Universidad Zaragoza
1542

Trabajo Fin de Grado

Grado en Ingeniería Informática

RobotWars: Desarrollo de un videojuego para el aprendizaje de programación

RobotWars: Development of a videogame for the learning of computer programming

Autor

David Carmona Casado

Director

Eduardo Mena Nieto

Escuela de Ingeniería y Arquitectura 2019 – 2020

AGRADECIMIENTOS

Después de un intenso período de dos años, hoy es el día: escribo este apartado de agradecimientos para finalizar mi trabajo de fin de grado. Ha sido un período de aprendizaje intenso, no solo en el campo técnico, sino también a nivel personal. Escribir este trabajo ha tenido un gran impacto en mí y es por eso por lo que me gustaría agradecer a todas aquellas personas que me han ayudado y apoyado durante este proceso.

En primer lugar, me gustaría agradecer a mis padres, Virginia y Manuel, por sus sabios consejos y su comprensión. Han supuesto en todo momento un apoyo moral muy importante, ayudándome a superar los obstáculos a nivel psicológico que un proyecto de tal envergadura lleva consigo.

Además, me gustaría darles las gracias a mis compañeros. En especial a Catalin, Adrián, Daniel y León, que me han ayudado en todo lo posible. Por el apoyo en todo momento y cariño hacia mí. Sin ellos esta experiencia no habría sido posible y he aprendido de cada uno de ellos.

También quiero hacer una mención especial a Almudena, mi pareja. Gracias por haberme acompañado en esta etapa de mi vida sin importar las dificultades, por ayudarme en momentos de necesidad y por ser mi psicóloga personal en algunas ocasiones.

Finalmente, me gustaría agradecer a mi director, Eduardo Mena Nieto, por haber tutorizado y enfocado el proyecto como es debido. Me ha sabido guiar en todo momento y enseñarme nuevas referencias que me han sido de gran ayuda para el proyecto.

Resumen ejecutivo

A lo largo de la historia, los educadores siempre han buscado conectar con los alumnos para poder captar su interés y hacer que la educación se convierta en una experiencia vital. El pedagogo Johann Heinrich Pestalozzi conseguía esto en el siglo XIX, enseñando a niños a contar bloques de madera. Sus sucesores han seguido su legado y hoy en día utilizan variedad de medios con los que motivar a sus alumnos, en algunos casos juegos de acción. Esta metodología se denomina gamificación, que se define como la técnica de aprendizaje que traslada la mecánica de los juegos al ámbito educativo-profesional con el fin de conseguir mejores resultados. Con el advenimiento de las tecnologías de la información, los educadores tienen a su disposición un nuevo medio con el que llegar al alumnado. Esta revolución tecnológica está cambiando el modelo educativo actual.

En este trabajo de fin de grado, se explora la idea partiendo de cero en el desarrollo de RobotWars: un videojuego de batalla entre tanques donde el usuario podrá programar el comportamiento de cada uno de ellos mediante un lenguaje de programación sencillo, con el objetivo final de que los más jóvenes aprendan a programar. El jugador no interactúa directamente con los tanques durante el juego, que se controlan de forma automática según su implementación que constituye la inteligencia artificial del mismo. Esta temática nos da pie al objetivo principal de este trabajo, aprender los conocimientos básicos de programación de una forma sencilla y divertida siendo aplicable en actividades docentes.

Para lograr esto, se han abordado diversos problemas. En concreto, se ha priorizado el diseño e implementación de la gestión de colisiones del videojuego y movimientos básicos, el diseño de la gramática del lenguaje de programación y la implementación del analizador sintáctico. Por último, acoplar el compilador a la parte gráfica del videojuego.

Como conclusión, este proyecto ha logrado crear una tecnología educativa atractiva para los alumnos con la que los profesores pueden mejorar la enseñanza de la programación a edades más tempranas.

Índice

Capítulo 1

Introducción

La revolución tecnológica ha mejorado indudablemente la calidad de vida de nuestra sociedad y ha facilitado el desarrollo del saber humano. La educación es una disciplina que históricamente se ha beneficiado del uso de la tecnología. Este tipo de aprendizaje gana terreno en las metodologías de formación debido a su carácter lúdico, trasladar el modelo de juego en actividades docentes, esto se denomina gamificación.

Para explorar esta idea, se ha propuesto desarrollar un videojuego de programación en 2D denominado RobotWars, cuya inspiración viene de otro juego llamado Robocode [1] (Figura 1.1), algo más potente y complejo, ya que se programa en lenguajes como Java o .NET, y, sin embargo, en este juego se usará un lenguaje más sencillo para programar [2], debido a que uno de sus objetivos principales será utilizarlo en actividades docentes destinado a un público juvenil. Para ello se ha creado un lenguaje muy simple (del que se hablará más adelante en este documento) que hará el juego más fácil y dinámico para los usuarios.

El objetivo principal de este trabajo de fin de grado es crear un videojuego desde cero abordando todos los aspectos necesarios tanto en el enfoque técnico como en el artístico.

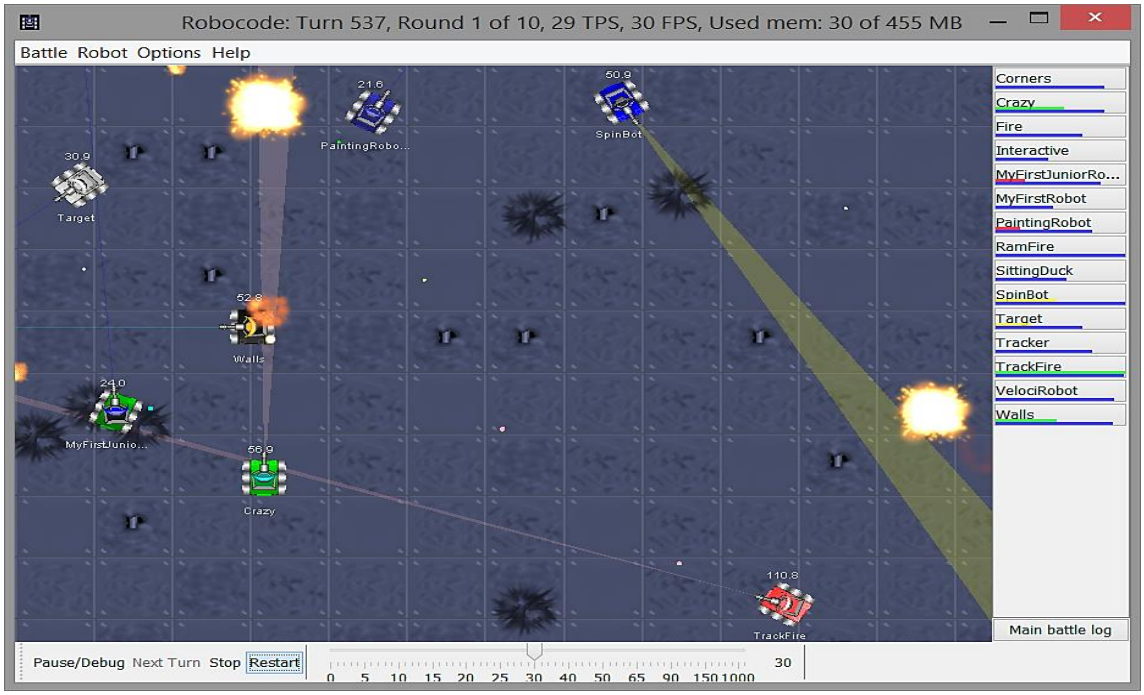


Figura 1.1: Robocode

La estructura de este documento constará de varios apartados bien diferenciados, empezando por el análisis del proyecto en el que se recogen los requisitos planteados, así como los casos de uso detallados con su flujo principal y alternativo.

A continuación, se realizará el diseño del sistema, donde se detalla la estructura del videojuego en paquetes, las partes de un tanque, las ventajas y desventajas del lenguaje de programación junto con su repertorio de instrucciones, pilar fundamental del videojuego ya que nos aporta la herramienta para conectar con el jugador y ofrecerle un aprendizaje atractivo. Una vez se ha diseñado el lenguaje, se tendrá que definir la gramática del mismo. Por otra parte, se necesita un entorno con el cual el jugador pueda programar sus propios tanques. Para ello, RobotWars, tiene su propio editor de texto para facilitar la labor de programación del usuario. Además, el juego ofrece diferentes tipos de partida, como batallas *All vs All*, donde todos los tanques luchan entre todos, o batallas *Team Deathmatch*, batallas por equipos donde la estrategia juega un papel importante.

Posteriormente, se implementará la aplicación en base al diseño explicado anteriormente, juntando dos partes principales de este trabajo. La parte gráfica y la lógica del juego, explicando detalladamente la gestión de colisiones y los movimientos de los objetos basados en transformaciones afines [3]. Por otro lado, se encuentra el compilador, parte fundamental del juego, ya que se encargará de comprobar si el programa es correcto y notificará al usuario mediante un mensaje si el programa del tanque está listo para usarse o existe algún error.

Obtenido el resultado final del sistema, se realizan las pruebas pertinentes para asegurar su correcto funcionamiento. Para ello, se mostrarán unas gráficas que describirán el grado de satisfacción de los usuarios encuestados en diferentes aspectos del videojuego.

Por último, se encuentran los apartados de conclusiones y bibliografía donde se enumeran las citas bibliográficas consultadas a lo largo del documento.

Un apartado a señalar es el apartado de Anexos donde se detallan las implementaciones de apartados anteriores a través de gráficas y anotaciones. Por ejemplo, mostrando paso por paso, las acciones del analizador sintáctico desde que el usuario pulsa el botón de compilar hasta que finaliza el proceso de compilación.

1.1. Objetivo y alcance

El objetivo de este proyecto es analizar, diseñar e implementar un videojuego educativo, dirigido a un sector de público juvenil, con un tutor con ciertos conocimientos de informática para guiarlos durante el juego. Este TFG está enfocado en integrar metodologías aprendidas durante el grado de Ingeniería Informática.

Lo más importante es que el videojuego ayude a los alumnos a aprender y afianzar conceptos de programación, pero sería muy beneficioso que además les resultase atractivo y sencillo de manejar. Tampoco es deseable que el videojuego sea muy complejo, algo que podría distraer a los alumnos del objetivo final que es el de aprender.

Gracias a que los juegos de acción son un entretenimiento muy popular, sus mecánicas de disparar, atacar y defender nos resultan a todos muy familiares. Es por ello por lo que el videojuego simulará un sencillo mapa en el que se enfrentarán diferentes tanques.

El desarrollo del videojuego se englobaría en dos grandes áreas. Por un lado, el apartado gráfico y lógico del juego, como pueda ser la gestión de colisiones y desarrollo de mecánicas que confieren al videojuego una sencillez gráfica, pero con una jugabilidad atractiva. Por otro lado, el desarrollo del compilador para la lectura e interpretación de los programas que serán implementados por el usuario.

1.2. Metodología y alcance

El trabajo se enmarca en un contexto técnico enfocado en la resolución de los objetivos previamente planteados. Para ello se propone seguir el ciclo de vida del software: definición de requisitos, análisis, diseño, implementación, pruebas, validación y mantenimiento.

La herramienta principal a utilizar será el IDE Eclipse[1]. Eclipse soporta librerías gráficas que permiten el desarrollo de videojuegos, en este caso, Swing [4]. Se ha elegido utilizar Java [5] como lenguaje de programación debido a los conocimientos adquiridos durante el grado por lo que no supone un extra de esfuerzo en la curva de aprendizaje. Hay que mencionar que Eclipse tiene ciertas limitaciones ya que no tiene una interfaz diseñada para el desarrollo de videojuegos, tampoco existe una tienda de recursos (todos los componentes de un juego hechos por artistas o diseñadores en vez de programadores) predefinidos, como sprites, animaciones o paquetes de sonido. Dicho esto, el desarrollo

[1] https://www.eclipse.org/

de RobotWars en este entorno confiere una complejidad extra sobre otros motores gráficos especializados e implica que todo está implementado desde cero (gráficos, lógica del juego, sprites…).

Para la construcción del compilador se ha utilizado JavaCC[2] , un generador de analizadores sintácticos para usar con aplicaciones Java. Se puede añadir como plug-in en Eclipse. Esta herramienta aporta la potencia necesaria para poder leer programas escritos en el lenguaje que nosotros deseemos y generar las acciones necesarias por cada token.

Además, se han usado otras herramientas como Paint3D[3] para la edición de sprites o herramientas online para conversión de formatos de archivo [6] o redimensionar imágenes [7].

Todas las herramientas mencionadas anteriormente son totalmente gratuitas, aunque sí que existen versiones de pago.

1.3. Motivación

La principal motivación para realizar el presente proyecto ha sido la creación de una aplicación que requiere utilizar técnicas muy diferentes de diversos campos combinadas en el mundo de la informática, tales como los gráficos, la ingeniería del software, los procesadores de lenguajes, etc. Por todo ello, este trabajo de fin de grado constituye una gran forma de poner en práctica la mayoría de los conocimientos que se han adquirido durante el grado.

Además, otro tema muy interesante es la gamificación[4], una técnica de aprendizaje que traslada la mecánica de los juegos al ámbito educativo-profesional con el fin de conseguir mejores resultados. Esto sumado a la posible dedicación en un futuro a la industria del videojuego ha motivado a escoger este tema como trabajo de fin de grado.

[2] https://marketplace.eclipse.org/category/free-tagging/javacc
[3] https://www.microsoft.com/es-ar/p/paint-3d/9nblggh5fv99#activetab=pivot:overviewtab
[4] https://www.educaciontrespuntocero.com/noticias/gamificacion-que-es-objetivos/

Capítulo 2

Análisis

Para realizar un análisis que posteriormente sirva para la toma de decisiones en el diseño del proyecto se han realizado distintos estudios. A continuación, se ha realizado una captura de posibles requisitos del sistema (Apartado 2.1).

2.1. Requisitos

A continuación, se presentan los requisitos funcionales y no funcionales que se plantearon al comienzo del trabajo (Tabla 2.1 y Tabla 2.2). En estas tablas, además de los propios requisitos, se detalla la prioridad que se dio inicialmente a cada requisito y si se ha cumplido o no al final del trabajo.

ID	Descripción	Prioridad	Cumplido
RF1	El videojuego permitirá iniciar una batalla	Alta	Sí
RF2	La aplicación permitirá abrir batallas guardadas	Baja	Sí
RF3	El videojuego permitirá guardar batallas	Baja	Sí
RF4	El sistema incluirá un editor de texto para programar	Alta	Sí
RF5	La aplicación permitirá cambiar opciones de configuración	Media	Sí
RF6	El videojuego incluirá un manual de ayuda	Baja	Sí
RF7	El videojuego permitirá añadir tanques al campo de batalla	Alta	Sí
RF8	El sistema permitirá eliminar tanques seleccionados	Alta	Sí
RF9	La aplicación incluirá un sistema predictivo para ayudar a la programación	Alta	Sí
RF10	La aplicación incluirá un compilador para interpretar los programas	Alta	Sí
RF11	El videojuego permitirá compilar ficheros en el lenguaje	Alta	Sí
RF12	El sistema permitirá funciones de editado del fichero (deshacer, rehacer, cortar, copiar, pegar, borrar y seleccionar todo)	Media	Sí

RF13	El videojuego permitirá crear un nuevo fichero	Alta	Sí
RF14	El videojuego permitirá abrir ficheros existentes	Alta	Sí
RF15	La aplicación permitirá guardar ficheros	Alta	Sí

Tabla 2.1: Requisitos Funcionales

ID	Descripción	Prioridad	Cumplido
RNF1	El videojuego podrá ejecutarse en sistemas operativos Windows.	Alta	Sí
RNF2	El sistema podrá ejecutarse en sistemas operativos Mac.	Baja	No
RNF3	El sistema podrá ejecutarse en sistemas operativos Linux	Baja	No
RNF4	Los programas serán guardados en ficheros externos	Alta	Sí
RNF5	El listado de tanques se cargará de un directorio externo	Alta	Sí
RNF6	El control del videojuego podrá ser tanto por teclado como programado	Media	No
RNF7	Los programas serán escritos en el lenguaje	Alta	Sí
RNF8	El lenguaje tendrá un conjunto de instrucciones básicas.	Alta	Sí
RNF9	El lenguaje permitirá instrucciones de control (if, while)	Alta	Sí

Tabla 2.2: Requisitos No Funcionales

2.2. Casos de uso

En la siguiente figura (Figura 2.3) se detallan los casos de uso del videojuego y, a continuación, la especificación de cada uno de ellos.

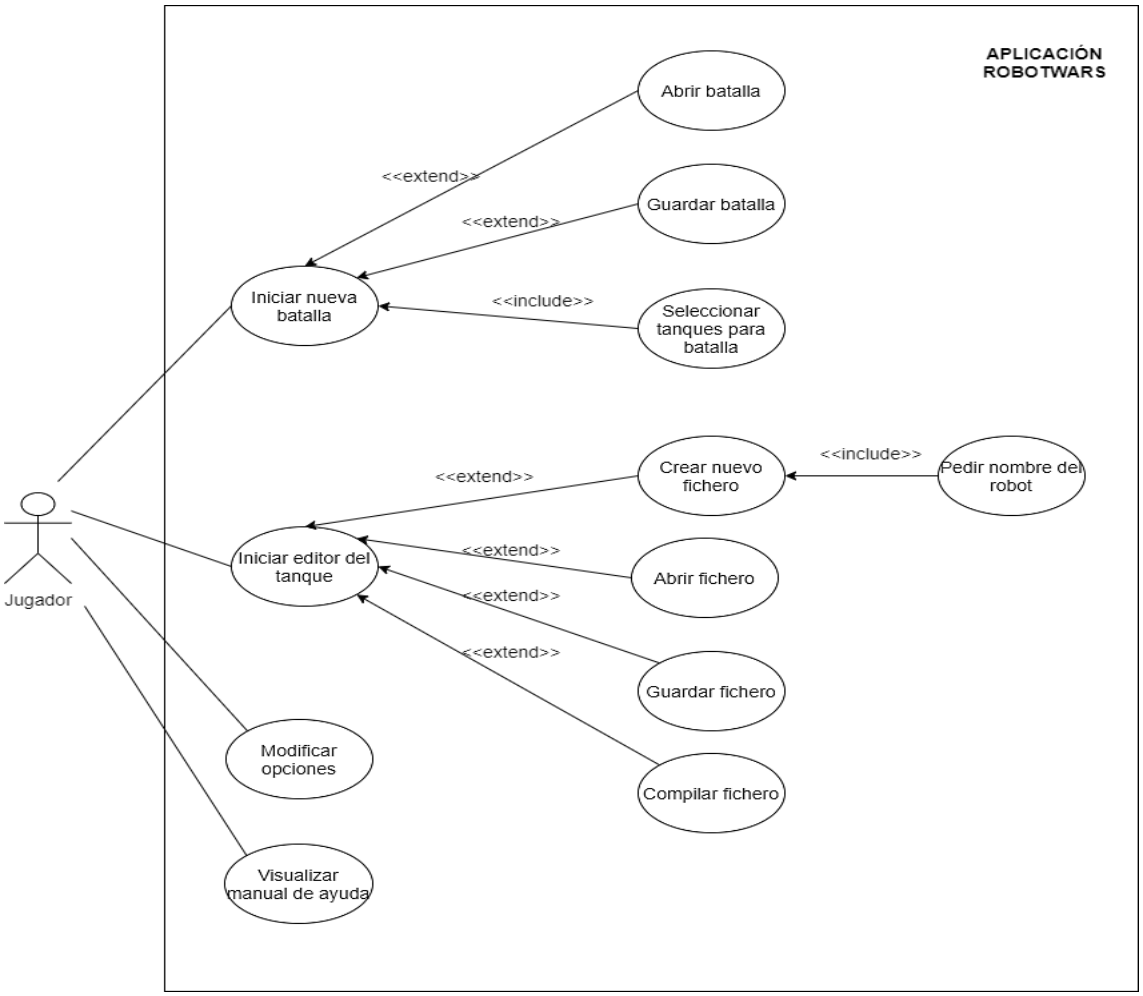


Figura 2.3: Diagrama casos de uso

Iniciar nueva batalla

Este caso de uso permite al jugador iniciar una nueva batalla donde deberá seleccionar los tanques que desee.

- Precondición: El jugador debe estar en el menú principal.
- Postcondición: El jugador accede a la pantalla de selección de tanques
- Flujo de eventos principal:
    1. El caso de uso comienza cuando el jugador selecciona la opción “New Battle” en el menú principal.
    2. La aplicación muestra la pantalla de selección de tanques y termina el caso de uso.

Seleccionar tanques para batalla

Este caso de uso permite al jugador seleccionar los tanques que se enfrentarán durante la partida.

- Precondición: El jugador debe estar en la pantalla de selección de tanques y debe haber al menos un tanque creado.
- Postcondición: El jugador empieza una nueva partida con los tanques seleccionados.
- Flujo de eventos principal:
    1. El caso de uso comienza cuando el jugador selecciona un tanque de la lista de tanques disponibles.
    2. Añade el tanque pulsando el botón "Add".
    3. Mientras se desee, el jugador puede seleccionar más tanques.
    4. El jugador pulsa el botón "Start Battle".
    5. La aplicación empieza una partida con los tanques seleccionados y termina el caso de uso.
- Flujo de eventos alternativos.
    2. El jugador añade todos los tanques disponibles pulsando el botón "Add All".
    3. El jugador elimina el tanque seleccionado pulsando el botón "Remove".
        3. El jugador elimina todos los tanques pulsando el botón "Remove All".
        4. El jugador cancela la partida pulsando el botón "Cancel" y la aplicación vuelve al menú principal.

Abrir batalla

Este caso de uso permite al jugador abrir una batalla ya existente y jugar a partir de su contenido.

- Precondición: El jugador debe estar en el menú principal.
- Postcondición: El jugador empezará una batalla a partir de los tanques guardados.
- Flujo de eventos principal:
    1. El caso de uso comienza cuando el jugador selecciona la opción "Open" en el menú principal.

2. La aplicación muestra un cuadro de diálogo para elegir una batalla de las existentes.
3. El jugador selecciona una batalla y pulsa el botón de “Abrir”.
4. La aplicación muestra la ventana de selección de tanques con los tanques guardados.
5. El jugador pulsa el botón de “Start”.
6. La aplicación empieza la batalla con los tanques seleccionados y termina el caso de uso.

- Flujo de eventos alternativo:

4.a. El jugador pulsa el botón de “Cancel” y vuelve al menú principal.

4.b. El jugador añade algún tanque o borra alguno seleccionado.

Guardar batalla

Este caso de uso permite al jugador guardar la batalla que está jugando actualmente.

- Precondición: El jugador debe estar en la pantalla de batallas.
- Postcondición: Se guardará la batalla y el jugador se quedará en la pantalla de batallas.
- Flujo de eventos principal:
  1. El caso de uso comienza cuando el jugador selecciona la opción “Save” en la pantalla de batallas.
  2. La aplicación muestra un cuadro de diálogo para elegir una batalla de las existentes o darle un nombre nuevo.
  3. El jugador selecciona una batalla y pulsa el botón de “Save”.
  4. La aplicación reanuda la batalla y termina el caso de uso.
- Flujo de eventos alternativos:

2. El jugador pulsa el botón de “Cancel” y vuelve a la pantalla de batallas.

Iniciar editor del tanque

Este caso de uso permite al jugador abrir el editor para empezar a programar el comportamiento del tanque.

- Precondición: El jugador debe estar en el menú principal.
- Postcondición: El jugador accede a la pantalla del editor de código.

- Flujo de eventos principal:
    1. El caso de uso comienza cuando el jugador selecciona la opción “Editor Tank” en el menú principal.
    2. La aplicación muestra la pantalla del editor de código y termina el caso de uso.

Crear nuevo fichero

Este caso de uso permite al jugador crear un fichero nuevo donde podrá implementar el comportamiento del tanque.

- Precondición: El jugador debe estar en la pantalla del editor de código.
- Postcondición: El jugador implementará con las instrucciones oportunas el comportamiento del tanque.
- Flujo de eventos principal:
    1. El caso de uso comienza cuando el jugador selecciona la opción “New” en la pantalla del editor de código.
    2. La aplicación muestra un cuadro de diálogo donde pedirá el nombre del tanque y termina el caso de uso.

Pedir nombre del robot

Este caso de uso permite al jugador introducir un nombre con el que será guardado el tanque y su fichero asociado.

- Precondición: El jugador debe pulsar la opción de crear un nuevo fichero.
- Postcondición: El jugador empezará a implementar el comportamiento del tanque.
- Flujo de eventos principal:
    1. El jugador introduce el nombre del tanque en el input del diálogo.
    2. El jugador pulsa el botón de “Accept”.
    3. La aplicación muestra una nueva hoja para escribir código y termina el caso de uso.
- Flujo de eventos alternativos:
    2. El jugador pulsa el botón de “Cancel” y vuelve a la pantalla del editor de código.

Abrir fichero

Este caso de uso permite al jugador abrir un fichero ya existente y programar a partir de su contenido.

- Precondición: El jugador debe estar en la pantalla del editor de código.
- Postcondición: El jugador empezará a implementar el comportamiento del tanque a partir del fichero existente.
- Flujo de eventos principal:

  1. El caso de uso comienza cuando el jugador selecciona la opción "Open" en la pantalla del editor de código.

  2. La aplicación muestra un cuadro de diálogo para elegir un fichero de los existentes.

  3. El jugador selecciona un fichero y pulsa el botón de "Open".

  4. La aplicación muestra una hoja con el código existente de ese fichero.

- Flujo de eventos alternativo:
  3. El jugador pulsa el botón de "Cancel" y vuelve a la pantalla del editor de código.

Guardar fichero

Este caso de uso permite al jugador guardar el fichero donde ha programado el comportamiento del tanque.

- Precondición: El jugador debe estar en la pantalla del editor de código.
- Postcondición: Se guardará el fichero y el jugador se quedará en la pantalla del editor de código.
- Flujo de eventos principal:
  1. El caso de uso comienza cuando el jugador selecciona la opción "Save" en la pantalla del editor de código.
  2. La aplicación muestra un cuadro de diálogo para elegir un fichero de los existentes o darle un nombre nuevo.
  3. El jugador selecciona un fichero y pulsa el botón de "Save".
  4. La aplicación vuelve a la hoja donde estaba programando el jugador.

- Flujo de eventos alternativos:
    3. El jugador pulsa el botón de “Cancel” y vuelve a la pantalla del editor de código.

Compilar fichero

Este caso de uso permite al jugador compilar el fichero que está programando para verificar si es correcto sintáctica y semánticamente.

- Precondición: El jugador debe crear un fichero o abrir uno existente y el fichero debe estar guardado.
- Postcondición: La aplicación mostrará un mensaje informando si la compilación ha sido correcta o existe algún fallo.
- Flujo de eventos principal:
    1. El caso de uso comienza cuando el jugador selecciona la opción “Compile” en la pantalla del editor de código.
    2. La aplicación muestra un mensaje informando si la compilación ha sido correcta o existe algún fallo.
    3. El jugador pulsa el botón de “Accept”.
    4. La aplicación vuelve a la hoja de programación y termina el caso de uso.
- Flujo de eventos alternativos:
    2. Si el fichero no ha sido guardado, la aplicación muestra un mensaje informando de ello y pregunta al jugador si desea guardar.
    3. Si el usuario pulsa “Yes”, el fichero se guarda y se vuelve al paso 2 del flujo principal.
    3. Si el usuario pulsa “No”, la aplicación muestra un mensaje de error informando que se debe guardar antes de compilar y vuelve a la hoja de programación sin realizar ninguna acción.
    3. Si el usuario pulsa “Cancel”, se vuelve a la hoja de programación sin realizar ninguna acción.

Modificar opciones

Este caso de uso permite al jugador cambiar distintas opciones del videojuego.

- Precondición: El jugador debe estar en el menú principal.
- Postcondición: La aplicación cambiará el parámetro oportuno.

- Flujo de eventos principal:
    1. El jugador selecciona la opción “Settings” en el menú principal.
    2. La aplicación muestra por pantalla las pestañas disponibles: “Visualization”, “Graphics”, “Audio”.
    3. El jugador elige una de las pestañas para las que quiere modificar algún parámetro.
    4. La aplicación muestra las opciones detalladas de la pestaña que ha elegido el jugador.
    5. El jugador modifica los parámetros que desee.
    6. La aplicación actualiza los parámetros modificados.

Visualizar manual de ayuda

Este caso de uso permite al jugador visualizar los requisitos del sistema y leer el repertorio de instrucciones con las que se puede programar.

- Precondición: El jugador debe estar en el menú principal.
- Postcondición: El jugador obtiene información de la aplicación.
- Flujo de eventos principal:
    1. El jugador selecciona la opción “Help” en el menú principal.
    2. La aplicación muestra una ventana con información de la aplicación, requisitos del sistema y el repertorio de instrucciones con el que aprender a programar.

Capítulo 3

Diseño

Se ha realizado el diseño del sistema acorde con los requisitos y objetivos del proyecto. Se ha optado por un sistema modular y escalable, intentando facilitar nuevas funcionalidades sin un coste en tiempo excesivo.

3.1. Arquitectura del sistema

En esta sección se detalla la estructura del videojuego (Figura 3.1), incluyendo todas las partes en la que está dividido el código.

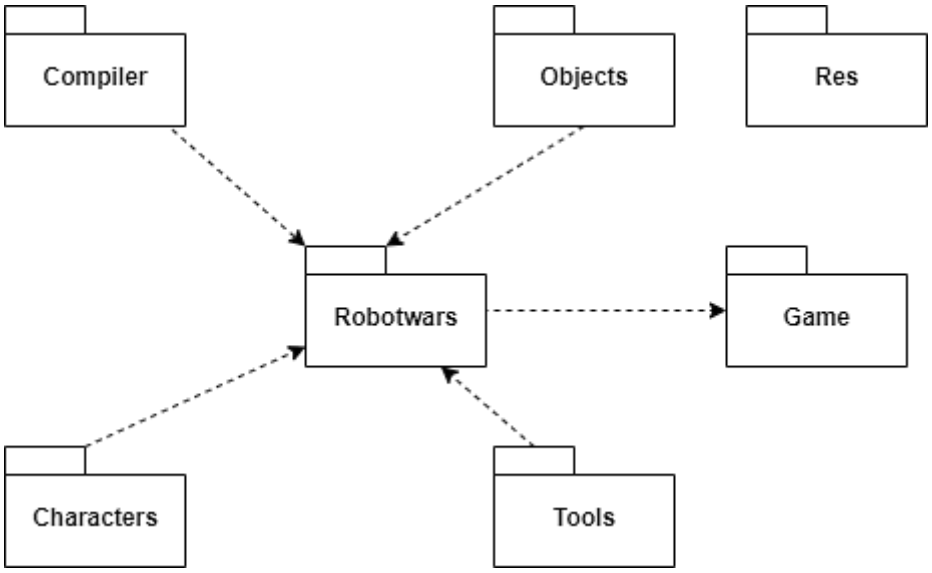


Figura 3.1: Diagrama de paquetes

Paquete "Characters"

- Character: Representa los campos comunes de todos los personajes: posición en el mapa, vida del personaje, boundingBox, un identificador y un nombre.
- Tank: Clase principal de un tanque. Controla el inicio y la ejecución de todos los componentes de un tanque.

Paquete “Compiler”

Este paquete es propio de JavaCC por lo que auto genera ciertas clases propias para el correcto funcionamiento del compilador.

- MyNewGrammar: Gestiona la compilación de los ficheros implementados y la correcta ejecución de las instrucciones cuando se inicia una batalla.

Paquete “Game”

- Estado: Representa los estados por los que pasa el juego. Se almacena en una estructura de tipo enumeración.
- MaquinaEstados: Gestiona los estados del juego. Se va actualizando cada vez que se cambia de ventana.

Paquete “Objects”

- Box: Gestiona toda la estructura de los obstáculos.
- Bullet: Gestiona la información de las balas de los tanques. Gestión de colisiones.
- GameObject: Representa los campos comunes de todos los objetos: posición en el mapa, bounding box, si es rompible o no.
- Mine: Gestiona la información de las minas que colocan los tanques. Gestión de colisiones.

Paquete “Res”

Un recurso son datos (sprites, audio, texto, etc.) a los que un programa necesita acceder de una manera que es independiente de la ubicación del código del programa. Este paquete se encarga de gestionar estos recursos a los que el juego necesitará acceder a lo largo de su ejecución.

Paquete “Robotwars”

- Animation: Pinta las animaciones de las explosiones.
- CharLimitDocument: Componente para limitar el número de caracteres en la descripción de los tanques.
- CheckListItem: Encapsula un objeto con dos campos que será renderizado.
- CheckListRenderer: Gestiona el aspecto gráfico de una lista de checkbox.

- Configuration: Interfaz que almacena campos genéricos para la gestión de los movimientos de los tanques.
- Controller: Gestiona el movimiento de las balas en cada tick y las explosiones.
- GameMain: Inicializa la aplicación
- RobotwarsGame: Clase principal, gestiona y unifica todas las partes del juego, inicialización del campo de batalla, menús, gestión de ficheros, etc…
- Sound: Gestiona la reproducción de efectos FX y música del juego.

Paquete "Tools"

- ImageTool: Carga los sprites que hay en el paquete de Resources.
- PanelSprite: Se encarga de pintar y refrescar cada una de las ventanas del juego.
- PlayThread: Thread que se encarga de reproducir todos los sonidos del juego.
- TextLineNumber: Componente que mostrará los números de línea del editor.

En el diagrama de estados que se muestra a continuación (Figura 3.2), puede observarse el flujo al iniciar una nueva batalla.

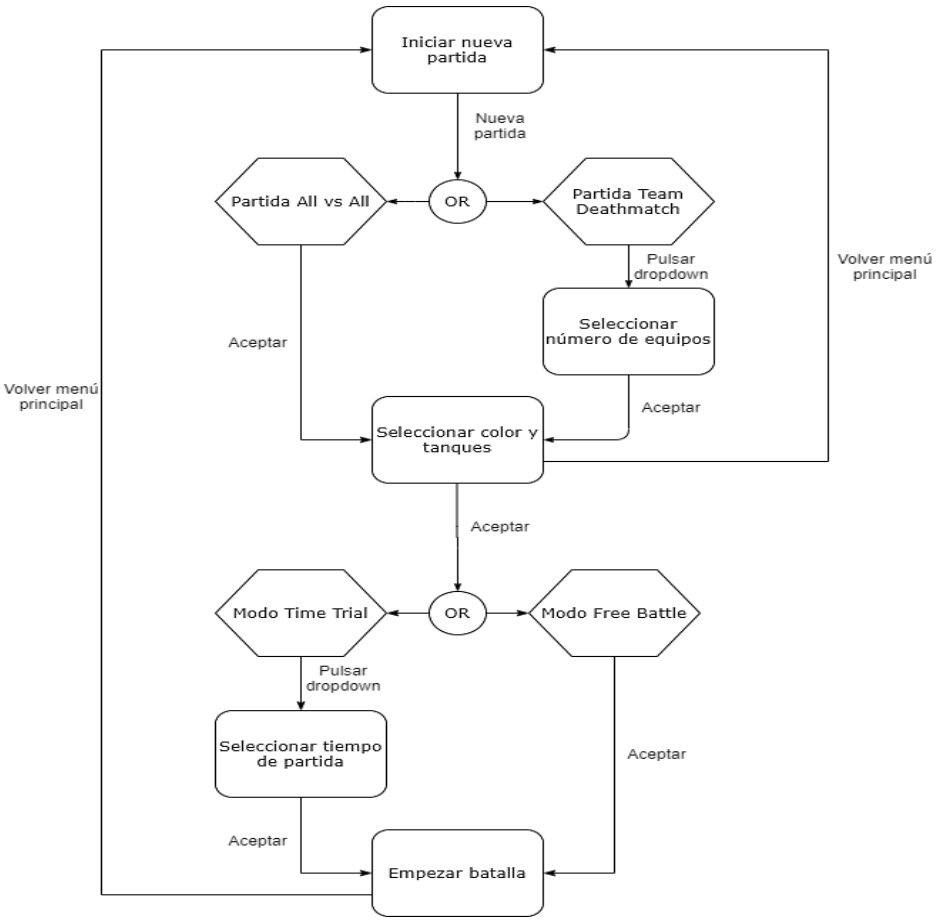


Figura 3.2: Diagrama de estados al iniciar batalla

3.2. Partes de un tanque

Los tanques se componen de tres partes: el cuerpo, el cañón y el radar (Figura 3.3).

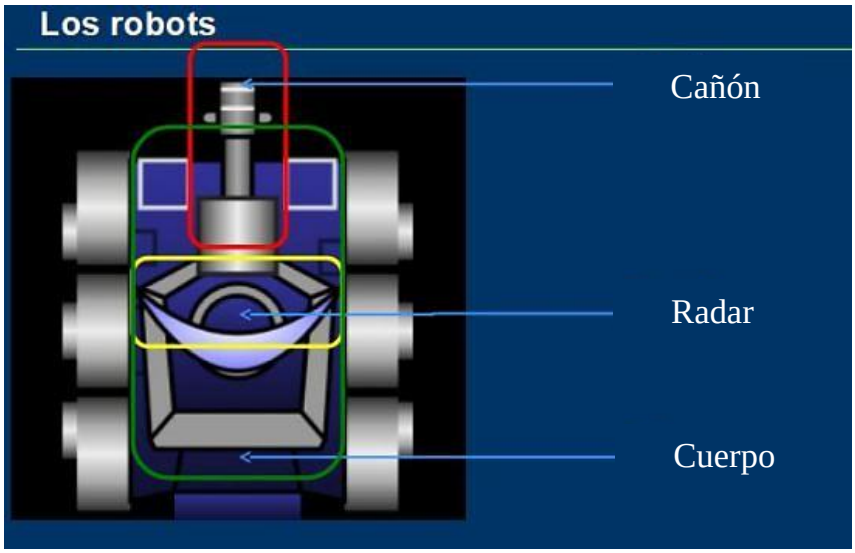


Figura 3.3: Partes de un tanque

El cañón rota sobre el eje central del tanque y dicha rotación es controlada desde programación. A diferencia del cañón, el radar no es direccional, pero da información de todo lo que hay a su alrededor en un radio concreto.

Tanto el radar como el cuerpo tienen instrucciones asociadas que pueden dar información del campo de batalla. El cañón y el cuerpo del tanque pueden girar de manera independiente, por tanto, hay instrucciones que nos dan información de cada uno de ellos, por ejemplo, los grados a los que se encuentran.

Con “freeInFront” podemos saber si en frente del tanque existe un obstáculo. Es importante este dato para realizar otras instrucciones, como girar a la derecha o a la izquierda y no estar avanzando en vano. Existen instrucciones que aportan la información contraria, por ejemplo, si el frente está ocupado o si la parte de atrás del tanque está libre.

Con “scanTank” podemos saber si existe un tanque a un radio establecido.

Hay más instrucciones disponibles para consultar tu estado y, por supuesto, para disparar que se detallarán más adelante.

3.3. Lenguaje de programación

El pilar fundamental del juego se basa en el lenguaje de programación creado para este trabajo. El lenguaje es sencillo por diversos motivos:

- No hay declaración de variables
- No hay declaración de funciones
- No hay tipos de datos
- No existen estructuras de datos, como arrays o matrices
- No permite la creación de objetos o clases
- No permite estructuras de control como for, switch o do while

Este trabajo de fin de grado se enfoca en el aprendizaje de conceptos básicos de programación sin olvidar el entretenimiento, por tanto, se decidió que los puntos mencionados anteriormente aumentaban la complejidad del lenguaje sin aportar un excesivo conocimiento a conceptos básicos que se pueden aprender con instrucciones más sencillas. Además, se decidió que las instrucciones estuvieran en inglés, ya que el objetivo principal de este trabajo es el de aprender, así el usuario se beneficia en ambos ámbitos, obtener conocimientos de programación y de inglés.

Sin embargo, el lenguaje es potente para el contexto específico en el que se usa, está orientado a programar tanques y sus mecánicas asociadas. Algunas instrucciones son propias de lenguajes más convencionales, como las que se enumeran a continuación:

- Permite estructuras de control como if o while
- Tiene constantes booleanas como true o false.
- Tiene comparadores aritméticos, como mayor (>), menor (<), menor o igual (<=), mayor o igual (>=), igual (=) o distinto (<>)
- Permite la parametrización de algunas instrucciones

Esta es la base de la que se sustenta, pero tiene muchas más instrucciones orientadas a la programación de tanques. Un aspecto muy importante en el juego es la orientación del tanque en el eje de coordenadas del mapa, muchas de las instrucciones se basan en ello o devuelven como resultado información relacionada con los grados del tanque.

En la siguiente figura (Figura 3.4) se ilustra el eje de coordenadas asociado a cada tanque.

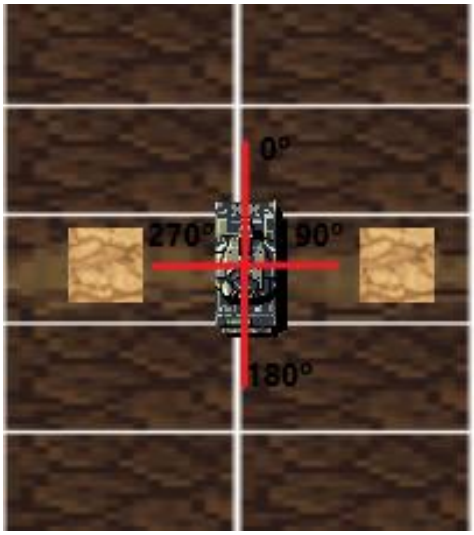


Figura 3.4. Eje de coordenadas de un tanque

Con la ilustración anterior se puede entender el funcionamiento de todas las instrucciones que emplean los grados como información en el campo de batalla. El repertorio de instrucciones del juego se enumera a continuación:

- Instrucción ***begin***: indica el comienzo del programa. Hay que escribirlo siempre para que el compilador detecte el comienzo.
- Instrucción ***end***: indica el final del programa. Hay que escribirlo siempre para que el compilador detecte el final.
- Instrucción ***advance(meters):*** el tanque avanza los metros indicados como parámetro. El parámetro es opcional. Si no se especifica, avanzará un metro.
- Instrucción ***back(meters):*** el tanque retrocede los metros indicados como parámetro. El parámetro es opcional. Si no se especifica, retrocederá un metro.
- Instrucción ***turnLeft(degrees):*** el tanque gira a la izquierda los grados indicados como parámetro. El parámetro es opcional. Si no se especifica, girará quince grados.
- Instrucción ***turnRight(degrees):*** el tanque gira a la derecha los grados indicados como parámetro. El parámetro es opcional. Si no se especifica, girará quince grados.

- Instrucción ***turnCanonLeft(degrees):*** el cañón gira a la izquierda los grados indicados como parámetro. El parámetro es opcional. Si no se especifica, girará quince grados.
- Instrucción ***turnCanonRight(degrees):*** el cañón gira a la derecha los grados indicados como parámetro. El parámetro es opcional. Si no se especifica, girará quince grados.
- Instrucción ***fire():*** el tanque dispara el cañón.
- Instrucción ***dropMine():*** el tanque suelta una mina en la casilla donde se encuentre el tanque. Esta mina puede explotar cuando un tanque pase por encima (cualquier tanque, incluido el que ha puesto la mina) o cuando sea disparada por otro tanque.
- Instrucción ***scanMine():*** devuelve los grados a los que se encuentra la mina más cercana respecto a la posición y orientación del tanque.
- Instrucción ***scanTank():*** devuelve los grados a los que se encuentra el tanque enemigo más cercano respecto a la posición y orientación del tanque. Esta instrucción es muy relevante ya que, en función del tipo de partida seleccionada, devolverá cualquier tanque cercano (All vs All) o devolverá el tanque si es del equipo contrario (Team Deathmatch).
- Instrucción ***freeInFront():*** devuelve cierto si no hay ningún obstáculo en frente.
- Instrucción ***busyInFront():*** devuelve cierto si hay algún obstáculo en frente.
- Instrucción ***freeBack():*** devuelve cierto si no hay ningún obstáculo detrás.
- Instrucción ***busyBack():*** devuelve cierto si hay algún obstáculo detrás.
- Instrucción ***random(min, max):*** devuelve un número aleatorio entre el intervalo pasado como parámetro. Esto nos proporciona realizar comportamientos aleatorios, con los que un usuario rival, aunque quiera ver el código de tanques enemigos, le resultará impredecible.
- Instrucción ***hasHit():*** devuelve cierto si en la instrucción anterior, el tanque colisionó con algo. Esta instrucción es útil, para escaparse de esquinas y no quedarse atrapado.
- Instrucción ***notHasHit():*** devuelve cierto si en la instrucción anterior, el tanque no colisionó con nada.
- Instrucción ***angleCanon():*** devuelve el ángulo del cañón (de 0 a 360 grados).

- Instrucción ***angleTank():*** devuelve el ángulo del tanque (de 0 a 360 grados).
- Instrucción ***true:*** devuelve cierto siempre. Sirve para hacer bucles infinitos.
- Instrucción ***false:*** devuelve falso siempre. Nunca hará la condición o el bucle.

El lenguaje también contiene estructuras de control, es decir, condiciones y bucles. Las condiciones tienen la siguiente estructura:

```
"if" condition "do"
        instruction
"endIf"
["else"
        instruction]
```

Los bucles tienen la siguiente estructura:

```
"while" condition "do"
        instruction
"endWhile"
```

Todo lo explicado en este punto está sintetizado en un apartado del manual de usuario, para que el jugador pueda familiarizarse con el lenguaje lo antes posible y sacarle el máximo partido.

Como se puede apreciar, la mayoría de estas instrucciones están orientadas al campo de batalla y al movimiento de los tanques, de ahí que el lenguaje sea de propósito específico, únicamente destinado a realizar batallas entre tanques de la manera más divertida posible.

3.4. Gramática del lenguaje

Al diseñar el analizador sintáctico [8] hay que pensar la gramática del lenguaje. Es la parte principal en el diseño del compilador [9], ya que permitirá reconocer el programa. Transcribirla a JavaCC es muy fácil teniendo la gramática definida (Figura 3.5).

En nuestro caso el lenguaje es sencillo, se compone de una lista de sentencias, siempre empezando con la instrucción “begin” y terminando con la instrucción “end”. Nuestras sentencias pueden ser una o más instrucciones, condiciones o bucles. En las instrucciones de control se pueden usar funciones booleanas (muchas instrucciones solo devuelven cierto o falso) o comparaciones, ya que hay instrucciones más complejas que devuelven un resultado numérico y se pueden comparar con otro número. Con esta jerarquía se puede formar un lenguaje básico pero lo suficientemente potente como para aprender nociones

básicas de programación y divertirse. En siguientes apartados, se explicará su implementación (Apartado 4.2).

```
Programa ::= (<INIT>)? bloque_sentencias

bloque_sentencias ::= <PRINCIPIO> lista_sentencias <FIN>

lista_sentencias ::= (sentencia)+

sentencia ::= invocacion_accion
| seleccion
| mientras_que

invocacion_accion ::= <AVANZA> <ABRIRPAREN> (factor)? <CERRARPAREN>
| <RETROCEDE> <ABRIRPAREN> (factor)? <CERRARPAREN>
| <GIRAIZQ> <ABRIRPAREN> (factor)? <CERRARPAREN>
| <GIRADER> <ABRIRPAREN> (factor)? <CERRARPAREN>
| <DISPARAR> <ABRIRPAREN> <CERRARPAREN>
| <SOLTARMINA> <ABRIRPAREN> <CERRARPAREN>
| <GIRACANONIZQ> <ABRIRPAREN> (factor)? <CERRARPAREN>
| <GIRACANONDER> <ABRIRPAREN> (factor)? <CERRARPAREN>

seleccion ::= <IF> expresion <DO> lista_sentencias
(<ELSE> lista_sentencias)? <STOPIF>

mientras_que ::= <WHILE> expresion <DO> lista_sentencias <STOPWHILE>

expresion ::= <FRENTELIBRE> <ABRIRPAREN> <CERRARPAREN>
| <FRENTEOCUPADO> <ABRIRPAREN> <CERRARPAREN>
| <ATRASLIBRE> <ABRIRPAREN> <CERRARPAREN>
| <ATRASOCUPADO> <ABRIRPAREN> <CERRARPAREN>
| <DETECTMINA> <ABRIRPAREN> <CERRARPAREN> operadorRelacional factorCondicion
| <SCANRADAR> <ABRIRPAREN> <CERRARPAREN> operadorRelacional factorCondicion
| <RANDOM> <ABRIRPAREN> factor <COMMA> factor <CERRARPAREN> operadorRelacional factorCondicion
| <CHOQUE> <ABRIRPAREN> <CERRARPAREN>
| <NOCHOQUE> <ABRIRPAREN> <CERRARPAREN>
| <ANGULOCANON> <ABRIRPAREN> <CERRARPAREN> operadorRelacional factorCondicion
| <ANGULOTANK> <ABRIRPAREN> <CERRARPAREN> operadorRelacional factorCondicion
| <VERDAD>
| <FALSO>

operadorRelacional ::= <MAYOR>
| <MENOR>
| <IGUAL>
| <MAI>
| <MEI>
| <NI>

factorCondicion ::= (minus)? <CONSTENTERA>

factor ::= (minus)? <CONSTENTERA>

minus ::= <MINUS>
```

Figura 3.5: Gramática del lenguaje

3.5. Programación de tanques

Para programar los robots no se necesitará ningún IDE externo, todo lo necesario está en el mismo RobotWars (Figura 3.6).

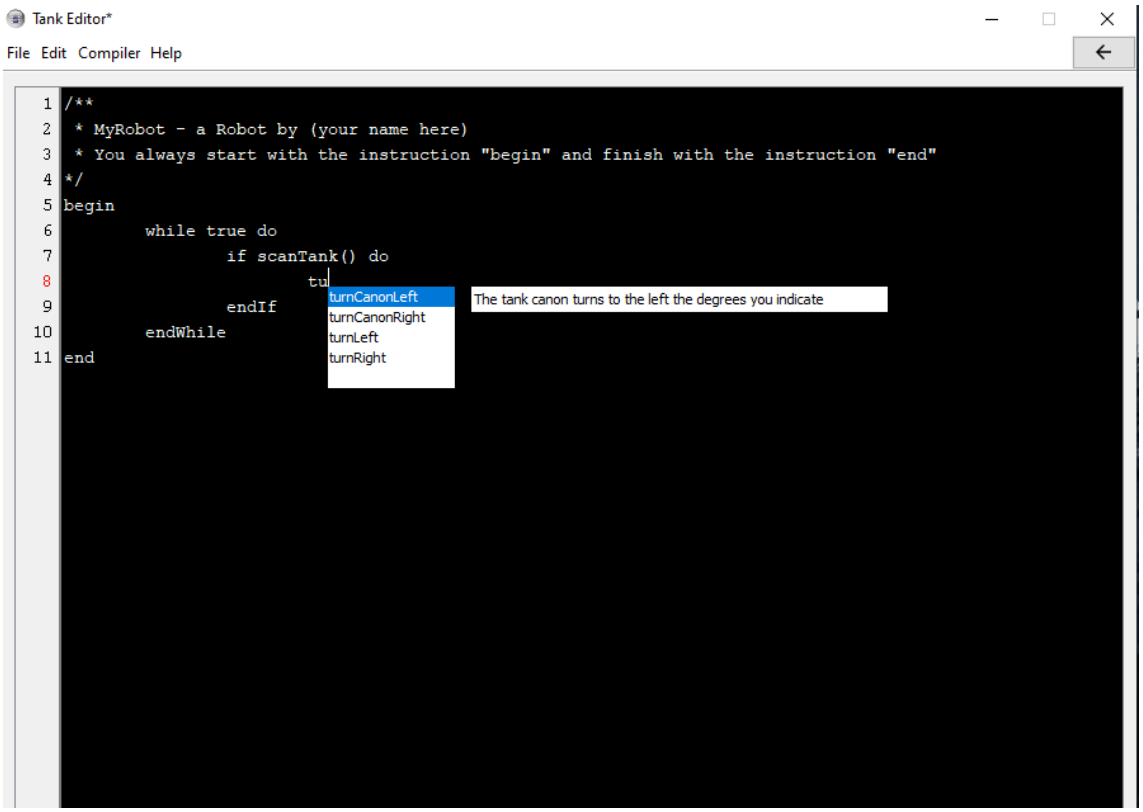


Figura 3.6: Editor de código

El editor es sensible al contexto, es decir, sugiere posibles instrucciones mientras el jugador está escribiendo, con esto conseguimos facilitar la labor de programación. Además, existe el menú *Help*, donde al clickar se abrirá una ventana nueva e independiente al juego con todo el repertorio de instrucciones que el lenguaje permite y sus explicaciones, con lo cual el jugador puede programar a la vez que mira el manual de ayuda.

El editor tiene todas las funciones principales de un entorno profesional, como crear un nuevo programa, guardar programa y abrir un programa existente [10]. Por otra parte, también ofrece funciones de edición (deshacer, rehacer, cortar, copiar, pegar y seleccionar todo), todas ellas con atajos de teclado[5] para mejorar la usabilidad del editor y conferir un aspecto más profesional.

A continuación, se mostrará un tanque (*colocaMinas*) de ejemplo que ofrece el juego para empezar batallas si un jugador lo desea (Figura 3.7). Es uno de los tanques más sencillos para comentar su código y explicar a nuevos usuarios las instrucciones y

[5] https://docs.oracle.com/javase/tutorial/uiswing/misc/keybinding.html

funcionalidades que el editor permite antes de empezar a programar por primera vez un tanque propio.

```
begin
        while true do
                /**
                 * Patrol loop. Advance until it collides with an object
                */
                while notHasHit() do
                        advance()
                        /**
                         * Drop a mine randomly
                        */
                        if random(1,100) >= 99 do
                                dropMine()
                        endIf
                endWhile
                turnRight(90)
                /**
                 * Loop to get out of any corner
                */
                while hasHit() do
                        if freeInFront() do
                                advance(5)
                        else
                                back(10)
                        endIf
                        turnRight(90)
                endWhile
        endWhile
end
```

Figura 3.7: Ejemplo tanque ColocaMinas

La estrategia de este tanque es la siguiente: avanza por el mapa hasta que colisiona con algún objeto (digamos que patrulla por el mapa), mientras está patrullando va colocando minas de manera aleatoria, en este caso, con una probabilidad muy baja (instrucción random). Cuando choca con algún objeto, gira hacia la derecha siempre que pueda, sino retrocede y vuelve a intentar girar hasta que el movimiento lo permita (el bucle con la instrucción hasHit facilita esta labor). Vuelve a empezar con el mismo comportamiento de manera indefinida.

Cuando el jugador haya terminado su programa, deberá pulsar el botón “Compile” y se mostrará un mensaje indicando si el programa es correcto o por el contrario tiene algún fallo. Todos los tanques que se hayan programado tienen que estar compilados correctamente para mostrarse en la ventana de selección de tanques, de lo contrario se guardará únicamente el programa, pero no podrá utilizarse en el campo de batalla. En apartados posteriores, se detallará su funcionamiento (Anexo B.3).

3.6. La forma de combate

3.6.1. Tipos de partida

Al iniciar una batalla en RobotWars, el juego sugiere dos tipos de partida. Por un lado, el tipo *All vs All*, donde todos los tanques tienen que pelear entre ellos, todos son enemigos. Se ofrece una partida más caótica sin pérdida de diversión.

Por otro lado, el tipo *Team Deathmatch*, partida por equipos donde el usuario elegirá en primera instancia el número de equipos que desea poner a pelear. Para cada equipo se seleccionará un color (a modo de distinción) y los integrantes que lo componen. Este tipo de partidas, le ofrecen una mayor jugabilidad a la aplicación, ya que confiere un punto más estratégico, donde los tanques pueden atacar de forma coordinada y puede ser más atractivo para el jugador.

La lógica del juego variará en función del tipo de partida elegida. Se explicará en el siguiente apartado. En las partidas *All vs All*, el ganador será un tanque mientras que en las partidas *Team Deathmatch*, el ganador será un equipo. Ambos modos, se pueden seleccionar desde la ventana de tipos de juego (Figura 3.8).

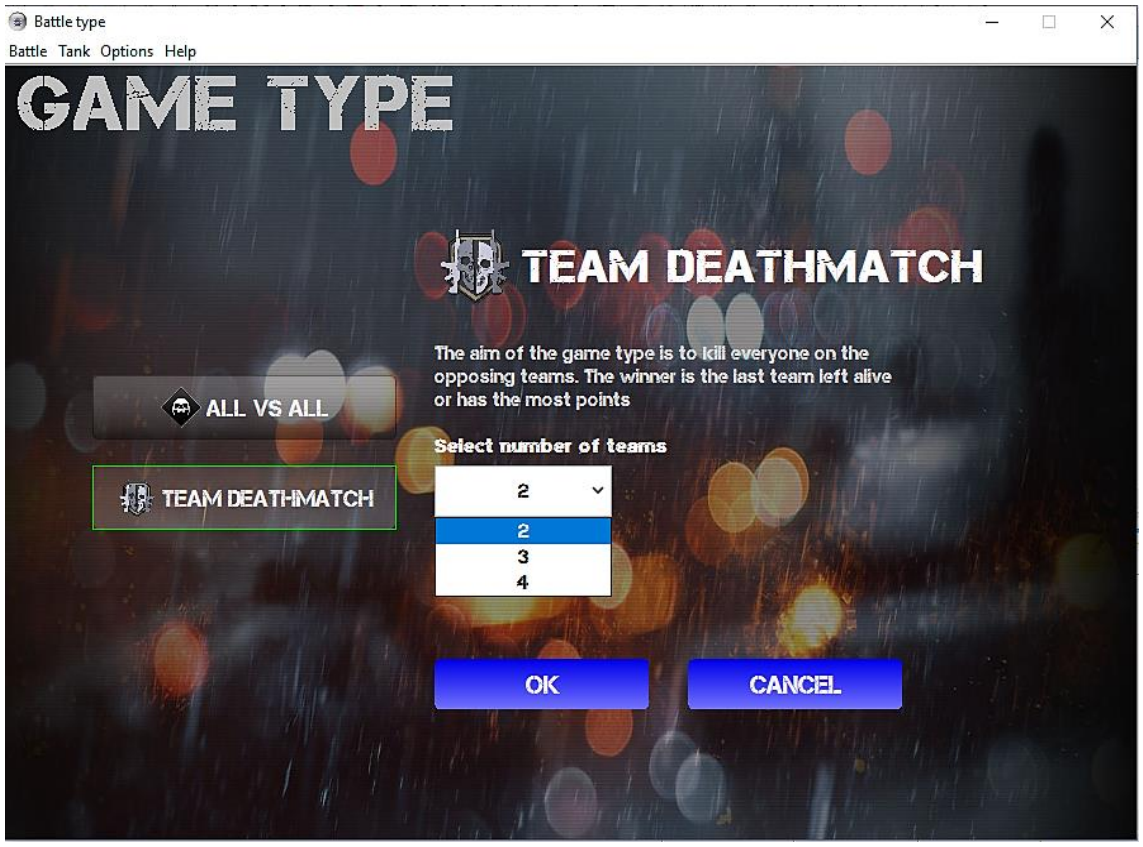


Figura 3.8: Tipos de partida

3.6.2. Modos de juego

RobotWars proporciona dos modos de juego. Por un lado, el modo *Time Trial* donde el usuario elegirá el tiempo de la partida. Su finalidad es simple, una vez el cronómetro llegue a cero, gana el tanque con más vida o el equipo con más integrantes dependiendo el tipo de partida que se haya seleccionado previamente. El mapa del juego tiene varios pasillos, que se pueden dividir como filas y columnas, al fin y al cabo, es un mapa rectangular. En este modo de juego, el mapa dispone de una fila menos que el otro modo para crear batallas más intensas. Por otro lado, el modo *Free Battle*, batalla sin límite de tiempo donde gana el último tanque que quede en pie o el equipo que elimine al resto de tanques enemigos dependiendo el tipo de partida que se haya seleccionado previamente. Ambos modos, se pueden seleccionar desde la ventana de modos de juego (Figura 3.9).

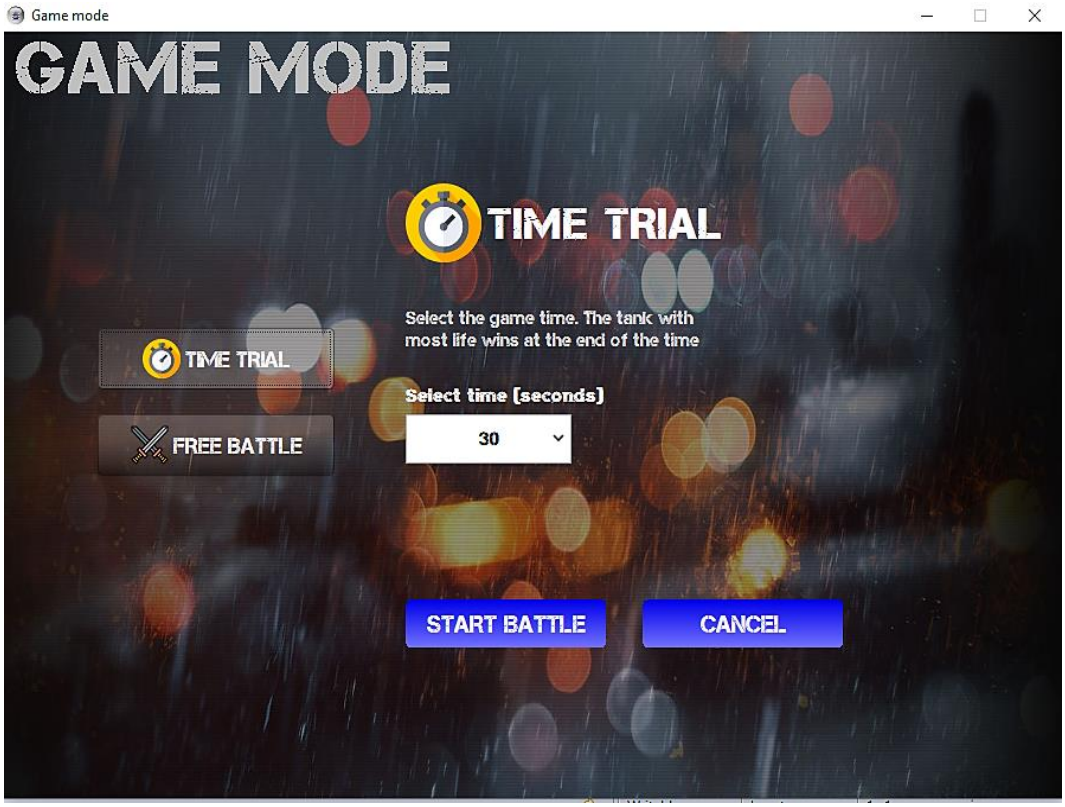


Figura 3.9: Modos de juego

Cada tanque aparecerá de manera aleatoria por el mapa, siempre y cuando la posición no sea la misma para ambos, que se realizará un ajuste para mover uno de los tanques a otra posición. Los tanques tienen 100 puntos de vida y no pueden disparar de manera muy continuada, tienen que pasar 2 segundos entre cada uno de los disparos, así evitar que los tanques se conviertan en metralletas. También son capaces de soltar minas. Éstas explotan al pasar un tanque por encima o al ser disparada por otro tanque.

Capítulo 4

Implementación

Una vez finalizada la fase de diseño (apartado 3) se procede a realizar la implementación de los componentes que van a ser utilizados en la solución final de la aplicación. La fase de implementación es la más extensa durante el ciclo de vida del sistema. En ella, se implementa el código para el proyecto de trabajo que está en relación con las especificaciones de las etapas anteriores. Durante la fase de implementación se han realizado cambios considerando que aportan mayor eficiencia y eficacia al sistema que se detallarán en apartados posteriores (Anexo B).

La implementación de la aplicación se confecciona en varias iteraciones. Durante cada iteración se realizan las pruebas pertinentes para asegurar el correcto funcionamiento, asegurando así la calidad de la solución final.

4.1. Gestión de colisiones

La gestión de colisiones es una de las partes principales al comienzo de la aplicación. En este tipo de juegos, los objetos, en este caso, los tanques, están en constante movimiento entre ellos, además de los propios obstáculos que hay repartidos por el mapa, por lo que los movimientos y los cálculos de las posiciones tienen que ser muy precisos.

Para la gestión de colisiones se decidió crear una figura geométrica llamada Bounding Box[6]. Esta figura está formada por cuatro puntos en el plano, es decir, es el rectángulo mínimo que engloba al objeto (Figura 4.1).

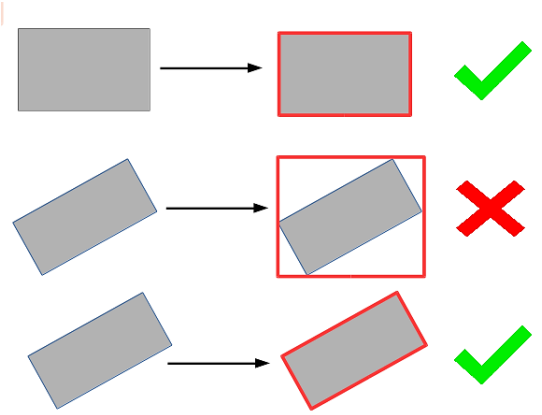


Figura 4.1: Bounding Box

[6] https://en.wikipedia.org/wiki/Minimum_bounding_box

Al avanzar el tanque, hay que calcular su nueva posición en el mapa, que no se llevará a cabo hasta comprobar que en esa nueva posición no hay nada que se interponga.

Uno de los problemas iniciales que se detectaron fue que el recalculo de esta delimitación era incorrecto, ya que se calculaba a partir de la bounding box anterior. Esto conllevaba que el rectángulo fuera cada vez mayor por lo que las colisiones ocurrían sin contacto aparente. La solución consistió simplemente en crear una bounding box nueva, tras cada movimiento del objeto, con los puntos nuevos recalculados.

Dado que los tanques se pueden mover y rotar con una granularidad indeterminada (se necesitarían muchos sprites del tanque que no disponía), se decidió por usar una transformación afín para obtener el movimiento del sprite. En geometría, una transformación afín [3] consiste en una transformación lineal seguida de una traslación (Figura 4.2)[7].

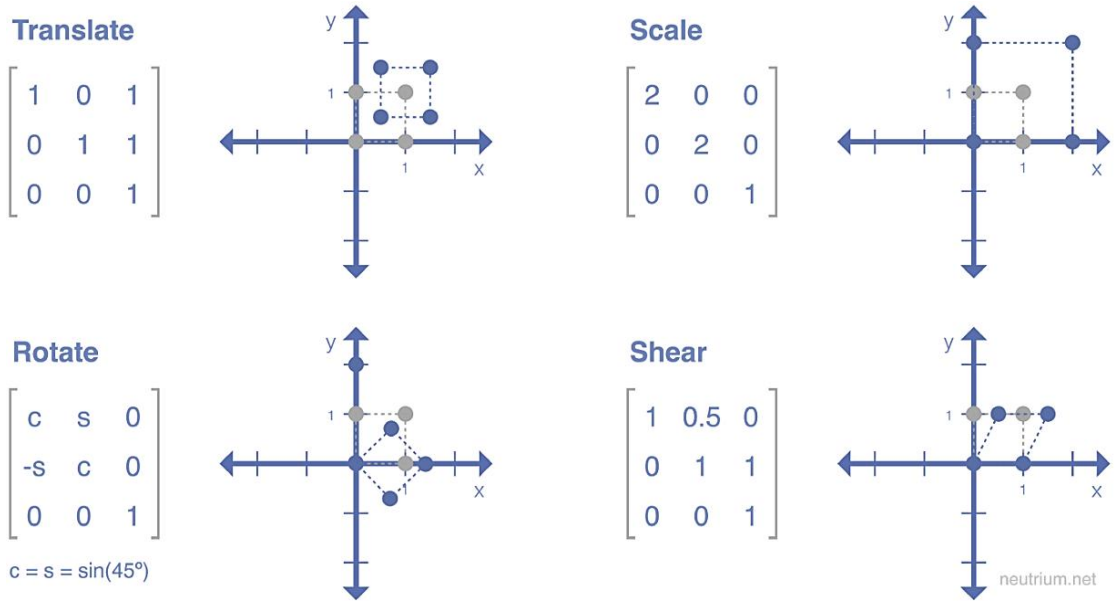


Figura 4.2: Posibles transformaciones

Las balas también requieren una transformación afín [3] cuando son disparadas, pero en su caso, solo se necesita, movimiento de traslación. Al igual que las explosiones y las minas. Tanto las balas como las minas se crean con la posición del cañón, aunque hay que realizar algún pequeño ajuste para que se vean correctamente.

[7] https://medium.com/mlait/affine-transformation-image-processing-in-tensorflow-part-1-df96256928a

4.2. Compilador

El compilador es la parte más importante en la programación del juego. JavaCC es la herramienta que permite leer una gramática y convertirla a un programa Java que pueda reconocer esa gramática [11]. Asimismo, provee otras funciones relacionadas con el analizador sintáctico como la construcción de un árbol, acciones asociadas y la depuración.

El proceso de compilación se divide en varias fases (Figura 4.3): analizador léxico, analizador sintáctico, analizador semántico, generación de código y optimización de código [9]. En este trabajo, se han omitido las últimas tres partes (semántico, generación de código y optimización), debido a la sencillez del lenguaje, ya que no requiere declaración de variables.

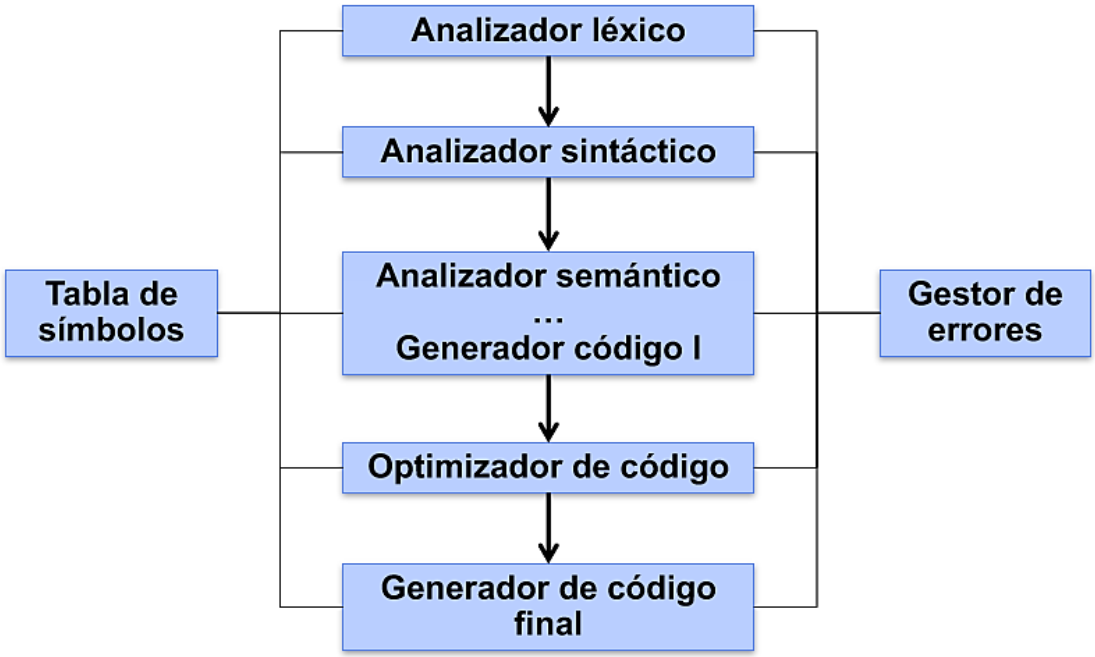


Figura 4.3: Fases del compilador

El analizador léxico es la primera fase del compilador que recibe como entrada el programa escrito por el usuario y produce una salida compuesta de tokens. Estos tokens serán necesarios para la siguiente etapa. En cualquier lenguaje de programación es necesario establecer patrones para caracteres especiales (como el espacio en blanco, los comentarios o los retornos de carro) que la gramática pueda reconocer sin que constituya un token en sí y que ignorará (Figura 4.4).

```
SKIP :
{
  " "
| "\r"
| "\t"
| "\n"
| < "//" (~["\r", "\n"])* >
| < "/*" > : COMMENT
| < "/**" > : COMMENT
}

< COMMENT > SKIP:
{
  < "*/" >: DEFAULT
| < ~[] >
}
```

Figura 4.4: Tokens para ignorar

Después hay que definir todos los tokens que componen el lenguaje (Figura 4.5 y Figura 4.6), es decir, todas las instrucciones y palabras reservadas, incluidos los puntos y coma y los paréntesis.

```
TOKEN : /* KEYWORDS */
{
  < INIT: "MyRobot - a Robot by (your name here)" >
| < AVANZA : "advance" >
| < RETROCEDE : "back" >
| < GIRAIZQ : "turnLeft" >
| < GIRADER : "turnRight" >
| < DISPARAR: "fire" >
| < VERDAD: "true" >
| < FALSO: "false" >
| < SOLTARMINA: "dropMine" >
| < DETECTMINA: "scanMine" >
| < SCANRADAR: "scanTank" >
| < GIRACANONIZQ: "turnCanonLeft" >
| < GIRACANONDER: "turnCanonRight" >
| < FRENTELIBRE: "freeInFront" >
| < FRENTEOCUPADO: "busyInFront" >
| < ATRASLIBRE: "freeBack" >
| < ATRASOCUPADO: "busyBack" >
| < RANDOM: "random" >
| < IF: "if" >
| < ELSE: "else" >
| < WHILE: "while" >
| < DO: "do" >
| < STOPIF: "endIf" >
| < STOPWHILE: "endWhile" >
| < FIN: "end" >
| < PRINCIPIO: "begin" >
| < CHOQUE: "hasHit" >
| < NOCHOQUE: "notHasHit" >
| < ANGULOCANON: "angleCanon" >
| < ANGULOTANK: "angleTank" >
}
```

Figura 4.5: Tokens del lenguaje

```
TOKEN: /*VALORES*/
{
  < CONSTENTERA: (["0"-"9"])+ >
}

TOKEN: /*SYMBOLS*/
{
 < ABRIRPAREN: "(" >
| < CERRARPAREN: ")" >
| < COMMA: "," >
| < MAYOR: ">" >
| < MENOR: "<" >
| < IGUAL: "=" >
| < MAI: ">=" >
| < MEI: "<=" >
| < NI: "<>" >
| < MINUS: "-" >

}
```

Figura 4.6: Símbolos del lenguaje

El analizador sintáctico es la segunda fase del compilador y tiene dos funciones principales. En primer lugar, analizar todas las cadenas de símbolos de acuerdo con las reglas de la gramática (Anexo B.4.2). Para ello, se definen todos los tokens necesarios y la gramática formal en formato BNF (apartado 3.4). La segunda función consiste en realizar las acciones asociadas a cada instrucción. JavaCC permite colocar acciones en las funciones después de un terminal, esta funcionalidad nos ayudará a gestionar los movimientos del tanque.

Capítulo 5

Resultados

Finalmente se muestra el resultado final de la aplicación donde se mostrarán algunas ventanas y las pruebas pertinentes realizadas por usuarios.

5.1. Versión final del juego

A continuación, se muestran una serie de capturas de la versión final del videojuego (Figura 5.1, 5.2, 5.3 y 5.4). Mostrando la pantalla de batallas con las posibles combinaciones de partidas que ofrece el juego: dos tipos de partida, *All vs All* y *Team Deathmatch*. Dos modos juego, *Time Trial* y *Free Battle*. Con estas imágenes se puede observar las posibilidades del juego para ofrecer una diversión al jugador al mismo tiempo que aprende conocimientos de programación.

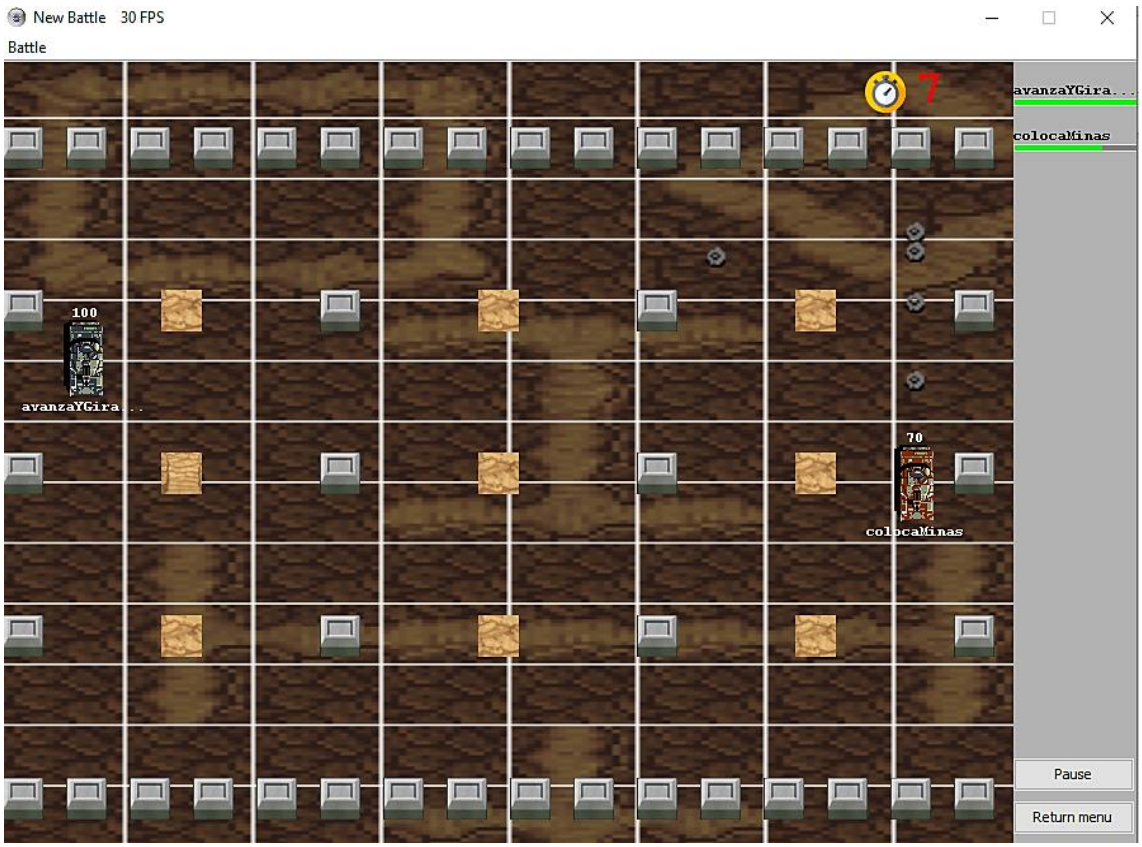


Figura 5.1: Batalla All vs All con modo Time Trial

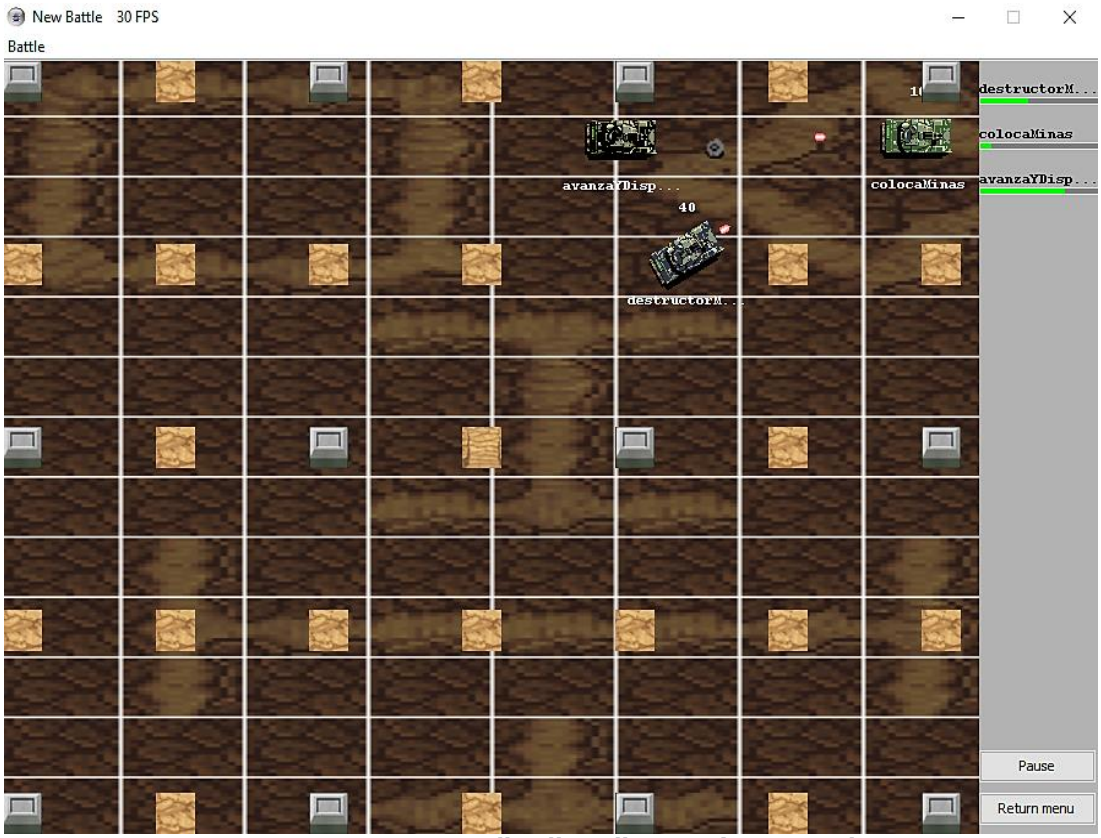


Figura 5.2: Batalla All vs All con modo Free Battle

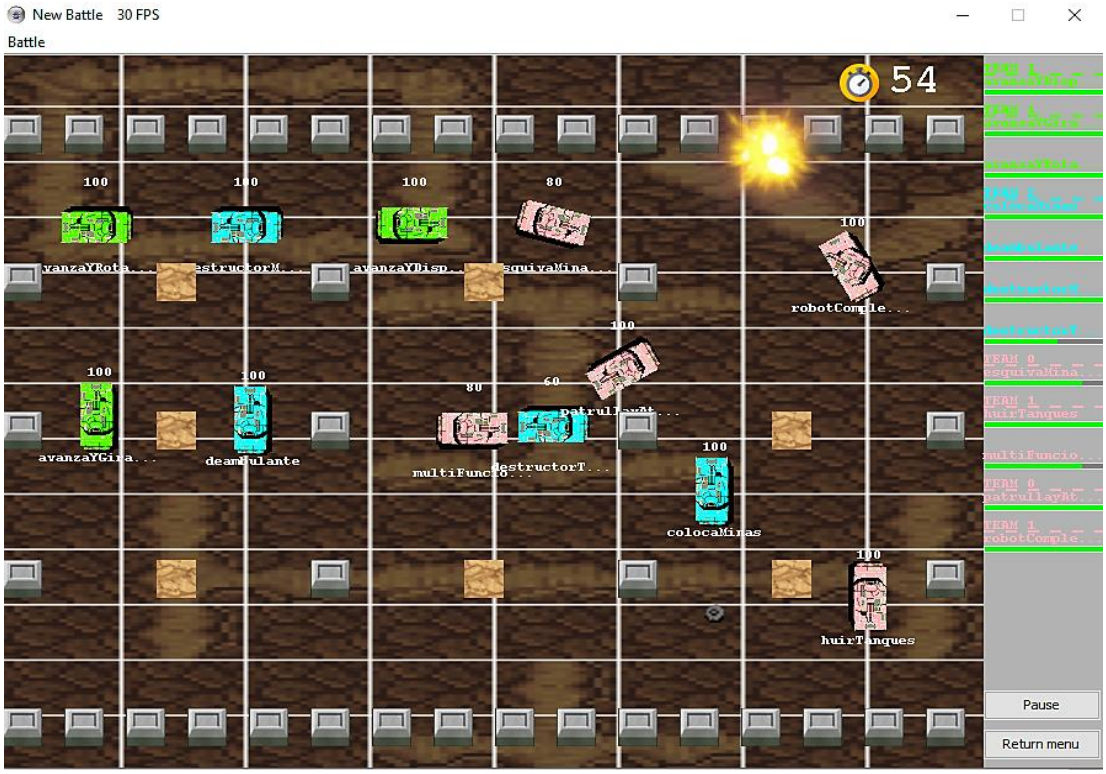


Figura 5.3: Batalla Team Deathmatch (tres equipos) con modo Time Trial

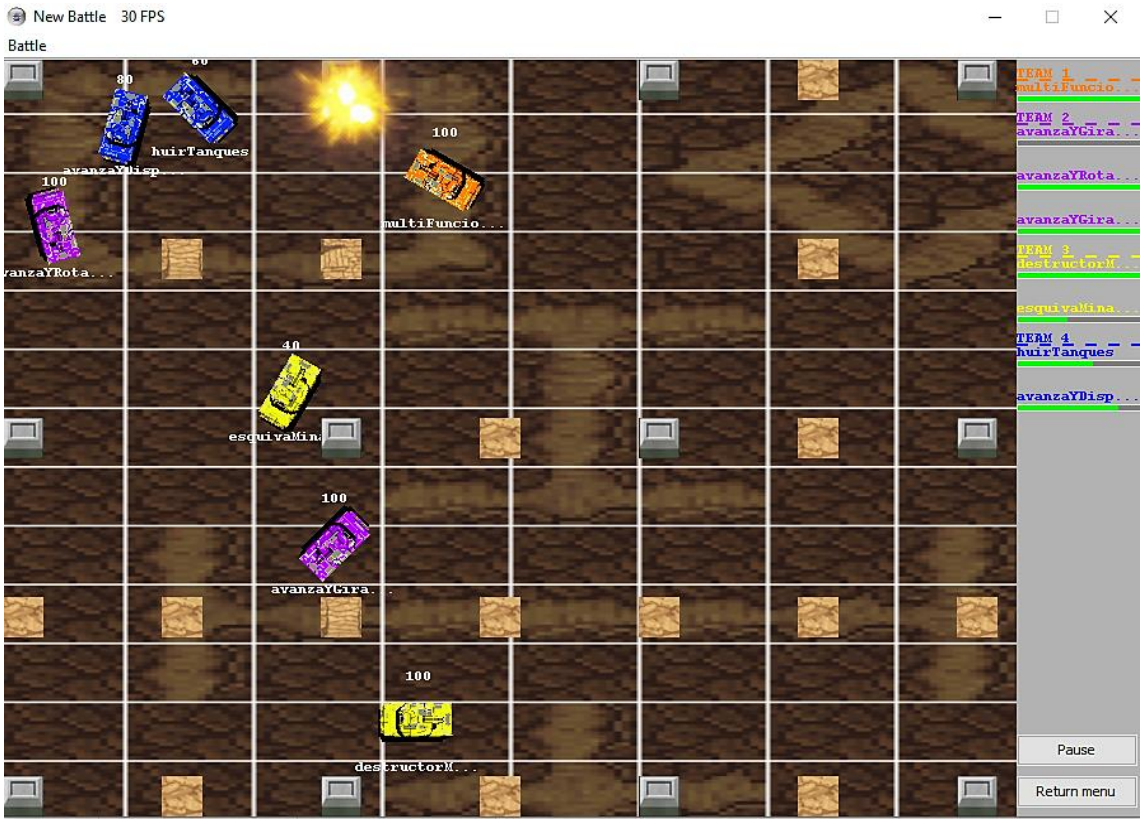


Figura 5.4: Batalla Team Deathmatch (cuatro equipos) con modo Free Battle

5.1.1. Antialiasing

Se puede observar claramente (Figura 5.5) como el antialiasing [13] reduce de forma significativa los artefactos (se observa principalmente en la sombra del tanque y el nombre) y mejora la calidad de la imagen.

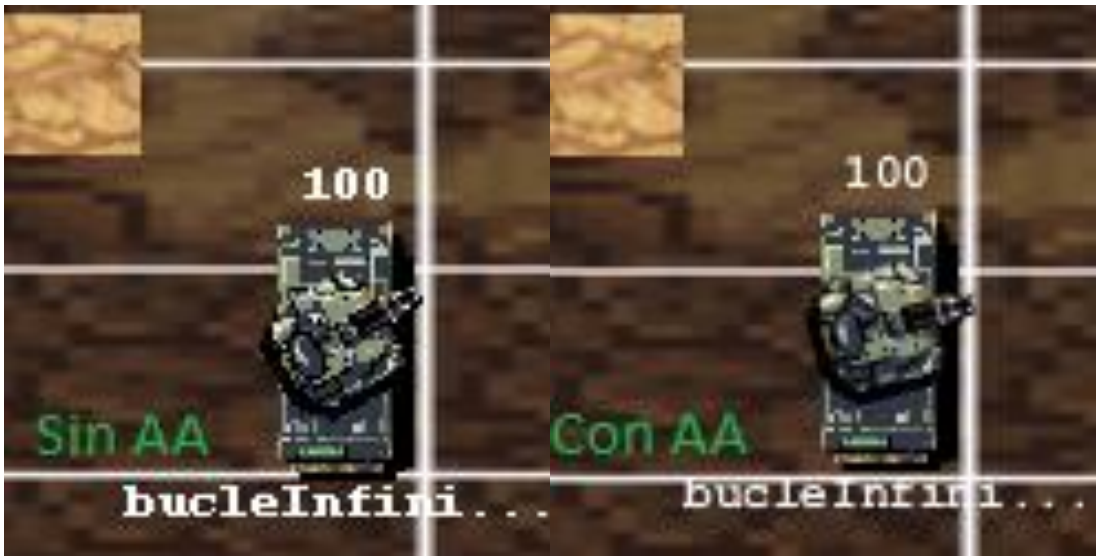


Figura 5.5: Comparativa con zoom

5.2. Evaluación de los usuarios

Tras finalizar la implementación del juego, se empezó la fase de pruebas para comprobar la correcta jugabilidad del videojuego. Para ello, se seleccionaron diez usuarios con diferentes conocimientos sobre informática y videojuegos que evaluaron diferentes aspectos: la navegación en los menús, la facilidad de programación y la jugabilidad. De los diez usuarios, se fragmentan en las siguientes edades:

- El 30% de los usuarios tienen entre 15 y 18 años
- El 50% de los usuarios tienen entre 19 y 25 años
- El 20% de los usuarios tienen entre 26 y 30 años

5.2.1. Navegación en los menús

En primer lugar, se hicieron pruebas para comprobar la navegación entre los menús (Figura 5.6). Casi todos los usuarios entendieron a la primera la navegación por la aplicación, como empezar una nueva batalla y abrir el editor (al 70% le parece excelente o muy buena). En general, los resultados son satisfactorios, no hay mención de ningún problema importante en la navegación exceptuando un usuario de 15 años que necesitó un tiempo mayor para la comprensión de los menús.

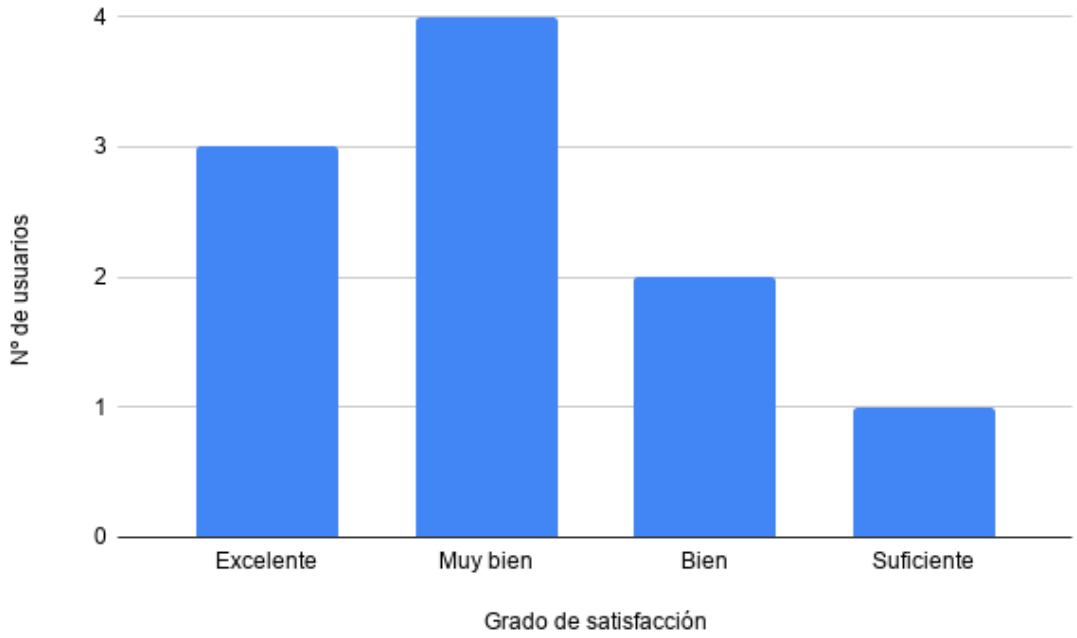


Figura 5.6: Nivel de adecuación en la navegación por los menús

5.2.2. Facilidad de programación

A continuación, se probó el nivel de adecuación cuando se empieza a programar un tanque (Figura 5.7). Algunos usuarios (con más conocimientos de informática y videojuegos), entendieron fácilmente el uso del editor de texto. Sin embargo, la mayoría tuvieron problemas a la hora de advertir como empezar un programa de cero y aprenderse las instrucciones que permitía el lenguaje. Para solucionar este problema se plantea como posible mejora la inclusión de un tutorial al comienzo del juego que explique detalladamente la creación de programas, su compilación y su uso posterior al finalizar la programación. El juego actualmente cuenta con un manual de usuario que puede ser de utilidad antes de empezar a jugar (pocas veces es leído antes de jugar por parte de los usuarios).

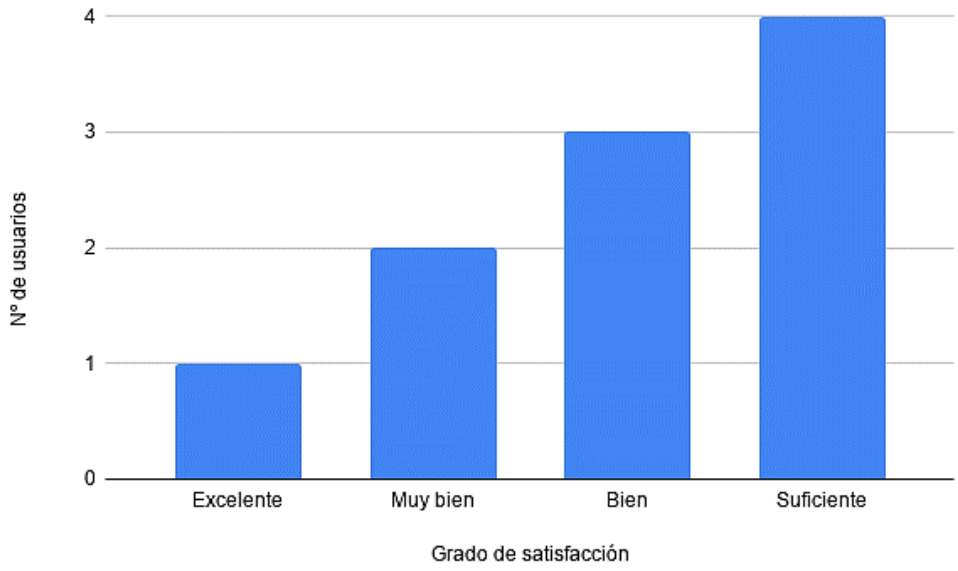


Figura 5.7: Nivel de adecuación al comienzo de la programación

5.2.3. Jugabilidad

También se preguntó a los usuarios por el nivel de jugabilidad del videojuego (Figura 5.8). Algunos usuarios encontraron puntos positivos en las batallas, con cierto grado de diversión cuando se conocían bien todas las mecánicas del juego y sus limitaciones. Muy pocos usuarios sufrieron una pequeña frustración de las batallas ya que algunas pueden resultar largas. De nuevo, predomina un nivel de agrado alto (el 70% de los usuarios les parece excelente o muy buena).

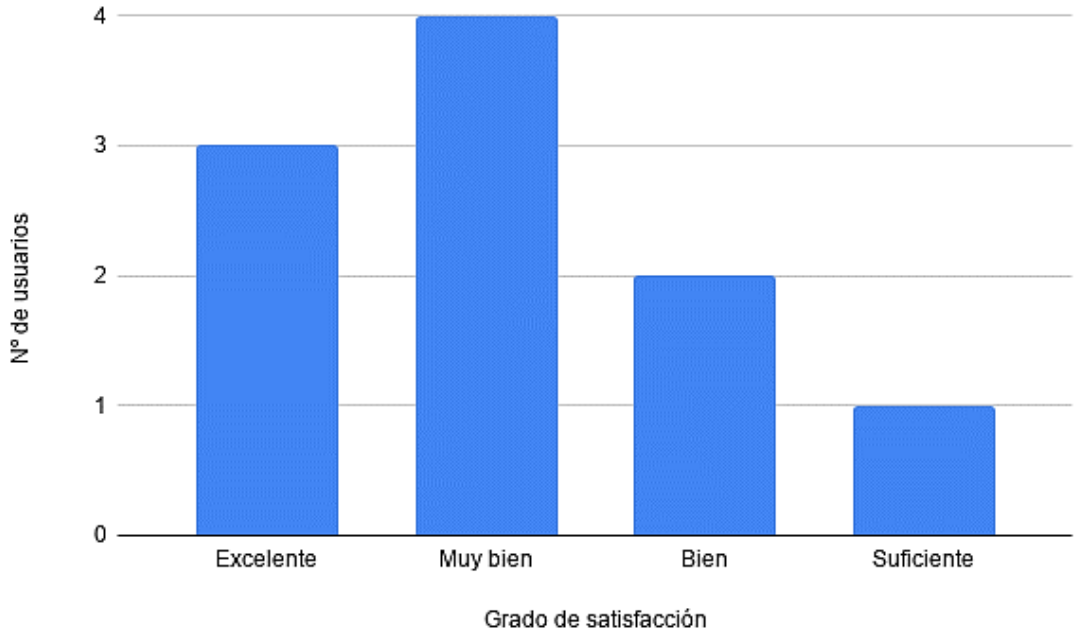


Figura 5.8: Nivel de jugabilidad

Capítulo 6

Conclusiones

Es al finalizar el desarrollo del TFG cuando se puede mirar atrás y ver el recorrido que éste ha tenido. Se comenzó este proyecto con el objetivo principal de crear un videojuego que pudiese utilizarse con el fin de enseñar a programar.

Para llevarlo a cabo, se han empleado una gran variedad de tecnologías y herramientas. Esto ha significado adoptar el flujo de trabajo propio de la industria del videojuego, es decir, crear contenido digital para después importarlo en un entorno de desarrollo. Además, se han puesto en práctica conocimientos de una gran cantidad de asignaturas de la carrera: Ingeniería del Software, Videojuegos, Procesadores de Lenguajes, Programación de Sistemas Concurrentes y Distribuidos, Estructura de Datos y Algoritmos. La combinación de estos conocimientos, especialmente en lo referente a la programación y desarrollo de un proyecto software, ha sido fundamental para el desarrollo de este trabajo.

Gracias a todo ello se ha conseguido agregar la funcionalidad necesaria para que se pueda introducir contenido educativo y para que los alumnos se diviertan aprendiendo. En definitiva, puede afirmarse que se ha logrado crear una técnica de aprendizaje, la gamificación.

En el transcurso de la realización de este proyecto se ha invertido una gran cantidad de tiempo y esfuerzo. Cada una de las herramientas utilizadas ha requerido de su propia curva de aprendizaje. Aunque gran parte de este proyecto se ha realizado con Eclipse, con una librería básica que es Java Swing [4], Java ofrece diferentes posibilidades adicionales (con otras librerías) que no se han llegado a explorar en este trabajo. Aun así, se ha aprendido a manejar los elementos más importantes del software y a estructurar internamente un videojuego.

Como se ha recalcado en la introducción de esta memoria, la gamificación representa el futuro de la educación. El videojuego creado en este TFG, RobotWars, aporta una nueva herramienta que se suma a otras iniciativas ya establecidas en el mundo académico.

Como última reflexión, cabe señalar que los trabajos de fin de grado son esenciales tanto para la formación de los alumnos como para el mundo académico. A los alumnos nos permite poner a prueba gran parte de los conocimientos aprendidos en la carrera, pero también nos da la oportunidad de innovar y enfocar de manera novedosa el problema planteado. Llegados a este punto, la valoración personal de haber realizado un trabajo propio con la libertad para elegir el tema deseado, sin ninguna imposición por parte del director, hace que su desarrollo y su finalización sean muy satisfactorios. Este tipo de trabajos son los que ofrecen un enriquecimiento personal y profesional que pone la brocha final a tantas horas de dedicación y esfuerzo durante el transcurso del grado.

6.1. Posibles ampliaciones

Durante la realización de este proyecto y gracias al *feedback* que se recibió durante la fase de pruebas, se han extraído algunas ideas que podrían implementarse en posibles ampliaciones:

- Aportar mayor potencia al lenguaje, con declaración de variables o añadiendo alguna instrucción más. Es una mejora que resultaría muy útil a los alumnos ya que aumentaría considerablemente la diversión del videojuego.
- Un requisito inicial (de baja prioridad) era la posibilidad de jugar en sistemas Linux y Mac. En ambas plataformas se ha probado y aunque el juego arranca, la gestión de los ficheros es incorrecta. Habría que implementar un sistema más genérico que tuviese cabida para estos sistemas de archivos. Con esta mejora, incluiríamos nuevos usuarios potenciales, sin restringir el sistema operativo de su ordenador.
- Añadir una funcionalidad online al videojuego sería el siguiente paso en su desarrollo. Podría reutilizarse la gran mayoría del trabajo realizado, habría que añadir una infraestructura modelo cliente-servidor para llevarlo a cabo. Además, podría crearse un ranking para generar cierta competición entre jugadores.
- Incluir un tutorial la primera vez que se juega para guiar al usuario explicando detalladamente la creación de programas, el repertorio de instrucciones, la interfaz y el uso posterior de los tanques creados.
- Añadir un mayor número de mapas que el jugador pudiera seleccionar antes de empezar una nueva batalla. Mejoraría la jugabilidad considerablemente.

6.2. Cronograma

En la Figura 6.1 se refleja, de manera global, el desarrollo de las tareas realizadas a lo largo del proyecto. Esto ha sido posible de gestionar gracias a que durante toda la realización del proyecto se ha llevado una contabilidad diaria de cada tarea a partir de anotaciones internas. En total, la realización de este trabajo ha llevado un total de 621 horas.

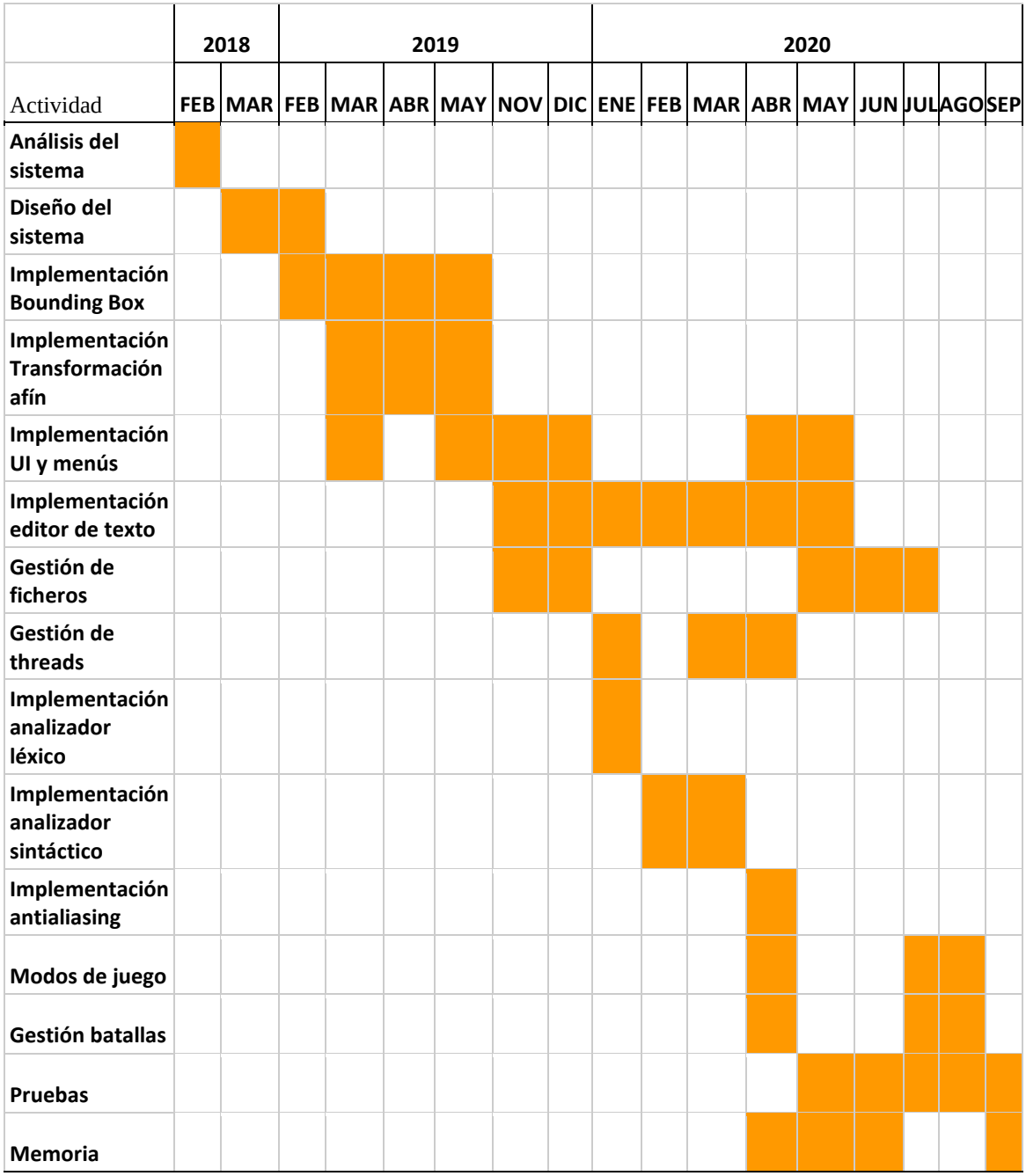


Figura 6.1: Cronograma esfuerzos realizados

Capítulo 7

Bibliografía

[1] Lambert M Surhone, Mariam T Tennoe, Susan F Henssonow. *Robocode*. Betascript Publishing. 2011

[2] CODE Project. For Those who code. https://www.codeproject.com/Articles/50377/Create-Your-Own-Programming-Language [Online; Accedido 3-Enero-2020]

[3] Wikipedia- Transformación afín -Wikipedia, the free enclycopedia. https://es.wikipedia.org/wiki/Transformaci%C3%B3n_af%C3%ADn [Online; Accedido 24-Abril-2020]

[4] Yang, *Java Swing Tutorials* -Herong's Tutorial Examples. 2017

[5] Hurtado Gil, Sandra Victoria. *Conceptos avanzados de programación con JAVA.* Universidad, ICESI, 2002

[6] JPG to PNG. Free online tool. https://jpg2png.com/ [Online; Accedido 16-Abril-2019]

[7] Online PNG tools https://onlinepngtools.com/resize-png [Online; Accedido 16-Abril-2019]

[8] Anthony J. Dos Resis. *Compiler construction using Java, Javacc and Yacc.* Willey, 2012

[9] T. Copeland. *Generating Parsers With JavaCC.* Second Edition. Centennial Books, 2013.

[10] GeeksforGeeks A computer science portal for geeks https://www.geeksforgeeks.org/java-swing-create-a-simple-text-editor/ [Online; Accedido 8-Diciembre-2019]

[11] S. Galvez Rojas, *Traductores, Compiladores e Intérpretes* OCW- Universidad de Málaga, 2011. https://ocw.uma.es/pluginfile.php/1025/mod_resource/content/0/Capitulo_5.pdf [Online; Accedido 22-Enero-2020]

[12] Zang, *Computer Graphics Using Java 2D&3D*. Pearson, 2017.

[13] Wikipedia- Antialiasing -Wikipedia, the free enclycopedia https://es.wikipedia.org/wiki/Antialiasing [Online; Accedido 6-Junio-2020]

[14] HOBBYCONSOLAS https://www.hobbyconsolas.com/reportajes/que-es-antialiasing-videojuegos-hobby-basics-257305 [Online; Accedido 6-Junio-2020]

[15] Java™ Documentación. Los tutoriales de Java™ Cómo usar el subsistema Focus https://docs.oracle.com/javase/tutorial/uiswing/misc/focus.html [Online; Accedido 8-Junio-2020]

[16] Java™ Documentación. Los tutoriales de Java™ https://docs.oracle.com/javase/tutorial/uiswing/components/textarea.html [Online; Accedido 7-Diciembre-2019]

[17] Ejemplos Java y C/Linux http://www.chuidiang.org/java/timer/timer.php [Online; Accedido 2- Mayo-2020]

[18] Java™ Plataform Standard Ed. 77. Paquete Javax.sound.sampled Class AudioFormat https://docs.oracle.com/javase/7/docs/api/javax/sound/sampled/AudioFormat.html [Online; Accedido 19-Marzo-2020]

[19] Java Documentation. Capítulo 4: reproducción de audio https://docs.oracle.com/javase/8/docs/technotes/guides/sound/programmer_guide/chapter4.html [Online; Accedido 20-Marzo-2020]

[20] Java™ Ejemplo editor Pane http://www.chuidiang.org/java/ejemplos/JEditorPane-JTextPane/EjemploJEditorPaneHtml.java.html [Online; Accedido 22-Abril-2020]

Anexos

A. Diseño preliminar

Se detallan todos los recursos creados en la fase de diseño de la aplicación como bocetos y las instrucciones del lenguaje inicial.

Durante la fase de diseño del juego, se realizaron unos bocetos para clarificar el aspecto de la aplicación y saber cómo iba a ser la distribución de todos los componentes de cada una de las ventanas. Hay que recalcar que algunas ventanas han podido sufrir ciertas modificaciones a lo largo del desarrollo software pero se puede observar como la gran mayoría se han mantenido prácticamente intactas.

Además, se diseñó una máquina de estados permitiendo obtener a alto nivel el mapa de navegación de la aplicación y cuál iba a ser el flujo del usuario a través de todas las ventanas (Figura A.1).

El diagrama consta de nueve estados y las transiciones entre ellos dependerán de la opción seleccionada en la aplicación.

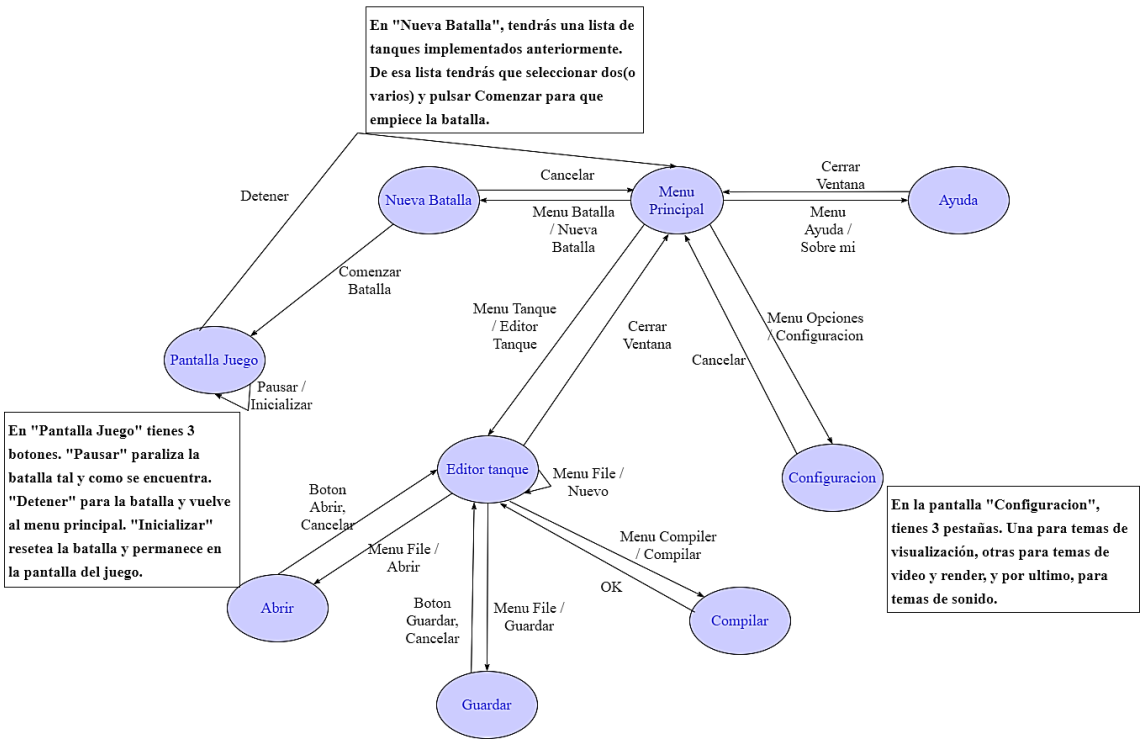


Figura A.1: Diagrama de estados

A continuación, se muestran los bocetos correspondientes a la ventana principal (Figura A.2, A.3, A.4, A.5 y A.6).

Figura A.2: Ventana principal

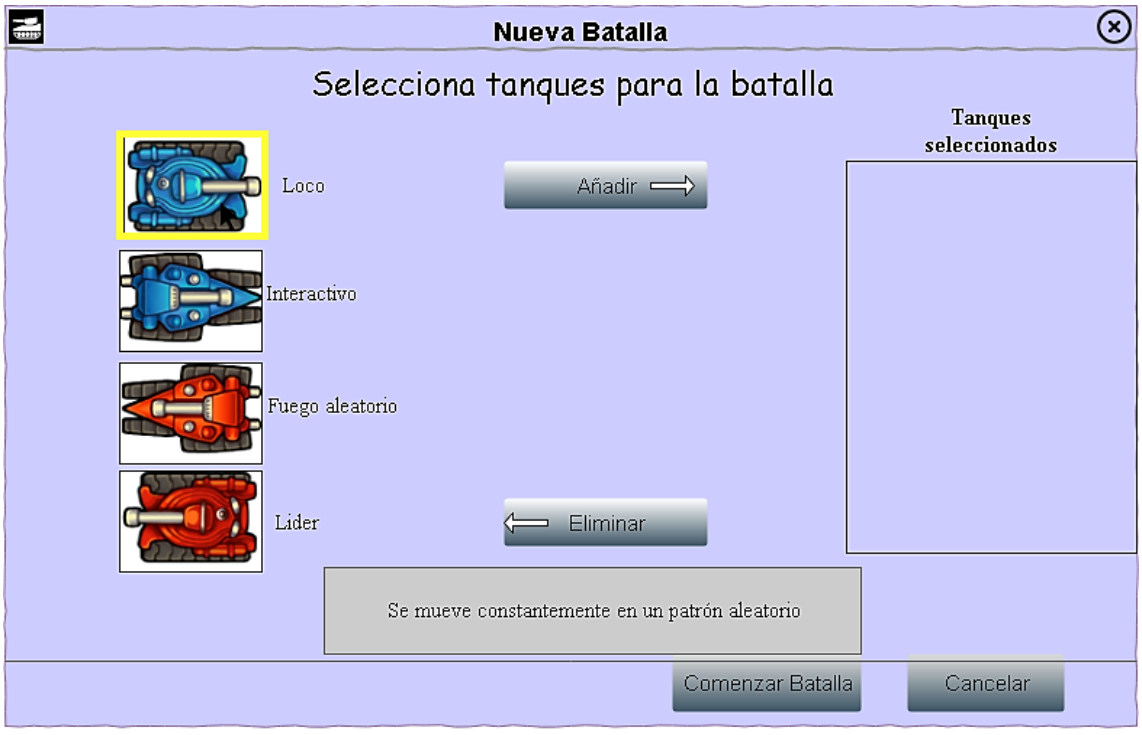
Figura A.3: Ventana principal (Batalla)

Figura A.4: Ventana principal (Tanque)

Figura A.5: Ventana principal (Opciones)

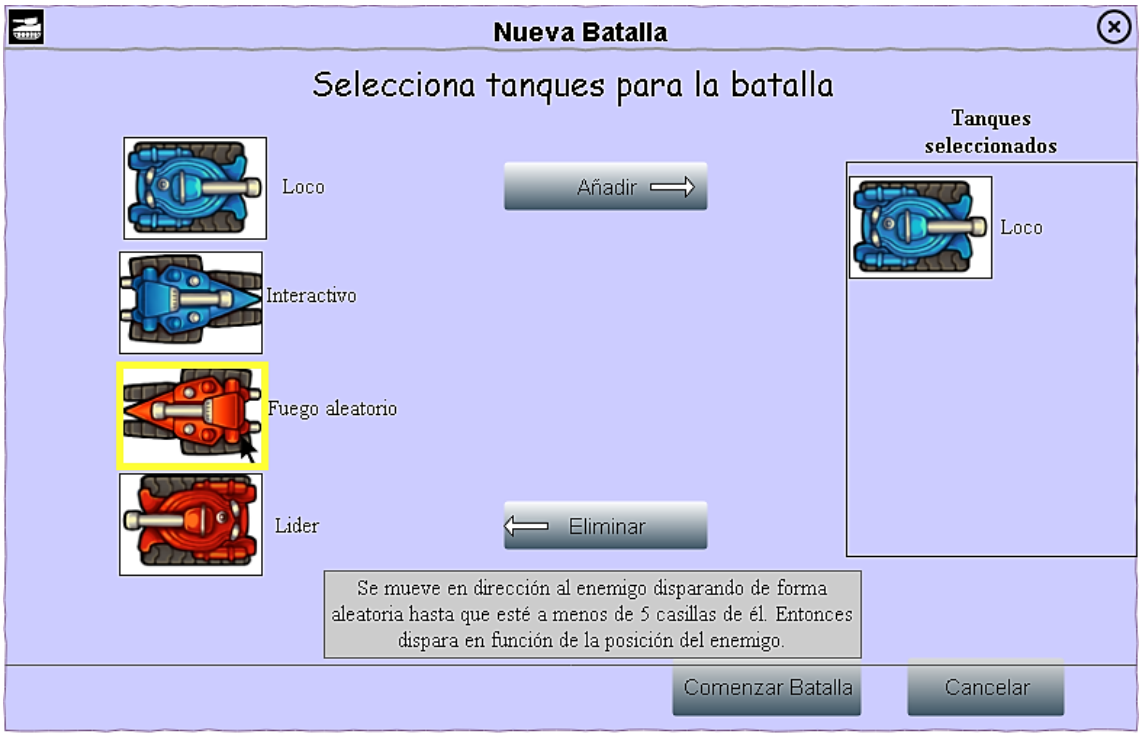
Figura A.6: Ventana principal (Ayuda)

Posteriormente, se crearon los bocetos correspondientes a la ventana de selección de tanques (Figura A.7 y A.8).



**Los nombres de los tanques son guardados por el programador. Se podría meter alguna breve descripción de su principal punto fuerte al lado del tanque cuando lo seleccionas.

Figura A.7: Nueva batalla 1



**Los nombres de los tanques son guardados por el programador. Se podría meter alguna breve descripción de su principal punto fuerte al lado del tanque cuando lo seleccionas.

Figura A.8: Nueva batalla 2

A continuación, se muestra el boceto correspondiente a la ventana de batallas (Figura A.9).

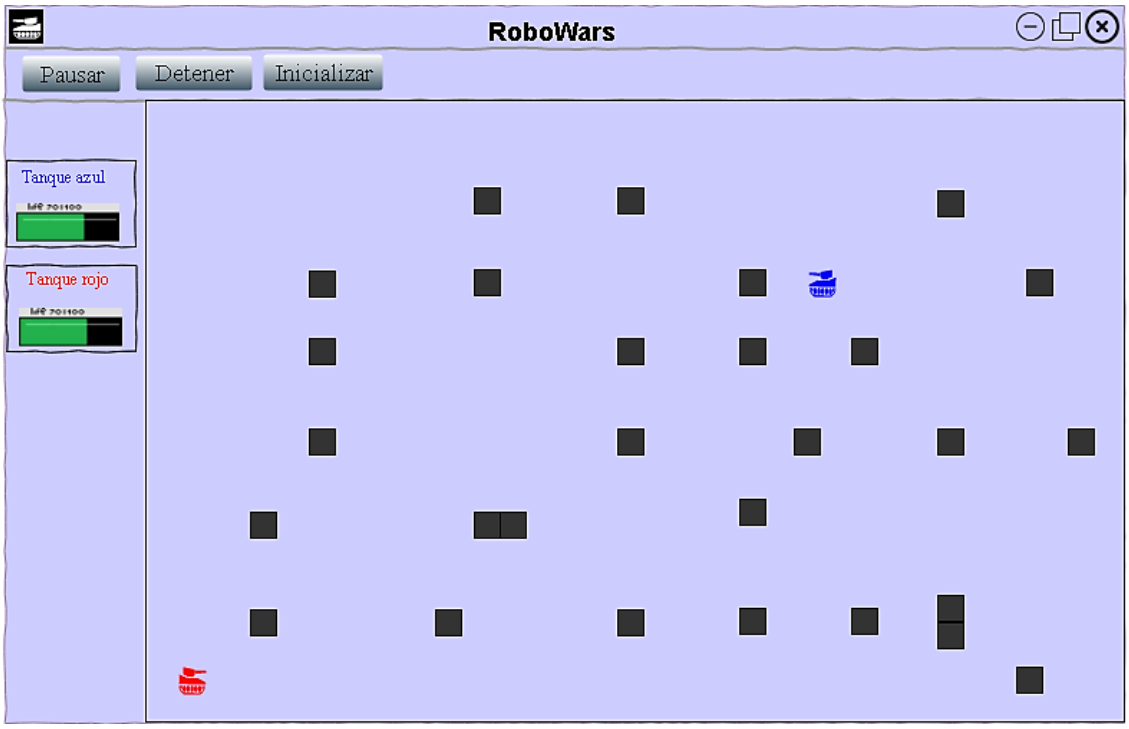


Figura A.9: Ventana Batallas

Posteriormente, se diseñaron los bocetos correspondientes a la ventana del editor de código (Figura A.10, A.11, A.12 y A.13).

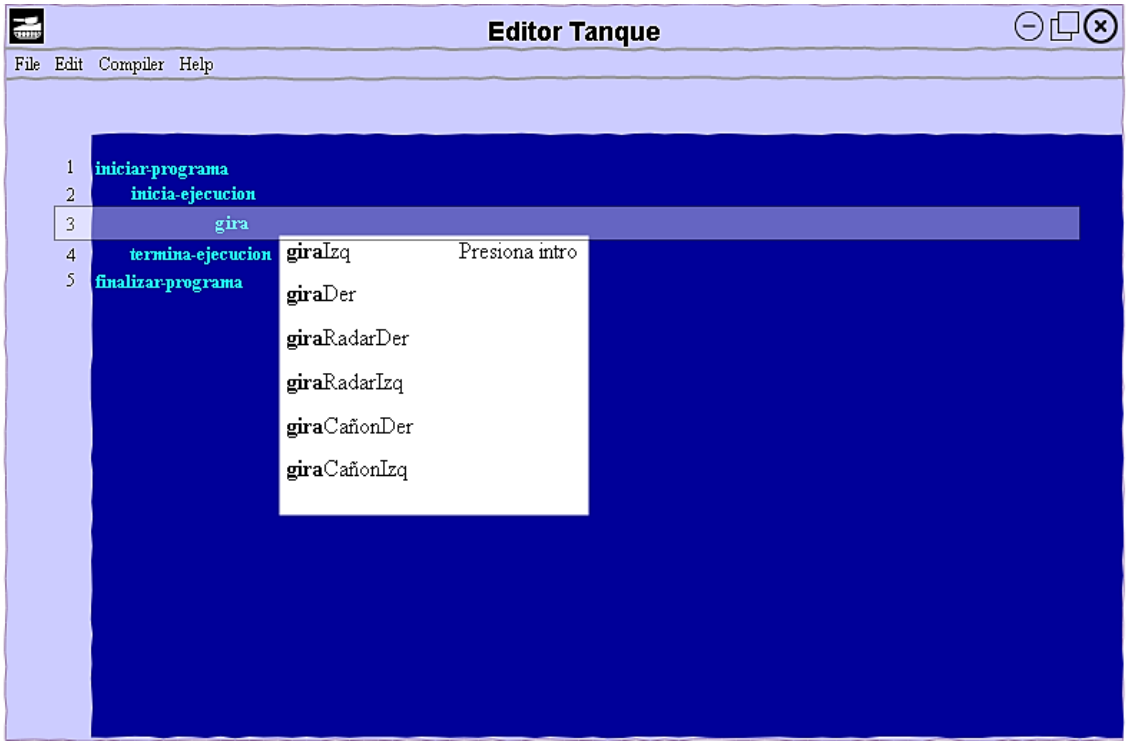


Figura A.10: Editor código

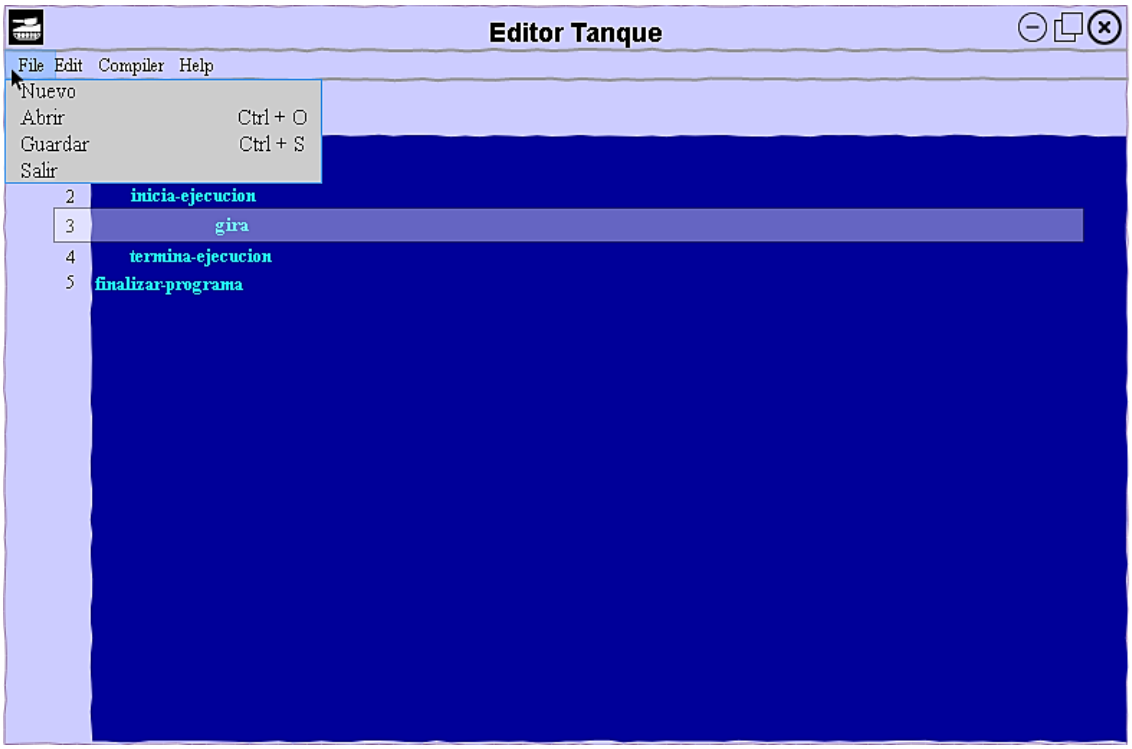


Figura A.11: Editor código (File)

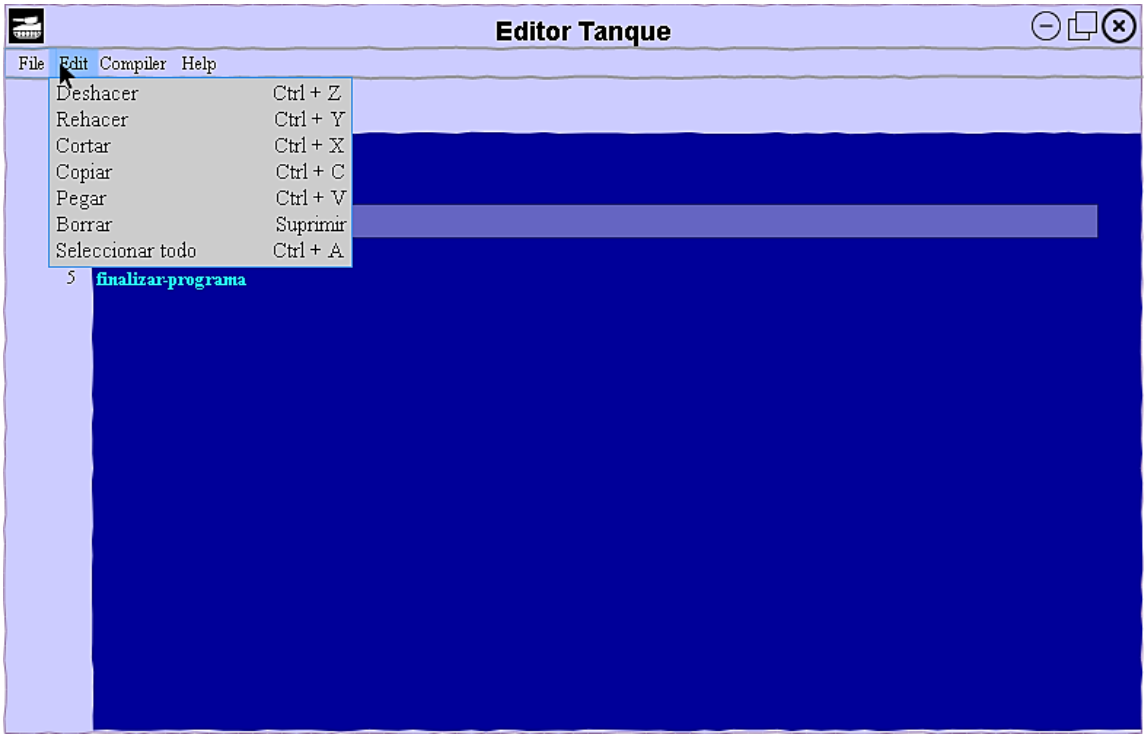


Figura A.12: Editor código (Edit)

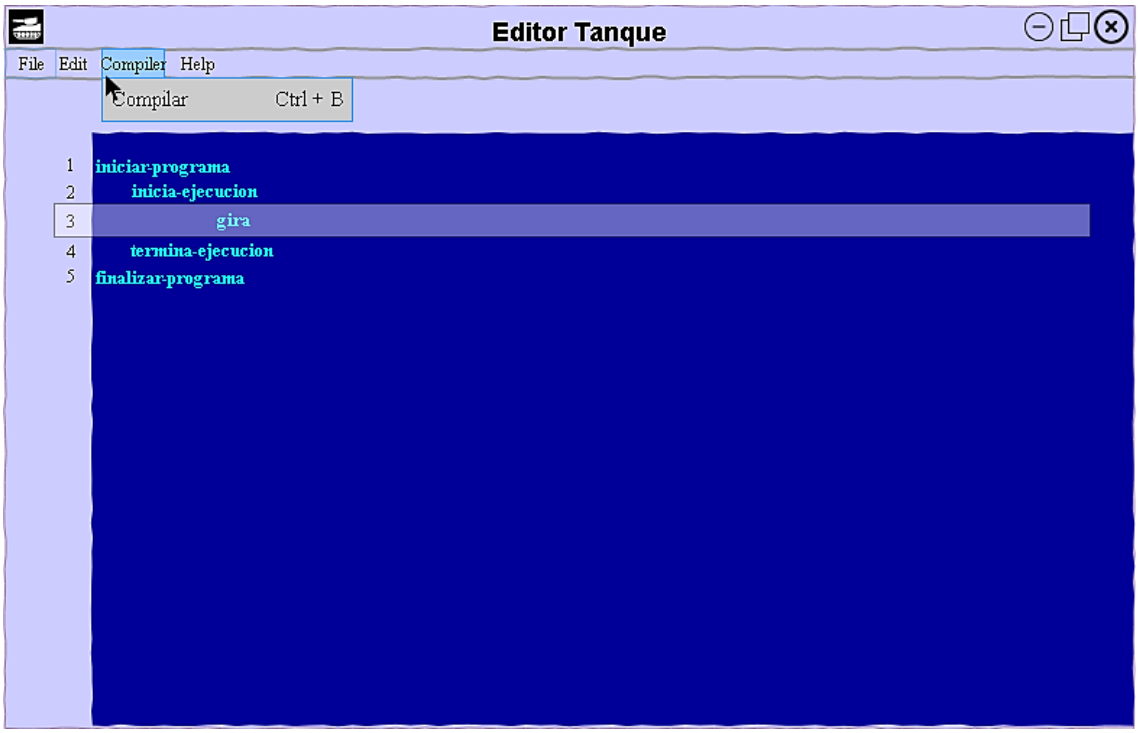


Figura A.13: Editor código (Compiler)

A continuación, se muestran los bocetos correspondientes con la ventana de configuración (Figura A.14, A.15 y A.16). Se diseñaron tres pestañas, una de visualización, con las opciones relacionadas con los textos mostrados en la batalla a modo de información. Otra pestaña de vídeo, para configurar opciones gráficas como la resolución de la pantalla. Finalmente, la pestaña de sonido, para ajustar las opciones de audio del juego como el volumen o los sonidos FX.

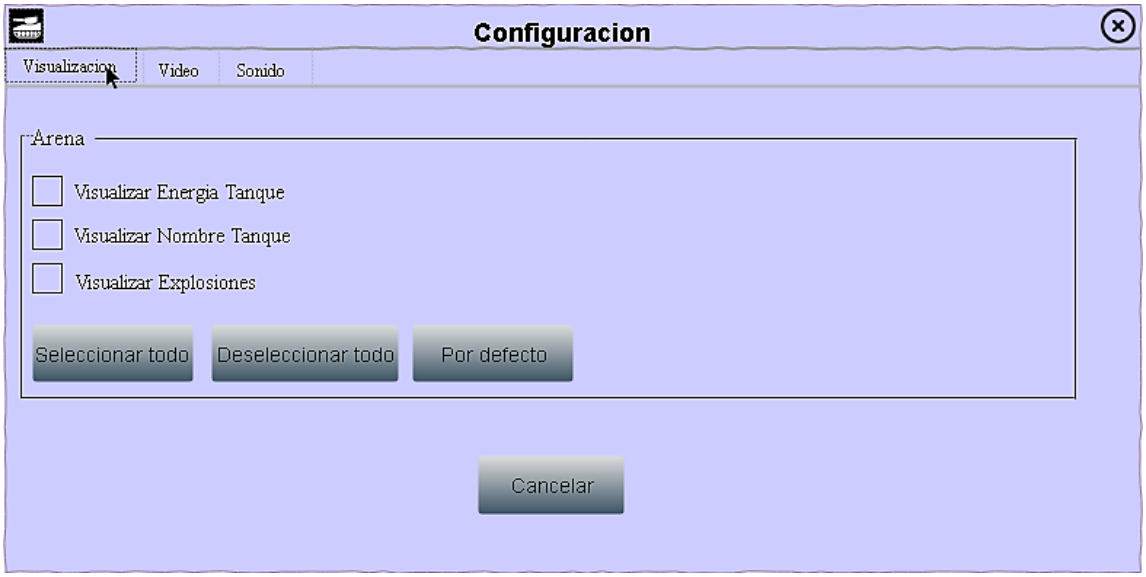


Figura A.14: Ventana configuración (Visualización)

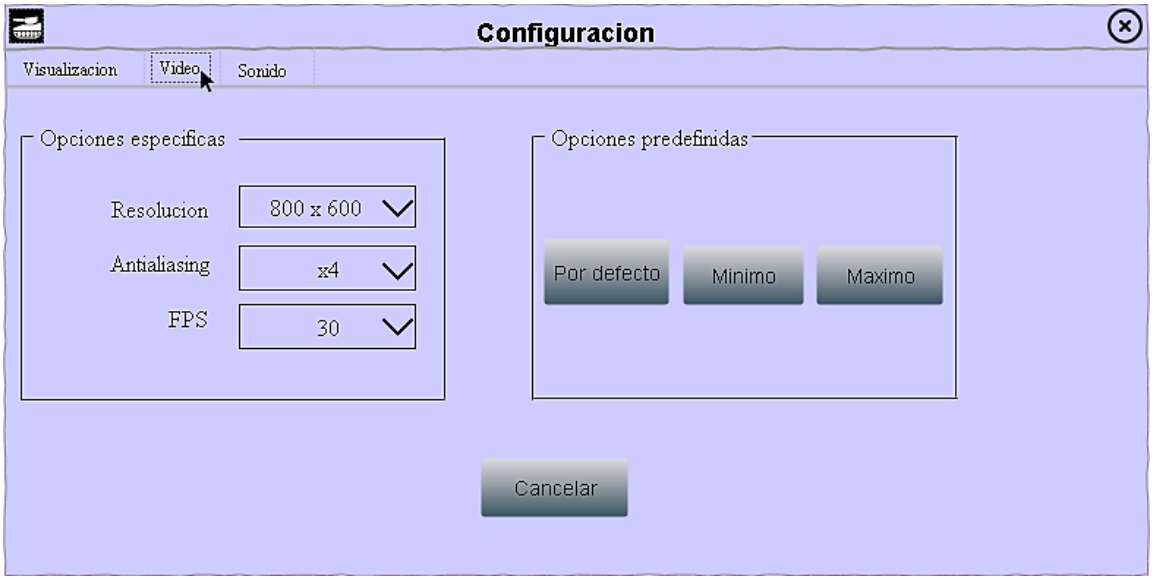


Figura A.15: Ventana configuración (Video)

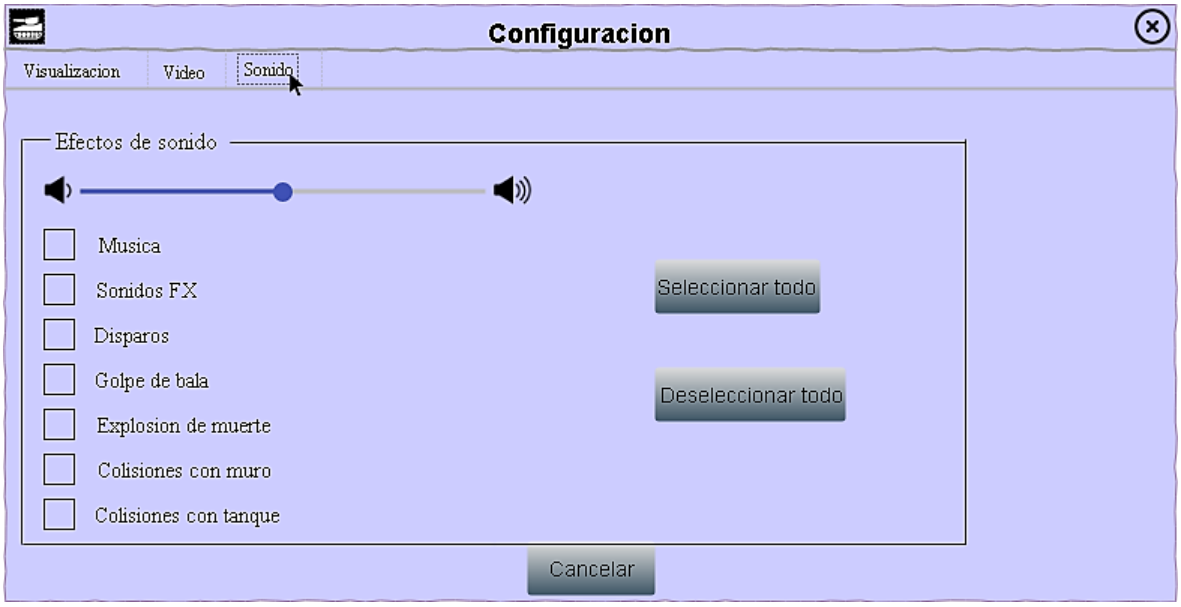


Figura A.16: Ventana configuración (Sonido)

Por último, se diseñó el boceto de la pantalla de ayuda (Figura A.17).

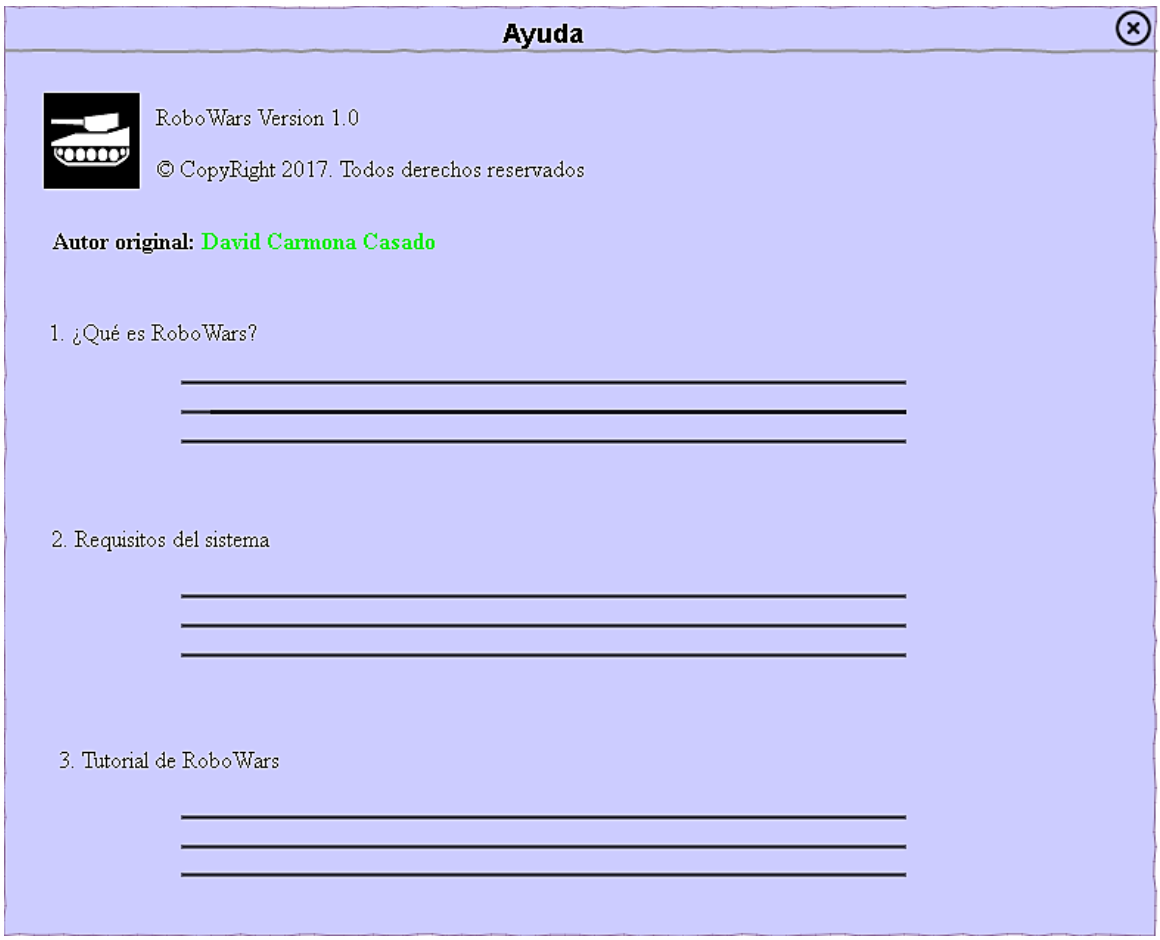


Figura A.17: Ventana ayuda

Junto con los bocetos, se diseñó el repertorio de instrucciones que iba a permitir el lenguaje. Este repertorio ha sufrido alguna pequeña modificación respecto a esta fase, pero las instrucciones principales y su funcionalidad se han mantenido (Figura A.18).

```
Instrucciones básicas
-----------------------------------------
avanza() => avanza una casilla
giraIzq() => gira a la izquierda 90°
giraDer() => gira a la derecha 90°
disparar() => dispara un misil
soltarMina() => suelta una mina en la casilla que se encuentra el tanque. Esta mina
puede explotar al pasar un tanque por encima o al ser disparada por otro tanque.
detectarMina() => devuelve true si hay una mina a "x" casillas
scanRadar() => devuelve true si hay un tanque a "x" casillas
giraRadarDer() => gira el radar 90 grados a la derecha
giraRadarIzq() => gira el radar 90 grados a la izquierda
giraCañonDer() => gira el cañon 90 grados a la derecha
giraCañonIzq() => gira el caño 90 grados a la izquierda
frenteLibre() => devuelve true si no hay ningun muro en frente
mirandoNorte() => devuelve true si el tanque está actualmente orientado hacia el Norte
mirandoSur() => devuelve true si el tanque está actualmente orientado hacia el Sur
mirandoEste() => devuelve true si el tanque está actualmente orientado hacia el Este
mirandoOeste() => devuelve true si el tanque está actualmente orientado hacia el Oeste


Instrucciones de control
-----------------------------------------
ExpresionSi:= "si" termino "hacer"
    expresion
["sino"
    expresion]

ExpresionMientras:= "mientras" termino "hacer"
    expresion

Consideraciones
-----------------------------------------
Un thread por cada tanque.
Se permite la posibilidad de crear un mapa con caracteres para especificar
el escenario. Cada caracter representa un tile. Los pasillos se dejaran con caracteres
en blanco.
```

Figura A.18. Repertorio instrucciones (Inicial)

B. Implementación

Se detallan los problemas planteados en la fase de implementación junto con las soluciones desarrolladas.

B.1. Gestión de colisiones

B.1.1. Bounding Box

Para implementar esta idea, se ha necesitado la clase *Path2D*[8], esta clase tiene métodos que nos permiten comprobar la intersección entre dos figuras. Como la bounding box, es un rectángulo, para cada objeto se necesitarán cuatro puntos que deberán ser unidos para obtener la forma deseada. Cada vez que el tanque avance o rote, habrá que calcular su nueva bounding box y comprobar si existe alguna intersección. El cálculo en el eje X es el siguiente:

$$sumaX = \sin(gradosTank) * speed$$

Donde *gradosTank*, son los grados del tanque (en radianes), respecto a su posición inicial y *speed*, es una constante que hay que ajustar al comienzo del desarrollo para obtener un movimiento realista (se utilizará en siguientes apartados). Hay un booleano que controla si se quiere avanzar o retroceder, en este último caso, el cálculo es el mismo, pero negado. El cálculo en el eje Y es similar, pero hay que realizar un ajuste, ya que a parte de los grados del tanque hay que saber en qué sentido ha girado.

Con estos nuevos valores se calcula la nueva posición que será provisional. A continuación, se llama a una función que recorre todos los obstáculos y comprueba si la bounding box de cada uno de ellos intersecta con la nueva posición del tanque. En función de esto, devuelve un booleano indicando si puede avanzar y, por tanto, la nueva posición se hará efectiva o no. Esta comprobación se repite con el resto de los tanques y con las minas, es decir, con todos los objetos visibles en el mapa. Para llevar a cabo la nueva posición, solo hay que sumar la posición en X actual a lo calculado en la fórmula anteriormente mencionada. Ídem con el eje Y.

Al rotar el tanque también hay que calcular su nueva posición en el mapa. El cálculo es el siguiente:

[8] https://docs.oracle.com/javase/7/docs/api/java/awt/geom/Path2D.html

$$tempX = puntoX(i) - centroRectanguloX$$

$$tempY = puntoY(i) - centroRectanguloY$$

Donde *puntoX*(i) es la posición en el eje X del punto i, siendo i el índice correspondiente a los cuatro puntos que forman la bounding box. *CentroRectanguloX* es el centro del tanque en el eje X. *PuntoY*(i) y *centroRectanguloY*, representan lo mismo, pero en el eje Y.

$$rotatedX = tempX * \cos(gradosTank) - tempY * \sin(gradosTank)$$

$$rotatedY = tempX * \sin(gradosTank) + tempY * \cos(gradosTank)$$

Una vez calculada la nueva posición (provisional) el procedimiento es exactamente igual que al avanzar.

B.1.2. Transformación afín

Es una transformación que preserva las relaciones de colinealidad entre puntos, es decir, puntos que recaían sobre una misma línea (o sobre un mismo plano) antes de la transformación, son preservadas [3].

Para implementar esta idea, se ha necesitado la clase *AffineTransform*[9] que permite realizar los movimientos de traslación y rotación que se necesitan en el juego. Cada tanque necesita dos matrices de transformación (o matriz ampliada), una para el cuerpo del tanque y otra para el cañón. En primer lugar, hay que instanciar ambos objetos pasando como parámetro la posición del tanque y del cañón. Cuando un tanque avanza, se necesita hacer un movimiento de traslación, en nuestro caso, se necesita operar en la dirección del eje Y para obtener el movimiento deseado.

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & -speed \\ 0 & 0 & 1 \end{pmatrix}$$

En la matriz anterior estamos indicando que el movimiento de traslación debe ser sobre el eje Y. La distancia de traslación entre la coordenada origen y destino es la variable *speed* (la constante definida en apartados anteriores). En caso de querer retroceder se usa la misma matriz, pero con la variable *speed* positiva. No hay que olvidar que, al moverse

[9] https://docs.oracle.com/javase/7/docs/api/java/awt/geom/AffineTransform.html

el cuerpo del tanque, debe moverse el cañón en la misma posición por lo que habrá que asignar el resultado de la matriz anterior al cañón.

Para rotar tanto el tanque como el cañón, se necesita un movimiento de rotación, que se consigue rotando los grados deseados junto con el centro del tanque o el cañón. Esta operación es equivalente a la siguiente secuencia de transformaciones:

$$translate(anchorx, anchory)$$
$$rotate(theta)$$
$$translate(-anchorx, -anchory)$$

Donde *anchorx* es el centro del tanque en el eje X y *anchory* es el centro del tanque en el eje Y, *theta* son los grados de rotación del tanque. Como se puede observar, primero se realiza la traslación de las coordenadas para que el punto de anclaje esté en el origen, después se hace la rotación del nuevo origen y finalmente se realiza la traslación para restaurar las coordenadas del punto de anclaje original. En este tipo de movimientos el cuerpo del tanque y el cañón son independientes, ambos pueden tener grados diferentes sobre sus ejes.

Las balas tienen el mismo comportamiento que al avanzar un tanque, los cálculos son los mismos que en el apartado anterior (apartado B.1.1). En este punto de la implementación se detectó un problema importante. Las comprobaciones para las intersecciones entre objetos no tenían en cuenta cuál estaba más cerca del origen de la bala por lo que ocasionaba comportamientos extraños durante el juego [12]. Simplemente si se detectaba una colisión se dejaba de iterar y se daba por hecho que había intersectado con ese objeto. Para solventarlo, hay que recorrerse todos los objetos, si se han detectado colisiones entre varios, hay que averiguar cual está más cerca, para ello se usó la clase *Point2D*[10], donde existen métodos que devuelven la distancia entre dos puntos en el plano.

En este caso, es importante validar si intersecta con algún objeto, si es así, se crea la matriz ampliada de la explosión que se usará para iniciar la animación de la misma. Las matrices de todos los objetos del juego se utilizarán para pintarlos en el campo de batalla ya que hay dos posibilidades a la hora de pintar un sprite: mediante su posición X e Y o mediante su transformación afín [3], se decidió usar esta última ya que reflejaba con mayor precisión la posición de los objetos.

[10] https://docs.oracle.com/javase/7/docs/api/java/awt/geom/Point2D.html

B.2. Mejoras gráficas

Para mejorar la calidad de imagen se ha implementado la técnica de *antialiasing* [13]. A continuación, se explica la técnica de forma detallada.

B.2.1. Antialiasing

El concepto de aliasing se produce cuando una imagen (un sprite, la textura, etc.) tiene una resolución inferior a la que es capaz de mostrar la pantalla y, por tanto, se muestra con bordes escalonados que resultan irreales [14].

El antialiasing busca corregir ese defecto mediante una técnica: toma el color del píxel que está en el borde y, a su lado, dibuja otro píxel cuyo color está entre el suyo y el color de lo que haya al fondo. En la Figura B.1, la A de la izquierda muestra aliasing, mientras que en la derecha se ha aplicado antialiasing para ocultar el defecto.

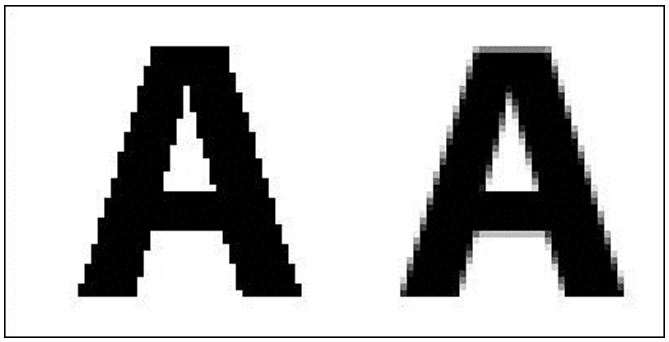


Figura B.1: Ejemplo de antialiasing

En el menú de configuración del juego, se ha creado una opción para activar esta técnica. Es un *dropdown* con tres valores, On, Off, Default. Para ello se han especificado tres métodos diferentes para aplicar *antialising* en función del usuario.

La clase *Graphics2D*[11] se puede usar para especificar si se desea que los objetos se rendericen[12] lo más rápido posible o si se prefiere que la calidad de renderizado sea lo más alta posible.

Para establecer o cambiar los atributos de renderizado, hay que construir un objeto *RenderingHints*[13](similar a un HashMap). Si se desea cambiar alguno de estos atributos,

[11] https://docs.oracle.com/javase/7/docs/api/java/awt/Graphics2D.html
[12] https://es.wikipedia.org/wiki/Renderizaci%C3%B3n
[13] https://docs.oracle.com/javase/tutorial/2d/advanced/quality.html

hay que especificar la clave-valor para el atributo correspondiente. A continuación (Figura B.2), se muestran los tipos de atributos.

Hint	Key	Values
Antialiasing	KEY_ANTIALIASING	VALUE_ANTIALIAS_ON VALUE_ANTIALIAS_OFF VALUE_ANTIALIAS_DEFAULT
Alpha Interpolation	KEY_ALPHA_INTERPOLATION	VALUE_ALPHA_INTERPOLATION_QUALITY VALUE_ALPHA_INTERPOLATION_SPEED VALUE_ALPHA_INTERPOLATION_DEFAULT
Color Rendering	KEY_COLOR_RENDERING	VALUE_COLOR_RENDER_QUALITY VALUE_COLOR_RENDER_SPEED VALUE_COLOR_RENDER_DEFAULT
Dithering	KEY_DITHERING	VALUE_DITHER_DISABLE VALUE_DITHER_ENABLE VALUE_DITHER_DEFAULT
Fractional Text Metrics	KEY_FRACTIONALMETRICS	VALUE_FRACTIONALMETRICS_ON VALUE_FRACTIONALMETRICS_OFF VALUE_FRACTIONALMETRICS_DEFAULT
Image Interpolation	KEY_INTERPOLATION	VALUE_INTERPOLATION_BICUBIC VALUE_INTERPOLATION_BILINEAR VALUE_INTERPOLATION_NEAREST_NEIGHBOR
Rendering	KEY_RENDERING	VALUE_RENDER_QUALITY VALUE_RENDER_SPEED VALUE_RENDER_DEFAULT
Stroke Normalization Control	KEY_STROKE_CONTROL	VALUE_STROKE_NORMALIZE VALUE_STROKE_DEFAULT VALUE_STROKE_PURE
Text Antialiasing	KEY_TEXT_ANTIALIASING	VALUE_TEXT_ANTIALIAS_ON VALUE_TEXT_ANTIALIAS_OFF VALUE_TEXT_ANTIALIAS_DEFAULT VALUE_TEXT_ANTIALIAS_GASP

Figura B.2: Atributos de renderizado

Como se puede observar en la figura anterior, Java nos permite establecer tres grados diferentes de antialiasing, activado, desactivado o una opción por defecto, que resulta una solución intermedia entre las anteriores. Antes de pintar los objetos del juego, se comprueba qué opción ha elegido el usuario y se establece las opciones de renderizado. El juego por defecto tiene desactivado el *antialising*.

B.3. Programación de tanques

El editor está formado por un *JTextArea*[14], donde se escriben todas las instrucciones, asociado a un *JScrollPanel*[15] que permite poner barras de scroll al componente anterior cuando el programa tiene un tamaño elevado. Además, se ha implementado otra clase *TextLineNumber*[16], que permite toda la lógica para mostrar los números de línea. Este componente tiene que usar la misma altura para cada línea y es parametrizable, se puede

[14] https://docs.oracle.com/javase/7/docs/api/javax/swing/JTextArea.html
[15] https://docs.oracle.com/javase/7/docs/api/javax/swing/JScrollPane.html
[16] https://tips4java.wordpress.com/2009/05/23/text-component-line-number/

especificar la alineación horizontal de los dígitos, a la izquierda, centrado o a la derecha. Remarcándose en rojo el número de la línea en la que se encuentra el cursor.

El editor es sensible al contexto y sugiere un listado de las instrucciones más parecidas con el comienzo de la palabra que el usuario escribe. Para ello se ha creado una bolsa de palabras, con todas las instrucciones que el lenguaje permite. Cuando el documento cambia, hay un Listener[17] para leer el contenido actual de la última instrucción y buscar qué palabras de nuestra bolsa comienzan con ese prefijo. Si hay match entonces se lanza un thread que te sugiere las instrucciones deseadas. Se puede navegar por ese listado con las flechas (como en cualquier otro editor), al pulsar la tecla Enter, se escribe la instrucción completa seleccionada por el usuario. Uno de los problemas principales al implementar esta parte, fue el foco entre componentes [15], cuando mostraba la lista de instrucciones sugeridas, el foco recaía sobre el editor de texto (no sobre la lista) entonces el usuario no podía moverse con las flechas por el listado, ya que lo único que se conseguía era mover el cursor del editor. Sólo podía seleccionar la instrucción clicando con el ratón, siendo algo incómodo a la hora de programar. Para solventarlo, se añadieron unos *KeyListener*[18] mapeados a las flechas del teclado, si se pulsaba la tecla Abajo o Arriba y el componente de la lista era visible entonces se cambiaba el foco al listado, en caso contrario, lo seguía manteniendo el editor. Cuando seleccionas una de las instrucciones entonces el foco vuelve al editor.

Por otro lado, mientras escribes, la aplicación muestra una breve descripción de la instrucción sugerida. Para colocar correctamente esta descripción al lado del listado, se tuvo que mapear la posición del cursor del documento al sistema de coordenadas de la pantalla [16].

El editor ofrece la posibilidad de añadir una descripción al tanque que se está programando (Figura B.3). Está limitado a 70 caracteres para que sea una descripción breve y concisa de los puntos fuertes que tendrá el tanque (aunque el usuario puede escribir lo que desee).

[17] https://docs.oracle.com/javase/tutorial/uiswing/events/intro.html
[18] https://docs.oracle.com/javase/tutorial/uiswing/events/keylistener.html

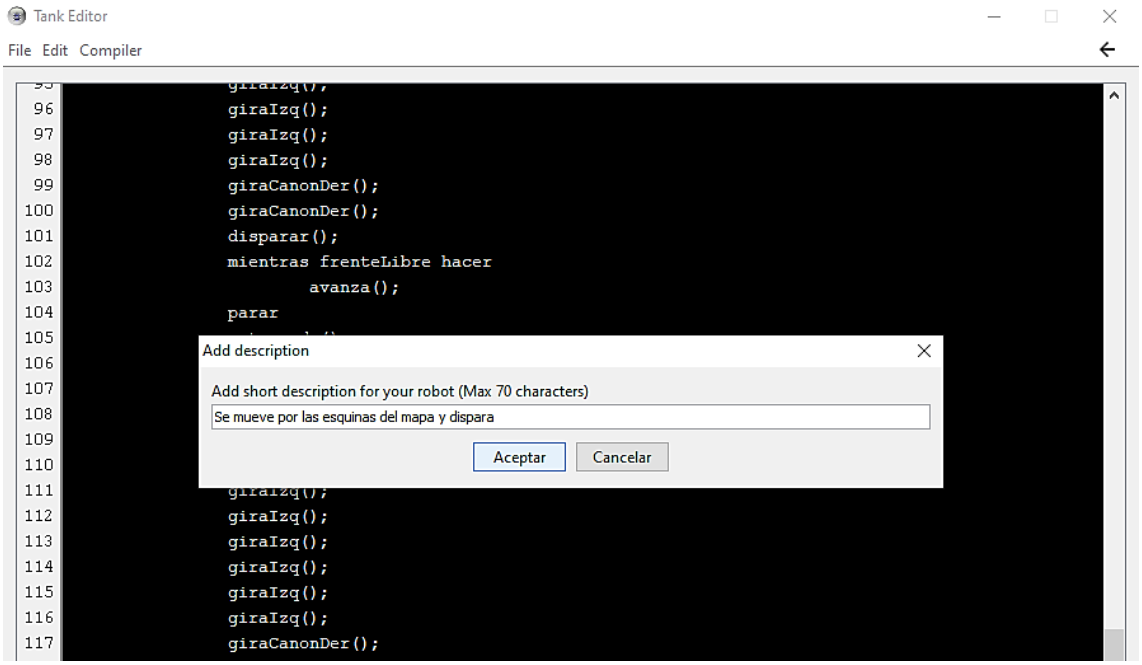


Figura B.3: Añadir descripción

Así cuando el jugador quiera seleccionar un tanque en la ventana previa a la batalla, podrá visualizarla y decidirse entre uno u otro en función de esa información.

Además, el editor ofrece todas las funciones principales (deshacer, rehacer, cortar, copiar, pegar, seleccionar todo), todas ellas con atajos de teclado. De esta forma, se ofrece una programación más fluida e intuitiva para el jugador.

En el siguiente apartado, se detallará la gestión de los programas creados por el usuario.

B.3.1. Gestión de ficheros

Cuando se lanza el juego por primera vez, pregunta dónde se quiere guardar los programas que escribirá el usuario (Figura B.4). Por defecto, se instalan en el disco local C: y crea una carpeta denominada *myRobots*. Pero el usuario puede cambiar esta ruta a la que desee. Esta carpeta generada por el videojuego, será persistente para todas las ejecuciones posteriores que internamente guardará información relevante como la ruta que ha seleccionado el usuario previamente (es importante guardarla porque el usuario podrá modificarla desde el juego como se detallará en este apartado posteriormente), además de las carpetas de ejemplos, tanto de tanques como de batallas.

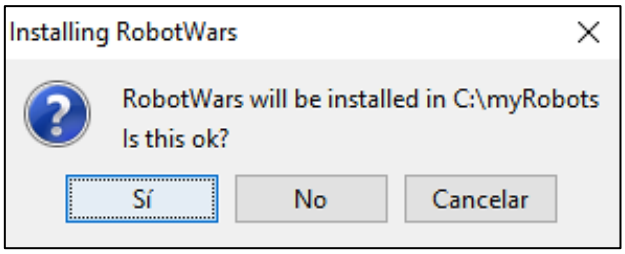


Figura B.4: Mensaje al arrancar el juego.

El juego tiene una carpeta de tanques de ejemplo para que el usuario se familiarice con el lenguaje y vea las mecánicas del juego antes de empezar a programar de cero. Cada tanque, conlleva un fichero asociado, que es su programa, el comportamiento implementado por el usuario en el editor de texto. La carpeta de ejemplos se añadió dentro del workspace del proyecto de Eclipse, de esta forma, cuando se lanzaba el juego desde Eclipse, leía estos ficheros del workspace y los creaba en la carpeta que el usuario hubiera indicado. El problema ocurre cuando se exporta el proyecto a un archivo *.jar*[19], este archivo puede haber sido exportado a cualquier ruta del sistema de ficheros de la plataforma utilizada y, por tanto, no sabe dónde buscar los ficheros de ejemplo. Para ello, cada vez que arranca el juego, se mueve la carpeta de ejemplos del workspace al directorio *Temp*, que es donde se crean todos los archivos temporales, y de ahí, leer todos los ficheros y moverlos a la ruta establecida por el usuario. Cuando el juego se cierra, entonces la carpeta es borrada en el directorio de temporales y no deja rastro en la plataforma, así que se consideró una solución bastante limpia.

Una vez que el usuario empiece a programar, puede optar por crear un nuevo programa de cero o abrir uno ya existente. Cada vez que se empiece a programar un nuevo tanque, se crea un fichero *.txt* en la ruta que indicó el jugador al arrancar el juego por primera vez. Será persistente durante todas las ejecuciones posteriores del juego. Si se desea abrir un programa existente, se mostrará un navegador de archivos (Figura B.5), que permitirá al usuario seleccionar el fichero correspondiente. La aplicación guarda el último directorio visitado para no tener que empezar desde la raíz del directorio siempre, de esta forma, mejoramos la usabilidad de la aplicación.

[19] https://es.wikipedia.org/wiki/Java_Archive

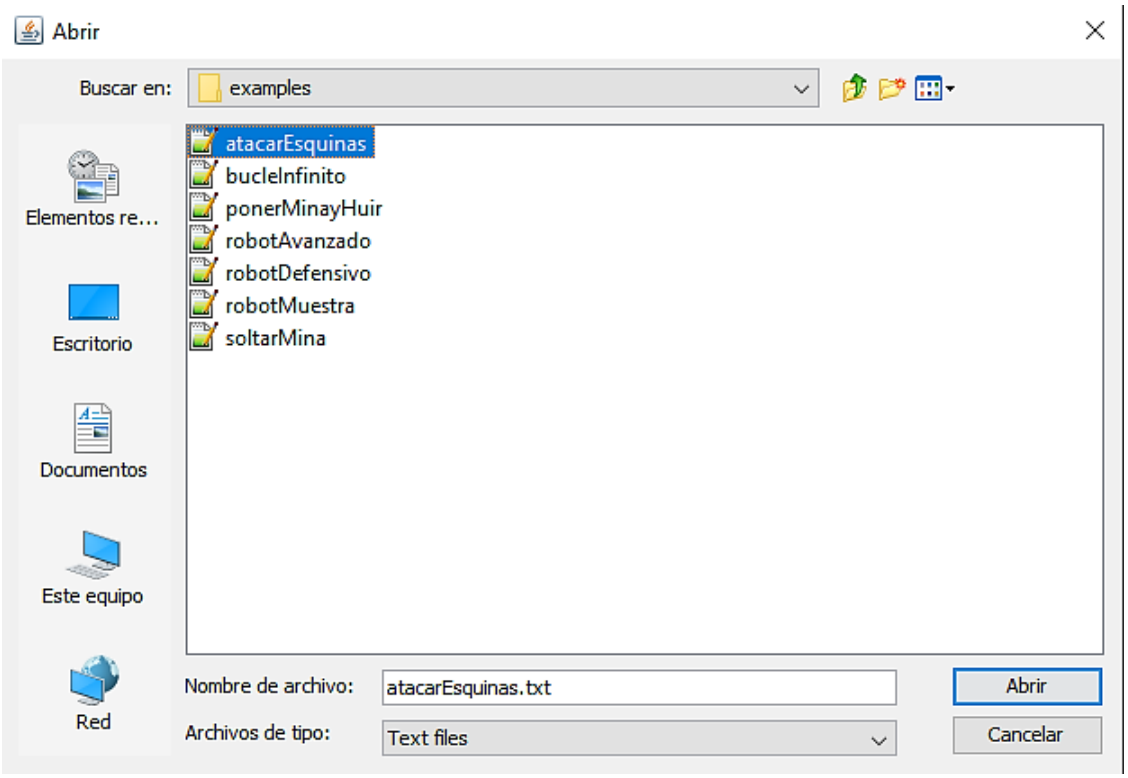


Figura B.5: Navegador de archivos

Por otra parte, el juego proporciona la posibilidad de cambiar la ruta de guardado desde el menú de configuración. Desde ahí, se pueden añadir directorios y habilitar el que se desee. Al cambiar la ruta de guardado, la aplicación realiza un volcado de todos los ficheros y directorios de la ruta origen a la ruta destino. En un principio, se hacía una operación de mover todos los ficheros, pero en determinados casos, ocurría un error que indicaba que ese fichero estaba siendo usado por otro proceso. Para solventar este problema se decidió hacer una copia de todos los ficheros a la ruta nueva y después borrar los ficheros de la ruta origen, de esta forma, obtenemos el mismo resultado, pero sin ocasionar el error mencionado anteriormente.

B.3.2. Gestión de batallas

Al igual que los tanques, el juego tiene una carpeta de batallas de ejemplo para que el usuario pueda observar cómo se comportan los tanques desde un principio antes de empezar a programar. Desde el menú principal, se puede crear una batalla nueva o abrir

una ya existente, en este último caso, se abrirá un navegador de archivos (Figura B.6) que permitirá al usuario seleccionar la batalla correspondiente.

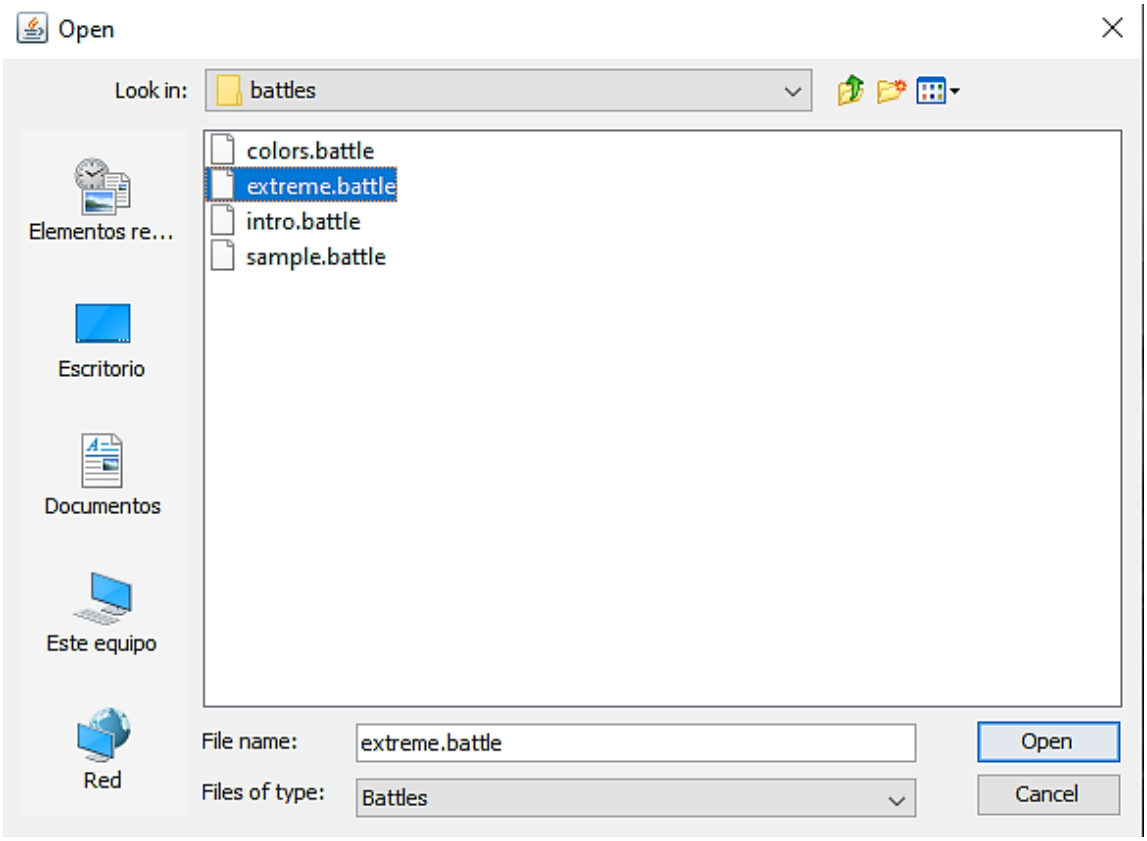


Figura B.6: Abrir batalla

Por otra parte, mientras una batalla está en juego, se puede guardar la misma. El navegador es equivalente al mostrado anteriormente. Este guardado genera un fichero *.battle* en la carpeta de batallas. Se necesitan siete parámetros para guardar una batalla: el nombre de los tanques en juego, los colores asociados a cada uno (índices), el equipo al que pertenecen en el caso que la partida sea Team Deathmatch (si es All vs All, el parámetro es -1), el número de equipos (si es All vs All, el parámetro es 0), el tipo de partida (All vs All / Team Deathmatch), el modo de juego (Free Battle / Time Trial) y el tiempo en el caso de que la batalla sea Time Trial (si es Free Battle, el parámetro es 0). Con estos parámetros, podremos recuperar la información pertinente a la hora de ejecutar una batalla. Otro aspecto importante es saber si la batalla que está en juego ha sido ya guardada previamente o no, ya que, si se pulsa el botón de Save y la batalla no ha sido guardada, se abrirá el navegador de archivos para que el usuario escriba el nombre que desea ponerle. Por el contrario, si la batalla ya ha sido guardada anteriormente, el juego realiza el guardado sin preguntar el nombre. Para ello, hay que comprobar que los

nombres de los tanques de la batalla en ejecución no estén en ningún fichero de batallas, al igual que los colores asociados a los tanques.

B.3.3. Gestión de threads

Cada tanque tiene que ejecutarse simultáneamente y de forma independiente para poder llevar a cabo las batallas. Por tanto, cada tanque es un thread que necesita leer e interpretar su programa asociado. Es importante gestionar correctamente la ejecución de las instrucciones. Los ordenadores actuales tardan nanosegundos en ejecutar cada línea de código, por tanto, leer las instrucciones asociadas al tanque y ejecutarlas ocurrían excesivamente rápido, no daba tiempo al ojo humano a visualizar correctamente lo que se había programado y los movimientos eran erráticos. Para ello, se decidió añadir un *sleep* después de cada instrucción [17], el tiempo que se interrumpe el thread hay que ajustarlo bien para que proporcione unos movimientos reales, pero sin ocasionar problemas de rendimiento.

Cuando el jugador decide empezar una nueva batalla con los tanques seleccionados, la aplicación recorre la lista de tanques y empieza su ejecución. Mientras se desarrolla la batalla, el usuario puede pulsar el botón Stop para pararla, entonces la aplicación recorre la lista de threads y detiene su ejecución, al mismo tiempo, vacía ésta lista para que no queden esos tanques para futuras ejecuciones. Ocurre lo mismo cuando una batalla termina porque uno de los tanques haya salido victorioso.

Además de los tanques existe otro thread, el que se encarga de la música del juego y los sonidos FX. En el menú de configuración, hay un apartado de sonido para ajustar el volumen del juego, activar o desactivar la música o los sonidos FX. En este punto, se detectó un problema, la desactivación de la música no era instantánea, es decir, cuando el usuario pulsaba la casilla para desactivarla, el sonido tardaba varios segundos hasta que desaparecía, se detectaba un *delay*. En ese momento, se usaba la interfaz *Clip*[20] que era la más usada en la comunidad de videojuegos en Java, pero se tuvo que investigar en una clase que ofreciera una mejora de rendimiento en cuanto a las acciones del usuario.

Un *delay* de 10 a 100 milisegundos es aceptable para audio en tiempo real. En todos los sistemas con audio existen latencias (Figura B.7), las latencias muy bajas no

[20] https://docs.oracle.com/javase/7/docs/api/javax/sound/sampled/Clip.html

funcionarán en todas las plataformas y más de 100 milisegundos probablemente sean molestos para el usuario.

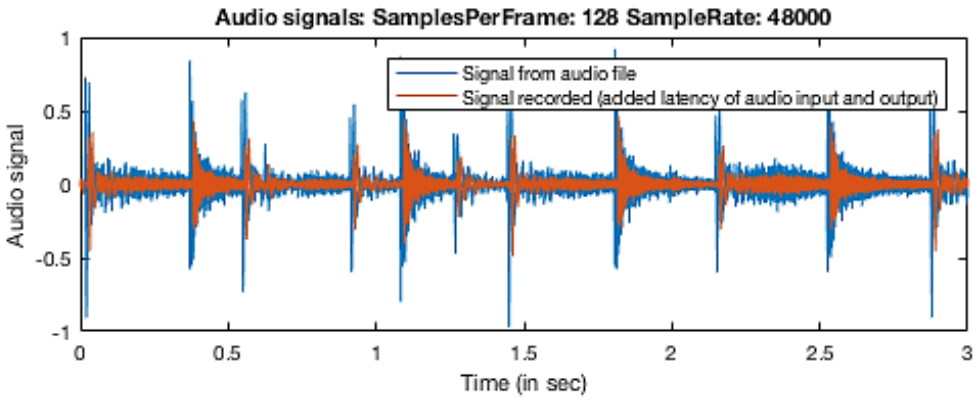


Figura B.7: Latencia en una señal de audio

Para ello, se usó la clase *AudioInputStream*[21], la cual permite gestionar al programador, la longitud de bytes a leer de los datos de audio de entrada y el formato que se desea [18]. Permite gestionar el sonido a más bajo nivel, con esta clase, lo importante es ajustar el número de bytes que se leen en cada fotograma, para obtener un equilibrio entre eficiencia y rendimiento [19]. Con esta mejora, se obtuvieron unos resultados positivos y la desactivación de la música pasó de tener un *delay* de varios segundos a ser prácticamente instantánea.

Hay que recalcar que lo mencionado anteriormente y la gestión de los sonidos de los tanques se realiza mediante un thread. Se encarga de leer la entrada de datos y escribirlo por el buffer de salida que conllevará la reproducción del sonido. Cuando el thread termina de reproducir la música, comprueba en qué estado del juego está, dependiendo de ello, vuelve a empezar o se detiene la ejecución.

B.4. Compilador

El proceso comienza cuando el usuario selecciona la opción Compile (Figura B.8) en el editor de texto. Primero comprueba si el programa ha sufrido alguna modificación desde que se abrió, si es así, muestra un mensaje preguntando al jugador si desea guardarlo. El programa siempre tiene que estar guardado antes de compilar.

[21] https://docs.oracle.com/javase/7/docs/api/javax/sound/sampled/AudioInputStream.html

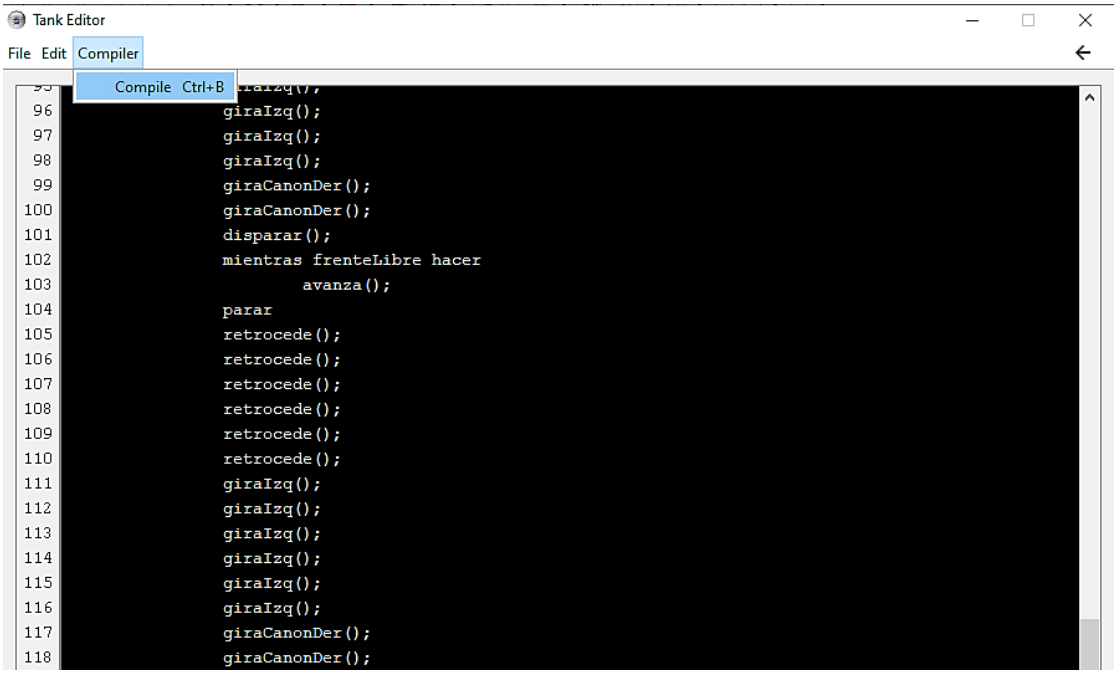


Figura B.8: Compilar programa.

En este proceso, se notificará al editor si la compilación ha sido exitosa o no, es decir, si el programa escrito por el usuario es correcto sintácticamente, siendo éste el que muestre un mensaje al usuario. En un principio, los errores mostrados eran los propios de JavaCC, pero se observó que algunos mensajes eran demasiado crípticos (hay que pensar en todos los usuarios potenciales), sin tener conocimientos de informática podían resultar difíciles de entender.

En primer lugar, hay que rescatar el error de JavaCC y transformar al lenguaje natural algunas directrices. Por ejemplo, en algunos mensajes, se escribía <EOF> para indicar fin de fichero, por lo que se pasó a traducir al lenguaje natural. Lo mismo con otro tipo de símbolos ASCII, como \n, \t, etc… También se añadió algo más de información a los mensajes en ciertos casos, por ejemplo, si faltaba por cerrar un bucle o una condición, el mensaje lo notifica expresamente (anteriormente no era así), si falta algún paréntesis a la instrucción (ya sea de apertura o de cierre), si falta cerrar el programa (con la instrucción "end"). El editor también marca en rojo la línea donde sucede el error para facilitar al jugador la visualización del posible fallo. Además, cuando el error es léxico, la aplicación te sugiere la lista de instrucciones que más se asemeje a la palabra que haya escrito el usuario erróneamente. Hay que facilitar al máximo la programación al usuario teniendo en cuenta que no todos van a tener conocimientos informáticos. Cuando la compilación ha sido exitosa, se genera un fichero *.code* con el mismo nombre que el usuario haya

escrito para ese tanque, de esta forma la aplicación tiene constancia de qué tanques están listos para usarse y cuáles no.

Una de las funciones del analizador [9] es realizar las acciones asociadas a cada token. Se va a ejemplificar con la instrucción *advance()* pero el funcionamiento es similar con el resto de instrucciones.

En nuestra gramática existe la función *invocacion_accion*, esta función recibe cuatro parámetros: el objeto tanque, un booleano para saber si estamos analizando el programa o estamos en batalla, otro booleano para indicar si estamos dentro de una condición que ha sido cierta o falsa y una cadena de caracteres indicando cual ha sido la instrucción anterior (si la hay). Cuando el analizador detecta la instrucción *advance*(), hay que comprobar primero si estamos en proceso de compilación o si estamos en batalla y necesitamos ejecutar las acciones asociadas. En este caso, vamos a detallar las acciones que se realizan cuando se empieza una batalla. Después hay que comprobar si la instrucción anterior ha sido una condición o un bucle y si ha sido cierta o no. Para ello, nuestra función *expresion*, devuelve cierto o falso en función de la condición que se ha llevado a cabo, este parámetro se va pasando de forma recursiva al resto de funciones. Si ha sido falso, saltamos esta instrucción y no se realizan las acciones. Si por el contrario es cierto, se llama a la función avanzar del tanque, que se encarga de calcular la nueva posición y realizar el movimiento. Finalmente, se duerme el thread durante un tiempo como ya se comentó (apartado B.3.2.).

Este comportamiento se repite con el resto de las instrucciones. Se llamará a la función del tanque correspondiente dependiendo de la instrucción que se haya leído.

C. Manual de usuario

C.1. Requisitos del sistema

Para ejecutar correctamente RobotWars, debe instalarse Java 8 Standard Edition (SE) o una versión más reciente de Java en su sistema. Se pueden utilizar tanto Java Runtime Environment (JRE) como Java Developer Kit (JDK). Hay que tener en cuenta que el JRE no incluye el compilador estándar de Java (javac), pero el JDK sí. Por lo tanto, es suficiente ejecutar RobotWars en el JRE.

El espacio mínimo que se requiere en disco para la correcta ejecución es de 140 MB. La carpeta generada por el juego, no está considerada en el requisito anterior ya que es mínimo el espacio ocupado (unos 8 KB).

Durante el juego se puede cambiar la resolución. Las resoluciones elegidas son las siguientes:

- **Ventana pequeña**: 900x650
- **Ventana media**: 1280x720
- **Ventana grande**: 1920x1080

Requisitos mínimos:

- SO: Windows 7, 8, 8.1, 10
- Procesador: Intel Core i3 2.80GHz
- Memoria: 2GB de RAM
- Tarjeta Gráfica: NVIDIA GT 340 / ATi Radeon HD 5550
- DirectX: Versión 11

Requisitos recomendados:

- SO: Windows 10
- Procesador: Intel Core i5 3.00GHz
- Memoria: 4GB de RAM
- Tarjeta Gráfica: NVIDIA GTX 770 / AMD R9 280X
- DirectX: Versión 11

C.2. Guía de ayuda

Una vez se ha iniciado el videojuego, esta es la primera pantalla que se muestra (Figura C.1)

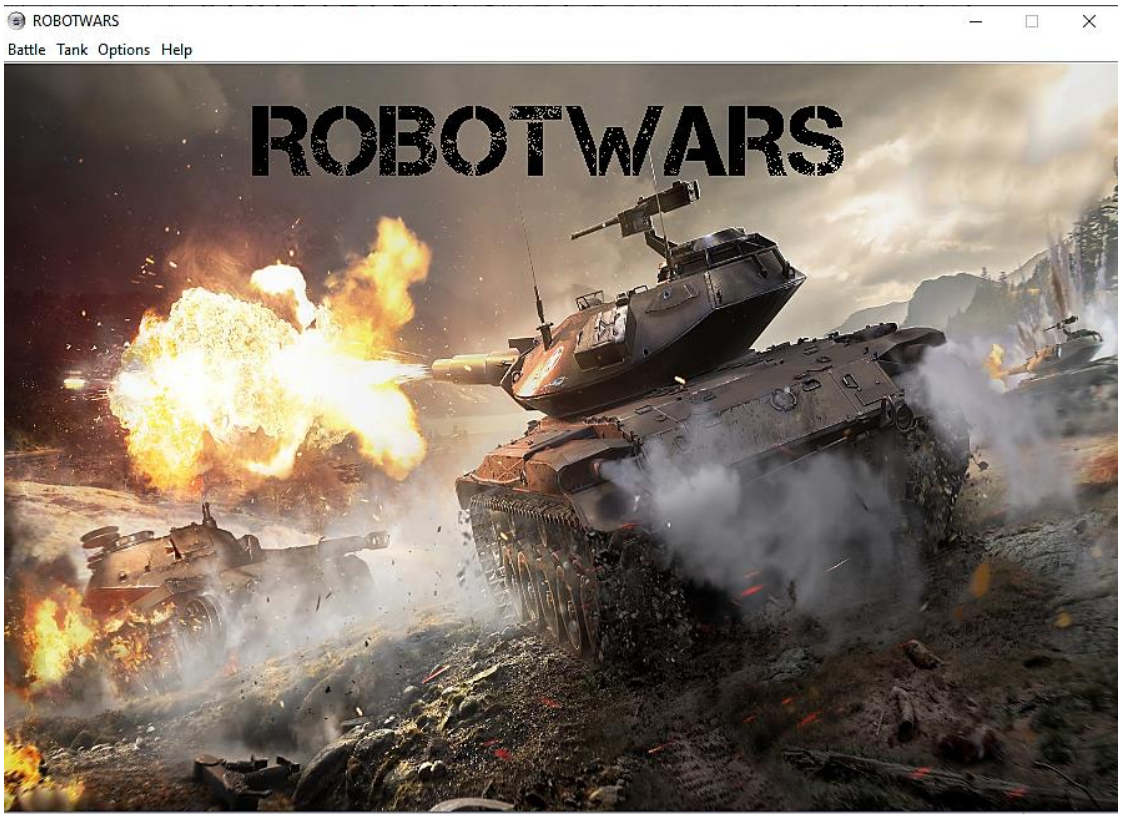


Figura C.1: Ventana principal

Desde la ventana principal se puede acceder a todas las posibilidades de RobotWars. La pantalla está compuesta por cuatro opciones dispuestas sobre la barra de menú.

Desde la opción *Battle*, se puede acceder a la ventana de selección de tanques mediante el submenú *New Battle*. Si hacemos click sobre esta opción, accederemos a la ventana de selección de tanques (Figura C.2). En esta ventana, se comienza con los tanques de ejemplo que proporciona la aplicación. Nos permite elegir trece colores diferentes para el tanque seleccionado: azul, negro, azul oscuro, verde, verde claro, azul claro, rosa claro, naranja, rosa, morado, rojo, blanco y amarillo. En la parte izquierda, tendremos la lista de tanques disponibles para seleccionar en el campo de batalla. En la parte central, los botones para añadir o borrar los tanques seleccionados. En la parte derecha, tendremos la

lista de tanques seleccionados para pelear. Finalmente, los botones para empezar la batalla o cancelar y volver al menú principal.

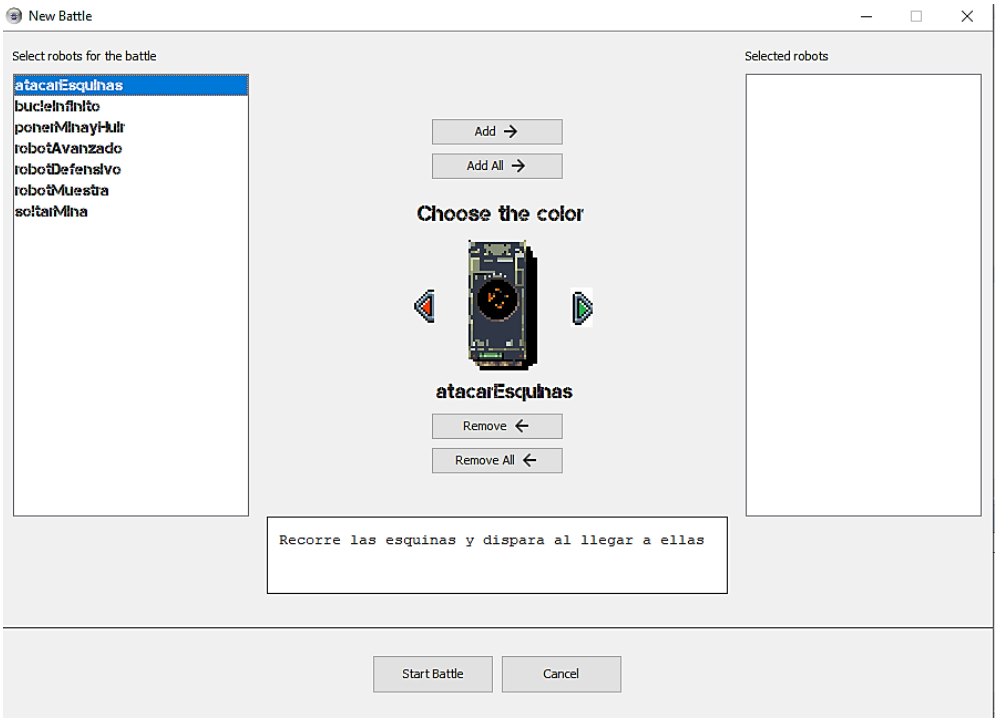


Figura C.2: Ventana de selección

Una vez, hemos elegido los tanques para pelear. Se pulsa el botón *Start Battle* y comenzará la batalla en la ventana del juego (Figura C.3).

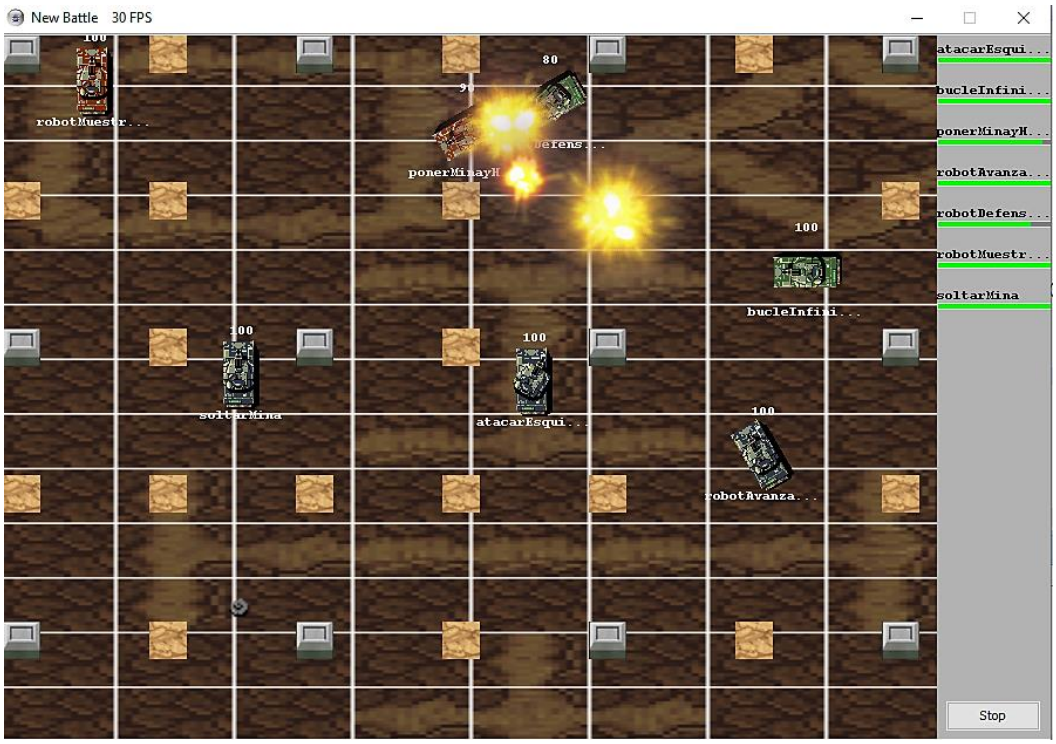


Figura C.3: Ventana de batallas

Por otra parte, desde el menú *Tank Editor*, se puede acceder a la ventana del editor de texto. Si hacemos click sobre esta opción, accederemos a la ventana del editor de texto. La ventana mostrará una imagen de fondo por defecto usada como transición.

Para crear un tanque nuevo, se hace click sobre la opción *File*, después seleccionamos *New (Figura C.4)*. En la parte superior derecha existe un botón con una flecha con la que se puede volver en cualquier momento al menú principal.

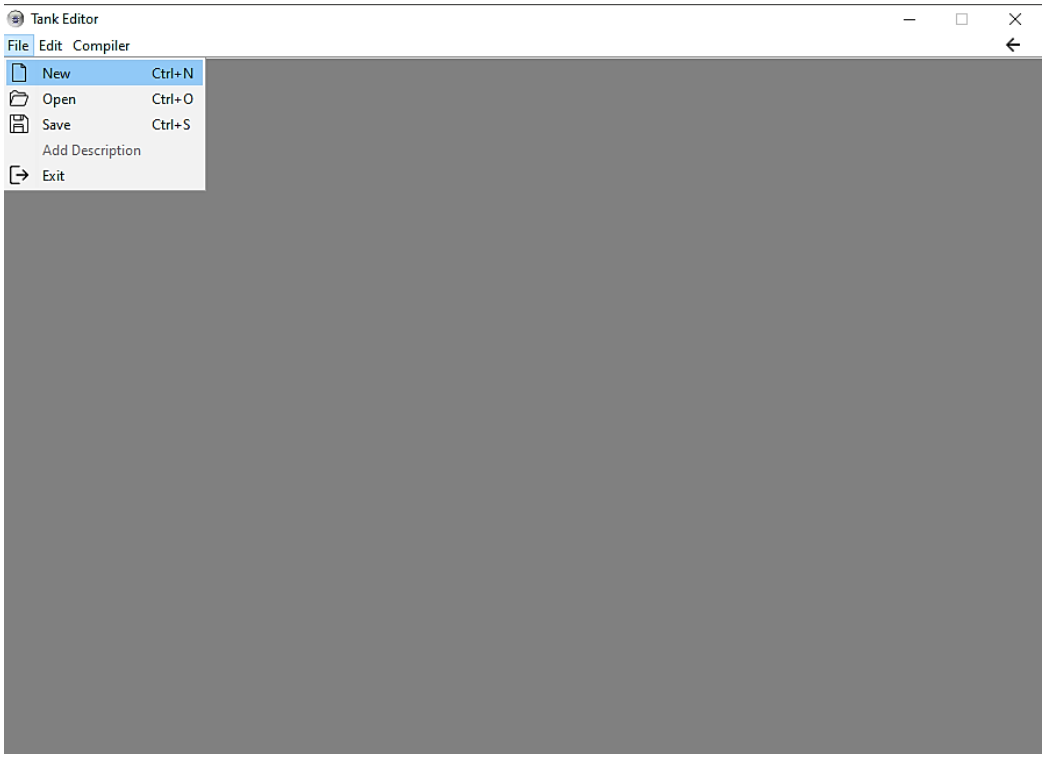


Figura C.4: Crear nuevo tanque.

En este momento nos aparecerá un dialogo para introducir el nombre del robot que vamos a crear (Figura C.5). El nombre no puede ser vacío o contener espacios, el propio input lo avisa. En caso contrario, mostrará un mensaje de advertencia indicando al usuario el problema. Inicialmente el programa asociado a este tanque tendrá el mismo nombre que introduzca el usuario.

Más adelante, se mostrará como abrir y guardar estos programas, existiendo la posibilidad de cambiar el nombre cuando se desee.

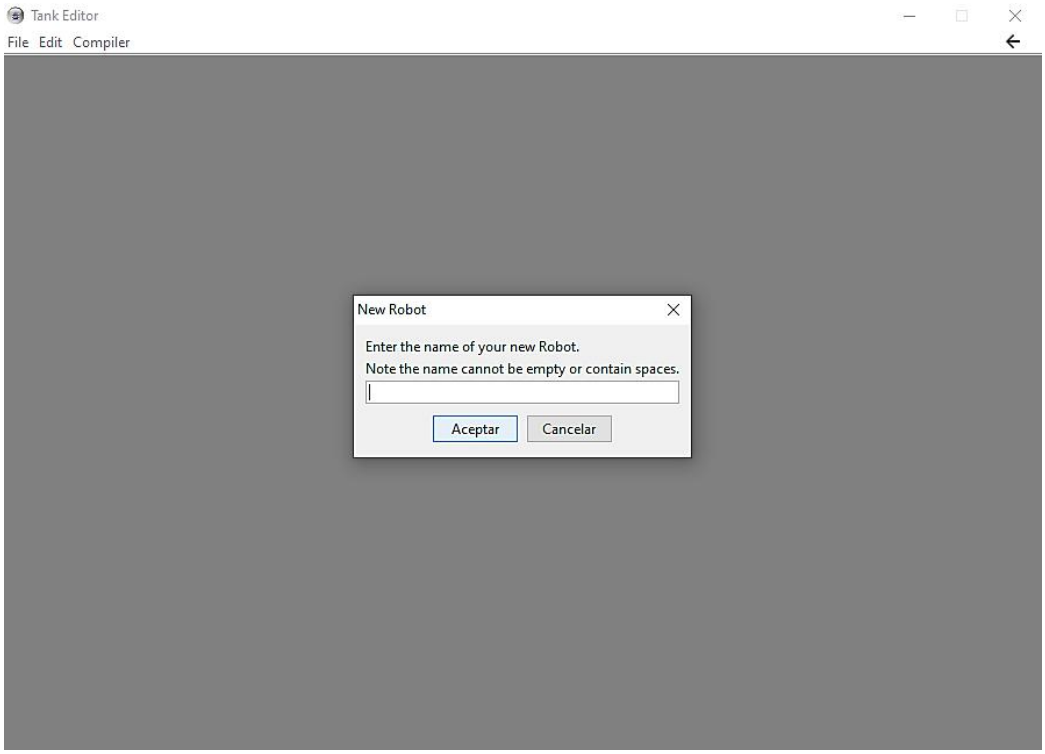


Figura C.5: Introducir nombre del robot

Al introducir el nombre, se mostrará el editor de texto para empezar a programar (Figura C.6).

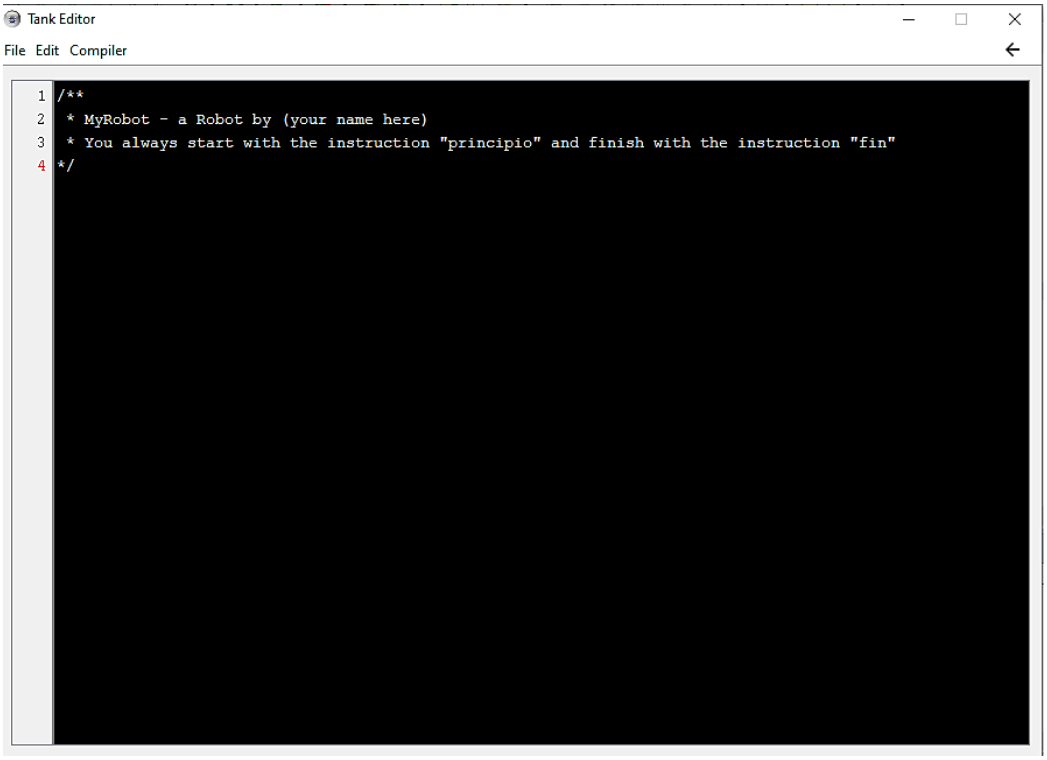


Figura C.6: Editor de texto

Aquí el jugador es cuando programa el comportamiento del tanque. Una vez cree que está terminado y desea probarlo, hay que compilarlo para saber que está escrito correctamente.

Como ya se ha comentado en apartados anteriores, es imprescindible que el programa haya sido guardado previamente para compilar. Sino ha sido así, la aplicación se encarga de preguntar si deseas guardarlo, cuando se pulsa la opción para compilar. Para ello, pulsaremos sobre la opción *Compiler*, después seleccionamos *Compile* (Figura C.7).

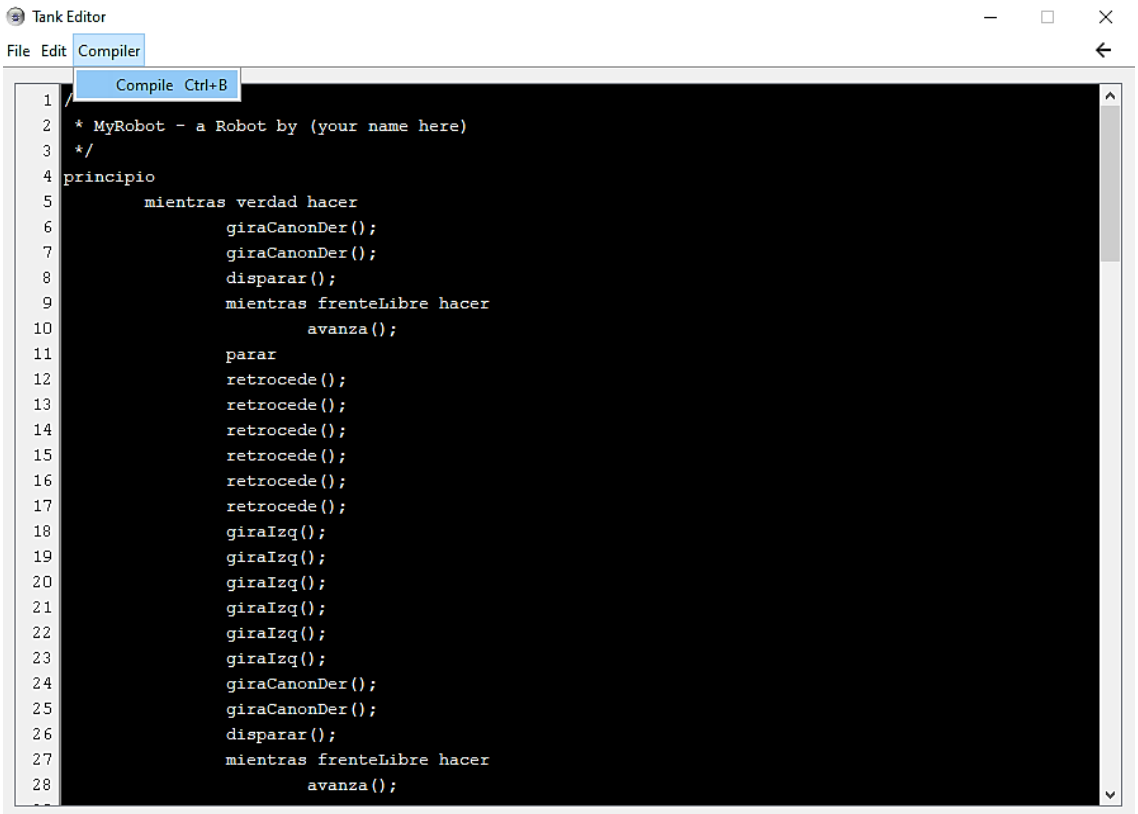


Figura C.7: Compilar programa

Como se puede observar, también existe un atajo de teclado pulsando las teclas Ctrl + B. De esta forma, se ofrece una experiencia más dinámica para jugadores con ciertos conocimientos de informática que, en principio, pueden estar más acostumbrados al uso de estos atajos. Aunque es una utilidad disponible para todo tipo de jugadores.

Todas las opciones del editor tienen asociado un atajo de teclado confiriéndole un aspecto más profesional.

El juego nos mostrará un mensaje informando si la compilación ha sido exitosa o no.

Si la compilación ha sido errónea, el mensaje intentará guiar al usuario para solucionar el problema, sugiriendo posibles instrucciones si se ha escrito mal alguna instrucción e incluso señalando en rojo el error como se ha comentado en apartados anteriores. Se mostrarán ambas capturas para ejemplificar los dos casos (Figura C.8 y C.9).

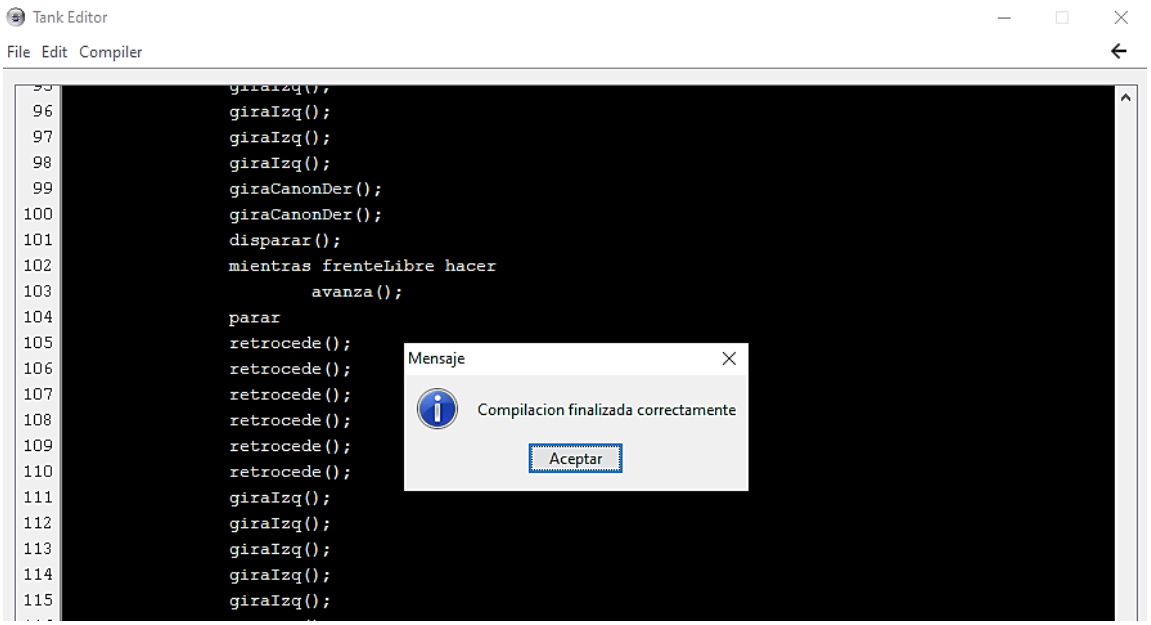


Figura C.8: Compilación exitosa

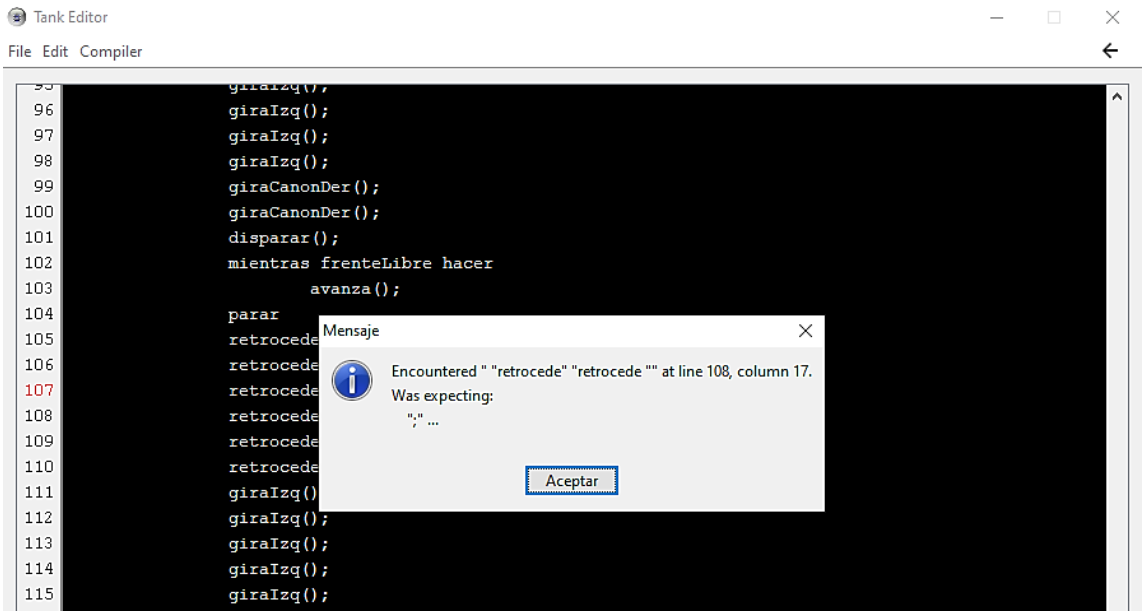


Figura C.9: Compilación fallida

Desde el editor puedes abrir tanques ya existentes, desde la opción *File*, después *Open*. Se abrirá un cuadro de diálogo para seleccionar el fichero correspondiente (Figura C.10). Para guardar es equivalente el diálogo, los programas se pueden guardar desde la opción *File*, después *Save*.

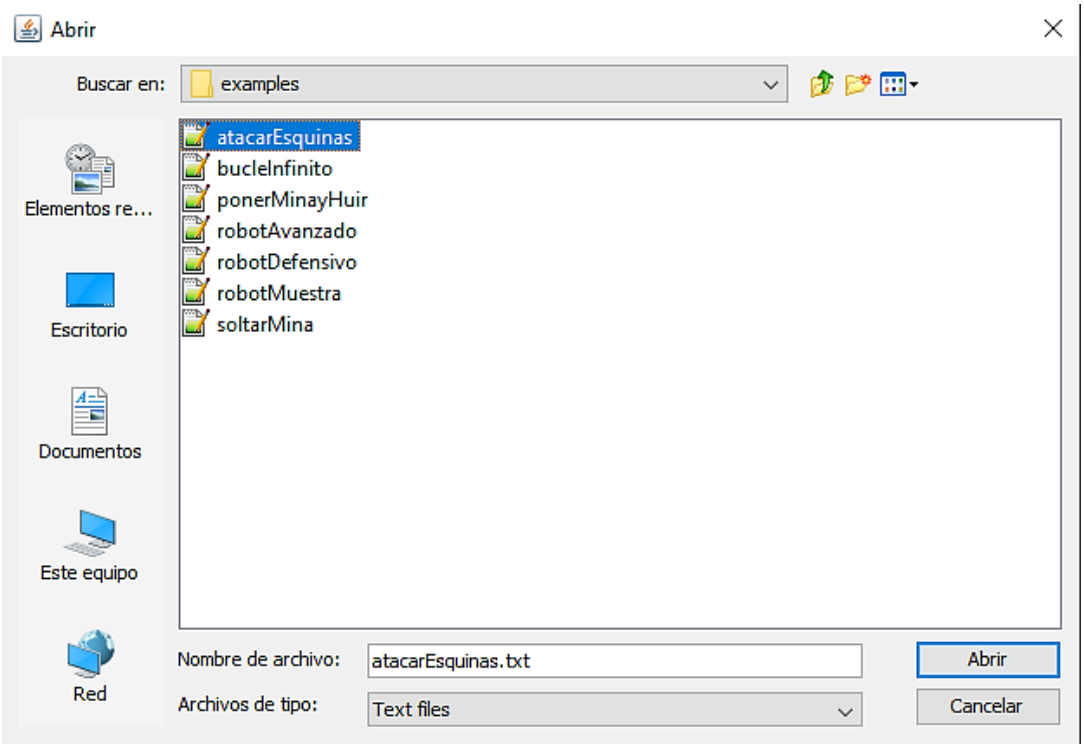


Figura C.10: Abrir fichero existente

El juego tiene un menú de configuración, por el cual se pueden cambiar diferentes características del juego. Se accede desde el menú *Settings*. La primera ventana que aparece son opciones visuales (Figura C.11), para elegir si se quiere ver el nombre de los tanques, su vida, las explosiones, etc… Por defecto, están todas las opciones activadas, para ofrecer en primera instancia una jugabilidad más atractiva al usuario. Ya que el juego ha sido optimizado en la medida de lo posible, no existe un deterioro del rendimiento por usar estas opciones. Existen también tres botones para mejorar la usabilidad de la aplicación que permiten seleccionar o deseleccionar todas las opciones pulsando uno de esos botones.

Si se desea que los cambios sean persistentes, hay que pulsar el botón *Save*. Por el contrario, si se desea volver al menú principal, hay que pulsar el botón *Cancel*.

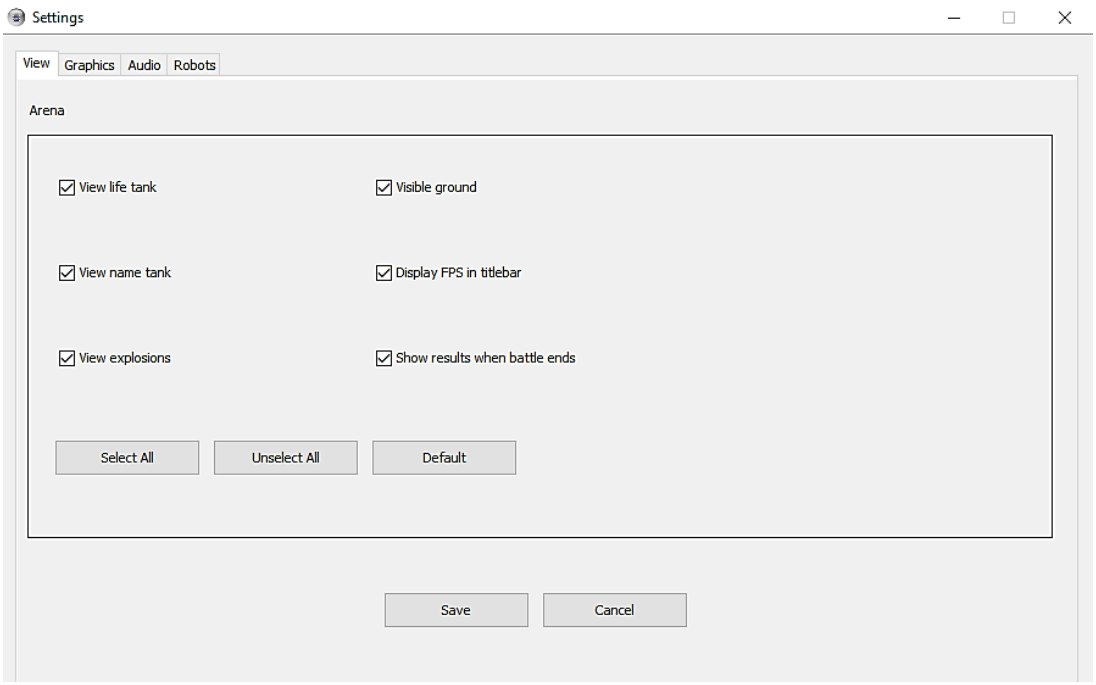


Figura C.11: Pestaña visual

La siguiente pestaña es la de gráficos (Figura C.12). Desde ahí se puede cambiar la resolución de la pantalla, los FPS y el antialiasing que se mencionó en apartados anteriores.

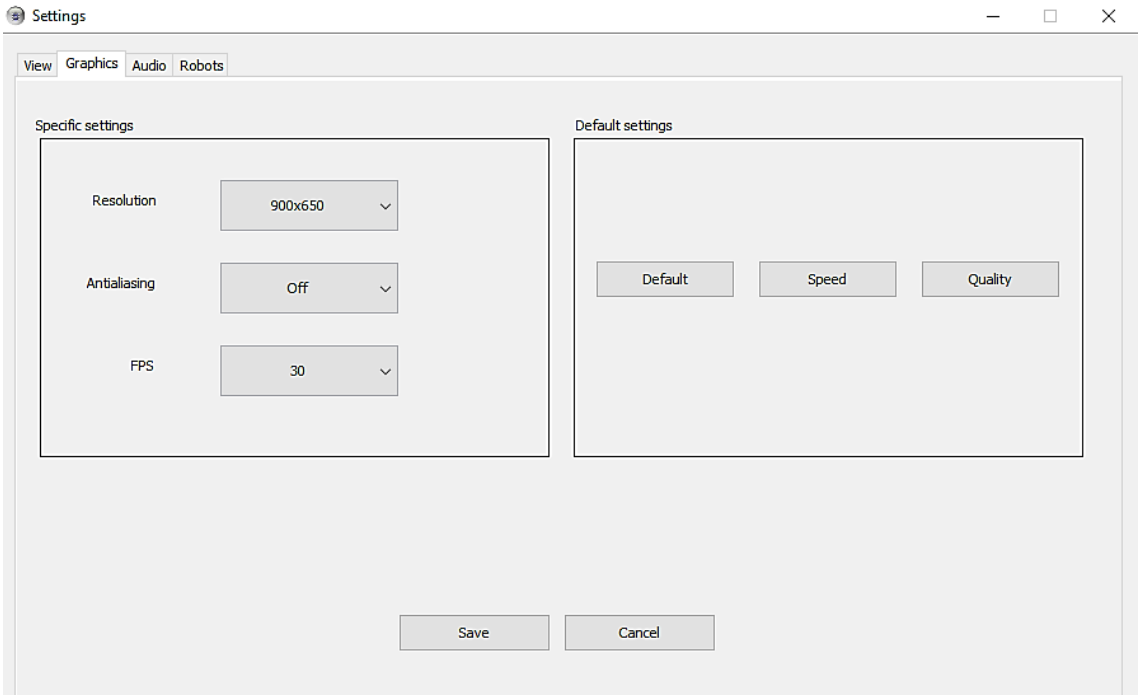


Figura C.12: Pestaña de gráficos

La aplicación también tiene una pestaña de sonido (Figura C.13) donde se puede cambiar el volumen del juego, quitarlo si se desea o quitar los sonidos FX (los pertenecientes a los tanques, como disparar o moverse).

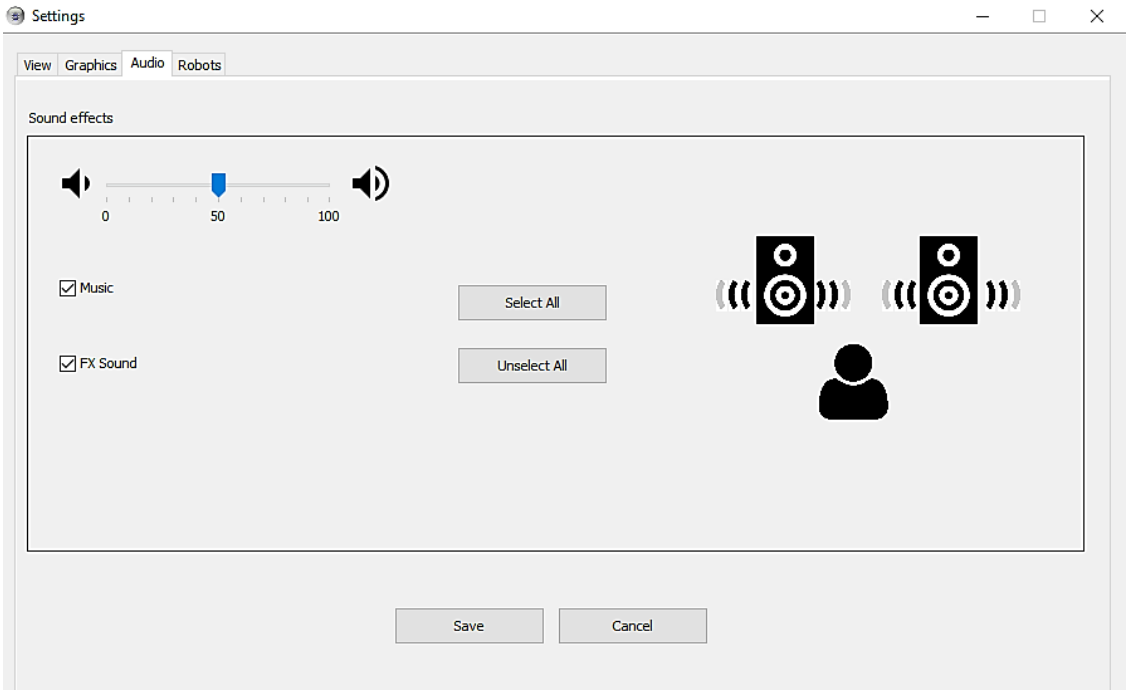


Figura C.13: Pestaña de sonido

Finalmente existe otra pestaña para cambiar la ruta donde se guardan los programas implementados por el jugador, la pestaña Robots. Por defecto se instalan en el disco local C: pero esto es modificable desde esta ventana (Figura C.14).

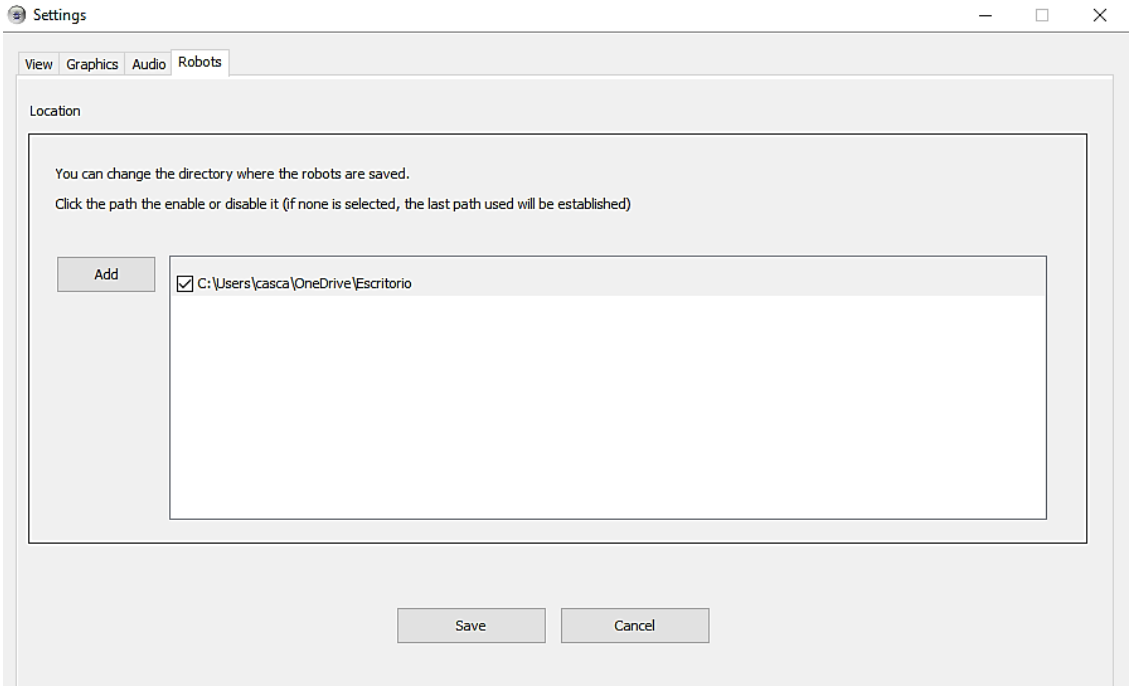


Figura C.14: Pestaña de directorio de guardado

Al hacer click sobre el botón “Add”, aparecerá un cuadro de diálogo, similar al que se usa para abrir o guardar programas (Figura C.15). Ahí se podrá seleccionar un directorio que será usado para futuras ejecuciones del juego donde se guardarán los programas.

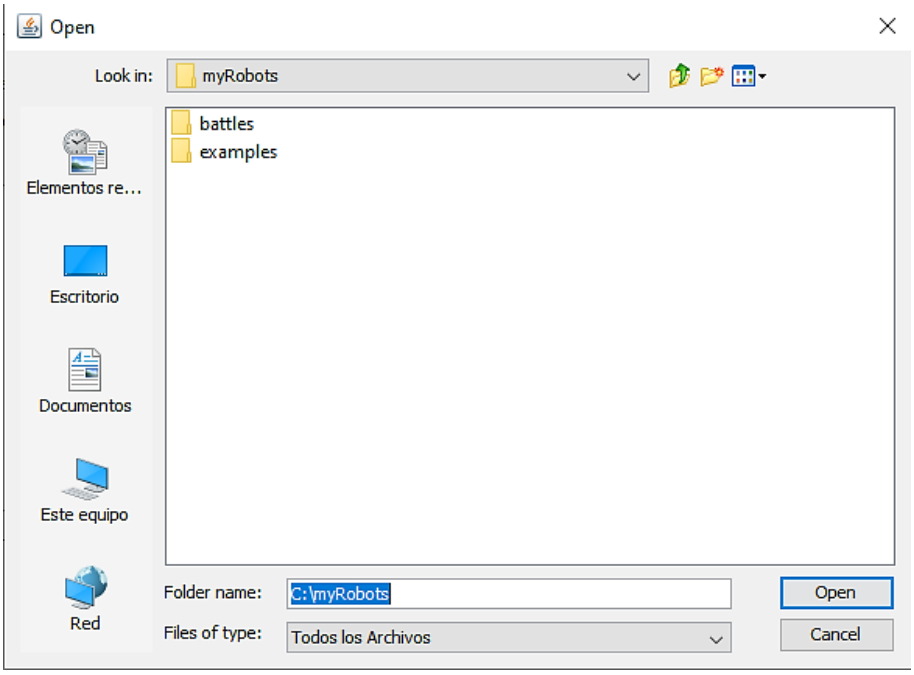


Figura C.15: Añadir directorio

Por último, el juego ofrece un manual de ayuda integrado (Figura C.16). Desde la opción *Help*, después seleccionas *User Guide*. En esta ventana se explica al usuario todo lo relacionado para comenzar a programar un tanque de cero, el repertorio de instrucciones y su sintaxis [20].

Figura C.16: Ventana de manual de ayuda

Existe otro menú, llamado *Read Me*, centrado en los requisitos de sistema para lanzar el juego correctamente (Figura C.17).

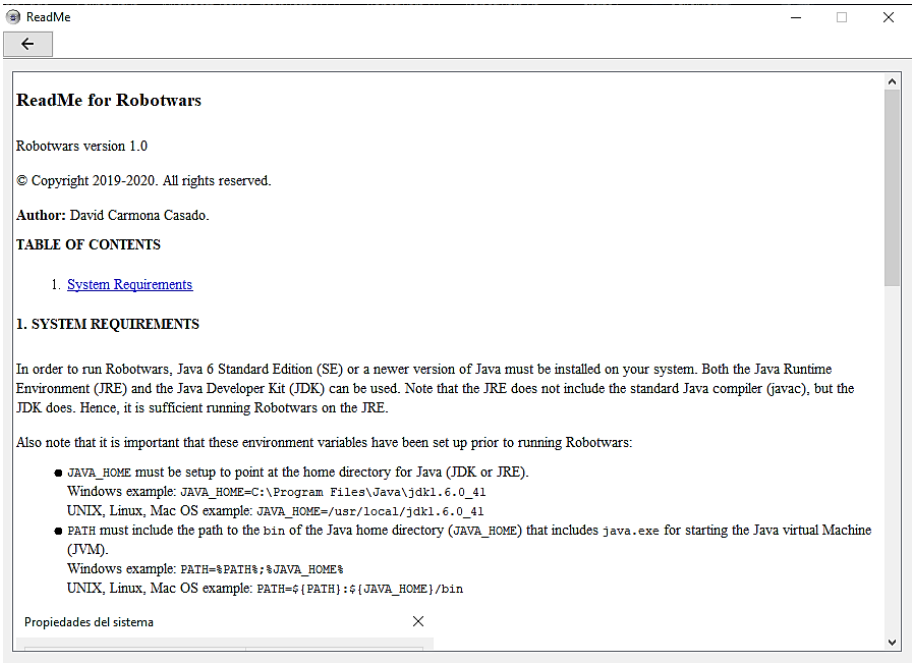


Figura C.17: Requisitos del sistema

C.3. Ejemplos de tanques

A continuación, se mostrará un tanque de ejemplo que ofrece el juego para empezar batallas si un jugador lo desea (es el tanque más complejo e inteligente). Los ejemplos más representativos tienen comentarios en el código para explicar al usuario el comportamiento de los mismos, así se facilita en la medida de lo posible la adaptación del usuario al lenguaje y al entorno.

El tanque se llama *multiFuncion* (Figura C.18, C.19, C.20, C.21). Este tanque es el más completo de los que proporciona el juego, por ende, su código es más extenso y con una complejidad algo superior. Tiene comentarios en el código para entender su funcionamiento fácilmente ya que el objetivo principal del juego es facilitar al usuario las labores de programación.

La estrategia de este tanque es la siguiente: avanza por el mapa hasta que colisiona con algún objeto (digamos que patrulla por el mapa), mientras tanto escanea con el radar minas cercanas (instrucción *scanMine*), si existe alguna intenta esquivarlas dándose la vuelta y yéndose hacia el lado contrario. Además, escanea por si hay tanques cerca (instrucción *scanTank*), si existe alguno, se posiciona hacia los grados pertinentes y

avanza hacia él mientras le dispara (una vez que detecta un tanque va a por él hasta destruirlo). Si choca con algún objeto, dispara (por si el obstáculo es un tanque enemigo) y gira hacia la izquierda siempre que pueda, sino retrocede y vuelve a intentar girar hasta que el movimiento lo permita (el bucle con la instrucción *hasHit* facilita esta labor). Si retrocediendo detecta que no puede escaparse, entonces intenta avanzar y girar, de esta forma, hacemos un tanque robusto ante cualquier situación. Al intentar escapar de esquinas, siempre escanea también minas cercanas, para evitar que explote alguna en alguno de sus movimientos de escapatoria. Vuelve a empezar con el mismo comportamiento de manera indefinida.

Como se puede apreciar, este tanque es bastante completo y se creó para que el jugador pueda visualizar qué tipos de instrucciones son más importantes y cuáles menos. Para realizar un tanque "inteligente" es imprescindible que tenga una estrategia de combate bien definida. El radar es una pieza fundamental en el juego para obtener información de todo lo que nos rodea y poder actuar ante ello.

```
5  begin
6          while true do
7                  /**
8                   * Patrol loop. It will advance until it collides with an object
9                  */
10                 while notHasHit() do
11                         advance()
12                         /**
13                          * Check if there are mines nearby to avoid them
14                         */
15                         if scanMine() <> -1 do
16                                 if scanMine() < 340 do
17                                         /**
18                                          * If I have free front I can go
19                                         */
20                                         if scanMine() > 20 do
21                                                 advance()
22                                         /**
23                                          * Else turn 180 degrees to run away
24                                         */
25                                         else
26                                                 turnRight(180)
27                                         endIf
28                                 else
29                                         turnRight(180)
30                                 endIf
31                         endIf
```

Figura C.18: Ejemplo tanque MultiFuncion (Parte 1)

```
32                      /**
33                       * Check if there are tanks nearby
34                      */
35                      if scanTank() <> -1 do
36                              /**
37                               * Depending on the position of the enemy
38                               * we position ourselves to shoot later
39                              */
40                              if scanTank() >= 45 do
41                                      if scanTank() <= 180 do
42                                              turnRight(120)
43                                      endIf
44                                      if scanTank() <= 270 do
45                                              turnLeft(120)
46                                      endIf
47                                      if scanTank() <= 350 do
48                                              turnLeft(30)
49                                      endIf
50                              else
51                                      if scanTank() >= 10 do
52                                              turnRight(30)
53                                      endIf
54                              endIf
55                              advance(5)
56                              fire()
57                      endIf
58              endWhile
```

Figura C.19: Ejemplo tanque MultiFuncion (Parte 2)

```
59              /**
60               * We come out of the loop because we have collided
61               * with something, if it's a tank, we shoot
62              */
63              if scanTank() <= 20 do
64                      fire()
65              endIf
66              if scanTank() >= 340 do
67                      fire()
68              endIf
69              /**
70               * If we don't have a tank in front, we try
71               * to turn until we get out of the corner
72              */
73              if scanTank() > 20 do
74                      if scanTank() < 340 do
75                              turnLeft(90)
76                              /**
77                               * Loop to exit any corner
78                              */
79                              while hasHit() do
80                                      if freeInFront() do
81                                              advance(5)
82                                      else
83                                              /**
84                                               * Be careful when leaving the corner
85                                               * in case we have mine behind
86                                              */
```

Figura C.20: Ejemplo tanque MultiFuncion (Parte 3)

```
94                                      endIf
95                                      turnLeft(90)
96                              endWhile
97                      endIf
98              endIf
99              if scanTank() = -1 do
100                     turnLeft(90)
101                     while hasHit() do
102                             if freeInFront() do
103                                     advance(5)
104                             else
105                                     /**
106                                              * Be careful when leaving the corner
107                                              * in case we have mine behind
108                                             */
109                                             if scanMine() >= 160 do
110                                                     if scanMine() <= 200 do
111                                                             back()
112                                                     endIf
113                                             else
114                                                     back(10)
115                                             endIf
116                             endIf
117                             turnLeft(90)
118                     endWhile
119             endIf
120     endWhile
121 end
```

Figura C.21: Ejemplo tanque MultiFuncion (Parte 4)