Una Indagación sobre el Comportamiento de Lenguajes de Programación Sencillos Basados en Bloques

J. Ángel Velázquez Iturbide

Escuela Técnica Superior de Ingeniería Informática
Universidad Rey Juan Carlos
Madrid, Spain
angel.velazquez@urjc.es

Resumen: Una de las dificultades que presenta el aprendizaje de la programación, en comparación con otras disciplinas, es que los programas tienen asociado un comportamiento dinámico que el profano no percibe. Diversos autores han puesto énfasis en la necesidad de enseñar explícitamente dicho comportamiento, en forma de modelos conceptuales del ordenador implicado por el lenguaje (también llamados "máquinas nocionales"). Aunque los lenguajes basados en bloques son más fáciles de aprender que los textuales, también es necesario desarrollar modelos conceptuales que expliquen su comportamiento. En este artículo se presenta un estudio realizado para conocer con todo detalle el comportamiento de dos lenguajes sencillos basados en bloques, Code.org y ScratchJr, especialmente de este último, dada su mayor complejidad. El artículo presenta la planificación del estudio y los resultados obtenidos. Como trabajos futuros, prevemos desarrollar una máquina nocional de ScratchJr que facilite al alumno el aprendizaje del comportamiento de sus programas.
Palabras clave: Enseñanza de la informática, Estado de un programa, Máquina nocional, Programación basada en bloques, Code.org, ScratchJr.

Abstract: One of the main difficulties on learning to program, compared to learning other disciplines, is that programs have associated a dynamic behavior which is not perceived by novices. Different authors have emphasized the need of explicitly teaching such a behavior, as conceptual models of the computer implied by the programming language (also known as "notional machines"). Although block-based languages are easier to learn than textual languages, it is also necessary to develop conceptual models which explain their behavior. In this article, we present a study conducted to know in full detail the behavior of two block-based languages, namely Code.org and ScratchJr, especially the latter, given its higher complexity. The article presents the experimental setting and the results obtained. In the near future, we intend to develop a notional machine for ScratchJr which will hopefully assist the novice in learning the behavior of his/her programs.
Key words: Computer Science education, Program state, Notional machine, Block-based programming, Code.org, ScratchJr.

1. Introducción

En el ámbito de la investigación en enseñanza de la informática, el aprendizaje de la programación ocupa un papel destacado. La programación es una de las primeras materias de informática que se estudian y la que presenta mayor dificultad. Las razones de esta dificultad son múltiples: necesidad de aprender simultáneamente un lenguaje de programación y la resolución algorítmica de problemas, naturaleza abstracta de los problemas planteados, alto nivel cognitivo de la asignatura, falta de conocimientos previos relacionados con la materia, etc. Puede encontrarse más información en diversos estudios (p.ej. Robins, 2019; Gomes y Mendes, 2007; Luxton-Reilly *et al.*, 2018).

En la última década ha aparecido una nueva generación de lenguajes de programación basados en bloques, de los que el más conocido es Scratch (Resnick *et al.*, 2009). Estos lenguajes eliminan algunas dificultades para su aprendizaje (Bau, Gray, Kelleher, Sheldon y Turbak, 2017), siendo incluso una alternativa para la introducción a la misma en la universidad, aunque su uso no está exento de problemas (Martínez-Valdés, Velázquez-Iturbide y Hijón-Neira, 2017). Sin embargo, se usan principalmente en la educación preuniversitaria, con niños y jóvenes. Estos alumnos también padecen dificultades y desarrollan malas concepciones (Swidan, Hermans y Smit, 2018).

Un recurso didáctico que últimamente se está resaltando para mejorar el aprendizaje de la programación es la explicación del comportamiento dinámico de los programas, es decir, de su ejecución. La programación de computadores se diferencia de otras materias en que el texto estático de un programa tiene asociada una semántica dinámica. Por tanto, un alumno debe aprender simultáneamente los aspectos estáticos y dinámicos de un lenguaje de programación. Los aspectos estáticos son de naturaleza lingüística (p.ej. su sintaxis), mientras que su dinámica remite a un modelo del comportamiento del programa, al menos implícito. Dicho modelo explica el efecto, a veces oculto, de las instrucciones sobre un hipotético computador, máquina virtual o "máquina nocional" (Sorva, 2013).

Aunque los estudios sobre las dificultades de los alumnos para comprender el comportamiento de los programas se remontan a los años 70 (Mayer, 1979), son du Boulay, O'Shea y Monk (1981) quienes utilizan por primera vez el término "máquina nocional". Es un modelo del ordenador implicado por las construcciones del lenguaje de programación. Por tanto, es un modelo conceptual que debe utilizarse para explicar a los alumnos el comportamiento de un lenguaje de programación. No es una descripción del procesador real ni una definición formal de la semántica del lenguaje de programación ni el modelo mental que el alumno construye al aprender la materia. Una máquina nocional puede explicarse con texto, diagramas o incluso mostrarse en alguna herramienta software (p.ej. un entorno de programación, un depurador o un sistema de visualización de programas).

No se han hecho esfuerzos para diseñar máquinas nocionales para lenguajes basados en bloques, como Scratch (Resnick *et al.*, 2009), ni siquiera hay disponibles descripciones detalladas de la dinámica de estos lenguajes (Seppälä *et al.*, 2019). Puede haber varias razones para esta carencia. Por un lado, son lenguajes cuyo efecto se hace visible con los movimientos de los personajes en un "escenario", por lo que puede parecer innecesaria ninguna otra explicación. Asimismo, los atributos de sus elementos son más difíciles de representar que en los lenguajes convencionales, ya que son de naturaleza visual o auditiva en lugar de valores en memoria. Sin embargo, es importante el diseño de máquinas nocionales para estos lenguajes dada la importancia creciente de la enseñanza preuniversitaria de la programación (Velázquez-Iturbide, 2018), que deberá enseñarse a niños, jóvenes e incluso a sus profesores.

En el presente artículo se describen con detalle dos lenguajes basados en bloques, Code.org y ScratchJr. Hemos comenzado con lenguajes sencillos para afrontar el reto con más probabilidades de éxito y para adquirir experiencia antes de abordar lenguajes más complejos, como Scratch. Se identifican sus principales construcciones, se describe su comportamiento y, destacadamente, se caracteriza el estado de un programa en ejecución.

La estructura del artículo es la siguiente. En las dos siguientes secciones se describe el comportamiento de los lenguajes en orden creciente de complejidad, primero Code.org y después ScratchJr. En cada sección se presenta el lenguaje, se describe la metodología usada para la indagación, se describe su dinámica y se analiza el soporte dado actualmente a la misma. Terminamos con nuestras conclusiones y líneas de trabajo presente y futuro.

2. Comportamiento de Code.org

Code.org es una organización no gubernamental estadounidense. Ofrece cursos gratuitos de programación, para los que han utilizado imágenes de juegos y películas populares de sus colaboradores, como La Edad de Hielo, Plantas contra Zombis, Angry Birds o Minecraft. Sucesivamente mostramos el aspecto del entorno y el lenguaje, la metodología

de la indagación, y una descripción de la dinámica del lenguaje.

2.1. Programas del "Curso Express de Prelectores"

En el sitio web de Code.org (https://studio.code.org/) se encuentran recursos para profesores y alumnos, entre ellos varios cursos. En concreto, el "Curso Express de Prelectores" (https://studio.code.org/s/pre-express-2019/) es adecuado para niños de 4 a 8 años. Hemos seleccionado este curso porque sería adecuado como preludio a ScratchJr.

El curso consta de 12 lecciones, agrupadas en 3 secciones:

1. Secuencias, formada por 6 lecciones.
2. Bucles, formada por 4 lecciones.
3. Eventos, formada por 2 lecciones.

Cada lección se basa en algún micromundo y consta de ejercicios y quizá uno o dos vídeos explicativos. Muchas lecciones son similares porque permiten practicar con los mismos conceptos de programación, aunque usando personajes y fondos distintos. Por concreción, hemos seleccionado el micromundo de la ardilla Scrat, de la película La Edad de Hielo. Recordemos que Scrat siempre está intentando coger una bellota de roble.

Los programas a desarrollar en Code.org se limitan a la resolución de puzles, en el sentido de Pelánek y Effenberger (2020). Son retos planteados de forma precisa, que pueden resolverse siguiendo reglas claramente definidas.

La Figura 1 contiene parte de la pantalla mostrada al seleccionar el ejercicio 1 de la lección de bucles con Scrat. En la parte izquierda, se presenta un micromundo. La ardilla y la bellota se encuentran en dos casillas distintas de una cuadrícula. Las casillas con color blanco, liso, representan hielo que la ardilla puede pisar sin peligro, mientras que las casillas "agrietadas" representan hielo frágil, que se romperá si la ardilla lo pisa. El área azul es agua. Todo el área de juego tiene unas dimensiones de 8×8, aunque las casillas con agua no están delineadas.

Figura 1. Presentación parcial en Code.org del ejercicio 1 de bucles de Scrat

A la derecha del área de juego hay otras dos áreas, que la Figura 1 muestra parcialmente. La zona superior derecha contiene el enunciado del puzle, mientras que la zona inferior derecha engloba un área de bloques y un espacio de trabajo, que contienen bloques que representan diversas acciones. El enunciado indica qué debe hacerse con los bloques presentados para que la ardilla Scrat llegue, sin caer al agua, desde su casilla hasta la casilla de la bellota. Los bloques pueden arrastrarse desde el área de bloques al espacio de trabajo y ensamblarse con otros bloques, formando una secuencia de instrucciones para Scrat (es decir, un programa) que se lee de arriba abajo. El programador sólo puede resolver el puzle presentado en cada ejercicio.

La solución del puzle de la Figura 1 es sencilla: basta con que Scrat dé cinco pasos a su derecha. Esto puede conseguirse de varias formas. La Figura 2 muestra dos soluciones, una basada en una secuencia de 5 pasos a la derecha y la otra en repetir 5 veces la acción de dar un paso a la derecha.

2.2. Metodología de la Investigación

Se han realizado todos los ejercicios de las lecciones 1 ("aprende a arrastrar y soltar"), 2 ("secuencias con Scrat") y 7 ("bucles con Scrat"). Dado lo sencillo del lenguaje, no ha sido necesaria ninguna otra

planificación para la indagación. Asimismo, se presenta su comportamiento tan detalladamente que casi puede considerarse una máquina nocional, sólo que descrita verbalmente.

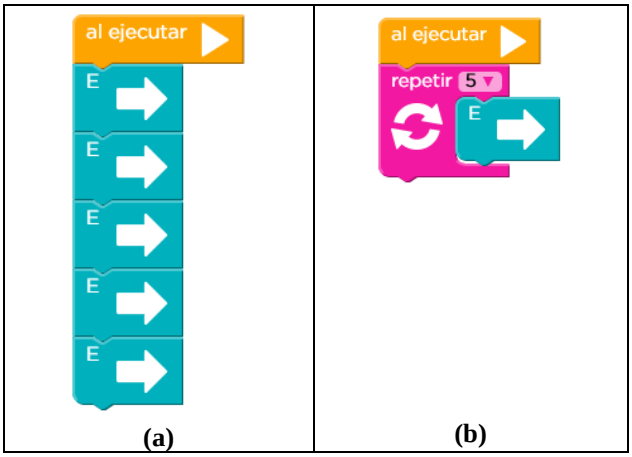

Figura 2. Dos programas alternativos para avanzar 5 pasos a la derecha: (a) sin bucles, (b) con un bucle

2.3. Resultados sobre la Ejecución de Code.org

El comportamiento de estos puzles es sencillo, aunque no todos sus elementos son evidentes. Obviamente, la parte más sencilla consiste en definir el comportamiento de sus instrucciones. Existen 4 bloques de movimiento (arriba, abajo, a la derecha y a la izquierda, véase Figura 1), cuyo efecto consiste en que Scrat avance un paso en la dirección especificada en el bloque.

Los bloques pueden ensamblarse formando una secuencia, y su ejecución también se realiza en secuencia, uno tras otro de arriba a abajo. Cada bloque se ejecuta de forma indivisible y sólo tras su terminación se continúa por el siguiente.

Por último, el bucle disponible en Code.org simplemente itera un número dado de veces (véase Figura 1). Aunque en los casos más sencillos, su comprensión es fácil, hay que definirlo de forma precisa porque puede englobar más de un bloque de movimiento, e incluso otros bucles (véase Figura 3).

Sea la siguiente explicación de la ejecución de un bucle, suponiendo que se ha seleccionado un número *n* de repeticiones (en el menú que se despliega a la derecha de su etiqueta "repetir", véase Figura 2):

1. Tomamos nota de que empezamos la primera iteración. Podemos llevar la cuenta de las iteraciones realizadas usando un contador *i*; empezamos con *i*=1.
2. Ejecutamos en secuencia las instrucciones englobadas por el bucle.
3. Si la iteración realizada debe ser la última, es decir, si *i*=*n*, termina la ejecución del bucle y continuamos ejecutando el bloque que esté ensamblado en la parte inferior del bucle. En caso contrario, incrementamos el contador *i* en 1 y volvemos al paso 2.

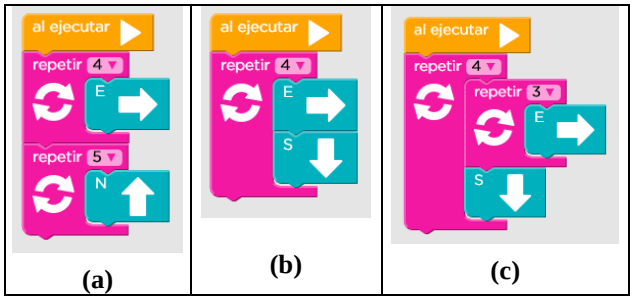

Figura 3. Tres programas con distintos anidamientos de bloques de movimiento y de bucle

No obstante las reglas anteriores, la ejecución mental de un programa Code.org se hace de forma intuitiva. Para hacerlo de forma precisa, debemos introducir el concepto de estado de un programa. Se define como el conjunto de valores necesarios para controlar la ejecución del programa. En el micromundo de Scrat, el estado de un programa consta de:

- Valor de cada casilla. Una casilla puede tener hielo sólido, hielo quebrado o agua.
- Situación de Scrat y de la bellota. Es la casilla donde se encuentra cada uno.
- Bloque activo del programa. Durante la ejecución del programa, los bloques se van ejecutando en secuencia, por lo que debemos saber cuál se está ejecutando en cada momento.

La existencia de bucles y de anidamiento de bloques hace necesaria la declaración de reglas aún más precisas sobre el bloque activo durante la ejecución de un bloque "repetir":

- Se añade el contador *i* al estado del bloque

"repetir", que se va actualizando durante su ejecución.

- Cada bloque "repetir" tiene su propio contador.
- El bloque activo oscila entre el propio bloque de bucle (cuando actualiza y comprueba el valor del contador *i*) y la secuencia de bloques de su cuerpo (cuando se ha determinado que debe ejecutarse).

Finalmente, hay que aclarar cómo se controla y evoluciona la ejecución de un programa. La ejecución de un programa arranca siempre que el usuario pulsa la tecla "Ejecutar" o "Paso" (véase Figura 1). En el primer caso, se ejecuta el programa completo, mientras que en el segundo, se ejecuta el primer bloque no iterativo (más la ejecución de bucles estrictamente necesaria para alcanzar dicho bloque). El botón "Paso" puede pulsarse repetidamente. En cualquier caso, la ejecución del programa se acompaña de una animación sonora del micromundo.

La ejecución de un programa puede encontrarse en varios estados:

- Estado inicial. Es el estado en que arranca la ejecución de un programa, coherente con el enunciado del problema.
- Estado intermedio. El programa está en ejecución pero aún no ha terminado.
- Estado final de éxito. El programa ha acabado habiendo alcanzado el objetivo buscado, es decir, Scrat ha llegado a la bellota. El programa Code.org termina en cuanto Scrat coge la bellota, aunque queden bloques o iteraciones por ejecutar.
- Estado final de fracaso. El programa ha acabado, sin haber alcanzado el objetivo buscado. Es decir, Scrat no ha llegado a la bellota porque se ha quedado corto o incluso se ha ahogado.

2.4. Soporte de Code.org a la Máquina Nocional

El lenguaje de programación usado en Code.org es muy simple. Aun así, hemos visto que su comprensión conlleva tener en cuenta varios detalles y las interacciones entre los distintos elementos del lenguaje y del entorno. Evidentemente, sería deseable que Code.org diera soporte al usuario para que pueda seguir la ejecución del programa, es decir, soporte para su rastreo (*tracing*). Veamos que sólo proporciona un soporte parcial.

La facilidad más destacada que se proporciona al usuario es la posibilidad de ejecutar un programa paso a paso, es decir, bloque a bloque. Puede saberse cuál es el bloque activo porque se resalta con un marco amarillo (véase Figura 4).

Sin embargo, Code.org no proporciona información sobre el proceso de ejecución de un bucle. En un bucle, siempre se resalta algún bloque interno, pero nunca el propio bloque "repetir", por lo que el proceso de control del bucle queda oculto al programador.

Esta falta de información puede ser grave cuando se ejecutan bucles anidados. Por ejemplo, el bucle más interno de la Figura 3(c) solamente se resalta como muestra la Figura 4(a), sin que podamos saber en cada momento la iteración que está realizando ninguno de los dos bucles. Mientras no se ejecuta el bucle más interno, se tiene algo más de información, ya que entonces sabemos que se está ejecutando el bloque de movimiento que le sigue (véase Figura 4(b)), aunque seguimos ignorando el estado de ejecución del bucle más externo.

Aunque sea un detalle menor, también hemos comprobado que, en algunas situaciones de terminación con fracaso, señala erróneamente la casilla donde se debería romper el hielo.

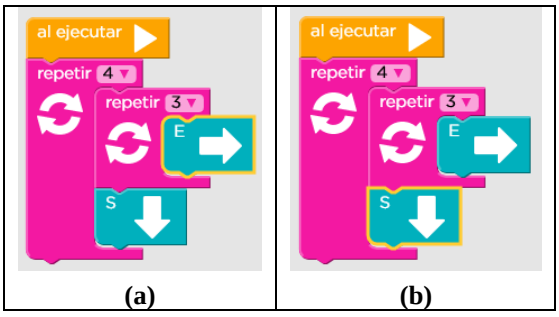

Figura 4. Dos estados intermedios en la ejecución de un programa con bucles anidados

.

Figura 5. Pantalla principal de ScratchJr

3. Comportamiento de ScratchJr

ScratchJr (Flannery *et al.*, 2013) es un lenguaje de programación visual diseñado para introducir a niños de 5 a 7 años de edad en la programación. ScratchJr es un derivado del popular lenguaje Scratch que permite programar sin necesidad de saber leer. Puede encontrarse más información en su página web (https://www.scratchjr.org/).

3.1. Programas ScratchJr

La Figura 5 muestra la interfaz de usuario de ScratchJr. Podemos destacar varias zonas:

- Escenario. Está situada en la parte central de la pantalla y es el área donde transcurre la acción. En la Figura, pueden verse un gato y un pollo en un paisaje campestre.
- Páginas. Está situada en la parte superior derecha, donde se muestran los distintos fondos en los que transcurrirá la acción del programa. En la Figura se muestra la única página existente (el paisaje) más la posibilidad de añadir otras páginas (signo más).
- Personajes. Este área está situada en la parte superior izquierda y es la zona donde se muestran los personajes de la página actual del programa. La etiqueta del personaje que se encuentra seleccionado aparece agrandada y enmarcada en naranja (en la Figura, el gato).
- Controles. Encima del escenario aparecen varios controles, entre ellos la bandera verde que se utiliza para iniciar la ejecución de un programa.
- Zona de programación. En esta zona situada en la parte inferior de la pantalla, se construye el programa del personaje seleccionado. Su programa puede estar formado por varias secuencias de bloques ("guiones", *scripts*) que pueden ejecutarse en paralelo. Obsérvese en la Figura que los bloques de un guión se disponen en horizontal, de izquierda a derecha.
- Categorías y paleta de bloques. El escenario y la zona de programación están separadas por una franja horizontal. En su parte izquierda aparecen seis iconos que representan las seis categorías de bloques. A su derecha, aparecen los bloques disponibles en la categoría seleccionada; en la Figura, muestra los ocho bloques de la categoría de movimiento (color azul).

El usuario de ScratchJr puede desarrollar libremente cualquier programa sin estar limitado por un reto (como en Code.org). Un programa se crea mediante el arrastre de bloques desde la paleta de bloques a la zona de programación y la particularización de los parámetros del bloque, si los tiene.

3.2. Metodología de la Investigación

Nuestra indagación sobre ScratchJr se realizó en dos fases. Puede encontrare con todo detalle en Velázquez-Iturbide (2021).

En una primera etapa, nos familiarizamos con el lenguaje ScratchJr, ya que conocíamos sus principales características pero no habíamos desarrollado ningún programa. Por tanto, desarrollamos los programas contenidos en las nueve tarjetas disponibles en el sitio web de ScratchJr (2021). También desarrollamos algunos programas más, sin ningún plan.

En una segunda fase, analizamos bloque a bloque sus características. Con frecuencia, sabíamos o intuíamos ciertos comportamientos, pero debíamos estar totalmente seguros de ellos. En algunos casos, un bloque producía un efecto sobre otros bloques, o había que usar varios bloques coordinadamente. La indagación también se amplió a diversas propiedades o al comportamiento de guiones, personajes, escenarios y páginas.

Sólo teníamos un instrumento para comprobar los comportamientos, los recursos disponibles por medio de su entorno de programación. Por tanto, inferimos visualmente el comportamiento de los bloques. Según el caso, usamos las siguientes formas de obtener información visualmente:

- Activando la rejilla. Esta facilidad, proporcionada por un icono situado sobre el escenario, activa el sistema de coordenadas en el mismo.
- Observando el movimiento absoluto de los personajes o comparando el avance o velocidad relativo de varios caracteres.
- Haciendo que los personajes dijeran algo (se muestra un bocadillo de comic).

Para algunos bloques, fue suficiente con arrastrar y soltar el bloque desde la paleta de bloques a la zona de programación y ejecutarlo (pulsándolo). En otras ocasiones, se planificó y desarrolló un programa para comprobar ciertos comportamientos. Desarrollamos varios programas de prueba, dos de ellos variaciones de las tarjetas disponibles en el sitio web de ScratchJr (2021). No hacía falta que fueran demasiado complejos, normalmente con 2 ó 4 personajes. Resumimos una selección de los programas:

1. Bloques “Comenzar al tocar” y “Parar”. Diseñamos un programa con dos caracteres para comprobar la facilidad de separar dos caracteres que se tocan, y terminar la ejecución del programa completo de una forma controlada.
2. Bloque “Fijar velocidad”. Adaptamos la tarjeta 3 de ScratchJr (2021), donde tres personajes alineados verticalmente avanzan de izquierda a derecha. Comprobamos qué otros bloques se veían afectados al fijar la velocidad. También comprobamos si su efecto era persistente de acople ejecución a otra del programa, y si la propiedad de velocidad se asociaba con un guión o con un personaje.
3. Bloque “Parar”. Usamos dos personajes, de los que uno ejecutaba un bloque “parar”. Queríamos saber si este bloque afecta a todos los guiones de su personaje, incluyendo el guión donde se encuentra el bloque, así como a otros personajes.
4. Bloques “Comenzar con mensaje” y “Enviar mensaje”. Se construyó un programa con dos personajes que se comunicaban por medio de un mensaje. El objetivo era comprobar si los mensajes son un mecanismo de sincronización entre dos procesos paralelos o como una invocación a subrutina. Después, también estudiamos varios detalles relacionados, como qué sucede si el personaje que recibe el mensaje no termina su ejecución, si no tiene un bloque de fin, o si ningún personaje recibe el mensaje.
5. Bloques “Comenzar con mensaje” y “Enviar mensaje”. Adaptamos la tarjeta 9 de ScratchJr (2021), con una estructura más compleja de envío y recepción de mensajes para reforzar nuestra comprensión de su comportamiento como invocación de subrutinas. También modificamos el programa de forma que un mensaje fuera recibido por tres personajes. Indagamos el comportamiento del envío de

mensajes cuando los tres receptores hacían lo mismo y cuando un personaje tardaba más en terminar su tarea o incluso no terminaba.

3.3. Programa de Muestra

Incluimos en esta sección uno de los programas bosquejados antes, el número 4 (véase la Figura 6). Contiene un diálogo entre dos personajes (Gato y Pollo) para comprobar si el emisor siempre retrasa su ejecución hasta que el guión activado termina. Utilizamos bloques "Decir" para rastrear la ejecución del programa. Se usó un diálogo como medio de comprobar visualmente la secuencia de ejecución de los bloques de los guiones. El hipotético diálogo (en inglés) era:

1. Gato: "Hi, Chicken". "If you are touched, notify me".
2. Pollo: "All right".
3. Gato: "Thank you, Chicken".
4. Cuando Pollo es tocado, salta, dice "Now, Cat!" y envía un mensaje. Gato da las gracias y desaparece.

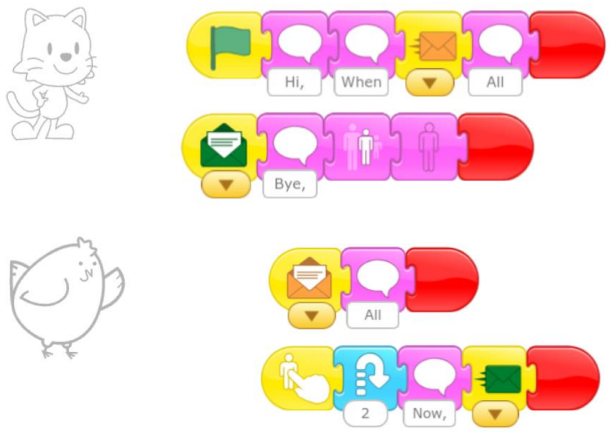

Figura 6. Programa de prueba número 4

Tal y como se explica arriba, este programa básico se modificó (en los guiones de Pollo) para comprobar algunas cuestiones adicionales. Por ejemplo, reemplazar el bloque "Fin" en el primer guión del pollo por un bloque "Repetir indefinidamente" permite comprobar que el guión de Gato que envió el mensaje permanece parado, dado que el guión llamado no termina.

3.3. Resultados sobre la Ejecución de ScratchJr

Presentamos, por falta de espacio, un resumen de los hallazgos de la investigación. El lector interesado puede encontrar más detalle en Velázquez-Iturbide (2021).

Como muestra la Figura 5, existen seis categorías de bloques, cada una distinguible mediante un color:

- Bloques de movimiento (color azul).
- Bloques de apariencia (color rosa).
- Bloques de sonido (color verde).
- Bloques disparadores (color amarillo).
- Bloques de control (color naranja).
- Bloques de finalización (color rojo).

Las dos primeras categorías permiten cambiar atributos de los personajes mediante su movimiento o cambio de aspecto. El movimiento de los personajes tiene lugar en un escenario de tamaño fijo, en el que se mueven de forma circular, es decir, si desaparecen por un extremo, aparecen por el extremo contrario. La tercera categoría de bloques permite reproducir sonidos.

Las restantes tres categorías contienen bloques de control de la ejecución, que deben colocarse en partes distintas de un programa ScratchJr: los bloques disparadores se colocan al comienzo de un guión, los bloques de finalización se colocan al final y los bloques de control, en medio. Puede comprobarse viendo la forma de cada tipo de bloques.

El comportamiento de ScratchJr es similar al de Code.org en algunos elementos básicos: bloque de inicio de la ejecución (bloque disparador "Al presionar bandera verde", véanse Figuras 5 y 6), bloques de movimiento, secuencia de bloques y bucle de repetición. Obviamente, hay variaciones en los bloques de movimiento, ya que se introducen nuevos bloques (p.ej. saltar o girar) y se pueden parametrizar con un número de pasos. Asimismo, se introduce la categoría de apariencia. Ambas extensiones amplían los atributos de los personajes (p.ej. su orientación espacial o su tamaño), pero no implican un cambio drástico en la concepción de la máquina nocional.

Algunos bloques de control permiten cambiar algo la ejecución secuencial "normal" de un guión. El bloque "Esperar" provoca que la ejecución de un guión se pare durante varias décimas de segundo. El bloque "Fijar velocidad" permite variar la velocidad de ejecución de algunos bloques. Sin embargo, estos bloques no alteran el modelo conceptual consistente en la ejecución secuencial de bloques. Su influencia se limita a permitir un mayor control del tiempo para conseguir efectos multimedia.

El bloque de finalización "Repetir indefinidamente" introduce un cambio conceptual algo mayor en el flujo de ejecución, ya que produce un bucle infinito. Sin embargo, no es un cambio grande, siendo incluso más fácil seguir su ejecución que la del bucle "Repetir", ya que no necesita tener un contador asociado.

Veamos las principales novedades que introduce ScratchJr sobre el comportamiento de Code.org. Una novedad destacada es que no existe un programa monolítico. Cada personaje puede tener asociado un programa propio. Además, cada personaje puede tener varios guiones (véanse el gato y el pollo de la Figura 6, cada uno con dos guiones). Los guiones que forman un programa pueden ejecutarse en paralelo. Si suponemos que la máquina nocional dispone de un solo "procesador", esto implica que los bloques de los distintos guiones se ejecutan de alguna forma alterna y equitativa, aunque imprevisible.

El número de guiones que se ejecutan en paralelo en un programa ScratchJr puede variar en el tiempo. Cada guión inicia su ejecución cuando sucede un evento que es detectado por su bloque disparador. Hay cuatro clases de eventos:

- Al presionar la bandera verde.
- Al pulsar al personaje.
- Al recibir un mensaje.
- Al tocar al personaje.

Los dos primeros eventos responden a acciones del usuario, uno para iniciar la ejecución del programa y otro en cualquier momento de la ejecución del programa. El tercer evento permite sincronizar la ejecución de varios guiones, normalmente de personajes diferentes. Finalmente, el cuarto evento tiene un carácter más imprevisible, como vemos más adelante.

Un matiz del envío y recepción de mensajes puede inducir a error a los aprendices de ScratchJr con experiencia en programación concurrente. Por el nombre de los bloques, el programador puede pensar en un paso de mensajes síncrono. Sin embargo, el envío de un mensaje en ScratchJr es más parecido a una llamada a subprograma (Flannery *et al.*, 2013). Es decir, el guión que envía un mensaje se queda en espera hasta que termina el guión que ha iniciado su ejecución al recibir el mensaje.

Los restantes bloques de las categorías de control y de finalización aún amplían más los comportamientos posibles. La ejecución del bloque "Parar" en un guión de un personaje produce la interrupción de los demás guiones del personaje. El cambio más drástico en el flujo de ejecución lo produce el bloque de finalización "Ir a página". Este bloque produce un cambio de fondo en el escenario, la desaparición de todos los personajes de la página anterior y la aparición de los personajes de la nueva página. Obviamente, se para la ejecución de los guiones activos de los personajes de la página anterior y se inician los guiones de los nuevos personajes que tengan un bloque disparador "Al presionar bandera verde". Por tanto, las páginas delimitan el ámbito de fondos, personajes y guiones.

Faltan por analizar dos cuestiones que se trataron en el comportamiento de Code.org. La primera es el estado de un programa ScratchJr. El elemento principal es la página en ejecución. Dependiendo de la página activa, el estado de un programa consta de:

- Atributos de cada personaje: posición por defecto (en ambos ejes), posición actual, orientación espacial (en grados), texto que está diciendo, tamaño por defecto, tamaño actual, y si es visible.
- Guiones en ejecución. Para cada guión, debe conocerse el bloque en ejecución. En algunas situaciones, también es necesario conocer: si el guión está esperando la terminación de otro guión (tras un envío de mensaje), y el estado de la ejecución de algunas instrucciones individuales ("Repetir", "Esperar", tiempo transcurrido de la ejecución de bloques afectados por un bloque "Fijar velocidad").

- Otros valores: fondo mostrado en cada página, estado de la interpretación de un sonido.

Una segunda cuestión es el estado de la ejecución de un programa. Ya hemos visto que la ejecución de un guión puede iniciarse con varios bloques disparadores. En contrapartida, esta versatilidad puede producir poca claridad y control del usuario sobre el comportamiento del programa. Como se comentó antes, el usuario suele tocar la bandera verde para iniciar la ejecución. Una vez que el programa se encuentra en ejecución, el icono de la bandera verde se convierte en un hexágono rojo, cuya pulsación permite al usuario parar todos los guiones. El usuario también puede iniciar la ejecución de un programa al tocar un personaje, aunque es más normal que esta acción la realice una vez que el programa se está ejecutando.

Sin embargo, un guión puede iniciar su ejecución debido a que dos personajes se estén tocando. Esta situación puede ser imprevisible en un programa, pudiéndose producir situaciones desconcertantes para el usuario. Por ejemplo, supongamos que el usuario ha parado con el icono de hexágono rojo un programa en el que los personajes se mueven. Si dos personajes se están tocando, puede iniciarse inmediatamente la ejecución de sus guiones con el bloque disparador "Comenzar al tocar". El usuario puede volver a pararlo y el guión se seguirá iniciando hasta que los personajes se separen (bien por acción del usuario o por movimiento de los personajes).

3.4. Soporte de ScratchJr a la Máquina Nocional

En el sitio web de ScratchJr se encuentran publicados dos documentos sobre el lenguaje: definición de los bloques y programas de ejemplo. Sin embargo, ambos documentos explican insuficientemente la dinámica de los programas.

ScratchJr también da algún soporte para mostrar el estado de un programa en ejecución:

- Página activa, resaltándola con un marco naranja.
- Bloques activos. En cada guión se resalta el bloque activo mediante un cambio de tono.
- Posición de un personaje. Puede activarse el control de cuadrícula, haciendo visible sobre el propio escenario las coordenadas de la posición del personaje activo.

Sin embargo, este soporte es insuficiente. En primer lugar, no se proporciona al usuario un control de avance paso a paso, como en Code.org. En segundo lugar, los bloques disparadores y el bucle "Repetir" no se resaltan cuando se están ejecutando. Por último, aunque es fácil determinar la posición y los bloques activos de los guiones del personaje activo, se desconocen para los demás personajes, así como el resto de sus atributos.

4. Conclusiones

Se ha descrito el comportamiento de dos lenguajes de bloques sencillos, Code.org y ScratchJr, sobre todo los bloques de control de la ejecución, como los bucles. La Tabla 1 muestra los principales conceptos de programación presentes en cada lenguaje.

Tabla 1. Principales conceptos de programación

Code.org	**ScratchJr**
Secuencia Bucle	Secuencia Bucle Paralelismo Eventos Llamada a subrutina (envío y recepción de mensajes) Ámbito

También hemos visto que una comprensión completa de la ejecución de un programa exige conocer el estado del programa y conocer las condiciones de inicio y terminación de una ejecución del programa. Asimismo, hemos visto que los entornos proporcionados por sus creadores dan información incompleta sobre el estado de la ejecución, es decir, dan poco soporte a ninguna máquina nocional que se desarrolle.

Durante el mes de mayo del curso académico 2020-21 se ha intentado evaluar la comprensión del comportamiento de ScratchJr en la asignatura "Las TIC en la Educación", obligatoria del segundo cuatrimestre de primer curso del Grado de Educación Infantil de la Universidad Rey Juan Carlos. Se ha evaluado el grado de conocimiento de programación alcanzado en dos grupos, uno presencial y otro

online. La impartición del tema de programación ha variado entre ambos grupos. El grupo presencial utiliza los materiales de cursos pasados, mientras que para el grupo online se han desarrollado apuntes que presentan los comportamientos aquí descritos.

Se han realizado varias pruebas de conocimiento pero, debido a la pandemia, todas se han realizado online salvo el examen final. Por varios indicios, solamente podemos considerar fiables los resultados obtenidos en el examen final, que se realizó presencialmente. Hicieron el examen 69 alumnos del grupo presencial y 49 del online. El examen incluía 6 preguntas de tipo test sobre ScratchJr. Los resultados han sido muchos mejores para el grupo online (media igual a 4'8 sobre 6) que para el grupo presencial (2'65), con resultados estadísticamente significativos (p<<0'05).

Los comportamientos presentados para ScratchJr no constituyen una máquina nocional porque no están organizados como un modelo conceptual detallado y completo. Sin embargo, contiene los elementos principales para diseñar una máquina nocional, sobre todo la definición de estado de ejecución de sus programas. Actualmente estamos dando una estructura a las reglas de comportamiento presentadas para ScratchJr de forma que podamos calificarlas de máquina nocional. También queremos comprobar si el comportamiento descrito para el escenario de Scrat es suficiente para explicar el comportamiento de los programas de otros escenarios de Code.org.

Más adelante, se pretende desarrollar un sistema de rastreo/depuración de programas basado en sus respectivas máquinas nocionales. Asimismo, se espera realizar una investigación similar sobre Scratch, comenzando por una indagación sobre los aspectos menos claros del lenguaje.

Agradecimientos

Este trabajo se ha financiado con el proyecto de investigación e-Madrid-CM (P2018/TCS-4307) de la Comunidad Autónoma de Madrid (también financiado con fondos estructurales FSE y FEDER).

Referencias

Bau, D., Gray, J., Kelleher, C., Sheldon, J. y Turbak, F. (2017). Learnable programming: Blocks and beyond. *Communications of the ACM, 60*(6), 72-80, doi: 10.1145/3015455.

du Boulay, B., O'Shea, T. y Monk, J. (1981). The black box inside the glass box: Presenting computing concepts to novices. *International Journal of Man-Machine Studies*, *14*(3), 237-249, doi: 10.1016/S0020-7373(81)80056-9.

Flannery, L. P., Kazakoff, E. R., Bontá, P., Silverman, B., Bers, M. U. y Resnick, M. (2013). Designing ScratchJr: Support for early childhood learning through computer programming. En *Proceedings of the 12th International Conference on Interaction Design and Children, IDC '13*, pp. 1-10, doi: 10.1145/2485760.2485785.

Gomes, A. y Mendes, A. J. (2007). Learning to program – difficulties and solutions. En *Proceedings of the International Conference on Engineering Education, ICEE 2007*, Coimbra, Portugal: Academic Press, 2007.

Luxton-Reilly, A., Simon, Albluwi, I., Becker, B. A., Giannakos, M., Kumar, A. N., Ott, L., Paterson, J., Scott, M. J., Sheard, J. y Szabo, C. (2018). Introductory programming: A systematic literature review. En *ITiCSE'18 Companion*, pp. 55-106, doi: 10.1145/3293881.3295779.

Martínez-Valdés, J. A., Velázquez-Iturbide, J. Á. y Hijón-Neira, R. (2017). A (relatively) unsatisfactory experience of use of Scratch in CS1. En *Proceedings of 5th International Conference on Technological Ecosystems for Enhancing Multiculturality, TEEM'17*, 6 pp., doi: 10.1145/3144826.3145356.

Mayer, R. E. (1979). A psychology of learning BASIC. *Communications of the ACM*, *22*(11), 589-593, doi: 10.1145/359168.359171.

Paredes-Barragán, P. y Velázquez-Iturbide, J. Á. (2021). Evaluación del rendimiento académico sobre ScratchJr en el Grado en Educación Infantil. *Serie de Informes Técnicos DLSI1-URJC*, 2021-05,

Universidad Rey Juan Carlos. Recuperado de http://lite.etsii.urjc.es/technical-reports/.

Pelánek, R. y Effenberger, T. (2020). Design and analysis of microworlds and puzzles for block-based programming. *Computer Science Education*, doi: 10.1080/08993408.2020.1832813.

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B. y Kafai, Y. (2009). Scratch: Programming for all. *Communications of the ACM*, *52*(11), 60–67, doi: 10.1145/1592761.1592779.

Robins, A. V. (2019). Novice programmers and introductory programming. En S. A. Fincher y A. V. Robins (Eds.), *The Cambridge Handbook of Computing Education Research* (pp. 327-376), Cambridge University Press.

ScratchJr (2021). *ScratchJr – Teach – Activities*. Recuperado de https://www.scratchjr.org/teach/activities

Seppälä, O., Ball, T., Barik, T., Becker, B. A., Denny, P., Duran, R., Sorva, J. y Velázquez-Iturbide, J. Á. (2019). Notional machines for Scratch and Python. En M. Guzdial, S. Krishnamurthi, J. Sorva y J. Vahrenhold (Eds.), Notional Machines and Programming Language Semantics in Education, *Dagstuhl Reports*, *9*(7), 21, Dagstuhl Publishing, doi: 10.4230/DagRep.9.7.1.

Sorva, J. (2013). Notional machines and introductory programming education. *ACM Transactions on Computing Education*, *13*(2), article 8, doi: 10.1145/2483710.2483713.

A. Swidan, F. Hermans y M. Smit (2018). Programming misconceptions for school students. En *Proceedings of the International Conference on Cumputing Education Research, ICER'18*, pp. 151-159, doi: 10.1145/3230977.3230995.

Velázquez-Iturbide, J. Á. (coord..) (2018). Informe del grupo de trabajo SCIE/CODDII sobre la enseñanza preuniversitaria de la informática. Sociedad Científica Informática de España. Recuperado de https://www.scie.es/wp-content/uploads/2021/02/Informe-SCIE-CODDI-2018-06.pdf.

Velázquez-Iturbide, J. Á. (2021). Una indagación del comportamiento del lenguaje ScratchJr. *Serie de Informes Técnicos DLSI1-URJC*, 2021-03, Universidad Rey Juan Carlos. Recuperado de http://lite.etsii.urjc.es/technical-reports/.

Weintrop, D. (2019). Block-based programming in computer science education. *Communications of the ACM*, *62*(8), 22-25, doi: 10.1145/3341221.