Universidad de La Laguna

Escuela Superior de Ingeniería y Tecnología
Sección de Ingeniería Informática

Trabajo de Fin de Grado

Pensamiento computacional. Herramientas y aplicaciones.

"*Computational thinking. Tools and applications*"

Eliana Abdel Majid Hassan

La Laguna, 4 de julio de 2016

D^a. **Coromoto León Hernández**, con N.I.F. 78.602.216-W profesora Titular de Universidad adscrita al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutora.

D. **Eduardo Segredo González**, con N.I.F. 78.564.242-Z investigador de la Edinburgh Napier University, como cotutor.

C E R T I F I C A (N)

Que la presente memoria titulada:

"*Pensamiento computacional. Herramientas y aplicaciones.*"

ha sido realizada bajo su dirección por *D^a*. **Eliana Abdel Majid Hassan**, con N.I.F. 45.865.478-M.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 4 de julio de 2016

Agradecimientos

A mi tutora Coromoto y a mi cotutor Eduardo, por apoyarme y ayudarme en este largo proceso. Por enseñarme el verdadero concepto de pensamiento computacional.

A mis padres Ra'ed e Imán por darme la oportunidad de aprender y confiar en mis conocimientos.

A mis hermanas Ilia y Rania por ser las mejores hermanas del mundo.

A mis abuelos, tíos y primos por todo el apoyo recibido siempre y por no dejarme caer nunca.

A mi tía Laila por haberme empujado hacia el mundo de la Informática.

A mi amiga Bea por su apoyo incondicional todos estos años.

A mis amigos por acompañarme todo este tiempo.

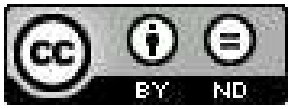
A mis compañeros de la facultad, por todos los momentos juntos.

A la computación por enseñarnos a resolver problemas.

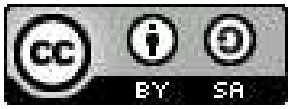
"Scientia potentia est"

Licencia

*Si NO quiere permitir que se compartan las adaptaciones de tu obra y quieres permitir usos comerciales de tu obra indica:



* Si quiere permitir que se compartan las adaptaciones de tu obra mientras se comparta de la misma manera y quieres permitir usos comerciales de tu obra (licencia de Cultura Libre) indica:



* Si quiere permitir que se compartan las adaptaciones de tu obra y quieres permitir usos comerciales de tu obra (licencia de Cultura Libre) indica:

Resumen

El objetivo de este proyecto ha sido estudiar el diseño y la arquitectura software de las herramientas existentes para el desarrollo del pensamiento computacional. Además, se han desarrollado casos de estudio de algoritmos evolutivos con varias herramientas de programación visual.

El pensamiento computacional concibe la idea entender la mente humana como un sistema de procesamiento de la información muy similar o incluso idéntico al de una computador. Se ha pretendido estudiar las herramientas que conforman este hecho con el objetivo de analizar sin son aptas para promover este paradigma de la computación.

Palabras clave: *pensamiento computacional, computación, algoritmo evolutivo, scratch, programación visual.*

Abstract

The purpose of this project has been the software design and architecture of the existing tools for the development of computational thinking. Furthermore, study cases of evolutionary algorithms have been developed with different tools of visual programming.

Computational thinking comes up with the idea of understanding the human mind as a processing system of information very similar, or even identical, to the one within a computer.

The aim is to study the tools that define this fact with the objective of analyzing if they are suitable to promote this paradigm of computation.

Keywords: computational thinking, computer studies, evolutionary algorithm, scratch, visual programing.

Índice general

[1] Herramienta perteneciente al MIT

[2] Manera de comprobar el correcto funcionamiento de un módulo de código

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

Según los currículos de la educación actual, los ciclos de primaria no optan por clases de informática, sólo se ha incluido como actividad extraescolar. Sin embargo, a nivel de secundaria, la asignatura de Informática se incluye tanto en la rama de tecnología como en asignaturas optativas. Esto se debe a que hoy en día, la mayoría de las instituciones están orientadas a la «Alfabetización digital» ya que toda labor realizada tal como organizar, entender, analizar y evaluar la información se lleva a cabo utilizando la tecnología digital.

No obstante, después de analizar este hecho vamos más allá del término «Informática» y llegamos al término "Computación". Numerosos países han uncluido ya la asignatura "Computación"en el sentido de «Resolver problemas» con un ordenador realizando el denominado "Pensamiento computacional". Por lo tanto, nos encontramos con un término bastante extenso en el que predomina la resolución de problemas que se encuentran en el día a día.

La capacidad de esta resolución de problemas se va incrementando a lo largo del tiempo. Comenzamos a introducirnos en las nuevas tecnologías hasta llegar a tener pensamientos mecánicos de resolución y además, a afrontar estos problemas con una serie de pasos ordenados con los que organizamos los hechos hasta llegar a una posible o posibles soluciones.

1.1. Antecedentes y estado actual del tema

En el Anexo II del RD 1393/2007 se establecen las materias básicas por rama de conocimiento para las enseñanzas universitarias oficiales. Informática sólo aparece en la rama de Ingeniería y Arquitectura. Se infiere de esta carencia, que los estudiantes que llegan a la universidad ya tienen una alfabetización digital.

Sin embargo, la velocidad con la que se ha producido el desarrollo tecnológico y el aumento del protagonismo de las nuevas tecnologías de la información y la comunicación permite presagiar que en poco tiempo las actuales tecnologías informáticas quedarán obsoletas. Por ello es necesario que la formación que reciban los alumnos de bachillerato en la materia Informática no se limite tan solo al conocimiento intrínseco del uso de las tecnologías actuales y a sus utilidades prácticas inmediatas, sino que incida en toda una serie de destrezas que les permitan adecuarse a las que irán surgiendo, esto es, se hace necesario que desarrolle el pensamiento computacional.

Existen en el mercado varias herramientas como Alice, AppInventor, Scratch, etc. que promueven la resolución de problemas mediante la programación visual[1] basada en bloques. Numerosas organizaciones se dedican a la promoción de estas herramientas organizando eventos como 'hour of code (code.org)', 'week of code (codeweek.eu)', etc.

En el presente proyecto se pretende realizar un análisis detallado de estas y otras herramientas, con la finalidad de conocer si son aptas para el desarrollo de habilidades relacionadas con el pensamiento computacional.

1.2. Objetivos

Acorde con los antecedentes de este proyecto, los objetivos de éste constituyen los siguientes puntos:

- Realizar un análisis detallado de herramientas, como Scratch, AppInventor, Alice.., con la finalidad de conocer si son aptas para el desarrollo de habilidades relacionadas con el pensamiento computacional.
- Estudiar el diseño y la arquitectura software de Scratch, así como el análisis de sus funcionalidades.
- Desarrollar casos de estudio con algoritmos evolutivos aplicados a estas herramientas que promueven el pensamiento computacional y realizar una comparativa entre ellos.

[1] Lenguaje de programación que permite a los usuarios la realización de programas con elementos gráficos en vez de con elementos textuales

1.3. Motivación

Respecto al punto anterior, este trabajo surge a raíz de que solamente a nivel de secundaria se incluyen asignaturas de Informática, tanto en ramas tecnológicas como en asignaturas optativas en últimos cursos. Nos encontramos ante lo denominado "Alfabetización digital", ya que toda labor realizada la llevamos a cabo utilizando la tecnología digital.

Vamos más allá del término Informática y llegamos al término "Computación" donde interpretamos este concepto como la capacidad que tenemos para resolver problemas de cualquier índole, esto es, tener un problema y llevar a cabo una serie de pasos para llegar a una solución. Finalmente, de este concepto deriva el término "Pensamiento computacional". El desarrollo del pensamiento computacional cada día se hace más extenso a medida que aumentan el número de tecnologías.

1.4. ¿Qué es el pensamiento computacional?

Según Henderson et al. en su trabajo titulado "Computational Thinking" [11] el pensamiento computacional es una metáfora universal de razonamiento utilizado tanto por hombres y máquinas. Es el núcleo de todas las disciplinas modernas: Ciencia, Tecnología, Ingeniería y Matemáticas (Science, Technology, Engineering and Mathematics, STEM) y es intrínseco a todas las otras disciplinas de la A a la Z.

Por lo tanto, entendiendo el estado actual del tema, llegamos a entender el término computación como la capacidad para resolver problemas. El pensamiento computacional va más allá y concibe la idea entender la mente humana como un sistema de procesamiento de la información muy similar o incluso idéntico al de una computador.

Actualmente, predominan en el mercado varias herramientas como Alice, AppInventor, Scratch, Logo, Greenfoot, etc., que promueven la resolución de problemas mediante la programación visual basada en bloques.

1.5. Herramientas

A continuación se describirán algunas herramientas que promueven el pensamiento computacional, definiendo sus propios lenguajes de programación y sus propias funcionalidades:

1.5.1. Scratch[2]

Según Maloney et al [12] en su trabajo "The Scratch Programming Language and Environment"el término Scratch hace referencia a un entorno de programación visual que permite a usuarios (tanto usuarios pequeños, como docentes y padres) aprender programación a través de la realización de proyectos de cualquier índole, así como creación de juegos e historias animadas.

Destacamos el término de programación visual ya que se lleva a cabo el desarrollo de las diversas aplicaciones en un entorno visual amigable y con facilidad de utilización para el usuario; en este caso, programación orientada a bloques en los que cada bloque recoge una funcionalidad. La última versión de esta herramienta está escrita en el lenguaje ActionScript, propiedad de Adobe Flash, y hace uso de Adobe Flex.

Figura 1.1: Logo general de Scrath del MIT

[2]Herramienta perteneciente al MIT

En las siguientes imágenes, propias de la herramienta Scratch, pondremos un ejemplo con sus distintas funcionalidades.

- En la figura 1.2 se muestra el escenario o backdrop[3] de Scratch, en el que se encuentran los Sprites[4] a los que se le añadirán las funcionalidades por bloques.

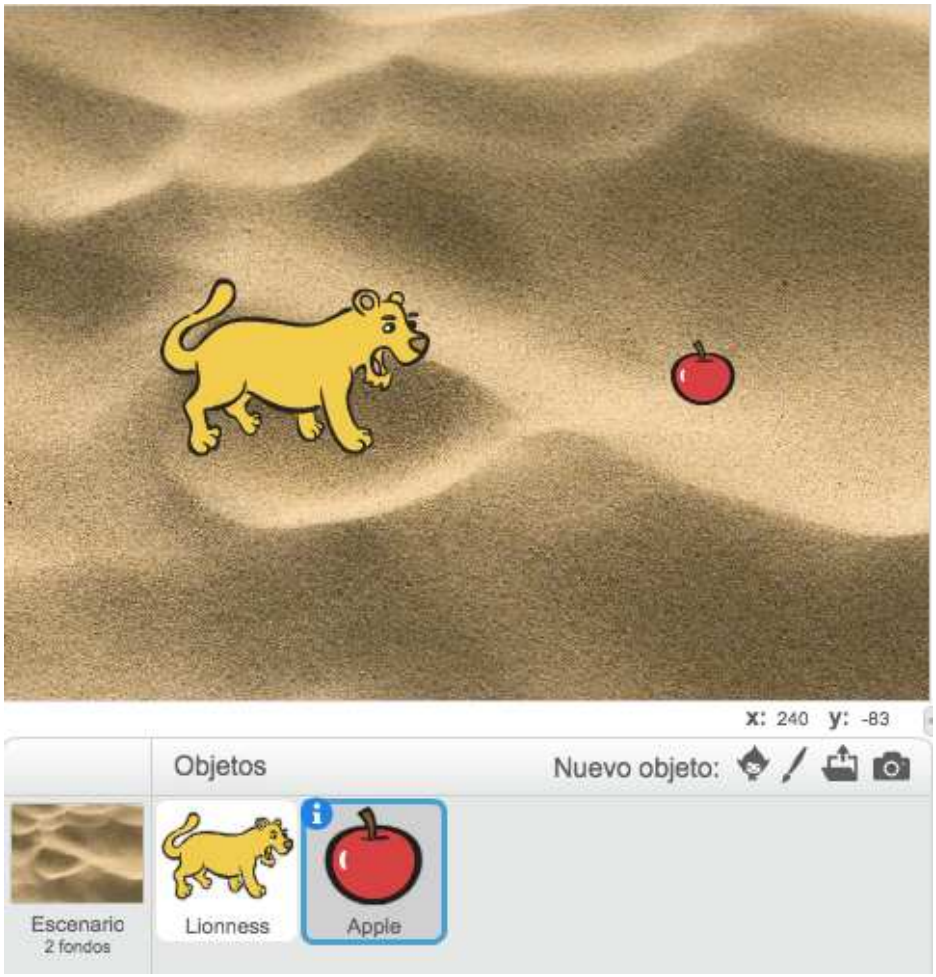


Figura 1.2: Escenario de Scratch

[3]Escenario en el que se realizan las animaciones mediante los bloques de Scratch

[4]El término Sprite fue popularizado por Jay Miner y se trata de un tipo de mapa de bits dibujados en la pantalla de ordenador por hardware gráfico especializado.

- En la figura 1.3 se muestran los distintos bloques posibles para darle funcionalidad a los sprites del escenario.

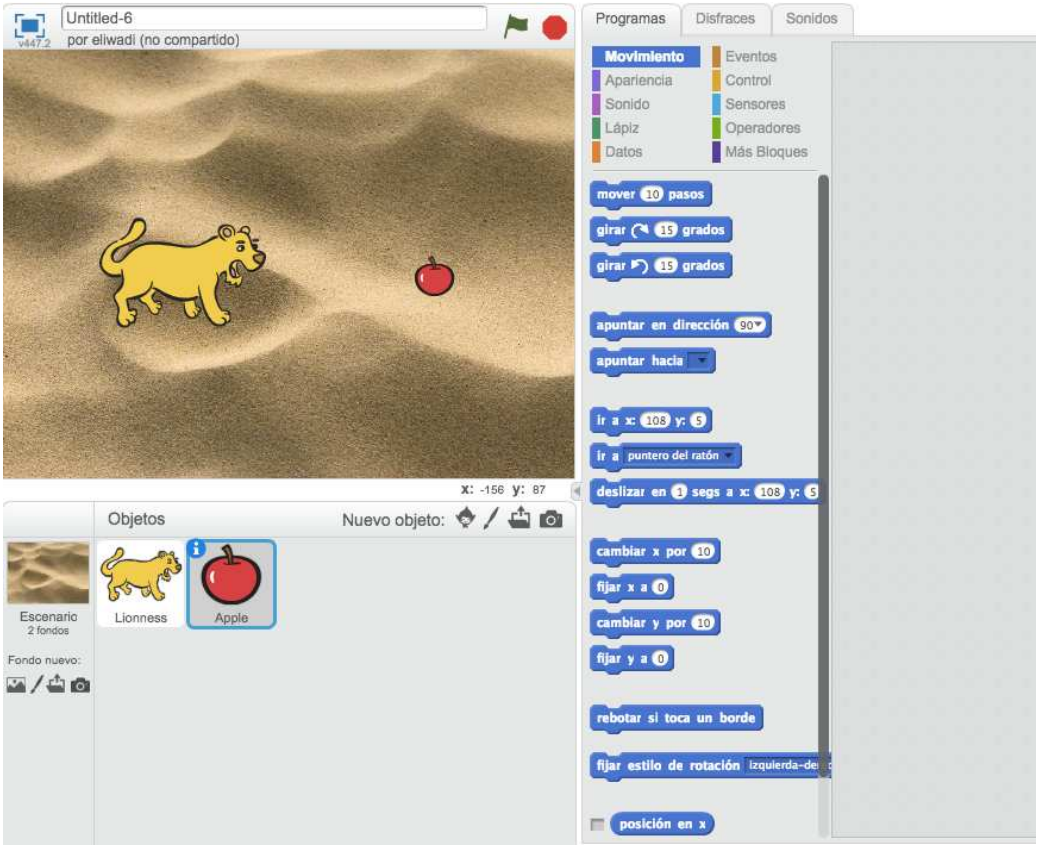


Figura 1.3: Funcionalidades divididas en bloques

1.5.2. AppInventor

AppInventor[13], creada por Google Labs, es una plataforma gratuita para la creación de aplicaciones software orientada a la programación visual utilizada en sistemas operativos Android. Además, se utilizan bloques para el desarrollo de las funcionalidades a partir de un conjunto de herramientas básicas. Una vez que se cree la aplicación con los bloques correspondientes y sus funcionalidades se lanzará el emulador de Android para observar el funcionamiento de la aplicación. Este emulador se llama aiStarter, y es un servicio en segundo plano que creará la simulación del dispositivo Android. Esta herramienta está desarrollada en Java y es utilizada principalmente para el desarrollo de aplicaciones móviles.

Figura 1.4: Logo general de AppInventor del MIT

A continuación se verán unas imágenes propias de la herramienta AppInventor, en las que pondremos un ejemplo con sus distintas funcionalidades.

- En la figura 1.5 se muestra el escenario en AppInvetor en el que creamos un ejemplo de dos sprites en un canvas (lienzo en el que se llevan a cabo todas las funcionalidades a través de bloques).

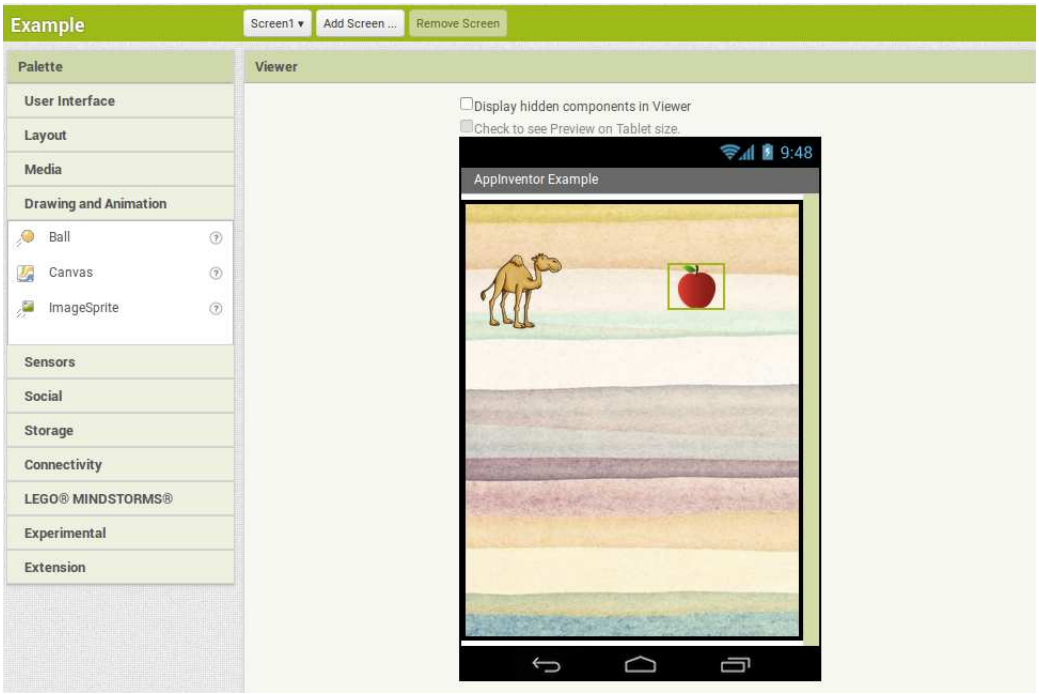


Figura 1.5: Escenario en AppInventor

- En la figura 1.6 se muestran los componentes de la aplicación y sus características modificables.

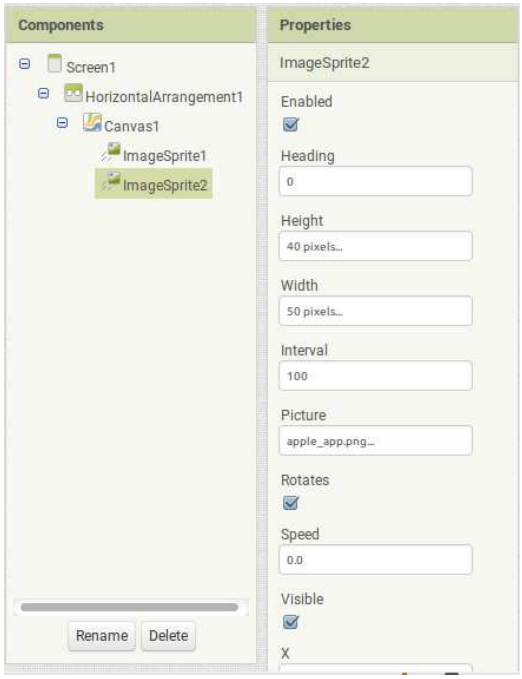


Figura 1.6: Componentes con sus características en AppInventor.

- El panel de bloques que añadirán funcionalidades a los sprites del ejemplo se muestran en la figura 1.7

Figura 1.7: Panel de bloques en AppInventor.

1.5.3. Alice

Alice [4], otra herramienta que promueve el pensamiento computacional, consta de un entorno educativo abierto y libre orientado a objetos. Está programado en Java y utiliza un entorno sencillo basado en *arrastrar y soltar* al igual que las otras herramientas de programación visual. Además, con esta herramienta se pueden desarrollar animaciones mediante modelos 3D.

Figura 1.8: Logo general de Alice

A continuación se verán unas imágenes propias de la herramienta Alice, en las que pondremos un ejemplo con sus distintas funcionalidades.

- En la figura 1.9 observamos el escenario en el que se encuentran los dos objetos seleccionados para añadir funcionalidades. Tanto el escenario como los objetos se podrán seleccionar del álbum propio de la herramienta. Estos álbumes se verían en la figura 1.11

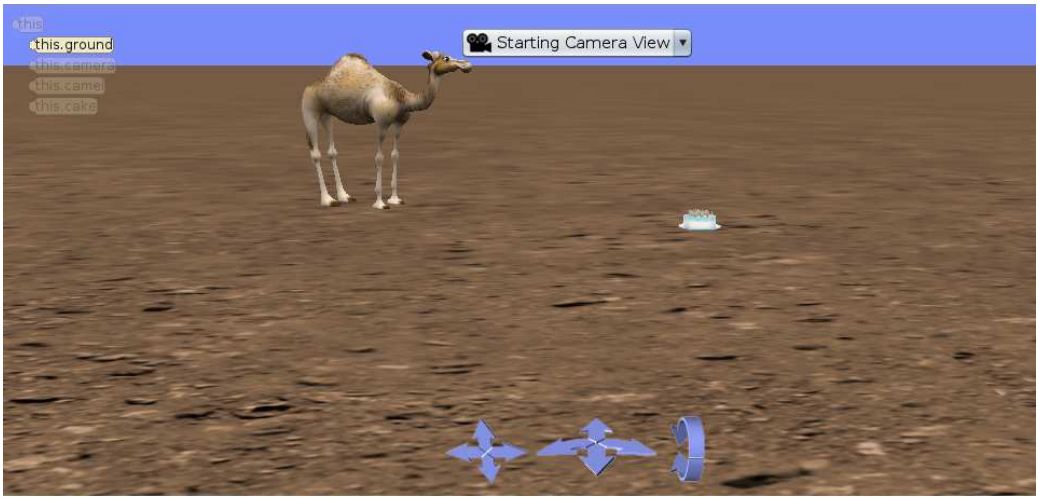


Figura 1.9: Escenario en Alice

- En la figura 1.10 se establecen las posibles funcionalidades asociadas a los objetos o Sprites en el escenario seleccionado.

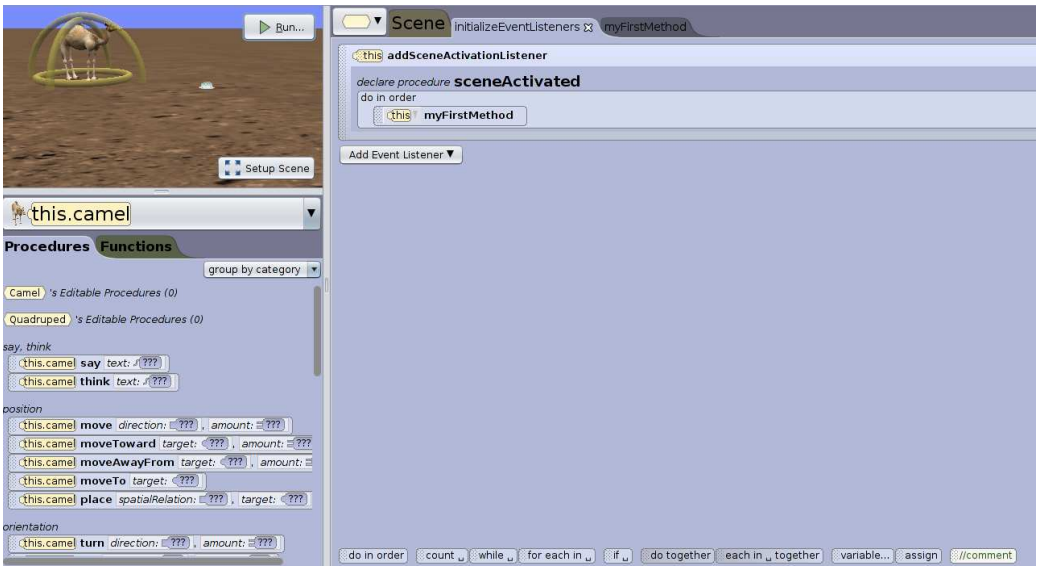


Figura 1.10: Posibles funcionalidades asociadas a los objetos (Sprites) en el escenario

- La lista de Sprites con posibilidad de selección para la aplicación se muestran en la figura 1.11

Figura 1.11: Posibles sprites para el escenario

- Información de los objetos que se encuentran en el escenario.

Figura 1.12: Información de los objetos (Sprites) del escenario

1.5.4. Greenfoot

Greenfoot [7] es una herramienta interactiva con propósitos educativos cuyo objetivo es la creación de aplicaciones gráficas en dos dimensiones. Al igual que Alice está desarrollada en Java y es compatible con cualquier sistema operativo. En general, cualquiera compatible con JVM (Java Virtual Machine). La diferencia entre este entorno y Alice es el número de dimensiones. La interfaz gráfica de Greenfoot consta de dos partes esenciales, el **World Classes** y el **Actor Classes**. En primer lugar, el **World Classes** constituye los fondos del escenario ante el que nos encontramos (entiéndase escenario como entorno en el que desarrollamos la aplicación) y en segundo lugar, el **Actor Classes** constituye los objetos del escenario y a los que le daremos funcionalidad para crear la aplicación deseada.

Figura 1.13: Logo general de Greenfoot

A continuación se verán unas imágenes propias de la herramienta Greenfoot, en las que pondremos un ejemplo con sus distintas funcionalidades.

- En el primer paso hemos añadido una imagen de arena que constituirá el escenario (o backdrop) en el que nos encontramos. El escenario se asocia con el campo World, y a su vez se pueden añadir más escenarios que constituirán clases de world. Estas clases se ven definidas en la parte derecha de la imagen.

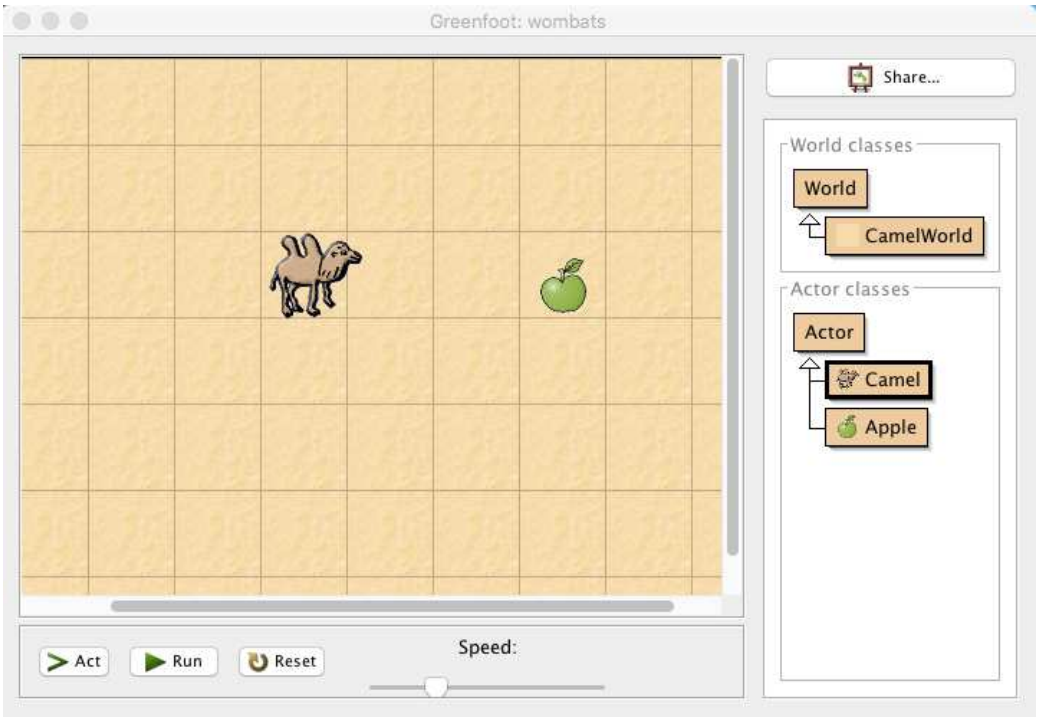


Figura 1.14: Ejemplo del escenario en Greenfoot

- En segundo lugar, observamos un ejemplo de creación de objetos. Según la clase que tengamos de actores, en nuestro caso Camel y Apple podemos crear un objeto de estas clases, creando varios Camel y varios Apple.

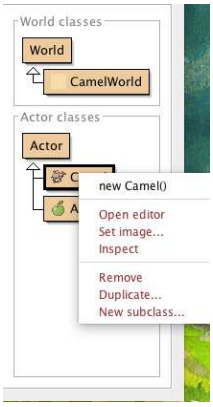


Figura 1.15: Ejemplo de la creación de objetos en Greenfoot

- Por último, al seleccionar click derecho en el objeto creado podremos asignarle las funcionalidades que veamos convenientes para el desarrollo de la aplicación. Entre ellas, posicionamiento de localizaciones, movimientos, imágenes etc.

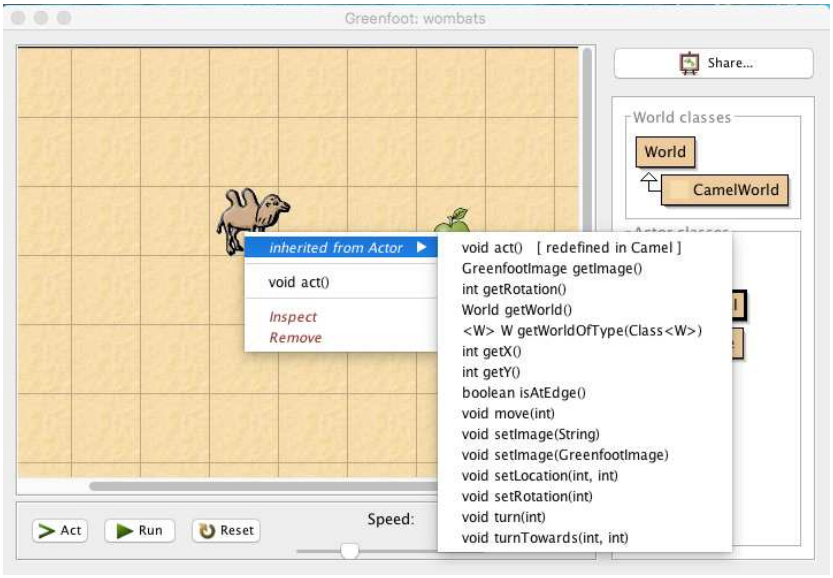


Figura 1.16: Ejemplo de funciones en Greenfoot

1.5.5. Logo

Logo [8], lenguaje de programación de alto nivel, fue diseñado con fines didácticos por Danny Bobrow, Wally Feurzeig y Seymour Papert y constituye un entorno basado en una parte funcional y en una parte estructurada, además amigable y de fácil uso. Por esta razón es el lenguaje de programación preferido para trabajar con niños y jóvenes. Fue creado con la finalidad de utilizarlo en la enseñanza de la programación, ya que proporciona soporte para manejo de listas, archivos y E/S.

Para el desarrollo de este lenguaje se han basado en las características del lenguaje Lisp y consta de varias versiones. Una de las características destacables de este entorno es la posibilidad de producir "gráficos tortuga", es decir, dar instrucciones virtuales a una tortiga (a través de bloques), etc.

Posteriormente, han surgido varias últimas versiones de Logo. Entre ellas la herramienta **Scratch**.

1.6. Revisión bibliográfica

A continuación haremos una revisión bibliográfica de la herramienta elegida para introducir el concepto de computación y el pensamiento computacional. Esta herramienta a la que nos referimos es Scratch.

Armoni et al. [9] en su trabajo titulado "From Scratch to 'Real' programming" hacen un estudio de cómo pasar de los conceptos aprendidos con bloques a realizar una programación en código real.

Sin embargo, antes de llevar a cabo programación en código real debemos estudiar la programación orientada a bloques. Para ello, se hace un estudio bibliográfico de artículos relacionados con herramientas de programación visual.

En el artículo titulado "The Scratch Programming Language and Environment" Maloney et al. [12] hacen un estudio del entorno de esta plataforma, así como su interfaz de usuario, el uso de los tipos de datos, realización de datos concretos, la interacción del usuario con el programa, la ejecución del programa sin errores visibles, etc. Destacamos la utilización de los Sprites como

objetos. El término Sprite fue popularizado por Jay Miner y se trata de un tipo de mapa de bits dibujados en la pantalla de ordenador por hardware gráfico especializado. Sin embargo, con el paso del tiempo esta definición de 'Sprite' terminó siendo cualquier pequeño mapa de bits que se dibuje en la pantalla, incluso si tiene que moverlo todo el procesador central y no cuenta con hardware especializado.

Además de estudiar la arquitectura y funcionalidad de la herramienta Scratch, en el artículo "Scratch: Programming for All"escrito por Mitchel Resnick et al. [15] explican que cualquier problema en base de programación se puede resolver utilizando Scratch. Asimismo, cualquier introducción que se quiera hacer a la programación se puede hacer con Scratch, al destacar de forma positiva por su interfaz usable y accesible además de intuitiva.

Capítulo 2

Herramienta Scratch

En este capítulo se hablará de Scratch, sus características principales, sus distintas versiones, aplicaciones de las mismas y de su arquitectura en la que explicaremos las herramientas utilizadas.

2.1. Definición

Scratch es un entorno de programación visual desarrollada por el MIT que permite a distintos usuarios (tanto pequeños, como docentes y padres) aprender programación mientras trabajan en proyectos de creación de juegos e historias animadas. Destacamos el término de programación visual ya que llevamos a cabo el desarrollo de las diversas aplicaciones con un entorno visual amigable y fácil de utilizar para el usuario, en este caso programación orientada a bloques, en los que cada bloque recoge una funcionalidad.

Según Maloney [12], el entorno de Scratch fue diseñado para promover la programación a través de respuesta inmediata de los scripts realizados a través de los bloques. Estos bloques se ejecutan realizando los pasos para poner en movimiento a los Sprites que vayamos a utilizar en el programa.

En líneas generales, Scratch es la herramienta más destacada hoy en día cuando hablamos de pensamiento computacional. No solamente por su utilización en centros y eventos relacionados con computación alrededor del mundo, sino por su facilidad de uso y su interfaz intuitiva a la hora de buscar solución a los problemas dados. Por ello, Resnick et al. [15] destacan Scratch como la mejor herramienta de programación visual, ya que no requiere conocimientos previos

de programación, a diferencia con otras herramientas de programación visual.

En cuanto al entorno de esta plataforma destacamos varios puntos importantes sobre su interfaz de usuario y su funcionamiento. Estos puntos se aplican para cualquier versión de Scratch.

- **Ventana única en la interfaz gráfica.**

  Scratch contiene una única ventana en la que se encuentran todas las funcionalidades propias para la realización de programas y/o aplicaciones. Es una entorno multi-panel ya que es una única ventana con varios paneles con el objetivo de que todas las funcionalidades sean visibles al usuario. Además, cada uno de los comandos que contienen bloques están diferenciados por colores ayudando a los usuarios a adaptarse a los bloques.

  En la figura 2.1 se ve la interfaz gráfica de Scratch con sus correspondientes paneles:

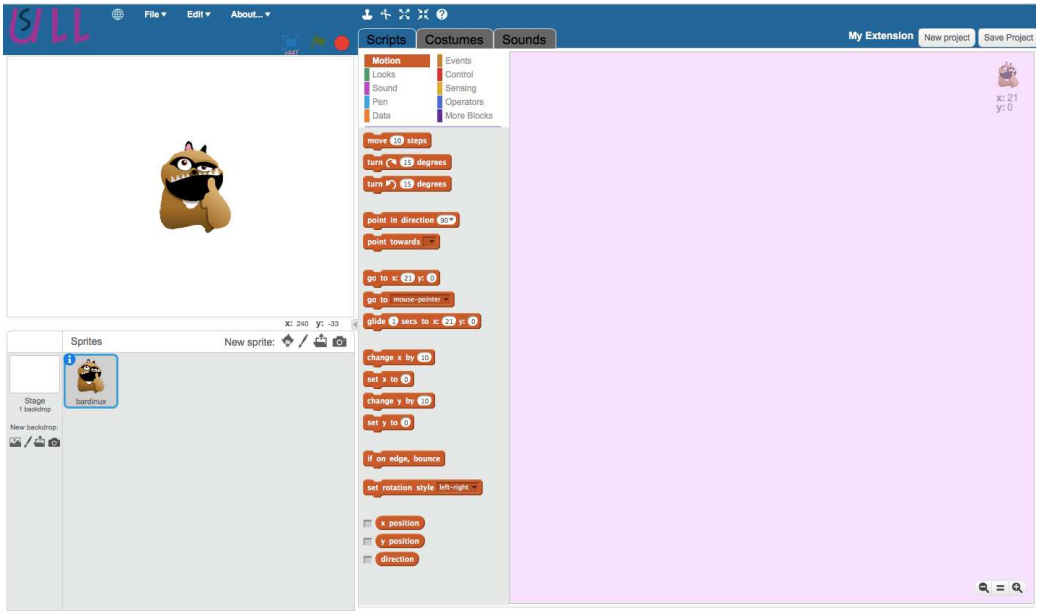


Figura 2.1: Interfaz de Scratch modificada.

Esta versión se ha adaptado cambiando los colores de los bloques y modificando los colores de la interfaz. Como se puede observar en la imagen, hay varios paneles en los que se llevará a cabo la "implementación"de la aplicación.

- **Ejecución en tiempo real.**

Una clave de Scrath es la ejecución en tiempo real. No requiere un paso de compilación previo. Al seleccionar los bloques de la paleta de comandos correspondiente y asociarlos a algún Sprite se ejecuta en el mismo momento la funcionalidad deseada. Esto ayuda a los usuarios a tener una retroalimentación rápida y precisa.

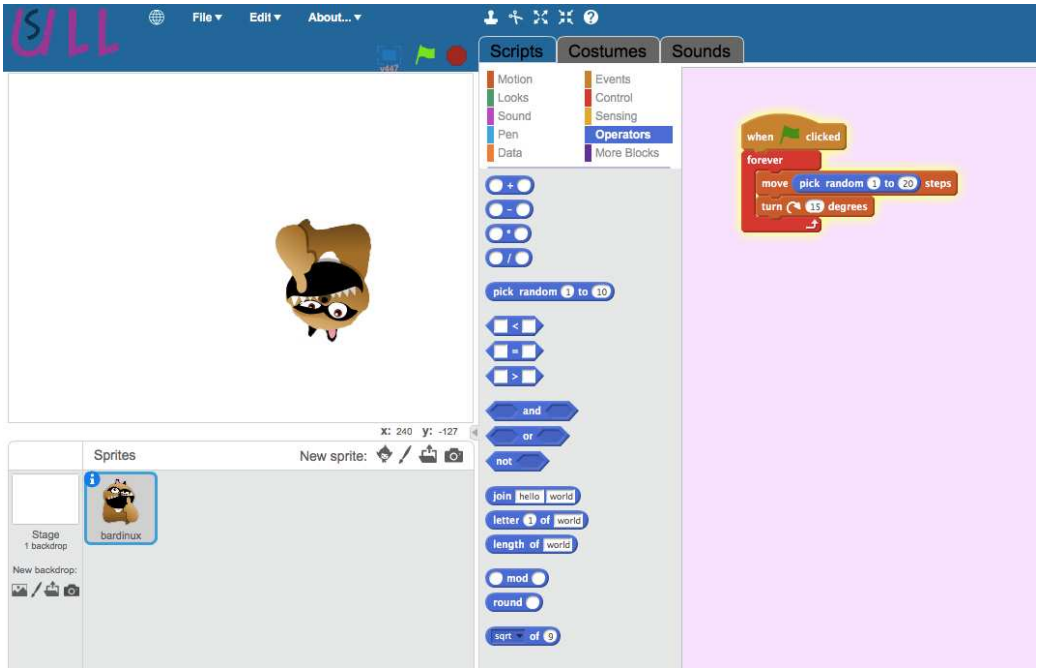


Figura 2.2: Ejemplo de ejecución en tiempo real.

Cualquier modificación que se realice en los bloques hechos cambiará en tiempo real el funcionamiento del Sprite elegido.

- **Ejecución de Scripts visible.**

  Otro dato a destacar de Scratch es la retroalimentación visual para observar la ejecución del Script. Cuando un script (o programa) está en funcionamiento se rodea de un borde amarillo brillante. Esto es primordial ya que ayuda al usuario a saber qué bloque está en ejecución y que funcionalidad se está llevando a cabo. Además, así se verificará si alguno de los grupos de bloques no se está ejecutando.

Figura 2.3: Ejemplo de script en funcionamiento

- **Carencia de mensajes de error.**

  Scratch carece de mensajes de error a la hora de establecer los bloques. Los errores de sintaxis son eliminados ya que los bloques se ensamblan solo en casos en los que tenga sentido el ensamblaje entre ellos. Además, Scratch también se esfuerza por eliminar los errores en tiempo de ejecución haciendo que todos los bloques se ejecuten aunque haya algún error en el funcionamiento.

- **Realización de datos concretos, variables y listas.**

  Scratch permite el establecimiento de variables y listas. Propone estos datos para que el usuario pueda manipular cada uno de ellos según sus objetivos. Con ello, establece en el escenario un monitor de variables y un monitor de listas con el objetivo de observar los cambios realizados en estos datos. Las diferentes funciones de estos dos tipos de datos son:

Figura 2.4: Ejemplo de creación de variables.

Figura 2.5: Selección de variables y listas.

Figura 2.6: Componentes de las variables

Figura 2.7: Componentes de las listas

En cuanto al lenguaje de programación destacamos varios puntos importantes de la plataforma:

- **Sintaxis de bloques.**

  Los programas (scripts) de Scratch se construyen mediante la concatenación de los bloques propios de Scratch representando bloques de condición, expresiones y estructuras de control.

  Para situar estos bloques de comandos en la zona de programas se utilizará el mecanismo de «arrastrar y soltar» con el objetivo de construir funcionalidades. Las formas de estos bloques sugerirán como ensamblar los distintos bloques ya que el mecanismo nombrado anteriormente rechazará el ensamblaje entre dos bloques en el caso de que no tenga sentido.

Figura 2.8: Sugerencia de ensamblaje.

A continuación se diferenciarán los distintos tipos de bloques propios de Scratch:

- **Bloque de comando**.

  Los bloques de comandos se caracterizan por tener un "hoyo"en la parte superior y una protuberancia correspondiente en la parte inferior. Estos bloques se pueden unir para crear una secuencia de comandos llamada Pila (o stack).

Figura 2.9: Ejemplo de bloque de comandos.

- **Bloque de función**
  Los bloques de función devuelven un valor. Además, no constan de ningún «hoyo» y de ninguna protuberancia en su estructura.

Figura 2.10: Ejemplo de bloque de función.

- **Bloque de disparador o trigger**
  Los disparadores constan de una protuberancia curva en su parte superior. Comienza a ejecutar las sentencias anidadas en él cuando el evento del disparador se inicia.

Figura 2.11: Ejemplo de bloque disparador o trigger.

- **Bloque de estructuras de control**
  Los bloques de estructuras contienen espacios en las que se posicionarán las sentencias que se vayan a ejecutar si se cumple la condición establecida.

Figura 2.12: Ejemplo de bloque de control de estructuras.

- **Tipos de datos**
  Scratch posee tres tipos de datos principales: booleano (boolean), número (number) y cadena (string). Estos son los únicos tipos de datos que pueden

ser utilizados en expresiones, en variables o devueltos en funciones ya hechas.

En Scratch, la forma del espacio del parámetro indica el tipo de dato que espera y la forma del bloque de función indica el tipo que se ha devuelto. Scratch solo permite a un bloque de función ser insertado en un espacio de parámetro si el resultado no viola las restricciones de los tipos de datos.

Figura 2.13: Tipo de dato booleano.

Figura 2.14: Tipo de dato Number.

Figura 2.15: Tipo de dato String.

- **Sprites: Modelo de objetos**

  Los sprites son objetos, encapsulan estados y comportamientos, esto es variables y programas (scripts). Sin embargo, aunque posea esto, no tiene ni clases ni herencia por lo que es un lenguaje basado en objetos pero no un lenguaje orientado a objetos.

  Cada sprite tiene su propia zona de programas. Esto tiene sus ventajas, ya que cada Script es independiente y no existen mezclas entre programas. Con esto, se incrementa la facilidad para utilizar Scratch y además, cualquier cambio que se realice en los programas de un Sprite, sólo tendrán efecto en este Sprite.

  Sin embargo, esto tiene su lado negativo también, ya que tendrá complicaciones en el momento de establecer comportamientos similares para cada

uno de los sprites. No existe una jerarquía de clases, ni herencia entre ellas para difundir las funcionalidades a las clases hijas.

Por último, otra de las ventajas de las últimas versiones de Scratch es la comodidad de la opción «Duplicate» ya que no se tendrá que copiar manualmente los programas de los Sprites en el caso de que se quieran utilizar los mismos.

- **Comunicación inter-sprite**

  Como mencionamos anteriormente, Scratch no puede llamar a los programas de otros Sprites directamente (Cada Sprite tiene sus propios programas). En lugar de eso, Scratch utiliza el término llamado "difusión o broadcast"para permitir la comunicación y sincronización entre Sprites.

  Una difusión dispara todos los programas (los ejecuta) en todos los Sprites que empiezan con el bloque disparador "when I receive «msg»". La difusión de Scratch sigue el modelo **uno a muchos** ya que la difusión puede lanzar muchos programas (posiblemente en muchos Sprites); «débilmente acoplado» ya que no importa cuantos receptores hay y "asíncrono"porque el comando de "difusión"no espera hasta que el programa lanzado se haya completado.

Figura 2.16: Comunicación Inter-sprite.

- **Procedimientos**

  Versiones tempranas de Scratch han tenido el mecanismo de creación de procedimientos. Sin embargo, muchos usuarios confundían el término procedimiento con el término difusión ya que los dos asociaban un nombre con una colección de comandos. Para simplificar la herramienta los procedimientos fueron eliminados antes de que Scratch fuera lanzado. Sin

embargo, en la versión 2.0 de Scratch se han añadido estos procedimientos con el objetivo de realizar conjuntos de bloques específicos para alguna acción.

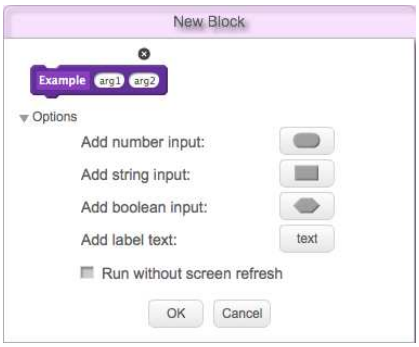


Figura 2.17: Selección del nombre del procedimiento y los tipos de datos de los argumentos si los hubiese.

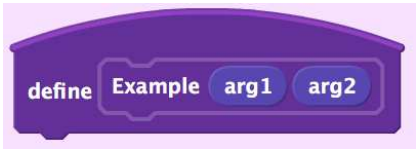


Figura 2.18: Vista del procedimiento.

- **Concurrencia**

  La concurrencia o multihilo ha sido considerada una ténica de programación bastante avanzada. En este caso, los Sprites de Scratch presentan concurrencia ya que pueden llevar a cabo varias tareas a la vez. Esto tiene un efecto bastante positivo en el desarrollo de aplicaciones ya que se pueden llegar a hacer aplicaciones bastante complejas.

Figura 2.19: Ejemplo de concurrencia de un Sprite

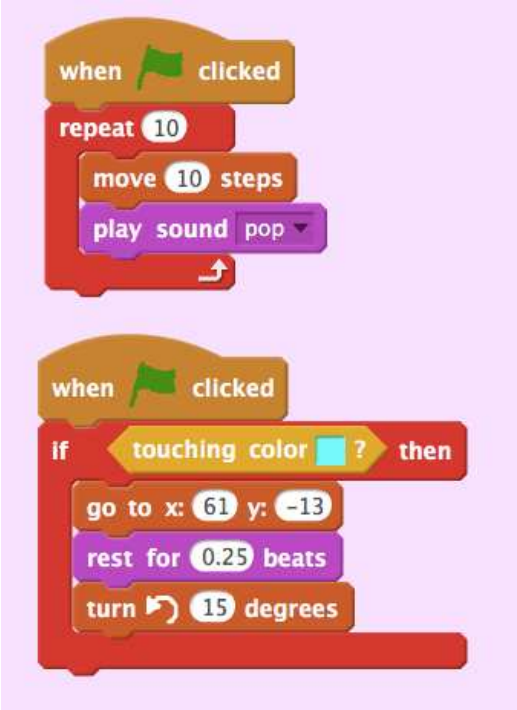


Figura 2.20: Ejemplo de concurrencia de un Sprite

2.2. Aplicaciones

A lo largo de la trayectoria de Scratch se han llevado a cabo numerosas aplicaciones desarrolladas con esta plataforma, entre ellas 'Hour of code' en la que Scratch ofrecía una serie de niveles en cada una de las aplicaciones observando el código en JavaScript que generaban cada uno de estos niveles (ordenados por orden de dificultad).

Numerosas instituciones realizan cada año eventos y cursos orientados al pensamiento computacional y a Scratch concretamente, ya que es la herramienta mayormente utilizada por colegios y por usuarios.

En líneas generales, podemos concluir que está orientado principalmente al uso educativo en las instituciones.

2.3. Versiones

Scratch comenzó a utilizarse en el año 2003 con su primera versión Scratch 1.0. Posteriormente se fue mejorando esta aplicación hasta llegar a la última versión de escritorio, Scratch 1.4. Estas dos versiones fueron desarrolladas en Squeak, una implementación de Smalltalk[1] [14].

La segunda versión de Scratch, Scratch 2.0, está desarrollada bajo Adobe Flash escrita en ActionScript. A diferencia de la versión 1.4 disponemos en este caso de dos versiones, la versión online (propia de la plataforma del MIT, Massachusetts Institute of Technology) y la versión offline, tanto para trabajar en versión de escritorio como para desarrollar código a partir del código fuente propio de Scratch (a esta parte se le llama Scratch 2.0 offline de desarrollo).

En nuestro caso, se ha cogido el código fuente de scratch y se ha modificado la interfaz con el objetivo de orientarla a una propia versión de Scratch.

Una de las modificaciones que se han introducido en la versión de Scratch 2.0 ha sido la implementación de procedimientos y el aumento del número de bloques y características de los Sprites. Además, la interfaz de usuario se ha visto modificada positivamente respecto a Scratch 1.4.

[1]Lenguaje de programación orientado a objetos desarrollado por el grupo de investigación Xeroy PARC con propósitos educacionales

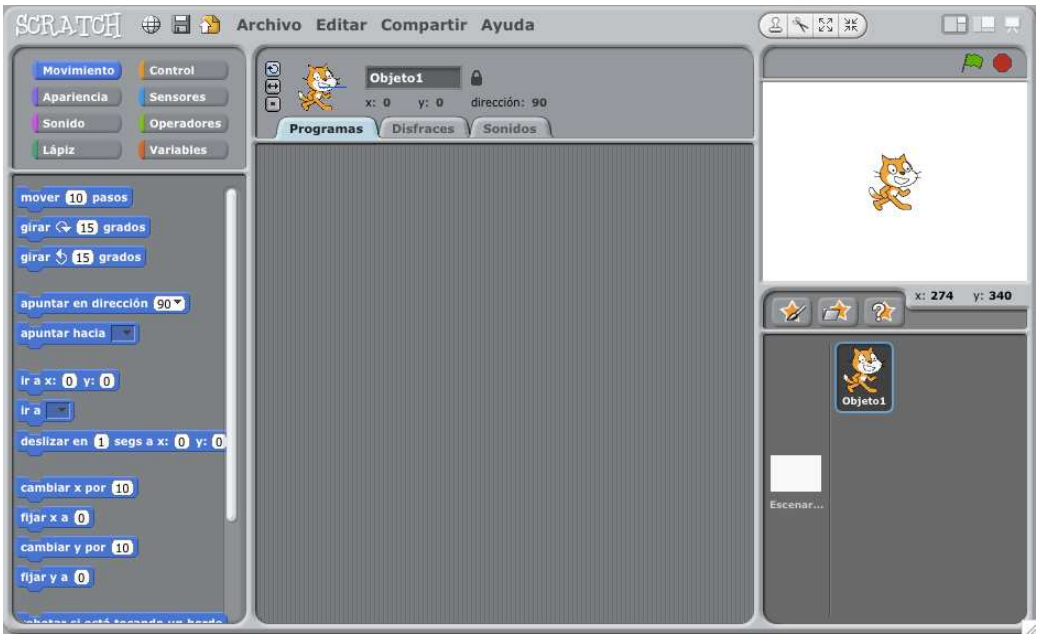


Figura 2.21: Scratch 1.4

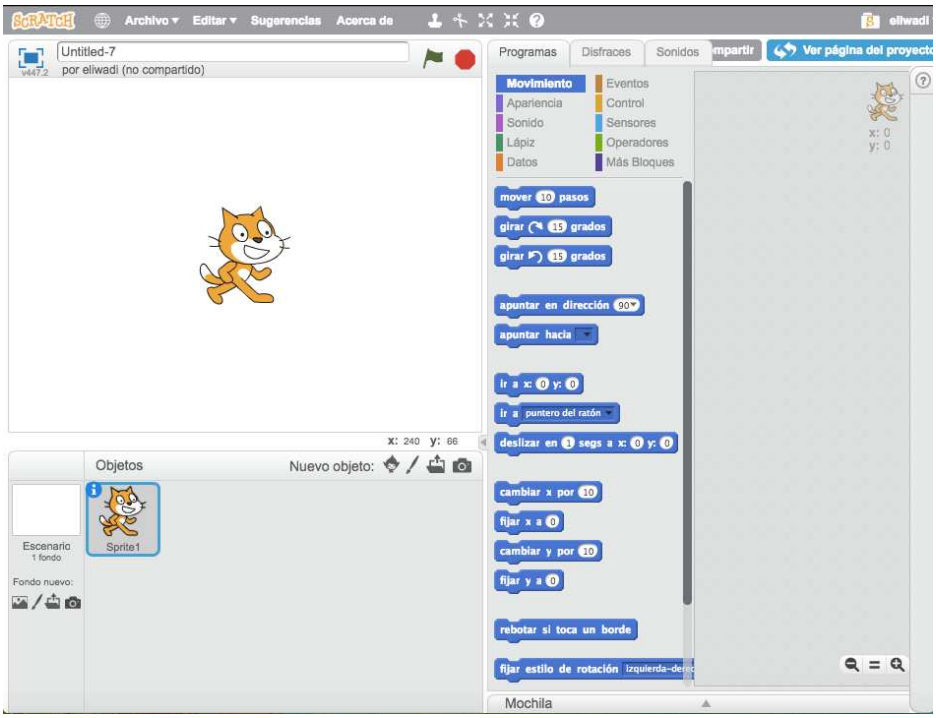


Figura 2.22: Scratch 2.0 y Scratch 2.0 Offline de Escritorio

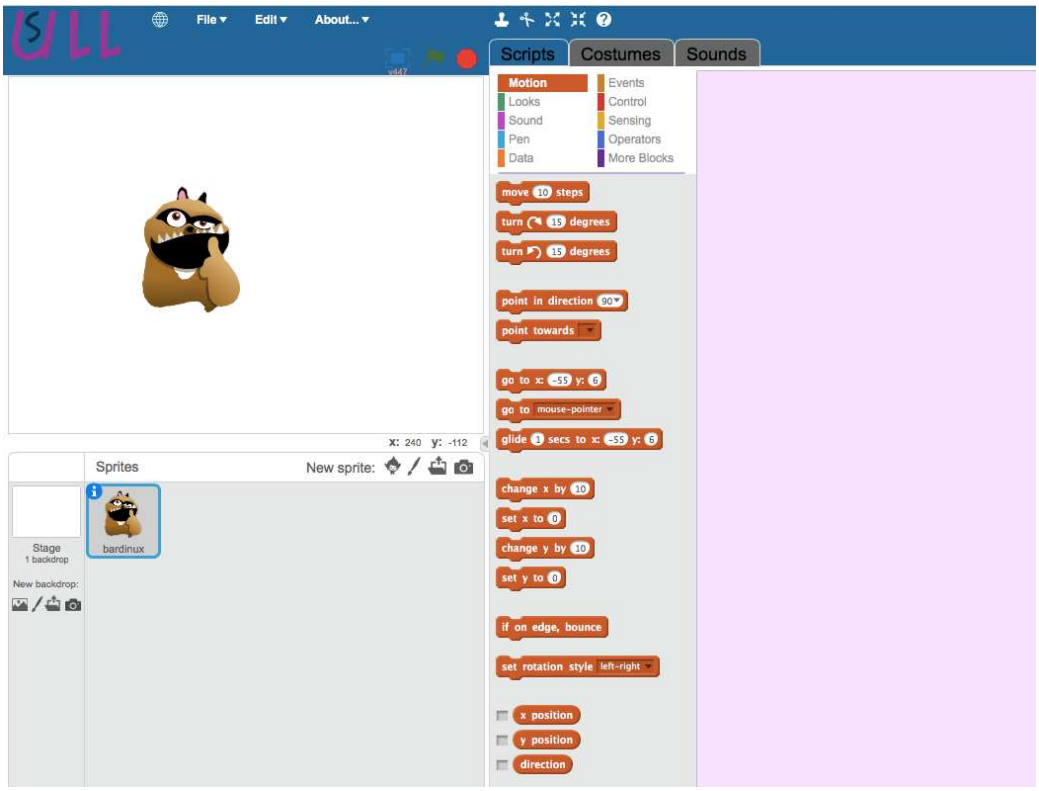


Figura 2.23: Scratch 2.0 Offline de Desarrollo

2.4. Arquitectura

2.4.1. Estructura de directorios

Scratch posee una estructura de directorios extensa. Entre estos directorios se encuentran algunos como, bloques, extensiones, primitivas, UI, media, y numerosos ficheros intrínsecos en estos directorios. Todos éstos poseen subdirectorios, los cuales a su vez poseen ficheros anidados que se utilizarán para la posterior compilación y prueba de la aplicación.

Cada uno de ellos posee una funcionalidad, por lo que la mayoría son ficheros de propósito específico. Una de las características de las clases de los ficheros es la herencia, ya que de esta forma se establecen las interconexiones entre todos los ficheros de la aplicación. El fichero principal es Scratch.as y posee todas las anidaciones de los directorios en los que se encuentra todo el ćodigo fuente de Scratch.

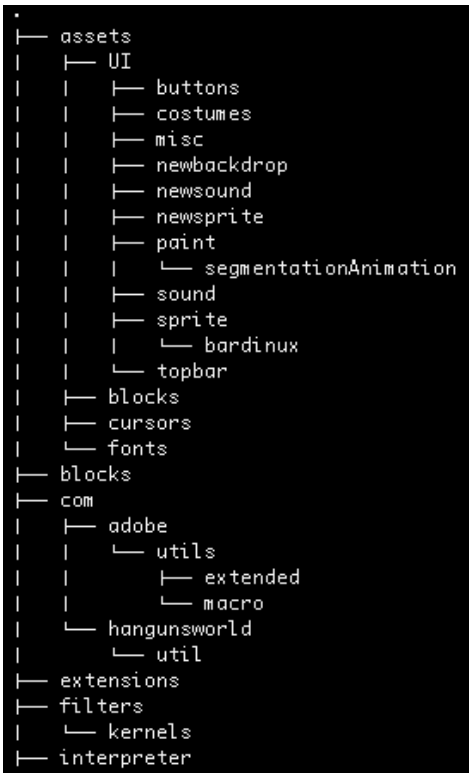


Figura 2.24: Estructura de directorios.

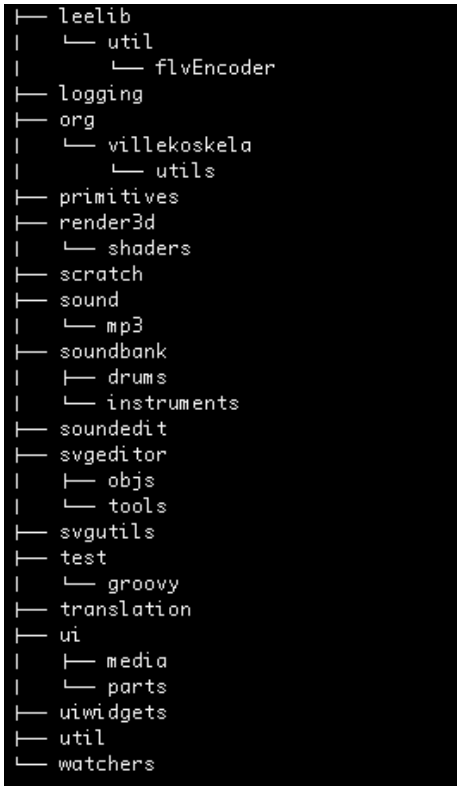


Figura 2.25: Estructura de directorios.

2.4.2. Lenguaje ActionScript

Adobe ActionScript [1] (fecha de lanzamiento en 1997) es el lenguaje de programación de la plataforma Adobe Flash. Originalmente desarrollado como una forma para que los desarrolladores programen de forma más interactiva. La programación con ActionScript permite mucha más eficiencia en las aplicaciones de la plataforma Flash para construir animaciones de todo tipo, desde simples a complejas, ricas en datos e interfaces interactivas.

La última versión de este lenguaje es utilizada en las últimas versiones de Adobe Flash y Flex

Todos los ficheros de la estructura de directorios de Scratch pertenecen a este lenguaje de programación.

2.4.3. Adobe Flash

Adobe Flash [2] es una aplicación de reproducción multimedia. Soporta numerosas aplicaciones web, aplicaciones de escritorio. móviles, gráficos, animaciones, etc. Permite reproducir archivos en formato SWF, los cuales son creados con la herramienta Adobe Flash 2, con Adobe (Apache) Flex o con otras herramientas. En nuestro caso creamos este archivo mediante Adobe Flex, construyendo la estructura de directorios a partir de la herramienta Gradle[2].

2.4.4. Apache Flex

Adobe Flex [3] es un término que agrupa una serie de tecnologías para dar soporte al despliegue y desarrollo de Aplicaciones Enriquecidas de Internet, basadas en su plataforma propietaria Flash.

Flex fue inicialmente lanzado como una aplicación de la J2EE o biblioteca de etiquetas JSP que compilaba el lenguaje de marcas Flex (MXML) y ejecutaba mediante ActionScript aplicaciones Flash (archivos SWF binarios). Versiones posteriores de Flex soportan la creación de archivos estáticos que son compilados, y que pueden ser distribuidos en línea sin la necesidad de tener una licencia de servidor.

La última versión del SDK es la 4.11.0, y se liberó bajo licencia Apache, versión 2.

2.5. Tecnologías utilizadas

2.5.1. Gradle

En este proyecto se utiliza Gradle [6] como herramienta de automatización de la construcción del código (build). Se apoya en Groovy y en un DSL (Domain Specific Language) para trabajar con un lenguaje sencillo y claro a la hora de construir el build. Además, dispone de una gran flexibilidad que permite trabajar con ella utilizando otros lenguajes, en nuestro caso con ActionScript, con disponibilidad por otro lado de un sistema de gestión de dependencias

[2]Herramienta de automatización de construcción de código

sólido.

Para el uso de esta herramienta se ejecuta el comando **./gradlew build** en la estructura donde se encuentre el directorio de códigos fuente (/src) y los ficheros propios de la tecnología Gradle. Una vez ejecutada la acción se creará un directorio llamado **build/** en el que habrá un fichero propio de adobe flash llamado **Scratch.swf**. Al abrir este fichero mostrará su contenido en el navegador predeterminado del sistema.

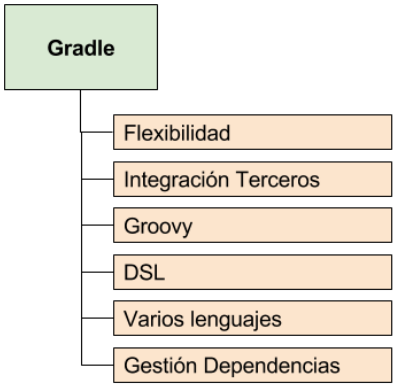


Figura 2.26: Características de Gradle.

2.5.2. Ficheros JSON

Los sprites (imágenes) de Scratch se forman apartir de este tipo de ficheros. Como se estableció en el punto anterior, cada Sprite es un objeto, por lo que el almacenamiento de estas imágenes se lleva a cabo mediante la ordenación de objetos. En este caso, ficheros JSON. Los ficheros JSON se encargan de identificar y gestionar datos. Una gran ventaja de estos ficheros es que puede ser leído por cualquier lenguaje de programación. Por lo tanto, puede ser usado para el intercambio de información entre distintas tecnologías.

A continuación veremos el método utilizado para obtener estos ficheros JSON del servidor externo de Scratch. Una vez obtenidos estos ficheros se procesan

para ser leídos y así obtener los sprites correspondientes mostrándolos en la UI[3] de Scratch.

```
public function getMediaLibrary(libraryType:String, whenDone:Function):URLLoader
{
                var url:String = getCdnStaticSiteURL() + 'media/' + libraryType
                    + 'Library.json';
                return serverGet(url, whenDone);
}
```

En este método se almacena la URL del servidor externo de Scratch. Después de esto se obtienen los sprites del servidor externo. Se hace una llamada a esta URL [4] que contiene un directorio llamado **media/** el cual contiene varios ficheros JSON por cada una de las categorías de Scratch. Estos ficheros poseen llamadas a los ficheros JSON asociados a cada imagen. Una vez obtenidos se procesan mediante los métodos pertenecientes al fichero JSON.as de la estructura de directorios.

La estructura de estos ficheros consta de varias claves (nombre del atributo) con sus correspondientes valores (valor del atributo):

- **N**ombre del objeto.
- **Sonidos:** En el caso de que un sprite tenga sonidos propios se establecerá un conjunto llamado "Sonidos"que contendrá a su vez los siguientes atributos principales:
  - Nombre del sonido.
  - ID del sonido.
  - Fichero del sonido.
- **Costumes o disfraces:** En el caso de que un sprite tenga varias formas se establecerá un conjunto llamado "Costumes"que contendrá a su vez los siguientes atributos principales:
  - Nombre del costume.
  - Fichero del costume.
  - Posición x e y.

[3] User Interface, Interfaz de usuario de Scratch

[4] Uniform Resource Locator

2.5.3. Ficheros de traducción .po

Los ficheros de traducción .po son ficheros en los que se encuentra cada una de las traducciones que vayamos a hacer en nuestra aplicación. Cada idioma o traducción se encuentra en un fichero de este tipo.

En el fichero propio de la estructura de directorios de Scratch llamado Server.as posee un método propio de lectura de estos ficheros. Este método hace una llamada al servidor externo y selecciona del directorio **/locale** el fichero correspodiente al lenguaje seleccionado en la interfaz de usuario.

```
public function getPOFile(lang:String, whenDone:Function):void {
                serverGet('locale/' + lang + '.po', whenDone);
                //DialogBox.notify('Se ha leido fichero');
}
```

Estos ficheros contienen identificadores (msgid) y textos (msgstr) que debe aparecer en sustitución del identificador. Por cada sentencia a traducir habrá un identificador que será la sentencia original y un texto que será la sentencia traducida. En el siguiente ejemplo se muestra una traducción de un texto:

```
msgid "Example"
msgstr "Ejemplo"
```

2.6. Pruebas unitarias[5] con FlexUnit

En esta sección se nombrará las pruebas unitarias realizadas orientadas a Adobe Flash y Apache Flex. Estas pruebas se llevan a cabo con la herramienta FlexUnit, la cual se basa en pruebas unitarias para proyectos utilizados con Apache Flex y ActionScript. Además, realizamos la compilación del código fuente a través de la herramienta Gradle, por lo que estarán orientadas a proyectos utilizados con esta herramienta.

Existen varias versiones de FlexUnit, pero en este caso se utiliza FlexUnit 4.1 ya que trabajamos con el gestor de construcción GradleFx. En el caso de Scratch se podrán utilizar las pruebas para comprobar el correcto funcionamiento de los bloques, la importación correcta de sprites, la correcta traducción de la interfaz u otras funcionalidades que se quieran comprobar.

Para realizar estas pruebas se llevan a cabo los siguientes pasos obtenidos de la página oficial que integra GradleFx con FlexUnit[5]:

1. En primer lugar, se debe especificar las dependencias de FlexUnit descargando las librerías específicas necesarias desde su sitio y luego desplegarlos en el repositorio o dependencias de uso basado en archivos. Una vez definidas las dependecias habrá que definirlas en el fichero de construcción de código fuente.

   - Dependencias en el repositorio

```
dependencies {
        test group: 'org.flexunit', name: 'flexunit-tasks', version:
            '4.1.0-8', ext: 'swc'
        test group: 'org.flexunit', name: 'flexunit', version:
            '4.1.0-8', ext: 'swc'
        test group: 'org.flexunit', name: 'flexunit-cilistener', version
            : '4.1.0-8', ext: 'swc'
}
```

   - Cuando tengamos instalado FlexUnit en nuestra máquina:

```
def flexunitHome = System.getenv()['FLEXUNIT_HOME']  //FLEXUNIT_HOME is
    an environment variable referencing the FlexUnit install location
dependencies {
    test files("${flexunitHome}/flexunit-4.1.0-8-flex_4.1.0.16076.swc",
               "${flexunitHome}/flexUnitTasks-4.1.0-8.jar",
               "${flexunitHome}/flexunit-cilistener-4.1.0-8-4.1.0.16076.
                   swc",
}
```

[5]Manera de comprobar el correcto funcionamiento de un módulo de código

2. Por consiguiente, se tendrá que especificar la ubicación del archivo ejecutable de Flash Player. Gradle utiliza la variable de entorno FLASH PLAYER EXE por convención que debe contener la ruta al ejecutable. Si no se utiliza esta variable de entorno se puede reemplazar con la propiedad "flexUnit.command". El ejecutable se descarga desde la página oficial de Adobe Flash.

3. Por último,se siguen las siguientes partes:

   - Utilizar src/test/actionscript como el directorio fuente de las clases de los Test.
   - Utilizar src/resources como el directorio de los recursos de los tests.
   - Cada uno de los nombres de los ficheros Test terminarán con el nombre "Test.as".

4. Para iniciar las pruebas unitarias se utilizará la siguiente línea de comandos integrada en Gradlefx:

```
$./gradlew test
```

Capítulo 3

Casos de estudio: Algoritmos evolutivos

En este punto se nombrará el concepto de algoritmos evolutivos, así como sus paradigmas y los operadores genéticos utilizados en éstos. Posteriormente, se hará un caso de estudio de estos algoritmos con dos herramientas de programación visual orientadas a bloques, Scratch y AppInventor.

3.1. Algoritmos evolutivos

3.1.1. Definición

Un algoritmo evolutivo es un método de optimización y búsqueda de soluciones basado en postulados de evolución biológica. Los algoritmos evolutivos poseen tres paradigmas principales, la programación evolutiva, las estrategias evolutivas y los algoritmos genéticos, en los cuales nos centraremos para desarrollar los casos de estudio.

3.1.2. Algoritmos genéticos

Según Jonathan E. Rowe en su trabajo titulado "Genetic Algorithm Theory" [16] define el concepto de algoritmo genético dependiendo de una generación constituida por una población (de tamaño N) de individuos (tamaño M) y de la

utilización de operadores genéticos (selección, cruce, mutación) para producir la siguiente población de individuos. Asimismo, son métodos utilizados para la resolución de problemas de búsquedas y optimización basado en llegar a la solución más óptima deseada.

Desde otro punto de vista, Goldberg en su libro "Genetic Algorithms in Search, Optimization and Machine Learning"[10] hace referencia al algoritmo genético de John Holland (1965) estableciendo que cualquier algoritmo genético constituye un algoritmo de búsqueda basado en la mecánica de selección natural y genética natural.

Por último, según el proceso llamado "Markov Chains"hay un número finito de posibles poblaciones de tamaño N. La probabilidad de producir una población particular en una generación dependerá solamente de la anterior población de individuos.

Según este último postulado, para producir la siguiente generación de individuos se deben seguir los siguientes pasos N veces, los cuales constituyen un algoritmo genético simple o canónico:

- Seleccionar dos individuos de la población.
- Realizar un cruce para formar una descendencia.
- Llevar a cabo la mutación.
- Insertar el resultado en la nueva población.

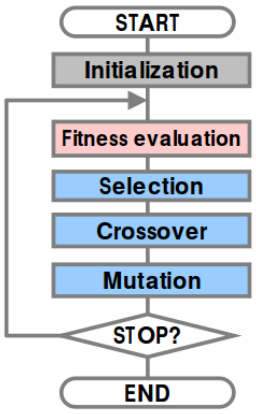


Figura 3.1: Algoritmo genético simple o canónico

3.1.3. Operadores genéticos

A continuación, se hará una pequeña definición para cada uno de los operadores propios de los algoritmos genéticos que se utilizarán en los casos de estudio.

Selección

En este proceso, se seleccionan los individuos de los que se realizará el cruce. Estos individuos se seleccionan según su función de adaptación. Se entiende por función de adaptación como el nivel de adaptación del individuo en el medio correspondiente[16].

Cruce

En el proceso de cruce, se seleccionan dos individuos y se produce una descendencia de la mezcla de éstos. Para cada i,j y k perteneciente al conjunto (omega) necesitamos especificar la probabilidad de cruce entre i y j para que produzca k. La función heurística de esta probabilidad se denomina como:

$$C(p)_k = p_i p_j r(i, j, k) \tag{3.1}$$

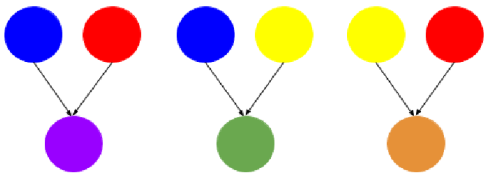


Figura 3.2: Ejemplo de cruce.

En este ejemplo se lleva a cabo un cruce entre tres parejas de individuos distintos. El cruce es el resultado de la mezcla de colores entre ellos. En el punto siguiente se hará una demostración con las herramientas elegidas para promover el pensamiento computacional.

Mutación

En el caso del operador de mutación se aplica a cada uno de los descendientes de manera individual. Este caso consiste en la alteración aleatoria de la información del invividuo. Por lo tanto, mutar un individuo de la población producirá otro individuo distinto.

3.2. Casos de estudio

En los siguientes puntos se detallarán los algoritmos evolutivos realizados a través de dos herramientas orientadas al pensamiento computacional. Se ha creado un caso llamado S-Balls en el que tres Sprites interactuarán entre sí explicando los operadores genéticos del paradigma de Algoritmo genético, estos son, selección, cruce y mutación. Además, se hará una comparativa entre las distintas herramientas teniendo en cuenta la facilidad de uso y la rapidez de realización al encontrarnos con un problema.

Estamos ante un escenario en el que poseemos una población de individuos (sprites) de los cuales se seleccionarán dos con el objetivo de realizar el cruce entre ellos. Una vez obtenido un descendiente de estos se procederá a mutar uno de ellos.

3.2.1. Caso de estudio con Scratch

En este caso, se ha realizado el caso de estudio S-Balls con Scratch en el que tres sprites (tres pelotas de colores primarios) se mueven en el escenario (backdrop) hasta que dos de estos individuos se cruzan y se produce la descendencia de este cruce (mezcla de colores primarios). En el caso de las mutaciones, cada X tiempo aleatorio una de las pelotas mutará y cambiaría.

A continuación se verán unas imágenes de este caso de estudio en el que se muestra los distintos operadores básicos de los algoritmos genéticos y su aportación en este caso.

- En este primer caso se observa la población de individuos en el escenario.

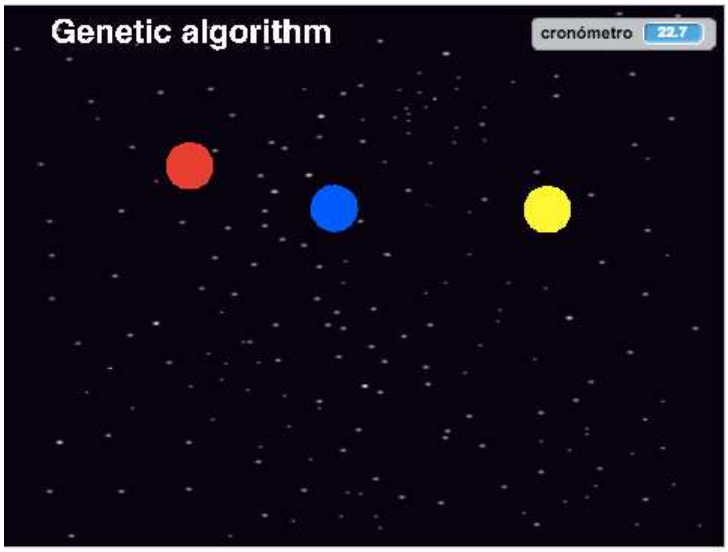


Figura 3.3: Población de individuos

- En las figuras 3.4 y 3.5 se establece un cruce entre los distintos individuos, creando así descendientes de los mismos.

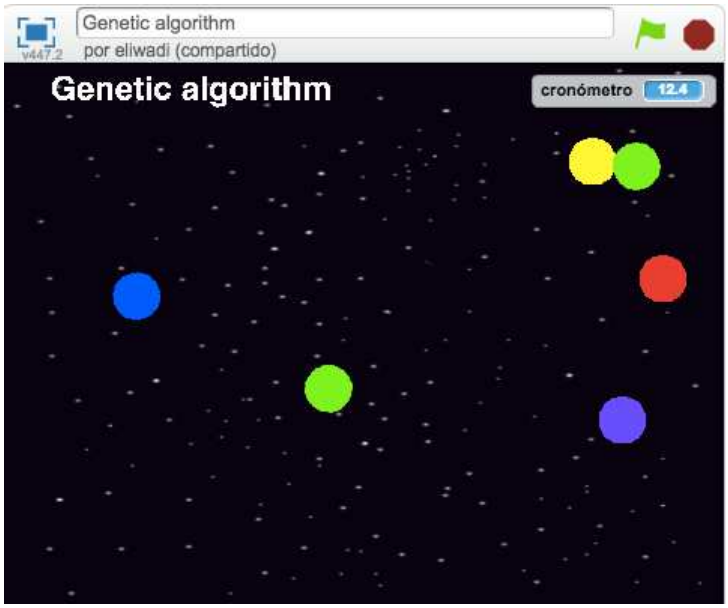


Figura 3.4: Cruce de individuos en Scratch 1

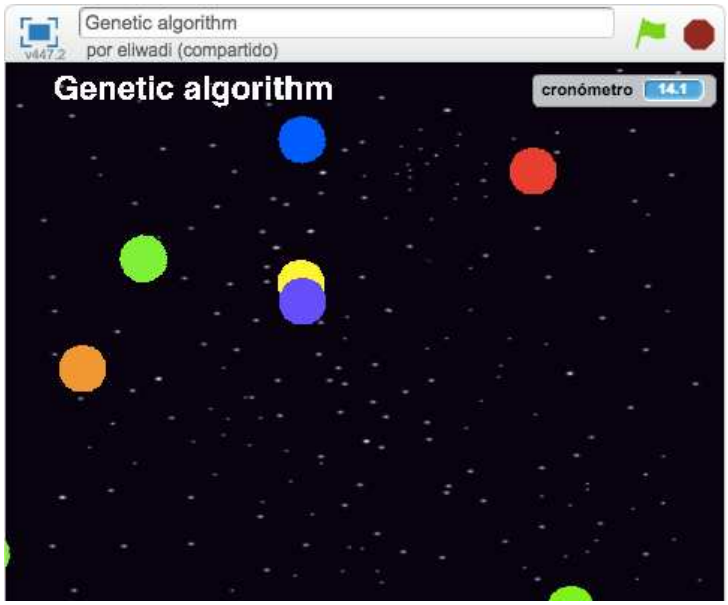


Figura 3.5: Cruce de individuos en Scratch 2

- Por último, el ejemplo de una mutación. En este caso la pelota roja.

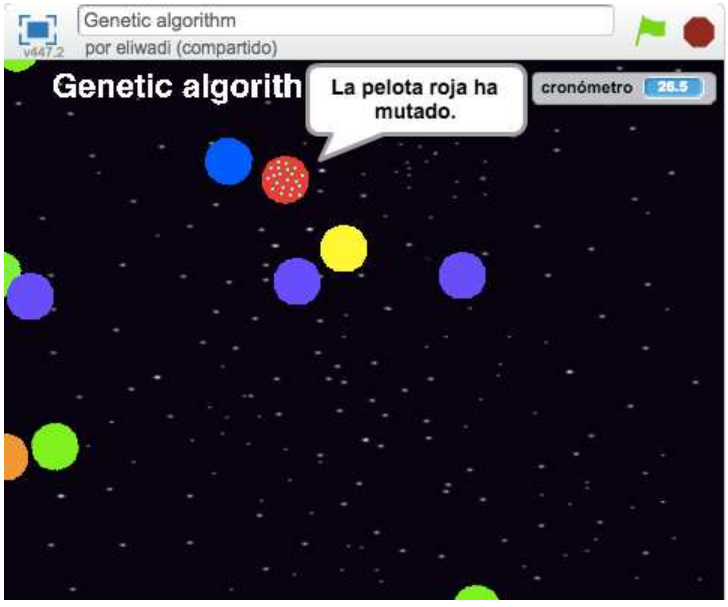


Figura 3.6: Mutación de un individuo de la población en Scratch

3.2.2. Caso de estudio con AppInventor

A continuación se verá el caso de estudio S-Balls aplicado a AppInventor. Esta herramienta constituye un entorno orientado a la programación visual utilizada en sistemas operativos Android. Consiste en desarrollo de aplicaciones mediante bloques utilizando el emulador aiStarter de Android. Para emular el contenido de la aplicación se necesita este emulador en segundo plano. En primer lugar se explicará cómo establecer la conexión con el emulador de AppInventor y en segundo lugar las funcionalidades añadidas a la aplicación con el objetivo de su funcionamiento como algoritmo evolutivo. Los pasos para llevar a cabo la conexión con AppInventor han sido:

1. Realización de la aplicación mediante bloques en la interfaz propia de AppInventor o en su versión de escritorio.
2. **Conexión con el emulador de Android.**

   Para este paso se requiere la instalación del emulador de Android en el sistema operativo deseado. En el caso de Linux se ha instalado la aplicación en **/usr/google/appinventor/commands-for-AppInventor**. Una vez dentro de este directorio lanzamos el ejecutable **./aiStarter** para iniciar el emulador en segundo plano.
3. Visualización de la aplicación mediante simulación de un dispositivo

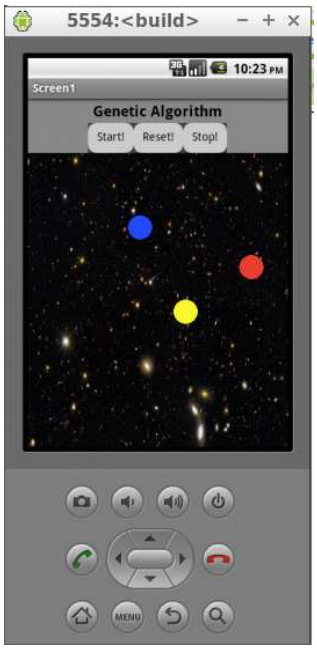


Figura 3.7: Población de individuos en AppInventor

Figura 3.8: Cruce de Individuos en AppInventor

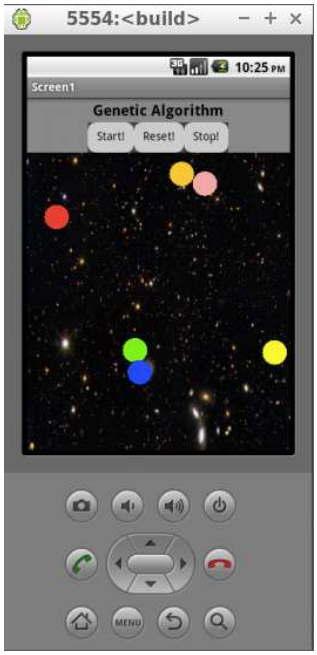


Figura 3.9: Mutación de un individuo de la población en AppInventor.

3.3. Comparativa entre Scratch y AppInventor

En este punto se llevará a cabo una comparativa entre las dos herramientas utilizadas en el caso de estudio del Algoritmo Evolutivo. Los siguientes puntos establecerán comparaciones entre estas dos herramientas, tanto en funcionamiento de las herramientas como en las interfaces gráficas de cada una:

1. Destaca la facilidad de Scratch en el momento de llevar a cabo la sincronización de bloques para los cruces y las mutaciones de los individuos de la población. Esta ventaja es la independencia de cada Sprite, ya que cada uno posee su propia zona de programas. En el caso de AppInventor todos los programas de los Sprites se realizan en la misma ventana o zona, por lo tanto aumenta la complejidad en el momento de posicionar los bloques.

2. AppInventor es una herramienta orientada a usuarios con conocimientos previos de programación, ya que los nombres de los bloques utilizados presentan similitud con los métodos predefinidos en programas. Un ejemplo de esto es: " When «Sprite».collideWith". En cambio, Scratch posee bloques con nombres más intuitivos para los usuarios, por lo que no requiere conocimientos previos de programación.

3. La realización del cruce de los Sprites se ha hecho clonando uno de los Sprites y modificando su color cada vez que se produce un cruce. Esto hace que se generen Sprites dinámicamente según se toquen los dos Sprites padres. En AppInventor esto no se puede llevar a cabo ya que no existe la función "Clone". Debido a esto, se deben crear los Sprites manualmente. Esto es una gran desventaja para AppInventor, ya que limita en este sentido la creación infinita de Sprites.

4. Scratch y AppInventor poseen dos ventajas comunes. Por un lado, cada uno de ellos funciona en tiempo real. Por lo que por cualquier cambio que se haga en el programa los usuarios reciben una retroalimentación del funcionamiento. Por otro lado, al establecer los bloques, si el bloque se intenta posicionar en un espacio y éste es incorrecto, no se podrá posicionar el bloque.

5. En general, la interfaz de Scratch destaca por su accesibilidad y usabilidad. AppInventor no destaca en este sentido, tanto por su posición de elementos en la interfaz gráfica como por sus colores. Sin embargo, al ser una herramienta orientada al desarrollo de aplicaciones en Android destaca por su multitud de funcionalidades orientadas a trabajar con dispositivos móviles como el movimiento de los Sprites con *sensores*.

Capítulo 4

Conclusiones y líneas futuras

4.1. Conclusiones

El pensamiento computacional ha constituido uno de los movimientos más importantes en la tecnología actual. Este hecho concibe la idea del pensamiento humano análogo al de una máquina. Hoy en día, el mundo de la tecnología se encuentra tanto explícitamente como implícitamente alrededor del mundo y al estar alfabetizados digitalmente hacemos uso diario de éstas.

Todas las herramientas analizadas que promueven el pensamiento computacional constan de ventajas e inconvenientes, tanto de implementación como de interfaz del usuario. La más óptima de ellas es Scratch ya que destaca por su usabilidad, su accesibilidad y su facilidad de uso. Constituye una interfaz gráfica bastante intuitiva además de cumplir los Requisitos de Accesibilidad del Software según la norma española UNE 139803 referente a Accesibilidad de la web.

Otra ventaja de esta plataforma es la desigualdad de edades de usuarios que deseen incrementar sus conocimientos de programación, ya que cualquier usuario independientemente de su edad podrá utilizar la interfaz sin problemas mayores. Otras herramientas orientadas a la computación como Alice o AppInventor se centran principalmente en usuarios con conocimientos previos de programación, ya que los bloques establecidos en la interfaz constituyen nombres de métodos de programación propiamente dichos con sus correspondientes tipos de datos.

4.2. Líneas futuras

4.2.1. Integración de Scratch con Hardware

Una de las líneas futuras interesantes en el desarrollo de Scratch es la integración de ésta en placas Arduino[1] con el objetivo de no sólo insertar a la población en el mundo de la programación sino integrarse también en el mundo de la electrónica. Las placas Arduino cada vez son más utilizadas en la enseñanza con el propósito de realizar circuitos simples y así fomentar la docencia en este campo.

Asimismo existen más dispositivos con los que se podrá integrar Scratch. Por ejemplo, las placas Raspberry Pi[2] y derivados constituyen un microprocesador a través del cual se podrá insertar Scratch. A través de este microordenador o microcomputador se podrá almacenar Scratch y este a su vez con un periférico para la utilización de la herramienta. Otras placas como Banana Pi o FPGAs también podrían ser utilizadas para ello.

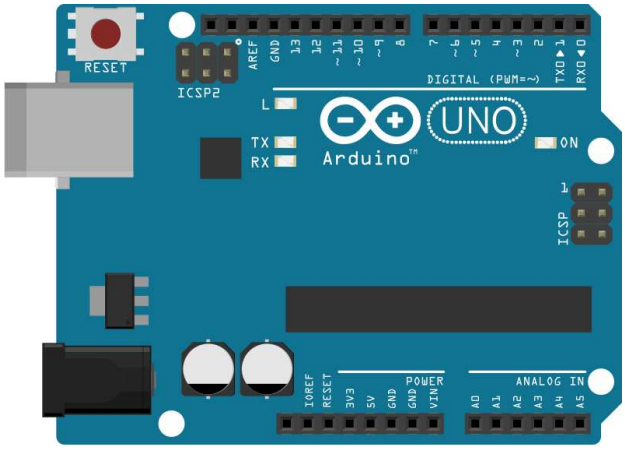


Figura 4.1: Placa Arduino con sus correspondientes partes.

[1]Arduino constituye una placa de desarrollo software y hardware compuesta por circuitos que forman un microcontrolador y un entorno de desarrollo para programar esta propia placa (IDE)

[2]Microprocesador u ordenador de placa reducida

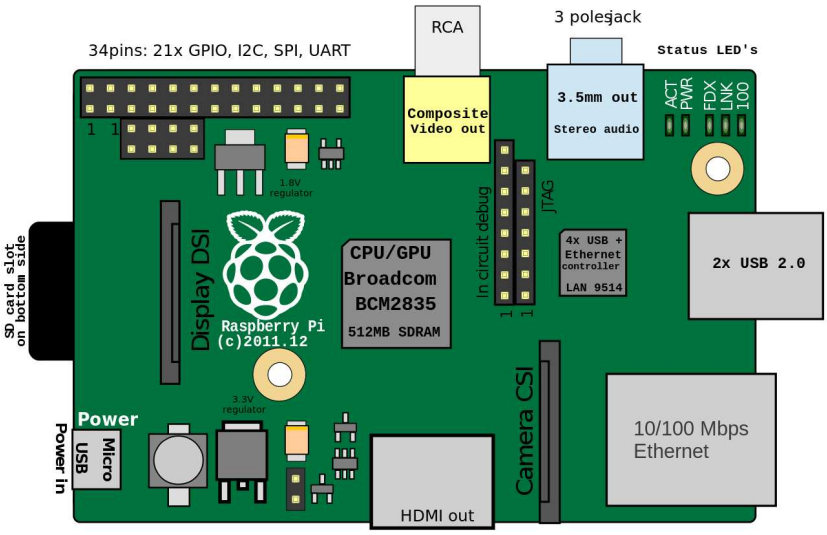


Figura 4.2: Placa Raspberry Pi con sus correspondientes partes.

4.2.2. Almacenamiento de datos en servidores externos

Scratch hace uso de un servidor externo para la obtención y el procesamiento de los Sprites y ficheros .po utilizados en su interfaz. La versión 2.0 offline de desarrollador de Scratch no procesa estas imágenes ni estos ficheros ya que pertenecen al MIT, por lo que son datos protegidos. Una de las posibles modificaciones de este código fuente de Scratch podría ser la obtención de los Sprites de un servidor propio en el que almacenemos tanto Sprites como ficheros de traducción y otros datos.

La adaptabilidad de esta fuente de datos podría ser un gran avance para los futuros desarrolladores de Scratch de la misma institución.

Capítulo 5

Summary and Conclusions

Computational thinking has constituted one of the most important movements in actual technology, considering that it conceives the fact of human thinking as a machine. Nowadays, the world of technology is, both implicitly and explicitly, around the world, and since we are digitally literates, we use them on a daily basis.

All tools analyzed that promote computational thinking consist of advantages and disadvantages, both for implementation and user interface. The most ideal of them is Scratch, because of its usability, accessibility and ease of use. It constitutes a very intuitive graphic interface, as well as obeying the Requirements of Software Accessibility, according to the spanish rule UNE 139803, regarding web accessibility.

Another benefit of this platform is the inequality of the users' ages that wish to increase their knowledge about programming, considering that any user, regardless of his age, will be able to use the interface without facing major problems. Other tools oriented to computation, such as Alice or AppInventor, are focused mainly in users with previous knowledge of programming, since the blocks that are established in the interface constitute names of programming methods, in the strict sense, with their corresponding types of data.

Capítulo 6

Presupuesto

En este capitulo se hará un presupuesto según el tiempo empleado en el desarrollo de este proyecto.

6.1. Presupuesto

Tabla 6.1: Tabla de presupuestos por hora €/h

Tareas	**Horas**	**Presupuesto**
Revisión bibliográfica	60h	5€/h
Escritura de Antecedentes	30h	10€/h
Puesta en marcha de las herramientas	80h	20€/h
Pruebas realizadas	70h	15€/h

Tabla 6.2: Presupuesto total €

Tareas	**Horas**	**Presupuesto**
Total	240h	3.250€

En total el presupuesto sería 3.250€por 240h de realización de trabajo.

Bibliografía

[1] Actionscript documentation @ONLINE. http://goo.gl/jfreZn.

[2] Adobe flash documentation @ONLINE. https://goo.gl/1e21s2.

[3] Adobe flex documentation @ONLINE. https://goo.gl/nWz69L.

[4] Alice official page @ONLINE. http://www.alice.org/index.php.

[5] Flexunit with gradlefx. http://doc.gradlefx.org/en/latest/flexunit.html.

[6] Gradle documentation @ONLINE. http://gradle.org/.

[7] Greenfoot official page @ONLINE. http://www.greenfoot.org/door.

[8] Logo bibliography @ONLINE. http://el.media.mit.edu/logo-foundation/s.

[9] Michal Armoni, Orni Meerbaum-Salant, and Mordechai Ben-Ari. From scratch to 'real' programming. *Trans. Comput. Educ.*, 14(4):25:1–25:15, February 2015.

[10] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1989.

[11] Peter B. Henderson, Thomas J. Cortina, and Jeannette M. Wing. Computational thinking. *SIGCSE Bull.*, 39(1):195–196, March 2007.

[12] John Maloney, Mitchel Resnick, Natalie Rusk, Brian Silverman, and Evelyn Eastmond. The scratch programming language and environment. *Trans. Comput. Educ.*, 10(4):16:1–16:15, November 2010.

[13] Massachusetts Institute of Technology MIT. App inventor bibliography @ONLINE. http://appinventor.mit.edu/explore/get-started.html.

[14] Massachusetts Institute of Technology MIT. Smalltalk @ONLINE. https://wiki.scratch.mit.edu/wiki/Smalltalk.

[15] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, and Yasmin Kafai. Scratch: Programming for all. *Commun. ACM*, 52(11):60–67, November 2009.

[16] Jonathan E. Rowe. Genetic algorithm theory. In *Proceedings of the 9th Annual Conference Companion on Genetic and Evolutionary Computation*, GECCO '07, pages 3585–3608, New York, NY, USA, 2007. ACM.