UNIVERSIDAD DE OVIEDO

ESCUELA DE INGENIERÍA INFORMÁTICA

TRABAJO FIN DE GRADO

"Simulador de robots para la asignatura de Software para robots"

DIRECTOR: Cristian González García

AUTOR: Diego Fernández Suárez

Agradecimientos

A mi familia en general, les agradezco haber estado siempre ahí apoyándome, tanto en los buenos momentos como en los malos y sobre manera durante estos últimos años de carrera, aguantándome en mis periodos de estrés, ansiedad, nervios o desánimo; que no han sido pocos. Muchas gracias por todo y sobre manera por animarme siempre a seguir adelante con mis sueños.

A todos los profesores del grado que han contribuido de una forma u otra a mi aprendizaje durante todos estos años. En especial, a Cristian; el tutor de mi proyecto, que siempre ha estado a mi disposición y me ha resuelto cuantas dudas me han surgido; además de proporcionar ideas en todo momento para mejorar la calidad del proyecto.

Gracias a mis *betatesters*, por reservar un poco de tiempo de vuestras vidas para ayudarme a realizar las pruebas del sistema y darme algo de *feedback* sobre la aplicación.

Por último y no menos importante, a mis compañeros de grado y amigos (tanto del grado como de fuera). Gracias por apoyarme en todo momento y por hacer más ameno el tiempo que he pasado estudiando esta carrera. Gracias por ayudarme en los momentos en los que lo he necesitado, me llevo un gran recuerdo de toda esta etapa de mi vida.

Resumen

Hoy en día existen varios sistemas que simulan el comportamiento de Arduino, muchos de ellos de uso gratuito. Pero ninguno de ellos cubre en su totalidad las necesidades de la asignatura de Software para Robots.

Por ello surge este proyecto, el cual va a consistir en la creación de una aplicación o simulador que permita a los usuarios (alumnos de la asignatura de Software para Robots) desarrollar los programas (o sketches) que se usan para especificar el comportamiento de los robots.

En concreto se va a desarrollar un sistema que simule dos tipos de robots: un actuador lineal y un robot móvil (dos realmente, uno con dos sensores infrarrojos para el seguimiento de líneas y otro con cuatro sensores del mismo tipo). Permitirá en todo momento ajustar los pines de la placa Arduino a los que se conectan los elementos, además de realizar la compilación del código y, dependiendo de este, mostrar errores, alertas y/o ejecutar el código.

El sistema requiere de dos partes bastante diferenciadas: el *front-end*, que será la interfaz gráfica de la aplicación, la animación de los robots una vez se ejecuta el código y la salida y/o entrada por consola; y el *back-end* que gestiona todas las fases de la compilación de código, las llamadas a librerías, la gestión de archivos y de estado de los componentes y placas de Arduino.

Palabras Clave

Simulador de Arduino, Compilador, Python, ANTLR4, Tkinter, Robot móvil, Actuador lineal.

Abstract

There are some systems that simulate the behavior of Arduino; most of them are free. However, none of them fills all the needs of the subject of robotic software.

That is the origin of this project; it will consist of the creation of an application or simulator that allows the users (students of the subject) to develop the programs (or sketches) that are used to specify the behavior of the robots.

A system that simulates two types of robots will be created: a linear actuator and a mobile robot (which will be two, one with two infrared sensors and one with four of the same type). It will always allow the adjusting of the pins of the Arduino board to which the elements will be connected, besides doing the compilation and, depending on it, showing errors, warnings and/or executing the code.

The system requires two differentiated parts: the front-end, which will be the graphical interface of the application, the animation of the robots once the code is executed and the input and/or output by console; and the back-end, which will manage all compiler phases, library calls, file management, and the state of the components and Arduino boards.

Keywords

Arduino simulator, Compilator, Python, ANTLR4, Tkinter, Mobile robot, Linear actuator.

Índice General

Índice de Figuras

Índice de Tablas

Capítulo 1. Memoria del Proyecto

En este capítulo se resume el porqué de la realización de este proyecto y lo que se pretende con su realización.

1.1 Resumen de la Motivación, Objetivos y Alcance del Proyecto

El proyecto está motivado por un problema de la asignatura de Software para Robots; que se ha visto agravado por la pandemia del COVID-19. Dicho problema es la dificultad para probar diferentes ejercicios de la asignatura en los que es necesario utilizar unos elementos de hardware concretos: un robot móvil y un actuador lineal.

Hoy en día, para probar los ejercicios, es necesario hacerlo de manera presencial en la Escuela de Ingeniería Informática; requiriendo realizarlas en un aula preparada para ello o, en su defecto, probarlo en las propias clases de prácticas de laboratorio. Esto es un problema por tres razones principalmente:

- Los horarios, ya que es necesario reservar previamente esa aula común para todos los alumnos de la asignatura, lo que genera tiempos de espera para poder realizar los ejercicios.
- La obligación de los alumnos a desplazarse a la universidad, problema que se acentúa según la distancia que tengan que recorrer los alumnos para ello.
- Algunas veces se puede dar la situación de que no haya baterías disponibles para probar los robots.

Los objetivos de este proyecto son:

- Desarrollar una aplicación para poder realizar las pruebas de los ejercicios sin tener que recurrir al hardware necesario; es decir, crear un simulador que los alumnos puedan emplear desde donde quieran. Dicha aplicación deberá:
    - Proporcionar una forma de visualizar los movimientos del robot móvil y del actuador lineal.
    - Permitir la codificación de ambos dispositivos en un lenguaje idéntico a Arduino.
    - Dar una salida por consola en caso de que el código tenga fallos o se necesite imprimir mensajes
- Diseñar el software y la arquitectura del sistema de tal manera que sea fácil de mantener y, en caso de que sea necesario, de ampliarlo.
- Permitir al usuario gestionar el archivo de código de forma fácil, permitiendo crear archivos, importarlos, guardarlos y editarlos.
- Crear una interfaz fácil e intuitiva para el usuario.

El alcance del proyecto será únicamente la implementación de los dos dispositivos anteriormente mencionados en este apartado.

1.2 Resumen de Todos los Aspectos

En este apartado se resumen los diferentes apartados que se tratarán a lo largo del presente documento.

1.2.1 Introducción

La asignatura de Software para Robots que se imparte en el Grado de Ingeniería del Software en los cursos de tercero o cuarto como asignatura optativa tiene un problema que se ha visto agravado a raíz de la pandemia del COVID-19, y es que ciertos robots (en concreto los robots móviles y el actuador lineal) no pueden ser probados por los alumnos sin que estos tengan que ir presencialmente a un aula o laboratorio de la escuela para ello.

En un contexto de normalidad, el depender de piezas concretas únicamente disponibles en la asignatura genera problemas de tiempo y organización a los alumnos, no así de desplazamiento ya que deben acudir a la escuela normalmente. En el contexto de la pandemia, teniendo que aislarse para evitar contagios, genera problemas de desplazamientos y, además de ello, también genera riesgo para los alumnos.

Aunque llegue (o haya llegado, dependiendo del contexto) el día del final de la pandemia y la vuelta a la normalidad, el problema subyacente de los horarios para los alumnos seguirá presente. Es por ello por lo que surge este proyecto.

El sistema que se desarrollará pretende ofrecer a los alumnos (y realmente a cualquier usuario de Arduino) una herramienta para poder ejecutar y probar sketches que implementen el movimiento de los dos robots sin falta de recurrir al hardware físico necesario.

1.2.2 Análisis y diseño

El proyecto que se ha desarrollado será un simulador para la implementación de código para los robots móviles o el actuador lineal. La funcionalidad que se pretende ofrecer a los usuarios es la siguiente:

- Importar y guardar archivos.
- Permitir el desarrollo de código a través de un editor.
- Mostrar el comportamiento esperado de los robots mediante una simulación en 2 dimensiones desde un punto de vista cenital.
- Mostrar toda salida del código (errores, advertencias y texto) mediante una consola.
- Permitir una configuración sencilla de los robots (cambiar pines y cambiar de robots).

1.2.3 Planificación y presupuesto

Se ha realizado una planificación y un presupuesto de manera que el proyecto se pueda realizar en 4 meses y con un presupuesto de 36.541,85 €. No obstante, la planificación final ha resultado en 9 meses y un presupuesto de unos 78.546,37 €.

1.2.4 Implementación

Para implementar el sistema se ha decidido usar el lenguaje de programación Python, así como la herramienta ANTLR4 para realizar la parte del compilador y tkinter para la interfaz gráfica. Para el desarrollo se utilizarán Visual Studio Code y Notepad++ (principalmente el primero de los dos entornos).

1.2.5 Pruebas

Las pruebas serán ejecutadas una vez finalizado cada módulo, para así facilitar el proceso de desarrollo de la aplicación. Se arreglarán todos los fallos detectados durante las pruebas antes de proceder al desarrollo del siguiente módulo.

Las pruebas de usabilidad y accesibilidad se han realizado al finalizar el desarrollo de la aplicación, para así obtener una retroalimentación completa del funcionamiento del sistema desarrollado.

Capítulo 2. Introducción

En este apartado se explicará las razones por las que se desarrolla este proyecto, además de establecer los objetivos que este se plantea. Por último, se expondrán las diferentes alternativas disponibles con respecto al sistema que se pretende implementar, como a las librerías que se van a emplear finalmente para el desarrollo, además de las alternativas para la implementación del compilador.

2.1 Justificación del Proyecto

El proyecto tiene su origen en la necesidad de probar ciertos ejercicios de la asignatura Software para Robots que requieren un hardware específico. Dicho hardware es, en concreto, un actuador lineal y un robot móvil, ambos diseñados por la escuela y disponibles en la facultad. Sería bastante complejo facilitar la disponibilidad de los robots para todos los alumnos. En tal caso, o bien sería la universidad la que tuviese que proveer el material, o bien los alumnos deberían comprar e imprimir los materiales necesarios.

Todo ello, hace que los alumnos tengan que desplazarse hasta la universidad para realizar las tareas que involucran al hardware. El principal problema surge a raíz de la pandemia del COVID-19, que hizo que las clases pasasen de ser presenciales a ser a distancia. Esta situación hizo que los alumnos tuviesen que desplazarse a la facultad, lo que implicaba bastante riesgo, ya que al tener que ir a la universidad, estos se exponen más al contagio. Ahora, en el curso actual (2021/22), se ha retomado la presencialidad de las clases, aunque esta va sujeta a que la evolución de la pandemia sea favorable.

Además, existe el problema de que se debe reservar una sala para realizar las tareas, lo que complica la gestión de horarios entre alumnos que cursan la asignatura. Por último, y aunque los alumnos deben desplazarse en algún momento para probar la tarea, el número de desplazamientos necesarios puede llegar a ser excesivo dependiendo del caso o incluso muy costoso, dependiendo de la lejanía y/o el transporte disponible.

Esto es problemático por varias causas:

- En el contexto de la pandemia pone en riesgo la salud de los alumnos ya que, al forzarles a desplazarse, estamos aumentando el riesgo de que se contagien de la enfermedad.
- Con anterioridad a la pandemia ya era un problema, ya que se tenía que reservar un aula para la realización de los ejercicios. Esto genera un problema con los horarios, ya que los alumnos deben de ponerse de acuerdo o adaptar su horario a las necesidades de otros.
- Además, el desplazamiento hasta la universidad puede ser costoso y/o complicado para los alumnos, sobre todo cuanto más lejos vivan del centro.

El proyecto tiene la intención de crear un sistema a través del cual se pueda:

- Crear e importar archivos que sean compatibles con el IDE de Arduino.
- Programar en el lenguaje propio de Arduino.
- Simular el comportamiento de los dos robots mencionados anteriormente en este apartado.
- Mostrar por consola toda salida que se deba realizar por dicho medio, incluyendo los errores que se puedan dar en el código.

El sistema resultado del proyecto será una aplicación para simular, que permita al usuario (los alumnos) programar los robots diseñados por la escuela utilizando el lenguaje de Arduino, además de poder visualizar que movimientos van a realizar los robots y la salida que se pueda generar por consola. Otras soluciones que se encuentran a través de internet no incorporan la posibilidad de probar los robots. Por ejemplo, Tinkercad, que es la opción más famosa en cuanto a simular Arduino se refiere, no incorpora la funcionalidad que necesitamos, por ello surge este proyecto.

2.2 Objetivos del Proyecto

El objetivo principal del proyecto es crear un sistema que permita a los alumnos realizar de una forma más simple los ejercicios que requieren de un actuador lineal o de un robot móvil para su realización.

Para ello, se plantean los siguientes objetivos:

1. Crear un sistema de simulación de estos robots compatible con Windows.
2. Implementar una interfaz gráfica de usuario intuitiva y fácil de entender para el usuario final.
3. Proporcionar una forma de visualizar los movimientos de ambos dispositivos.
4. Permitir al usuario escoger que tipo de dispositivo va a utilizar para probar.
5. Permitir la codificación en un lenguaje idéntico a Arduino.
6. Dar la capacidad al usuario de poder rehacer o deshacer las acciones que hayan llevado a cabo en la codificación.
7. Dar una salida por consola
   7.1. Que muestre los fallos de código
   7.2. Que permita imprimir mensajes
8. Diseñar el software y la arquitectura de forma que sea fácil de mantener y de ampliar.
9. Permitir al usuario gestionar los archivos de código, permitiendo crear archivos, importarlos, guardarlos y editarlos.
10. Con respecto a la creación de archivos, dar la posibilidad de decidir si se crea un archivo preparado para el actuador lineal, para el robot móvil o en blanco.

Para poder probar si funciona un robot concreto, debe de escogerse dicho robot y su configuración debe de ser correcta (es decir, los pines deben ser correctos).

2.3 Estudio de la Situación Actual

En este apartado se muestran las diferentes alternativas al sistema que vamos a desarrollar en este proyecto. Se describirán las funcionalidades de cada una y se comparará con nuestro sistema; en que se parecen y en que discrepan.

También se mostrarán las diferentes alternativas tecnológicas que se han tenido en cuenta a la hora de desarrollar el sistema, y por qué finalmente se han escogido unas frente a otras. Estas tecnologías están relacionadas tanto con el desarrollo del sistema, es decir, con las librerías y/o *frameworks* usados; como con el lenguaje de programación usado.

Además, se desarrollarán las alternativas posibles a la hora de diseñar el compilador y por qué no han sido escogidas.

2.3.1 Evaluación de Alternativas

En este apartado se estudiarán diferentes aplicaciones que ofrecen funcionalidades similares al sistema que se va a desarrollar.

2.3.1.1 Tinkercad

Tinkercad **[Tinkercad11]** es una aplicación web gratuita que permite realizar diseños 3D, electrónica y codificación. En concreto, nos da la opción de poder crear circuitos con componentes como los que se usan en la asignatura de Software para Robots: placas de Arduino, placas de prueba, pulsadores, leds…

El programa permite diseñar previamente los circuitos, con la posibilidad de que utilicen otros dispositivos que no sean Arduino; y el código que se ejecutará posteriormente en ese circuito, ofreciendo la posibilidad de programar con bloques, con texto o con los dos a la vez.

2.3.1.1.1 Ventajas

- Gran variedad de componentes, sensores y opciones a la hora de diseñar los circuitos.
- Permite guardar en nuestra cuenta online nuestros diseños.
- Muy intuitiva y fácil de usar.
- Es gratuita.
- Permite registrarse muy rápidamente

2.3.1.1.2 Inconvenientes

- No tiene como componentes el robot móvil y el actuador lineal.
- No está pensada para circuitos complejos o avanzados.
- No es una aplicación pensada exclusivamente para circuitos.

2.3.1.1.3 Conclusiones

Tinkercad es una aplicación web muy útil para realizar diseños básicos con Arduino, así como aprender a usar esta tecnología. Al ser una aplicación web nos permite guardar nuestros diseños en la nube y podemos acceder muy rápido a ella.

Aunque es una aplicación muy útil y puede ser usada por los alumnos de la asignatura para la primera parte de esta, donde los componentes usados no son tan complejos, la aplicación no se adapta 100% a nuestras necesidades; buscamos una aplicación que implemente los dos robots que necesitamos.

Figura 2.1 Logo de Tinkercad

2.3.1.2 Wokwi

Se trata de otra aplicación online, en este caso de código abierto, para simular circuitos electrónicos y de Arduino **[Wokwi19]**. Está pensada para aprender a programar con Arduino y prototipar. Soporta diferentes placas de Arduino: UNO, Mega y Nano entre otras, además de una gran variedad de componentes, como leds, sensores, pantallas, teclados…

Para usar la aplicación no hace falta registrarse, pero sí que hace falta para poder guardar nuestros diseños. Se puede utilizar para ello nuestra cuenta de Google, de GitHub o de email. Una vez registrados ya podríamos guardar sin problemas.

La aplicación ofrece la posibilidad de añadir componentes a nuestro diseño, así como eliminarlos o modificarlos. En la parte izquierda se puede hacer el programa que ejecutará el Arduino o editar los componentes que tiene el circuito.

2.3.1.2.1 Ventajas

- Gran variedad de placas.
- Muchas opciones de edición para los circuitos.
- Permite guardar los diseños en tu cuenta.
- Es gratuita.

2.3.1.2.2 Inconvenientes

- Falta de implementación del robot móvil y del actuador lineal.
- La documentación de la aplicación es escasa.

2.3.1.2.3 Conclusiones

Wokwi es una aplicación web bastante avanzada que nos permite diseñar circuitos y programas para estos, pudiendo guardarlos en nuestra cuenta en la página y ofreciendo una gran variedad de componentes.

Aún a pesar de ser una aplicación bastante avanzada y útil para poder simular una gran variedad de componentes, no tiene aquellos que buscamos y por ello queda descartado como alternativa.

Figura 2.2 Logo de Wokwi

2.3.1.3 PICSimLab

Se trata de una aplicación de escritorio para Windows, Ubuntu o MacOS que permite emular diferentes placas similares a las de Arduino, además de la propia Arduino UNO usada en la asignatura **[PICSimLab16]**.

Permite diseñar nuestros propios circuitos, además de cargar archivos HEX o cargar directamente desde el propio IDE de Arduino. Es un programa bastante útil para usuarios avanzados, integrando en un mismo programa diferentes placas.

2.3.1.3.1 Ventajas

- Gran variedad de placas ajenas a Arduino.
- Es gratuita.

2.3.1.3.2 Inconvenientes

- Falta de implementación del actuador lineal y del robot móvil.
- No implementa muchos componentes.
- La interfaz gráfica de usuario es bastante poco intuitiva.
- No se puede programar directamente en la aplicación, hay que hacerlo desde el IDE de Arduino y cargar el código compilado.

2.3.1.3.3 Conclusiones

PICSimLab es un simulador muy enfocado a la emulación de diferentes placas, entre ellas varias de Arduino. Sin embargo, no contiene una gran cantidad de componentes y está muy enfocada a usuarios avanzados. Por último, y lo que descarta totalmente esta aplicación es que no implementa los dos robots que necesitamos.

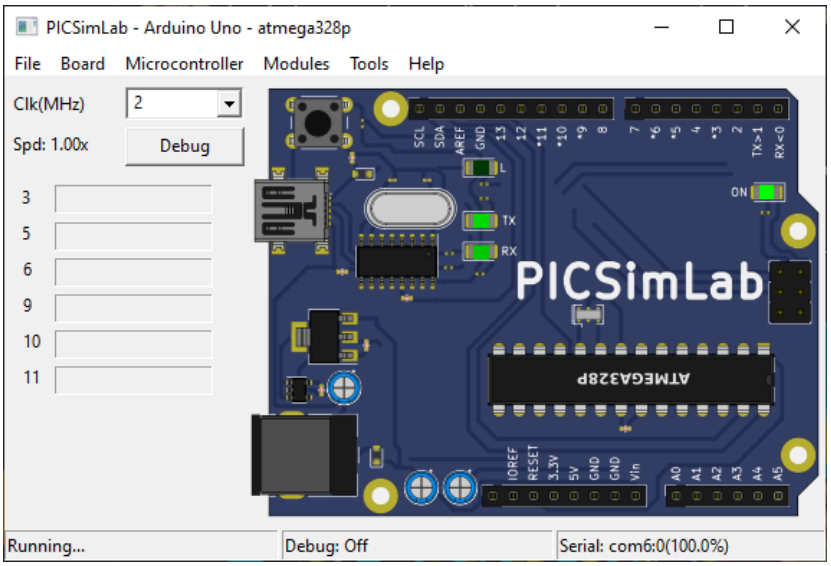


Figura 2.3 Interfaz gráfica de PICSimLab

2.3.1.4 UnoArduSim

Esta es una aplicación educativa en la cual, como en las anteriores, puedes diseñar tus propios circuitos y programas **[UnoArduSim17]**. En este caso, no tiene demasiada libertad a la hora de añadir o quitar componentes.

La aplicación permite la programación de código en el lenguaje de Arduino, además de ofrecernos el valor de las variables que declaremos a lo largo del código. Vienen algunos componentes ya incorporados en el propio diseño, dejando a nuestra elección si usar un componente determinado o no.

A pesar de ello, esto no quiere decir que no podamos eliminarlos del diseño, podemos establecer su cantidad a cero; o aumentarla según las posibilidades del programa, ya que el número total de componentes que se pueden usar está limitado.

2.3.1.4.1 Ventajas

- Es una aplicación de escritorio con todos los componentes ya integrados en ella.
- Tiene un depurador de código, lo que facilita el aprendizaje de Arduino.
- Permite ver las variables en ejecución.
- Permite escoger los componentes que vamos a usar.
- Es gratuita.

2.3.1.4.2 Inconvenientes

- Falta la implementación del actuador lineal y del robot móvil.
- Su interfaz gráfica es muy recargada y compleja, lo que dificulta un aprendizaje rápido de la herramienta.
- La forma de añadir o eliminar componentes no es intuitiva para el usuario, teniendo que hacerlo desde un submenú.

2.3.1.4.3 Conclusiones

UnoArduSim es una aplicación de escritorio enfocada al aprendizaje de Arduino, que nos permite codificar en la propia aplicación y escoger los componentes que vamos a usar de entre una lista limitada de ellos.

Lo que descarta el uso de esta aplicación para resolver nuestro problema es que no implementa como posibles componentes los dos robots que necesitamos.

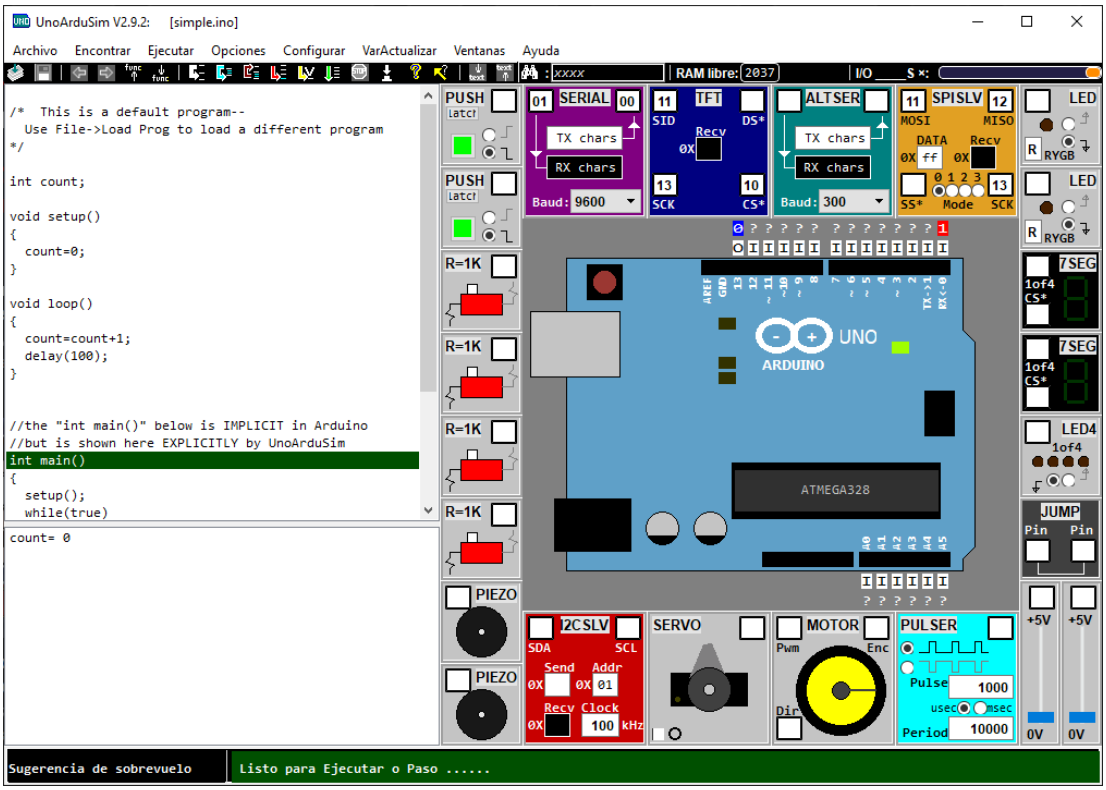


Figura 2.4 Interfaz gráfica de UnoArduSim

2.3.2 Estudio de Alternativas de Lenguaje

En este apartado se estudiarán los diferentes lenguajes alternativos en los que se podría desarrollar el proyecto.

2.3.2.1 Java

Java **[Java96]** es un lenguaje de programación orientado a objetos y una plataforma informática. Hoy en día, es uno de los lenguajes de programación más usados a lo largo del mundo, muchas aplicaciones e incluso videojuegos como Minecraft están implementadas en este lenguaje. Fue comercializado por primera vez en 1995 por Sun Microsystems. En la actualidad pertenece a Oracle.

La sintaxis de Java es similar a la de otros lenguajes de programación conocidos, como pueden ser C o C++. Los ficheros Java contienen clases o interfaces, que se definen mediante su

declaración y su implementación, esta última está rodeada por corchetes ({,}). Los métodos componen las clases, y pueden ser públicos, privados o protegidos.

Java no tiene herencia múltiple ni permite la sobrecarga de operadores, si bien es cierto que permite herencia a través de clases abstractas y de interfaces, siendo estas últimas una manera de especificar métodos (sin implementación) que van a tener que implementar las clases que hereden de ella. Las clases abstractas sí que permiten implementar métodos, aunque dan la opción de crear métodos abstractos, que deben ser implementados por las clases que hereden de ella, de manera similar a las interfaces.

El software desarrollado en este lenguaje se compila a bytecode, pudiendo este ser ejecutado en cualquier máquina virtual de Java, lo que hace que Java sea muy versátil y compatible con gran parte del hardware existente.

2.3.2.1.1 Ventajas

- Independencia con respecto a la plataforma, lo que permite desarrollar software multiplataforma de una manera muy sencilla.
- Comunidad muy amplia.
- Se orienta mayormente hacia el paradigma orientado a objetos, haciéndolo un lenguaje más sencillo.
- Estudiado y utilizado durante gran parte del grado.

2.3.2.1.2 Inconvenientes

- Su rendimiento en determinadas circunstancias puede llegar a ser muy pobre.
- No permite programar a bajo nivel.
- No hay control sobre el recolector de basura.

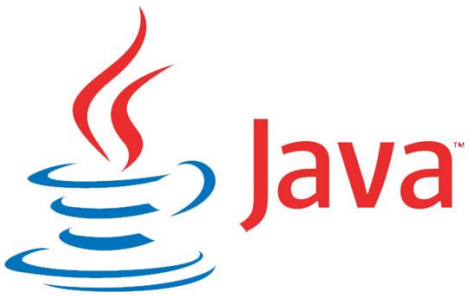


Figura 2.5 Logo de Java

2.3.2.1.3 Conclusiones

Aunque Java es un lenguaje de programación muy usado y perfectamente válido para este proyecto, he decidido descartarlo por la simple razón de que me he planteado hacer este proyecto en un lenguaje de programación que no fuese este, ya que lo usamos durante la mayor parte de la carrera. De esta manera puedo aprender a utilizar en profundidad nuevos lenguajes de programación.

2.3.2.2 C#

C# **[CSharp00]** es un lenguaje de programación multiparadigma desarrollado por Microsoft, formando este parte de la plataforma .NET. Es un lenguaje de programación muy popular y nos permite usar entre otros paradigmas el orientado a objetos, el funcional y el estructurado; aunque está principalmente enfocado al orientado a objetos.

Su sintaxis es similar a la que tienen otros lenguajes como C o Java, con clases y métodos delimitados por corchetes. Aunque a diferencia de los otros C, este no necesita los archivos .h o de cabecera, que se usan para declarar las clases en C y C++.

El lenguaje de programación comenzó su desarrollo en al año 1999, y fue anunciado públicamente en el año 2000, aunque su versión 1.0 no sería publicada hasta 2002.

2.3.2.2.1 Ventajas

- Es un lenguaje de programación simple y moderno.
- Soporta diferentes paradigmas, entre ellos el orientado a objetos.
- Permite programar con o sin punteros, dependiendo de la configuración del proyecto.
- Permite controlar la recolección de basura y así recuperar recursos no usados.
- Es un lenguaje de programación muy seguro.

2.3.2.2.2 Inconvenientes

- Al ser un lenguaje perteneciente a .NET de Microsoft, es dependiente de Windows.
- Aunque es un lenguaje bastante simple, tiene una curva de aprendizaje bastante pronunciada, sobre todo con LINQ.

2.3.2.2.3 Conclusiones

C# es un lenguaje de programación muy útil si se pretende usar paradigma orientado a objetos. Tiene muchas características interesantes como LINQ o las variables de tipo implícito. Sin embargo, al tener una curva de aprendizaje tan pronunciada, este queda descartado para la realización del proyecto.

Figura 2.6 Logo de C#

2.3.2.3 JavaScript

Es un lenguaje de programación interpretado, muy usado en el desarrollo web. Se trata de un lenguaje de programación que solo utiliza un hilo y que soporta entre otros paradigmas la orientación a objetos y el paradigma funcional, lo que hace a este lenguaje multiparadigma **[JavaScript95]**.

El estándar para JavaScript es ECMAScript, que es soportado actualmente por todos los navegadores modernos como Chrome, Edge o Firefox. Actualmente se encuentra por la 11ª edición, también conocida como ECMAScript 2020, publicada en junio de 2020.

Su sintaxis es similar a la de Java o C, en el sentido de que las funciones se delimitan con corchetes, pero el lenguaje tiene una sintaxis más particular con respecto a otros lenguajes. Las variables se definen mediante la palabra "var", no mediante el tipo. Tampoco ofrece funcionalidades de entrada/salida, aunque se puede usar "console" para imprimir mensajes por la consola de depuración.

2.3.2.3.1 Ventajas

- Es un lenguaje de programación muy popular y con el auge del desarrollo web cada día está más de moda.
- Es compatible con cualquier navegador.
- Es un lenguaje muy versátil gracias a entornos como Node.js.
- Se ejecuta con rapidez.

2.3.2.3.2 Inconvenientes

- Sin el estándar ECMAScript, JavaScript no es compatible de la misma manera con todos los navegadores, siendo muy recomendable ceñirse a ECMA.
- Puede ser inseguro si el programador no tiene precaución.

2.3.2.3.3 Conclusiones

JavaScript es un lenguaje de programación orientado al desarrollo web y que goza de una gran popularidad. Lo que descarta a este lenguaje de programación es que sería necesario desarrollar una página web en vez de una aplicación de escritorio.

Figura 2.7 Logo de JavaScript

2.3.3 Estudio de Alternativas de Librerías de Interfaz Gráfica de Usuario

En este apartado se van a tratar las diferentes librerías alternativas en las que se podría desarrollar la interfaz gráfica de usuario.

2.3.3.1 PyQT5

Se trata de una librería desarrollada por Riverbank Computing, que está desarrollada mediante el conjunto de librerías Qt **[PyQT516]**. Dicho conjunto es usado a través de múltiples plataformas y sistemas operativos para la creación de Interfaces Gráficas de Usuario. Qt permite el acceso a diferentes funcionalidades, como la localización, acceso a contenido multimedia, conectividad con NFC y/o bluetooth, un navegador basado en Chromium y también permite el desarrollo de interfaces de usuario tradicionales.

El objetivo de esta librería es juntar el lenguaje de programación Python con el *framework* Qt, implementando más de 35 módulos de extensión.

2.3.3.1.1 Ventajas

- Es de código abierto.
- Está implementado para el desarrollo de aplicaciones multiplataforma.
- Ofrece muchos componentes para la interfaz, todos ellos con diseños básicos para cada plataforma soportada.
- Tiene varios sitios web desde donde aprender a usar la librería.

2.3.3.1.2 Inconvenientes

- Carece de documentación específica para Python.
- Requiere bastante tiempo aprender a usar las características más avanzadas.

2.3.3.1.3 Conclusiones

PyQT5 es una librería que a nivel de funcionalidad encaja perfectamente en las necesidades del sistema que estamos desarrollando en este proyecto. Sin embargo, la falta de documentación específica de la librería hace que sea más costoso de aprender que tkinter (la librería que finalmente se ha seleccionado).

Figura 2.8 Logo de PyQT

2.3.3.2 wxPython

Es un *toolkit* multiplataforma que traduce la librería wxWidgets, escrita en C++, para usarla con el lenguaje Python **[wxPython98]**. Es una librería de código abierto, al igual que wxWidgets, y es una de las alternativas principales a tkinter. El desarrollo de la librería traducida comenzó en el año 1996, y las primeras versiones de wxPython modernas fueron lanzadas en el año 1998.

Hoy en día, tiene como plataformas soportadas: Windows, MacOS y Linux. Su intención, además de implementar wxWidgets, es hacerlo de una manera que la haga mejor y más rápida. Además, hay un proyecto comenzado en 2010 y llamado Proyecto Phoenix que pretende limpiar la implementación y el código de wxPython y, además, hacerlo compatible directamente con Python.

La versión actual es la 4.x y se lanzó por primera vez en el año 2017.

2.3.3.2.1 Ventajas

- Es una librería muy grande y con muchos componentes.
- Es muy flexible.
- Tiene una gran comunidad y muchos sitios web desde los que formarse en la librería.
- Es de código abierto.

2.3.3.2.2 Inconvenientes

- Requiere ser descargada e instalada, al no ser nativa de Python.
- Su curva de aprendizaje es bastante pronunciada.
- Su documentación es escasa.
- Tiene bastantes bugs y errores.

2.3.3.2.3 Conclusiones

Esta librería, aunque cumple con los requisitos planteados para la aplicación, resulta ser muy compleja. Sobre manera, la falta de documentación y la dificultad de aprendizaje hacen que esta opción sea descartada.

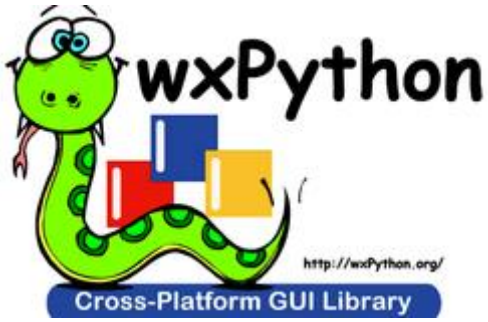


Figura 2.9 Logo de wxPython

2.3.3.3 Kivy

Kivy **[Kivy11]** es una librería de código abierto para el desarrollo de aplicaciones en Python. Al igual que otras alternativas, es una librería multiplataforma, que actualmente soporta Linux, Windows, MacOS, iOS y Raspberry Pi.

Tiene soporte para "multi-touch" UI, es decir, permite desarrollar aplicaciones no solamente para usar con teclado y ratón, sino que también permite que se use una pantalla táctil como modo de interacción.

Su primera versión fue lanzada el 1 de febrero de 2011, y su versión actual es la 2.0.0, lanzada el 10 de diciembre de 2020. Kivy es la evolución de "PyMT", siendo recomendado su uso para nuevos proyectos.

2.3.3.3.1 Ventajas

- Es de código abierto.
- Sus componentes son fáciles de usar.
- Su rendimiento es bastante bueno.

2.3.3.3.2 Inconvenientes

- La interfaz de usuario no es nativa en sus respectivas plataformas.
- No tiene una gran comunidad.
- Su documentación no es demasiado extensa.

2.3.3.3.3 Conclusiones

Kivy es una librería con mucho potencial para aplicaciones que estén pensadas para ordenador y para dispositivos móviles. Este no es el caso con nuestro sistema, con lo que queda descartada.

Además, su aprendizaje es más difícil, al no tener una gran documentación y al ser su comunidad bastante pequeña.

Figura 2.10 Logo de Kivy

2.3.4 Estudio de Alternativas de herramientas para la implementación del Compilador

En este apartado se explicarán las diferentes herramientas y librerías alternativas en las que se podría desarrollar el compilador.

2.3.4.1 PyParsing

Esta librería surge como una alternativa a lex/yacc, ofreciendo al desarrollador una serie de clases que se usan para construir la gramática directamente en código Python **[PyParsing04]**. La técnica usada por esta librería es *Parsing Expression Grammars* (PEGs), lo que facilita la resolución de ambigüedades.

Se trata de una librería de código abierto y tiene una gran documentación, además de ejemplos de cómo se puede usar. La primera versión data del año 2004, y actualmente se encuentra en la versión 3.0.6, lanzada el 12 de noviembre de 2021.

2.3.4.1.1 Ventajas

- Es de código abierto.
- Puede llegar a ser más simple que otras librerías de análisis léxico y sintáctico.

2.3.4.1.2 Inconvenientes

- Su comunidad no es demasiado grande, lo que dificulta la resolución de problemas y el aprendizaje.
- No es muy adecuada a gramáticas complejas.

2.3.4.1.3 Conclusiones

PyParsing es una librería muy interesante al ser PEG, pero como la gramática que se debe usar en este sistema es bastante compleja, no se adecua correctamente a las necesidades del proyecto, por lo que queda descartada.

2.3.4.2 PLY

Sus siglas significan Python Lex-Yacc y, como las herramientas que implementa (Lex y Yacc), sirve para generar compiladores de lenguajes de programación **[PLY01]**. Utiliza LR-parsing, lo que lo hace idóneo para gramáticas grandes, además de proporcionar una comprobación de errores bastante extensiva.

La versión original de PLY fue desarrollada en el año 2001, y estaba pensada para usarse como una introducción a la construcción de compiladores por parte de estudiantes. La actual versión es la 4.0, cuyo principal objetivo ha sido la modernización para las versiones de Python de 3.6 en adelante.

2.3.4.2.1 Ventajas

- Está implementado en Python.
- Es fácil de integrar.
- Es de código abierto.
- Su detección de errores es buena.
- Tiene una documentación muy extensa.

2.3.4.2.2 Inconvenientes

- No soporta EBNF (Notación de Backus-Naur Extendida).
- Sus conflictos de shift-reduce y de ambigüedad son bastante difíciles de resolver.

2.3.4.2.3 Conclusiones

PLY es una gran librería para el desarrollo de compiladores. Sin embargo, el hecho de que no soporte EBNF y sus complicaciones a la hora de desarrollar la gramática hacen que al final esta no sea la alternativa escogida.

2.3.4.3 PyPEG

PyPEG **[PyPEG09]** es un *framework* que permite construir tu propio analizador en Python, soportando además Unicode. Para definir la gramática se emplea la sintaxis PEG. Es bastante verboso y cabe destacar que no produce un árbol, sino que produce una estructura basada en la gramática definida.

Su versión actual es la 2.x y el proyecto lleva desarrollándose desde el año 2009. Es de código abierto y tiene una documentación bastante concisa.

2.3.4.3.1 Ventajas

- Es de código abierto.

2.3.4.3.2 Inconvenientes

- Su comunidad es pequeña.
- Es verboso.
- Es bastante difícil de aprender.

2.3.4.3.3 Conclusiones

PyPEG es un *framework* muy interesante para analizar documentos o texto, sin embargo, no está demasiado orientado al desarrollo de lenguajes de programación. Además, al no tener una gran comunidad ni muchos tutoriales, resulta bastante difícil de aprender.

2.3.5 Estudio de Alternativas de diseño del compilador

En este apartado se van a estudiar las diferentes posibilidades que se han considerado a la hora de desarrollar el compilador del simulador.

2.3.5.1 Compilador a bytecode

Esta opción consiste en compilar el código Arduino a *bytecode* (o algún lenguaje ensamblador inventado por el desarrollador). La implementación del compilador sería igual hasta la fase de generación de código, donde se obtendría, en vez de código Python, código ensamblador o *bytecode*. Esto implicaría también cambiar la forma de ejecutar el código generado.

2.3.5.1.1 Ventajas

- No se depende de que el código Python generado sea correcto.
- La fase de generación es más simple, pues las instrucciones son más concretas y varían menos.
- Los módulos que ejecutan el código no requieren de un módulo que es generado por el programa, evitando importar módulos manualmente.

2.3.5.1.2 Inconvenientes

- Requiere realizar gestión de la memoria propia.
- Implica la creación de un analizador de *bytecode*.
- El juego de instrucciones puede llegar a ser muy complejo.
- Requiere la creación de una máquina virtual.

2.3.5.1.3 Conclusiones

Aunque esta opción nos simplifique el trabajo en las fases de generación y de ejecución, realmente complica mucho más otras fases, como la ejecución de código, teniendo que llegar a realizar un analizador para el *bytecode*. Esta fase no hace falta en el transpilador que se ha escogido finalmente, con lo que la opción escogida finalmente es más simple.

2.3.5.2 Compilador de Arduino

La siguiente opción consiste en emplear el compilador de Arduino normal (el que usa, entre otros, el IDE oficial) para generar el código binario que finalmente sería ejecutado en la placa base Arduino y posteriormente analizarlo para simular su comportamiento en la aplicación que se va a desarrollar.

2.3.5.2.1 Ventajas

- El compilador ya está desarrollado.

- Al emplear el compilador oficial se asegura el hecho de que el código empleado siempre será correcto.
- Se podría reutilizar el IDE de Arduino para realizar la programación y así se simplificaría la aplicación (eliminando el editor).

2.3.5.2.2 Inconvenientes

- Requiere crear un analizador de código ensamblador (de la arquitectura AVR).
- La creación de un analizador requeriría operaciones a nivel de binario, que son más complejas y delicadas.
- La implementación sería muy sensible a cambios:
    - En el compilador de Arduino.
    - En el juego de instrucciones de AVR.

2.3.5.2.3 Conclusiones

Aunque es posible que esta opción sea algo más simple y rápida a nivel de implementación, la realidad es que sería muy sensible a cambios por parte de Arduino y de AVR, con lo que el mantenimiento se volvería mucho más costoso. Por ello, la opción del transpilador vuelve a salir ganadora, ya que es la opción más estable de las cuatro.

2.3.5.3 Compilador externo

La última opción es utilizar un compilador ya desarrollado y adaptarlo al proyecto para realizar la simulación.

2.3.5.3.1 Ventajas

- No haría falta implementar un compilador.
- Es mucho más simple a la hora de compilar el código.

2.3.5.3.2 Inconvenientes

- Surge una dependencia de código a una librería externa, con lo que en caso de cambios en la librería haría que se tuviesen que hacer cambios en el sistema.
- Puede no adaptarse al sistema correctamente.
- No hay librerías de este tipo disponibles para Python (que se haya encontrado).

2.3.5.3.3 Conclusiones

Aunque esta opción nos ahorraría el hecho de hacer un compilador, la dependencia externa de código hace que la aplicación sea menos estable que con la opción del transpilador. Además, el hecho de que no haya opciones realmente viables de librerías de este tipo para Python hace que esta opción sea descartada.

Capítulo 3. Aspectos Teóricos

En este capítulo se explicarán los diferentes aspectos teóricos que han influido significativamente al sistema o se han usado durante el desarrollo de este.

3.1 Métrica 3

Métrica 3 **[Metrica01]** es una metodología de planificación, desarrollo y mantenimiento de sistemas de información. Pretende sistematizar las actividades de todo el ciclo de vida del software. Parte de la anterior versión de Métrica (la 2.1), mejorando en general la metodología proporcionada por esta versión.

Su uso es libre, a cambio de que se haga constar en la obra su autoría original (el ministerio de Administraciones Públicas y el ministerio de Hacienda). Los principales objetivos de esta metodología son:

- Proporcionar o definir sistemas de información que ayuden a alcanzar los objetivos de la organización.
- Dotar a la organización de productos software que satisfagan las necesidades de los usuarios.
- Mejorar la productividad de los departamentos de Sistemas y TIC (Tecnologías de la Información y las Comunicaciones).
- Facilitar la comunicación entre los participantes.
- Facilitar la operación, mantenimiento y uso del software obtenido.

Este documento está desarrollado empleando esta metodología, pero de forma adaptada, contemplando solo aquellos apartados que se han considerado más adecuados para un proyecto de fin de carrera.

3.2 Scrum

Scrum **[Scrum86]** es un marco de trabajo (o metodología, aunque llamarla así es técnicamente incorrecto) ágil, de las más famosas y usadas del mundo (más de 12 millones de personas la usan todos los días), a través del cual las personas implicadas en un proyecto pueden abordar problemas complejos. Antes de explicar cómo funciona Scrum, se deben explicar una serie de conceptos:

- *Backlog* del producto, es una lista priorizada y ordenada de las funcionalidades y mejoras que se deben desarrollar sobre el producto.
- Sprint, es el periodo durante el cual se va a realizar el trabajo, siendo recomendable que dicha duración sea constante de un sprint a otro.
- Sprint *backlog*, es la lista de las funcionalidades que se desarrollarán durante el sprint.

Scrum está compuesto por una serie de actores, que son los siguientes:

- *Product owner* o dueño del producto, que representa al cliente y *stakeholders* del proyecto. Es el que proporciona el *backlog* del producto.
- Scrum máster, es el líder del equipo, y ayuda a todos los actores a entender cómo funciona scrum, sus normas y sus prácticas.
- Equipo o desarrolladores, que son los que desarrollarán el producto (aunque por lo general, el scrum máster también puede estar involucrado en el desarrollo).

El proceso seguido en Scrum es el siguiente:

1. Se obtiene un *backlog* del producto, que es una lista ordenada de las funcionalidades y mejoras a desarrollar por el equipo.
2. Los ítems del *backlog* son elegibles en la planificación del sprint, que será realizada por todos los miembros del equipo al comienzo del sprint, obteniendo así el sprint *backlog*, que será el conjunto de tareas que se realizarán a lo largo del sprint.
3. Todos los días suceden una serie de eventos:
    a. El scrum diario, que tiene como objetivo comentar las novedades en el desarrollo del anterior, así como comentar qué se va a hacer durante el día. Estas reuniones tienen una serie de características: solo pueden durar 15 minutos, y requiere que participen en ella los desarrolladores, siendo opcional la presencia del *scrum máster* y el *product owner*.
    b. El equipo, después de la reunión, realizará las tareas que tenía previstas y que ha comentado durante la reunión.
4. Al final del sprint se realiza una reunión en la que están involucrados todos los actores para analizar el resultado del sprint. La reunión se llama sprint *review*.
5. Después de esa revisión, el equipo se reúne para realizar una retrospectiva del sprint, analizando cómo han trabajado y planificando como mejorar ese trabajo.

Este marco de trabajo pretende ser más flexible que metodologías tradicionales y evitar los problemas derivados de estas, como y los retrasos o los sobrecostes de los proyectos.

3.3 Compilador

Un compilador es un programa que transforma el código fuente de un lenguaje de programación a código máquina o a código de otro lenguaje de programación (lo que lo convertiría en un transpilador) **[Sheldon22]**. El compilador lee los archivos de código, lo analiza y lo traduce al lenguaje deseado.

Los compiladores pueden traducir a un lenguaje máquina optimizado para el sistema operativo que se esté usando o incluso traducir a *bytecode* en vez de código máquina, que será ejecutado dentro de una máquina virtual (es el caso de Java, por ejemplo).

En todo caso, el compilador debe asegurarse que el lenguaje de salida (sea código máquina, bytecode u otro lenguaje de programación) sea correcta. Para ello, los compiladores funcionan de la siguiente manera:

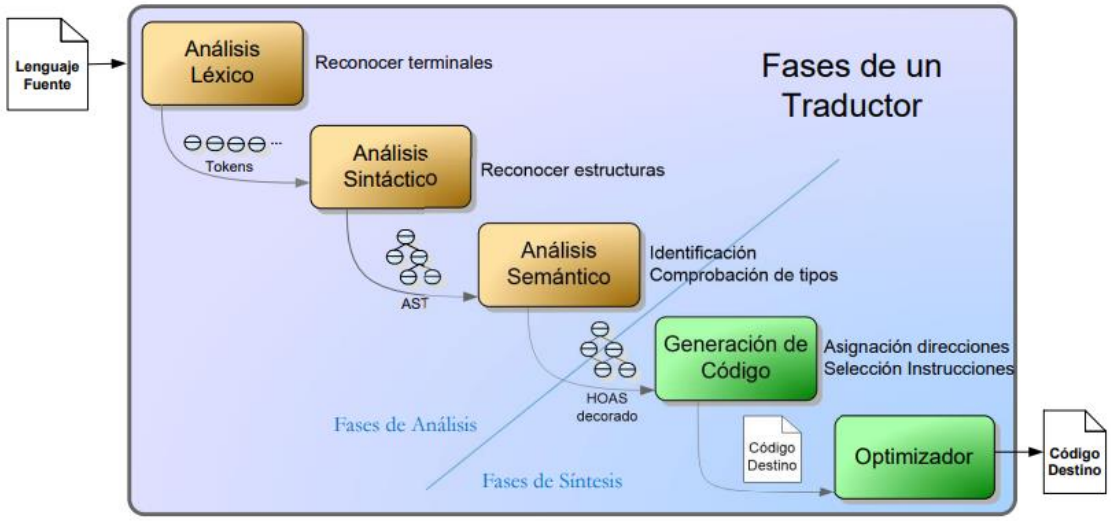


Figura 3.1 Fases de un compilador [Izquierdo20]

- Análisis léxico: En esta fase se divide el código en lexemas (agrupaciones de caracteres que conforman las sentencias del lenguaje), que luego son convertidos en *tokens* (conjunto de lexemas que pueden ser tratados como uno) con vistas a las dos siguientes fases de análisis.
- Análisis sintáctico: Se verifica que la sintaxis del código introducido es correcta y válida, basada en las reglas del lenguaje (también se le llama a este proceso *parsing*). Es en esta fase en la que se crea el Árbol de Sintaxis Abstracta (AST por sus siglas en inglés), que representa la estructura lógica de los elementos específicos del código.
- Análisis semántico: Se verifica la validez de la lógica del código. Algunas comprobaciones de esta fase son tipos, declaración de variables, ámbito de variables (si son declaradas para una función o para el programa entero), etc.
- Generación de código: Si el código pasa correctamente las fases de análisis anteriores, entonces se generará el código de salida, bien sea otro lenguaje de programación, *bytecode*, código ensamblador u otros.
- Optimizador: Optimiza el código generado por la anterior fase. Es usado en la mayoría de los lenguajes potentes (C o C++ por nombrar algunos), para mejorar el rendimiento final de los programas.

3.4 Gramática Libre de Contexto (GLC)

Una gramática libre de contexto (GLC de aquí en adelante) **[Wikipedia01]** es una gramática formal que, como tal, puede ser definida mediante la siguiente 4-tupla:

$$G = (V_t, V_n, P, S)$$

Dónde:

- V_t es un conjunto finito de símbolos terminales.
- V_n es un conjunto finito de símbolos no terminales.

- P es un confunto finito de producciones o reglas de producción.
- S es el símbolo inicial y forma parte de los no terminales.

Los elementos de P se definen de la siguiente forma:

$$V_n \rightarrow (V_t \cup V_n)^*$$

Lo que quiere decir que los símbolos no terminales se definen mediante la unión de cero o más símbolos no terminales y símbolos terminales, o bien símbolos no terminales y terminales de forma aislada.

Para describir GLCs, la notación más usada es la forma Backus-Naur (BNF de aquí en adelante), y será la usada para describir la gramática empleada por el compilador definido en este proyecto.

Existen dos formas principales de derivar la cadena inicial dentro de una GLC:

- Derivación por la izquierda (LL): Se reemplaza primero el no terminal que esté más hacia la izquierda, es la forma más simple de derivar. Esta derivación será la usada por nuestra gramática.
- Derivación por la derecha (LR): Es lo opuesto a la derivación por la izquierda, se empieza siempre por el no terminal de la derecha.

3.5 Arduino

Arduino **[Arduino05]** es una compañía y comunidad que se dedica al diseño, manufactura y soporte de dispositivos electrónicos y de software para controlarlos. Su principal objetivo es permitir a todo tipo de personas (desde profesionales hasta aficionados) acceder y utilizar de una forma sencilla dispositivos electrónicos y tecnologías digitales.

Arduino es usado en diferentes ámbitos, como el profesional, el estudiantil o en casa; bien sea para proyectos grandes o pequeños. En Arduino trabajan junto con empresas muy importantes en el mundo tecnológico, como Bosch, Google, Intel o Raspberry Pi.

Arduino como placa base es una placa que emplea los microcontroladores AVR y que, mediante el uso de pines, permite transmitir y recibir información o señales eléctricas a componentes electrónicos, tales como leds, servomotores o botones. En general, las placas permiten conectarse al ordenador a través de USB, y es a través de este medio por donde se suben los programas (o sketches) a la placa, para ser ejecutados.

Figura 3.2 Logo de Arduino

3.6 UML

Universal Modeling Language (Lenguaje Unificado de Modelado o UML) **[UML04]** es un lenguaje empleado para el modelado (o diseño) de los sistemas de software. El realizar modelos o diagramas antes de la fase de desarrollo de los sistemas ayuda a que se reduzcan las posibilidades de que un proyecto falle a la hora de cumplir las expectativas iniciales.

Hay dos versiones diferentes de UML, la 1.0 y la 2.0, que es una revisión de la versión 1.0 y sus actualizaciones. UML define un total de trece tipos de diagramas divididos en tres categorías:

- Diagramas de estructura, que incluye el diagrama de clases, de objetos, de componentes, de estructura compuesta, de paquetes y de despliegue.
- Diagramas de comportamiento, que incluye el diagrama de casos de uso, de actividades y de máquina de estado.
- Diagramas de interacción, que incluye el diagrama de secuencia, de comunicación, de tiempos y de interacción.

Figura 3.3 Logo de UML

3.7 Asignatura de Software para Robots

El simulador se va a usar en el contexto de la asignatura de software para robots, en concreto en dos partes de la asignatura: la práctica 4 de la asignatura, que corresponde con los actuadores lineales y su programación; y las prácticas 7, 8 y 9.1, que corresponden a los robots móviles y el seguimiento de caminos **[González21]**.

En el caso del actuador lineal, se ha implementado de forma que se puedan realizar todas las actividades, tanto obligatorias (mover el actuador con el joystick, limitar el movimiento y realizar el movimiento automático) como opcionales (sistema de coordenadas).

El caso de los robots móviles es el mismo. Para el robot de dos sensores de luz se pueden hacer todas las actividades (seguir líneas, encontrar el circuito y esquivar obstáculos). En el caso del de cuatro sensores, se puede implementar la escapada y resolución de laberintos

siguiendo todos los métodos propuestos en las prácticas (mano derecha o izquierda, Tremaux o libre), así como el concurso de resolución de laberintos planteado en la práctica 9.1.

Capítulo 4. Planificación del Proyecto y Presupuesto Iniciales

En este capítulo se va a desglosar tanto la planificación inicial como el presupuesto inicial; ambos estando sujetos a cambios durante el desarrollo del proyecto, que serán explicados en el Capítulo 12.

4.1 Planificación Inicial

En esta sección se define la planificación seguida a lo largo de la ejecución del proyecto, dividiéndose en los siguientes bloques:

1. Fase inicial, donde se realizará la reunión inicial, así como las diferentes actividades previas al desarrollo como tal del sistema: presupuesto y planificación iniciales, además de un análisis inicial de lo que se va a realizar en el proyecto.
2. Análisis del sistema. En esta fase se define el sistema y los requisitos de deberá cumplir. Para ello, es necesario identificar los subsistemas que lo componen, así como las clases a través de un diagrama; casos de uso y escenarios.
3. Diseño del sistema, donde se establecerá la arquitectura y las clases del sistema de manera definitiva. En este bloque también se realizarán los diferentes diagramas necesarios para establecer el diseño del sistema, además de establecer el diseño de la interfaz y del plan de pruebas a realizar en el futuro.
4. Implementación del sistema. El desarrollo del sistema, siguiendo lo establecido en las anteriores fases.
5. Pruebas del sistema. Mientras se desarrolla el sistema se van a comenzar a realizar las pruebas que no requieran que el sistema esté finalizado. En caso contrario, se esperará a la finalización de la implementación.
6. Documentación, incluye la documentación de la memoria, los diferentes manuales del sistema y la presentación final del proyecto.

En la siguiente tabla se muestran los diferentes bloques divididos en las diferentes tareas que los componen, después de la tabla se expone el diagrama de Gantt del proyecto:

Nombre de tarea	Duración	Comienzo	Fin
Fase inicial	**3,13 días**	**lun 11/10/21**	**jue 14/10/21**
Reunión inicial	1 hora	lun 11/10/21	lun 11/10/21
Análisis inicial del proyecto	1 día	lun 11/10/21	mar 12/10/21
Planificación inicial del proyecto	2 días	mar 12/10/21	jue 14/10/21
Presupuesto inicial	2 días	mar 12/10/21	jue 14/10/21
Análisis	**9 días**	**jue 14/10/21**	**mié 27/10/21**
Definición del sistema	1 día	jue 14/10/21	vie 15/10/21
Requisitos del sistema	3 días	vie 15/10/21	mié 20/10/21
Identificación de los subsistemas	2 días	mié 20/10/21	vie 22/10/21

Diagrama de clases preliminar	1 día	vie 22/10/21	lun 25/10/21
Análisis de casos de uso y escenarios	2 días	lun 25/10/21	mié 27/10/21
Diseño del sistema	**15 días**	**mié 27/10/21**	**mié 17/11/21**
Arquitectura del sistema	3 días	mié 27/10/21	lun 01/11/21
Diseño de clases	3 días	lun 01/11/21	jue 04/11/21
Realización de diagramas	**5 días**	**jue 04/11/21**	**jue 11/11/21**
Diagrama de estados	1 día	jue 04/11/21	vie 05/11/21
Diagrama de Interacción	2 días	vie 05/11/21	mar 09/11/21
Diagrama de Actividades	2 días	mar 09/11/21	jue 11/11/21
Diseño de la interfaz	1 día	jue 11/11/21	vie 12/11/21
Especificación técnica del plan de pruebas	3 días	vie 12/11/21	mié 17/11/21
Implementación del sistema	**25 días**	**mié 17/11/21**	**mié 22/12/21**
Decidir tecnologías y estándares	**7 días**	**mié 17/11/21**	**vie 26/11/21**
Escoger lenguaje de programación	2 días	mié 17/11/21	vie 19/11/21
Escoger entorno de desarrollo	1 día	vie 19/11/21	lun 22/11/21
Escoger librerías	2 días	lun 22/11/21	mié 24/11/21
Investigar estándares y normas a seguir	2 días	mié 24/11/21	vie 26/11/21
Módulo de Interfaz gráfica de Usuario	2 días	vie 26/11/21	mar 30/11/21
Módulo de controlador	5 días	mar 30/11/21	mar 07/12/21
Módulo de Actuador Lineal	**5 días**	**mar 07/12/21**	**mar 14/12/21**
Desarrollo del compilador	3 días	mar 07/12/21	vie 10/12/21
Desarrollo de la representación gráfica	2 días	vie 10/12/21	mar 14/12/21
Módulo de Robot Móvil	**6 días**	**mar 14/12/21**	**mié 22/12/21**
Desarrollo del compilador	4 días	mar 14/12/21	lun 20/12/21
Desarrollo de la representación gráfica	2 días	lun 20/12/21	mié 22/12/21
Pruebas del sistema	**9 días**	**mié 17/11/21**	**mar 30/11/21**
Pruebas unitarias	2 días	mié 17/11/21	vie 19/11/21
Pruebas de integración y del sistema	3 días	vie 19/11/21	mié 24/11/21
Pruebas de usabilidad y accesibilidad	2 días	mié 24/11/21	vie 26/11/21
Pruebas de rendimiento	2 días	vie 26/11/21	mar 30/11/21
Documentación	**60 días**	**jue 14/10/21**	**jue 06/01/22**
Memoria del proyecto	**36 días**	**jue 14/10/21**	**vie 03/12/21**
Documentar desarrollo	25 días	jue 14/10/21	jue 18/11/21
Conclusiones	1 día	jue 18/11/21	vie 19/11/21
Ampliaciones	1 día	vie 19/11/21	lun 22/11/21
Planificación final	5 días	lun 22/11/21	lun 29/11/21
Presupuesto final	4 días	lun 29/11/21	vie 03/12/21
Manuales del sistema	**4 días**	**vie 03/12/21**	**jue 09/12/21**
Manual de usuario	1 día	vie 03/12/21	lun 06/12/21
Manual de instalación	1 día	lun 06/12/21	mar 07/12/21
Manual de ejecución	1 día	mar 07/12/21	mié 08/12/21
Manual del programador	1 día	mié 08/12/21	jue 09/12/21
Revisión final del proyecto	7 días	mié 22/12/21	vie 31/12/21
Presentación	**4 días**	**vie 31/12/21**	**jue 06/01/22**
Preparación	**3 días**	**vie 31/12/21**	**mié 05/01/22**
Preparación del contenido	2 días	vie 31/12/21	mar 04/01/22
Realización de diapositivas	1 día	mar 04/01/22	mié 05/01/22
Exposición	1 día	mié 05/01/22	jue 06/01/22

Tabla 4.1 Planificación inicial

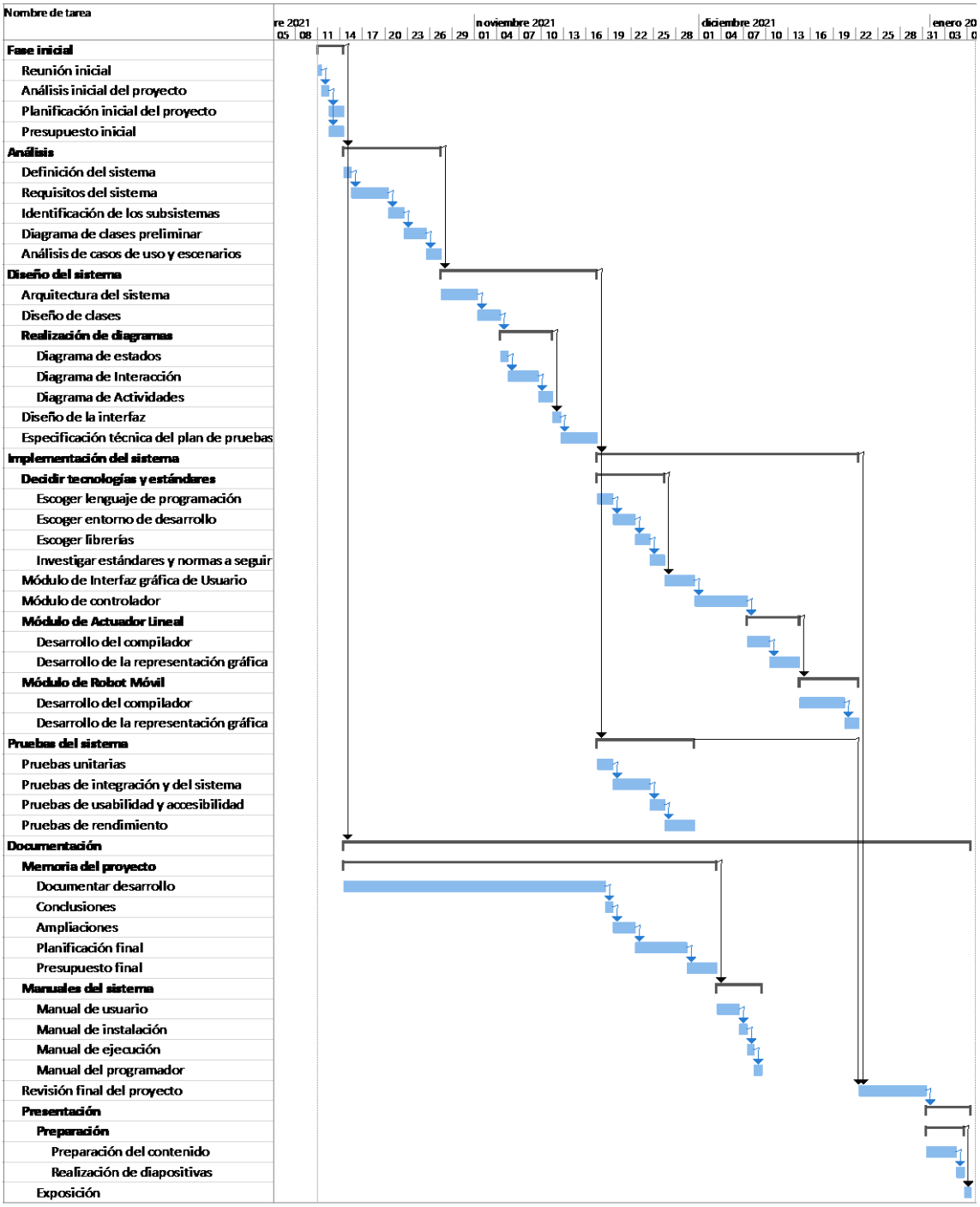


Figura 4.1 Diagrama de Gantt (con planificación)

4.2 Presupuesto Inicial

En este apartado se expone el presupuesto del proyecto, que está basado en la planificación realizada en el apartado 4.1. El apartado se divide en dos apartados, el primero del presupuesto detallado, que corresponde a la empresa; y el segundo del presupuesto simplificado, que corresponde al cliente.

Se ha simulado el rol de una empresa real, teniendo en cuenta empleados en los puestos necesarios y sus sueldos, tratando de aproximarse lo máximo posible a su sueldo real. En este caso se ha seguido el Convenio colectivo estatal de empresas de consultoría y estudios de mercado y de la opinión pública **[BOE18]**.

4.2.1 Desarrollo de Presupuesto Detallado (Empresa)

En este apartado se expone el presupuesto detallado, es decir, el de la empresa.

4.2.1.1 Costes directos

Los costes por trabajador se han obtenido del BOE del día 6 de marzo de 2018 **[BOE18]**.

4.2.1.1.1 Costes del desarrollo y del personal

Para establecer el coste del personal para este proyecto se ha decidido calcularlo de dos formas. La primera es la representada en la siguiente tabla, que desglosa el coste del personal según el bloque o fase del proyecto:

Concepto	Horas	Total
Fase inicial	25	962,69 €
Análisis	72	3.753,78 €
Diseño	120	3.091,08 €
Implementación	200	2.386,08 €
Pruebas	72	779,40 €
Documentación	480	5.375,55 €
Formación	16	479,55 €
	Total	16.828,14 €

Tabla 4.2 Costes de desarrollo

La segunda forma es según el número de horas trabajadas en el proyecto por parte del trabajador. Como se puede observar, el coste total es el mismo:

Concepto	Cantidad	Coste por hora	Horas trabajadas	Total
Jefe de Proyecto	1	18,04 €	185	3.337,71 €
Experto en Ciberseguridad	1	17,50 €	144	2.520,06 €
Ingeniero de Software	1	16,59 €	251	4.165,05 €
Desarrollador	1	11,93 €	457	5.452,20 €
Tester	1	10,83 €	125	1.353,13 €
			Total	16.828,14 €

Tabla 4.3 Costes del personal

4.2.1.1.2 Costes de licencias

En el caso de las licencias necesarias para el proyecto, podemos distinguir dos casos:

- Las licencias que se pagan solo una vez, en este caso solo Windows 10. Su precio por unidad se ha calculado obteniendo la amortización de la licencia. Para realizar dicho cálculo se divide el número de meses que se va a usar en el proyecto, en este caso 4; entre la vida útil que se le supone, en este caso 24 meses. El resultado final es el porcentaje amortizado (un 16,67%).
- Las licencias mensuales, que son el resto de las licencias. En este caso, se cobra directamente por cada mes usado, teniendo en cuenta el número de unidades.

Concepto	Cantidad	Uso (meses)	Coste total (unidad)	Total
Microsoft Office 365	5	4	10,50 €	210,00 €
Windows 10	5	4	24,17 €	120,83 €
GitHub	5	4	4,00 €	80,00 €
Microsoft Project	5	4	8,40 €	168,00 €
			Total	578,83 €

Tabla 4.4 Costes de licencias

4.2.1.1.3 Costes del material

Para calcular el coste del material se ha calculado la amortización de cada producto. Para ello, se divide el número de meses que se va a usar en el proyecto entre los meses de vida útil total correspondientes a cada material. El porcentaje obtenido se aplica sobre el precio unitario total y con ello se obtiene el coste que tiene cada unidad en el proyecto.

Finalmente, para obtener el total, se multiplica ese precio unitario en el proyecto por la cantidad de elementos de ese material que serán necesarios.

Concepto	Cantidad	Uso (meses)	Vida útil (meses)	Precio Unitario	Amortización	Total
Ordenador	5	4	72	859,64 €	5,56%	238,79 €
Monitor	5	4	120	199,00 €	3,33%	33,17 €
Escritorio	5	4	180	219,58 €	2,22%	24,40 €
Silla	5	4	60	74,84 €	6,67%	24,95 €
Teclado	5	4	36	52,85 €	11,11%	29,36 €
Ratón	5	4	36	23,52 €	11,11%	13,07 €
Placa Arduino	1	2	2	20,99 €	100,00%	20,99 €
Robot Móvil	1	2	2	59,00 €	100,00%	59,00 €
Actuador Lineal	1	2	2	49,36 €	100,00%	49,36 €
					Total	493,08 €

Tabla 4.5 Costes del material

4.2.1.2 Costes indirectos

En este apartado se desglosan los gastos que no tienen una relación directa con el desarrollo del sistema, es decir, que no son generados directamente por el desarrollo del producto.

Concepto	Coste/mes	Meses	Total
Electricidad	208,32 €	4	833,28 €
Dietas	55,24 €	4	220,96 €
Internet	38,00 €	4	152,00 €
Alquiler del local	620,00 €	4	2.480,00 €
Agua	15,64 €	4	62,56 €
Material de oficina	79,27 €	1	79,27 €
Limpieza	859,61 €	4	3.438,44 €
		Total	7.266,51 €

Tabla 4.6 Costes indirectos

4.2.1.3 Resumen

Una vez desglosados los gastos, procedemos a calcular el coste total del proyecto, que se muestra en la siguiente tabla de forma resumida:

Concepto	Total
Material	493,08 €
Licencias	578,83 €
Desarrollo del proyecto	16.828,14 €
Suma de Costes directos	17.900,05 €
Costes indirectos	7.266,51 €
Suma de costes	25.166,56 €
Beneficios (20%)	5.033,31 €
Subtotal	30.199,88 €
IVA (21%)	6.341,97 €
Total	36.541,85 €

Tabla 4.7 Resumen de costes

Como se puede observar, se suman costes directos e indirectos y se les aplica el beneficio deseado en el proyecto (en este caso es un 20%). Todo esto resulta en el subtotal, al cual se le aplica el IVA y, con ello, se obtiene el coste total del proyecto.

4.2.2 Desarrollo de Presupuesto Simplificado (Cliente)

A continuación, se muestra el presupuesto del cliente. Se trata de un desglose simplificado de los costes que hemos indicado en el apartado 4.2.1. Los costes indirectos se han distribuido entre las diferentes etapas del desarrollo del proyecto, de ahí la diferencia en los costes. El coste de material y licencias es el mismo que en el presupuesto detallado.

El subtotal se obtiene sumando todos los costes de las filas anteriores. El IVA se aplica al subtotal y, una vez obtenido, se suma al subtotal. El resultado es el coste total del proyecto.

Concepto	Total
Desarrollo del sistema	19.758,62 €
Documentación	7.132,67 €
Formación	2.236,67 €
Material	493,08 €
Licencias	578,83 €
Total	30.199,88 €
IVA (21%)	6.341,97 €
Total + IVA	36.541,85 €

Tabla 4.8 Presupuesto Simplificado

Capítulo 5. Análisis

Este apartado contendrá toda la especificación de requisitos y toda la documentación del análisis de la aplicación, a partir de la cual se elaborará posteriormente el diseño.

5.1 Definición del Sistema

En este subapartado se determinará el alcance final del sistema, es decir, se establecerán los límites de este; siempre teniendo en cuenta sus características.

5.1.1 Determinación del Alcance del Sistema

En este proyecto se desarrollará una aplicación que permita realizar y probar los ejercicios que dependen de un actuador lineal y de un robot móvil a los alumnos de la asignatura de Software para Robots. Para ello, se va a construir un compilador que permita al usuario escribir en el lenguaje Arduino el código que se necesite para hacer funcionar a los robots, además de una interfaz gráfica que permita escribir el código, una representación gráfica de los robots y una consola que permita al usuario comprobar la salida que proporcione el código que haya escrito.

Dicho compilador actuará sobre el código escrito en un editor de texto, que mostrará al usuario las líneas en las que se encuentra cada sentencia, además de permitir realizar *scroll* vertical y lateral, y las acciones de hacer y deshacer.

Cada robot tendrá una representación gráfica, en la cual se mostrará las acciones que realice cada uno en función del código escrito. El compilador, en vez de generar código ensamblador, deberá generar código del lenguaje empleado para el desarrollo de este proyecto. Dicho código se ejecutará en el momento en el que se decida hacer clic en el botón de ejecutar, y se detendrán en el momento en el que se haga clic en el botón parar.

Además, se desarrollará una salida por consola, que permita al usuario conocer la existencia de errores o imprimir diferentes datos que sean considerados necesarios a través de las funcionalidades proporcionadas por el lenguaje de programación.

El sistema será accesible y ejecutable desde Windows.

Se proporcionará la posibilidad de guardar el código en archivos con la misma extensión que aquellos que funcionan en el entorno de desarrollo de Arduino (.ino), además de permitir cargar datos desde ese tipo de ficheros.

5.2 Requisitos del Sistema

En este subapartado se obtendrán los requisitos del sistema, además de identificar a los diferentes actores que interactuarán con él y los diferentes casos de uso de la aplicación.

5.2.1 Obtención de los Requisitos del Sistema

En la siguiente tabla se muestran los requisitos del sistema a desarrollar, tanto funcionales como no funcionales

5.2.2 Requisitos de Interfaces Externas

Código	Nombre Requisito	Descripción del Requisito
RIE1	Interfaces de Software	
RIE1.1	Disponibilidad en Plataformas	El sistema deberá estar disponible en Windows
RIE2	Interfaces de Usuario	
RIE2.1	Estándares de Interfaz de Usuario	El sistema deberá seguir los estándares propios de cada uno de los sistemas operativos mencionados en RIE1.1

Tabla 5.1 Requisitos de Interfaces Externas

5.2.2.1 Requisitos Funcionales

Código	Nombre Requisito	Descripción del Requisito
R.1	Edición de Código	
R1.1	Programación en Arduino	El sistema permitirá al usuario programar en el lenguaje de programación de Arduino.
R1.2	Ejecución del Código	El sistema permitirá la ejecución del código escrito.
R1.3	Cargar archivos	El sistema permitirá cargar el código desde archivos.
R1.4	Guardar archivos	El sistema permitirá guardar el código escrito en un archivo.
R.2	Representación gráfica	
R2.1	Representación gráfica del código	El sistema ejecutará el código escrito en el editor mediante la representación gráfica de los movimientos que origina.
R2.2	Representación de los Robots	El sistema representará gráficamente los robots.
R2.2.1	Representación gráfica del Robot Móvil	El sistema representará gráficamente el robot móvil.

R2.2.1.1		La representación gráfica deberá mostrar correctamente los movimientos del robot.
R2.2.1.2		La representación gráfica deberá mostrar los inputs de los siguientes sensores:
R2.2.1.2.1		Sensor de ultrasonidos.
R2.2.1.2.2		Sensores de infrarrojos para la detección de la línea.
R2.2.1.3		La representación gráfica deberá mostrar en qué dirección se mueven los servomotores.
R2.2.1.4		La representación gráfica mostrará las interacciones del robot móvil con su entorno, que podrán ser las siguientes:
R2.2.1.4.1		Seguir el camino, que será de color negro.
R2.2.1.4.2		Salirse del camino. El exterior del camino estará será de color blanco.
R2.2.1.4.3		Esquivar obstáculos.
R2.2.1.5		La representación gráfica también deberá mostrar un laberinto, cuyo camino esté representado en los mismos colores que los mencionados en R2.2.1.4.1 y R2.2.1.4.2.
R2.2.2	Representación gráfica del Actuador Lineal	El sistema representará gráficamente el actuador lineal.
R2.2.2.1		El sistema mostrará correctamente los movimientos del actuador lineal.
R2.2.2.2		El sistema mostrará la interacción de la pieza móvil del actuador lineal con los botones situados en los extremos del robot.
R2.2.2.3		El sistema deberá mostrar en qué dirección se mueven los servomotores.
R3	Salida por consola	
R3.1	Reporte de errores	El sistema deberá mostrar por consola los errores que tenga el código
R3.1.1	Tipos de errores	Los errores de código podrán ser de los siguientes tipos:
R3.1.1.1		Errores léxicos (errores en la escritura de las palabras del código).
R3.1.1.2		Errores sintácticos (errores en la estructura del código).

R3.2	Salida por consola	El sistema deberá mostrar por consola aquellos parámetros que se especifiquen en el código escrito en el editor.
R4	Log de errores	
R4.1	Anotación de errores	El sistema notificará a través de un archivo los errores que vaya cometiendo el alumno.
R4.1.1	Tipos de errores	En el archivo se incluirán todos los errores que se vayan cometiendo, siendo los errores del mismo tipo que los indicados en R3.1.1.

Tabla 5.2 Requisitos funcionales

5.2.2.2 Requisitos de desarrollo

Código	Nombre Requisito	Descripción del Requisito
RD1	Restricciones de estándares	
RD1.1	Metodología	El sistema deberá ser desarrollado en la metodología Scrum.
RD2	Restricciones de idiomas	
RD2.1	Idiomas disponibles	El sistema deberá estar disponible en el lenguaje español.
RD2.2	Internacionalización	El sistema deberá facilitar la futura internacionalización de la aplicación.

Tabla 5.3 Requisitos de desarrollo

5.2.2.3 Requisitos no Funcionales

Código	Nombre Requisito	Descripción del Requisito
RU1	Requisitos de usuario	
RU1.1	Conocimiento de Arduino	El usuario deberá conocer el lenguaje a Arduino.
RU1.2	Conocimiento de los robots	El usuario deberá conocer en que puertos de las placas se conectan los diferentes componentes de ambos robots.
RT1	Requisitos Tecnológicos	
RT1.1	Sistemas Operativos compatibles	El sistema deberá funcionar en los siguientes sistemas operativos (especificando lo establecido en RIE1.1):
RT1.1.1		Windows 10
RT1.1.2		Windows 11
RT1.2	Ejecución sencilla	El sistema deberá ser fácil de ejecutar en todas las plataformas mencionadas.

Tabla 5.4 Requisitos no funcionales

5.2.3 Identificación de Actores del Sistema

Los actores principales del sistema son los siguientes:

- Alumno/Usuario: son los actores principales del sistema. Serán quienes interactúen con el programa, desarrollando código y comprobando que este funciona correctamente; tanto por los resultados que puedan salir por consola como por la representación animada de lo que harían los robots con el código ejecutado.
- Profesor: será un actor que usará menos el sistema. Principalmente lo podrá usar para ejemplificar la tarea que los alumnos deben hacer. También podrá usar el programa para resolver dudas puntuales que les puedan surgir a los alumnos

5.2.4 Especificación de Casos de Uso

5.2.4.1 Diagrama de casos de uso

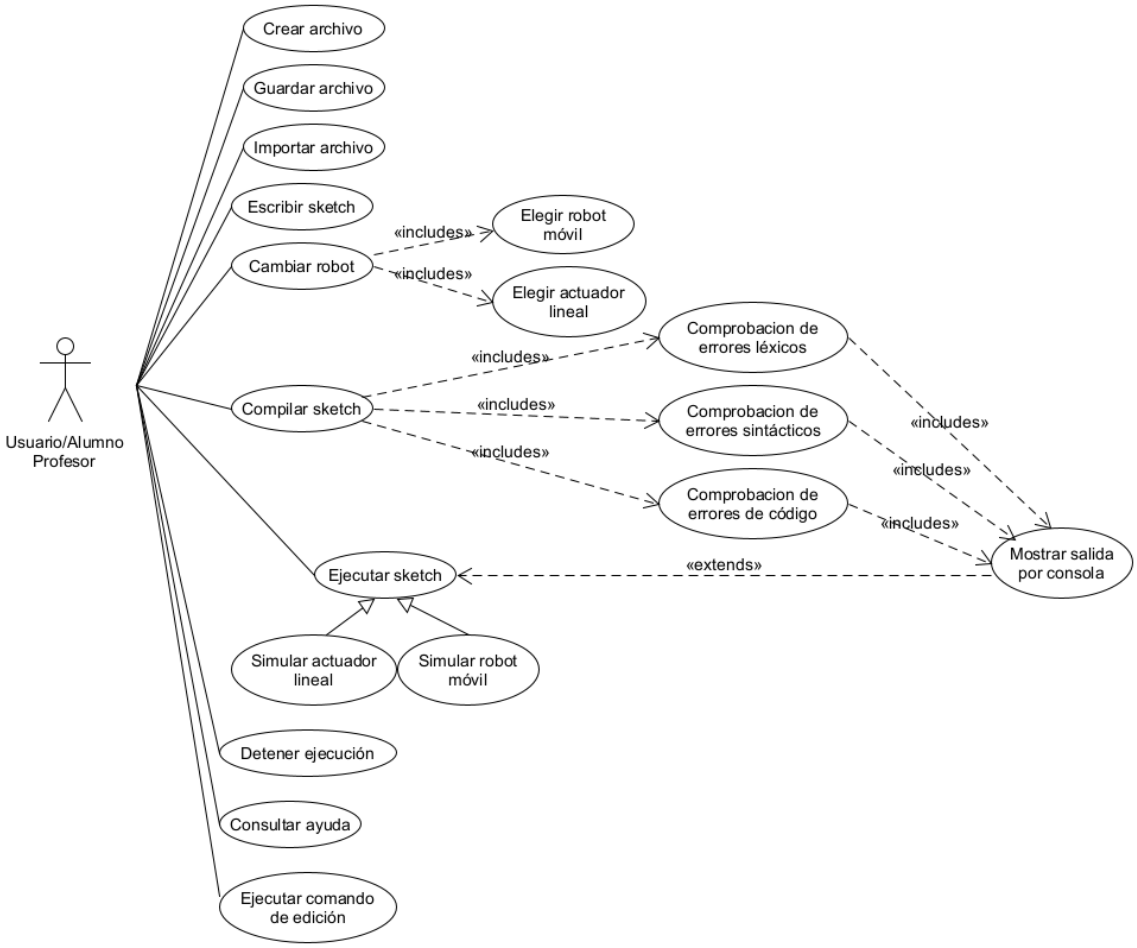


Figura 5.1 Diagrama de casos de uso del sistema

5.2.4.2 Descripción de los casos de uso

Nombre del Caso de Uso
Crear archivo
Descripción
El usuario podrá crear un nuevo archivo. Una vez creado, se mostrará en el editor de código, únicamente incluyendo los métodos “setup” y “loop”. Se hace por defecto al abrir el simulador.

Nombre del Caso de Uso
Guardar archivo
Descripción
El usuario podrá guardar el sketch en cualquier momento. Si el archivo es importado, se podrá guardar sin especificar un nuevo archivo. En todo caso, se podrá guardar el código a uno nuevo, que tendrá la extensión “.ino”.

Nombre del Caso de Uso
Importar archivo
Descripción
El usuario podrá importar un sketch ya existente. Dicho sketch deberá ser un archivo con la extensión “.ino”. El contenido del sketch será mostrado en el editor de código.

Nombre del Caso de Uso
Escribir sketch
Descripción
El usuario podrá en todo momento escribir código en la parte del simulador reservada para ello (el editor de código). Dicho código podrá guardarse en un archivo en todo momento, según lo explicado en el caso de guardar archivo.

Nombre del Caso de Uso
Cambiar robot
Descripción
En el momento de simular el código, el usuario podrá elegir que robot simular. Podrá escoger entre dos robots: • El robot móvil • El actuador lineal En todo caso, deberá escoger uno de los dos

Nombre del Caso de Uso
Elegir robot móvil
Descripción
Este es uno de los robots que se podrá elegir para simular. En este caso, se mostrará un robot móvil con todos aquellos obstáculos u objetos necesarios para realizar las pruebas pertinentes.

Nombre del Caso de Uso
Elegir actuador lineal
Descripción
Este es el otro robot que se podrá escoger. En este caso, solo es necesario mostrar el propio actuador lineal, ya que todas las actividades a realizar lo involucran solamente a él.

Nombre del Caso de Uso
Compilar sketch
Descripción
El usuario podrá compilar el sketch. Este proceso mostrará todos los errores cometidos a la hora de escribir el código. Cabe destacar la diferencia entre compilar y ejecutar, aunque si ejecutamos también se realizará la compilación antes de ejecutar como tal.

Nombre del Caso de Uso
Comprobación de errores léxicos
Descripción
En la fase de compilación se comprobarán todos los errores léxicos que pueda tener el sketch para ser reportados al usuario. Tendrán lugar cuando una cadena no encaje en ninguno de los patrones del lenguaje.

Nombre del Caso de Uso
Comprobación de errores sintácticos
Descripción
En la fase de compilación se comprobarán todos los errores sintácticos que pueda tener el sketch para ser reportados al usuario. Estos errores suceden cuando el programa no cumple las reglas del lenguaje.

Nombre del Caso de Uso
Comprobación de errores de código
Descripción
Esta será la última fase de compilación. Reportará al usuario errores varios, principalmente relacionados con si se están programando los robots correctamente o no.

Nombre del Caso de Uso
Ejecutar sketch
Descripción
El usuario podrá ejecutar el sketch en cualquier momento. Para ello, el código deberá compilar. La ejecución implicará que se simulará el comportamiento del robot elegido.

Nombre del Caso de Uso
Simular actuador lineal
Descripción
Esta es una de las dos opciones de ejecución. Podrá ejecutarse si el código tiene los componentes necesarios para poder simular el comportamiento del robot. En caso contrario, el actuador no podrá hacer nada.

Nombre del Caso de Uso
Simular robot móvil
Descripción
Se trata de la segunda opción de la simulación o ejecución. Como con el actuador lineal, funcionará si se configuran todos los componentes necesarios en el sketch.

Nombre del Caso de Uso
Mostrar salida por consola
Descripción
El usuario visualizará por consola la salida del compilador y/o del código escrito en el sketch. En este último caso no siempre habrá salida, dependerá del propio usuario.

Nombre del Caso de Uso
Detener ejecución
Descripción
El usuario podrá detener la ejecución del sketch siempre y cuando se esté ejecutando. En caso contrario, esta opción no estará disponible, ya que no será necesaria. Tampoco es necesario parar la simulación para modificar el código.

Nombre del Caso de Uso
Consultar ayuda
Descripción
El usuario deberá tener acceso a un manual de ayuda. Para ello, en la barra superior habrá un botón de ayuda desde el cuál acceder a dicho manual.

Nombre del Caso de Uso
Ejecutar comando de edición
Descripción
Este caso de uso describe a las diferentes acciones de edición que podrá ejecutar el usuario sobre el sketch presente en el editor de código. Dichas acciones serán: • Deshacer • Rehacer

5.3 Identificación de los Subsistemas en la Fase de Análisis

En este apartado se descompondrá el sistema a desarrollar en diferentes subsistemas. En el subapartado 5.3.1 se describirán los diferentes subsistemas que se han distinguido y en el apartado 5.3.2 se describirá como se comunican los diferentes subsistemas.

5.3.1 Descripción de los Subsistemas

En esta sección se enumeran todos los subsistemas identificados inicialmente en la aplicación. Los subsistemas son agrupaciones de paquetes y clases que tienen un objetivo propósito común. Ejemplos de subsistemas pueden ser todas las clases que manejen la base de datos (subsistema “base de datos”), clases que agrupen un conjunto de servicios relacionados, etc. A continuación, se muestra el desglose de los diferentes subsistemas que componen el sistema a desarrollar, así como una breve descripción de cada uno. Los subsistemas son los siguientes:

- IGU (Interfaz Gráfica de Usuario)
- Compilador
- Representación Gráfica
- Consola
- Librerías

El siguiente diagrama muestra los subsistemas y sus interacciones:

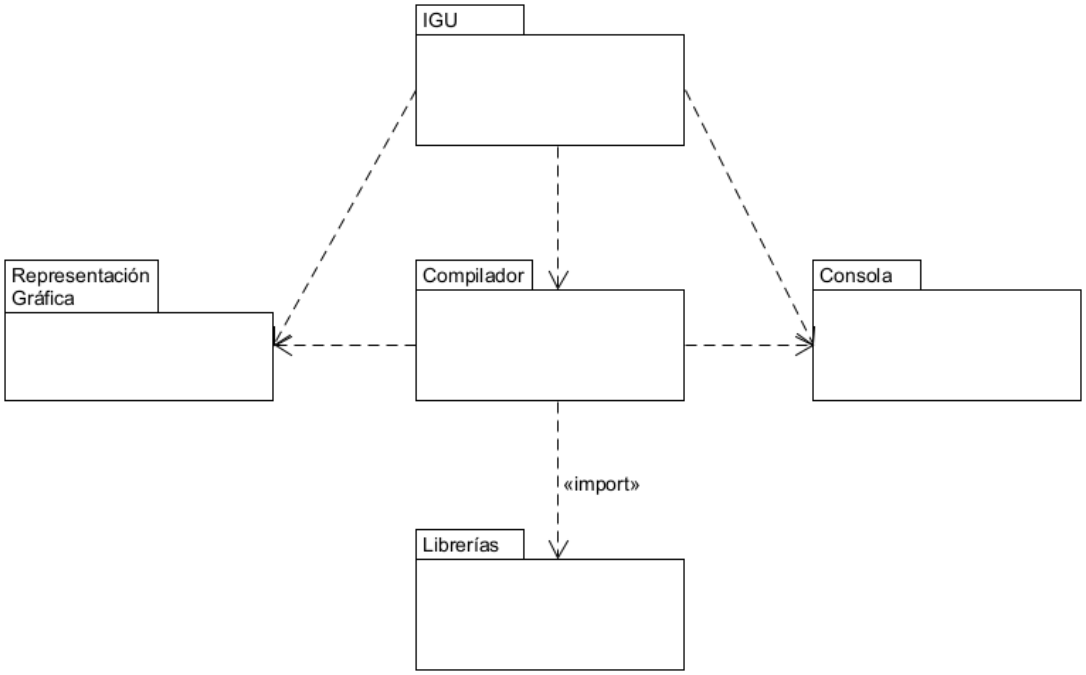


Figura 5.2 Diagrama de Subsistemas

5.3.1.1 IGU

Se trata del subsistema que se encargará de recibir la entrada por parte del usuario, así como mostrar la salida propia del programa. Es decir, actuará como entrada y salida del programa. La IGU (Interfaz Gráfica de Usuario) se comunicará con el compilador para ejecutar el código, con la representación gráfica para mostrar lo que hacen los robots con dicho código, y con la consola para mostrar la salida o bien de errores, o bien de mensajes que imprima el usuario voluntariamente a través del código.

5.3.1.2 Compilador

El subsistema del compilador se encarga de transformar el sketch de Arduino a código ejecutable. Todo el subsistema es encargado de todas las acciones de compilación, empezando por el análisis léxico, siguiendo por la transformación del "parse tree" obtenido a un "AST", continuando por el análisis sintáctico y la posterior generación de código.

Es un subsistema clave para el programa, ya que es el que se encargará de generar las diferentes salidas (por consola y la salida gráfica que muestra los movimientos de los robots). Entrará en acción cuando se compile o ejecute el sketch programado.

5.3.1.3 Representación Gráfica

Se encargará de mostrar las imágenes de los robots realizando las acciones que han sido programadas. En el caso del actuador lineal, mostrará únicamente el robot, ya que solo tiene una pieza móvil y un par de botones que pueden limitar su movimiento.

En el caso del robot móvil hay que mostrar varias cosas:

- El propio robot, que es quien se va a mover.
- Obstáculos, para comprobar que el robot no se va a chocar con nada después en una práctica real.
- Una pista (color negro, por similitud con la práctica real) por la que se pueda mover el robot. En ella se probarán la mayoría de los ejercicios.

Recibirá las instrucciones del subsistema del compilador directamente.

5.3.1.4 Consola

Se encargará de toda salida escrita que sea necesaria, ya sea porque se deban mostrar errores de carácter léxico o sintáctico, o bien se necesite dar consejos al usuario de algo que esté programando mal o que esté en una dirección incorrecta en el desarrollo del código; o bien simplemente el usuario necesite imprimir mensajes a través de consola como salida por cualquier razón. Recibirá los mensajes del compilador de manera directa.

5.3.1.5 Librerías

Es un subsistema estrechamente relacionado con el compilador. En él se encuentran todas las librerías de Arduino necesarias para la programación de los sketches. Se programarán solamente aquellas que sean estrictamente necesarias para la realización de los ejercicios de la asignatura, aunque estará optimizado para su posterior ampliación, si se diese el caso de que fuese necesario.

5.3.2 Descripción de los Interfaces entre Subsistemas

Una vez identificados los subsistemas, debemos también describir cómo será la comunicación entre los mismos. En el caso de nuestro sistema, los distintos subsistemas se comunicarán de forma local, empleando la llamada a los diferentes métodos que proporcionen las diferentes clases que compongan el subsistema.

En algún caso podrá ser necesario la comunicación entre subsistemas empleando archivos de tipo JSON o XML.

5.4 Diagrama de Clases Preliminar del Análisis

En este apartado se muestran y describen las clases que potencialmente tendrá el sistema. Estas clases pueden no ser las definitivas, pero se aproximarán bastante a lo que finalmente será la estructura de la aplicación. En el apartado 5.4.1 se mostrará el diagrama de clases obtenido del análisis, teniendo en cuenta los requisitos y casos de uso.

En el apartado 5.4.2 se describirán las diferentes clases obtenidas y mostradas en el anterior subapartado.

5.4.1 Diagrama de Clases

A continuación, se muestra el diagrama de clases preliminar del análisis, obtenido de los requisitos y los casos de uso. Primero se muestra el diagrama completo, más adelante se desglosa para facilitar su lectura:

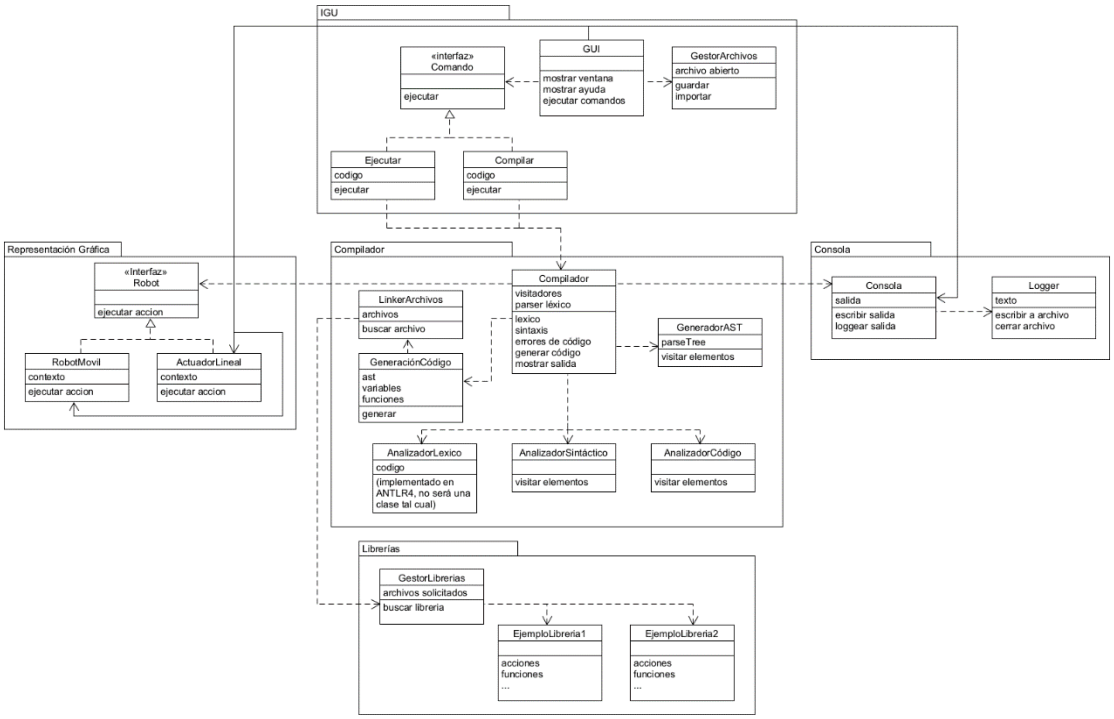


Figura 5.3 Diagrama de clases completo

Por razones de visibilidad, se muestran a continuación una imagen de cada subsistema con sus correspondientes clases:

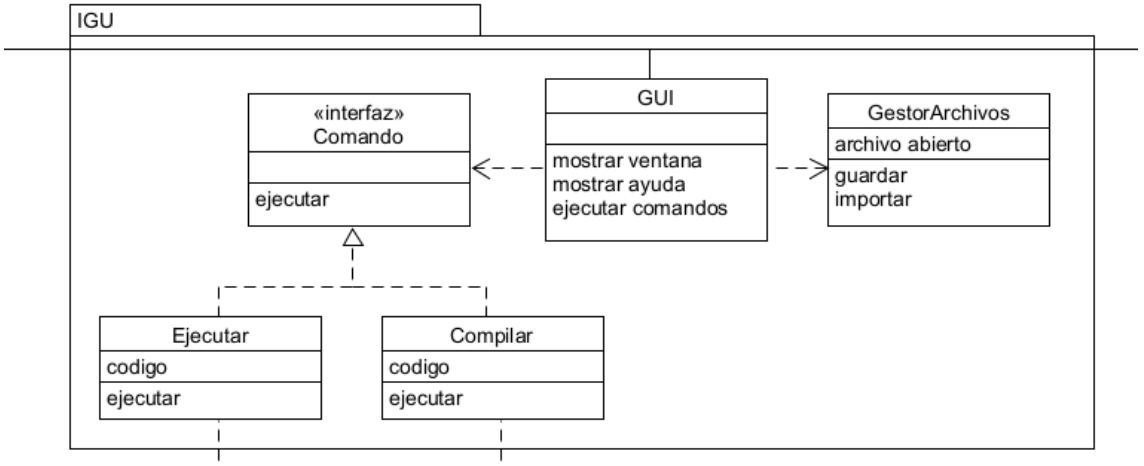


Figura 5.4 Clases del Subsistema IGU

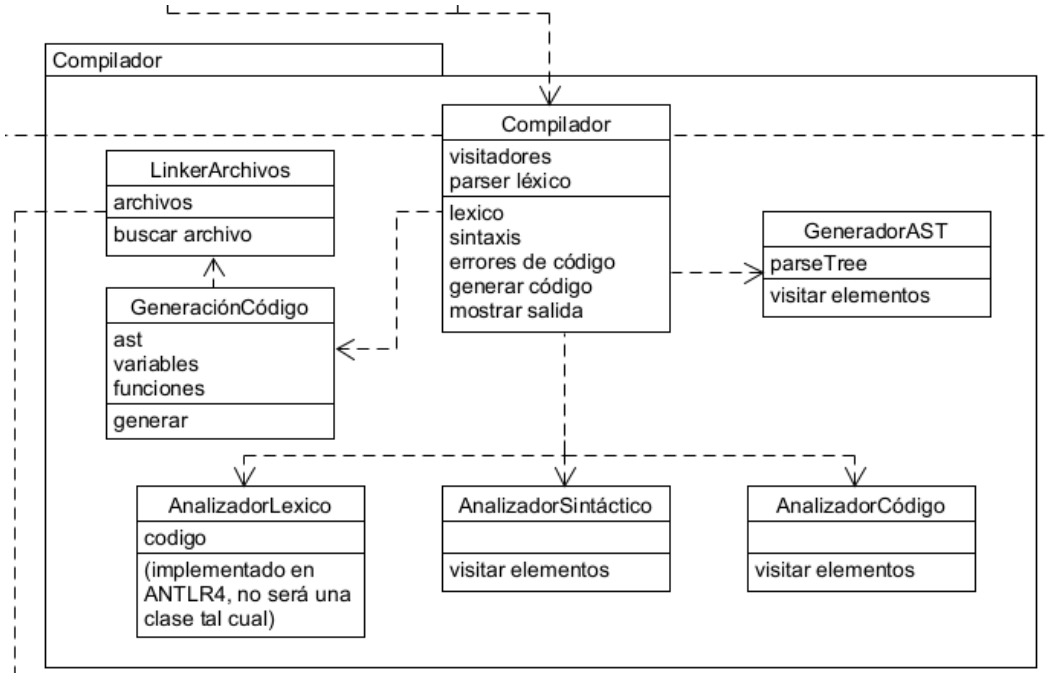


Figura 5.5 Clases del Subsistema Compilador

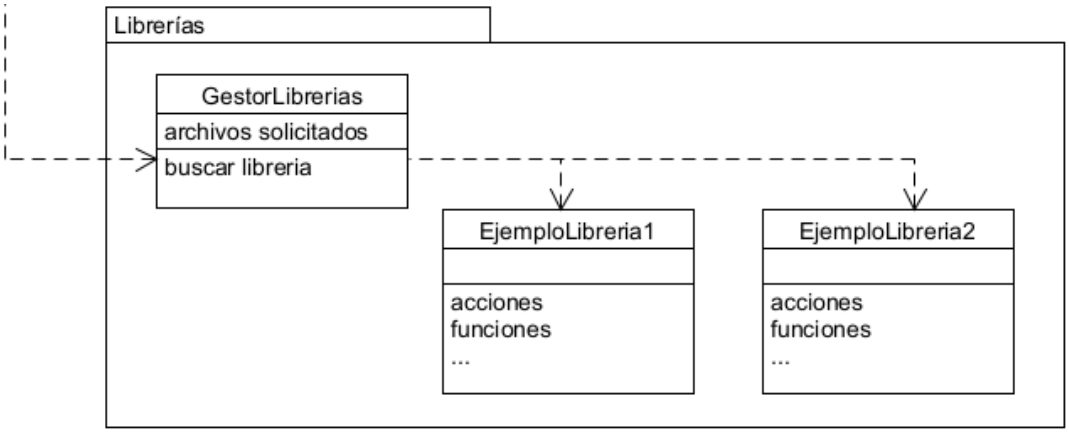


Figura 5.6 Clases del Subsistema Librerías

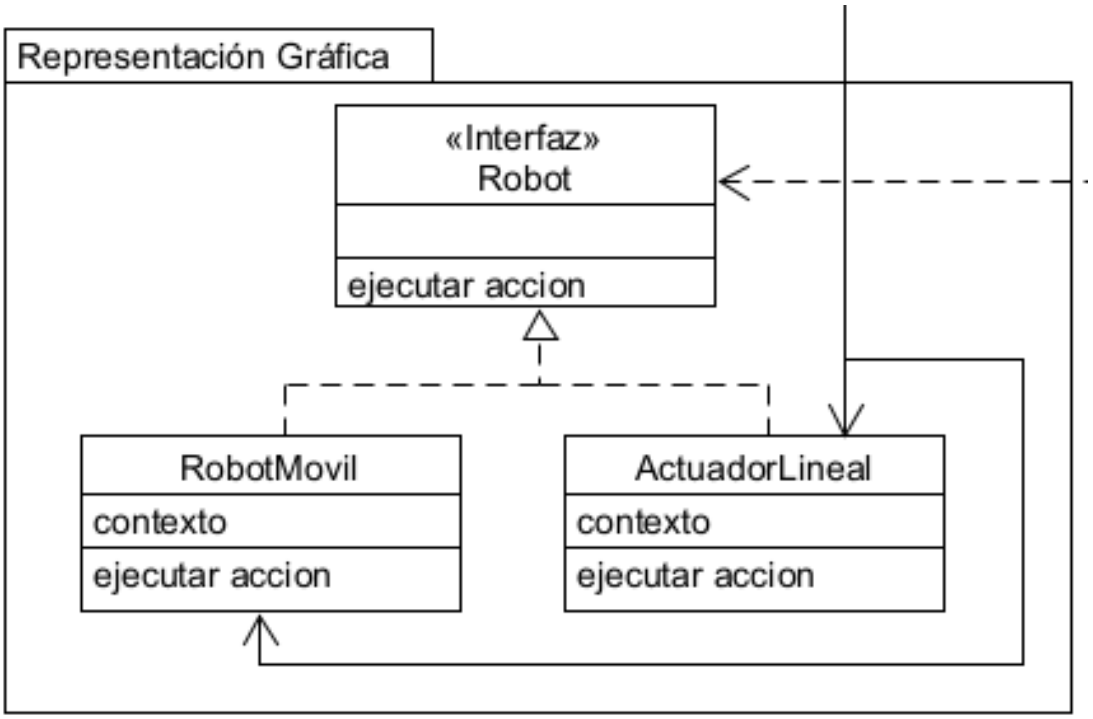


Figura 5.7 Clases des Subsistema Representación Gráfica

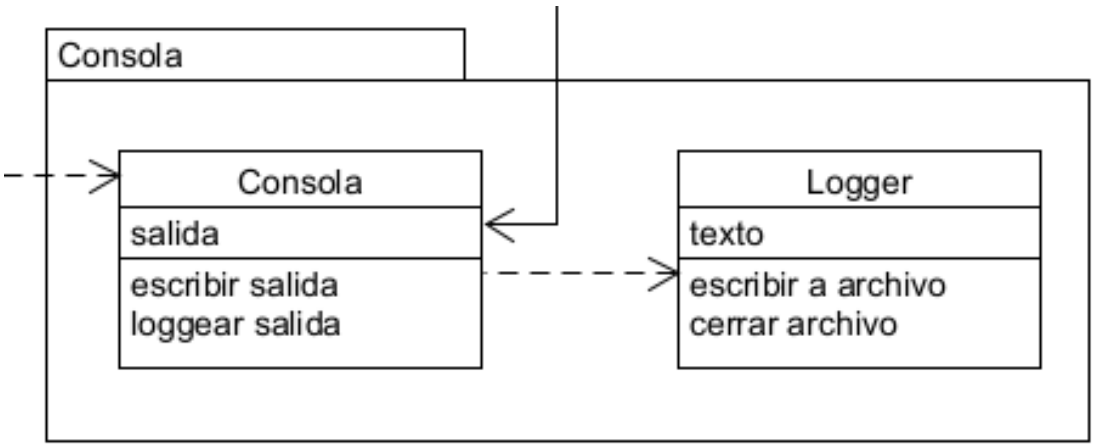


Figura 5.8 Clases del Subsistema Consola

Cabe destacar que al ser este diagrama de clases preliminar y al estar desarrollado empleando una metodología ágil, este diagrama está sujeto a adiciones (sobre manera) y a eliminación de clases, aunque las clases propuestas son necesarias según los casos de uso y requisitos obtenidos.

5.4.2 Descripción de las Clases

A continuación, se explicará la funcionalidad que se considera que tendrá cada clase según el análisis realizado. Este subapartado se dividirá según los subsistemas explicados en el apartado 5.3.

5.4.2.1 IGU

Nombre de la Clase
GUI
Descripción
Clase dedicada a la creación de la ventana de aplicación que se le mostrará al usuario y a través de la cual deberá realizar todas las interacciones necesarias
Responsabilidades
Se encargará de crear la ventana de aplicación y de recibir las interacciones del usuario con la interfaz, incluido el desarrollo del sketch de Arduino
Métodos Propuestos
mostrar ventana: Creará y mostrará la ventana principal de la aplicación, incluyendo los componentes que la forman. **mostrar ayuda:** Mostrará la ventana de ayuda que se creará una vez finalizada la aplicación y su funcionalidad. **ejecutar comandos:** Ejecutará los comandos que el usuario ejecute a través de los diferentes botones proporcionados. Dichos comandos estarán bajo la interfaz Comando

Nombre de la Clase
GestorArchivos
Descripción
Será la clase que realice la creación de los archivos para guardar los sketches, así como importarlos
Responsabilidades
Se encargará de la gestión de archivos, es decir, de su creación, guardado e importado
Atributos Propuestos
archivo abierto: Será el archivo que contenga el sketch que se está desarrollando actualmente.
Métodos Propuestos
guardar: Escribe al sketch al archivo que está actualmente abierto o, en su defecto, lo escribe a un nuevo archivo con el nombre propuesto por el usuario. **importar:** Abre un archivo seleccionado por el usuario y copia su contenido al editor de texto del GUI. (puede darse la situación de que sea necesario un método **crear**, aunque en esta fase no es del todo seguro)

Nombre de la Clase
Comando
Descripción
Interfaz que describe los métodos que debe de implementar cada comando del GUI
Responsabilidades
Establece un contrato que todo comando que lo implemente debe cumplir para poder ser ejecutado en el GUI
Métodos Propuestos

ejecutar: Realiza la acción que ha sido seleccionada o ejecutada por parte del usuario al pulsar un botón o cualquier otro elemento interactivo de la GUI. Todo elemento necesario para la ejecución del comando será pasado a través del constructor de la propia clase.

Nombre de la Clase
Ejecutar
Descripción
Clase que hereda de comando y que ejecuta el sketch
Responsabilidades
Se encargará de ejecutar el comando compilar primero, ya que es necesario que no haya errores antes de ejecutar, y luego pasará a ejecutar el código compilado, dando paso a los métodos que mostrarán la salida del programa
Métodos Propuestos
ejecutar: Realizará una llamada a la clase compilar, para posteriormente pasar la salida de esta (si es correcta) a la clase adecuada para su posterior ejecución.

Nombre de la Clase
Compilar
Descripción
Clase que hereda de Comando y que compila el sketch
Responsabilidades
Esta clase compilará el código escrito en el editor de texto, mostrando (si los hubiese) los errores por salida de consola, empleando para ello la clase adecuada. (puede que no interactúe directamente con la consola)
Métodos Propuestos
ejecutar: Enviará el código al compilador para que este detecte los errores (si los hay). No hace falta que genere el código (teóricamente), pero probablemente se implemente de esa manera de todos modos.

5.4.2.2 Compilador

Nombre de la Clase
Compilador
Descripción
Compila el código escrito en el editor de texto de GUI
Responsabilidades
Compilará el código del sketch. Para ello, lo pasará por las diferentes fases de comprobación de errores: errores de tipo léxico, sintáctico y otros errores de código. Mostrará la salida necesaria por consola: errores si los hay o que todo se ha compilado correctamente. También se encargará de llamar a la clase responsable de generar el código a partir del AST.
Atributos Propuestos

visitadores: Son los diferentes visitadores necesarios para analizar el código y mostrar los errores, así como generar el código a partir del AST obtenido del sketch. **parser léxico:** Al estar usando ANTLR4, el parser léxico es una clase especial, ya generada una vez escrita la gramática.
Métodos Propuestos
léxico: Se encargará de enviar el código al analizador léxico para su análisis, la salida se redirige al generador de AST. **sintaxis:** Se encargará de enviar el AST al analizador sintáctico para su análisis. **errores de código:** Se encargará de analizar el AST en busca de otros errores no cubiertos. **generar código:** Generará el código una vez estén descartados los errores. **mostrar salida:** A partir del código generado, mostrará la salida de la ventana principal

Nombre de la Clase
GeneradorAST
Descripción
Genera el árbol AST con el código correspondiente
Responsabilidades
Se encargará de generar el AST a partir del parse tree generado por ANTLR4. Esto es necesario antes de buscar errores de tipo sintáctico u otros errores.
Atributos Propuestos
parseTree: El árbol generado por el analizador léxico a partir de la gramática definida.
Métodos Propuestos
visitar elementos: Visita los elementos del parse tree (empleando el patrón visitor) para generar el AST correspondiente.

Nombre de la Clase
AnalizadorLexico
Descripción
Busca errores léxicos en el código
Responsabilidades
Buscará errores léxicos con respecto a la gramática empleada y generará el parse tree que corresponda al código
Métodos Propuestos
(generado automáticamente por ANTRL4)

Nombre de la Clase
AnalizadorSIntactico
Descripción
Busca errores sintácticos en el código
Responsabilidades
Buscará errores sintácticos en el AST generado.
Métodos Propuestos
visitar elementos: Visitará los nodos del AST en busca de errores sintácticos

Nombre de la Clase
AnalizadorCodigo
Descripción
Busca otros errores en el código no detectados en las anteriores fases de análisis del compilador
Responsabilidades
Buscará errores de otros tipos en el AST generado. Estos errores serán por lo general cometidos por el usuario a la hora de programar el robot, es decir, serán errores con respecto a cómo se está programando el robot, con respecto a si se va por buen camino a la hora de desarrollar el sketch
Métodos Propuestos
visitar elementos: Visitará los nodos del AST en busca de otros errores

Nombre de la Clase
GeneracionCodigo
Descripción
Generará el código a ejecutar a partir del AST
Responsabilidades
Convertir el AST en código ejecutable, mostrando así la salida al usuario a través de la ventana principal
Atributos Propuestos
AST: El árbol de sintaxis abstracta obtenido. **variables:** Guardará las variables obtenidas del análisis del AST **funciones:** Guardará las funciones obtenidas del análisis del AST
Métodos Propuestos
generar: Generará el código a partir del AST obtenido anteriormente

Nombre de la Clase
LinkerArchivos
Descripción
Busca los archivos incluidos en el sketch necesarios
Responsabilidades
Se encargará de unir las librerías por defecto de Arduino y aquellos archivos incluidos por parte del usuario al código generado
Atributos Propuestos
archivos: Lista de los archivos a unir con el código generado.
Métodos Propuestos
buscar archivo: Buscará el archivo correspondiente que se le pida por parámetro y lo devolverá para su unión.

5.4.2.3 Librerías

Nombre de la Clase
GestorLibrerías
Descripción
Busca las librerías necesarias
Responsabilidades
Buscará las librerías entre los archivos que conozca y devolverá aquella que corresponda con el archivo solicitado
Atributos Propuestos
archivos: Lista con los archivos que se necesitan
Métodos Propuestos
buscar librería: Busca las librerías necesarias para la ejecución del sketch

Nombre de la Clase
EjemploLibreriaN
Descripción
Librerías programadas para la ejecución del sketch
Responsabilidades
Ejecutar los métodos que pertenezcan a la librería y que haya pedido el usuario

5.4.2.4 Representación Gráfica

Nombre de la Clase
Robot
Descripción
Interfaz de la que heredan ambos robots
Responsabilidades
Mostrar el contrato que deben cumplir los robots para poder mostrarse como salida
Métodos Propuestos
ejecutar acción: Ejecutará cada acción que le sea ordenada al robot en cuestión

Nombre de la Clase
RobotMovil
Descripción
Representación gráfica del Robot Móvil. Hereda de la interfaz Robot
Responsabilidades
Representar gráficamente las acciones que debería hacer el robot en la realidad
Métodos Propuestos
ejecutar acción: Ejecutará la acción en cuestión que se le ordene al robot

Nombre de la Clase
ActuadorLineal
Descripción
Representación gráfica del Actuador Lineal. Hereda de la interfaz Robot
Responsabilidades
Representar gráficamente las acciones que debería hacer el actuador en la realidad
Métodos Propuestos
ejecutar acción: Ejecutará la acción en cuestión que se le ordene al actuador

5.4.2.5 Consola

Nombre de la Clase
Consola
Descripción
Mostrará la salida por consola de la aplicación
Responsabilidades
Mostrar el texto que se le pase como parámetro a través de la consola de la ventana principal
Atributos Propuestos
salida: El texto que se deberá mostrar por consola
Métodos Propuestos
escribir salida: Mostrará la salida pasada a la clase a través de la consola de la Ventana Principal. **loggear salida:** Pasará al logger la salida para ser archivada en un log.

Nombre de la Clase
Logger
Descripción
Creará un archivo con el *feedback* recibido por el alumno
Responsabilidades
Escribir todos los errores y *feedback* dado al alumno por parte de la aplicación
Atributos Propuestos
texto: La información que será añadida al archivo log
Métodos Propuestos
escribir a archivo: Escribirá el texto recibido por la clase a un archivo log. **cerrar archivo:** Si queda algo por escribir en el log, lo escribirá y cerrará el archivo log (generalmente se ejecutará cuando se cierre la aplicación)

5.5 Análisis de Casos de Uso y Escenarios

En este apartado se realizará el análisis de los casos de uso y la creación de los escenarios. Cada subapartado corresponde a un caso de uso.

5.5.1 Crear archivo

Crear archivo	
Precondiciones	-
Postcondiciones	Se creará un sketch de Arduino con los métodos obligatorios “setup” y “loop”.
Actores	Alumno/usuario y profesor.
Descripción	1. El usuario pincha en la opción de menú archivo. 2. El usuario pincha en el botón crear archivo dentro del menú. 3. El sistema pregunta al usuario si está seguro de crear un nuevo archivo. 4. El sistema crea el nuevo archivo con los métodos mencionados.
Variaciones (escenarios secundarios)	• **Escenario alternativo 1:** El usuario cancela la creación del nuevo archivo: ○ El sistema confirmará la cancelación de la creación del archivo.
Excepciones	-
Notas	-

Tabla 5.5 Escenario de crear Archivo

5.5.2 Guardar archivo

Guardar archivo	
Precondiciones	El archivo debe contener algo de código. El archivo debe de compilar correctamente.
Postcondiciones	El sistema escribirá al archivo el código que actualmente tenga el editor de código.
Actores	Alumno/usuario y profesor.
Descripción	1. El usuario en cuestión pinchará sobre el botón guardar. 2. El sistema guardará el sketch en el archivo correspondiente. 3. El sistema confirmará el guardado del archivo.
Variaciones (escenarios secundarios)	• **Escenario alternativo 1:** No se ha creado todavía un archivo al que guardar: ○ El sistema pedirá al usuario que guarde el

	archivo en una localización de su preferencia. ○ El sistema pedirá al usuario un nombre para el archivo. ○ El sistema guardará el archivo y confirmará al usuario su guardado. • **Escenario alternativo 2:** El usuario utiliza el botón guardar del menú archivo: ○ El sistema actúa de la misma manera que con el botón
Excepciones	• Que el guardado haya sido incorrecto por cualquier razón: ○ El sistema comunicará al usuario el error que corresponda.
Notas	-

Tabla 5.6 Escenario de Guardar archivo

5.5.3 Importar archivo

Importar archivo	
Precondiciones	Debe de existir el archivo que se vaya a importar. El archivo debe de tener la extensión ".ino".
Postcondiciones	El código del archivo será escrito en el editor de código.
Actores	Usuario/alumno y profesor.
Descripción	1. El usuario en cuestión busca el archivo. 2. El sistema escribe el contenido del archivo en el editor de código y registra el nombre del archivo para poder sobrescribirlo. 3. El sistema comunica mediante ventana emergente que se ha importado el archivo correctamente.
Variaciones (escenarios secundarios)	• **Escenario alternativo 1:** El usuario utiliza el botón importar del menú archivo: ○ El sistema actúa de la misma manera que con el botón.
Excepciones	• Que el importado sea incorrecto por cualquier razón: ○ El sistema comunicará al usuario el error que corresponda.
Notas	-

Tabla 5.7 Escenario de Importar archivo

5.5.4 Escribir sketch

Escribir Sketch	
Precondiciones	-
Postcondiciones	El código escrito por el usuario se reflejará en el editor de código.
Actores	Usuario/alumno y profesor.
Descripción	1. El usuario en cuestión escribirá el código a través del teclado de su ordenador. 2. El sistema reflejará dicho código en el editor de código.
Variaciones (escenarios secundarios)	• **Escenario alternativo 1:** El usuario escribe código pegándolo: ○ El sistema recibe dicho código.
Excepciones	-
Notas	-

Tabla 5.8 Escenario de Escribir Sketch

5.5.5 Cambiar robot

Cambiar robot	
Precondiciones	-
Postcondiciones	El robot elegido será el que se dibuje en la representación de código una vez se pinche en ejecutar.
Actores	Alumno/usuario y profesor.
Descripción	1. El usuario pincha en la lista desplegable de la barra de botones. 2. El usuario escoge el robot que quiera (robot móvil o actuador lineal). 3. El sistema cambia el robot que va a representar. 4. El sistema comunica mediante ventana emergente que se ha cambiado el robot.
Variaciones (escenarios secundarios)	-
Excepciones	• El sistema puede fallar si el cambio de robot no finaliza correctamente: ○ Se notificará el error al usuario mediante una ventana emergente.
Notas	-

Tabla 5.9 Escenario de Cambiar robot

5.5.6 Elegir robot móvil

Elegir robot móvil	
Precondiciones	Debe de estar abierta la lista desplegable.
Postcondiciones	El robot que se dibujará en la representación será el robot móvil.
Actores	Usuario/alumno y profesor.
Descripción	1. Se pincha en la opción de la lista que corresponde con el robot móvil. 2. El sistema cambia de robot al robot móvil. 3. El sistema comunica al usuario que el cambio se ha realizado con éxito.
Variaciones (escenarios secundarios)	• **Escenario alternativo 1:** El robot móvil es el que está seleccionado: ○ El sistema no tendrá que hacer nada
Excepciones	• El cambio de robot puede fallar ○ El sistema reportará el fallo mediante un mensaje de error en una ventana emergente.
Notas	-

Tabla 5.10 Escenario de Elegir el robot móvil

5.5.7 Elegir actuador lineal

Elegir actuador lineal	
Precondiciones	Debe de esta abierta la lista desplegable.
Postcondiciones	El robot que se dibujará en la representación será el actuador lineal.
Actores	Usuario/alumno y profesor.
Descripción	1. Se pincha en la opción de la lista que corresponde con el actuador lineal. 2. El sistema cambia de robot al actuador lineal. 3. El sistema comunica al usuario que el cambio se ha realizado con éxito.
Variaciones (escenarios secundarios)	• **Escenario alternativo 1:** El actuador lineal está seleccionado: ○ El sistema no tendrá que hacer nada.
Excepciones	• El cambio de robot puede fallar: ○ Se mostrará el fallo por un mensaje de

	error en una ventana emergente.
Notas	-

Tabla 5.11 Escenario de Elegir actuador lineal

5.5.8 Compilar sketch

Compilar sketch	
Precondiciones	El editor de código no está vacío.
Postcondiciones	El sistema reporta al usuario si tiene algún fallo en el Sketch.
Actores	Alumno/usuario y profesor
Descripción	1. El usuario en cuestión debe pinchar en el botón compilar (en principio en el submenú herramientas, no se descarta ponerlo como botón). 2. El sistema compilará el sketch, pasándolo por las fases necesarias de detección de errores, pero sin llegar a ejecutarlo. 3. El sistema reportará al usuario por consola sus errores de cualquier carácter. 4. El sistema escribirá los errores en un archivo log.
Variaciones (escenarios secundarios)	-

Excepciones	• El usuario trata de compilar un archivo vacío:
	  ○ El sistema deberá comunicar al usuario mediante una ventana emergente su fallo.
	• El usuario escribe código que no funciona con el robot seleccionado:
	  ○ El sistema deberá reportar mediante una ventana emergente y por consola dicha situación.

Notas	-

Tabla 5.12 Escenario de Compilar sketch

5.5.9 Comprobación de errores léxicos

Comprobación de errores léxicos	
Precondiciones	El usuario debe de haber pulsado el botón compilar
Postcondiciones	Se proporcionará un listado de errores léxicos (si los hay). Se generará un parse tree.
Actores	Sistema
Descripción	1. El sistema recibirá el sketch escrito en el editor de código.

	2. El sistema analizará dicho sketch. 3. El sistema generará un listado de errores léxicos. 4. El sistema generará un "parse tree".
Variaciones (escenarios secundarios)	-
Excepciones	• La fase de análisis puede fallar por alguna razón: o El sistema comunicará mediante ventana emergente el fallo al usuario. o Dicho error será escrito en el log.
Notas	-

Tabla 5.13 Escenario de Comprobación de Errores léxicos

5.5.10 Comprobación de errores sintácticos

Comprobación de errores sintácticos	
Precondiciones	El usuario debe haber pulsado el botón compilar. El compilador debe haber ejecutado la fase de comprobación de errores léxicos. El compilador debe haber generado el AST.
Postcondiciones	Un listado de errores sintácticos.
Actores	Sistema.
Descripción	1. El sistema analizará el AST en busca de errores sintácticos 2. El sistema generará una lista de errores sintácticos.
Variaciones (escenarios secundarios)	-
Excepciones	• El análisis del AST puede fallar por alguna razón: o El sistema deberá comunicar la razón mediante una ventana emergente o mediante consola. o Dicho error deberá ser escrito en el log.
Notas	-

Tabla 5.14 Escenario de Comprobación de errores sintácticos

5.5.11 Comprobación de errores de código

Comprobación de errores de código	
Precondiciones	El usuario debe haber pulsado el botón compilar. El compilador debe haber ejecutado la fase de comprobación de errores sintácticos.
Postcondiciones	Un listado de errores de código.
Actores	Sistema.

Descripción	1. El sistema analizará el AST en busca de otros errores de código no cubiertos. 2. El sistema generará un listado de errores de código.
Variaciones (escenarios secundarios)	-

Excepciones	• El análisis del AST puede fallar por alguna razón:
	  ○ El sistema deberá comunicar la razón mediante una ventana emergente o mediante consola.
	  ○ Dicho error deberá ser escrito en el log.

Notas	-

Tabla 5.15 Escenario de Comprobación de errores de código

5.5.12 Ejecutar sketch

Ejecutar sketch	
Precondiciones	El editor de código no debe de estar vacío. Debe seleccionarse uno de los dos robots disponibles.
Postcondiciones	Se mostrará los movimientos del robot por la representación gráfica. Se mostrará la salida necesaria por consola.
Actores	Usuario/alumno y profesor.
Descripción	1. El usuario pinchará sobre el botón ejecutar. 2. El sistema realizará toda la fase de compilación. 3. El sistema generará el código a partir del AST. 4. El sistema mostrará la salida en forma de los movimientos del robot a través de la representación gráfica. 5. El sistema mostrará la salida necesaria por consola.
Variaciones (escenarios secundarios)	• **Escenario alternativo 1:** El alumno pulsa el botón ejecutar del menú herramientas: ○ El sistema deberá responder de la misma forma.

Excepciones	• La compilación puede dar errores:
	  ○ El sistema no deberá de mostrar la salida que se daría sin errores (es decir, ni “prints” por consola ni los robots representados).
	• La generación de código puede fallar por alguna razón:
	  ○ El sistema deberá reportar el fallo por consola o por ventana emergente si es más grave.

	○ Dicho error deberá ser escrito en el log.
Notas	-

Tabla 5.16 Escenario de Ejecutar sketch

5.5.13 Simular actuador lineal

Simular actuador lineal	
Precondiciones	Debe estar seleccionado como robot el actuador lineal. El código debe de compilar. El usuario debe haber pulsado el botón ejecutar.
Postcondiciones	Se mostrará por la zona de representación gráfica el comportamiento simulado del actuador lineal.
Actores	Sistema.
Descripción	1. El sistema recibirá el AST. 2. El sistema generará el código necesario para simular el actuador lineal. 3. El sistema dibujará la simulación del actuador en la representación gráfica. 4. El sistema mostrará la salida por consola necesaria.
Variaciones (escenarios secundarios)	-
Excepciones	• La fase de generación de código puede fallar por alguna razón: ○ El error deberá ser mostrado por consola o por ventana emergente si es más crítico. ○ Dicho error será escrito en el log.
Notas	-

Tabla 5.17 Escenario de Simular actuador lineal

5.5.14 Simular robot móvil

Simular robot móvil	
Precondiciones	El sketch debe compilar. El usuario debe haber pulsado el botón ejecutar. El robot seleccionado debe ser el robot móvil.
Postcondiciones	El sistema mostrará por la zona de representación gráfica el comportamiento del robot móvil simulado.
Actores	Sistema.
Descripción	1. El sistema recibirá el AST. 2. El sistema generará el código necesario para simular el robot. 3. El sistema dibujará la simulación del robot en la representación gráfica.

	4. El sistema mostrará la salida por consola necesaria.
Variaciones (escenarios secundarios)	-
Excepciones	• La fase de generación de código puede fallar por alguna razón: ○ El error deberá ser mostrado por consola o por ventana emergente si es más crítico. ○ Dicho error será escrito en el log.
Notas	-

Tabla 5.18 Escenario de Simular robot móvil

5.5.15 Mostrar salida por consola

Mostrar salida por consola	
Precondiciones	La fase de compilación debe de estar terminada. El código debe de estar ejecutándose.
Postcondiciones	Se mostrarán por texto los errores cometidos en el sketch. Se mostrará la salida escrita con el método “print”.
Actores	Sistema.
Descripción	1. El sistema envía los mensajes a la consola. 2. La consola muestra los mensajes recibidos.
Variaciones (escenarios secundarios)	-
Excepciones	-
Notas	-

Tabla 5.19 Escenario de Mostrar salida por consola

5.5.16 Detener ejecución

Detener ejecución	
Precondiciones	El sistema debe de estar ejecutando el sketch.
Postcondiciones	La salida por consola se detendrá. El dibujo en la zona de representación gráfica se parará.
Actores	Usuario/alumno y profesor.
Descripción	1. El usuario debe pulsar el botón parar. 2. El sistema detendrá la salida por consola. 3. El sistema detendrá el dibujo del robot que se esté simulando.
Variaciones (escenarios secundarios)	• **Escenario alternativo 1:** El usuario decide pulsar el botón parar del menú herramientas: ○ El sistema actuará de la misma forma.

Excepciones	-
Notas	-

Tabla 5.20 Escenario de Detener ejecución

5.5.17 Consultar ayuda

Consultar ayuda	
Precondiciones	-
Postcondiciones	El sistema mostrará una ventana emergente con el manual de ayuda al usuario.
Actores	Usuario/alumno.
Descripción	1. El usuario debe pulsar sobre el menú ayuda. 2. El usuario debe pulsar sobre el botón obtener ayuda. 3. El sistema mostrará en una ventana emergente el manual de ayuda al usuario.
Variaciones (escenarios secundarios)	-
Excepciones	-
Notas	-

Tabla 5.21 Escenario de Consultar ayuda

5.5.18 Ejecutar comando de edición

Ejecutar comando de edición	
Precondiciones	Debe existir código que editar.
Postcondiciones	El código será editado de alguna forma.
Actores	Usuario/alumno y profesor.
Descripción	1. El usuario ejecutará el comando de edición que estime oportuno. 2. El sistema realizará la acción de edición correspondiente.
Variaciones (escenarios secundarios)	-
Excepciones	-
Notas	-

Tabla 5.22 Escenario de Ejecutar comando de edición

5.6 Análisis de Interfaces de Usuario

Este apartado tratará el análisis inicial de la interfaz de usuario. Para ello, en el apartado 5.6.1 se realizará la descripción de la interfaz, que consistirá en la descripción de la pantalla o pantallas que tendrá el sistema, mientras que el apartado 5.6.2 tratará la comunicación de fallos y el tratamiento de entradas.

5.6.1 Descripción de la Interfaz

En la siguiente imagen se muestra el primer mockup que se ha hecho de la aplicación (a mano a través de OneNote):

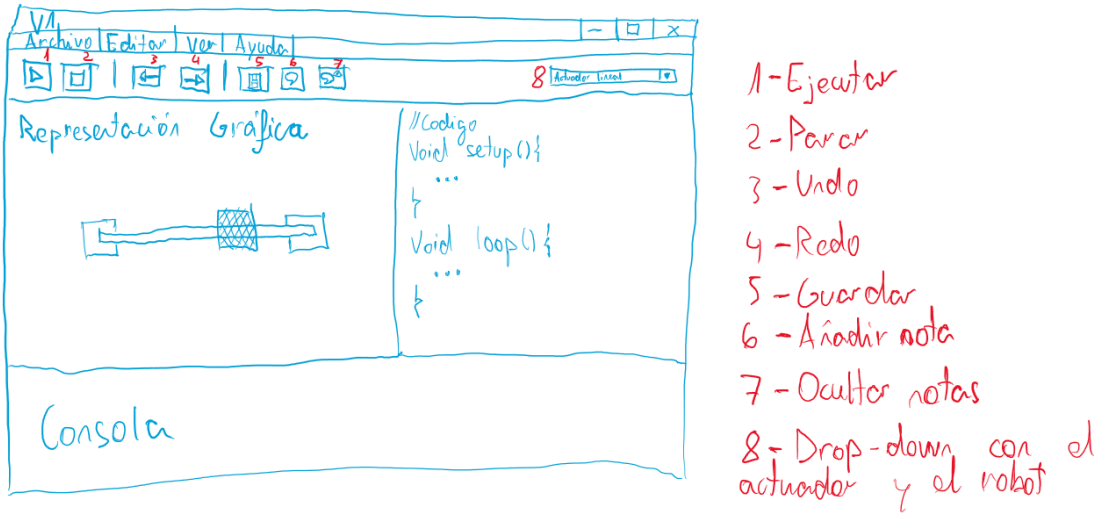


Figura 5.9 Mockup preliminar del Sistema

Como podemos ver, la ventana principal contiene una barra de menú, una barra de botones, y tres contenedores que sirven para mostrar las salidas y para introducir el código.

La barra de menú tiene cuatro opciones:

- Archivo: Permite crear, importar o guardar los archivos con los sketches que escribamos en el editor de texto. En el caso del guardado, dará dos opciones, guardar o guardar como, en el último caso, creará un nuevo archivo en el que guardar.
- Editar: Permitirá realizar las diferentes acciones de edición propias de un editor (deshacer, rehacer, copiar, pegar…).
- Ver: Dará diferentes opciones de personalización de la IGU (añadir números de línea, quitar una perspectiva, etc.
- Ayuda: Mostrará la ayuda del sistema y también la información de la aplicación (su versión entre otros).

En cuanto a la barra de herramientas, vemos siete botones y una lista desplegable:

- Ejecutar: Permitirá compilar y ejecutar el código, para posteriormente mostrar la salida por la representación gráfica y la consola.
- Parar: Para la ejecución del programa, es decir, no se mostrará más salida.
- Undo: Deshacer la última acción de edición.
- Redo: Rehacer la última acción desecha.
- Guardar: Guarda el sketch en el archivo correspondiente si ya existe, y si no le pide al usuario crear uno nuevo.
- Añadir nota: Permitiría añadir notas al código (funcionalidad descartada por los comentarios de código)
- Ocultar notas: Ocultaría las notas creadas anteriormente (descartada).
- Lista desplegable: Permite cambiar entre los dos robots.

Finalmente están los tres paneles o componentes, que son los que se describen a continuación:

- Representación gráfica, que nos mostrará los robots realizando las acciones que hayan sido programadas.
- Editor de código, que como su propio nombre indica nos permitirá editar el sketch de Arduino. Se le podrá añadir, a voluntad del usuario, el número de línea correspondiente a cada sentencia de código
- Consola, que mostrará la salida por texto; bien sean errores de código de diferente índole, o bien sea salida intencionada por el usuario, mediante código en el sketch (método "print").

A continuación, se muestra un diseño del mockup refinado con los cambios mencionados con respecto a las notas y otras funcionalidades, mediante draw.io:

Figura 5.10 Mockup refinado del sistema

Los cambios en esta versión del mockup son:

- Eliminación de los botones correspondientes a las notas.
- Añadido en el editor de código el número de línea por defecto.
- Cambiadas diferentes opciones de la barra de menú:
    - Vista se reemplaza por Window (ventana).
    - Añadidos las opciones de Options y Tools. La primera permitirá acciones como cambiar la fuente y su tamaño y la segunda utilizar herramientas como ejecutar o parar.

Por último, cabe destacar que los tres paneles (la representación, el código y la consola) son paneles cuyo tamaño se puede cambiar. Es decir, sus separadores se pueden mover y, con ello, cambiar el tamaño del panel.

5.6.2 Descripción del Comportamiento de la Interfaz

Durante el uso habitual por parte de cualquier usuario de la aplicación, van a suceder errores de diversa índole. Por ello, serán tratados de diferentes formas:

Los errores que sean de un carácter más importante o crítico, como una carga de archivo incorrecta o un guardado incorrecto, se mostrarán mediante mensaje en una ventana emergente, localizada en el centro de la ventana principal.

En cuanto a errores más propios de la ejecución de código, estos serán mostrados por consola, ya que van a ser errores de compilación y, como hemos visto en anteriores apartados, la consola está pensada precisamente para esos casos de errores, entre otros.

5.7 Especificación del Plan de Pruebas

En esta se diseña y expone el plan de pruebas de la aplicación y sus funciones, así como todos los mecanismos que se utilizarán para detectar errores y corregirlos ya en la fase de implementación.

Debe destacarse que todas las pruebas serán implementadas al terminar cada módulo. En algunos casos, si la clase implementada es muy compleja, se implementarán una vez terminada la clase correspondiente.

Mayormente, las pruebas implementadas serán unitarias; aquellas que sirven para poner a prueba el correcto funcionamiento de un módulo de código o clase. Aunque también se implementarán pruebas de usabilidad y pruebas de integración (de algún módulo, sobre manera el compilador).

5.7.1 Pruebas unitarias

En este apartado se muestran las pruebas unitarias que se realizarán de la aplicación. Dichas pruebas se realizarán para comprobar si funcionan bien las partes más pequeñas del código.

En total, se realizarán pruebas unitarias para los 18 casos de uso mencionados en el apartado 5.5, cubriendo en todo caso las situaciones descritas en ellos.

Caso de Uso 1: Crear archivo	
Prueba	**Resultado Esperado**
Crear un archivo con el editor de código vacío	Debería crearse un archivo con los métodos por defecto
Prueba	**Resultado Esperado**
Crear un archivo cuando el editor tiene el código por defecto	El editor no cambiará en ningún caso.
Prueba	**Resultado Esperado**
Crear un archivo cuando el editor tiene código programado	El sistema deberá preguntar al usuario para confirmar la acción. El editor de código se cambiará a los métodos por defecto
Prueba	**Resultado Esperado**
Cancelar la Operación	El editor permanece sin cambios. El sistema notificará la cancelación de la acción.

Tabla 5.23 Test de Caso de uso 1

Caso de Uso 2: Guardar archivo	
Prueba	**Resultado Esperado**
Guardar por primera vez un archivo	El sistema creará el nuevo archivo y guardará el contenido de código en él.
Prueba	**Resultado Esperado**
Guardar en un archivo ya existente	El sistema sobrescribirá el archivo ya existente con el código tal y como está en el editor.

Tabla 5.24 Test de Caso de uso 2

Caso de Uso 3: Importar archivo	
Prueba	**Resultado Esperado**
Importar un archivo existente	El sketch deberá mostrarse correctamente en el editor de código.
Prueba	**Resultado Esperado**
Importar un archivo con la extensión incorrecta	El sistema debe notificar que no se puede abrir el archivo.

Tabla 5.25 Test de Caso de uso 3

Caso de Uso 4: Escribir sketch	
Prueba	**Resultado Esperado**
Escribir código	El sketch deberá reflejar el código escrito por el usuario.

Tabla 5.26 Test de Caso de uso 4

Caso de Uso 5: Cambiar robot	
Prueba	**Resultado Esperado**
Cambiar al robot móvil	El sistema debe notificar que el cambio se ha realizado con éxito. El sistema debe reflejar en el desplegable que se está usando el robot móvil.
Prueba	**Resultado Esperado**
Cambiar al actuador lineal	El sistema debe notificar que el cambio se ha realizado con éxito. El sistema debe reflejar en el desplegable que se está usando el actuador lineal.

Tabla 5.27 Test de Caso de uso 5

Caso de Uso 6: Elegir robot móvil	
Prueba	**Resultado Esperado**
Intentar elegir el robot móvil cuando este ya está seleccionado	El sistema no debe hacer nada.
Prueba	**Resultado Esperado**
Pasar del actuador lineal al robot móvil	El sistema notificará que el cambio ha tenido éxito. El robot referenciado en código deberá ser el móvil.

Tabla 5.28 Test de Caso de uso 6

Caso de Uso 7: Elegir actuador lineal	
Prueba	**Resultado Esperado**
Intentar elegir el actuador lineal cuando este ya está seleccionado	El sistema no debe hacer nada.
Prueba	**Resultado Esperado**
Pasar del robot móvil al actuador lineal	El sistema notificará que el cambio ha tenido éxito. El robot referenciado en código deberá ser el actuador lineal.

Tabla 5.29 Test de Caso de uso 7

Caso de Uso 8: Compilar sketch	
Prueba	**Resultado Esperado**
Comprobar que el sketch compila correctamente	El sistema notificará que el sketch ha sido compilado correctamente por consola.
Prueba	**Resultado Esperado**
Comprobar que el sketch no compila correctamente (con errores de todo tipo)	El sistema notificará solo los errores léxicos.
Prueba	**Resultado Esperado**
Comprobar que el sketch tiene solo errores léxicos	El sistema notificará todos los errores léxicos.
Prueba	**Resultado Esperado**
Comprobar que el sketch tiene solo errores sintácticos	El sistema notificará todos los errores sintácticos
Prueba	**Resultado Esperado**
Comprobar que el sketch tiene solo otros errores	El sistema notificará todos los errores de otro tipo.

Tabla 5.30 Test de Caso de uso 8

Caso de Uso 9: Comprobación de errores léxicos	
Prueba	**Resultado Esperado**
No hay errores léxicos	El sistema genera un “parse tree”.
Prueba	**Resultado Esperado**
Hay errores léxicos	El sistema genera una lista de errores.

Tabla 5.31 Test de Caso de uso 9

Caso de Uso 10: Comprobación de errores sintácticos	
Prueba	**Resultado Esperado**
No hay errores sintácticos	El sistema no genera nada.
Prueba	**Resultado Esperado**
Hay errores sintácticos	El sistema genera una lista de errores.

Tabla 5.32 Test de Caso de uso 10

Caso de Uso 11: Comprobación de errores de código	
Prueba	**Resultado Esperado**
No hay errores de código	El sistema no genera nada.
Prueba	**Resultado Esperado**
Hay errores de código	El sistema genera una lista de errores.

Tabla 5.33 Test de Caso de uso 11

Caso de Uso 12: Ejecutar sketch	
Prueba	**Resultado Esperado**
No hay código que ejecutar	El sistema notifica que el código no es ejecutable.
Prueba	**Resultado Esperado**
El código no compila	El sistema notifica los errores de compilación por consola.
Prueba	**Resultado Esperado**
El código compila y es de robot móvil	El sistema representa los movimientos del robot móvil en la representación gráfica.
Prueba	**Resultado Esperado**
El código compila y es de actuador lineal	El sistema representa los movimientos del actuador lineal en la representación gráfica
Prueba	**Resultado Esperado**
El código compila y es de robot móvil, pero esta seleccionado el actuador lineal	El sistema notifica que se está representando el robot equivocado.
Prueba	**Resultado Esperado**
El código compila y es de actuador lineal, pero está seleccionado el robot móvil	El sistema notifica que se está representando el robot equivocado.

Tabla 5.34 Test de Caso de uso 12

Caso de Uso 13: Simular actuador lineal	
Prueba	**Resultado Esperado**
El código es de actuador lineal	El sistema representa correctamente el movimiento del actuador lineal.
Prueba	**Resultado Esperado**
El código no es de actuador lineal	El sistema notifica que se ha seleccionado el robot equivocado.

Tabla 5.35 Test de Caso de uso 13

Caso de Uso 14: Simular robot móvil	
Prueba	**Resultado Esperado**
El código es de robot móvil	El sistema representa correctamente el movimiento del robot móvil.
Prueba	**Resultado Esperado**
El código no es de robot móvil	El sistema notifica que se ha seleccionado el robot equivocado.

Tabla 5.36 Test de Caso de uso 14

Caso de Uso 15: Mostrar salida por consola	
Prueba	**Resultado Esperado**
Se muestran los errores de compilación	El sistema mostrará por consola los distintos errores de compilación.
Prueba	**Resultado Esperado**
Se muestran los mensajes impresos por pantalla del usuario	El sistema mostrará todos los mensajes que el usuario ha decidido imprimir.

Tabla 5.37 Test de Caso de uso 15

Caso de Uso 16: Detener ejecución	
Prueba	**Resultado Esperado**
Pulsar el botón parar	El sistema detiene la representación del robot correspondiente. El sistema detiene la salida por consola.

Tabla 5.38 Test de caso de uso 16

Caso de Uso 17: Consultar ayuda	
Prueba	**Resultado Esperado**
El usuario pulsa el botón de ayuda	El sistema muestra el manual de ayuda al usuario en una ventana emergente

Tabla 5.39 Test de Caso de uso 17

Caso de Uso 18: Ejecutar comando de edición	
Prueba	**Resultado Esperado**
Deshacer	El sistema deshace la última acción.
Prueba	**Resultado Esperado**
Rehacer	El sistema rehace la última acción que ha sido deshecha.

Tabla 5.40 Test de Caso de uso 18

5.7.2 Pruebas de integración

A continuación, se muestran las diferentes pruebas de integración que se van a realizar. Estas pruebas se realizarán al finalizar la programación de cada subsistema de la aplicación, y servirán para comprobar que, efectivamente, los subsistemas trabajan correctamente en conjunto.

Subsistema de Compilador	
Prueba	**Resultado Esperado**
Error léxico	El sistema muestra por consola los errores léxicos
Prueba	**Resultado Esperado**
Error sintáctico	El sistema muestra por consola los errores sintácticos
Prueba	**Resultado Esperado**
Error de código	El sistema muestra por consola los errores de código
Prueba	**Resultado Esperado**

Error léxico y sintáctico	El sistema muestra por consola los errores léxicos
Prueba	**Resultado Esperado**
Error léxico y de código	El sistema muestra por consola los errores léxicos
Prueba	**Resultado Esperado**
Error sintáctico y de código	El sistema muestra por consola los errores sintácticos
Prueba	**Resultado Esperado**
Los tres errores a la vez	El sistema muestra por consola los errores léxicos
Prueba	**Resultado Esperado**
Sin errores	El sistema muestra los mensajes programados por el usuario

Tabla 5.41 Test de integración de subsistema de compilador

Subsistema de representación gráfica	
Prueba	**Resultado Esperado**
Movimiento de teclado	El robot se mueve en la dirección que pretende el usuario
Prueba	**Resultado Esperado**
Movimiento de código	El robot se mueve según lo esperado en el código introducido
Prueba	**Resultado Esperado**
Cambio de robot	El robot escogido se muestra una vez se pulsa sobre ejecutar

Tabla 5.42 Test de integración de subsistema de representación gráfica

Subsistema de consola	
Prueba	**Resultado Esperado**
Se escribe una sentencia print en el código	Se muestra el mensaje escrito en el print
Prueba	**Resultado Esperado**
Se escribe un bucle while (advertencia)	Se muestra una advertencia sobre el uso
Prueba	**Resultado Esperado**
Se escribe un error de código	Se muestra el error por consola

Tabla 5.43 Test de integración de subsistema de consola

Subsistema de librerías	
Prueba	**Resultado Esperado**
Se prueba un método implementado	El método probado funciona correctamente
Prueba	**Resultado Esperado**
Se prueba un método sin implementar	Se muestra una advertencia por consola

Tabla 5.44 Test de integración de subsistema de librerías

5.7.3 Pruebas de usabilidad

En este apartado se muestran las pruebas de usabilidad, que son las siguientes:

Pruebas de usabilidad	
Prueba	**Resultado Esperado**
Compilar código	1. Abrir la aplicación 2. Escribir un sketch que no compile 3. Comprobar que se muestren los errores por consola 4. Escribir un sketch que compile 5. Comprobar que se muestre que se compila correctamente por consola
Prueba	**Resultado Esperado**
Ejecutar código	1. Escribir un sketch que no compile 2. Comprobar que no se ejecute el sketch 3. Escribir un sketch que compile 4. Comprobar que se ejecuta el sketch 5. Escribir alguna sentencia print 6. Comprobar que se muestra salida por consola
Prueba	**Resultado Esperado**
Ver ayuda	1. Pinchar el menú ayuda 2. Pinchar el botón de ayuda 3. Comprobar que se muestra la ayuda 4. Navegar a cualquier página aleatoria 5. Comprobar que explica la funcionalidad correspondiente
Prueba	**Resultado Esperado**
Editar código	1. Escribir código 2. Copiar una parte del código 3. Pegarlo en otra línea del editor 4. Cortar una parte del código 5. Pegarlo en otra línea del editor 6. Deshacer la última acción 7. Rehacer la acción deshecha

Tabla 5.45 Pruebas de usabilidad

Capítulo 6. Diseño del Sistema

En este apartado se describirá el diseño del sistema que se ha obtenido a partir del análisis realizado en el anterior apartado. Esto se realizará a través de diferentes diagramas: de paquetes, de componentes, de despliegue, de interacción, de actividades, etc.

Por último, también se mostrará la especificación técnica del plan de pruebas a realizar.

6.1 Arquitectura del Sistema

En este subapartado se mostrarán los diagramas relacionados con la arquitectura del sistema, empezando por el de paquetes, siguiendo por el de componentes y terminando con el de despliegue.

6.1.1 Diagramas de Paquetes

A continuación, se muestra el diagrama de paquetes del proyecto, que representa la estructura lógica del sistema, no la física. Esto es debido a que Python tiene tres elementos (paquetes, módulos y clases) que influyen a dicha estructura.

En algunos paquetes de Python, los módulos se corresponderán con una clase, mientras que en otros se corresponderán con una colección de clases y, por ello, se corresponderían con un paquete en otros lenguajes de programación (como Java). A su vez, ciertos módulos no contienen ninguna clase, sino que son solo funciones. Estos cuentan como paquete en el diagrama, aunque luego tengan su representación propia dentro del diagrama de clases. Otros módulos sí que se corresponderán con solo una clase, en cuyo caso, no serán paquetes dentro del diagrama.

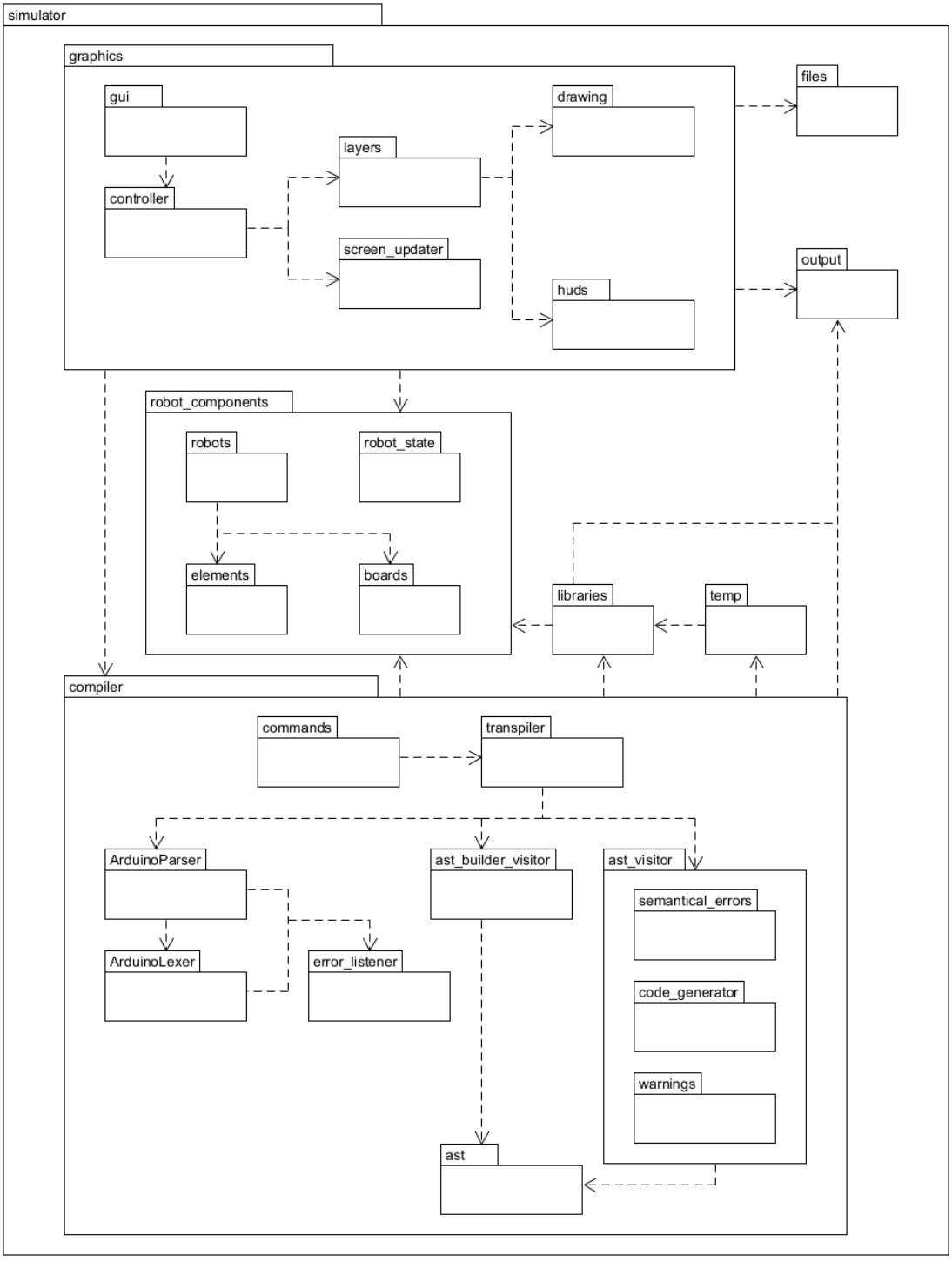


Figura 6.1 Diagrama de paquetes del sistema

Como se puede ver en el diagrama, el sistema está formado por un paquete único que luego se subdivide en 7 paquetes más pequeños: “files”, “output”, “temp”, “libraries”, “graphics”, “robot_components” y “compiler”. Estos tres últimos se subdividen a su vez en varios paquetes, que serán explicados a continuación, junto con los otros paquetes que no tienen subdivisiones:

6.1.1.1 Paquete files

Este paquete contiene el acceso a archivos, bien sean de código (.ino) o bien sean de configuración de la aplicación (.json), conteniendo estos últimos la información de configuración de los robots.

6.1.1.2 Paquete output

Contiene toda la parte de la salida por consola de la aplicación, así como la generación de los logs dicha salida para su posterior análisis (para errores del usuario o incluso fallos en la aplicación propiamente dicha).

6.1.1.3 Paquete libraries

Este paquete contiene la implementación de los diferentes métodos de las librerías de Arduino que son usadas en la programación de los robots del sistema. En concreto String, Servo, Serial y la librería Standard.

6.1.1.4 Paquete temp

Este es un paquete especial. Es generado una vez se traduce el código de Arduino a Python, y contiene el sketch listo para ejecutarse, una vez sea importado por la clase encargada de ello. En resumen, este paquete será el que contenga el resultado de la compilación.

6.1.1.5 Paquete graphics

Se trata del paquete que contiene el *front-end* de la aplicación y la animación del robot, así como los cambios de estado de este que dependan de su posición y contexto dentro de la propia animación (por ejemplo, si está sobre la pista o no, o sí tiene en frente un obstáculo o no). Se subdivide en los siguientes paquetes.

6.1.1.5.1 Paquete gui

Paquete que contiene todo lo relacionado con la representación de la ventana, es decir, de la interfaz gráfica. Interactúa con el paquete controlador de manera continua, por tanto, son paquetes muy relacionados entre sí.

6.1.1.5.2 Paquete controller

Este paquete contiene la lógica encargada de realizar la comunicación entre el paquete gui y el resto de los paquetes que interactúan con el "estado gráfico" del robot, es decir, el comportamiento de la animación del robot.

6.1.1.5.3 Paquete layers

Se encarga de realizar las operaciones de movimiento y de interacción con el entorno del robot. Es decir, contiene toda la lógica encargada de mover el robot, así como de proveer el estado correcto a los diferentes elementos que contienen el robot.

6.1.1.5.4 Paquete screen_updater

Este paquete contiene la lógica que anima el movimiento del robot. Se trata de un módulo sin clases.

6.1.1.5.5 Paquete drawing

Contiene la lógica relacionada con el dibujado del robot (no confundir con animación), es decir, la lógica que realiza el dibujado de las diferentes formas necesarias para mostrar el robot y su entorno.

6.1.1.5.6 Paquete huds

Contiene las diferentes clases que realizan la función de *Heads Up Display*, para mostrar datos del robot y como se está comportando en la ejecución.

6.1.1.6 Paquete robot_components

Este paquete contiene otros paquetes más pequeños que se encargan de manejar el estado del robot, en este caso aquel estado más cercano a Arduino: que leen los sensores, como se están moviendo los servos, si los botones están pulsados, etc.

Esta información será la que usarán más tarde los métodos del paquete “libraries”.

6.1.1.6.1 Paquete robots

Contiene todo lo relacionado con los robots y su estructura. Es decir, las clases que representan el robot como un conjunto de elementos conectados a una placa de Arduino. Permite asignar y desasignar los pines de los elementos.

6.1.1.6.2 Paquete robot_state

Este es un paquete extraño, solo se usa para los *delays* programados en el sketch y para finalizar la ejecución en caso de que sea finalizada mediante la función de Arduino empleada para ello. No obstante, su comportamiento podría ser expandido en el futuro, además de estar relacionado con la placa del robot en el sentido de que contiene información sobre lo que esta debe “ejecutar”.

6.1.1.6.3 Paquete elements

Contiene la lógica de los elementos empleados para construir los robots: servos, sensores de luz y de ultrasonidos, botones y joystick. Dicha lógica puede ser cambiar el valor que leen o que reciben dichos elementos.

6.1.1.6.4 Paquete boards

Contiene la lógica de las placas de Arduino empleadas en los robots. Permitirá el cambio de pines, condicionado a que sean del tipo correcto una vez sean asignados, así como la lectura de valores de los elementos.

6.1.1.7 Paquete compiler

Este paquete contiene la parte del compilador dentro de lo que se considera *back-end* de la aplicación. Se divide en los siguientes paquetes

6.1.1.7.1 Paquete commands

Encargado de ejecutar las diferentes acciones del compilador, es decir, compilar, ejecutar el método “setup” y ejecutar continuamente el método “loop”.

6.1.1.7.2 Paquete transpiler

Este paquete es el compilador (transpilador, ya que traduce el código a otro lenguaje) propiamente dicho. Se encarga de llamar al resto de paquetes que realizan las diferentes fases de la compilación.

6.1.1.7.3 Paquete ArduinoParser

Junto con “ArduinoLexer”, genera el *parse tree* que produce el sketch. Se encarga de los errores sintácticos. Es generado por ANTLR4

6.1.1.7.4 Paquete ArduinoLexer

Se encarga de los errores léxicos. Es generado por ANTLR4

6.1.1.7.5 Paquete error_listener

Edita el texto de la salida de errores dada por los dos paquetes mencionados en los anteriores subapartados.

6.1.1.7.6 Paquete ast_bulder_visitor

Este paquete contiene la lógica que transforma el *parse tree* en un AST propiamente dicho.

6.1.1.7.7 Paquete ast

Contiene todas las clases nodo del AST, con sus características correspondientes que permiten las fases de análisis semántico y posteriores.

6.1.1.7.8 Paquete ast_visitor

Este paquete contiene la definición de la clase “visitor” base que recorrerá las diferentes clases del paquete “ast”. Se divide en varios paquetes, cada uno se corresponde con una fase.

6.1.1.7.8.1 Paquete semantical_errors

Contiene la lógica necesaria para la realización del análisis semántico del sketch proporcionado por el usuario.

6.1.1.7.8.2 Paquete code_generator

Contiene la lógica para realizar la generación del código Python que será ejecutado posteriormente para realizar la simulación.

6.1.1.7.8.3 Paquete warnings

Contiene la lógica que permite la comunicación de advertencias de implementación al usuario de la aplicación.

6.1.2 Diagramas de Componentes

A continuación, se muestra el diagrama de componentes de la aplicación:

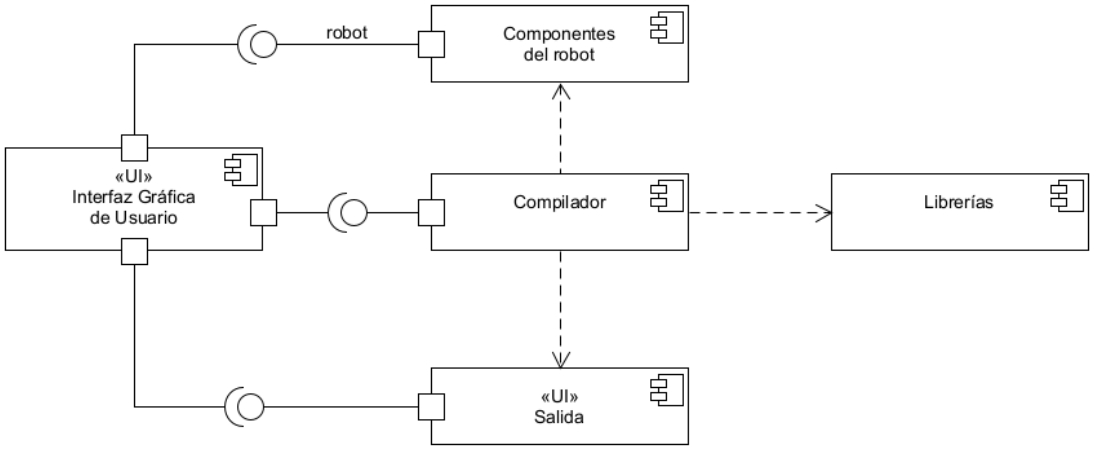


Figura 6.2 Diagrama de componentes de la aplicación

La aplicación está formada por 5 componentes: la Interfaz gráfica de usuario, con la que interactúa el usuario del sistema para programar, ejecutar la simulación, etc.; los componentes de robot, que mantienen el estado de esos componentes: en que pin están, lo que leen, etc.

Luego, está la Salida, que se encarga de la salida de texto, advertencias y errores, tanto por consola como por log. El compilador, que se encarga de la compilación, análisis y generación de código.

Por último, el último componente es del de las Librerías, que se encarga de las operaciones que se pueden realizar en Arduino.

6.1.3 Diagramas de Despliegue

El siguiente el diagrama de despliegue de la aplicación:

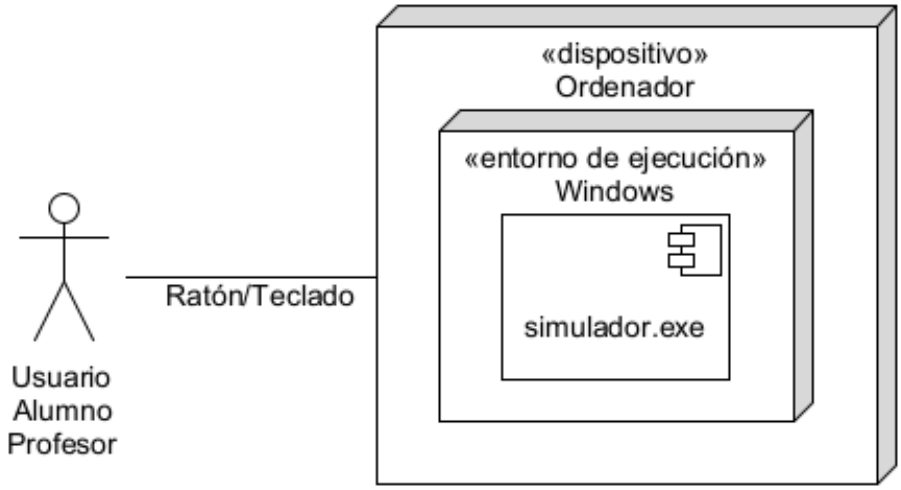


Figura 6.3 Diagrama de despliegue del sistema

Como se puede ver, se expone que la aplicación es *standalone*, es decir, no requiere de servicios externos para funcionar. El Ordenador no requiere de demasiados requisitos. Debe de ser un PC preferiblemente, ya que la funcionalidad de la interfaz gráfica se puede ver afectada si se usa otro dispositivo. El entorno de ejecución podrá ser Windows 10 (no iOS, al no tener uno no puedo proporcionar una aplicación que funcione; ni Ubuntu, por problemas de compatibilidad de ciertas librerías con el sistema operativo).

6.2 Diseño de Clases

6.2.1 Diagrama de clases

A continuación, se va a mostrar un diagrama general del sistema, que hace hincapié en las relaciones entre las clases que lo componen. Después, se desglosarán las clases en mayor profundidad, para mostrar sus métodos y sus atributos.

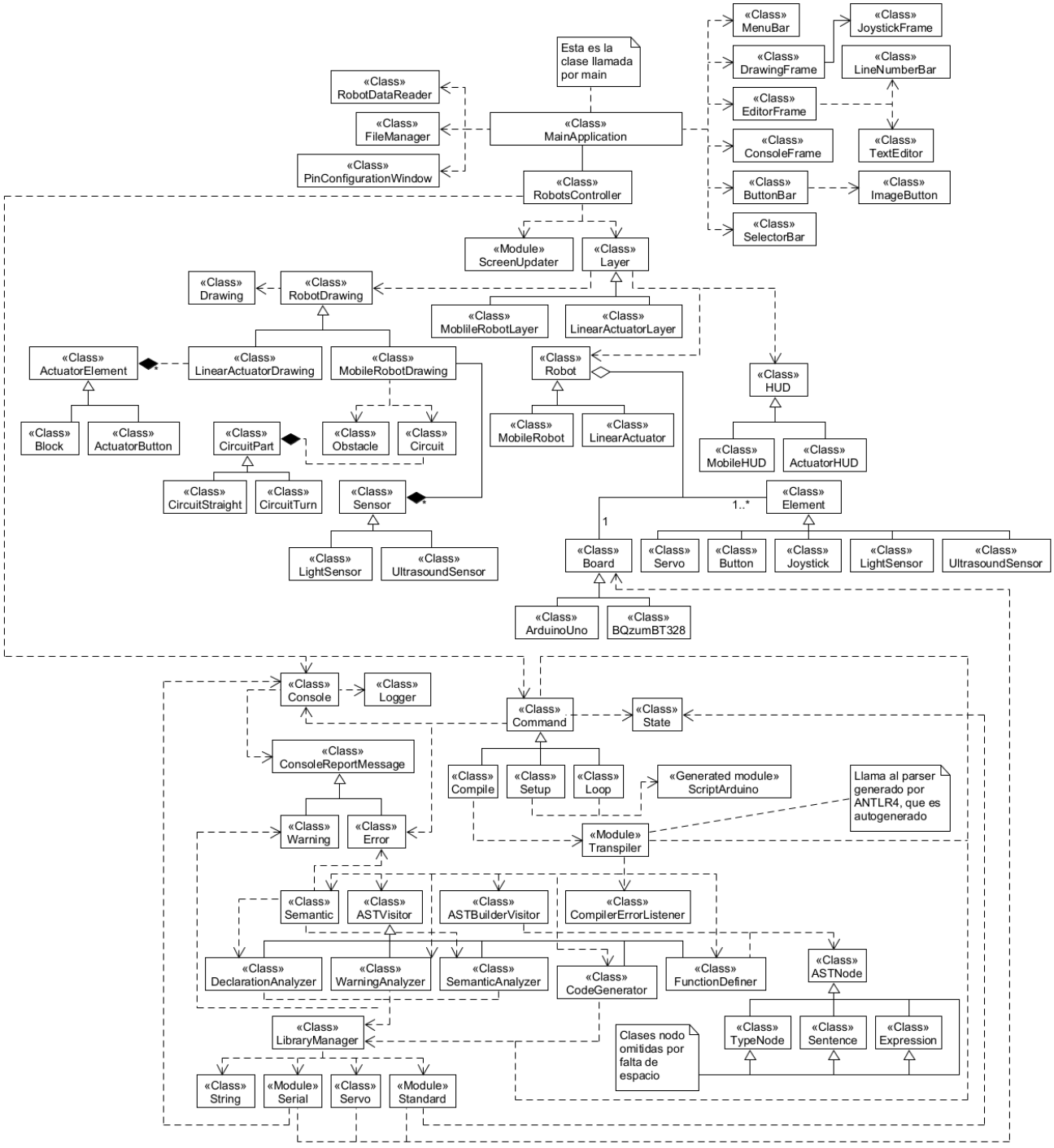


Figura 6.4 Diagrama de clases simplificado (relaciones)

Los diagramas de clase han sido generados por ingeniería inversa, es decir, son el resultado final del proyecto. Sin embargo, esto no es del todo cierto, al tener que haber realizado ciertos arreglos no planificados debido a las limitaciones de Python y el *framework* de interfaz gráfica empleado. Estos problemas serán explicados más adelante en el apartado 7.5.1.

En resumidas cuentas, la "clase main" (en Python la llamada a "main" es diferente a otros lenguajes) es "MainApplication". Esta llama a "RobotsController" para las instrucciones que requieren de clases externas a la interfaz gráfica. La clase "RobotsController" es la que llama a "ScreenUpdater" para animar el robot y a "Layer" para actualizarlo en cada movimiento.

A su vez, "Layer" debe mantener actualizados los "Drawing", "HUD" y "Robot" que le correspondan (dependiendo de si es un robot móvil o un actuador lineal). Las clases derivadas de "Drawing" delegan ciertos comportamientos a otras clases:

- En el caso de "LinearActuatorDrawing", delega el comportamiento de sus botones y otros elementos a clases "ActuatorElement", controlando cada una de ellas el componente homónimo.
- En el caso de "MobileRobotDrawing", similarmente, se delega el comportamiento de los sensores a las clases Sensor. Se añade que hay dos elementos más que afectan al dibujo de este, los obstáculos ("Obstacle") y los circuitos ("Circuit" y las clases de que se compone, "CircuitParts").

Las clases derivadas de "Robot" se refieren a una implementación concreta del estado en que estarían los robots compuestos por la placa y los diferentes elementos. Cada robot estará compuesto de los elementos ("Element") correspondientes:

- Robot móvil ("MobileRobot"): "Servo", "LightSensor" y "UltrasoundSensor".
- Actuador lineal ("LinearActuator"): "Servo", "Button", "Joystick".

A su vez, cada uno de esos robots solo tendrá una placa de Arduino (la Arduino para el actuador y la BQ para el robot móvil). Es a través de esta clase que se realiza la comunicación entre librerías ("String", "Serial", "Servo" y "Standard") y elementos.

Dichas librerías implementan los métodos de la documentación de Arduino **[ArduinoRef].**

El encargado de compilar ("Compile") y ejecutar los métodos "setup" y "loop" ("Setup" y "Loop"), serán las clases que heredan de "Command".

En el caso de "Setup" y "Loop", llaman al módulo generado por el compilador ("ScriptArduino"), mientras que en el caso de "Compile", llaman al método "transpile" de "Transpiler".

Este último módulo coordina todas las fases de compilación, desde el análisis léxico y sintáctico realizado por las clases generadas por ANTLR4, pasando por la generación del AST realizada por la clase "ASTBuilderVisitor", continuando con el análisis semántico realizado por la clase "Semantic", que llama a las clases "DeclararationAnalyzer" (que se encarga de comprobar si las variables y funciones se definen) y la clase "SemanticAnalyzer", que realiza el análisis semántico de tipos y otros errores relacionados con la semántica. Los dos últimos

pasos ya serían la generación de código (“CodeGenerator”) y de advertencias (“WarningAnalyzer”).

Las últimas 4 clases mencionadas forman parte del patrón “visitor”, visitando la estructura de clases nodo proporcionada por las clases que heredan de “ASTNode”.

Por último, los comandos acceden a “State” para comprobar si el código se ha terminado de ejecutar o si se está en medio de un “delay”. A su vez, toda la salida se realiza a través de “Console”, que permite que se le pase texto simple, errores (“Error”) o advertencias (“Warning”). Esta salida se escribe a un log (“Logger”).

6.2.2 Desglose de las clases

En este subapartado se desglosarán las clases por paquete, mostrando los métodos y variables correspondientes a cada una y, en caso de que sea necesario, explicando su funcionamiento interno.

Hay que destacar que las variables o métodos no tendrán en muchos casos un tipo establecido. Esto es porque Python tiene tipado dinámico y, aunque esté mezclado con tipado fuerte, no es que la variable se deba declarar de un tipo en su definición (como en Java, que declaramos una variable como tipo nombre = valor).

Además, no se declarará el tipo de las variables, ya que en Python no es que exista como tal los métodos privados o protegidos, más bien aquellos que necesiten esa protección comenzarán por “__” y “_” respectivamente, aunque esto no impide que sean accedidas desde una clase externa si el programador se empeña. En resumidas cuentas, no tiene sentido declarar su nivel de protección.

6.2.2.1 Paquete compiler

6.2.2.1.1 Paquete commands

La figura 6.5 representa las clases del paquete “commands”, que son “Command” de la que heredan “Compile”, “Setup” y “Loop”. Implementa el patrón de diseño “command”, con un método “execute”.

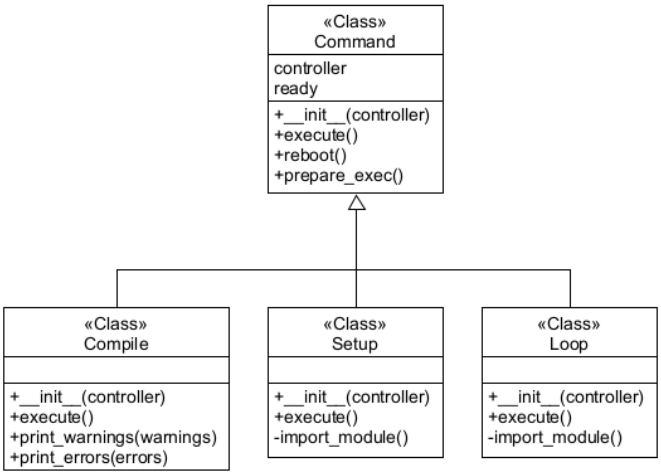


Figura 6.5 Clases del paquete commands

6.2.2.1.2 Paquete transpiler

La Figura 6.6 representa el módulo del paquete "transpiler". El module quiere decir que es solo un método el que está instanciado en el paquete, es decir, que no está asociado a ninguna clase. Se encarga de coordinar la fase de compilación del código.

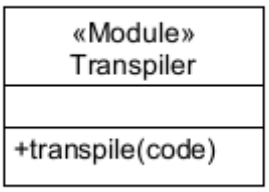


Figura 6.6 Módulo del paquete transpiler

6.2.2.1.3 Paquete ArduinoParser, ArduinoLexer y error_listener

Estas clases se ha decidido no incluirlas en el diagrama, las dos primeras están generadas automáticamente por ANTLR4 y no se trabaja directamente con ellas más que para generar el árbol sintáctico y/o los errores de código.

Respecto a "error_listener", simplemente traduce los errores por defecto de ANTLR4 al formato escogido para el proyecto, no es muy relevante.

6.2.2.1.4 Paquete ast_builder_visitor

Esta clase hereda del "visitor" generado por ANTLR4, y es la que realiza la conversión de *parse tree* a AST. Para ello, emplea el patrón "visitor", visitando la estructura en árbol de las clases que componen el *parse tree* de ANTLR4.

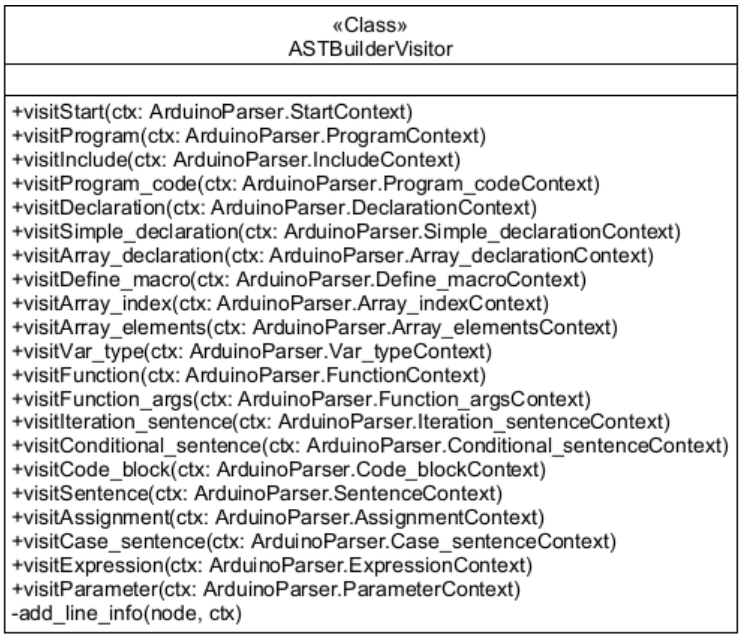


Figura 6.7 Clase del paquete ast_builder_visitor

6.2.2.1.5 Paquete ast

Este paquete es el que más clases contiene del sistema en general. Se trata de la estructura que forma el arbol sintáctico del lenguaje de programación en cuestión. Son parte del patrón visitor, en el sentido de que son las clases nodo del árbol y, por ello, las que implementan el método accept. Para abreviar, se mostrarán las clases más relevantes y aquellas que sean prototipo para otras.

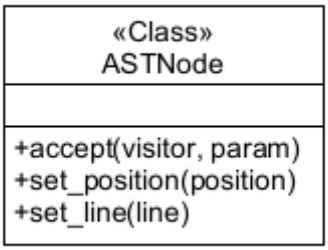


Figura 6.8 Clase base del AST, heredan todas de ella

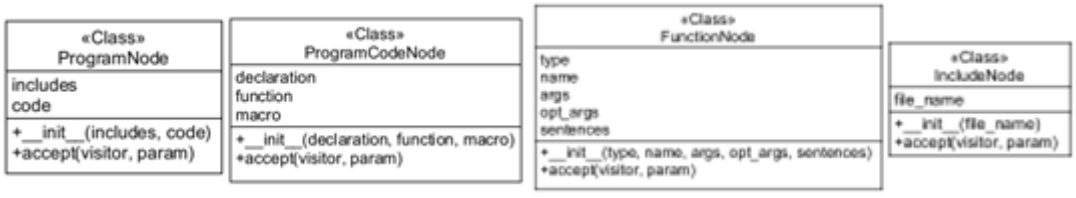


Figura 6.9 Clases que heredan directamente de ASTNode y no heredan de otro nodo

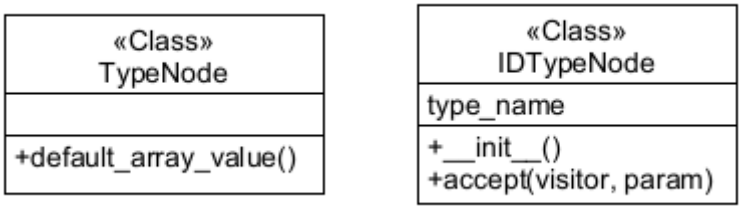


Figura 6.10 Clase TypeNode y subclase de esta IDTypeNode

Respecto a las clases de la Figura 6.10, aquellas que heredan de "TypeNode" son todas iguales, excepto "IDTypeNode", que contiene el nombre del tipo. El resto de los tipos son aquellos que tiene la librería standard de Arduino.

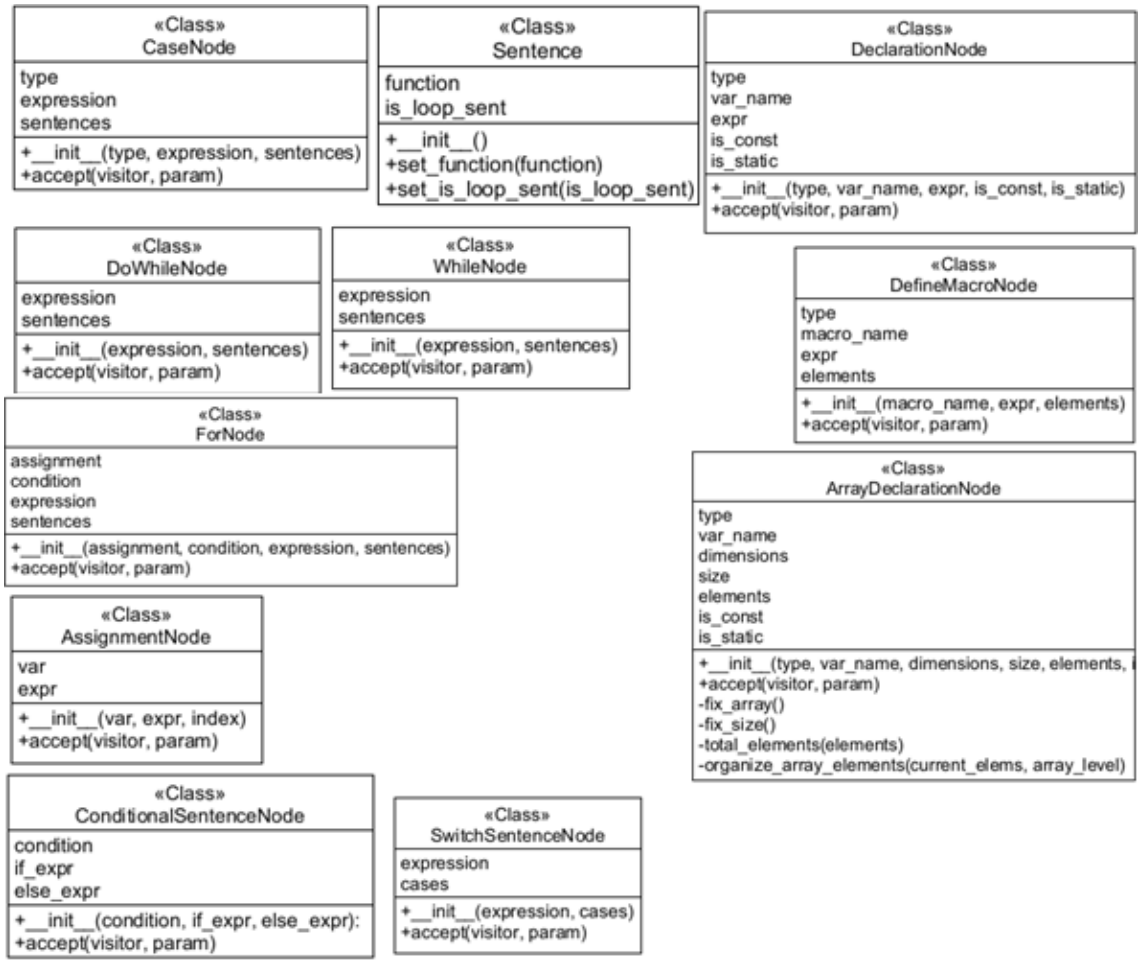


Figura 6.11 Clase Sentence y subclases que heredan de ella

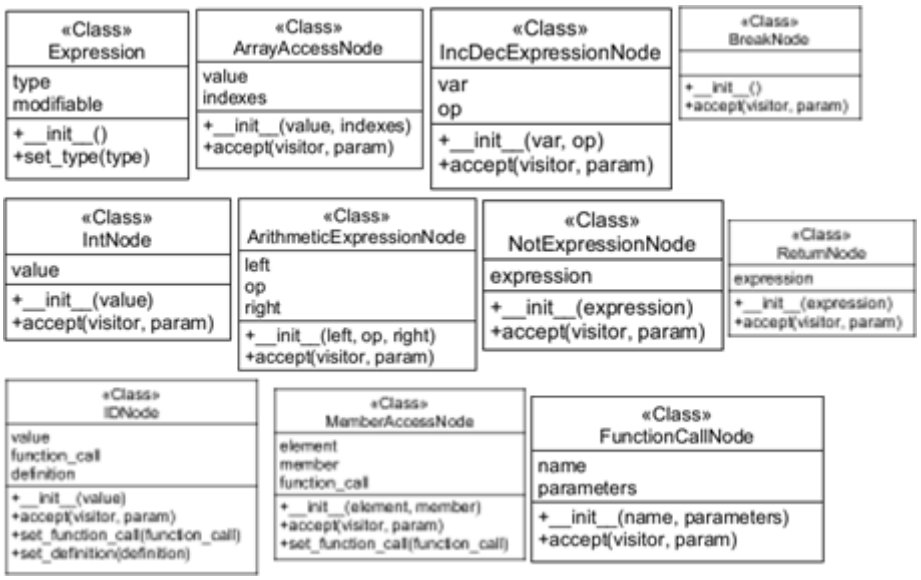


Figura 6.12 Clase Expression y subclases que heredan de ella

Hay varias expresiones que se repiten en cómo se estructura la clase. "ArithmeticExpressionNode" es exactamente igual que otras operaciones como comparación o expresiones booleanas. "NotExpressionNode" es idéntica a su correspondiente en su versión de operaciones con bits. "BreakNode" es igual a "ContinueNode". "IntNode" es idéntica al resto de expresiones correspondientes a un tipo como "String" o "bool", salvo "IDNode" que está presente en la figura.

6.2.2.1.6 Paquete ast_visitor, semantical_errors, code_generator y warnings

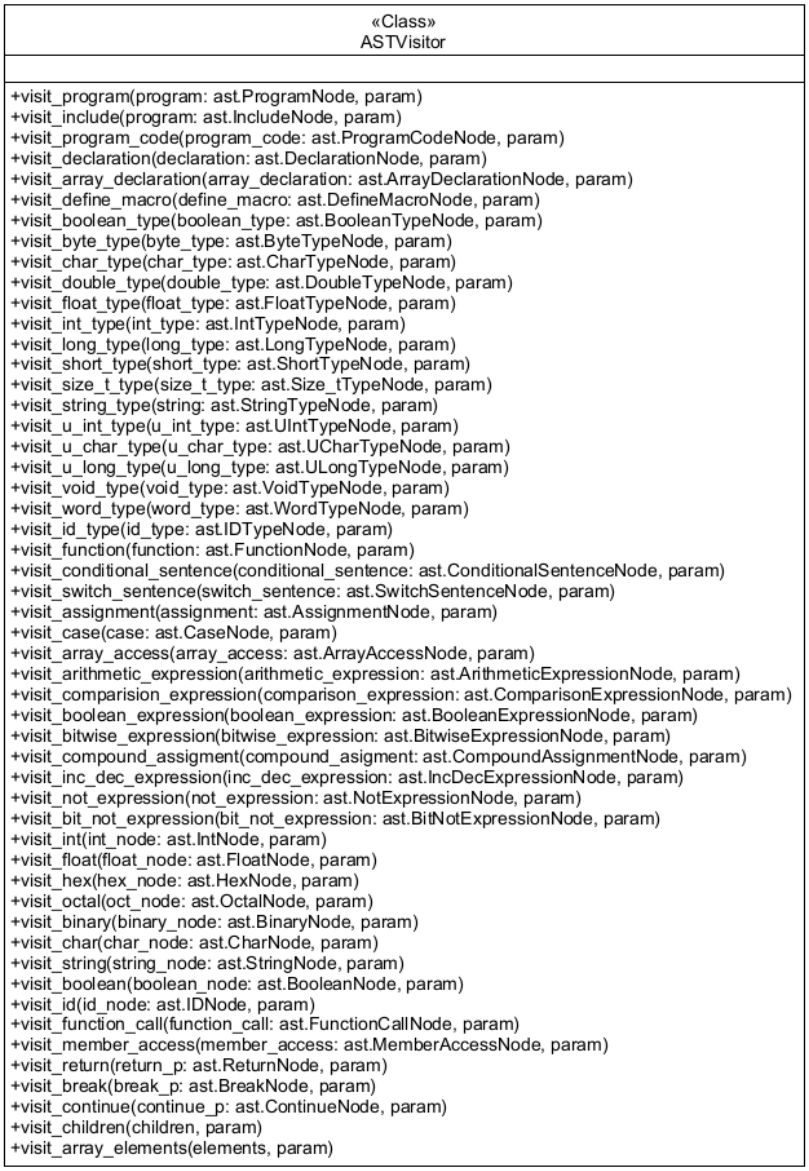


Figura 6.13 Clase ASTVisitor

De la clase de la Figura 6.13 heredan las clases: “DeclarationAnalyzer”, “SemanticAnalyzer”, “WarningAnalyzer”, “CodeGenerator” y “FunctionDefiner”. Estas clases son las encargadas del análisis semántico, la generación de código y la generación de advertencias.

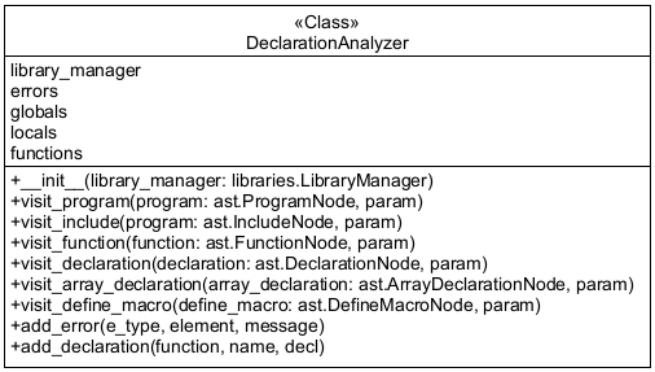


Figura 6.14 Clase DeclarationAnalyzer

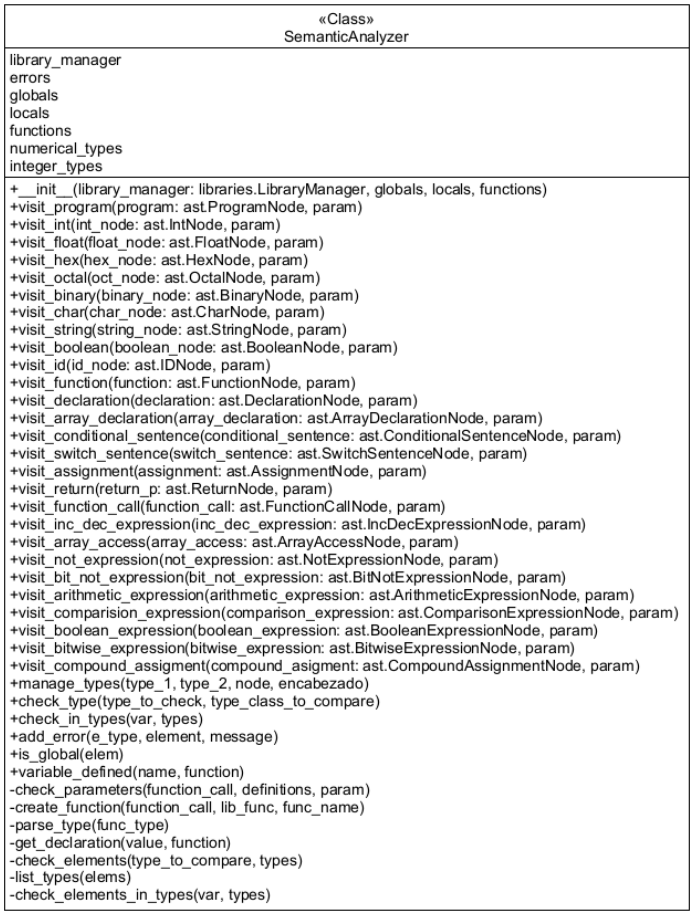


Figura 6.15 Clase SemanticAnalyzer

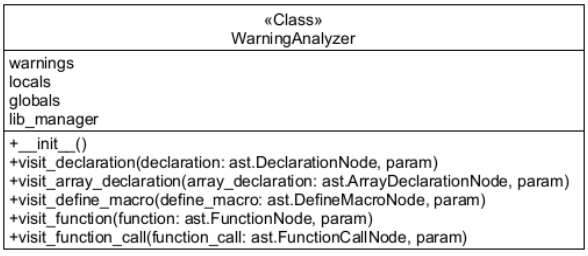


Figura 6.16 Clase WarningAnalyzer

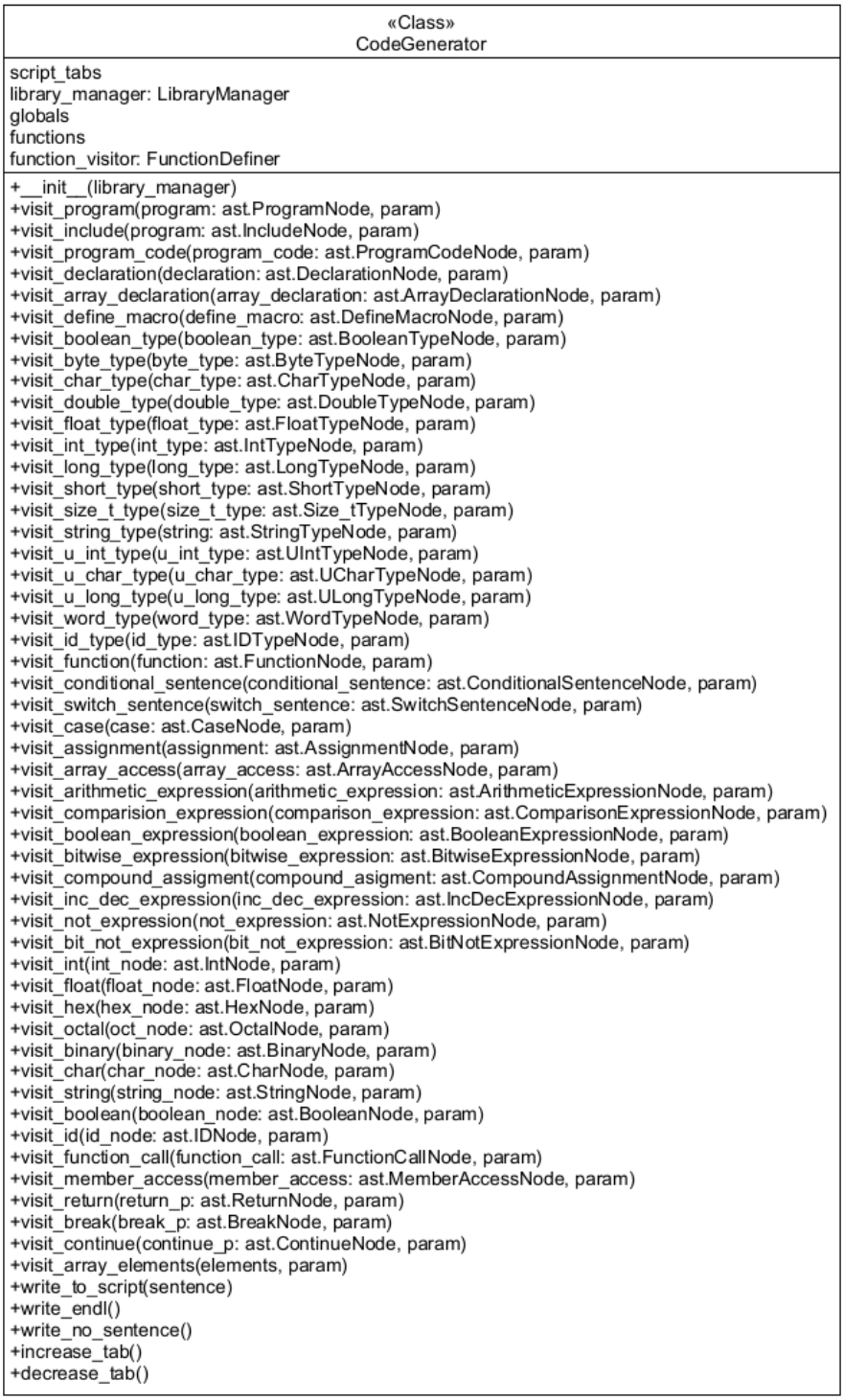


Figura 6.17 Clase CodeGenerator

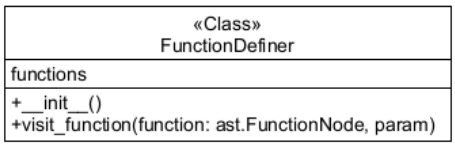


Figura 6.18 Clase FunctionDefiner

6.2.2.2 Paquete files

Las clases de este paquete se muestran en la Figura 6.19. “RobotDataReader” se encarga de leer la configuración de los robots del archivo JSON, mientras “FileManager” se encarga de abrir y guardar los sketches.

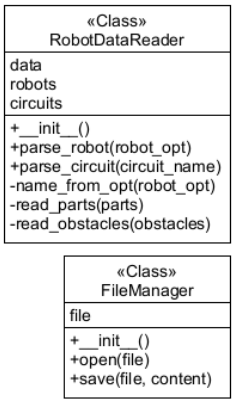


Figura 6.19 Clase del paquete files

6.2.2.3 Paquete graphics

6.2.2.3.1 Paquete gui

La clase “MainApplication” mostrada en la figura 6.20 es aquella que gestiona la interfaz gráfica y resulta ser el “main” de la aplicación.

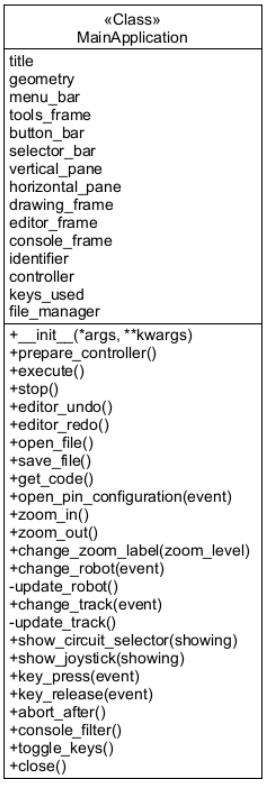


Figura 6.20 Clase MainApplication

6.2.2.3.2 Paquete controller

La clase de la Figura 6.21 se encarga de comunicar la Interfaz Gráfica de Usuario (o vista) con el resto de la lógica de la aplicación. Gestiona la animación y el estado del robot (en el sentido de que llama a las clases en las que delega dichas funciones).

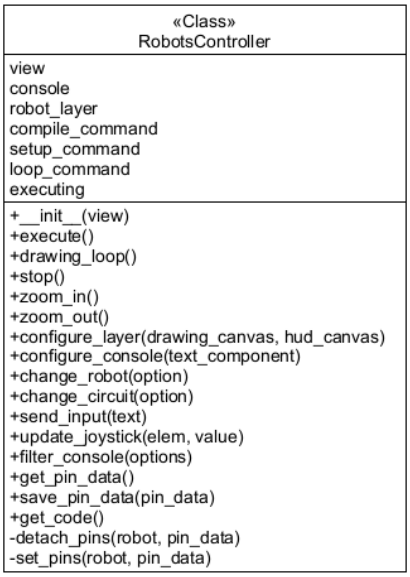


Figura 6.21 Clase RobotsController

6.2.2.3.3 Paquete layers

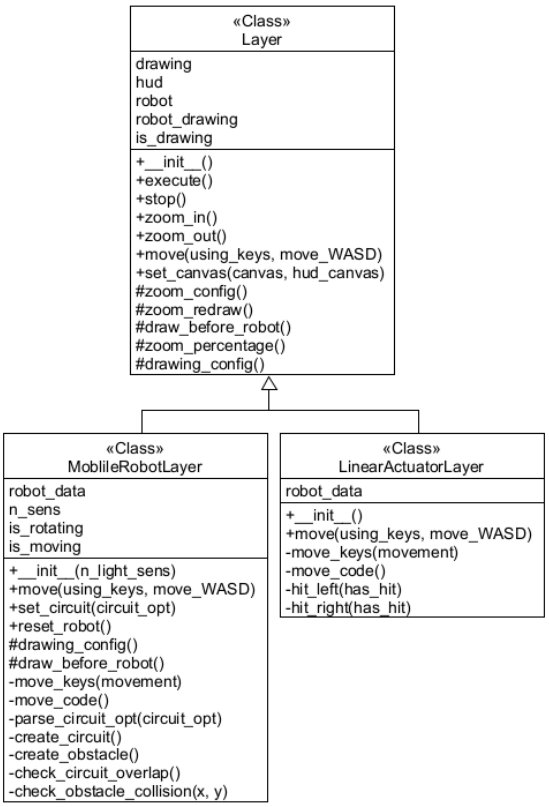


Figura 6.22 Clase Layer y sus subclases

Las clases de la Figura 6.22 se encargan de toda la lógica de movimiento del robot, así como de sus sensores y elementos. También se encarga de llamar a las clases de "drawing", que dibujan los elementos.

6.2.2.3.4 Paquete screen_updater

El módulo de la Figura 6.23 se encarga de animar el movimiento de los robots. Se trata de un método, no una clase.

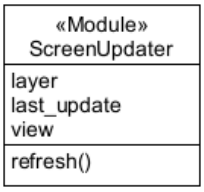


Figura 6.23 Módulo ScreenUpdater

6.2.2.3.5 Paquete drawing

En este paquete se distinguen dos tipos de clase según su función. La primera es la de la Figura 6.24. Se encarga de todos los métodos de dibujado que son necesarios para representar a los robots.

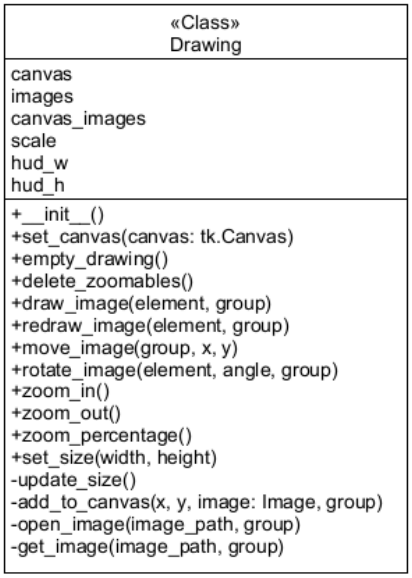


Figura 6.24 Clase Drawing

La segunda son las clases de las Figura 6.25, Figura 6.26, Figura 6.27, Figura 6.28 y Figura 6.29.

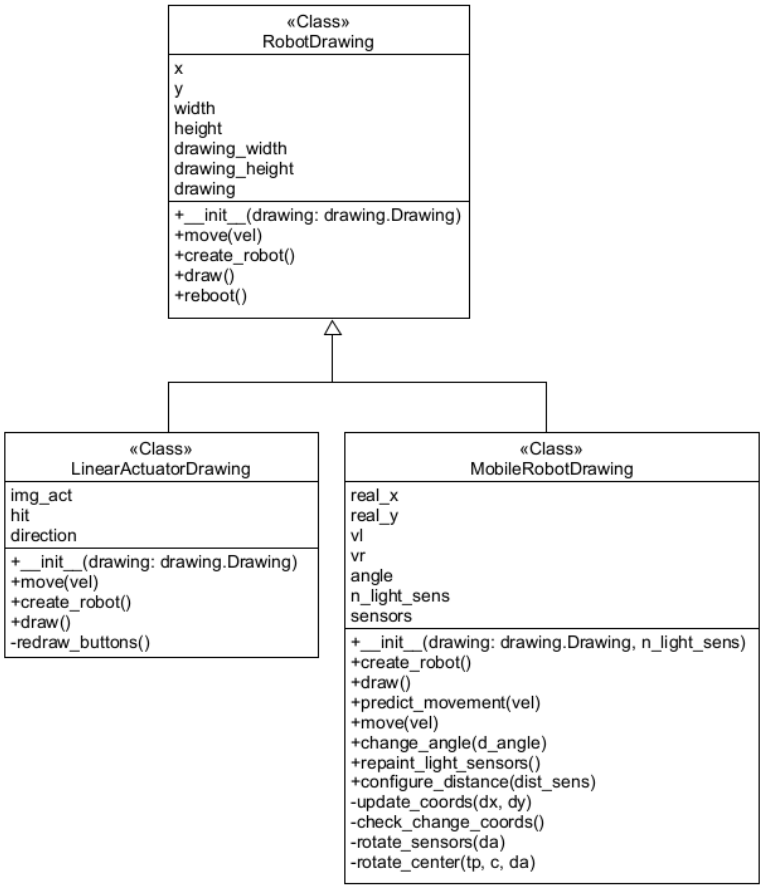


Figura 6.25 Clase Robot Drawing y subclases de esta

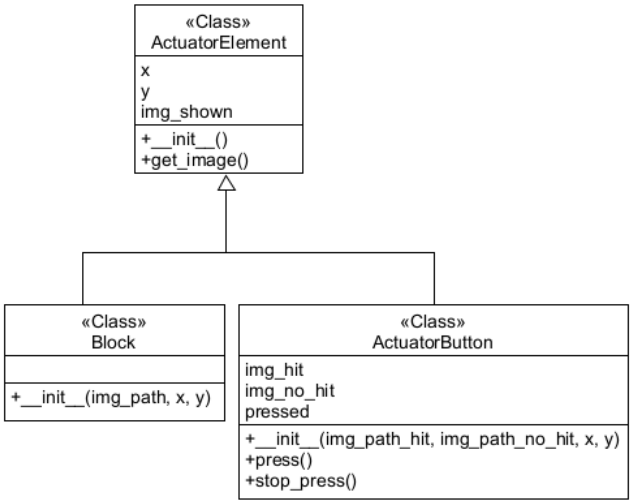


Figura 6.26 Clase ActuatorElement y subclases de esta

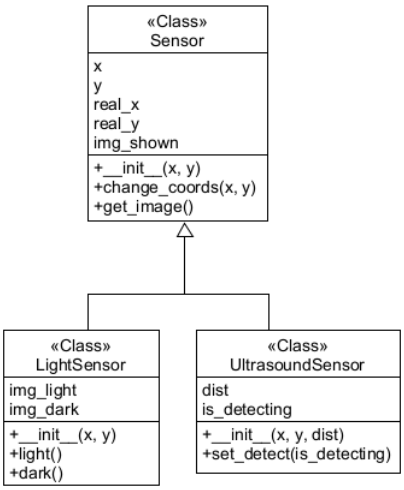


Figura 6.27 Clase Sensor y subclases de esta

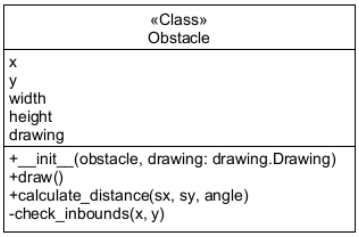


Figura 6.28 Clase Obstacle

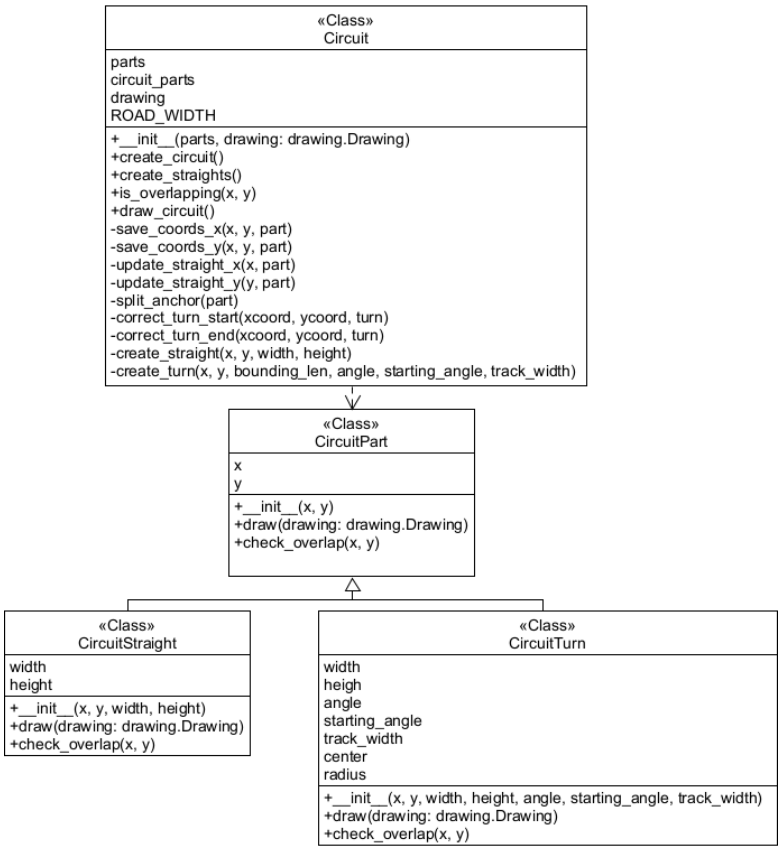


Figura 6.29 Clase Circuit y clases en las que delega

Estas clases se encargan de la parte de gestión de los robots más cercana al dibujo, es decir, si se tiene que mover, aquí es donde se mueve de forma efectiva. Además, contiene los elementos con los que interactúan los robots y que afectan al estado de los elementos propios de estos.

6.2.2.3.6 Paquete huds

Las clases de este paquete muestran la información del estado actual de robot: movimiento de servos, botones pulsados, lectura de sensores, etc.

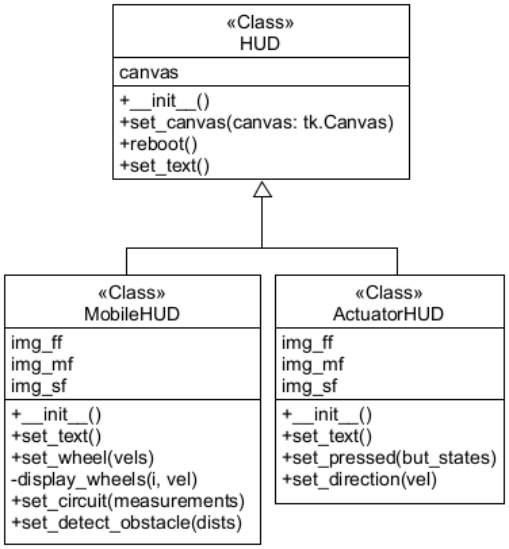


Figura 6.30 Clase HUD y subclases

6.2.2.4 Paquete libraries

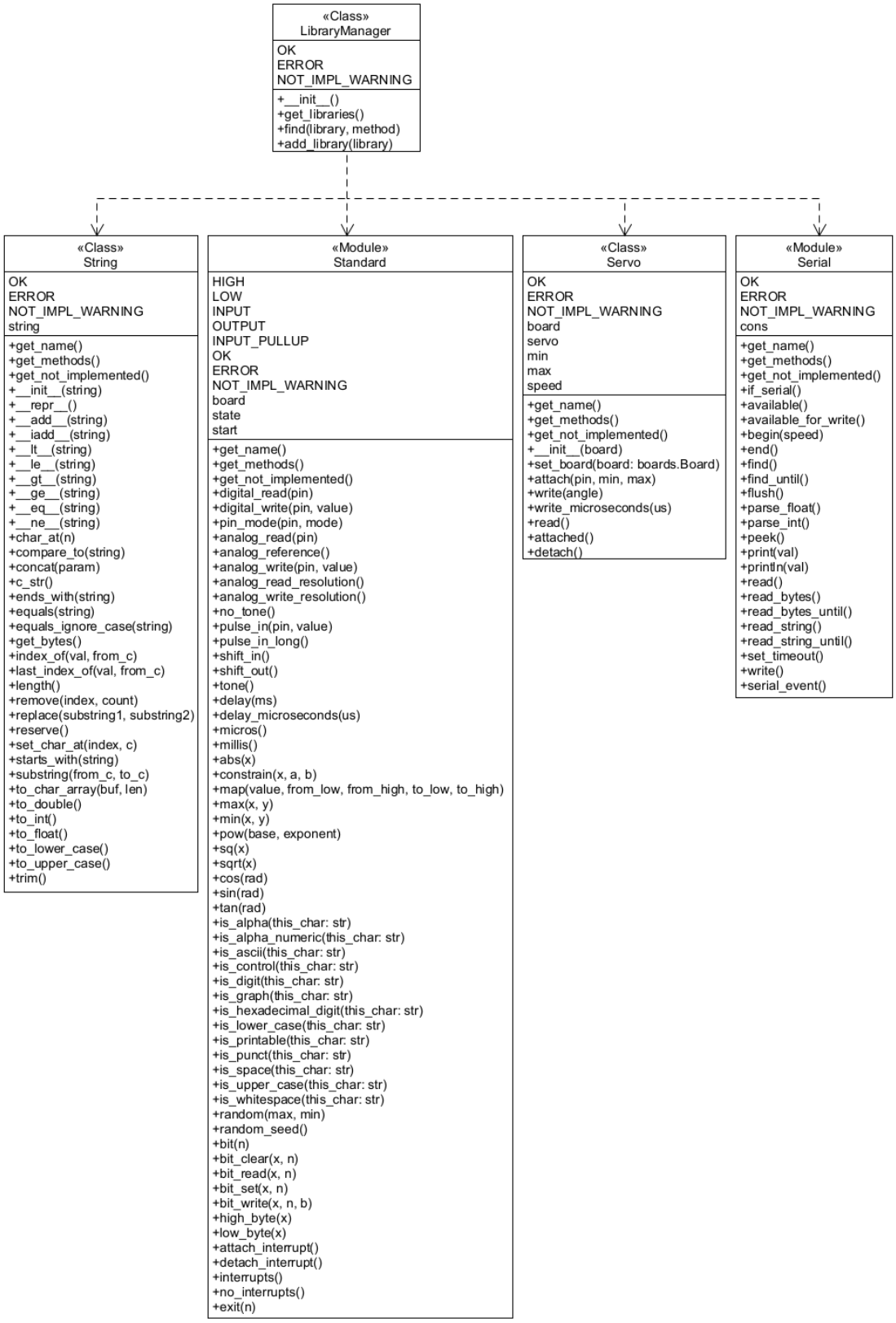


Figura 6.31 Clase LibraryManager y clases/módulos de librerías (String, Standard, Servo y Serial)

Las clases y/o módulos de la Figura 6.31 se encargan de proporcionar los métodos de las librerías de Arduino, de tal manera que el código generado se comporte de una manera coherente con la realidad.

Dichas clases son dependientes de los robots (excepto "String"), ya que pueden actualizar o leer su estado. Por ejemplo, la librería "Standard" puede leer o escribir a pines o "Serial" puede asignarse a un pin.

6.2.2.5 Paquete output

Este paquete se encarga de la salida por consola y de *loggear* dicha salida. Las clases que heredan de "ConsoleReportMessage" son las que almacenan los errores y advertencias, que más tarde son mostradas en las clases "Console" y "Logger" (que es llamada por "Console").

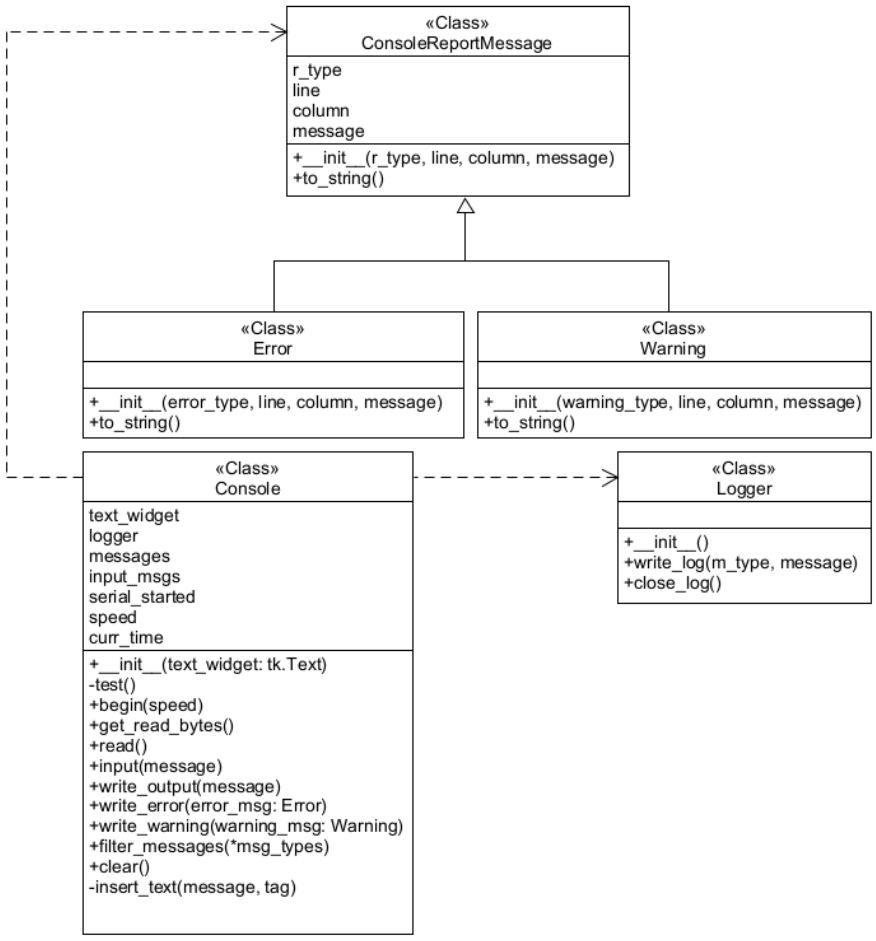


Figura 6.32 Clases del paquete output

6.2.2.6 Paquete robot_components

6.2.2.6.1 Paquete robots

Las clases mostradas en la Figura 6.33 se encargan de gestionar el estado de los robots, es decir, lo que leen los sensores, como giran los servomotores, botones pulsados o no, etc.

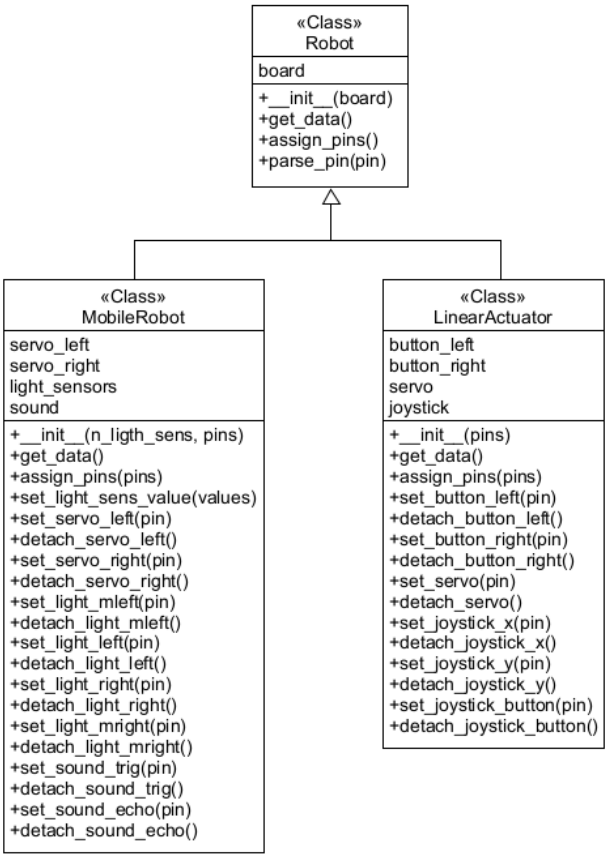


Figura 6.33 Clase Robot y subclases

6.2.2.6.2 Paquete robot_state

Esta clase controla el estado de la ejecución dentro de la placa Arduino. Establece si hay un *delay* o si se ha terminado de ejecutar el código.

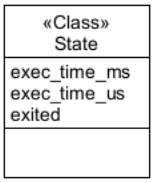


Figura 6.34 Clase State

6.2.2.6.3 Paquete elements

En la Figura 6.35 se muestran las clases de este paquete. Son las clases que representan los elementos de los robots. Contienen el estado que se actualiza mediante las clases del paquete robots (pines, lectura, etc.).

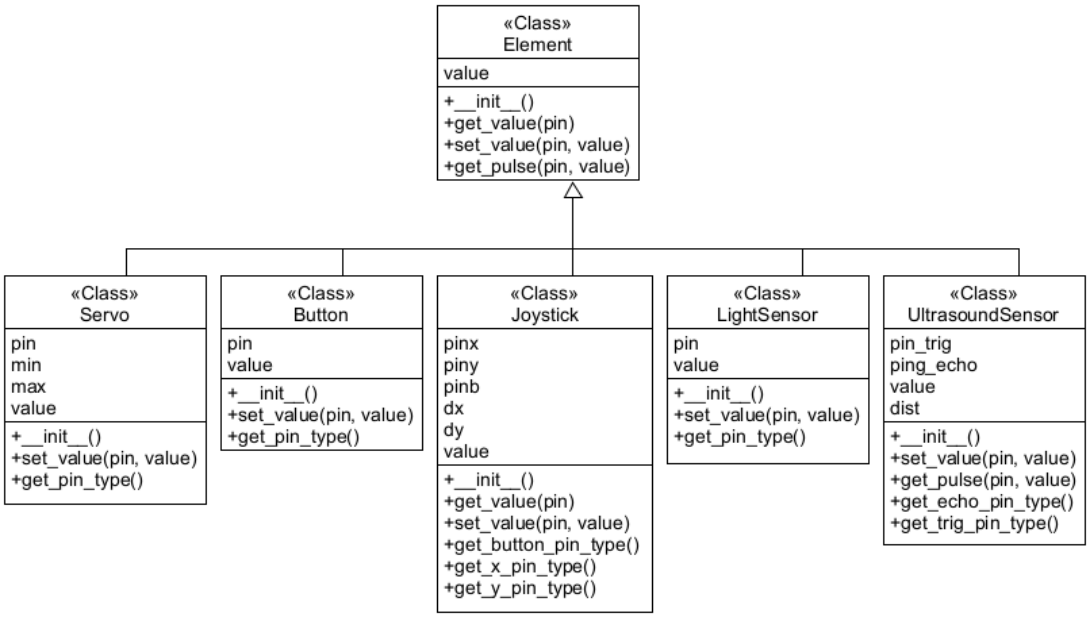


Figura 6.35 Clase Element y subclases

6.2.2.6.4 Paquete boards

La Figura 6.36 muestra la clase "Board" y los diferentes tipos de placa que se usan. Esta clase gestiona todo lo relacionado con los pines y las lecturas y escrituras de valores de los robots (del paquete robots).

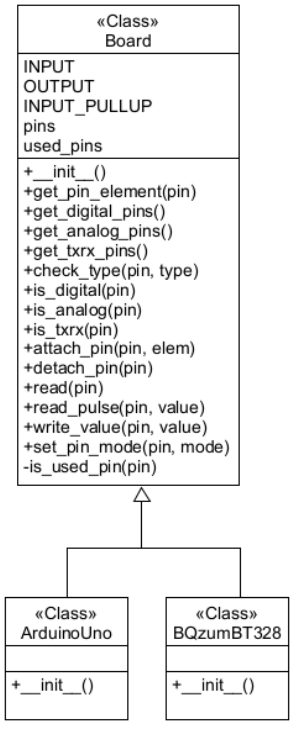


Figura 6.36 Clase Board y subclases

6.2.2.7 Paquete temp

Este es un paquete especial. Contiene el módulo generado que se ejecutará al compilar el código del sketch. Se muestra en la siguiente figura:

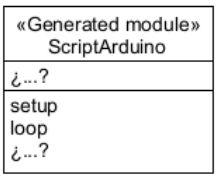


Figura 6.37 Módulo generado ScriptArduino

6.2.3 Patrones de diseño

A la hora de desarrollar el sistema se emplean varios patrones de diseño. Este apartado describirá de forma breve los patrones usados y el porqué de su uso. Para más información sobre los patrones se puede consultar **[Gamma95]** y **[Shvets14]**.

6.2.3.1 Visitor

El patrón "Visitor" representa una operación sobre los elementos de una estructura de objetos (en el caso que nos ocupa, un árbol de objetos nodo), definiendo nuevas operaciones sin necesidad de cambiar las clases sobre las que se opera.

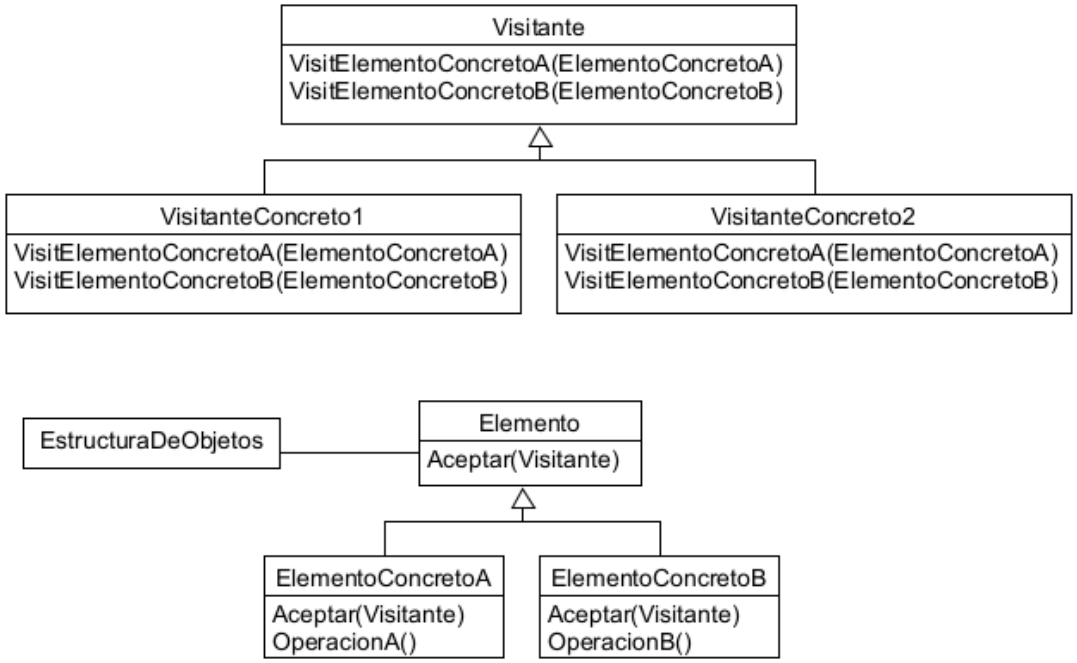


Figura 6.38 Estructura del patrón Visitor

"Visitante" declara una operación "visitar" para cada elemento concreto, que implementan los visitantes concretos; definiendo la operación que se realizará sobre cada elemento concreto. La clase elemento y las que implementan dicha clase definirán una operación "aceptar", que toma el visitante como argumento y que realiza una llamada al método "visit" correspondiente.

Este patrón de diseño se emplea en el paquete "compiler", y más concretamente en las clases "ArduinoVisitor" (de la que implementa "ASTBuilderVisitor"), y "ASTVisitor" (de la que implementan "DeclarationAnalyzer", "WarningAnalyzer", "SemanticAnalyzer", "CodeGenerator" y "FunctionDefiner"). La primera clase ("ASTBuilderVisitor") se encarga de recorrer el *parse tree* y generar el AST y las clases derivadas de "ASTVisitor" se encargan de diferentes fases del compilador ("DeclarationAnalyzer" y "SemanticAnalyzer" del análisis

semántico, “CodeGenerator” y “FunctionDefiner” de la generación de código y “WarningAnalyzer” del análisis de alertas).

6.2.3.2 Command

Este patrón encapsula una petición en un objeto, conteniendo toda la información sobre la petición, permitiendo parametrizar a los clientes con diferentes peticiones y poner en cola su ejecución (además de permitir realizar funcionalidades como deshacer y rehacer).

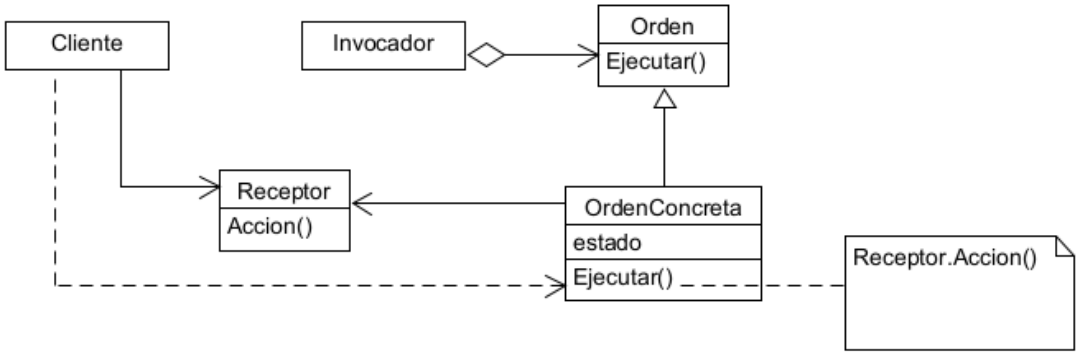


Figura 6.39 Estructura del patrón Command

“Orden” será la interfaz que permita ejecutar la operación, y de ella implementa “OrdenConcreta”, que será la clase que, una vez recibida la llamada por su interfaz, realizará la orden ejecutar correspondiente, llamando al método que corresponda de la clase receptora. “Cliente” será la clase que cree la “OrdenConcreta” y que establezca el “Receptor” de la orden. Por último, el “Receptor” será la clase que realice las operaciones asociadas a la “Orden”.

Este patrón se usa en el módulo “command”, en las tres clases que lo forman (“Compile”, “Setup” y “Loop”), que son las clases que se encargan de ejecutar la compilación, el método “setup” y el método “loop”.

6.2.3.3 Template Method

Template Method define en una operación el esqueleto de un algoritmo, delegando uno o varios de sus pasos en las subclases. La consecuencia de esto es que las subclases pueden redefinir los pasos de un algoritmo sin cambiarlo.

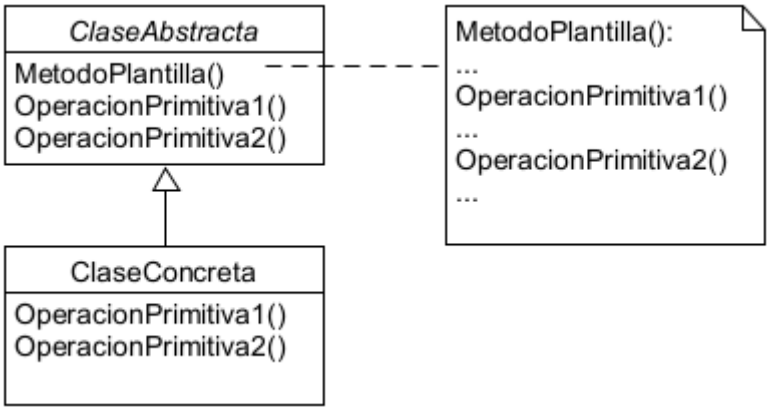


Figura 6.40 Estructura del patrón Factory Method

La "ClaseAbstracta" define el método que contiene la plantilla del algoritmo, así como las operaciones primitivas que las subclases podrán redefinir. La plantilla del algoritmo llamará a las operaciones primitivas (y otras operaciones que no se puedan redefinir) para definir la estructura del algoritmo.

Este patrón se usa en el módulo "layers", en el método "execute".

6.3 Diagramas de Interacción y Estados

En este apartado se definirán diagramas de secuencia para los casos de uso y escenarios definidos en el apartado 5.5.

6.3.1 Casos de uso 1, 2, 3

El diagrama de la Figura 6.41 nos muestra los pasos realizados para importar un archivo, guardarlo posteriormente y crear un archivo nuevo de cero. En este caso asumimos que el archivo a importar existe.

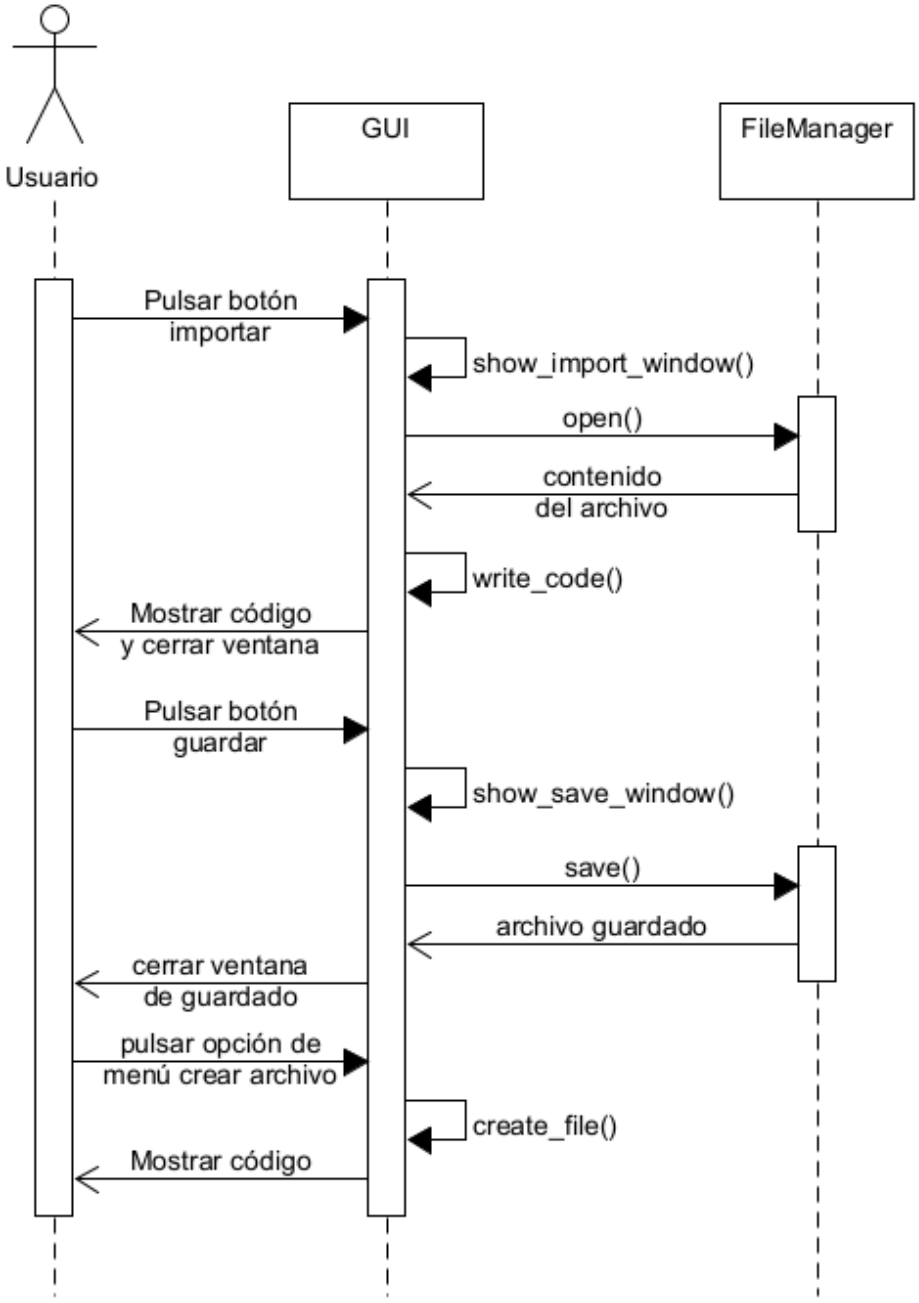


Figura 6.41 Diagrama de secuencia - casos de uso 1, 2 y 3

6.3.2 Casos de uso 5, 6, 7

En el diagrama de la Figura 6.42 se muestra el proceso que involucra los casos de uso 5, 6 y 7, que son cambiar un robot por otro. Este es un caso genérico, es decir, no se aplica para uno de los robots en concreto.

Si se concretase un robot, el diagrama cambiaría en el último método ejecutado, que es el de "update_track", ya que, en caso del robot móvil, este método representaría la pista que usará el robot; mientras que para el actuador lineal no haría nada, pues no necesita de un circuito.

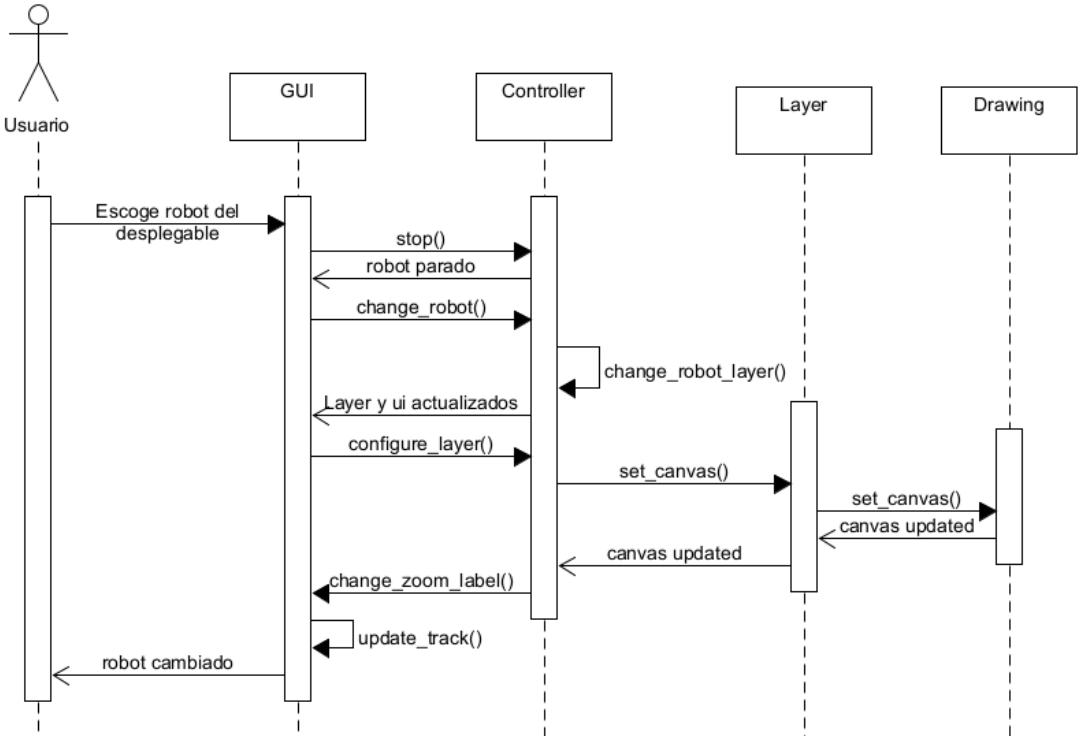


Figura 6.42 Diagrama de secuencia – casos de uso 5, 6 y 7

6.3.3 Casos de uso 8, 9, 10, 11

El diagrama de la Figura 6.43 representa los casos de uso 8, 9, 10 y 11 que corresponden a las fases de compilación del compilador (o transpilador).

El caso representado es uno en el que no hay errores, puesto a que, si los hay, el proceso de compilación debería detenerse justo al terminar la fase en la que se encontraron los errores mencionados.

Sin embargo, eso no se aplica a las advertencias, puesto a que el código se ejecutará de todas formas si hay alguna advertencia.

Este diagrama no varía si se cambia el robot que se va a representar durante la ejecución del sketch compilado.

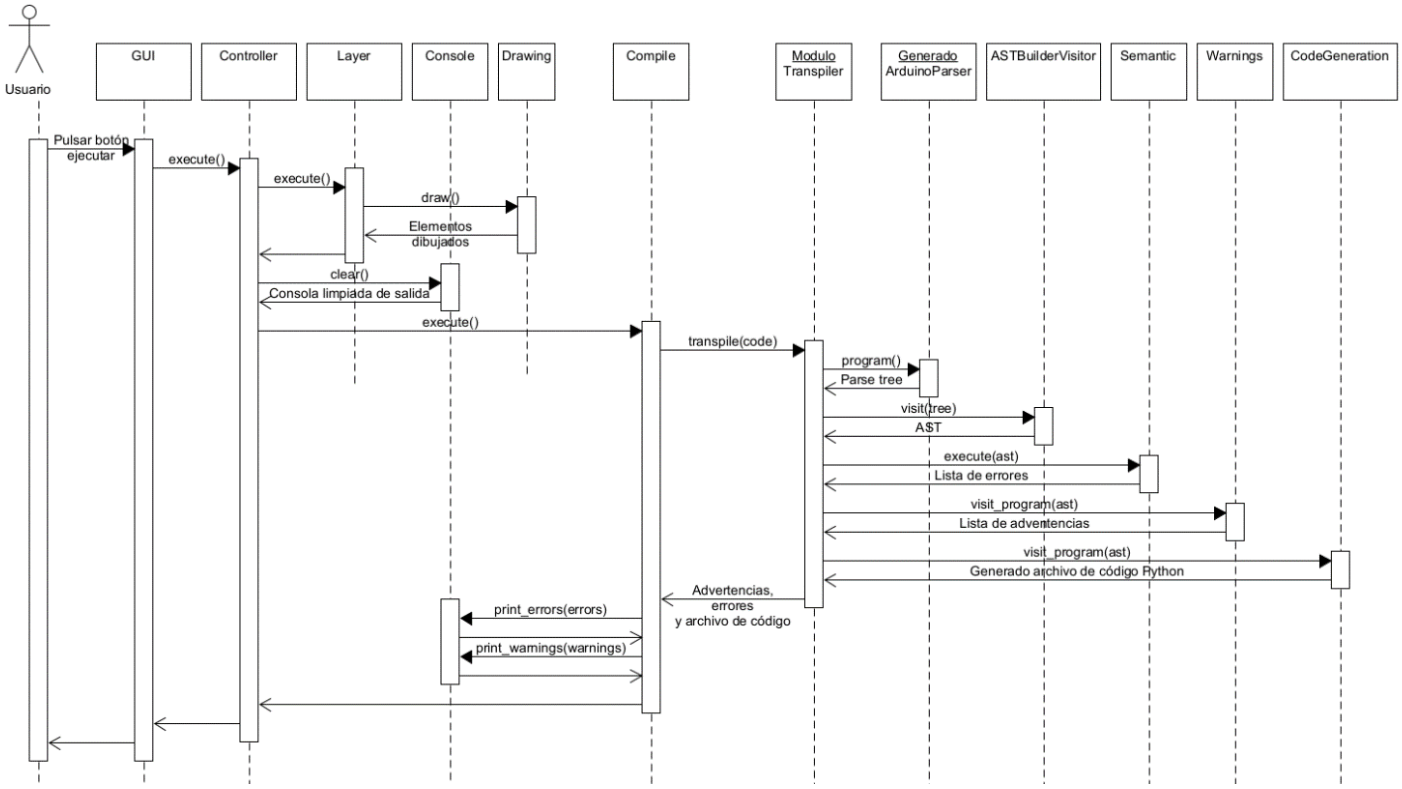


Figura 6.43 Diagrama de secuencia - casos de uso 8, 9, 10, 11

6.3.4 Casos de uso 12, 13, 14

El diagrama de la Figura 6.44 representa a los casos de uso 12, 13 y 14 que son ejecutar el sketch y simular ambos robots. Como en el caso de la Figura 6.42, el caso es uno genérico, aunque realmente no habría diferencias debidas al robot.

La ejecución parte de donde terminó la compilación del sketch. Es decir, este diagrama de secuencia sería una continuación de aquel de la Figura 6.43.

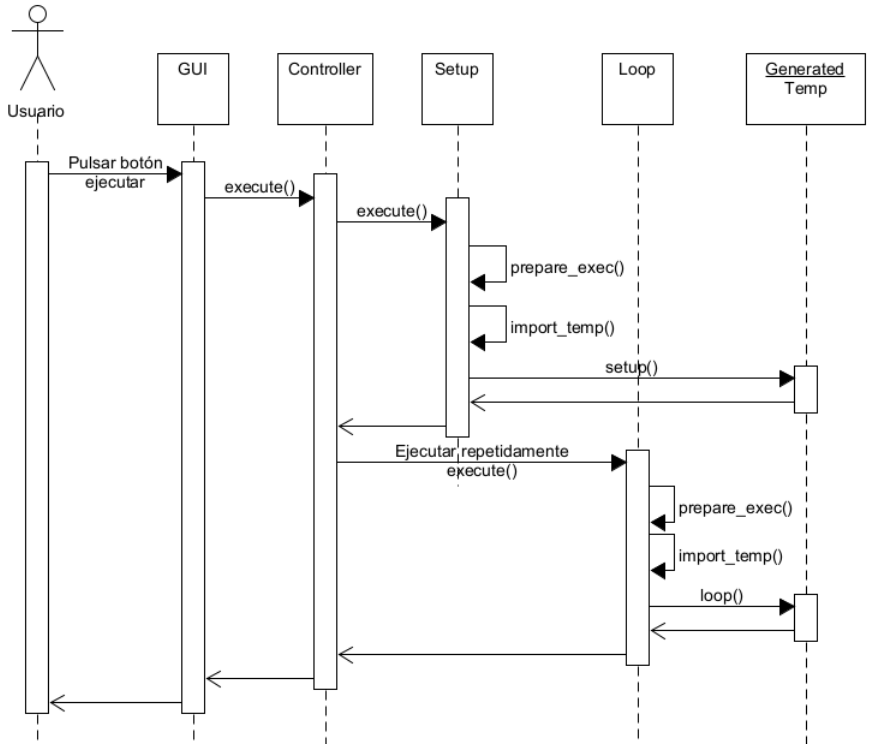


Figura 6.44 Diagrama de secuencia – casos de uso 12, 13 y 14

6.3.5 Caso de uso 16

El diagrama de la Figura 6.45 representa al caso de uso 16, que corresponde a la detención de la ejecución. El proceso de detención será el mismo sea cual sea el robot escogido.

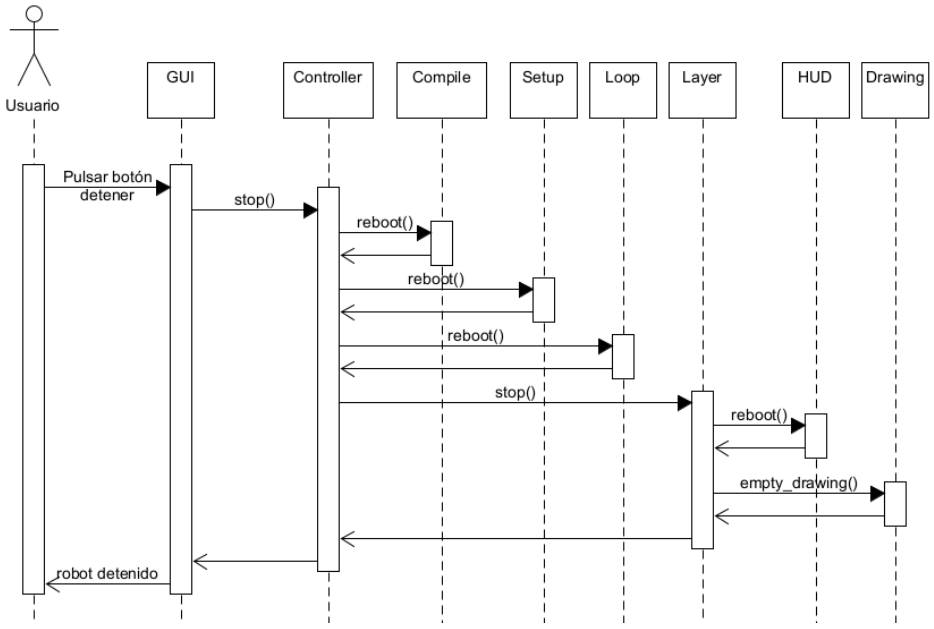


Figura 6.45 Diagrama de secuencia – caso de uso 16

6.4 Diagramas de Actividades

En este apartado se muestra (en la Figura 6.46) un diagrama de actividad del proceso principal de la aplicación, que es compilar y ejecutar el sketch programado por el usuario.

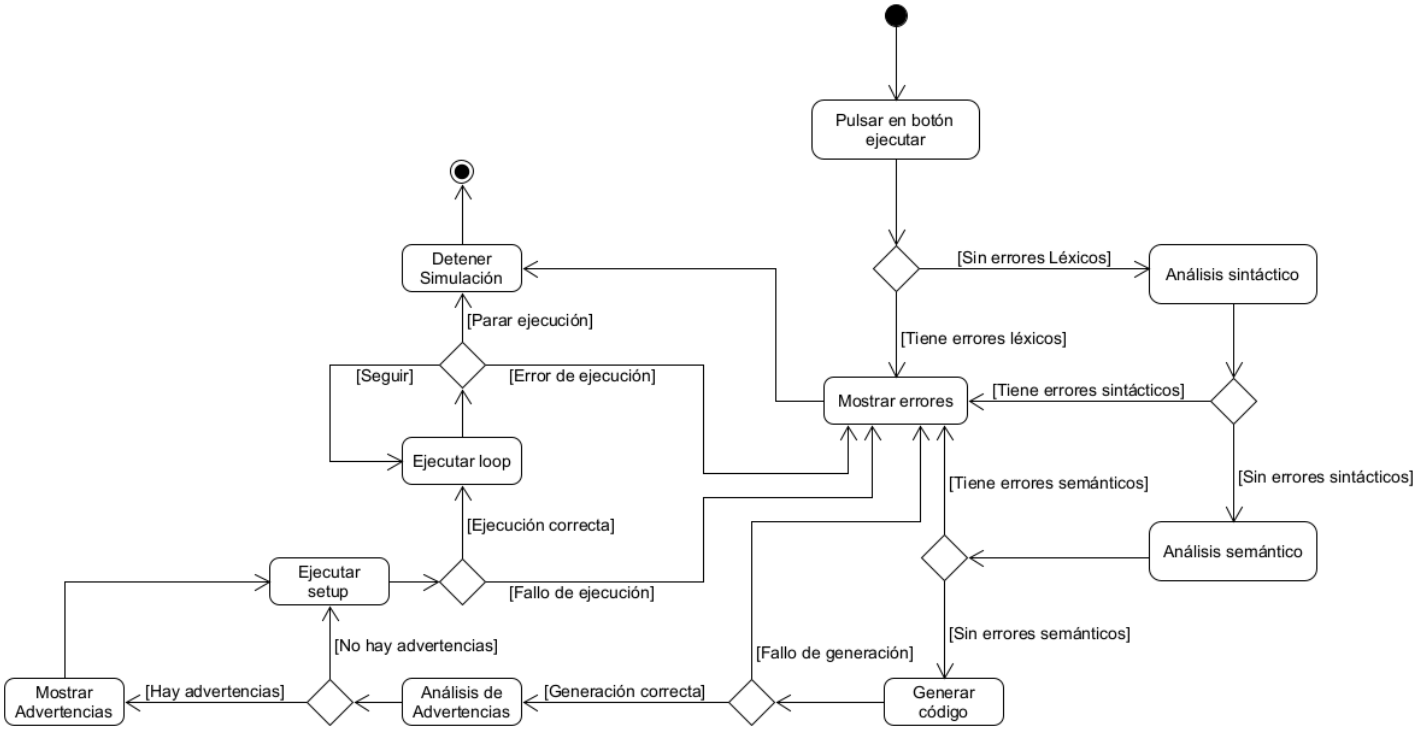


Figura 6.46 Diagrama de actividad

En este diagrama se describe un proceso de compilación y ejecución genérico. Cuando la ejecución haya terminado, el usuario recibirá la retroalimentación por parte del programa, es decir, si falla el código, recibirá los errores que ha cometido. Si no falla, el programa se ejecutará correctamente y sin mostrar ningún fallo por consola.

6.5 Diseño de la Interfaz

El diseño de la interfaz gráfica se corresponde en gran medida a aquel propuesto en el apartado 5.6.

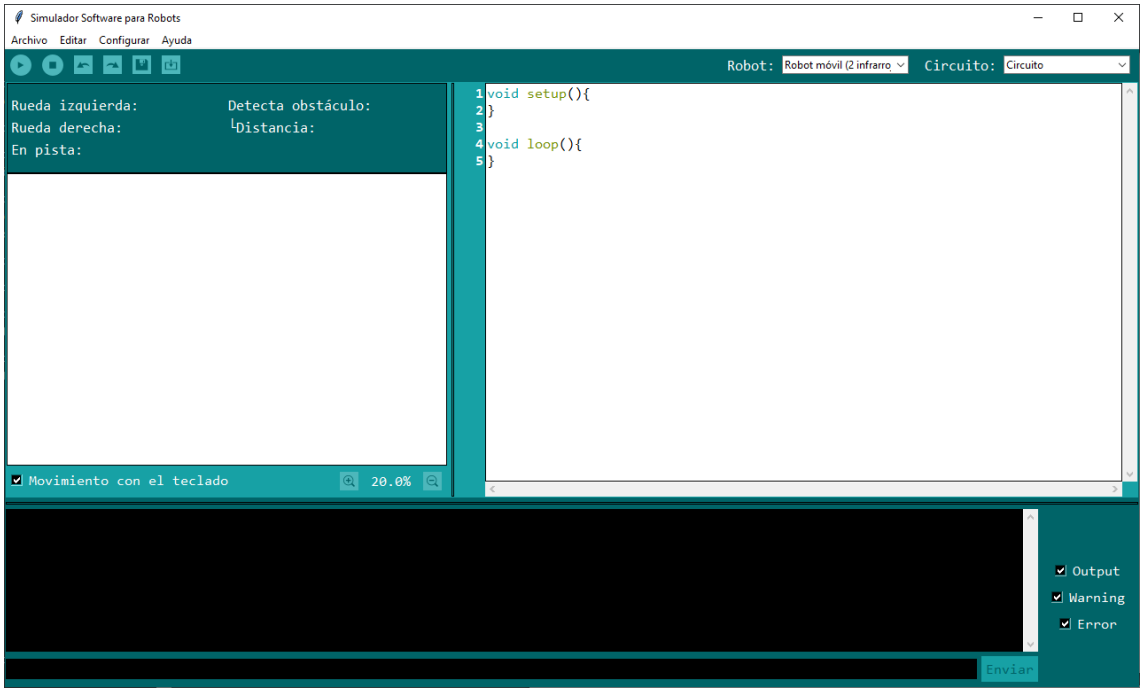


Figura 6.47 Diseño inicial de la interfaz

Como se puede ver hay pocas diferencias en la barra de herramientas. Se han eliminado menús que no se iban a usar como herramientas o ventana y se ha renombrado opciones a configuración.

La barra de ejecución solo añade un desplegable más en caso de que sea robot móvil, que es un desplegable para escoger circuitos como se enseña en las siguientes figuras:

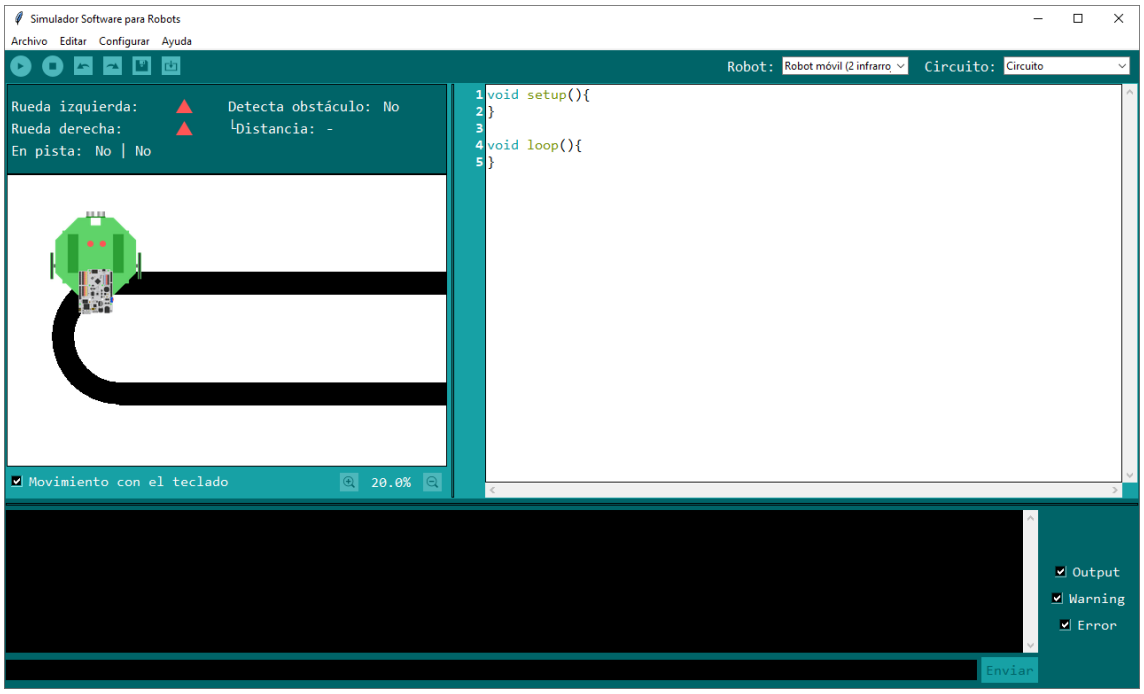


Figura 6.48 Diseño de la interfaz con el circuito "Circuito"

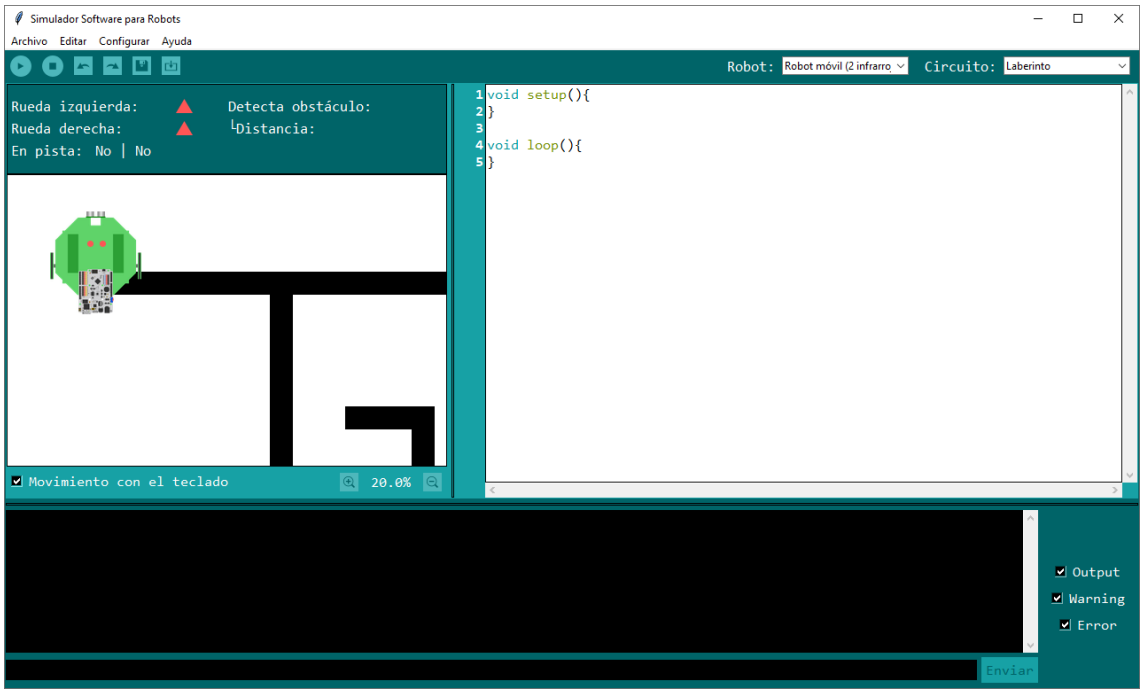


Figura 6.49 Diseño de la interfaz con el circuito "Laberinto"

Como se puede ver en la primera imagen, el circuito elegido ha sido "Circuito", mientras que en la segunda ha sido "Laberinto". En el caso del actuador lineal, este desplegable no existe, lo que hace que la barra de herramientas sea la misma que se propuso:

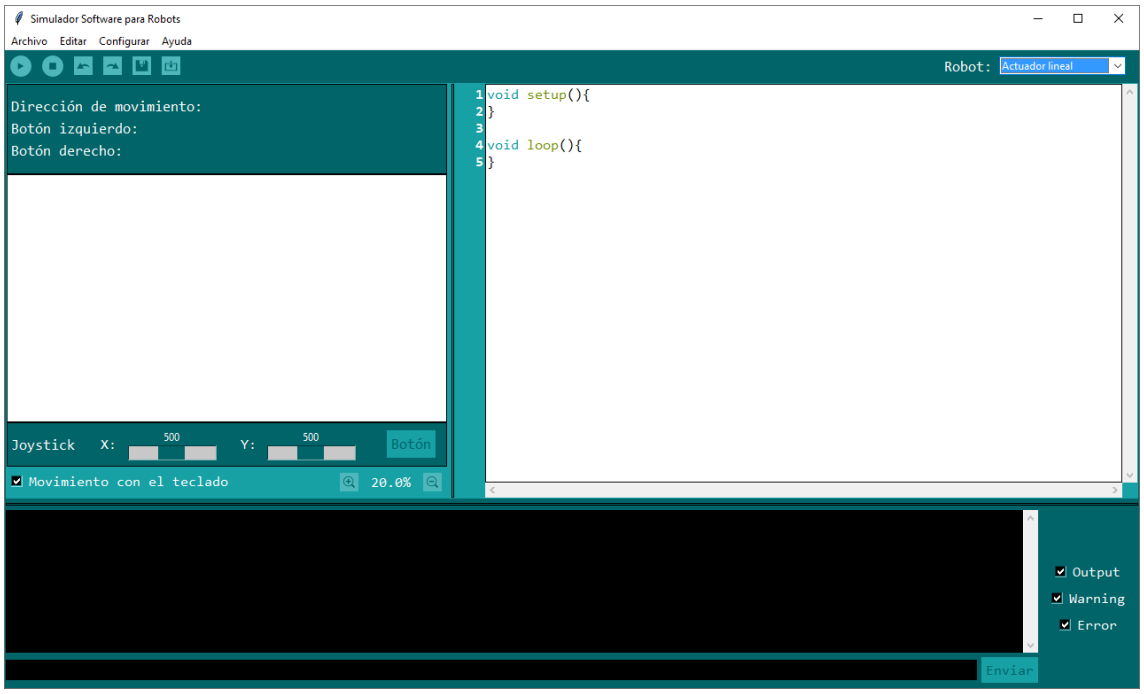


Figura 6.50 Diseño de la interfaz para el actuador lineal

En las figuras anteriores se puede ver que en la parte de la representación gráfica se han añadido varias cosas:

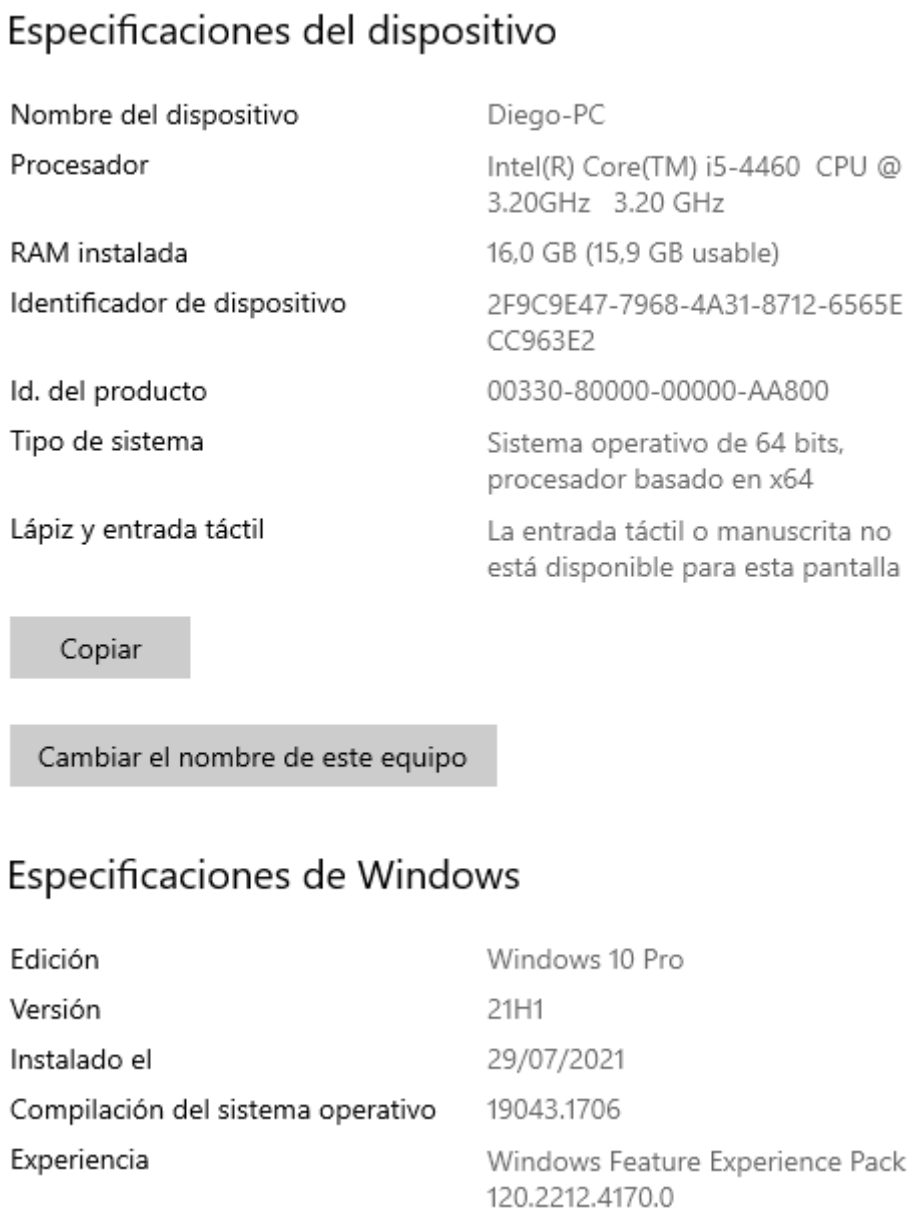
- En ambos casos se ha añadido un HUD, para el robot móvil mostrará la dirección y velocidad (mediante color: rojo – lento, amarillo – despacio, azul – rápido), Si los sensores de luz ven pista o no y si se detecta un obstáculo y la distancia en caso afirmativo. Para el actuador lineal, muestra la misma flecha que el robot móvil (en este caso horizontal) para indicar la dirección en que se mueve el robot. Además, muestra si los botones que se sitúan al final del actuador están pulsados o no.
- Además, se ha añadido una barra que contiene un *checkbox* que permite activar o desactivar el movimiento por teclado del robot (esto es para posicionarlo al antojo del usuario). Además, esa barra contiene la ampliación a la que se ve la imagen que representa a los robots.
- Por último, en la pantalla del actuador lineal, se ha añadido una barra que simula un joystick (en este caso la dirección en que se ha movido) y el botón de este.

En el caso del editor de código, se ha añadido dos barras de *scroll* (horizontal y vertical). Las líneas de texto son como estaban propuestas y, además, se ha realizado coloreado sintáctico para el código.

Por último, la consola cambia en el sentido de que se le añade un menú para filtrar la salida de la consola y un campo de texto para permitir la entrada por consola.

6.6 Especificación Técnica del Plan de Pruebas

Antes de comenzar a especificar el plan de pruebas, se especificará las características del ordenador en que se van a realizar las pruebas.



Especificaciones del dispositivo

Nombre del dispositivo	Diego-PC
Procesador	Intel(R) Core(TM) i5-4460 CPU @ 3.20GHz 3.20 GHz
RAM instalada	16,0 GB (15,9 GB usable)
Identificador de dispositivo	2F9C9E47-7968-4A31-8712-6565ECC963E2
Id. del producto	00330-80000-00000-AA800
Tipo de sistema	Sistema operativo de 64 bits, procesador basado en x64
Lápiz y entrada táctil	La entrada táctil o manuscrita no está disponible para esta pantalla

Especificaciones de Windows

Edición	Windows 10 Pro
Versión	21H1
Instalado el	29/07/2021
Compilación del sistema operativo	19043.1706
Experiencia	Windows Feature Experience Pack 120.2212.4170.0

Figura 6.51 Características del sistema

Volumen	Distribución	Tipo	Sistema de archivos	Estado	Capacidad	Espacio disponible	% disponible
(C:)	Simple	Básico	NTFS	Correcto (Arranque, Archivo de paginación, Volcado, Partición de datos básicos)	465,12 GB	246,39 GB	53 %
(Disco 0 Partición 1)	Simple	Básico		Correcto (Partición de sistema EFI)	100 MB	100 MB	100 %
(Disco 0 Partición 3)	Simple	Básico		Correcto (Partición de recuperación)	556 MB	556 MB	100 %
Datos (E:)	Simple	Básico	NTFS	Correcto (Partición primaria)	1863,01 GB	1225,58 GB	66 %

Figura 6.52 Características de almacenamiento

Cabe destacar que ciertas pruebas no se desarrollarán en este ordenador, sobre manera para probar que el sistema funciona correctamente en otros ordenadores y para que ciertos usuarios interesados en el proyecto prueben el sistema.

Al ser una aplicación *standalone* (sin necesidad de recursos externos), solo se prueba en un ordenador (excepto pruebas mencionadas anteriormente), sin necesidad de otros externos.

6.6.1 Pruebas Unitarias

Las pruebas unitarias se realizarán sobre aquella funcionalidad que ejecute operaciones básicas. Esto quiere decir que en el caso de la generación de código (por ejemplo), no se realizarán pruebas unitarias, pues es una funcionalidad compleja del código.

En caso de ser funcionalidad compleja, no quedará más remedio que realizar pruebas de forma manual, este será el caso de la interfaz gráfica (animaciones, insertar código, consola, cambio de robots, etc.) y la parte de generación de código del compilador.

A su vez, otras operaciones muy básicas y dependientes de librerías básicas de Python (como la lectura de archivos), no necesitará pruebas automatizadas, pues la programación de esta funcionalidad es similar en todos los proyectos de este tipo y funciona correctamente. No obstante, se probará después de finalizar la programación de este tipo de módulos el correcto funcionamiento de estos.

Esto implica que las pruebas unitarias que se van a automatizar son aquellas de análisis léxico, sintáctico y semántico, además de la generación del AST. Para automatizar dichas pruebas se empleará unittest para Python.

Las pruebas unitarias se realizarán siempre al finalizar la programación del módulo o clase que se haya programado. En caso de ser correcta, se continuará desarrollando, pero si falla se corregirá inmediatamente el error que haya sucedido.

A continuación, se muestra una serie de tablas con las pruebas a realizar:

Pruebas de gramática	
Escribir un sketch con operaciones aritméticas	Los tokens generados se corresponden con los esperados
Escribir un programa con definiciones de arrays, que sean de una dimensión y de varias dimensiones	
Escribir un sketch con accesos a arrays	
Escribir un sketch con operaciones a nivel de bit	
Escribir un sketch con sentencias break y continue	
Escribir un sketch con operaciones de comparación	
Escribir un sketch con operaciones compuestas (es decir, +=, -=, etc.)	
Escribir un sketch que contenga un bucle do	

while	
Escribir un sketch que contenga un else	
Escribir un sketch que contenga un bucle for	
Escribir un sketch que contenga un if	
Escribir un sketch que contenga una sentencia return	
Escribir un sketch que contenga un bloque switch case	
Escribir un sketch comprobando las asignaciones a variables	
Escribir un sketch que contenga un bucle while	

Tabla 6.1 Pruebas unitarias de gramática

Pruebas de AST	
Escribir un sketch que genere nodos de sentencias include	El nombre de los archivos especificados en la prueba debe coincidir con los especificados en el sketch
Escribir un sketch que genere definiciones de variables globales	El tipo, nombre y valor (si lo hay) de la variable debe de ser aquel que se ha declarado en el sketch. Las variables que se declaren como constantes deben de tener el valor de is_const a verdadero. Si la definición global es un array, el tamaño de ese array debe coincidir con el que se haya definido.
Escribir un sketch que contenga definiciones de funciones propias	El tipo de las funciones debe coincidir con el tipo definido en el sketch. El nombre de la función debe coincidir con el nombre definido en el sketch. Los parámetros deben coincidir en número, nombre y tipo con los definidos en el sketch. El número de sentencias de la función debe ser el mismo que el definido en el sketch.
Escribir un sketch que contenga los diferentes valores terminales del lenguaje de programación	El tipo y valor de esos terminales debe coincidir con lo definido en el sketch El tipo de las variables a las que sean asignados debe ser el mismo que se ha establecido en el sketch.
Escribir un sketch que contenga llamadas a funciones	El nombre de la llamada a la función debe coincidir con el definido en el sketch. El número y tipo de parámetros debe coincidir con lo definido en la función que se está llamando del sketch.
Escribir un sketch que contenga sentencias condicionales	Las condiciones de las sentencias deben coincidir con lo establecido en el sketch. Las sentencias dentro de un if deben coincidir con las establecidas en el sketch. Ídem para else. Las sentencias case switch deben de tener la misma variable sobre la que cambian que en el sketch. El valor del case será el mismo que en el sketch.

	Las sentencias de case y default deberán ser el mismo número que las definidas en el sketch.
Escribir un sketch que contenga bucles	Se debe comprobar que el tipo de bucle es el establecido en el sketch para cada bucle. La condición de parada del bucle debe ser la misma que la establecida en el sketch. Para los bucles for, el valor del incremento y de la definición inicial deberán coincidir con lo establecido en el sketch.
Escribir un sketch que contenga asignaciones a variables	El nombre de la variable debe coincidir con lo establecido en el sketch. El valor de la expresión que se asigna debe coincidir con lo establecido en el sketch. El operador de asignación debe coincidir con el establecido en el sketch.
Escribir un sketch que contenga operaciones a nivel de bits	El valor establecido a izquierda y derecha del operador debe ser el mismo que el establecido en el sketch. El valor del operador debe de ser el mismo que el establecido en el sketch.
Escribir un sketch que contenga definiciones locales (dentro de función)	El nombre, tipo y valor asignado (si hay este último) debe coincidir con lo establecido en el sketch. Si la variable es una constante, el valor de is_constant debe ser verdad.
Escribir un sketch que contenga operaciones del tipo ++ o --	El nombre de la variable a incrementar y el operador debe coincidir con lo establecido en el sketch.
Escribir un sketch que contenga operaciones de control de ejecución (tipo return, break o continue)	El tipo del nodo debe ser el correspondiente a la sentencia del sketch. El valor de return debe coincidir con lo establecido en el sketch.
Escribir un sketch que contenga operaciones aritméticas	El valor establecido a izquierda y derecha del operador debe ser el mismo que el establecido en el sketch. El valor del operador debe de ser el mismo que el establecido en el sketch.
Escribir un sketch que contenga operaciones de comparación y de booleanos	El valor establecido a izquierda y derecha del operador debe ser el mismo que el establecido en el sketch. El valor del operador debe de ser el mismo que el establecido en el sketch.
Escribir un sketch que compruebe la definición de arrays	El nombre, dimensiones, tamaños y elementos del array deben coincidir con lo establecido en el sketch.
Escribir un sketch con accesos a arrays	El nombre del array y el índice accedido deben coincidir con lo establecido en el sketch.

Tabla 6.2 Pruebas unitarias de AST

Pruebas de comprobaciones de errores de código	
Escribir un sketch que contenga errores sintácticos	El tipo del error debe coincidir con el tipo de error esperado. El número de errores de cada sketch debe coincidir con los errores introducidos en el sketch. El mensaje de error que se muestre por consola deberá coincidir con el mensaje esperado.
Escribir un sketch que contenga errores de definición de setup y loop	
Escribir un sketch que contenga errores de tipado	
Escribir un sketch que contenga errores de declaración de variables y funciones	
Escribir un sketch que contenga errores de break, continue y return	
Escribir un sketch que contenga errores de acceso a arrays	
Escribir un sketch que contenga errores relacionados con las librerías de código	
Escribir varios sketches que no contengan fallo alguno	El número de errores deberá ser 0, coincidiendo con el compilador de Arduino original

Tabla 6.3 Pruebas unitarias de comprobación de errores de código

Pruebas de advertencias	
Escribir un sketch que contenga las sentencias de código que tengan advertencias	El número de advertencias, tipo y mensaje deben coincidir con los esperados según el contenido del sketch.

Tabla 6.4 Pruebas unitarias de advertencias

6.6.2 Pruebas de Integración y del Sistema

Como se ha descrito en el anterior apartado, las pruebas se ejecutarán después del desarrollo de cada módulo, permitiendo continuar con la siguiente fase de desarrollo si la prueba es correcta, pero obligando a corregir los fallos si hay algún problema durante la ejecución de estas.

La integración de cada módulo se probará de la siguiente forma: primero se probará que lo que se hizo anteriormente funciona de forma correcta para después probar la integración del nuevo módulo con el conjunto formado por los módulos ya integrados anteriormente.

En el caso de las pruebas del compilador, los datos a introducir son un sketch de Arduino, cualquiera que cumpla los requisitos establecidos por la prueba. La salida esperada dependerá del sketch proporcionado a modo de entrada. Ídem para las pruebas de la consola y librerías.

Para las pruebas de representación gráfica, la prueba que usará código será utilizando un sketch. En cuanto a la prueba de movimiento con teclado, se usará las teclas WASD. Respecto al cambio de robot, la prueba será pulsar sobre la lista que tiene los robots y cambiarlo.

6.6.3 Pruebas de Usabilidad y Accesibilidad

Los tipos de usuarios que ayudaron en las pruebas de usabilidad y accesibilidad serán de un perfil de estudiante de Ingeniería Informática del Software de 3er o 4º curso o recientemente graduado que hayan cursado la asignatura de Software para robots (o que la estén cursando). Deben manejarse correctamente o con soltura en el lenguaje Arduino. Debe tener conocimientos de cómo funciona Arduino, sus placas, sensores, componentes y código para poder explorar el potencial de la aplicación.

Cada uno de estos usuarios realizará las pruebas desde su propio ordenador personal (o uno que tengan a su alcance en su defecto).

Para la realización de las pruebas se les explica a los usuarios que la aplicación sirve para simular los ejercicios del robot móvil y el actuador lineal de la asignatura. Se les explica que la aplicación sirve para ejecutar los sketches y simularlos con una animación. Cada usuario deberá después utilizar la aplicación y evaluar todas las características que vean de la aplicación, reportando mejoras y fallos que se puedan encontrar.

Se les otorgará un cuestionario a través de Google Forms, que es muy recomendable que rellenen para reportar lo que se hayan encontrado en la aplicación y cómo la valoran.

6.6.3.1 Diseño de Cuestionarios

En este apartado se detallarán los cuestionarios empleados a la hora de realizar las pruebas de usabilidad y accesibilidad.

6.6.3.1.1 Cuestionario de Evaluación

Se realizarán los siguientes cuestionarios:

- **1º**: **Preguntas de carácter general** a través de las cuales se determine el tipo de usuario y su nivel de conocimiento informático. En este caso nos centraremos más en el conocimiento de Arduino y el uso que le dan.
- **2º**: **Actividades guiadas** para hacer con nuestra aplicación.
- **3º**: **Batería de preguntas cortas** con los distintos aspectos de la aplicación que se pretendan evaluar.
- **4º**: **Observaciones**, para que el usuario aporte todo lo que considere oportuno de nuestra aplicación.

6.6.3.2 Actividades de las Pruebas de Usabilidad

6.6.3.2.1 Preguntas de carácter general

A continuación, se muestra el cuestionario de preguntas de carácter general:

¿Usa un Arduino frecuentemente?
1. Todos los días 2. Varias veces a la semana 3. Ocasionalmente 4. Nunca o casi nunca
¿Ha usado alguna vez Arduino durante el último año?
1. Si, es parte de mi trabajo o profesión 2. Lo uso básicamente para ocio 3. Solo lo he empleado durante la asignatura de software para robots 4. No, hace más de un año que cursé la asignatura
¿Qué busca Vd. Principalmente en este programa?
1. Que sea fácil de usar 2. Que sea intuitivo 3. Que sea rápido 4. Que tenga todas las funciones necesarias
¿Le resulta familiar el diseño del software de la prueba y su estética?
1. Sí, es muy familiar 2. Sí 3. Puede 4. No demasiado 5. No
¿Ha usado alguna vez algún software de simulación de Arduino parecido a este?
1. Sí 2. No 3. No me acuerdo
¿Está graduado de la carrera?
1. Sí 2. No
¿Ha cursado la asignatura?
1. Sí 2. No

Tabla 6.5 Cuestionario general

6.6.3.2.2 Actividades guiadas

A continuación, se muestran una serie de actividades guiadas a realizar sobre la aplicación para probarla:

Actividad	Tiempo Máximo	Realizado con éxito
Abrir y ejecutar el programa	< 1 minuto	
Escribir un sketch básico	< 5 minutos	
Ejecutar sketch	< 2 minutos	
Parar ejecución	< 1 minuto	
Guardar un sketch	< 3 minutos	
Importar un sketch	< 3 minutos	
Configurar pines de un robot	< 5 minutos	
Cambiar tipo de robot	< 1 minuto	
Filtrar errores de la consola	< 1 minuto	
Filtrar advertencias de la consola	< 1 minuto	
Filtrar mensajes de texto de la consola	< 1 minuto	
Crear un nuevo sketch desde archivo	< 2 minutos	
Deshacer y rehacer una acción	< 1 minuto	
Cambiar circuito dentro del robot móvil y ejecutar	< 5 minutos	
Mover robot móvil con las teclas	< 2 minutos	
Simular comportamiento de código de los ejercicios (vale el hecho para la asignatura si está disponible)	< 7 minutos	
Salir del programa a través del menú archivo	< 1 minuto	

Tabla 6.6 Actividades guiadas

6.6.3.2.3 Preguntas Cortas sobre la Aplicación y Observaciones

En este apartado se muestra el cuestionario de preguntas cortas a realizar al usuario que ha probado el sistema:

Facilidad de Uso	Totalmente de acuerdo	De acuerdo	Neutral	En desacuerdo	Totalmente en desacuerdo
¿Sabe dónde está dentro de la aplicación?					
¿Existe ayuda para las funciones en caso de que tenga dudas?					
¿Le resulta sencillo el uso de la aplicación?					
¿Le resulta intuitivo el sistema de simulación de los robots?					
¿Le ha resultado sencilla la					

configuración de pines?					
¿Le resulta intuitiva la salida y entrada mediante consola?					
Funcionalidad	**Totalmente de acuerdo**	**De acuerdo**	**Neutral**	**En desacuerdo**	**Totalmente en desacuerdo**
¿Funciona cada tarea como Vd. Espera?					
¿El tiempo de respuesta de la aplicación es muy grande?					
¿Ha echado en falta alguna funcionalidad en el sistema?					
Calidad del Interfaz					
Aspectos gráficos	**Totalmente de acuerdo**	**De acuerdo**	**Neutral**	**En desacuerdo**	**Totalmente en desacuerdo**
El tipo y tamaño de letra es adecuada					
Los iconos e imágenes usados son adecuados					
Los colores empleados son					
La estética es adecuada					
Diseño de la Interfaz	**Totalmente de acuerdo**	**De acuerdo**	**Neutral**	**En desacuerdo**	**Totalmente en desacuerdo**
¿Le resulta fácil de usar?					
¿El diseño de las pantallas es claro y atractivo?					
¿Cree que el programa está bien estructurado?					
Manual de ayuda	**Si**		**No**		**A veces**
¿Necesitó acceder al manual de ayuda?					
Opinión					
Relevancia para la asignatura	**Totalmente de acuerdo**	**De acuerdo**	**Neutral**	**En desacuerdo**	**Totalmente en desacuerdo**
¿Considera que el simulador facilita los ejercicios que emplean los robots simulados?					
¿Cree que le sería beneficioso este simulador a la hora de cursar la asignatura?					
¿Cree que deberían añadirse otros robots o componentes?					

Aspecto	Totalmente de acuerdo	De acuerdo	Neutral	En desacuerdo	Totalmente en desacuerdo
¿Considera que la similitud de diseño con el IDE de Arduino facilita la adaptación?					
Observaciones					
Cualquier comentario del usuario					

Tabla 6.7 Preguntas cortas

6.6.3.2.4 Cuestionario para el responsable de las pruebas

A continuación, se muestra un cuestionario para que sea realizado por el responsable de las pruebas:

Aspecto Observado	Notas
El usuario comienza a trabajar de forma rápida por las tareas	
Tiempo en realizar cada tarea	
Errores leves cometidos	
Errores graves cometidos	
Comentarios de usuarios	
Faltas de funcionalidad	
¿Se ha consultado la ayuda por parte del usuario?	
¿Ha necesitado el usuario mucho tiempo para realizar las tareas?	
Grado de satisfacción del usuario	
Notas del responsable	

Tabla 6.8 Cuestionario para el responsable de las pruebas

6.6.3.2.5 Pruebas de Accesibilidad

En este subapartado se mostrarán las pruebas de accesibilidad de la aplicación. Para ello, se empleará la siguiente tabla, señalando el resultado de la prueba y explicando cualquier detalle que sea necesario. La mayoría de las pruebas/preguntas utilizadas son las propuestas por Thomas Hamilton **[Hamilton22]**

Prueba	Resultado	Notas
¿La aplicación contiene mnemónicos como alternativa a todas las operaciones con ratón?		
¿Se proveen instrucciones en el manual de usuario?		
¿Es sencillo el uso de la aplicación usando la documentación?		
¿Hay atajos para acceder a los menús?		
¿Soporta todos los sistemas operativos?		
¿Son apropiados para todos los usuarios los colores de la aplicación?		
Las imágenes e iconos, ¿Son fáciles de entender?		
Las alertas de la aplicación, ¿Tienen sonido?		
¿Se puede cambiar la fuente y su tamaño?		

No se usa el color como único medio para transmitir información.		

Tabla 6.9 Pruebas de accesibilidad

6.6.4 Pruebas de Rendimiento

Para las pruebas de rendimiento, realizaremos *profiling* de la aplicación ya que, al ser una aplicación que no funciona a través de servidor, esta no tendrá usuarios concurrentes. Por tanto, no necesita de pruebas de carga.

La herramienta que se empleará para el *profiling* será cProfile de Python. Esta librería se ejecuta junto con la aplicación y mide cuanto tiempo de la ejecución se pasa en cada función, y a su vez, cuanto tiempo usa por llamada.

Dentro de lo posible se intentará mejorar el rendimiento de los métodos que consuman demasiado tiempo y recursos. Aunque no en todos los casos podrá ser así, debido a la naturaleza de Python y que este lenguaje sea a efectos prácticos de un solo hilo.

Capítulo 7. Implementación del Sistema

En este capítulo se explicarán diferentes detalles referentes a la implementación del sistema. Primero, se comentarán los estándares y normas seguidos a la hora de desarrollar el proyecto. En el siguiente apartado se expondrán los lenguajes de programación usados a la hora de desarrollar el sistema. En el tercer apartado se mostrarán las diferentes herramientas empleadas a la hora de desarrollar el sistema. En el último apartado del capítulo, se expondrán los diferentes problemas encontrados durante del desarrollo; así como una explicación detallada de las clases y su funcionamiento.

7.1 Estándares y Normas Seguidos

Durante el desarrollo de este proyecto se ha seguido las convenciones de código propuestas por la guía de estilo PEP8 **[PEP8]**. Aunque no se ha seguido al 100%, sobre todo en cuanto a longitud de líneas se refiere; si se han intentado seguir en gran medida, de tal manera que se mejore la legibilidad del código.

Sobre manera, se ha aplicado las convenciones de nombrado especificadas y las de comentarios. La validación del código se ha realizado empleando “pycodestyle”. Los errores de los archivos autogenerados se han ignorado, pues igualmente volverían a tener errores una vez se tengan que volver a generar.

Además, se han ignorado las discrepancias con la convención de código que tengan que ver con la longitud de línea (por comodidad personal).

7.2 Lenguajes de Programación

En este apartado se van a mostrar los diferentes lenguajes de programación empleados para el desarrollo del sistema. Antes de ello, se debe destacar el hecho de que, aunque el proyecto contenga código Java, esto no quiere decir que sea considerado un lenguaje usado para el desarrollo. Esto es debido a que el código Java es generado por ANTLR4 y usado simplemente por las clases y módulos relacionados con ese código.

Por ello, y al no ser en ningún caso código desarrollado por parte del programador, no se expondrá en este apartado el lenguaje Java. Tampoco se expondrá JSON, al ser un formato para el intercambio de datos y no un lenguaje de programación.

7.2.1 Python

Como ya se ha mencionado varias veces a lo largo de esta documentación, se ha empleado como lenguaje principal de desarrollo el lenguaje Python **[Python91]**. En concreto la versión de Python 3.10.4.

Python es un lenguaje de programación multiparadigma, es decir, tiene varios paradigmas, siendo los más destacados el procedimental, el orientado a objetos y el funcional. También incorpora, entre otros: tipado dinámico, módulos, excepciones y clases.

Los puntos más fuertes de Python son que su sintaxis es muy clara, que es portable y que es multiplataforma (Linux y Windows entre otros). Además de todo lo anterior, es de código abierto y es gratuito.

Figura 7.1 Logo de Python

Python ha sido usado como lenguaje de desarrollo tanto en el *back-end*, que cubre toda la parte del compilador, los estados de los robots y acceso a ficheros; como en el *front-end*, que cubre toda la interfaz gráfica y la salida y entrada por consola.

7.2.2 ANTLR

El lenguaje empleado por ANTLR4 **[ANTLR89]** sirve para el desarrollo de los archivos con extensión “g4”, que son aquellos que definen la gramática del compilador. Dichos archivos se le envían a la herramienta de generación de los parser de ANTLR4 (el archivo de extensión “jar” descargado de la web), para generar el *lexer* y el *parser* que serán los que realicen las fases de análisis léxico y sintáctico del compilador. La versión de ANTLR usada es la 4.10.1.

7.3 Herramientas y Programas Usados para el Desarrollo

En este apartado se van a describir las diferentes herramientas y programas usados durante el desarrollo, así como comentar el uso que se les ha dado y, en caso de que la tengan; su versión.

7.3.1 Visual Studio Code

Visual Studio Code **[VSCode15]** es un editor de código (que no entorno de desarrollo integrado o IDE) desarrollado por Microsoft, que incluye soporte para ejecución y depuración del código; así como control de cambios (repositorios). Está disponible en macOS, Linux y Windows. Durante todo el desarrollo del proyecto, se ha empleado la versión más actualizada del programa (siendo la 1.68.1 actualmente).

Este programa permite el uso de extensiones para ampliar su funcionalidad. En el caso de este proyecto, se han usado las siguientes:

- ANTLR4 grammar syntax support, que realiza el coloreado sintáctico de los archivos de extensión "g4", además de sugerencias durante la codificación y validaciones sintácticas (mostradas en el archivo y por salida de consola).
- Pylance, que sirve para que, junto con la extensión Python, se proporcione un buen soporte al lenguaje Python. Algunas características más relevantes de la extensión son el realizar auto importados, el completado de código, la sugerencia de parámetros o la navegación de código.
- Python, junto con Pylance, realiza el soporte de Python, con características como el formateado de código o permitir las pruebas y la depuración; así como la refactorización de código.
- vscode-pdf, que permite leer documentos PDF en el programa sin falta de usar un programa externo. Es útil a la hora de leer el manual de usuario y comprobar es correcto.

El programa se ha empleado para el desarrollo de todo el código que conforma el sistema, desde el compilador hasta la interfaz gráfica.

Figura 7.2 Logo de Visual Studio Code

7.3.2 Notepad++

Se trata de otro editor de código (en este caso es de código abierto), desarrollado por Don Ho y cuya versión actual es la 8.4.2. Como en el caso de Visual Studio Code, la versión que se ha utilizado ha sido siempre la más actualizada. Está escrito en C++ y es un programa rápido y ligero, con lo que es muy eficiente en términos de rendimiento y recursos **[NotepadPP03]**.

Este programa se ha utilizado para la edición de archivos del proyecto que requieran búsqueda y reemplazamiento avanzados (ya que la búsqueda y reemplazo de Visual Studio Code es mucho más básica).

Además, se ha empleado para leer archivos de código externos al sistema, sobre manera por la rapidez de ejecución del programa, que se ejecuta mucho más rápido que Visual Studio Code y que permite, por tanto, un acceso más rápido a esos archivos.

Figura 7.3 Logo de Notepad++

7.3.3 pip

Pip **[Pip08]** es el instalador de paquetes de Python. Por tanto, su utilidad en el proyecto ha sido instalar los diferentes módulos y paquetes externos empleados por el sistema (por ejemplo, tkinter o pillow). Como en los casos anteriores, se ha empleado la versión más actualizada, que en este momento es la 22.1.1.

7.3.4 git y GitHub

El software git **[Git07]** y el servicio GitHub **[GitHub08]** se han usado (en conjunto) para realizar el control de versiones del proyecto. En el caso de GitHub, este se ha empleado además para almacenar el proyecto; es decir, se ha empleado como repositorio. Ambos funcionan en conjunto en el sentido de que GitHub emplea git como medio para realizar el control de versiones del proyecto. La versión utilizada de git es la "2.35.1.windows.2".

Además, GitHub se ha empleado para gestionar las *pull request* del proyecto (es decir, realizar un *merge* de la rama de desarrollo a la principal). Se planea en un futuro emplearlo como un lugar en el que acceder libremente (código abierto) al código del software, así como para reportar errores del programa.

Figura 7.4 Logos de GitHub (izquierda) y git (derecha)

7.3.5 Draw.io

Draw.io **[DrawIO17]** es una aplicación online de código abierto para la creación de diagramas, desde diagramas de clases, pasando por diagramas entidad-relación e incluso *mockups* de interfaces gráficas. Esta última funcionalidad es para lo que se ha usado este software. En concreto se ha realizado el segundo *mockup* del apartado 5.6.

Figura 7.5 Logo de draw.io

7.3.6 UMLet

UMLet **[UMLet02]** es una herramienta para la creación de diagramas UML de código abierto. Sus mayores ventajas son su simplicidad y su ligereza. Se ha empleado para realizar todos los diagramas UML que contiene esta documentación, desde diagramas de clase, diagramas de ejecución, diagramas de paquetes, diagramas de estado, etc. La versión del software empleada ha sido la 15.0.0.

Figura 7.6 Logo de UMLet

7.3.7 OneDrive

OneDrive **[OneDrive07]** es un servicio de almacenamiento en la nube ofrecido por Microsoft. Al ser el utilizado por nuestra Universidad, se ha empleado para almacenar todo lo relacionado con esta documentación: actas de reunión, diagramas UML, copias de seguridad de la documentación, etc. Fue lanzado en febrero del año 2014.

Figura 7.7 Logo de OneDrive

7.4 Librerías Usadas para el Desarrollo

En este apartado se mostrarán las librerías que han sido empleadas finalmente para el desarrollo del sistema.

7.4.1 ANTLR4

ANTLR (ANother Tool for Language Recognition) **[ANTLR89]** es un generador de analizadores (*parser*) para la lectura, procesado, ejecución o traducción de texto estructurado o código fuente. Genera analizadores a partir de una GLC, en concreto de tipo LL(*) **[Parr11]**.

ANTLR se usa en proyectos importantes, como Twitter, que lo usa en su buscador a la hora de analizar las peticiones; Oracle, que usa en SQL developer y sus herramientas de migración o NetBeans, que analiza el código de C++ con esta herramienta.

Figura 7.8 Logo de ANTLR

ANTLR4 es una herramienta bastante simple de usar, simplemente se debe escribir la gramática del lenguaje que se va a analizar y usar el ejecutable de Java para generar el *parser*. Además, la forma de escribir la gramática es muy simple y bastante similar a la BNF.

El mayor problema que tiene ANTLR es que, cada vez que se haga una modificación en la gramática, se debe ejecutar el archivo Java. Otro inconveniente es que el archivo "jar" genera archivos de código Java que son necesarios para los analizadores (léxico y sintáctico) que hacen más pesado y complejo el proyecto.

Se ha escogido ANTLR por su sencillez y facilidad de uso, simplificando mucho el desarrollo del compilador, además de la rapidez de adaptación a la herramienta, debido a que ya se usó con anterioridad.

7.4.2 Tkinter

Tkinter **[Tkinter88]** es la interfaz por defecto de Python para el kit de herramientas de GUI Tk. Está disponible en la gran mayoría de sistemas Linux, además de Windows. Aunque es la librería gráfica más usada (o de las más usadas) de Python, no está mantenida por Pyhon.

Tkinter da soporte a muchas versiones TCL/TK (con soporte multihilo o no). Además, añade mucha lógica para facilitar procesos de implementación.

Las mayores ventajas de tkinter es su rapidez de uso, ya que solo requiere un comando para instalarlo. Además, una vez se aprende lo básico de la librería, se vuelve muy intuitivo y sencillo de usar.

Los mayores problemas de tkinter surgen cuando se debe desarrollar un sistema complejo. Si ese subsistema requiere hilos, tkinter no es seguro para la programación multihilo. Si se requieren bucles que procesen mucha información, esto hará que la interfaz se congele y no responda. Además, el propio tkinter no contiene ciertos elementos de interfaz gráfica básicos, como *checkboxes*, con lo que hace falta importar ttk, que es una nueva familia de widgets para tkinter.

Se ha escogido esta librería porque es bastante simple de usar y no consume demasiados recursos del ordenador. Además, tiene una gran comunidad de desarrolladores detrás, lo que facilita la resolución de dudas sobre la librería. El hecho de que la documentación sea extensa y bastante buena también se puede considerar una razón por la que se ha escogido.

7.4.2.1 Pillow

Pillow es una biblioteca de manipulación de imágenes. Se necesita en el proyecto para manejar las imágenes de los diferentes robots y de los botones de la barra de herramientas, por lo que su uso es obligatorio. Se usa en conjunto con tkinter.

7.5 Creación del Sistema

En este apartado se van a explicar los aspectos con los que se ha lidiado durante el desarrollo de la aplicación, empezando por los problemas encontrados y siguiendo con una descripción detallada de las clases que componen el sistema.

7.5.1 Problemas Encontrados

En este apartado se explicarán los problemas encontrados a lo largo del proyecto, con especial énfasis en el problema expuesto en el apartado 7.5.1.4, ya que afecta a la aplicación dado a que solo ha podido ser parcialmente resuelto.

7.5.1.1 Permitir el cambio de tamaño de los paneles

El primer problema encontrado fue a la hora de desarrollar la interfaz gráfica. Desde un primer momento se diseñó la aplicación con la idea de permitir cambiar el tamaño de los componentes, para así hacerla más flexible y facilitar la vista y organización al usuario.

Tkinter implementa para ello una clase que es “PanedWindow”, que resulta ser el foco del problema. Desde el primer *mockup* de la aplicación, la intención era dos paneles superiores divididos por una barra vertical y debajo otro panel separado de los superiores por una barra

horizontal; ambas barras se podrían arrastrar, de tal manera que se cambia el tamaño del componente.

El problema surge en el momento en que queremos organizar los componentes de aquella manera. Fue necesario investigar en profundidad la documentación de "tkinter", que para este caso determinado era un poco escasa. Tras mucho investigar, se encontró la solución, y es que el constructor de este elemento de la GUI tiene una opción "orient" que permite orientar la colocación de los componentes gráficos, es decir, "orient" en vertical no quiere decir que la barra sea vertical, sino que la barra es horizontal; los componentes se insertan primero uno arriba y luego otro abajo. Esto es de la forma opuesta para la orientación horizontal.

7.5.1.2 Números de línea

El siguiente problema surge después de finalizar el *mockup* refinado, ya que se añadió esta funcionalidad. Desgraciadamente, "tkinter" no proporciona una manera sencilla de implementar esta funcionalidad. Después de mucho investigar por internet, se dio con una solución propuesta por Bryan Oakley **[Oakley13]**, que ha sido adaptada al proyecto según las necesidades que se debían cubrir.

Sobre manera, los mayores cambios han sido gráficos, cambiando el color de fondo y la fuente (tamaño y color también); además de adaptar el editor de texto propuesto para añadir posteriormente el resaltado de sintaxis.

7.5.1.3 Cargado de imágenes

Este es un problema muy simple por su naturaleza, pero que llevó algo de tiempo de investigación. La forma en la que funciona el colector de basura de Python hace que, si las imágenes no se añaden a una lista o son instancias de clase, estas no se muestren (al menos en "tkinter"). La solución es la indicada en el problema.

7.5.1.4 Problema de los "bucles"

Este es el problema más grande de la aplicación, y dada la complejidad del problema, este solamente ha podido ser parcialmente resuelto. La primera vez que se intentó ejecutar código con un bucle o un delay, surgía un problema. Ese problema es que, al no actualizarse el estado del robot por estar dentro de un bucle, el bucle se volvía infinito.

Para que sea más fácil de entender, vamos a explicar lo que conduce al problema:

1. Se entra al bucle.
2. Se itera.
3. Como el dibujo no se ha actualizado, el estado del robot no cambia.
4. Si el bucle depende del estado del robot (por ejemplo, que un sensor de luz vea la pista), entonces el bucle sigue iterando.
5. Con ello, se ha obtenido un bucle infinito

El caso del *delay* es ligeramente diferente. Usar el *delay* de Python implicaría parar el hilo, con lo que se pararía el dibujo también, aunque en este caso, si se sale de la situación, eso sí, una vez se acabe el tiempo.

El problema parte de que Python, a efectos de ejecución, usa un solo hilo, por lo que la programación multihilo realmente no lo es. Esto no tendría consecuencia alguna si el programa pudiese ejecutarse en un solo hilo, pero el hecho de que se deba ejecutar el código compilado hace que debamos depender de otro hilo.

Normalmente, la solución el problema de solo tener un hilo sería crear otro proceso de Python, que ejecute el programa del "otro hilo". Sin embargo, esta opción no es realmente viable, porque los procesos no comparten memoria (y la opción que permite compartirla es excesivamente compleja). Por tanto, el problema no se puede resolver (o más que no se puede, en el contexto actual, no es viable) empleando otro procesador lógico.

La "solución" ha sido crear una clase que, cada vez que es llamada, actualiza gráficamente la ventana creada con "tkinter". Dicho código está presente en el módulo "screen_updater", y en concreto es la función "update_idletasks". Al llamar al módulo al final de cada bucle y dentro del método delay de la librería estándar, el robot se puede seguir dibujando cada 15 milisegundos (para generar 60 fotogramas por segundo) y, por ello, el estado del robot puede cambiar en cada iteración, lo que rompe el bucle infinito (siempre que se cumpla la condición).

En el caso del *delay*, resuelve el problema de que no hace falta que se llame al método homónimo de Python, sino que se puede hacer un bucle infinito hasta que pasa el tiempo y que el robot se siga dibujando. Por ello, funciona como lo hace en la realidad el método en Arduino.

Pero esta solución genera un problema, y es que interfaz gráfica no es responsiva (no permite interacciones) mientras se esté ejecutando el *delay* o el bucle. No es un problema grande, ya que se pueden realizar todas las acciones correctamente (y se ha encontrado que el uso de los mnemónicos es infalible en caso de que no responda); pero genera casos en los que, si se pulsa un botón durante el tiempo de no respuesta, este botón no genere un evento de pulsación, lo que hace que no se realice lo que quiere el usuario en algunas ocasiones (salvo uso de mnemónicos, como se ha comentado).

7.5.1.5 Resaltado de sintaxis

El problema surge debido a que "tkinter" no proporciona una forma sencilla de colorear palabras. El problema se solucionó utilizando expresiones regulares para buscar palabras y colorearlas.

Para saber que palabras deberían ser coloreadas, se añadió en el fichero "assets" un archivo de texto con la palabra y la clave del color. Dicha palabra es buscada por la expresión regular y, una vez encontrada la palabra, se devuelve la línea en que está, la columna en que empieza y la línea en que acaba y la columna en que acaba.

Pero a la hora de colorear nuevas palabras, se debe hacer que se coloree el cuadro de texto cada poco tiempo (un segundo o algún otro tiempo poco apreciable). El primer enfoque para actualizar el color era borrar todo el coloreado y volver a pintar uno a uno, pero esto generaba una especie de parpadeo.

La solución final al problema está basada en las televisiones de tubo (de rayos catódicos), ya que funciona de una forma muy similar. Se va borrando el color y coloreando línea a línea. De esta manera, no se genera parpadeo alguno.

Cabe destacar que esta funcionalidad generaba el mismo problema que los bucles de falta de respuesta de la interfaz, problema solucionado gracias a los generadores de Python. Esta puede que también sea la solución definitiva al problema de los bucles (para quitar la falta de respuesta, pero requiere un estudio que consumirá bastante tiempo)

7.5.2 Descripción Detallada de las Clases

La documentación del código de la aplicación se encuentra en la carpeta adjunta "documentaciónPython", accediendo al archivo simulator.html (véase el apartado 14.2).

Capítulo 8. Gramática y analizadores

En este capítulo se explicarán los diferentes aspectos relacionados con la Gramática Libre de Contexto (GLC) y las diferentes fases del compilador. La GLC reconocerá un subconjunto de las instrucciones reconocidas por el compilador de Arduino (es un subconjunto debido a que se han dejado fuera instrucciones que nunca van a ser usadas, como el nada recomendado goto o los punteros). Los elementos reconocidos por la gramática se explicarán en el siguiente apartado.

8.1 Backus Naur Form (BNF) de la GLC

En este apartado se muestra la BNF de la gramática libre de contexto.

```
<start> ::= <program> EOF

<program> ::= <include> <program_code>

<include> ::= | "#include" STRING_CONST | "#include" "<" ID "." ID ">"

<program_code> ::= | <declaration> ";" | <function> | <define_macro>

<declaration> ::= <simple_declaration> | <array_declaration>
    | ("const" | "static") <declaration>

<simple_declaration> ::= <var_type> ID "=" <expression> | <var_type> ID

<array_declaration> ::= <var_type> ID <array_index> "=" <expression>
    | <var_type> ID <array_index> "=" <array_elements>
    | <var_type> ID <array_index>

<define_macro> ::= "#define" ID <expression>
    | "#define" ID <array_elements>

<array_index> ::= | <array_index> "[" INT_CONST "]" | "[" "]"

<array_elements> ::= "{" <array_elements> "," <array_element> "}" | <array_element>

<array_element> ::= "{" <array_element> "," <expression> "}"| <expression>

<var_type> ::= "bool" | "boolean" | "byte" | "char" | "double" | "float"
    | "int" | "long" | "short" | "size_t" | "String" | "unsigned int"
    | "unsigned char" | "unsigned long" | "void" | "word" | ID

<function> ::= <var_type> ID "(" <function_args> ")" "{" <sentence> "}"

<sentences> ::= | <sentences> <sentence>

<function_args> ::= | <function_args> "," <declaration> | <declaration>

<iteration_sentence> ::= "while" "(" <expression> ")" <code_block>
    | "do" <code_block> "while" "(" <expression> ")" ";"
```

```
    | "for" "(" <simple_declaration> ";" <expression> ";" <expression> ")"
<code_block>

<conditional_sentence> ::= "if" "(" <expression> ")" <code_block> "else" <code_block>
    | "if" "(" <expression> ")" <code_block>
    | "switch" "(" <expression> ")" "{" <case_sentence> "}"

<case_sentence> ::= "case" <expression> ":" <sentences> | "default" ":" <sentences>

<code_block> ::= "{" <sentences> "}" | <sentence>

<sentence> ::= <declaration> ";"
    | <iteration_sentence>
    | <conditional_sentence>
    | <assignment> ";"
    | <expression> ";"
    | <define_macro>
    | "return" <expression> ";" | "return" ";"
    | "break" ";"
    | "continue" ";"

<assignment> ::= <expression> "=" <expression>

<expression> ::= "true"
    | "false" | LOW | HIGH | ANALOG_PIN | INPUT | INPUT_PULLUP | OUTPUT
    | HEX_CONST | OCTAL_CONST | BINARY_CONST | INT_CONST | FLOAT_CONST
    | CHAR_CONST | STRING_CONST | ID
    | "(" <expression> ")"
    | <expression> "." ID
    | ID <array_index>
    | <expression> "(" parameter ")"
    | <conversion>
    | <expression> ("++"|"--")
    | ("++"|"--") <expression>
    | ("!"|"~") <expression>
    | <expression> ("*"|"/"|"%") <expression>
    | <expression> ("+"|"-") <expression>
    | <expression> (BIT_SHIFT_R|BIT_SHIFT_L) <expression>
    | <expression> (">" | ">=" | "<=" | "<") <expression>
    | <expression> ("=="|"!=") <expression>
    | <expression> "&" <expression>
    | <expression> "^" <expression>
    | <expression> "|" <expression>
    | <expression> "&&" <expression>
    | <expression> "||" <expression>
    | <expression> ("%="|"&="|"*="|"+="|"-="|"/="|"^="|"|="|"=") <expression>

<conversion> ::= ("(unsigned int)"|"(unsigned long)") <expression>
    | "(" <type_convert> ")" <expression>
    | "String" "(" <expression> ")"
    | <type_convert> "(" <expression> ")"

<type_convert> ::= "byte" | "char" | "float" | "int" | "long" | "word"

<parameter> ::= | <parameter> "," <expression>
```

Figura 8.1 BNF de la gramática

Dentro de esta gramática se pueden contemplar una serie de terminales que no van ni entrecomillados ni entre los símbolos de mayor y menor. Dichos terminales (exceptuando LOW, HIGH, ANALOG_PIN, INPUT, INPUT_PULLUP y OUTPUT, que son constantes y representan la constante de Arduino homónima) representan los siguientes valores:

- BIT_SHIFT_R y BIT_SHIFT_L representan los operadores >> y << respectivamente.
- ANALOG_PIN representa los pines analógicos, que en Arduino son los pines definidos por el intervalo [A0-A5].
- BINARY_CONST representa las constantes binarias (0b seguido de ceros y unos).
- OCTAL_CONST representa a las constantes en base ocho (0 seguido de valores cuyos dígitos están comprendidos en el intervalo [0-7]).
- HEX_CONST representa a las constantes hexadecimales (0x seguido de valores del cero al quince, siendo los valores del diez al quince las letras [A-F])
- INT_CONST. Es una constante entera (puede ser negativa o positiva).
- FLOAT_CONST. Es una constante decimal (puede ser negativa o positiva).
- CHAR_CONST. Es un valor de carácter.
- STRING_CONST. Es una cadena.
- ID representa el nombre de las variables, funciones, sentencias *declare* y librerías.

En esta gramática se han definido las siguientes características de Arduino (refiriéndonos a él como lenguaje):

- Sentencias *Include* para importar las librerías.
- Declaración de variables, arrays (ambos locales o globales), macros (sentencias *declare*) y funciones.
- Tipos de variables o de retorno de funciones, además de las conversiones que ofrece Arduino (por eso los tipos de conversión están a parte de los tipos normales, no se puede hacer conversión a todo).
- Bucles de tipo *while*, *for* y *do while*.
- Sentencias condicionales de tipo *if-else* y *case-switch*.
- Operaciones aritméticas, lógicas, *booleanas* y a nivel de bit; además de asignación compuesta (asignación y operación, por ejemplo “+=”).
- Sentencias *return*, *break* y *continue*.

No se han declarado algunas características, o bien por su complejidad (punteros, aunque se usan poco en la asignatura); o bien por ser innecesarios o poco recomendados (*goto* es un ejemplo de ello).

8.2 Análisis previos

La gramática definida detecta por sí sola cualquier tipo de error léxico y/o sintáctico. esto lo hace gracias a ANTLR4, que genera los analizadores léxico y sintáctico. El primero de ellos comprobará cualquier tipo de error léxico, además de analizar el código con el fin de obtener los tokens.

Dichos tokens son después enviados al analizador sintáctico, el cuál analiza todos los errores sintácticos (aquellos que tienen que ver con cómo se ha combinado los lexemas introducidos) y genera un *parse tree*. Dicho árbol es convertido a un AST con el fin de facilitar las siguientes fases de análisis.

Para comprobar los errores de carácter semántico se ha desarrollado una clase que recibe el AST y analiza los errores (si es que los hay) de dicho tipo. Dichos errores pueden ser de tipado, de declaración de variables o funciones y si el acceso a una variable es correcto o no (por ejemplo, intentar acceder a una variable de otra función).

La última fase de análisis es el análisis de advertencias. Esta es una fase extra con respecto a las esperadas en la estructura del compilador definida en el punto 3.3. Advierte al usuario de sentencias del código que, aunque no tengan nada incorrecto en las anteriores fases, o bien no sean recomendadas (por ejemplo, los bucles); o bien no estén implementadas (como es el caso de algunos métodos de las librerías).

En caso de que se detecte algún error o advertencia, estos serán posteriormente mostrados mediante la consola incorporada en la interfaz gráfica. Entre la información mostrada de ambos estará el tipo de error o advertencia, así como la línea y columna en la que se encuentran y el mensaje que indica el error correspondiente.

8.3 Generación de Código

Una vez superadas todas las fases de análisis descritas en el anterior apartado, se procede a la generación de código. Dicha generación convertirá los datos almacenados en el AST a código Python. Se debe destacar que, si hay algún error (pero no advertencia) en las anteriores fases de análisis, nunca se ejecutará el código generado.

Hay una serie de características que tiene el generado de código:

- Debido a que Python define de una manera distinta las variables globales y las variables locales, a principio de cada método se debe declarar cada variable global empleada utilizando la palabra reservada de Python “global”.
- En cuanto a los bucles:
    - El bucle *for* de Arduino y el de Python difieren en estructura, con lo que la sintaxis debe ser adaptada de un lenguaje a otro, no siendo así con el *while*.
    - El bucle *do while* no existe en Python, con lo que se debe hacer un bucle *while* infinito y utilizar una sentencia condicional al final del bucle que, si se llega a cumplir, ejecute la sentencia break y, por tanto, imite el comportamiento del bucle.
- Python no soporta los operadores “++” y “--”, con lo que dichas operaciones se traducen a variable “+=” o “-=” (respectivamente) uno.
- Todas las librerías básicas (siendo estas “Standard”, “Serial” y “String”) se importan por defecto, ya que no requieren de una sentencia *Include* en los *sketches*.
- La sintaxis de las sentencias *case-switch* en Python cambia ligeramente, siendo *case-match*. El resto de la expresión es prácticamente idéntico.

Si por alguna razón tiene lugar alguna excepción durante el proceso de generación de código, se mostraría mediante la consola de la interfaz gráfica que un error ha sucedido durante la compilación (esto también incluye las fases de análisis anteriores).

8.4 Ejecución

Una vez finalizada la fase de generación de código, se procederá a ejecutarlo. Esto se realiza a través de dos clases implementadas en el módulo “commands”. Dichas clases son “Setup” y “Loop”, que llamarán a las funciones homónimas dentro del script generado.

El código se ejecutará de “manera infinita”, aprovechándose del método “after” de tkinter, que permite programar una llamada a una función cada n milisegundos. De esta manera, se genera una especie de bucle infinito, aunque realmente no lo sea.

El código accederá a la información de los robots a través de los métodos de las librerías programadas y ejecutará las acciones que se le indiquen según la información proporcionada por la parte de animación del robot.

Capítulo 9. Desarrollo de las Pruebas

En este capítulo se resumirá el resultado de las pruebas desarrolladas para el proyecto, empezando por las pruebas unitarias, seguido de las pruebas de integración, continuando con las pruebas de usabilidad y accesibilidad y finalizando con las pruebas de rendimiento.

9.1 Pruebas Unitarias

En este apartado se mostrarán los resultados de las pruebas unitarias descritas en el apartado 6.6.1, así como las anotaciones realizadas sobre estas pruebas (si las hay).

Pruebas de gramática		
Escribir un sketch con operaciones aritméticas	Los tokens generados se corresponden con los esperados	Correcto
Escribir un programa con definiciones de arrays, que sean de una dimensión y de varias dimensiones		Correcto
Escribir un sketch con accesos a arrays		Correcto
Escribir un sketch con operaciones a nivel de bit		Correcto
Escribir un sketch con sentencias break y continue		Correcto
Escribir un sketch con operaciones de comparación		Correcto
Escribir un sketch con operaciones compuestas (es decir, +=, -=, etc.)		Correcto
Escribir un sketch que contenga un bucle do while		Correcto
Escribir un sketch que contenga un else		Correcto
Escribir un sketch que contenga un bucle for		Correcto
Escribir un sketch que contenga un if		Correcto
Escribir un sketch que contenga una sentencia return		Correcto
Escribir un sketch que contenga un bloque switch case		Correcto
Escribir un sketch comprobando las asignaciones a variables		Correcto
Escribir un sketch que contenga un bucle while		Correcto

Tabla 9.1 Pruebas de la gramática

Pruebas de AST		
Escribir un sketch que genere nodos de sentencias include	El nombre de los archivos especificados en la prueba debe coincidir con los especificados en el sketch	Correcto
Escribir un sketch que genere definiciones de variables globales	El tipo, nombre y valor (si lo hay) de la variable debe de ser aquel que se ha declarado en el sketch. Las variables que se declaren como constantes deben de tener el valor de is_const a verdadero. Si la definición global es un array, el tamaño de ese array debe coincidir con el que se haya definido.	Correcto
Escribir un sketch que contenga definiciones	El tipo de las funciones debe coincidir con el tipo definido en el sketch.	Correcto

de funciones propias	El nombre de la función debe coincidir con el nombre definido en el sketch. Los parámetros deben coincidir en número, nombre y tipo con los definidos en el sketch. El número de sentencias de la función debe ser el mismo que el definido en el sketch.	
Escribir un sketch que contenga los diferentes valores terminales del lenguaje de programación	El tipo y valor de esos terminales debe coincidir con lo definido en el sketch El tipo de las variables a las que sean asignados debe ser el mismo que se ha establecido en el sketch.	Correcto
Escribir un sketch que contenga llamadas a funciones	El nombre de la llamada a la función debe coincidir con el definido en el sketch. El número y tipo de parámetros debe coincidir con lo definido en la función que se está llamando del sketch.	Correcto
Escribir un sketch que contenga sentencias condicionales	Las condiciones de las sentencias deben coincidir con lo establecido en el sketch. Las sentencias dentro de un if deben coincidir con las establecidas en el sketch. Ídem para else. Las sentencias case switch deben de tener la misma variable sobre la que cambian que en el sketch. El valor del case será el mismo que en el sketch. Las sentencias de case y default deberán ser el mismo número que las definidas en el sketch.	Correcto
Escribir un sketch que contenga bucles	Se debe comprobar que el tipo de bucle es el establecido en el sketch para cada bucle. La condición de parada del bucle debe ser la misma que la establecida en el sketch. Para los bucles for, el valor del incremento y de la definición inicial deberán coincidir con lo establecido en el sketch.	Correcto
Escribir un sketch que contenga asignaciones a variables	El nombre de la variable debe coincidir con lo establecido en el sketch. El valor de la expresión que se asigna debe coincidir con lo establecido en el sketch. El operador de asignación debe coincidir con el establecido en el sketch.	Correcto
Escribir un sketch que contenga operaciones a nivel de bits	El valor establecido a izquierda y derecha del operador debe ser el mismo que el establecido en el sketch. El valor del operador debe de ser el mismo que el establecido en el sketch.	Correcto
Escribir un sketch que contenga definiciones	El nombre, tipo y valor asignado (si hay este último) debe coincidir con lo establecido en el	Correcto

locales (dentro de función)	sketch. Si la variable es una constante, el valor de is_constant debe ser verdad.	
Escribir un sketch que contenga operaciones del tipo ++ o --	El nombre de la variable a incrementar y el operador debe coincidir con lo establecido en el sketch.	Correcto
Escribir un sketch que contenga operaciones de control de ejecución (tipo return, break o continue)	El tipo del nodo debe ser el correspondiente a la sentencia del sketch. El valor de return debe coincidir con lo establecido en el sketch.	Correcto
Escribir un sketch que contenga operaciones aritméticas	El valor establecido a izquierda y derecha del operador debe ser el mismo que el establecido en el sketch. El valor del operador debe de ser el mismo que el establecido en el sketch.	Correcto
Escribir un sketch que contenga operaciones de comparación y de booleanos	El valor establecido a izquierda y derecha del operador debe ser el mismo que el establecido en el sketch. El valor del operador debe de ser el mismo que el establecido en el sketch.	Correcto
Escribir un sketch que compruebe la definición de arrays	El nombre, dimensiones, tamaños y elementos del array deben coincidir con lo establecido en el sketch.	Correcto
Escribir un sketch con accesos a arrays	El nombre del array y el índice accedido deben coincidir con lo establecido en el sketch.	Correcto

Tabla 9.2 Pruebas del AST

Pruebas de comprobaciones de errores de código		
Escribir un sketch que contenga errores sintácticos	El tipo del error debe coincidir con el tipo de error esperado. El número de errores de cada sketch debe coincidir con los errores introducidos en el sketch. El mensaje de error que se muestre por consola deberá coincidir con el mensaje esperado.	Correcto
Escribir un sketch que contenga errores de definición de setup y loop		Correcto
Escribir un sketch que contenga errores de tipado		Correcto
Escribir un sketch que contenga errores de declaración de variables y funciones		Correcto
Escribir un sketch que contenga errores de break, continue y return		Correcto
Escribir un sketch que contenga errores de acceso a arrays		Correcto
Escribir un sketch que contenga errores relacionados con las librerías de código		Correcto

Escribir varios sketches que no contengan fallo alguno	El número de errores deberá ser 0, coincidiendo con el compilador de Arduino original	Correcto

Tabla 9.3 Pruebas de comprobaciones de errores de código

Pruebas de advertencias		
Escribir un sketch que contenga las sentencias de código que tengan advertencias	El número de advertencias, tipo y mensaje deben coincidir con los esperados según el contenido del sketch.	Correcto

Tabla 9.4 Pruebas de advertencias

En la siguiente figura se puede observar como todas las pruebas realizadas han sido superadas con éxito:

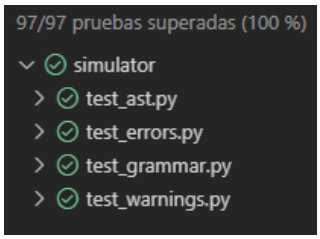


Figura 9.1 Pruebas ejecutadas desde Visual Studio Code (con unittest)

9.2 Pruebas de Integración y del Sistema

En este apartado se mostrarán los resultados de las pruebas de integración del sistema, mostrando las anotaciones realizadas sobre los resultados de cada una.

Subsistema de Compilador	
Prueba	**Resultado Esperado**
Error léxico	El sistema muestra por consola los errores léxicos
Resultado	Comportamiento esperado
Prueba	**Resultado Esperado**
Error sintáctico	El sistema muestra por consola los errores sintácticos
Resultado	Comportamiento esperado
Prueba	**Resultado Esperado**
Error de código	El sistema muestra por consola los errores de código
Resultado	Comportamiento esperado
Prueba	**Resultado Esperado**
Error léxico y sintáctico	El sistema muestra por consola los errores léxicos
Resultado	Comportamiento esperado
Prueba	**Resultado Esperado**
Error léxico y de código	El sistema muestra por consola los errores léxicos
Resultado	Comportamiento esperado
Prueba	**Resultado Esperado**
Error sintáctico y de código	El sistema muestra por consola los errores sintácticos
Resultado	Comportamiento esperado
Prueba	**Resultado Esperado**
Los tres errores a la vez	El sistema muestra por consola los errores léxicos
Resultado	Comportamiento esperado
Prueba	**Resultado Esperado**
Sin errores	El sistema muestra los mensajes programados por el usuario
Resultado	Comportamiento esperado

Tabla 9.5 Test de integración de subsistema de compilador

Subsistema de representación gráfica	
Prueba	**Resultado Esperado**
Movimiento de teclado	El robot se mueve en la dirección que pretende el usuario
Resultado	Comportamiento esperado
Prueba	**Resultado Esperado**
Movimiento de código	El robot se mueve según lo esperado en el código introducido
Resultado	Comportamiento esperado
Prueba	**Resultado Esperado**
Cambio de robot	El robot escogido se muestra una vez se pulsa ejecutar
Resultado	Comportamiento esperado

Tabla 9.6 Test de integración de subsistema de representación gráfica

Subsistema de consola	
Prueba	**Resultado Esperado**
Se escribe una sentencia print en el código	Se muestra el mensaje escrito en el print
Resultado	Comportamiento esperado
Prueba	**Resultado Esperado**
Se escribe un bucle while (advertencia)	Se muestra una advertencia sobre el uso
Resultado	Comportamiento esperado
Prueba	**Resultado Esperado**
Se escribe un error de código	Se muestra el error por consola
Resultado	Comportamiento esperado

Tabla 9.7 Test de integración de subsistema de consola

Subsistema de librerías	
Prueba	**Resultado Esperado**
Se prueba un método implementado	El método probado funciona correctamente
Resultado	Comportamiento esperado
Prueba	**Resultado Esperado**
Se prueba un método sin implementar	Se muestra una advertencia por consola
Resultado	Comportamiento esperado

Tabla 9.8 Test de integración de subsistema de librerías

9.3 Pruebas de Usabilidad y Accesibilidad

En este apartado se explicarán y mostrarán las pruebas de usabilidad y accesibilidad realizadas sobre el sistema. Comenzando por las pruebas de usabilidad y terminando con las de accesibilidad.

9.3.1 Pruebas de Usabilidad

En este apartado se mostrarán las pruebas de usabilidad realizadas por los diferentes usuarios que han probado la aplicación.

9.3.1.1 Preguntas de carácter general

En primer lugar, se mostrarán los resultados obtenidos de las preguntas de carácter general, primero de forma global y después usuario a usuario.

9.3.1.1.1 Resultados generales de las pruebas

Los siguientes son los resultados generales de las preguntas de carácter general:

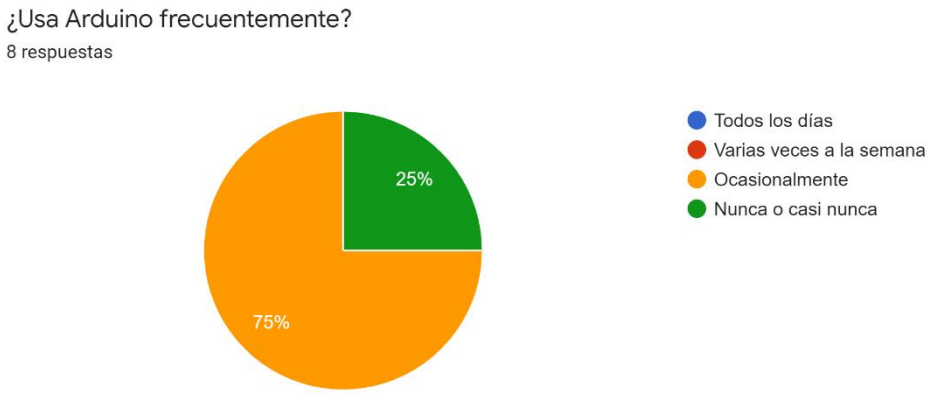


Figura 9.2 Uso frecuente de Arduino

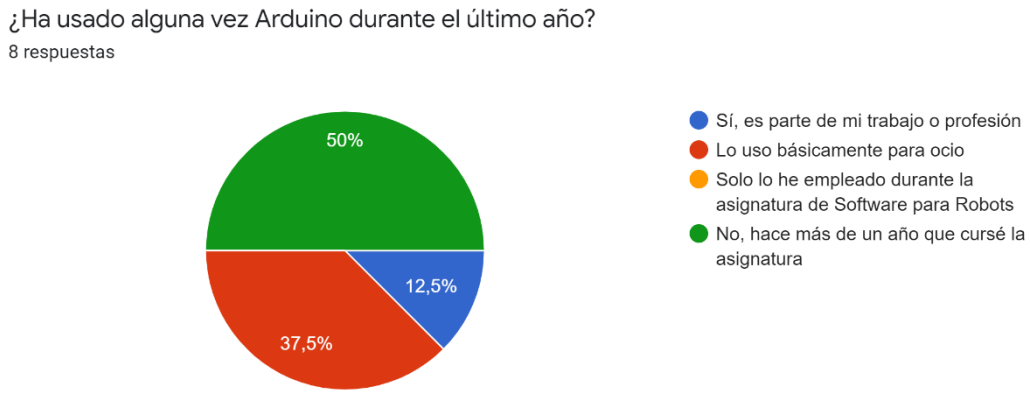


Figura 9.3 Uso de Arduino durante el último año

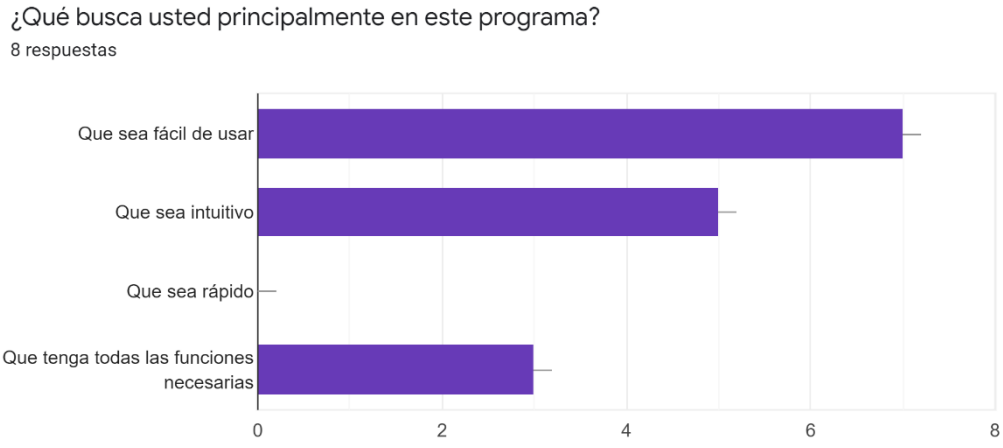


Figura 9.4 Expectaciones sobre el programa

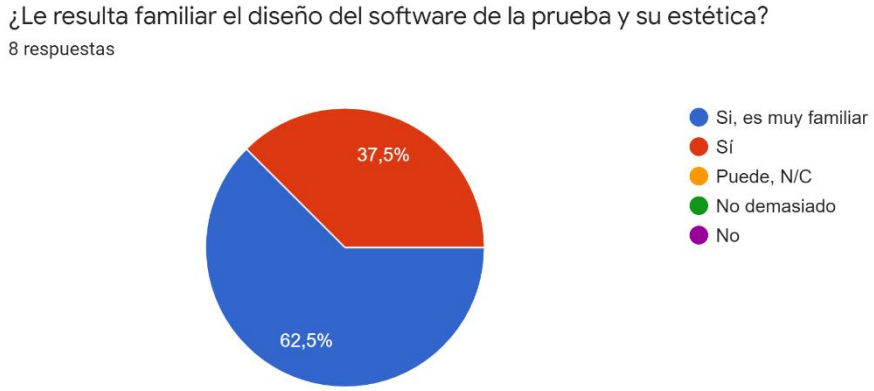


Figura 9.5 Similitud de diseño

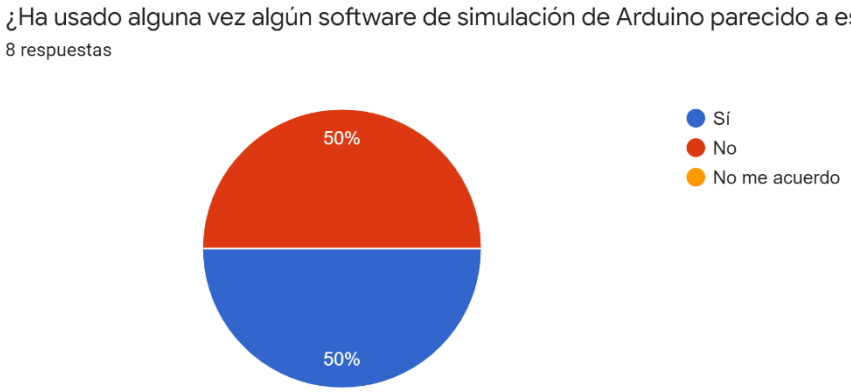


Figura 9.6 Uso de software de simulación

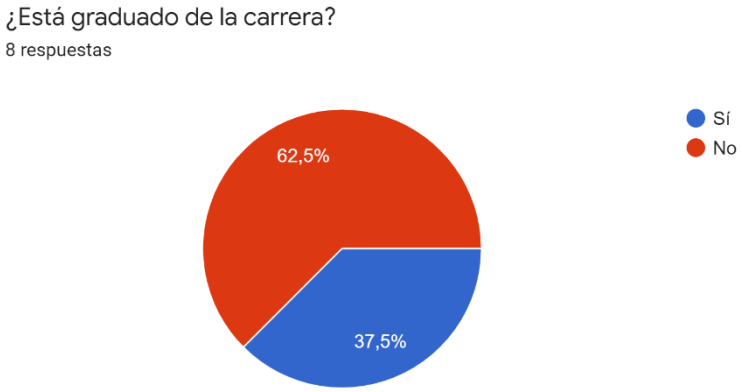


Figura 9.7 Usuario graduado

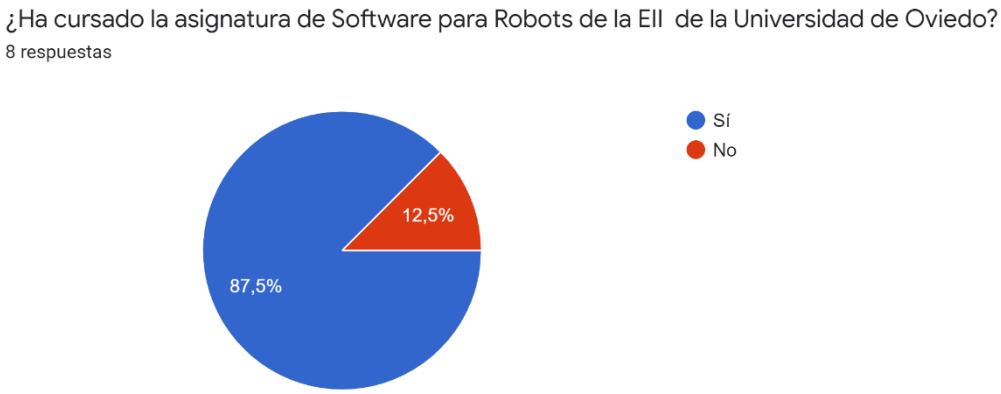


Figura 9.8 Usuario que cursó la asignatura

9.3.1.1.2 Resultados individuales de las pruebas

Los siguientes son los resultados individuales de las pruebas:

1 - ¿Usa un Arduino frecuentemente?
1. Todos los días 2. Varias veces a la semana 3. Ocasionalmente 4. Nunca o casi nunca
2 - ¿Ha usado alguna vez Arduino durante el último año?
1. Si, es parte de mi trabajo o profesión 2. Lo uso básicamente para ocio 3. Solo lo he empleado durante la asignatura de software para robots 4. No, hace más de un año que cursé la asignatura
3 - ¿Qué busca Vd. Principalmente en este programa?
1. Que sea fácil de usar 2. Que sea intuitivo 3. Que sea rápido 4. Que tenga todas las funciones necesarias
4 - ¿Le resulta familiar el diseño del software de la prueba y su estética?
1. Sí, es muy familiar 2. Sí 3. Puede 4. No demasiado 5. No
5 - ¿Ha usado alguna vez algún software de simulación de Arduino parecido a este?
1. Sí 2. No 3. No me acuerdo
6 - ¿Está graduado de la carrera?
1. Sí 2. No
7 - ¿Ha cursado la asignatura?
1. Sí 2. No

Tabla 9.9 Preguntas de carácter general

N.º	Usuario 1	Usuario 2	Usuario 3	Usuario 4	Usuario 5	Usuario 6	Usuario 7	Usuario 8
1	3	3	3	3	4	4	3	3
2	2	2	2	4	4	4	4	1
3	1	1, 2, 4	1, 2	1	1, 2	1, 4	1	1, 2, 4
4	1	2	2	1	1	1	1	2
5	2	2	2	2	1	1	1	1
6	1	1	2	2	2	2	1	2
7	1	1	1	1	1	2	1	1

Tabla 9.10 Resultados individuales de las preguntas de carácter general

9.3.1.2 Actividades guiadas

En este apartado se expondrán los resultados de las actividades guiadas, todas ellas fueron realizadas por un grupo reducido de personas con respecto a las preguntas cortas y las preguntas generales.

Primero se mostrarán los resultados globales de las pruebas mediante una gráfica. Después se mostrarán los resultados individuales de los cuatro usuarios a los que se les ha solicitado la cumplimentación de estas cuatro pruebas.

9.3.1.2.1 Resultados generales de las pruebas

A continuación, se muestran a través de unos gráficos circulares, los resultados obtenidos en las actividades guiadas de una forma general.

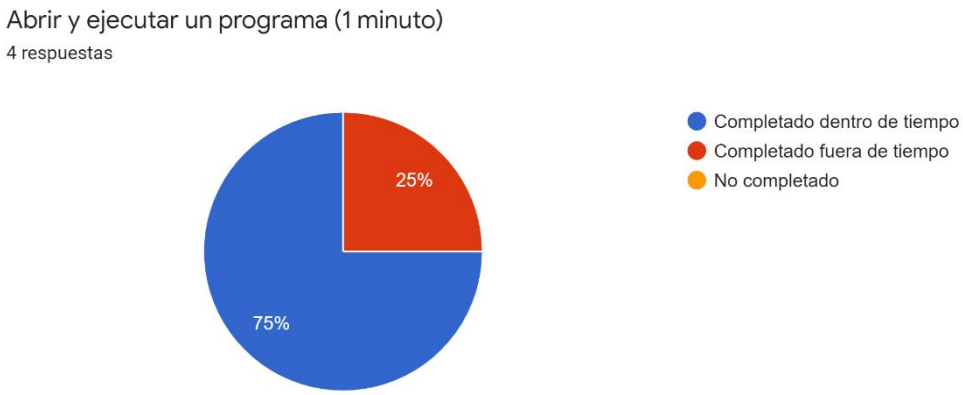


Figura 9.9 Abrir y ejecutar un programa

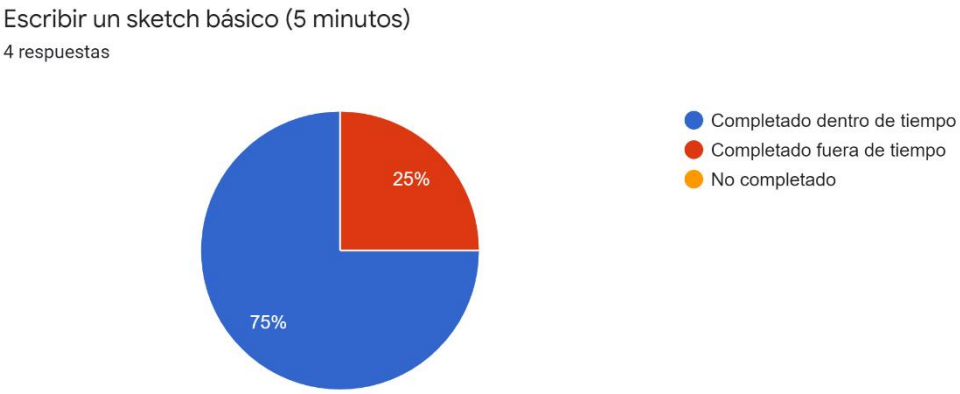


Figura 9.10 Escribir un sketch básico

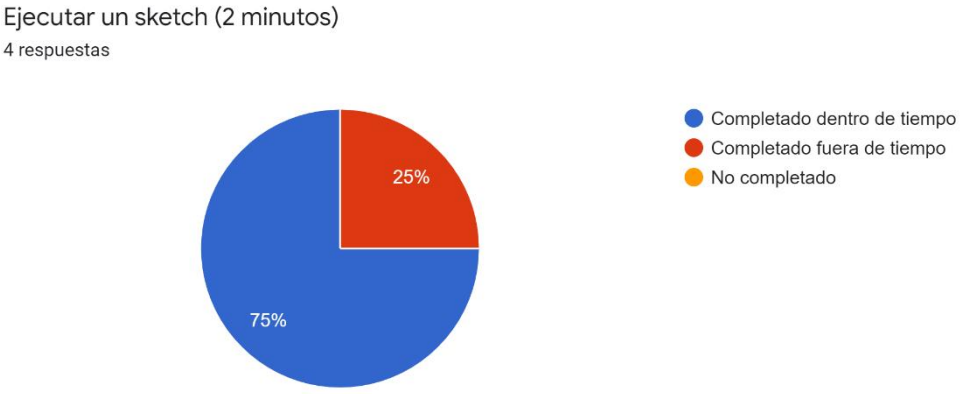


Figura 9.11 Ejecutar un sketch

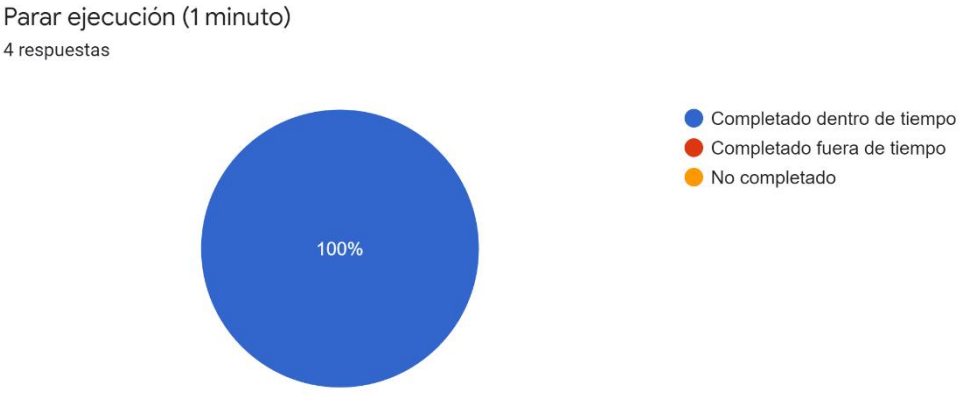


Figura 9.12 Parar ejecución

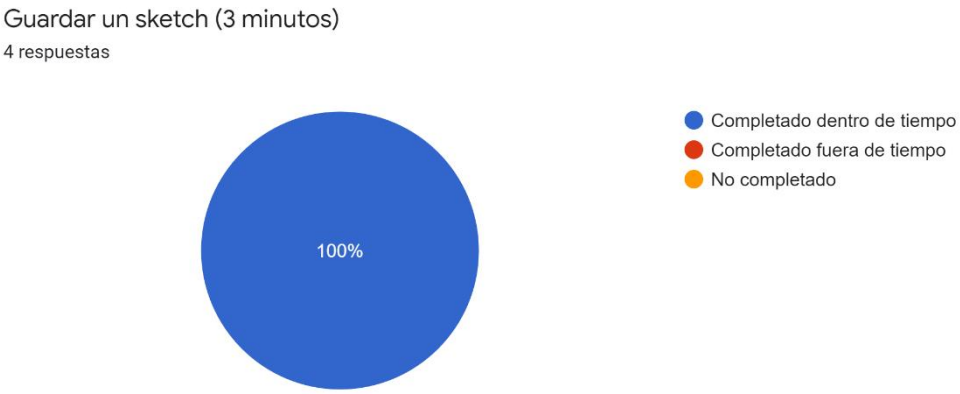


Figura 9.13 Guardar un sketch

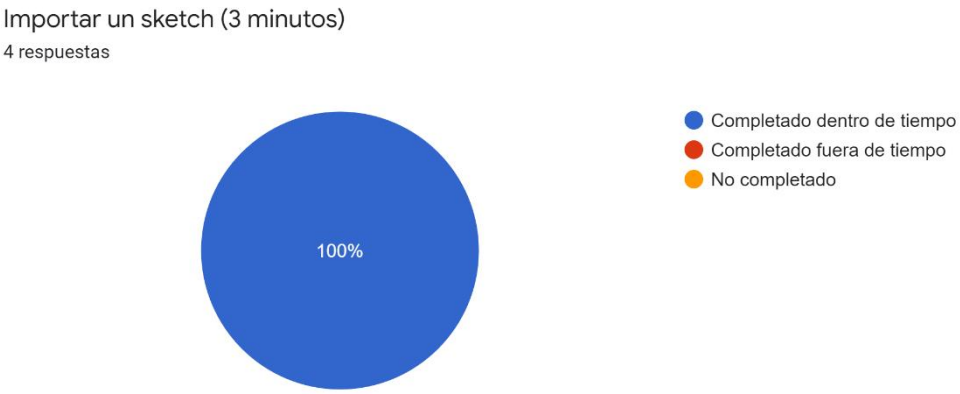


Figura 9.14 Importar un sketch

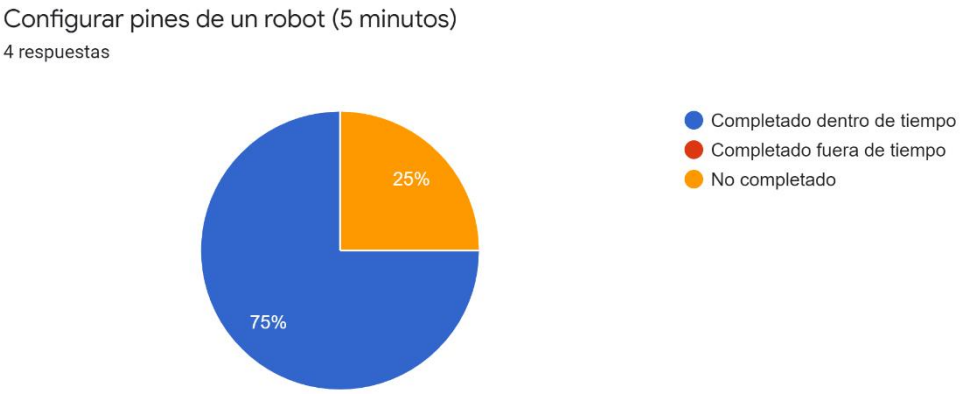


Figura 9.15 Configurar pines de un robot

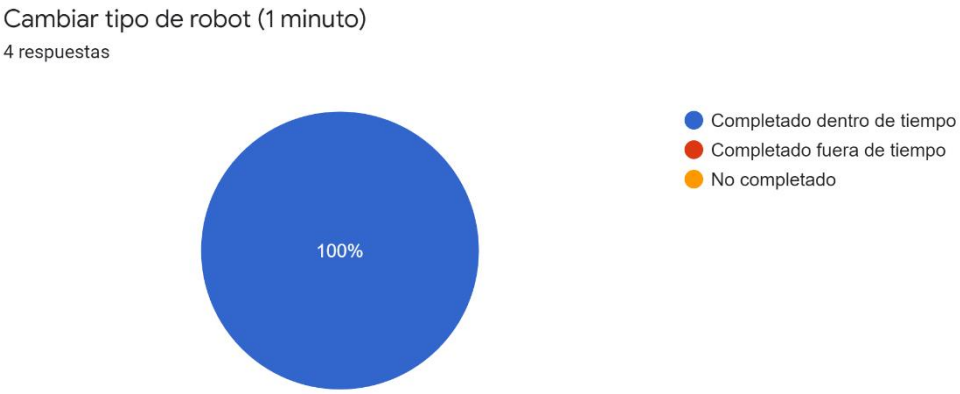


Figura 9.16 Cambiar tipo de robot

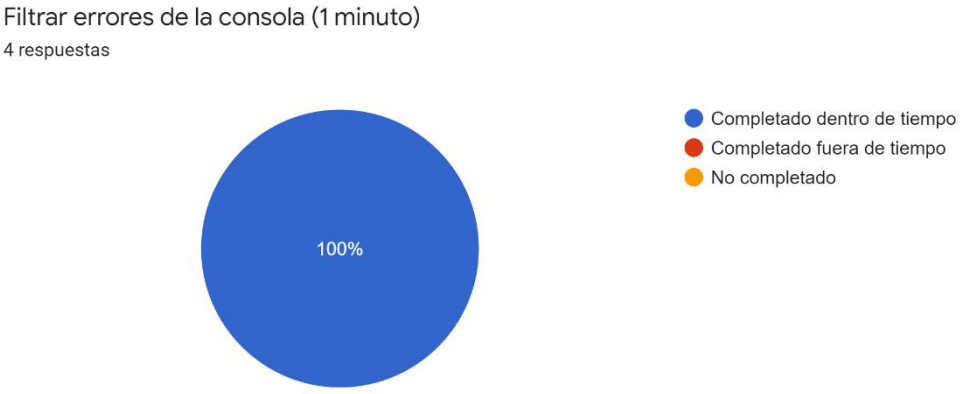


Figura 9.17 Filtrar errores de la consola

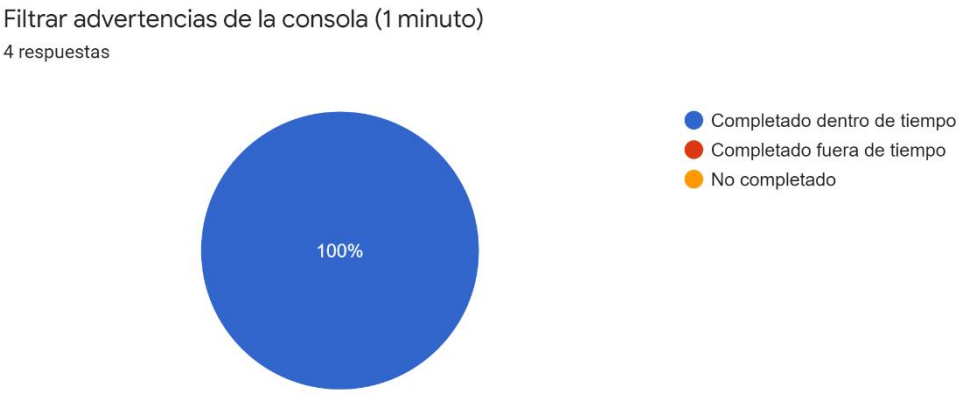


Figura 9.18 Filtrar advertencias de la consola

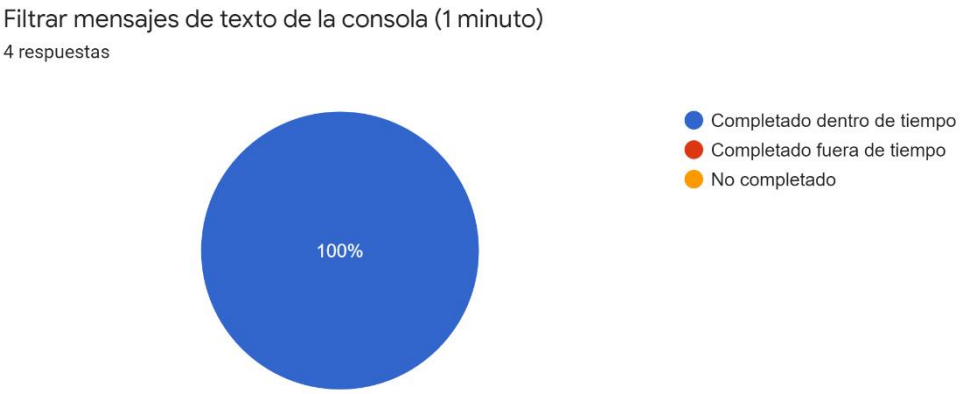


Figura 9.19 Filtrar mensajes de texto de la consola

Figura 9.20 Crear un nuevo sketch desde el menú archivo

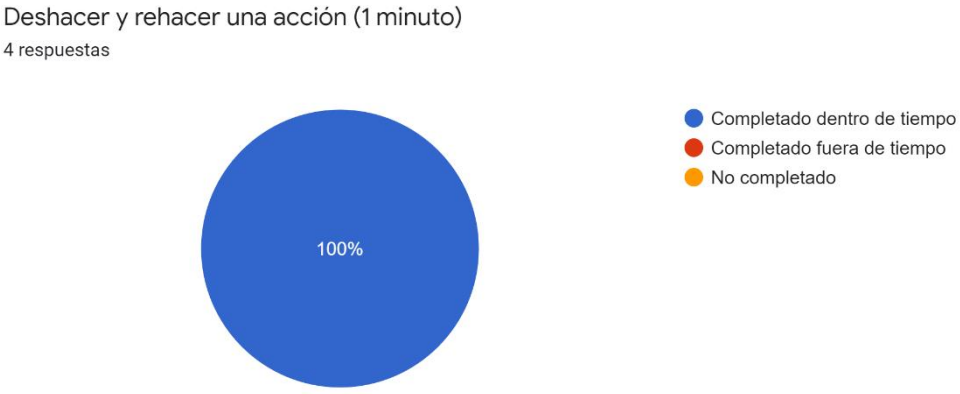


Figura 9.21 Deshacer y rehacer una acción

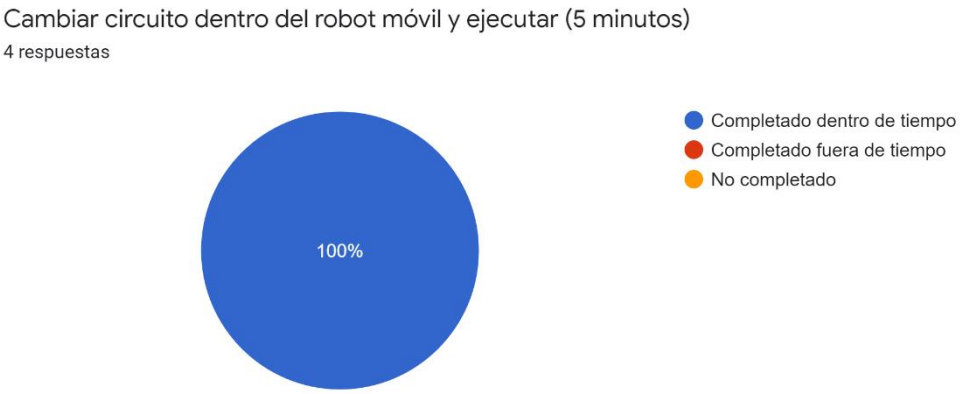


Figura 9.22 Cambiar circuito dentro del robot móvil y ejecutar

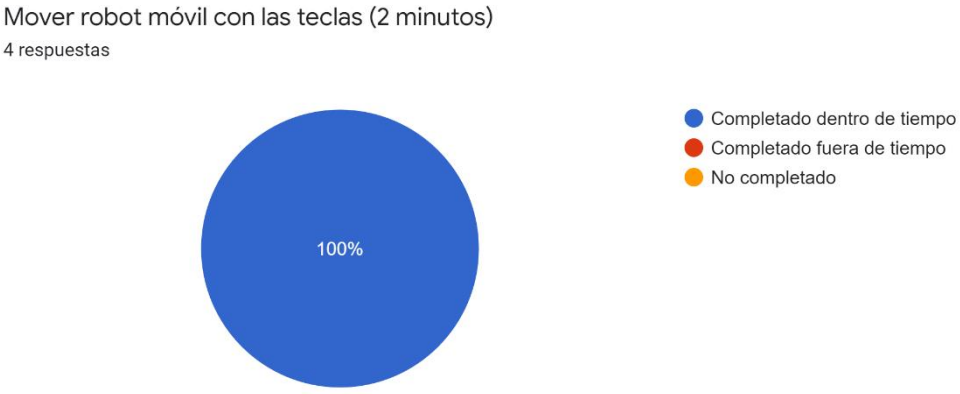


Figura 9.23 Mover robot móvil con las teclas

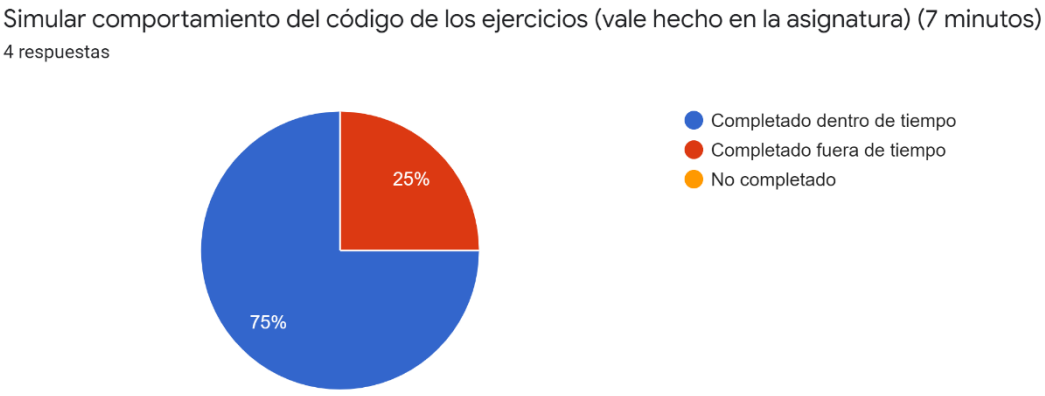


Figura 9.24 Simular comportamiento del código de los ejercicios

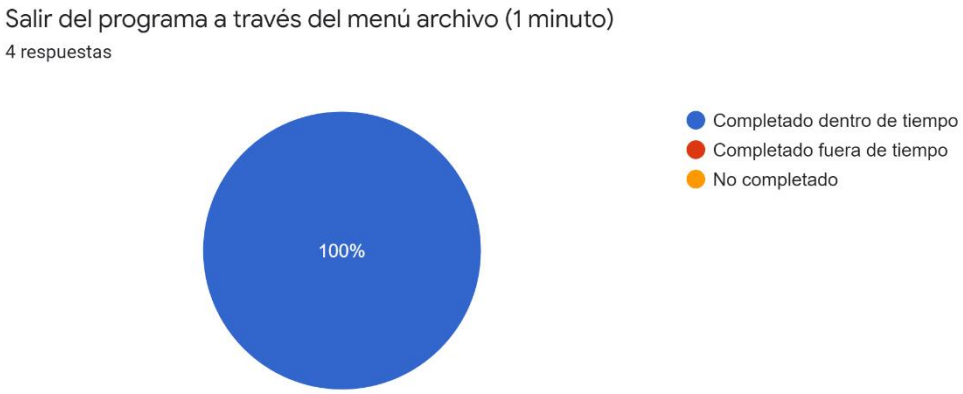


Figura 9.25 Salir del programa a través del menú archivo

9.3.1.2.2 Resultados individuales de las pruebas

A continuación, se muestra una tabla con los resultados individuales de cada usuario que ha realizado las actividades:

Actividad	**Tiempo Máximo (minutos)**	**Realizado con éxito**			
		Usuario 1	**Usuario 2**	**Usuario 3**	**Usuario 4**
Abrir y ejecutar el programa	< 1	Si	Sin tiempo	Si	Si
Escribir un sketch básico	< 5	Si	Sin tiempo	Si	Si
Ejecutar sketch	< 2	Sin tiempo	Si	Si	Si
Parar ejecución	< 1	Si	Si	Si	Si
Guardar un sketch	< 3	Si	Si	Si	Si
Importar un sketch	< 3	Si	Si	Si	Si
Configurar pines de un robot	< 5	Si	Si	No	Si
Cambiar tipo de robot	< 1	Si	Si	Si	Si
Filtrar errores de la consola	< 1	Si	Si	Si	Si
Filtrar advertencias de la consola	< 1	Si	Si	Si	Si
Filtrar mensajes de texto de la consola	< 1	Si	Si	Si	Si
Crear un nuevo sketch desde archivo	< 2	Si	Si	Si	Si
Deshacer y rehacer una acción	< 1	Si	Si	Si	Si
Cambiar circuito dentro del robot móvil	< 5	Si	Si	Si	Si

y ejecutar					
Mover robot móvil con las teclas	< 2	Si	Si	Si	Si
Simular comportamiento de código de los ejercicios	< 7	Si	Si	Si	Si
Salir del programa a través del menú archivo	< 1	Si	Si	Si	Si

Tabla 9.11 Actividades guiadas

9.3.1.3 Preguntas cortas sobre la aplicación

A continuación, se expondrán los resultados obtenidos de las preguntas cortas realizadas a los usuarios que han probado la aplicación. Primero se realizará un resumen general sobre las contestaciones realizadas por los usuarios y después se resumirán sus respuestas de forma individual.

9.3.1.3.1 Resultados generales de las pruebas

Los siguientes son los resultados generales de las preguntas cortas sobre la aplicación:

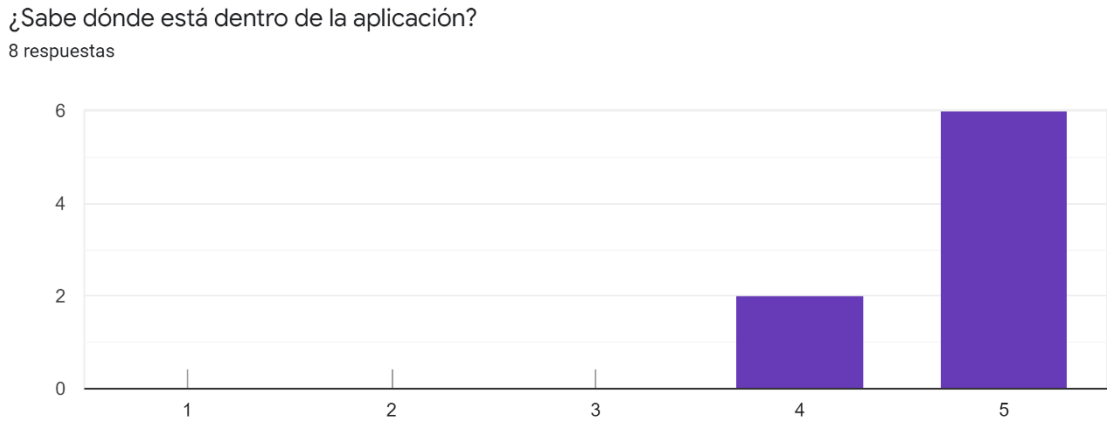


Figura 9.26 Navegabilidad de la aplicación

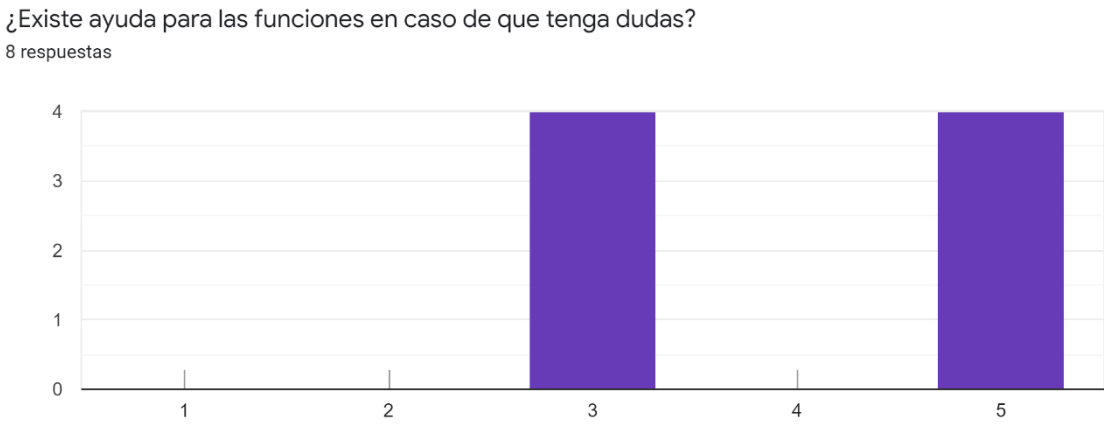


Figura 9.27 Ayuda de la aplicación

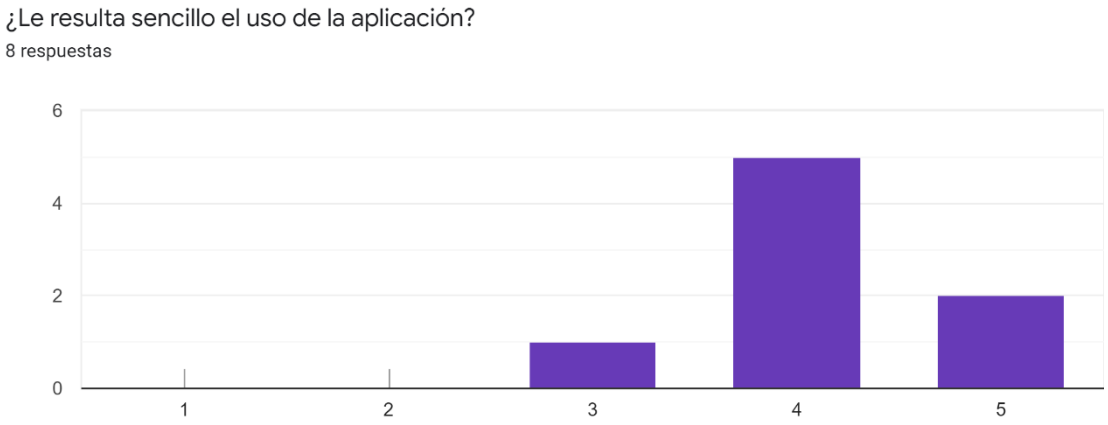


Figura 9.28 Sencillez de la aplicación

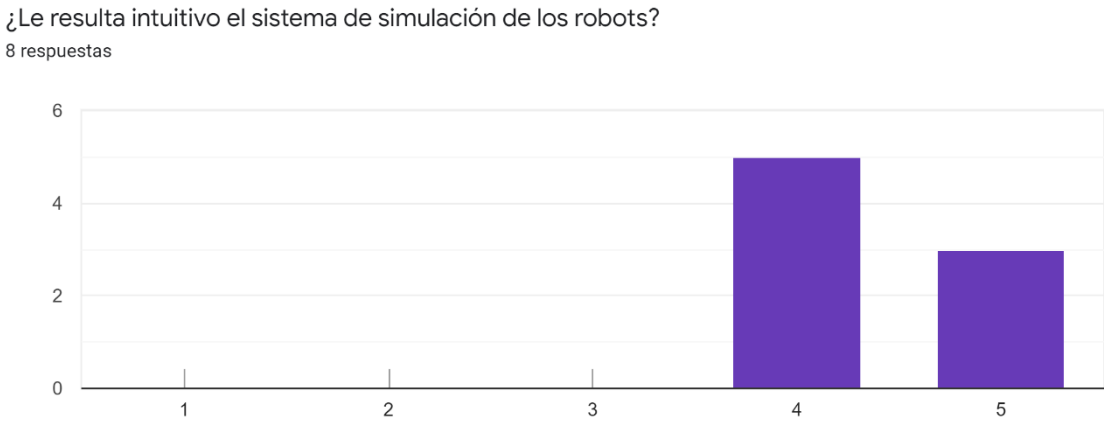


Figura 9.29 Representación gráfica intuitiva

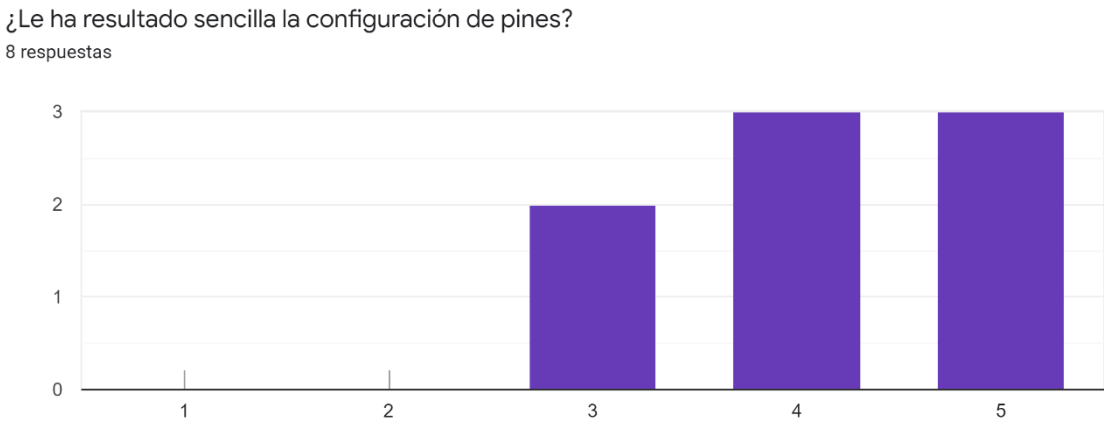


Figura 9.30 Sencillez de configuración de pines

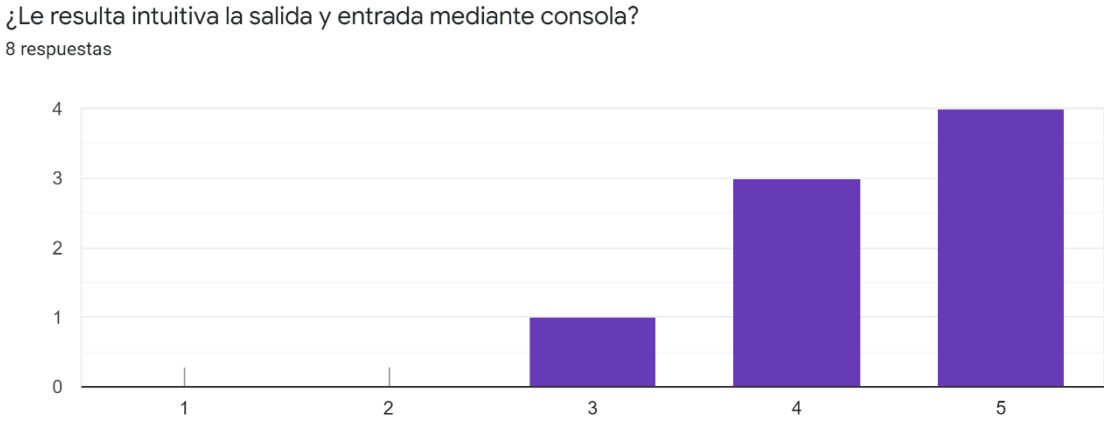


Figura 9.31 Consola intuitiva

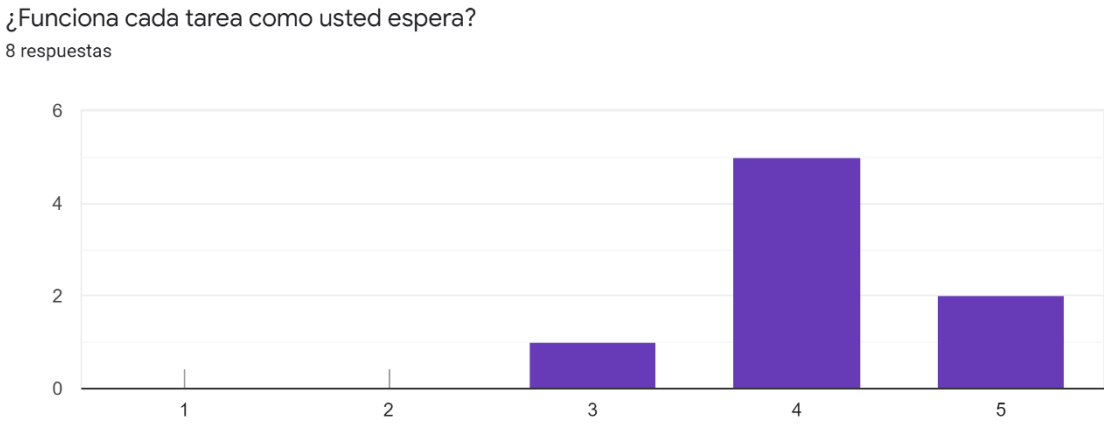


Figura 9.32 Funcionamiento de cada tarea

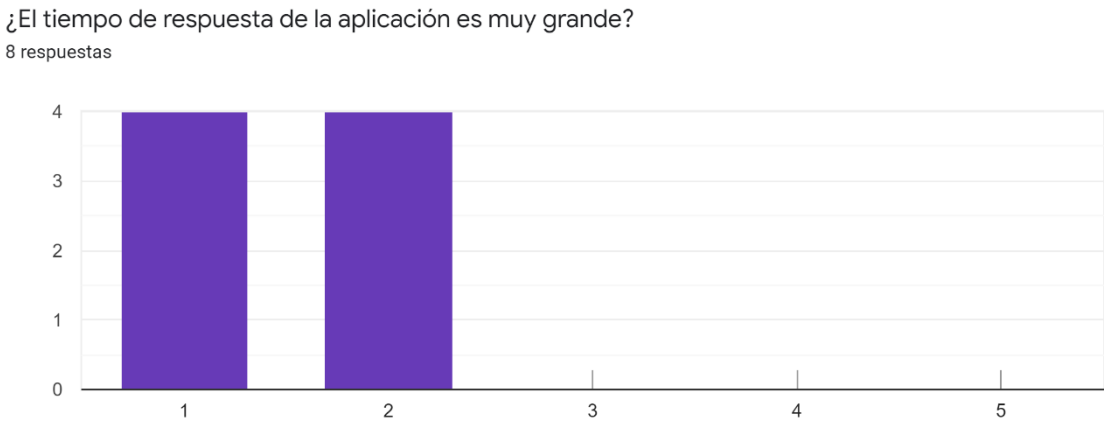


Figura 9.33 Tiempo de respuesta

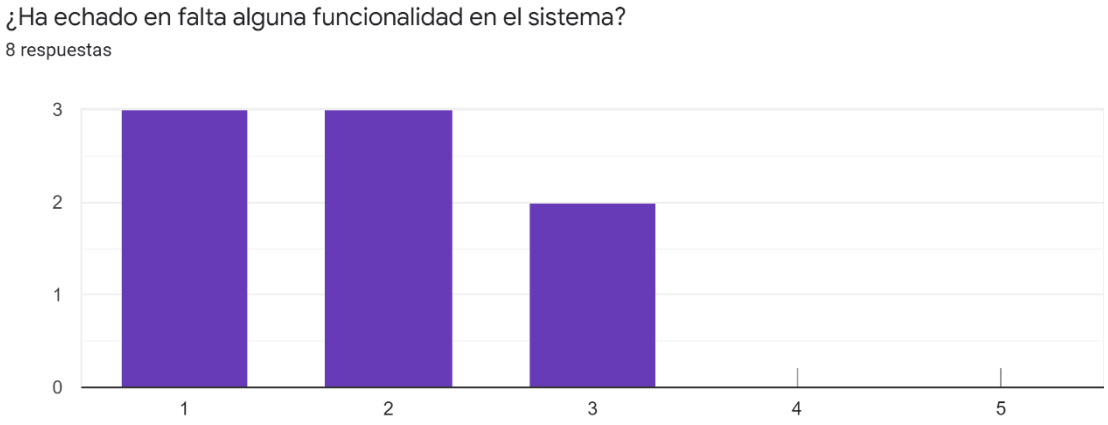


Figura 9.34 Falta de funcionalidad

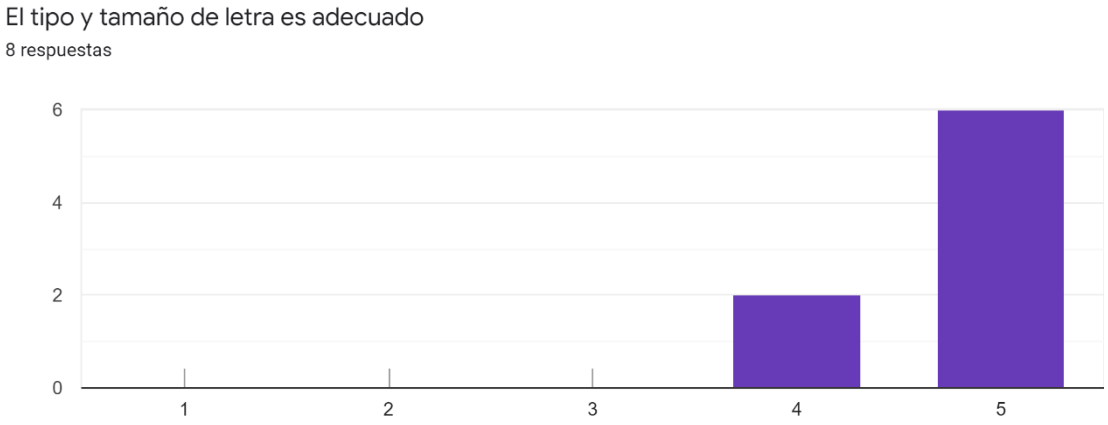


Figura 9.35 Tipo y tamaño de letra

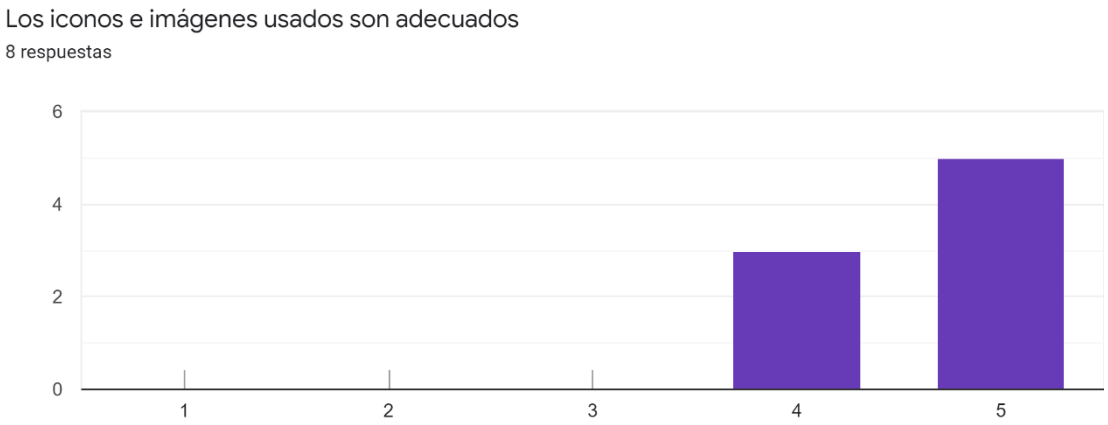


Figura 9.36 Iconos e imágenes adecuadas

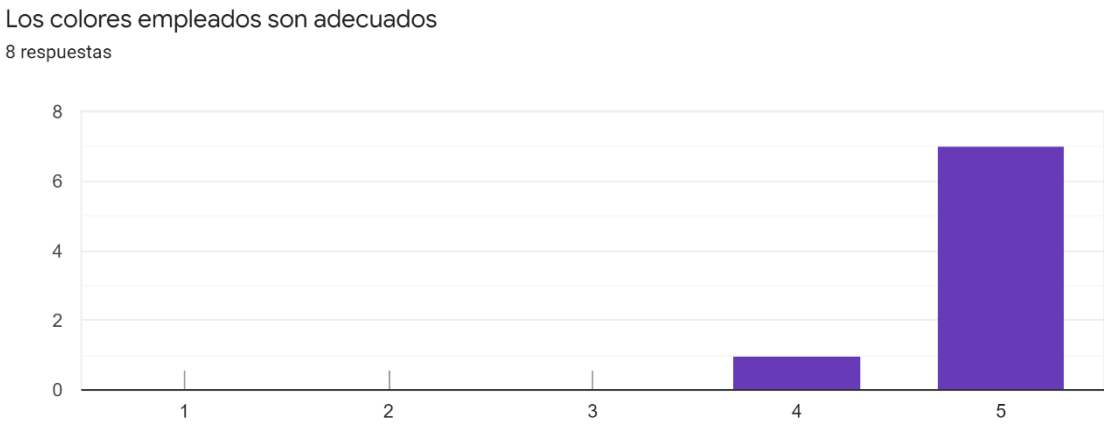


Figura 9.37 Colores adecuados

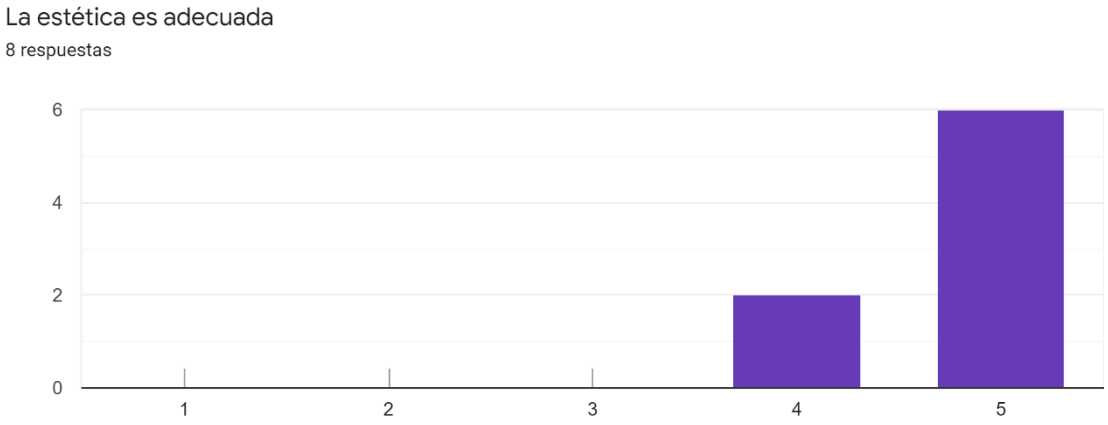


Figura 9.38 Estética adecuada

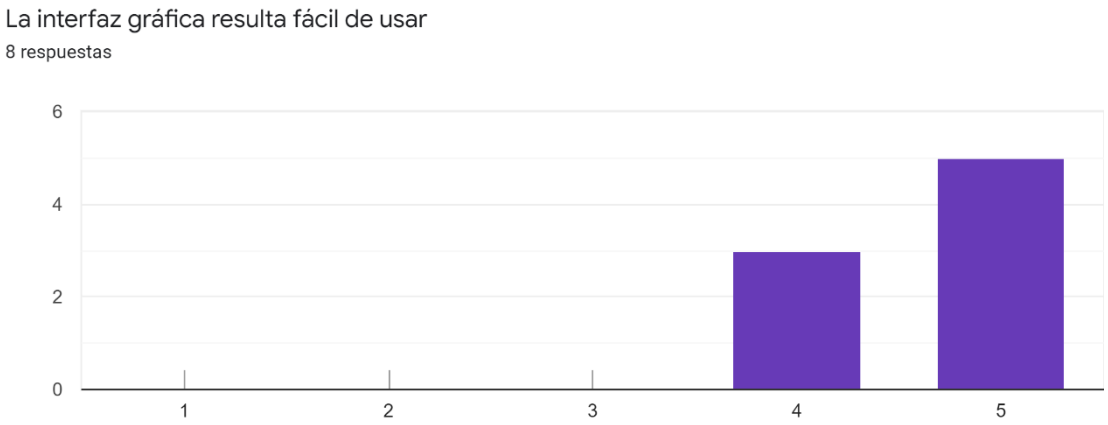


Figura 9.39 Interfaz gráfica sencilla

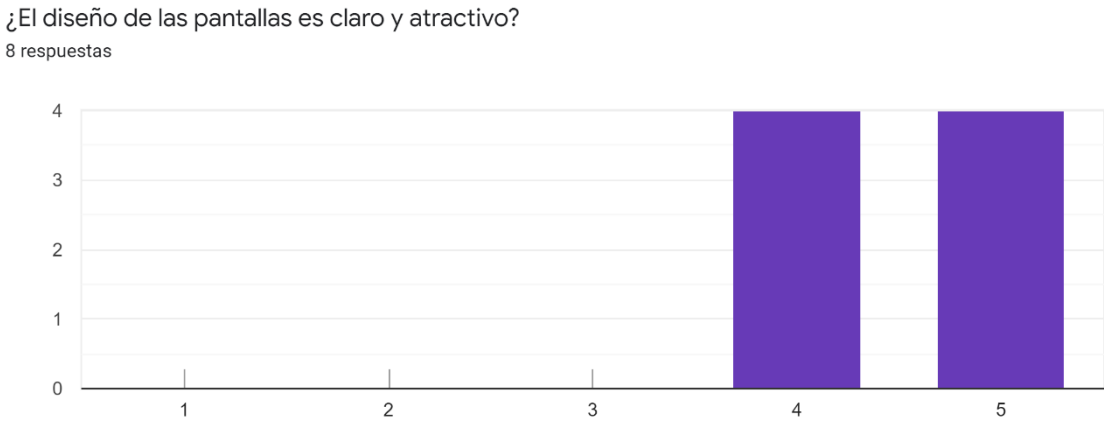


Figura 9.40 Diseño de pantallas claro y atractivo

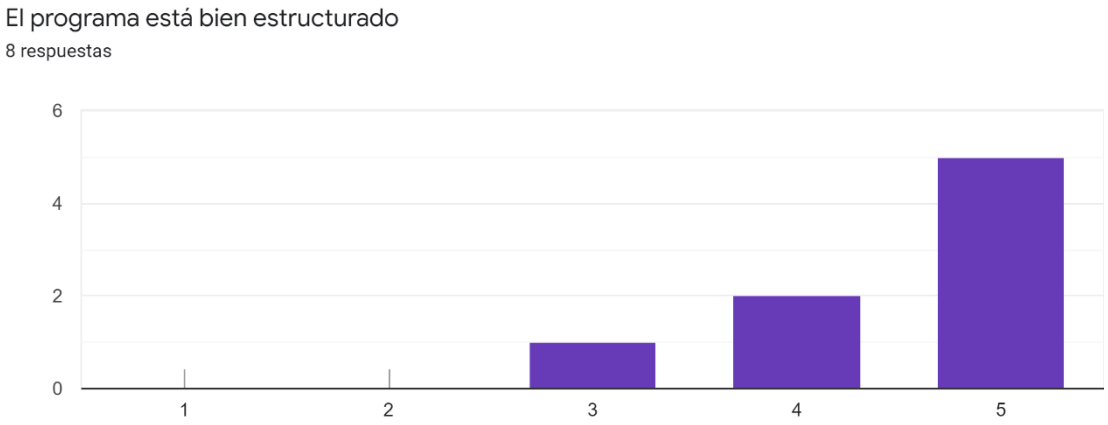


Figura 9.41 Programa bien estructurado

Figura 9.42 Necesidad del manual de ayuda

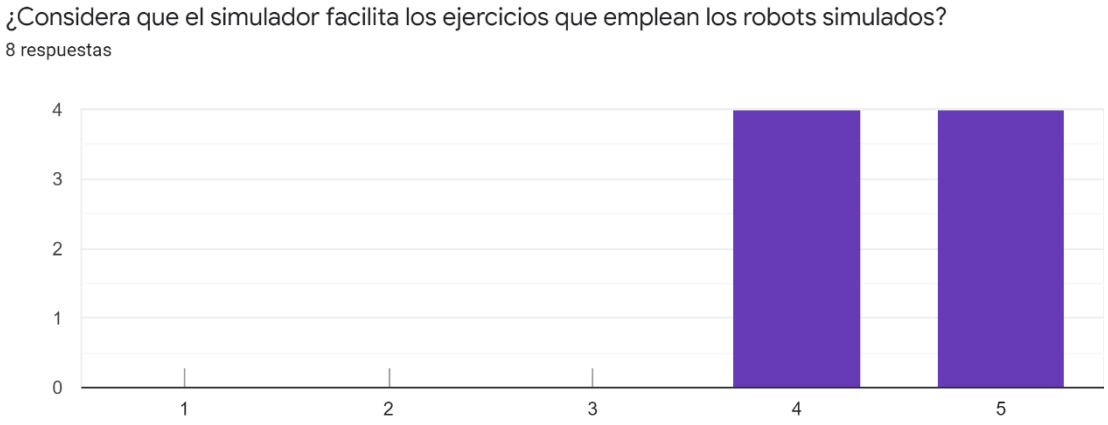


Figura 9.43 ¿Facilita los ejercicios?

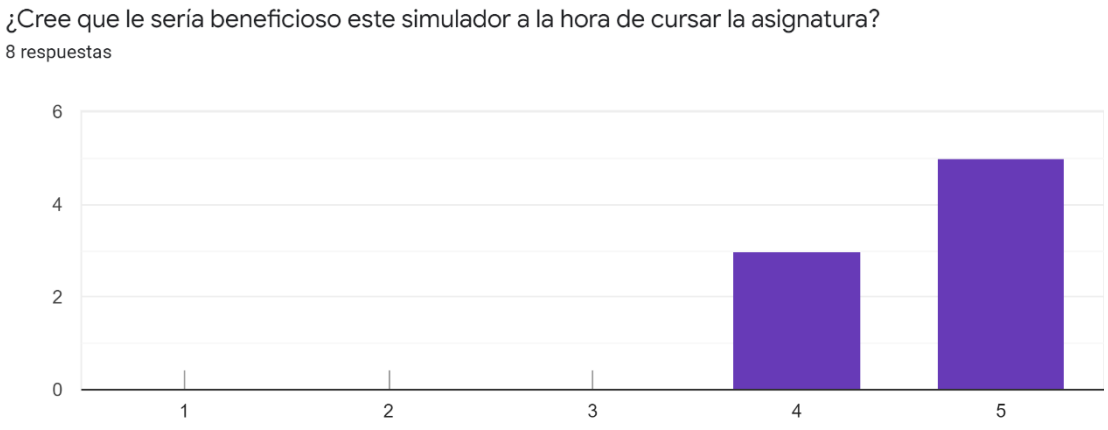


Figura 9.44 Beneficio para la asignatura

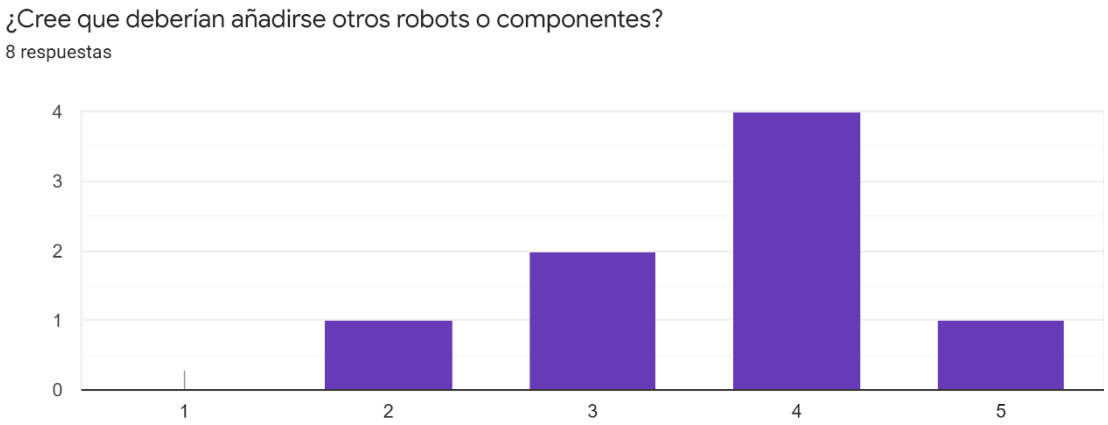


Figura 9.45 Necesidad de otros componentes

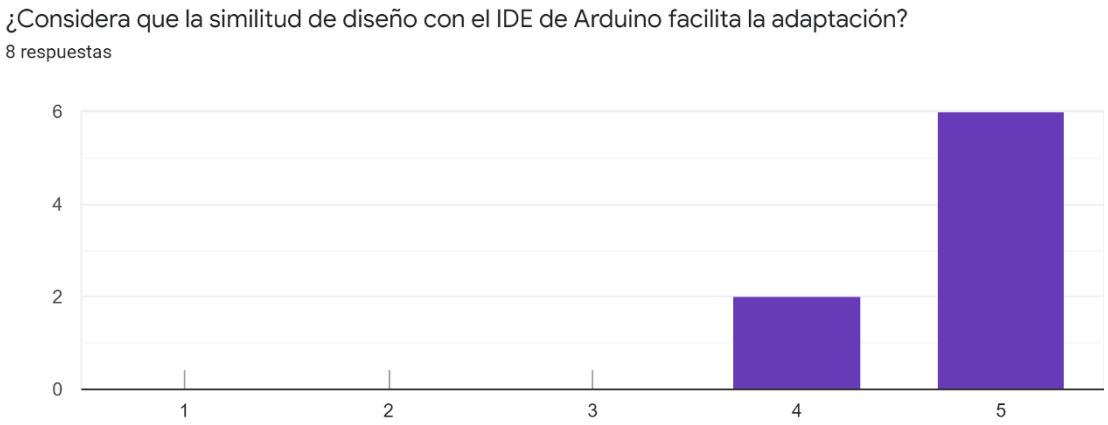


Figura 9.46 Adaptación facilitada

Además, se han obtenido la siguiente retroalimentación por parte de los usuarios:

“Yo igual hubiese preferido que el botón de "Movimiento con el teclado" hubiese estado encima de la pantalla donde se muestra el robot, en vez de debajo de ésta, lo hubiese visto antes en vez de tardar en darme cuenta de que el programa no se estaba ejecutando porque no estaba deseleccionada esta casilla”

“Excelente trabajo. Ojalá haber tenido este simulador cuando cursé la asignatura; me habría ahorrado un montón de "horas extra" teniendo que reservar el laboratorio para poder trabajar con los robots de la escuela.”

9.3.1.3.2 Resultados individuales de las pruebas

En la siguiente tabla se muestran los resultados individuales de las preguntas cortas:

Facilidad de Uso								
Usuario	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
¿Sabe dónde está dentro de la aplicación?	5	4	5	5	5	5	5	4
¿Existe ayuda para las funciones en caso de que tenga dudas?	3	3	3	3	5	5	5	5
¿Le resulta sencillo el uso de la aplicación?	4	4	3	4	5	5	4	4
¿Le resulta intuitivo el sistema de simulación de los robots?	5	4	4	5	4	5	4	4
¿Le ha resultado sencilla la configuración de pines?	4	4	5	4	5	3	3	5
¿Le resulta intuitiva la salida y entrada mediante consola?	4	4	3	5	5	5	4	5
Funcionalidad								
¿Funciona cada tarea como Vd. Espera?	5	4	4	5	4	3	4	4
¿El tiempo de respuesta de la aplicación es muy grande?	1	1	2	2	1	1	2	2
¿Ha echado en falta alguna funcionalidad en el sistema?	2	3	2	1	1	1	3	2
Aspectos gráficos								
El tipo y tamaño de letra es adecuada	5	4	5	5	5	5	5	4
Los iconos e imágenes usados son adecuados	5	4	5	4	5	5	5	4
Los colores empleados son	5	5	5	5	5	5	5	4
La estética es adecuada	5	4	5	5	5	5	5	4
Diseño de la Interfaz								
¿Le resulta fácil de usar?	5	4	4	5	5	5	4	5
¿El diseño de las pantallas es claro y atractivo?	5	4	5	4	5	5	4	4
¿Cree que el programa está bien estructurado?	5	4	5	5	5	4	3	5

Manual de ayuda								
Usuario	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
¿Necesitó acceder al manual de ayuda?	A veces	No	A veces	Si	A veces	No	Si	Si

Relevancia para la asignatura								
Usuario	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
¿Considera que el simulador facilita los ejercicios que emplean los robots simulados?	5	5	4	4	5	5	4	4
¿Cree que le sería beneficioso este simulador a la hora de cursar la asignatura?	5	5	5	4	5	4	4	5
¿Cree que deberían añadirse otros robots o componentes?	4	4	2	3	4	3	4	5
Aspecto								
¿Considera que la similitud de diseño con el IDE de Arduino facilita la adaptación?	5	5	5	4	5	5	4	5

Observaciones	
Usuario 5	Excelente trabajo. Ojalá haber tenido este simulador cuando cursé la asignatura; me habría ahorrado un montón de "horas extra" teniendo que reservar el laboratorio para poder trabajar con los robots de la escuela.
Usuario 6	Yo igual hubiese preferido que el botón de "Movimiento con el teclado" hubiese estado encima de la pantalla donde se muestra el robot, en vez de debajo de ésta, lo hubiese visto antes en vez de tardar en darme cuenta de que el programa no se estaba ejecutando porque no estaba deseleccionada esta casilla

Tabla 9.12 Preguntas cortas

9.3.1.4 Cuestionario para el responsable de las pruebas

En este apartado se mostrarán los resultados del cuestionario respondido por el responsable de las pruebas.

Aspecto Observado	Notas
El usuario comienza a trabajar de forma rápida por las tareas	Si
Tiempo en realizar cada tarea	Dentro del adecuado en la mayoría de los casos
Errores leves cometidos	Pocos (relacionados con la falta de conocimiento de Arduino)
Errores graves cometidos	Pocos (relacionados con la falta de conocimiento de Arduino)
Comentarios de usuarios	Preguntas sobre la ayuda y ciertos elementos de la interfaz
Faltas de funcionalidad	Algunas leves, principalmente avisos emergentes
¿Se ha consultado la ayuda por parte del usuario?	Si en la mayoría de los casos
¿Ha necesitado el usuario mucho tiempo para realizar las tareas?	No
Grado de satisfacción del usuario	8/10
Notas del responsable	Deberían realizarse mejoras de calidad de vida principalmente

Tabla 9.13 Cuestionario para el responsable rellenado

9.3.2 Pruebas de Accesibilidad

En este apartado se mostrarán los resultados de las pruebas de accesibilidad realizadas sobre el sistema desarrollado.

Prueba	Resultado	Notas
¿La aplicación contiene mnemónicos como alternativa a todas las operaciones con ratón?	Correcto	Faltaría unir la ayuda a la tecla F1
¿Se proveen instrucciones en el manual de usuario?	Correcto	El manual podría ampliarse
¿Es sencillo el uso de la aplicación usando la documentación?	Correcto	Ciertos aspectos pueden seguir siendo confusos
¿Hay atajos para acceder a los menús?	Correcto	
¿Soporta todos los sistemas operativos?	Incorrecto	Debido a problemas de las librerías tkinter y logger
¿Son apropiados para todos los usuarios los colores de la aplicación?	Correcto	
Las imágenes e iconos, ¿Son fáciles de entender?	Correcto	
Las alertas de la aplicación, ¿Tienen sonido?	Incorrecto	No se sabe si tkinter deja realizar esto
¿Se puede cambiar la fuente y su tamaño?	Incorrecto	No obstante, el tamaño de la fuente es suficientemente grande
No se usa el color como único medio para transmitir información.	Correcto	

Tabla 9.14 Pruebas de accesibilidad rellenadas

9.3.3 Conclusiones de las pruebas de Usabilidad y Accesibilidad

Respecto a las pruebas de usabilidad se sacan las siguientes conclusiones y se pretende realizar las siguientes ampliaciones (cambios una vez presentado el proyecto):

- Mejorar el manual de ayuda.
- Añadir más partes de la asignatura para poder realizar pruebas.
- Mejorar el sistema de configuración de pines, quizá por uno que represente la placa de Arduino.
- Mejorar la salida por consola, en concreto hacerla más sencilla de usar e intuitiva, reordenar componentes y fusionar el campo de entrada y el de salida.

Respecto a las pruebas de accesibilidad se deben hacer los siguientes cambios:

- Implementar el simulador en otros sistemas operativos (Linux y iOS principalmente).
- Añadir sonido a las alertas de la aplicación.
- Permitir el cambio de tamaño y tipo de fuente de texto.

Todos estos cambios se implementarán como ampliación de la aplicación, es decir, se realizarán a futuro. Esto es debido a la complejidad de los cambios propuestos, con el tiempo disponible actualmente es imposible llegar a implementarlos a tiempo para la entrega de este. Estos cambios son importantes, pero no determinantes de cara a la asignatura de Software para Robots y de cara a los requisitos del cliente.

9.4 Pruebas de Rendimiento

En este apartado se va a mostrar los resultados del *profiling* realizado sobre la aplicación. Para ello se ejecutó la aplicación, se importó un archivo, se ejecutó y se esperó a que finalizase, se paró la ejecución y se salió del programa. Los resultados del *profiling* son los mostrados en la siguiente figura.

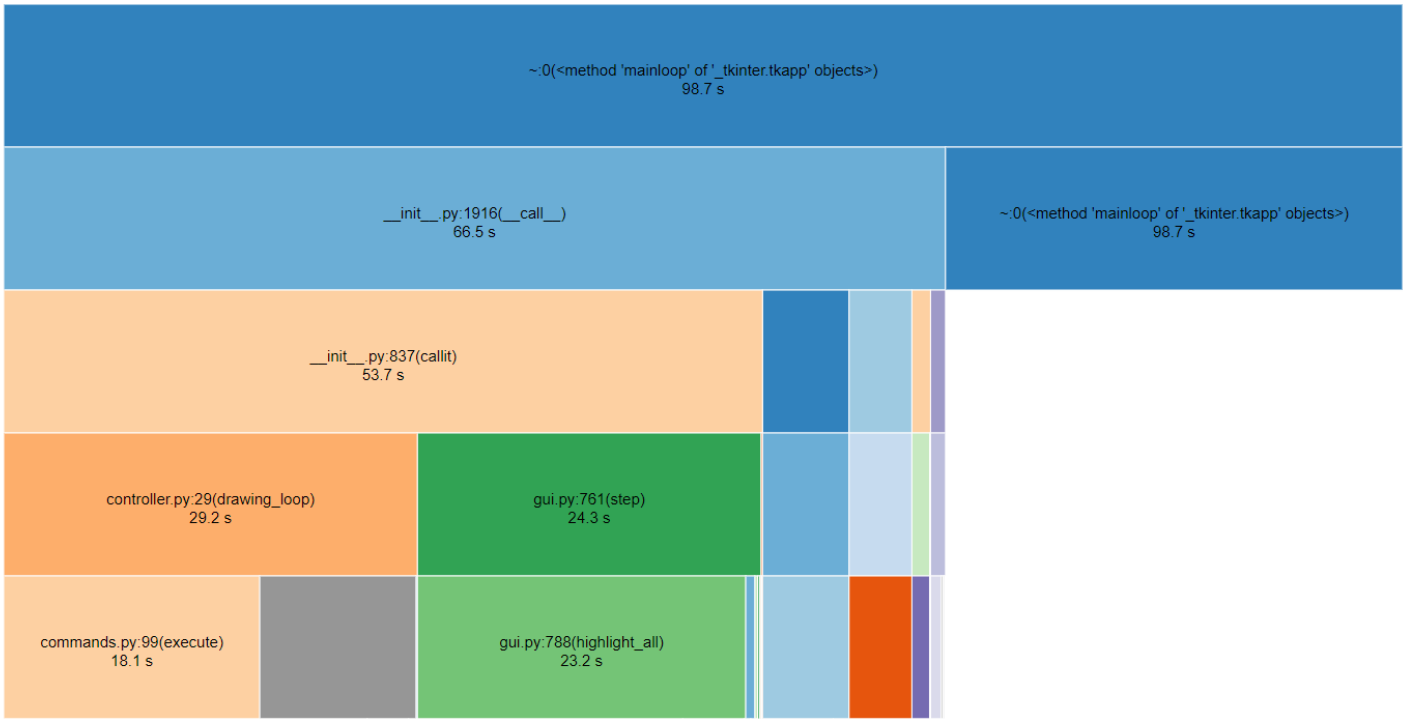


Figura 9.47 Resultados generales del profiling

En ella podemos ver que la ejecución del programa duró 98.7 segundos en total, de los cuáles 66.5 de ellos se emplearon en el método constructor del "gui" y el total de ellos en el "mainloop" de tkinter. Dicho método es el que ejecuta la interfaz gráfica continuamente, y sobre el cuál debe girar toda la ejecución del programa. Hay que destacar sobre ese método que no se puede emplear programación multihilo, pues no es seguro y genera muchos fallos (esto es la fuente de varios de los problemas mencionados en el Capítulo 7.

Desde el constructor es desde donde se crean el resto de las clases necesarias para la ejecución del sistema, además de ser también el punto desde donde se llaman a los métodos de estas clases. Podemos ver que se divide en dos partes: la propia llamada al constructor y la llamada a la ejecución de la interfaz gráfica. Lo primero toma 53.7 segundos y lo segundo 6.12. El resto de los métodos no visibles se relacionan con la llamada al bucle principal de la interfaz.

En la siguiente figura se muestra en mayor profundidad el *profiling* de la aplicación por si fuese de interés, no se va a desglosar poco a poco por la complejidad del diagrama, sin embargo, se mostrarán después de la figura las conclusiones extraídas de la prueba de rendimiento.

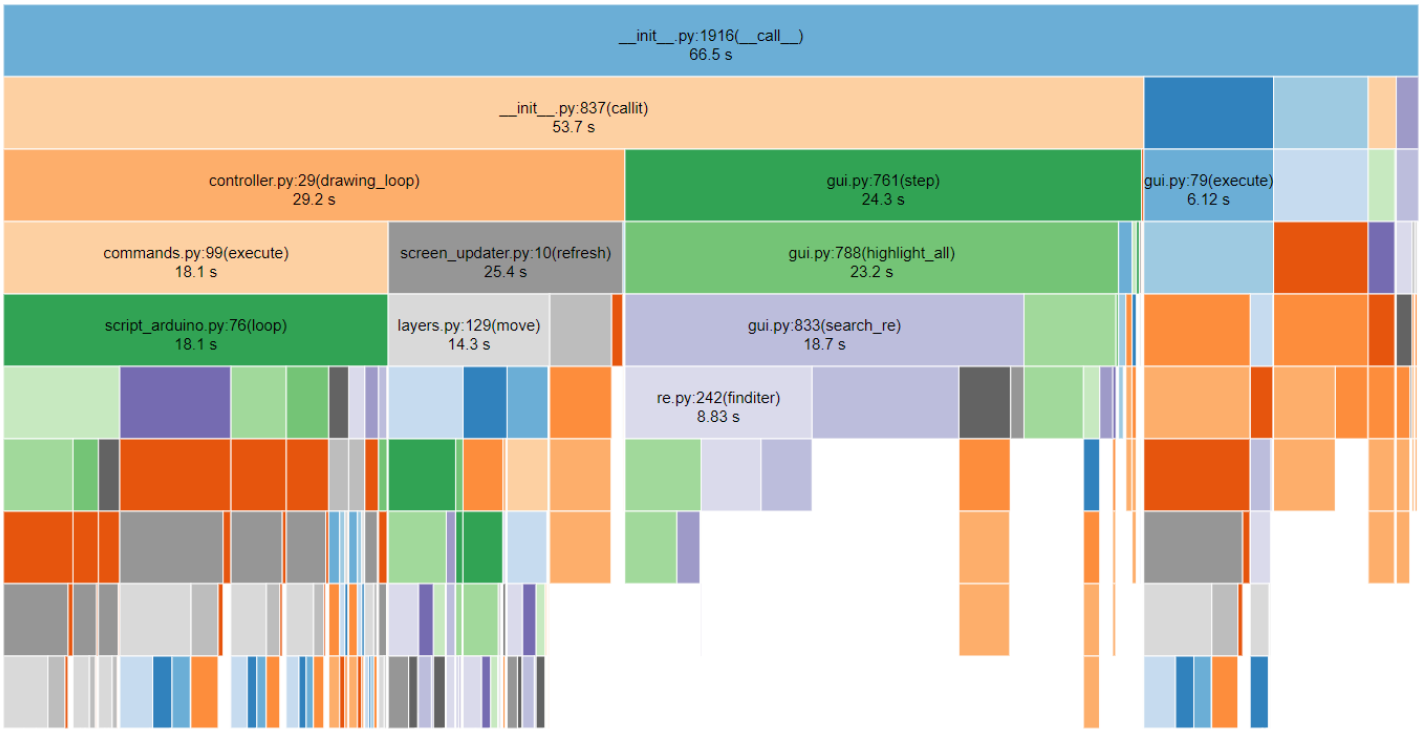


Figura 9.48 Resultados ampliados del profiling

Podemos observar por los tiempos de ejecución que el módulo que debería ser optimizado en una mayor manera es el de generación de código, pues "script_arduino.py" es el método que más consume, a parte de la interfaz gráfica (sobre la cuál no se pueden realizar muchas mejoras de rendimiento, pues dichas mejoras dependen de una librería externa).

Esto es debido a varios factores, lo primero es que, una vez se ejecuta el sketch, el script generado se estará ejecutando en bucle (no infinito, pues se necesita actualizar la interfaz gráfica). Pero también se debe a los bucles (*for* y *while*), que hacen que el programa se "bloquee" en ellos, sobre manera cuando son bucles que duran más tiempo de lo normal.

Este bloqueo lo que hace es que la interfaz gráfica se bloquee, no permitiendo al usuario interactuar y, por tanto, todo el tiempo que se pase en esa situación es como si la aplicación no respondiese.

Sin embargo, estas mejoras son complejas, pues habría que analizar toda la generación de código y que sentencias de código Python son óptimas y se ejecutan en menos tiempo, para corregir aquellas que se usen y que no sean óptimas. Por ello, será algo que se realizará en un futuro. Además, el rendimiento general de la aplicación es satisfactorio, como se ha mostrado en las pruebas de usabilidad. Por ello, no se considera necesaria ninguna mejora en este sentido, sino que será trabajo que se realizará en las ampliaciones

Capítulo 10. Manuales del Sistema

En este capítulo se mostrarán los diferentes manuales del sistema. El primero es el manual de instalación, que indica los pasos que se deben seguir para instalar el sistema. El segundo es el manual de ejecución, que indica los pasos que se deben seguir para ejecutar el sistema. El tercero es el manual de usuario, que indica el funcionamiento del programa y orienta al usuario a la hora de usar el sistema. Por último, está el manual del programador, que es un manual orientativo para aquellos programadores que quieran modificar o seguir desarrollando el sistema.

10.1 Manual de Instalación

En este apartado se realizará la explicación de todos los pasos necesarios para instalar el sistema, empezando por los requisitos.

El sistema deberá instalarse en un ordenador que tenga como sistema operativo Windows. Sería preferible que fuese Windows 10, ya que sobre este se ha probado la correcta ejecución del sistema. No obstante, el sistema debería funcionar correctamente instalándolo en cualquier versión posterior de Windows (Windows 11 en adelante) y en cualquier versión de Windows que cuente con soporte técnico hoy en día.

Ahora, se mostrarán los pasos necesarios para instalar el sistema de forma correcta:

1. Se descarga el archivo comprimido (.zip) que contiene los archivos necesarios para el correcto funcionamiento del ejecutable, además del propio ejecutable (.exe). Al descargarlo, deberíamos tener un archivo cuyo nombre será del tipo “simulador-SR-win”.
2. Se descomprime el archivo comprimido (en el directorio que queramos almacenar el programa y sin falta de descomprimir en un nuevo directorio, el archivo comprimido ya contiene un directorio que contiene el sistema). Deberíamos tener una situación similar a la de la figura 9.1.

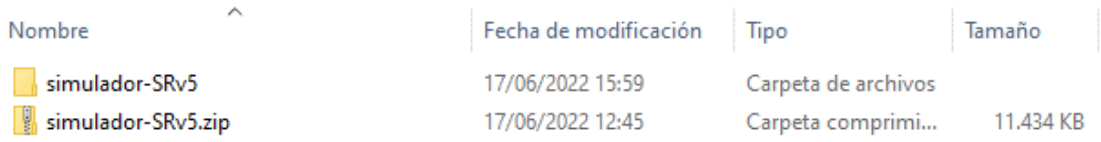
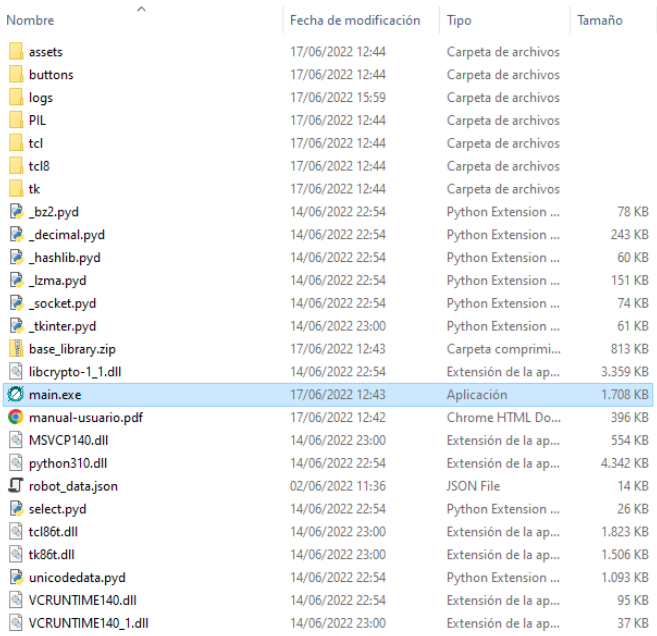


Figura 10.1 Directorio del Sistema – Descomprimido

3. El sistema ya estaría instalado. Solo hace falta ejecutar el programa principal. Para ello, accedemos a la carpeta descomprimida, en la que vemos lo mostrado en la Figura 10.2.



Nombre	Fecha de modificación	Tipo	Tamaño
assets	17/06/2022 12:44	Carpeta de archivos	
buttons	17/06/2022 12:44	Carpeta de archivos	
logs	17/06/2022 15:59	Carpeta de archivos	
PIL	17/06/2022 12:44	Carpeta de archivos	
tcl	17/06/2022 12:44	Carpeta de archivos	
tcl8	17/06/2022 12:44	Carpeta de archivos	
tk	17/06/2022 12:44	Carpeta de archivos	
_bz2.pyd	14/06/2022 22:54	Python Extension ...	78 KB
_decimal.pyd	14/06/2022 22:54	Python Extension ...	243 KB
_hashlib.pyd	14/06/2022 22:54	Python Extension ...	60 KB
_lzma.pyd	14/06/2022 22:54	Python Extension ...	151 KB
_socket.pyd	14/06/2022 22:54	Python Extension ...	74 KB
_tkinter.pyd	14/06/2022 23:00	Python Extension ...	61 KB
base_library.zip	17/06/2022 12:43	Carpeta comprimi...	813 KB
libcrypto-1_1.dll	14/06/2022 22:54	Extensión de la ap...	3.359 KB
main.exe	17/06/2022 12:43	Aplicación	1.708 KB
manual-usuario.pdf	17/06/2022 12:42	Chrome HTML Do...	396 KB
MSVCP140.dll	14/06/2022 23:00	Extensión de la ap...	554 KB
python310.dll	14/06/2022 22:54	Extensión de la ap...	4.342 KB
robot_data.json	02/06/2022 11:36	JSON File	14 KB
select.pyd	14/06/2022 22:54	Python Extension ...	26 KB
tcl86t.dll	14/06/2022 23:00	Extensión de la ap...	1.823 KB
tk86t.dll	14/06/2022 23:00	Extensión de la ap...	1.506 KB
unicodedata.pyd	14/06/2022 22:54	Python Extension ...	1.093 KB
VCRUNTIME140.dll	14/06/2022 22:54	Extensión de la ap...	95 KB
VCRUNTIME140_1.dll	14/06/2022 23:00	Extensión de la ap...	37 KB

Figura 10.2 Directorio que contiene el sistema

El archivo que necesitamos ejecutar el main.exe, que está señalado en la Figura 10.2. Si lo ejecutamos, ya podremos ver el sistema en funcionamiento.

Vamos a describir el contenido de este directorio. Primero, en cuanto a archivos, vemos una serie de archivos cuya extensión es “pyd” y “dll”. Estos archivos son bibliotecas de enlace dinámico, es decir, archivos con código ejecutable cargados bajo demanda por el sistema **[Wikipedia].** El archivo comprimido base_library.zip contiene archivos de extensión “pyc”, que son archivos de *bytecode* compilado.

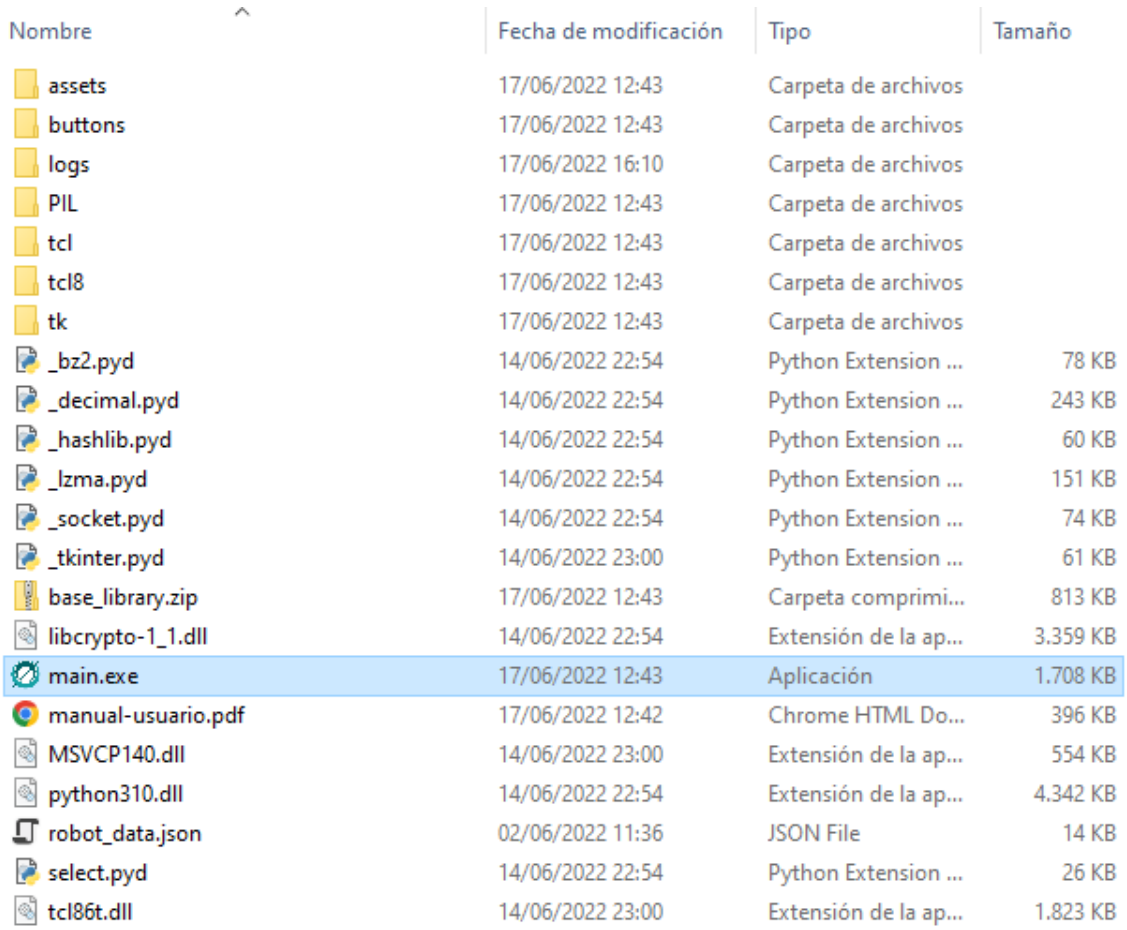
El archivo manual-usuario.pdf es el manual de usuario (apartado 10.3), que se ejecutará en el momento en que el usuario pulse sobre la opción “Manual de ayuda” del menú “Ayuda”. Por último, “robot_data.json” contiene los datos necesarios para configurar los circuitos y los robots por defecto. Si se quiere cambiar los pines por defecto, se puede hacer en este archivo (aunque no es recomendable).

Las carpetas “assets” y “buttons” contienen las imágenes png usadas para el *front-end* de la aplicación, en concreto “buttons” contiene los botones de la barra de herramientas y “assets” los gráficos de los robots (y el icono de la aplicación).

Respecto al resto de las carpetas (no “logs”, esa es generada posteriormente junto a “temp”; la primera para los logs y la segunda para el archivo de código compilado), son las correspondientes a las librerías usadas por el sistema (contienen archivos de extensión pyd).

10.2 Manual de Ejecución

En este apartado se explicarán los pasos necesarios para ejecutar el sistema. En este caso, al ser un sistema *standalone*, dichos pasos son pocos. Simplemente basta con entrar en la carpeta del sistema (descrita en el apartado 10.1) y ejecutar el archivo main.exe, como se muestra en la Figura 10.3.



Nombre	Fecha de modificación	Tipo	Tamaño
assets	17/06/2022 12:43	Carpeta de archivos	
buttons	17/06/2022 12:43	Carpeta de archivos	
logs	17/06/2022 16:10	Carpeta de archivos	
PIL	17/06/2022 12:43	Carpeta de archivos	
tcl	17/06/2022 12:43	Carpeta de archivos	
tcl8	17/06/2022 12:43	Carpeta de archivos	
tk	17/06/2022 12:43	Carpeta de archivos	
_bz2.pyd	14/06/2022 22:54	Python Extension ...	78 KB
_decimal.pyd	14/06/2022 22:54	Python Extension ...	243 KB
_hashlib.pyd	14/06/2022 22:54	Python Extension ...	60 KB
_lzma.pyd	14/06/2022 22:54	Python Extension ...	151 KB
_socket.pyd	14/06/2022 22:54	Python Extension ...	74 KB
_tkinter.pyd	14/06/2022 23:00	Python Extension ...	61 KB
base_library.zip	17/06/2022 12:43	Carpeta comprimi...	813 KB
libcrypto-1_1.dll	14/06/2022 22:54	Extensión de la ap...	3.359 KB
main.exe	17/06/2022 12:43	Aplicación	1.708 KB
manual-usuario.pdf	17/06/2022 12:42	Chrome HTML Do...	396 KB
MSVCP140.dll	14/06/2022 23:00	Extensión de la ap...	554 KB
python310.dll	14/06/2022 22:54	Extensión de la ap...	4.342 KB
robot_data.json	02/06/2022 11:36	JSON File	14 KB
select.pyd	14/06/2022 22:54	Python Extension ...	26 KB
tcl86t.dll	14/06/2022 23:00	Extensión de la ap...	1.823 KB

Figura 10.3 Archivo ejecutable del sistema

Una vez ejecutado, nos saldrán dos ventanas, la consola (que mostrará salida de Python si es que la hay, incluidos errores); y la ventana principal, que será la que nos permitirá utilizar toda la funcionalidad de la aplicación. Las ventanas se muestran en la Figura 10.4.

Para cerrar el sistema correctamente hay dos opciones:

- Cerrar desde el propio botón de cerrar de la ventana de la aplicación.
- Cerrar desde la opción dada en el menú Archivo de la aplicación.

No es recomendable cerrar la aplicación cerrando la consola o usando el administrador de tareas, es mejor cualquiera de las opciones propuestas anteriormente. Por ello, es mejor dejar estas opciones como las últimas.

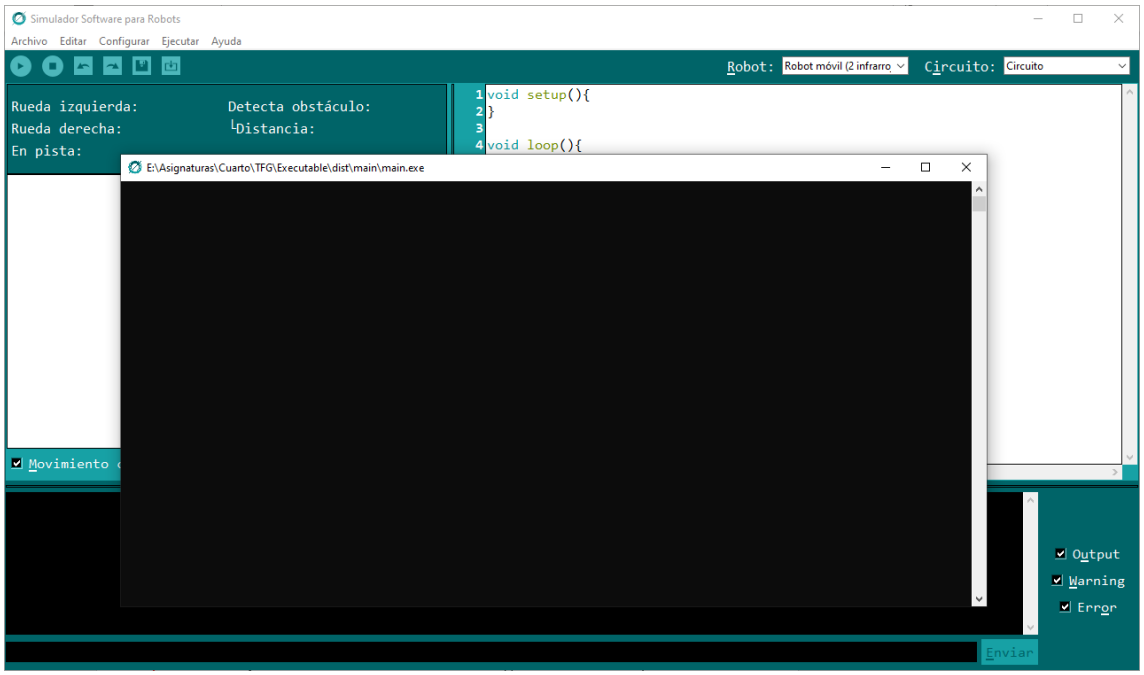


Figura 10.4 Consola y Aplicación superpuestas

10.3 Manual de Usuario

En este apartado se desarrollará el manual de usuario, que será incluido dentro de la aplicación como manual de ayuda para los usuarios y que pretende explicarles las funcionalidades de la manera más simple posible.

10.3.1 Visión general del sistema

Al abrir la aplicación nos encontraremos con la ventana que contiene toda la funcionalidad del sistema.

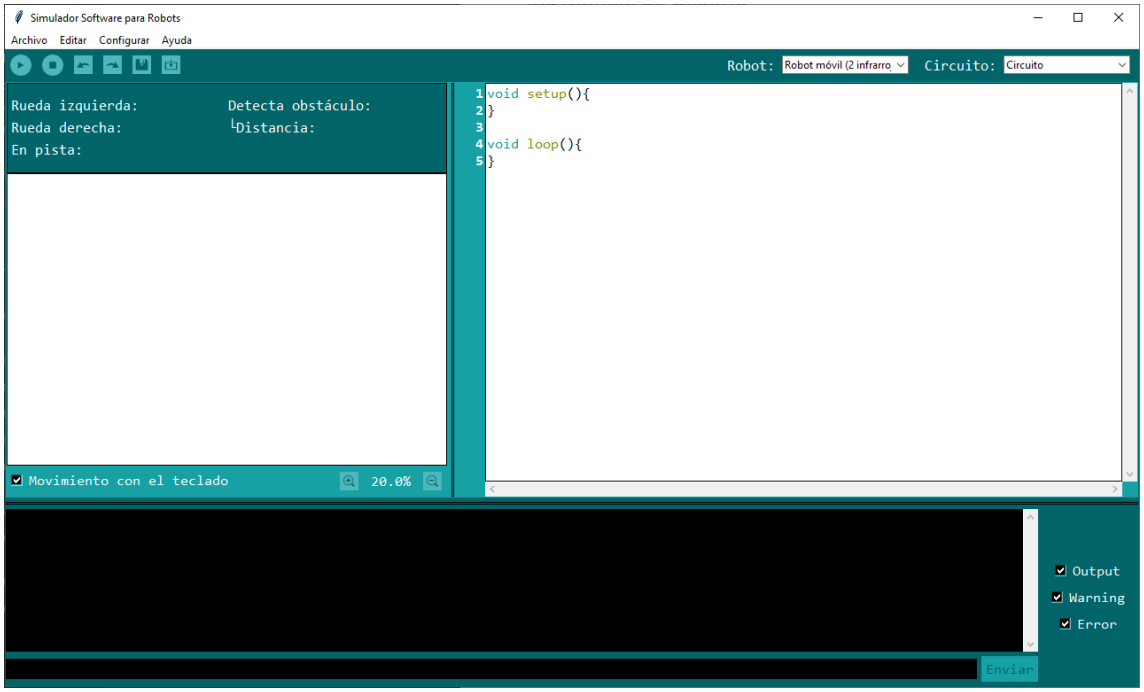


Figura 10.5 Pantalla principal del sistema

En ella podemos ver varios elementos:

- Arriba del todo encontramos el menú de la aplicación.
- Debajo justo del menú, encontramos la barra de herramientas, que contiene todas las herramientas básicas para ejecución, edición y apertura de sketches.
- Después, la interfaz se divide en tres componentes principales:
    - Arriba a la izquierda se encuentra la representación gráfica de los robots.
    - Arriba a la derecha se encuentra el editor de código.
    - Abajo se encuentra la consola de entrada/salida.

Se explicará en los siguientes apartados las funcionalidades que se podrán encontrar en cada componente mencionado.

10.3.1.1 Menú

Comenzaremos describiendo la funcionalidad del menú de la aplicación. En la Figura 10.6 se pueden ver las opciones ofrecidas por dicho menú.

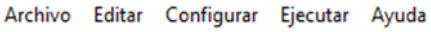


Figura 10.6 Menú de la aplicación

La primera opción es archivo, si pulsamos sobre ella, veremos que surge el menú desplegable mostrado en la Figura 10.7.

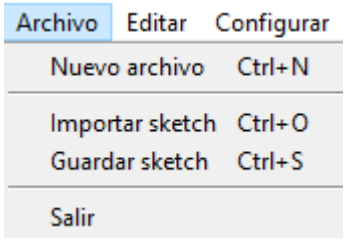


Figura 10.7 Menú archivo

En este menú podemos ver cuatro opciones:

- Nuevo archivo: nos permitirá crear la estructura de un nuevo sketch. Antes de crear el nuevo archivo nos saldrá un diálogo como el de la Figura 10.8. Si queremos crear el archivo escogeremos sí, en otro caso basta con escoger no o cerrar el diálogo.

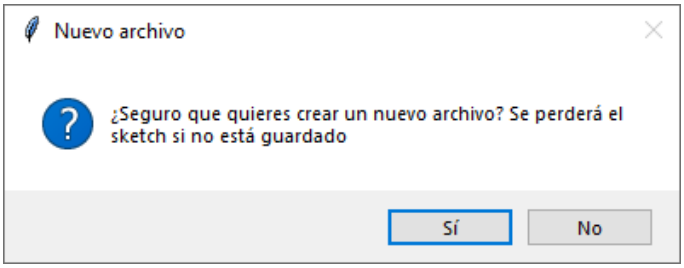


Figura 10.8 Diálogo de confirmación – Crear nuevo archivo

- Importar sketch: Nos permitirá abrir un sketch que queramos editar. Emplea el diálogo típico de apertura de archivos de otras aplicaciones parecidas.
- Guardar sketch: Nos permitirá guardar un sketch que hallamos creado/editado. Usa el diálogo clásico de guardado, con lo que podemos sobrescribir el archivo o guardar en el archivo que nosotros escojamos.
- Salir: Permite salir de la aplicación, al igual que el botón cerrar de la ventana, solo que, en este caso, lanza un diálogo de confirmación como el de la Figura 10.9.

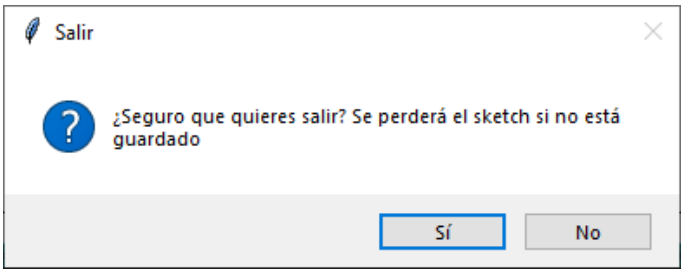


Figura 10.9 Diálogo de confirmación – Salir de la aplicación

La segunda opción es el menú editar, si pulsamos sobre ella, veremos el menú mostrado en la Figura 10.10.

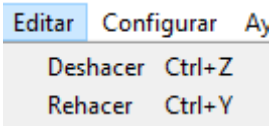


Figura 10.10 Menú editar

En este menú vemos dos opciones, que son deshacer y rehacer. Dichas opciones implican el que se deshaga o rehaga acciones del editor de código, sean editar, borrar, añadir o cualquier otra acción que tenga que ver con el sketch. Esto implica incluso que se puede deshacer el crear archivo o importar uno nuevo.

La siguiente opción es el menú de configuración, que contiene una sola opción, Configurar pines, cuyo mnemónico es 'ctrl + p'. Si accedemos a dicha opción, abriremos la siguiente ventana:

Figura 10.11 Ventana de configuración de pines del robot

La ventana que se muestra en la Figura 10.11 varía según el robot que esté elegido en la aplicación. La ventana mostrada en la figura anterior es para el robot móvil con 2 sensores de infrarrojos. Mientras que las figuras siguientes muestran las otras dos opciones, la primera el robot móvil de 4 infrarrojos y la segunda el actuador lineal.

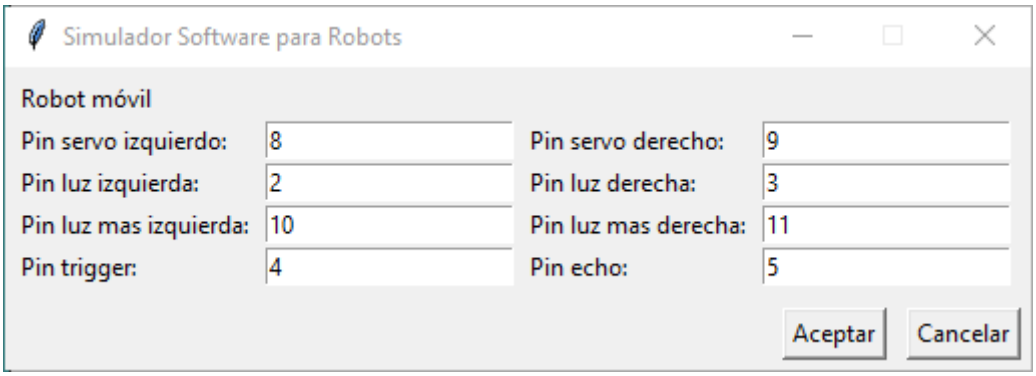


Figura 10.12 Ventana de configuración - Robot móvil de 4 sensores infrarrojos

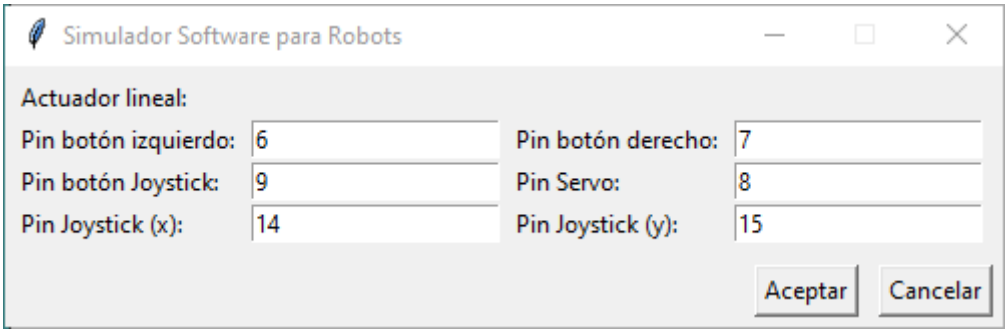


Figura 10.13 Ventana de configuración – Actuador lineal

Como podemos ver, si el pin es analógico, como en el caso de las ordenadas del joystick, se muestra el número de pin que correspondería en número. Esto no impide que se pueda escribir el pin con la usual forma A0-A5, de hecho, es recomendable que se haga así para evitar confusiones.

En la cuarta opción vemos el menú ejecutar, que se puede ver en la Figura 10.14. Da cuatro opciones, Ejecutar, que comienza la simulación, Detener, que la detiene, Ampliar, que amplía la imagen de la simulación y Reducir, que la reduce.

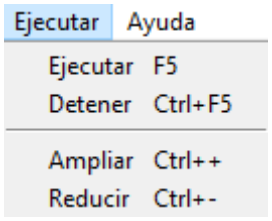


Figura 10.14 Menú ejecutar

Con respecto a la quinta y última opción, es el menú de ayuda, que incorpora él acerca de la aplicación y este manual de usuario.

10.3.1.2 Barra de herramientas

La barra de herramientas de la aplicación se encuentra en la parte superior de la aplicación, justo debajo del menú de esta. La Figura 10.15 y la Figura 10.16 muestran las variaciones de dicha barra.

Figura 10.15 Barra de herramientas – Actuador lineal

Figura 10.16 Barra de herramientas – Robot móvil

En la primera figura, se ve la barra de herramientas del actuador lineal y en la segunda de ambos robots móviles. La única diferencia entre ambas es que, en el caso del robot móvil; se añade una opción para poder cambiar el circuito que recorrerá el robot.

Como se puede ver en la Figura 10.16, si pasamos por encima de los botones de la izquierda, se mostrará a la derecha de estos la opción que se va a ejecutar.

Las opciones de este menú son las siguientes:

- Ejecutar: Ejecuta el sketch escrito en la parte del editor de código.
- Detener: Detiene la ejecución del sketch si este se está ejecutando.
- Deshacer: Como en editar, deshace las acciones que se han realizado a la hora de editar el código.
- Rehacer: Rehace las acciones que se han realizado a la hora de editar el código.
- Guardar: Guarda el sketch de la misma forma que la opción del menú archivo.
- Importar: Importa el sketch de la misma forma que la opción del menú archivo.
- Robot: Permite cambiar el tipo de robot (y lo que cambie de la interfaz gráfica con él).
- Circuito (solo robots móviles): Permite cambiar el circuito que recorrerá el robot móvil.

10.3.1.3 Representación gráfica de los robots

Debajo de la barra de herramientas de la aplicación, hacia la izquierda se encuentra la parte de la representación gráfica de los robots. Su apariencia es la que muestra las dos figuras siguientes.

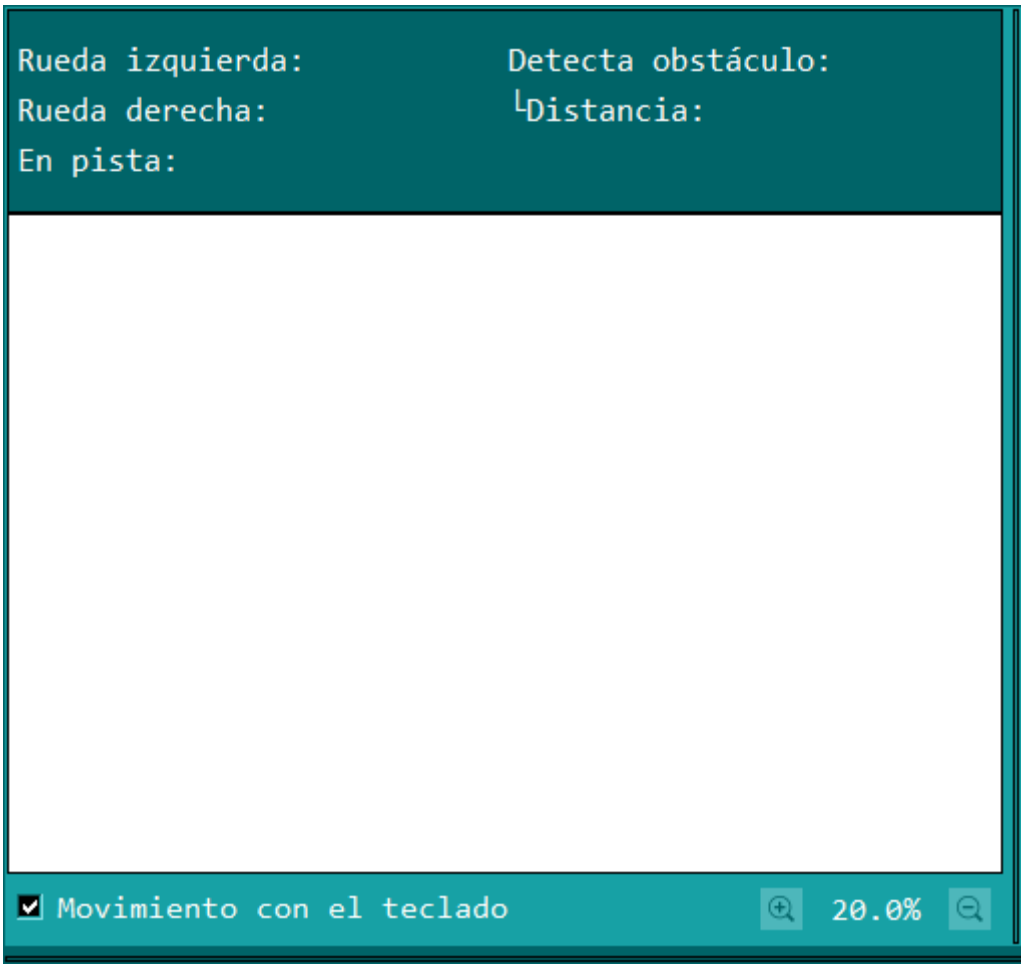


Figura 10.17 Representación gráfica – Robot móvil – Sin ejecutar

Como vemos en la Figura 10.17, la representación gráfica cuenta con un HUD que muestra durante la ejecución los datos de que realiza cada componente del robot. Debajo está un recuadro en blanco, que será donde se muestre la simulación del robot. Por último, y abajo del todo, se ve una checkbox para activar o desactivar el movimiento por teclado. Esto es **IMPORTANTE**. Para ejecutar el código (automáticamente), se debe desactivar esta casilla. Si no, el robot estará quieto (a menos que pulsemos las teclas de movimiento WASD, esto es útil para posicionar el robot a nuestro gusto).

La parte derecha de esta barra inferior es la ampliación del dibujo, con el botón izquierdo se acerca y con el derecho se aleja. Se puede realizar lo mismo con el scroll, hacia delante amplia, hacia atrás aleja. La ampliación recomendada en una pantalla de 1920x1080 es 20% con el programa en pantalla completa, de esta manera se podrá ver todo lo que haga el robot.

Figura 10.18 Representación gráfica – Actuador lineal – Sin ejecutar

Como vemos en la Figura 10.18, cambia ligeramente el HUD y se añade una nueva barra. El resto de funcionalidad es la misma que en la Figura 10.17.

La nueva barra que se ha añadido es la que permite simular el joystick. El botón del joystick funciona pulsando el botón. La dirección del joystick funciona según la Figura 10.19.

0, 0	512, 0	1023, 0
0, 512	512, 512	1023, 512
0, 1023	512, 1023	1023, 1023

Figura 10.19 Movimiento del joystick

A continuación, se muestran tres figuras mostrando cada robot en modo ejecución (es decir, representados gráficamente).

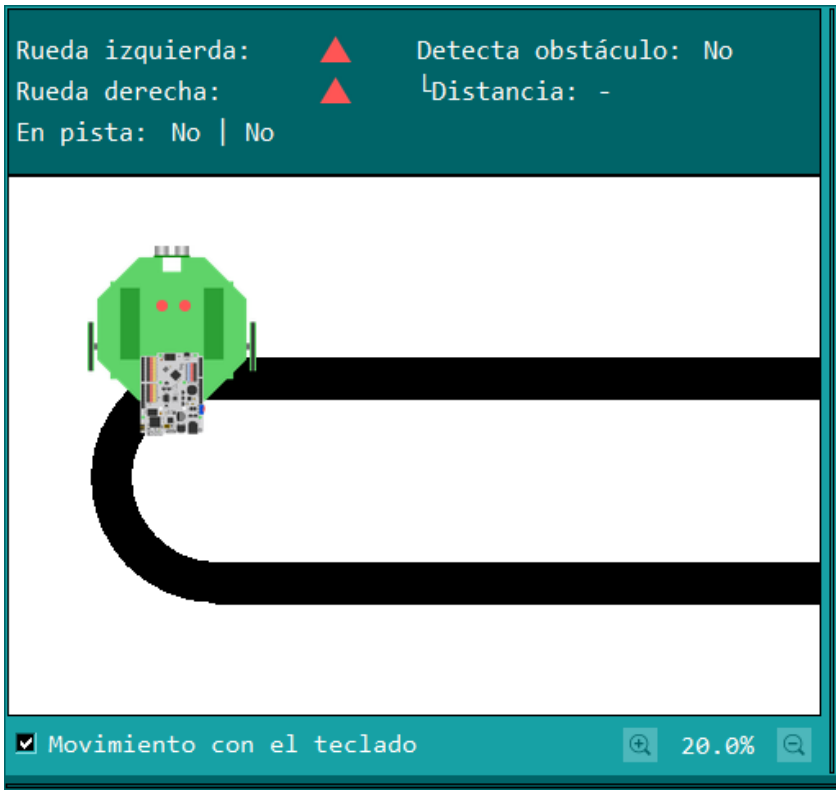


Figura 10.20 Representación gráfica – Robot móvil 2 - Ejecutando

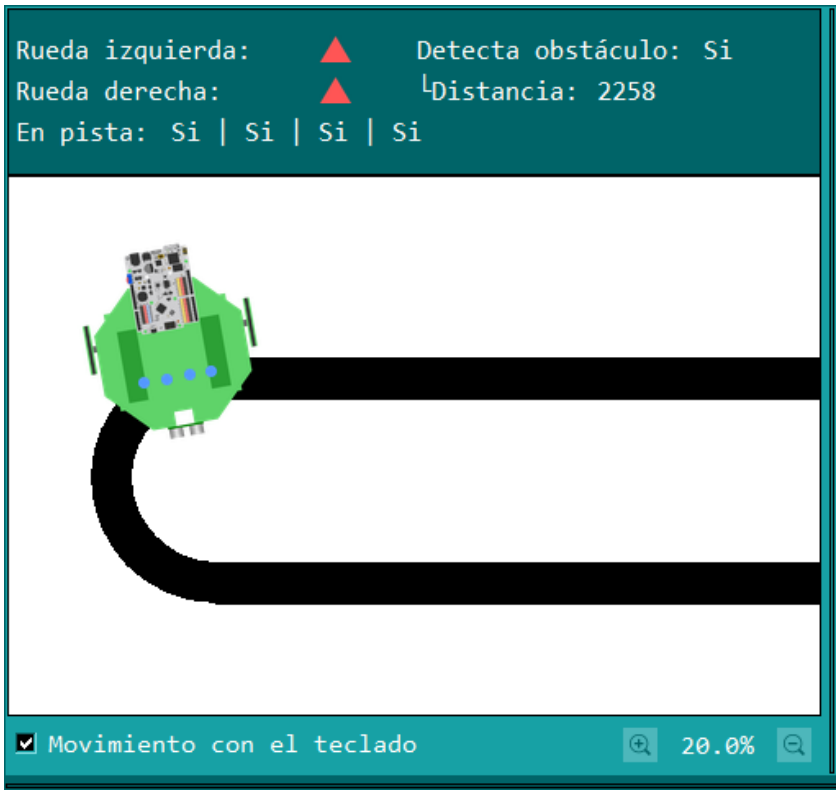


Figura 10.21 Representación gráfica – Robot móvil 4 - Ejecutando

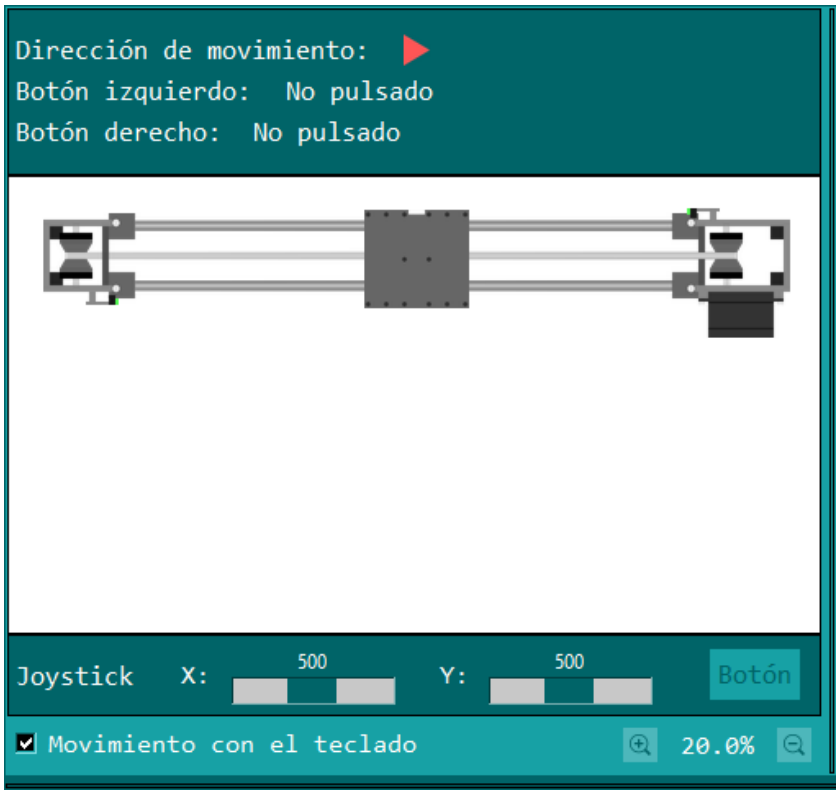


Figura 10.22 Representación gráfica – Actuador lineal – Ejecutando

En la Figura 10.20 podemos ver como se vería el robot móvil de 2 sensores de luz en su representación gráfica. La posición de los sensores de luz se puede ver mediante los círculos rojos. Dichos círculos se colorean de azul si el sensor detecta la pista. Si eso sucede, el HUD mostrará (en el lado correspondiente) si el sensor está en pista o no.

Respecto al HUD, también mostrará en qué dirección se mueve en cada rueda. El color de la flecha mostrará la velocidad del servo (azul – rápido, amarillo – velocidad media, rojo – lento o parado).

El HUD también mostrará si el sensor de ultrasonidos detecta un obstáculo, y si lo detecta mostrará la distancia hacia el obstáculo (no equivalente a la real).

En la Figura 10.21 se ve que lo único que cambia en cuanto a la representación gráfica del robot móvil de 4 sensores es el número de círculos (sensores) y, en el HUD, se muestra en la parte de en pista lo que detectan los 4 sensores.

Con respecto a la Figura 10.22, en el robot vemos los dos botones que hacen de tope a cada extremo del actuador lineal. Si el bloque del actuador toca un botón, la parte que colisiona con el bloque se vuelve roja, como se muestra en la Figura 10.23. Si no, estará verde como en la Figura 10.22.

En el caso del HUD, se muestra en qué dirección se mueve el servo y su velocidad de la misma forma que en los robots móviles y también se muestra si los botones están siendo pulsados o no.

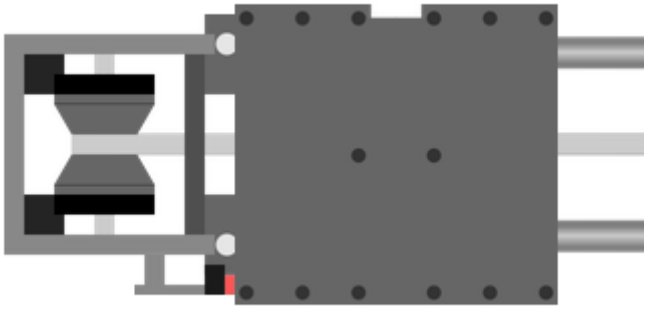


Figura 10.23 Actuador lineal - El botón tope está pulsado

10.3.1.4 Editor de código

A la derecha de la representación gráfica está el editor de código. La Figura 10.24 muestra cómo se estructura.

Figura 10.24 Editor de código

El editor tiene coloreado sintáctico (el mismo que Arduino IDE). Además, a la izquierda se muestra el número de línea correspondiente (esto facilita la búsqueda de errores y advertencias.

Se puede navegar a través del código usando las barras de *scroll* lateral y vertical. Se volverán activas en caso de ser necesarias. Se pueden usar utilizando la rueda del ratón. El editor no autocompleta el código, eso sería una funcionalidad para implementar en el futuro.

10.3.1.5 Consola de entrada/salida

Debajo de los dos últimos componentes se sitúa la consola, que se muestra en la Figura 10.25.

Figura 10.25 Consola

En la parte superior izquierda tenemos la consola como tal. Mostrará la salida del sketch, con un código de colores en el cual rojo es un error, amarillo una advertencia y blanco un mensaje normal. Estos mensajes se pueden filtrar con los *checkboxes* de la derecha (se pueden usar a la vez varios).

En la parte inferior hay un campo de texto que es el que se debe usar si se quiere hacer una entrada por consola. La entrada se envía usando el botón enviar.

10.3.1.6 Otras características para tener en cuenta

La aplicación se bloquea a todo tipo de interacción si se mete en un bucle o un *delay*, esta situación se termina en el momento que se sale del bucle (si no se vuelve infinito) o del *delay*. El arreglo de este problema ya está siendo desarrollado, pero mientras tanto, si hace falta cambiar algo en la interfaz gráfica cuando se bloquea, algunas veces funciona al realizar la interacción con la interfaz bloqueada. Bajo experiencia personal del desarrollador es mejor realizar la interacción justo al principio del bucle o *delay*, la mayoría de las veces suele funcionar (no todas).

Muchos errores de ejecución pueden ser resultado de una mala configuración de los pines, por lo que es recomendable revisarlos siempre que se vaya a ejecutar el programa. Si se quiere cambiar los pines por defecto del programa, se puede en el archivo “robot_data.json”, aunque no es muy recomendable tocar dicho archivo.

Cuando el botón ejecutar o parar se vuelve amarillo es que se está ejecutando alguna tarea, en el momento que se vuelve al color normal, dicha tarea se terminó de realizar.

10.4 Manual del Programador

En este manual se incluirán todas las recomendaciones que puedan ayudar con el desarrollo a los futuros desarrolladores o cualquiera que quiera ampliar el sistema o modificarlo.

Dentro del proyecto se han dejado comentarios y descripciones de métodos y clases en aquellos que se ha considerado necesario. No obstante, se explicará en este manual estos y otros detalles que quizá puedan pasar desapercibidos a ojos del programador.

El manual se dividirá por paquetes de Python (toda carpeta con un archivo "__init__.py"), además de un apartado que explicará otros detalles que no encajen correctamente en ningún paquete de la aplicación. Siguiendo el orden alfabético, empezamos por el paquete "compiler".

10.4.1 Compilador (Transpilador)

El compilador se encuentra dentro del paquete "compiler", donde encontramos los archivos mostrados en la Figura 10.26.

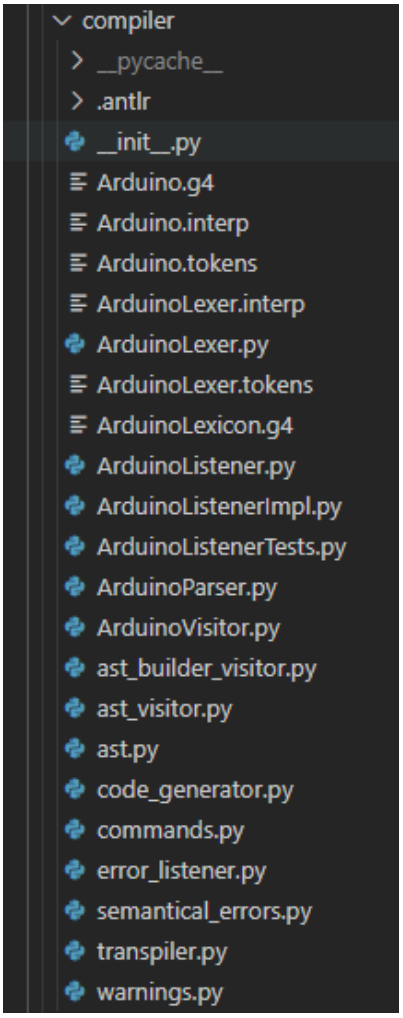


Figura 10.26 Archivos del paquete compiler – Visual Studio Code

Dentro de este archivo podemos diferenciar varios tipos de archivo según su relevancia para el programador:

- Archivos irrelevantes: “ArduinoLexer.tokens”, “ArduinoLexer.interp”, “Arduino.interp”, “Arduino.tokens”. Estos archivos son autogenerados por ANTRL4 (el archivo jar de ANTRL4 que se puede descargar de su web), y no tienen relevancia alguna en ninguna clase desarrollada (no así en clases autogeneradas de ANTLR4.
- Archivos relevantes: Dentro de estos, podemos ver dos clases:
    - Autogenerados: Son aquellos que genera ANTR4. Dentro de esta categoría están “ArduinoParser”, “ArduinoVisitor”, “ArduinoListener”, “ArduinoLexer”.
    - Aquellos módulos que se implementan a partir de autogenerados; módulos como “ArduinoListenerImpl” y “ArduinoListenerTests” (que solo tienen utilidad para el testeo y la observación del *parse tree* generado por el *parser*) y “ast_builder_visitor“.
    - Desarrollados: Son aquellos desarrollados a mano, que incluyen a los módulos restantes del paquete

Primero de todo vamos a explicar cómo funciona el módulo “motor” del paquete, “transpiler”. Este módulo contiene un método (“transpile”), que será el que llame a los métodos de otros paquetes forman las fases del compilador (o transpilador). A modo de resumen, esta clase llama al *lexer* y *parser* (realmente el *lexer* se pasa como parámetro al *parser*), al “ast_builder_visitor” para generar el AST, además del análisis semántico (“semantical_errors”), generación de código (“code_generator”) y generación de alertas (“warnings”).

El módulo encargado de llamar al “transpiler” será “commands”, que contiene tres comandos (del patrón *Command*): “Compile” (el que llama a “transpiler”), “Setup” (que llama a la función “setup” del sketch compilado) y “Loop” (que llama a la función “loop” del sketch compilado).

Gran parte de las clases de este paquete implementan el patrón *Visitor*, esas clases se muestran en la Figura 10.27. En resumen:

- “ast_builder_visitor” implementa el *Visitor* definido por “ArduinoVisitor", que es el que recorre la estructura en árbol del *parse tree* generado por “ArduinoParser”.
- “ast_visitor” será el *Visitor* del que hereden el resto de los módulos señalados. Este *Visitor* en el que recorre el AST definido en el módulo homónimo.

Estos módulos que implementan el patrón *Visitor* realizarán las operaciones sobre las estructuras en árbol utilizando todos los métodos “visit_x” que contengan, siendo necesario añadir solamente aquellos métodos de este tipo que vayamos a utilizar.

Por ejemplo, si quiero operar sobre las sentencias “#include”, entonces solamente incluiré el método “visit_include”, no haría falta incluir ningún método más. Esto facilitará la comprensión de código; además de ahorrarnos líneas de código innecesarias y facilitar así la rapidez con la que buscamos algo dentro de un módulo.

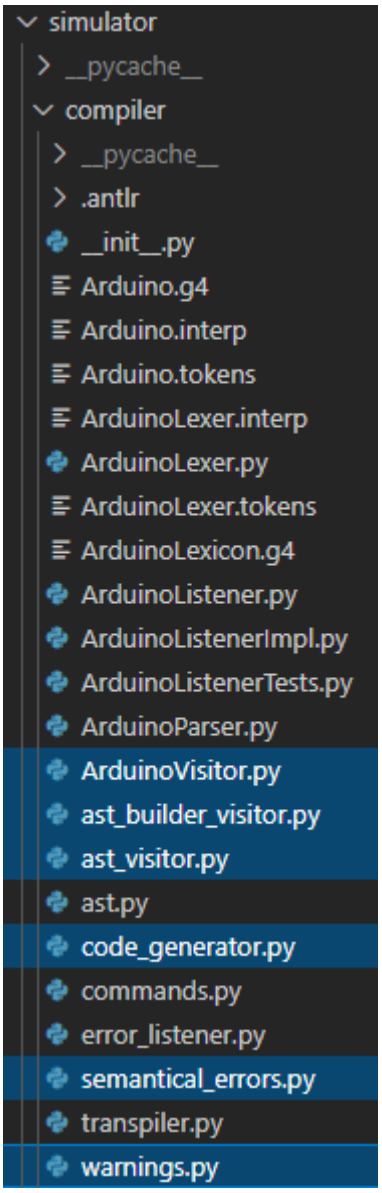


Figura 10.27 Módulos que implementan el patrón visitor

Respecto a “error_listener”, es una clase que sirve para cambiar la estructura de los errores generados por ANTLR4, es decir, en vez de mostrar el texto de los errores por defecto se muestra el texto que se ha creado para el simulador. Son muy parecidas a las clases *Visitor*, de hecho, si el programador lo desea; son fácilmente sustituibles por otro. Estas clases tienen dos métodos para cada nodo del árbol: “enterx” y “exitx”, el primero se ejecuta al entrar en el nodo y el segundo al salir de él. Estas clases, como ya se ha comentado, solo se han creado con el propósito de probar la estructura del *parse tree*.

El módulo “code_generator” está desarrollado de tal manera que genera un archivo de extensión .py, es decir, genera código Python. Si se hereda de “ast_visitor” se podría hacer otro generador de código que genere otro lenguaje.

Para añadir alguna expresión nueva al compilador (por ejemplo, punteros), los pasos a seguir son los siguientes:

1. Añadimos las expresiones léxicas correspondientes a ArduinoLexicon.g4
2. Añadimos las expresiones sintácticas correspondientes a Arduino.g4
3. Generamos la gramática con el "jar" proporcionado por ANTLR4, que se puede descargar de su web.
4. Añadimos los nodos AST necesarios en el módulo homónimo.
5. Añadimos la lógica necesaria para generar esos nodos a "ast_builder_visitor", dentro del método "visit" correspondiente.
6. Implementar la lógica de análisis semántico necesaria.
7. Se implementa la lógica de generación de código.
8. Finalmente se añaden las advertencias necesarias en "warnings" (si las hubiera).

10.4.2 Lectura de archivos (sketches y configuración)

En el paquete files se encuentran los métodos relacionados con la apertura y el guardado de archivos (hay solo tres excepciones de archivos no gestionados por este módulo, se explican en el paquete correspondiente). En concreto, solo hay un módulo: "files_reader". Este contiene dos clases:

- Las clases "listener" "FileManager": Se encarga de importar y guardar los sketches. Se le debe proporcionar el nombre y dirección del archivo (combinados). En el caso de guardar, también se proporciona el código del sketch.
- "RobotDataReader": Se encarga de leer "robot_data.json", es decir, lee todos los datos de configuración de los robots y de las pistas presentes en el archivo. Tiene dos métodos principales:
    - "parse_robot", que se encargará de proveer la configuración de los robots.
    - "parse_circuit", que se encargará de proveer la configuración de los circuitos (rectas, curvas y obstáculos).

El archivo "robot_data.json" contiene dos listas, una de robots ("robots") y otra de circuitos ("circuits"). Cada robot tiene un nombre ("name") y una lista de elementos ("elements"). Cada elemento tiene a su vez un nombre ("name") y un pin ("pin"). En el caso de los circuitos, cada circuito tiene un nombre ("name"), una lista de partes ("parts") y otra de obstáculos ("obstacles"). Las partes se distinguen de la siguiente manera:

- Si "type" es "straight", entonces tendrá la orientación ("orient") (eje "x" o "y"), la distancia ("dist") (en píxeles) que recorre, donde se va a fijar la siguiente parte ("anchor next") ("end", "1/4", "mid", "3/4", "start"; además de poder guardar para otra futura pieza con cualquiera de las anteriores añadiendo "/espacio/ save" y recuperando el punto de fijado con "/espacio/ load". Además, se pueden guardar más puntos de fijación en una recta con la propiedad guardar puntos de fijado ("save anchors").
- Si "type" es "turn", entonces tiene el ángulo de comienzo ("starting_angle") (en que ángulo empieza la curva), el ángulo que recorre ("angle") y la longitud de su *bounding box* ("bounding_len").

Cada obstáculo tiene la coordenada x (“x”), la coordenada y (“y”), anchura (“width”) y altura (“height”).

Si se quiere editar los colores del resaltado de sintaxis, esto se debe hacer desde “assets/colors.txt”. Dentro de este archivo podemos cambiar los colores entre los disponibles. Si queremos definir un nuevo color, esto debe hacerse desde la clase “TextEditor”, en los métodos “create_tags” y “remove_tags” (es intuitivo).

Las imágenes que se añadan (bien como botones o bien como modelos para los robots) deben de abrirse directamente desde el módulo o clase que la usa. Esto es por como Python gestiona la memoria, de esta manera funciona correctamente (aunque no es muy deseable). Los colores del análisis sintáctico se podrían mover a este módulo, quizá sería lo más correcto. En este caso se deja así por conveniencia y rapidez, además de que se considera que la lógica está muy acoplada y es muy poca como para generar una nueva clase.

10.4.3 Front-end

En el paquete “graphics” se encuentra toda la funcionalidad relacionada con el *front-end* de la aplicación. En concreto, podemos encontrarnos los módulos que muestra la figura 9.28. El módulo “gui” contiene toda la funcionalidad relacionada con la ventana de la aplicación, “controller” será el que se comunique con el resto de los módulos.

Los módulos “hud” y “drawing” son los que se encargan de dibujar los “canvas” de “tkinter” correspondientes al HUD y al dibujo de los robots respectivamente.

Con respecto a “layers” y “robot_drawings”, el primero se encarga de gestionar el movimiento (tanto por sketch como por teclado “WASD"), además de reiniciar el robot si se necesita (es decir, cambiar el circuito). También se encarga del ejecutar y parar la simulación, de ampliar o reducir la imagen y de configurar el “canvas” (estas dos últimas delegando en “drawing”).

Con respecto al segundo, se encarga de la parte de los robots correspondiente a su animación (no confundir con el módulo “robots” de “robot_components”, que se encarga de la parte del estado de los robots). Este módulo pedirá el cambio del estado de los robots según lo que se vayan encontrando en el entorno de la animación.

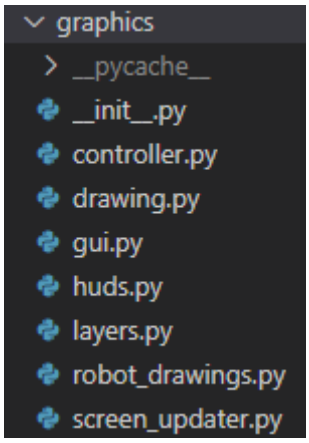


Figura 10.28 Módulos del paquete graphics – Visual Studio Code

10.4.4 Librerías de Arduino

En el paquete "libraries" podemos observar cuatro librerías de Arduino ("Servo", "String", "Serial" y la librería "Standard"). Dentro de los módulos homónimos, se pueden encontrar los métodos, aunque no todos están implementados. Para añadir un nuevo método, hay que seguir estos pasos:

1. Añadir el método a "get_methods". Se debe añadir como clave del diccionario el nombre exacto del método en Arduino. Como valor se debe escribir una tupla del tipo (tipo, nombre del método Python, lista de tipos de parámetros (en orden y con opcionales entre paréntesis), y si hace falta que el valor de un parámetro sea editado por la función (-1 si no, índice del parámetro en caso contrario).
2. Implementar el método en cuestión.
3. Añadir la librería al constructor de la clase "LibraryManager" de "libs" (si la librería es nueva).

Si se siguen bien estos pasos, salvo que la librería se use de una manera especial (como Serial, que mencionas la clase para llamar al método, como un método estático de Java); entonces, la librería debería funcionar directamente. En caso de que no sea así, deberá arreglarse desde el método "visit_function_call" correspondiente.

10.4.5 Entrada y salida

El paquete output contiene el módulo "console", que a su vez contiene varias clases:

- "ConsoleReportMessage": del que heredan:
  - "Error": Sirve para mostrar los errores por consola.
  - "Warning": Sirve para mostrar las advertencias por consola.
- "Console": Escribirá los mensajes que le sean pasados a la consola. Además, pasa dichos mensajes al "logger" para que los añada al log actual.
- "Logger": Escribe los mensajes de "console" en un log, además de aquellos fallos o tareas que escriba el propio Python en el log.

10.4.6 Componentes del robot (estado de Arduino)

El paquete “robot_components” contiene la funcionalidad relacionada con el estado del robot (es decir, la simulación de lo que lee la placa de los componentes conectados a ella). Se divide en cuatro módulos, “boards”, que contiene la funcionalidad de la placa (conectar y desconectar pines, leer valores, etc.), “elements”, que contiene la funcionalidad de los elementos como servos, joysticks, botones, etc. Además de “robot_state”, que contiene tres atributos, dos relacionados con el tiempo (sirve para realizar “delays”) y si se ha terminado de ejecutar el código del robot o no. Por último, el módulo robots contiene la lógica de los robots (une la placa con los elementos).

10.4.7 Carpeta temp

El código generado del sketch se añade a una carpeta “temp”. El módulo “command” realiza la importación del módulo creado y lo ejecuta (realmente es la única clase que nos interesa que lo ejecute).

10.4.8 Pruebas unitarias

Las pruebas unitarias están en el mismo directorio que el módulo que llama a la aplicación. Esto es para que las pruebas se ejecuten con los mismos nombres de paquetes y módulos importados (cuidado a la hora de realizar *imports*, estos pueden dar muchos problemas).

10.4.9 Añadir un nuevo robot

Si se quiere añadir un nuevo robot, habría que:

1. Añadir dicho robot en el módulo robots del paquete “robot_components”
2. Añadir el robot en el “robot_drawings” de “graphics”, construyendo toda la lógica de dibujado.
3. Añadir el robot en “layers”, para gestionar su movimiento y comportamiento.
4. Gestionar todas las opciones de cambiar el robot dentro del módulo “gui” de “graphics”.
5. Añadir un nuevo “hud” al módulo homónimo de “graphics”.
6. Añadir todos los gráficos de representación del robot.

Capítulo 11. Conclusiones y Ampliaciones

En este apartado se muestran las conclusiones sacadas del proyecto por parte del desarrollador, además de las posibles ampliaciones que se puedan hacer al sistema.

11.1 Conclusiones

El sistema desarrollado cumple e incluso supera las expectativas que se tenían en un principio. Principalmente, se ha realizado una labor más exhaustiva de desarrollo que la que se preveía en un inicio, con la consecuencia posterior de que el presupuesto y la planificación finales difieren mucho de las iniciales, lo cual no es bueno y es un error que debo tener muy en cuenta a futuro.

Aún a pesar de no haber cumplido con mis expectativas personales de terminar el proyecto a finales del 2021 o principios del 2022 (debido a la extensión del proyecto y otras circunstancias personales), me siento muy orgulloso del resultado obtenido. Si bien es cierto que quiero seguir desarrollando el proyecto y terminar de pulirlo, pues hay una serie de detalles del proyecto que no me han dejado satisfecho (principalmente la forma de ejecutar los bucles).

Este proyecto me ha hecho coger mucha experiencia en Python y aprender a usar tkinter, de modo que ahora puedo desenvolverme de una mejor manera tanto en el lenguaje como en la librería.

Además, este proyecto (junto con la asignatura de Diseño de Lenguajes de Programación) me ha hecho darme cuenta de lo complejas que son las gramáticas y la cantidad de trabajo que hay detrás de los lenguajes de programación; además de haber aprendido a crear un lenguaje de programación complejo (no olvidemos que Arduino es en realidad un subconjunto pequeño de C++).

En definitiva, estoy contento en general con el proyecto desarrollado, he aprendido a organizarme mejor a nivel de trabajo y, sobre manera, he aprendido a que no se debe subestimar la planificación y presupuestos iniciales. Me queda el mal sabor de boca de haber realizado una planificación tan pobre al inicio del proyecto, pero es algo que voy a mejorar para la próxima vez.

Con respecto al simulador, como exalumno que soy de la asignatura de Software para Robots, lo veo como una herramienta muy útil y que, por lo menos a mí, me hubiera gustado tener a la hora de desarrollar los sketches de ambos robots, el actuador lineal y todo lo relacionado con el móvil. Gran parte de lo que me llevó a escoger este proyecto como trabajo de fin de grado es poder realizar este simulador de modo que aquellos que cursen en un futuro la asignatura no tengan los mismos problemas que me encontré en su día cuando la cursé.

11.2 Ampliaciones

Este sistema puede tener (y tendrá) una gran cantidad de ampliaciones, pues la gramática no cubre todas las características de Arduino, además de que el propio simulador está pensado para poder añadir más robots en un futuro. Las futuras ampliaciones son:

- Arreglar los diferentes métodos de "visit_function_call", pues están llenos de fallos derivados del diseño inicial por cómo se ha ido ampliando el proyecto, principalmente debido a cómo están implementadas las librerías y como se realiza el acceso a métodos le librerías y objetos (es decir, como se analiza una sentencia del tipo "Serial.print"). Esto aportará mayor facilidad al futuro mantenimiento del sistema.
- Cambiar en la generación de código de bucles *for* y *while* a generadores. Esto hará que la interfaz gráfica responda correctamente en todo momento y no se quede congelada.
- Subrayado de los errores y advertencias en el editor de código. Esto facilitará a los usuarios la localización de errores y advertencias en el código.
- Realizar un sistema gráfico para el cambio de los pines. Esto será más intuitivo para el usuario, que está acostumbrado a la placa en el mundo real.
- Asignar la tecla "F1" a la ayuda, cumpliendo así los estándares en ese sentido.
- Mejorar el sistema de consola y de *logging*. Si algo ha dejado claro las pruebas de usabilidad es que el punto débil de la aplicación es la consola. Esto proporcionará una mejor experiencia de usuario.
- Crear una página en la que poder reportar errores del simulador, para así facilitar el proceso de mejora y de solución de problemas futuro. Incluso se puede usar el propio repositorio del proyecto en un futuro.
- Refinamiento de la interfaz de usuario. Ciertos botones o componentes deberían ser reordenados para mejorar la experiencia de usuario y la navegabilidad de la aplicación.
- Mejorar el rendimiento del código generado (el código de "script_arduino.py") y, en general, mejorar el proceso de generación de código.
- Mejorar el manual de ayuda al usuario, ampliando las funcionalidades que cubre y mejorando las explicaciones.
- Añadir más funcionalidad en forma de componentes de Arduino, para así permitir a los alumnos simular todas las prácticas de la asignatura que requieran de ello.
- Implementar el simulador de tal manera que se pueda ejecutar en otras plataformas (Linux y iOS).
- Añadir sonido a las notificaciones y alertas de la aplicación.
- Permitir cambiar el tipo y tamaño de la fuente desde el simulador.

Capítulo 12. Planificación del Proyecto y Presupuesto finales

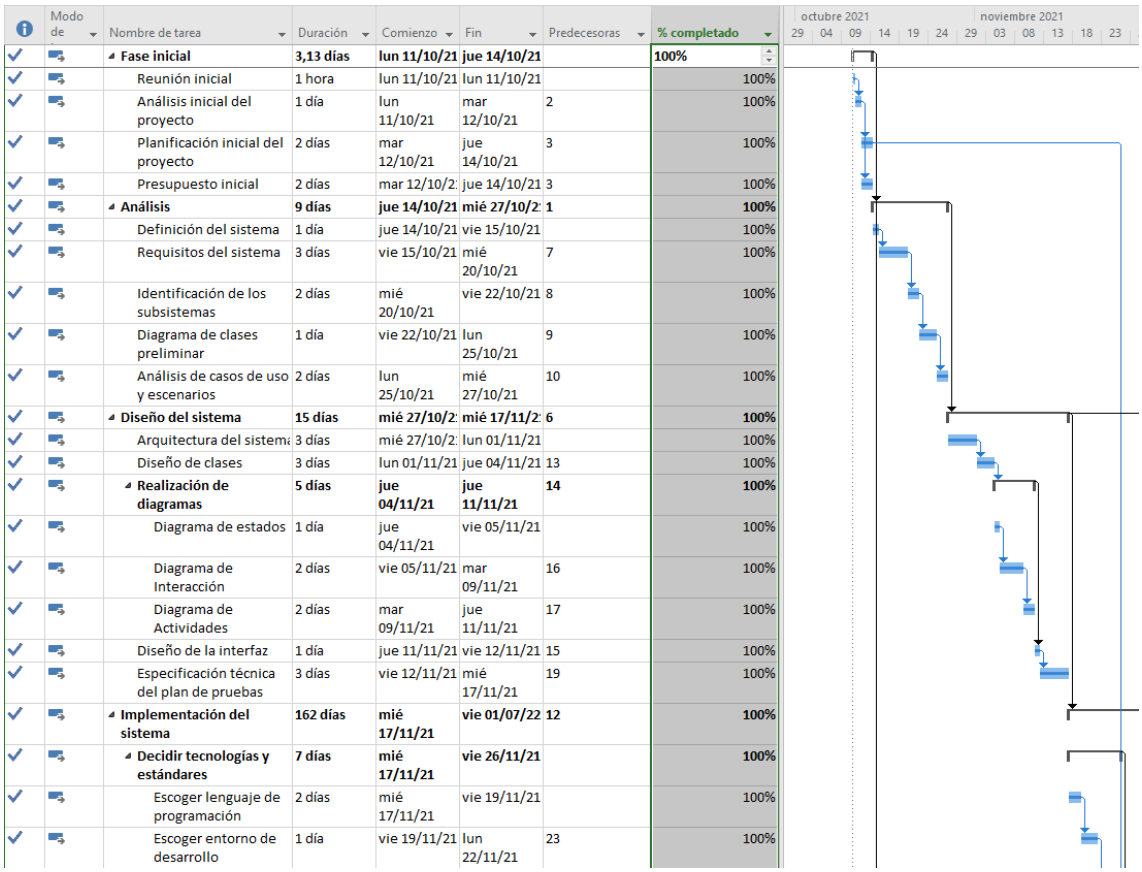
En este capítulo se mostrarán y explicarán la planificación y presupuesto finales del proyecto, actualizados con todos los cambios que han ido surgiendo a lo largo del proyecto.

12.1 Planificación Final

En esta sección se añadirá la planificación final y se explicará las diferencias respecto a planificación inicial y el porqué de estas. Principalmente, las diferencias se encuentran en la fase de implementación y de documentación del proyecto. Debido a una gran subestimación del tiempo que debería tomar la implementación de los diferentes módulos, se pueden notar unas diferencias muy grandes con respecto a la planificación inicial.

Además de subestimar el tiempo que iba a llevar implementar tanto el compilador como las pruebas de este y la interfaz gráfica, influye el hecho de que por distintas circunstancias (tanto del proyecto como personales) se cambió la fecha intencionada de presentación del proyecto de enero a junio. Esto ha hecho que el tiempo que finalmente ha tomado el desarrollo del proyecto sea del doble, y que, por ello, el presupuesto también se haya duplicado (en vez de tomar 4 meses, tomó 9 meses finalmente).

En esta planificación no se muestra una interrupción del proyecto durante unos 14 días en el mes de abril. Esta interrupción debida a, primero una enfermedad y después a un problema con el ordenador sobre el que se desarrolló el proyecto no será tenida en cuenta, pues no es excesivamente significativa (y se recupera con más trabajo en otros días). En las siguientes figuras se muestra la planificación final en todo detalle.



Modo de	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras	% completado
✓	**Fase inicial**	**3,13 días**	**lun 11/10/21**	**jue 14/10/21**		100%
✓	Reunión inicial	1 hora	lun 11/10/21	lun 11/10/21		100%
✓	Análisis inicial del proyecto	1 día	lun 11/10/21	mar 12/10/21	2	100%
✓	Planificación inicial del proyecto	2 días	mar 12/10/21	jue 14/10/21	3	100%
✓	Presupuesto inicial	2 días	mar 12/10/2	jue 14/10/21	3	100%
✓	**Análisis**	**9 días**	**jue 14/10/21**	**mié 27/10/2**	**1**	**100%**
✓	Definición del sistema	1 día	jue 14/10/21	vie 15/10/21		100%
✓	Requisitos del sistema	3 días	vie 15/10/21	mié 20/10/21	7	100%
✓	Identificación de los subsistemas	2 días	mié 20/10/21	vie 22/10/21	8	100%
✓	Diagrama de clases preliminar	1 día	vie 22/10/21	lun 25/10/21	9	100%
✓	Análisis de casos de uso y escenarios	2 días	lun 25/10/21	mié 27/10/21	10	100%
✓	**Diseño del sistema**	**15 días**	**mié 27/10/2**	**mié 17/11/2**	**6**	**100%**
✓	Arquitectura del sistema	3 días	mié 27/10/2	lun 01/11/21		100%
✓	Diseño de clases	3 días	lun 01/11/21	jue 04/11/21	13	100%
✓	**Realización de diagramas**	**5 días**	**jue 04/11/21**	**jue 11/11/21**	**14**	**100%**
✓	Diagrama de estados	1 día	jue 04/11/21	vie 05/11/21		100%
✓	Diagrama de Interacción	2 días	vie 05/11/21	mar 09/11/21	16	100%
✓	Diagrama de Actividades	2 días	mar 09/11/21	jue 11/11/21	17	100%
✓	Diseño de la interfaz	1 día	jue 11/11/21	vie 12/11/21	15	100%
✓	Especificación técnica del plan de pruebas	3 días	vie 12/11/21	mié 17/11/21	19	100%
✓	**Implementación del sistema**	**162 días**	**mié 17/11/21**	**vie 01/07/22**	**12**	**100%**
✓	**Decidir tecnologías y estándares**	**7 días**	**mié 17/11/21**	**vie 26/11/21**		**100%**
✓	Escoger lenguaje de programación	2 días	mié 17/11/21	vie 19/11/21		100%
✓	Escoger entorno de desarrollo	1 día	vie 19/11/21	lun 22/11/21	23	100%

Figura 12.1 Planificación final – Parte 1

Modo de	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras	% completado
	Escoger librerías	2 días	lun 22/11/21	mié 24/11/2:	24	100%
	Investigar estándares y normas a seguir	2 días	mié 24/11/21	vie 26/11/21	25	100%
	Módulo de compilador	**85 días**	**vie 26/11/21**	**vie 25/03/22**	**22**	**100%**
	Implementar gramática	7 días	vie 26/11/21	mar 07/12/21		100%
	Implementar AST	10 días	jue 16/12/21	jue 30/12/21	40	100%
	Implementar errores semáticos	14 días	vie 28/01/22	jue 17/02/22	41	100%
	Implementar generación de código	12 días	mié 09/03/22	vie 25/03/22	42	100%
	Implementar alertas	2 días	vie 26/11/21	mar 30/11/2:	4	100%
	Módulo de Interfaz gráfica de Usuario	50 días	vie 25/03/22	vie 03/06/22	27	100%
	Módulo de librerías	7 días	vie 03/06/22	mar 14/06/2:	33	100%
	Módulo de archivos	2 días	mar 14/06/2:	jue 16/06/22	34	100%
	Módulo de robots	7 días	jue 16/06/22	lun 27/06/22	35	100%
	Módulo de salida	4 días	lun 27/06/22	vie 01/07/22	36	100%
	Pruebas del sistema	**83 días**	**mar 30/11/2:**	**vie 25/03/22**	**12**	**100%**
	Pruebas unitarias	**71 días**	**mar 30/11/2:**	**mié 09/03/2:**		**100%**
	Pruebas de gramatica	7 días	mar 07/12/2:	jue 16/12/21	28	100%
	Pruebas de AST	21 días	jue 30/12/21	vie 28/01/22	29	100%
	Pruebas de errores	14 días	jue 17/02/22	mié 09/03/2:	30	100%
	Pruebas de alertas	1 día	mar 30/11/2:	mié 01/12/2:	32	100%
	Pruebas de integración y del sistema	3 días	mié 09/03/22	lun 14/03/22	39	100%
	Pruebas de usabilidad y accesibilidad	7 días	lun 14/03/22	mié 23/03/22	44	100%
	Pruebas de rendimiento	2 días	mié 23/03/22	vie 25/03/22	45	100%
	Documentación	**197 días**	**jue 14/10/21**	**lun 18/07/22**	**1**	**94%**
	Memoria del proyecto	**179 días**	**jue 14/10/21**	**mié 22/06/2:**		**100%**
	Documentar desarrollo	178 días	jue 14/10/21	mar 21/06/22		100%
	Conclusiones	0,25 días	mar 21/06/2:	mar 21/06/2:	49	100%
	Ampliaciones	0,25 días	mar 21/06/2:	mar 21/06/2:	50	100%
	Planificación final	0,25 días	mar 21/06/2:	mar 21/06/2:	51	100%

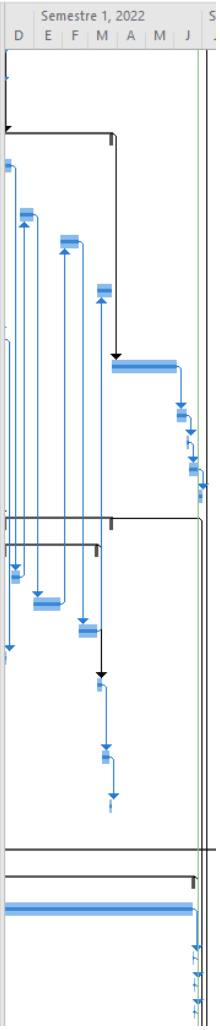


Figura 12.2 Planificación final – Parte 2

Modo de	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras	% completado
	Planificación final	0,25 días	mar 21/06/2:	mar 21/06/2:	51	100%
	Presupuesto final	0,25 días	mar 21/06/2:	mié 22/06/2:	52	100%
	Manuales del sistema	**1,99 días**	**mié 22/06/2:**	**vie 24/06/22**	**48**	**100%**
	Manual de usuario	1 día	mié 22/06/2:	jue 23/06/22		100%
	Manual de instalación	0,33 días	jue 23/06/22	jue 23/06/22	55	100%
	Manual de ejecución	0,33 días	jue 23/06/22	jue 23/06/22	56	100%
	Manual del programador	0,33 días	jue 23/06/22	vie 24/06/22	57	100%
	Revisión final del proyecto	7 días	vie 01/07/22	mar 12/07/22	38;21	0%
	Presentación	**4 días**	**mar 12/07/2:**	**lun 18/07/22**	**59**	**0%**
	Preparación	**3 días**	**mar 12/07/2:**	**vie 15/07/22**		**0%**
	Preparación del contenido	2 días	mar 12/07/22	jue 14/07/22		0%
	Realización de diapositivas	1 día	jue 14/07/22	vie 15/07/22	62	0%
	Exposición	1 día	vie 15/07/22	lun 18/07/22	61	0%

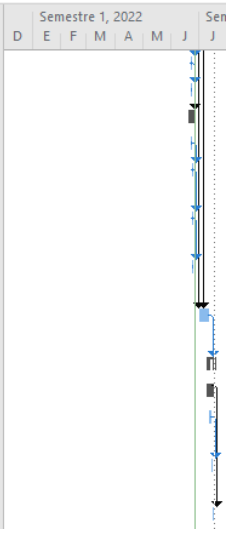


Figura 12.3 Planificación final – Parte 3

12.2 Presupuesto Final

A continuación, se mostrará el presupuesto final una vez completado el desarrollo del sistema. Debido a como se subestimó la fase de implementación y prueba del sistema, los sobrecostes de este son realmente notables. Se han añadido las horas que se reflejan en la planificación final al presupuesto (cambiando únicamente las fases de desarrollo y prueba, aunque documentar lleve más tiempo, no es una tarea que se realice 8 horas al día, sino que se realiza media hora, por ejemplo).

A parte de esto, se incluye la compra de un ratón (pues en realidad se tuvo que comprar, se me rompió en el parón de abril). Otras diferencias se deben al cambio de uso de los materiales de 4 meses a 9 meses por la extensión del proyecto. Tras realizar el presupuesto final se nota una diferencia de 42.004,52 €, para un coste final del proyecto de unos 78.546,37 €. Dicho sobrecoste se debe a una mala planificación inicial del proyecto.

Presupuesto final detallado				
Ítem	**Subítem**	**Concepto**	**Subtotal**	**Total**
1		Desarrollo del sistema		**49.792,62 €**
	1	Fase inicial	4.829,78 €	
	2	Análisis	7.620,87 €	
	3	Diseño	6.958,17 €	
	4	Implementación	19.328,91 €	
	5	Pruebas	11.054,89 €	
2		Documentación		**9.242,64 €**
3		Formación		**4.346,64 €**
4		Material		**953,62 €**
	1	Ordenador	537,28 €	
	2	Monitor	74,63 €	
	3	Escritorio	54,90 €	
	4	Silla	56,13 €	
	5	Teclado	66,06 €	
	6	Ratón	35,28 €	
	7	Placa Arduino	20,99 €	
	8	Robot Móvil	59,00 €	
	9	Actuador Lineal	49,36 €	
5		Licencias		**578,83 €**
	1	Microsoft Office 365	210,00 €	
	2	Windows 10	120,83 €	
	3	GitHub	80,00 €	
	4	Microsoft Project	168,00 €	
			Total	**64.914,35 €**
			IVA (21%)	**13.632,01 €**
			Total + IVA	**78.546,37 €**

Tabla 12.1 Presupuesto final del proyecto

Capítulo 13. Referencias Bibliográficas

En este capítulo se muestran todas las referencias bibliográficas usadas en el proyecto, tanto de libros y artículos como de páginas web.

13.1 Libros y Artículos

[González21] González G., Cristian. "Prácticas de la asignatura Software para Robots (4.1, 4.2, 7.1, 7.2, 8.1, 8.2 y 9.1)". Universidad de Oviedo. 2021

[BOE18] "Resolución de 22 de febrero de 2018, de la Dirección General de Empleo, por la que se registra y publica el XVII Convenio colectivo estatal de empresas de consultoría y estudios de mercado y de la opinión pública.". Gobierno de España. 2018. ISBN: 0212-033X

[Izquierdo20] Izquierdo C., Raúl. "Diseño de Lenguajes de Programación (Apuntes) – Conceptos básicos". Universidad de Oviedo. 2020.

[Parr11] Parr, Terrence. "LL(*): The Foundation of the ANTLR Parser Generator". Universidad de San Francisco. 2011.

[Gamma95] Gamma, Erich; Helm, Richard; Johnson, Ralph; Vlissides, John. "Patrones de diseño. Elementos de software orientado a objetos reutilizable". Addison Wesley. 2003. ISBN: 978-84-7829-059-8.

13.2 Referencias en Internet

[Tinkercad11] "Tinkercad". https://www.tinkercad.com/. 2021.

[Wokwi19] "Wokwi". https://wokwi.com/. 2021.

[PICSimLab16] Gambôa Lopes, L. Claudio. "PICSimLab - Prog. IC Simulator Lab.". https://sourceforge.net/projects/picsim/. 2021.

[UnoArduSim17] Simmons, Stan. "Simulator Download". https://www.sites.google.com/site/unoardusim/services. 2021.

[Java96] "Java | Oracle". https://www.java.com/es/. 2021.

[CSharp00] "Documentos de C#: inicio, tutoriales y referencias | Microsoft Docs". https://docs.microsoft.com/es-es/dotnet/csharp/. 2021.

[JavaScript95] "JavaScript | MDN". https://developer.mozilla.org/es/docs/Web/JavaScript. 2021.

[PyQT516] "Riverbank Computing | Introduction". https://pypi.org/project/PyQt5/ . 2021.

[wxPython98] "Welcome to wxPython!". https://www.wxpython.org/. 2021.

[Kivy11] "Kivy: Cross-platform Python Framework for NUI Development". https://kivy.org/#home. 2021.

[PyParsing04] "Welcome to PyParsing's documentation!". https://pyparsing-docs.readthedocs.io/en/latest/. 2021.

[PLY01] "PLY (Python Lex-Yacc)". http://www.dabeaz.com/ply/. 2021

[PyPEG09] "pyPEG – a PEG Parser-Interpreter in Python". https://fdik.org/pyPEG/. 2021.

[ANTLR89] "ANTLR". https://www.antlr.org/. 2021.

[Metrica01] "PAe – Métrica v.3". https://administracionelectronica.gob.es/pae_Home/pae_Documentacion/pae_Metodolog/pae_Metrica_v3.html. 2022

[Scrum86] "Home | Scrum.org". https://www.scrum.org/. 2022

[Sheldon22] Sheldon, Robert. "What is a compiler?". https://www.techtarget.com/whatis/definition/compiler. 2022

[Arduino05] "Arduino – Home". https://www.arduino.cc/. 2022

[ArduinoRef05] "Arduino Reference". https://www.arduino.cc/reference/en/?_gl=1*1x13cb2*_ga*Mzc4NTYyMzk1LjE2NTEwNzAwMDQ.*_ga_NEXN8H46L5*MTY1NTA0MjQ0MC41My4xLjE2NTUwNDI0NDQuNTY.-. 2022

[UML04] "Welcome To UML Web Site!". https://www.uml.org/. 2022

[Hamilton22] Hamilton, Thomas. "Accessibility Testing Tutorial: What is, Tools & Examples". https://www.guru99.com/accessibility-testing.html. 2022

[Wikipedia01] "Wikipedia". https://www.wikipedia.org/. 2022

[PEP801] "PEP8 – Style Guide for Python Code". https://peps.python.org/pep-0008/. 2022

[Python91] "Welcome to Python.org". https://www.python.org/. 2022

[PythonDocs91] "Python Frequently Asked Questions". https://docs.python.org/3/faq/. 2022

[VSCode15] "Visual Studio Code – Code editing. Redefined". https://code.visualstudio.com/. 2022

[NotepadPP03] "Notepad++". https://notepad-plus-plus.org/. 2022

[Pip08] "pip". https://pypi.org/project/pip/. 2022

[Git07] "Git". https://git-scm.com/. 2022

[GitHub08] "Where the world builds software". https://github.com/. 2022

[DrawIO17] "diagrams.net". https://draw.io/. 2022

[UMLet02] "UMLet - Free UML Tool for Fast UML Diagrams". https://www.umlet.com/. 2022

[OneDrive07] "Almacenamiento personal en la nube: Microsoft OneDrive". https://www.microsoft.com/es-es/microsoft-365/onedrive/online-cloud-storage . 2022

[Oakley13] Oackley, Bryan. "Tkinter adding line number to text widget answer". https://stackoverflow.com/questions/16369470/tkinter-adding-line-number-to-text-widget . 2021

[Tkinter88] "tkinter – Interface de Python para tcl/tk – documentación de Python". https://docs.python.org/es/3/library/tkinter.html . 2022

[Shvets14] Shvets, Alexander. "Refactorización y patrones de diseño". https://refactoring.guru/es . 2022

Capítulo 14. Apéndices

En este capítulo se añaden los diferentes apéndices del proyecto.

14.1 Glosario y Diccionario de Datos

En este apartado se exponen los diferentes términos empleados en el documento que requieren de explicación:

- AST: Siglas de Árbol de Sintaxis Abstracta (del inglés *Abstract Syntax Tree*), es un árbol ordenado, que parte de una raíz y que representa la estructura sintáctica de una cadena (o entrada) de una Gramática Libre de Contexto (GLC).
- GLC: Siglas de Gramática Libre de Contexto (o CFG en inglés, que significa *Context Free Grammar*). Explicado en profundidad en el apartado 3.4.
- Parse tree: es un árbol con raíz ordenado que representa la estructura sintáctica de una cadena (de acuerdo con una GLC)
- Front-end: Se trata de la parte de un sistema informático que se encargará de procesar la entrada y salida del sistema o, en otras palabras, se encargará de la interacción con el usuario.
- Back-end: Se trata de la parte de un sistema informático que se encarga de todo el procesamiento lógico de la aplicación o, en otras palabras, la parte interna de la aplicación, su motor.
- Actuador lineal: En el caso de este proyecto, se trata de un dispositivo que convierte el movimiento rotatorio de un servomotor en un movimiento lineal. En el caso del usado para la asignatura de Software para Robots está integrado por un servomotor, un joystick y dos botones que funcionan como límites de movimiento.
- Robot móvil: Se trata de una máquina automática que se puede mover en cualquier ambiente dado. En el caso de la asignatura de Software para robots estará formado por dos servomotores, tendrá dos o cuatro sensores infrarrojos para el seguimiento de la pista y un sensor de ultrasonidos.
- Python: es un lenguaje de programación interpretado.
- Parser: Del inglés, significa analizador. En el caso del compilador, examinará el código en busca de errores léxicos, sintácticos o semánticos y/o advertencias.
- Lexema: Agrupación de caracteres que conforman las sentencias del lenguaje.
- Tokens: conjunto de lexemas que pueden ser tratados como uno solo.
- JSON: Siglas de *JavaScript Object Notation*. Se trata de un lenguaje usado para el almacenamiento y/o intercambio de datos.

14.2 Contenido Entregado en el Archivo adjunto

En este apartado se mostrará el contenido entregado en el archivo adjunto.

14.2.1 Contenidos

En este apartado se realiza la descripción del contenido del archivo adjunto (directorios y para qué sirve cada cosa), la descripción de esta documentación y la de cualquier material que adicionalmente se entregue en la presentación.

14.2.1.1 Estructura general directorios del Archivo adjunto

En la siguiente table se muestra la estructura general de directorios del archivo adjunto.

Directorio	Contenido
./ Directorio raíz del Archivo adjunto	Contiene un fichero leeme.txt explicando toda esta estructura y el icono del proyecto
./simulator-robotic-software	Contiene toda la estructura de directorios del proyecto para desarrollo.
./ejecucion	Ficheros utilizados para la ejecución del proyecto.
./documentacion	Contiene toda la documentación asociada al proyecto.
./documentacion/img	Directorio que contiene las imágenes utilizadas en la documentación.
./documentacion/uml	Ficheros que genera la herramienta (Rose, *ArgoUML*, etc.) con la que se han generado los diagramas UML y de entidad relación.
./documentacionPython	Contiene la documentación de los diferentes módulos que componen la aplicación
./herramientas	Contiene los ficheros de instalación de las herramientas utilizadas para el desarrollo o puesta en marcha del proyecto. Como las herramientas pesan cerca de 100 mb juntas, se ha decidido incluir un archivo “leeme.txt” con los enlaces de descarga correspondientes.

Tabla 14.1 Estructura general del archivo adjunto

14.2.1.2 Estructura de Directorios de desarrollo

Se muestra aquí el contenido del directorio de desarrollo de la tabla anterior, incluyendo todos los directorios que deben depender del mismo.

Directorio	Contenido
./ Directorio raíz de "desarrollo"	Contiene los ficheros de proyecto, además de un archivo "build.py" para construir el ejecutable, el manual de usuario en pdf y "robot_data.json", que contiene los datos de los robots
./assets	Contiene las imágenes de los robots, así como de los HUDs y el icono de la aplicación. Además, contiene las palabras clave para el coloreado sintáctico
./buttons	Contiene las imágenes de los botones de la barra de herramientas superior.
./simulator	Directorio raíz del proyecto en sí, del código. Contiene cuatro tests que se ejecutan con "unittest" y un módulo main, que sirve para ejecutar la aplicación.
./simulator/compiler	Contiene la parte correspondiente al compilador (análisis léxico, sintáctico y semántico; además de advertencias y todo lo relacionado al AST.
./simulator/files	Contiene el módulo que gestiona el acceso a archivos ".ino" y al fichero de configuración ".json"
./simulator/graphics	Contiene todo lo relacionado con el *front-end* de la aplicación, desde la propia aplicación en sí, hasta las animaciones de los robots.
./simulator/libraries	Contiene la implementación de las cuatro librerías de Arduino necesarias para programar correctamente los robots: String, Servo, Standard y Serial.
./simulator/output	Contiene el módulo que gestiona la entrada, salida y logs del sistema.
./simulator/robot_components	Contiene el módulo que gestiona el estado de los diferentes componentes de los robots, además del estado de los propios robots en sí.
./tests	Contiene toda la documentación relativa al proyecto, incluyendo los ficheros generados por herramientas de generación de documentación automática como *Javadoc* o similar.
./tests/error-tests	Contiene archivos ".txt" con código Arduino empleados en las pruebas implementadas en el archivo "../simulator/test_errors.py"
./tests/file-tests	Contiene archivos ".txt" con código Arduino empleados en las pruebas implementadas en

	el archivo "../simulator/test_ast.py"
./tests/grammar-tests	Contiene archivos ".txt" con código Arduino empleados en las pruebas implementadas en el archivo "../simulator/test_grammar.py"
./tests/warning-tests	Contiene archivos ".txt" con código Arduino empleados en las pruebas implementadas en el archivo "../simulator/test_warnings.py"

Tabla 14.2 Estructura del directorio de desarrollo

14.2.2 Código Ejecutable e Instalación

Para ejecutar el código basta con ejecutar el módulo "main.py", sin ninguna acción adicional.

Para instalar la aplicación basta con, o bien descargar el archivo comprimido del repositorio del proyecto y descomprimirlo en el directorio que se quiera, o bien copiar el código del directorio ejecución y realizar lo mismo que con el directorio descomprimido. Una vez dentro del directorio, basta con ejecutar el archivo ".exe" presente. Para más información consultar el apartado 10.1 y el apartado 10.2.

14.3 Código Fuente

El código fuente se encuentra en la carpeta "ejecucion" de los archivos adjuntos, su contenido está explicado brevemente en el apartado 14.2.1.

No obstante, y si el código no está disponible en dicho directorio por falta de espacio a la hora de subir los archivos u otro tipo de problemas, entonces el código se puede encontrar en el repositorio del proyecto.

14.4 Actas de reuniones

En este apartado se muestran las actas de reuniones.

Acta de reunión

Fecha

15 de octubre de 2021 a las 11:45 (CEST).

Resumen

En esta reunión se han tratado los siguientes puntos:

- La planificación inicial del proyecto.
  - Se ha llegado a la conclusión de que hay que dividir mejor las tareas de implementación y diseño general del sistema.
- El presupuesto inicial del proyecto, extrayendo las siguientes conclusiones:
  - Se tiene que añadir la amortización de los materiales y licencias de software establecidos en el presupuesto.
  - Se debe revisar el sueldo-coste de los trabajadores (debe incluir Seguridad Social y otros costes).
  - Falta añadir el beneficio que quiere la empresa.
  - Falta aplicar el IVA a todo el presupuesto.
- Un mockup programado en el lenguaje en el que se va a desarrollar la aplicación, del cual se ha establecido que es mejor que se pueda poner en pantalla completa, manteniendo la misma ratio de tamaño de los componentes.

Siguiente reunión

Se ha establecido como tareas:

1. Implementar las mejoras anteriormente resumidas.
2. Investigar más sobre el propio diseño del sistema.

Medio de la reunión

La reunión se ha realizado de forma telemática, utilizado el programa Microsoft Teams.

Acta de reunión

Fecha

28 de octubre de 2021 a las 11:45 (CEST).

Resumen

En esta reunión se han tratado los siguientes puntos:

- La planificación inicial del proyecto.
    - Se han expuesto las tareas del desarrollo que se van a realizar, dividiéndolas de una mejor manera con respecto a la última reunión (15 de octubre).
- El presupuesto inicial del proyecto.
    - Se ha expuesto el presupuesto inicial completado en su totalidad, incluyendo ambos (el de empresa y el del cliente).

Siguiente reunión

Para la siguiente reunión se ha acordado:

- Añadir presupuesto y planificación inicial a la documentación
- Realizar el análisis del sistema
- Comenzar con el desarrollo del sistema

Medio de la reunión

La reunión se ha realizado de forma telemática, utilizando el programa Microsoft Teams.

Acta de reunión

Fecha

9 de diciembre de 2021 a las 11:45 (CET).

Resumen

En esta reunión se han tratado los siguientes puntos:

- Se han expuesto los modelos que se van a usar para la representación de los robots, habiendo escogido el 2D como forma de representación final.
- Se han mostrado varios arreglos realizados a la interfaz gráfica de usuario.

Siguiente reunión

Para la siguiente reunión se ha acordado:

- Mejorar el dibujo del robot móvil.
- Realizar una serie de mejoras a la interfaz gráfica:
    - Ajustar el padding entre componentes.
    - Establecer los separadores correctos entre componentes.

Medio de la reunión

La reunión se ha realizado de forma telemática, utilizando el programa Microsoft Teams.

Acta de reunión

Fecha

25 de noviembre de 2021 a las 11:45 (CET).

Resumen

En esta reunión se han tratado los siguientes puntos:

- Se ha establecido definitivamente a Python como lenguaje de desarrollo de la aplicación.
- Se han expuesto las librerías que se van a usar para el desarrollo de la aplicación.
- Se ha debatido la conveniencia de representar los robots en 3D o en 2D.

Siguiente reunión

Para la siguiente reunión se ha acordado:

- Decidir definitivamente entre representación 2D y representación 3D.
- Realizar la representación de los robots de acuerdo con lo escogido anteriormente.

Medio de la reunión

La reunión se ha realizado de forma telemática, utilizando el programa Microsoft Teams.

Acta de reunión

Fecha

23 de diciembre de 2021 a las 11:45 (CET)

Resumen

En esta reunión se han tratado los siguientes puntos:

- Mención de que se ha completado el punto 2 de la documentación.
- Petición de revisión del punto 5.2.1 de la documentación (Requisitos del sistema).
- Presentación de mejoras en la interfaz gráfica de usuario:
    - Se han añadido barras de desplazamiento verticales y horizontales a la parte del editor de texto de la aplicación.
    - Se pretende añadir los números de línea al editor, se ha mencionado que se está experimentando con ello.
- Presentación de plazos para las navidades.

Siguiente reunión

Para la siguiente reunión se ha acordado:

- Retrasar la fecha del 6 de enero que hubiese correspondido la reunión al 13 de enero.
- Se ha establecido que se va a completar el punto 5 de la documentación.
- Se ha puesto como objetivo terminar los puntos 6 y 7 de la documentación.
- Se ha puesto como objetivo tener desarrollado la mayor parte del sistema en la siguiente reunión.

Medio de la reunión

La reunión se ha realizado de forma telemática, utilizando el programa Microsoft Teams.

Acta de reunión

Fecha

13 de enero de 2022 a las 11:45 (CET)

Resumen

En esta reunión se han tratado los siguientes puntos:

- Se han tratado los avances en el compilador:
  - Se ha presentado la gramática usada para el lenguaje.
  - Se han mostrado las pruebas unitarias empleadas para probar que la generación de tokens es correcta.
  - Se ha enseñado las maneras de mostrar los tokens generados por el programa.
- Se ha notificado la finalización del punto de la documentación asociado a los requisitos del sistema.
- Se han resuelto las siguientes dudas:
  - El compilador en vez de generar código va a llamar directamente a los métodos que simulen el comportamiento de los robots.
  - Los pines del servomotor se comprobarán cuando se asocien por el método “attach”.
- Se ha establecido que el proyecto será presentado finalmente en la convocatoria de junio correspondiente al año 2022.

Siguiente reunión

Para la siguiente reunión se ha acordado:

- Finalizar las pruebas unitarias para probar la generación de tokens.
- Avanzar en el patrón “Visitor” necesario para la fase de “Generación de código”, tener programado al menos la estructura de nodos del AST (Árbol de Sintaxis Abstracta).

Medio de la reunión

La reunión se ha realizado de forma telemática, utilizando el programa Microsoft Teams.

Acta de reunión

Fecha

27 de enero de 2022 a las 11:45 (CET)

Resumen

En esta reunión se han tratado los siguientes puntos:

- Se han mostrado los avances con respecto al Árbol de Sintaxis Abstracta (AST).
    - Se ha mostrado las diferentes clases de nodos del AST.
    - Se ha enseñado la clase “Visitor” que genera el árbol.
    - Todo ello se ha probado con un ejemplo.
- Se ha enseñado el diagrama UML de casos de uso, correspondiente al punto 5.2.4 de la documentación del proyecto.
- Se ha comunicado un fallo con el repositorio durante la semana anterior a la reunión.
    - Dicho fallo fue provocado por no realizar un “fetch” antes de realizar el “push” al repositorio.

Siguiente reunión

Para la siguiente reunión se han establecido como objetivos:

- Terminar el punto 5 de la documentación.
- Realizar las pruebas unitarias con respecto a la generación del AST, en caso de no poder terminarlas, avanzar lo máximo posible.
- Comenzar la implementación de las librerías estándar de Arduino necesarias para el funcionamiento del simulador en caso de terminar las tareas anteriores.

Medio de la reunión

La reunión se ha realizado de forma telemática, utilizando el programa Microsoft Teams.

Acta de reunión

Fecha

10 de febrero de 2022 a las 11:45 (CET).

Resumen

En esta reunión se han tratado los siguientes puntos:

- Se ha comentado la finalización del punto 5 de la documentación.
- Se ha reportado el avance con respecto a las pruebas del AST:
  - Las pruebas planificadas en un principio ya han sido finalizadas.
- Se han indicado los cambios con respecto a la definición de arrays:
  - Ahora se pueden crear arrays multidimensionales de todas las formas que se puede en Arduino.
- Se han indicado los cambios realizados al generador de AST:
  - Cambiar la forma en la que se asignaban ciertos valores de manera que no se asigne el contexto del parse tree.
  - Adaptar el "Visitor" a las mejores de los arrays.
- Se han resuelto una serie de dudas:
  - Se ha decidido mantener por el momento las sentencias "define".
  - Se ha decidido no implementar las terminaciones de enteros u (unsigned) y L (long).

Siguiente reunión

Para la siguiente reunión se han establecido como objetivos:

- Realizar la detección de errores sintácticos y otros tipos de errores.
- Probar dicha detección.
- En caso de terminar las tareas anteriores, comenzar con las librerías estándar y la generación de código.

Medio de la reunión

La reunión se ha realizado de forma telemática, utilizando el programa Microsoft Teams.

Acta de reunión

Fecha

24 de febrero de 2022 a las 11:45 (CET).

Resumen

En esta reunión se han tratado los siguientes puntos:

- Se ha comentado un imprevisto con el hardware usado para la realización del proyecto:
  - El ratón del ordenador empleado se ha estropeado, siendo necesario reemplazarlo por otro. Con ello se han perdido 4 días de trabajo, con las consecuencias de reorganización que ello conlleva.
- Se han explicado los avances realizados:
  - Se ha creado la clase "Visitor" del AST.
  - Se han mejorado ciertos puntos de la gramática formal del compilador:
    - Se han fusionado las reglas que analizaban las sentencias "#include" del código.
    - Se ha cambiado el nombre de las reglas "Definition" a "Declaration".
    - Se han fusionado y simplificado las reglas que analizaban las declaraciones.
  - A consecuencia del anterior punto, se han actualizado el AST, así como su generador y las pruebas pertinentes para reflejar los cambios realizados.
- Se han tratado las siguientes dudas:
  - No se deberán implementar clases ni structs.
  - Los "#define" y el código en general deberá respetar el tamaño máximo especificado por la placa "Arduino UNO".
  - El acceso a métodos y constantes de clases y librerías se realizará de una manera simple, buscando los métodos correspondientes según el nombre de la clase.

Siguiente reunión

Para la siguiente reunión se han establecido como objetivos:

- Cumplir los objetivos no realizados la anterior semana por los imprevistos anteriormente comentados.
- Arreglar la documentación:
  - Insertar las referencias no insertadas y/o no actualizadas.
  - Corregir lo expuesto en el punto 5 según lo comentado por parte del tutor.

Medio de la reunión

La reunión se ha realizado de forma telemática, utilizando el programa Microsoft Teams.

Acta de reunión

Fecha

10 de marzo de 2022 a las 11:45 (CET).

Resumen

En esta reunión se han tratado los siguientes puntos:

- Se han mencionado las implementaciones nuevas en el lenguaje formal del compilador:
  - Las expresiones de incremento y decremento (i++ o i-- y sus variantes) se han movido junto con el resto de las expresiones, reordenándolas por su prioridad.
- El desarrollo de la forma de analizar y reordenar los arrays:
  - Se realiza de la misma forma que el propio compilador de Arduino (ver referencias).
  - De esta forma se descarta representar los arrays en árbol, que es como se expresa en el analizador sintáctico.
- La implementación de dos nuevas pruebas para el AST:
  - Una para probar la declaración de arrays (nombre de la variable, tamaño, elementos y dimensiones).
  - La otra se encarga de probar las llamadas a arrays (nombre del array a llamar e índice que se busca).
- Definición de la gramática abstracta con el fin de acotar los errores que debe probar el analizador semántico.
- Inicio de la implementación del analizador semántico del lenguaje.

Siguiente reunión

Para la siguiente reunión se ha acordado:

- Sacar los mensajes de error a un archivo de internacionalización.
- Implementar el analizador semántico y sus pruebas.
- Comenzar el desarrollo de las librerías de Arduino necesarias para la ejecución del programa (y si es posible, terminarlo).

Medio de la reunión

La reunión se ha realizado de forma telemática utilizando el programa Microsoft Teams.

Acta de reunión

Fecha

24 de marzo de 2022 a las 11:45 (CET).

Resumen

En esta reunión se han tratado los siguientes puntos:

- Se ha comunicado un retraso de una semana imprevisto, debido a enfermedad.
- El desarrollo del analizador semántico del compilador y las modificaciones debidas a este:
    - Modificado el acceso a los valores de los arrays, en vez de analizarlos en árbol, se analizan en lista.
    - Modificado la llamada a funciones, ahora se considera una regla de tipo expresión, y se analiza imitando la EBNF de C++.
    - Modificado el acceso a miembros de clases, de manera similar a lo mencionado con respecto a la llamada a funciones.
    - El desarrollo de dos analizadores semánticos, uno para las declaraciones y otro para los demás errores semánticos.
- El desarrollo de las pruebas del analizador semántico:
    - Se han probado los casos en los que se generan errores.
    - Los errores probados son de declaración, tipos, índice de acceso a arrays y sentencias especiales; principalmente.
- Se han actualizado las pruebas realizadas anteriormente sobre el AST y la gramática, para adaptarlos a los cambios realizados en esta última.

Siguiente reunión

Para la siguiente reunión se ha acordado:

- Realizar pruebas positivas sobre los errores (tanto sintácticos como semánticos).
- Comenzar con el desarrollo de la representación gráfica y de la salida por consola.
- En caso de terminar las anteriores tareas antes de lo previsto, desarrollar las librerías de Arduino necesarias, adaptadas al simulador.

Medio de la reunión

La reunión se ha realizado de forma telemática utilizando el programa Microsoft Teams.

Acta de reunión

Fecha

7 de abril de 2022 a las 11:45 (CEST).

Resumen

En esta reunión se han tratado los siguientes puntos:

- La realización de una reimplementación de la parte gráfica de la aplicación.
- La implementación de ambos robots:
    - Para el actuador lineal, se ha implementado la representación gráfica completa y el movimiento que va a realizar.
    - Para el robot móvil se ha programado el movimiento en los dos ejes (x e y), además de la representación completa del robot y de la pista de ensayos.
- La implementación del dibujo que representará las acciones de los robots.
- La conexión entre la parte de la interfaz gráfica de usuario y la representación gráfica de los robots.

Siguiente reunión

Para la siguiente reunión se ha acordado:

- Mejorar el movimiento de los robots, suavizándolo (evitando “que pegue saltos”.
- Implementar el resto de comportamiento que falta en ambos robots:
    - En el actuador lineal el tope de movimiento y el cambio de botones a pulsado si se llega al límite.
    - En el robot móvil crear los obstáculos, el intercambio de estados de sensores según el contexto y tanto la superposición del sensor de luz con la pista como la detección de un obstáculo por parte del sensor de ultrasonidos.
- Si se termina todo lo anterior (siguiendo el orden establecido):
    - Implementar las librerías que servirán para ejecutar el código.
    - Realizar el coloreado de código en la interfaz gráfica.
    - Implementar la salida por consola.
    - Implementar los errores del usuario al programar el robot.
    - Implementar la fase de “generación de código”.

Medio de la reunión

La reunión se ha realizado de forma telemática utilizando el programa Microsoft Teams.

Acta de reunión

Fecha

21 de abril de 2022 a las 11:45 (CEST).

Resumen

En esta reunión se han tratado los siguientes puntos:

- Avances en la GUI (Interfaz Gráfica de Usuario):
  - Implementación del Actuador lineal: sus bordes, botones y movimiento.
  - Implementación del Robot móvil: su movimiento, rotación, detección de obstáculos y detección de pista.
  - Comentado que, aunque el robot es hexagonal, su bounding box (o caja de colisión) es cuadrada.
  - Se ha comentado la intención de mostrar los datos del robot a través de un HUD:
    - En caso del robot móvil mostrará el movimiento de las ruedas, si detecta o no pista en cada sensor de luz y si detecta o no obstáculos con alguno de los dos sensores de ultrasonidos.
    - En el caso del actuador lineal se mostrará el movimiento del robot y si los botones están pulsados o no.
- Se ha comentado la posibilidad de que no se puedan hacer más pruebas unitarias, debido a la complejidad que tendrían estas.

Siguiente reunión

Para la siguiente reunión se ha acordado:

- Terminar la GUI.
- Terminar la consola y el logger.
- Comenzar a desarrollar las librerías.
- Si todo esto se termina:
  - Desarrollar la generación de código.
  - Desarrollar la detección de errores restante.

Medio de la reunión

La reunión se ha realizado de forma telemática utilizando el programa Microsoft Teams.

Acta de reunión

Fecha

5 de mayo de 2022 a las 11:45 (CEST).

Resumen

En esta reunión se han tratado los siguientes puntos:

- Se ha acordado el cambio de reuniones bisemanales (cada 2 semanas) a una reunión semanal en lo que resta de proyecto.
- Se han mostrado los avances en el front end:
    - Se ha añadido un checkbox para cambiar entre movimiento con teclado y movimiento con código.
    - Se ha añadido la funcionalidad a los filtros de consola.
    - Se ha creado una ventana para configurar los puertos usados de la placa de Arduino.
    - Se ha introducido la funcionalidad de realizar entrada por consola mediante un campo de texto y un botón
- Se han mostrado los avances en el back end:
    - Se han implementado los métodos de las librerías necesarios para que los robots funcionen.
    - Se ha preparado el movimiento de los robots para poder realizarse por código.
    - Se ha implementado la consola para la salida errores, alertas y texto normal, además de loggear los mensajes a un archivo.
    - Se ha comenzado el desarrollo de la generación de código.

Siguiente reunión

Para la siguiente reunión se ha acordado:

- Dejar terminado (o casi terminado) la aplicación.
- Realizar la siguiente reunión dentro de una semana a la misma hora.
- Investigar la generación del programa ejecutable para diversas plataformas.

Medio de la reunión

La reunión se ha realizado de forma telemática utilizando el programa Microsoft Teams.

Acta de reunión

Fecha

12 de mayo de 2022 a las 11:45 (CEST).

Resumen

En esta reunión se han tratado los siguientes puntos:

- Se han mostrado los avances finales en materia de interfaz gráfica de usuario:
  - Se han añadido dos opciones de robot móvil, una con dos sensores de luz y otra con 4 sensores de luz.
  - Se ha cambiado la implementación del joystick (necesario para el movimiento del actuador lineal): ahora se representa mediante dos componentes 'slider' para cada eje del joystick y un botón para la pulsación del joystick.
  - Se ha terminado de implementar la configuración de pines de la placa Arduino (la que corresponda según el robot y la placa que tiene cada uno).
- Se han mostrado los arreglos en materia de compilador y generación de errores semánticos (además de las librerías del compilador):
  - Se han creado los métodos de 'libraries.getMethods()', que devuelven una tupla que contiene el tipo que retorna la función, la función en sí y los tipos de los parámetros (opcionales y obligatorios) que necesita la función.
  - Se han implementado los últimos errores semánticos, que corresponden a las funciones no definidas por el usuario (contenidas en librerías por defecto).
  - Se han añadido ciertas constantes del compilador de Arduino:
    - Pines analógicos (A0-A5, que corresponden con los valores 14-19)
    - LOW y HIGH, que corresponden con los valores 0 y 1 respectivamente.
    - INPUT, OUTPUT, INPUT_PULLUP, que corresponden con '0x0', '0x1' y '0x2' respectivamente.

Siguiente reunión

Para la siguiente reunión se ha acordado:

- Avanzar lo máximo posible (sino terminar) la parte de programación del proyecto, es decir, terminar la generación de código y los errores básicos que se deban añadir.
- Revisar (sino para la siguiente reunión, lo antes posible) el punto de alternativas para añadir el porque se escogió realizar un 'transpilador'.

Medio de la reunión

La reunión se ha realizado de forma telemática utilizando el programa Microsoft Teams.

Acta de reunión

Fecha

19 de mayo de 2022 a las 11:45 (CEST).

Resumen

En esta reunión se han tratado los siguientes puntos:

- Avances del proyecto:
  - Se ha separado la interfaz gráfica (vista) del controlador del robot; es decir, el controlador realizará la gestión de los robots y mandará a la vista que los muestre según su estado.
  - Se ha terminado la fase de detección de errores semánticos del código.
  - Se ha realizado la fase de generación de código Python, todavía en progreso.
- Problemas del proyecto:
  - Se a llegado a un impasse con respecto a la ejecución del código Arduino, especialmente si este tiene bucles. Cuando un sketch de Arduino tiene bucles, esto genera que Python haga lo siguiente:
    - En caso de trabajar en un único hilo, la interfaz gráfica se congela.
    - En caso de trabajar con múltiples hilos, el programa no funciona paralelamente, debido a que Python funciona realmente utilizando un único hilo.

Siguiente reunión

Para la siguiente reunión se ha acordado:

- Trabajar en lo restante de la programación del proyecto.
- En caso de no poder solucionar el problema de los bucles, marcar en la documentación como problema a resolver a futuro.
- En caso de terminar con la programación, realizar la documentación.

Medio de la reunión

La reunión se ha realizado de forma telemática utilizando el programa Microsoft Teams.

Acta de reunión

Fecha

26 de mayo de 2022 a las 11:30 (CEST).

Resumen

En esta reunión se han tratado los siguientes puntos:

- Avances del proyecto:
  - Se han creado todos los circuitos/bancos de prueba para probar los robots móviles, tanto el de 2 sensores infrarrojos como el de 4 sensores infrarrojos.
  - Se han probado los tres robots (incluido el actuador lineal) para probar que funcionan correctamente.
  - Se han arreglado los fallos que impedían el correcto funcionamiento de cada uno de los robots.
- Solución de problemas del proyecto:
  - Se ha solucionado el problema de los bucles. La solución empleada ha sido actualizar manualmente la interfaz gráfica de usuario al final de cada iteración del bucle.
- Problemas del proyecto:
  - El ejercicio 8.2 de laberintos no funciona correctamente por un error en la fase de análisis de tipos dentro de la fase de análisis semántico.

Siguiente reunión

Para la siguiente reunión se ha acordado:

- Arreglar el fallo del ejercicio 8.2.
- Investigar sobre el coloreado de código.
- Probar el simulador con casos reales enviados por el tutor.
- Realizar las pequeñas correcciones restantes que necesita el proyecto.

Medio de la reunión

La reunión se ha realizado de forma telemática utilizando el programa Microsoft Teams.

Acta de reunión

Fecha

2 de junio de 2022 a las 11:45 (CEST).

Resumen

En esta reunión se han tratado los siguientes puntos:

- Avances del proyecto:
  - Se han implementado los botones y acciones de deshacer y rehacer, tanto en Interfaz gráfica como en submenús.
  - Se han implementado los botones de importar y guardar archivo, tanto en interfaz gráfica como en submenús.
  - Se ha implementado el resaltado/coloreado de sintaxis.
  - Se han implementado las advertencias al usuario (de código no recomendable o de métodos no implementados).
  - Se han añadido mnemónicos a cada acción de menú que lo requiera (todas menos salir y acerca de).
- La parte de programación del proyecto queda así finalizada, faltando un par de detalles:
  - La ayuda al usuario.
  - La internacionalización del programa (o al menos, la preparación de este para ello).

Siguiente reunión

Para la siguiente reunión se ha acordado:

- Avanzar lo máximo posible en la documentación, sobre todo en aquellos puntos que requieran más trabajo.
- Empezar a redactar los diferentes manuales requeridos.
- Probar con código hecho por alumnos de la asignatura los diferentes robots.

Medio de la reunión

La reunión se ha realizado de forma telemática utilizando el programa Microsoft Teams.

Acta de reunión

Fecha

16 de junio de 2022 a las 12:30 (CEST).

Resumen

En esta reunión se han tratado los siguientes puntos:

- Aplicación:
  - Se ha generado el programa ejecutable para Windows. También se ha probado que funcione en un ordenador con Python instalado y en otro sin instalar.
  - Se ha cancelado la versión de Ubuntu por problemas relacionados con librerías externas usadas por el proyecto.
  - Se han arreglado varios bugs y se ha añadido el manual de usuario como manual de ayuda.
- Documentación:
  - Se ha revisado el punto 6, indicando las correcciones y las partes añadidas después de la revisión del punto.
  - Se ha añadido las pruebas de accesibilidad.
  - Se ha comentado que se va a borrar la parte del diagrama de despliegue correspondiente a Linux.

Siguiente reunión

Para la siguiente reunión se ha acordado:

- Terminar los manuales de la documentación.
- Terminar el punto 7 de la documentación.
- Avanzar en los puntos restantes.
- Si se hace todo lo anterior, revisar el punto 2.
- Cambiar el manual de usuario por el mismo, pero sin número de apartado en los títulos.

Medio de la reunión

La reunión se ha realizado de forma telemática utilizando el programa Microsoft Teams.

Acta de reunión

Fecha

23 de junio de 2022 a las 12:30 (CEST).

Resumen

En esta reunión se han tratado los siguientes puntos:

- Avances de la documentación.
    - Se han mostrado los avances dentro del Capítulo 3, y se han comentado los subapartados que se van a realizar en dicho capítulo.
    - Se ha comentado las tareas restantes por realizar y el orden en que van a ser realizadas.

Siguiente reunión

Para la siguiente reunión se ha acordado:

- Terminar la documentación.

Medio de la reunión

La reunión se ha realizado de forma telemática utilizando el programa Microsoft Teams.

Acta de reunión

Fecha

28 de junio de 2022 a las 14:00 (CEST).

Resumen

En esta reunión se han tratado los siguientes puntos:

- Finalización de la documentación: se han terminado los siguientes puntos: 1.2.5, 2.3.5, 3, 7.4, 8, 9, 11, 12.
- Se han aclarado los pasos a seguir para solicitar la presentación del trabajo.

Siguiente reunión

Para la siguiente reunión se ha acordado:

- Revisar la documentación:
    - Localizar errores gramaticales.
    - Arreglar errores técnicos que se encuentren.
    - Finalizar anexos.

Medio de la reunión

La reunión se ha realizado de forma telemática utilizando el programa Microsoft Teams.

Acta de reunión

Fecha

05 de julio de 2022 a las 10:30 (CEST).

Resumen

En esta reunión se han tratado los siguientes puntos:

- Se han comentado los diferentes cambios hechos a la documentación tras las revisiones, sobre manera se han realizado correcciones ortográficas y gramaticales, y se ha actualizado el punto 9 de pruebas con los resultados finales de las pruebas de usabilidad.
- Se han comentado los pasos a seguir una vez entregado el trabajo, es decir, la presentación y los contenidos que debe tener.

Siguiente reunión

Para la siguiente reunión se ha acordado:

- Subir la documentación y archivos asociados a ella.
- Comenzar el desarrollo de la presentación.

Medio de la reunión

La reunión se ha realizado de forma telemática utilizando el programa Microsoft Teams.