces, escenarios y programas de un proyecto a otro**. Si se inicia sesión, se puede abrir la mochila dentro de cualquier proyecto. La mochila facilita la reutilización de elementos de otros proyectos, así como partir de uno o más proyectos para crear nuevos.
- Los **objetos son vectoriales**, lo que permite aumentar su tamaño sin que pierdan resolución.
- El **editor de sonidos** ofrece muchas posibilidades para grabar y editar sonidos e incluirlos en los proyectos Scratch.
- Los **bloques de manejo de vídeo** abren una puerta interesante para nuevos proyectos tipo Microsoft Xbox + Kinect. Se puede utilizar la cámara Web del ordenador para programar la interacción con los objetos de un proyecto mediante el movimiento de las manos o del cuerpo.
- Se pueden **almacenar variables y listas en el sitio Web de Scratch**, lo cual permite crear encuestas en línea o listas de puntuación, entre otro tipo de recursos.
- Se pueden utilizar **nuevos bloques para leer tanto la fecha como la hora del sistema**.

### 2.1.1. Interfaz gráfica de Scratch

Los proyectos Scratch se construyen con objetos móviles programables o *sprites*. Además, la apariencia de cada objeto puede modificarse a través del uso de diferentes disfraces, los cuales pueden dibujarse en el Editor de Pinturas o importarse del disco duro del ordenador. Para decirle a un objeto qué debe hacer se deben encajar un conjunto de bloques para formar pilas denominadas programas o *scripts*. Los bloques de un programa se ejecutan de manera secuencial, desde arriba hacia abajo siguiendo el orden en el que se han apilado.

La Figura 2.2 muestra la interfaz gráfica de Scratch. En su parte superior izquierda se encuentran los diferentes menús. El icono de **Lenguaje**, con forma de globo, permite cambiar el idioma de la interfaz gráfica de Scratch. El menú **Archivo** permite crear un nuevo proyecto, guardar el proyecto actual, cargar/descargar un proyecto desde/hacia el disco duro y deshacer una acción. El menú **Editar** permite recuperar el último bloque,

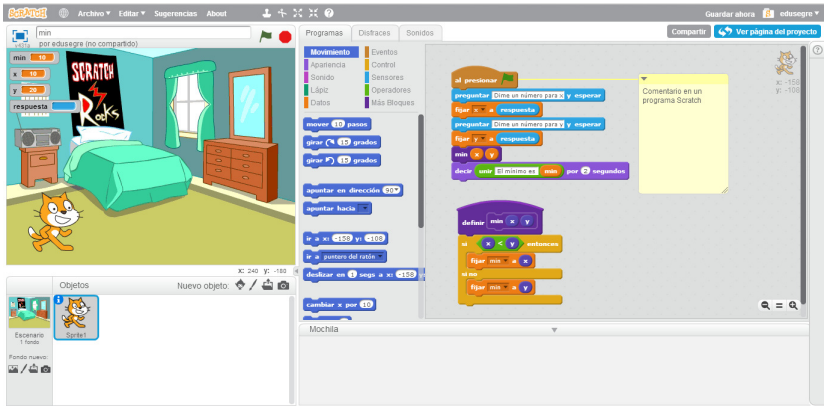


Figura 2.2: Interfaz gráfica de Scratch

programa, objeto, disfraz o sonido borrado, disminuir el tamaño del escenario y ver la ejecución del programa paso a paso. Por último, el menú de **Sugerencias** permite acceder a la página de ayuda de Scratch. entorno A la derecha de los menús se encuentra la **Barra de Herramientas**. Primero debe seleccionarse una herramienta, y posteriormente, se debe hacer clic sobre otros objetos para llevar a cabo una acción. Siguiendo un orden de izquierda a derecha en la barra, las herramientas disponibles son las siguientes:

- **Duplicar**: Duplica objetos, disfraces, sonidos, bloques y programas.
- **Borrar**: Borra Objetos, disfraces, sonidos, bloques y programas.
- **Aumentar el tamaño de un objeto**.
- **Disminuir el tamaño de un objeto**.
- **Ayuda**: Muestra una ventana de ayuda en la parte derecha de la interfaz gráfica.

El **Escenario** (Figura 2.3) es donde las historias, juegos y animaciones cobran vida en Scratch. Tiene unas dimensiones de 480 por 360 píxeles, correspondiéndose el centro al píxel $x = 0$ e $y = 0$. Los diferentes objetos se mueven e interactúan con otros objetos situados en el escenario. Moviendo el ratón por el escenario se puede saber en qué coordenadas se encuentra el puntero exactamente. El botón **Modo Presentación**, situado en la parte superior izquierda del escenario, permite ver los proyectos en modo pantalla completa. La **Bandera Verde**, ubicada en la parte superior derecha del escenario, ofrece una manera fácil para poder comenzar a ejecutar simultáneamente varios programas pertenecientes a un proyecto. Al hacer clic en la bandera verde, todos aquellos programas que comiencen con el bloque *Al presionar la bandera verde* comenzarán a ejecutarse de manera simultánea. Por último, al pulsar la **Señal de Parada**, situada a la derecha de la bandera verde, se detendrá la ejecución de todos los programas del proyecto.


Figura 2.3: Componentes de la interfaz gráfica de Scratch: el escenario


Figura 2.4: Componentes de la interfaz gráfica de Scratch: la lista de objetos


Figura 2.5: Ventana de información de un objeto

Al crear un nuevo proyecto en Scratch, por defecto, siempre se inicializa con el objeto **Gato**. No obstante, se pueden crear nuevos objetos en Scratch. Para ello, pueden utilizarse los siguientes botones:

- Importar un objeto disponible en Scratch.
- Pintar un nuevo objeto usando el **Editor de Pinturas**.
- Importar una imagen guardada en algún directorio del disco duro.
- Capturar una imagen con la cámara del ordenador.

Por otro lado, para eliminar algún objeto, deben seleccionarse las tijeras de la barra de herramientas y hacer clic sobre el objeto que se desee eliminar. Todas las funciones anteriores se encuentran disponibles en la parte superior derecha de la **Lista de Objetos** (Figura 2.4). La lista de objetos muestra imágenes en miniatura de todos los objetos utilizados en un proyecto Scratch. Debajo de cada miniatura se muestra el nombre del objeto, y seleccionando un objeto haciendo clic sobre el mismo pueden editarse sus programas, disfraces y sonidos. Además, haciendo clic con el botón derecho sobre un objeto, éste se puede mostrar/esconder, exportar, duplicar o borrar. Cabe mencionar que no sólo los objetos pueden cambiar su apariencia, sino también los escenarios. Para ver y editar programas, fondos y sonidos asociados al escenario se debe hacer clic en el icono del escenario ubicado a la izquierda de la lista de objetos.

Para acceder a la **información de un objeto**, se debe hacer clic sobre el icono azul de información que aparece en la esquina superior izquierda de su imagen en miniatura en la lista de objetos. Al hacer clic se muestra una ventana (Figura 2.5) con información como el nombre del objeto, el cual puede modificarse, la posición y dirección en la que se moverá el objeto, el estilo de rotación, si el objeto puede ser arrastrado y si se muestra en el escenario. Cabe mencionar que el estilo de rotación permite controlar cómo se visualiza el disfraz a medida que el objeto cambia de dirección. Las opciones disponibles, presentadas de izquierda a derecha en la ventana de información, se enumeran a continuación:

- **Rotar**. El disfraz rota a medida que el objeto cambia de dirección.

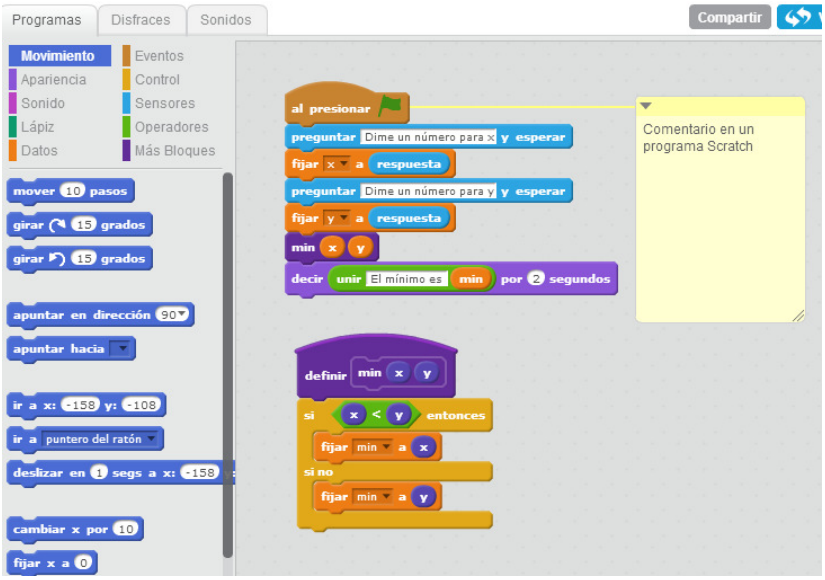



Figura 2.6: Componentes de la interfaz gráfica de Scratch: la paleta de bloques y el área de programas

- **Izquierda–derecha**. El disfraz mira o a la izquierda o a la derecha.
- **No-rotar**. El disfraz nunca rota, a pesar de que el objeto cambie de dirección.

Dos de los componentes más importantes que proporciona la interfaz gráfica de Scratch son la **Paleta de Bloques** y el **Área de Programas**, que se muestran en la Figura 2.6. Para programar un objeto determinado o el escenario, se deben arrastrar bloques desde la paleta de bloques hasta el área de programas. Para ejecutar un bloque determinado se debe hacer clic sobre el mismo y para crear programas se deben ir encajando bloques en el área de programas formando pilas. Se debe hacer clic sobre cualquier punto de una pila de bloques para ejecutar el programa correspondiente, teniendo en cuenta un orden secuencial de los bloques desde arriba hacia abajo. Además, se pueden asociar tantos programas como se desee a un objeto o al escenario. Una funcionalidad muy útil que proporciona Scratch es la de copia de programas entre objetos. Para copiar un programa de un objeto a otro, se debe arrastrar la pila del programa correspondiente hasta la imagen del objeto destino ubicado en la lista de objetos. Por último, para limpiar el área de programas, se debe hacer clic con el botón derecho sobre la misma y seleccionar la opción limpiar en el menú desplegable. También se pueden añadir comentarios a los programas, tal y como se muestra en la Figura 2.6.

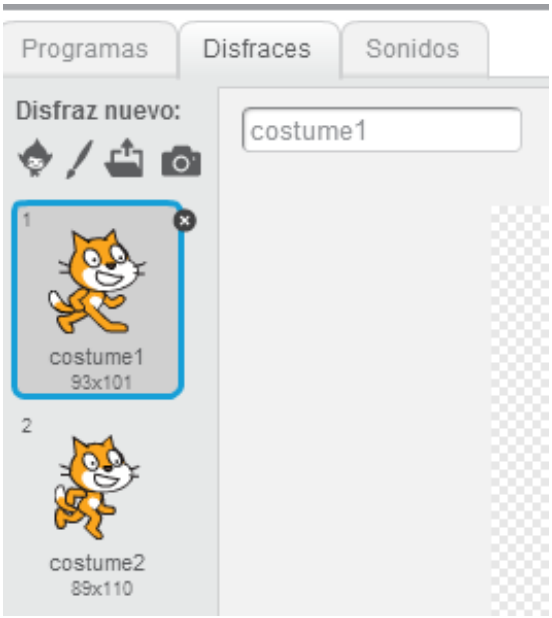



Figura 2.7: Componentes de la interfaz gráfica de Scratch: la pestaña de disfraces

Una vez se haya seleccionado un objeto en la lista de objetos, y haciendo clic en la pestaña **Disfraces** situada en la parte superior izquierda de la paleta de bloques y del área de programas, se pueden ver y editar los disfraces asociados a dicho objeto. En la Figura 2.7 se puede observar como se han asociado dos disfraces diferentes al objeto Gato. Existen cuatro maneras diferentes de crear nuevos disfraces:

- Importar un objeto disponible en Scratch.
- Dibujar un nuevo disfraz en el **Editor de Pinturas**.
- Importar un archivo de imagen desde el disco duro.
- Tomar una foto con la cámara del ordenador.

Scratch reconoce muchos formatos de imágenes: JPG, BMP, PNG o GIF (incluyendo GIF animados), entre otros. Cada disfraz tiene un número que se muestra en la esquina superior izquierda de su imagen en miniatura. El orden de los disfraces puede establecerse arrastrando las imágenes en miniatura de cada uno de ellos, cambiando también el número asignado a los disfraces. Si se hace clic con el botón derecho sobre la imagen en miniatura de uno de los disfraces, o bien se puede convertir en un nuevo objeto, o bien se puede exportar una copia del disfraz a un archivo separado.


Figura 2.8: Componentes de la interfaz gráfica de Scratch: la pestaña de sonidos

Por otro lado, haciendo clic sobre la pestaña de **Sonidos**, ubicada a la derecha de la pestaña de Disfraces (véase la Figura 2.8), y una vez se ha seleccionado un objeto en la lista de objetos, se pueden ver y editar los sonidos del mismo. Para asociar sonidos a un objeto se puede:

- Seleccionar un sonido de la librería de sonidos.
- Grabar un nuevo sonido.
- Importar un archivo de audio.

Al igual que Scratch soporta múltiples formatos para las imágenes, también lo hace para los sonidos, siendo formatos válidos MP3, WAV, AIF o AU, por ejemplo.

Para finalizar, es importante mencionar que Scratch cuenta con un **Editor de Pinturas**, que permite crear y/o editar disfraces y fondos. En lo que respecta a su funcionalidad y apariencia (Figura 2.9), se asemeja mucho a Paint, el editor gráfico que viene por defecto con el sistema operativo Windows, con la salvedad de que el Editor de Pinturas permite manejar imágenes en formato vectorial.

### 2.1.2. Bloques de Scratch

Scratch proporciona tres tipos principales de bloques, a los cuales se puede acceder a través de la Paleta de Bloques:

- **Bloques para apilar**. Estos bloques tienen protuberancias salientes y/o muescas en la parte superior y pueden encajarse unos con otros para formar pilas. Algunos de estos bloques contienen en su interior un área de ingreso de información, que puede ser, por ejemplo, un número o un menú desplegable del que se puede seleccionar una

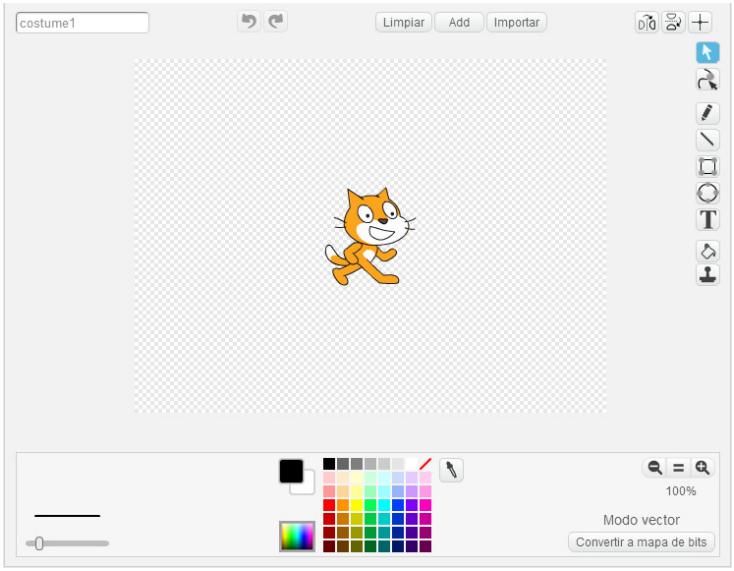



Figura 2.9: Componentes de la interfaz gráfica de Scratch: el editor de pinturas

opción. Otros bloques de este tipo tienen forma de boca en la que se pueden insertar otros bloques para apilar.

- **Sombreros**. Estos bloques tienen la parte superior redondeada, y se ubican como el primer elemento en una pila. Por lo general, esperan a que suceda un evento, como por ejemplo que se presione una tecla, para entonces ejecutar los bloques que se encuentran apilados debajo de ellos.
- **Sensores**. Este tipo de bloques han sido diseñados para encajar en el área de ingreso de información de otros bloques. Los sensores con bordes redondeados informan sobre números o cadenas de texto y encajan en bloques que tienen espacios redondeados o rectangulares. Los sensores con bordes en punta informan valores booleanos y encajan dentro de bloques con espacios que terminan en punta o son rectangulares. Haciendo clic en cualquier sensor se puede ver su valor actual. Algunos de los sensores tienen un *checkbox* que permite mostrar un monitor en el escenario. Este monitor muestra el valor actual del sensor, y a medida que el valor del sensor cambia, también lo hace el monitor de manera automática. Un monitor puede mostrar el valor de un sensor de diferentes formas: en un espacio pequeño de sólo lectura junto al nombre del sensor, en un espacio grande de sólo lectura sin mostrar el nombre del sensor, y por último, en un deslizable que permite modificar el valor del sensor. Esta última modalidad, no obstante, sólo está disponible para monitorizar valores de variables creadas por el usuario.


Figura 2.10: Ejemplos de los tres tipos de bloques disponibles en Scratch: bloques para apilar, sombreros y sensores


Figura 2.11: Monitor de una lista en el escenario y bloques específicos para operar con listas en la Paleta de Bloques

En la Figura 2.10 se pueden observar ejemplos de los tres tipos de bloques mencionados con anterioridad.

Las **listas** en Scratch permiten almacenar tanto números como cadenas de caracteres. Para crear una lista se debe ir a la categoría **Datos** de la Paleta de Bloques y hacer clic en el botón **Crear una lista**. Una vez creada la lista, aparece el monitor de la lista en el escenario, así como un conjunto de bloques específicamente diseñados para operar con listas en la categoría Datos de la Paleta de Bloques (véase la Figura 2.11). Las listas pueden importarse/exportarse desde/hacia un fichero TXT, haciendo clic con el botón derecho sobre el monitor de la lista ubicado en el escenario.

Por último, las **cadenas de caracteres** se forman mediante la unión de letras, palabras u otro tipo de caracteres. En Scratch, las cadenas se pueden guardar en variables o en listas declaradas por el usuario, y existen bloques específicamente diseñados para operar con cadenas. Las cadenas se evalúan con el valor cero en bloques de operaciones matemáticas, así como en bloques que contienen áreas de información numéricas.

Para profundizar en la funcionalidad de cada uno de los bloques que Scratch proporciona se recomienda la lectura de la sección de bloques existente en la wiki de Scratch.

## 2.2. Desarrollo del pensamiento computacional a través de Scratch

A pesar de que Scratch puede utilizarse como lenguaje de iniciación en la programación a los que no tienen experiencia previa en este ámbito, el objetivo principal que Scratch persigue es el de programar para aprender. Este objetivo reencarna el sueño de Seymour Papert, pupilo de Piaget y creador del lenguaje de programación Logo en el MIT como herramienta construccionista. Con posterioridad, Mitchel Resnick utilizó las ideas construccionistas de Papert y el concepto de los bloques Lego para desarrollar Scratch.

Scratch se basa en la teoría construccionista del aprendizaje, la cual viene a decir que el alumnado construye de manera individual sus propios esquemas mentales con el objetivo de comprender el entorno que les rodea. Según Piaget y su teoría constructivista, el aprendizaje es más una reconstrucción del conocimiento que una transmisión del mismo. El construccionismo de Papert parte de la idea anterior y además añade que el aprendizaje es más efectivo cuando el alumnado tiene la experiencia de construir un producto significativo como parte de una actividad determinada. Precisamente es eso es lo que sucede cuando los estudiantes elaboran medios interactivos haciendo uso de un lenguaje de programación visual como Scratch. No sólo construyen productos significativos como animaciones, historias interactivas o videojuegos, sino que también tienen la oportunidad de experimentar durante el proceso de construcción.

Según palabras de Resnick, cuando una persona programa, además de comprender ideas computacionales y matemáticas, también aprende a resolver problemas, diseñar proyectos y comunicar ideas. Las personas no están simplemente aprendiendo a programar, sino que están programando para aprender. Programar es como escribir, y por lo tanto, no sólo los ingenieros en informática deben saber programar, al igual que no sólo deben saber escribir

los novelistas. Del mismo modo que las personas pueden expresarse mediante la escritura, también pueden hacerlo mediante la programación de medios interactivos con Scratch.

Scratch ofrece un contexto y un conjunto de oportunidades para contribuir en la conversación activa sobre pensamiento computacional. Teniendo en cuenta la definición de sus autores en [Brennan and Resnick, 2012], el pensamiento computacional puede definirse como *"procesos de pensamiento involucrados en formular problemas y encontrar sus soluciones de manera que las mismas estén representadas de una forma tal que puedan ejecutarse o llevarse a cabo por un agente que procesa información, ya sea un humano o una máquina"*.

Realizando una búsqueda en bases de datos científicas de reconocido prestigio, el número de trabajos en los que se trata de desarrollar el pensamiento computacional a través del uso de Scratch no es muy elevado. Por ejemplo, llevando a cabo una búsqueda en Scopus de artículos que contengan en su título, resumen y/o palabras clave la combinación de "Scratch" y *"computacional thinking"*, el número de artículos encontrados es de 29. Por otro lado, la búsqueda con los términos "Scratch" y "pensamiento computacional" no devuelve ningún resultado. Si se utiliza la base de datos disponible en la Web Of Science (WOS), los resultados obtenidos son prácticamente los mismos que en Scopus. Por último, realizando una búsqueda en Dialnet de documentos que contengan el término "pensamiento computacional" se obtiene un único artículo, el cual versa sobre el desarrollo del pensamiento computacional en la formación en matemática discreta [Flores, 2011], pero que nada tiene que ver con el desarrollo del pensamiento computacional a través de un lenguaje de programación visual como Scratch. Todo esto es un claro de indicador de que esta línea de investigación es novedosa, quedando aún mucho trabajo por hacer relacionado con la misma.

De los trabajos encontrados, cabe mencionar la revisión que hacen [Lye and Koh, 2014] de la tendencia actual sobre investigación empírica del desarrollo computacional a través de la programación en primaria y secundaria. Los autores proponen que debería aumentarse la presencia del pensamiento computacional en las sesiones de clase pertenecientes a los estudios de primaria y secundaria. En concreto, esta presencia debería centrarse en las prácticas y en las perspectivas computacionales, dos de las dimensiones del pensamiento computacional (véase la Tabla 1.1 de la Sección 1.1). Para ello, podría diseñarse un entorno de PBL construccionista que incluya actividades de procesamiento de información, andamiaje y reflexión para desarrollar tanto las prácticas computacionales, como las perspectivas computacionales. En este trabajo se pretende seguir la propuesta anterior, con el objetivo final de tratar de medir el impacto que un lenguaje de programación visual como Scratch puede tener sobre el pensamiento computacional.

El *andamiaje*, teoría propuesta por Jerome Bruner a finales de los cincuenta, es un proceso en el cual los adultos ofrecen apoyo a los niños y niñas hasta el momento en el que puedan aplicar nuevas habilidades y estrategias de manera independiente. A medida que los niños y niñas comienzan a demostrar destreza en la realización de cierta tarea, la ayuda o asistencia se reduce gradualmente, con el objetivo de que sean más responsables que los adultos de su propio aprendizaje. La teoría del andamiaje se encuentra estrechamente

relacionada con el concepto de *zona de desarrollo próximo* [Vygotsky, 1989], que viene a ser la distancia entre el nivel real de desarrollo, determinado por la capacidad de un niño/a de resolver de manera independiente un problema, y el nivel de desarrollo potencial, determinado por la capacidad de resolver un problema bajo la supervisión y apoyo de un adulto o en colaboración con un igual más competente. Teniendo en cuenta el papel del profesorado, y una sociedad cada vez más inmersa en el uso de las TIC, la utilización de un lenguaje de programación visual como Scratch debería ir acompañada de una batería de actividades que permita promover el desarrollo del pensamiento computacional. Dichas actividades consistirían en la formulación de problemas que sean capaces de generar retos en el aprendizaje del alumnado, con el objetivo de implantar en las aulas la teoría del construccionismo y el concepto de andamiaje. Además, el uso del ordenador como herramienta para resolver los problemas planteados ofrece una experiencia más gratificante.

Los párrafos que se exponen a continuación pretenden llevar a cabo un breve resumen de las publicaciones que mayor interés han despertado durante la búsqueda realizada, sobre todo teniendo en cuenta la relación que mantienen con el estudio que se pretende realizar en este trabajo. Teniendo en cuenta lo anterior, en [Grover et al., 2015] se propone la creación y testeo de un curso introductorio a la informática para alumnado de secundaria titulado "Fundamentos para Pensamiento Computacional Avanzado". El objetivo del curso es preparar y motivar al alumnado perteneciente a los niveles de secundaria en la resolución de problemas mediante el uso de algoritmos. Las conclusiones a las que llegan los autores son que los estudiantes, una vez superan el curso, aumentan significativamente su capacidad de razonamiento algorítmico, son capaces de transferir lo aprendido en un lenguaje de programación visual basado en bloques como Scratch a un lenguaje de programación textual y, por último, interiorizan que la computación es una disciplina tan importante como lo son las matemáticas o la lengua y literatura.

Otro estudio interesante es la comparativa que lleva a cabo [Chang, 2014] de los lenguajes de programación visual Alice y Scratch. Dicha comparativa se realizó en el ámbito de un curso introductorio a la programación para alumnado con bajo rendimiento, en el que se analizó la relación entre su compromiso, entusiasmo y disfrute en el proceso de aprendizaje. El autor concluye que el análisis de las anteriores variables permite a los docentes seleccionar el lenguaje de programación visual más adecuado para tratar de desarrollar habilidades propias del pensamiento computacional en este tipo de alumnado de características peculiares.

Por otro lado, [Boechler et al., 2014] llevan a cabo un análisis de la importancia de incorporar el desarrollo de las habilidades del pensamiento computacional no sólo en los currículos de primaria y secundaria, sino también en niveles universitarios, dado que dichas habilidades son *"un amplificador intelectual esencial para todas las disciplinas científicas y profesionales"*, y sin duda las nuevas generaciones deben aprender a desarrollarlas. Para lograrlo, los autores proponen la utilización de lenguajes de programación visual como Scratch para realizar tareas como el desarrollo de videojuegos. Así podría comprenderse cómo la experiencia previa del alumnado influye en su habilidad para desarrollar el pensamiento computacional a través de la creación de videojuegos en Scratch.

Por último, en [Roscoe et al., 2014] se utiliza, entre otras tecnologías, App Inventor, Scratch y Arduino como herramientas para desarrollar habilidades del pensamiento computacional gracias a la programación de videojuegos y construcción de robots. [Ruthmann et al., 2010] presenta un trabajo similar, pero en este caso, se pretende desarrollar el pensamiento computacional mediante la creación de composiciones musicales en Scratch.

# Capítulo 3

# Estudio del impacto de Scratch en el desarrollo del pensamiento computacional

Este capítulo aborda dos de los objetivos principales del presente trabajo. En primer lugar, se describe la metodología a seguir en el estudio diseñado para medir el impacto de utilizar un lenguaje de programación visual como Scratch en el desarrollo de competencias propias del pensamiento computacional. Por otro lado, se exponen los resultados obtenidos de la aplicación de instrumentos de medición del pensamiento computacional a varios grupos de alumnado con diferentes características. Estos resultados permiten establecer el punto de partida para aplicar en su totalidad el estudio diseñado. El capítulo se organiza tal y como sigue. En la Sección 3.1 se expone la metodología del estudio diseñado, incluyendo los instrumentos de desarrollo y medición del pensamiento computacional que se utilizan en el mismo. A continuación, en la Sección 3.2 se describen los resultados obtenidos de la aplicación de los instrumentos de medición del pensamiento computacional a diferentes grupos de alumnado de primaria y secundaria.

## 3.1. Metodología del estudio

En esta sección se pretende exponer la metodología a utilizar en el estudio diseñado para medir el impacto de utilizar Scratch en el desarrollo del pensamiento computacional en estudiantes pertenecientes a los niveles de primaria y secundaria. También se realiza una exposición de los diferentes instrumentos de recopilación de información que pueden utilizarse para llevar a cabo este estudio.

Se identifican tres fases bien diferenciadas a la hora de desarrollar el estudio de como afecta Scratch a la evolución del pensamiento computacional. Son las siguientes:

1. **Aplicación de instrumentos de medición del pensamiento computacional**. El objetivo de esta primera fase es la recopilación de datos que midan el grado de uso del pensamiento computacional por parte del alumnado. Cabe mencionar que, debido a restricciones de tiempo y de disponibilidad del alumnado, en el presente trabajo sólo se ha podido aplicar esta primera fase del estudio, dejando para un futuro próximo la aplicación de las restantes fases.
2. **Desarrollo del pensamiento computacional en un entorno construccionista de aprendizaje basado en problemas**. Siguiendo las recomendaciones propuestas por [Lye and Koh, 2014], esta fase consiste en la realización de una batería de actividades de procesamiento de información, andamiaje y reflexión por parte del alumnado con el objetivo de desarrollar, principalmente, las dimensiones de las prácticas y las perspectivas computacionales, ambas pertenecientes al pensamiento computacional. Para ello, se utiliza el lenguaje de programación visual Scratch, además de otros instrumentos que serán descritos en la Sección 3.1.2. Esta fase debe aplicarse durante un periodo de tiempo significativo, como por ejemplo, un cuatrimestre o un curso académico completo.
3. **Aplicación de instrumentos de medición del pensamiento computacional**. En esta última fase se vuelven a aplicar los instrumentos de medición del pensamiento computacional que se utilizaron durante la primera fase. El objetivo es estudiar la evolución del alumnado respecto al grado de uso del pensamiento computacional después de haberse visto involucrado en la realización de la batería de actividades propuestas en la fase anterior.

### 3.1.1. Instrumentos de medición del pensamiento computacional

En este trabajo se utilizan los instrumentos de medición del pensamiento computacional aplicados en [López, 2014]. Estos instrumentos constan de un total de cinco actividades que permiten que el alumnado trabaje cuatro de las principales habilidades del pensamiento computacional. Estas habilidades son el pensamiento lógico, la abstracción, el pensamiento algorítmico y la planificación cognitiva. En el presente trabajo se obtuvieron datos de las cinco actividades. No obstante, sólo se muestran los resultados de las actividades *"dibuja y ordena los objetos"* y *"gánate los puntos"*. Se han seleccionado estas dos actividades dado que son las que demandan en mayor grado el uso de la abstracción y del pensamiento algorítmico.

#### Actividad "dibuja y ordena los objetos"

La actividad "dibuja y ordena los objetos" es una variante de la propuesta por [Morra, 2001], que trata de desarrollar, principalmente, la habilidad de la abstracción. En esta actividad aparecen los dibujos de un balón, un lápiz, un libro y un oso, y se le solicita al alumnado que dibuje dichos objetos dentro de dos rectángulos siguiendo un conjunto de instrucciones ya

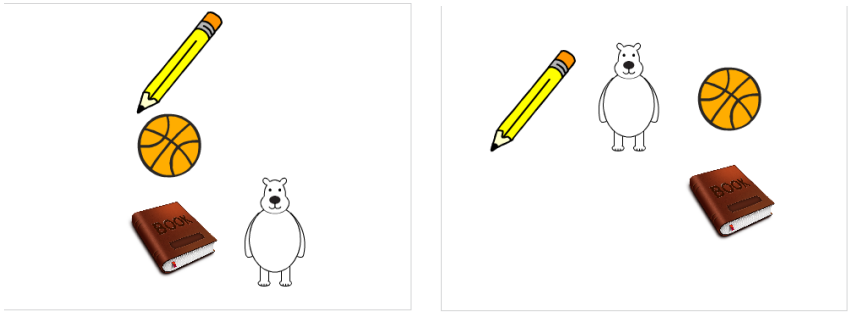


Figura 3.1: Posible solución a la actividad "dibuja y ordena los objetos" para el primer esquema (izquierda) y para el segundo (derecha)

establecido para cada uno de los rectángulos. Esta actividad se muestra en la Sección A.2 del Apéndice A. El resultado que se obtiene de la actividad, por cada estudiante, consiste en dos esquemas con una distribución espacial diferente de los cuatro objetos. Una posible solución a esta actividad se muestra en la Figura 3.1.

Para realizar cada uno de los esquemas, el alumnado tiene que interpretar, de manera secuencial, tres instrucciones previamente definidas con el objetivo de poder establecer la relación espacial entre tres pares de objetos para distribuirlos en el rectángulo. Cuando un estudiante lee una de las instrucciones, integra nueva información que modifica el esquema. Además, toda esta información debe permanecer en la memoria. Las instrucciones se basan en el *"Test de Descripción Espacial"* propuesto por [Ehrlich and Johnson-Laird, 1982], y que consiste en presentar la relación espacial entre varios objetos mediante sentencias secuenciales de la forma "A está debajo de B", "B está a la izquierda de C" y "D está a la derecha de C". Se trata de una descripción sencilla de una relación espacial de dos dimensiones (encima–debajo, izquierda–derecha) entre cuatro objetos que ilustra el uso de un esquema mental.

En esta actividad, el primer esquema utiliza un conjunto de instrucciones semicontinuas de la forma A–C, C–D y A–B. Por otro lado, el segundo esquema utiliza un conjunto de instrucciones discontinuas de la forma D–B, C–A, A–D, es decir, dos objetos diferentes en la primera instrucción, otros dos objetos diferentes en la segunda instrucción y, por último, un objeto de la primera instrucción y otro de la segunda en la tercera instrucción. Según [Ehrlich and Johnson-Laird, 1982], el alumnado menor de 10 años presenta dificultades a la hora de procesar instrucciones discontinuas como consecuencia de la cantidad de información que tienen que memorizar.

Para resolver esta actividad, el estudiante debe procesar, paralelamente, las instrucciones que conforman la secuencia. De este modo, construye el esquema para que todos los objetos quepan en el espacio disponible en el rectángulo. A continuación, debe dibujar los objetos en el orden establecido. En el caso del primer esquema, si se empieza a dibujar desde arriba

hacia abajo y de izquierda a derecha, siguiendo la primera instrucción, podría llegarse a la tercera instrucción sin contar con el espacio suficiente en el rectángulo para dibujar el lápiz encima del balón. En cambio, el segundo esquema pretende clarificar si el alumnado es capaz de transferir la estrategia utilizada durante el desarrollo del primer esquema a la construcción del segundo, o si por el contrario, es capaz de mejorar su estrategia.

Con el objetivo de evaluar el nivel de desempeño de los estudiantes a la hora de acometer esta actividad, se tiene en cuenta la rúbrica mostrada a continuación, la cual se basa no sólo en la cantidad de instrucciones correctamente procesadas e incluidas en el esquema, sino también la transferencia del procedimiento utilizado para resolver satisfactoriamente ambos esquemas. La rúbrica es la siguiente:

- **Básico**. El alumno/a omite información o confunde las relaciones entre objetos en la construcción de cada esquema. Comete distintos tipos de errores y correcciones en ambos esquemas. Hace correcciones, pero sin lograr construir el esquema correcto una vez se han implementado las correcciones. Omite objetos o los repite en posiciones incorrectas teniendo en cuenta las instrucciones dadas. No existen diferencias entre el procedimiento seguido para construir el primer esquema y el procedimiento de elaboración del segundo.
- **Intermedio**. El alumno/a comprende la composición de ambos esquemas y es capaz de transferir el procedimiento utilizado para construir el primer esquema a la construcción del segundo. En los esquemas se integran las relaciones entre objetos de manera apropiada, con correcciones tales como borrar un elemento y dibujarlo nuevamente o dibujarlo al final en el espacio que queda disponible, pero siempre cumpliendo con las instrucciones dadas. Repite objetos, pero se conservan las relaciones espaciales propuestas en las instrucciones. Se da un menor número de errores a la hora de construir el segundo esquema respecto a la construcción del primero.
- **Avanzado**. El alumno/a demuestra que comprende toda la información contenida en las instrucciones y es capaz de construir de manera satisfactoria ambos esquemas. Se dan correcciones, como por ejemplo, el borrar un objeto para volver a dibujarlo en el primer esquema. Esto viene a indicar que el alumno/a dibuja y ordena los objetos del primer esquema a medida que lee cada instrucción. No obstante, para el segundo esquema, lee las tres instrucciones para integrarlas en un único esquema mental.

#### Actividad "gánate los puntos"

Con esta actividad se pretende determinar la capacidad del alumnado para comprender y usar estructuras de control, sobre todo aquellas de tipo condicional. Para poder llevarla a cabo, los estudiantes deben basarse en el cumplimiento de dos condiciones. Por un lado, buscar palabras que tengan las últimas tres letras iguales a las de otra palabra, y por el otro, buscar las que comiencen por 'p' y acaben en 'a'.

Para realizar la búsqueda de palabras que cumplan las dos condiciones anteriores, al alumnado se le proporciona un fragmento, consistente en tres estrofas, del poema *"Pastorcita"* de Rafael Pombo. Además, también se le proporciona dos instrucciones para conseguir puntos, así como una instrucción de asignación de puntos. Esta actividad se muestra en la Sección A.3 del Apéndice A.

Para su resolución, el alumnado debe mantener activas las condiciones durante toda la lectura del poema. Adicionalmente, también debe ser capaz de aplicar el sistema de puntuación. A continuación, se muestra una posible solución a la actividad "gánate los puntos":

```
Pastorcita                                  Puntos

Pastorcita perdió sus ovejas                1+5+5+5_____
¡y quién sabe por dónde andarán!            5+5+5_______
No te enfades, que oyeron tus quejas        5+5_________
y ellas mismas bien pronto vendrán.         5+5+5+5_____

Y no vendrán solas, que traerán sus colas,  5+5+5+5+5+5_
Y ovejas y colas gran fiesta darán.         5+5+5_______
Pastorcita se queda dormida,                1+5+5_______
y soñando las oye balar.                    5___________

Se despierta y las llama enseguida,         5+5_________
y engañada se tiende a llorar.              5___________
No llores, pastora, que niña que llora      1+5+5+5_____
bien pronto la oimos reir y cantar.         5+5_________


                              Puntos totales:  163_________
```

Con el objetivo de evaluar el nivel de desempeño de los estudiantes a la hora de desarrollar esta actividad, se tiene en cuenta la siguiente rúbrica:

- **Básico**. El alumno/a no hace uso de las condiciones para desarrollar la actividad. No se observa una búsqueda sistemática de información. No comprende, aparentemente, las instrucciones o no identifica, dentro del texto, palabras que cumplan con las dos condiciones proporcionadas.
- **Intermedio**. El alumno/a comprende, al menos, una de las dos instrucciones condicionales proporcionadas. No obstante, no se evidencia una búsqueda sistemática de información, ni tampoco una aplicación de la condición en la totalidad del texto. Por ejemplo, el alumno/a sólo señala las palabras que cumplen una de las dos condiciones o señala únicamente algunas palabras, sobre todo aquellas que se encuentran en la primera estrofa o que tienen cierto grado de similitud con las del ejemplo.
- **Avanzado**. El alumno/a comprende ambas condiciones y las aplica de manera meticulosa en la totalidad del texto. El alumnado que muestra este nivel de desempeño

es capaz de manejar con solvencia estructuras de control. En este grupo se encuentran las respuestas en las que se señalan todas las palabras dentro del texto que cumplen con ambas condiciones, obteniéndose la máxima puntuación posible.

### 3.1.2. Actividades en un entorno construccionista de aprendizaje basado en problemas

Tal y como se ha mencionado con anterioridad, en el presente trabajo se sigue la recomendación propuesta por [Lye and Koh, 2014] de uso de un entorno construccionista de aprendizaje basado en problemas para desarrollar las competencias y habilidades propias del pensamiento computacional. En un entorno con estas características, los docentes deben potenciar que el alumnado trabaje, entre otros tipos, actividades de resolución de problemas basadas en andamiaje.

Una de las principales ventajas del andamiaje es la motivación que puede llegar a provocar en los estudiantes, ya que permite enseñarles conceptos y habilidades con las que no se encuentran familiarizados al mismo tiempo que les permite trabajar con herramientas sofisticadas a la par que divertidas. Una de estas herramientas, sin duda, es el lenguaje de programación visual Scratch.

En este estudio se propone, como paso previo a la resolución de un problema mediante el uso de Scratch, la utilización de la plantilla de análisis de problemas que se muestra en la Sección A.1 del Anexo A. Esta plantilla permite formular el problema de manera clara y concisa, fijar los resultados esperados, identificar el conjunto de datos disponible, ser capaz de detectar las restricciones del problema y, por último, proporcionar los pasos a seguir para lograr los resultados esperados partiendo de los datos disponibles.

Por otro lado, en las fases inicial y final del estudio presentado en este trabajo se aplican los instrumentos de medición del pensamiento computacional, los cuales permiten obtener, principalmente, información cuantitativa. Con el objetivo de poder recaudar información cualitativa que complemente a la cuantitativa, se propone el uso de la herramienta de observación de clases del ISTE[1] (del inglés, *ISTE Classroom Observation Tool (ICOT)*). Esta herramienta permite, a través de un conjunto de cuestiones específicamente diseñadas para tal fin, guiar la observación de una serie de aspectos clave relacionados con el grado de integración de las TIC en un aula. La Sección A.4 del Anexo A muestra una captura de pantalla del ICOT.

#### Ejemplos de actividades de resolución de problemas basadas en andamiaje

En el presente trabajo se propone que el alumnado se enfrente a un conjunto de actividades de andamiaje que impliquen la resolución de problemas, como las diseñadas

[1]Esta herramienta puede descargarse desde el siguiente enlace. También se encuentra disponible una versión traducida al castellano. Para aplicarla, se recomienda la lectura de su manual de usuario.

por [Webb and Rosson, 2013]. Algunos ejemplos de este tipo de actividades, las cuales pueden tomarse como modelo para realizar nuevos diseños, se comentan en los siguientes párrafos. No son más que un pequeño extracto de la infinidad de actividades que pueden diseñarse para desarrollar habilidades propias del pensamiento computacional mediante el uso de un lenguaje de programación visual como Scratch.

La primera actividad, consistente en una historia, tiene dos objetivos principales. Por un lado, que el alumnado se familiarice con el lenguaje y la interfaz gráfica proporcionados por Scratch, y por el otro, introducir al alumnado algunos conceptos computacionales y terminología que utilizarán en actividades posteriores. El ejemplo de andamiaje en este caso es una historia en la que varios amigos planean una excursión. Indagando en la historia, los estudiantes descubren que un personaje llamado Billy desaparece de la escena, ya que Billy no está de acuerdo con la idea que ha tenido uno de sus amigos para pasar el día. Para que el alumnado pueda descubrir y resolver estos problemas, se le guía en el uso de un procedimiento de resolución. Por ejemplo, se les ayuda a introducir una nueva escena que ilustre que el grupo de amigos acepta la idea propuesta por Billy para pasar el día. Estas modificaciones incluyen el reemplazo o la adición de bloques para controlar el movimiento de los personajes, lo que dicen, e incluso cuando deberían estar presentes en la escena y cuando no. No obstante, antes de implementar la solución en Scratch, deben hacer uso de la plantilla de análisis de problemas para identificar los personajes y las escenas que aparecen en la historia, los problemas existentes, que aspectos cambiarían en la historia para resolver dichos problemas y los bloques que utilizarían para implementar la solución en Scratch. Por otro lado, los docentes deben documentar todo lo acontecido durante el desarrollo de la actividad a través de la herramienta ICOT.

El ejemplo de andamiaje para la segunda actividad consiste en una historia con la que los estudiantes interactúan mediante un sensor de inclinación de Lego. En esta historia, un personaje llamado Holly se mueve por su habitación recogiendo objetos que debe meter en su maleta. Los estudiantes controlan los movimientos de Holly gracias al uso del sensor de inclinación. Pronto, el alumnado descubre que Holly sólo se puede mover hacia arriba y hacia abajo, y que para corregir el problema debe modificar los bloques de las sentencias condicionales para permitir que Holly también se pueda mover a la izquierda y a la derecha. Al igual que con el otro ejemplo de actividad ilustrado en el párrafo anterior, antes de implementar la solución en Scratch, los estudiantes deben recopilar toda la información posible en la plantilla de análisis de problemas, mientras que el profesorado debe utilizar la herramienta ICOT para documentar en su totalidad el desarrollo de la actividad.

## 3.2. Análisis de los datos recopilados

Esta sección lleva a cabo una descripción, tanto del contexto, como de los grupos de alumnado de primaria y secundaria sobre los que se han realizado las pruebas. Dichas pruebas se han aplicado durante el periodo de prácticas en un centro educativo y han consistido en la aplicación de los instrumentos de medición del pensamiento computacional

expuestos en la Sección 3.1.1, ya que no ha sido posible realizar el estudio completo debido a restricciones de tiempo y de disponibilidad del alumnado. Para finalizar la sección, también se muestran y discuten los resultados obtenidos en las pruebas.

### 3.2.1. Descripción del contexto

El Instituto de Educación Secundaria (IES) Domingo Pérez Minik se encuentra situado en el término municipal de San Cristóbal de La Laguna, y dentro de éste, en la Curva de Gracia, Avenida de los Menceyes nº 72. Su principal ámbito de influencia está formado por los barrios de Gracia, Finca España y la zona alta de La Cuesta. Los barrios de Barrio Nuevo y La Verdellada, también en La Laguna, aunque fuera de la zona de influencia, están muy cercanos. Esta zona presenta unas ciertas singularidades con respecto al resto del municipio. Se trata de zonas que aglutinan una población creciente con variedad de situaciones sociales, cercano a la zona universitaria de alquiler, viviendas unifamiliares y casas de autoconstrucción, donde se respira un alto nivel de seguridad ciudadana. El centro se sitúa al lado del Instituto de Astrofísica de Canarias, del museo de las Ciencias y el Cosmos, y de las facultades de Ciencias Económicas y Empresariales y de Derecho de la Universidad de La Laguna. También es importante mencionar que se encuentra rodeado de zonas ajardinadas y buenas infraestructuras comunicativas a nivel de cercanía de dos paradas de tranvía, buen servicio de guaguas, acceso fácil de vehículos, facilidad de aparcamientos, acceso a autopista del Norte, y también a la conexión con la autopista del Sur. Por último, existen multitud de servicios próximos tal y como pueden ser oficinas bancarias, farmacias, supermercados, comercios y restauración creciente. En su inicio, en el curso 90–91, el IES Domingo Pérez Minik fue creado para dar respuesta a la población estudiantil de la zona de La Verdellada, Finca España y Cercado Mesa, que no podía ser absorbida por los institutos adyacentes. Con la implantación anticipada de la Ley Orgánica General del Sistema Educativo (LOGSE) en el curso 95–96 y la posterior creación de los Distritos Educativos, se produjo una localización del alumnado procedente de Finca España, La Cuesta, La Higuerita y Ofra Barrio Nuevo, ya que los colegios del distrito son el Centro de Educación Infantil y Primaria (CEIP) Clorinda Salazar y el CEIP Las Mantecas.

El municipio de San Cristóbal de La Laguna tiene un total de 153.187 habitantes, ocupando el segundo puesto en número de habitantes tanto de la isla de Tenerife como de toda la provincia de Santa Cruz de Tenerife. Además, a estos datos hay que sumarle la población flotante de estudiantes de otras islas. Del total de habitantes mencionado con anterioridad, 10.430 son extranjeros, lo que viene a representar un 6,8 % de la población total. Los países que mayor número de habitantes aporta al municipio son europeos (Alemania, Francia, Rumanía y Reino Unido), africanos (Marruecos) y americanos (Argentina, Colombia y Ecuador),aunque también hay un colectivo asiático significativo. Otros datos de interés podrían ser los 24 núcleos de población con los que cuenta el municipio, repartidos en una superficie municipal de 102,06 $Km^2$. Teniendo en cuenta todo lo anterior, la densidad de población es de unos 1.501 Hab./$Km^2$.

Comercial y urbana en la zona centro y sur del municipio y turística en la costa norte. En la zona sur del municipio se encuentran los principales polígonos industriales, como Los Majuelos, Las Torres de Taco, Las Mantecas o Las Chumberas, entre otros. En ellos se concentran fabricas—almacenes—de alimentación y manufacturas, exportaciones y el mayor número de grandes centros comerciales de gran tamaño de la rama de la alimentación, bricolaje, automóviles y equipamientos domésticos. En el casco histórico y en los 180 barrios del municipio se dispersa el comercio tradicional y la pequeña industria familiar. Es por todo lo anterior que la mayoría de la población trabaja en el sector servicios. También hay que resaltar que en el municipio se encuentra la Universidad de La Laguna, con aproximadamente 20.323 alumnos matriculados durante el curso 2014–2015, así como el Aeropuerto de Tenerife Norte, el Hospital Universitario de Canarias y el Archivo Histórico Provincial.

### 3.2.2. Descripción de los grupos de alumnado

El alumnado del IES Domingo Pérez Minik proviene de familias trabajadoras, de nivel económico medio. Las características del alumnado que llega son muy diversas, ya que variados son sus orígenes, sus culturas, sus valores, sus objetivos, sus modelos familiares, sus entornos y sus capacidades. En este centro existe alumnado con unos niveles académicos muy buenos, alumnado al que le cuesta llegar a los objetivos, alumnado extranjero, alumnado con necesidades especiales, alumnado con distintas discapacidades, alumnado con recursos y alumnado sin ellos, alumnado muy joven y alumnado que supera los veinte años. Sin duda, una amplia variedad de perfiles que tienen cabida porque son un ejemplo en joven del entorno que nos rodea. En las familias, al igual que con el alumnado, encontramos mucha variedad, si bien en los últimos años viene dándose una mayor cantidad de modelos familiares, desestructuración y padres menos colaboradores con el centro, debido esto último sin duda a sus circunstancias personales y profesionales.

Los instrumentos de pensamiento computacional se aplicaron a un total de 54 alumnos y alumnas pertenecientes a tres niveles diferentes:

- **Sexto curso de Primaria**[2]. El alumnado tiene edades comprendidas entre los 11 y los 13 años. Hay un total de 19 estudiantes con 11 años, 9 estudiantes con 12 años y 2 estudiantes de 13 años.
- **Cuarto curso del Programa de Diversificación Curricular (PDC)**. En este caso el alumnado tiene edades comprendidas entre los 15 y los 19 años. En total, hay 2 estudiantes de 15 años, tres estudiantes de 16 años, 8 estudiantes de 17 años, 3 estudiantes de 18 años, y por último, un estudiante de 19 años. Es importante destacar que parte del alumnado de este curso presenta dificultades específicas en el aprendizaje, por lo que resulta interesante estudiar el grado de uso del pensamiento computacional por parte de este tipo de niños y niñas.

[2]Este grupo de estudiantes no pertenece al IES Domingo Pérez Minik, sino al CEIP Tomé Cano, un centro ubicado en un contexto distinto al del primer centro.

Tabla 3.1: Número de alumnos y alumnas según nivel de desempeño para la tarea "dibuja y ordena los objetos"

| | **Básico** | **Intermedio** | **Avanzado** | **Total** |
|---|---|---|---|---|
| **Sexto de Primaria** | 11 | - | 19 | 30 |
| **Cuarto de PDC** | 1 | 3 | 13 | 17 |
| **Primero de FPB** | 2 | 1 | 4 | 7 |
| **Total** | 14 | 4 | 36 | 54 |

- **Primer curso de la Formación Profesional Básica (FPB) en Informática y Comunicaciones**. En este nivel los estudiantes tienen edades comprendidas entre los 16 y los 18 años, habiendo 4 estudiantes con 16 años, 2 estudiantes con 17 años y 1 estudiante con 18 años. Al igual que ocurre con el grupo de cuarto curso de PDC, en este grupo existe alumnado con dificultades específicas en el aprendizaje. No obstante, la gran mayoría se ha matriculado en este ciclo para no quedar excluidos del sistema educativo. Los principales motivos suelen ser un alto grado de absentismo en etapas de escolarización previas o un bajo rendimiento académico que imposibilita titular.

### 3.2.3. Resultados y discusión de las pruebas

En esta sección se muestra información cuantitativa obtenida de la aplicación de los instrumentos de medición del pensamiento computacional a los grupos de alumnado descritos en la sección anterior. Esta información cuantitativa permite tener una visión global del nivel de desarrollo de habilidades relacionadas con el pensamiento computacional para cada uno de los grupos de estudiantes analizado, los cuales presentan características muy diversas entre sí. Por otro lado, la aplicación en su totalidad del estudio expuesto en este trabajo permitiría obtener información cualitativa complementaria a la cuantitativa que podría utilizarse para realizar un análisis mucho más exhaustivo.

#### Análisis de resultados obtenidos en la prueba "dibuja y ordena los objetos"

La Tabla 3.1 muestra información sobre el número de alumnos y alumnas clasificados según su nivel de desempeño en la actividad "dibuja y ordena los objetos". Por otro lado, la Figura 3.2 muestra la misma información pero en formato de porcentajes acumulados.

Se debe recordar que la actividad "dibuja y ordena los objetos", que no sólo desarrolla la habilidad de abstracción, sino también las de planificación y paralelismo, requiere que el alumnado procese paralelamente la secuencia de instrucciones proporcionada para poder construir un esquema de manera que al dibujarlo quepa dentro del espacio delimitado por un rectángulo. Esta actividad busca establecer la cantidad de información que el alumno o alumna es capaz de retener en su memoria. De este modo, el nivel avanzado

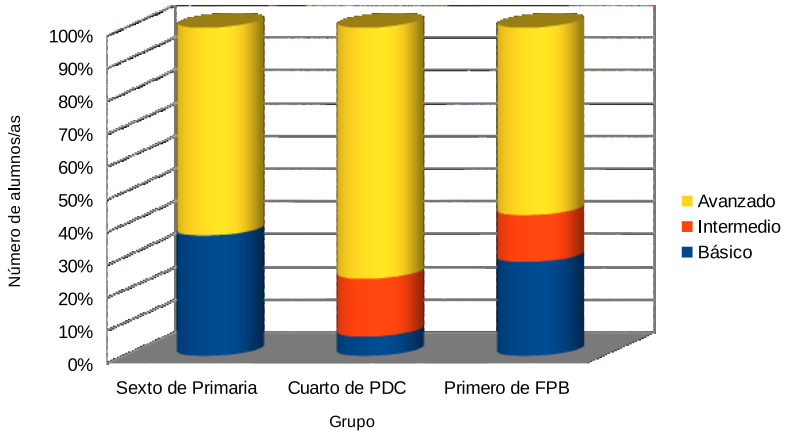



Figura 3.2: Porcentaje acumulado de alumnos y alumnas según nivel de desempeño para la actividad "dibuja y ordena los objetos"

lo muestran aquellos estudiantes capaces de proyectar en su mente el esquema completo antes de dibujarlo.

Observando los resultados, puede deducirse que, como norma general, la mayor parte del alumnado perteneciente a los diferentes grupos analizados presentó un grado de desempeño avanzado. Esto significa que el alumnado fue capaz de comprender e integrar toda la información proporcionada en las instrucciones en un único esquema. La consecuencia directa de lo anterior es que, de manera extendida, el alumnado demostró un alto grado de desarrollo de habilidades relacionadas con la abstracción, la planificación y el paralelismo.

Sin embargo, también hay que decir que un número significativo de estudiantes de cada grupo mostró un grado de desempeño básico. Los estudiantes con estas características no fueron capaces de comprender toda la información contenida en las instrucciones (omitieron información) o de establecer las relaciones necesarias para ubicar correctamente los objetos en el espacio disponible, lo que se traduce en un déficit en el desarrollo de habilidades relacionadas con la abstracción.

Teniendo en cuenta los niveles de desempeño obtenidos por cada grupo de alumnado por separado, es importante mencionar que los grupos de sexto de primaria y cuarto del PDC cuentan con un mayor porcentaje de alumnado que mostró un nivel de desempeño avanzado que el grupo de primero de la FPB en informática y comunicaciones. Las diferencias, no obstante, no fueron significativas. Este hecho, en parte comprensible, podría deberse a la falta de motivación detectada en gran parte del alumnado perteneciente al grupo de la FPB en informática y comunicaciones. La gran mayoría de veces en las que se impartió docencia a dicho grupo durante el periodo de prácticas en el centro, el alumnado presentó una

Tabla 3.2: Número de alumnos y alumnas según nivel de desempeño para la tarea "gánate los puntos"

| | **Básico** | **Intermedio** | **Avanzado** | **Total** |
|---|---|---|---|---|
| **Sexto de Primaria** | 17 | 13 | - | 30 |
| **Cuarto de PDC** | 5 | 12 | - | 17 |
| **Primero de FPB** | - | 7 | - | 7 |
| **Total** | 22 | 32 | - | 54 |

actitud muy pasiva, a excepción de la sesión durante la que desarrollaron las actividades pertenecientes al instrumento de medición del pensamiento computacional, en la que su comportamiento fue muy correcto y mostraron bastante interés. Muy probablemente, este tipo de actitud en el aula se deba a la problemática particular de cada alumno y alumna, lo que viene a suponer un impacto negativo en su rendimiento, y por lo tanto, en el desarrollo de cualquier tipo de actividad a la que tengan que enfrentarse.

Por último, los resultados también indican que la edad del alumnado perteneciente a los diferentes grupos analizados, así como el contexto o centro en el que desempeñan su actividad, no son, aparentemente, variables que afecten al grado de desarrollo de las habilidades relacionadas con la abstracción. No obstante, para dar un mayor soporte a todas las afirmaciones expuestas con anterioridad debería llevarse a cabo el estudio completo presentado en este trabajo. De este modo, se podría arrojar un poco más de luz a estas y otras cuestiones.

#### Análisis de resultados obtenidos en la prueba "gánate los puntos"

La Tabla 3.2 muestra información sobre el número de alumnos y alumnas clasificados según su nivel de desempeño en la actividad "gánate los puntos". La Figura 3.3, por otro lado, muestra la misma información pero en formato de porcentajes acumulados.

Es importante recordar que la actividad "gánate los puntos" requiere que los estudiantes sean capaces de interpretar y aplicar reglas condicionales, manteniéndolas activas durante la lectura del texto, con el objetivo de identificar las palabras que cumplan con las condiciones predefinidas. Esta actividad, al igual que la anterior, también requiere del alumnado la habilidad de comprensión lingüística, de modo que pueda entender el enunciado proporcionado.

La primera conclusión que puede extraerse de los resultados es que, a diferencia de lo observado para la primera, ningún alumno o alumna mostró un grado de desempeño avanzado en el desarrollo de esta actividad, independientemente del grupo analizado. Repasando la rúbrica establecida, lo anterior indica que el alumnado no fue capaz de comprender ambas condiciones y/o de aplicarlas a la totalidad del texto, es decir, no fue capaz de manejar con solvencia sentencias condicionales. Sin embargo, un número significativo de estudiantes de cada grupo presentó un nivel de desempeño intermedio, es decir, el alumnado pertene-

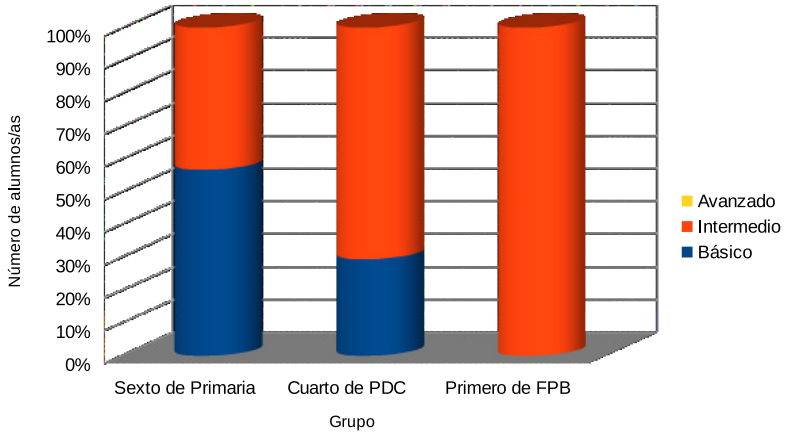



Figura 3.3: Porcentaje acumulado de alumnos y alumnas según nivel de desempeño para la actividad "gánate los puntos"

ciente a esta categoría fue capaz de comprender y de aplicar una de las dos sentencias condicionales.

La segunda conclusión que puede extraerse es que en este caso la edad si que parece ser un factor influyente a la hora de comprender, interpretar y aplicar una sentencia condicional u otro tipo de sentencias de control. La Figura 3.3 muestra claramente como el número de alumnos y alumnas que mostraron un nivel de desempeño intermedio aumenta a medida que también lo hace el nivel académico, y por tanto, la edad media del alumnado.

# Capítulo 4

# Conclusiones y trabajo futuro

En el presente trabajo se ha llevado a cabo una recopilación bibliográfica sobre el estado del arte del pensamiento computacional y los lenguajes de programación visual en el contexto educativo. Además, se ha profundizado en numerosos conceptos relacionados con el pensamiento computacional como pueden ser el pensamiento algorítmico o el aprendizaje basado en problemas.

Dicha revisión de la literatura también ha permitido analizar los lenguajes de programación visual más ampliamente utilizados en diferentes ámbitos de aplicación y atendiendo a diversas taxonomías. Teniendo en cuenta el contexto educativo, se ha realizado un estudio más exhaustivo de algunos de los lenguajes de programación visual más extendidos: Alice, App Inventor, StarLogo y Scratch.

Tras haber llevado a cabo dicho análisis, se ha seleccionado Scratch como herramienta base para diseñar el estudio presentado en este trabajo. Se han repasado las características más importantes que la interfaz gráfica de Scratch proporciona, los materiales que pueden utilizarse para aprender a manejarlo y se han revisado las bases de datos científicas más importantes en busca de trabajos previos en los que se haya estudiado la influencia del uso de un lenguaje de programación como Scratch en el desarrollo de habilidades propias del pensamiento computacional. El bajo número de resultados obtenidos en la búsqueda anterior es un claro indicador de que esta línea de investigación es novedosa, quedando aún mucho trabajo por hacer.

Otro de los objetivos a alcanzar en este trabajo ha sido el diseño de un estudio que permita medir el impacto que tiene un lenguaje de programación visual como Scratch para el desarrollo del pensamiento computacional en estudiantes de primaria y secundaria. Dicho estudio consta de tres fases principales bien diferenciadas en las que se utilizan diferentes instrumentos para el desarrollo y medición del pensamiento computacional. Tanto la primera como la última fase se centran en la obtención de datos cuantitativos, mientras que la segunda, desarrollada en un entorno construccionista de aprendizaje basado en problemas, está más enfocada a la obtención de información cualitativa, para lo que se usan, entre

otros instrumentos de recopilación de información, una plantilla de análisis de problemas por parte del alumnado y la herramienta de observación ICOT por parte del profesorado.

La estancia en un centro educativo durante el periodo de prácticas no fue suficientemente larga como para llevar a cabo el estudio completo. Por ello, sólo se desarrolló la primera fase, consistente en la aplicación de los instrumentos de medición del pensamiento computacional a diferentes grupos de primaria y secundaria. De dicho estudio previo se puede concluir que por lo general el alumnado de primaria y secundaria muestra un alto grado de desarrollo de habilidades relacionadas con la abstracción, la planificación y el paralelismo. Sin embargo, no muestra un nivel de desempeño tan alto en otro tipo de facetas como el manejo de estructuras de control.

La principal línea de trabajo futuro consistirá en la aplicación de las restantes fases del estudio diseñado. De este modo se podrá, por un lado, obtener información cualitativa que complemente a los datos cuantitativos ya recopilados, y por el otro, conocer si realmente el uso de Scratch influye significativamente en la mejora de habilidades propias del pensamiento computacional.

# Apéndice A

# Instrumentos para el desarrollo y medición del pensamiento computacional

## A.1. Plantilla para el desarrollo del pensamiento algorítmico

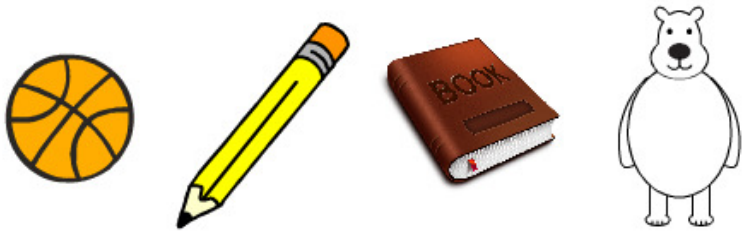
Cabe mencionar que la plantilla presente en esta sección puede descargarse en formato gráfico desde el siguiente enlace.

```
################################################################################
#                            Análisis de Problemas                             #
################################################################################
#                                                                              #
# Formular el problema:                                                        #
#                                                                              #
#                                                                              #
#                                                                              #
################################################################################
#                                                                              #
# Resultados esperados:                                                        #
#                                                                              #
#                                                                              #
#                                                                              #
################################################################################
#                                                                              #
# Datos disponibles:                                                           #
#                                                                              #
#                                                                              #
#                                                                              #
#                                                                              #
################################################################################
```

```
################################################################################
#                                                                              #
# Restricciones:                                                               #
#                                                                              #
#                                                                              #
#                                                                              #
################################################################################
#                                                                              #
# Procesos necesarios (en pseudocódigo):                                       #
# 1.                                                                           #
# 2.                                                                           #
# 3.                                                                           #
# 4.                                                                           #
# 5.                                                                           #
# 6.                                                                           #
# 7.                                                                           #
# 8.                                                                           #
# 9.                                                                           #
# 10.                                                                          #
# 11.                                                                          #
# 12.                                                                          #
# 13.                                                                          #
# 14.                                                                          #
# 15.                                                                          #
# 16.                                                                          #
################################################################################
```

## A.2. Plantilla de la actividad “dibuja y ordena los objetos”

Observa los siguientes cuatro objetos:

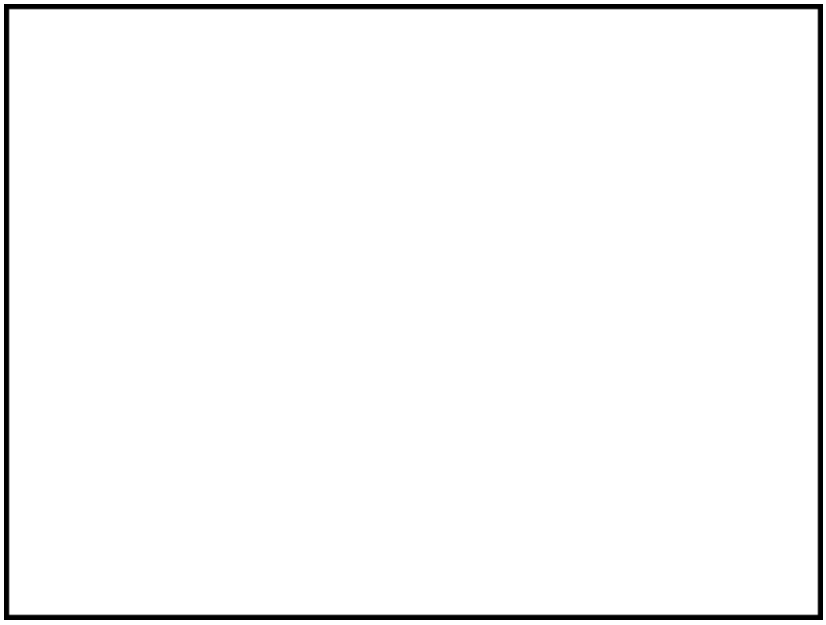



Dibújalos dentro del cuadro, de acuerdo a las siguientes instrucciones:

1. El balón está encima del libro.
2. El libro está al lado del oso.
3. El balón está debajo del lápiz.

Dibújalos de nuevo, pero ahora siguiendo las siguientes instrucciones:

1. El oso está al lado del lápiz.
2. El libro está debajo del balón.
3. El balón está al lado del oso.



## A.3. Plantilla de la actividad "gánate los puntos"

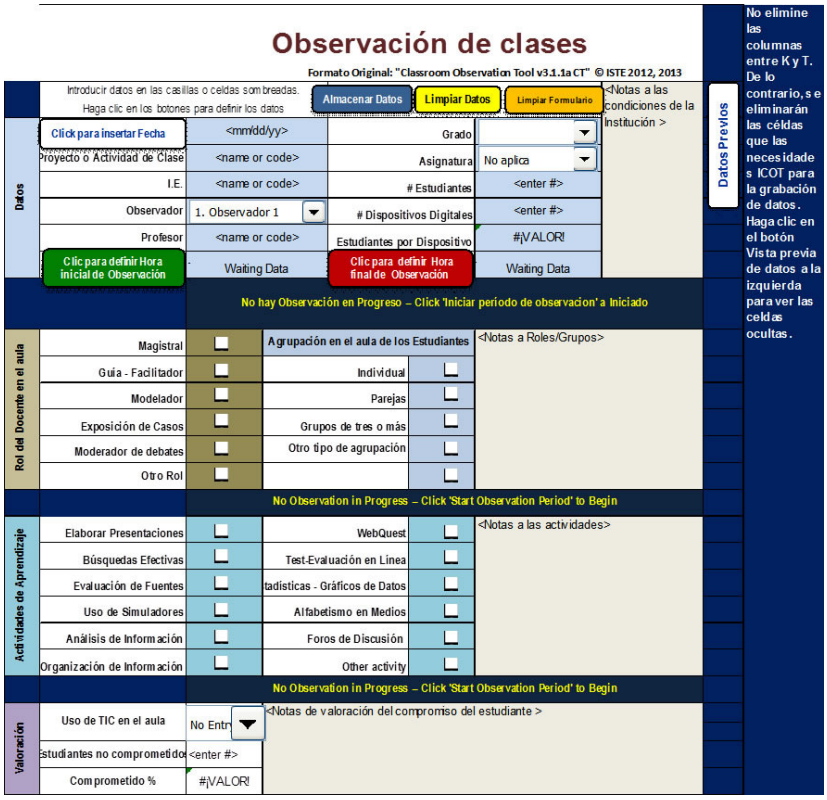
A continuación, se muestra un fragmento del poema "Pastorcita" de Rafael Pombo. Buscando en el texto puedes conseguir puntos de la siguiente manera:

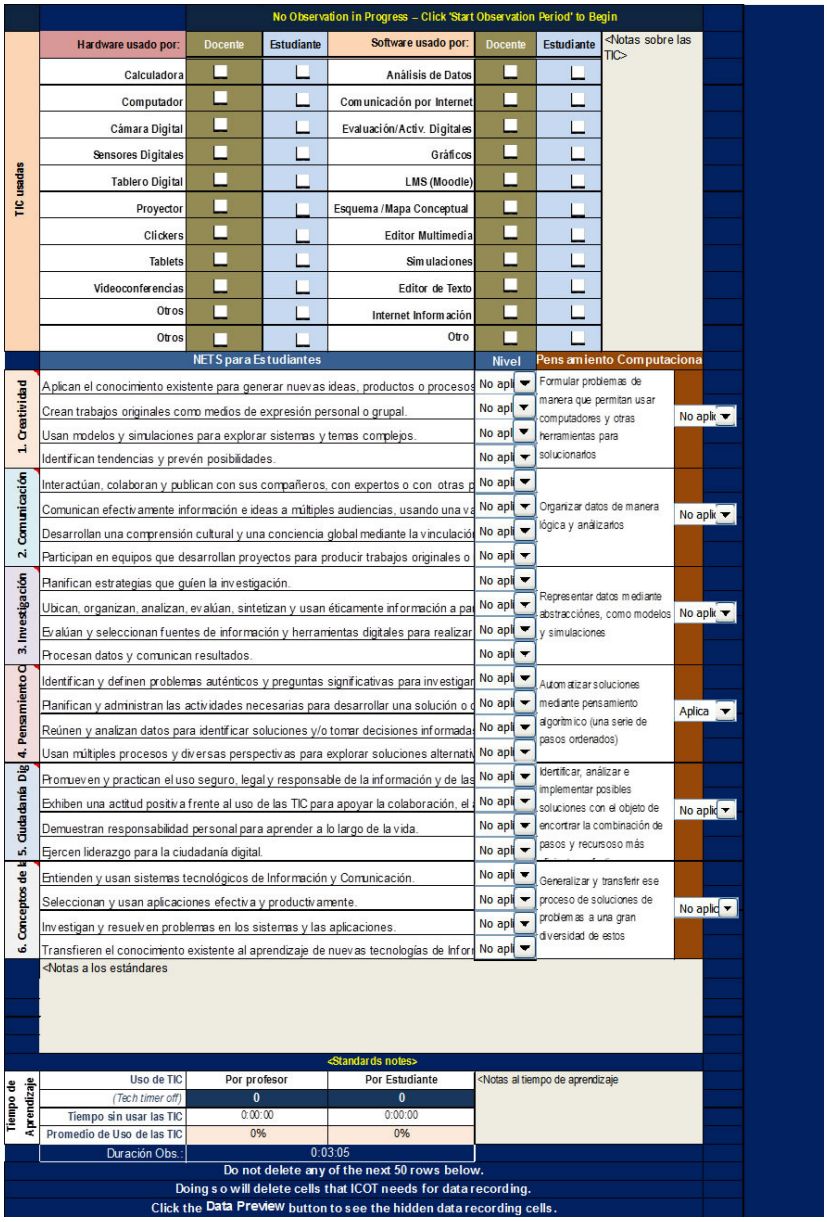
- Por cada palabra que tenga las **tres últimas letras iguales a las de otra palabra**, **consigues 5 puntos**.
- Por cada palabra que comience con **p** y termine con **a**, **consigues 1 punto**.

Para cada verso, subraya las palabras que te hayan permitido conseguir puntos y anota al lado el número de puntos conseguidos, tal y como se muestra en el ejemplo.

```
Pastorcita                                Puntos

Pastorcita perdió sus ovejas              1 + 5_______
¡y quién sabe por dónde andarán!           ____________
No te enfades, que oyeron tus quejas      5____________
y ellas mismas bien pronto vendrán.        ____________

Y no vendrán solas, que traerán sus colas,  ____________
Y ovejas y colas gran fiesta darán.         ____________
Pastorcita se queda dormida,                ____________
y soñando las oye balar.                    ____________

Se despierta y las llama enseguida,         ____________
y engañada se tiende a llorar.              ____________
No llores, pastora, que niña que llora      ____________
bien pronto la oimos reir y cantar.         ____________


                          Puntos totales:   ____________
```

## A.4. Herramienta de observación de clases del ISTE





No Observation in Progress – Click 'Start Observation Period' to Begin

TIC usadas

| Hardware usado por: | Docente | Estudiante | Software usado por: | Docente | Estudiante | <Notas sobre las TIC> |
|---|---|---|---|---|---|---|
| Calculadora | ☐ | ☐ | Análisis de Datos | ☐ | ☐ | |
| Computador | ☐ | ☐ | Comunicación por Internet | ☐ | ☐ | |
| Cámara Digital | ☐ | ☐ | Evaluación/Activ. Digitales | ☐ | ☐ | |
| Sensores Digitales | ☐ | ☐ | Gráficos | ☐ | ☐ | |
| Tablero Digital | ☐ | ☐ | LMS (Moodle) | ☐ | ☐ | |
| Proyector | ☐ | ☐ | Esquema / Mapa Conceptual | ☐ | ☐ | |
| Clickers | ☐ | ☐ | Editor Multimedia | ☐ | ☐ | |
| Tablets | ☐ | ☐ | Simulaciones | ☐ | ☐ | |
| Videoconferencias | ☐ | ☐ | Editor de Texto | ☐ | ☐ | |
| Otros | ☐ | ☐ | Internet Información | ☐ | ☐ | |
| Otros | ☐ | ☐ | Otro | ☐ | ☐ | |

| NETS para Estudiantes | Nivel | Pensamiento Computacional |
|---|---|---|
| 1. Creatividad: Aplican el conocimiento existente para generar nuevas ideas, productos o procesos | No apli | Formular problemas de manera que permitan usar computadores y otras herramientas para solucionarlos: No aplic |
| 1. Creatividad: Crean trabajos originales como medios de expresión personal o grupal. | No apli | |
| 1. Creatividad: Usan modelos y simulaciones para explorar sistemas y temas complejos. | No apli | |
| 1. Creatividad: Identifican tendencias y prevén posibilidades. | No apli | |
| 2. Comunicación: Interactúan, colaboran y publican con sus compañeros, con expertos o con otras p | No apli | Organizar datos de manera lógica y analizarlos: No aplic |
| 2. Comunicación: Comunican efectivamente información e ideas a múltiples audiencias, usando una v | No apli | |
| 2. Comunicación: Desarrollan una comprensión cultural y una conciencia global mediante la vinculació | No apli | |
| 2. Comunicación: Participan en equipos que desarrollan proyectos para producir trabajos originales o | No apli | |
| 3. Investigación: Planifican estrategias que guíen la investigación. | No apli | Representar datos mediante abstracciónes, como modelos y simulaciones: No aplic |
| 3. Investigación: Ubican, organizan, analizan, evalúan, sintetizan y usan éticamente información a pa | No apli | |
| 3. Investigación: Evalúan y seleccionan fuentes de información y herramientas digitales para realizar | No apli | |
| 3. Investigación: Procesan datos y comunican resultados. | No apli | |
| 4. Pensamiento C: Identifican y definen problemas auténticos y preguntas significativas para investigar | No apli | Automatizar soluciones mediante pensamiento algorítmico (una serie de pasos ordenados): Aplica |
| 4. Pensamiento C: Planifican y administran las actividades necesarias para desarrollar una solución o c | No apli | |
| 4. Pensamiento C: Reúnen y analizan datos para identificar soluciones y/o tomar decisiones informada | No apli | |
| 4. Pensamiento C: Usan múltiples procesos y diversas perspectivas para explorar soluciones alternati | No apli | |
| 5. Ciudadanía Dig: Promueven y practican el uso seguro, legal y responsable de la información y de las | No apli | Identificar, análizar e implementar posibles soluciones con el objeto de encontrar la combinación de pasos y recursoso más: No aplic |
| 5. Ciudadanía Dig: Exhiben una actitud positiva frente al uso de las TIC para apoyar la colaboración, el | No apli | |
| 5. Ciudadanía Dig: Demuestran responsabilidad personal para aprender a lo largo de la vida. | No apli | |
| 5. Ciudadanía Dig: Ejercen liderazgo para la ciudadanía digital. | No apli | |
| 6. Conceptos de I: Entienden y usan sistemas tecnológicos de Información y Comunicación. | No apli | Generalizar y transferir ese proceso de soluciones de problemas a una gran diversidad de estos: No aplic |
| 6. Conceptos de I: Seleccionan y usan aplicaciones efectiva y productivamente. | No apli | |
| 6. Conceptos de I: Investigan y resuelven problemas en los sistemas y las aplicaciones. | No apli | |
| 6. Conceptos de I: Transfieren el conocimiento existente al aprendizaje de nuevas tecnologías de Infor | No apli | |

<Notas a los estándares

<Standards notes>

Tiempo de Aprendizaje

| Uso de TIC | Por profesor | Por Estudiante | <Notas al tiempo de aprendizaje |
|---|---|---|---|
| (Tech timer off) | 0 | 0 | |
| Tiempo sin usar las TIC | 0:00:00 | 0:00:00 | |
| Promedio de Uso de las TIC | 0% | 0% | |
| Duración Obs.: | 0:03:05 | | |

Do not delete any of the next 50 rows below.
Doing so will delete cells that ICOT needs for data recording.
Click the Data Preview button to see the hidden data recording cells.

# Bibliografía


[Ajiro and Tsuchida, 2005] Ajiro, T. and Tsuchida, K. (2005). A bit-level concurrent visual programming language (A-BITS) and a base computation model (APC) for its development. In *Visual Languages and Human-Centric Computing, 2005 IEEE Symposium on*, pages 269–271.

[Alice.org, 2015] Alice.org (2015). Sitio Web oficial de Alice. `http://www.alice.org/index.php`.

[Amelunxen et al., 2006] Amelunxen, C., Konigs, A., Rotschke, T., and Schurr, A. (2006). MOSL: Composing a Visual Language for a Metamodeling Framework. In *Visual Languages and Human-Centric Computing, 2006. VL/HCC 2006. IEEE Symposium on*, pages 81–84.

[Appinventor.mit.edu, 2015] Appinventor.mit.edu (2015). Sitio Web oficial de MIT App Inventor. `http://appinventor.mit.edu/explore/`.

[Barr and Stephenson, 2011] Barr, V. and Stephenson, C. (2011). Bringing Computational Thinking to K-12: What is Involved and What is the Role of the Computer Science Education Community? *ACM Inroads*, 2(1):48–54.

[Baytak and Land, 2011] Baytak, A. and Land, S. (2011). An investigation of the artifacts and process of constructing computers games about environmental science in a fifth grade classroom. *Educational Technology Research and Development*, 59(6):765–782.

[Begel and Graham, 2005] Begel, A. and Graham, S. (2005). Spoken programs. In *Visual Languages and Human-Centric Computing, 2005 IEEE Symposium on*, pages 99–106.

[Benzi et al., 1999] Benzi, F., Maio, D., and Rizzi, S. (1999). VISIONARY: a Viewpoint-based Visual Language for Querying Relational Databases. *Journal of Visual Languages & Computing*, 10(2):117 – 145.

[Boechler et al., 2014] Boechler, P., Artym, C., Dejong, E., Carbonaro, M., and Stroulia, E. (2014). Computational thinking, code complexity, and prior experience in a videogame-building assignment. In *Advanced Learning Technologies (ICALT), 2014 IEEE 14th International Conference on*, pages 396–398.

[Bragg and Driskill, 1994] Bragg, S. and Driskill, C. (1994). Diagrammatic-graphical programming languages and DoD-STD-2167A. In *AUTOTESTCON '94. IEEE Systems Readiness Technology Conference. 'Cost Effective Support Into the Next Century', Conference Proceedings.*, pages 211–220.

[Brennan and Resnick, 2012] Brennan, K. and Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. In *Annual American Educational Research Association meeting*, Vancouver, BC, Canada.

[Burnett and Baker, 1994] Burnett, M. M. and Baker, M. J. (1994). A Classification System for Visual Programming Languages. *Journal of Visual Languages & Computing*, 5(3):287 – 300.

[Catrobat.org, 2015] Catrobat.org (2015). Sitio Web oficial de Catrobat. `http://www.catrobat.org/`.

[Chang, 2014] Chang, C.-K. (2014). Effects of using Alice and Scratch in an introductory programming course for corrective instruction. *Journal of Educational Computing Research*, 51(2):185–204.

[Cormen et al., 2009] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009). *Introduction to Algorithms, Third Edition*. The MIT Press.

[CSTA, 2011] CSTA (2011). Operational Definition of Computational Thinking for K–12 Education.

[Ehrlich and Johnson-Laird, 1982] Ehrlich, K. and Johnson-Laird, P. (1982). Spatial descriptions and referential continuity. *Journal of Verbal Learning and Verbal Behavior*, 21(3):296 – 306.

[Erwig, 1994] Erwig, M. (1994). DEAL-a language for depicting algorithms. In *Visual Languages, 1994. Proceedings., IEEE Symposium on*, pages 184–185.

[Flores, 2011] Flores, A. (2011). Desarrollo del Pensamiento Computacional en la Formación en Matemática Discreta. *Lámpsakos*, (5):28–33.

[Futschek, 2006] Futschek, G. (2006). Algorithmic thinking: The key for understanding computer science. In Mittermeir, R., editor, *Informatics Education – The Bridge between Using and Understanding Computers*, volume 4226 of *Lecture Notes in Computer Science*, pages 159–168. Springer Berlin Heidelberg.

[Grover and Pea, 2013] Grover, S. and Pea, R. (2013). Computational Thinking in K–12: A Review of the State of the Field. *Educational Researcher*, 42(1):38–43.

[Grover et al., 2015] Grover, S., Pea, R., and Cooper, S. (2015). Designing for deeper learning in a blended computer science course for middle school students. *Computer Science Education*, 25(2):199–237.

[Hague and Payton, 2011] Hague, C. and Payton, S. (2011). Digital literacy across the curriculum. *Curriculum Leadership*, 9(10).

[Hmelo-Silver, 2004] Hmelo-Silver, C. (2004). Problem-based learning: What and how do students learn? *Educational Psychology Review*, 16(3):235–266.

[Howland et al., 2006] Howland, K., Good, J., and Robertson, J. (2006). Script Cards: A Visual Programming Language for Games Authoring by Young People. In *Visual Languages and Human-Centric Computing, 2006. VL/HCC 2006. IEEE Symposium on*, pages 181–186.

[Hundhausen et al., 2004] Hundhausen, C., Wingstrom, J., and Vatrapu, R. (2004). The Evolving User-Centered Design of the Algorithm Visualization Storyboarder. In *Visual Languages and Human Centric Computing, 2004 IEEE Symposium on*, pages 62–64.

[Jonassen et al., 2008] Jonassen, D., Howland, J., Marra, R., and Crismond, D. (2008). *Meaningful learning with technology (3rd ed.)*. Pearson/Merrill Prentice Hall.

[Kafai and Burke, 2013] Kafai, Y. B. and Burke, Q. (2013). Computer Programming Goes Back to School. *Phi Delta Kappan*, 95(1):61–65.

[Kafai et al., 2011] Kafai, Y. B., Fields, D. A., and Burke, W. Q. (2011). Entering the Clubhouse: Case Studies of Young Programmers Joining the Online Scratch Communities. *Journal of Organizational and End User Computing (JOEUC)*, 22(2):21–35.

[Kelleher and Pausch, 2005] Kelleher, C. and Pausch, R. (2005). Lowering the Barriers to Programming: A Taxonomy of Programming Environments and Languages for Novice Programmers. *ACM Comput. Surv.*, 37(2):83–137.

[Lin and Liu, 2012] Lin, J. and Liu, S.-F. (2012). An investigation into parent-child collaboration in learning computer programming. *Educational Technology and Society*, 15(1):162–173. cited By 2.

[Logo.media.mit.edu, 2015] Logo.media.mit.edu (2015). The Logo Foundation. `http://el.media.mit.edu/logo-foundation/index.html`.

[Lookingglass.wustl.edu, 2015] Lookingglass.wustl.edu (2015). Sitio Web oficial de Looking Glass. `http://lookingglass.wustl.edu/`.

[Lye and Koh, 2014] Lye, S. Y. and 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(0):51 – 61.

[López, 2014] López, J. C. (2014). Impacto de Scratch en el desarrollo del pensamiento algorítmico. Master's thesis, Universidad Icesi.

[Mills, 2010] Mills, K. A. (2010). A Review of the "Digital Turn" in the New Literacy Studies. *Review of Educational Research*, 80(2):246–271.

[Morales and Landa, 2004] Morales, P. and Landa, V. (2004). Aprendizaje Basado en Problemas Problem-based Learning. *Theoria*, 13:145–157.

[Morra, 2001] Morra, S. (2001). On the Information-Processing Demands of Spatial Reasoning. *Thinking and Reasoning*, 7(4):347–365.

[National Research Council, 1999] National Research Council (1999). *Being Fluent with Information Technology*. The National Academies Press, Washington, D.C., USA.

[National Research Council, 2012] National Research Council (2012). *A framework for K-12 science education: Practices, crosscutting concepts, and core ideas*. The National Academies Press, Washington, D.C., USA.

[Ng, 2012] Ng, W. (2012). Can we teach digital natives digital literacy? *Computers & Education*, 59(3):1065 – 1078.

[Pinet and Lbath, 2000] Pinet, F. and Lbath, A. (2000). A Visual Modelling Language for Distributed Geographic Information Systems. In *Proceedings of the 2000 IEEE International Symposium on Visual Languages (VL'00)*, VL '00, pages 75–, Washington, DC, USA. IEEE Computer Society.

[Polya, 1957] Polya, G. (1957). *How to solve it*. Princeton University Press.

[Ratcliff and Anderson, 2011] Ratcliff, C. C. and Anderson, S. E. (2011). Reviving the Turtle: Exploring the Use of Logo with Students with Mild Disabilities. *Computers in the Schools*, 28(3):241–255.

[Resnick et al., 2009] Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., and Kafai, Y. (2009). Scratch: Programming for All. *Commun. ACM*, 52(11):60–67.

[Roscoe et al., 2014] Roscoe, J., Fearn, S., and Posey, E. (2014). Teaching computational thinking by playing games and building robots. In *Interactive Technologies and Games (iTAG), 2014 International Conference on*, pages 9–12.

[Ruthmann et al., 2010] Ruthmann, A., Heines, J. M., Greher, G. R., Laidler, P., and Saulters, II, C. (2010). Teaching computational thinking through musical live coding in scratch. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, SIGCSE '10, pages 351–355, New York, NY, USA. ACM.

[Slnova.org, 2015] Slnova.org (2015). Sitio Web oficial de StarLogo Nova. `http://www.slnova.org/`.

[Smith et al., 2000] Smith, D. C., Cypher, A., and Tesler, L. (2000). Programming by Example: Novice Programming Comes of Age. *Commun. ACM*, 43(3):75–81.

[Snap.berkeley.edu, 2015] Snap.berkeley.edu (2015). Sitio Web oficial de Snap. `https://snap.berkeley.edu/`.

[Starlogo.mit.edu, 2015] Starlogo.mit.edu (2015). Sitio Web oficial de StarLogo. `http://education.mit.edu/starlogo/`.

[Starlogo.tng.mit.edu, 2015] Starlogo.tng.mit.edu (2015). Sitio Web oficial de StarLogo TNG. `http://education.mit.edu/projects/starlogo-tng`.

[Tangney et al., 2010] Tangney, B., Oldham, E., Conneely, C., Barrett, S., and Lawlor, J. (2010). Pedagogy and Processes for a Computer Programming Outreach Workshop: The Bridge to College Model. *Education, IEEE Transactions on*, 53(1):53–60.

[Theodorou and Kordaki, 2010] Theodorou, C. and Kordaki, M. (2010). Super Mario: a collaborative game for the learning of variables in programming. *International Journal of Academic Research*, 2(4):111–118.

[Triantafyllou and Timcenk, 2013] Triantafyllou, E. and Timcenk, O. (2013). Applying Constructionism and Problem Based Learning for Developing Dynamic Educational Material for Mathematics At Undergraduate University Level. In *Proceedings of the 4th International Research Symposium on Problem-Based Learning*, Kuala Lumpur.

[User Interface Group, 1995] User Interface Group, U. (1995). Alice: Rapid prototyping for virtual reality. *IEEE Comput. Graph. Appl.*, 15(3):8–11.

[Vygotsky, 1989] Vygotsky, L. S. (1989). *El desarrollo de los procesos psicológicos superiores*. Crítica.

[Webb and Rosson, 2013] Webb, H. and Rosson, M. B. (2013). Using scaffolded examples to teach computational thinking concepts. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, SIGCSE '13, pages 95–100, New York, NY, USA. ACM.

[Wing, 2006] Wing, J. (2006). Computational Thinking. *Communications of the ACM, CACM*, 3(49):33–35.