RED-Revista de Educación a Distancia, 46(6). 15-Sept.-2015
DOI: 10.6018/red/46/6
http://www.um.es/ead/red/46/Basogain.pdf

Pensamiento Computacional a través de la Programación: Paradigma de Aprendizaje

Computational Thinking Trough Programming: A Learning Paradigm

Xabier Basogain Olabe
Universidad del País Vasco / Euskal Herriko Unibertsitatea. España.
xabier.basogain@ehu.es

Miguel Ángel Olabe Basogain
Universidad del País Vasco / Euskal Herriko Unibertsitatea. España.
miguelangel.olabe@ehu.es

Juan Carlos Olabe Basogain
Christian Brothers University. Estados Unidos de América.
jolabe@cbu.edu

Resumen
Este artículo presenta el concepto del Pensamiento Computacional y cómo puede ser integrado en el aula a través del diseño e implementación de proyectos de programación. Se describe la necesidad, el propósito y las principales características del Pensamiento Computacional. Se muestra con varios ejemplos cómo se pueden desarrollar los elementos fundamentales del Pensamiento Computacional utilizando un lenguaje programación. La última sección del artículo muestra el contenido y los resultados del curso "Pensamiento Computacional en la Escuela" impartido en la modalidad MOOC (*Massive Open Online Courses*) en la plataforma Miríada X.

Palabras Clave
Pensamiento Computacional, Programación, Paradigma de Aprendizaje.

Abstract
This article presents the framework of Computational Thinking and how it can be integrated into the classroom through the design and implementation of programming projects. It describes the need, purpose and main features of Computational Thinking. It describes with several examples how to develop the key elements of Computational Thinking using a programming language. The last section of the paper shows the content and outcomes of the course "Computational Thinking in School" taught as a MOOC (Massive Open Online Courses) in the platform Miríada X.

Keywords
Computational Thinking, Programming, Learning Paradigm.

INTRODUCCIÓN

El estudio formal de las competencias computacionales en las escuelas de primaria y secundaria ha sido reconocido por muchas instituciones y administraciones. A modo de ejemplo, Inglaterra, a partir del año académico 2014-15, ha incluido formalmente el estudio del Pensamiento Computacional y programación de ordenadores como parte del plan de estudios de la educación primaria y secundaria, como se describe en el "currículo nacional en Inglaterra: Estudio de Programa de Computación" (Department for Education England, 2013).

La organización Code.org promueve la idea de que todos los alumnos deben tener la oportunidad de aprender programación. Esta organización en su portal web ofrece materiales de programación y promociona que las escuelas incorporen más programación en su currículum. El reto 'la hora de programación' ha logrado que miles de estudiantes de 180 países tengan una primera experiencia con la computación. Esta iniciativa cuenta con el apoyo de personajes públicos relevantes de Microsoft, Facebook y del mundo de la tecnología en general (Code, 2013).

Existen varios entornos de programación diseñados para usuarios noveles en programación. Ente ellos podemos destacar Alice (Alice, 2015), Greenfoot (Greenfoot, 2015) y Scratch (Scratch, 2015). Scratch es un entorno gráfico de programación que permite a los usuarios (principalmente niños de 8 a 16 años) aprender a programar, realizando proyectos personales como crear juegos, contar historias y realizar animaciones. Scratch se construye sobre las ideas del constructivismo de Logo (Kafai and Resnick, 1996), (Papert, 1980) y Etoys (Kay, 2010). Un objetivo clave del diseño de Scratch es apoyar el aprendizaje autodidacta a través de práctica personal y de la colaboración con otros (Maloney et. al, 2010), (Resnick, 2009).

El portal LearnScratch.org está siendo utilizado en la actualidad por más de 5000 centros escolares de todo el mundo (principalmente en Estados Unidos, Reino Unido y Canadá). Ha sido creado para proveer material docente que permita la integración de Scratch de forma directa en el aula. Estos materiales docentes han sido diseñados y orientados para la creación de proyectos Scratch que presentan de forma incremental mayor complejidad y dificultad en su realización.

Uno de los retos en introducir formalmente el estudio de los lenguajes de programación en las escuelas de primaria y secundaria es el número limitado de profesores cualificados para enseñar esta materia. Por esta razón, los portales LearnScratch.org y AprendiendoScratch.org (la versión en castellano de LearnScratch) han sido desarrollados con la metodología denominada enseñanza colaborativa, donde los maestros pueden introducir a sus alumnos no sólo la sintaxis de Scratch sino también, y más importante, el desarrollo de estrategias de diseño (Olabe and Rouèche, 2008), (Olabe, Basogain y Olabe, 2010). Los video tutoriales creados en LearnScratch y AprendiendoScratch desempeñan el papel de introducir a los estudiantes los conceptos y estrategias de diseño. El maestro en clase tiene el papel de apoyar a los estudiantes, promocionar la colaboración y debate, motivar la creatividad de los alumnos, etc.
Además de la metodología de enseñanza colaborativa también se promociona el paradigma del construccionismo: los estudiantes aprenden creando nuevas estructuras en sus mentes que representan los nuevos conceptos aprendidos. Este proceso de

aprendizaje es óptimo cuando al mismo tiempo se crea la misma estructura por los alumnos de una forma física, por ejemplo con los bloques de lego, o conectando bloques de Scratch para formar un proyecto operativo. Este proceso de aprendizaje haciendo 'artefactos', 'programas', permite a los alumnos construir y comprobar, reconstruir, modificar, mejorar, etc. (Papert, 1991).

Este proceso de aprendizaje difiere mucho del que se desarrolla en la mayoría de las actividades en las áreas de matemáticas y ciencias de nuestras escuelas. Por ejemplo, analizando los problemas planteados en el test de PISA (Pisa, 2015) en matemáticas, la gran mayoría de las preguntas pueden resolverse en Scratch con un único bloque de resolución de operaciones aritméticas. Los proyectos realizados por los estudiantes que utilizan Scratch muestran que el cerebro humano es capaz de comprender, diseñar y construir proyectos con muchos bloques y de muchos tipos. Este proceso de aprendizaje a través de Scratch favorece el desarrollo de competencias y habilidades para analizar y resolver problemas utilizando técnicas y estructuras que se utilizan en las ciencias de la computación (Basogain *et al.*, 2012), (Olabe *et al.*, 2011).

El Pensamiento Computacional es "un enfoque para resolver un determinado problema que empodera la integración de tecnologías digitales con ideas humanas. No reemplaza el énfasis en creatividad, razonamiento o pensamiento crítico pero refuerza esas habilidades al tiempo que realza formas de organizar el problema de manera que el computador pueda ayudar" (CSTA and ISTE, 2011).

En los siguientes apartados del artículo se describe de forma general el concepto de Pensamiento Computacional, y dos características relevantes de éste a través del desarrollo paso a paso de dos proyectos en el entorno de programación Scratch (los proyectos del juego Pong y el juego Pack-Man). La última sección describe el objetivo, contenido y resultados del curso "Pensamiento Computacional en la Escuela" impartido en la plataforma Miríada X en formato MOOC (*Massive Open Online Courses*).

PENSAMIENTO COMPUTACIONAL

El Pensamiento Computacional es una metodología basada en la implementación de los conceptos básicos de las ciencias de la computación para resolver problemas cotidianos, diseñar sistemas domésticos y realizar tareas rutinarias. Esta nueva forma de abordar los problemas nos permite resolver con eficacia y éxito problemas que de otra forma no son tratables por una persona.

La principal promotora del Pensamiento Computacional, Jeannette Wing (Wing, 2006), (Wing, 2011) introduce esta nueva forma de abordar los problemas basados en el potencial que ofrece la computación, tanto cuando se realiza con la ayuda de los ordenadores o en las propias personas. Wing describe con detalle las características y propiedades del Pensamiento Computacional. La Tabla I presenta de forma resumida los principales conceptos asociados al Pensamiento Computacional.

Nº	Característica
1	Reformular un problema a uno parecido que sepamos resolver por reducción, encuadrarlo, transformar, simular
2	Pensar Recursivamente
3	Procesar en Paralelo

4	Interpretar código como datos y datos como código
5	Generalizar análisis dimensional
6	Reconocer ventajas y desventajas del solapamiento
7	Reconocer coste y potencia de tratamiento indirecto y llamada a proceso
8	Juzgar un programa por simplicidad de diseño
9	Utilizar Abstracción y descomposición en un problema complejo o diseño de sistemas complejos
10	Elegir una correcta representación o modelo para hacer tratable el problema
11	Seguridad en utilizarlo, modificarlo en un problema complejo sin conocer cada detalle
12	Modularizar ante múltiples usuarios
13	Prefetching y caching anticipadamente para el futuro
14	Prevención, protección, recuperarse de escenario peor caso
15	Utilizar razonamiento heurístico para encontrar la solución
16	Planificar y aprender en presencia de incertidumbre
17	Buscar, buscar y buscar más
18	Utilizar muchos datos para acelerar la computación
19	Límite tiempo/espacio y memoria/potencia de procesado

Tabla I.- Conceptos y características de Pensamiento Computacional.

Una primera y errónea idea que se puede tener del Pensamiento Computacional es creer que es una materia exclusiva para personas del ámbito de la ingeniería informática y computación. Existe un interés y esfuerzo creciente en incorporar el Pensamiento Computacional a través de proyectos, juegos, entornos de programación, etc. en el currículum de escuelas y universidades. Algunos de estos esfuerzos están orientados a estudiantes jóvenes, en particular a las mujeres, en introducir la programación de ordenadores y el Pensamiento Computacional (Repenning, Webb and Ioannidou, 2010), (Google, 2015).

El equipo Scratch de MIT define el Pensamiento Computacional como un conjunto de conceptos, prácticas y perspectivas que se basan en las ideas del mundo de la informática. Los estudiantes al programar y compartir proyectos de Scratch, comienzan a desarrollarse como pensadores computacionales: aprenden conceptos básicos de computación y matemáticas, y a la vez también aprenden estrategias de diseño, resolución de problemas, y otras formas de colaboración (ScratchEd Team, 2015).

Las personas que desarrollan estas técnicas basadas en el ordenador están en disposición de resolver problemas complejos no sólo por sacar provecho de la potencia computacional de los ordenadores sino también por la capacidad de los lenguajes de ordenador en describir sistemáticamente un problema en varias capas de abstracción y de describir el interface entre dichas capas sin ambigüedad. Esta habilidad aumenta de forma absoluta la complejidad de los problemas reales para los cuales podemos encontrar una solución buena y eficiente.

PENSAMIENTO COMPUTACIONAL UTILIZANDO PROGRAMACIÓN

Esta sección muestra el desarrollo de los conceptos del Pensamiento Computacional a través de la realización de dos proyectos en el entorno de programación Scratch. Se han seleccionado los conceptos nº 1 y nº 9 de la Tabla I, por su relevancia dentro del grupo de conceptos listados:

1) los conceptos de la abstracción y descomposición.
2) el concepto de redefinir un problema en la forma de un problema conocido del que se tiene una solución.

Estos conceptos son descritos paso a paso utilizando Scratch como entorno para el desarrollo de los mismos.

Abstracción y Descomposición en resolución de tareas de complejidad alta

Utilizaremos la creación del juego Pong como proyecto de tareas complejas grandes para ilustrar dos elementos esenciales del Pensamiento Computacional: abstracción y descomposición para resolver tareas complejas grandes.

La Figura 1 muestra el diseño del juego Pong escrito en Java. El funcionamiento del juego es sencillo: una bola amarilla rebota entre dos paletas, una está controlada por el jugador a través del ratón, y la otra está controlada por el programa del juego.

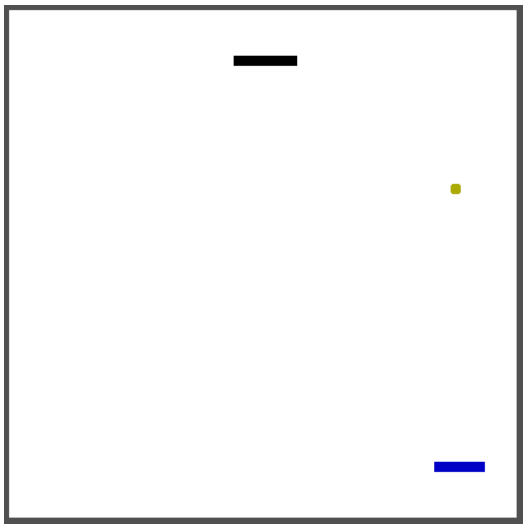

Figura 1.- Diseño del juego Pong escrito en Java.

El código del programa incluye un extenso número de métodos, variables y estructuras. El programa tiene 346 líneas de código y requiere un alto nivel de formación para comprender su funcionamiento y comportamiento, además de una gran experiencia para poder desarrollarlo (Falstad, 2015). La Figura 2 muestra parte del listado del código del programa del juego Pong escrito en Java.

```
327     public void stop() {
328     if (engine != null && engine.isAlive()) {
329         engine.stop();
330     }
331     engine = null;
332     }
333
334     public boolean handleEvent(Event evt) {
335     if (evt.id == Event.MOUSE_MOVE) {
336         paddles[0].setTarget(evt.x);
337         return true;
338     } else if (evt.id == Event.MOUSE_DOWN) {
339         ball.startPlay();
340         return true;
341     } else {
342         return super.handleEvent(evt);
343     }
344     }
345
346 }
```

Figura 2.- Listado parcial del programa juego Pong escrito en Java.

La Figura 3 muestra otro juego de Pong, en este caso implementado con Scratch (LearnScratch 2015a). La bola azul rebota en las paredes y en la paleta rectangular negra que está controlada por el jugador. Un marcador lleva la cuenta del número de veces que el jugador ha golpeado la bola. Finalmente, si la paleta no logra golpear la bola y ésta llega a la línea roja del fondo el juego termina.

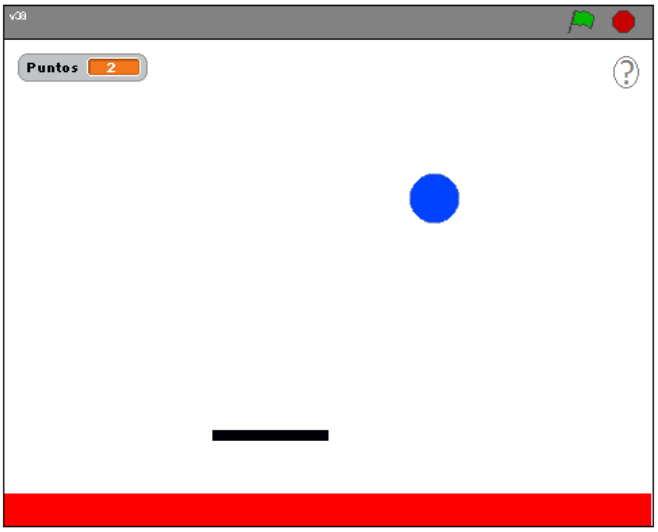

Figura 3.- Diseño del juego Pong realizado con Scratch.

Los dos elementos del Pensamiento Computacional que desarrollamos en la creación del juego Pong para resolver tareas complejas grandes son los siguientes:

- Abstracción.- significa identificar la esencia del proceso que se desea crear eliminando todos los detalles superfluos. En este sentido reduciremos los elementos del juego Pong para incluir sólo aquellos componentes y características que son esenciales para el juego.
- Descomposición.- es el proceso por el cual dividimos el problema en partes, y cada una de ellas en sus correspondientes componentes básicas para transformar un problema grande y complejo en un conjunto pequeño de tareas sencillas e interdependientes.

Movimiento de la Bola

La bola tiene diferentes tipos de comportamiento en el juego, incluyendo su movimiento constante, la interacción con la paleta, la interacción con el fondo rojo, etc. El objetivo de esta parte es proveer a la bola la característica de movimiento continuo en el espacio donde se produce el juego. Un sencillo bloque 'mover pasos' (Figura 4) proporciona la base para este comportamiento. Cada vez que se ejecuta este bloque, por ejemplo haciendo doble clic sobre él, la bola se mueve 4 pasos (en un escenario de 480x360 pixeles) en el sentido en curso del *sprite* u objeto.

Figura 4.- Bloque 'mover pasos'.

Para lograr el movimiento continuo el bloque mover se repite continuamente introduciéndolo como parte del cuerpo del bloque de repetición 'por siempre' (Figura 5).

Figura 5.- Movimiento continuo con el bloque 'por siempre'.

Se supone que si la bola se mueve continuamente en un sentido hacia adelante en línea recta en un escenario de dimensiones finitas, pronto alcanzará una de las paredes/bordes del escenario. Scratch ha sido diseñado de forma que las coordenadas de un sprite siempre rebotan en los límites del escenario, y por consiguiente cuando la bola alcanza un borde permanecerá en dicha posición y la funcionalidad del bloque 'mover' queda anulada. Para este tipo de eventos, que ocurren con cierta frecuencia debido a las dimensiones finitas del escenario de Scratch, el menú Movimiento incluye el bloque 'rebotar si toca un borde' (Figura 6).

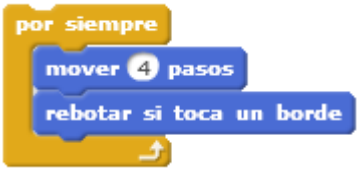

Figura 6.- Bloque 'rebotar si toca un borde'.

Los tres bloques de la Figura 6 proveen a la bola un movimiento continuo en el escenario, rebotando cuando alcanza cualquiera de los bordes del escenario.

Finalmente, podemos añadir un bloque gestor de eventos que activará el *script* o el programa cuando el evento deseado se produce (Figura 7).

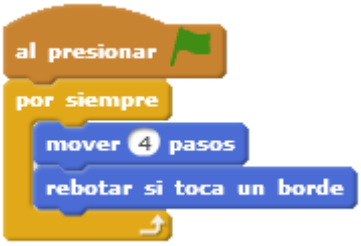

Figura 7.- El bloque 'al presionar bandera verde' inicia el movimiento continuo.

En este caso el gestor de eventos seleccionado es ‘al presionar la bandera verde’. Esta aparente sencilla estructura provee gran flexibilidad en la creación de programas con múltiples procesos concurrentes.

La estructura de los eventos y del gestor de eventos en Scratch incluye tres componentes principales: el evento, el gestor de eventos y el *script* (programa) asociado a cada evento. En el ejemplo anterior, el evento es el hecho físico de hacer clic con el ratón en la bandera verde que está en el vértice superior derecha del entorno Scratch.

El gestor de eventos en Scratch lo constituyen un serie de bloques fáciles de reconocer por ser bloques que tienen redondeada su parte superior. En este caso el gestor de eventos está constantemente monitorizando todas las actividades y en especial está esperando que la bandera verde sea presionada por el ratón. Cuando esto ocurra se activará el *script* asociado con este evento, y para este ejemplo el bloque ‘por siempre’ se encarga de mantener la bola en movimiento permanente.

La Paleta

En este punto el proyecto contiene sólo un *sprite*, la bola, con sólo un *script*, que se encarga del movimiento continuo de la bola.

El siguiente paso es crear un nuevo *sprite* que representará la paleta. La representación gráfica, el disfraz, de la paleta es un simple rectángulo.

La Figura 8 muestra el escenario con la bola y el nuevo *sprite* creado para la paleta.

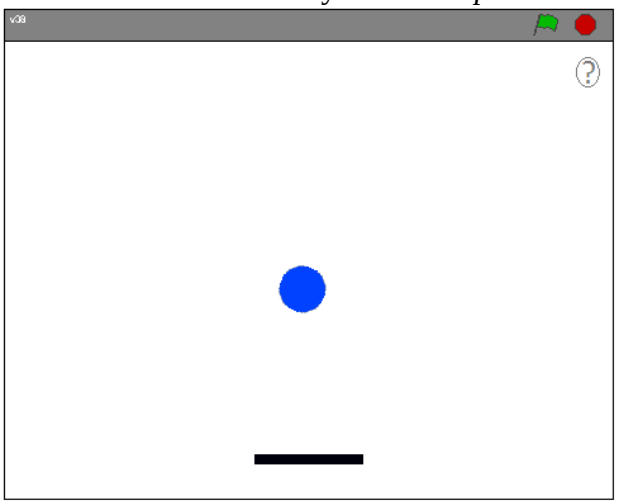

Figura 8.- Escenario con dos *sprites*: la pelota y la paleta.

El funcionamiento de la paleta la controlará el ratón del ordenador. El jugador podrá mover la paleta moviendo el ratón. En concreto la coordenada x de la paleta coincidirá con la coordenada x del ratón.
Scratch ofrece la capacidad de fijar la coordenada x de los *sprites* mediante el bloque que se muestra en la Figura 9.

Figura 9.- Bloque ‘fijar x a ‘.

El funcionamiento de la paleta consiste en seguir permanentemente la posición horizontal del ratón, y por tanto el valor para fijar la coordenada x de la paleta no será un valor particular, como el valor 0 que aparece en la figura, sino que será el valor concreto de la coordenada x del ratón.

Scratch presenta varios bloques denominados reporteros que no implementan ninguna acción concreta como es mover unos pasos o fijar una coordenada x, sino que ofrecen información. En otros lenguajes de programación, los métodos o funciones que implementan acciones pero que no devuelven ninguna información son declarados tipo *void*. El reportero de Scratch que ofrece la coordenada x actual del ratón está representado en la Figura 10.

Figura 10.- Reportero 'posición x del ratón'.

Los bloques que representan acciones y los bloques que son reporteros se diferencian gráficamente. Los bloques que representan acciones como 'mover pasos' o 'fijar x' tienen un corte en su parte superior y una pestaña en la parte inferior que indican su capacidad de apilarse para crear una secuencia de acciones. Los reporteros como 'posición x del ratón' tienen formas redondeadas y no pueden apilarse. Si pueden insertarse en los bloques que tengan un su interior un espacio en blanco como el valor 0 (por defecto) del bloque 'fijar x a'.
La Figura 11 muestra cómo el reportero 'posición x del ratón' ha sido colocado como argumento de la acción del bloque 'fijar x a'. El efecto de esta combinación es que cuando el bloque es ejecutado la coordenada x de la paleta se fijará al valor actual de la coordenada x del ratón, y por tanto parecerá que la paleta está controlada por la localización del ratón.

Figura 11.- Reportero 'posición x del ratón' como argumento del bloque 'fijar x a'.

Para lograr que la paleta siga continuamente la posición horizontal del ratón el bloque debe ser repetido continuamente. Este proceso es muy parecido al movimiento continuo de la bola. La Figura 12 muestra el *script* que permite que la paleta sea controlada por el ratón.

Figura 12.- Posición de la paleta controlada por la 'posición x del ratón'.

Se puede añadir un gestor de eventos que activará este *script* cuando un evento particular se produzca. La Figura 13 muestra el *script* completo que controla la paleta donde el evento hacer 'clic sobre la bandera verde' de Scratch hará que siga la coordenada x del ratón.

Figura 13.- *Script* completo del movimiento de la paleta.

En este punto del proyecto tenemos ya un ejemplo de procesamiento en paralelo donde dos *scripts*, uno que controla el movimiento de la bola y otro que controla el movimiento de la paleta, y ambos se ejecutan a la vez.

También como los dos *scripts* son activados por el mismo gestor de eventos, cuando se hace clic en la bandera verde, podemos ver un caso de sincronismo de tareas correspondientes a diferentes *sprites*.

El funcionamiento del proyecto hasta este momento es que los dos *sprites*, la bola y la paleta están sin interacción entre ellos. Si los trayectos de los dos *sprites* se cruzan en cualquier momento no hay ningún efecto en ninguno de ellos. El efecto de que la bola rebote en la paleta cuando ambos estén en contacto no está todavía desarrollado en el proyecto. Esto es parte del proceso de abstracción y descomposición.

Hemos empezado reduciendo al máximo el comportamiento de los elementos del proyecto e incrementalmente añadimos complejidad incorporando comportamientos y relaciones entre los objetos.

El rebote

El proceso de descomposición nos permite separar elementos de la abstracción del proyecto en *scripts* separados. En este sentido modelamos el movimiento constante de la bola con el *script* estudiado anteriormente.

Podemos añadir ahora como un *script* separado el comportamiento del rebote cuando la bola golpea la paleta. La combinación de dos *scripts* será la que proveerá de un comportamiento más complejo: movimiento constante más el rebote si hay contacto con la paleta.

El efecto en la bola cuando hace contacto con la paleta es que la pelota cambia su dirección y continúa su movimiento. Scratch ofrece un bloque para cambiar la dirección de un *sprite* (ver Figura 14).

Figura 14.- Bloque 'apuntar en dirección'.

La simulación realística del rebote se logra cambiando la dirección por la dirección reflejada a la dirección antes del contacto. Esta nueva dirección se implementa restando 180 grados a la actual dirección del *sprite* (Figura 15).

Figura 15.- Operador resta: 180 menos el reportero de la dirección del *sprite*.

Este es otro ejemplo de la utilización de reporteros en Scratch. Los dos bloques, el azul y el verde tienen forma redondeada indicando su funcionalidad como reporteros y también cómo se interconectan entre ellos; no se apilan como los bloques de acción y se colocan en los huecos de los bloques donde sustituirán los valores previos por defecto.

La Figura 15 muestra el reportero azul de dirección que provee la actual dirección del *sprite*, en este caso la dirección de la pelota. Este reportero se colocará en uno de los dos espacios del reportero resta, que informa sobre la diferencia de los valores colocados en los espacios. En este caso el reportero verde provee un valor que es la diferencia entre 180 grados y el valor actual de la dirección. Como hemos indicado anteriormente este valor representa la nueva dirección de un objeto después de haber sido reflejado en la colisión con otro objeto.

Ahora podemos utilizar el bloque de acción 'apuntar en dirección' introducida anteriormente para actualizar la dirección de la pelota después del impacto. La Figura 16 muestra el anidamiento de los bloques de acción y reporteros.

Figura 16.- Anidamiento de acciones y reporteros.

Como en el caso del movimiento continuo de la pelota, o el seguimiento continuo de la paleta por el ratón, nos interesaría monitorizar constantemente los posibles contactos entre pelota y paleta. Diferente a los otros dos casos, el efecto del rebote no ocurrirá a la vez, sino que sólo cuando la pelota y la paleta choquen. Scratch ofrece para este tipo de eventos una estructura controlada que cuenta estos tipos de condiciones, cuando algo es verdadero o falso, cuando dos *sprites* chocan o no. La Figura 17 muestra el bloque 'si entonces'.

Figura 17.- Bloque 'si entonces'.

La diferencia significativa es que el cuerpo del bloque, el *script* anidado dentro de su cuerpo, se ejecutará sólo si el reportero colocado en su espacio es verdadero. La forma del espacio es parecida pero diferente de los reporteros anteriores que tenían bordes redondeados. En este caso, la forma es hexagonal o con bordes rectos. En Scratch estas formas significan que el reportero informará sobre una cantidad booleana, un valor que sólo puede tener dos posibles estados, verdadero o falso.

Esto es diferente por ejemplo del reportero 'posición x del ratón' que informa un valor entre -240 y 240. Formalmente un reportero que informa sobre una cantidad booleana es denominado predicado.

Scratch provee varios predicados, o reporteros booleanos que proveen información útil. En nuestro caso estamos interesados en determinar cuándo la pelota y la paleta chocan. La Figura 18 muestra el predicado que nos provee esta información.

Figura 18.- Bloque '¿tocando?'

La Figura 19 muestra la implementación del efecto del rebote cuando la pelota y la paleta chocan con la combinación de los tres bloques citados. El bloque 'si entonces' monitoriza el valor del predicado 'tocando Paleta', y cuando éste valor es verdadero se realizarán las acciones del cuerpo del bloque cambiando la dirección de la pelota y por tanto realizando el efecto del rebote. Para que esta acción esté continuamente operativa, se añade el bloque 'por siempre' como se muestra en la Figura 19.

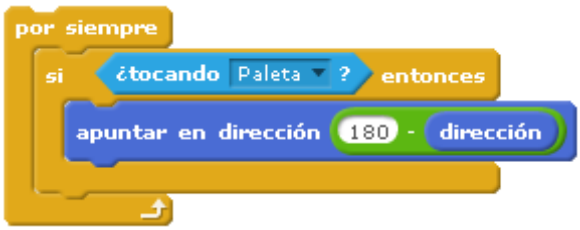

Figura 19.- *Script* del efecto rebote de la pelota al tocar la paleta.

Activamos este *script* con el gestor de eventos al 'presionar bandera verde' (Figura 20) extendiendo la sincronización de procesos descritos anteriormente. En este momento el proyecto incluye dos *sprites* y tres *scripts*, implementando el movimiento continuo de la pelota, el seguimiento continuo del ratón, y el efecto del rebote de la pelota.

Figura 20.- Inicio del efecto rebotar al presionar bandera verde.

En un proceso de refinamiento incremental podemos modelar el efecto del rebote con formas más sofisticadas. En este punto en la mente del programador está claro qué *script* implementa la acción del rebote. Las acciones del rebote están en el cuerpo del bloque 'si entonces' a diferencia de los otros bloques que forman parte del *script* del rebote (Figura 20).

El bloque por siempre y al presionar bandera verde son sólo el gestor de eventos y la estructura controlada que regulan cuando el efecto del rebote se producirá. Por esta razón, si queremos hacer un efecto de rebote más sofisticado, por ejemplo añadir un sonido cuando el choque ocurra, bastaría con añadir un bloque de sonido al cuerpo del *script*.

La Figura 21 muestra un *script* modificado donde el efecto del rebote ahora se realiza con dos acciones; cuando la pelota choca con la paleta hay un sonido Pop, y además la pelota cambia de dirección. Este es otro ejemplo del proceso de abstracción en el cual inicialmente sólo se ha implementado el elemento esencial del rebote, el cambio de dirección, y más tarde se ha añadido un efecto secundario, el sonido Pop. La estrategia

de un continuo refinamiento del proyecto se puede realizar una vez la estructura fundamental ha sido definida y construida.

Figura 21.- Efecto secundario del rebote: tocar sonido Pop.

Fin del Juego

Un último comportamiento en la implementación del juego tiene por objetivo detener todos los procesos cuando la paleta falla a la hora de golpear la pelota y ésta en su trayectoria toca el fondo rojo. Las reglas del juego considera esta situación como el final del juego. Scratch provee una estructura controlada para finalizar todos los procesos, todos los *scripts* de todos los *sprites* con un único bloque. La Figura 22 muestra el bloque 'detener todos'.

Figura 22.- Bloque 'detener todos'.

Hay que señalar que este bloque tiene un corte pero no tiene una pestaña, indicando que ningún bloque puede ser adjuntado después de él por definición, deteniendo y haciendo irrelevantes los subsecuentes bloques.

Este bloque sólo se ejecutará una vez, al final del juego, y este final del juego puede producirse en cualquier momento, por tanto el *script* debe examinar continuamente si la condición de final del juego se ha producido. Scratch provee un bloque de control para implementar este tipo de procesos donde se está esperando que un evento se produzca. La Figura 23 muestra el bloque 'esperar hasta que'.

Figura 23.- Bloque 'esperar hasta que'.

Este bloque detiene la ejecución del *script* hasta que el predicado en su espacio se convierte en verdadero. En ese caso el *script* reanuda la ejecución y los bloques apilados después suyo son ejecutados

La condición para la finalización del juego es cuando la pelota toca el fondo rojo. Scratch provee un predicado para determinar si un *sprite* está en contacto con un color particular. La Figura 24 muestra el bloque 'tocando el color'.

Figura 24.- Bloque ¿tocando el color?

La versión definitiva del *script* que implementa el final del juego se muestra en la Figura 25 donde se combina los bloques anteriores, y el gestor de eventos utilizado en *scripts* anteriores.

Figura 25.- *Script* de final de juego.

Marcador

En este punto el proyecto está completamente operacional, incluye dos *sprites* y cuatro *scripts* pequeños que implementan para la pelota su movimiento continuo, el efecto del rebote y el final del juego, y para la paleta su seguimiento continuo del ratón. La metodología de diseño ha consistido en identificar los comportamientos fundamentales del proyecto y sus correspondientes implementaciones mediante la construcción y testeo de sencillos *scripts*.

Se puede incluir elementos adicionales al proyecto para mejorar sus características o añadir nuevas funcionalidades. Por ejemplo se puede añadir un marcador que registre el número de veces que el jugador ha golpeado la pelota antes de que finalice el juego. Esto requiere la creación de una variable que podemos llamarle Puntos. Scratch provee un bloque para manipular variables. La Figura 26 muestra dos bloques que nos permite inicializar el número de puntos a cero e incrementar el número de puntos.

Figura 26.- Bloques de la variable Puntos.

El número de puntos debe incrementarse de uno en uno cada vez que la paleta hace contacto con la pelota. Anteriormente se ha estudiado con detalle el *script* que implementa el rebote, y el cambio de dirección de la pelota junto con el sonido Pop. La Figura 27 muestra el *script* actualizado, donde además de las dos acciones iniciales se incrementa de uno en uno el número de puntos que representa el número de veces que la paleta ha chocado con la pelota.

Figura 27.- *Script* del efecto Rebote que incluye incrementar los puntos del Marcador.

Una vez que el juego ha finalizado uno puede volver a empezar un nuevo juego presionando la bandera verde, que activa los *scripts* del proyecto que fueron detenidos por el bloque 'detener todos'. Sería aconsejable que cada vez que un nuevo juego comienza el marcador reflejara dicha situación poniendo el marcador a 0 puntos como se muestra en la Figura 28.

Figura 28.- Inicialización del Marcador.

Este último *script* puede considerarse en cualquier lugar donde se inicializan valores del juego. Este tipo de *scripts* de inicialización son comunes en muchos proyectos y son activados sólo al comienzo de la ejecución del juego. Se puede actualizar este *script* de inicialización incluyendo otras tareas como las que se muestran en la Figura 29.

Figura 29.- Inicialización de diferentes tareas.

En esta versión del *script* de inicialización el número de puntos es ajustado a cero, la posición inicial de la pelota es cualquier lugar de la parte superior del escenario, y la dirección inicial de la pelota es elegida de una forma aleatoria.

Reformulación de un problema a un problema similar del que ya se conoce su solución.

Utilizaremos la creación del juego Pack-Man como proyecto para ilustrar otro elemento esencial del Pensamiento Computacional: Reformulación de un problema a un problema similar del que ya se conoce su solución.

La Figura 30 muestra el diseño del juego Pack-Man escrito en Java (Postma, 2015).

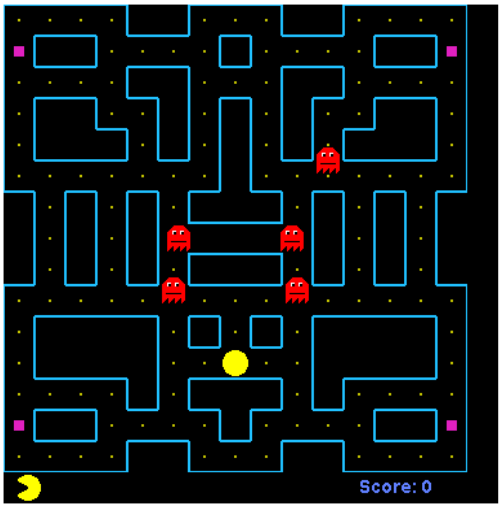

Figura 30.- Diseño del juego Pack-Man escrito en Java.

El código, como se ha visto anteriormente para el juego Pong, incluye un número extenso de métodos, variables y estructuras. Tiene 800 líneas de código y constituye un proyecto complejo. La Figura 3 muestra parcialmente el código Pack-Man escrito en Java.

```
807  public void run()
808  {
809    long  starttime;
810    Graphics g;
811
812    Thread.currentThread().setPriority(Thread.MAX_PRIORITY);
813    g=getGraphics();
814
815    while(true)
816    {
817      starttime=System.currentTimeMillis();
818      try
819      {
820        paint(g);
821        starttime += 40;
822        Thread.sleep(Math.max(0, starttime-System.currentTimeMillis()));
823      }
824      catch (InterruptedException e)
825      {
826        break;
827      }
828    }
829  }
```

Figura 31.- Listado parcial del código de Pack-Man en Java.

La Figura 32 muestra un proyecto similar implementado con Scratch (LearnScratch, 2015b). Este proyecto presenta una versión simplificada del juego en varios aspectos, pero incluye nuevas características avanzadas como el cambio de nivel de dificultad cada vez que se alcanza el objetivo del juego que es tocar el cuadrado de color rojo.

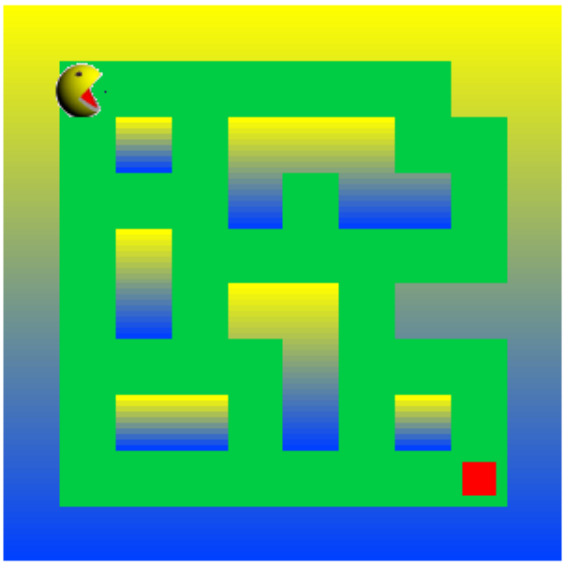

Figura 32.- Diseño de Pack-Man en Scratch.

La creación de este proyecto Scratch para construir el juego Pack-Man nos permite ilustrar otro de los elementos esenciales del Pensamiento Computacional, la Reformulación de un problema a un problema similar del que ya se conoce su solución. En el desarrollo de proyectos y en la creación de soluciones para resolver problemas específicos, encontramos con frecuencia la presencia de retos que tienen características, propiedades, formas, comportamientos, etc. de problemas que hemos resuelto con anterioridad. No siempre la similitud de los problemas es significante, pero la experiencia de proyectos anteriores ofrece un gran valor en la resolución del proyecto en curso. Una de las definiciones de la Inteligencia la describe como la habilidad de utilizar experiencias pasadas para resolver problemas.

A la hora de observar y analizar un nuevo problema, debemos tratar de aislar las partes significativas del funcionamiento e intentar ajustar sus características con aquellas que hemos analizado anteriormente en otros problemas. De esta manera podemos comenzar abordar la solución del nuevo proyecto desde un punto de vista donde de la experiencia anterior puede ser de gran ayuda.

Esta actitud tiene implicaciones importantes en el proceso de aprendizaje y en el proceso de resolver nuevos problemas. Las técnicas aprendidas en el proceso de resolver problemas relacionados no sólo nos ayudan a resolver problemas nuevos sino que además el proceso de aprendizaje adquiere mayor valor añadido.

Además, al actualizar una solución de un problema anterior a las características de un problema nuevo y diferente, desarrollamos incrementalmente el valor de la solución anterior, poniéndola disponible para resolver un conjunto amplio de problemas. Nuestro conocimiento se incrementa con su utilización y la experiencia amplía su propio alcance.

Un juego básico de Pack-Man

La Figura 32 muestra el diseño general de una versión básica del juego Pack-Man. En la implementación del juego no hay enemigos del Pack-Man y el objetivo es llegar a través del laberinto al cuadrado rojo lo más rápido posible. Se puede utilizar un reloj

para medir la habilidad de cada jugador registrando el tiempo que requiere para alcanzar el objetivo. Una vez que se alcanza el objetivo, el diseño del laberinto cambia, y empieza un nuevo escenario del juego. Por simplicidad sólo se ha incluido dos diseños para el laberinto. Después de alcanzar el segundo objetivo el proyecto vuelve al diseño inicial.

Música de fondo

La música de fondo, repetitiva y continua es quizá la característica más recordada por los jugadores del juego original de Pack-Man. El efecto de la música es irrelevante para la ejecución del propio juego; sin embargo la estética y otros elementos secundarios en el proyecto desempeñan un papel importante, y su ausencia en la implementación del juego puede afectar a la calidad del resultado final.

La implementación de la música de fondo es uno de los primeros problemas que se abordan cuando se introduce Scratch; por ejemplo la creación de relatos sencillos normalmente incluyen una animación sencilla y una música como sonido de fondo.

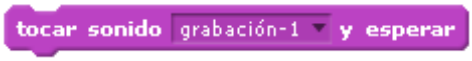

Figura 33.- Bloque 'tocar sonido y esperar'.

La Figura 33 muestra el bloque 'tocar sonido y esperar' que es un sencillo *script* que reproducirá una música grabada hasta que finalice.

La música de fondo no está necesariamente asociada a ningún *sprite* en particular; por eso su localización puede ser independiente de los *sprites* y podría ubicarse como parte del Escenario. Esto es similar a que en un ballet la música de la orquesta no está en un bailarín particular sino que está con todos, pudiéndose poner la música en cualquier lugar de forma separada.

En un sencillo relato con música de fondo, la narración del relato o la canción se reproduce una única vez. Sin embargo, en un juego no es posible predecir con exactitud la duración del mismo por parte del jugador.

Una sencilla solución es repetir la música una y otra vez mientras el juego continúa. En el caso particular del juego Pack-Man, el fragmento de música grabada tiene una duración de sólo cuatro segundos mientras el juego puede durar minutos o más tiempo.

Figura 34.- *Script* para la música de fondo del Pack-Man.

La Figura 34 muestra el *script* definitivo que implementa la música de fondo. El fragmento corto de música es repetido continuamente durante el juego. Además el

script es iniciado por el gestor de eventos 'al presionar bandera verde' que sincronizará todos los demás *scripts* del juego.

Animación Pack-Man

Otra característica que los jugadores recuerdan del juego original es la animación de Pack-Man que simula la acción de comer unos puntos pequeños según se desplaza por el laberinto. Este efecto provee a Pack-Man la característica de un ser vivo.

Cuando se introduce Scratch, los estudiantes trabajan sin animación. El cambio periódico del disfraz de un *sprite*, si se hace bien, es lo que consigue dar una sensación real de animación.

Un ejemplo típico de este proyecto es la animación de un animal corriendo.

Figura 35.- Disfraces de un guepardo en movimiento.

La Figura 35 muestra los cuatro primeros disfraces de un guepardo en el proceso de correr. El *script* básico que se muestra en la Figura 36 cambia el disfraz del *sprite* cada décima de segundo. Esto produce una imagen realística del guepardo corriendo. El tiempo del bloque 'esperar segundos' puede ser modificado para cambiar el efecto visual, desde un movimiento lento a una carrera rápida.

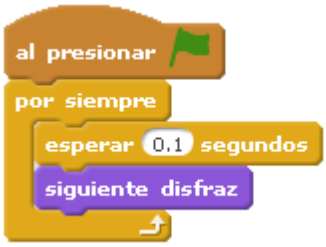

Figura 36.- *Script* para realizar el cambio de disfraz cada 0.1 segundos.

En el caso del juego Pack-Man el efecto de animación es importante pero no requiere el nivel de detalle y precisión que la carrera del guepardo. El *script* utilizado será el mismo, ya que la animación puede abstraerse como el cambio periódico del disfraz, pero el número de disfraces es mucho más reducido. La Figura 37 muestra los dos disfraces utilizados para producir el efecto de animación del Pack-Man.

Figura 37.- Disfraces de Pack-Man.

Movimiento de Pack-Man a través del laberinto

En el juego original de Pack-Man como se muestra en la Figura 30 el laberinto ha sido definido como un conjunto de paredes, en este caso de color azul claro. El Pack-Man se mueve automáticamente a través de los pasillos del laberinto si está orientado en la dirección correcta alineado con el camino. El jugador controla la posición del Pack-Man dirigiéndolo con cuatro teclas de dirección. Se puede pensar en Pack-Man como un robot que se mueve constantemente sin interrupción donde el jugador sólo controla la dirección en la que se desplaza Pack-Man.

Un movimiento continuo es una tarea relativamente fácil de implementar con Scratch. Para permitir que Pack-Man sólo se mueva en los caminos del laberinto y no atraviese las paredes es necesario realizar otros análisis.

En un juego introductorio implementado en Scratch una pelota rebotaba en las paredes del escenario moviéndose de forma libre. En el centro del escenario había un pequeño círculo rojo. Cada vez que la pelota pasaba sobre el objetivo rojo sonaba el sonido Pop. La Figura 38 muestra el *script* que implementa esta acción.

Figura 38.- *Script* que toca el sonido Pop cuando se toca el color rojo.

Aunque esta operación parece diferente del movimiento del Pack-Man a través del laberinto verde, existen varias similitudes importantes. En los dos casos se desea implementar una acción (mover o reproducir un sonido Pop) cuando una condición es cierta (la pelota está sobre el objetivo rojo, o el Pack-Man está dentro del laberinto verde). Por consiguiente una estructura similar implementará las dos acciones.

La Figura 39 muestra el *script* que implementa el movimiento del Pack-Man a través del laberinto. La estructura es similar a la reproducción del sonido de la pelota en movimiento.

Figura 39.- Movimiento de Pack-Man dentro del laberinto.

La acción a implementar, moverse en el laberinto, la realiza el bloque 'mover 2 pasos'. La condición de saber si el Pack-Man está en el laberinto o ha alcanzado una pared necesita un análisis más pormenorizado de sus disfraces.

La Figura 40 muestra que los disfraces de Pack-Man incluyen un puntito negro delante de su boca. Según Pack-Man se mueve en la zona verde, el color negro del disfraz toca el color verde. Cuando se acaba el camino, el punto negro toca la pared amarilla y la condición de que el punto negro toque el color verde no se cumple, y por tanto se detiene el *script*, y el Pack-Man no avanza más.

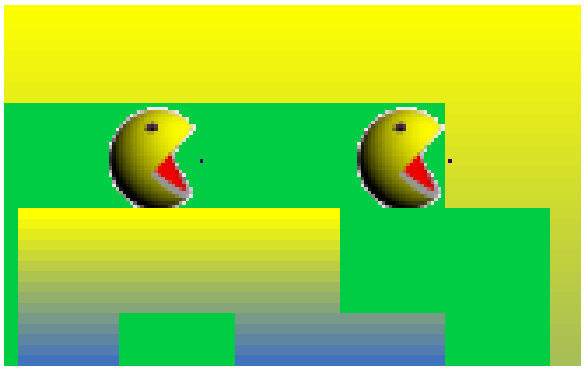

Figura 40.- Puntito negro de Pack-Man tocando color verde o amarillo.

La Figura 41 muestra el predicado que controla el *script* del movimiento del Pack-Man, en concreto la condición de saber si el Pack-Man está en el laberinto o ha alcanzado una pared.

Figura 41.- Predicado que indica si Pack-Man toca o no toca color verde.

Está claro que la estructura de un problema resuelto anteriormente es útil para resolver el nuevo problema pero se hace necesario desarrollar una estrategia concreta: el solape de los colores negro y verde, para gestionar los requerimientos del nuevo problema. A la vez que se desarrolla una nueva estrategia, se crea una nueva experiencia y el valor de los problemas resueltos anteriormente se hace cada vez más completo.

La destreza de reconocer similitudes en dos problemas diferentes como en este caso se basa en nuestra capacidad de representar con el lenguaje Scratch la esencia de los dos problemas, haciendo visible la relación entre ambos.

Guiado de Pack-Man

El juego original permite al jugador controlar el Pack-Man mediante el control de la dirección a la que se orienta. Como se observa en la Figura 30, la distribución en el laberinto está orientada de norte a sur o de este a oeste. Por tanto el control de Pack-Man permite orientarlo hacia arriba, abajo, izquierda y derecha.

En un juego en que se ha simulado el movimiento de un tren, el jugador puede moverlo a la izquierda o derecha en pequeños pasitos haciendo presión en las teclas flecha izquierda y flecha derecha respectivamente.

La Figura 42 muestra los dos *scripts* que implementan el control de un sencillo tren.

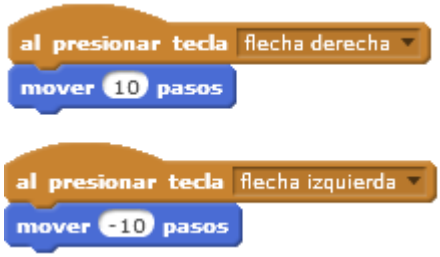

Figura 42.- Control del movimiento con las teclas flechas derecha e izquierda.

La relación con el control de Pack-Man es evidente sólo que en ese caso del tren tenemos dos controles, mover a la derecha y mover a la izquierda, asociados a dos acciones, mover 10 pasos o mover -10 pasos (en Scratch –10 pasos se interpreta como mover 10 pasos en sentido contrario). Similarmente en el caso de Pack-Man hay un conjunto de controles, cuatro controles, asociados con las cuatro acciones. La Figura 43 muestra la implementación del control de Pack-Man.

Figura 43.- Controles para el movimiento de Pack-Man.

Es conveniente destacar que aunque hay cuatro *scripts* separados, desde el punto de vista del estudiante que implementa el proyecto, deben ser interpretados como una única abstracción: el control de la dirección de Pack-Man, que en este caso tiene cuatro posibles opciones. Hay estrategias que implementan este control en un único *script* con estructuras tipo ‘si’ anidadas, pero en este caso presentamos esta implementación por el bien de la simplicidad.

Cambio de Nivel

La única funcionalidad del juego que falta por implementar es cambiar el laberinto una vez que Pack-Man alcanza el objetivo rojo.

El Pack-Man atraviesa los pasillos del laberinto con el control que realiza el jugador. En el caso de que Pack-Man toque una pared dejará de andar hasta que el jugador cambie su dirección y permita la continuación del paseo. La acción deseada cuando eventualmente Pack-Man alcance el cuadrado rojo es cambiar el laberinto para empezar un nuevo nivel. Esta funcionalidad es muy similar a aquella en la que la pelota hace sonar un sonido cuando pasa por encima del objetivo rojo, o que Pack-Man se mueva hacia adelante cuando su punto negro toca el color verde. El efecto deseado es realizar una acción cuando se produce una condición.

El laberinto es implementado como un *sprite* que contiene un disfraz por cada nivel del juego. La Figura 44 muestra dos niveles del juego implementado en este ejemplo.

Figura 44.- Disfraces del laberinto.

Los diferentes diseños del laberinto tienen las siguientes características: los pasillos son de color verde; las paredes son de cualquier color menos verde; y el objetivo es un cuadrado rojo ubicado en cualquier sitio del laberinto. Siguiendo estas instrucciones se podría añadir varios disfraces al *sprite*, y por tanto añadiendo varios niveles al juego.

El *script* para cambiar los disfraces cuando Pack-Man encuentra el objetivo se muestra en la Figura 45.

Figura 45.- *Script* para el cambio de nivel del juego.

Reconocemos la estructura del *sprite* y por tanto la similitud en la esencia de las tareas implementadas. La acción a implementar es cambiar el disfraz con el bloque 'siguiente disfraz', y la condición es que el color rojo del *sprite*, el objetivo cuadrado rojo, toque el punto negro del *sprite* de Pack-Man.

CURSO PENSAMIENTO COMPUTACIONAL EN LA ESCUELA

Existen diversas iniciativas formativas sobre Pensamiento Computacional en formato MOOC (*Massive Open Online Courses*). Entre ellas destacan los cursos 'Pensamiento Computacional en la Escuela' en Miríada X (Basogain, Olabe y Olabe, 2015), 'Code Yourself! An Introduction to Programming' en Coursera (Coursera, 2015), 'CS002x Programming in Scratch' en EdX (edX, 2015), y 'Creative Computing online workshop' en Harvard Graduate School of Education (Harvard, 2015). Son cursos que tratan la temática de la programación y la computación tanto para jóvenes estudiantes como para docentes y público en general. Estos cursos tienen planteamientos afines y complementarios, todos ellos presentan objetivos orientados a la enseñanza de la Programación a través de Scratch.

Objetivo del curso

El curso 'Pensamiento Computacional en la Escuela' tiene como objetivo doble: 1) establecer una definición sobre Pensamiento Computacional, y 2) aplicar dicha definición en la resolución de problemas en la escuela y en la vida cotidiana. En el curso se establece la siguiente definición: "Pensamiento Computacional es el proceso de reconocimiento de aspectos de computación en el mundo que nos rodea, y la aplicación de herramientas y técnicas de computación para entender y razonar acerca tanto de los sistemas y procesos naturales como de los artificiales".

Esta definición presenta una visión holística del conjunto de las propiedades del Pensamiento Computacional. En ella se incluyen las características que los principales impulsores del Pensamiento Computacional como Seymour Papert, Jeannette Wing, y MIT Media Lab proponen para este nuevo paradigma de pensamiento (Papert, 1980), (Disessa, 2000), (Wing, 2006).

La figura 46 muestra la sección del curso en la plataforma Miríada X en la que se describe la definición del Pensamiento Computacional.

Figura 46.- Definición de Pensamiento Computacional.

El segundo objetivo del curso, la aplicación de la definición del Pensamiento Computacional, se lleva a cabo a través de la programación y de diferentes tipos de problemas-proyectos que resuelven en el curso.

La computación se realiza mediante el lenguaje de programación Scratch. Este lenguaje permite leer y escribir programas de ordenador que resuelven problemas. Los programas se construyen mediante técnicas y métodos propios de la computación indicados con anterioridad en la Tabla I.

Contenidos del curso

Las materias del curso han sido seleccionadas y organizadas para lograr el objetivo del curso. El curso se ha estructurado en los siguientes 5 módulos:

Módulo 1.- Pensamiento Computacional en la Escuela + Proyecto Introducción a Scratch
Módulo 2.- Lenguaje y Entorno de Programación + Proyecto Juego Pong – Un jugador
Módulo 3.- Programación: Conceptos y Métodos + Proyecto Juego Pong – Dos jugadores
Módulo 4.- Control Automático y Cibernética + Proyecto Coche sin conductor
Módulo 5.- Ciencias de la Vida + Proyecto Mariposa en el invernadero
El primer módulo describe el paradigma del Pensamiento Computacional y su aplicación en el ámbito escolar y en el día a día. El proyecto Scratch propuesto

establece un primer contacto con el lenguaje y la publicación de proyectos en el portal scratch.mit.edu.

Los módulos 2 y 3 abordan a través de Scratch los principios fundamentales del lenguaje de programación y los métodos de programación (Guttag, 2013).

El módulo 2 describe los tres mecanismos del lenguaje: primitivas (comandos, funciones, estructura de control, evento/disparo), combinación (grupos o pilas de comandos, funciones dentro de comandos) y abstracción (más bloques, definir procedimientos). El módulo 3 describe los conceptos básicos de programación: algoritmos, descomposición de problemas, patrones, abstracción y concurrencia (*multi-threading*).

En estos módulos además se describe el entorno de programación Scratch detallando sus paneles y los modos de edición/ejecución de un programa. Se detallan el escenario, las paletas de bloques, los objetos, los *scripts*, clones, comunicación entre objetos y procedimientos. El entorno Scratch además facilita la sintaxis sin errores, la información de funciones, la visibilidad de los datos, y la mochila (recurso para la reutilización de código y objetos).

Los proyectos Scratch juego de Pong - un jugador, y juego de Pong - dos jugadores tienen los siguientes objetivos: a) introducir Scratch, b) el diseño modular o descomposición, c) la abstracción, d) la interacción entre objetos y e) el uso de variables.

Los módulos 4 y 5 acercan el Pensamiento Computacional a dos áreas presentes en nuestro entorno cotidiano. En estos dos módulos se introducen los conceptos de control automático, cibernética, modelado y simulación de ciencias de la vida. El propósito es conocer mejor los mundos naturales y artificiales que nos rodean.

El módulo 4 introduce la ingeniería de control en el contexto de la teoría general de sistemas y en el ámbito escolar de STEM (*Science, Technology, Engineering, Mathematics*) (Stem, 2015). Se describen los siguientes términos propios de la ingeniería de control: diagrama de bloques, realimentación, comparador, consigna, error, acción de control, controlador, sensor, actuador y perturbación. Estos conceptos facilitan la comprensión de mundos artificiales de nuestro entorno doméstico y laboral que están controlados por sistemas diseñados por el hombre. Este módulo introduce también el concepto de la cibernética: una disciplina asociada al gobierno de sistemas. La cibernética es un área interdisciplinaria de la teoría de sistemas, control, comunicación, información y teoría de la computación.

El módulo 5 describe los conceptos de modelado y simulación de sistemas. Se estudia el modelado y simulación de sistemas continuos y discretos, sistemas sencillos y complejos, y sistemas naturales y artificiales.

Scratch facilita la implementación del control automático y la cibernética a través de sus bloques de decisión y comunicación. Scratch también facilita el modelado y simulación; Scratch como un lenguaje sirve para modelar sistemas, y como un entorno de programación sirve para ejecutar y simular modelos de sistemas.

Los proyectos Coche sin conductor y Mariposa en el invernadero son proyectos que introducen al estudiante a los sistemas autorregulados, y a la simulación de sistemas biológicos. Las figuras 47 y 48 muestran el diseño de los proyectos Scratch del Coche sin Conductor y Mariposa en el invernadero respectivamente; ambos proyectos son del curso 'Pensamiento Computacional en la Escuela'. En ambas figuras se utiliza el editor online de Scratch.

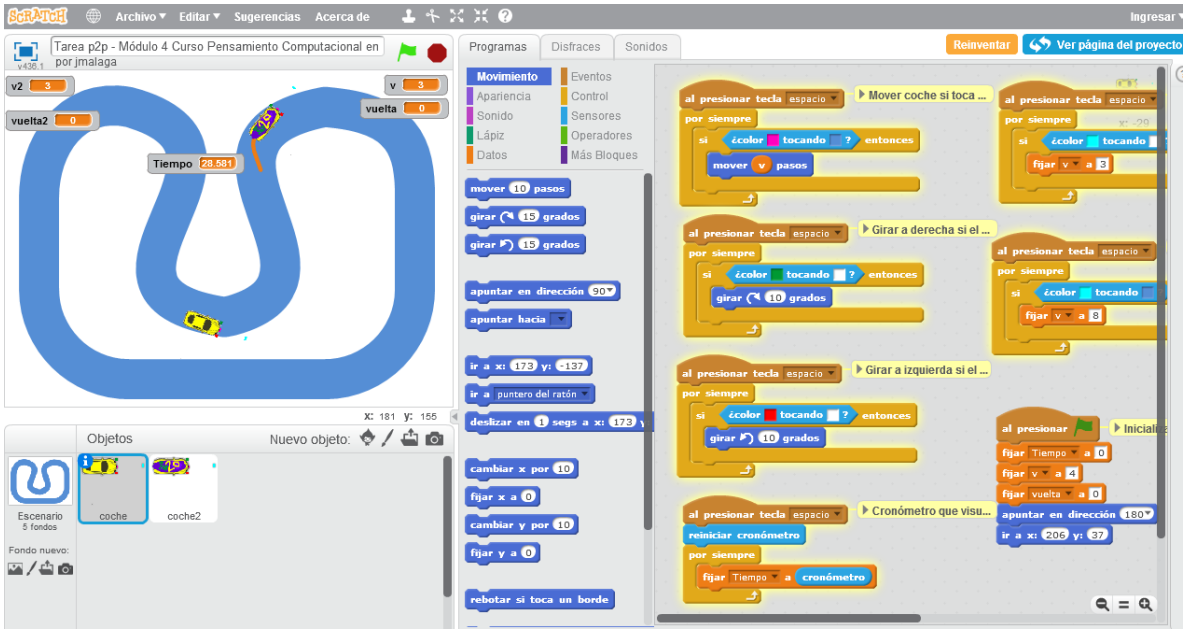

Figura 47.- Proyecto Coche sin conductor del curso Pensamiento Computacional en la Escuela.

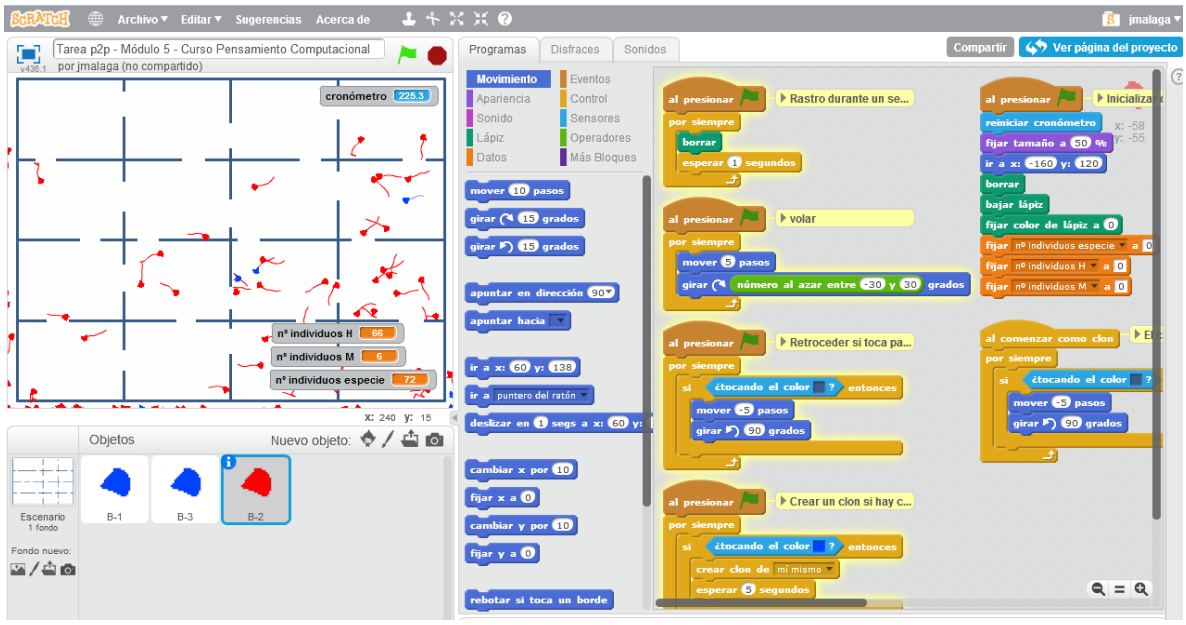

Figura 48.- Proyecto Mariposa en el invernadero del curso Pensamiento Computacional en la Escuela.

Resultados

Al finalizar la primera edición del curso MOOC 'Pensamiento Computacional en la Escuela' el pasado mes de mayo del 2015, los alumnos cumplimentaron la encuesta final del curso. La encuesta recoge diversos datos demográficos del alumnado y algunos datos sobre su opinión del curso.

La encuesta tiene 491 casos válidos. El 58,66% del alumnado es hombre y el 41,34% es mujer, con edades comprendidas entre menos de 18 años (2,85%) y más de 65 años (0.61%), estando la mayoría del alumnado en la franja 36-50 años (46,44%), y el resto distribuido en las franjas 18-25 años (14,87%), 26-35 años (23,42%) y 51-65 años (11,81%).

La Tabla II muestra la relación cruzada entre la actividad profesional del alumnado y su conocimiento sobre Programación antes de iniciar el curso MOOC.

El 57,88% sabía programar antes de iniciar el curso, y el 42,16% no sabía programar. Las dos terceras partes del alumnado del curso realizan su profesión en el ámbito de la docencia, mientras que el otro tercio (30,14%) su profesión no tiene que ver con la docencia.

Profesión	***Conocimiento de Programación***		
	No	Si	Total
Docente Educación Primaria	52.00	27.00	79.00
	65.82%	34.18%	100.00%
	25.12%	9.51%	16.09%
	10.59%	5.50%	16.09%
Docente Educación Secundaria	47.00	84.00	131.00
	35.88%	64.12%	100.00%
	22.71%	29.58%	26.68%
	9.57%	17.11%	26.68%
Docente Educación Superior	28.00	46.00	74.00
	37.84%	62.16%	100.00%
	13.53%	16.20%	15.07%
	5.70%	9.37%	15.07%
Docente No Reglada	20.00	39.00	59.00
	33.90%	66.10%	100.00%
	9.66%	13.73%	12.02%
	4.07%	7.94%	12.02%
No Docente	60.00	88.00	148.00
	40.54%	59.46%	100.00%
	28.99%	30.99%	30.14%
	12.22%	17.92%	30.14%
Total	207.00	284.00	491.00
	42.16%	57.84%	100.00%
	100.00%	100.00%	100.00%
	42.16%	57.84%	100.00%

Tabla II.- Actividad profesional y conocimiento sobre Programación.

La siguiente Tabla III muestra la relación cruzada entre la actividad profesional del alumnado y su respuesta a la pregunta "¿Propondrías aprender Scratch?" como propuesta de aprender Programación a través de un lenguaje.

Destaca el pequeño porcentaje de alumnos que indican que no propondría aprender Programación (1,43%), y el pequeño porcentaje que propondría aprender Programación sólo a los estudiantes de ingeniería informática (1,22%).

La tercera parte del alumnado (30.55%) propondría aprender Programación a los alumnos de las escuelas.

La mayoría del alumnado (66.80%) propondría aprender Programación a cualquier persona con interés en aprender a resolver nuevos tipos de problemas de una forma diferente a lo convencional.

Profesión	***Proponer aprender Programación***				Total
	Si, a los alumnos de las escuelas	Si, a cualquier persona con interés	Si a los estudiantes de ingeniería informática	No	
Docente Educación Primaria	34.00	43.00	1.00	1.00	79.00
	43.04%	54.43%	1.27%	1.27%	100.00%
	22.67%	13.11%	16.67%	14.29%	16.09%
	6.92%	8.76%	.20%	.20%	16.09%
Docente Educación Secundaria	52.00	76.00	1.00	2.00	131.00
	39.69%	58.02%	.76%	1.53%	100.00%
	34.67%	23.17%	16.67%	28.57%	26.68%
	10.59%	15.48%	.20%	.41%	26.68%
Docente Educación Superior	21.00	50.00	2.00	1.00	74.00
	28.38%	67.57%	2.70%	1.35%	100.00%
	14.00%	15.24%	33.33%	14.29%	15.07%
	4.28%	10.18%	.41%	.20%	15.07%
Docente No Reglada	15.00	44.00	.00	.00	59.00
	25.42%	74.58%	.00%	.00%	100.00%
	10.00%	13.41%	.00%	.00%	12.02%
	3.05%	8.96%	.00%	.00%	12.02%
No Docente	28.00	115.00	2.00	3.00	148.00
	18.92%	77.70%	1.35%	2.03%	100.00%
	18.67%	35.06%	33.33%	42.86%	30.14%
	5.70%	23.42%	.41%	.61%	30.14%
Total	150.00	328.00	6.00	7.00	491.00
	30.55%	66.80%	1.22%	1.43%	100.00%
	100.00%	100.00%	100.00%	100.00%	100.00%
	30.55%	66.80%	1.22%	1.43%	100.00%

Tabla III.- Actividad profesional y propuesta de aprender Programación.

La siguiente Tabla IV muestra la relación cruzada entre la actividad profesional del alumnado y su respuesta a la pregunta "¿Aplicarías el conocimiento aprendido?" en el curso.

El porcentaje de alumnado que no aplicaría el conocimiento aprendido es bajo (3,86%).

El 20,98% aplicaría el conocimiento aprendido en la escuela y el 17,52% lo aplicaría en la vida diaria.

La mayoría del alumnado (58,04%) aplicaría el conocimiento aprendido en el curso MOOC en la escuela y en la vida diaria.

Profesión	***Aplicar el Conocimiento Aprendido***				Total
	Si, en la escuela	Si, en la vida diaria	Si, en la escuela y en la vida diaria	No	
Docente Educación Primaria	27.00	1.00	47.00	4.00	79.00
	34.18%	1.27%	59.49%	5.06%	100.00%
	26.21%	1.16%	16.49%	23.53%	16.09%
	5.50%	.20%	9.57%	.81%	16.09%
Docente Educación Secundaria	46.00	3.00	81.00	1.00	131.00
	35.11%	2.29%	61.83%	.76%	100.00%
	44.66%	3.49%	28.42%	5.88%	26.68%
	9.37%	.61%	16.50%	.20%	26.68%
Docente Educación Superior	13.00	3.00	57.00	1.00	74.00
	17.57%	4.05%	77.03%	1.35%	100.00%
	12.62%	3.49%	20.00%	5.88%	15.07%
	2.65%	.61%	11.61%	.20%	15.07%
Docente No Reglada	5.00	4.00	47.00	3.00	59.00
	8.47%	6.78%	79.66%	5.08%	100.00%
	4.85%	4.65%	16.49%	17.65%	12.02%
	1.02%	.81%	9.57%	.61%	12.02%
No Docente	12.00	75.00	53.00	8.00	148.00
	8.11%	50.68%	35.81%	5.41%	100.00%
	11.65%	87.21%	18.60%	47.06%	30.14%
	2.44%	15.27%	10.79%	1.63%	30.14%
Total	103.00	86.00	285.00	17.00	491.00
	20.98%	17.52%	58.04%	3.46%	100.00%
	100.00%	100.00%	100.00%	100.00%	100.00%
	20.98%	17.52%	58.04%	3.46%	100.00%

Tabla IV.- Actividad profesional y aplicación del conocimiento adquirido en el curso.

CONCLUSIONES

Este artículo presenta el Pensamiento Computacional a través de la Programación como un elemento esencial en la educación de los estudiantes de primaria y secundaria.

Describe los componentes fundamentales y enumera los principios del Pensamiento Computacional.

Se describen dos proyectos de programación que ilustran paso a paso la implementación del Pensamiento Computacional utilizando Scratch como entorno de desarrollo.

Además se incluye una serie de referencias seleccionadas que describen los campos del Pensamiento Computacional y los entornos de programación para su implementación como son Scratch, Alice o Greenfoot.

El Pensamiento Computacional ya está incorporándose como una parte integral de la educación en países como Inglaterra.

Los investigadores están desarrollando entornos de diseño y materiales docentes curriculares para su integración en la educación.

Los maestros y profesores serán formados en este campo del Pensamiento Computacional a través de entornos basados en ordenador, Internet y MOOCs.

Y finalmente, los estudiantes aprenderán los conceptos esenciales y técnicas del Pensamiento Computacional que les prepararán mejor para los retos de las próximas décadas.

Presentación del artículo: 5 de junio de 2015
Fecha de aprobación: 5 de agosto de 2015
Fecha de publicación: 15 de septiembre de 2015

Basogain, X., Olabe, M.A. y Olabe, J.C., (2015). Pensamiento Computacional a través de la Programación: Paradigma de Aprendizaje. *RED. Revista de Educación a Distancia, 46(6).* 30 de Septiembre de 2015. Consultado el (dd/mm/aa) en http://www.um.es/ead/red/46

REFERENCIAS

Alice [Software de Ordenador] (2015). http://www.alice.org/

Basogain, X., et al. (2012, Septiembre 5-7). Mathematics Education through Programming Languages. En Universidad de Oviedo, *21st Annual World Congress on Learning Disabilities Book of Abstracts, Resúmenes*. Artículo presentado en 21st Annual World Congress on Learning Disabilities, Oviedo (553-559). Oviedo: Ediciones dela Universidad de Oviedo. ISBN: 978-84-8317-936-9 .

Basogain, X., Olabe, M.A. y Olabe, J.C. (2015, Junio 11-12). *Curso Pensamiento Computacional en la Escuela: Diseño e Implementación en Miríada X*. Ponencia presentada en las XXIII Jornadas Universitarias de Tecnología Educativa, JUTE 2015. Badajoz, España.

Code.org [Blog] (2013). Anybody can learn. Recuperado 2 de Junio de 2015, de http://code.org

Coursera [Portal Web] (2015). Code Yourself! An Introduction to Programming. Recuperado 2 de Junio de 2015, de https://www.coursera.org/course/codeyourself

CSTA and ISTE (2011). *Computational Thinking Leadership Toolkit,* first edition 2011. Computer Science Teachers Association (CSTA) and International Society for Technology in Education (ISTE). Recuperado 2 de Junio de 2015, de http://www.iste.org/docs/ct-documents/ct-leadershipt-toolkit.pdf?sfvrsn=4

Department for Education England. (2013). *National curriculum in England: computing programmes of study - key stages 1 and 2*. Ref: DFE-00171-2013. Recuperado 2 de Junio de 2015, de http://goo.gl/NW1mHH.

Disessa, A. (2000). *Changing minds: Computers, learning, and literacy*. Cambridge: MIT Press.

edX [Portal Web] (2015). CS002x Programming in Scratch. Harvey Mudd College. Recuperado 2 de Junio de 2015, de https://www.edx.org/course/programming-scratch-harveymuddx-cs002x

Falstad, P. (2015). Java Pong Game. Recuperado 2 de Junio de 2015, de http://www.falstad.com/pong/

Google (2015). Google's Exploring Computational Thinking Initiative. Recuperado 2 de Junio de 2015, de http://www.google.com/edu/computational-thinking/ 4

Greenfoot [Software de Ordenador] (2015). Recuperado 2 de Junio de 2015, de http://www.greenfoot.org/

Guttag, J. (2013) *Introduction to Computation and Programming Using Python*, revised and expanded edition. Cambridge: MIT Press.

Harvard [Portal Web] (2015). Creative Computing online workshop. Harvard Graduate School of Education. Recuperado 2 de Junio de 2015, de https://creative-computing.appspot.com/

Kafai, Y. and Resnick, M., Eds. (1996). *Constructionism in Practice: Designing, Thinking, and Learning in a Digital World*. Lawrence Erlbaum Associates, Inc., Mahwah, NJ.

Kay, A. (2010). Squeak etoys, children, and learning. Recuperado 2 de Junio de 2015, de http://www.squeakland.org/resources/articles/

LearnScratch [Portal Web] (2015a). Lesson 12: Juego de Pong. Recuperado 2 de Junio de 2015, de http://learnscratch.org/sc3-u3/sc3-l12 /

LearnScratch [Portal Web] (2015b). Lesson 11: Pack-Man Game. Recuperado 2 de Junio de 2015, de http://learnscratch.org/sc3-u3/sc3-l11

Maloney, J., Resnick, M., Rusk, N., Silverman, B., and Eastmond, E. 2010. The Scratch Programming Language and Environment. *ACM Trans. Comput. Educ.* 10, 4, Article 16 (November 2010),15 pages. Recuperado 2 de Junio de 2015, de http://doi.acm.org/10.1145/1868358.1868363

Olabe, J.C. and Rouèche, C. (2008). A Website to Disseminate Scratch Using Video Tutorials: LearnScratch.org. Conference Scratch@MIT 2008. July 24-26, Massachusetts Institute Technology, MIT Cambridge, MA, USA.

Olabe, J.C., Basogain, X. and Olabe, M.A. (2010). Teaching and Learning Scratch in Schools Around the World. Conference Scratch@MIT 2010. August 11-14, Massachusetts Institute Technology, MIT Cambridge, MA, USA.

Olabe, J.C., Basogain, X., Olabe, M. A., Maiz, I. and Castaño, C. (2011). Programming and Robotics with Scratch in Primary Education. Book title: *Education in a technological world: communicating current and emerging research and technological efforts*. Publisher: Formatex Research Center, A. Méndez-Vilas (Ed.) Dec-2011. ISBN: 978-84-939843-3-5. pp: 356-363

Papert, S. (1980). *Mindstorms: Children, Computers, and Powerful Ideas*. BasicBooks, New York.

Papert, S. (1991). *Situating constructionism*. In I. Harel & S. Papert (Eds.), Constructionism. 1-11. Norwood, NJ: Ablex

Postma, B. [Source Code] (2015). PacMan Game. Recuperado 2 de Junio de 2015, de http://www.brianpostma.com/Applets/PacMan.zip

PISA (2015). *The Programme for International Student Assessment (PISA)*. El Programa para la Evaluación Internacional de Alumnos de la OCDE. Recuperado 2 de Junio de 2015, de http://www.oecd.org/pisa/

Repenning, A., Webb, D., Ioannidou, A., (2010). Scalable Game Design and the Development of a Checklist for Getting Computational Thinking into Public Schools, The 41st ACM Technical Symposium on Computer Science Education, SIGCSE 2010, (Milwaukee, WI), ACM Press.

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., & Kafai, Y. (2009). Scratch: programming for all. *Communications of the ACM*, 52(11), 60-67.

Scratch [Software de Ordenador] (2015). MIT Media Lab. Recuperado 2 de Junio de 2015, de http://scratch.mit.edu/

ScratchEd Team [Portal Web] (2015). Computational Thinking webinars. Recuperado 2 de Junio de 2015, de http://scratched.gse.harvard.edu/content/1488

Stem (2015). U.S. Department of Education. S*cience, Technology, Engineering and Math: Education for Global Leadership*. Recuperado 2 de Junio de 2015, de http://www.ed.gov/stem

Wing, J. (2006). Computational thinking. *Communications of the ACM*, 49 (3), 33-36. Recuperado 2 de Junio de 2015, de http://dx.doi.org/10.1145/1118178.1118215

Wing, J. (2011). Research notebook: Computational thinking—What and why? *The Link Magazine, Spring*. Carnegie Mellon University, Pittsburgh. Recuperado 2 de Junio de 2015, de http://www.cs.cmu.edu/link/research-notebook-computational-thinking-what-and-why