CoreWarUCM: Simulador Oficial

Por:
Stiven Arias Giraldo
Ricardo Sulbarán Socas
Carlos Romero García

UNIVERSIDAD COMPLUTENSE
MADRID

Grado en Desarrollo de Videojuegos
Facultad de Informática

Dirigido por

José Luis Vázquez Poletti
Juan Carlos Fabero Jiménez

Madrid, 2021-2022

Agradecimientos

Queremos agradecer a nuestros compañeros, con ellos hemos reído, hemos sufrido y hemos ido logrando año tras año objetivos y metas nuevos. También a nuestros profesores, un gran cuerpo de profesionales que velan y se preocupan por sus alumnos, siempre a nuestra disposición. Por último, agradecer a aquellas personas externas a la universidad que nos han ayudado a mejorar este proyecto, en especial a los amigos de Ricardo, quienes son profesionales en temas de diseño y nos dedicaron su tiempo a darnos consejos sobre cómo quedaría mejor nuestra aplicación.

Autorización y difusión de uso

Se autoriza a la UCM (Universidad Complutense de Madrid) a difundir y usar el trabajo realizado con fines académicos y mencionado a los autores, tanto la memoria como el código.

Stiven Arias Giraldo
Ricardo Sulbarán Socas
Carlos Romero García

Madrid, 2021-2022

Este documento esta realizado bajo licencia Creative Commons: "Reconocimiento-CompartirIgual 4.0 Internacional".

Índice general

Resumen **9**

Abstract **11**

1. Introducción **13**
1.1. Motivaciones . . . 13
1.2. Objetivos . . . 13
1.3. Estructura del documento . . . 14

1. Introduction **16**
1.1. Motivation . . . 16
1.2. Objectives . . . 16
1.3. Document structure . . . 17

2. Estado del arte **19**
2.1. CorewarUCM . . . 19
2.2. Unity . . . 19

3. Metodología y tecnología **22**
3.1. Scrum . . . 22
3.2. Plan de proyecto . . . 24
3.2.1. Documentación de las tecnologías . . . 24
3.2.2. Planteamiento inicial del proyecto . . . 24
3.2.3. Diseño de la aplicación por Scrum . . . 25
3.3. Tecnologías usadas . . . 26
3.3.1. Unity y `C#` . . . 26
3.3.2. Runtime File Browser . . . 26
3.3.3. JetBrains Rider . . . 26
3.3.4. Ares . . . 26
3.3.5. RedCode . . . 26
3.3.6. Adobe Photoshop . . . 26
3.3.7. Overleaf . . . 26

4. Diseño de la aplicación **28**
4.1. Descripción . . . 28
4.2. Puntos a tratar . . . 28
4.3. Diseño general . . . 29
4.3.1. Diseño del menú principal . . . 29
4.3.2. Diseño *In-Game* . . . 31
4.4. Ejemplo de uso . . . 33
4.4.1. Competidor . . . 33
4.4.2. Gestor de un torneo . . . 34

5. Arquitectura e implementación **37**
5.1. Capas de la aplicación 37
5.1.1. Escena de Unity:*Loader* 37
5.1.2. Escena de Unity:*MainMenu* 38
5.1.3. Escena de Unity: *GameScene* 40
5.2. Flujo de ejecución 42

6. Resultados **45**
6.1. Discusión de los resultados 45
6.2. Problemas al proyecto 46
6.3. Contribución personal 46
6.3.1. Contribuciones realizadas por el grupo 46
6.3.2. Contribuciones realizadas por Carlos 47
6.3.3. Contribuciones realizadas por Ricardo 48
6.3.4. Contribuciones realizadas por Stiven 48

7. Conclusiones y trabajo a futuro **51**
7.1. Conclusiones 51
7.2. Trabajo a futuro 51

7. Conclusions and future work **54**
7.1. Conclusions 54
7.2. Future work 54

Anexos **57**
A. Manual de usuario CorewarUCM 57

Índice de figuras **64**

Bibliografía **65**

Resumen

El objetivo principal es plantear una alternativa para el simulador **ARES** [1] mediante **Unity** para su uso en los torneos, sobre todo adaptando la interfaz y la parte gráfica a un estilo más moderno.
Por tanto, la idea principal es extraer las funcionalidades más relevantes de ARES y adaptarlas en Unity. Actualmente se encuentra anticuado y a los concursantes, principalmente jóvenes, les motiva una estética más colorida y con más efectos especiales como la que queremos plantear.

Palabras clave: Unity, CoreWar, C#, simulador, torneo, Redcode, interfaz, ARES, ensamblador, modernización

Abstract

The main objective is to make an alternative to the simulator **ARES** [1] using **Unity** for tournaments, focusing in the user interface and the graphic part, turning it into a more modern style.
Therefore, the main idea is to extract the most relevant functionalities from ARES and to adapt them with Unity. Currently, it is old and the most of the contestants are young, so they will be more motivated with a colorful aesthetic with special effects like we want to do.

Key words: Unity, CoreWar, `C#`, simulator, tournament, Redcode, interface, ARES, assembler, modernization

Capítulo 1

Introducción

1.1. Motivaciones

Durante el desarrollo de la carrera y, a lo largo de los años, nos hemos ido encontrado con multitud de tareas, materias y conocimientos nuevos que han ido enriqueciendo cada una de nuestras habilidades como desarrolladores. Esto ha sido posible también gracias a la motivación que hemos podido obtener por parte del profesorado de la universidad. Así pues, en nuestro tercer año de carrera, una de las optativas que podíamos escoger era **Ciberseguridad en videojuegos** impartida por **José Luis Vázquez-Poletti**.

Desde el primer momento de la materia ya se nos informó sobre los objetivos diversos de la asignatura, así como de las limitaciones que íbamos a sufrir en cuanto al contenido. Esto es debido al poco tiempo que tendríamos para ésta, un cuatrimestre, el equivalente a 6 créditos. El objetivo principal de la asignatura era obtener unos conocimientos básicos sobre la ciberseguridad, a partir de los cuales nosotros podríamos continuar formándonos en función de nuestras ganas de seguir en este campo.

Así pues, uno de los incentivos que recibimos a lo largo del cuatrimestre fue el torneo organizado por la universidad, **CoreWarUCM** [2], donde tanto los alumnos de la facultad como cualquier otra persona externa a la universidad pudiera participar. Dicho torneo consistiría en diseñar y crear una estrategia, basada en las reglas oficiales del torneo, para desarrollar y crear un pequeño malware que pusiese en evidencia el virus del contrincante. Ambos programas lucharían uno contra el otro a través del programa, **ARES**, el cual determina quién ha creado el virus más potente.

Cabe destacar que, los programas creados por los participantes están escritos en lenguaje ensamblador directamente en ficheros de texto (con extensión .redcode o .red), pudiendo utilizar diversas instrucciones para noquear al rival. Redcode es un lenguaje que se asemeja al lenguaje ensamblador, creado específicamente para ser interpretado por un simulador, y más simplificado que un lenguaje ensamblador condicional. Dentro de «El manual básico de CoreWar» [3] se pueden observar todos los detalles para comenzar a crear un virus para el torneo.

Como decíamos, este torneo no solo sirve para que los estudiantes se interesen más por la asignatura y por la ciberseguridad en general, sino también para aprender sobre el uso del lenguaje ensamblador y para atraer a la gente a este tipo de estudios. Además, existen premios físicos para los ganadores del torneo, lo cual siempre es llamativo y atractivo para las personas.

1.2. Objetivos

Como hemos dicho anteriormente, queremos crear un simulador moderno, llamativo y atractivo para que la gente pueda disfrutar de los combates durante el torneo organizado anualmente (**CoreWarUCM**) por la

universidad. Además, una de las claves del proyecto será transmitir de la mejor manera posible lo que está pasando durante las batallas, puesto que durante la simulación suele resultar tedioso o difícil de entender, así como crear del mismo programa un espectáculo visual.

1.3. Estructura del documento

El trabajo ha sido desarrollado a lo largo de siete capítulos más los anexos. En los capítulos se profundizarán los conceptos marcados en el resumen del siguiente índice de contenidos:

- **Capítulo 1**. Introducción. Marca el comienzo del trabajo y la motivación del mismo.
- **Capítulo 2**. Estado del arte. Se habla de las herramientas que se han utilizado, de las aplicaciones similares en las que nos hemos inspirado y las tecnologías usadas.
- **Capítulo 3**. Metodología y tecnología. Se detalla la metodología usada para llevar a cabo el proyecto y se profundiza en las tecnologías usadas.
- **Capítulo 4**. Diseño de de la aplicación. Muestra final de la aplicación, explicación de la interfaz y diseño de la misma.
- **Capítulo 5**. Arquitectura e implementación. Se explicará el funcionamiento de cada una de las escenas de Unity y sobre cómo se producen las transiciones entre ellas.
- **Capítulo 6**. Resultados. Retrospectiva del proyecto y discusión sobre los resultados finales.
- **Capítulo 7**. Conclusiones y trabajo a futuro. Resumen de los resultados y mejoras o contenido adicional que se puede incluir en el proyecto en futuras versiones.
- **Anexos A**. Manual de usuario CorewarUCM. Detalla un manual de usuario sencillo y directo para los que quieran hacer uso de la aplicación.

Los capítulos uno y siete se pueden encontrar también traducidos al inglés como se específica en la normativa de los Trabajos de Fin de Grado de la UCM.

Capítulo 1

Introduction

1.1. Motivation

During the degree, we have encountered with so many tasks and knowledge which have improved all of our developer skills. This has been possible also due to motivation from the teachers. So, in our third year of the university, one of our optional subjects was **Ciberseguridad en videojuegos** taught by **José Luis Vázquez-Poletti**.

Since the first moment of the subject we were informed about the main objectives, also about the limitations we would have related to the content. The time of the semester was short, around four months, equivalent to six credits. The main objective of the course was to obtain basics knowledge about cybersecurity, from which we could continue investigating and improving our skills.

So, one of the incentives we had through the semester was the tournament managed by the university, **CoreWarUCM** [2], where the students from the college and people from another places could join. That championship consists in designing and creating an strategy, based on the official rules of the competition, to develop and manufacture a small piece of malware that exposes the enemy's virus. Both programs fight each other inside the simulator, **ARES**, which determines who have created the best virus.

We have to mention that the programs created by participants have been written in assembly language inside text files (with .redcode or .red extension), so they can use different instructions to knock rival down. Redcode is a language similar to assembly language, created specifically to be interpreted by a simulator, and more simplified than a conditional assembly code. Inside the «El manual básico de Corewar» [3] you can see every detail about how to created a virus for the tournament.

As we said, the tournament is not just useful to make the subject and cybersecurity more interesting for the students, but also, to learn about the usages of assembly language and to engage people in this kind of studies. In addition, there are physical prizes for the winners, which is always flashy and attractive for the people.

1.2. Objectives

As we said before, we want to make a modern simulator, visible and attractive to make the people enjoy the fights during the annual competition (**CoreWarUCM**) managed by university. In addition, on of the keys of the project will be to communicate in the best way what is going on inside the battles, because the simulation could be tedious or hard to understand and also to make a visual show with the game.

1.3. Document structure

The project has been developed through seven chapters with the annexes. Into the chapters, the concepts inside the table of content will be explained deeply:

- **Chapter 1**. Introduction. It marks the beginning of the work and the motivation of it.
- **Chapter 2**. State of the art. It talks about the tools that have been used, the similar applications that we have been inspired by and the technologies used.
- **Chapter 3**. Methodology y technology. It details the methodology used to make the project and it deeps into the used techs.
- **Chapter 4**. Application design. Final sample of the application, explanation of the interface and design of the same.
- **Chapter 5**. Architecture and implementation. It will explain the functionality about Unity scenes and about the transitions between scenes.
- **Chapter 6**. Results. Retrospective of the project and discussion about the final results of the project.
- **Chapter 7**. Conclusions and future work. Resume about the results and improvement or additional content which could be included in future versions.
- **Annexes A**. CorewarUCM user manual. It details a simple and direct user manual for those who want to use the application.

Chapter one and seven can also be found in English as it is specified in the regulation for End-of-Degree Projects of the UCM.

Capítulo 2

Estado del arte

2.1. CorewarUCM

CorewarUCM es el torneo oficial de la Universidad Complutense de Madrid donde los contrincantes luchan por la victoria. Cada uno debe crear y diseñar su propio *malware*, su propio código con el que llevarse el premio.

Está basado en **Core War**, un juego de programación nacido en 1984. Se celebra anualmente en el recinto ferial de Madrid **IFEMA**, las inscripciones son gratuitas y cualquiera puede participar.

Los premios que se consiguen en el concurso son cedidos por **Ubisoft**, una empresa de videojuegos francesa que tiene sedes en España.

Para empezar existen dos modos de juego: **Normal1v1** y **Torneo**.

El primero se usa para entrenar, principalmente, pues es un enfrentamiento único donde gana el mejor virus. Se escogen los dos archivos con los que se quiere probar y se le da jugar. Luego se muestra la simulación de la memoria y el estado del combate, de manera que se puede observar lo que sucede durante la batalla. Tras pasar el tiempo del combate, se nombra a un ganador y se puede volver a jugar el mismo combate o regresar al menú principal.

Por otro lado, en el modo torneo se pueden añadir diversos archivos a una lista. Se puede personalizar quién enfrenta a quién y el orden de los combates. Tras cada combate se muestran la opción de repetir combate, continuar con el siguiente, decidir ganador y volver al menú principal. La opción de decidir ganador es debida a que durante el evento, los jueces son los que dictaminan realmente quién ha sido el ganador. Como sucede en el enfrentamiento Normal1V1, los combates se visualizan de la misma manera. Finalmente, cuando todos los combates terminen, se puede ver la tabla de puntuaciones indicando la clasificación.

2.2. Unity

Por otro lado, para el desarrollo del juego se ha utilizado **Unity**, un motor de videojuegos que ofrece una interfaz visual, un editor y *scripting* con `C#` para desarrollar videojuegos de una manera más sencilla. Ha sido la herramienta más fundamental para el desarrollo. Gracias a ello y a su gran comunidad de desarrolladores, se ha podido agilizar bastante el ritmo de desarrollo.

Como **IDE** para progamar con Unity nos decantamos por **JetBrains Rider** debido a su versatilidad, maleabilidad y su capacidad multiplataforma.

También se ha utilizado un *plug-in* de Unity **Runtime File Browser** [20] que consiste en un pequeño programa que ayuda al usuario a visualuzar cuadros de diálogo para guardar y cargar archivos durante la ejecución del juego. Además, proporciona su propia versión de la interfaz mostrando un explorador de archivos en tiempo real del sistema.

Capítulo 3

Metodología y tecnología

En esta sección explicaremos qué metodología preferimos utilizar para llevar a cabo el proyecto y en qué consiste.

3.1. Scrum

La metodología **Scrum** nos ha acompañado a lo largo de toda la carrera en prácticamente todos los proyectos que hemos llevado a cabo. Es una metodología ágil e iterativa que nos permite tener actualizado el proyecto de manera constante y cada pocos periodos de tiempo.

Principalmente se basa en tener *sprints* (pueden ser semanales, cada dos semanas...) en los que se escoge una serie de tareas y se llevan a cabo durante el tiempo planeado. Además, se incluyen reuniones frecuentes (diarias, cada dos o 3 días...) en las que se expone lo que ha hecho cada persona desde la última reunión, si ha tenido problemas, etc, de manera que todo el grupo pueda permanecer actualizado ante el trabajo del resto.

El proceso de planificación de esta metodología se divide en:

- **Product Backlog**: Lo primero es planificar el proyecto entero en la medida de lo posible. A partir de la idea principal del proyecto o del documento de diseño, se obtienen todas las tareas para completarlo, también llamadas historias de usuario. Se trata de *minitrabajos* a corto plazo de los diferentes componentes que tendría el proyecto final. Por ejemplo: diseño del Menú principal, diseño del menú Normal1v1...Además, al ser un proceso iterativo, algunas tareas que *a priori* parecían simples, se pueden dificultar y dividirse nuevamente en dos o más subtareas.
- **Sprint Planning**: Se planifica el tiempo que va a durar el próximo sprint y cuáles van a ser los objetivos más prioritarios. Se dividen los objetivos a realizar para cada miembro del grupo.
- **Sprint Backlog**: A partir del *Product Backlog* se obtienen las tareas del sprint. Se escogen aquellas tareas con mayor nivel de prioridad para que los miembros del grupo las realicen durante el sprint. Cada vez que se termine una tarea se notifica en el registro utilizado y se puede coger una nueva. Así hasta completar las tareas del sprint. Si se han acabado todas los trabajos antes de que termine el sprint se podrían añadir nuevas tareas para continuar avanzando o se podría emplear el tiempo en otros aspectos del proyecto.
- **Daily Scrum**: Reuniones periódicas durante el sprint para proporcionar *feedback* al resto del equipo.
- **Sprint Review**: Se realiza una discusión sobre la posibilidad de añadir o reducir más tareas al sprint, informe de posibles errores encontrados en algún aspecto del proyecto, etc.

- **Sprint Retrospective**: Una vez se ha terminado el sprint, llega el momento de comprobar los resultados. ¿Qué ha ido bien?, ¿qué ha ido mal?, son preguntas que surgen cada vez que se llega a esta etapa. El objetivo es mejorar la gestión del desarrollo, mejorar la planificación de los siguientes sprints y mejorar la calidad del proyecto. Tras acabar esta etapa, se regresa de nuevo al *Sprint Planning* para volver a comenzar el proceso.

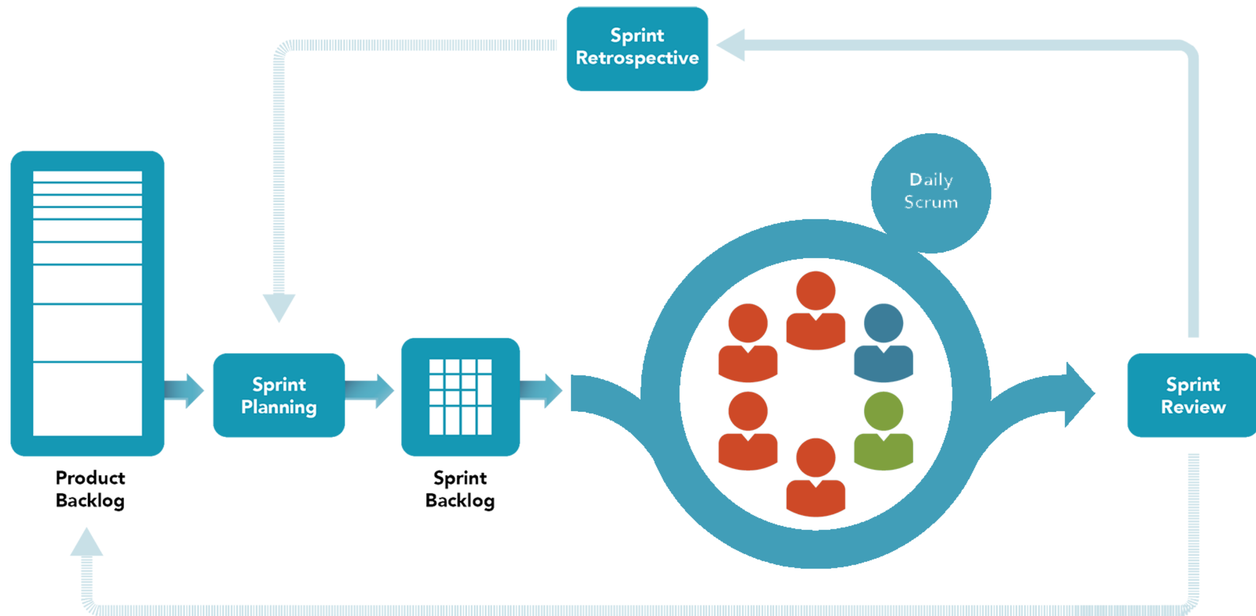


Figura 3.1: Esquema metodología Scrum

Como llevamos tiempo usando esta metodología (casi desde el inicio de la carrera), estamos acostumbrados a su procedimiento. Es por ello que cuanto más la usamos más nos familiarizamos con ella y más beneficios le sacamos. Nos hemos guiado por los siguientes principios:

- **Auto-organización**: Este principio se centra en los trabajadores de hoy, que entregan un valor significativamente mayor cuando se auto-organizan y esto resulta en una mejor participación de los equipos y en la propiedad compartida de lo conseguido; y un entorno innovador y creativo que sea más propicio para el crecimiento.

- **Colaboración**: Este principio se centra en las tres dimensiones fundamentales relacionadas con el trabajo colaborativo: la conciencia, la articulación y la apropiación. También aboga por la gestión de proyectos como un proceso compartido de creación de valor con equipos que trabajan e interactúan juntos para ofrecer el mayor valor.

- **Time-boxing**: Este principio describe cómo el tiempo se considera una restricción limitante en Scrum, y se utiliza para ayudar a gestionar eficazmente la planificación y ejecución de proyectos. Al tener hitos pequeños o entregas en un corto periodo de tiempo, el desarrollo de las tareas se agiliza.

- **Desarrollo iterativo**: Este principio es el principal exponente del desarrollo de videojuegos. Crear proyectos de este tipo necesita una actualización constante de contenido, adaptación a cualquier tipo de cambio y revisión periódica de los contenidos que se quieren desarrollar. Son varias las veces en las que algún aspecto del diseño puede no ser el correcto y necesitar de cambios.

3.2. Plan de proyecto

Antes de comenzar el proyecto se tuvieron varias reuniones con el tutor en las cuales se acordó una división en dos etapas del proyecto. Estas dos etapas serían la fase de preparación de Scrum y de la memoria y la fase de implementación.

- **Documentación de las tecnologías**
- **Preparación de Scrum**
- **Desarrollo y pruebas**
- **Documentación de la aplicación**

3.2.1. Documentación de las tecnologías

Para comenzar a preparar la organización mediante Scrum, primero había que decidir que tecnologías se usarían para el desarrollo. Decidimos usar **Unity**[4] rápidamente, debido a su gran capacidad multiplataforma, la familiaridad con el programa y su gran accesibilidad de cara a trabajos futuros. Además, el programa otorga prácticamente todo lo que necesitamos para el desarrollo, teniendo también diferentes *plug-in* de libre acceso en su base de datos. Entre ellos, destacamos el que ya se mencionó con anterioridad, **Runtime File Browser** [20]. Este pequeño programa nos sirve para generar un explorador de archivos en tiempo de ejecución con el que poder seleccionar aquellos virus que se quieran usar en las batallas. Por otro lado, para realizar pruebas se utilizaba ARES, un entorno ya construido donde poder cargar cada uno de los archivos. El propio programa viene de serie con ejemplos escritos en **RedCode**[11] muy útiles para comprobar su funcionamiento. Además, queríamos que toda nuestra aplicación se desarrollara con licencia libre. Este proceso fue bastante sencillo ya que existe documentación de todas las tecnologías citadas anteriormente.

3.2.2. Planteamiento inicial del proyecto

Una vez visto con qué tecnologías podríamos desarrollar la aplicación, comenzamos a ver cómo crearla. Al principio, lo que planteamos es que fuera un programa con una interfaz sencilla, bonita y rápida de procesar. Además, a diferencia de Ares, el objetivo también era empaquetar y organizar las interfaces, ya que en Ares se encuentra todo en el mismo nivel, eliminando así muchos elementos innecesarios que no íbamos a necesitar.

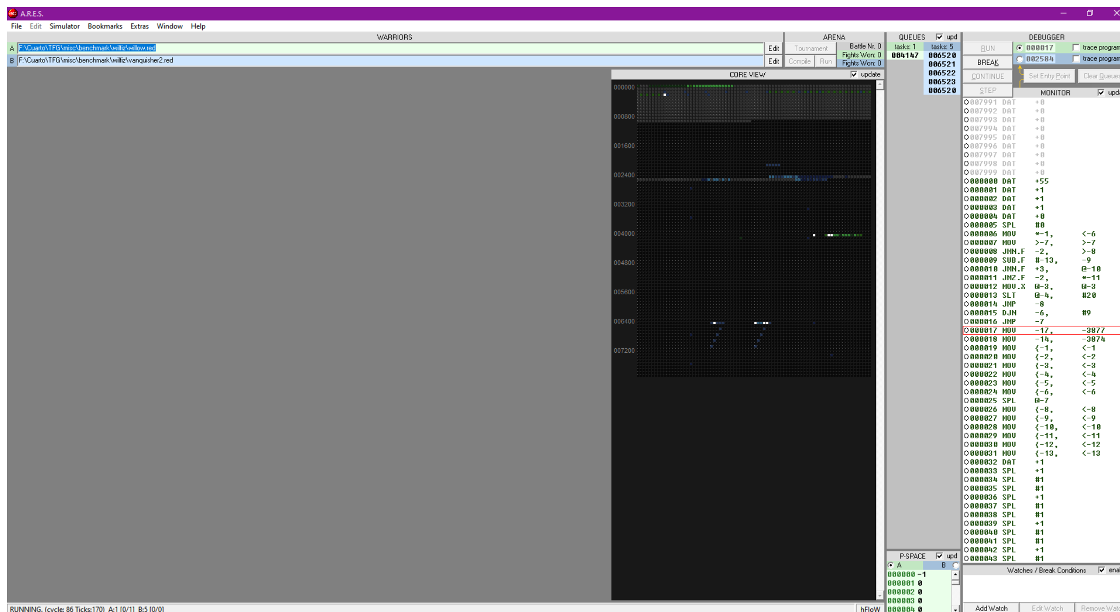


Figura 3.2: Interfaz de Ares

3.2.3. Diseño de la aplicación por Scrum

Lo primero que hicimos para oficializar la planificación fue crear un espacio de trabajo en **Trello**[7], una aplicación web que permite compartir un tablero **Kanban**[8] almacenado en la nube. Kanban es un método para gestionar el trabajo que surgió en «*Toyota Production System (TPS)*». Se trata de un método visual de gestión de proyectos que permite a los equipos visualizar sus flujos de trabajo y la carga de trabajo. En un tablero Kanban, el trabajo se muestra en un proyecto en forma de tablero organizado por columnas.

En Trello, el tablero creado empieza vacío y solo da la opción de insertar columnas, tantas como se necesiten. Además, cada una de estas columnas es muy versátil porque permite añadir tarjetas con múltiples opciones de configuración. Se pueden añadir colores a la tarjeta, imágenes y descripciones, títulos, *checklist*, etc, lo que permite un mayor entendimiento de la tarea en cuestión.

Así pues, es el momento de decidir cuántas columnas queremos y cuál va a ser el contenido de cada una. Esta fase se corresponde a la creación de un **Product Backlog**, mencionado anteriormente, donde se almacena el proyecto fragmentado en diferentes tareas. Entonces, decidimos que sería buena idea poner 6 columnas: «Product Backlog», todas las tareas que quedan por completar del proyecto; «Stiven», «Carlos», «Ricardo», donde cada uno iba a poner las tareas que tuviera en marcha; «Dropped», para las tareas que se acaban descartando; y «Done», para los trabajos ya finalizados. Además, dentro de cada columna, las tareas correspondientes estarían organizadas por colores: «Product Backlog», amarillo o rojo; «Stiven», «Carlos», «Ricardo» con diferentes tonos de azul; «Done», verde; y «Dropped» no tendría indicador de color. Por otro lado, aquellas tareas que tuvieran dependencias y no pudieran realizarse estarían marcadas con un color rojo dentro del Product Backlog, marcando con una *checklist* dentro de la tarjeta las dependencias de la historia de usuario.

En un primer momento, todas las tareas que fueron planteadas, eran demasiado generales como para poder llevarlas a cabo en los posteriores *sprints*. Por tanto, en los posteriores meses del planteamiento inicial, según íbamos avanzando con el proyecto, también fuimos fragmentando todas las tareas que fueran demasiado grandes (esta es la parte positiva de un proceso iterativo como este).

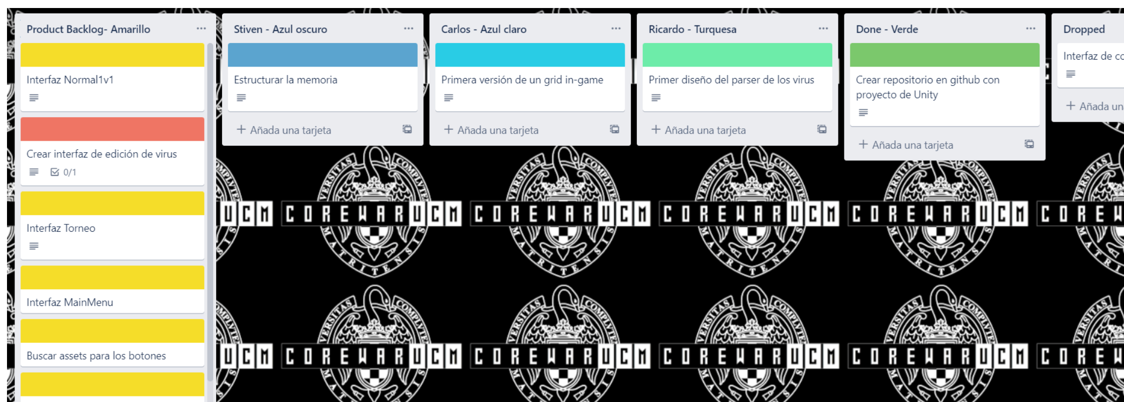


Figura 3.3: Una de las primeras versiones de Trello

Una vez decidido dónde y cómo se almacenaría el **Product Backlog**, era el momento de decidir cómo iba a ser la organización del proyecto. Para empezar, el **Sprint Planning** sería mensual, ya que en el primer momento del proyecto no era tan necesario apresurar la realización de las tareas. Posteriormente, dicho plan cambiaría a *sprints* semanales según se iban acercando los últimos meses del curso. Y, por otro lado, la organización del **Daily Scrum** se decidió que sería semanal, es decir, una reunión a la semana (en principio los domingos). Sin embargo, estas reuniones no serían necesarias en múltiples ocasiones, debido a que la comunicación estaba planteada desde un principio con **Discord**[9], un sistema de comunicación online que

permite la creación y gestión de múltiples canales, tanto de texto como de audio, alojados en un servidor. Este servidor estaría compuesto por los integrantes del proyecto y por el director, lo que permite una mayor capacidad de *feedback* por parte de todos.

Una vez claras todas las decisiones administrativas y de gestión, comenzaría la ejecución de la metodología Scrum a lo largo del curso.

3.3. Tecnologías usadas

En esta sección se verán las tecnologías usadas para desarrollar el proyecto

3.3.1. Unity y C#

Unity[4] es el motor de videojuegos usado para llevar a cabo este proyecto. Todas las herramientas, componentes y *scripts* están montados y empaquetados dentro del motor. `C#`[14] es el lenguaje de programación que se usa para desarrollar videojuegos con Unity.

3.3.2. Runtime File Browser

El propio «**Unity Asset Store**» [5] ofrece una gran cantidad de recursos de *assets*, donde se pueden encontrar diferentes herramientas. **Runtime File Browser** [20] permite agregar al proyecto un explorador de archivos en tiempo de ejecución de una *build* de Unity.

3.3.3. JetBrains Rider

Rider[10] de JetBrains es un IDE .NET multiplataforma basado en la plataforma IntelliJ y ReSharper. Es usada en el proyecto para programar los scripts de Unity.

3.3.4. Ares

Ares[1] es el programa que se usa para enfrentar los diferentes virus en una batalla. Se ha usado principalmente como fuente de inspiración y como centro de pruebas, para así comprobar y saber cómo funciona exactamente la lógica del simulador.

3.3.5. RedCode

Redcode[11] es un lenguaje que se asemeja al lenguaje ensamblador, creado específicamente para ser interpretado por un MARS y más simplificado que un lenguaje ensamblador condicional. Es el lenguaje en el que están escritos los virus de los participantes.

3.3.6. Adobe Photoshop

Adobe Photoshop [12] es un editor de fotografías que se ha usado en severas ocasiones a lo largo del desarrollo del proyecto. Gracias a este programa hemos podido desarrollar con eficacia los *assets* utilizados durante el desarrollo, pudiendo así retocar animaciones, imágenes, verificar que las fuentes usadas sean correctas, etc.

3.3.7. Overleaf

Es la plataforma[13] utilizada para la realización de la memoria de este TFG. Se ha realizado mediante la programación LATEX.

Capítulo 4

Diseño de la aplicación

4.1. Descripción

El torneo del Corewar es un evento que se lleva haciendo ya varios años. Siempre se ha recurrido a *OBS* para retransmitir las partidas de una forma más animada. Además, ARES contiene muchas características que son completamente innecesarias de cara a preparar el torneo para los participantes. También destaca que todo el contenido del programa está ubicado en una interfaz única un tanto abrumadora y poco intuitiva.

Es por ello que, con el desarrollo de esta alternativa, **CoreWarUCM** reduce funcionalidades innecesarias para tener un programa más directo y sencillo. También, cuenta con diferentes menús organizados en diferentes interfaces, de manera que todo sea más fácil de encontrar y fatigue menos al jugador a primera vista.

Tiene dos modos principales: Normal1v1 y Torneo. Dentro del primero, el jugador puede poner a prueba su virus contra otro en una batalla directa, 1 contra 1. Por otro lado, el modo torneo, permite la agregación de hasta 32 participantes y permite la configuración de las batallas. Esto es así por petición del director del TFG, puesto que en los torneos oficiales, entre los jueces y demás directores del torneo, deciden cómo van a ser los enfrentamientos. Además, en ambos modos y dentro del combate, el usuario puede decidir un ganador antes de que se termine la partida (ya que las partidas se podrían alargar demasiado), volver al menú principal o simplemente observar la simulación de la memoria donde luchan los virus.

Cuando se terminen las partidas, aparece el resultado de la batalla, pudiendo haber un ganador o un empate. En cualquier caso, el usuario puede decidir quién ganó esa batalla, dado que en diversas ocasiones serán los jueces del torneo los que decidan quién ha salido victorioso de la batalla, repetirla o volver al menú principal.

4.2. Puntos a tratar

En las siguiente secciones se detalla el diseño general de la aplicación y los ejemplos de uso. Se muestran una serie de imágenes que reflejan el contenido de la aplicación en sus distintas capas.

4.3. Diseño general

A continuación se describirá el diseño de la aplicación en todas sus capas, explicando en qué consiste y mostrando las figuras correspondientes. Se separa en dos puntos: **Diseño del menú principal** y **Diseño *In-Game***.

4.3.1. Diseño del menú principal

Lo primero que se ve al iniciar el programa es la portada, seguida del menú principal:

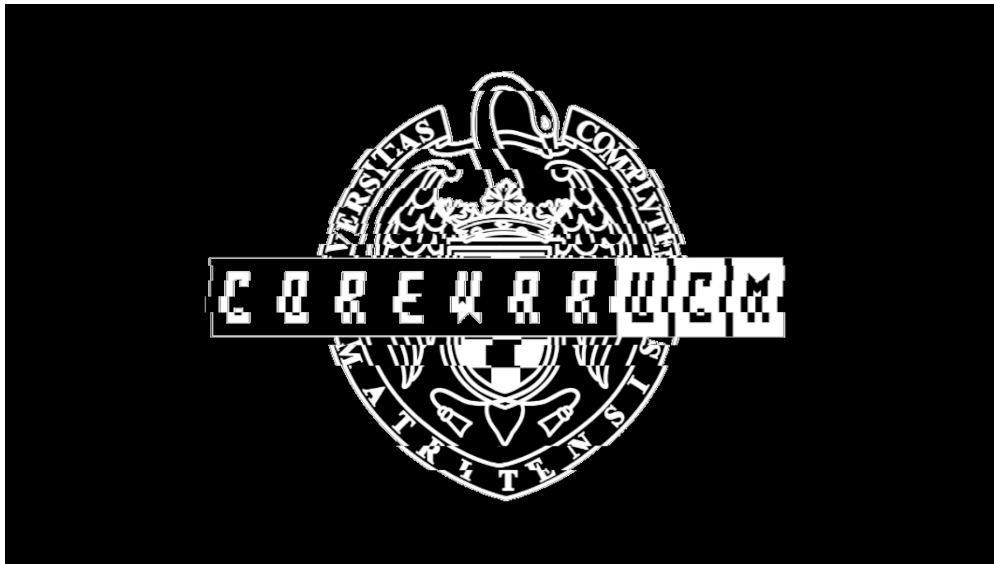


Figura 4.1a: Portada del juego

Figura 4.1b: Menú principal del juego

Dentro del modo Normal1v1 se pueden seleccionar dos opciones: Cargar virus y Editar virus:

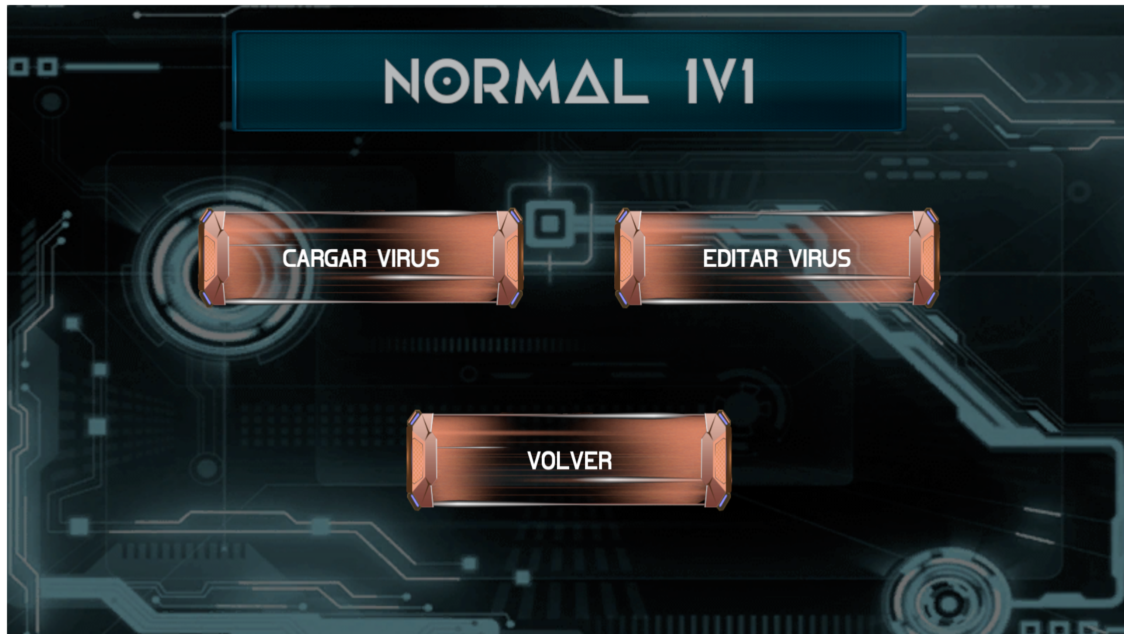


Figura 4.2: Interfaz: Normal1v1

El modo Torneo muestra una lista de los 32 participantes inicializados a *null*. Cada vez que se añada uno nuevo, se busca el primer hueco vacío de la lista de arriba hacia abajo. El botón *CONFIGURAR* permanece deshabilitado hasta que haya al menos 2 participantes activos:

Figura 4.3: Interfaz: Torneo

Una vez se le de al botón, se muestra la interfaz de configuración, donde se pueden añadir batallas y decidir quién luchará en el encuentro. Es similar al funcionamiento de la interfaz anterior, pero en lugar de añadiendo virus, se añaden batallas. Sin embargo, la funcionalidad de la interfaz no se pudo completar, por lo que su interactividad está deshabilitada.

Figura 4.4: Interfaz: Configuración del torneo

4.3.2. Diseño *In-Game*

Cuando se termina de realizar la carga de virus y las configuraciones necesarias, se accede al estado de la batalla donde se enfrentan los virus 1v1. Aquí se puede ver la simulación de la memoria, los nombres de los participantes y 4 botones, uno para volver al menú principal, otro para elegir un ganador y parar la batalla y dos que controlan la velocidad de la simulación (medida en instrucciones por segundo),

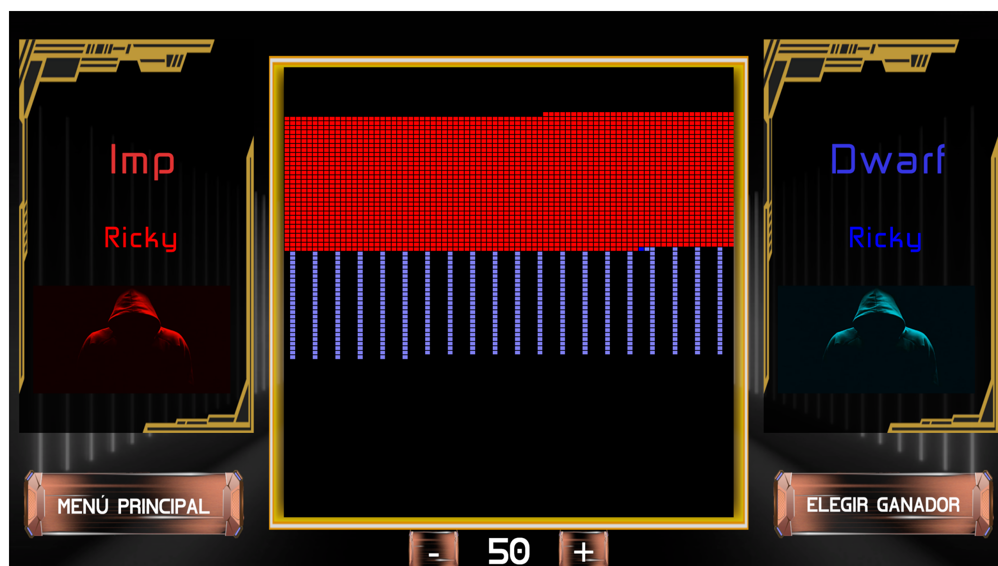


Figura 4.5: Batalla entre dos virus

Si se le da al botón del menú principal, se pausa la partida y se realiza una pregunta de confirmación:

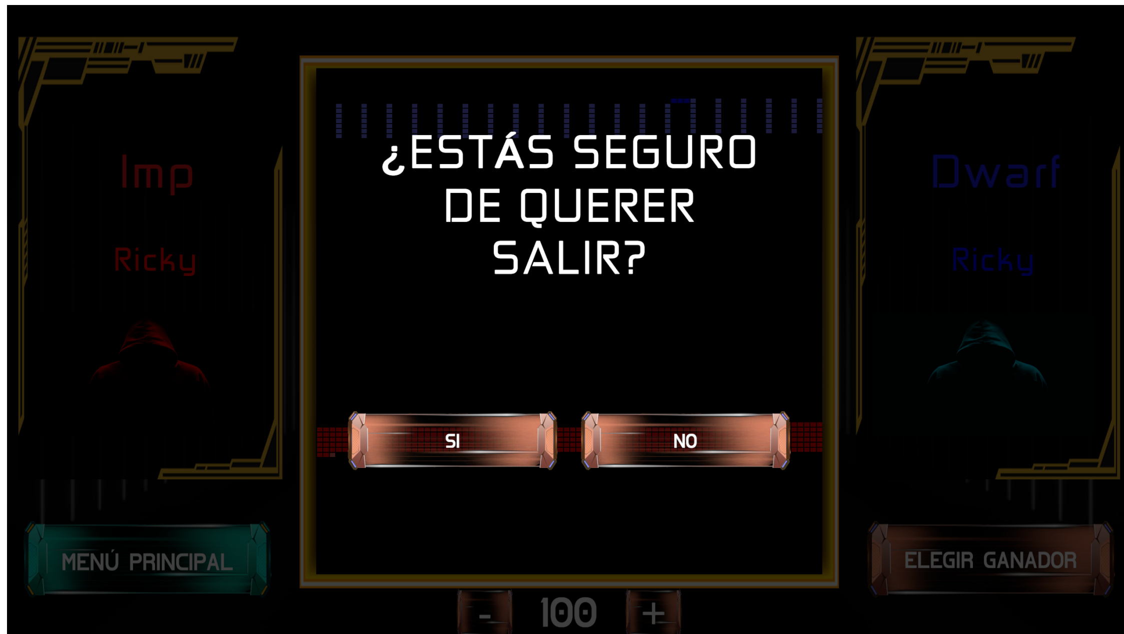
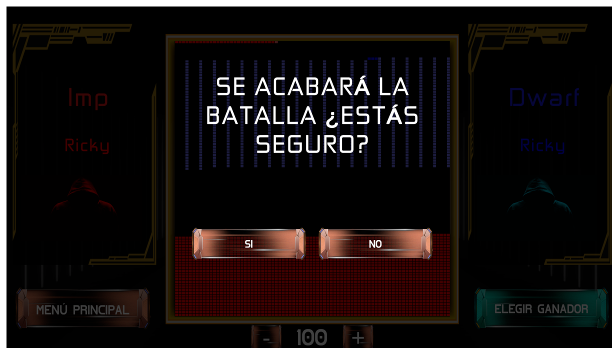


Figura 4.6: Volver al menú principal

La funcionalidad para elegir al ganador es similar a la opción previa. Sin embargo, ahora se hacen dos preguntas, la de confirmación y la de elegir el ganador:



(a) Pregunta de confirmación para elegir ganador

(b) Pregunta para elegir al ganador

Figura 4.7: Elegir ganador

Posterior a la elección del ganador, se muestra la interfaz de resultados. Evidentemente, si la batalla ha acabado por su propia cuenta también se mostrará la interfaz de resultados. La batalla puede haber acabado con un ganador o en empate. En el caso de que se haya elegido ganador, el botón de .ELEGIR GANADOR"que se observa, estará deshabilitado. Además, el botón ÇONTINUAR.aparece desactivado también, ya que es un botón pensado para el torneo, donde se puede continuar con la siguiente batalla. Por último, si no se está de acuerdo con el resultado de la batalla siempre se puede repetir:

Figura 4.8: Resultados de la batalla

4.4. Ejemplo de uso

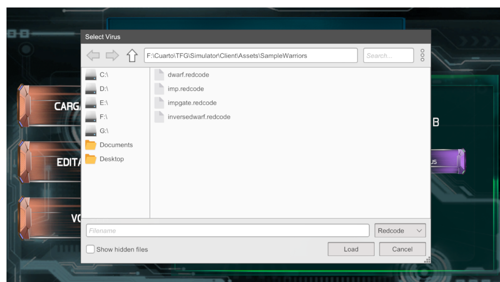
En este apartado se ve cómo utilizar la aplicación, dependiendo de los tipos de usuarios:

- Competidor
- Gestor del Torneo

4.4.1. Competidor

Su principal interés reside en el modo Normal1v1, debido a que es el acceso más rápido a un combate para poner a prueba su virus.

En la opción de cargar virus se pueden seleccionar los archivos con extensión *.redcode* o *.red*. La diferencia principal entre ambos es que el .redcode solo se puede generar desde el editor de la aplicación y funciona a modo archivo comprimido. Dentro se guarda el código .red y el avatar asociado, con extensión *.png*. Además, el botón *JUGAR* estará deshabilitado mientras solo haya un virus cargado:



(a) Selección de archivo

(b) Resultado al cargar virus

Figura 4.9: Normal 1v1 - Cargar Virus

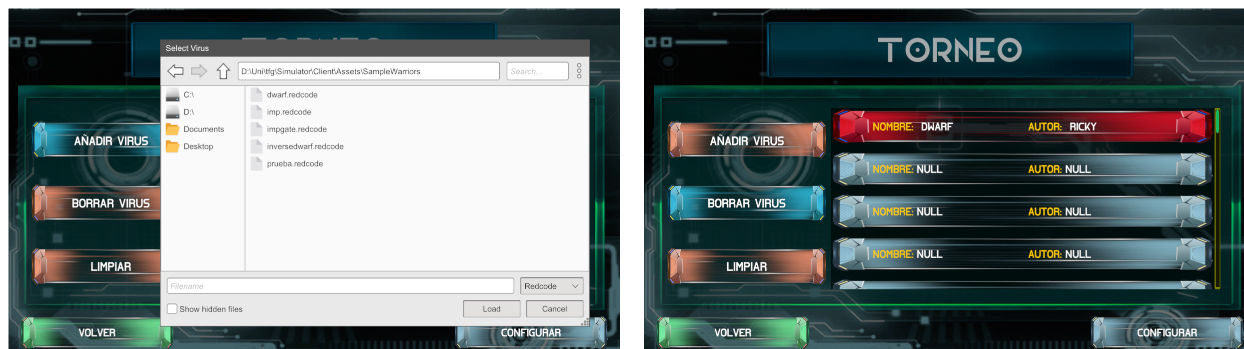
Por otro lado, la opción *Editar Virus* permite editar el virus que se seleccione sin necesidad de abandonar la aplicación:

Figura 4.10: Editor del virus

En cuanto esté listo, le dará al botón de *JUGAR* y se procederá al estado de la batalla explicado anteriormente. También puede ser relevante para el usuario organizar sus propios torneo, de forma que pueda practicar su propio virus contra otros muchos de una forma más fluida.

4.4.2. Gestor de un torneo

En el caso de un gestor del torneo, éste podría añadir a todos los participantes necesarios en la lista (hasta un máximo de 32). Puede eliminar participantes de la lista o puede limpiarla directamente.



(a) Selección de archivo (b) Eliminación de un participante

Figura 4.11: Adición y eliminación de un participante en el torneo

Una vez se hayan añadido los participantes, se procederá a la configuración del torneo. Ya se mencionó anteriormente que dicha configuración está deshabilitado. Dentro de la configuración se pueden añadir batallas y elegir a los contrincantes de cada duelo, de forma que sea el gestor quien determine cada uno de los enfrentamientos. Al igual que en la selección de competidores, se pueden borrar batallas seleccionadas y limpiar la lista entera. Una vez listo, se le dará al botón *JUGAR* y se pasará al estado de la batalla para dar comienzo al primer combate.

Dentro de la batalla, para el gestor puede ser interesante la opción de *ELEGIR GANADOR* para que la batalla no se alargue demasiado. También es él quién decidirá el comienzo de la batalla y quién decide, al finalizar un combate, cuándo puede proseguir el siguiente.

Capítulo 5

Arquitectura e implementación

5.1. Capas de la aplicación

A continuación se explicará la arquitectura y la implementación del proyecto. Por lo tanto, se comentarán 3 subsecciones clave que se corresponden a 3 escenas de Unity diferentes: **Loader**, **MainMenu** y **GameScene**.

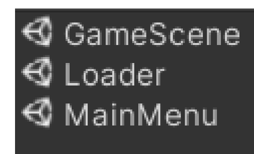


Figura 5.1: Escenas de Unity

5.1.1. Escena de Unity:*Loader*

Es la primera escena del juego (véase Figura 4.1a). En ella se muestra la portada del juego que actúa a modo de botón para acceder a la siguiente escena *MainMenu*.

Su funcionalidad es sencilla, el *GameObject* «**Loader**» cuenta con un componente *script* llamado *Loader.cs* que se encarga de cargar la última ruta utilizada en el programa. De esta forma, cuando se quiera cargar un virus no hay que volver a buscar la carpeta en cuestión. Dicho *GameObject* se observa en la siguiente figura en la parte lateral izquierda de la imagen.

La escena también cuenta con una animación *fade-out* para el cambio de escena.

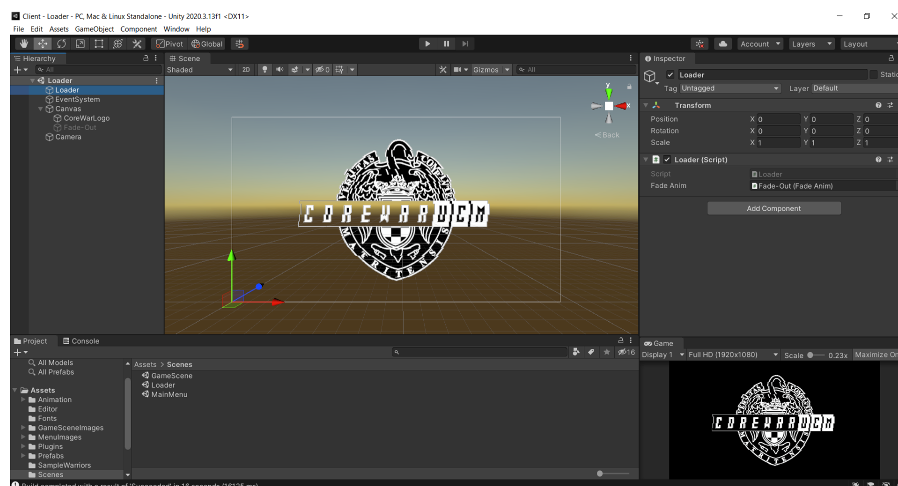


Figura 5.2: Escena de Unity: Loader

5.1.2. Escena de Unity: *MainMenu*

Comienza con un efecto *fade-in* y es la escena que sucede a la anterior. Se trata de una escena ya mucho más compleja que se caracteriza principalmente por su gestión de interfaces de usuario.

Existen 4 menús en total que se pueden observar en la jerarquía de *GameObjects* en la siguiente figura (lateral izquierdo de la imagen): *MainMenu*, *Normal1v1*, *Torneo*, *Config*.

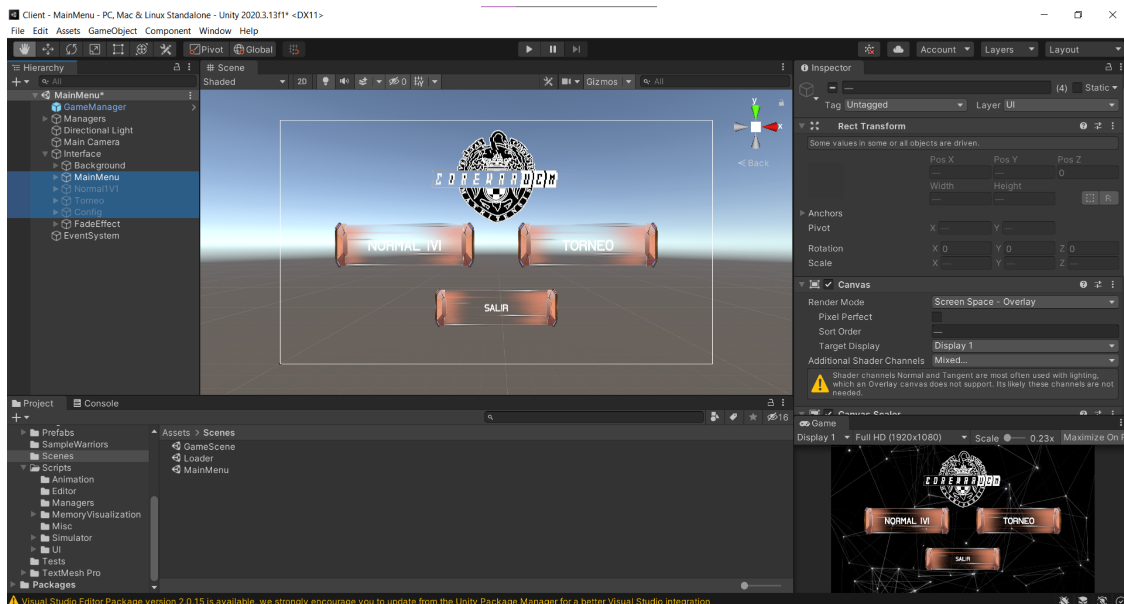


Figura 5.3: Escena de Unity: MainMenu

En general, para simplificar la funcionalidad y hacerla más sencilla, Unity permite agregar *callbacks* en los botones referenciando a otros *GameObjects* y a sus componentes. De esta manera, para navegar a través de los menús, estos *GameObjects* se activan y desactivan de forma sencilla con el componente *Button*:

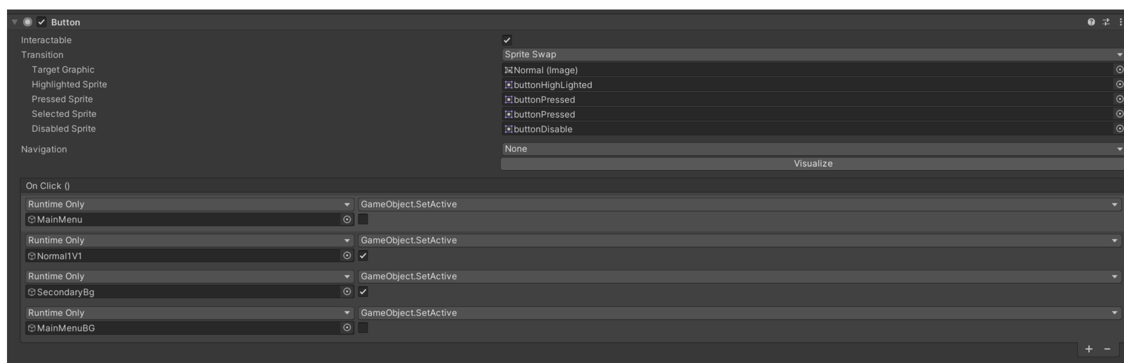


Figura 5.4: Ejemplo del componente *Button*

Es decir, Unity permite al usuario una gran accesibilidad a funciones varias sin necesidad de cablear código por debajo. Directamente, se pueden agregar y quitar componentes ya existentes que permiten hacer gran cantidad de cosas.

Además, con el editor de Unity también se pueden agregar animaciones tanto para los botones como para otro tipo de componentes. Esto es importante, porque algunas animaciones implementadas se resumen a

un simple archivo con extensión *.anim* de Unity, con el que podemos cambiar diversos parámetros para personalizar la animación. Otras animaciones en cambio se han construido mediante código, aunque su complejidad no es demasiado alta. Un ejemplo de esto es el efecto de parpadeo, aplicado en diferentes objetos del programa, que se implementa con el componente *BlinkEffect.cs*.

Es en esta escena es donde se crea por primera vez la entidad **GameManager**, la cual es la única en todo el flujo de la aplicación que sigue viva de principio a fin. Sirve de puente entre las entidades de cada escena y se encarga de crear y guardar una referencia a aquellos *managers* a los que se necesite un acceso rápido desde fuera. Al trabajar con Unity, una práctica muy común es la de implementar diferentes gestores para que monitoricen diferentes aspectos de la escena. Para permitir esa conexión fácil con otras entidades, el *GameManager* utiliza un patrón **Singleton**. En Unity, también existen diversos métodos que se sitúan en el flujo de ejecución del motor, donde se aloja también el bucle principal de los ejecutables que genera. El método *Awake* pertenece a dicho flujo de ejecución y se llama cuando una entidad se crea o se activa, para inicializar la instancia de la clase. De esta forma, es en dicho método donde reside la propia inicialización del singleton.

El script *GameManager.cs* tiene diversas variables declaradas públicas, de forma que desde el propio editor se puedan agregar los componentes a los que se hace referencia. Como estas referencias varían en función de la escena, el primero que se crea en el *MainMenu* no posee todas las referencias de aquellos componentes que están en otras escenas. Para solucionar esto, se crea una instancia del *GameManager* en cada escena y se le asignan las referencias que le corresponda a cada una desde el editor. Cuando se produce el cambio de escena (por ejemplo del *MainMenu* a *GameScene*) se llamará al Awake de la entidad *GameManager* que está en la nueva escena. Cuando esto ocurra comprobará si ya existe una entidad *GameManager* asignada al singleton y al confirmar esto se destruirá a si mismo. Previo a la destrucción, la entidad destruida le pasa la información que posea al singleton actual, de manera que sea éste quien tenga ahora las referencias que se necesitan.

Para manejar la carga y el guardado de los virus se ha implementado la clase **VirusIO**, con la funcionalidad de leer y guardar archivos utilizados por simulador para la ejecución de los programas. Implementa la librería **SimpleFileBrowser** [20] lo que permite abrir un buscador de archivos en tiempo de ejecución. Este buscador permite cargar archivos (filtrados solo por extensión *.redcode* y *.red*) o crear nuevos desde cero en el editor de virus del programa. Dentro de este editor (véase la figura 4.10) se observa que se puede también cargar iconos dentro del archivo mediante el botón **ICONO**. Una vez se quiera guardar el virus creado, el usuario puede escoger si guardarlo como .redcode o como .red, de forma que si se escoge la segunda opción no se guardará el icono cargado, en caso de haberlo. Como funciona de manera asíncrona , para su uso, la llamada del método **LoadVirus** del componente *VirusIO.cs* se realiza mediante una corrutina de Unity. Las corrutinas se usan para realizar la llamada a un método tras haber pasado un tiempo específico. Para controlar este flujo de ejecución se utilizan **Normal1v1Manager.cs** y **TournamentManager.cs**, que son componentes asociados a un *GameObject* diferente cada uno. Por el lado del modo Normal1v1, existe un método en su *manager* correspondiente llamado **LoadVirus** el cual está asociado al botón cargar virus del participante A o B (véase figura 4.9). Por el lado del torneo, la comunicación se produce a través del botón *AÑADIR VIRUS*, quien contiene un *callback* al método **AddToList** de *TorunamentManager* para añadir nuevos participantes a la lista. En ambos casos, la llamada a dicho método produce una comunicación con el *GameManager* para cargar el archivo que se quiere usar y la información cargada es devuelta al *manager* correspondiente para que se encargue de actualizar la visualización en pantalla de lo sucedido.

El guardado en «*Editar Virus*» también es asíncrono, pero debido a que no necesita mandar ningún dato simplemente recibe el virus a guardar sin necesidad de *callbacks*. El flujo de ejecución es similar, donde el botón para guardar virus se comunica con *EditorManager* con el método **SaveVrus** y éste, a su vez, con el *GameManager.cs*.

Por otro lado, dentro de los menús *Normal1v1* y *Torneo* se encuentra lo siguiente:

- **Normal1v1 - Cargar virus:** Para empezar, en Normal1v1, existe un efecto de desplazamiento de los botones, para que se reorganicen en el lado izquierdo de la pantalla (véase Figura 4.2 y Figura 4.10) al pulsar *CARGAR VIRUS* o *EDITAR VIRUS*. Dentro de las opciones para cargar el virus se observa un apartado para el virus A y otro para el virus B. En ambos casos, la funcionalidad es la misma y es la que se acaba de explicar hace unos párrafos. Con un click se abre un explorador de archivos (véase Figura 4.9) y se selecciona el virus que se quiere cargar. Una vez se tengan los dos preparados solo hay que darle a jugar. Este botón está enlazado a *Normal1v1Manager* para que se le comunique al *GameManager* que se quiere cambiar de escena. Una vez hecho esto, el programa compila los virus en la siguiente escena, de forma que si hay algún error en la ejecución de éstos, se notificará y la partida se verá bloqueada, dando opción solamente a volver al menú principal.
- **Normal1v1 - Editar virus:** Una herramienta interna que permite de manera sencilla crear y editar virus. Consiste de un panel donde el usuario puede escribir el código de sus programas y después guardarlo en un archivo para utilizarlo en las batallas. También puede cargar un archivo ya existente para modificarlo. Para implementar esta funcionalidad se utiliza un componente **InputField** de *Unity*, lo que permite fácilmente editar texto dentro del cuadro de texto asociado. Para la parte de carga o guardado se utiliza el componente **EditorManager** asociado que se comunica con el ya descrito *VirusIO* para la entrada y salida de datos o archivos.
- **Torneo:** Finalmente, la funcionalidad del modo Torneo es similar a la de cargar los virus, puesto que al darle al botón de añadir virus, aparece el explorador de archivos y se escoge el virus. Sin embargo, en este caso al funcionar a modo de lista, será el *TournamentManager* quien se encargue de su gestión. El botón para eliminar virus funciona también a través de *TournamentManager* así como el botón de limpiar. Posteriormente, cuando se tengan listos los participantes, se le dará al botón de configuración y se accederá a la interfaz de configuración de batallas, la cual se encuentra sin funcionalidad.

5.1.3. Escena de Unity: *GameScene*

Finalmente, la escena del juego es la más compleja de todas. Por un lado están las funcionalidades de las del simulador de la batalla y, por otro, las de UI.

Para la implementación de la lógica se se han usado principios de diseño *SOLID* [15], como el *Single Responsibility Principle*, y se ha hecho uso de útiles patrones de diseño, como el patrón *Observador* o *Proxy*. Gracias al buen uso de estos patrones y principios la aplicación es fácilmente extensible, y modificable. Además de esto, para la simulación se ha utilizado una gran cantidad de *test de unidad* para garantizar el funcionamiento de componentes básicos y algunos más complejos.

La siguiente figura es una vista a alto nivel de la arquitectura del simulador:

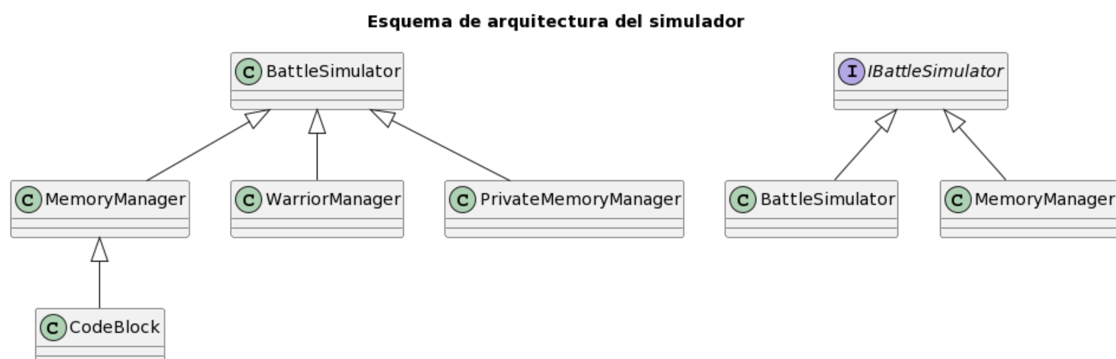


Figura 5.5: Esquema de arquitectura del simulador.

Como se puede observar se ha intentado separar las responsabilidades en distintos componentes con una jerarquía de niveles. En el nivel más alto se encuentra el **BattleSimulator**, que es la clase encargada de empezar la simulación, avanzarla y notificar a los observadores de posibles cambios en el estado. Además, contiene funcionalidades para ajustar la velocidad de ejecución de las instrucciones; en la *figura 4.5*, los botones que se observan en la parte inferior central son los que contienen la lógica descrita. Para completar estas tareas, tiene tres componentes clave que se encuentran en el siguiente nivel de la jerarquía:

- **WarriorManager**. Los virus de CoreWar son bastante complejos y gran parte de esta complejidad viene dada por la capacidad de los virus de tener múltiples procesos, por la existencia de múltiples virus en una misma batalla, etc. Es por eso que este componente se encarga de manejar a quién le corresponde el siguiente turno, cuáles de sus procesos ejecuta o dónde se encuentra en memoria.
- **CommonMemoryManager**. Es el componente con más complejidad detrás, pues es un contenedor de virus «*CodeBlocks*» y el campo de batalla a la vez. Este componente sirve como puente entre la simulación a bajo nivel y su representación a niveles más altos.
- **PrivateMemoryManager**. Otro concepto que añade complejidad en las últimas versiones de CoreWar es la capacidad de los virus de tener memoria privada. Este componente facilita su uso, ya que es bastante diferente a la memoria común.

El *CommonMemoryManager* es un contenedor de *CodeBlocks*, componentes de más bajo nivel que se encargan de determinar qué ocurre cuando la instrucción se ejecuta en unas circunstancias específicas. Todos tienen una interfaz común, pero se hace uso de herencia y polimorfismo dinámico para tratarlos indistintamente a niveles más altos.

Para facilitar la interacción con el resto del sistema, se usa la inyección de dependencias mediante métodos, de tal manera que la función más importante de estos bloques, llamada ***Execute***, toma como parámetro un objeto que se usa de puente entre el bloque y el resto del sistema.

Este objeto que se inyecta es el *CommonMemoryManager*, que ha implementado una interfaz común con el *BattleSimulator*. Haciendo uso del patrón *Proxy*, algunas de las implementaciones de esta interfaz en el *CommonMemoryManager* repercuten en simples llamadas al *BattleSimulator* y otras que puede manejar las maneja él.

Para poder hacer llegar todos estos datos al inicio de la escena, se necesita algo que guarde los virus seleccionados en escenas anteriores. Hay que recordar que *Unity* elimina todo cuando cambia de escena salvo que se le indique. Como ya se explicó en la escena *MainMenu*, el *GameManager* no desaparece entre escenas y sirve de puente entre las entidades. Una de las clases asociadas al *GameManager* que no implementa *MonoBehaviour* es el **VirusManager**. *MonoBehaviour* consiste en la clase base de la que derivan los componentes de script de Unity. *VirusManager*, se encarga de guardar todos los virus que han sido cargados y llevar la lógica de la pareja que se va a enfrentar. No está asociado al ciclo de ejecución de *Unity*, sino que la crea el *GameManager* en la construcción del *Awake*. Por otro lado, **BattleManager**, un componente asociado a una entidad vacía dentro de la escena, guarda una referencia al *UIManager* y al *BattleSimulator*. Cuando comienza la escena, el *GameManager* le pasa los virus que van a pelear y *BattleManager* se encarga de traspasarlos al simulador y al *UIManager* para que se inicialice todo.

El *frontend* de la simulación es bastante sencillo en cuanto a arquitectura se refiere. Existe un componente observador que se registra a los cambios que reporta el *BattleSimulator*, y los representa visualmente. Por ejemplo, cuando un bloque es ejecutado en memoria, un mensaje de ejecución es mandado y el *frontend* responde pintándolo del color correspondiente. Para poder representar las celdas se creó un *sprite* en interfaz con un *shader* asocidado para aumentar la velocidad de pintado, ya que se haría en paralelo en la GPU. Este *shader* itera $x*y$ veces (siendo x el ancho de la textura del *sprite* e y la altura) y pinta el téxel correspondiente con el color de la celda que toque.

Este sistema es muy extensible y permite reaccionar de distintas maneras a distintos mensajes en función de la configuración o las versiones del software. De manera relativamente sencilla se pueden añadir sistemas de partículas u otros efectos gráficos complejos con mucha facilidad de integración.

De la misma manera se podría hacer, por ejemplo, una interfaz puramente textual, pues toda la simulación es independiente de la visualización.

La estructura de esta escena es similar a la anterior en cuanto a interfaces se refiere. Cada uno de los menús utilizados producen una transición entre sí sin necesidad de abandonar la escena, sino que funciona mediante la activación y desactivación de *GameObjects*:

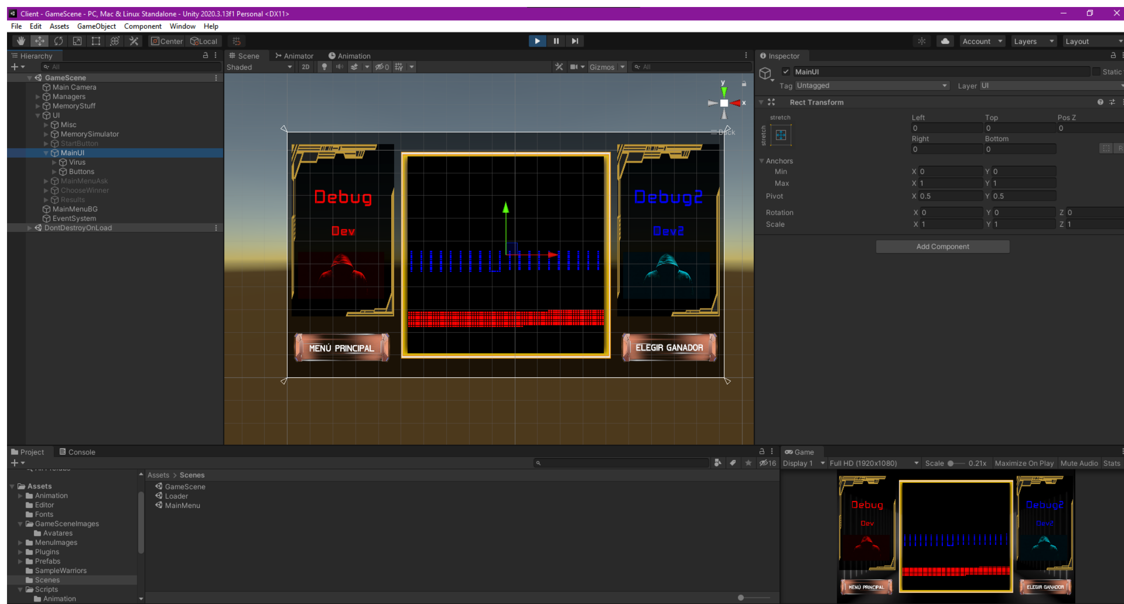


Figura 5.6: Escena de Unity: GameScene

Cada uno de los virus dispone de lo siguiente: nombre del virus, nombre del autor y un avatar. Cada virus está en un lateral de la pantalla siendo rojo el de la izquierda y azul el de la derecha. El *UIManager* guarda la referencia a todos los elementos de la escena para poder modificarlos. Cuando el *BattleManager* ha cargado sus datos, éste le pasa los virus al *UIManager* (como se indicó más arriba). Además, *UIManager* se encarga de modificar los elementos en pantalla para mostrar el nombre del virus y su autor.

5.2. Flujo de ejecución

- **Competidor**. El competidor utilizará principalmente el apartado de normal 1v1. Dentro puede seleccionar entre un combate donde probar sus programas o un editor donde crear o modificar los virus. Dentro del editor puede, mediante un editor de texto, crear nuevos virus. También puede modificar virus existentes para mejorarlos o cambiarlos y una vez terminado puede guardarlos en su ordenador. En el apartado de combate puede seleccionar dos virus y así entrar dentro de la simulación, donde se muestra la representación de la batalla.

- **Gestor del Torneo**. El apartado de torneo está diseñado para que tenga todas las funcionalidades necesarias para poder ejecutar los torneos de la CoreWar. Este apartado le permite, pues, seleccionar hasta 32 participantes desde el buscador de archivos, posteriormente, configurar los enfrentamientos (añadiendo y eliminando combates mediante una lista) y, finalmente, proceder a jugar. Sin embargo, a

pesar de su diseño, su ejecución no está implementada al 100 %, por lo que la interfaz de configuración de batallas permanece desactivado.

Además, dentro de la batalla, ambos usuarios pueden volver al menú principal en cualquier momento, modificar la velocidad de ejecución del simulador o elegir un ganador. Esta última opción está sobre todo pensada para los gestores de un torneo, de forma que se pueda acabar la batalla con antelación y así proseguir al siguiente combate.

Capítulo 6

Resultados

6.1. Discusión de los resultados

Empezando por la parte negativa de los resultados, el proyecto no está funcional en todas sus capas. Por diversos motivos, no hemos sido capaces de terminar con el modo torneo. Sin embargo, las batallas y la funcionalidad «*in-game*» del juego funcionan. Esto es importante porque el torneo consistiría en implementar la configuración de las batallas y posteriormente, un *ScoreManager* que se encargue de gestionar las puntuaciones de los participantes, de manera que cuando se pulse el botón para jugar el torneo se proceda de la misma manera que en «Normal1v1» al cambiar de escena. Además, habría que adecuar el contenido estético a la a aplicación para mantener cierta coherencia.

No se ha podido realizar pruebas con usuarios, pero sí hemos podido contar con el apoyo de compañeros del sector que se especializan en diseño. A partir de su punto de vista profesional, se han llevado a cabo labores de refactorización de recursos gráficos y del aspecto general del juego, como puede ser la colocación de los botones, los colores, la coherencia en los menús, etc.

Así pues, se destacan los siguientes puntos:

- **Programa extensible:** se ha creado un código fácilmente accesible para los futuros trabajos. También Unity otorga con el editor una gran expansibilidad en el proyecto y en los temas de diseño. Se pueden cambiar los recursos estéticos simplemente sustituyendo los recursos en el editor de Unity o directamente en los archivos *Assets* del programa.
- **Interfaz intuitiva:** se han creado botones grandes, legibles y literales, es decir, los botones representan exactamente lo que hacen sin perdida alguna. Esto provoca que el usuario pueda acceder al programa sin ningún inconveniente sobre su uso, como puede pasar con ARES a primera vista.
- **Interfaz modernizada:** como ya se venía comentando, uno de los objetivos al crear esta alternativa para el Corewar, era crear una interfaz más colorida, alegre, vistosa y moderna. Siguiendo un poco las tendencias actuales por las luces, el neón, lo cibernético, etc, se ha creado una estética que atrae a sus usuarios solo por su diseño.
- **Portátil:** Unity genera *builds* que no necesitan instalarse en el sistema. Directamente otorga un ejecutable junto con otros archivos protegidos para que se pueda arrancar el programa de manera rápida y sostenible. Además, no necesita de internet para funcionar.
- **Multiplataforma:** tanto para los amantes de Windows como para los de Linux, el programa es completamente funcional en ambas plataformas.

6.2. Problemas al proyecto

En el apartado de la simulación hubo grandes problemas desde el principio del proyecto. Inicialmente, se trató de implementar el juego de tal manera que no se simulara nada, sino que simplemente se viesen resultados de un simulador ya existente, para así centrar el foco en el apartado visual de la aplicación.

Esto parecía prometedor, pues la simulación es bastante compleja (con un conjunto de casi 20 instrucciones, 6 modificadores, 8 modos de direccionamiento, multiproceso, memoria privada...), pero resultó imposible. Ningún simulador aportaba el detalle que se necesitaba. Después de descifrar el depurador de «*pMars*», se consiguió tener un prototipo relativamente completo, pero con obstáculos que no podían superarse. El mayor obstáculo fue la dificultad para distinguir a quién le pertenece la memoria modificada.

El depurador muestra la ejecución de los virus paso a paso, y se puede investigar el estado de la memoria, detectar los cambios en memoria en cada turno y atribuirlos al virus correspondiente. El problema de este enfoque es que no se pueden distinguir quién fue el último en modificar la memoria, en el caso de que dos virus modifiquen la misma zona de memoria.

En la parte de visualización de la memoria también hubo numerosos problemas que llevaron a varias iteraciones de diseño e implementación. Inicialmente, se generaban una entidad en escena por cada celda de memoria. Si se supone el tamaño estándar utilizado esto supondría un total de 8000 entidades en pantalla. Esta primera versión, aunque funcional a nivel de pintar las celdas, tenía un rendimiento pésimo.

Inicialmente se atribuyó al renderizado, pensando que dibujar tantas entidades ralentizaba la simulación y resultó ser cierto, pero no por la razón que se pensó. El renderizado iba sorprendentemente rápido (aunque con margen de mejora), lo que ocurría era que la CPU estaba haciendo de cuello de botella. Resulta que al ser entidades separadas, Unity tiene que recorrerlas una a una para ejecutar sus componentes, lo que añadía 8000 iteraciones extra cada *frame*. Para solucionar esto se eliminaron las entidades y se creó una única entidad donde se simularían las celdas mediante el pintado con un shader(como ya se explica en la implementación).

Además, en el editor hubo problemas por el manejo que hace Unity de los componentes *InputFields*. La idea inicial era implementar un sistema similar al *intellisense* para indicar los errores del código, autocompletar y marcar con diferentes colores las partes del código, como los comentarios. El problema era que para poder implementar todo esto, no solo se tiene en cuenta la cantidad de revisiones de texto necesarias, sino que el mayor problema tenía que ver con el manejo que hace Unity de la edición dinámica de estilos de fuente. Para poder cambiarlo, por ejemplo, para marcar un comentario de verde, es necesario utilizar un lenguaje de marcado similar a **XML**, lo cual acarreaba numerosos problemas debido a los solapamientos que podían hacer diferentes estilos. Al final se decidió descartar para esta versión y dejar el editor simple, sin *intellisense*.

6.3. Contribución personal

6.3.1. Contribuciones realizadas por el grupo

Las siguientes son las tareas en las que se ha trabajado conjuntamente:

Búsqueda de herramientas similares

Al comenzar el proyecto, uno de los primeros pasos era buscar y entender herramientas que hicieran lo mismo que CorewarUCM. Sin embargo, solo se pudo encontrar una, ARES, ya comentada anteriormente. Así pues, tanto por separado como en grupo, cada miembro estuvo analizando y manejando el programa

para entender su funcionamiento. Se observó que el programa poseía diversas funcionalidades no muy bien accesibles y, dado que nuestro objetivo sería buscar una alternativa, se comenzaría una pequeña recogida de datos para resaltar todo aquello que se querría suprimir y todo aquello que se querría mantener.

Obtención de la información

Para conseguir crear tu propio virus hace falta algo más que saberse los comandos. A partir del *Manual del Corewar* el grupo comenzaría a extraer información acerca del funcionamiento de los programas y de las estrategias ya existentes. Además, también haría falta conocer qué es lo que hace cada línea de código de *RedCode*, ya que habría que simular su función, por lo que se buscaría información tanto de los profesores, como de internet y de los virus que vienen de ejemplo en el programa de ARES.

Además, también haría falta algo de información acerca de la logística del proyecto, al menos una pequeña base que sea útil para que el grupo pueda organizar el proyecto de una forma más ordenada. Esto hace referencia a la decisión de escribir la memoria en *LATEX*, ya que para ello primero habría que buscar información básica acerca de este lenguaje. Sin embargo, sería Stiven quien cogería el relevo de esta tarea.

Diseño del proyecto

Antes de comenzar a programar y a desarrollar, uno de los puntos más vitales del proyecto fue diseñar una buena metodología de trabajo y un buen reparto de tareas. De esta forma, en un primer momento, sin nada por lo que empezar, Carlos decidió crear un proyecto en Unity para empezar a probar cosas. Gracias a ello y al entendimiento de ARES, se comenzó a crear un sistema en *Trello* que permitiese ir volcando las ideas principales más otras un poco más creativas.

Además, a lo largo del proyecto, aunque Stiven estuviese encargado del diseño de interfaces o Ricardo estuviese encargado del diseño del grid del simulador de la batalla, se hacían reuniones para confirmar dichas decisiones. De esta manera, todos los integrantes del grupo estarían al tanto de todas las decisiones y así participarían todos en decisiones importantes.

6.3.2. Contribuciones realizadas por Carlos

Entre las contribuciones aportadas por Carlos se destaca:

Sistema de Entrada y Salida de datos

Para permitir a los usuarios manejar de manera cómoda sus programas, así como crear nuevos, se creó el sistema de *VirusIO*. Éste permite por medio de un buscador de archivos interno el acceso al disco del dispositivo para seleccionar los virus que serán utilizados en las peleas. También permite generar nuevos archivos desde el editor de la aplicación.

Editor

El editor interno, creado a partir de un *InputField* de Unity modificado, permite al usuario crear o modificar sus programas. Se pensó que era algo esencial para la comodidad del competidor, que no requiere de programas externos, pudiendo no solo crear sino probar sus virus directamente dentro del mismo programa.

Sistema de visualización de la simulación

Ya explicado en profundidad en el apartado dedicado a la GameScene, la aplicación necesitaría un sistema de visualización de la memoria y de las acciones ocurridas en el simulador. Para la optimización de GameScene y para evitar pérdidas de *frames*, provocando por un cuello de botella en la CPU, se vio necesaria la creación de un *shader* que calcule el color de los *texels* de la textura en función del estado de la memoria.

Gestión correcta del GameManager

Tras muchas *features* implementadas, el GameManager se empezó a convertir en un código que no se podría mantener. Teniendo en cuenta que es el *core* de la aplicación y que conecta todo, había que reescribirlo al completo cambiando su diseño. El objetivo es, pues, que conecte toda la arquitectura del programa y que delegue el trabajo a las clases y *managers* situados en cada escena.

Conexión FrontEnd-BackEnd

Como se puede observar en las contribuciones de los otros compañeros, ambos están colocados en extremos opuestos de la pipeline. Mientras que uno se centraba en la lógica del simulador y su *backend*, el otro se encargó del apartado de la UI y su *frontend*. Para poder conectar todo esto, especialmente el menú principal con el juego, el trabajo de Carlos se centró al completo (quitando lo citado anteriormente) en desarrollar lo necesario para que conectara la lógica de debajo con la interfaz. Además, habría que refactorizar código en algunos puntos para que se adaptara y se conectara correctamente con la otra parte.

6.3.3. Contribuciones realizadas por Ricardo

Entre las contribuciones aportadas por Ricardo se destaca:

Diseño e implementación de la arquitectura del simulador

Uno de los objetivos más claros durante el desarrollo del simulador fue facilitar la modificación y extensibilidad de éste. Por ello una atención extra a la arquitectura de este componente fue necesaria desde el principio. Para conseguir una arquitectura con la que se pudieran cumplir los objetivos, Ricardo dedicó bastante tiempo en investigar patrones de diseño y distintas ideas que podía aplicar hasta llegar a lo que se tiene hoy. El resultado es una simulación fácil de extender, testeada y con una buena separación de responsabilidades. De esta manera la interfaz ha quedado completamente separada de la simulación y se puede cambiar el renderizado y los efectos con relativa facilidad.

Enlazado de cambio de escenas, conservación de estado y carga de virus en simulación

Se hizo uso del patrón Singleton en el GameManager de Unity, un patrón muy estándar en el motor, para facilitar el manejo de escenas, entre otras cosas. Para la carga de virus en la simulación, el código se apoya en el parser de pMARS. El simulador tradicional de Corewar, en modo de depuración compila el código de los virus que se le pasan como argumentos y muestra una versión compilada por pantalla. Hacerlo de esta manera evita implementar la compilación en el simulador y la preocupación por cosas que se resuelven durante la compilación (preprocesado de etiquetas, resolución de modificadores, errores de compilación...).

Arquitectura y primeros bocetos de interfaz durante la simulación

La arquitectura de la interfaz durante la simulación permite registrar el controlador de UI a numerosos eventos que emite el simulador (modificación de celdas, ejecución de código..). De esta manera se puede añadir o modificar efectos en reacción a estos eventos. Y, como se ha hecho un sistema de eventos en la simulación, añadir eventos es bastante sencillo por ejemplo:

- Detectar el tipo de estrategia que usa un virus en particular
- Detectar si el valor de un dato decrementa o incrementa o cosas por el estilo

Y, consecuentemente, se puede hacer que la UI reaccione a éstos con facilidad gracias a la implementación modular que se consiguió.

6.3.4. Contribuciones realizadas por Stiven

Entre las contribuciones aportadas por Stiven se destaca:

Administración y gestión del proyecto

Una de las labores principales de Stiven fue llevar a cabo el rol de **Scrum Master** que consiste en la persona que se encarga de gestionar y administrar el proyecto dentro de la metodología Scrum. Sus tareas en este aspecto han sido pues: gestión de Trello (de sus tablas y de todas sus tareas), organizar reuniones, organizar el proyecto debidamente en directorios apropiados, informarse y maneter informados a los compañeros acerca de los requisitos necesarios para cumplimentar esta memoria y otros requisitos indispensables para el TFG, etc.

Además, también hay que resaltar su labor con la propia memoria del proyecto, pues fue quién se encargó de generar una estructura sólida con *Overleaf*, diseñando cada uno de los puntos que se deberían ir rellenando por parte del equipo y repartiendo las tareas entre cada uno de los miembros de manera clara y ordenada. Para ellos, estuvo informándose al respecto sobre el contenido de la memoria, tanto con el director como con ejemplos de otros TFGs de la facultad.

Búsqueda de recursos para el programa

Para llevar a cabo una aplicación de este calibre es necesario conseguir una gran cantidad de recursos. Uno de los objetivos principales del proyecto es crear una aplicación atractiva visualmente. Para ello, Stiven se ha encargado de buscar todos los recursos posibles, procesarlos por Photoshop, cambiar sus colores, transformar vídeos para obtener sus frames, etc. Por otro lado, no bastaría solo con eso, pues hay que probar su impacto dentro de la aplicación, hay que probar su escalado, su combinación con el resto de recursos, su nivel de reutilización en otras capas de la aplicación, etc. Así pues, para poder probar que todos los recursos funcionasen correctamente, se encargó, además, del diseño y de la implementación de la mayor parte de las interfaces de usuario.

Diseño e implementación de los menús

Partiendo del punto anterior, una de las necesidades en la búsqueda de recursos es probar su utilidad. Sin embargo, en los inicios del proyecto, no se disponía de ningún tipo de interfaz. Así, pues Stiven se encargaría de diseñar y crear los primero bocetos para el menú principal y así poder probar cada uno de los recursos encontrados. El primer paso fue crear el *MainMenu* y a raíz de eso se generaría los menús de *Normal1v1* y el del *Torneo*. Posteriormente, pasaría a implementar la lógica de los menús, la forma en la que se generan las transiciones, animaciones, lógica de los botones, etc. Tras esto, Carlos se encargaría de enlazar el menú principal con el GameScene. Finalmente, con una GameScene más funcional, Stiven se encargó de diseñar la interfaz de la escena y, como se ha explicado anteriormente, de probar la combinación de los diferentes *assets* encontrados para dicha escena.

Capítulo 7

Conclusiones y trabajo a futuro

7.1. Conclusiones

Este Trabajo de Fin de Grado se ha afrontado con la motivación de aprender, estudiar e investigar acerca de todo lo posible de cómo crear una aplicación para encontrar una alternativa a ARES. Se han realizado numerosas pruebas sobre ARES con todos los virus de ejemplo que venían. Para ello, se ha instalado y se ha aprendido a usarlo y como funciona. De esta manera, se ha podido destacar lo más relevante de la aplicación y lo que sería más usado de cara a simplificar el programa. Sin embargo, no se han podido completar todos los objetivos del grupo, destacando el modo Torneo.

En lo referente al nivel tecnológico, hemos mejorado nuestra habilidades con Unity y también hemos podido aplicar nuestros conocimientos previos sobre la aplicación y sobre la arquitectura del código. Por lo que concluimos, nos gustaría poder llegar a implantarlo en todos los torneos, ya que puede llegar a motivar más a sus participantes simplemente por el diseño del programa así como atraer nuevos competidores. Para ello, nos gustaría seguir en contacto con la facultad y con los posibles futuros estudiantes que continúen nuestro trabajo para poder aconsejar o explicar cualquier cosa en la medida de lo posible.

7.2. Trabajo a futuro

Finalmente, se han extraído las siguientes ideas destacables para un posible trabajo a futuro:

- **Mejoras en la estética del juego**: la estética del juego es coherente a rasgos generales, pero la verdad es que la conformidad con el resultado de la estética en GameScene no es del 100 %. La combinación de colores consideramos que no es del todo acertada y se podría mejorar para que sea mucho más homogéneo. Por otro lado, también nos hubiera gustado meter más efectos especiales (con el sistema de partículas de Unity, por ejemplo) de manera que lo que suceda durante el combate se viva con mayor intensidad.

- **Implementación del torneo**: de acuerdo a lo que se había planeado, la implementación del torneo iba a tener las siguientes características: lista de participantes, lista de batallas configurables, lista de puntuaciones, cuadro de emparejamientos, botón para continuar con la siguiente batalla en GameScene y finalmente alguna especie de podio para los ganadores del torneo. Por desgracia, solo se consiguió terminar en su totalidad la funcionalidad de la lista de participantes y el diseño de la configuración de batallas, por lo tanto, el resto de características mencionadas quedarían pendientes.

- **Adición de temas**: es común que cada año cambien los aspectos visuales del torneo. Es por ello, que nos hemos esforzado en que el programa sea lo más extensible, accesible y adaptable posible a nuevas versiones. Una de las cosas más interesantes que se podría incluir es la configuración de temas. Ya es común en muchos juegos de cualquier tipo la personalización de las interfaces del juego mediante paquetes temáticos. Pueden cambiar completamente la interfaz o simplemente cambiar los colores.

Para añadirlos al programa, Unity permite la creación de archivos con extensión *.asset* que permiten múltiples configuraciones. En ellos se pueden cargar, por ejemplo, los colores principales que se van a usar, los fondos para cada capa del juego, los *sprites* de los botones, etc. De esta forma, bastaría solo con cambiar el contenido de cada uno de estos paquetes y mediante un parámetro indicarle al juego que utilice un tema u otro.

- **Tutorial:** una de nuestras ideas iniciales fue crear un pequeño tutorial en algún punto del programa que permitiese al usuario aprender con el propio juego. De esta forma se fomentaría más su uso, daría más utilidad al jugador. Al incrementar su uso se podría incluso tener analíticas para mejorar la aplicación.
- **SinglePlayer/Retos:** más allá del simple tutorial que enseña lo básico del código, se podría implementar una serie de retos donde se debe programar un virus que derrote a un tipo concreto. Debido a la filosofía de piedra, papel, tijera que sigue Redcode estos retos enseñan cómo crear diferentes clases de virus dependiendo de la situación o el comportamiento buscado.

Capítulo 7

Conclusions and future work

7.1. Conclusions

This Final Degree Project has been faced with the motivation to learn, study and research about every possible way to develop an app and find an alternative to ARES. We made numerous test with ARES using all the virus available in the examples. To do this, we learnt to use it and how it works. Because of that we have been able to highlight the most relevant aspects of the application and what would be the most used in order to simplify the program. Nevertheless, we were not able to finish all the objectives, being tournament mode the most important.

Regarding the technological level, we have improved our skills with Unity and also we could apply our previous knowledge about the software and code architecture. So we conclude that we hope this project will reach all the tournaments, it could get to motivate participants simply by the design as well as attract new competitors. For this to be possible we wish to keep contact with the faculty and the possible future students that will continue with our work so they can know that they can seek counsel with us if they need it.

7.2. Future work

Finally, we have extracted the following ideas for a possible follow up work in the coming years:

- **Improving the application aesthetic**: the aesthetics of the application are generally consistent, but the truth is that our conformity with the current results in GameScene is not 100 % right now. We consider that the color combination is not the best and could be improved to make it much more homogeneous. On the other hand, we would also like to see more «*fx*» (like the particle systems Unity gives) so that the combat is experienced with more intensity.
- **Tournament implementation**: According to what had been planned, the tournament implementation was going to have the following features: list of contenders, list of configurable battles, lit of scores, pairing tables, continue button for the next battle in GameScene and some kind of podium for the winners. In the end, we only achieved to end the functionality of the creation of the list of contenders and the design of the configuration of battles, therefore the rest of features have to be implemented.
- **Themes**: is really common that every year the visual style of the tournament changes. That is why we took the effort to make it as flexible and adaptable as possible to new versions. One of the most interesting things to include is the theme configuration. Is really common in modern game to give the option to personalize the interface with thematic packages. It could overhaul the interface or simply change the colors. To add it to the application Unity gives you the possibility to create files with the *.asset* extension, this allows you to store multiple data types and settings. You could load, for example, the main colors that the program is going to use for each layer of the games, or the button *sprites*, etc.

With this it would be as simple as change the package content to completely and use a parameter to choose the one you want to use.

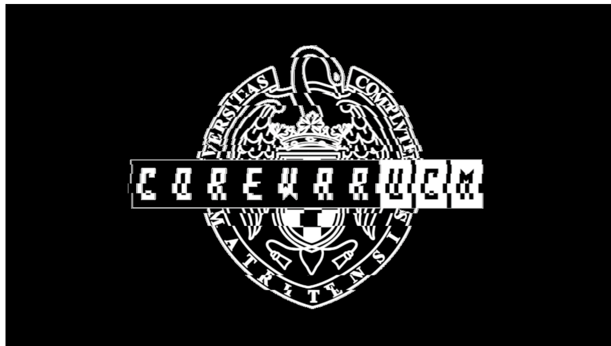
- **Tutorial:** one of the first ideas was to create a little tutorial at some point that could teach the base user how to code with the game. With this its use would be further encouraged , it would give more utility to the player. By increasing its use we could even use analytics to improve the application.
- **SinglePlayer/Challenges:** beyond the simple tutorial that teaches the basics of the code, a series of challenges could be implemented where you must program a virus that defeats a specific type. Due to the rock, paper, scissors philosophy that redcode follows, these challenges teach how to create different kinds of viruses depending on the situation or behavior sought.

Anexos

A - Manual de usuario CorewarUCM

Inicio del juego

Para empezar una partida debes tener en cuenta que este programa no está pensado para realizar *debug* con tus virus. Si tu virus tiene algún error de compilación simplemente no se ejecutará, pero es tu responsabilidad encontrar los fallos en tu archivo. Dicho esto, lo primero que te encuentras al iniciar el juego son las siguientes pantallas y, recuerda, siempre puedes cerrar el programa con el botón **Salir**:



(a) Portada del juego

(b) Menú principal del juego

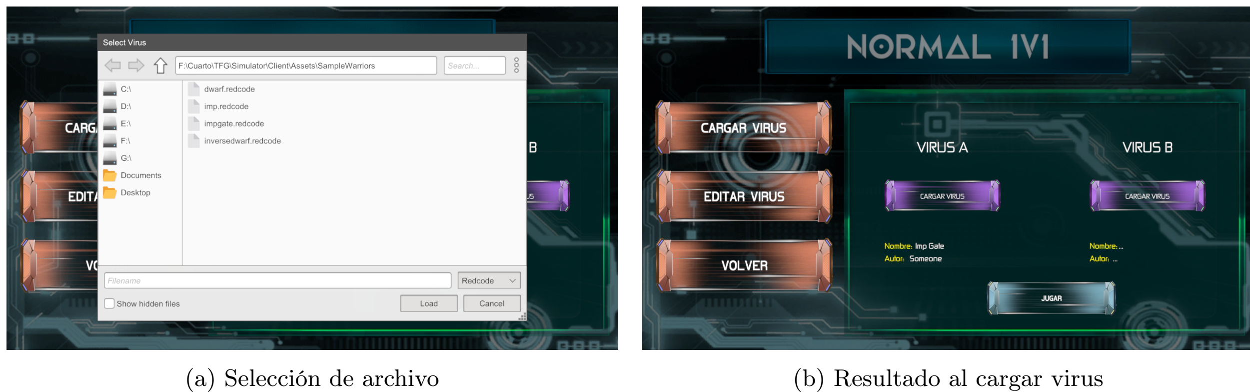
Figura A.1: Inicio del juego

Normal1v1

Dentro del modo Normal1v1 se pueden seleccionar dos opciones: Cargar virus y Editar virus:

Figura 4.4: Interfaz: Normal1v1

Con la opción cargar virus, se despliegan los botones con los que poder agregar tu archivo. Cada uno tiene un botón **Cargar virus** para seleccionar, directamente del explorador de archivos del sistema, el virus con extensión *.redcode* o *.red* que se desea cargar:



(a) Selección de archivo (b) Resultado al cargar virus

Figura A.3: Normal 1v1 - Cargar Virus

Una vez tengas listos los dos integrantes de la batalla puedes proceder a darle al botón **Jugar**, pero hasta que no haya dos archivos cargados, el botón estará deshabilitado.

Además, puedes modificar el nombre y el autor del virus directamente en este menú. Lo único que debes hacer es darle click encima del texto correspondiente y empezar a escribir:

Figura A.4: Cambio del nombre del virus

Con la opción **Editar Virus** puedes crear tu propio virus desde cero o bien puedes cargar uno ya existente y editarlo. De esta forma no tienes por qué abandonar la aplicación para seguir probando tus programas. También podrás añadirle una imagen para asociar al virus y que se muestre durante las batallas. Una vez tengas listo el nuevo virus recuerda darle al botón de guardar. Aquí podrás elegir guardarlo en formato *.red*

(formato estándar) o *.redcode* (formato con imágenes). Si has cargado una imagen y lo guardas como .red, la imagen no se guardará finalmente:

Figura A.5: Editor del virus

Torneo

Es importante destacar que la funcionalidad del modo torneo no está completa, sin embargo, te voy a enseñar cómo funciona.
El modo Torneo muestra una lista de los 32 participantes inicializados a *null*:

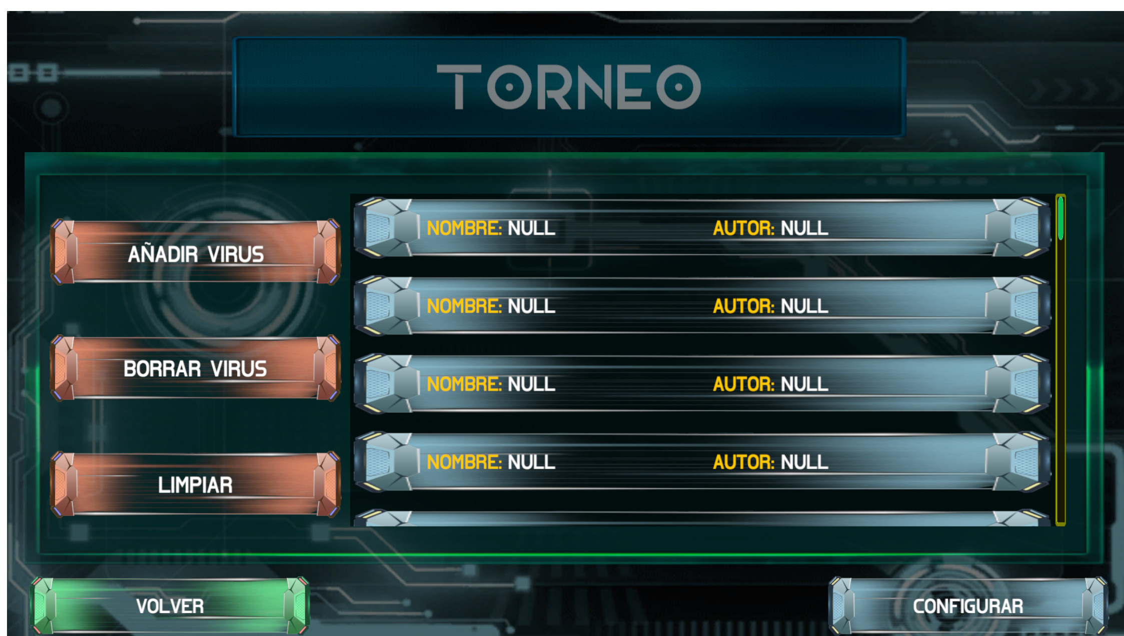


Figura A.6: Interfaz: Torneo

Se pueden añadir nuevos participantes mediante el botón **Añadir Virus**, de manera que se abre el explorador de archivos del sistema para seleccionar el virus con extensión *.redcode* o *.red* que se quiera cargar:

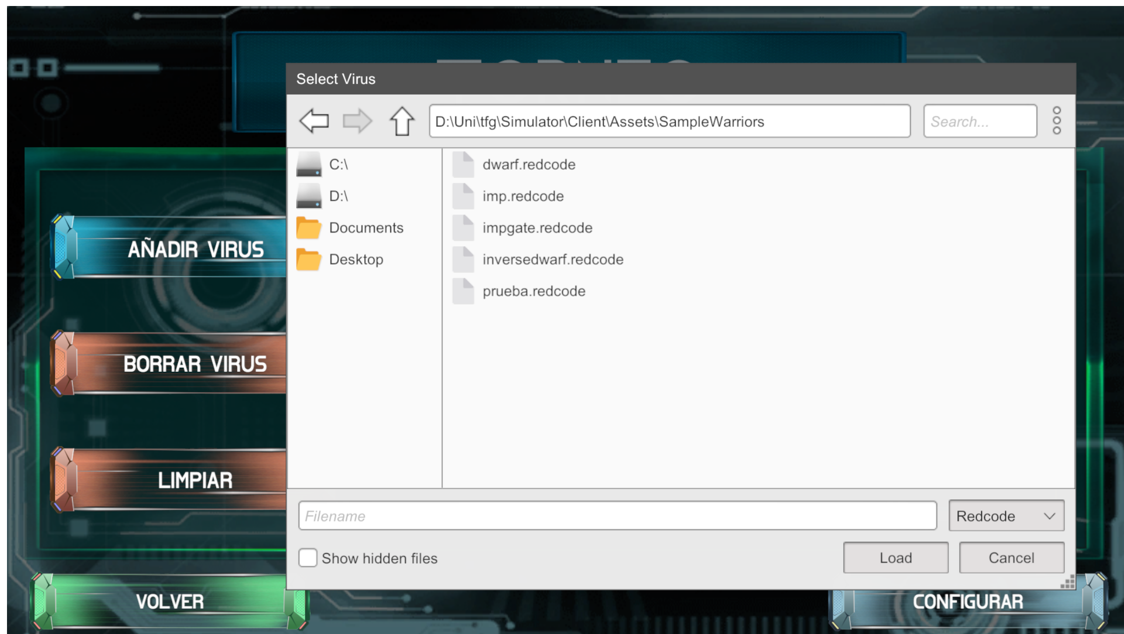


Figura A.7: Selección de archivo - Torneo

Por otro lado, si deseas eliminar un participante, solo debes darle click al virus correspondiente y, seguidamente, darle al botón **Borrar Virus**:

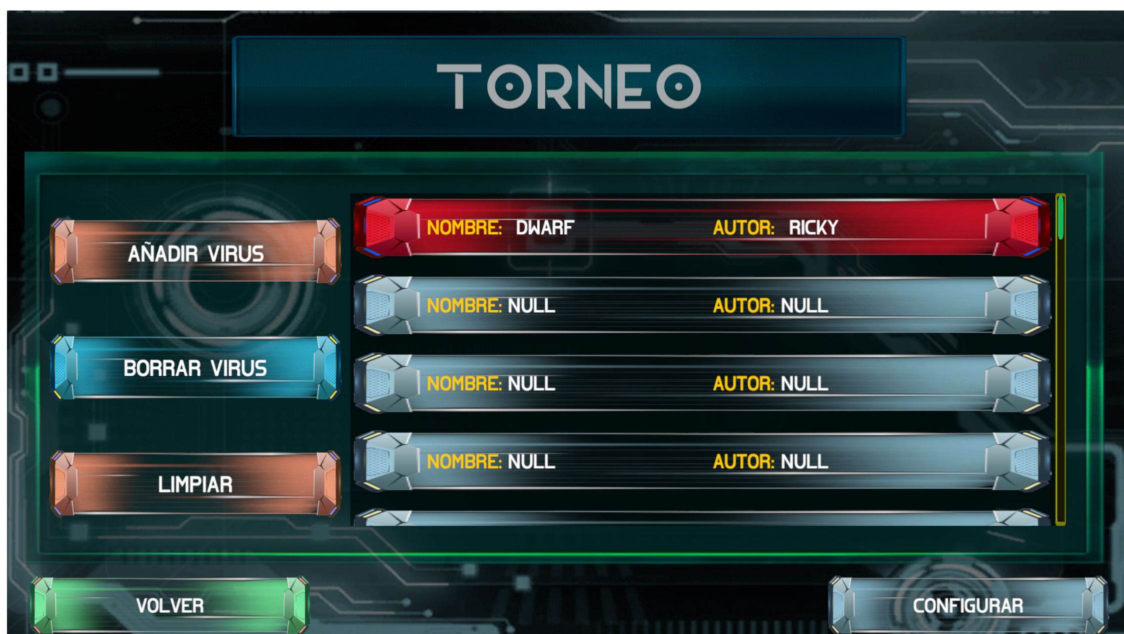


Figura A.8: Eliminación de un participante

Al igual que en el otro modo, el botón *Configurar* permanece deshabilitado hasta que haya al menos 2 participantes activos. Pero una vez se le de al botón, se muestra la interfaz de configuración, donde se pueden añadir batallas y decidir quién luchará en el encuentro. Es similar al funcionamiento de la interfaz

anterior, pero en lugar de añadiendo virus, se añaden batallas. Sin embargo, la funcionalidad de la interfaz no se pudo completar, por lo que su interactividad está deshabilitada.

Figura A.9: Interfaz: Configuración del torneo

Escena del juego

Cuando hayas terminado de configurar los virus que se van a enfrentar y le des al botón para jugar, accederás al siguiente menú. Aquí lo único que debes hacer es darle al botón de **Comenzar** para empezar la ejecución.

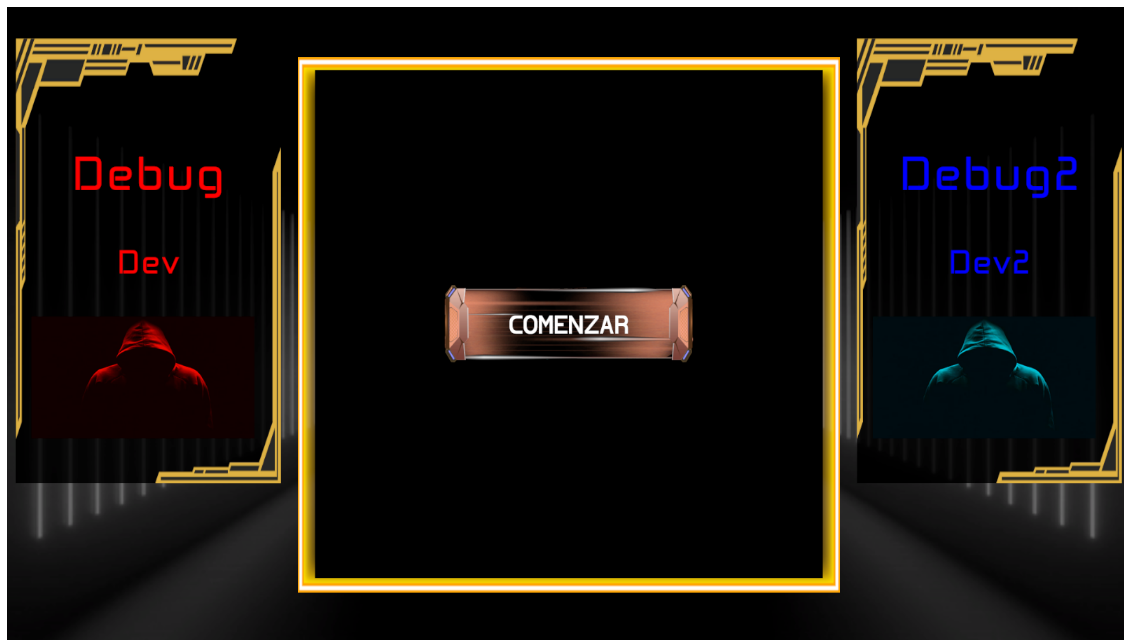


Figura A.10: Inicio de la batalla

Posteriormente, durante la ejecución del programa tienes 4 opciones: esperar a que se termine la ejecución,

volver al menú principal, modificar la velocidad de la simulación o o elegir un ganador. El valor numérico de la zona inferior central de la pantalla representa la velocidad actual, medida en instrucciones por segundo.

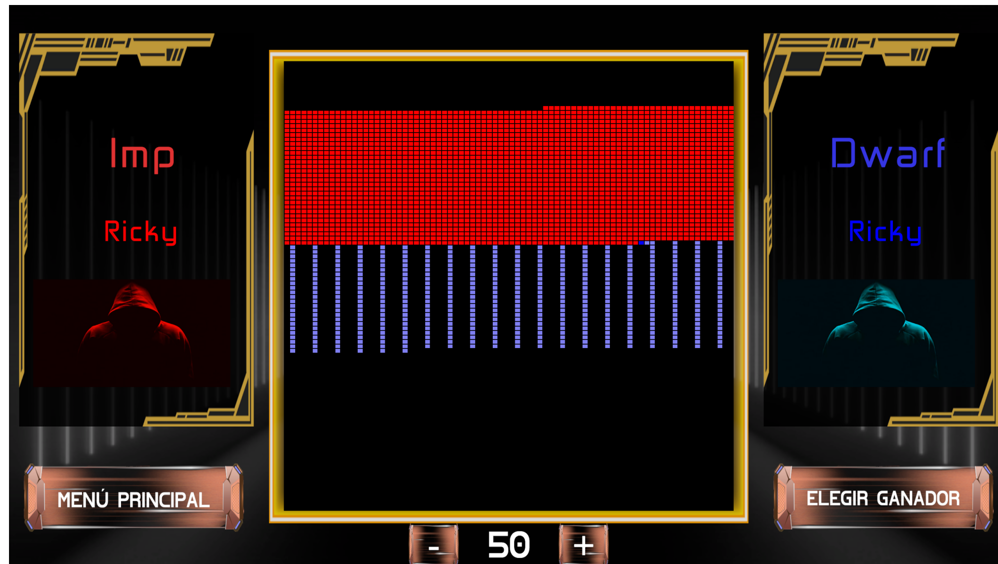


Figura A.11: Batalla entre dos virus

El último caso, «elegir un ganador», o está sobre todo pensando para el modo torneo, puesto que en el evento realizado por la Universidad Complutense de Madrid, CorewarUCM, los jueces son los que dictaminan muchas veces quién es el ganador del combate. En cualquier caso, si se elige un ganador, en la posterior pantalla de resultados aparecerá el botón para elegir ganador desactivado. Además, el botón **Continuar** aparecerá desactivado si se accede desde *Normal1v1* ya que dicho botón sirve para continuar con la siguiente batalla durante un torneo:

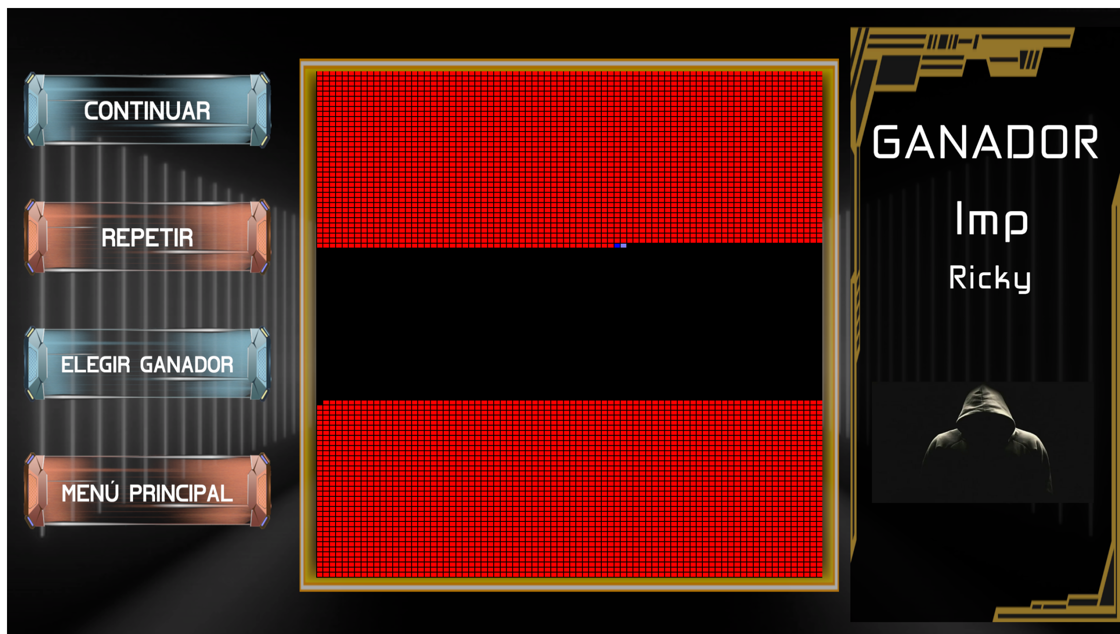


Figura A.12: Resultados de la batalla

Además, puedes repetir la batalla o volver al menú principal una vez acabada la batalla.

Índice de figuras

3.1. Esquema metodología Scrum . . . 23
3.2. Interfaz de Ares . . . 24
3.3. Una de las primeras versiones de Trello . . . 25

4.1a. Portada del juego . . . 29
4.1b. Menú principal del juego . . . 29
4.2. Interfaz: Normal1v1 . . . 30
4.3. Interfaz: Torneo . . . 30
4.4. Interfaz: Configuración del torneo . . . 31
4.5. Batalla entre dos virus . . . 31
4.6. Volver al menú principal . . . 32
4.7. Elegir ganador . . . 32
4.8. Resultados de la batalla . . . 33
4.9. Normal 1v1 - Cargar Virus . . . 33
4.10. Editor del virus . . . 34
4.11. Adición y eliminación de un participante en el torneo . . . 34

5.1. Escenas de Unity . . . 37
5.2. Escena de Unity: Loader . . . 37
5.3. Escena de Unity: MainMenu . . . 38
5.4. Ejemplo del componente *Button* . . . 38
5.5. Esquema de arquitectura del simulador. . . . 40
5.6. Escena de Unity: GameScene . . . 42

A.1. Inicio del juego . . . 57
4.4. Interfaz: Normal1v1 . . . 57
A.3. Normal 1v1 - Cargar Virus . . . 58
A.4. Cambio del nombre del virus . . . 58
A.5. Editor del virus . . . 59
A.6. Interfaz: Torneo . . . 59
A.7. Selección de archivo - Torneo . . . 60
A.8. Eliminación de un participante . . . 60
A.9. Interfaz: Configuración del torneo . . . 61
A.10. Inicio de la batalla . . . 61
A.11. Batalla entre dos virus . . . 62
A.12. Resultados de la batalla . . . 62

Bibliografía

[1] https://corewar.co.uk/ares.htm [Último acceso: 17.05.2022]

[2] https://corewar.ucm.es/ [Último acceso: 17.05.2022]

[3] https://fdist.ucm.es/corewar/CoreWar.pdf [Último acceso: 17.05.2022]

[4] https://unity.com/ [Último acceso: 17.05.2022]

[5] https://assetstore.unity.com/ [Último acceso: 17.05.2022]

[6] https://github.com/ [Último acceso: 17.05.2022]

[7] https://trello.com/ [Último acceso: 17.05.2022]

[8] https://asana.com/es/resources/what-is-kanban [Último acceso: 17.05.2022]

[9] https://discord.com/ [Último acceso: 17.05.2022]

[10] https://www.jetbrains.com/es-es/rider/ [Último acceso: 17.05.2022]

[11] https://vyznev.net/corewar/guide.html [Último acceso: 17.05.2022]

[12] https://www.adobe.com/es/products/photoshop.html [Último acceso: 17.05.2022]

[13] https://www.overleaf.com/ [Último acceso: 17.05.2022]

[14] https://docs.microsoft.com/es-es/dotnet/csharp/ [Último acceso: 17.05.2022]

[15] https://profile.es/blog/principios-solid-desarrollo-software-calidad/ [Último acceso: 17.05.2022]

[16] https://www.canva.com/_/aprende/49-tipografias-futuristas-gratis/ [Último acceso: 17.05.2022]

[17] https://mylivewallpapers.com [Último acceso: 17.05.2022]

[18] https://pixabay.com/ [Último acceso: 17.05.2022]

[19] https://png.is/ [Último acceso: 17.05.2022]

[20] https://assetstore.unity.com/packages/tools/gui/runtime-file-browser-113006#description [Último acceso: 17.05.2022]