Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

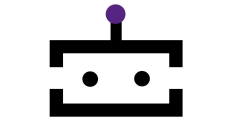
Trabajo de Fin de Grado

Grado en Ingeniería Informática

Reimplementación de Roblockly: mejorando y ampliando la herramienta de simulación de robótica educativa

Roblockly Reimplementation: improving and extending educational robotics simulation tool

Basilio Gómez Navarro



La Laguna, 8 de septiembre de 2021

D. **Eduardo Manuel Segredo González**, con N.I.F. 78564242Z profesor Ayudante Doctor adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor.

D. **Rafael Arnay del Arco**, con N.I.F. 78569591G profesor Contratado Doctor adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como cotutor.

C E R T I F I C A N

Que la presente memoria titulada:

*"*Reimplementación de Roblockly: mejorando y ampliando la herramienta de simulación de robótica educativa"

ha sido realizado bajo su dirección por D. **Basilio Gómez Navarro**, con N.I.F. 79096435-X.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 8 de septiembre del 2021

Agradecimientos

A mi familia, en especial a mi padre, mi madre y mi hermana, los cuales han sido un pilar fundamental con su cariño, confianza, apoyo incondicional y ayuda ante las adversidades surgidas durante la consecución de la carrera.

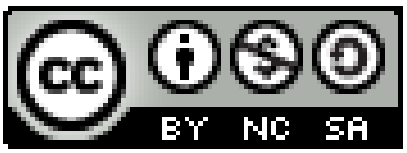
A mi novia y a mis amigos, por haberme acompañado y apoyado durante el transcurso de esta etapa, siendo siempre una fuente de ánimos, ayuda y cariño.

A mis compañeros y compañeras de carrera que me han tendido la mano siempre que lo he necesitado y con los que he compartido grandes experiencias a lo largo de estos años.

A Paula González Mora por la elaboración de la identidad corporativa y aplicación de la misma.

A Eduardo Manuel Segredo González y Rafael Arnay del Arco por toda la ayuda que me han brindado durante la elaboración de este TFG y la confianza que han depositado en mí para llevar a cabo dicho trabajo.

Licencia



Resumen

Vivimos en la era de la información en un mundo hiperconectado, repleto de computadores y dispositivos inteligentes que han cambiado por completo el modelo de vida con respecto al de hace un par de décadas. Es por esto, que desarrollar competencias como el *Pensamiento Computacional* [1] cada vez es más importante para el presente y el futuro. Así pues, para lograr esta tarea, en 2019 se inició un proyecto consistente en el desarrollo de una aplicación llamada Roblockly, pensada para ser utilizada principalmente en centros educativos de enseñanza obligatoria. Esta aplicación permite al usuario programar un robot virtual a través de un lenguaje de programación visual basado en bloques, con la finalidad de que posteriormente, este sea capaz de superar una serie de retos en un entorno simulado. Con esto se persigue fomentar el desarrollo del pensamiento computacional a través de la *Robótica Educativa* [2] y conseguir así que de una manera amena y lúdica, los estudiantes de dichos centros puedan ejercitar dicha competencia.

En este Trabajo de Fin de Grado se pretende ir más allá de la aplicación inicial, llevando a cabo una reimplementación de la misma y ampliando sus funcionalidades. Además de esto, se ha planteado el esbozo de un módulo de estadísticas que permita medir la evolución del usuario con el uso de la aplicación y por consiguiente, el desarrollo de su pensamiento computacional.

Palabras clave: Pensamiento computacional, robótica educativa, programación visual, Unity, simulador.

Abstract

We live in the information era, in a hyper-connected world, surrounded by computers and smart devices that have completely changed our lifestyles in the last two decades. Therefore, developing skills such as *Computational Thinking* [1] becomes increasingly more important nowadays and in the future. Roblockly, born in 2019, is a project that consists in the development of an application created to achieve this goal, conceived mainly to be used in secondary education or high schools.

This application allows the user to program a robot through a programming system based on block coding and overcome a series of challenges subsequently. This encourages the development of computational thinking through *Educational Robotics* [2] in students and enables them to achieve those skills in a playful manner.

This Bachelor's End of Degree Project intends to go beyond the initial application, reimplementing it and expanding its usefulness. Besides, the outline of a statistical module is set out to allow the evolution of the user of the application together with his or her development of computational thinking.

Key words: Computational thinking, educational robotics, visual programming, Unity, simulator.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1: Introducción

1.1 Motivación

La evolución tecnológica del último siglo, ha cambiado nuestro estilo de vida, pues ha afectado a todos los ámbitos de la sociedad, tanto en lo profesional como en lo personal. Consecuencia directa de esta evolución es la aparición del computador, que junto a la posterior llegada de internet y el surgimiento de dispositivos como teléfonos inteligentes, tabletas, televisores inteligentes, coches autónomos, etc., ha provocado que vivamos en lo que se conoce como la "Era de la información". El planeta entero está hiperconectado y esto ha ocasionado que se genere una cotidianidad tecnológica que precisa de ciertas destrezas y habilidades por parte del ser humano. Por ello, resulta evidente la importancia de que las personas desarrollen dichas competencias desde temprana edad, pues cada vez son y serán más importantes para el día a día. Entre estas competencias, se puede destacar el *Pensamiento Computacional*, que permite mejorar la creatividad, la capacidad de pensamiento crítico y estructurado, mejorar la capacidad de resolución de problemas y conseguir una relación mejor y más natural con el mundo que nos rodea [2].

Este TFG ha puesto el foco de atención en esta destreza mencionada previamente, motivado por la posibilidad de contribuir a la mejora y evolución de una aplicación llamada "Roblockly", que fue creada en 2019 y cuyo objetivo principal es fomentar el pensamiento computacional en estudiantes de la enseñanza obligatoria a través de la *Robótica Educativa* [2]. En concreto, consiste en una aplicación que permite a los usuarios simular la programación de un robot haciendo uso de un lenguaje de programación visual basado en bloques, para que posteriormente, este sea capaz de completar una serie de retos. Además, desde su origen, la aplicación ha mantenido la premisa de ser una herramienta de acceso gratuito online y de código abierto, de tal manera que para acceder a ella únicamente sea necesario un ordenador con conexión a internet, lo cual favorece las posibilidades de que cualquier centro educativo con unas infraestructuras informáticas básicas pueda acceder a dicha aplicación.

1.2 Objetivos

Este Trabajo de Fin Grado persigue llevar a cabo una refactorización del código, reimplementación y ampliación de funcionalidades de la primera versión de Roblockly, desarrollada por **Cristian Manuel Ángel Díaz** en su Trabajo de Fin de Grado "Robótica educativa y pensamiento computacional", elaborado en el curso académico 2018 - 2019 en el Grado de Ingeniería Informática de la Universidad de La Laguna. Además, también se pretende esbozar un módulo estadístico que permita recoger información sobre la soluciones diseñadas por el usuario y hacer

posible un seguimiento de su progreso con el uso de la aplicación. Todo esto cumpliendo siempre con la premisa de que sea una aplicación en línea, de acceso gratuito y código abierto.

Capítulo 2: Contextualización

2.1 Conceptos relevantes

El objetivo de este punto es profundizar en dos conceptos clave de este TFG ya mencionados previamente, por un lado el pensamiento computacional y por otro, la robótica educativa.

2.1.1 Pensamiento computacional

El pensamiento computacional es un término que tuvo su origen en ideas de Seymour Papert, pero fue desarrollado por Jeannette Wing, quien en 2006, publicó una columna de opinión en el número de marzo de la revista Communications of the ACM [3] en la que expuso que el pensamiento computacional surge como un concepto transversal que engloba un conjunto de habilidades y procesos cognitivos, y que dichos procesos y habilidades son indispensables para los expertos en Ciencias de la Computación, y para otras disciplinas científicas [4]. Asimismo, la persona que emplea el pensamiento computacional, es capaz de descomponer un problema en problemas más pequeños que, a su vez, son más fáciles de resolver. De esta manera, se puede reformular cada uno de estos problemas para facilitar su solución haciendo uso de estrategias de resolución de problemas familiares [5].

Además de esto, tal y como dice Miguel Zapata-Ros, al pensamiento computacional lo conforman elementos como: la creatividad, la resolución de problemas, el pensamiento abstracto, la recursividad, la iteración y los patrones, entre otros [1]. Estos componentes se pueden encontrar claramente en las fases de creación de un código, así pues, tal y como se menciona en [1], el proceso de creación de un código se compone de: una primera fase para la detección y delimitación del problema y su naturaleza; una segunda fase en la que se delimitan los métodos y disciplinas posibles a emplear en la resolución del problema; en tercer lugar, una fase de organización de la resolución tras la que se lleva a cabo el diseño de dicha resolución; en cuarto lugar, una fase algorítmica a la que le sigue la elaboración del código y por último, la fase de prueba y validación en la que se verifica la funcionalidad del código y se depuran los posibles errores que hayan podido surgir.

De la misma manera, dichos componentes se pueden apreciar en otros ámbitos de la vida cotidiana, lo cual pone de manifiesto la importancia del desarrollo del pensamiento computacional para la vida en general. Por consiguiente, tal y

como se puede observar, los beneficios del fomento de dicha competencia son muchos, y es por ello que una de las principales motivaciones de este Trabajo de Fin de Grado ha sido contribuir al desarrollo del mismo en estudiantes de enseñanza obligatoria.

2.2.2 Robótica educativa

Si bien el pensamiento computacional es uno de los elementos que conforma la esencia de este Trabajo de Fin de Grado, sin duda la robótica educativa es otro. Ésta apareció a principios de los años 90 [6] y se ha desarrollado notablemente en las últimas décadas en casi todos los países a la vez que su relevancia ha ido en aumento [2]. Así pues, de la misma manera en la que previamente se argumentaba la importancia que tiene el pensamiento computacional para el presente y el futuro, es una realidad que cada vez hay más robots en prácticamente todos los ámbitos de la sociedad, y es por ello que conceptos como este, han ido tomando cada vez mayor relevancia en el campo de la educación [2].

La robótica educativa, implica la resolución de retos o problemas de manera grupal o individual, a través del diseño y construcción de un robot que ha de poder ser programado a través de un ordenador para moverse, manipular objetos y llevar a cabo diferentes y determinados trabajos por medio de la interacción con su entorno [2]. Por lo tanto, la idea es que los estudiantes hagan uso de su ingenio y de las herramientas que se les proporcionan para programar y configurar dicho robot, de tal manera que sea capaz de superar el reto en cuestión.

Asimismo, cabe destacar que, además de lo mencionado previamente, también se pretende trabajar en el alumno competencias básicas necesarias en el mundo actual, tales como el aprendizaje colaborativo, la toma de decisiones en equipo y el pensamiento computacional, a la vez que trabaja temas multidisciplinares como la electrónica, la informática, la mecánica y la física, entre otros [2].

Por otra parte, tal y como se ha comentado en puntos anteriores, Roblockly fundamenta su dinámica de funcionamiento en la robótica educativa, es por ello que puede servir como un ejemplo perfecto para exponer las fases que se han de llevar a cabo durante una sesión docente de robótica educativa:

1. **Planteamiento del reto a superar**. En primer lugar, la persona docente propondrá el reto que deberán resolver los alumnos, quiénes deberán analizarlo para plantear una posible solución inicial. Durante esta fase, se trabajan habilidades como la descomposición de problemas y la abstracción, entre otras [7].
2. **Elección y configuración de sensores del robot.** Una vez propuesta una posible solución inicial, los alumnos deberán escoger el robot que vayan a utilizar y configurarlo con los sensores más convenientes para superar el reto.

Para ello, es imprescindible que los estudiantes tengan claro previamente cuál será el comportamiento que deberá llevar a cabo el robot y en base a ello, seleccionar los sensores apropiados. A lo largo de esta fase, interviene el reconocimiento de patrones y la creatividad [7].

3. **Programación del robot.** En esta fase, se diseñará e implementará el algoritmo que permita recopilar información de su entorno al robot y le permita actuar en consecuencia, de tal manera que pueda superar el reto propuesto [7]. Cabe destacar que la manera de llevar a cabo esta fase, puede variar según la tecnología y el material que se esté empleando para el desarrollo de la actividad. En el caso de Roblockly, para la programación del robot, se utiliza un sistema de programación visual por bloques al estilo *Blockly* [8], que permite al alumno una interacción más clara, sencilla y lúdica con la programación. Además de lo anteriormente mencionado, esta etapa es muy útil para el desarrollo del pensamiento algorítmico [7].
4. **Simulación.** Tras la programación del robot, se lleva a cabo la simulación del mismo para comprobar que los sensores utilizados y el algoritmo propuesto, funcionan correctamente. En caso de que no sea así, se deberán realizar las correcciones necesarias para que el objetivo se consiga adecuadamente.
5. **Exposición de resultados**. En este último paso, los alumnos mostrarán su diseño y propuesta de solución al resto de compañeros, a la vez que compartirán su experiencia durante el transcurso de la actividad.

Para concluir, cabe mencionar que desde Roblockly, se propone en todo momento un planteamiento lúdico que provoque la sensación de juego a los estudiantes y que por consiguiente, les motive a seguir avanzando y desarrollando soluciones para los distintos retos, a la vez que desarrollan todas las habilidades mencionadas previamente.

2.2 Antecedentes y estado actual del tema

Por lo que se refiere a aplicaciones y herramientas para llevar la robótica al ámbito educativo, existen más alternativas además de Roblockly que conviene conocer. Por ello, a continuación se muestra una sucinta recopilación de otros proyectos que guardan relación con la aplicación que se ha desarrollado en este TFG. Así mismo, la tabla 8.1 del apéndice “Tablas comparativas” del capítulo 8, muestra una comparativa entre las aplicaciones mencionadas a continuación y Roblockly.

Jabutí EDU Nube

Consiste en una plataforma virtual que tiene como objetivo posibilitar la enseñanza de robótica educativa a distancia. Está basada en el concepto de IoT (Internet of Things) y su planteamiento consiste en permitir que los alumnos puedan controlar dispositivos físicos a través de un portal web, que posee un entorno de programación basado en el lenguaje de programación LOGO [9]. Además, al ser un

entorno Open Hardware, permite trabajar con diferentes tipos de proyectos de robótica basados en ESP32, ESP 8266 y Arduino, lo cual permite que los distintos centros educativos puedan trabajar con sus propios desarrollos [9].

Otra característica de esta aplicación es la posibilidad de que varios usuarios puedan programar el mismo robot al mismo tiempo, ya que posee una herramienta de alternancia que permite al docente encargado, ejecutar el código de cada alumno una vez que haya sido revisado y validado por él mismo [9].

ROBOT-EDULPGC

Se trata de un proyecto de innovación educativa llevado a cabo por la Universidad de Las Palmas de Gran Canaria, en el que se plantea el diseño, implementación y puesta en práctica de una plataforma modular de robótica educativa de bajo coste [10], [11]. Guarda cierta similitud con Roblockly en cuanto a que es un proyecto que también se ha desarrollado mediante la integración de varios Trabajos de Fin de Grado y busca eliminar la barrera económica a través de su bajo coste [10], [11].

La idea principal de ROBOT-EDULPGC radica en el diseño de un robot educativo abierto y de bajo coste, que gracias a su modularidad, pueda ser utilizado en todos los niveles educativos (multidisciplinar), haciendo uso de hardware y software libres.

Open Roberta Lab

Open Roberta Lab es un entorno de programación en la nube que permite programar diversos tipos de robots educativos, entre los que se encuentran los robots *EV3* y *NXT* de *LEGO MINDSTORMS* [12], el robot *Bot'n Roll ONE A* [13] basado en Arduino, el *BBC micro bit* [14] y el *Calliope mini* [15], entre otros [16].

Para llevar a cabo la programación de dichos robots, utiliza un sistema de programación visual por bloques a través del meta-lenguaje de programación NEPO [16], el cual posee una capa de conexión hardware que le permite trasladar a los robots, los algoritmos que se desarrollen a través de su sistema de programación por bloques, elaborado haciendo uso de la librería de Google Blockly [8]. Además de esto, también posee una serie de funcionalidades y mejoras que le permiten una mejor integración con Open Roberta.

Por último, cabe destacar que esta aplicación es de acceso totalmente gratuito y puede ser utilizada sin necesidad de llevar a cabo ningún tipo de registro en su página web.

Zowi

Este último, tal vez es el proyecto que más dista del modelo de robótica educativa expuesto anteriormente, ya que se ha planteado como un robot educativo de uso personal, es decir, inicialmente no se ha planteado como un producto para ser llevado a las aulas. No obstante, por las características que posee y su funcionamiento, podría ser utilizado perfectamente en una sesión docente de robótica educativa salvo por su coste económico, el cual podría suponer un impedimento para algunos centros educativos.

Este robot destinado al público de menos edad [17] creado por la empresa española “bq”, está compuesto por una “cabeza” con forma de cubo en la que se encuentran sus “ojos”, formados por dos sensores de ultrasonidos y sonido respectivamente y dos “piernas” que le permiten desplazarse (véase la Fig. 2.1).

Para concluir, cabe mencionar que la compañía ha empleado como placa un modelo compatible con Arduino, [17] además de la posibilidad de programar el robot haciendo uso del lenguaje visual Bitbloq; el cual ha sido creado por “bq” y sigue la misma línea que los ya mencionados previamente, ya que también es un sistema de programación por bloques al estilo Google Blockly [17].

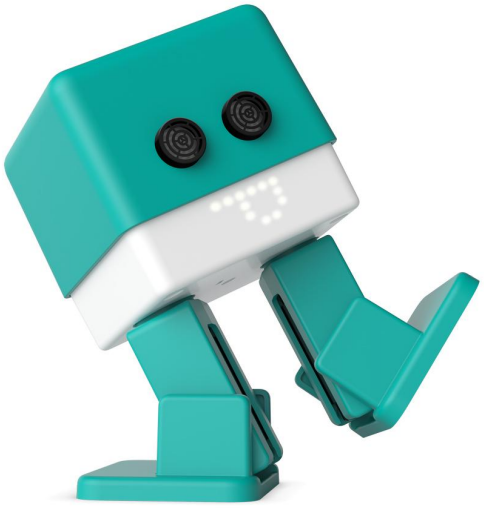


Fig. 2.1: Robot educativo Zowi

Capítulo 3: Metodología

3.1 Actividades a realizar

Para el desarrollo de este Trabajo de Fin de Grado, se han propuesto un total de 12 actividades a llevar a cabo para la consecución de los objetivos propuestos ya mencionados previamente.

1. Búsqueda de información bibliográfica necesaria para llevar a cabo los objetivos mencionados previamente, así como el desarrollo del capítulo 2, ya presentado en esta memoria.
2. En esta fase, se pretende llevar a cabo un estudio en profundidad de las prestaciones y características de la aplicación, con el objetivo de plantear posibles mejoras y la adición de funcionalidades que no posee actualmente. Así mismo, se pretende examinar el código y su estructura para llevar a cabo una refactorización del mismo en las secciones que lo precisen.
3. Refactorizar el código con las mejoras planteadas en la fase anterior.
4. En base a la bibliografía consultada, proponer nuevos robots y sensores que se puedan añadir a la aplicación para enriquecerla ampliando la capacidad de los distintos robots y aumentando así la variedad de retos posibles.
5. Creación de un nuevo módulo para la elección del robot y la configuración del mismo, a través de la selección de los sensores que se vayan a emplear en la resolución del reto propuesto.
6. Añadir los nuevos modelos de robots propuestos así como la lógica necesaria para que puedan llevar a cabo los movimientos requeridos.
7. Añadir los nuevos sensores propuestos así como la programación necesaria para su funcionamiento.
8. Creación de los retos necesarios para probar y utilizar los nuevos sensores añadidos.
9. Realizar una interfaz sencilla para la elección de los diferentes retos creados en la fase anterior.
10. Estudio y planteamiento sobre el esbozo de un primer prototipo de módulo estadístico que permita recoger información sobre la soluciones diseñadas por el usuario y hacer posible un seguimiento de su progreso con el uso de la aplicación.
11. Implementación del módulo estadístico.
12. Elaboración de la memoria del trabajo.

3.2 Plan de trabajo

Inicialmente, tal y como está planteada la programación del curso académico y de la asignatura de “Trabajo de Fin de Grado”, se propuso finalizar el proyecto antes de la fecha límite para la convocatoria de Junio, no obstante, se produjo una

serie de retrasos que dificultaron la finalización y presentación del mismo antes de la convocatoria de Septiembre.

A continuación, se muestra una tabla que ilustra la readaptación del plan de trabajo inicial, elaborado en base a la finalización y presentación de dicho proyecto en la convocatoria de septiembre del presente curso académico:

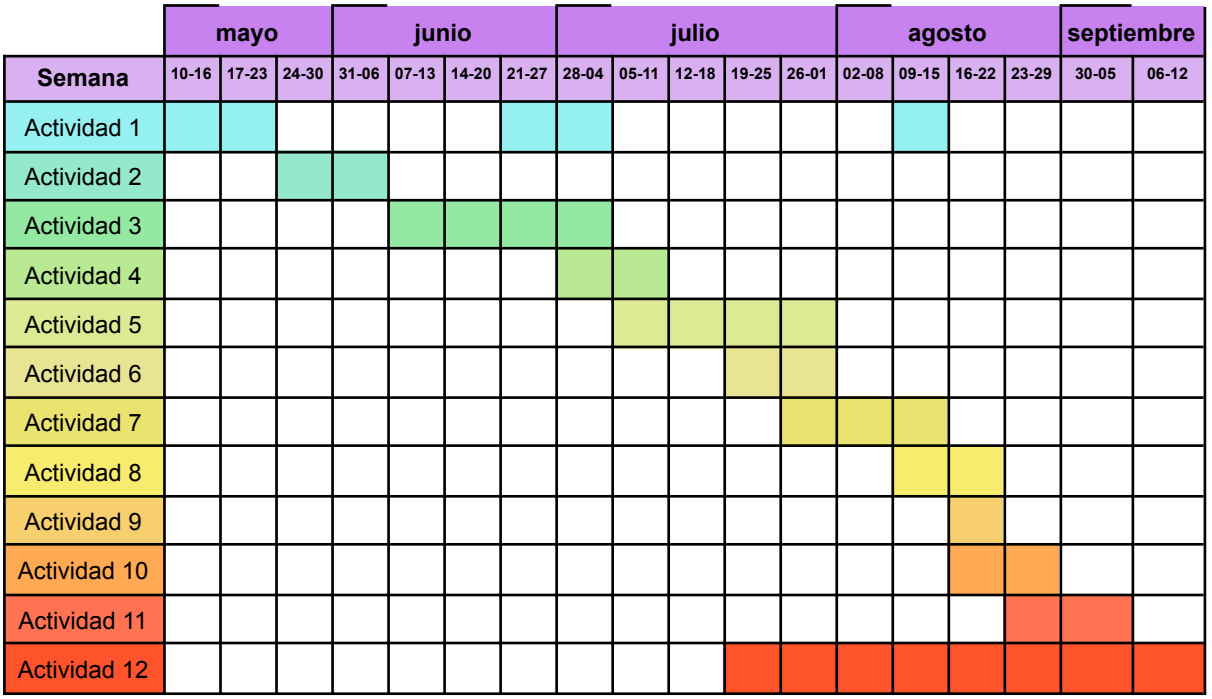


Tabla 3.1: Diagrama de Gantt con la planificación del proyecto

3.3 Tecnologías utilizadas

En esta sección se especifican las tecnologías empleadas durante el desarrollo de este proyecto, así como su funcionamiento y la manera en la que han sido utilizadas para la confección de este Trabajo de Fin de Grado.

Unity

Ésta ha sido la herramienta con la que se ha elaborado prácticamente la totalidad del proyecto, ya que el resto de tecnologías empleadas, se han utilizado como un complemento de Unity o como apoyo para otras áreas del trabajo. Dicha herramienta es un motor de videojuegos multiplataforma desarrollado por la empresa *Unity Technologies*, disponible como plataforma de desarrollo para Microsoft Windows, Mac OS y Linux [18].

Tal y como se ha mencionado anteriormente, es una herramienta multiplataforma, lo cual ha sido considerado como una característica de gran valor y utilidad para la realización de este TFG, ya que ofrece la posibilidad de exportar el

proyecto que se esté desarrollando en ella a HTML5 a través de WebGL [19], además de muchas otras plataformas. Contar con esta funcionalidad ha resultado fundamental para cumplir con el objetivo ya mencionado que consiste en facilitar el acceso a la aplicación a todos los usuarios de tal manera que sólo sea necesario un ordenador con conexión a internet.

Por otra parte, cabe mencionar que Unity cuenta con dos licencias principales para desarrolladores: Unity Professional y Unity Personal [18]. Esta última, ha sido la que se ha utilizado durante el desarrollo de Roblockly, pues permite el acceso a todas las prestaciones del motor con la única salvedad de que cuando se compila un proyecto, al ejecutarlo se muestra siempre al comienzo una breve animación que muestra el logotipo de Unity y la frase “Made with Unity”. El resto de restricciones están supeditadas a cuestiones económicas que por el momento, no han sido motivo de preocupación para la consecución del propósito de este trabajo.

Siguiendo la línea expuesta previamente sobre la utilización de Unity para la confección de Roblockly, conviene llevar a cabo un breve glosario que agrupe la descripción de los términos más relevantes de Unity empleados en el elaboración de este trabajo, pues en próximos capítulos, la redacción de esta memoria se apoyará en dichos términos para explicar el desarrollo del proyecto.

- **Escena:** Es una estructura de datos que contiene todos los elementos necesarios para la ejecución de un fichero .unity [20]. Permiten separar de manera muy favorable los distintos contextos y secciones de nuestra aplicación.
- **GameObject:** Es la clase base que compone todas los elementos de una escena de Unity. Una escena en Unity, viene por defecto con dos GameObjects: una luz y una cámara. Otros ejemplos de GameObjects son: un sensor, un robot, un botón, una rueda, un plano, una esfera, etc.
- **Jerarquía:** Por defecto está situada en el lateral izquierdo de la interfaz de usuario (IU) de Unity y representa la relación que guardan todos los GameObjects de una escena entre sí.
- **Cámara:** Es un tipo de GameObject que permite determinar el espacio de la escena que se va a renderizar durante la ejecución del proyecto.
- **Canvas:** Es otro tipo de GameObject utilizado para la renderización de elementos de la interfaz gráfica de usuario como pueden ser, texto, imágenes, botones, paneles, etc. Sin un canvas no es posible renderizar ni mostrar al usuario ninguno de los elementos mencionados previamente.
- **Prefab:** Es el concepto con el que se conoce a la consolidación de un GameObject. Un prefab se genera cuando se extrae un GameObject de una escena y se añade a otra escena o simplemente se guarda como plantilla para ser utilizado e instanciado en otros lugares del proyecto en los que sea necesario.

- **Componente**: Es el nombre que se le da a las propiedades y características que posee un GameObject.
    - **Transform**: Es la propiedad que almacena los datos de la posición, rotación y escala de un GameObject.
    - **Collider:** Es una propiedad que se le añade a un GameObject para los propósitos de colisiones físicas [21].
    - **RigidBody:** Es un componente utilizado para conseguir que el movimiento de un GameObject se realice a través del motor de físicas de Unity, de tal manera que su movimiento fuera tal y como el que describe un objeto físico en la realidad.
    - **Script:** Es el componente que permite que se le puedan programar determinados comportamientos y funcionalidades a un GameObject. Se pueden crear tantos scripts como se deseen y atribuirlos a uno o varios GameObjects.
    - **Material:** Como bien su nombre indica, hace referencia al material del cual está compuesto un GameObject.
- **Inspector:** Normalmente situado en el lateral derecho de la interfaz de Unity, permite observar y acceder a todas las componentes que posee un GameObject de la escena.
- **Delegado:** Es un elemento de la API de Unity, por lo que no se puede observar como la mayoría de elementos expuestos previamente. Se utiliza para referenciar a un método o función de otra clase cuando sucede un evento concreto en la aplicación. Por ejemplo, se puede utilizar para indicar al gestor de estadísticas que el robot superó un reto; así pues, cuando esto sucede, se genera desde la clase que controla la finalización del reto, un "aviso", también conocido como "evento" en el argot de Unity, que contiene cierta información que es recibida y procesada por la clase que gestiona las estadísticas de la aplicación.

Por último, cabe resaltar que Unity trabaja con C# y JavaScript como lenguajes de programación compatibles, además de con varias versiones diferentes de la propia aplicación. Para este trabajo, se ha empleado el lenguaje **C#** y la versión **2020.3.17f1** de Unity.

Git y GitHub

Para llevar a cabo un control de versiones en línea a lo largo del desarrollo de este trabajo, se ha hecho uso de dos tecnologías estandarizadas en el mundo de la programación y el desarrollo de software:

- Por una lado, **Git**, que es un sistema de control de versiones distribuido, de código abierto y gratuito [22].
- Por otra parte, **GitHub**, consistente en una plataforma de desarrollo colaborativo para alojar proyectos, generalmente de programación, utilizando Git [23].

Durante la elaboración de este Trabajo de Fin de Grado, se ha utilizado el un repositorio de Github, actualmente público, en el que se encuentra alojado todo el proyecto de Roblockly en Unity.

Enlace para acceder al respositorio con el proyecto de Roblockly actual:
https://github.com/alu0101049151/Roblockly_BasilioGN.git

Asimismo, GitHub cuenta con un servicio llamado “GitHub Pages” que permite desplegar una página web por cada repositorio público. Así pues, esto brinda la posibilidad de alojar Roblockly en la web y acceder a él a través de un enlace, lo cual resulta idóneo para lograr este objetivo propuesto inicialmente en el proyecto.

El despliegue de la versión de la aplicación desarrollada en este Trabajo de Fin de Grado, se ha realizado en el siguiente enlace:
https://alu0101049151.github.io/Roblockly_Online_Deploy/

UBlockly

UBlockly es una reimplementación para Unity de la librería de Google, “Blockly” [8] que ha sido desarrollada por Ling Mao (https://github.com/imagicbell). Esta librería es la misma que se empleó en la versión inicial de Roblockly, no obstante, aún se encuentra en desarrollo y desde entonces, ha adquirido nuevas prestaciones que no poseía la versión integrada en la implementación inicial de Roblockly. Es por ello que una de las tareas llevadas a cabo durante el desarrollo de este TFG, ha sido añadir estas nuevas funcionalidades a la versión del simulador confeccionada durante este trabajo.

Cabe destacar que a la hora de instalar la librería, surgieron una serie de problemas que dificultaron su correcta integración con el proyecto. En primer lugar, se tuvo que cambiar la versión de Unity del proyecto, para que fuera compatible con la última versión de UBlockly, ya que hasta ese entonces, se había estado trabajando con la versión de Unity 2019.1.0f.2, que era la misma en la que se había desarrollado la primera versión Roblockly. Esto supuso algunas complicaciones, pero afortunadamente tras realizar los ajustes convenientes, todo continuó funcionando correctamente. En segundo lugar, cabe mencionar que a pesar de ser una librería aún en desarrollo, la documentación que posee es un poco escueta en algunos aspectos, como por ejemplo el proceso de integración de la librería con Unity y los posibles problemas que puedan surgir dependiendo del sistema operativo utilizado. Tanto es así, que este fue el primer inconveniente que surgió con la librería, ya que al instalarla y seguir los pasos indicados en la documentación, no fue posible acceder a la escena de ejemplo que trae. Así pues, tras indagar y realizar una consulta a través del foro de *Issues* del repositorio de la librería, se llegó a la conclusión de que el sistema de rutas de directorios que sigue Unity cambia según el sistema operativo, ya que en Mac OS funciona perfectamente la instalación

de la librería pero en Microsoft Windows, hay que mover de manera manual los directorios *Source* y *UserData* a la ruta *Examples/Assets*. Por último, cuando se compiló por primera vez la aplicación y se probó como aplicación de escritorio y aplicación web, se pudo comprobar que el arrastre de los bloques a la zona de programación no funcionaba en ninguna de las dos plataformas, y que por lo tanto no era posible llevar a cabo ninguna programación con los bloques. Este problema supuso un punto de inflexión en el desarrollo del proyecto, ya que al tratarse de un error de funcionamiento interno de la propia librería, arreglarlo manualmente podría ser muy complejo además suponer un gasto de tiempo inasumible. Por lo tanto, se tomó la decisión de acudir de nuevo al foro de *Issues* del repositorio de la librería, plantear el problema sucedido y esperar una pronta respuesta. Afortunadamente, la desarrolladora Ling Mao fue rápida y aportó la solución al problema en apenas un par de días, permitiendo así continuar con el plan de trabajo previsto para la elaboración del proyecto.

Por otra parte, las nuevas funcionalidades que posee esta actualización de la librería, permiten que el usuario pueda depurar el algoritmo que haya realizado con los bloques, lo cual puede enriquecer notablemente la experiencia de uso de los estudiantes. Para utilizarlas, simplemente se debe programar con los bloques necesarios el algoritmo que quiera depurar y una vez lo tenga finalizado, antes de pulsar el botón de ejecución situado en la esquina superior derecha de la interfaz (véase la Fig. 3.1), debe marcar la casilla que pone *Debug* (véase la Fig. 3.2) y posteriormente comenzar la ejecución pulsando el nuevo botón de ejecución (véase la Fig. 3.3) que aparece en la misma posición que el anterior. Una vez comienza la ejecución del código y la depuración del mismo, se puede observar un pequeño punto verde en la esquina superior izquierda del bloque que está siendo ejecutado en ese momento (véase la Fig. 3.4).

Fig. 3.1: Botón de ejecución

Fig. 3.2: Casilla *Debug*

Fig. 3.3: Botón de ejecución durante la depuración

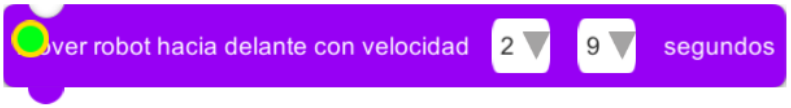


Fig. 3.4: Bloque en ejecución durante la depuración

En cuanto al resto de aspectos de la librería, cabe mencionar que su funcionamiento es bastante satisfactorio y una vez pasada una curva inicial de aprendizaje, su uso es muy sencillo, en especial, la programación y creación de bloques personalizados, gracias a una nueva documentación algo más elaborada que la que se encuentra en el repositorio.

Se puede acceder a esta nueva guía de uso a través del siguiente enlace:
https://hackmd.io/@beBvDP44ShyK5VorFbhGcw/H1Qbb1HBu

Por último, conviene mencionar a los bloques personalizados que se han creado corresponden a las categorías de *sensors* y *move,* las cuales se pueden encontrar definidas en la ruta: *Examples/Assets/Source/JsonBlocks*. Asimismo, la implementación de la lógica de cada bloque de cada categoría se encuentra en los ficheros *Sensors_CSharp y Move_CSharp* que a su vez se encuentran en la ruta: *Examples/Assets/Source/JsonBlocks/Script/CodeDB/CSharp/Interpreters.*

Visual Studio Code

Este es el entorno de desarrollo integrado (*IDE*) utilizado durante la elaboración del proyecto. En él se han programado todos los scripts que luego han sido aplicados como componentes a los distintos GameObjects de las escenas. Asimismo, también ha permitido llevar a cabo de manera muy sencilla, la depuración del código cuando ha sido necesario.

3.4 Fase de desarrollo

Este punto pretende abordar distintos aspectos sobre la elaboración del proyecto, así como detallar la implementación de algunos elementos del mismo.

3.4.1 Elaboración del código

De la misma manera que este TFG se ha apoyado en uno previo y tal y como se menciona en el “Capítulo 5: Conclusiones y líneas futuras”, es muy posible que este trabajo también sirva como base para otros venideros. Así pues, con el fin de conseguir un código de calidad, funcional, óptimo y comprensible, durante la confección de este Trabajo de Fin de Grado, se han seguido una serie de pautas de diseño y elaboración de código en Unity que se detallan a continuación:

Guía de estilos

Como base de las ideas mencionadas previamente, se ha intentado seguir una guía de estilos de programación en Unity con el lenguaje C#, no obstante, durante la fase de investigación y búsqueda de material bibliográfico en la que se indagó sobre esta cuestión, no se consiguió dar con una única guía que cubriera todos los puntos considerados para la elaboración de código, por lo que finalmente, se decidió usar una complementación de los diversos manuales encontrados.

Así pues, algunas de las “normas” que se emplearon a la hora de elaborar el código fueron las siguientes:

- Los nombres de métodos y clases han de escribirse en formato *PascalCase*.
- Los nombres de parámetros de funciones, siempre han de escribirse en estilo *camelCase*.
- Siempre se debe utilizar llaves en las sentencias if/ else if /else, salvo en el caso de que puedan ser escritos en una sola línea, en cuyo caso estaría permitido no utilizarlo.
- Para las llaves de las funciones, se debe emplear el estilo *Allman* [24].
- Los espacios de nombre se deben indicar en orden alfabético debajo de las directivas “using”, que se deben situar encima de los espacios de nombre.
- Evitar dejar espacios en blanco innecesarios.

A continuación se muestran los enlaces que permiten acceder a las distintas guías de estilo utilizadas:

1. https://github.com/dotnet/runtime/blob/main/docs/coding-guidelines/coding-style.md
2. https://docs.microsoft.com/en-us/dotnet/standard/design-guidelines/naming-guidelines
3. https://uvasgd.github.io/sgd-docs/unity/documentation.html (Sólo utilizada la convención de nombres).

Así mismo, cabe destacar que también se han empleado “normas” estándar como limitar a 80 el número de caracteres por línea de código, tabular utilizando 4 espacios, nombrar las variables, métodos y clases de la manera más autoexplicativa posible o programar todo el código en inglés.

Documentación del código

Con la intención de facilitar una mejor y más fácil comprensión del código desarrollado en este TFG a las personas que vayan a consultar o trabajar con él en futuras ocasiones, en cada script del proyecto se ha añadido una documentación que consta de 3 componentes:

1. Comentario de cabecera. En las primeras líneas de cada fichero de código, se encuentra un comentario como el siguiente:

```
/**
* Universidad de La Laguna
* Project: Roblockly
* Author: Basilio Gómez Navarro
* Email: alu0101049151@ull.edu.es
* File: SomeClass.cs: This file contains the SomeClass class
*       implementation to manage its different functionalities.
*/
```

Fig. 3.5: Comentario de cabecera de cada script

2. Comentario de cabecera de Clase y Métodos. En la línea inmediatamente superior a la declaración de una clase o un método de la misma, se encuentra un comentario que describe brevemente su funcionamiento y sus parámetros y valor de retorno en caso de ser un método o función que tome algún parámetro o devuelva algún valor, tal y como se muestra a continuación:

```
/// <summary>
/// Here goes the method functionality description.
/// </summary>
/// <param name="someParameter">
/// Here goes the “someParameter” description.
/// </param>
/// <returns>
/// Here goes the return value description.
/// </returns>
public bool SomeFunction(string somePrameter) {}
```

Fig. 3.6: Modelo de comentario de cabecera de clases y métodos

3. Comentarios de fin de línea. Sólo empleados en caso de ser necesarios. Son los comentarios de fin de línea estándar utilizados para aclarar o detallar algún aspecto del código programado.

Modularidad

Otro objetivo de cara a la programación ha sido conseguir un código lo más modular y encapsulado posible, procurando siempre que cada clase implementada desempeñe una única función, tal y como se promueve en el principio SOLID de responsabilidad única [25].

Optimización del código

Por último, también cabe mencionar que como premisa a la hora de programar, se ha buscado la eficiencia en el código desarrollado, de tal manera que el resultado final sea un código óptimo. No obstante, al no poseer un dominio muy elevado con la API de Unity, es posible que se hayan empleado algunas funciones de la misma que podrían ser sustituidas por otras más eficientes.

3.4.2 Modificaciones y mejoras propuestas al proyecto original

Tal y como se ha expuesto previamente, el objetivo de este Trabajo de Fin de Grado es la reimplementación y ampliación de funcionalidades de la primera versión de Roblockly. Así pues, en esta sección se pretende exponer sucintamente las ampliaciones y modificaciones llevadas a cabo con respecto a la versión inicial.

Si bien es cierto que en un principio se planteó la posibilidad de trabajar directamente sobre el proyecto de Unity de la versión inicial de Roblockly, finalmente

se descartó dicha opción ya que también se pretendía llevar a cabo una refactorización del código. Esto último requería comprender el funcionamiento del mismo casi a la perfección y debido a la falta de documentación de este, y la intención de rediseñar casi todas las escenas de la aplicación, se consideró mejor opción elaborar un nuevo proyecto de Unity y usar como referencia el anterior.

Para el rediseño de la aplicación se planteó la posibilidad de crear una identidad corporativa que integrase el color violeta característico de la Universidad de La Laguna [26]. Esto fue posible gracias a la colaboración de **Paula González Mora**, quien se encargó del diseño y aplicación de la misma.

A continuación se presentan dos elementos de la identidad corporativa mencionada previamente:

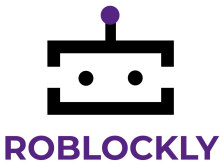


Fig. 3.7: Elemento 1

Fig. 3.8: Elemento 2

Apoyándose en dicho trabajo, se llevó a cabo el diseño de una nueva interfaz de usuario más intuitiva y fácil de utilizar, en la que se integraron modificaciones y novedades como la posibilidad de elegir diferentes modelos de robots, añadir nuevos sensores de una manera más práctica o seleccionar nuevos retos acorde con los nuevos sensores desarrollados, entre otras que se detallarán en el siguiente punto.

3.4.3 Desarrollo de las modificaciones y mejoras propuestas

En primer lugar, con la finalidad de facilitar la comprensión del proyecto de Unity elaborado durante el trabajo y del contenido que se expone en este punto, a continuación se desglosa brevemente la distribución de directorios llevada a cabo dentro del directorio “Assets” del proyecto:

1. *Images*. Contiene todas las imágenes empleadas en el proyecto.
2. *Plugins*. En él se encuentran los distintos complementos descargados de terceros utilizados en el proyecto.
3. *ProjectPrefabs*. Agrupa todos los *prefabs* generados a lo largo de la elaboración de la aplicación.
4. *Scenes*. Contiene todas las escenas utilizadas en el proyecto.
5. *Scripts*. En este directorio se encuentran todos los scripts programados durante este TFG.
6. *Textures*. Posee las texturas creadas manualmente para diferentes elementos del proyecto.

7. *ublockly-master*: En él se encuentra el contenido de la librería UBlockly.

Una vez aclarada la distribución de directorios del proyecto, en los siguientes puntos de esta sección se procede a detallar el desarrollo de las modificaciones y mejoras propuestas a la versión inicial de Roblockly.

3.4.3.1 Elección del robot

Una de las premisas consideradas para el nuevo diseño de la aplicación, consiste en darle una estética que se asemeje más a la de un videojuego. Un claro ejemplo de ello, es el nuevo módulo de elección del robot, pues resulta similar a la elección del avatar de un videojuego en tercera persona o del coche de un videojuego de carreras. Este módulo implementado en la escena ***IndividualSelectionMenu***, permite que a través de dos flechas situadas en ambos lados de la interfaz de usuario, este pueda visualizar las distintas opciones de robots disponibles para desarrollar los diferentes retos y elegir el que prefiera (véase la Fig 6.2).

Para llevar a cabo el cambio de robots, se ha creado un GameObject vacío llamado ***RobotHolder*** que contiene los distintos modelos de robots disponibles que se van activando cuando el usuario pulsa las flechas mencionadas previamente. Esto se ha conseguido mediante la implementación de la clase *RobotSelection,* que se encarga de la activación y desactivación de los robots y finalmente, de la confirmación del elegido, que a su vez, se consigue con la búsqueda de aquel GameObject que posee la etiqueta “CurrentRobot” y el almacenamiento de su posición dentro del objeto *RobotHolder* en el GameObject de Unity que contiene las preferencias del jugador (*PlayerPrefs)*.

3.4.3.2 Elección del reto

Tras la elección del robot, el usuario debe elegir el reto que va a llevar a cabo. Esto es así ya que una vez elegido el reto, deberá configurar el robot con los sensores necesarios para superar dicho reto.

La elección del reto, se lleva a cabo en la escena ***ChallengeSelection*** y básicamente consta de 4 botones con imágenes de cada uno de los retos denominados “Reto del laberinto”, “Reto del sendero blanco”, “Reto del sendero de colores” y “Reto de la plataforma móvil”, pensados para ser resueltos haciendo uso de los sensores de ultrasonido y contacto, infrarrojo, color y giróscopo respectivamente (véase la Fig 6.6). Dicha escena se puede considerar como una novedad de esta versión de Roblockly, ya que en la versión inicial, sólo se podía elegir entre dos retos y la elección se llevaba a cabo en la misma escena de montaje y configuración del robot.

3.4.3.3 Elección de los sensores

El módulo que permite seleccionar y añadir al robot los sensores necesarios para superar el reto elegido, ha sido reimplementado por completo en la escena ***IndividualSensorSelection***, ya que se ha propuesto una nueva interfaz que permite realizar esto de una manera más sencilla e intuitiva, a través del concepto de "arrastrar y soltar" (*drag and drop*).

En relación con el robot, en el primer fotograma de la escena, se lleva a cabo la búsqueda y activación del robot seleccionado previamente y se establece como un objeto indestructible entre escenas, lo cual se consigue gracias al método *DontDestroyOnLoad* de Unity. Una vez que éste aparece, el usuario puede rotarlo a su gusto clicando encima de él y arrastrando el cursor hacia el lado que quiera rotarlo.

A su vez, tal y como se muestra en la Fig. 6.7, esta IU está compuesta principalmente por un panel en el lateral izquierdo que contiene todos los sensores disponibles. Dicho panel se puede dividir en dos secciones: la parte superior que contiene a los sensores externos y la parte inferior que contiene los sensores internos. Así pues, como bien su nombre indica, los sensores de la parte superior, funcionan acoplados a la carrocería del robot, mientras que los internos, ya están añadidos por defecto en el interior del mismo y, simplemente hay que activarlos o desactivarlos haciendo click sobre la casilla que se encuentra al lado del nombre de cada uno. Para desarrollar esto último, se ha implementado un sistema de delegados de Unity (véase la Fig. 3.9) que se disparan cuando el usuario selecciona la casilla del sensor que quiere utilizar para "activar" o "desactivar" el código y la lógica correspondientes al sensor seleccionado.

```
public void Acivation()
{
   if (!gameObject.activeSelf)
   {
      gameObject.SetActive(true);
      CheckboxGyroscope(); // Delegado para activar el Giróscopo
   } else {
      gameObject.SetActive(false);
      DeactivateGyroscope(); // Delegado para desactivar el Giróscopo
   }
}
```

Fig. 3.9: Cógio del fichero *CheckGyroscope.cs* para la activación o desactivación del giróscopo

En cuanto a la selección y acople de los sensores externos, se ha desarrollado el sistema de "arrastrar y soltar" (*drag and drop)* mencionado previamente, el cual permite que el usuario simplemente tenga que hacer click sobre la imagen del sensor externo que quiera utilizar y, sin soltar el botón izquierdo del ratón, arrastre el sensor que ha aparecido en la posición del cursor hasta uno de los

puntos de anclaje habilitados en el robot. Estos puntos de anclaje se han denominado ***snap points*** durante el desarrollo del proyecto y su lógica se ha implementado en el script ***SnapController.cs****,* aplicado a cada uno de los robots. Cuando se selecciona un sensor externo, dichos puntos cambian su color a rojo, de tal manera que el usuario puede ver con facilidad dónde se encuentran ubicados y por consiguiente, dónde puede colocar el sensor elegido. Asimismo, si este no es aproximado suficientemente a uno de los *snap points* disponibles, desaparecerá para indicar que el anclaje no ha sido exitoso.

Además de esto, cabe destacar, que si se coloca un sensor en un *snap point* no deseado, existe la posibilidad de eliminarlo. Para ello, basta con hacer clic sobre el sensor en cuestión y seleccionar el botón “Eliminar” del panel que aparece en el lateral derecho de la interfaz tras pulsar sobre el sensor. De lo contrario, si se pulsa por equivocación sobre un sensor que ya ha sido anclado al robot o finalmente no se quiere eliminar, se puede cancelar la operación pulsando sobre el botón “Cancelar” del panel mencionado anteriormente.

Los puntos de anclaje resultan un elemento de gran importancia ya que, además de mostrar las posibles ubicaciones en las que se puede acoplar el sensor, permiten que este se coloque de una manera preestablecida en el script ***SnapController.cs****,* según la posición del punto de anclaje. De esta manera, si por ejemplo el sensor se coloca en la parte trasera del robot, se rotará automáticamente para que quede posicionado mirando hacia detrás, al igual que se rotará hacia un lado u otro si lo colocamos en algún lateral del robot. Dicho sistema de puntos de anclaje, aunque posee las ventajas mencionadas anteriormente, también necesita ser configurado manualmente para cada uno los robots, lo cual resulta un inconveniente a la hora de añadir futuros robots al proyecto, ya que será necesario llevar a cabo un proceso previo de configuración de los puntos de anclaje en cada uno de ellos.

Por último, resulta de interés mencionar el sistema de nomenclatura utilizado para los sensores. Cuando se ancla un sensor externo al robot, este se convierte en su hijo, por lo que si se quiere acceder posteriormente a él, mientras solamente haya uno de ese tipo, no habrá problema; no obstante, si se añaden varios sensores del mismo tipo, será necesario distinguirlos de alguna manera. Para ello, se ha empleado un sistema de nomenclatura que toma el valor de la etiqueta del *snap point* en el que se ha anclado y establece su nombre en base a ésta, ya que cada punto de anclaje posee una etiqueta diferente según la posición del robot en la que se encuentre ubicado. Dicho procedimiento, se lleva a cabo en el método ***SetSensorName*** de la clase ***SensorGeneric***, que transforma el nombre del GameObject añadiendo el tipo de sensor que es y su ubicación en el robot (véase la Fig. 3.10 y la Fig. 3.11). Este nombre es el que también se muestra en el panel lateral derecho que emerge tras pulsar el sensor en cuestión, tal y como se muestra en la Fig. 3.12

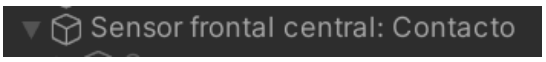


Fig. 3.10: Sensor de contacto frontal central en la jerarquía de la escena

Fig. 3.11: Robot con sensor de contacto en la posición frontal central

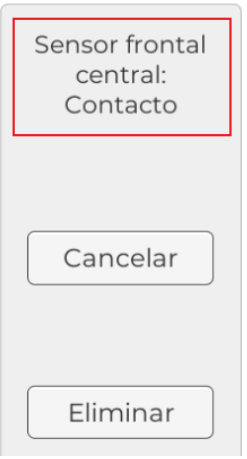


Fig. 3.12: Panel lateral derecho con el nombre del sensor

3.4.3.4 Funcionamiento del Robot

A lo largo de esta memoria, se ha reiterado en varias ocasiones el uso del robot y la importancia del papel que juega en el simulador; no obstante, no se ha llevado a cabo una explicación detallada de su funcionamiento y propiedades. Por ello, en este punto, se pretende abordar la descripción de la lógica empleada para simular su movimiento, la gestión de los sensores aplicados y sus características principales.

Todos los robots utilizados poseen 6 componentes:

- **Collider**. Necesario para detectar colisiones y otros eventos como los clics de ratón utilizados para rotar el robot en la escena de elección de sensores.
- **RigidBody**. Necesario para llevar a cabo el movimiento del robot, además de la detección de colisiones. No obstante, cabe destacar que el RigidBody del robot se encuentra por defecto en modo cinemático hasta que se lleva a cabo la resolución de un reto, ya que si se activa la física en las escenas de elección del robot o configuración de los sensores, con la mínima fuerza que se le aplique directa o indirectamente, es muy probable que el robot vaya a entrar en movimiento y que por consiguiente, desaparezca del campo de visión.
- ***RotateRobot.cs***. Es un script que permite llevar a cabo la rotación del robot en la escena de elección de sensores a través de la pulsación y arrastre del cursor sobre el robot, tal y como se ha mencionado anteriormente.
- ***SnapController.cs***. Es el script que contiene la lógica para llevar a cabo el acoplamiento de los sensores, tal y como se ha expuesto anteriormente.

- ***RobotManager.cs***. Este script contiene la implementación de la clase ***RobotManager***, desarrollada para gestionar el comportamiento del robot en general. Así pues, en ella se desempeñan funciones como el control del modo cinemático del RigidBody del robot, la activación del giróscopo, o el acceso a la información recogida por los sensores anclados, entre otras.
- ***RobotMotionController.cs***. Tal y como su nombre indica, es el script que contiene la lógica para ejecutar el movimiento del robot y todo lo relacionado con él.

Una vez detallados los componentes del robot, conviene profundizar en el último de ellos, pues es el que contiene el código que se ejecuta cuando el usuario utiliza bloques de la categoría "Mover". Antes de entrar en detalles, resulta pertinente aclarar, que el sistema de movimientos del robot ha sido otro módulo reimplementado por completo en esta versión de Roblockly. La principal diferencia con respecto a la versión inicial, radica en el uso de la función "MovePosition" del motor de físicas de Unity, la cual necesita que el robot cuente con un RigidBody para poder ser utilizada, tal y como se ha señalado anteriormente. Dicha función, ha sido empleada para mover el robot hacia delante y hacia detrás, ya que permite llevar a cabo el desplazamiento de un RigidBody desde una posición de origen a otra de destino. La integración de dicha función en el código, se ha realizado en el método ***MoveVerticalRobot*** (véase la Fig. 3.13) que a su vez es invocado desde el método ***MoveVerticalRobotInfinite*** (véase la Fig. 3.14), utilizado básicamente para poder llamar al método anterior desde una corrutina [27], ya que para llevar a cabo el movimiento del robot de manera visual y progresiva, es necesario que todos los métodos empleados para mover el robot, sean llamados desde una corrutina.

```
private void MoveVerticalRobot(float velocity, bool forward)
{
  Vector3 moveInput;
  if (forward)
  {
      moveInput = new Vector3(0, 0, DEFAULT_DISTANCE);
  } else { // If wants to move robot backwards;
      moveInput = new Vector3(0, 0, -DEFAULT_DISTANCE);
  }
      moveInput = transform.TransformDirection(moveInput); // Used to transform
      robot direction from local into world space.
      robotRigidbody.MovePosition(transform.position + moveInput *
      Time.deltaTime * velocity);
      RotateWheels(velocity);
}
```

Fig. 3.13: Método *MoveVerticalRobot* de la clase *RobotMotionController*

```
public IEnumerator MoveVerticalRobotInfinite(float velocity, bool forward)
{
   MoveVerticalRobot(velocity, forward);
   yield return null;
}
```

Fig. 3.14: Método *MoveVerticalRobotInfinite* de la clase *RobotMotionController*

Así mismo, tal y como en la versión inicial de la aplicación, se ha implementado la posibilidad de que el robot se pueda mover hacia delante o hacia detrás durante un tiempo determinado. Para conseguir esto, se codificado el método ***MoveVerticalRobotTime***, en el que se ha empleado una corrutina [25] de Unity que permite controlar el tiempo transcurrido entre una llamada y otra a la función ***MoveVerticalRobot*** (véase la Fig. 3.15).

```
public IEnumerator MoveVerticalRobotTime(float velocity, float timeToMove, bool
forward)
{
   if (timeToMove > 0)
   {
     float elapsedTime = 0.0f;
     while(elapsedTime <= timeToMove)
     {
        MoveVerticalRobot(velocity, forward);
        elapsedTime += Time.deltaTime;
        yield return null;
     }
   }
   yield return null;
}
```

Fig. 3.15: Método *MoveVerticalRobotTime* de la clase *RobotMotionController*

Por último, se debe detallar el movimiento de rotación del robot, ya que también se ha implementado de una manera diferente a la versión inicial de Roblocky. Con esta versión, el robot gira alrededor del eje vertical que pasa por su centro, lo cual no es del todo realista al tratarse de robots de cuatro ruedas, pero de cara a la programación de los retos, resulta más práctico. Así pues, para conseguir esto, se ha empleado la interpolación esférica entre dos vectores, a través de la función *Vector3.Slerp* [28] de Unity que permite rotar el robot con un ángulo concreto desde una posición de origen. Dicha función ha sido empleada en el método ***RotateRobot*** que recibe el valor del ángulo a rotar y la dirección del mismo (véase la Fig. 3.16).

```
public IEnumerator RotateRobot(float angleToRotate, string directionToRotate)
{
   if (directionToRotate == LEFT)
   {
       angleToRotate *= -1; // Changes to negative the selected Angle;
   }
   angleRotated += angleToRotate;
   Quaternion goal = Quaternion.Euler(0, angleRotated, 0);
   while (Quaternion.Angle(transform.rotation, goal) > 1.0f)
   {
       transform.rotation = Quaternion.Slerp(transform.rotation,
       Quaternion.Euler(0, angleRotated, 0), Time.deltaTime);
       yield return null;
   }
   transform.rotation = Quaternion.Euler(0, angleRotated, 0);
   yield return null;
}
```

Fig. 3.16: Método *RotateRobot* de la clase *RobotMotionController*

3.4.3.5 Funcionamiento de los sensores

Para la versión de Roblockly elaborada en este Trabajo de Fin de Grado, se planteó inicialmente la posibilidad de agregar 3 nuevos sensores: sensor de color, giróscopo y micrófono. No obstante, la propuesta del micrófono se hizo pensando en la posibilidad de implementar también retos colectivos de varios robots, de tal manera que se utilizara fundamentalmente para la comunicación entre dichos robots dentro de un mismo reto. Finalmente, se descartó la elaboración y desarrollo del módulo de retos colectivos y por lo tanto, la implementación del micrófono como sensor, careció de sentido. A pesar de ello, se ha dejado preparada la interfaz de elección de sensores y parte del código, de cara a futuras versiones de Roblockly.

Además de la integración del sensor de color y el giróscopo como nuevos sensores, también se reimplementó por completo la lógica del sensor de Ultrasondio. Así pues, sólo se han considerado como motivo de interés la explicación del funcionamiento del sensor de ultrasonido, giróscopo y color, puesto que el código de los sensores de contacto e infrarrojos se refactorizó, pero su funcionamiento sigue siendo el mismo que el de la versión inicial de Roblockly.

Antes de profundizar en el funcionamiento de los sensores nombrados previamente, cabe mencionar que cada GameObject que se instancia en la escena de elección de sensores cuando el usuario clica sobre una de las imágenes del panel lateral izquierdo, es un prefab del tipo de sensor externo que se pretende anclar, ubicado en el directorio *ProjectPrefabs/Sensors.* A cada uno de estos prefabs, se les ha aplicado una serie de componentes que también determinan el funcionamiento del sensor:

- **Collider**. Cada sensor cuenta con algún tipo de collider, pues para la resolución de ciertos retos es importante que pueda detectar colisiones.
- **RigidBody**. Necesario para que durante la configuración del mismo, se pueda clicar sobre él y desplegar el panel lateral derecho, además de la detección de colisiones. Al igual que el robot, los sensores tienen por defecto el RigidBody en modo cinemático, ya que de no ser así, al llevar a cabo el anclaje de los sensores en la escena de elección de sensores, el RigidBody cinemático del robot entraría en conflicto con el RigidBody no cinemático del sensor.
- Un script denominado ***DragObject.cs**,* que contiene la lógica para que el sensor pueda ser "arrastrado y soltado" con el ratón por el usuario.
- Un script específico denominado ***SensorTouch.cs**, **SensorUS.cs**, **SensorIR.cs*** y ***SensorColor.cs*** según el tipo de sensor seleccionado. Las clases implementadas en cada uno de estos scripts, heredan de la clase base desarrollada en el script ***SensorGeneric.cs***, que contiene una serie de métodos iguales para todos los sensores, entre los que se encuentran el método para establecer el nombre del sensor o eliminar el sensor del robot, entre otros.

Una vez aclarado cuáles son los componentes básicos de los sensores externos, se procede especificar el funcionamiento de los mencionados previamente:

Color

Para este sensor, se ha empleado la misma lógica utilizada con el sensor de infrarrojos, puesto que la única diferencia es el hecho de que el sensor de infrarrojo detecta solamente blanco y negro y el de color, un espectro más amplio. Así pues, en la implementación realizada para este TFG, se ha establecido que el sensor de color sea capaz de detectar rojo, azul, amarillo y verde. Para reproducir dicho comportamiento, se ha establecido el lanzamiento de un raycast [29] desde el led verde del sensor de tal manera que al impactar con un material, permite acceder al nombre o color del mismo. Estos materiales que representan los colores detectados, se encuentran en el directorio *Textures/Materials/Scenarios/ColorChallenge.*

Ultrasonido

Este sensor se utiliza para medir distancias, a través de la emisión de un sonido por medio de un altavoz y la medición del tiempo que tarda en volver con un micrófono. Por motivos obvios, esto no se puede implementar de manera exacta en Unity y es por ello que se ha tenido que simular a través de otros mecanismos.

En la primera versión de roblockly, el funcionamiento de dicho sensor, se basó en el lanzamiento de raycasts [29] dentro de tres bucles "for" anidados, ejecutados en el método *Update* de la clase que gestiona el cálculo de las distancias detectadas con este sensor. Esto suponía un elevado coste computacional ya que

su complejidad era $O(n^3)$, por lo que se planteó una mejora que aporta una solución más eficiente.

Así pues, para esta nueva implementación, se modificó el prefab del sensor de ultrasonido y se le añadió un GameObject con forma de cono en la parte delantera (véase la Fig. 3.17) con el fin de desactivar su renderizado y utilizarlo únicamente para detectar colisiones (véase la Fig. 3.18). De esta manera, cuando el collider del cono colisiona con algún objeto, se toma la posición de dicho objeto y se calcula la distancia entre el objeto y el sensor. Esta lógica, se encuentra implementada en el script *ConeCollider*, aplicado al propio cono del prefab del sensor de ultrasonido.

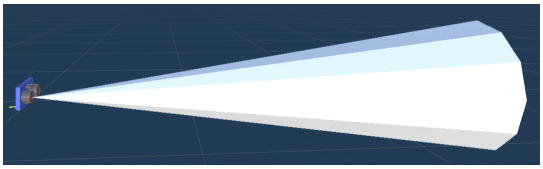


Fig. 3.17: Sensor ultrasonido con cono en la parte delantera

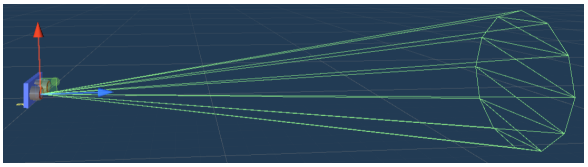


Fig. 3.18: Sensor ultrasonido con cono en la parte delantera sin renderizado (sólo collider)

Giróscopo

La implementación del giróscopo se ha desarrollado en base al reto de “La plataforma móvil”, ya que para su simulación se precisa de la esfera situada bajo dicha plataforma, y por ello, no es posible utilizar este sensor en otros retos. En él, se pretende que el robot sea capaz de moverse por la plataforma hasta encontrar el equilibrio en el centro de la misma y mantenerse un tiempo suficiente en la zona “roja” que representa el centro de la plataforma.

Para simular el comportamiento de dicho sensor, se ha trazado un vector perpendicular al eje X de la esfera en la que se apoya la plataforma y otro desde el centro de la esfera hasta el centro del robot (véase la Fig 3.19), de tal manera que, conociendo estos vectores, se puede calcular el ángulo de desviación del robot respecto al centro de la plataforma y por lo tanto, determinar si este está inclinado hacia atrás o hacia delante. Asimismo, con el fin de controlar que el robot solamente se mueva hacia delante o hacia atrás cuando usa el giróscopo, se ha eliminado la posibilidad de que el usuario pueda utilizar los bloques para rotar el robot. La codificación de dicho algoritmo se encuentra en el script ***SensorGyroscope.cs***.

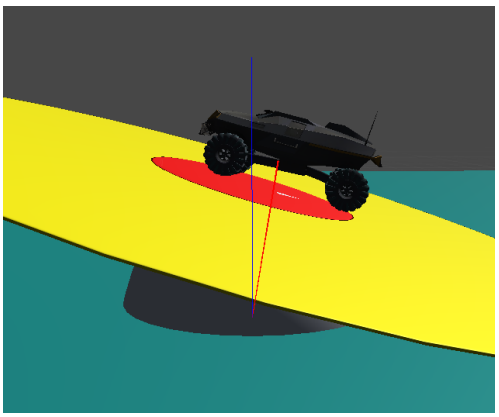


Fig. 3.19: Vectores trazados para calcular la inclinación del robot

3.4.3.6 Retos

Los escenarios de los retos de la versión inicial de Roblockly, son el material que más se ha podido reutilizar en la elaboración de esta nueva versión de la aplicación. Así pues, salvo en el caso del reto de “La plataforma móvil” y el del “Sendero de colores”, el resto de escenarios para los retos se han mantenido prácticamente iguales a los de la versión inicial de la aplicación. Así mismo, también se han integrado los botones “+” y “-” (véase la Fig. 3.20), desarrollados en la versión inicial de Roblockly para aumentar o disminuir el tamaño de los bloques utilizados, así como la lógica que controla que sólamente se muestren en la categoría de “sensores”, aquellos bloques que correspondan a los sensores añadidos al robot.

En cuanto a la interfaz de usuario implementada en las escenas de los distintos retos, no se han realizado cambios notables, más allá de los botones para la depuración del código ya mencionados en la sección 3.3 de la presente memoria. Así pues, en el resto de la interfaz, sólo se ha añadido el temporizador y el botón de menú general, que se explicará más adelante, además de la nueva distribución de colores llevada a cabo mediante la aplicación de la identidad corporativa ya mencionada en el punto 3.4.2.

Fig. 3.20: Botones para maximizar o minimizar el tamaño de los bloques

Por otra parte, aunque ya se han mencionado algunos retos elaborados para algún tipo de sensor concreto, resulta conveniente exponer brevemente un listado de todos ellos, detallando sucintamente sus principales características.

1. *Reto del laberinto*. Este reto consiste en que el robot recorra un laberinto hasta “capturar” una moneda que se encuentra en el final del mismo. Al igual que en la primera versión de Roblockly, se pretende que este reto sea

superado mediante el uso de los sensores de ultrasonido y de contacto (véase la Fig.3.21)

2. *Reto del sendero blanco*. Pensado para ser superado con el uso del sensor infrarrojo, este reto consiste en que el robot sea capaz de de recorrer un sendero formado por líneas blancas sobre un fondo negro, hasta llegar al final del mismo y "capturar" la moneda (véase la Fig.3.22).
3. *Reto del sendero de colores.* Está pensado para ser superado con el sensor de color, por lo que su planteamiento es el mismo el reto del sendero blanco. Así pues, en vez recorrer un sendero formado por líneas blancas y un fondo negro, debe seguir un sendero formado por distintos colores que tendrá que identificar y en base a ello, realizar ciertos movimientos que le permitan llegar al final del camino y "capturar la moneda" (véase la Fig.3.23).
4. *Reto de la plataforma móvil*. Es un reto que sólamente puede ser resuelto con el sensor giróscopo y precisa que el robot no posea ningún otro sensor acoplado a la carrocería. En él se pretende que, dado una posición inicial, el robot sea capaz de alcanzar el centro de la plataforma y mantener el equilibrio en la zona "roja" (véase la Fig.3.24).

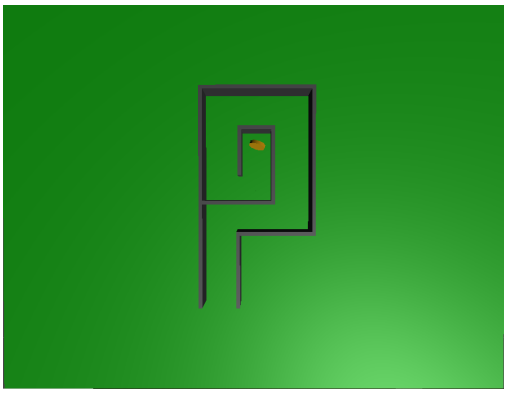


Fig. 3.21: Reto del laberinto

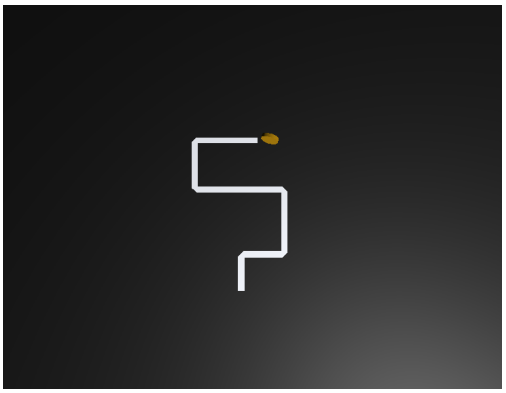


Fig. 3.22: Reto del sendero blanco

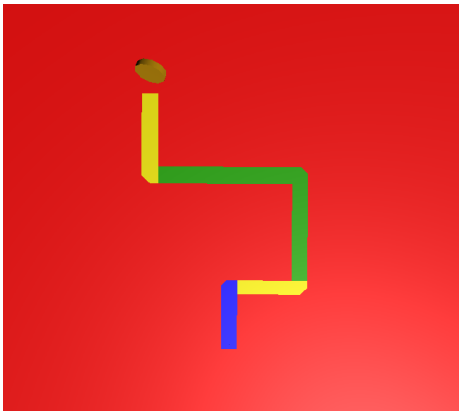


Fig. 3.23: Reto del sendero de colores

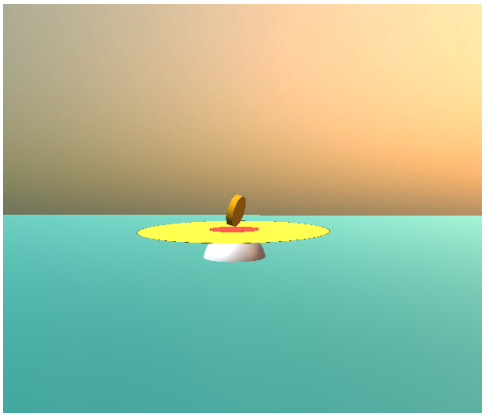


Fig. 3.24: Reto de la plataforma móvil

Una vez que ya se ha detallado el funcionamiento del movimiento del robot y los distintos tipos de sensores, resulta de interés especificar los bloques creados para cada uno de ellos.

Bloques para el movimiento

En cuanto a los bloques para llevar a cabo el movimiento, se observan los mismos que en la versión inicial de Roblockly. Así pues, tal y como se muestra en la Fig. 3.25, con estos bloques, el robot puede moverse hacia delante y atrás durante un tiempo determinado o no, rotar sobre sí mismo y detenerse.

Fig. 3.25: Bloques creados para el movimiento del robot

Bloques para los sensores

Con respecto a los sensores, salvo los bloques del sensor de color y el giróscopo, el resto son iguales a los ya implementados en la primera versión de Roblockly. En consecuencia, la versión actual de la aplicación cuenta con los siguientes bloques para los sensores:

- **Contacto:** Dos bloques que permiten detectar si el sensor ha chocado o no con algún elemento (véase la Fig. 3.26).
- **Ultrasonido:** Un bloque que devuelve la distancia del sensor con respecto al objeto con el que ha chocado (véase la Fig. 3.27).
- **Infrarrojo:** Dos bloques que permiten comprobar si el sensor ha detectado color blanco o negro (véase la Fig. 3.28).
- **Color:** Un bloque que permite comprobar si el sensor ha detectado rojo, amarillo, verde o azul (véase la Fig. 3.29).
- **Giróscopo:** Un bloque que permite verificar si el robot no está inclinado y otro que comprueba si el robot está inclinado hacia delante o hacia detrás (véase la Fig. 3.30).

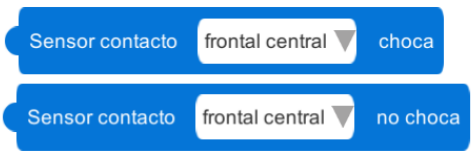


Fig. 3.26: Bloques para el sensor de contacto

Fig. 3.27: Bloque para el sensor de ultrasonido

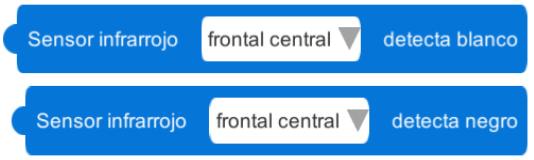


Fig. 3.28: Bloque para el sensor de infrarrojos

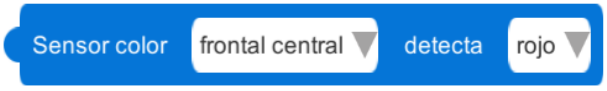


Fig. 3.29: Bloque para el sensor de color

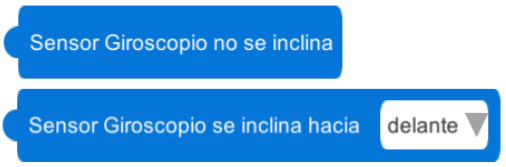


Fig. 3.30: Bloque para el sensor giróscopo

Por último, cabe destacar que a pesar de que la funcionalidad de la mayoría de bloques es la misma que la de la primera versión de Roblockly, la implementación de los mismos ha cambiado ligeramente debido a que, tal y como se explicó, para distinguir los distintos sensores acoplados al robot, se utiliza la localización en la que ha sido colocado cada uno. Así pues, si por ejemplo el usuario ha colocado un sensor de contacto en el punto de anclaje del lateral derecho, en el bloque correspondiente, deberá elegir la opción “derecho” tal y como se muestra en la Fig. 3.31. Asimismo, si elige un sensor en una posición del robot en la que no hay nada, emergerá un mensaje de error en la pantalla que le indicará al usuario que no se puede elegir dicho sensor y parará la ejecución del simulador (véase la Fig. 3.32).

Fig. 3.31: Elección del sensor del lado derecho

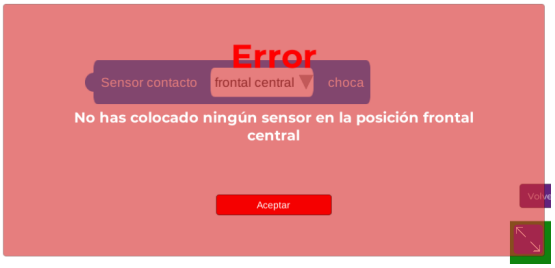


Fig. 3.32: Error de elección de posición en bloque de sensor

3.4.3.7 Módulo estadístico

Como ya se explicó en la sección 1.3, uno de los objetivos propuestos para el desarrollo de este Trabajo de Fin de Grado era esbozar el prototipo de un módulo de estadísticas que permita recoger información sobre la soluciones diseñadas por el usuario y hacer posible un seguimiento de su progreso con el uso de la misma. Así pues, en esta primera versión de dicho módulo, se ha implementado un sistema de

recolección de estadísticas básico, que permite almacenar para cada reto resuelto, el menor tiempo que ha tardado el estudiante en resolverlo, la solución con menos bloques y el tiempo promedio que ha tardado las distintas veces que lo ha resuelto. Además, en la sección de “solución con menos bloques”, también se indica el tiempo consumido para la resolución, los bloques utilizados y el porcentaje de acercamiento a la solución óptima (véase la Fig. 3.33). Esto último, se calcula comparando el número de bloques empleados con unos valores preestablecidos en el código que indican el menor número de bloques que pueden ser utilizados para resolver cada reto.

Se puede acceder al panel de estadísticas desde cualquier escena a través del menú de configuración, que se muestra al pulsar el botón de ajustes (véase la Fig. 3.34). Dicho menú, ha sido otra de las novedades añadidas a la interfaz del proyecto, y permite acceder a las estadísticas, volver al menú de inicio o salir de la aplicación (véase la Fig. 3.35). Para ver el los datos estadísticos mencionados anteriormente, simplemente hay que pulsar sobre el botón “Estadísticas” del menú de configuración y una vez ahí, seleccionar el botón del reto completado que se quiera visualizar (véase la Fig. 3.36). El código para gestionar la lógica de dicho menú de configuración, se ha desarrollado en la clase ***SettingsMenu*** implementada en el script ***SettingsMenu.cs***.

Por otra parte, la lógica del módulo de estadísticas se ha desarrollado a través de la clase implementada en el fichero ***StatisticsManager.cs***, que a su vez, ha sido aplicado al GameObject *StatisticsManager* de la escena de elección del robot. Dicho objeto, prevalece entre las diferentes escenas gracias a la función *DontDestroyOnLoad* de Unity [30], permitiendo que se pueda acceder a él para consultar o guardar información desde cualquier escena de la aplicación. La recopilación de datos, se lleva a cabo cuando el robot “atrapa” la moneda que se encuentra al final del reto, ya que se dispara un evento que permite recoger toda la información necesaria detallada previamente. Dicho evento es lanzado desde la clase ***CoinBehaviour***, implementada en el script ***CoinBehaviour.cs***, añadido al GameObject de la moneda que se encuentra al final de cada reto.

Para concluir, cabe mencionar que la interfaz también cuenta con un temporizador, que comienza a funcionar desde que el usuario entra en la escena del reto, y para cuando se dispara el evento mencionado anteriormente. La lógica de dicho temporizador ha sido implementada en el fichero ***TimerBehaviour.cs***, aplicado al GameObject *TimerPanel* (véase la Fig. 3.37) del canvas de la escena de cada reto.

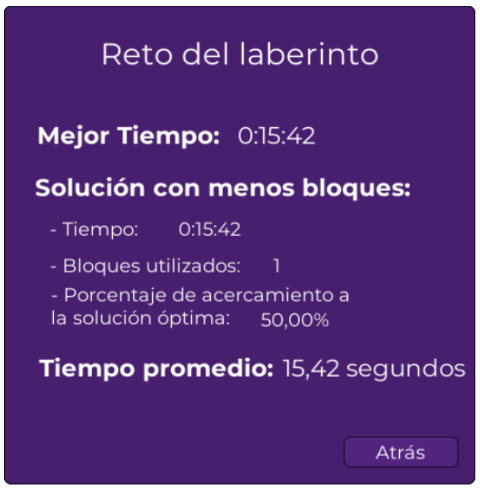


Fig. 3.33: Interfaz del módulo estadístico

Fig. 3.34: Botón de ajustes

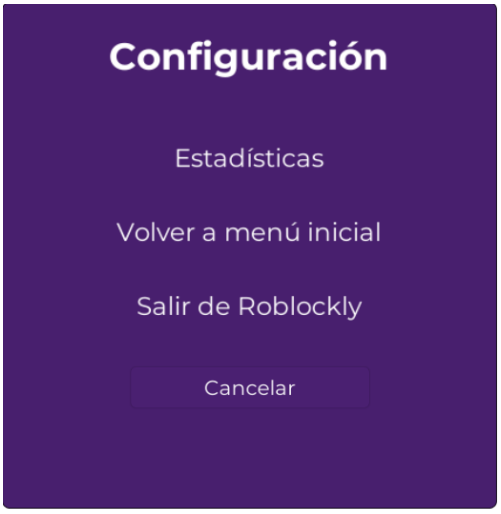


Fig. 3.35: Menú de configuración

Fig. 3.36: Interfaz del menú de estadísticas

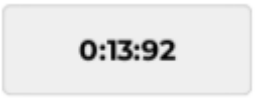


Fig. 3.37: Temporizador de interfaz de los retos

Capítulo 4: Caso de uso

Una vez explicado el funcionamiento de los distintos elementos que conforman la aplicación, resulta de interés, llevar a cabo un caso de uso, explicando paso a paso un ejemplo práctico de ejecución y resolución de un reto.

Así pues, en primer lugar, debemos seleccionar la opción de “Reto individual” en el menú inicial de la aplicación (véase la Fig. 6.1).

Fig. 6.1: Menú inicial de Roblockly

Seguidamente, se debe elegir el robot que se vaya a utilizar para la resolución del reto. Para ello, es posible visualizar las distintas opciones de robots disponibles, pulsando en los botones en forma de flecha que se encuentran en los laterales de la pantalla (véase las Fig. 6.2, 6.3, 6.4 y 6.5). Así pues, una vez que se haya seleccionado el robot deseado, simplemente se le debe pulsar sobre el botón “continuar” que se encuentra en la parte inferior de la pantalla. Para este caso de uso, se ha utilizado el Robot 4 (véase la Fig. 6.5).

Fig. 6.2: Robot 1

Fig. 6.3: Robot 2

Fig. 6.4: Robot 3

Fig. 6.5: Robot 4

Una vez elegido el robot que se va a utilizar, se transita a la escena de elección del reto, la cual muestra el nombre y una captura de pantalla de los distintos retos disponibles (véase la Fig. 6.6). Para hacer la selección, simplemente se debe pulsar sobre la imagen del reto a resolver y seguidamente, sobre el botón “continuar”.

Fig. 6.6: Menú para la elección del reto

En tercer lugar, se lleva a cabo la configuración del robot para la resolución del reto mediante el anclaje de los sensores que se estimen oportunos. En este caso, se ha propuesto solucionar el “Reto del sendero blanco”, el cual está pensado para ser superado mediante el uso del sensor infrarrojo. Concretamente, la idea es colocar dos sensores infrarrojos en los puntos de anclaje frontal derecho y frontal izquierdo respectivamente, de tal manera que mientras ambos sensores detecten el color blanco, el robot avanzará hacia delante, sin embargo, desde que uno de ellos detecte el negro, el robot parará y realizará un giro de noventa grados hacia el lado que se le indique, con la intención de que ambos sensores detecten de nuevo el color blanco y, por consiguiente, el robot pueda continuar avanzando hacia delante hasta llegar a la moneda (veáse la fig. 6.7).

Como se ha mencionado previamente, para añadir al robot los sensores que se vayan a utilizar, se puede pulsar y arrastrar el ratón sobre el mismo para así rotarlo y poder acceder más fácilmente a los distintos puntos de anclaje. Por lo tanto, una vez encontrados los puntos deseados, simplemente es necesario pulsar sobre la imagen del sensor de infrarrojos, y arrastrar el sensor de infrarrojos que aparece en la posición del cursor del ratón hasta el *snap point* deseado.

Después de acoplar al robot los sensores necesarios y pulsar el botón de “continuar”, se carga la escena de simulación (véase la Fig. 6.8) en la que se lleva a cabo la programación y ejecución del algoritmo diseñado. Así pues, para programar

al robot, se deberá ir colocando y encajando entre sí los diferentes bloques que se vayan a emplear, pulsando sobre las distintas categorías que se muestran en la columna lateral izquierda y arrastrando a la zona del medio los bloques deseados (véase la Fig. 6.9).

Fig. 6.7: Menú de selección y acople de sensores

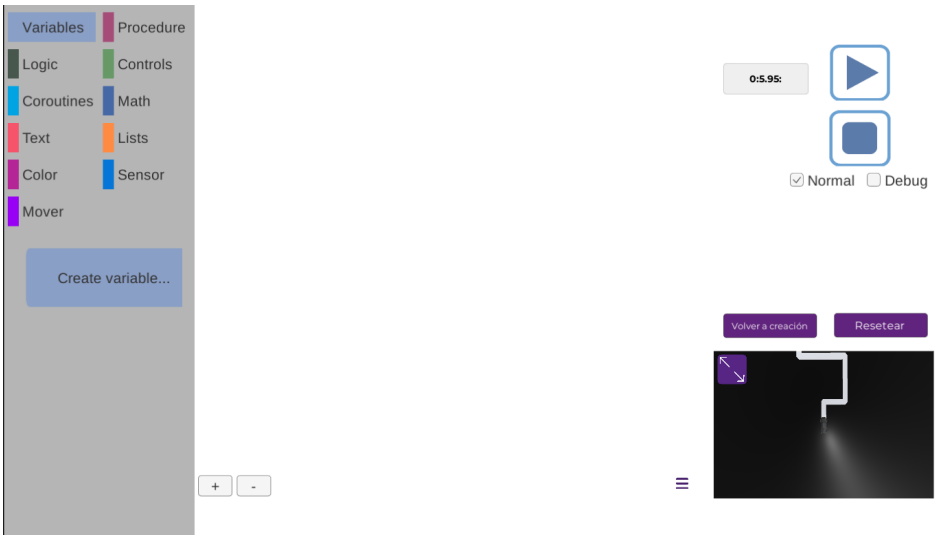


Fig. 6.8: Escena de programación y simulación del reto

Cuando ya se ha programado el algoritmo, se procede a la ejecución del mismo. Para ello, simplemente es necesario hacer click sobre el botón de ejecución (véase la Fig. 3.1) y visualizar el comportamiento del robot a través de la ventana de visualización del reto que se encuentra en la esquina inferior derecha. En caso de querer visualizar la simulación de manera más precisa, simplemente será necesario pulsar sobre el botón de ampliación de ventana que se encuentra en la esquina superior izquierda de la misma, de tal manera que la ventana de visualización del reto se mostrará con mayo tamaño en el medio de la pantalla (véase la Fig. 6.10).

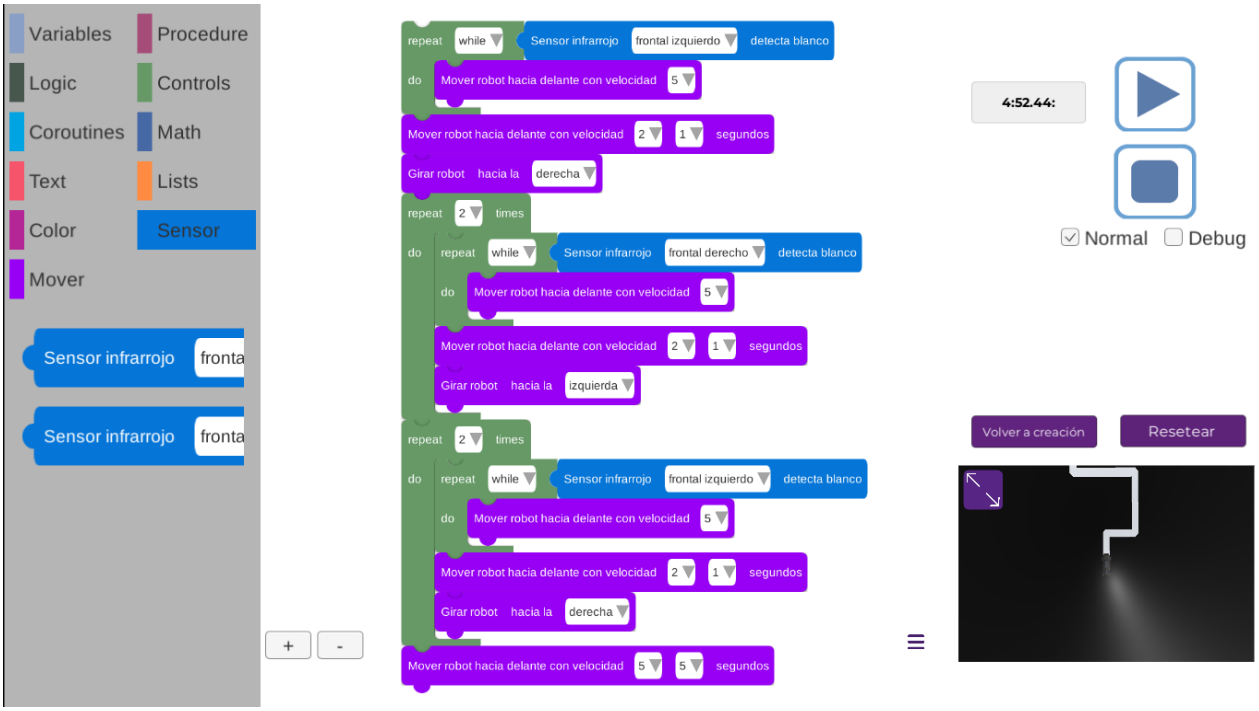


Fig. 6.9: Programación de la solución mediante bloques

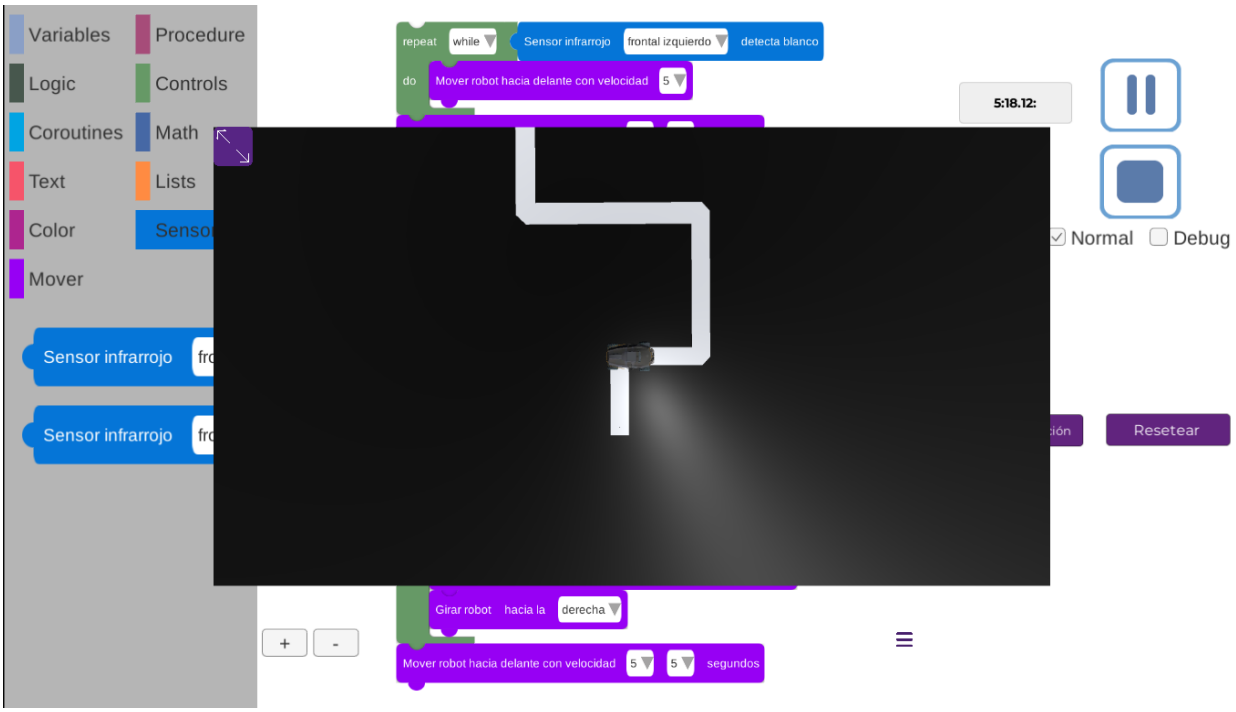


Fig. 6.10: Ampliación de la ventana de visualización de la simulación

Por último, una vez que el robot supera el reto “atrapando” la moneda del final del camino blanco, se detiene el temporizador (véase la Fig. 6.11), y entonces es posible consultar los datos estadísticos de la solución diseñada para resolver el reto. Para ello, se debe pulsar sobre el botón de ajustes (véase la Fig. 3.35) que se encuentra en la esquina inferior derecha de cada una de las escenas mencionadas previamente y, seguidamente, en el botón de “estadísticas” del menú de configuración que se muestra (véase la Fig. 3.36). Una vez ahí, se debe pulsar sobre el botón del reto resuelto, que en este caso ha sido el “Reto del sendero blanco” (véase la Fig. 6.12), y acceder al panel con la información de la resolución llevada a cabo, tal y como se muestra en la Fig. 6.13.

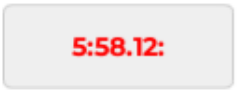


Fig. 6.11: Temporizador detenido tras “atrapar” la moneda

Fig. 6.12: Panel con el botón del reto resuelto

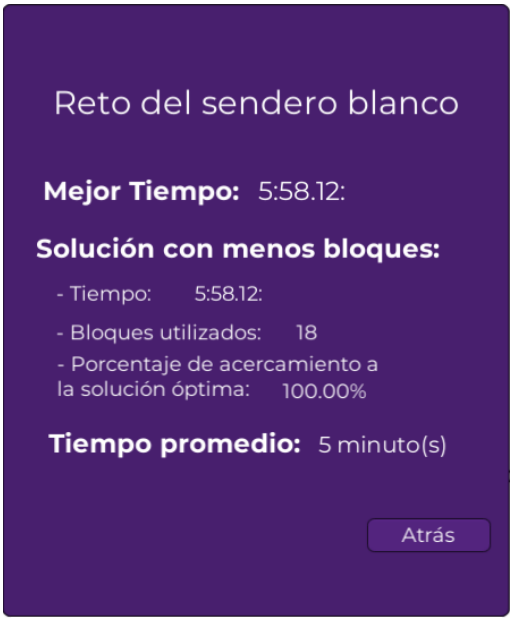


Fig. 6.13: Panel con los datos estadísticos del reto resuelto

Capítulo 5: Conclusiones y líneas futuras

Tras la elaboración de este Trabajo de Fin de Grado, se puede afirmar que se ha cumplido con los objetivos establecidos, puesto que el resultado ha sido una nueva versión ampliada y mejorada de la aplicación Roblockly.

Asimismo, aunque no era uno de los objetivos propuestos inicialmente, el hecho de haber cambiado toda la interfaz gráfica de usuario por un diseño semejante al de un videojuego, podría ser considerado como un punto de motivación para los estudiantes a la hora de utilizar la aplicación, ya que fomentaría el interés de cada uno por el mundo de la robótica educativa, a la vez que desarrollarían el pensamiento computacional.

Por otra parte, aunque en este TFG se ha elaborado una nueva versión de Roblockly con nuevas funcionalidades y características ya expuestas a lo largo de esta memoria, aún quedan abiertos varios frentes de trabajo que podrían ser abordados en futuras versiones del simulador. Entre ellos, se encuentran la posibilidad de programar varios robots en un mismo escenario, de tal manera que puedan interactuar a través del sensor de color, ya implementado en este TFG, o el desarrollo del micrófono como sensor, tal y como se ha explicado al comienzo del punto 3.4.3.5. Otra funcionalidad que se podría añadir en futuras versiones, es la creación de un sistema de perfiles de usuarios, que permita a cada uno almacenar las distintas soluciones que ha ido programando para superar los diversos retos propuestos. De esta manera, también se abrirían las puertas a la ampliación del módulo de estadísticas, ya que se podría recoger una mayor cantidad de información que permitiría llevar a cabo un seguimiento más exhaustivo de la evolución del usuario en la aplicación, y por lo tanto, del desarrollo de su pensamiento computacional.

Para concluir, conviene mencionar de nuevo el valor que aporta, el hecho de que la aplicación sea accesible de manera gratuita a través de internet, ya que con la llegada de la pandemia del COVID-19, se ha puesto de manifiesto la utilidad e importancia de herramientas y servicios online que faciliten la docencia virtual.

Capítulo 6: Summary and Conclusions

After having finished this Degree Final Dissertation, we can conclude that the established objectives have been accomplished, as the result has been a new, expanded and improved version of the Roblockly application.

Likewise, although it was not one of the initial objectives, the fact of having changed the entire graphical user interface for a design similar to that of a video game could be considered motivating for students when using the application. Thus, this would promote their interest in educational robotics while developing computational thinking.

On the other hand, although a new version of Roblockly has been developed in this Dissertation, with new functionalities and characteristics already exposed throughout this essay, several work fronts are still open, which could be addressed in future versions of the simulator. Among them, there is the possibility of programming several robots in the same scenarios, in such a way that they can interact through the colour sensor, already implemented in this Dissertation, or the development of the microphone as a sensor, as explained at the beginning of point 3.4.3.5. Another functionality that could be added in future versions is the creation of a user profile system, which will allow the storage of different solutions programmed to overcome the various challenges proposed. In this way, this would make the expansion of the statistics module possible, since a greater amount of information could be collected, allowing a more exhaustive monitoring of the user's evolution in the application and, therefore, of the development of their computational thinking.

To conclude, it is worth mentioning its valuable contribution again, the fact that the application is freely accessible through the Internet, as, with the arrival of the COVID-19 pandemic, the usefulness and importance of online tools and services that facilitate virtual teaching is obvious.

Capítulo 7: Presupuesto

A continuación se muestra una tabla con el desglose de tareas realizadas, horas invertidas en su realización y coste de cada una. Así mismo, cabe destacar que para la elaboración de este presupuesto se ha establecido un precio de **20€ por hora** y no se ha tenido en cuenta el precio del equipo utilizado.

Tarea	Horas empleadas	Coste
Planteamiento y desarrollo de mejoras y nuevas funcionalidades para la aplicación.	24h	480€
Búsqueda de nuevos modelos de robots a integrar y construcción del módulo para la elección del robot que se vaya a utilizar.	32h	640€
Estudio de nuevos sensores posibles y creación del módulo para añadirlos al robot.	64h	1.280€
Reimplementación del código de movimiento del robot.	32h	640€
Implementación del código de funcionamiento de los nuevos sensores y reimplementación del código del sensor de ultrasonido.	40h	800€
Creación de retos	18h	360€
Creación de la interfaz para elección del reto	10h	200€
Creación del menú de configuración	16h	320€
Propuesta e implementación de un prototipo de módulo estadístico.	24h	480€
Elaboración de la memoria del trabajo.	64h	1.280€
Total	324h	6.480€

Tabla 7.1: Presupuesto del proyecto

Capítulo 8: Apéndice 1: Tablas comparativas

	Roblockly				
	Programación de un robot	Pensado para las aulas	Programación visual por bloques	Acceso gratuito	Acceso en línea
Jabutí EDU Nube	X	X			X
ROBOT-EDULPGC		X			
Open Roberta Lab	X		X	X	X
Zowi	X		X		

Tabla 8.1: Comparativa entre características de Roblockly y aplicaciones mencionadas en el punto 2.2.2

Capítulo 9: Referencias y bibliografía

9.1 Referencias

[1] M. Zapata-Ros, “Pensamiento computacional: Una nueva alfabetización digital,” *Revista de Educación a Distancia (RED)*, no. 46, sep. 2015. [En línea]. Disponible en: https://www.um.es/ead/red/46/zapata.pdf

[2] K. Pittí Patiño, I. Moreno, L. Muñoz, J. R. Serracín, J. Quintero, and J. Quiel, “La robótica educativa, una herramienta para la enseñanza-aprendizaje de las ciencias y las tecnologías,” *Education in the Knowledge Society (EKS)*, vol. 13, no. 2, pp. 74–90, jul. 2012.

[3] “Pensamiento computacional,” *Wikipedia*, ag. 13, 2021. [En línea]. Disponible en: https://es.wikipedia.org/wiki/Pensamiento_computacional (Accedido: 13-my-2021).

[4] J. M. Wing, “Computational thinking,” *Communications of the ACM*, vol. 49, no. 3, pp. 33–35, Mar. 2006.

[5] B. Ortega-Ruipérez and M. M. A. Brouard, “Robótica DIY: pensamiento computacional para mejorar la resolución de problemas,” *Revista Latinoamericana de Tecnología Educativa - RELATEC*, vol. 17, no. 2, pp. 129–143, dic. 2018. [En línea]. Disponible en: https://relatec.unex.es/article/view/3313

[6] “Robótica educativa,” *Wikipedia*, jul. 19, 2021. [En línea]. Disponible en: https://es.wikipedia.org/wiki/Rob%C3%B3tica_educativa (Accedido: 15-my-2021).

[7] C. M. Ángel-Díaz, E. Segredo, R. Arnay, and C. León, “Simulador de Robótica Educativa para la promoción del Pensamiento Computacional,” *Revista de Educación a Distancia (RED)*, vol. 20, no. 63, abr. 2020. [En línea]. Disponible en: http://dx.doi.org/10.6018/red.410191

[8] “Introduction to Blockly ,” *Google Developers*. [En línea]. Disponible en: https://developers.google.com/blockly/guides/overview (Accedido: 16-my-2021).

[9] L. Kucuk and L. Añais, “Jabutí EDU, Plataforma de Robótica Educativa IOT open Hardware,” Apr. 2021.

[10] D. Aponte Núñez *et al.*, “Integración del diseño e implementación de la electrónica de una plataforma robótica educativa multidisciplinar como soporte al aprendizaje,” jul. 2020. [En línea]. Disponible en: http://dx.doi.org/10.4995/inred2020.2020.11939. (Accedido: 16-my-2021).

[11] A. Martí Gil *et al.*, “Proceso de aprendizaje en la fabricación integrada de una plataforma robótica educativa multidisciplinar,” jul. 2020. [Online]. Disponible en: http://dx.doi.org/10.4995/inred2020.2020.11960. (Accedido: 16-my-2021).
[12] “MINDSTORMS®,” *The LEGO® Group*. https://www.lego.com/es-es/themes/mindstorms (Accedido: 16-my-2021).
[13] J. C.- botnroll.com, “Bot’n Roll ONE A Robot.” http://botnroll.com/onea_en/ (Accedido: 16-my-2021).
[14] “Micro:bit Educational Foundation,” *Micro:bit*. https://www.microbit.org/ (Accedido: 16-my-2021).
[15] “CALLIOPE.” https://calliope.cc/es/ (Accedido: 16-my-2021).
[16] “Open Roberta,” *Wikipedia*, Jul. 03, 2021. [En línea]. Disponible en: https://en.wikipedia.org/wiki/Open_Roberta (Accedido: 16-my-2021).
[17] J. Penalva, “Probamos Zowi, un robot con cerebro Arduino que puede dar más de lo que aparenta,” *Xataka*, nov. 02, 2019. [En línea]. Disponible en: https://www.xataka.com/analisis/probamos-zowi-un-robot-con-cerebro-arduino-que-puede-dar-mas-de-lo-que-aparenta (Accedido: 16-my-2021).
[18] “Unity (game engine),” *Wikipedia*, Sep. 03, 2021. [En línea]. Disponible en: https://en.wikipedia.org/wiki/Unity_(game_engine) (Accedido: 18-my-2021).
[19] “WebGL,” *Wikipedia*, jun. 11, 2021. [En línea]. Disponible en: https://es.wikipedia.org/wiki/WebGL (Accedido: 18-my-2021).
[20] U. Technologies, “Unity - Scripting API: Scene.” [En línea]. Disponible en: https://docs.unity3d.com/ScriptReference/SceneManagement.Scene.html (Accedido: 18-my-2021).
[21] U. Technologies, “Unity - Scripting API: Collider.” [En línea]. Disponible en: https://docs.unity3d.com/ScriptReference/Collider.html (Accedido: 20-my-2021).
[22] “Git.” [En línea]. Disponible en: https://git-scm.com/ (Accedido: 21-my-2021).
[23] “GitHub,” *Wikipedia*, jul. 15, 2021. [En línea]. Disponible en: https://es.wikipedia.org/wiki/GitHub (Accedido: 21-my-2021).
[24] “Indentation style,” *Wikipedia*, Aug. 12, 2021. [En línea]. Disponible en: https://en.wikipedia.org/wiki/Indentation_style#Allman_style (Accedido: 21-my-2021).
[25] “Principio de responsabilidad única,” *Wikipedia*, Nov. 03, 2020. [En línea]. Disponible en:

https://es.wikipedia.org/wiki/Principio_de_responsabilidad_%C3%BAnica (Accedido: 21-jun-2021).
[26] “Color, Marca” *Universidad de La Laguna*. [En línea]. Disponible en: https://www.ull.es/portal/marca/color/ (Accedido: 27-jun-2021).
[27] U. Technologies, “Unity - Manual: Coroutines.” [En línea]. Disponible en: https://docs.unity3d.com/Manual/Coroutines.html (Accedido: 28-jul-2021).
[28] U. Technologies, “Unity - Scripting API: Vector3.Slerp.” [En línea]. Disponible en: https://docs.unity3d.com/ScriptReference/Vector3.Slerp.html (Accedido: 28-jul-2021).
[29] U. Technologies, “Unity - Scripting API: Physics.Raycast.” [En línea]. Disponible en: https://docs.unity3d.com/ScriptReference/Physics.Raycast.html (Accedido: 29-jul-2021).
[30] U. Technologies, “Unity - Scripting API: Object.DontDestroyOnLoad.” [En línea]. Disponible en: https://docs.unity3d.com/ScriptReference/Object.DontDestroyOnLoad.html (Accedido: 09-ag-2021).

9.2 Bibliografía

[31] C. M. Ángel Díaz, “Robótica educativa y pensamiento computacional,” Trabajo de Fin de Grado, ULL, San Cristóbal de La Laguna, España, 2019. [En línea]. Disponible en: http://riull.ull.es/xmlui/handle/915/15460
[32] “Era de la información,” *Wikipedia*, Sep. 06, 2021. [En línea]. Disponible en: https://es.wikipedia.org/wiki/Era_de_la_informaci%C3%B3n (Accedido: 20-my-2021).
[33] “Jeannette Wing,” *Wikipedia*, Feb. 18, 2021. [En línea]. Disponible en: https://es.wikipedia.org/wiki/Jeannette_Wing (Accedido: 27-jun-2021).
[34] “What is LEGO WeDo?,” *LEGO Engineering*, May 30, 2013. http://www.legoengineering.com/what-is-lego-wedo/ (Accedido: 28-jul-2021).
[35] “What is Unity? Everything you need to know,” *Android Authority*, Mar. 20, 2021. https://www.androidauthority.com/what-is-unity-1131558/ (Accedido: 03-ag-2021).