Simulador de robótica educativa para la promoción del pensamiento computacional

Educational robotics simulator for fostering computational thinking

Cristian Manuel Ángel-Díaz
Universidad de La Laguna. San Cristóbal de La Laguna, España
cristian.angel.38@ull.edu.es

Eduardo Segredo
Universidad de La Laguna. San Cristóbal de La Laguna, España
esegredo@ull.edu.es

Rafael Arnay
Universidad de La Laguna. San Cristóbal de La Laguna, España
rarnayde@ull.edu.es

Coromoto León
Universidad de La Laguna. San Cristóbal de La Laguna, España
cleon@ull.edu.es

RESUMEN
Este trabajo presenta una herramienta Web libre y gratuita que facilita a cualquier centro educativo la enseñanza de conceptos básicos sobre robótica y programación y que, al mismo tiempo, permite desarrollar habilidades relacionadas con el pensamiento computacional: descomposición, abstracción, reconocimiento de patrones y pensamiento algorítmico. Esta herramienta permite diseñar y personalizar un robot móvil a través del uso de distintos tipos de sensores. Tras su creación, el robot se podrá poner a prueba en un entorno de simulación mediante distintos retos. El comportamiento del robot se puede definir a través de un lenguaje de programación visual basado en bloques que permite mover el robot a partir de la información del entorno recogida por los sensores.

Palabras Clave: Pensamiento computacional, robótica educativa, simulación, programación visual.

ABSTRACT
This work presents a free software tool that facilitates the teaching of basic robotics and programming concepts at any educational institution. At the same time, it allows the development of computational thinking skills to be carried out: decomposition, abstraction, pattern recognition and algorithmic thinking. This tool allows the design and configuration of a robot through the specification of different types of sensors. After designing the robot, its behaviour can be simulated by means of different challenges proposed to the user. This behaviour is defined through a block-based visual programming language. Blocks allow actions that the robot has to perform based on the information gathered by the different sensors to be defined in order to pass a challenge.

Keywords: Computational thinking, educational robotics, simulation, visual programming.

INTRODUCCIÓN

Saber programar hoy en día es una necesidad transversal, como lo es la lectura y la escritura debido, en parte, a que todos nos encontramos vinculados a la tecnología, sea cual sea nuestra ocupación. Estar formados en esta disciplina permitirá que nos podamos relacionar de forma más natural con el mundo que nos rodea, además de mejorar la creatividad, la capacidad de pensamiento crítico y estructurado y mejorar nuestra capacidad de resolver problemas gracias al *Pensamiento Computacional* (Perez Paredes & Zapata Ros, 2018; "Try Computational Thinking," 2020; Wing, 2008; Zapata-Ros, 2015). Por estos motivos y muchos otros, la enseñanza de este tipo de competencias es más necesaria que nunca, por lo que deben ser introducidas desde edades tempranas con el objetivo de que las generaciones venideras estén preparadas para la sociedad de las nuevas tecnologías.

La motivación principal del desarrollo de este proyecto surge de la posibilidad de contribuir a que se impartan asignaturas cuyos contenidos se encuentren directamente relacionados con el pensamiento computacional, incluyendo dichos contenidos en los currículos educativos de enseñanza obligatoria. En el caso que nos ocupa, nuestra aportación consiste en ofrecer herramientas de simulación de código abierto y gratuitas que puedan ser fácilmente accesibles por la mayor cantidad de centros educativos posible. Para lo anterior, hemos apostado por la *Robótica Educativa* (Moreno et al., 2012) como metodología para la enseñanza de las competencias anteriormente mencionadas. La docencia en las materias que involucran este tipo de metodología se basa en la propuesta de un reto o problema que debe ser superado por el alumnado a través de su ingenio y de las herramientas con las que cuenta para hacer que un robot supere dicho reto. Gracias a que el punto de partida consiste en un problema concreto, se pueden aplicar y desarrollar las principales habilidades relacionadas con el pensamiento computacional para su resolución: descomposición del problema, abstracción, reconocimiento de patrones. Por último, también interviene el pensamiento algorítmico, dado que se debe diseñar un algoritmo que controle el comportamiento del robot a la hora de resolver el reto planteado. La herramienta que aquí se presenta permite crear un robot que puede personalizarse añadiendo sensores de diferentes tipos, los cuales permiten obtener información acerca de un entorno simulado y, por lo tanto, definir acciones que puede realizar el robot teniendo en cuenta dicha información. Se ha optado por simular un robot móvil dado que aporta una mayor libertad a la hora que crear programas que definan su comportamiento, en comparación, por ejemplo, con un robot estático de tipo manipulador.

El robot virtual se basa en un modelo diferencial, es decir, tiene dos ruedas motrices que le permiten girar sobre sí mismo, avanzar y retroceder. Esta configuración es muy común entre los kits de robótica educativa. Asimismo, los sensores que incorpora el robot virtual son los más frecuentemente utilizados en los proyectos educativos. Nuestra intención es ir incorporando nuevos tipos de sensores a medida que se añadan nuevos retos. En este sentido, una ventaja de usar una herramienta virtual es que se pueden simular sensores más complejos, de tipo de barrido láser, por ejemplo, sin un sobrecoste añadido. Este tipo de sensores incrementaría mucho el coste de un robot real y dificultaría, por tanto, su uso en los centros educativos.

La definición del comportamiento que seguirá el robot para superar los retos se lleva a cabo a través de un lenguaje de programación visual basado en bloques. Por un lado, se

pone a disposición del usuario un conjunto de bloques que permiten especificar sentencias básicas disponibles en cualquier lenguaje de programación, como pueden ser bucles o condiciones, entre otras. Además, también se ha diseñado un conjunto de bloques que proporcionan, por un lado, la información del entorno de simulación obtenida a través de los sensores y, por el otro, el repertorio de acciones que puede llevar a cabo el robot, en concreto, avanzar, retroceder, girar y detenerse. La combinación de los diferentes bloques disponibles permite el diseño de diferentes programas cuyo objetivo será superar el reto planteado.

Teniendo en cuenta lo anterior, la principal contribución de este trabajo es, tal y como se ha indicado anteriormente, el hecho de que la herramienta de simulación propuesta sea de código abierto y gratuita. De este modo, cualquier centro educativo la tiene a su disposición para desarrollar la capacidad de resolución de problemas de su alumnado a través del pensamiento computacional, además de introducir conceptos básicos sobre robótica y programación.

Tras considerar diferentes opciones, nos decantamos por proporcionar dicha herramienta como una aplicación Web, y de este modo, permitir maximizar su alcance respecto a la cantidad de potenciales usuarios. Otra gran ventaja, al tratarse de una herramienta de simulación, es que los centros educativos no tienen que invertir recursos económicos para la adquisición de robots físicos, los cuales pueden llegar a ser significativamente elevados. Solo se requiere de un ordenador con un navegador Web compatible y una conexión a Internet.

Si bien es cierto que, desde el punto de vista motivacional, el uso de un simulador despierta menor interés en el alumnado en comparación con el uso de robots físicos, no hay que perder de vista que uno de los principales objetivos de esta herramienta es maximizar su difusión en los centros educativos. La escasez de recursos económicos no debería ser un impedimento a la hora de promover competencias relacionadas con la robótica educativa y el pensamiento computacional.

Por último, cabe mencionar que la aparición de proyectos relacionados con esta temática podría motivar a otros desarrolladores a crear herramientas que contribuyan a tal fin.

ANTECEDENTES Y ESTADO DEL ARTE

Esta sección comienza con una breve descripción del pensamiento computacional, así como con las principales habilidades que desarrolla. Seguidamente, se lleva a cabo una breve descripción de las fases que intervienen en una sesión docente basada en pensamiento computacional. Por último, se lleva a cabo un análisis sobre herramientas similares a nuestra propuesta, haciendo especial hincapié a las contribuciones que aportamos.

Pensamiento computacional

Podemos definir el pensamiento computacional como la capacidad de un individuo de afrontar un problema por medio del uso de habilidades relacionadas con las Ciencias de la Computación (Perez Paredes & Zapata Ros, 2018; “Try Computational Thinking,” 2020; Zapata-Ros, 2015).

Tal y como dijo Beatriz Ortega-Ruipérez: "*La persona que emplea un pensamiento computacional puede descomponer el problema en pequeños problemas que sean más fáciles de resolver, y reformular cada uno de estos problemas para facilitar su solución por medio de estrategias de resolución de problemas familiares.*" (Ortega-Ruiperez & Asensio Brouard, 2018).

Podemos sacar como conclusión que, aplicando el pensamiento computacional, se pueden resolver problemas complejos de un modo más efectivo al habitual.
Las principales habilidades que fomenta el pensamiento computacional son las siguientes:

- **Descomposición del problema**. Consiste en dividir el problema original en subproblemas que sean más sencillos de resolver. La combinación de todas las soluciones de los subproblemas proporcionará la solución del problema original.
- **Reconocimiento de patrones**. Se basa en tratar de identificar similitudes entre los distintos subproblemas, las cuales permitirán atajar su resolución en el caso de que varios compartiesen la misma solución o parte de la misma.
- **Abstracción:** Se trata del grado de detalle con el que se trata un problema o subproblema, analizando sus propiedades por capas e ignorando aquella información que no es relevante en cada momento, de modo que se puedan destacar aquellas propiedades diferenciadoras del resto (Zapotecatl López, 2020).
- **Pensamiento algorítmico**. Consiste en proporcionar una serie de pasos bien definidos, o un algoritmo, que permita resolver el problema, tratando de ser lo más genérico posible. Lo anterior permite que problemas similares puedan ser resueltos con un algoritmo igual o similar al diseñado desde un primer momento.

Robótica educativa

La pensamiento computacional (Moreno et al., 2012) es una metodología didáctica que usa robots como medio para el desarrollo de competencias a través de la resolución de retos o problemas. Dichos retos deberán ser superados por el alumnado haciendo acopio del ingenio y de las herramientas que les sean proporcionadas.

Por lo general, en una sesión docente basada en esta metodología, el profesorado comienza proporcionando unas nociones básicas a los estudiantes, para a continuación, proponer un problema que debe ser resuelto. El aula en la que se desarrolla la actividad cuenta con todo el material necesario, principalmente, hardware y software para, respectivamente, construir y programar el robot.

Podemos separar en distintas fases el desarrollo de una sesión docente de robótica educativa:

- **Planteamiento del problema**. Se trata de investigar y analizar el reto propuesto con el objetivo de comenzar a proponer una solución. En este punto intervienen habilidades como la descomposición y la abstracción, entre otras.
- **Diseño y creación del robot**. A raíz de lo descubierto en la primera fase, se comienza a diseñar y construir un robot que reúna los requisitos necesarios para superar el problema planteado, teniendo en mente cuál debe ser el comportamiento del mismo y si dispone de los componentes necesarios para poder obtener la información y ejecutar las acciones requeridas. Entre las

habilidades que intervienen en esta fase, cabe mencionar el reconocimiento de patrones y la creatividad.

- **Programación del comportamiento**. En este punto se debe diseñar e implementar un algoritmo que permita al robot recopilar información del entorno y actuar en consecuencia, siempre teniendo en mente la resolución del problema propuesto. El pensamiento algorítmico es una de las habilidades más importantes a desarrollar en este punto.
- **Simulación**. En esta fase se deberá probar tanto el diseño como el comportamiento del robot, y si fuera necesario, se llevarán a cabo tantas modificaciones como sean necesarias para obtener un resultado satisfactorio, es decir, la resolución del problema planteado.
- **Documentación y exposición**. Se muestran evidencias de que el robot ha superado el reto planteado y el alumnado comparte su diseño con el resto de compañeros.

Tal y como puede observarse, las fases anteriores permiten desarrollar una gran cantidad de habilidades. Entre ellas, cabe mencionar todas y cada una de las habilidades que fomenta el pensamiento computacional. Al mismo tiempo, también se promueven otro tipo de habilidades muy importantes, y que además son de un carácter más social y comunicativo, como pueden ser el trabajo en equipo y la expresión oral.

Por último, es importante destacar que esta metodología incluye una componente lúdica, dado que los retos o problemas se suelen plantear a modo de juego. Lo anterior convierte al pensamiento computacional en una metodología ideal a la hora de introducir al alumnado conceptos relacionados con la tecnología.

Proyectos relacionados

En la literatura actual, pueden encontrarse herramientas software y hardware cuyo propósito es similar al nuestro. En las siguientes secciones se describirán brevemente algunas de las más destacadas.

CoppeliaSim

Es un simulador de robótica cuya interfaz (Fig. 1) puede recordar a la de Unity, tecnología que se describirá en una sección posterior. Para programar un robot se pueden usar varios lenguajes de programación ampliamente conocidos, entre los que se encuentran C/C++, Python, Java y Matlab.

Brinda la posibilidad de implementar de forma rápida nuestros diseños de robots y permite un uso, tanto a nivel industrial, como a nivel educativo. Cuenta con muchas funcionalidades como, por ejemplo, cálculos de cinemática directa/inversa o la capacidad de agregar sensores a nuestro robot para poder interactuar con el entorno. Es compatible con Linux, MacOS y Windows y cuenta con una versión de código abierto, y otra privada, con funcionalidades más avanzadas (“CoppeliaSim,” 2020).

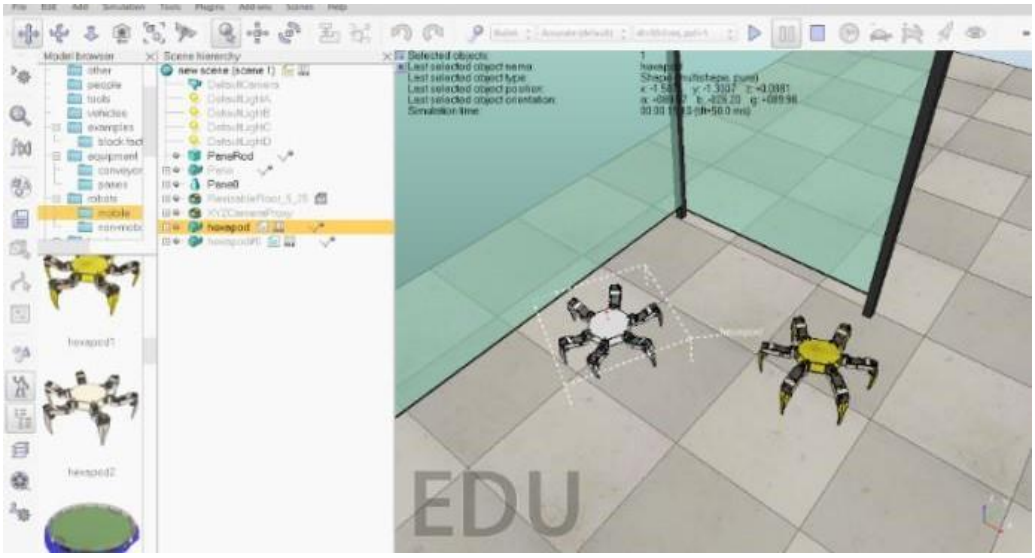

Fig. 1. CoppeliaSim

CoderZ

Es una solución basada en la nube orientada exclusivamente al ámbito educativo. Permite usar tanto Blockly ("Blockly," 2020) para los iniciados como la codificación en Java para la creación de robots virtuales en 3D.

Cuenta con un plan de estudios predefinido y muchas herramientas que ayudan a la enseñanza a través de este entorno, como presentaciones, modelos de evaluación y plantillas que permiten hacer demostraciones. No es una solución gratuita, lo que puede dificultar a aquellos centros educativos más modestos desde el punto de vista económico ("CoderZ," 2020; "Educators - CoderZ," 2020). La Fig. 2 muestra la interfaz proporcionada por esta herramienta.

Fig. 2. CoderZ

OpenRoberta

OpenRoberta (Jost, Ketterl, Budde, & Leimbach, 2015; "Open Roberta Lab," 2020) es un entorno web muy completo para programar diferentes tipos de dispositivos, incluyendo algunos robots bien conocidos como el mBot. Utiliza un lenguaje de programación visual propio basado en bloques, llamado NEPO, e integra un entorno de simulación en dos dimensiones.

Dash

A priori podría parecer un simple juguete para niños (Fig. 3), pero este producto proporciona un número significativo de posibilidades. Cuenta con una gran cantidad de sensores con los que puede moverse y sortear obstáculos. Además, permite su programación a través de dispositivos móviles con iOS o Android por medio de Blockly, lo cual lo hace accesible desde edades tempranas. Es una buena solución para el hogar, a pesar de que el precio sea algo elevado.

Fig. 3. Dash

Contribuciones

En este apartado nos centraremos en enumerar las distintas aportaciones que proporciona la herramienta presentada en este trabajo respecto al resto de recursos disponibles en la actualidad:

- **Es gratuita**. Resulta muy complicado implementar soluciones educativas de este tipo en los centros educativos, ya que la mayoría no disponen de los recursos económicos suficientes para adquirirlas. Este hecho provoca que solo unos pocos centros puedan disponer de este tipo de herramientas. Nuestra principal motivación es que gran parte del alumnado debería poder acceder a este tipo de herramientas, independientemente del presupuesto del centro educativo, así como de la situación económica de sus familias. Además, proporcionar herramientas gratuitas facilita enormemente incluir contenidos relacionados en los planes de estudios.
- **Es código abierto**. La gran mayoría de proyectos de software educativo han sido desarrollados con fines comerciales, por lo que el código o cualquier recurso relacionado con la herramienta suele ser privado. Este proyecto ha sido creado sin ánimo de lucro, por lo que se encuentra registrado bajo una licencia de código

abierto, permitiendo que otros desarrolladores que estén interesados puedan aprovechar, modificar o mejorar la herramienta.

- **Es simple**. Por lo general, estas herramientas de simulación suelen ser bastante complejas, tanto por las funcionalidades que ofrecen, como por la interfaz proporcionada, estableciendo barreras para aquellos usuarios más inexpertos. En el caso que nos ocupa, ofrecemos una simplificación de los simuladores convencionales.

DISEÑO Y DESARROLLO DEL SIMULADOR

En primer lugar, en la Sección 3.1 se lleva a cabo una breve introducción a las tecnologías y recursos utilizados para implementar la herramienta de simulación propuesta, denominada *UBlockly-Robot* (De La Paz González, 2018; Manuel & Díaz, 2019; Rivarés, 2018; *Roblockly Source Code*, 2020). A continuación, en la Sección 3.2, se describen las principales funcionalidades que proporciona el simulador, especificando detalles acerca de las soluciones técnicas adoptadas para llevar a cabo la implementación de las mismas.

Tecnologías y recursos utilizados

En las siguientes secciones se describirán las principales tecnologías utilizadas para la implementación del simulador, el motor gráfico *Unity* y la implementación de la librería Blockly para Unity, denominada *UBlockly*.

Unity

Unity ("Unity," 2020) es un motor gráfico desarrollado por Unity Technologies. Es uno de los motores más usados a nivel mundial para la creación de contenido interactivo tanto 2D como 3D, generalmente, videojuegos. Está disponible para la gran mayoría de sistemas operativos y los proyectos pueden ser exportados a una gran lista de plataformas. Entre las más destacadas podemos nombrar los dispositivos móviles, las videoconsolas de última generación y los principales navegadores Web.
Para este proyecto se ha utilizado la versión Unity Personal 2019.1.0f2 (64-bit) de Unity3D, ya que para la simulación del robot se requería de un entorno tridimensional en el que se pudiera representar las posibles colisiones que pudiera tener el robot con los elementos del entorno.

En Unity3D, la programación se puede realizar a través de los lenguajes C# y JavaScript. Además, cuenta con un editor visual muy potente e intuitivo que agiliza el desarrollo de los proyectos, pudiendo crear con facilidad entornos 3D propios, o importar recursos creados por otros desarrolladores de la comunidad a través de su *Asset Store*. Estos recursos van desde modelos 3D, hasta texturas o efectos de sonido.

A continuación, se explican algunos de los recursos más importantes de Unity:

- **GameObject**: Es el elemento principal de Unity. Dichos elementos se encuentran contenidos dentro de las escenas ("Escenas - Unity Manual," 2020). Como tal, el *GameObject* es un elemento vacío, que de forma nativa solo cuenta con un componente que define su posición, rotación y escala. Debemos añadirle otros componentes para cambiar las propiedades y el comportamiento de este en la escena.

Un *GameObject* podría ser desde una piedra o un árbol, hasta una luz que ilumine los elementos de la escena.

- **Componente**: Son aquellos elementos que al añadirse a un *GameObject* modifican sus propiedades. Estos son los algunos de los componentes más importantes:
    - **Scripts**: Por medio de la codificación de scripts podremos tanto alterar parámetros de nuestro objeto como modificar el comportamiento de éste en la escena como, por ejemplo, haciendo que responda con una determinada acción tras la pulsación de una tecla ("Scripts - Unity Manual," 2020).
    - **Transform**: Es un componente obligatorio para cualquier *GameObject*. Sirve para localizarlo dentro de la escena, además de para cambiar su rotación y su escala.
    - **Mesh**: Este componente permite definir la malla de puntos con la que se establece cómo será renderizado el *GameObject* en la escena.
    - **Collider**: Con este componente se puede configurar una malla de colisión para un *GameObject*. Si dos mallas de este tipo se encuentran, se puede indicar que ha habido contacto entre ellas. Cabe destacar que la malla de colisiones es totalmente independiente a la malla de renderizado de un objeto.
    - **Rigidbody**: Al añadir un componente de este tipo a un objeto, se indica que se encuentra bajo el control de las físicas del motor 3D. Esto quiere decir que los *GameObjects* que tengan dicho componente se verán influenciados por la gravedad y otro tipo de fuerzas, que pueden ser, tanto fruto de una colisión con un elemento de la escena, como con su aplicación a través de un script ("Rigidbody - Unity Manual," 2020).
- **Escena**: Contiene los entornos y menús creados a base de *GameObjects*. Cada escena debe contener al menos una cámara para poder visualizar lo que esté pasando durante la ejecución del programa y una luz direccional para iluminar el entorno.
- **Prefab**: Tras haber creado y modificado un *GameObject*, existe la posibilidad de guardar un modelo del mismo con todas sus propiedades. Este modelo puede ser usado posteriormente para ser instanciado en las escenas de un proyecto. Además, si modificamos un *prefab*, automáticamente todas las instancias que existan de este tendrán los cambios que hemos añadido a nuestras escenas ("Prefabs - Unity Manual," 2020).

UBlockly

UBlockly ("UBlockly," 2020) es una reimplementación de Google Blockly ("Blockly," 2020) desarrollada con Unity. Permite implementar en Unity, de una forma sencilla y eficiente, bloques Blockly para definir un lenguaje de programación visual propio.

En un inicio se barajó la posibilidad de utilizar Google Blockly directamente. Finalmente, se descartó esta posibilidad, ya que surgieron varios problemas e inconvenientes:

- La versión Web requiere un complemento de terceros.
- No se admiten funciones de Unity como las corrutinas.
- La flexibilidad de diseño de interacción de la interfaz de usuario (UI) es baja.
- El coste en lo que respecta al tiempo era bastante elevado y, además, sin garantías de éxito.

Por estos motivos se decidió investigar hasta finalmente encontrar UBlockly. Esta librería consiguió ahorrar tiempo de desarrollo y hacer que el proyecto fuera más sólido, ya que todo estaría desarrollado únicamente con tecnología Unity. UBlockly cuenta con tres módulos que se comunican entre sí. Estos son *Code, Model* y *UI* (Fig. 4).

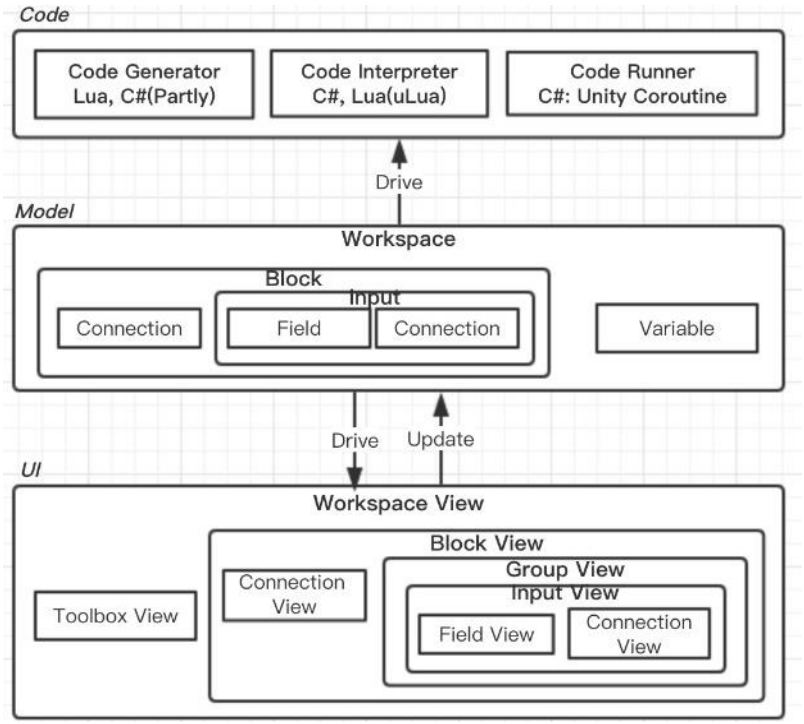

Fig. 4. Esquema de funcionamiento de UBlockly

El módulo *Code* es el núcleo de funcionamiento de UBlockly. Tras la creación de un script visual, permite generar el código C# fruto de la unión de los bloques apilados en el área de trabajo. Ya que C# es un lenguaje estático que no permite generar código de forma dinámica, requiere de la interpretación el código fuente asociado al bloque.

El módulo *Model* está constituido por el espacio de trabajo que contiene las variables y los bloques. Las variables son globales a todo el espacio de trabajo y los bloques representan a un programa ejecutable, que puede tener (o no) una salida, de forma análoga a una función en programación tradicional. Un bloque debe tener, al menos, un tipo de conexión para que se pueda conectar con otro bloque. Estas conexiones pueden ser de entrada, salida, anterior y posterior. Las conexiones entrada–salida permiten que un bloque reciba un parámetro a través de otro bloque, que, tras su ejecución, genera un valor de salida. Por otro lado, las conexiones de tipo anterior–posterior indican el orden de ejecución de estos. Los bloques son definidos en ficheros JSON con su estructura y propiedades correspondientes.

El módulo *UI* se ocupa de gestionar toda la parte visual con la que interactuará el usuario. Este módulo se encarga de crear los bloques a partir de la estructura definida en los ficheros JSON, o de modificar el tamaño de estos en el área de trabajo en tiempo de ejecución, si fuera necesario, entre otras funciones.

Por defecto, UBlockly proporciona varias categorías de bloques, como los bloques condicionales o bloques de repetición (bucles). En el caso que nos ocupa, para complementar los bloques proporcionados por defecto, se implementaron bloques específicos orientados a la ejecución de las acciones (movimiento) y a la recopilación de información del entorno de simulación (sensores) del robot.

En la Fig. 5 y la Fig. 6, se ilustra un pequeño ejemplo de cómo se crea un bloque en UBlockly:

- **Type** Nombre del bloque a implementar.
- **Message0**. Constituye el mensaje que verá el usuario escrito dentro del bloque. Puede contener menciones a diccionarios.
- **Args**. Es la sección en la que se definen los argumentos del bloque. Existen varios tipos de campos para los distintos tipos de datos: numérico, booleano, cadena de caracteres o listas.
- **PreviousStatement y NextStatement**. Permiten especificar si el bloque tiene conexiones de tipo anterior—posterior.
- **Output**. Permiten indicar si el bloque devuelve un valor y su tipo.

```
{
    "type": "move_robot_forward",
    "message0:"%{BKY_MOVE_ROBOT}
                %{BKY_MOVE_ROBOT_FORWARD}
                con velocidad %1",
    "args0": [
        {
            "type": "field_number",
            "name": "DISTANCE",
            "value": 1,
            "min": 1,
            "max": 5,
            "int": true
        }
```

Fig. 5. Fichero de definición de bloques

```
"MOVE_ROBOT": "Mover robot",
"MOVE_ROBOT_FORWARD": "hacia delante",
```

Fig. 6. Fichero de diccionario

La Fig. 7 ilustra el bloque resultante de las definiciones establecidas en la Fig. 5 y Fig. 6.

Fig. 7. Ejemplo de bloque

Principales funcionalidades

En esta sección se describen las principales funcionalidades que proporciona la herramienta de simulación UBlockly-Robot. Tal y como ilustra la Fig. 8, existen dos bloques de funcionalidades bien diferenciados: funcionalidades relacionadas con la creación o construcción del robot y funcionalidades relacionadas con su simulación.

Respecto a la creación, se puede editar el robot personalizando sus sensores. Por una parte, se proporciona la posibilidad de añadir un nuevo sensor, donde se deberá escoger su tipo y ubicación en el robot. Se puede elegir entre sensores de infrarrojos, ultrasonidos y de contacto. Una vez seleccionado el tipo, se podrán especificar diferentes parámetros, como pueden ser su precisión o distancia de medición.

Al mismo tiempo, se permite editar un sensor que ya haya sido creado, seleccionándolo de la estructura del robot, tras hacer clic con el ratón sobre el mismo. Con dicha acción se podrán ajustar los parámetros del sensor, tal y como se ha explicado anteriormente. También se puede seleccionar un sensor previamente añadido para borrarlo.

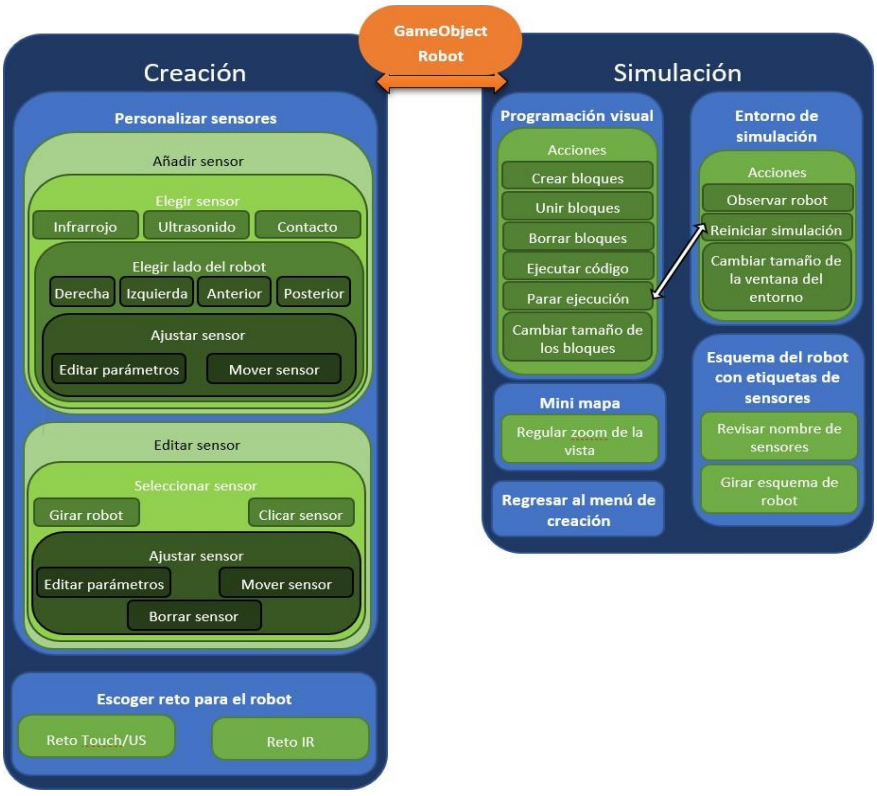

Fig. 8. Esquema de funcionalidades de UBlockly-Robot

Una vez construido el robot, se puede seleccionar uno de los retos disponibles. Una vez seleccionado el reto, se cargará la escena de simulación, en la que podremos ver varios elementos en pantalla. Uno de los más destacables es el área de trabajo para la programación visual, en la que será posible definir el comportamiento del robot apilando los distintos tipos de bloques (lógica, sensores y acción), también gracias a la ayuda de un pequeño esquema del robot que permite identificar con facilidad los sensores que han sido añadidos.

Existen algunas funcionalidades en este simulador que enriquecen la experiencia del usuario como, por ejemplo, la posibilidad de cambiar el tamaño de los bloques del área de trabajo o de la ventana del entorno de simulación. No obstante, una de las más interesantes es la capacidad de reiniciar la simulación y detener la ejecución de los programas, lo que permite a los usuarios una mayor interacción a la hora de desarrollar y corregir sus programas.

Por último, desde el entorno de simulación se puede volver a la sección de construcción del robot para modificarlo, tanto en el caso de que el reto seleccionado haya sido superado y se desee construir un nuevo robot adaptado para el siguiente reto, como para corregir su diseño actual.

Diseño de bloques específicos con UBlockly

Una de las tareas más importantes a la hora de desarrollar la herramienta de simulación, fue la implementación de los bloques específicos de movimiento y recopilación de información mediante sensores.

La primera categoría de bloques que se implementó fue la de movimiento. Esta consiste en las acciones que el robot es capaz de desempeñar: desplazarse hacia delante, hacia atrás, girar y detenerse.

Tal y como se describió con anterioridad, para definir estas acciones, se tiene que editar un conjunto de ficheros. Uno de los ficheros define los bloques pertenecientes, en este caso, a la categoría de movimiento. Otro fichero contiene un diccionario con los mensajes que muestran los bloques pertenecientes a la categoría (Fig. 6). El último establece las propiedades específicas para cada bloque de movimiento donde las más importantes, tal y como se ha indicado previamente, son si el bloque en cuestión toma valores de entrada y de qué tipo, el nombre del bloque, el mensaje que muestra, y su color (Fig. 5).

Tras editar los ficheros correspondientes, UBlockly dará como resultado un *prefab* del bloque perteneciente a la categoría de movimiento (véase la Fig. 9).

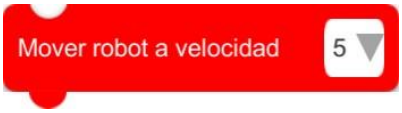

Fig. 9. Ejemplo de bloque de movimiento

Para finalizar la creación del bloque, tan solo restaría asociarle un fragmento de código para poder definir su comportamiento, tal y como ilustra la Fig. 10. En este caso, el código fuente se traduce en aplicar una fuerza al robot que permite su desplazamiento hacia adelante a partir de su posición actual.

```
public IEnumerator DoMove(int distance){

        robot.GetComponent<Rigidbody>().AddForce(Vector3.forward * distance * 100);
        yield return null;
```

Fig. 10. Fichero de definición de comportamiento

A continuación, se enumeran las distintas acciones (bloques) que conforman la categoría de movimiento:

- Avance y retroceso con velocidad variable durante un tiempo predefinido.
- Avance y retroceso con velocidad variable durante un tiempo indefinido.
- Giro hacia la derecha o izquierda con grados ajustables.
- Giro hacia la derecha o izquierda de manera indefinida.
- Detención o parada.

En el siguiente ejemplo se observa la definición de la función de avance del robot, en la que aprovechamos las propiedades de los componentes *HingeJoint* para hacer que las ruedas desarrollen movimiento por sí mismas (Fig. 11). Estos componentes unen dos objetos por medio de un elemento rotativo, de modo que, si unimos las ruedas y el robot, al girar la rueda, el robot se desplazará.

```
public IEnumerator DoMoveForward(int distance,int time){
    robot_rb.constraints = RigidbodyConstraints.FreezeRotation;
    wheel_r_rb.drag = 5; wheel_l_rb.drag = 5; wheel_c_rb.drag = 5;
    JointMotor motor_r,motor_l,motor_c;
    motor_r = wheel_r.motor; motor_l = wheel_l.motor; motor_c = wheel_c.motor;

    motor_r.targetVelocity = 300*distance;
    motor_l.targetVelocity = -300*distance;
    motor_c.targetVelocity = -300*distance/3;

    motor_r.force = 200*distance;
    motor_l.force = 200*distance;
    motor_c.force = 200*distance/3;

    wheel_r.motor = motor_r; wheel_l.motor = motor_l; wheel_c.motor = motor_c;
```

Fig. 11. Código asociado al bloque de movimiento que permite avanzar

El código fuente anterior sirve tanto para movimiento de avance indefinido como para avanzar durante un periodo de tiempo predefinido. El código fuente que permite al robot retroceder sería exactamente igual, con la salvedad de que el signo de la velocidad de las ruedas pasa a ser negativo.

En cuanto al desarrollo del código fuente que implementa los giros del robot, se aprovechó que cada rueda tiene asociada un motor independiente. Por ejemplo, si se hace que la rueda derecha gire hacia delante y la izquierda hacia atrás, se consigue un giro del robot hacia la izquierda.

En concreto, el fragmento de código de la Fig. 12 establece la condición de parada de un giro hacia la izquierda, el cual es similar al que implementa la condición de parada de un giro hacia la derecha.

```
...//Accionamiento de las ruedas
while (girando & angle != MAX_VALUE){
        if                  (final_angle                  ==
            robot.GetComponent<Transform>().eulerAngles.y)   {
            girando= false;
        }
        if ((girando) & (final_angle <= epsilon || final_angle >= 360-epsilon){
            if (initial_angle > final_angle){ //Ángulo final cercano a 0 por la derecha
                if (robot.GetComponent<Transform>().eulerAngles.y  -  epsilon  <=
                    final_angle){ girando = false;
            }}
            else{ //Ángulo final cercano a 0 por la izquierda
                if (robot.GetComponent<Transform>().eulerAngles.y  -  epsilon  >=
                    final_angle){ girando = false;
        }}}
        if  ((girando)  &  (robot.GetComponent<Transform>().eulerAngles.y  <
           final_angle)  &  (robot.GetComponent<Transform>().eulerAngles.y  <=
           initial_angle) & (!inverted))
        else{
            if  ((girando)  &  (robot.GetComponent<Transform>().eulerAngles.y  >
                final_angle)){ inverted = true;
            }
            else{
```

Fig. 12. Condiciones de parada del giro a izquierda

Las condiciones de parada únicamente se activarían en el caso en el que queramos girar una cantidad determinada de grados. Si el giro no tiene un ángulo final predefinido, el robot seguirá girando a la izquierda hasta que algún otro bloque haga que se detenga.

Respecto al código de detención del robot, tiene una implementación bastante simple, ya que lo que se hace es desactivar el motor y aumentar el rozamiento de las ruedas hasta que no puedan seguir rotando (Fig. 13).

```
    public IEnumerator DoStop(){
        robot_rb.constraints = RigidbodyConstraints.FreezeRotation;
        wheel_r.useMotor = false; wheel_r_rb.drag = 9999999; wheel_l.useMotor
        =  false;  wheel_l_rb.drag  =  9999999;  wheel_c.useMotor  =  false;
        wheel_c_rb.drag = 9999999; yield return new WaitForSeconds(1);
```

Fig. 13. Código de detención de movimiento

Al concluir la definición del comportamiento de los bloques, y por supuesto, haber creado la estructura de estos debidamente para crear sus *prefabs*, obtendríamos los bloques que podemos ver en la Fig. 14.

Fig. 14. Repertorio de bloques pertenecientes a la categoría de movimiento

Existe una segunda categoría de bloques, relacionados con la recopilación de información del entorno de simulación a través de los sensores del robot. En concreto, se han implementado bloques para recoger información a través de los diferentes tipos de sensores existentes: infrarrojos, ultrasonido y contacto.
La principal diferencia con los bloques de la categoría de movimiento es que los bloques de sensores deben retornar la información recopilada del entorno para poder condicionar las acciones del robot.

Después de definir su comportamiento y estructura, obtendremos el repertorio de bloques pertenecientes a la categoría de sensores (Fig. 15). Tal y como puede observarse, y al existir la posibilidad de añadir varios sensores de cada uno de los tipos a un robot, los bloques permiten la selección de uno de dichos sensores a través de un identificador numérico. La Fig. 16 muestra la paleta de bloques completa en el entorno de simulación que permite programar el comportamiento del robot.

Fig. 15. Repertorio de bloques pertenecientes a la categoría de sensores

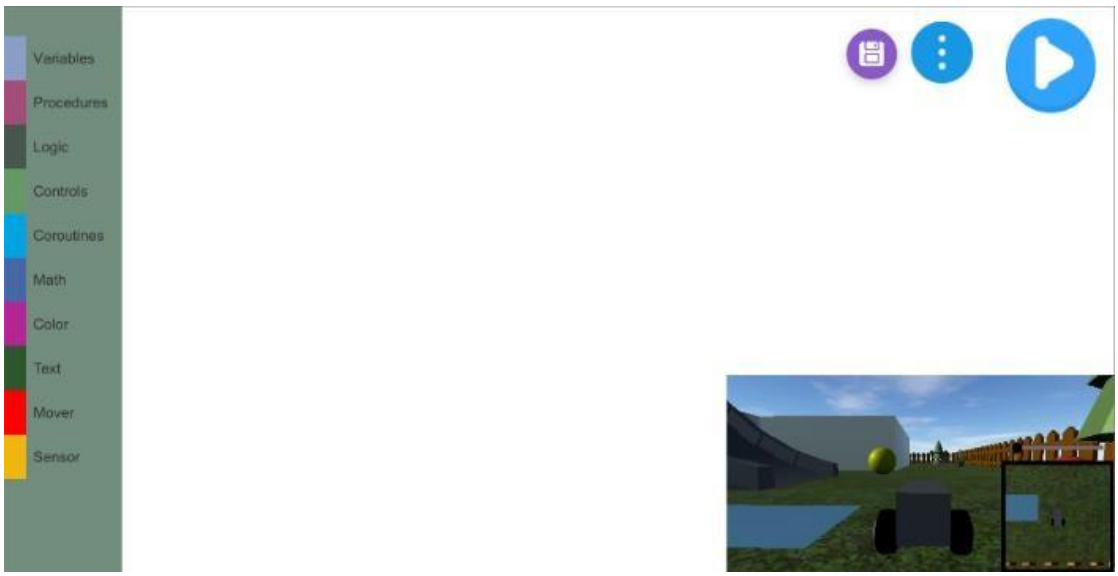

Fig. 16. Escena de simulación que muestra las diferentes categorías de bloques a la izquierda

Funcionalidades adicionales e interfaz de usuario

En este apartado se detallan algunas características adicionales del simulador, a nivel funcional y a nivel de interfaz, que mejoran la experiencia de usuario.

Una de estas funcionalidades consiste en mostrar las etiquetas de los sensores al usuario, para que pueda distinguirlos fácilmente, sobre todo aquellos del mismo tipo. Esta información se muestra tanto durante la construcción del robot, como en el entorno de simulación (Fig. 17). Además, al seleccionar un sensor para editarlo, el color de su etiqueta cambia, lo cual facilita identificar qué sensor se está modificando.

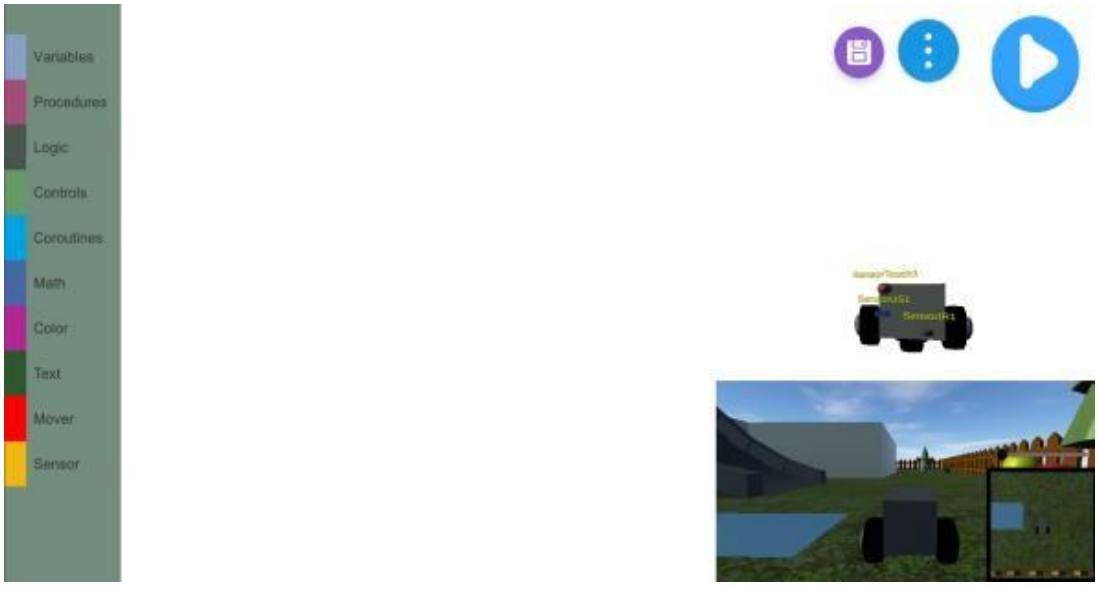

Fig. 17. Esquema del robot en la escena de simulación en la parte central derecha de la interfaz

Con el fin de hacer más sencilla la tarea de recordar el diseño concreto del robot llevado a cabo, el entorno de simulación cuenta con una miniatura del mismo, tal y como ilustra la Fig. 17. Lo anterior permite, por ejemplo, no solo conocer el tipo y número de sensores añadidos durante la fase de programación del robot, sino también identificar cada uno de los sensores añadidos a través de sus correspondientes etiquetas. La miniatura del robot puede visualizarse desde diferentes ángulos si se pulsan las teclas de las flechas del teclado.

Con el fin de mejorar la visualización del entorno de simulación se ha añadido un botón que permite mostrarlo a pantalla completa, tal y como se ilustra en la Fig. 18.

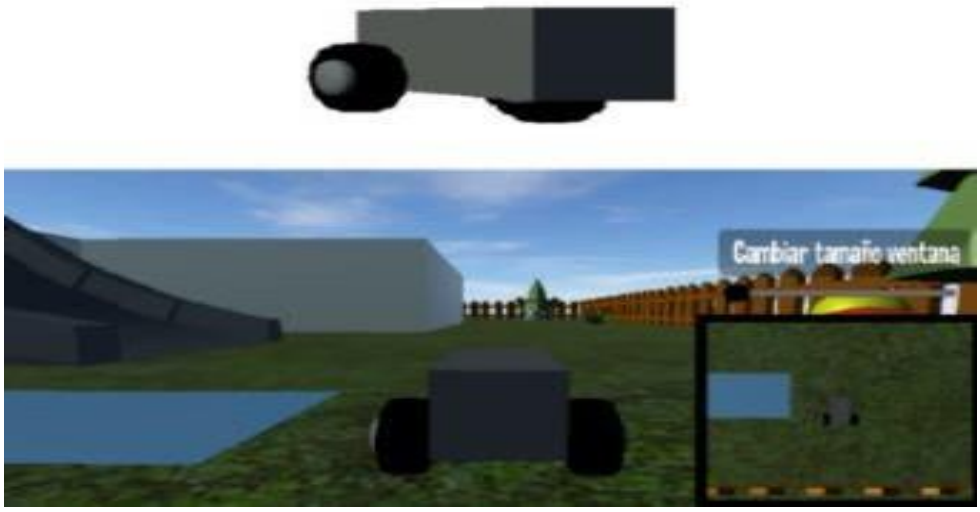

Fig. 18. Botón de cambio de tamaño de ventana situado encima de la vista cenital de la escena

También se ha incorporado la posibilidad de reiniciar la simulación del robot. De este modo, si el usuario se ha equivocado a la hora de programar su comportamiento, al pulsar el botón de reinicio, se detendrá la simulación y se llevará al robot hasta el punto de inicio, conservando los bloques previamente utilizados en el área de trabajo (Fig. 19).

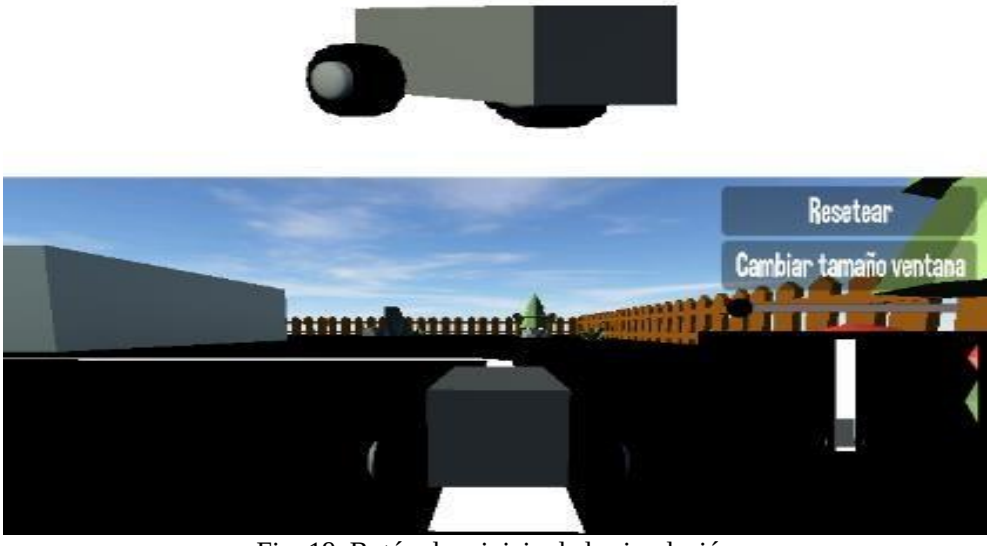

Fig. 19. Botón de reinicio de la simulación

El usuario, por equivocación, podría tener que modificar el diseño de su robot, por lo que se ha incluido un botón en el entorno de simulación que permite volver al entorno de construcción.

Además, en la paleta de bloques, solo se muestran aquellos bloques asociados a sensores que se hayan utilizado en la construcción del robot. Lo anterior quiere decir que si, por ejemplo, no añadimos ningún sensor de contacto, los bloques asociados a dicho tipo de sensores no estarán disponibles para su uso en la paleta de bloques. En la Fig. 20 se ilustra lo anterior.

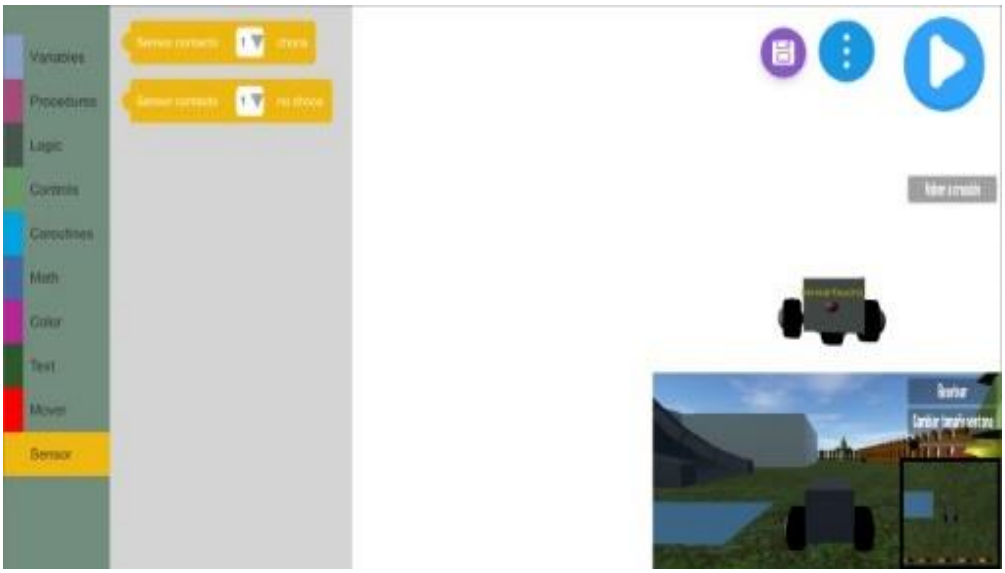

Fig. 20. Categoría de sensores con los bloques no permitidos ocultos

Por último, a pesar de poder arrastrar el cursor para poder ver la totalidad de nuestro programa en el área de trabajo, cabe la posibilidad de que dicha tarea se vuelva incómoda en el caso de que existan muchos bloques en la misma. Por ello, se han incluido unos botones en la esquina inferior izquierda por medio de los cuales se puede aumentar y disminuir el tamaño de los bloques que se encuentran en el área de trabajo (Fig. 21).

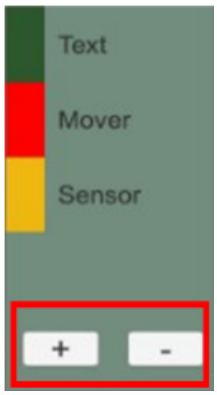

Fig. 21. Botones para aumentar/disminuir el tamaño del programa del área de trabajo

Creación de retos

En la actualidad, UBlockly-Robot proporciona dos retos predefinidos, aunque no conllevaría un trabajo excesivo añadir nuevos retos en la herramienta.

A través de estos retos se pueden trabajar habilidades relacionadas con el pensamiento computacional y que ya han sido mencionadas con anterioridad como la descomposición de problemas o el pensamiento algorítmico. La descomposición de problemas surge desde que se le plantea el reto al alumnado ya que, por un lado, se ha decidir qué sensores forman del robot y su posicionamiento y, por otro lado, se debe diseñar un algoritmo que, haciendo uso de los sensores seleccionados, permita resolver el reto planteado.

Uno de los retos se encuentra destinado a trabajar con los sensores de infrarrojos, consistente en que el robot haga uso de dicho tipo de sensor para seguir una línea blanca hasta alcanzar una moneda. Un sensor de infrarrojos permite discriminar entre dos colores: blanco y negro. Teniendo en cuenta lo anterior, se ideó un reto consistente en un entorno con un camino blanco sobre un fondo negro. Para superar el reto, se debe aprovechar la información que proporcionan los sensores de infrarrojos en cada momento para tratar de alcanzar la moneda situada al final del camino blanco.

Como se puede observar en la Fig. 22**,** el reto cuenta con un camino que tiene tanto rectas como curvas, por lo que el usuario deberá hacer buen uso de los sensores de infrarrojos para superar este reto.

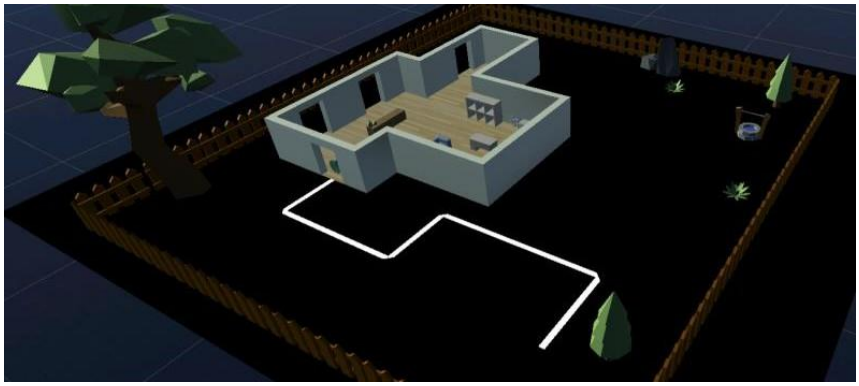

Fig. 22. Entorno de simulación del reto para los sensores de infrarrojos

Fig. 23. Configuración del robot en el reto para los sensores de infrarrojos

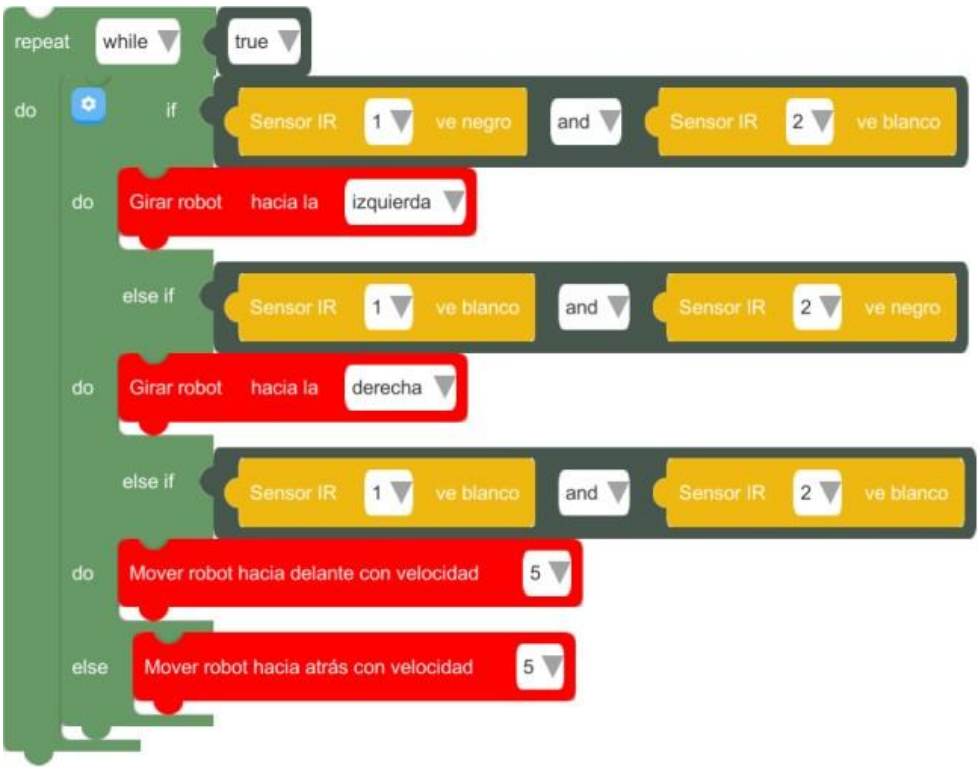

Fig. 24. Programa que permite superar el reto para los sensores de infrarrojos

Una posible solución consistiría, en primer lugar, en la creación de un robot que cuente con dos sensores de infrarrojos. Los sensores deben estar situados en la parte delantera del robot, uno en cada extremo, y lo más cerca posible al suelo, tal y como se muestra en la Fig. 23.

A continuación, se debe programar el comportamiento del robot a través de un programa similar al que se muestra en la Fig. 24. Dicho programa permite que el robot pueda, en cada momento, llevar a cabo una de las cuatro acciones enumeradas a continuación:

1. Si ambos sensores detectan el color blanco, el robot **avanza**.
2. Si el sensor situado a la derecha detecta el color negro y el sensor ubicado en la izquierda del robot detecta el color blanco, el robot realizará un **giro hacia la izquierda**.
3. De un modo similar al anterior, si el sensor situado a la izquierda detecta el color negro, mientras que el sensor derecho detecta el color blanco, el robot realizará un **giro hacia la derecha**.
4. Si ambos sensores detectan el color negro, el robot **retrocederá**.

El objetivo del segundo reto es la utilización de los sensores, o bien de contacto, o bien de ultrasonido, para que el robot navegue por un laberinto hasta capturar una moneda. Un sensor de contacto permite indicar, mediante un valor booleano (verdadero o falso), si el mismo ha colisionado con un objeto. Al mismo tiempo, un sensor de ultrasonidos proporciona información acerca de la distancia a la que se encuentra un posible obstáculo.

Teniendo en cuenta lo anterior, se diseñó un reto que pudiera solucionarse usando alguno de estos dos tipos de sensores. El objetivo final de este reto que el robot sea capaz de navegar por un laberinto hasta alcanzar una moneda situada en su salida.

Tal y como puede observarse en la Fig. 25 y Fig. 26, el camino cuenta con varias curvas y paredes que dificultan el recorrido del robot. Para resolver este reto, será necesaria la creación de un robot similar a los mostrados en la Fig. 27 y Fig. 28, en el caso de seleccionar el sensor de contacto o el sensor de ultrasonidos, respectivamente.

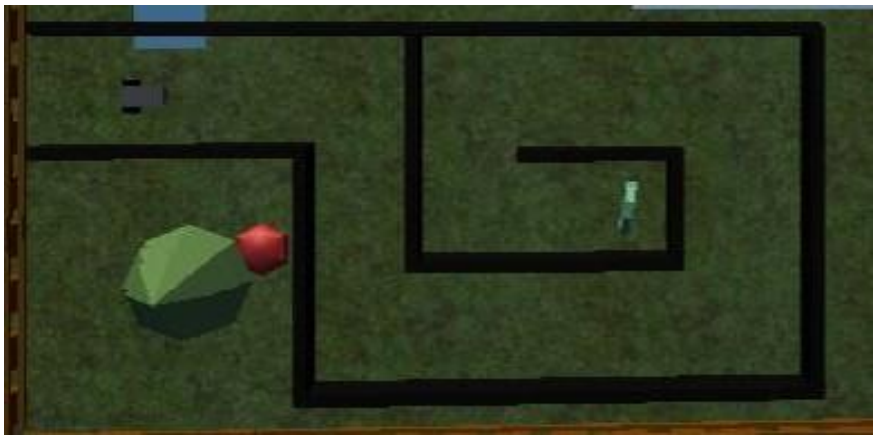

Fig. 25: Vista cenital del reto para los sensores de contacto y ultrasonidos

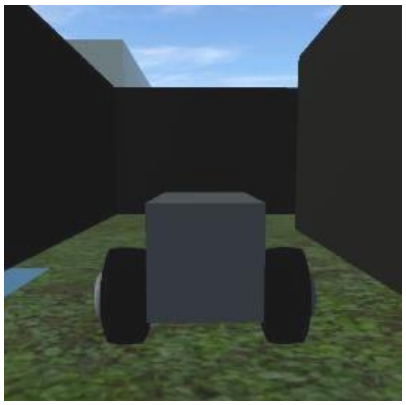

Fig. 26: Vista tridimensional del reto para los sensores de contacto y ultrasonidos

Fig. 27. Robot construido con un sensor de contacto

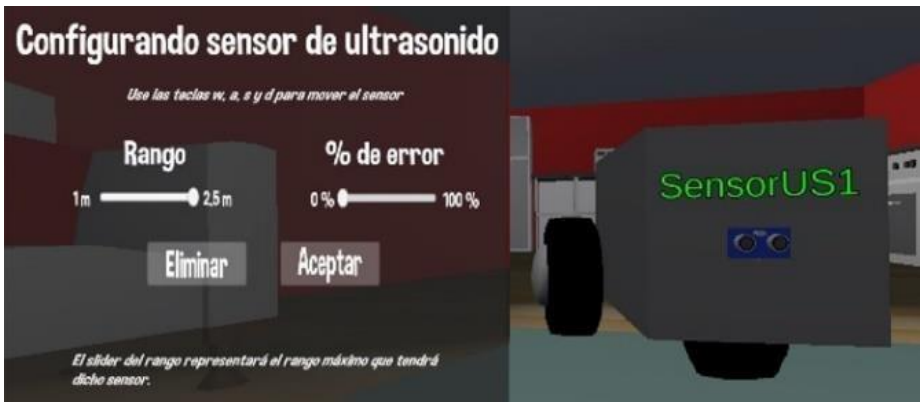

Fig. 28. Robot construido con un sensor de ultrasonidos

La Fig. 29 y la Fig. 30 ilustran posibles programas que permiten resolver con éxito el reto planteado. Ambos son bastante similares en cuanto a las condiciones con las que cuentan, ya que, para ambos casos, hay que realizar el mismo recorrido, que consiste en girar a la **derecha 90 grados una única vez** y girar a la **izquierda 90 grados cinco veces**. No obstante, mientras que el robot con el sensor de contacto debe avanzar siempre y cuando no exista contacto, el robot con el sensor de ultrasonidos avanza hasta que la distancia entre la pared y el mismo sea inferior a cierto umbral preestablecido.

Fig. 29. Programa asociado al robot con sensor de contacto

Fig. 30. Programa asociado al robot con sensor de ultrasonidos

CASO DE USO: RETO DEL SENSOR DE ULTRASONIDOS

Al iniciar la herramienta ("Roblockly Simulator," 2020), se muestra el menú ilustrado en la Fig. 31.

Fig. 31. Menú de inicio

En primer lugar, se debe pulsar sobre la opción "Crear un robot" para acceder al entorno de construcción. Una vez aquí, se debe personalizar el robot para poder superar alguno de los dos retos propuestos. El caso de uso de esta sección está enfocado en la resolución del reto del sensor de ultrasonidos.

Para añadir el sensor de ultrasonidos, debemos dirigirnos a la parte izquierda de la interfaz, en la que tenemos las opciones para añadir sensores, y pulsar sobre el botón "Ultrasonido". Una vez hayamos pulsado este botón, aparecerán algunas opciones en la interfaz, que nos permitirán elegir la ubicación del sensor en la estructura del robot.

Fig. 32. Añadiendo un sensor de ultrasonidos a la parte frontal del robot

En este caso, vamos a pulsar sobre la opción "Parte frontal" (Fig. 32). Cuando pulsemos ese botón se nos desplegará un menú con distintas opciones para configurar el sensor. En este caso, se puede modificar el rango de detección de objetos del sensor y la precisión de aciertos que va a tener. Si se arrastran las barras situadas a la izquierda, se pueden alterar dichos parámetros. Para cambiar la ubicación del sensor, dentro de la parte frontal del robot, se pueden pulsar las teclas W, A, S y D del teclado.

En el caso que nos ocupa, se debe situar el sensor, aproximadamente, en el centro de la parte frontal con la configuración de precisión y rango que puede observarse en la Fig. 33. Una vez finalizada la configuración del sensor, se debe pulsar el botón "Aceptar".

Fig. 33. Configuración del sensor ultrasonido

Una vez se vuelva al entorno de construcción, se puede pulsar sobre el botón "Reto Touch/US" para acceder al entorno de simulación con el reto correspondiente cargado. En el caso de que se haya cometido algún error durante la configuración del sensor de ultrasonidos, pulsando sobre dicho sensor se podrá, tanto modificar su configuración, como eliminarlo (Fig. 34).

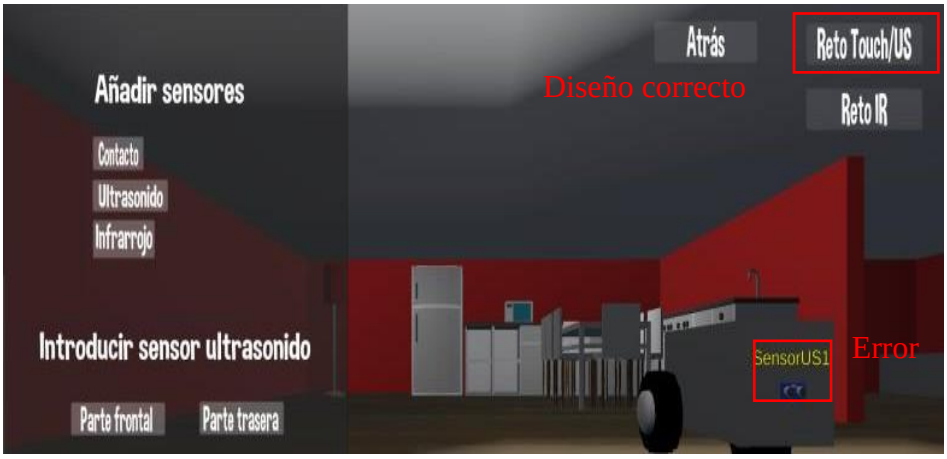

Fig. 34. Preparativos para iniciar el reto de sensor ultrasonido

Al pulsar sobre el botón del reto, se cargará la escena de simulación junto con el área de trabajo sobre la que se podrá editar el programa asociado al comportamiento del robot. Para comenzar a programar, se debe pulsar sobre alguna de las categorías de la paleta de bloques, momento en el que se desplegarán los bloques pertenecientes a dicha categoría. Acto seguido, se elegirá alguno de esos bloques y se arrastrará hasta el área de trabajo (Fig. 35).

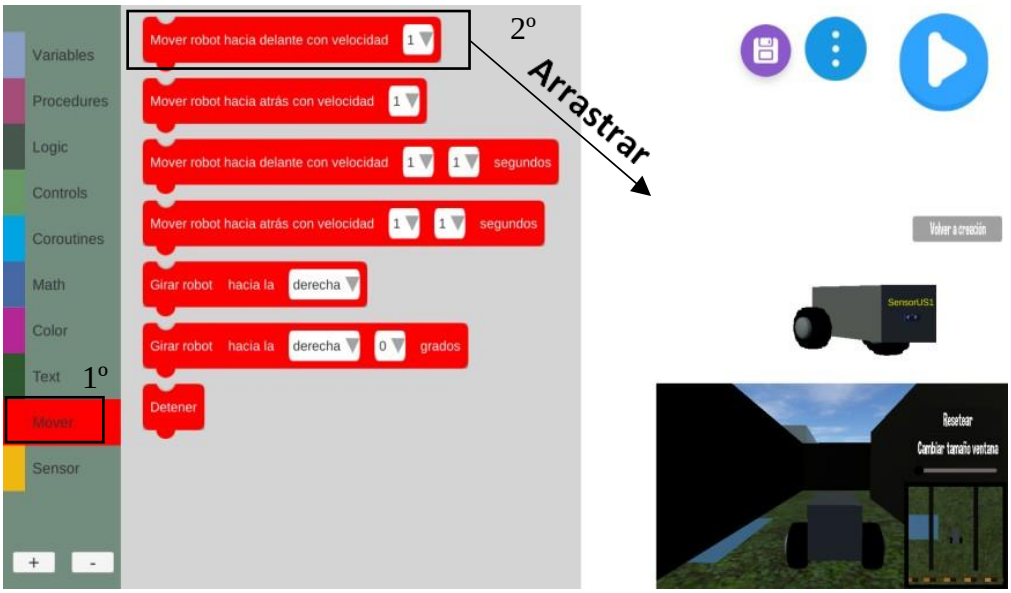

Fig. 35. Arrastrar un bloque al área de trabajo

Se deben ir arrastrando y apilando los bloques correspondientes hasta tener el programa que se muestra en la Fig. 30**.** Encadenar bloques es tan sencillo como arrastrar un bloque al área de trabajo y acercarlo a aquel bloque con el que queremos apilarlo. Cuando estén lo suficientemente cerca, las muescas por las que van a encajar los bloques cambiarán de color. Si en ese preciso momento soltamos el bloque que está siendo arrastrado, ambos bloques quedarán apilados (Fig. 36).

Fig. 36. Encajar dos bloques

Algunos bloques requieren de la especificación de algún parámetro (velocidad, ángulo de giro, dirección o identificador del sensor, entre otros), que puede ser una opción de entre un conjunto de posibles valores o un valor numérico. Cuando se pulsa sobre alguno de estos parámetros, aparecerá un cuadro de diálogo en el que se podrá especificar el valor del parámetro en cuestión. La Fig. 37 y la Fig. 38 ilustran lo anterior.

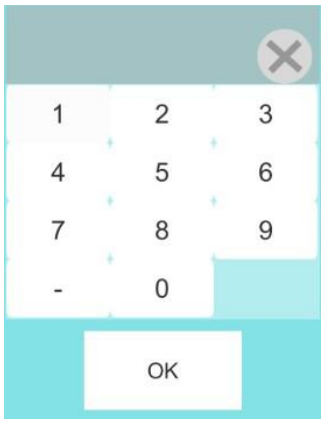

Fig. 37. Cuadro de diálogo: introducción de un número

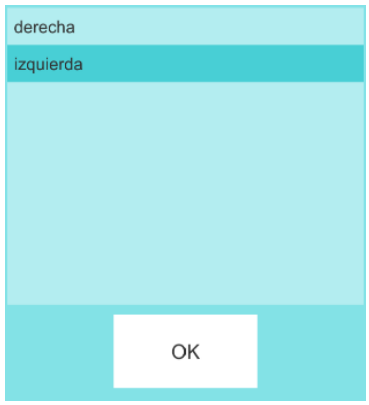

Fig. 38. Cuadro de diálogo: selección de una opción

Después de haber apilado el conjunto de bloques adecuado obtendremos un programa que permitirá al robot superar el reto propuesto. Para simular el comportamiento del robot en el entorno, se debe pulsar en el botón situado en la parte superior derecha. Una vez haya dado comienzo la simulación, un punto verde indica qué bloque se está ejecutando en cada momento. Además, también se puede observar la interacción del robot con el entorno de simulación.

Si en cualquier momento se desea reiniciar la simulación, se debe pulsar sobre el botón "Resetear". Por último, si se consigue superar el reto y se desea volver al entorno de construcción del robot, será suficiente pulsar sobre el botón "Volver a creación" y cambiar el diseño del robot para resolver otro reto. Todas las opciones anteriores quedan ilustradas en la Fig. 39.

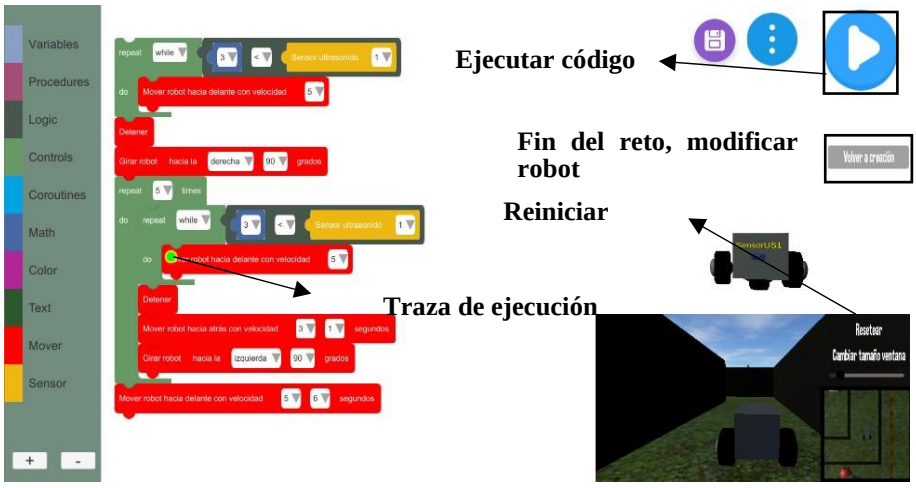

Fig. 39. Simulación y posibles opciones

CONCLUSIONES

A modo de conclusión general, se pueden dar por cumplidos los objetivos que se plantearon al inicio del desarrollo de la herramienta de simulación UBlockly-Robot. Se ha conseguido desarrollar un simulador gratuito, de código abierto y accesible, que fomenta el desarrollo de las principales habilidades relacionadas con el pensamiento computacional, a través del uso del pensamiento computacional como metodología didáctica. Gracias a que se trata de una herramienta software, que no requiere más de un navegador Web compatible y de una conexión a Internet, prácticamente cualquier centro educativo podría acceder a la misma, evitando al mismo tiempo, la inversión de recursos económicos, muchas veces no disponibles, en la adquisición de hardware o licencias software que suelen tener un elevado coste.

El presente trabajo sienta las bases de una herramienta que, desde nuestra modesta opinión, tiene mucho potencial de crecimiento, por lo que en futuras líneas de trabajo se podrían añadir más retos, además de catalogarlos por orden de dificultad, e incluso crear un módulo que permita la creación de nuevos retos, de manera fácil e intuitiva, por parte de los usuarios. Teniendo en cuenta lo anterior, sería un gran valor añadido que la herramienta formara parte de una comunidad educativa online, de modo que se pudieran compartir los diferentes retos creados por los usuarios, además de añadir algún sistema que permita puntuar y hacer clasificaciones según la destreza de los usuarios a la hora de resolver dichos retos compartidos.

Otra de las posibles mejoras que se podría desarrollar en el futuro serían la consideración de otros tipos de estructuras del robot que permitan diferentes distribuciones de las ruedas u otro tipo de elementos, como elementos móviles por medio de los cuales el robot pudiese recoger y transportar objetos del entorno de simulación. Lo anterior, además, podría dar bastante juego a la hora de plantear nuevos retos.

Presentación del artículo: 15 de enero de 2020
Fecha de aprobación: 21 de marzo de 2020
Fecha de publicación: 30 de abril de 2020

Ángel-Díaz, C. M., Segredo, E., Arnay, R. y León, C. (2020). Simulador de robótica educativa para la promoción del pensamiento computacional. *RED. Revista de Educación a Distancia, 20(*63). DOI: http://dx.doi.org/10.6018/red.410191

Agradecimientos
El presente trabajo se enmarca en el seno del Aula Cultural de Pensamiento Computacional de la Universidad de La Laguna. Por ello, quisiéramos agradecer la colaboración de todos los miembros involucrados.

Financiación
Esta investigación no ha recibido ninguna subvención específica de los organismos de financiación en los sectores públicos, comerciales o sin fines de lucro.

REFERENCIAS

Blockly. (2020). Retrieved March 9, 2020, from https://developers.google.com/blockly/

CoderZ. (2020). Retrieved March 9, 2020, from https://gocoderz.com/coding-robots/

CoppeliaSim. (2020). Retrieved March 9, 2020, from https://www.coppeliarobotics.com/

De La Paz González, E. (2018). *"Simulador de robots para fomentar el pensamiento computacional a través de lenguajes de programación visual" "Robot simulator for promoting the computational thinking through visual programming languages."*

Educators - CoderZ. (2020). Retrieved March 9, 2020, from https://gocoderz.com/educators/

Escenas - Unity Manual. (2020). Retrieved March 9, 2020, from https://docs.unity3d.com/es/current/Manual/CreatingScenes.html

Jost, B., Ketterl, M., Budde, R., & Leimbach, T. (2015). Graphical programming environments for educational Robots: Open Roberta - Yet another one? *Proceedings - 2014 IEEE International Symposium on Multimedia, ISM 2014*, 381–386. https://doi.org/10.1109/ISM.2014.24

Manuel, C., & Díaz, Á. (2019). *Robótica educativa y pensamiento computacional*.

Moreno, I., Muñoz, L., Serracín, J. R., Quintero, J., Patiño, K. P., & Quiel, J. (2012). La robótica educativa, una herramienta para la enseñanza-aprendizaje de las ciencias y las tecnologías. *Teoría de La Educación. Educación y Cultura En La Sociedad de La Información*, *13*(2), 74–90.

Open Roberta Lab. (2020). Retrieved March 9, 2020, from https://lab.open-roberta.org/

Ortega-Ruiperez, B., & Asensio Brouard, M. M. (2018). DIY robotics: computational thinking based patterns to improve problem solving. *Revista Latinoamericana De Tecnologia Educativa-Relatec*, *17*(2), 129–143. https://doi.org/10.17398/1695-288X.17.2.129

Perez Paredes, P., & Zapata Ros, M. (2018). El pensamiento computacional, análisis de una competencia clave. *New York: Create Space Independent Publishing*.

Prefabs - Unity Manual. (2020). Retrieved March 9, 2020, from https://docs.unity3d.com/es/current/Manual/Prefabs.html

Rigidbody - Unity Manual. (2020). Retrieved March 9, 2020, from https://docs.unity3d.com/es/current/Manual/class-Rigidbody.html

Rivarés, A. R. (2018). *Robótica Educativa mediante Programación Visual Educational Robotics through Visual Programming*.

Roblockly Simulator. (2020). Retrieved March 9, 2020, from https://computational-thinking.github.io/Roblockly/

Roblockly Source Code. (2020). Retrieved March 9, 2020, from https://github.com/Computational-Thinking/Roblockly-Source

Scripts - Unity Manual. (2020). Retrieved March 9, 2020, from https://docs.unity3d.com/es/current/Manual/CreatingAndUsingScripts.html

Try Computational Thinking. (2020). Retrieved March 13, 2020, from https://www.edsurge.com/news/2018-02-25-the-5th-c-of-21st-century-skills-try-computational-thinking-not-coding

UBlockly. (2020). Retrieved March 9, 2020, from https://github.com/imagicbell/UBlockly

Unity. (2020). Retrieved March 9, 2020, from https://unity.com/

Wing, J. M. (2008). Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, *366*(1881), 3717–3725. https://doi.org/10.1098/rsta.2008.0118

Zapata-Ros, M. (2015). Pensamiento computacional: Una nueva alfabetización digital. *Revista de Educación a Distancia*, (46).

Zapotecatl López, J. L. (2020). Capítulo 2: Abstracción. Retrieved March 9, 2020, from http://www.pensamientocomputacional.org/index.php/curso-pc/leccion-abstraccion