Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

# Trabajo de Fin de Grado

Grado en Ingeniería Informática

---

# "Simulador de robots para fomentar el pensamiento computacional a través de lenguajes de programación visual"

*"Robot simulator for promoting the computational thinking through visual programming languages"*

Eduardo de la Paz González

---

La Laguna, 30 de junio de 2018

D. **Rafael Arnay del Arco**, con N.I.F. 78.569.591-G Profesor Ayudante Doctor adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor

D. **Eduardo Manuel Segredo González**, con N.I.F. 78.564.242-Z Profesor Asociado 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:

*"Simulador de robots para fomentar el pensamiento computacional a través de lenguajes de programación visual"*

ha sido realizada bajo su dirección por D. **Eduardo de la Paz González**,

con N.I.F. 54.109.482-G.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 30 de junio de 2018

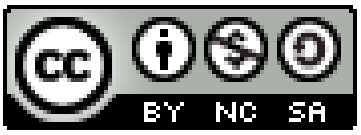
# Agradecimientos

En primer lugar, quiero dar las gracias a mi familia por apoyarme siempre.

En segundo lugar, me gustaría agradecer a Rafa y Eduardo, tutor y cotutor de este TFG, por la ayuda prestada y su colaboración durante estos meses de trabajo.

Por último, agradecer a mis compañeros del grupo M14 el apoyo y la convivencia durante estos cuatro años en el grado.

# Licencia



# Resumen

*La programación y el pensamiento computacional son competencias que la mayoría de las personas no suelen tener en su vida, y las pocas que sí, las adquieren en estudios universitarios en determinados grados que tratan específicamente estas materias. Sin embargo, el conocimiento de estas competencias puede ampliar nuestro pensamiento, mejorando muchas capacidades, como la visión y resolución de problemas.*

*En este proyecto, se busca integrar ambos conceptos en edades más tempranas de nuestra vida, fomentando así la adquisición de estas competencias a cualquier persona, independientemente de la rama en la cual hayan enfocado sus estudios posteriores. La simulación robótica, a partir de códigos generados a través de programación visual, tiene la ventaja de ser de fácil comprensión y, por lo tanto, poder aplicarse en escuelas e institutos. Esto permite promocionar esta aplicación dentro del sector educativo, que es la finalidad principal del proyecto.*

*En este simulador, se simula el comportamiento de un robot en un entorno virtual. Este robot viene determinado por un conjunto de características configuradas previamente en otro módulo. El código que interpreta, generado desde un módulo de programación visual a través de bloques, es la segunda entrada del simulador, que une ambos módulos y los ejecuta en un entorno, simulando su comportamiento con las órdenes recibidas y mostrando la información relativa al robot, como su movimiento o sus sensores, en la interfaz de simulación.*

**Palabras clave:** Simulador, Programación visual, Programación, Pensamiento computacional, Robótica educativa

# Abstract

*Programming and computational thinking are skills that most people do not usually have in their life, and the few who have them, acquire them in university studies, in certain degrees that specifically teach these matters. However, knowledge of these competences can broaden our thinking, improving many skills, such as vision and resolution of problems.*

*This project seeks to integrate both concepts at younger ages of our lives, thus promoting the acquisition of these skills to anyone, regardless of the branch in which they have focused their later studies. Robot simulation, based on generated code through visual programming, has the advantage of being easy to understand, therefore, be applied at schools and high schools. This allows promoting this application within education sector, which is the main purpose of the project.*

*In this simulator, the behaviour of a robot is simulated in a virtual environment. This robot is determined by a set of features, previously configured in another module. The code that it interprets, generated from a visual programming through blocks module, is the second input of the simulator, which join both modules and executes them in an environment, simulating their behaviour as the orders received, and showing the information related to the robot, as its movement and sensors, in the simulation interface.*

**Keywords:** Simulator, Visual programming, Programming, Computational thinking, Educational robotics

# Índice general

# Índice de figuras

# Índice de tablas

# Capítulo 1
# Introducción

En este apartado, se abordarán temas como la motivación del alumno para escoger este trabajo entre los demás presentados, la problemática que se intenta resolver con él, los objetivos que se fijaron en un inicio y que se persiguen en esta línea de trabajo. Por último, se adjunta la planificación inicial que se propuso y sobre la cual se partió para el desarrollo, junto con un pequeño esquema donde se relacionan las entradas y salidas comunes de los tres módulos que componen la aplicación final.

## 1.1 Motivación

La motivación principal que me ha llevado a escoger este TFG es la utilidad que puede llegar a tener si se consigue promocionar dentro del sector educativo. Con este proyecto, se puede llegar a multitud de edades en las cuales todavía no se tiene desarrollado el pensamiento computacional como tal.

Es por ello que proyectos como éste, a través de la programación visual, pueden ayudar a fomentar estas competencias y "abrir la mente" del alumnado a edades más tempranas. Creo que esto, sin duda, puede repercutir positivamente en su manera de enfocar posteriormente cualquier problema de la vida cotidiana.

Además de introducirlos al mundo de la programación que, desde mi punto de vista, debería de ser una competencia que cualquier persona debería de desarrollar en su etapa estudiantil, este conjunto de ideas citadas anteriormente ha sido la principal fuente de motivación para realizar este trabajo y el enfoque que le he dado.

## 1.2 Problemática

Actualmente, nos encontramos en una situación en la cual existen pocas herramientas que combinen la simulación de robots en entornos 3D y la programación visual con fines educativos. En cuanto a la simulación de robots en un entorno 3D, existen distintas alternativas tanto de software libre como comerciales. A continuación, se citan algunos ejemplos:

**Gazebo** [1] es un simulador de entornos 3D que posibilita evaluar el comportamiento de un robot en un mundo virtual. Permite diseñar robots de forma personalizada y crear mundos virtuales usando herramientas sencillas. Otro ejemplo para la simulación de robots es **USARSim** [2], un simulador 3D gráfico de alta fidelidad que usa el potente motor gráfico Unreal Engine usado, por ejemplo, en

muchos juegos de éxito de Epic Games como Fortnite.

Los simuladores comentados anteriormente son ambos de software libre. Sin embargo, existen otras alternativas comerciales. Algunos ejemplos de ellas son:

**Webots** [3]: es un software para simular robots móviles, ampliamente utilizado con fines educativos. Permite construir robots a través de la definición geométrica y dinámica de las partes que lo componen.

**Microsoft Robotics Developer Studio** [4]: es un entorno desarrollado en Windows para el control y simulación de robots. El desarrollador puede acceder fácilmente a los sensores y actuadores del robot.

Todos estos entornos son, en su mayoría, muy costosos como para aplicarse a la educación. Además, suelen estar orientados a entornos profesionales o semiprofesionales, donde prima mucho el nivel de detalle de la robótica y hay muy poco nivel de abstracción, lo que dificultaría el aprendizaje para alumnos que se están inicializando en la materia.

En cuanto a la unión de la robótica con programación visual, no he encontrado en toda mi etapa de documentación, un proyecto que cumpla las características que persigue el aquí presentado. Los programas enfocados a la programación visual suelen ser muy simples y con temáticas menos “técnicas” que la robótica.

Por lo tanto, creo que existe un vacío en el campo donde se intentará desarrollar este proyecto.

## 1.3 Alternativas propuestas

Este proyecto presenta una alternativa al software existente, con un enfoque más lúdico, con el objetivo de captar a un público que se esté iniciando tanto en la robótica como en la programación, como ya se comentó anteriormente. Solamente la inclusión de programación visual sería algo novedoso dentro del sector. Si a esto le sumamos nociones de robótica, como la creación y simulación de robots con diferentes tipos de características, convierten a esta aplicación en una propuesta muy interesante desde el punto de vista educativo.

Por último, cabe añadir que el coste de esta aplicación sería infinitamente inferior a cualquier software de licencia, y, obviamente, muy inferior también a cualquier proyecto que incluya hardware robótico. Este apartado la hace aún más atractiva de cara a su implantación, puesto que casi cualquier centro podría permitirse la inversión. Sin embargo, la mayor restricción que actualmente presenta la aplicación es que está desarrollada como aplicación de escritorio. Esto hace que, por el momento, su uso esté limitado a centros donde se disponga de un equipamiento adecuado que pueda soportar el proyecto. Para futuras mejoras, se ha planteado la migración de la aplicación a la web, permitiendo así el acceso de cualquiera en

cualquier ordenador que tenga un navegador y conexión a internet, independientemente de sus características hardware.

## 1.4 Objetivos perseguidos

En este apartado, se presentará una lista de puntos con los objetivos principales perseguidos desde un inicio y que, en su mayoría, se han podido alcanzar.

En los apartados anteriores se ha desarrollado más en detalle la motivación del proyecto y las soluciones que aporta. En este listado, se citan algunos objetivos ya comentados anteriormente y que han sido de importancia para saber enfocar el desarrollo de este.

- Unir programación visual con robótica.
- Introducir la robótica en la educación previa a unos estudios universitarios.
- Introducir la programación a través de programación visual.
- Fomentar el pensamiento computacional.
- Presentar una alternativa de coste inferior a los actuales simuladores.
- Realizar un simulador que se adapte a un uso educativo.
- Realizar una aplicación accesible para casi cualquier dispositivo, que no requiera de unas características hardware de coste elevado.
- Promover la programación en edades preuniversitarias.

## 1.5 Organización y planificación inicial

En este apartado, se expondrá la planificación inicial para este proyecto. Esta organización fue pactada entre ambas partes (tutores y alumno) y se recogió por escrito en el anteproyecto de este trabajo de fin de grado.

Las diferentes tareas y funcionalidades por implementar se recogieron en una lista, además de un diagrama de Gantt con la planificación por meses del desarrollo del trabajo. Estas funcionalidades fueron la primera hoja de ruta a seguir para el desarrollo de la aplicación y se cumplieron casi en su totalidad. A continuación, se adjuntan tanto las diferentes actividades propuestas como el diagrama de Gantt original. No obstante, en el apartado 4 de esta memoria se entrará más detalladamente en las funcionalidades de la aplicación. Por su parte, los obstáculos encontrados durante el desarrollo y que hicieron cambiar algunas de las actividades recogidas en el planteamiento inicial se detallarán en el apartado 3.6. Así, las funcionalidades definidas en un principio fueron las siguientes:

- **Diseño de menús**

- **Diseño de objetos**
  - Entorno
  - Obstáculos
  - Elementos con los que interactuar
- **Creación de distintos niveles**

- **Incorporaciones de otros módulos**
  - Incorporación del robot configurado
  - Incorporación de las instrucciones del robot
- **Simulación del robot**

El diagrama temporal seguido fue el siguiente:

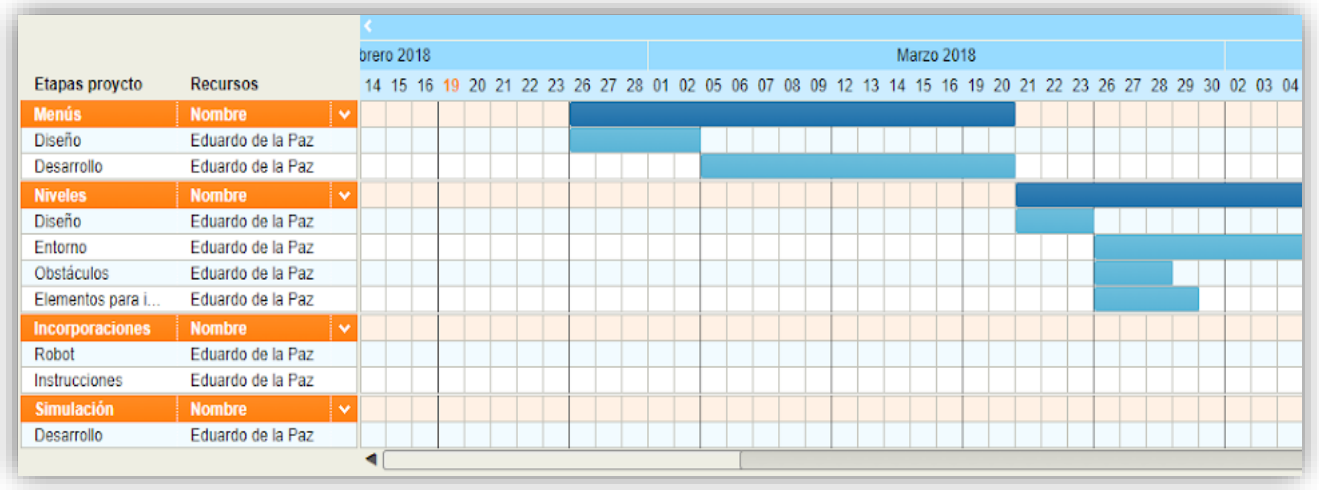



**Figura 1: Diagrama de Gantt, planificación de marzo**

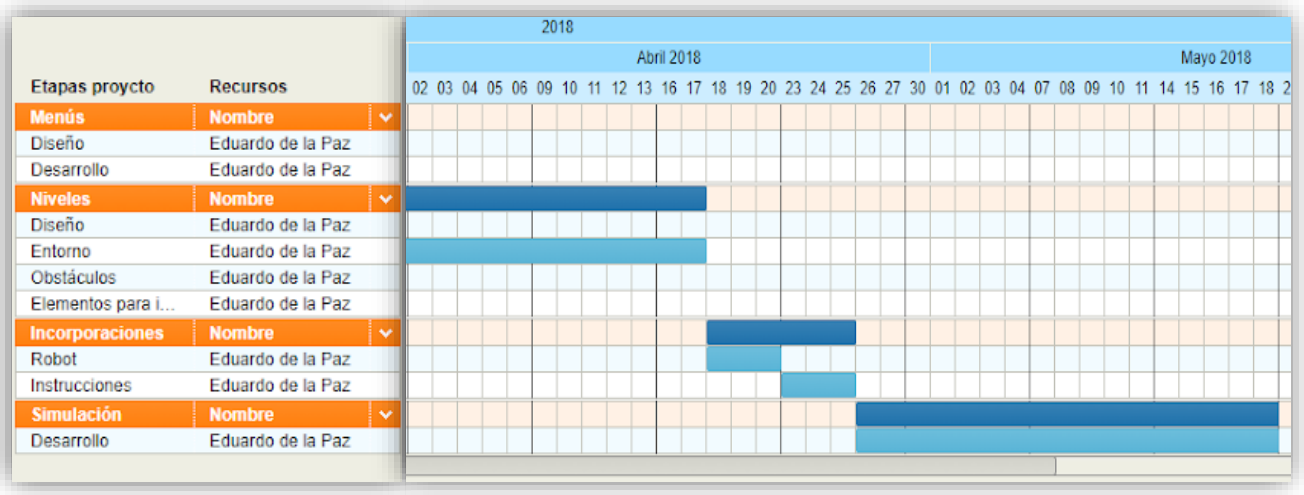



**Figura 2: Diagrama de Gantt, planificación de abril y mayo**

## 1.6 Esquema del flujo entre los módulos

A continuación, se adjunta un esquema en donde se detalla el funcionamiento del este proyecto, junto con los otros dos restantes, los cuales conforman la aplicación.

La lógica del programa comienza con la configuración del robot deseado en el módulo 1. En esta parte, el usuario personaliza el robot con diversas características. Una vez el robot es configurado, se exporta hacia los módulos 2 y 3.

En cuanto al módulo 2, recibe a través de un fichero de configuración XML, todos los ajustes que han sido añadidos al robot, para así habilitar los bloques del lenguaje de programación visual disponibles, dependiendo de las características que le han sido añadidas. Cuando procesa este fichero, muestra al usuario la posibilidad de crear el comportamiento que posteriormente simulará el robot, a través de un lenguaje de programación visual específicamente diseñado para tal fin. Finalizado el proceso de programación por bloques, se genera el código correspondiente a esos bloques en JavaScript y es exportado al último módulo, encargado de la simulación.

En este último módulo, se recogen tanto las instrucciones que debe seguir el robot como el objeto perteneciente al robot, y se simula su comportamiento dentro de un entorno. En los siguientes capítulos se desarrollará en más profundidad todos los detalles relativos a esta última parte. En la siguiente página, se adjunta el diagrama representativo del funcionamiento.

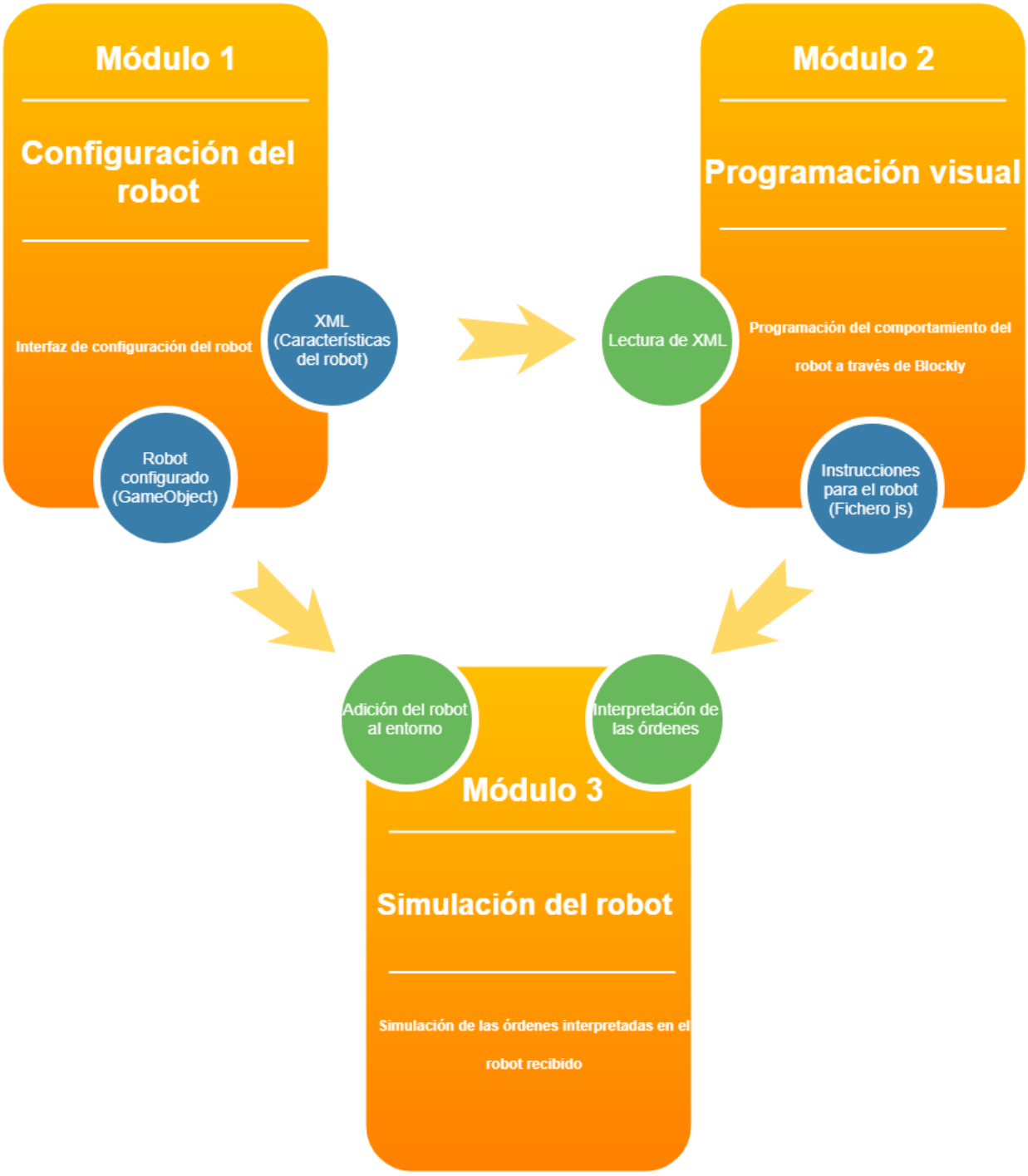



**Figura 3: Diagrama de flujo entre módulos**

# Capítulo 2
# Antecedentes

Una vez introducido el proyecto, en este capítulo, se expondrán las materias relacionadas con éste, así como algunos proyectos similares al desarrollado.

## 2.1 Materias y temas relacionados con el proyecto

### 2.1.1 Robótica Educativa

La robótica educativa es un sistema de enseñanza, (y aprendizaje por parte del alumno), en el cual se potencia las habilidades y competencias en los alumnos a través de actividades relativas a la robótica que se presentan al usuario en forma de retos. Estos retos se presentan principalmente, de forma mental y, posteriormente, en forma física. Cuando hablamos de robots físicos, estos artefactos pueden ser construidos con multitud de piezas y materiales formando dispositivos desde lo más simple hasta muy complejos, y son controlados por un sistema computacional interno. La ejecución de tareas en estos dispositivos de forma experimental es lo que más tarde se definirá como simulación. Este trabajo está enfocado principalmente a la robótica educativa. De hecho, está ligado a la educación en dos campos diferentes, ya que los alumnos pueden familiarizarse con la robótica a la vez que comienzan a descubrir programación visual para resolver los problemas.

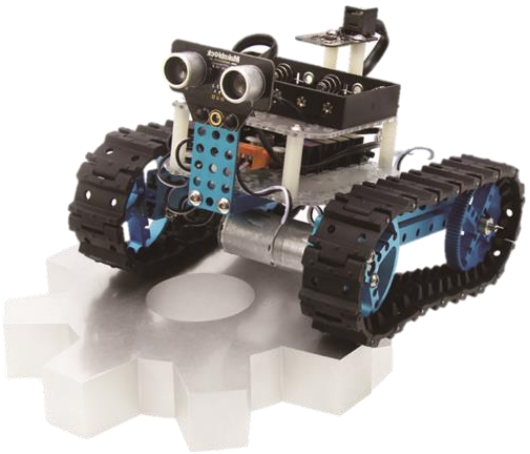


**Figura 4: Robot móvil [20]**

### 2.1.2 Simulación

Para encontrar la definición más formal de simulación, citaré la de Robert Shannon, que en su libro *Systems simulation: the art and science* [7] [8]*,* define: “Simulación es una técnica numérica para conducir experimentos en una computadora digital. Estos experimentos comprenden ciertos tipos de relaciones matemáticas y lógicas, las cuales son necesarias para describir el comportamiento y la estructura de sistemas complejos del mundo real a través de largos períodos". A grandes rasgos, define lo que se trabaja principalmente en este simulador. Se experimenta con robots de forma virtual y se traslada a un entorno de características reales. Además, a esto se le añade, que el robot responde a unas instrucciones recogidas de un código generado por bloques, lo que aumenta el grado de dificultad en la simulación y hace que sea algo más complejo de implementar físicamente. En este proyecto no solo se simula el movimiento del robot en un entorno. Se incluye la simulación de los sensores del robot, así como su interacción con los objetos de la escena, mostrando la información que devuelven en todo momento. Como se verá más adelante, la interfaz del programa devuelve en cada instante los parámetros de posición y orientación del robot, como la de los sensores ya mencionados.

### 2.1.3 Programación Visual

La programación visual [15] contiene elementos necesarios para poder desarrollar pequeñas aplicaciones a través de un entorno de bloques visuales muy amigable para el usuario y enfocado a programadores que se estén iniciando en la materia. Estos lenguajes usan un sistema de arrastrar y soltar bloques en lugar de tener que escribir todo el código en texto plano como normalmente se programa. Estos lenguajes proporcionan muchas ventajas [16] con respecto a los lenguajes tradicionales para usuarios no expertos, como pueden ser niños en edades tempranas.

Una de las principales ventajas que tienen estos lenguajes es que son de fácil aprendizaje. Esto es debido a que los usuarios no deben recordar una larga lista de comandos o sintaxis específica. Simplemente, cada vez que abren el entorno de programación visual, tienen disponible la lista con todos los bloques disponibles según el contexto.

La siguiente gran ventaja que presenta la programación visual es la ausencia de errores sintácticos. Cuando hablamos de niños iniciándose en la programación, lo que prima es que aprendan a resolver programas a través de ella y no, que pierdan mucho tiempo depurando algún paréntesis no cerrado, un punto y coma olvidado, etc. Es por ello que este tipo de lenguajes presentan bloques, los cuales solo pueden ser unidos con otros bloques si son sintácticamente correctos. Es decir, los bloques se unen como un puzle, donde solo encajan las piezas que producirían un código sin errores de sintaxis.

Por último, la programación visual presenta al usuario una interfaz amigable a través de iconos y colores, que le produce al programador, (normalmente enfocado a

niños) más atención e interacción que un archivo con texto plano. En la siguiente imagen se presenta un pequeño ejemplo de la comparación de la programación por bloques y su correspondencia en lenguaje de texto, en este caso, JavaScript.

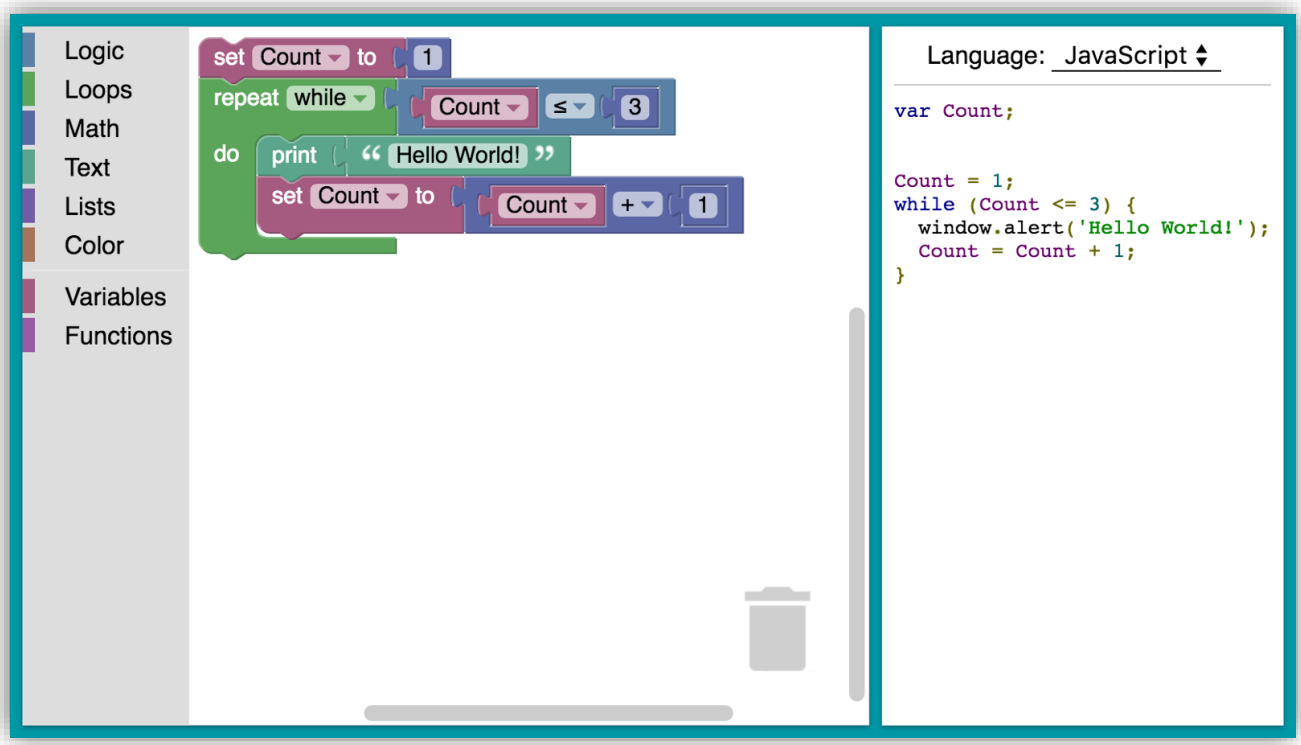



**Figura 5: Comparación entre programación visual y su salida en texto [17]**

# 2.2 Proyectos similares

### 2.2.1 Robot Mesh Studio, Virtual VEX IQ Robot Online

Robot Mesh es una empresa estadounidense dedicada al soporte de material robótico para actividades educativas en Estados Unidos y Canadá [25]. Además de material robótico físico, presenta software con el cual programar dichos robots o simularlos previamente, para analizar su comportamiento. Este programa, llamado Robot Mesh Studio, presenta múltiples posibilidades de personalización, y, a través de éste, se han creado algunos programas para uso educativo. En su proyecto realizado el año pasado: 2017 Hour of Code Activities with Robot Mesh Studio, se creó una aplicación de escritorio, disponible también online, para el aprendizaje de programación de robots [26].

La aplicación, permite en la ventana izquierda programar un VEX IQ Robot virtualmente, a través de programación visual, y simular su comportamiento a su derecha en la ventana “Mimic”.

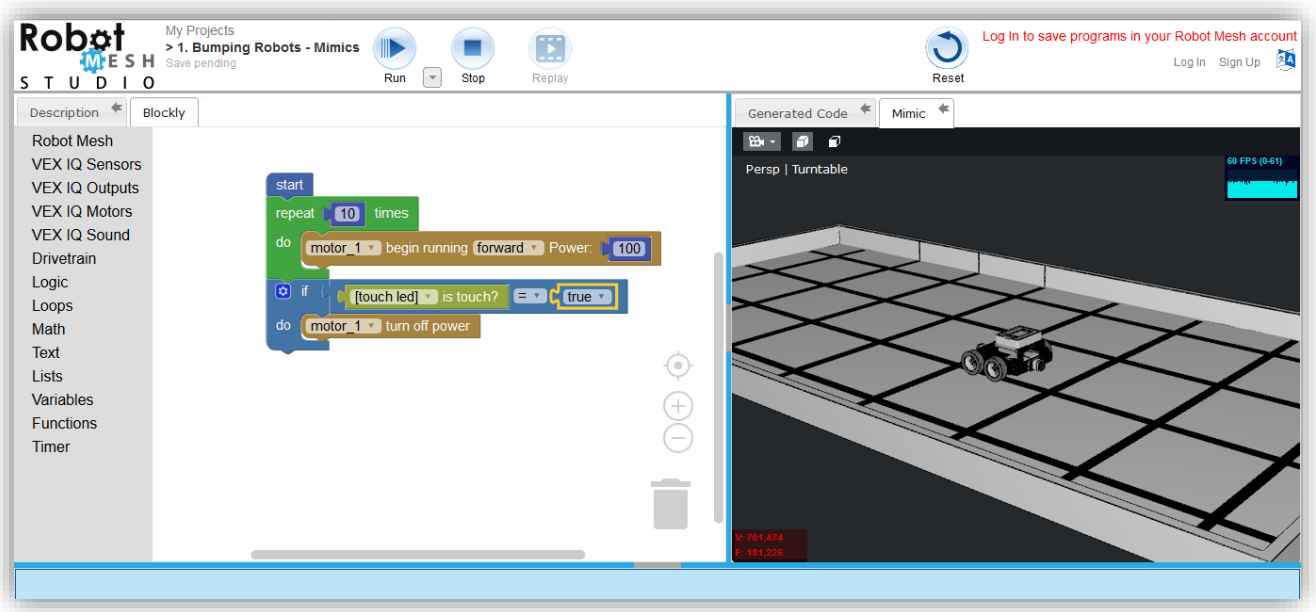



**Figura 6: Interfaz de Robot Mesh Studio**

La licencia educativa de este software es de 49.99 dólares estadounidenses al año, con posibilidad de agregar hasta 50 alumnos. Sin embargo, las pequeñas demostraciones presentes en su página web son de uso gratuito e ilimitado.

Este proyecto está relacionado directamente con esta aplicación ya que su funcionamiento es muy parecido y el objetivo final es bastante similar: introducir a la programación a través de la programación visual y la robótica. Además, este proyecto está integrado en la web, facilitando el acceso desde cualquier lugar y con cualquier ordenador. En el apartado 5 de esta memoria, se especifica que ésta es una de las líneas futuras de la aplicación. Exportarla a la web y no depender de un hardware concreto que soporte la aplicación de escritorio, por lo tanto, Robot Mesh Studio, es un muy buen ejemplo de implementación en el que fijarse y tomar ideas.

# Capítulo 3
# Metodología

En el capítulo anterior se ha presentado toda la información relativa al proyecto, necesaria para entender, a grandes rasgos, los temas relacionados con este trabajo. En este capítulo, se expondrán las diferentes herramientas utilizadas, con una breve explicación a modo de introducción al lector. Además, se describirán los recursos utilizados y consultados para la realización de la memoria. Por último, se describen las diferentes fases de desarrollo que se han seguido, junto con los problemas encontrados y las soluciones tomadas.

## 3.1 Metodología de trabajo

La metodología de trabajo seguida en este proyecto fue definida desde la primera reunión y llevada a cabo durante el resto del tiempo de trabajo. En un principio, se definieron los hitos o metas a desarrollar por el alumno. Definidos los objetivos finales, el alumno trabajaba individualmente en ellos. Cada dos semanas, se llevaba a cabo una reunión de control, para presentar lo trabajado hasta ese punto, los problemas encontrados, y las modificaciones tomadas. Aquí los tutores valoraban y exponían su punto de vista para continuar con el desarrollo. Además de las reuniones, también se contactaba por correo electrónico, para dudas puntuales, dada la gran disposición de los tutores a ayudar al alumno.

El sistema de reuniones cada 2 semanas, hizo que se mantuviese el contacto en todo momento, y que, a pesar, de otras materias presentes para el alumno durante el cuatrimestre, no se pausase el desarrollo de este trabajo durante largas etapas, logrando así, finalizar el trabajo progresivamente durante este tiempo.

## 3.2 Herramientas para el desarrollo

### 3.2.1 Unity

Unity es un motor de videojuegos multiplataforma creado por Unity Technologies. Se trata de una potente herramienta de desarrollo para los principales sistemas operativos, y cuyas aplicaciones son exportables a entornos de escritorio, consolas, dispositivos móviles, entornos web etc. [10] Unity ha sido la principal herramienta de desarrollo, donde se realiza toda la simulación, y en la que se trabaja con todos los scripts que se mencionarán en los próximos capítulos. Una de las grandes ventajas de Unity como motor, es la gran biblioteca de “Assets” que dispone en su **Asset Store** [13]. Un asset es una representación de cualquier ítem que puede

ser utilizado en un juego o proyecto. Un asset podría venir de un archivo creado fuera de Unity, tal como un modelo 3D, un archivo de audio, una imagen, o cualquiera de los otros tipos de archivos que Unity soporta. Sin embargo, también existe la posibilidad de descargarlos de la tienda mencionada anteriormente. Dentro de la Asset Store, existen multitud de archivos de gran utilidad, se trata de una colección de más de 4.400 paquetes de Assets en una amplia gama de categorías, incluyendo modelos 3D, texturas y materiales, sistemas de partículas, música y efectos de sonido, tutoriales y proyectos, paquetes de scripts, extensiones para el editor y servicios en línea… [14] He dispuesto de algunos de ellos para la realización del entorno de simulación. En apartados posteriores, se citará cada uno de ellos y su función dentro del proyecto. He elegido Unity, aparte de las razones ya mencionadas, porque es el único motor de videojuegos con el que he tenido experiencia anterior (concretamente en una asignatura del grado), y esto me ahorraba el tiempo de aprendizaje y de adaptación a otro motor diferente. Alternativas a Unity para el desarrollo de esta aplicación pueden ser Blender o Unreal Engine. No obstante, no las consideré en ningún momento puesto que mis conocimientos en Unity bastaban para los requerimientos del proyecto y esto me facilitó la iniciación del simulador.

### 3.2.2 GitHub

GitHub es una plataforma de desarrollo colaborativo para alojar proyectos utilizando el sistema de control de versiones Git [27]. Su función principal es la de almacenar el código de un determinado programa, así como sus diferentes versiones, documentación, despliegue, …

He trabajo con GitHub debido a varias características que hacen que sea una plataforma de mucha ayuda para poder operar de forma cooperativa con mis tutores. Una de ellas es GitHub Classroom, una herramienta que automatiza la creación de repositorios, el acceso y el control a los archivos de forma muy intuitiva para que los profesores puedan distribuir elementos y tareas a los alumnos. Además, dentro de cada proyecto alojado en GitHub, un profesor con permiso para acceder a los archivos puede hacer uso de los “issues”. Un issue no es más que una nota que alguien externo al proyecto puede dejar al autor del repositorio para alertarle/informarle sobre bugs, fallos o posibilidades de mejora principalmente. Estas issues son gestionadas por los propietarios del repositorio, cerrándolas cuando se crea conveniente.

### 3.2.3 Microsoft Office Word 2016

Microsoft Office Word es un potente editor y procesador de textos desarrollado por la empresa Microsoft. He usado Word debido a que es el editor con el que más familiarizado estoy desde hace muchos años y que mejor conozco sus funcionalidades. Esto me ha permitido poder desarrollar la memoria en un tiempo bastante más corto que si hubiera usado otro editor. Sin embargo, existen otras alternativas que también tuve en cuenta a la hora de elegir un programa para la

redacción de la memoria. LibreOffice, una alternativa de software libre con funcionalidades muy parecidas a Word; u Overleaf, una plataforma online colaborativa de redacción, edición y publicación de textos en formato LaTeX, fueron las alternativas en un primer momento antes de decantarme por Word. La primera de ellas no fue elegida debido a que considero que pudiendo tener acceso a Word, las posibilidades que te ofrece son superiores a LibreOffice, además de una interfaz mucho más amigable en mi opinión. Overleaf por su parte, fue descartada debido a que no tengo un conocimiento amplio del sistema de composición de textos LaTeX, y me llevaría bastante tiempo aprender a usarla antes de poder empezar a redactar la memoria. No obstante, LaTeX es una de las mejores opciones para la creación de artículos científicos, por lo que consideraré pasarlo a dicho formato en un futuro, así como aprender la sintaxis para mi vida profesional.

## 3.3 Recursos utilizados

### 3.3.1 Assets

Para la generación del entorno de simulación, se ha hecho uso de variedad de assets disponibles en la Asset Store de Unity. A continuación, se listan todos paquetes utilizados, así como su función dentro del proyecto. Cabe destacar que todos los paquetes son de coste gratuito y se encuentran en la tienda disponibles para la descarga en cualquier momento. Los assets incluidos son:

- **AxeyWorks, Low Poly Free Pack:** Todos los elementos de decoración externos, como vallas, árboles, flores, césped, fuente, etc. [21].
- **BigFurniturePack:** Todos los modelos de decoración interna a la casa, como el mobiliario del salón y del baño [22].
- **CasaModel:** Gameobject de la casa, generado desde SketchUp e importado como asset a Unity.
- **Gradientes:** Directorio con los gradientes utilizados en todos los textos pertenecientes a los botones del menú inicial.
- **Ground textures pack:** Texturas variadas de tierra, césped, nieve, etc.
- **JurassicJS:** Intérprete para Unity de código JavaScript. El funcionamiento de este asset, fundamental para el desarrollo del proyecto, se amplía en el siguiente apartado de esta memoria [18].
- **TextMesh Pro:** Asset para la creación de textos con una amplia variedad de formatos, colores, fuentes, tamaños… Amplía las posibilidades para personalizar textos que trae Unity por defecto [23].
- **Wispy Sky:** Paquete con el skybox añadido a la escena. Un skybox genera el cielo y el horizonte de una escena de Unity, simulando cómo sería el mundo alrededor de los objetos representados en el programa. Este efecto aporta

un alto grado de realidad al cielo del simulador [24].

### 3.3.2 Jurassic

Jurassic es un paquete desarrollado en ECMAScript, destinado a implementar JavaScript en .NET. Su uso no está pensado para usuarios finales, sino para ser integrado en programas .NET por los desarrolladores para compilar y ejecutar código JavaScript. Su asset en Unity, tiene un funcionamiento exactamente igual, salvo que, en lugar de .NET, el lenguaje utilizado es C#. Por lo tanto, compila y ejecuta código JavaScript para poder integrarlo dentro del C# propio de los scripts de Unity.

El descubrimiento de este asset fue un gran avance dentro del desarrollo de la simulación de la aplicación, ya que permite interpretar el código devuelto por el módulo de programación visual y darle el comportamiento que se desee. Además, la estructura de este intérprete permite definir nuevas funciones dentro del código, lo que permite poder personalizar el comportamiento del robot para cada función que reciba JavaScript.

A continuación, se procede a explicar el funcionamiento básico de este asset en nuestro proyecto, así como algunas capturas de código.

#### *Funcionamiento de Jurassic*

Para comenzar con este complemento, lo primero que se debe hacer es crear una instancia de Jurassic en nuestro proyecto de la siguiente manera:

```
engine = new ScriptEngine();
```

Una vez iniciado Jurassic en nuestro objeto “engine”, la ejecución de un código en JavaScript se puede realizar leyendo desde fichero o desde una variable que almacene el código.

```
//Ejecución desde string

engine.Execute(codeString);
```

```
//Ejecución desde fichero

engine.ExecuteFile(filePath);
```

En este caso, además de ejecutar el motor de este intérprete, se ha tenido que realizar esta tarea en un hilo nuevo. Esto es, se ha definido un hilo nuevo en el cual se va a procesar la interpretación del código, independiente del hilo principal de Unity, el cual se encarga de actualizar los frames y actualizar la posición y orientación del robot. Así, la iniciación del intérprete en este proyecto queda de la siguiente manera:

```
if (GUILayout.Button ("Execute")) {
            interprete = new Thread (Ejecutar);
            interprete.Start ();
}
```

De la misma manera, una vez se finaliza la simulación abortamos el hilo creado. En el siguiente esquema, se muestra un resumen del funcionamiento de los hilos:

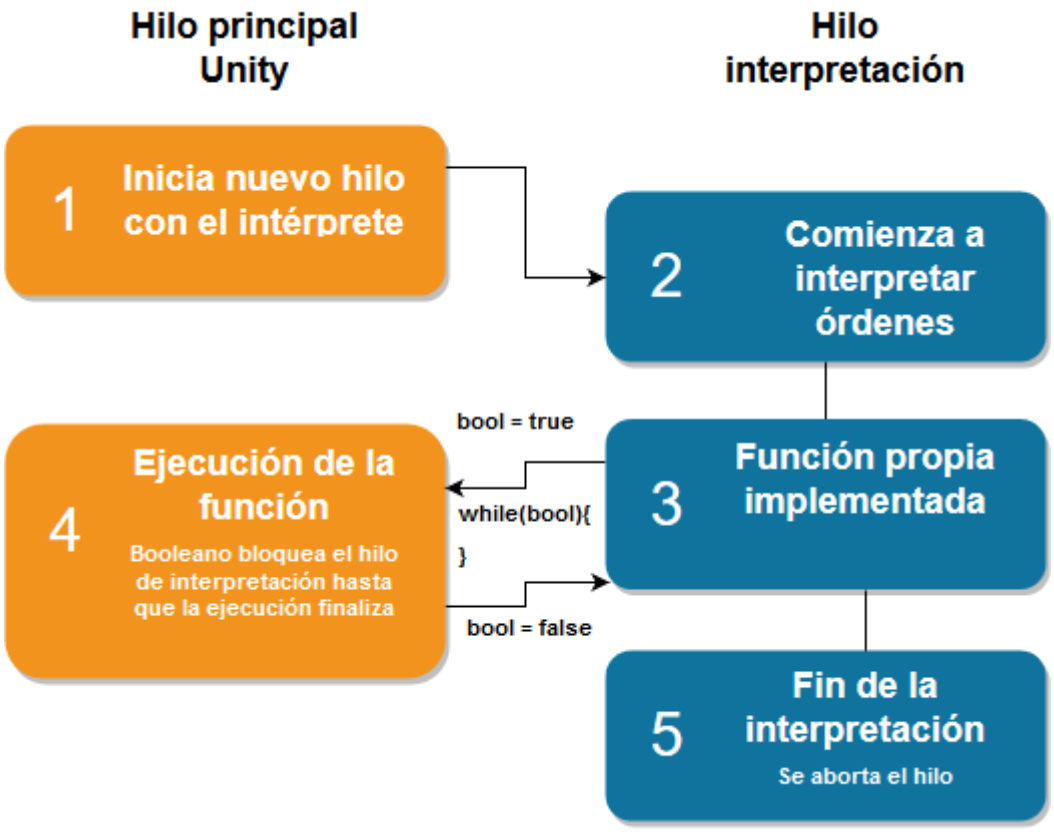



**Figura 7: Diagrama de funcionamiento de los hilos**

El funcionamiento de este intérprete es así de sencillo. Sin embargo, el verdadero potencial es el de poder declarar tus propias funciones, clases, y variables e implementar su funcionamiento. Las siguientes líneas de código, se corresponden a la creación de las funciones que recibiremos desde el módulo de bloques y que debemos definir su comportamiento.

```
engine.SetGlobalFunction ("Girar", new System.Action<int> (jsGirar));
engine.SetGlobalFunction ("Avanzar", new System.Action<double, int> (jsAvanzar));
engine.SetGlobalFunction ("SensorLaser", new System.Func<float> (jsSensorLaser));
engine.SetGlobalFunction ("SensorIR", new System.Func<bool> (jsSensorIR));
engine.SetGlobalFunction ("SensorUS", new System.Func<float> (jsSensorUS));
engine.SetGlobalFunction ("TouchSensor", new System.Func<bool> (jsTouchSensor));
engine.SetGlobalFunction ("Laser2D", new System.Func<float> (jsLaser2D));
```

Como se puede observar, le podemos añadir a nuestra variable engine, las funciones que nosotros deseemos, (jsGirar y jsAvanzar y los 5 sensores disponibles en este caso), y que definiremos su comportamiento previamente. La principal restricción de estas declaraciones son los tipos soportados. Jurassic actualmente acepta solamente los tipos de datos más básicos para sus funciones, debido a ello, si queremos operar con algún derivado de estos tipos, tendremos que realizar la conversión a ellos dentro de las propias funciones. En la siguiente tabla se muestran los tipos de datos soportados:

**Tabla 1: Tipos de datos soportados en Jurassic**

| C# type name | .NET type name | JavaScript type name |
|---|---|---|
| bool | System.Boolean | boolean |
| int | System.Int32 | number |
| double | System.Double | number |
| string | System.String | string |
| Jurassic.Null | Jurassic.Null | null |
| Jurassic.Undefined | Jurassic.Undefined | undefined |
| Jurassic.Library.ObjectInstance (or a derived type) | Jurassic.Library.ObjectInstance (or a derived type) | object |

Una vez definidas las funciones para el intérprete, podemos definir su comportamiento en Unity. En el siguiente ejemplo, se muestra la función jsAvanzar, la cual interviene en el movimiento de avance del robot.

```
public void jsAvanzar(double v, int s) {

    segundos = s;
    velocidad = (float)v;
    calculartiempo = true;
    _avanzando = true;
    while (_avanzando) {

    }

}
```

Como podemos ver, podemos definir la función que espera el intérprete en JavaScript y desarrollar su comportamiento dentro del programa. La metodología seguida para todas las funciones ha sido la de activar booleanos internos, para cada

una de las diferentes funciones que recibe el simulador. Esto permite poder tener el control temporal de todas y cada una de las órdenes que recibe el robot, y así poder ejecutarlas secuencialmente, respetando la finalización de una para el inicio de la siguiente. El hilo correspondiente a la interpretación queda esperando mientras el booleano interno realiza la operación correspondiente. Una vez finaliza el movimiento, asigna a false el valor del booleano y devuelve el control al hilo que interpreta que seguiría leyendo las siguientes órdenes. El código completo de los principales scripts se adjuntará en el apartado 8 de este documento, para consultar el resto de los códigos se adjunta a pie de página el enlace al repositorio del proyecto[1].

En el siguiente esquema, se puede apreciar un ejemplo de lo explicado anteriormente con la función avanzar:

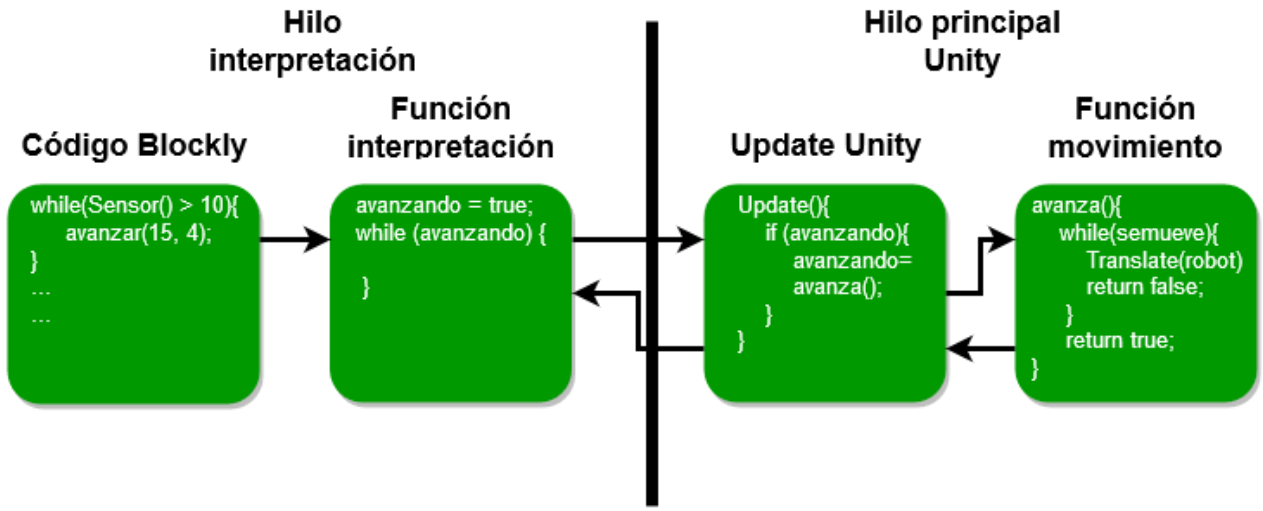



**Figura 8: Diagrama de hilos**

[1] Repositorio Github: https://github.com/ULL-ESIT-INF-TFG-1718/tfg-2017-2018-roboedu-compthink-alu0100893267

Con esto, se completa el funcionamiento del complemento Jurassic relativo a la interpretación del código JavaScript. Los demás elementos propios del lenguaje, como pueden ser los condicionales, los bucles o las declaraciones, se procesan directamente en el parseador de Jurassic, interpretándose directamente y ejecutándose en Unity. En la bibliografía de este documento, se asigna un enlace al repositorio de Jurassic [19], donde se puede encontrar el código fuente del parseador.

## 3.4 Fases de desarrollo

En este apartado, se resumen las diferentes fases de desarrollo por las que transcurrió este proyecto, desde su definición hasta su finalización.

### 3.4.1 Unión de los diferentes módulos

En la primera fase del proyecto, tuve que reunirme varias veces con mis tutores para la definición del alcance de la aplicación. Los principales puntos a tener en cuenta eran cómo unir el simulador a los otros módulos, además de fijar un estándar para dicha unión y empezar a trabajar sobre él. Se decidió unir los módulos de la siguiente manera:

- Unión con el módulo de configuración del robot: Se desarrollaría en Unity al igual que el simulador, por lo tanto, la forma más sencilla es exportar como asset el robot una vez es configurado e importarlo en el simulador. Esta unión se puede realizar en tiempo de ejecución gracias a la unificación de criterios para nombrar los componentes del robot y a la determinación de un directorio en concreto para depositar el robot y que el simulador lo encuentre.
- Unión con el módulo de programación visual: Este módulo, se desarrollaría en Blockly. Blockly es una librería desarrollada en JavaScript para la creación de lenguajes de programación visual a través de bloques. Establecer un nexo común entre el simulador y Blockly fue más complejo que el anterior módulo. Para enlazarlo, decidimos exportar el código generado por el usuario al seleccionar diferentes bloques a un fichero que se depositaría en un directorio fijado previamente. Una vez el fichero se encuentra en la carpeta, el simulador puede cargar, en tiempo de ejecución, las instrucciones generadas por la programación visual. Sin embargo, estas instrucciones deben ser interpretadas antes de aplicarse directamente al robot. Además, se exportan en lenguaje JavaScript, mientras que Unity trabaja principalmente con C#. Es por ello, que se insertó un intérprete de JavaScript para Unity (Jurassic), el cual lee el fichero correspondiente, interpreta las órdenes designadas por el código y las aplica al robot. El funcionamiento del intérprete se explicará con más detalle en el apartado de desarrollo de la aplicación.

### 3.4.2 Definición de las funcionalidades de la aplicación

En segundo lugar, se definieron los puntos a evaluar en la aplicación y que quedaron reflejados en el anteproyecto del trabajo. Así, se acordó que el proyecto debía contar con los siguientes apartados:

- Crear un módulo que permita cargar el robot diseñado.
- Crear un módulo que permita interpretar las instrucciones del apartado de programación visual y las simule en el robot.
- Crear una interfaz atractiva y funcional.
- Crear un módulo para la definición de nuevos entornos.

Estos apartados fueron la guía para el desarrollo de la aplicación y se ampliarán en el siguiente apartado de esta memoria.

### 3.4.3 Desarrollo de la aplicación

El siguiente apartado a desarrollar, fue la aplicación principal. Aquí, se pueden diferenciar tres líneas diferentes de trabajo. Estas tres líneas serían: el diseño y desarrollo del entorno, de la interfaz, el de la simulación (donde se incluyen todas las tareas relativas al movimiento del robot), y la conexión con los otros módulos, necesaria para cerrar el funcionamiento completo de la aplicación. Todo este proceso se explica ampliamente en el apartado 3.5 de la presente memoria, donde se detalla el desarrollo punto por punto de los elementos citados anteriormente.

### 3.4.4 Desarrollo de la memoria

El desarrollo de la memoria fue el último apartado del trabajo en realizarse. Todo el trabajo desarrollado durante los meses anteriores, se reflejan en esta memoria final del TFG. Este proceso, ha sido también supervisado en todo momento por los tutores académicos. La metodología para su final realización ha sido a través de borradores enviados a los tutores, hasta pulir la versión final para su entrega.

## 3.5 Desarrollo de la aplicación

### 3.5.1 Menú principal y entorno

En primer lugar, se comenzó realizando la interfaz básica. Aquí se realizó un menú de usuario preparado para cargar el robot creado y las instrucciones programadas. En el apartado de modos de uso, se representa cómo interactuar con dicho menú. Una vez realizado, se inició el desarrollo del simulador, comenzando por el entorno. Fue en este paso donde se agregaron todos los elementos decorativos e interactivos a la escena. He aquí un listado con los diferentes objetos:

- **Casa principal**

  Diseño en Google SketchUp del modelo y exportación a Unity.

  Formado por muros, suelo y techo (no visible para mejorar la iluminación en la simulación).

  - Baño
    - Decoración básica de baño: ducha, lavabo, váter…
  - Salón
    - Decoración básica de salón: sofás, estanterías, cuadros, tv…
- **Jardín**
  - Elementos decorativos
    - Vallas
    - Árboles
    - Césped
    - Rocas
    - Plantas
    - Arco
  - Elementos interactivos
    - Monedas

En la siguiente imagen, se muestra una vista cenital del resultado final del diseño del interior de la casa. En los anexos se adjuntarán más imágenes desde el punto de vista del robot en la simulación.

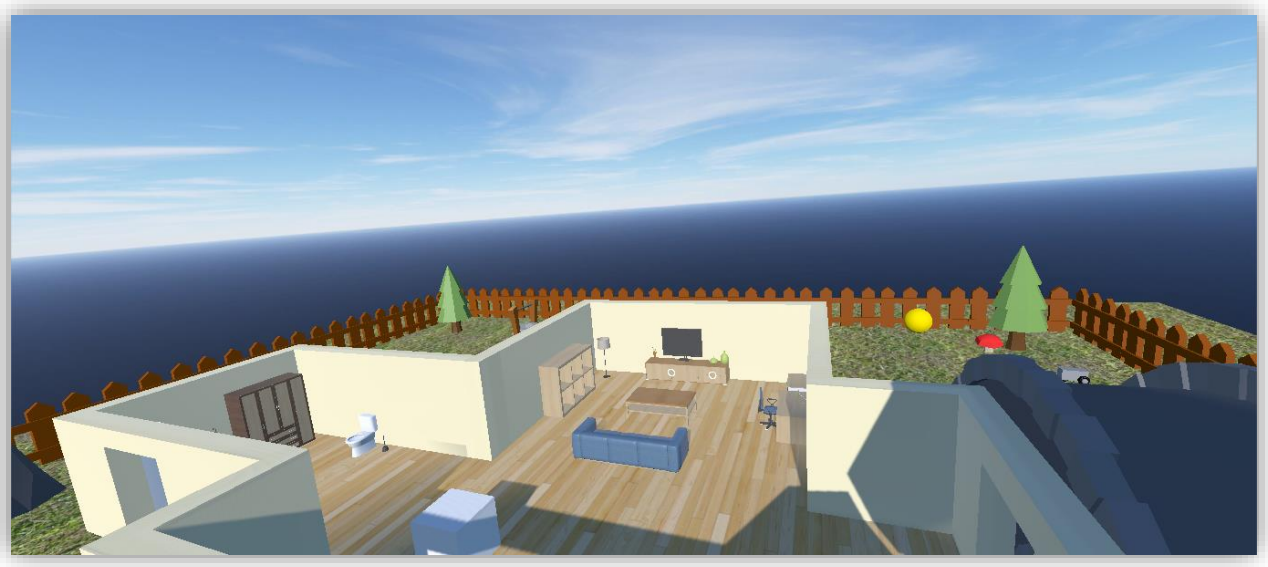


**Figura 9: Interior de la casa**

Además de los objetos, se añadió iluminación a la escena. Este proceso que se realiza con relativa facilidad en Unity, y dota a la escena de simulación de un realismo mucho mayor. La iluminación añadida a la escena fue creada con los siguientes parámetros:

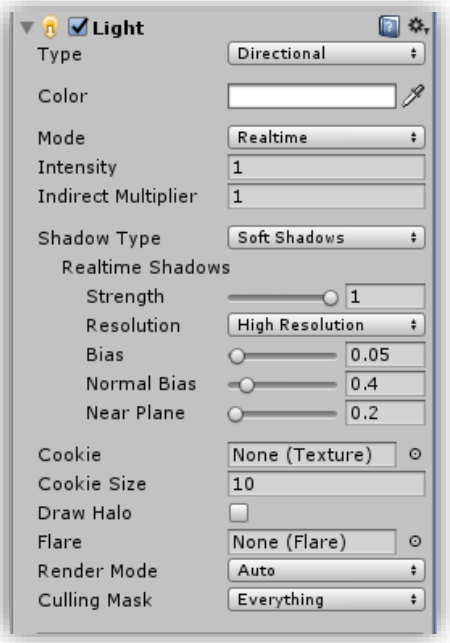



**Figura 10: Configuración de la iluminación**

Para finalizar el diseño del exterior, se añadieron tres “Reflection probe”, una en cada habitación de la casa. Este elemento de Unity es como una cámara esférica que guarda la información de los objetos de su alrededor y genera los reflejos de la luz en el punto en el que se sitúan, por lo que el realismo de la escena aumentó

significativamente. Después de este paso, se dio por concluido el diseño del exterior, y el trabajo se centró más en la simulación del robot en sí. El modelo final del entorno de simulación sería el siguiente:

**Figura 11: Entorno de simulación**

### 3.5.2 Simulación

El siguiente paso en el desarrollo de la aplicación fue el funcionamiento de la simulación. Para ello tuve que documentarme sobre las posibles entradas que recibiría la simulación de los otros módulos. En el apartado 3.6 se detallarán ampliamente los problemas encontrados a la hora de unir los diferentes módulos. Para la simulación del robot se debe contar con el robot físicamente, añadido como asset de Unity, y con las líneas del código que queremos que interprete y simule.

Para incluir el robot, se optó por una escena de Unity intermedia, la cual exporta directamente el robot creado en el módulo de configuración del robot al simulador. Este paso guarda el robot con toda su configuración y lo exporta directamente al entorno. Debido a que ambos módulos fueron programados en Unity, el proceso de unificación e importación de elementos fue bastante sencillo.

Para la interpretación del código obtenido del módulo de programación visual, el trabajo fue más complejo. En primer lugar, se inició una búsqueda para tratar de enlazar Blockly con Unity. El resultado de la búsqueda de proyectos similares concluyó en que la incorporación en una misma aplicación de escritorio de ambos entornos era inviable. Se optó entonces, por realizar un intérprete de JavaScript, para que el robot simulase las órdenes programadas. Este intérprete llamado Jurassic, se encuentra en la Asset Store de Unity, y su funcionamiento está explicado detalladamente en el apartado 3.3.2 de este documento. Llegado a este punto, me documenté sobre el funcionamiento de este intérprete para poder personalizarlo en mi proyecto. Su sintaxis es medianamente sencilla y de gran utilidad para este trabajo, ya que enlaza a la perfección los dos módulos menos similares. El

funcionamiento básico de este complemento de Unity se resume el apartado citado anteriormente. Para analizar más en profundidad sus posibilidades, existe una wiki en su propia página de GitHub enlace [19]. El siguiente paso en la simulación fue desarrollar todas las funciones posibles que recibiría el robot del módulo de bloques, para simular su comportamiento en el entorno. Para ello, se tuvieron que definir previamente todas las posibles funciones que recibiría el robot. Las funciones establecidas fueron las siguientes:

- Girar(x grados)

  Ordena al robot a girar x grados. Siendo los grados positivos si se quiere girar a la derecha y negativos si se quiere girar a la izquierda.

- Avanzar(x velocidad, t segundos)

  Comienza a avanzar el robot a x velocidad durante los t segundos establecidos.

- Laser()

  Devuelve una medida con la distancia a la cual se sitúa el objeto con el cual está colisionando el sensor láser (solo en caso de que haya un sensor laser incluido).

- Infrarrojo()

  Devuelve un booleano con la información que está retornando el sensor infrarrojo del robot, devolviendo verdadero en caso de que haya detectado un objeto y falso en caso contrario (solo en caso de que haya un sensor infrarrojo incluido).

- Ultrasonido()

  Devuelve una medida con la distancia a la cual se sitúa el objeto con el cual está colisionando el sensor de ultrasonido (solo en caso de que haya un sensor ultrasonido incluido).

- Contacto()

  Devuelve un booleano con la información que está retornando el sensor de contacto del robot, devolviendo verdadero en caso de que haya tocado un objeto y falso en caso contrario (solo en caso de que haya un sensor de contacto incluido).

- Laser2D()

  Devuelve una medida con la distancia a la cual se sitúa el objeto con el cual está colisionando el sensor láser 2D (solo en caso de que haya un sensor laser2D incluido).

Con la definición de las funciones, se dio por concluido el apartado de interpretación de órdenes y movimiento del robot. Llegados a este punto, se concluyó el apartado de simulación con la inclusión de algunas monedas flotantes, que hicieran al usuario tener un tipo de objetivo al cual llegar con las órdenes transmitidas al robot. Estos objetivos se sitúan aleatoriamente en el entorno y han de ser atrapados por el robot.

### 3.5.3 Interfaz de simulación

La última etapa en el desarrollo de la aplicación fue el diseño e implementación de la interfaz de simulación. Esta interfaz debía presentar los botones necesarios para interactuar con ella y ejecutar el código recibido, además de presentar información técnica sobre el entorno. Los elementos incluidos fueron los siguientes:

1. **Campo de código:** En este campo se incluye el código recibido del módulo de bloques.
2. **Botón de ejecución:** Botón para comenzar la simulación.
3. **Posición del robot:** Sección que indica la posición en (x, y, z) del robot en todo momento.
4. **Orientación del robot:** Sección que indica la orientación en (x, y, z) del robot en todo momento.
5. **Botón para la carga del código:** Botón que carga desde fichero las órdenes que debe seguir el robot.
6. **Información de todos los sensores presentes:** Lista con todos los sensores que fueron añadidos al robot con la información que éstos devuelven en todo momento.
7. **Minimapa:** Pequeño minimapa con un *slider* para personalizar el zoom en él.

En la siguiente imagen se aprecia un ejemplo de ejecución de la simulación:

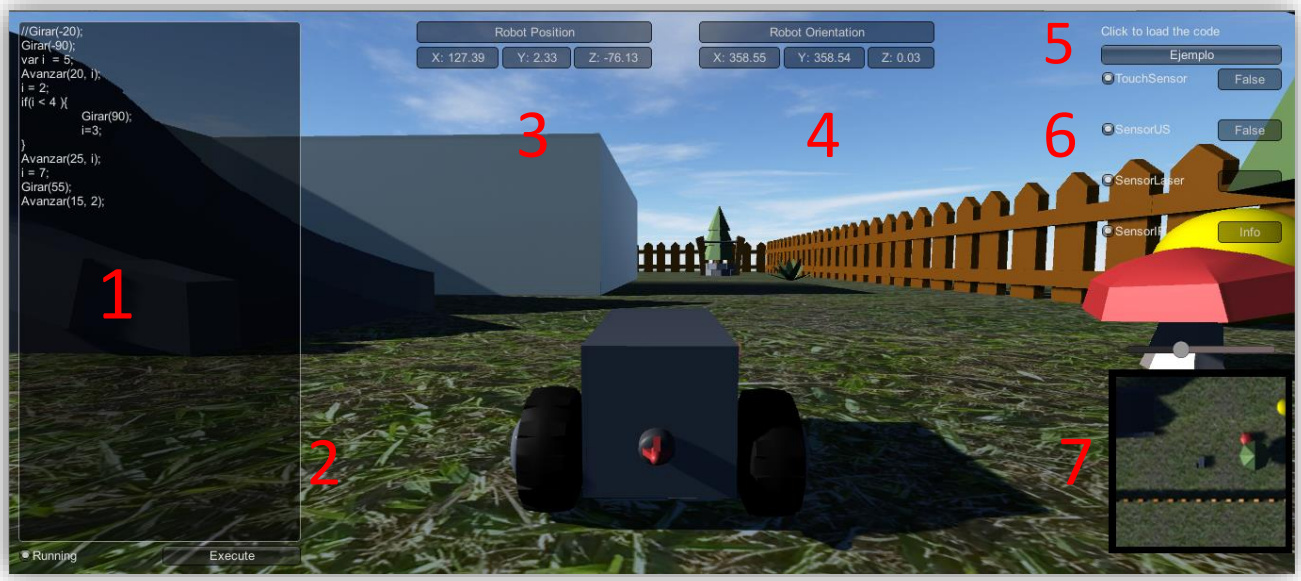



**Figura 12: Interfaz de simulación**

## 3.6 Problemas encontrados y modificaciones

Durante todo el tiempo de trabajo en este proyecto, han surgido algunos problemas o inconvenientes que han supuesto modificaciones a las líneas u objetivos marcados originalmente. Todos estos contratiempos han sido debatidos junto con los tutores de este proyecto para poner en común una solución. A continuación, se listan los problemas más significativos que fueron encontrados, así como la solución por la que se optó para solventarlos.

- **Unión con módulo de programación visual**

Uno de los primeros problemas encontrados, fue la unión de Blockly con Unity. En un primer lugar, se optó por intentar simular los bloques de Blockly dentro del propio entorno de simulación. Sin embargo, este proceso requería de bastante tiempo y tampoco contaba con un resultado óptimo garantizado. Por lo tanto, se optó por la comunicación a través de ficheros. Así, una vez finalizada la programación por bloques, el fichero de salida es almacenado en un directorio común, donde el simulador lee e interpreta el código generado.

- **Implementación de un intérprete de JavaScript para C#**

El mayor contratiempo que surgió en este proyecto fue la interpretación de órdenes desde Blockly a Unity. El código JavaScript generado por Blockly no puede ser ejecutado directamente en Unity, por lo que se pensó en desarrollar un parseador de este lenguaje a C#, que es el usado para los scripts de Unity. Este paso suponía un gasto temporal no contemplado en el inicio, no solo por su complejidad sino por los escasos precedentes de una herramienta así para Unity que se encontraron en proyectos similares. La solución de este problema se avanzó bastante al encontrar un asset que proporcionaba interpretación y ejecución de código externo en JavaScript en Unity. El complemento, llamado Jurassic, ahorró bastante tiempo de desarrollo y fue de gran ayuda en la elaboración. Su funcionamiento, se explica detalladamente en el punto 3.3.2.

- **Control de las instrucciones paso a paso**

Para simular el comportamiento del robot, es importante tener en cuenta que una instrucción no puede comenzar a la vez que se está ejecutando la anterior. En un inicio, se planteó definir dichas funciones dentro del método Update de Unity, el cual se ejecuta una vez por frame, en tiempo de ejecución. Esta opción hacía más fluido el movimiento, sin embargo, se pierde completamente el control del programa por parte del desarrollador ya que es el propio Unity el que se encarga de llamar al método. Esto imposibilitaba el poder controlar el orden secuencial, y el poder establecer tiempos entre órdenes, por ejemplo. La solución tomada al respecto fue la inclusión de corutinas a la escena. Como se mostrará en los algoritmos adjuntados en el apartado 8, las corutinas de Unity permiten ejecutar un método de forma independiente al hilo de ejecución del programa y esperar a que ésta termine para continuar con la ejecución. Esta solución fue la tomada en un principio, no obstante,

nos percatamos de que las corutinas se ejecutaban secuencialmente en el programa como se esperaba, pero el intérprete continuaba la ejecución del código.

Esto hizo modificar de nuevo el desarrollo de la interpretación de código. Finalmente, se optó por la inclusión de hilos a la aplicación. De esta manera, se inicia un hilo independiente al hilo principal de Unity en el cual se va a ejecutar el intérprete. Una vez el intérprete llama a alguna de las funciones que podemos definir, hacemos esperar este hilo en un bucle while, mientras el hilo principal de Unity realiza la operación deseada frame a frame hasta llegar al punto deseado o devolver el valor deseado en caso de un sensor. Cuando el trabajo relativo a esa función finaliza, se asigna a false un booleano global relativo a esa función desde el Update() de Unity. Este booleano hace al hilo de interpretación salir del bucle en el que se encontraba esperando y continuar con la lectura de la siguiente línea de código.

- **Modificación del entorno**

Entre las primeras tareas definidas, se propuso la generación del entorno dinámicamente, es decir, generar un entorno por defecto, y que posteriormente el usuario pudiera ir modificando las paredes, puertas, ventanas, etc. Esta tarea fue comenzada en un principio, no obstante, fue descartada más adelante al centrar en trabajo en el desarrollo del intérprete para la simulación que era uno de los puntos clave para un buen funcionamiento. La modificación del entorno sería un gran paso para la personalización de la aplicación y es uno de los puntos recogidos en las posibles líneas futuras del proyecto.

# Capítulo 4
# Resultados

Tras un amplio enfoque de las herramientas utilizadas, los recursos, y todas las fases de desarrollo de este proyecto, se procede a detallar el funcionamiento de la aplicación. En este apartado, se adjuntarán muchas capturas de pantalla de la aplicación, a modo de guía para el usuario.

## 4.1 Funcionalidad de la aplicación

Dentro del apartado de funcionalidad de la aplicación, se explican todas las diferentes posibilidades que ofrece al usuario durante su ejecución.

### 4.1.1 Carga del robot

El proceso de carga del robot se realiza a través de una escena intermedia de Unity. Una vez el robot es creado dentro el entorno de creación y configuración del robot, se exporta como asset y queda listo para ser añadido al simulador. Cuando dentro del simulador presionamos en cargar robot, se añaden, dinámicamente, algunas características necesarias para su funcionamiento dentro de este proyecto y queda guardado en la escena principal para su uso.

### 4.1.2 Carga de las instrucciones

En este apartado se detalla el proceso de unión con el módulo de programación visual. Para cargar las instrucciones, el simulador carga, desde el menú principal, el código generado por los bloques, que se encuentra alojado en un fichero situado en

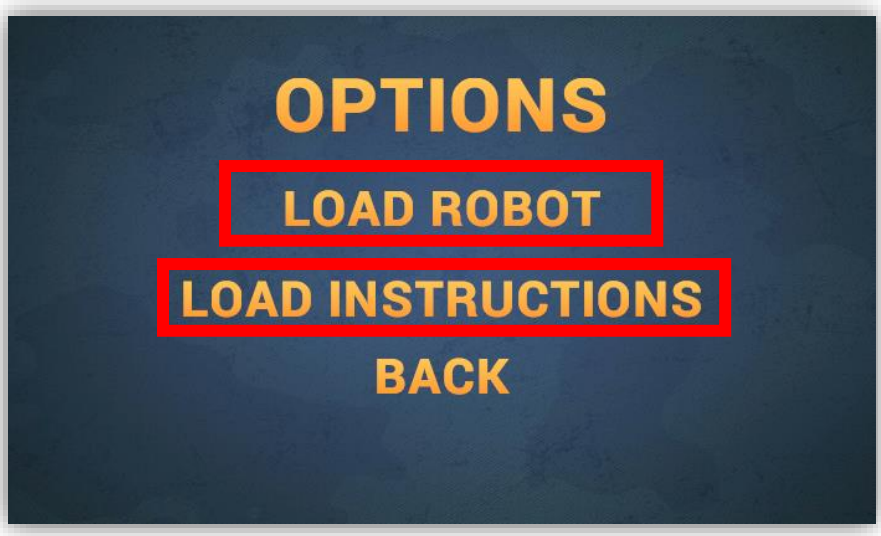



**Figura 13: Menú de carga de robot e instrucciones**

un directorio común. Este código, es guardado en una variable interna y pasado posteriormente al intérprete para su ejecución.

### 4.1.3 Simulación

Una vez cargado el robot y las instrucciones, la simulación puede ser realizada. Para ello, volvemos al menú principal, donde arrancamos el entorno de simulación pulsando Start.


**Figura 14: Menú principal**

La interfaz de simulación ofrece las funcionalidades citadas en el punto 3.5.3, que se desarrollan a continuación. Una vez iniciado el entorno de simulación, cargamos las instrucciones pulsando el botón situado en la parte superior derecha que indica el nombre del fichero a cargar, en este caso llamado “Ejemplo”. Cuando cargamos el código ejecutado, éste se visualiza en un gran recuadro a la izquierda de la pantalla, para comprobar las órdenes que está ejecutando el robot mientras se mueve. Para comenzar a ejecutar la simulación, basta con pulsar el botón situado en la esquina inferior izquierda que indica “Execute”.

Por otra parte, existen seis recuadros en la parte superior de la pantalla, tres relativos a la posición y otros tres relativos a la orientación. Aquí podemos comprobar en todo momento las coordenadas (x ,y ,z) del robot en ejecución. Todos estos elementos se encuentran resaltados en la siguiente ilustración.


**Figura 15: Elementos interactivos en la simulación.**

Otra de las funcionalidades de la simulación es la visualización de la información devuelta por los diferentes sensores que contiene el robot. En la parte superior derecha, se muestra dinámicamente, la información de los sensores disponibles en el robot configurado. Esta información se va actualizando conforme el robot se mueve, mostrando al usuario cuál es el valor de cada sensor realmente.

Por último, se ha añadido una funcionalidad extra, no planificada previamente, que ha sido la inclusión de un pequeño minimapa. Este minimapa ofrece una visión cenital del robot, útil para visualizar los elementos que tenemos alrededor de nuestro robot y que no se puedan ver porque tengamos una pared delante, por ejemplo. Además, se le ha añadido un *slider* a este minimapa, cuya función es la de alejar o acercar la vista aérea del robot.

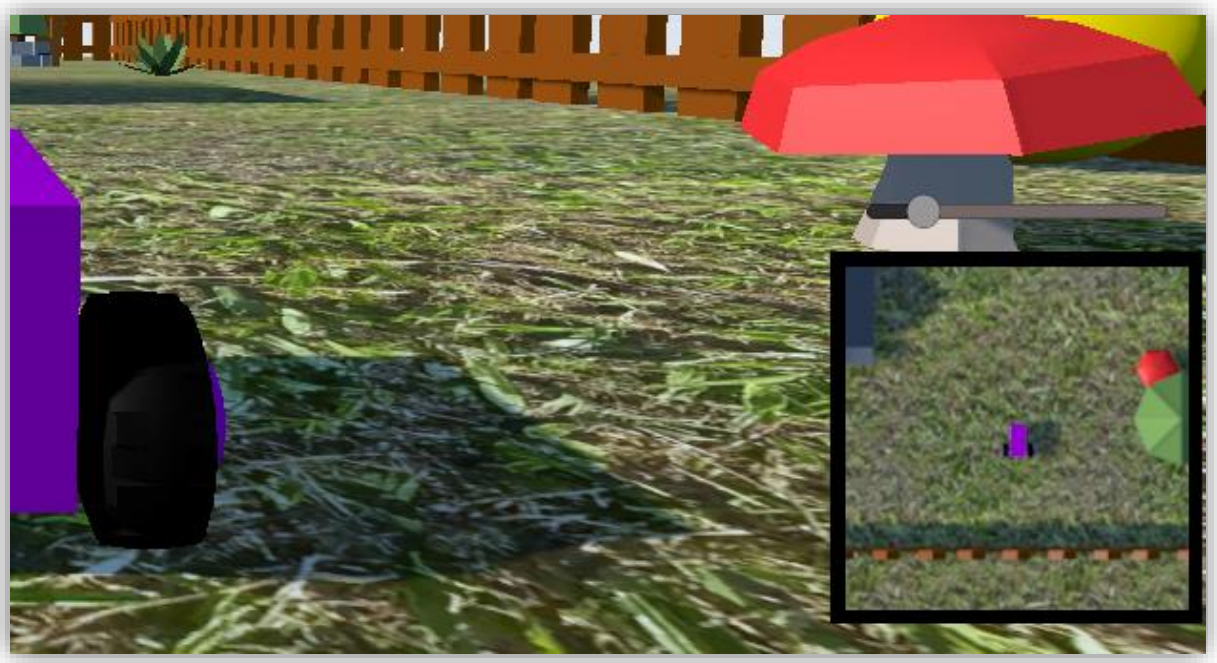


**Figura 16: Minimapa y slider, en tiempo de ejecución**

# Capítulo 5
# Conclusiones y líneas futuras

Como conclusión a este trabajo, se ha creado una aplicación que simula el comportamiento de un robot que recibe órdenes generadas por un lenguaje de programación visual. La aplicación final de escritorio, desarrollada en Unity, es capaz de cargar desde fichero tanto el código generado por el módulo de bloques como el robot generado desde el módulo de configuración, y sitúa ambos en el entorno. Ofrece al usuario retroalimentación de qué están mostrando los sensores en cada momento y cuál es la posición y la orientación del robot. Todo ello unificado en una misma interfaz gráfica, funcional y de fácil uso.

Se ha conseguido generar un entorno de simulación con diferentes elementos visuales que hacen que la simulación sea realista a la vez que visualmente atractiva, enfocada para usuarios en edad escolar. Este entorno además presenta algunos elementos interactivos, haciendo que el usuario tenga una meta u objetivo dentro del entorno.

Por otra parte, se ha conseguido unificar perfectamente la configuración del robot con el simulador, permitiendo al usuario guardar su robot configurado en tiempo de ejecución desde el módulo de configuración al de ejecución. Sincronizando ambos módulos para poder modificar dinámicamente la interfaz en función de los sensores añadidos y acceder a su información. De la misma manera, el módulo de programación visual se ha conseguido enlazar a través de un sistema de ficheros.

Todo esto ha hecho que los objetivos marcados en un principio hayan sido superados, para crear un proyecto novedoso, de gran ambición, y con mucha proyección de futuro para fomentar el pensamiento computacional en etapas escolares. El pensamiento computacional, como ya se detalló en la introducción del trabajo, comprende una serie de habilidades que puede generar grandes beneficios si se domina en etapas previas a las que actualmente están establecidas. Una de las metas de este proyecto es la de conseguir que los estudiantes sepan qué es la programación y para qué sirve, dándoles otra visión para resolver problemas y enseñándoles conceptos que, de no ser en estas etapas, es muy probable que no vuelvan a recibir.

Además, este proyecto establece un punto de partida para una aplicación aún mayor, con más elementos y características que por determinadas causas, como puede ser la falta de tiempo o de medios, no han podido ser implementadas. No obstante, en la siguiente página, se reflejarán algunas de las muchas líneas futuras que tiene la aplicación y las ventajas que éstas proporcionarían al proyecto en sí y a la incorporación de este trabajo en un entorno educativo. A continuación, se listan algunas de las líneas futuras relativas a este proyecto:

- **Implementación en la web**

  Una de las principales mejoras que presenta la aplicación de cara el futuro es la de poder integrar el proyecto dentro de un entorno web. Este paso, supondría que la aplicación fuese completamente independiente al software y hardware instalado en cada centro que se quiera ejecutar. Pudiendo ser accesible desde cualquier lugar en el que se disponga de un navegador y conexión a internet.

- **Posibilidad de modificación del entorno**

  Otra de las posibles líneas futuras es la de modificar el entorno antes de realizar la simulación. Este proceso permitiría un mayor grado de personalización de la aplicación y también mayor implicación del usuario en el proyecto, ya que tendría más capacidad para generar entornos a su gusto.

- **Posibilidad de cambio de instrucciones dinámicamente**

  Uno de los puntos tenidos en cuenta durante la realización de este proyecto fue la posibilidad de cambiar el comportamiento del robot en tiempo de ejecución mientras se ejecuta la simulación. Esto es, que el robot tenga la capacidad de parar de interpretar un determinado conjunto de órdenes y cambiar durante su transcurso a otro conjunto de instrucciones. Esta característica se descartó debido a que suponía un gran cambio a la lógica ya implementada. No obstante, es una buena línea de trabajo para modificar en un futuro.

- **Posibilidad de compactar los tres módulos en una sola aplicación**

  Hasta el momento, los tres módulos por separado han trabajado entre ellos a través de paso de ficheros, los cuales son secuenciales, es decir, el simulador no podría ser arrancado a no ser que encuentre un fichero con las instrucciones del segundo módulo y un objeto robot del primero. Además, una vez se llega al simulador, no existe la posibilidad de volver atrás y cambiar alguno de los parámetros configurados/programados.

  Si en un futuro esta aplicación se une en un mismo bloque, se abriría un amplio abanico de posibilidades de interacción entre módulos no implementados hasta el momento. Poder volver a la configuración del robot y cambiar sus características, volver atrás al módulo de programación y modificar el comportamiento del robot hasta lograr el deseado, entre otras posibilidades de interacción, son funcionalidades que ahora mismo no se pueden ofrecer y que le darían un mayor atractivo al proyecto.

# Capítulo 6
# Summary and Conclusions

At conclusion to this work, an application that simulates the behaviour of a robot which receives orders generated by a visual programming language has been developed. The final desktop application, developed in Unity, is capable of loading from file both the code generated by the visual programming language module and the robot generated by the robot configuration module, and places both in the environment. It offers the user feedback on what the sensors are showing every moment and what is the position and orientation of the robot. All these features are unified in a single graphic user interface, functional and easy to use.

A simulation environment has been generated. It has different visual elements that make the simulation realistic and visually attractive, focused on school-age users. This environment also presents some interactive elements, providing a set of challenges that can be solved by the users.

At the same time, the robot configuration module and the simulator have been integrated perfectly, allowing the user to save their configured robot at runtime from the configuration module to the simulation one. It has been synchronized both modules to be able to dynamically modify the graphical interface based on the sensors added and access their information. Similarly, the visual programming module has been linked through a file system.

All the above has meant that the aims set at the beginning of the project have been overcome, creating a new project, with great ambition and a lot of future projection to promote computational thinking in school stages. Computational thinking, as detailed in the introduction of the work, consist of a set of skills that may generate great benefits if it is mastered in previously stages to the currently established. One of the goals of this project is to get students to know what programming is and what is for, giving them another vision for solving problems and teaching them concepts, if not in these stages, it is very likely that they will not receive.

In addition, the project generates a start point for an even bigger application, with more elements and features which for certain causes, such as lack of time or resources, could not have been implemented yet. However, on the next page, we will see some of the many future lines this application has, and the advantages that these would provide to the project itself and to the incorporation of this work in an educational sector. Here are some of the future lines related to this project:

- **Web implementation**

  One of the main improvements in the application facing the future is to be able to integrate the project into a web environment. This step would suppose that the application could be independent to the software and hardware installed in each school that wants to implement it, being able to be accessible from any place where you have a browser and Internet connection.

- **Possibility of modification of the environment**

  Another of the possible future lines is the modification of the environment before executing the simulation. This process allows a better degree of personalization of the application and the greater involvement of the user in the project as well, who may have more capacity to generate environments to his liking.

- **Possibility of changing instructions dynamically**

  One of the points considered during the realization of this project was the possibility of changing the robot behaviour at runtime while the simulation is running. With this improve, the robot would have the ability to stop a set of commands and change to another set of instructions. This characteristic was discarded because it supposed a big change to the logic already implemented. However, it is a good line of work to modify in the future.

- **Possibility of compacting the three modules in a single application.**

  So far, the three modules have worked with each other through a file passing interface, which are sequential, that is, the simulator could not be started unless it finds a file with the instructions of the second module and a robot object of the first one. Furthermore, once the simulator is reached, there is no possibility of going back and changing any of the configured / programmed parameters.

  In the future, it would be desirable that this application joins modules all in a unique platform. Hence, a wide range of interaction possibilities would be opened that have not been implemented yet. Being able to return to the configuration of the robot and change its characteristics, going back to the programming module and modify the behaviour of the robot until achieving the desired one, among other possibilities of interaction, are functionalities that cannot be offered right now and would give a greater attractive to the project.

# Capítulo 7
# Presupuesto

A continuación, se muestra el presupuesto correspondiente a la elaboración de este proyecto.

## 7.1 Presupuesto

**Tabla 2: Resumen de presupuesto**

| Tareas | Horas | Precio/hora | Presupuesto |
|---|---|---|---|
| Revisión Bibliográfica | 10 h | 10 €/h | 100 € |
| Desarrollo de la aplicación | 150 h | 15 €/h | 2.250 € |
| Pruebas de la aplicación | 80 h | 15 €/h | 1.200 € |
| Desarrollo de la memoria | 50 h | 15 €/h | 750 € |
| **Total** | **290 h** | | **4.300 €** |

# Capítulo 8
# Scripts relativos a la simulación

## 8.1 Algoritmo de giro del robot

```
public void calculaPosGiro(){
        anguloFinal = (int) transform.rotation.eulerAngles.y + angulo;
        posinicial = transform.rotation.eulerAngles.y;
        if (anguloFinal > 360) {
            anguloFinal -= 360;
        }
        else if (anguloFinal < 0) {
            anguloFinal += 360;
        }
}

public bool girar(){

        // Giro a derecha
        if (angulo >= 0) {

            if ((transform.rotation.eulerAngles.y > anguloFinal) & (transform.rotation.
eulerAngles.y >= posinicial) & (!inverted)) {
                transform.Rotate (0, Time.deltaTime * 20, 0, Space.World);
                return true;
            } else if (transform.rotation.eulerAngles.y < anguloFinal) {
                inverted = true;
                transform.Rotate (0, Time.deltaTime*20, 0, Space.World);
                    return true;
            }

        } else { //Giro a izquierda
            if ((transform.rotation.eulerAngles.y < anguloFinal) & (transform.rotation.
eulerAngles.y <= posinicial) & (!inverted)) {
                transform.Rotate (0, -Time.deltaTime*20, 0, Space.World);
                 return true;
            }
            else if (transform.rotation.eulerAngles.y > anguloFinal) {
                inverted = true;
                transform.Rotate (0, -Time.deltaTime*20, 0, Space.World);
                return true;
            }

        }
        return false;

    }
```

**Tabla 3: Algoritmo para el movimiento de giro**

## 8.2 Algoritmo de avance del robot

```
public void calculaTiempoAvance(){
        tiempoAvance = Time.time + segundos;

}

public bool avanzar(){

    if (Time.time < tiempoAvance ) {
        transform.Translate (Vector3.forward * velocidad * Time.deltaTime);
        return true;
    }
    return false;

}
```

**Tabla 4: Algoritmo para el movimiento de avance**

## 8.3 Algoritmo de acceso a un sensor

```
public bool accesoLaser(){
       LaserDetection distance = (LaserDetection) GameObject.Find("/Robot/SensorLaser")
.transform.GetChild(3).GetComponent(typeof(LaserDetection));

       if (distance.getDetection ()) {
           laserdistance = (double)distance.getDistanceHit ();
       } else
           laserdistance = double.MaxValue;
       return false;
}
```

**Tabla 5: Algoritmo de acceso a un sensor (Sensor láser)**

# Bibliografía


[1] Gazebo Robot Simulator. (Accedido 03-02-2018) http://gazebosim.org/

[2] USARSim, Robot simulator for research and education. (Accedido 05-02-2018) https://ieeexplore.ieee.org/document/4209284/

[3] Webots. (Accedido 04-02-2018) https://www.cyberbotics.com/

[4] Microsoft Robotics Developer Studio. (Accedido 05-02-2018) https://msdn.microsoft.com/en-us/library/bb648760.aspx

[5] Robótica Wikipedia. (Accedido 20-04-2018) https://es.wikipedia.org/wiki/Rob%C3%B3tica

[6] Robótica educativa Wikipedia. (Accedido 25-04-2018) https://es.wikipedia.org/wiki/Rob%C3%B3tica_educativa

[7] Shannon, Robert; Johannes, James D. (1976). «Systems simulation: the art and science». *IEEE Transactions on Systems, Man and Cybernetics*. 6(10). pp. 723-724

[8] Simulación Wikipedia. (Accedido 16-05-2018) https://es.wikipedia.org/wiki/Simulaci%C3%B3n

[9] Blockly Wikipedia. (Accedido 16-05-2018) https://en.wikipedia.org/wiki/Blockly

[10] Unity Wikipedia. (Accedido 16-05-2018) https://es.wikipedia.org/wiki/Unity_(motor_de_juego)

[11] Unity Manual. (Accedido 16-05-2018) https://docs.unity3d.com/Manual/index.html

[12] Unity Scripting Reference. (Accedido 16-05-2018) https://docs.unity3d.com/ScriptReference/

[13] Unity Asset Store. (Accedido 16-05-2018) https://assetstore.unity.com/

[14] Unity Asset Store Tutorial. (Accedido 19-05-2018) https://unity3d.com/es/learn/tutorials/s/asset-store

[15] Programación visual Wikipedia. (Accedido 19-05-2018) https://es.wikipedia.org/wiki/Programación_visual

[16] Graphical vs Text-Based Coding for Kids. (Accedido 23-05-18) https://www.techagekids.com/2016/07/graphical-vs-text-based-coding-for-

kids.html

[17] Blockly Interface. (Accedido 25-05-2018) https://developers.google.com/blockly/

[18] Jurassic, JavaScript Runtime Interpreter for Unity. (Accedido 27-05-2018) https://assetstore.unity.com/packages/tools/integration/jurassic-javascript-runtime-interpreter-for-unity-2345

[19] Jurassic Wiki. (Accedido 27-05-2018) https://github.com/paulbartrum/jurassic/wiki

[20] RVM CAD Robotic. (Accedido 06-06-2018) http://www.rvmcad.co.in/robotic.php

[21] AxeyWorks, Low Poly, Asset Store. (Accedido 11-06-2018) https://assetstore.unity.com/packages/3d/environments/low-poly-free-pack-58821

[22] BigFurniturePack, Asset Store. (Accedido 11-06-2018) https://assetstore.unity.com/packages/3d/props/furniture/big-furniture-pack-7717

[23] Text Mesh Pro, Asset Store. (Accedido 11-06-2018) https://assetstore.unity.com/packages/essentials/beta-projects/textmesh-pro-84126

[24] Wispy Sky, Asset Store. (Accedido 11-06-2018) https://assetstore.unity.com/packages/2d/textures-materials/sky/wispy-skybox-21737

[25] Robot Mesh. (Accedido 15-06-2018) https://www.robotmesh.com/

[26] Robot Mesh Studio, Virtual VEX IQ Online. (Accedido 15-06-2018) https://www.robotmesh.com/studio/258502

[27] Github. (Accedido 16-06-2018) https://github.com/