Trabajo de Fin de Grado

Modo multijugador en Unity3D

Multiplayer mode in Unity3D

Juan Antonio Oliva Pérez

La Laguna, 31 de agosto de 2018

Dña. **Carina Soledad González González**, con N.I.F. 54.064.251-Z profesora titular adscrita al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutora

C E R T I F I C A

Que la presente memoria titulada:

"Modo multijugador en Unity3D"

ha sido realizada bajo su dirección por D. **Juan Antonio Oliva Pérez**,

con N.I.F. 79.071.210-Q.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firma la presente en La Laguna a 31 de agosto de 2018

Agradecimientos

A mi madre **Coco**, por haber estado siempre ahí, ayudándome y apoyándome durante todos estos años. Nunca le podré agradecer lo suficiente todo lo que ha hecho por mí.

A mi padre **Nacho**, por apoyarme y ayudarme siempre que lo he necesitado.

A **Carina**, por haber aceptado este proyecto y ayudarme a terminar la memoria.

A **Alberto Erice**, por permitirme seguir usando su videojuego para la realización de este proyecto.

Licencia

Resumen

Los juegos online multijugador gozan hoy en día de una enorme popularidad y su éxito es tal que puede decirse que prácticamente todos los juegos conocidos tienen su versión online. Por tanto, dado mi interés en el mundo de los videojuegos, es un área que quiero y necesito conocer.

En consecuencia, el objetivo de este Trabajo de Fin de Grado es implementar el modo multijugador a un juego ya existente, que sólo tiene la opción de jugar en local contra la inteligencia artificial.

El videojuego base del que parto es un juego de estrategia, similar al clásico juego de mesa Risk, en cuya versión para un solo jugador trabajé durante mis prácticas de empresa.

Para realizar este trabajo primero se ha implementado una sala de espera, donde los jugadores crean o buscan partidas a las que unirse, para posteriormente iniciar la partida.

Una vez en la partida se ha implementado un sistema de paso de mensajes entre cliente y servidor y se ha conseguido que la partida este perfectamente sincronizada.

*Para realizar este proyecto se ha usado la plataforma **Unity3D** estudiándose especialmente sus clases y técnicas de **Networking** para los juegos multijugador. Tras el trabajo que hemos realizado hemos conseguido el principal objetivo que nos propusimos, es decir, que puedan jugar varios jugadores en una misma partida, aunque el juego todavía necesita una revisión en su diseño y en las mecánicas del mismo. Trabajar en este proyecto me ha ayudado a aumentar y profundizar mis conocimientos en las técnicas relacionadas con el mundo de la creación de videojuegos, un aprendizaje que espero ir aumentando para poder ponerlo en práctica en lo que, sin duda, me gustaría que fuera mi futuro profesional: la adaptación, mejora o creación de videojuegos.*

Palabras clave: Multijugador, videojuego, sala de espera, cliente, servidor, Unity3D, Networking.

Abstract

Multiplayer online games are in full force nowadays and they are such a hit that it could be said nearly all known games have an online version. Consequently, and due to my particular interest for the videogames world, it is an area that I want and need to manage.

Therefore, this Final Degree Project aims at implementing multiplayer mode to an already existing game, which has as the only choice, playing local against an artificial intelligence.

The videogame target I'm using is a strategy game, similar to the classic table game Risk, whose one-player version I worked with during my internship.

In order to make this work possible, firstly a lobby needs to be created for players to create or search for matches to follow, to initiate the game later on.

Once in the match, a message pathway has been implemented between client and server and a perfect synchronicity has been achieved.

To elaborate this task, platform ***Unity3D*** *has been used, with special interest in its sorts and techniques of* ***Networking*** *for multiplayer games. Finally, the main objective proposed has been fulfilled, that is, several players involved in the same match, if only the game needs further revision, specifically regarding design and mechanisms. Working on this project has helped me increase and deepen my knowledge in techniques related to the world of creating videogames, a discipline I hope to improve in time in order to become professional in adapting, improving of creating videogames in the near future.*

Keywords: Multiplayer, Videogame, Lobby, Client, Server, Unity3D, Networking

Índice general

Índice de figuras

Capítulo 1
Introducción

1.1 Descripción y Motivación

La expansión de Internet a principios de los 90 trajo consigo una nueva manera de acercarse al mundo de los videojuegos[(1)], porque a partir de ese momento se abría la posibilidad de poder jugar online, no solo con videojuegos de navegador, que se juegan desde la web, sino también -lo que ha supuesto una verdadera revolución- con videojuegos multijugador en los que pueden interactuar jugadores desde cualquier lugar del mundo.

Los juegos online multijugador[(2)] gozan hoy en día de una enorme popularidad y su éxito es tal que puede decirse que prácticamente todos los juegos conocidos tienen su versión online y la lista de adaptaciones y nuevas creaciones no deja de crecer[(3-4)]. Enfrentarse a otros jugadores es, sin duda, mucho más entretenido, adictivo y emocionante que competir solo uno mismo con una máquina, ya que nos permite interactuar con otras personas y comunicarnos con ellas. Esto ha sido aprovechado por los profesionales de la enseñanza y algunos terapeutas para trabajar habilidades sociales y de colaboración[(5)]. Además, la posibilidad de jugar sin límites geográficos ha hecho del inglés la lengua principal de este mundo virtual, lo que no deja de ser una buena manera de perfeccionar este idioma[(6)].

Pero sobre todo los juegos online son competitivos. No solo te enfrentas

por el puro placer de la competición, sino también puedes hacerlo de una forma más profesional e incluso llegar a ganar mucho dinero. Durante la década de los 70 y los 80 se realizaron competiciones de videojuegos sobre todo en Estados Unidos que tuvieron un enorme éxito, a partir de los 90 el incremento de las conexiones a Internet permitió dar el salto a las competiciones online y aparecieron los primeros torneos de e-Sports(7). Los e-Sports son competiciones de juegos multijugador que permiten el enfrentamiento directo entre dos o más participantes; estos compiten en igualdad de condiciones, ganando el que demuestra poseer más habilidad. Actualmente se cuentan por miles los jugadores que participan en las competiciones oficiales que son retransmitidas por la red y son seguidas por millones de personas, lo que genera unas ganancias de muchos millones de dólares. Los torneos profesionales cuentan, como los deportes en vivo, con entrenadores, jugadores suplentes, patrocinadores, etc. Además este tipo de juegos puede realizarse a través de cualquier plataforma capaz de conectarse a la red como el ordenador, consolas (PS4, Xbox,...), e incluso tablets y móviles, lo que permite su seguimiento y participación desde cualquier lugar.

Como jugador desde pequeño aficionado a los videojuegos, la carrera que elegí, Ingeniería Informática, me ha permitido ver desde dentro este mundo, despertando mi interés por la otra cara del espejo, el del análisis, perfeccionamiento o creación de videojuegos, especialmente para el modo multijugador. Una línea de trabajo en la que me gustaría apostar mi futuro profesional.

1.2 Objetivos

El objetivo de este Trabajo de Fin de Grado es implementar un "Modo multijugador Online" a un videojuego inspirado en el juego de estrategia *Risk*. El desarrollo de este juego para un solo jugador conformó la línea principal de investigación de mis prácticas de empresa. Se trata, pues, de dar un paso más,

no sólo revisando y perfeccionando lo ya hecho en esa primera etapa sino añadiendo, como ya he señalado, la opción multijugador al mismo, sin duda, un objetivo mucho más ambicioso que conlleva una serie de tareas complejas que iré explicando a lo largo de este Trabajo de Fin de Grado.

Para poder implementar el modo multijugador Online tendré que estudiar y analizar las distintas técnicas disponibles en la plataforma que voy a usar (**Unity3D**)[8], así como emplear técnicas de:

- Concurrencias.
- Redes:
    - Configuración de un **Servidor**: uno de los jugadores ejecutaría el código en modo **Host**, siendo **Cliente** y **Servidor**. Esto implica que muchas veces habrá pasos de mensajes desde el mismo dispositivo.
    - Configuración de un **Cliente**, que sería la configuración para el resto de jugadores.
- Gestión de memoria.
- Paso de mensajes entre los **Clientes** y el **Servidor**.

Los objetivos concretos a los que se encamina mi trabajo son los siguientes:

- Crear una "sala de espera" con las siguientes características:
    - Será el lugar donde todos los **Clientes** se conectan al juego (cada **Cliente** es un jugador de la partida). La partida puede contar con un mínimo de 2 jugadores y un máximo de 8.
    - Aquí los jugadores configurarán sus datos.
    - Se realiza la gestión que comprueba que todos los **Clientes** tienen los datos descargados y están listos para empezar.
    - Se implementará un sistema de pasos de mensajes (chat).

- El juego:
  - Arranque del juego.
  - Detectar movimientos de los jugadores.
  - Detectar el "fin de movimiento" del jugador.
  - Lanzar el "fin de turno" y "ejecutar" las órdenes de los jugadores.
  - Gestionar el paso de mensajes entre **Cliente** y **Host**.

1.3 Estructura del documento

La memoria del Trabajo Fin de Grado que aquí presentamos consta de 7 capítulos:

- **Introducción.** Dividida en tres apartados, explico en ellos los motivos por los que me interesa este proyecto, los objetivos a cumplir y la estructura del documento que es el que aquí desarrollo.

- **Antecedentes.** Hablo aquí del proyecto base a partir del cual he podido realizar este Trabajo de Fin de Grado. Ese proyecto se centró en el videojuego *Dominance* inspirado en el conocido juego de mesa *Risk*. Añado la descripción de las principales reglas de ambos juegos para que el lector pueda seguir sin dificultad la nueva versión que presento en este trabajo.

- **Herramientas y Tecnologías.** Dedico este capítulo a la descripción de las herramientas que he usado para realizar el proyecto (Unity3D y Networking) explicando las características más importantes y necesarias para poder entender el contenido del siguiente capítulo.

- **Desarrollo.** Este capítulo es el punto central de mi trabajo: la implementación del modo multijugador al videojuego *Dominance*. En él muestro cómo se ha ido desarrollado el trabajo, la metodología utilizada y los resultados que he obtenido.

- **Conclusiones y líneas futuras.** En las conclusiones repaso los resultados obtenidos con este proyecto con el fin de destacar lo que he aprendido y sus aplicaciones futuras pensando en la más que probable revisión de este juego y las nuevas implementaciones susceptibles de ser añadidas, así como la aplicación de estos resultados en otros proyectos afines.

- **Summary and conclusions.** Como indica el título, hago aquí un resumen del proyecto y de las conclusiones en inglés, tal y como es preceptivo.

- **Bibliografía.** En este último capítulo recogemos todas las referencias de la bibliografía consultada.

Capítulo 2

Antecedentes y estado actual.

2.1 Contexto

La base inicial de este Trabajo de Fin de Grado es un videojuego llamado *Dominance*. Se trata de un videojuego en el que estuve trabajando durante las Prácticas de Empresa realizadas durante el curso académico 2016-2017, desarrollando el modo de juego de un solo jugador contra una o varias IA (Inteligencia Artificial). La licencia de este juego pertenece a *Adamantite Software Factory,* que es una empresa de desarrollo de videojuegos para Pc/Mac y dispositivos móviles, especialmente orientada a juegos de estrategia/wargames.

Dominance es pues, un videojuego de estrategia[9] en el que hay que ir conquistando distintas regiones geográficas de un mapa específico, consiguiendo recursos y comprando unidades para aumentar el ejército que permitirá al jugador conseguir los objetivos. El juego está inspirado en el exitoso *Risk*, que desde el año 2000 ofrece versiones para para PS2, Xbox 360, PS3, iPhone, PC, Xbox One y PS4.

2.2 Risk

Risk[10] es un clásico juego de mesa[11] de estrategia y dominación del mundo en el que pueden jugar de dos a seis jugadores. Fue creado por el

director de cine francés Albert Lamorisse en 1950 y puesto a la venta en Francia en 1957, bajo el título *La Conquête du Monde*. Un año después, la empresa *Parker Brothers* adquiere los derechos del juego y tras realizar algunas modificaciones y cambiar su nombre lo pone a la venta en 1959. A lo largo de los años se han ido sucediendo las ediciones con diversas modificaciones que conciernen sobre todo a la presentación de las fichas de juego. Junto a las ediciones del juego tradicional se han sacado ediciones especiales o conmemorativas en las que también se han introducido algunas variantes, como por ejemplo las que tienen como tema central a *Napoleón*, *El Señor de los Anillos*, *Star Wars*, *Narnia*, o la dedicada a *Juego de Tronos* aparecida en 2015.

Figura 1. Mi primer juego Risk

El tablero representa al mundo dividido en 6 continentes con un color distintivo; cada continente se divide a su vez en territorios hasta sumar 42: 6 para África, 12 para Asia, 4 para Australia, 7 para Europa, 9 para Norteamérica y 4 para Sudamérica. Las miniaturas representan a los ejércitos, que tienen tres tipos de unidades: Infantería, Caballería y Artillería. Además, hay dos juegos de cartas, unas que representan a cada uno de los territorios

más dos comodines y 14 que señalan un objetivo que el jugador debe alcanzar para ganar la partida. Por último, hay tres dados rojos que se usan para el ataque y dos azules para la defensa. Para iniciar el juego se nombra un general entre los jugadores que controlará las cartas y colocará los refuerzos, se reparten los territorios según las cartas que toquen a cada jugador y se montan las fuerzas que le correspondan. A partir de aquí comienza el juego propiamente dicho, en el que se piden refuerzos, se entra en combate y se lucha para alcanzar el objetivo final o misión secreta determinado por la carta objetivo que haya tocado a cada jugador.

2.3 *Dominance* para un solo jugador

Con el fin de una mejor comprensión del videojuego *Dominance* y de los elementos que lo componen, a continuación expondremos los pasos a seguir en una partida contra la IA (Inteligencia Artificial), explicando las distintas funciones del mismo. Siguiendo su desarrollo podremos darnos cuenta de que las reglas del juego de mesa original[10] son fácilmente reconocibles.

Al ejecutar el juego nos aparece un menú principal con 5 opciones:

Figura 2. Menú Principal

- **Nueva Partida**: Para empezar a configurar una partida.
- **Cargar Partida**: Para retomar una partida que hayamos guardado previamente.
- **Multijugador**: Aquí es donde llevaremos a cabo los objetivos de este Trabajo de Fin de Grado, por lo tanto, una vez que lo desarrollemos tal como explicaremos en los capítulos siguientes, deberá llevarnos a la escena donde se encuentra la sala de espera para que los jugadores puedan conectarse y unirse a la partida.
- **Créditos**: Nos muestra las personas implicadas en el videojuego.
- **Salir**: Para cerrar la aplicación.

Tras iniciar una nueva partida se pasa a la pantalla de configuración:

Figura 3. Pantalla de configuración

- Selección de jugadores: en el panel de la izquierda se selecciona el número de jugadores que participarán en la partida (el jugador real y sus oponentes IA) y se les asigna un color.

- Datos de la partida: el panel de la derecha indica el número de regiones iniciales con las que empieza cada jugador, la fuerza de las unidades neutrales del mapa[1] y la dificultad de la Inteligencia Artificial a la que te enfrentas.
- En este mismo panel el botón “Back” permite volver al menú principal y el botón “Start” comenzar la partida.

Figura 4. Interfaz y mapa del juego

Una vez ha empezado la partida podemos destacar varios elementos:

- Las regiones marcadas con una bandera son las regiones conquistadas por el jugador, cada una del color que se le asignó al principio.

[1] Las unidades neutrales son las unidades que hay inicialmente en una región que deben ser derrotadas para poder capturarla.

- Junto a la bandera aparece un contador con el número de unidades que se poseen en esa región.
- En la parte superior de la pantalla se encuentran los recursos de que dispone el jugador (Oro, Hierro y Maná). Estos recursos se usan para adquirir unidades para nuestro ejército en la tienda.
- A la derecha se encuentran tres botones que se encargan de (en orden descendente) el menú de pausa, finalizar el turno y abrir la tienda para comprar unidades.

Cuando abrimos la tienda aparecen bajo los recursos de los que disponemos las unidades del ejército que ofrece nuestro juego:

Figura 5. Pantalla de la tienda

Las unidades, ordenadas de débil a fuerte, son: Milicia, Guerrero, Caballero, Mago y Gigante. Podemos seleccionar el número de unidades que queremos comprar de entre las disponibles, es decir, las que aparecen iluminadas, pulsando el contador que se encuentra debajo de cada una de

ellas y según los recursos de que dispongamos. Se pulsa el botón verde de la esquina superior derecha para aceptar la compra o el botón rojo de la esquina superior izquierda para cancelar la operación. Las unidades compradas podrán colocarse en cualquiera de las regiones que tenemos conquistadas.

Una vez colocadas en la región elegida, seleccionamos una región adyacente y decidimos con cuantas unidades queremos atacar para conquistarla. Una vez hemos terminado de hacer nuestros movimientos (comprar, colocar, mover o atacar) pulsamos el botón para finalizar el turno, se resuelven los ataques y se muestra un resumen con las batallas que han ocurrido durante ese turno.

Figura 6. Información del final de turno

Al pulsar el botón "play" de cada batalla, el jugador puede ver una representación de los combates que han tenido lugar en ella.

En una batalla, cada unidad que compone el ejército atacante lucha de forma individual contra una unidad del ejército defensor. Cada vez que una de estas muere viene la siguiente, hasta que uno de los ejércitos se quede sin unidades.

Figura 7. Representación del combate

Una vez terminada la presentación, se muestra un resumen indicando el número de vencedores, el de derrotados y cuántas unidades sobreviven.

Figura 8. Resultado de la batalla

La partida termina cuando un jugador ha derrotado/conquistado todas las regiones de los demás jugadores.

Esta es la estructura y funcionamiento del juego tal y como se encuentra en estos momentos. A partir de aquí se llevará a cabo el eje central de este trabajo: implementar el modo multijugador.

Capítulo 3

Herramientas y tecnologías

3.1 Entornos de desarrollo

Para la realización de este Trabajo de Fin de Grado se ha empleado **Unity3D**[8] como principal entorno de desarrollo. **Unity3D** es un motor para el desarrollo de videojuegos creado por *Unity Technologies*, que nos permite compilar y exportar el resultado de nuestro trabajo a distintas plataformas como PC, Android, iOS, consolas, etc.

El Software es gratuito y contiene todas las características necesarias para desarrollar videojuegos complejos. Permite así crear juegos tanto en 2D como en 3D, y también crear scripts, animaciones, inteligencia artificial, iluminación, sonido, etc.

Unity3D también nos ofrece un extenso manual que explica todos sus elementos, así como una gran documentación correspondiente a la parte de programación, lo que nos facilita bastante al aprendizaje y desempeño del trabajo a realizar.

Otro entorno de desarrollo que usamos es **Visual Studio 2017**[12], desarrollado por *Microsoft* capaz de soportar diversos lenguajes de programación. Contiene un editor de código que soporta *IntelliSense* (la herramienta para autocompletar código) y elementos de refactorización de código.

Visual Studio 2017 viene integrado en **Unity3D** y se usa sobre todo para el desarrollo de los scripts. Los lenguajes de programación soportados en **Unity3D** son C# y JavaScript. En nuestro caso usamos C# que es con el que estamos más familiarizados.

A continuación, veremos algunas de las características[13] más importantes de **Unity3D** que usamos en nuestro juego.

3.2 Características de Unity3D

- **Assets:** Son todos los elementos que componen nuestro proyecto. Pueden ser creados en el propio **Unity3D** o ser importados de fuera, como un modelo 3D, un archivo de audio, imágenes, etc. **Unity3D** también posee una tienda llamada *Assetstore* donde se puede adquirir una gran variedad de estos **Assets** para el juego en cuestión, como alguna textura o animación y en muchos casos de forma gratuita.

- **Escena:** Las escenas contienen los entornos y menús de nuestro juego. Se usan sobre todo para definir las distintas "pantallas" por las que el usuario o jugador irá moviéndose durante el juego.

- **GameObject:** Son todos los elementos que se añaden a la escena, a los que se les añaden otros elementos llamados **Components**, que son los que definen el comportamiento del **GameObject.** Un **GameObject** puede ser una luz, un personaje, un sonido, etc.

- **Script:** Cuando los **Components** que vienen integrados en **Unity3D** no son suficientes para lograr el funcionamiento deseado, se pueden crear otros mediante los **Scripts**. Los **Scripts** son archivos de código que hemos programado nosotros mismos usando **Visual Studio 2017** o

cualquier otro editor, para posteriormente incluirlo en el proyecto como un **Asset** y añadirlo al **GameObject** que se desee.

- **Prefabs:** Un **Prefab** o "elemento prefabricado" es un **Asset** que diseñamos previamente y guardamos como "plantilla", para que pueda ser usado posteriormente en cualquier escena. Es especialmente útil cuando se necesita generar varios elementos en el juego que son básicamente iguales, por ejemplo las banderas de nuestro juego. Tiene la ventaja de que si modificas este **Prefab** también se modifican todos los elementos creados a partir de ese **Prefab**.

- **Coroutines:** Las **Coroutines** o "Corrutinas" son funciones que se ejecutan en segundo plano del código principal, lo que se conoce comúnmente como "hilos". Se usan para que el juego realice algunas acciones necesarias sin que se "congele" esperando a que se termine de ejecutar.

3.3 El Network de Unity3D

Lo más interesante que nos ofrece **Unity3D** en relación con este proyecto son sus librerías y elementos relacionados con **Multiplayer** y **Networking**, donde podemos destacar un API scripting de alto nivel[14] (**High Level scripting API** o **HLAPI**), que es un sistema que nos permite construir las capacidades necesarias para los juegos de **Unity3D** en red.

Gracias a **HLAPI** tendremos acceso a comandos que cubren las tareas básicas para los juegos multijugador, que se encuentran bajo el espacio de nombres **UnityEngine.Networking**.

El sistema **HLAPI** incluye:

- Un **Servidor**: Es la instancia a la que se conectarán los jugadores que quieran jugar juntos en la misma partida. El **Servidor** debe encargarse de recoger los datos que van produciendo los jugadores y de transmitirlos a todos los demás en perfecta sincronía, como por ejemplo el número de unidades de cada jugador.

  El **Servidor** puede ser de dos tipos:

  - **Servidor dedicado:** Es la instancia del juego dedicada únicamente a actuar como servidor.
  - **Host**: Se refiere al cliente que puede actuar tanto de **Servidor** como de **Cliente.** En este caso el **Servidor** y el **Cliente** local pueden comunicarse entre sí usando funciones directas, mientras que el resto de **Clientes** estarían conectados en remoto y usarían las funciones de **Network** que nos proporciona **HLAPI**.

- Los **Clientes**: Son las instancias del juego que se conectan al **Servidor**, normalmente desde otro ordenador o dispositivo distinto al de este.

La clase más importante de **HLAPI** para gestionar los elementos de red es **NetworkManager**[15], que nos proporciona los siguientes métodos simples para iniciar o detener los **Clientes** o **Servidores:**

- **StartClient** y **StopClient:** Se usan para iniciar o parar el **Cliente**, usando por defecto los atributos **networkAddress** y **networkPort** que tengamos configurados en el **NetworkManager**. Aunque en nuestro caso usaremos los del atributo **matchMaker**, tal como se explicará más adelante.

- **StartServer** y **StopServer:** Se usan para iniciar o parar el **Servidor**, usando el atributo **networkPort** como el puerto de escucha.

- **StartHost** y **StopHost**: Sirven para iniciar o parar el **Host** (**Cliente** y **Servidor**). Serán los que usaremos en nuestro juego junto al de los **Clientes**.

- **StartMatchMaker** y **StopMatchMaker:** Estos métodos inician o detienen el **matchMaker**, utilizado para buscar partidas disponibles o dejar de buscar.

Lo reseñado hasta aquí junto con los **Servicios de Multijugador**[16] -uno de los varios **Servicios de Internet** que nos proporciona **Unity3D**- nos permitirá implementar la forma de conectar a los jugadores de una forma relativamente sencilla y nos proporciona las herramientas necesarias para crear las partidas, listar las partidas a las que unirse, etc.
Para integrar estos servicios con **HLAPI** se usa la clase **NetworkMatch**[17] que viene incluida en el **NetworkManager** -el **matchMaker** mencionado más arriba- utilizando concretamente los siguientes métodos:

- **CreateMatch**: Método que crea una nueva partida; el cliente que llama a este método se convierte en el **Host.** Este a su vez llama al "callback" **OnMatchCreate** que nos indicará si tuvo éxito, lo que llamaría a la función **StartHost**, o nos mostraría el error en el caso de que fallara.

- **ListMatches**: Método que nos devuelve una lista de partidas creadas y disponibles para que se unan jugadores. Este método llama a la función "callback" **OnMatchList,** donde puede verse el listado de tipo **MatchInfoSnapshot** que contiene toda la información correspondiente

a la partida y lo podemos usar para unirnos a ella. Para obtener este listado se debe haber llamado previamente a la función **StartMatchMaker**.

- **JoinMatch:** método que nos indica que el cliente que lo llamó desea unirse a una partida específica. Esta función debe llamarse después de obtener los resultados del método **ListMatches** y haber escogido una partida a la que unirse. En la función "callback" **OnMatchJoined** se comprueba si todo ha ido bien e inicia al jugador como cliente con la función **StartClient**.

Una vez terminado todo lo relacionado con la conexión, los **Clientes** y el **Host** tienen que ser capaces de comunicarse entre sí. Para ello hay una serie de atributos[18] que se pueden añadir como cabecera en los métodos o atributos de clase, para darles determinado comportamiento, la sintaxis se indica en el siguiente esquema:

```
[Atributo]
public void Funcion(){}
```

Los atributos que se pueden añadir son los siguientes:

- **ClientAttribute:** Se usa en las funciones con la etiqueta [Client] para hacer que sólo se ejecuten en el **Cliente**.

- **ClientRpcAttribute:** Se usa en las funciones con la etiqueta [ClientRpc] para permitirles ser invocadas en los clientes desde el servidor. Además, las funciones tienen que empezar por el prefijo "Rpc".

- **CommandAttribute:** Se usa en los métodos con la etiqueta [Command] para permitirles ser invocados en el servidor desde un cliente. El método tiene que empezar por el prefijo "cmd", que, por seguridad, sólo puede enviarse desde tu "Objeto de jugador", impidiendo así que se puedan controlar elementos de otros jugadores.

- **ServerAttribute:** Se usa en las funciones bajo la etiqueta [Server] para hacer que sólo se puedan ejecutar en el **Servidor.**

- **SyncEventAttribute:** Se usa en eventos bajo la etiqueta [SyncEvent] para permitir que se ejecuten en los **Clientes** siempre que el evento en cuestión sea llamado en el **Servidor**.

- **SyncVarAttribute:** Se usa en las variables de clase bajo la etiqueta [SyncVar] que permite que sean sincronizadas en todos los **Clientes** desde el **Servidor.**

- **TargetRpcAttribute:** Se usa con la etiqueta [TargetRpc]. Es similar a **ClientRpcAttribute** con la diferencia de que aquí se invocaría en un único cliente, que viene determinado por su **NetworkConnection**, usando la variable **connectionToClient**. El prefijo de estas funciones es "Target".

Capítulo 4
Desarrollo

4.1 Sala de espera (Lobby)

La sala de espera o **Lobby** es la zona previa al inicio de una partida donde se reúnen los jugadores que van a participar. En esta zona los jugadores configuran sus datos (nombre y color) e indican cuando están listos para comenzar.

Cuando los jugadores están listos y se cuenta con el número de participantes requerido para la partida (mínimo 2 y máximo 8), esta puede empezar.

4.1.1 Procedimiento

Cuando un jugador entra al modo multijugador tiene 3 opciones:

Figura 9. Opciones de Multijugador

- **Crear**: El jugador que crea la partida será el **Host** (anfitrión) y actuará tanto de **Servidor** como de **Cliente**. Podrá elegir un nombre para la partida, configurar la dificultad de las unidades neutrales del mapa e indicar con cuantas regiones empezarán los jugadores.

Figura 10. Menú de crear partida

- **Buscar**: A los jugadores les aparecerá una pantalla con un listado de los **Servidores** que hay activos para poder elegir uno al que unirse. Para ello verán el nombre de la partida, el número de jugadores que ya se ha unido y los huecos disponibles.

Figura 11. Búsqueda de partidas disponibles

Una vez dentro, los jugadores pueden cambiar el nombre que aparece por defecto y elegir un color de entre los que todavía no se hayan seleccionado. A continuación pulsarán el botón “preparado”. Cuando todos los jugadores están listos se inicia una cuenta atrás de 5 segundos y la partida comienza.

Figura 12. Pantalla del Lobby

Los jugadores, además, una vez conectados al **Lobby** tendrán acceso a una ventana de **chat** en la que podrán hablar con los demás jugadores.

Figura 13. Ventana de Chat

- **Volver**: Esta opción permite volver al menú principal. En el caso de que exista una partida creada, esta se destruirá y desconectará a los jugadores que se hubiesen unido a ella.

4.1.2 Implementación (Lobby.cs)

Para la implementación de la sala de espera (**Lobby**) hemos añadido una nueva escena de nombre **OnlineMatchConfiguration** que será la transición entre la escena del menú principal y la escena de la partida multijugador.

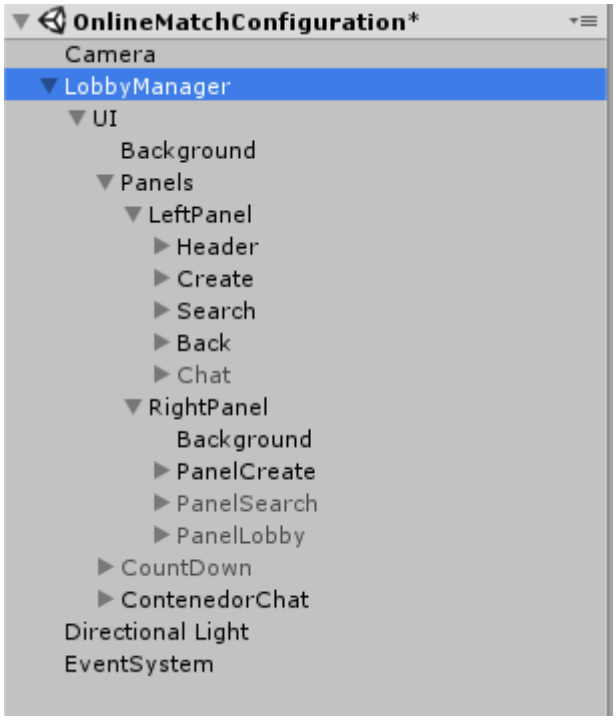

Figura 14. Jerarquía de la escena del Lobby

En esta jerarquía el elemento más importante es **LobbyManager** en el que he creado el script que permitirá manejar las funciones de la sala de espera y que he denominado **Lobby.cs**.

La clase **Lobby.cs** hereda de **NetworkLobbyManager**[19], que es un tipo especializado de **NetworkManager** que nos proporciona las herramientas necesarias para implementar una sala de espera multijugador previa a la escena principal de la partida.

Nos permite configurar un **Network** con:

- Número mínimo y máximo de jugadores.
- Comienzo automático cuando todos los jugadores están listos.
- Opciones para prevenir que un jugador se una a una partida que ya ha comenzado.
- Varias formas para configurar las opciones a escoger por los jugadores que están en el Lobby, como el color o el nombre.

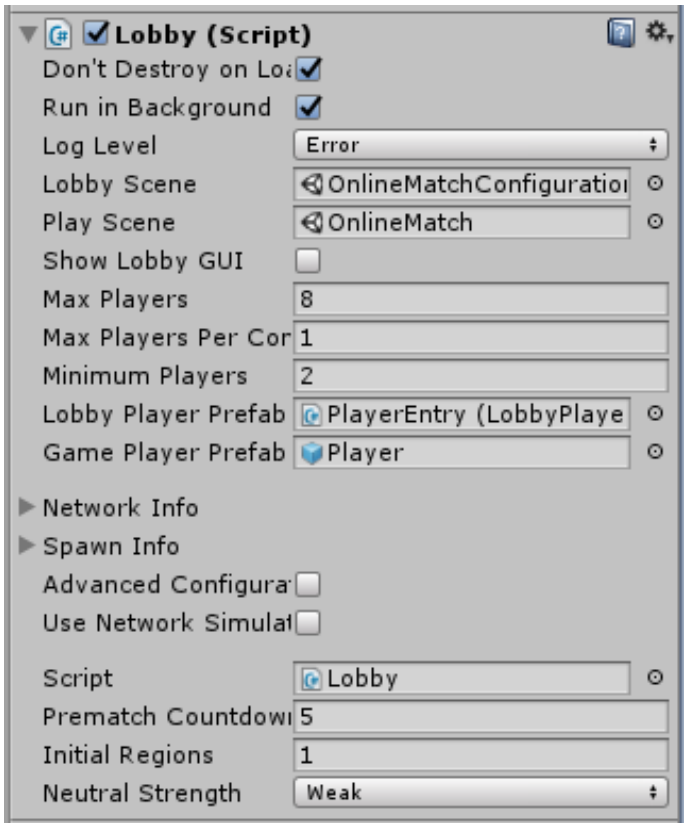

Figura 15. Lobby.cs visto en el editor de Unity3D

Como puede verse en la imagen superior hay dos tipos de **Prefabs** relacionados con el jugador: **PlayerEntry** y **Player**.

- **PlayerEntry:** Es el objeto de cada jugador dentro del Lobby. Se crea cuando un jugador se conecta y se mantiene hasta que se entra en la partida o se desconecta.
- **Player:** Es el objeto que se crea para cada jugador cuando la escena del **Lobby** cambia a la escena del juego y comienza la partida, tiene asociado el script **OnlinePlayer.cs**, que explicaremos más adelante.

4.1.3 Atributos de Lobby.cs

Entre los atributos de **Lobby.cs** cabe destacar los siguientes:

```
LobbyPanelCreate PanelCreate;
LobbyServerList ServerList;
LobbyPlayerList PlayerList;
```

Cada uno de estos atributos contiene los elementos necesarios para sus respectivos paneles, vistos en el apartado anterior (**Crear**, **Buscar** y **Lobby**), más un script asociado:

- **LobbyPanelCreate.cs**: Contiene los métodos necesarios para asignar el nombre de la partida, poder elegir la dificultad de las unidades neutrales y el número de regiones iniciales. Una vez pulsado el botón "crear" se invoca al método **lobby.cs** para cambiar del panel **Crear** al panel **Lobby**.

- **LobbyServerList.cs:** Contiene una lista de **Prefabs** de nombre **ServerEntry** en el que están almacenados los **Servidores** a los que el jugador puede unirse. En el caso de que no haya ninguno disponible se muestra el texto: "Ningún Servidor Encontrado". Cada uno de estos **Prefabs** contiene:

- Dos elementos “Text” con el nombre de la partida y el hueco disponible.
- El botón “Unirse” que, como en el caso anterior, llama al método para cambiar del panel **Buscar** al panel **Lobby**.

- **LobbyPlayerList.cs:** Es similar al anterior, tiene una cabecera con el nombre de la partida y una lista para almacenar los jugadores que se unen al **Lobby** mediante **Prefabs** de nombre **PlayerEntry** (figura 15). También contiene los métodos necesarios para añadir o eliminar jugadores. Este **Prefab** tiene asociado el script **LobbyPlayer.cs.**

- **LobbyPlayer.cs:** Este script lo primero que hace cuando se añade el **Prefab** al **Lobby** es comprobar si eres el jugador local (el dueño de ese **Prefab**). En este caso habilita los campos para cambiar el nombre, el color y poder pulsar el botón que indica si estás preparado. En el caso de ser el **Prefab** de otro jugador los deshabilita para que no puedas manipular sus datos.

  En relación con esto cabe destacar con más detalle algunas de sus acciones:

  Cada vez que el jugador cambia su nombre o su color llama a los métodos que se citan a continuación:

```
public void OnNameChanged(string str)
{
	CmdNameChanged(str);
}

public void OnColorClicked()
{
	CmdColorChange();
}
```

Nótese que llamamos al método con el comando “cmd” ya que estos datos tenemos que cambiarlos en el server y no localmente.

Estos métodos cambian los siguientes atributos:

```
[SyncVar(hook = "OnMyName")]
public string playerName = "";
[SyncVar(hook = "OnMyColor")]
public Color playerColor = Color.white;
```

Usando **SyncVar** para que estén sincronizados, las funciones **OnMyName** y **OnMyColor** son invocadas en cada cliente cada vez que su valor cambia.

El jugador también tiene los métodos necesarios para usar el chat:

```
[Command]
void CmdSend(string message)
{
	RpcRecieve(message);
}

[ClientRpc]
void RpcRecieve(string message)
{
	TxtChat.text += playerName + ">>" + message + "\n";
}
```

El jugador manda el mensaje al servidor con **cmdEnviar** para posteriormente enviárselo a todos los clientes con **RpcRecieve**.

Por último, tenemos los métodos relacionados con el botón que indica cuando los jugadores están listos. Cuando un jugador pulsa el botón “Preparado” llama a la función **SendReadyToBeginMessage** que viene ya definida en el **NetworkLobbyManager**. Con esto se invoca otra función denominada **OnClientReady** la cual hemos hecho “override” para controlar el paso del texto de “preparado” a “listo” en el caso del jugador local y de “...” a “listo” en el caso de los jugadores no locales.

4.1.4 Métodos destacados de Lobby.cs

Volviendo a la clase inicial **Lobby.cs** se pueden destacar los siguientes métodos:

- `public void ChangeToTab(Tab tab)`

Este método es el que se usa para cambiar de un panel a otro de entre los 3 disponibles (**Crear**, **Buscar** o **Lobby**). Juega con las distintas opciones según cuál sea el panel actual y a qué panel se va a cambiar:

- Cuando se entra en el panel **Buscar** se llama al método **StartMatchMaker**. Si el panel anterior era el **Lobby**, en primer lugar desconecta al jugador usando la función **StopHost** si este era el creador o **StopClient** si era un cliente.
- Al volver al panel de **Crear** se llama al método **StopMatchMaker** si se estaba buscando partida, o bien desconecta al jugador si estaba en el **Lobby** de la misma forma que en el caso anterior.
- Puedes llegar al panel **Lobby** o bien creando una partida o buscando y uniéndote a una ya creada. En el primer caso al entrar al **Lobby** se llama al método **CreateMatch** y en el segundo a **JoinMatch**.

- `public override void OnLobbyServerPlayersReady()`

Este método es invocado cuando todos los jugadores están listos. Cuando esto pasa se ejecuta una corrutina con un contador hacia atrás. Cuando llega a 0 se llama al método **ServerChangeScene** y carga la escena de la partida online.

- `public override bool OnLobbyServerSceneLoadedForPlayer(GameObject lobbyPlayer, GameObject gamePlayer)`

Este método es invocado cuando un jugador ha terminado de cambiar de la escena de **Lobby** a la escena de juego. En este método copiamos los

datos del jugador de **LobbyPlayer.cs** a **OnlinePlayer.cs,** que es el script que contiene el **Prefab** de nombre **Player** explicado anteriormente (figura 15) que se usará durante la partida.

4.2 El juego Online

Para realizar el juego se ha creado una nueva escena llamada **OnlineMatch** que contiene prácticamente los mismos elementos que la escena del juego en local. La diferencia está en que los scripts tienen que heredar de la clase **Networkbehaviour** para poder utilizar las funciones del **Network** de **Unity3D**.

También hay algunos elementos a los que hay que añadirle el componente **NetworkIdentity**, que se emplea para que pueda ser usado por la funciones de **Networking**. Un ejemplo de esto es el **prefab** que utiliza para generar las banderas del mapa, llamado **NetworkFlag**. Cada vez que deseemos crear una nueva bandera usamos el método **NetworkServer.Spawn()**, pasándole como parámetro el objeto que queremos instanciar, y este aparecerá para todos los clientes que participan en el juego.

Sin embargo, para elementos como las unidades del ejército, que son "solo números" sobre el mapa, aunque en realidad contienen bastante información asociada en su clase **Unit.cs** (ID, posición, estadísticas, vida, etc.) se necesita una forma de transmitir la información entre el servidor y el resto de clientes.

4.2.1 Paso de mensajes (problema y solución)

El principal problema que nos encontramos en el paso de mensajes de las funciones del **Network** de **Unity3D** es que no permiten el envío de datos complejos. Sólo nos permite tipos de datos primitivos como int, string, long, etc.

Para solucionarlo se han diseñado distintos **Structs** de datos primitivos, con los elementos indispensables necesarios para transmitir entre **Cliente** y **Servidor**, codificando los que sean necesarios usando **long** o **string.**

Son cuatro los **Structs** que hemos creado**:**

- **StructUnit:**

```
public struct StructUnit
{
      public long ID;            //ID de la unidad
      public string Owner;       //Nombre del dueño de la unidad
      public int Position;       //ID de la región donde está situada
      public int Destination;    //ID de la región a la que se dirige
      public int TypeID;         //ID del tipo de unidad
      public bool Attacking;     //Si está atacando
      public int Wounds;         //Heridas de la unidad
}
```

Este **Struct** se usa para almacenar la información de las unidades.

- **StructArmy:**

```
public struct StructArmy
{
      public long ID;            //ID del ejército
      public string Owner;       //Nombre del dueño del ejército
      public int Position;       //ID de la región donde está situado
      public int Destination;    //ID de la región a la que se dirige
}
```

Este **Struct** se usa para almacenar la información de los ejércitos.

- **StructBattleLog:**

```
public struct StructBattleLog
{
      public long ID;              //ID del BattleLog
      public string Attacker;      //Nombre del atacante
      public string Defender;      //Nombre del defensor
      public string CombatLogs;    //IDs de los CombatLogs
      public int From;             //ID de región de origen
      public int To;               //ID de la región destino
      public string AttackerUnits; //Número de Unidades atacantes
      public string DefenderUnits; //Número de Unidades defensoras
 }
```

Este **Struct** se usa para almacenar la información del resultado de las batallas que se muestran al final del turno. Para las variables de las unidades atacantes y defensoras se ha codificado el **string** de la siguiente manera: "XX,XX,XX,XX,XX"

Hay cinco grupos separados por comas equivalentes a los cinco tipos de unidad que hay en nuestro juego; en cada grupo, el primer número indica el total de unidades de ese tipo que participaron en la batalla y el segundo el número de unidades que quedaron vivas al final de esta.

- **StructCombatLog:**

```
public struct StructCombatLog
{
    public long ID;             //ID del CombatLog
    public string Log;          //string con los ataques realizados
    public StructUnit Attacker; //Unidad atacando
    public StructUnit Defender; //Unidad defendiendo
 }
```

Este **Struct** se usa para almacenar los resultados de cada combate individual entre unidades que tiene lugar en una batalla. La variable **Log** contiene el daño realizado en cada ataque en un **string** separado por comas.

4.2.2 Iniciando la partida

Al iniciarse el juego se invocan las funciones **OnStartClient** y **OnStartServer** según set trate del **Cliente** o del **Servidor**. A estos métodos les hemos hecho "override" para incluir la inicialización de los elementos de nuestro juego. Los clientes simplemente inicializan y actualizan las regiones del mapa, incluido el cliente que hace de **Host** (**Cliente** y **Servidor**).

El **Servidor** recorre cada región y le asigna los recursos de nuestro juego de forma aleatoria, también genera unidades neutrales aleatorias según la fuerza que hayamos configurado en el **Lobby**.

Una vez hecho esto el **Servidor** recorre cada uno de los objetos de jugador **OnlinePlayer** que hay en la escena (se trata de los **OnlinePlayer** que generamos al final del apartado anterior y que persisten al pasar a la escena de juego), los añade a la lista de jugadores del servidor y les asigna aleatoriamente sus regiones iniciales (cuya cantidad está también configurada en el **Lobby**). Una vez terminado este proceso genera el objeto **NetworkFlag** para colocar las banderas en cada región ocupada por un jugador y les asigna el color de dicho jugador.

Después de este proceso y antes de empezar el juego, cada objeto **OnlinePlayer** ejecuta su función inicial, donde inicializan datos básicos y llaman a esta función:

```
[Command]
void CmdConnectToServer()
{
  OnlineMatch.Instance.ServerInitializePlayer(connectionToClient, gameObject);
}
```

Este método, al estar situado bajo el atributo [Command], hace que se ejecute en el servidor desde el cliente. Le pasa como parámetros su **connectionToClient**, variable asociada a su **NetworkConnection,** para poder identificarlo posteriormente.

En el método **ServerInitializePlayer**, el servidor ejecuta tres acciones:

- Recorre todas las regiones iniciales de los jugadores y le devuelve la información al jugador usando el método **TargetChangeRegionOwner** para que actualice los dueños de esas regiones.

- Recorre también todas las regiones del mapa para enviarle al jugador los recursos (oro, hierro y maná) que hay en esa región mediante el método **TargetChangeRegionResources**.

- Recorre todas las unidades neutrales que ha generado en el mapa, las convierte en tipo **StructUnit** y se las envía al jugador para que actualice sus posiciones mediante el método **TargetSendUnit**.

  Nota: Los tres métodos indicados en los tres puntos anteriores son del tipo [TargetRpc] que se usa para que sean invocados en el cliente indicado en el **connectionToClient** que se le pasa por parámetro.

Terminadas todas las inicializaciones, el juego comienza normalmente y cada jugador puede hacer determinadas acciones (comprar unidades, colocarlas en el mapa, mover unidades o atacar regiones). Todas estas acciones ocurren en local y no serán sincronizadas hasta que el jugador le haya dado al botón para finalizar el turno.

4.2.3 Sincronización de fin de turno

Una vez un jugador ha terminado de hacer sus posibles acciones y le da al botón para finalizar turno, este (como **Cliente**) llama a los dos siguientes métodos:

```
[Command]
void CmdNextTurn()
{
  OnlineMatch.Instance.NextTurn(this);
}
```

Este primer método llama al método **NextTurn** que pertenece al servidor (recordemos que el atributo [Command] de un método nos permite ejecutar los métodos del servidor desde el cliente). **NextTurn** recibe como parámetro el objeto del jugador que ha finalizado el turno, a continuación, comprueba si dicho jugador figura en el listado de jugadores listos para el siguiente turno y si no aparece, lo añade.

El segundo método es llamado por el jugador para cada unidad que sea de su propiedad:

```
[Command]
void CmdSendUnit(Struct.StructUnit unit)
{
    OnlineMatch.Instance.SyncUnit(unit);
}
```

Este método, similar al anterior, llama al método del servidor **SyncUnit** que es el encargado de sincronizar las unidades. En primer lugar, busca si ya existe esa unidad en su lista de unidades (la del **Servidor**) mediante su ID; si ya existe, actualiza la información de la acción que está haciendo dicha unidad (moviéndose, atacando…). En caso de no existir esa unidad, la añade a la lista.

Además de estos dos métodos el jugador también invoca el método que sirve para mostrar en pantalla el panel de espera hasta que todos los jugadores hayan pasado su turno. El servidor enviará la orden de ocultarlo cuando esto ocurra.

Una vez el servidor comprueba que todos los jugadores han pasado su turno, es decir, cuando la lista de jugadores listos para el siguiente turno coincide con el número de jugadores de la partida, el servidor realiza las siguientes acciones:

- Aumenta el contador del turno y vacía la lista de jugadores listos para el siguiente turno.

- Crea los ejércitos de cada jugador agrupando sus unidades para posteriormente efectuar la fase de combate, guardando en las clases **BattleLog.cs** y **CombatLog.cs** toda la información relacionada (Enfrentamientos, Vencedores, Supervivientes, etc.).

- Actualiza el mapa atendiendo a lo que haya ocurrido. Primero elimina las unidades muertas y luego comprueba si las que han sobrevivido estaban atacando. En este caso actualiza la posición de las unidades a la región atacada y cambia el dueño de esa región. Una vez hecho esto llama al método **RpcChangeRegionOwner** para que cada cliente actualice el dueño de esa región. Recordemos que al tener el atributo [ClientRpc] se indica que el método es invocado en cada uno de los clientes desde el servidor.

- Avisa a los clientes que ya todos han finalizado su turno mediante el método **RpcNextTurn**, otro método con el atributo [ClientRpc]. Con este método el cliente oculta el panel de espera de fin de turno y luego comprueba si alguno de sus ejércitos ha participado en alguna batalla. En caso afirmativo, el cliente solicita al servidor el respectivo **BattleLog** para poder mostrar la información de la batalla. Para ello el jugador llama a un método con el atributo [Command] para que sea ejecutado en el servidor. Le pasa como argumentos los datos de su ejército y su variable **connectionToClient**, que es la variable asociada a su **NetworkConnection** que nos permitirá saber a qué cliente enviar la información. El servidor busca el **BattleLog** que coincida con los datos de la solicitud para posteriormente, convertir toda la información en **StructBattleLog** y sus respectivos **StructCombatLog**. Una vez hecho esto se los enviará al jugador mediante los métodos **TargetSendBattleLog** y **TargetSendCombatLog**. Recordemos que estos métodos tienen el atributo [TargetRpc] para que sean invocados en un único cliente determinado por la variable **connectionToClient** mencionada previamente.

  Una vez terminado todo este proceso y el jugador tenga ya la información necesaria, el cliente abre el panel que muestra los resultados del turno en el que se encuentra con la información recién adquirida.

- Comprueba si alguien ha ganado la partida. En caso afirmativo, en el método con el atributo [Client] llamado **OnNewTurn** (que se ejecuta únicamente en los clientes una vez empiezan un nuevo turno) saldrá el mensaje en la pantalla indicando al ganador.

Una vez en la pantalla de los resultados del turno, los jugadores podrán ver el resultado de sus batallas de la misma forma que en el juego offline. Una vez que el Cliente ha visto la información y cierra el panel hace una última petición al **Servidor** con el método **CmdFetchUnits** para sincronizar las unidades del tablero, pasándole de nuevo su variable **connectionToClient**. El servidor recorre todas las unidades y las va convirtiendo en **StrucUnit** para enviárselas de vuelta al cliente mediante el método **TargetSendUnit**. El cliente, una vez recibidas las unidades, las añade al mapa si son nuevas o actualiza su estado si ya existían.

Figura 16. Tres jugadores en plena partida

Capítulo 5
Conclusiones y líneas futuras

5.1 Conclusiones

Durante las prácticas de empresa trabajé en la versión para un jugador de un videojuego de estrategia llamado *Dominance*, inspirado en el juego de mesa *Risk*[10]. El tema de los videojuegos siempre me ha interesado y aquel primer trabajo sentó las bases para preparar el proyecto que aquí presento, la creación de una versión multijugador.

Mi Trabajo de Fin de Grado ha consistido en desarrollar el modo multijugador para que varios jugadores pudieran conectarse entre sí y competir partida tras partida. El objetivo ha sido llevado a cabo con éxito. Tras los ajustes necesarios, que se explican con detalle a lo largo de este trabajo, en este momento los jugadores pueden buscar partidas a las que unirse, crear las suyas propias, empezar a jugar y llevar a cabo una partida bien sincronizada hasta que alguno de ellos resulte vencedor.

La tarea no ha sido fácil, pues dedicarse al mundo de los videojuegos implica dedicar buena parte de nuestro esfuerzo al estudio de las distintas técnicas sobre cómo desarrollar los juegos multijugador. Puedo decir que el esfuerzo ha valido la pena y actualmente tengo una mejor idea de cómo está estructurado y cómo funciona este interesantísimo mundo del desarrollo de videojuegos multijugador online. He aprendido y puesto en práctica a lo largo

de este trabajo tareas como iniciar un servidor y que los clientes se conecten a él o la forma de pasar mensajes entre cliente y servidor para mantener el juego bien sincronizado, entre otras. También he podido fortalecer mis conocimientos sobre la plataforma Unity3D[8], una de las plataformas más populares que existen para crear videojuegos.

Solo me queda añadir que ha sido una experiencia enriquecedora, sobretodo porque es un tema que me interesa mucho, pues abre una línea de trabajo en la que me gustaría apostar mi futuro profesional.

5.2 Líneas futuras

Aunque en estos momentos, tras el proyecto que aquí presento, el juego está bastante completo, al haber podido añadirle la posibilidad multijugador online, considero que el juego está aún en fase Beta. El estudio profundo que tuve que hacer, tanto de la versión de un jugador como durante el proceso que ha culminado con la versión multijugador, ha sacado a la luz múltiples detalles que requieren una revisión que vaya perfeccionando el videojuego. Expongo a continuación algunos de estos puntos con las sugerencias que propongo y que me gustaría desarrollar en un futuro próximo para conseguir un juego más dinámico y divertido. Los posibles cambios y mejoras atañen tanto a la parte del diseño del juego como a su mecánica:

- Añadir alguna condición de victoria u objetivos adicionales. Actualmente un jugador gana cuando conquista todas las regiones de los demás jugadores. Esto, si los jugadores juegan bien, puede llevar a partidas eternas donde los implicados acabarían aburriéndose. Algunos objetivos adicionales podrían ser, por ejemplo, conquistar sectores completos del mapa o que los jugadores recibieran recompensas en forma de recursos o más unidades.

- Revisar las estadísticas de las unidades para que sean proporcionales a los recursos gastados en comprarlas. Ahora mismo la unidad de tipo "gigante" no es muy difícil de adquirir y es considerablemente más fuerte que las demás. Cuando los jugadores van aprendiendo a jugar se van dando cuenta de que solo vale la pena comprar "gigantes", por lo que el resto de unidades dejaría de usarse.

A pesar de lo dicho, el juego está en un proceso bastante avanzado, por lo que pienso que pronto podremos encontrarlo funcionando y disponible para su descarga. Sin duda, es un juego que entretiene bastante y que gustará a los jugadores amantes de los juegos de estrategia[9].

Capítulo 6

Summary and Conclusions

6.1 Summary and conclusions

During my internship I worked on a one-player version of a strategy videogame called *Dominance*, inspired in the table game *Risk*. The topic of videogames has always interested me and that first work inspired the project I'm presenting now, if only broadening the former one-player version to get into a multiplayer version.

The task I underwent was to develop multiplayer mode so several players could be connected among them and compete one match after the other. The goal has been fulfilled successfully. After some necessary adjustments, explained profusely throughout the project, players can search matches to join or create their own; and they can start playing and develop a well synchronized match until one of them wins.

The task was not easy, because getting involved in the world of videogames implies careful study of the different techniques about developing multiplayer games. I dare say that it has been productive and right now I know better about internal structures and functions of this most interesting discipline. Throughout this project I have learned and put into practice several tasks such as how to initiate a server that clients can be connected to, or how to open a message pathway between client and server to keep the game well synchronized, among other issues. I have also been able to

strengthen my knowledge about platform *Unity3D,* which stands as one of the most popular current platforms to create videogames.

I need only add that it has been a most enriching experience, above all, because it is a field that I am most involved in, and rather because it opens up a line of work I would choose in my professional path.

Capítulo 7
Bibliografía

1. Historia de los videojuegos [Internet], Wikipedia. 2018. Recuperado a partir de: https://es.wikipedia.org/wiki/Historia_de_los_videojuegos

2. Videojuegos multijugador [Internet]. 2018. Wikipedia. Recuperado a partir de: https://es.wikipedia.org/wiki/Videojuego_multijugador

3. List of cooperative video games [Internet]. Wikipedia. 2018. Recuperado a partir de: https://en.wikipedia.org/wiki/List_of_cooperative_video_games

4. List of grand strategy video games [Internet], Wikipedia. 2018. Recuperado a partir de:
https://en.wikipedia.org/wiki/List_of_grand_strategy_video_games

5. Sánchez i Peris, Francesc J. (Coord.) Videojuegos: una herramienta educativa del "homo digitalis". *Revista Electrónica Teoría de la Educación: Educación y Cultura en la Sociedad de la Información.* Vol. 9, nº 3. Universidad de Salamanca.
http://www.usal.es/~teoriaeducacion/rev_numero_09_03/n9_03_gonzalez_blanco.pdf

6. Matias, Como aprender inglés a través de videojuegos [Internet]. Kaplan Blog. 2015. Recuperado a partir de: https://www.kaplaninternational.com/latam/blog/como-aprender-ingles-a-traves-de-video-juegos

7. Deportes electrónicos [Internet]. Wikipedia. 2018. Recuperado a partir de: https://es.wikipedia.org/wiki/Deportes_electr%C3%B3nicos

8. Unity3D. 2018. Recuperado a partir de: https://unity3d.com/es

9. Strategies video game [Internet], Wikipedia. 2018. Recuperado a partir de: https://en.wikipedia.org/wiki/Strategy_video_game

10. Risk [Internet]. Wikipedia.2018. Recuperado a partir de: https://es.wikipedia.org/wiki/Risk

11. Attia Peter, La muy, muy larga pero imprescindible historia de los juegos de mesa [Internet]. Magnet. 2016, Recuperado a partir de: https://magnet.xataka.com/en-diez-minutos/la-larga-historia-de-los-juegos-de-mesa

12. Visual Studio. Microsoft. 2018. Recuperado a partir de: https://visualstudio.microsoft.com/

13. Unity | Creating Gameplay. Recuperado a partir de: https://docs.unity3d.com/Manual/CreatingGameplay.html

14. Unity | The Multiplayer High Level API. Recuperado a partir de: https://docs.unity3d.com/Manual/UNetUsingHLAPI.html

15. Unity | NetworkManager. Recuperado a partir de:
https://docs.unity3d.com/ScriptReference/Networking.NetworkManager.html

16. Unity Multiplayer services. Recuperado a partir de:
https://unity3d.com/es/unity/features/multiplayer

17. Unity | Documentation. NetworkMatch Recuperado a partir de:
https://docs.unity3d.com/ScriptReference/Networking.Match.NetworkMatch.html

18. Unity | Documentation. Networking Attributes. Recuperado a partir de:
https://docs.unity3d.com/ScriptReference/Networking.ClientAttribute.html

19. Unity | Documentation. NetworkLobbyManager. Recuperado a partir de:
https://docs.unity3d.com/Manual/class-NetworkLobbyManager.html