

Universidad  
Rey Juan Carlos

Escuela Técnica Superior de Ingeniería Informática

Grado en Diseño y Desarrollo de Videojuegos

Curso 2023 2024

Trabajo Fin de Grado

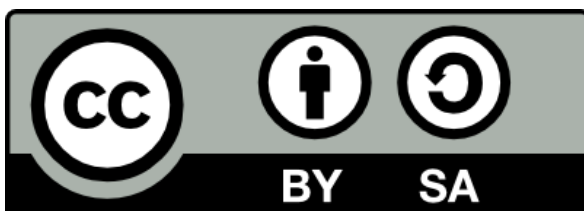
**DESARROLLO DE UN VIDEOJUEGO MULTIJUGADOR**

Autor: Álvaro García Sierra

Tutor: David María Arribas

---

Esta publicación tiene licencia Atribución-CompartirIgual 4.0 Internacional (CC BY-SA 4.0) <https://creativecommons.org/licenses/by-sa/4.0/deed.es>



---

## Agradecimientos

*A mis padres, por apoyarme en todo momento y creer siempre en mí, incluso cuando yo no lo hacía.*

*A mis amigos de Toledo, por poder contar siempre con ellos.*

*A mis amigos de la Universidad, sin ellos esta etapa habría sido mucho más difícil.*

*A todos vosotros, os quiero, gracias.*

---

## Resumen

El desarrollo de este Trabajo de Fin de Grado (TFG) se centra en la creación de un prototipo de videojuego del género *Battle Royale*, estilo Brawl Stars, diseñado tanto para dispositivos móviles como para PC. Para su creación se ha usado el motor Unity junto a la librería de Netcode for Gameobjects.

El proyecto ha abarcado la implementación básica de un videojuego, incluyendo todas las funcionalidades principales.

Este documento está compuesto por cinco capítulos. En el capítulo uno, llamado “Introducción”, se explicará a modo introductorio el origen de las motivaciones que impulsaron la realización de este proyecto, así como los desafíos que conlleva el desarrollo de los juegos multijugador. En este capítulo, también se detallarán los objetivos a cumplir para el proyecto.

En el capítulo dos, “Marco Teórico”, se introduce una investigación previa para el posterior desarrollo del proyecto, que abarca desde los inicios de los videojuegos multijugador hasta el estado del arte más actual. Esta investigación también abarca diferentes casos de estudio relevantes y las principales técnicas utilizadas para resolver los problemas de los juegos multijugador.

En el capítulo tres, se detalla el diseño del juego, sus controles, mecánicas y arte, para, posteriormente en el capítulo cuarto explicar la metodología utilizada, así como los requisitos funcionales y no funcionales. También se mostrarán los diferentes casos de uso y se nombrarán las tecnologías y herramientas utilizadas. Además, se entrará en detalle sobre la arquitectura del sistema y su implementación. A continuación, se realiza una discusión sobre los retos técnicos en la implementación del videojuego y sobre la validación del juego, realizada mediante diferentes *tests* con usuarios reales.

Por último, el capítulo quinto, “Conclusiones y trabajo futuro”, servirá para ofrecer los resultados obtenidos, conclusiones finales y se introducirán posibles mejoras para su futuro.

### **Palabras clave:**

Unity

Videojuego

Multijugador en línea

Netcode for Gameobjects

Cliente-servidor

---

## Abstract

The development of this final project focuses on creating a prototype of a *Battle Royale* video game, in the style of Brawl Stars, designed for both mobile devices and PC. The Unity engine was used for its creation along with the Netcode for Gameobjects library.

The project has covered the basic implementation of a video game, including all the main functionalities.

This document is composed of five chapters. In chapter one, "Introduction," the origin of the motivations that led to the realization of this project will be explained as an introduction, as well as the challenges involved in developing multiplayer games. This chapter will also detail the objectives to be achieved for the project.

In chapter two, a previous research for the subsequent development of the project is introduced, covering from the beginnings of multiplayer video games to the most current state of the art. This research also includes different relevant case studies and the main techniques used to solve the problems of multiplayer games.

In chapter three, the game's design, controls, mechanics, and art are detailed. Subsequently, in chapter four, the methodology used is explained, as well as the functional and non-functional requirements. The different use cases will also be shown, and the technologies and tools used will be named. Additionally, the system architecture and its implementation will be detailed. Following this, a discussion on the technical challenges in the implementation and on the validation of the game, carried out through various tests with real users, will take place.

Finally, chapter five, "Conclusions and Future Work", will serve to provide the obtained results, final conclusions, and will introduce possible improvements for the future.

### **Keywords:**

Unity  
Video game  
Online multiplayer  
Netcode for Gameobjects  
Client-server

# Índice

<b>1. Introducción</b>	<b>1</b>
1.1. Motivación . . . . .	1
1.2. Objetivos . . . . .	2
<b>2. Marco teórico</b>	<b>3</b>
2.1. Orígenes de los videojuegos multijugador . . . . .	3
2.2. Desafíos de los videojuegos multijugador . . . . .	6
2.3. Técnicas usadas en los videojuegos multijugador . . . . .	7
2.3.1. Enfoques utilizados para la sincronización de estado . . . . .	7
2.3.2. Técnicas de compensación de latencia . . . . .	10
2.3.3. Delay Netcode y Rollback Netcode . . . . .	12
2.4. Arquitecturas . . . . .	14
2.4.1. Cliente-servidor . . . . .	14
2.4.2. Peer-to-Peer (P2P) . . . . .	15
2.4.3. Arquitectura híbrida . . . . .	15
2.4.4. Modelo de cliente host . . . . .	15
2.5. Protocolos de transporte . . . . .	16
2.6. Estado del arte . . . . .	17
<b>3. Diseño del juego</b>	<b>21</b>
3.1. Sinopsis . . . . .	21
3.2. Controles . . . . .	21
3.3. Mecánicas . . . . .	21
3.4. Arte . . . . .	22
3.4.1. Interfaces de Usuario . . . . .	22
<b>4. Análisis y diseño</b>	<b>25</b>
4.1. Metodología empleada . . . . .	25
4.2. Requisitos . . . . .	25
4.2.1. Casos de uso . . . . .	27
4.3. Ejemplos de uso . . . . .	31
4.4. Proceso de desarrollo . . . . .	34
4.5. Diagrama de clases . . . . .	36
<b>5. Descripción informática</b>	<b>38</b>
5.1. Tecnologías y herramientas utilizadas . . . . .	38
5.1.1. Motor de videojuegos . . . . .	38
5.1.2. Elección de la librería multijugador . . . . .	39
5.1.3. Elección del proveedor de servicios . . . . .	41
5.1.4. Otras herramientas . . . . .	44
5.2. Arquitectura de red . . . . .	44
5.3. Manejo y propagación de errores . . . . .	49
5.4. Integración Unity Services . . . . .	50
5.5. Adapatación multiplataforma . . . . .	51
5.6. Retos técnicos . . . . .	52

---

<b>6. Validación</b>	<b>54</b>
6.1. Experimentos y pruebas . . . . .	54
<b>7. Conclusiones y trabajo futuro</b>	<b>58</b>
7.1. Conclusiones . . . . .	58
7.1.1. Logros alcanzados . . . . .	58
7.2. Trabajos futuros . . . . .	59
<b>Bibliografía</b>	<b>63</b>
<b>Anexos</b>	<b>64</b>
Anexo 1: Ejemplos de código . . . . .	64

## Índice de figuras

1.	<i>Tennis For Two</i> .Imagen obtenida de wikipedia.com . . . . .	3
2.	Diagrama funcionamiento <i>PLATO</i> www.tandfonline.com . . . . .	4
3.	Videojuego <i>MUD</i> . Imagen obtenida de as.com/meristation . . . . .	5
4.	Videojuego <i>Doom</i> . Imagen obtenida de wikipedia.com . . . . .	6
5.	Videojuego <i>Quake</i> . Imagen obtenida de engadget.com . . . . .	6
6.	Servidores dedicados de Valorant. Imagen obtenida de netduma.com . . . . .	20
7.	Estética del videojuego . . . . .	23
8.	Diagrama de navegación - Máquina de estados . . . . .	24
9.	Autenticación del usuario . . . . .	31
10.	Lista de salas . . . . .	31
11.	Creación de Lobby . . . . .	32
12.	Sala de la partida . . . . .	32
13.	Captura de la partida . . . . .	33
14.	Menú de estadísticas de la partida . . . . .	33
15.	Comienzo Sprint 1 . . . . .	34
16.	Comienzo Sprint 2 . . . . .	34
17.	Comienzo Sprint 3 . . . . .	35
18.	Comienzo Sprint 4 . . . . .	35
19.	Diagrama de clases reducido . . . . .	37
20.	Desfase de posición . . . . .	47
21.	Buffer de interpolación. Imagen obtenida de unity3d.com . . . . .	48
22.	Host Disconnected, mensaje de error . . . . .	49
23.	Error de conexión, mensaje de error . . . . .	50
24.	Unity Dashboard - Lobby . . . . .	51
25.	Unity Transform . . . . .	54
26.	Rendimiento en versión temprana del desarrollo . . . . .	55
27.	Rendimiento en última versión . . . . .	56
28.	Ancho de banda en versión temprana del desarrollo . . . . .	56
29.	Ancho de banda en última versión . . . . .	57

---

# 1. Introducción

## 1.1. Motivación

El impulso que ha motivado el desarrollo de este prototipo de videojuego multijugador ha surgido de la fascinación y el entusiasmo por la dinámica cambiante del mundo de los videojuegos. La creciente interconexión de las personas en la sociedad actual ha llevado a una demanda cada vez mayor de experiencias de juego compartidas y envolventes. En este aspecto, los videojuegos multijugador han tenido un papel importante, ofreciendo a los jugadores la oportunidad de sumergirse en mundos virtuales interactivos y colaborativos.

La creación de un videojuego multijugador representa tanto un reto técnico como una oportunidad para explorar la unión entre la creatividad y la tecnología. La diversidad en el diseño de videojuegos multijugador ofrece una amplia gama de posibilidades, desde la definición de mecánicas de juego únicas hasta la implementación de sistemas de red eficientes. Este proyecto representa un desafío en el aprendizaje de la maestría de las complejidades involucradas en la creación de experiencias multijugador.

La motivación detrás de este proyecto se fortalece aún más al considerar la evolución constante de la industria de los videojuegos multijugador. La posibilidad de contribuir a este panorama en constante cambio, explorando soluciones innovadoras y desafiando los límites técnicos, es un estímulo adicional. La oportunidad de aprender, adaptarse y aplicar conocimientos en un entorno en constante transformación brinda una experiencia única y enriquecedora.

## 1.2. Objetivos

Desarrollar un videojuego multijugador y multiplataforma eficiente es un desafío que suele requerir el esfuerzo de un equipo de desarrollo. Por tanto, el objetivo principal de este proyecto es establecer una base sólida mediante la creación de un prototipo funcional. Este prototipo servirá como punto de partida para iterar, probar y perfeccionar diferentes aspectos del juego, como la jugabilidad, la estabilidad de la red y la experiencia del usuario. Al centrarnos en la creación de un prototipo sólido, podemos asegurarnos de sentar las bases necesarias para el desarrollo futuro, permitiendo una evolución gradual hacia un producto final robusto.

La realización de este trabajo de fin de grado tiene los siguientes objetivos específicos:

- Investigar la evolución de los videojuegos multijugador así como las posibilidades y dificultades que representan.
- Investigar las diferentes técnicas y procesos para hacer frente a los retos del desarrollo de los videojuegos multijugador.
- Desarrollar un prototipo de videojuego multijugador aplicando el conocimiento adquirido e implementando técnicas de compensación de latencia.
- Implementar la capacidad multiplataforma (móvil y PC) para el videojuego.
- Validar el proyecto mediante diferentes pruebas para asegurar el correcto funcionamiento del videojuego.

## 2. Marco teórico

En este apartado se estudia la evolución de los videojuegos multijugador en línea, desde sus primeros días, hasta el estado del arte actual. También se presentan los distintos casos de estudio icónicos, los cuales han dejado un impacto significativo, ayudando en la evolución de este tipo de videojuegos gracias al uso de elementos o tecnologías innovadoras. También se explicarán diversas tecnologías o técnicas de red y compensación de latencia.

### 2.1. Orígenes de los videojuegos multijugador

En el marco de los videojuegos multijugador, *Tennis For Two* (figura 1), se puede considerar tanto el segundo videojuego de la historia después de OXO [1] como el primer videojuego multijugador local, al involucrar a dos jugadores. Este videojuego fue desarrollado por William Higginbotham en 1958 usando para ello un osciloscopio conectado a una computadora analógica. Este videojuego simulaba un campo de tenis mediante una línea horizontal, la cual representaba el terreno, y otra vertical, representando la red. Los jugadores debían elegir la velocidad y ángulo de la pelota mediante botones.

Figura 1: *Tennis For Two* .Imagen obtenida de wikipedia.com



Sin embargo, no fue hasta unos años después cuando se presentó el videojuego *Spacewar!*, el cual se puede considerar el primer videojuego multijugador en línea, al incorporar partidas de red de área local (LAN) en dos pantallas simultáneamente, aunque esto fuera en una versión posterior.

*Spacewar!* es un videojuego desarrollado por Steve Russell y un grupo de estudiantes en 1962 para demostrar la capacidad del Procesador de datos programado o PDP-1. Se trata de un videojuego de combate espacial, donde cada jugador controla una nave y debe destruir al jugador contrario disparando.

En esta década de 1960 se desarrolló la plataforma *PLATO* (Programmed Logic for Automatic Teaching Operations), la cual, aunque estuvo diseñada principalmente con fines educativos, supuso un gran avance para los videojuegos multijugador. Este proyecto utilizaba una arquitectura *cliente-servidor*, permitiendo a varios jugadores jugar juntos desde distintos terminales a través de una red (figura 2). Dichos terminales no ejecutaban ninguna tarea de cálculo o gestión por sí mismos, si no era el servidor el que aportaba toda la lógica del juego y distribuía la información a los jugadores mediante un switch.

Este sistema permitió el desarrollo de múltiples simulaciones y videojuegos. Podemos destacar *Empire* en el que podían jugar hasta 30 jugadores simultáneamente, y *Spasim* en el que podían hacerlo hasta hasta 32 personas, ambos desarrollados en 1973.

Figura 2: Diagrama funcionamiento *PLATO* [www.tandfonline.com](http://www.tandfonline.com)

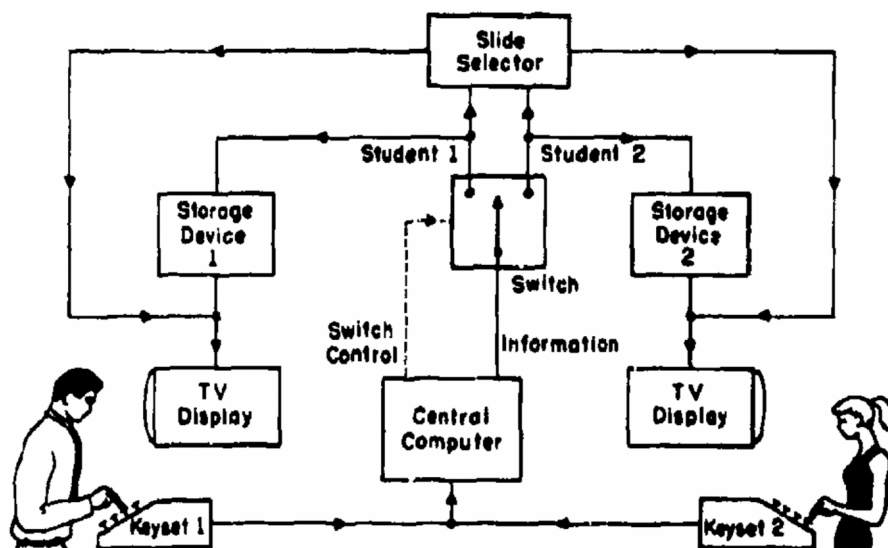


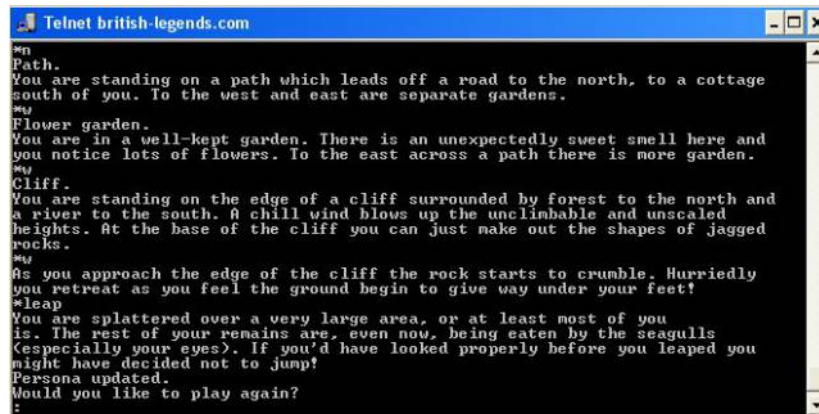
Fig. 1. General organization of PLATO II

Años más tarde, en la década de los 70 y gracias a los avances de *PLATO*, se desarrollaron los primeros **Multi-User Dungeon (MUD)** (figura 3). Este género de videojuegos ofrecía a varios jugadores interacción mediante la ejecución de comandos de texto para la toma de decisiones en un mundo virtual. Dichos juegos eran jugados a través de redes locales en sus orígenes y más tarde, a través de conexiones telefónicas. Los desarrolladores de *MUDs* fueron pioneros en la creación de mundos virtuales persistentes donde los jugadores podían explorar, interactuar y colaborar en tiempo real. Estos mundos estaban formados por descripciones en texto y permitían a los jugadores realizar una amplia variedad de acciones, desde luchar contra monstruos hasta comerciar con otros jugadores.

La estructura y la mecánica de juego utilizadas en los *MUDs* han influido en el

diseño de juegos en línea modernos. Elementos como la creación de personajes, la interacción social, la exploración de mundos virtuales y la cooperación entre jugadores son conceptos que se originaron en gran medida en los *MUDs*.

Figura 3: Videojuego *MUD*. Imagen obtenida de [as.com/meristation](http://as.com/meristation)



En 1993 fue lanzado el videojuego *Doom*, un videojuego de disparos multijugador creado por id Software (figura 4). Es ampliamente considerado como uno de los títulos más influyentes en la historia de los videojuegos y un referente en el género de los shooters en primera persona multijugador (*FPS* por sus siglas en inglés First Person Shooter).

Al contrario al modelo que se usaba en *PLATO*, la arquitectura de red en *Doom* se basaba en conexiones P2P (ver apartado 2.4.2), lo que significa que los jugadores se conectaban directamente entre sí en lugar de depender de un servidor central. *Doom* inicialmente utilizó el protocolo IPX/SPX para la conexión en red local. Más tarde, con la popularidad creciente de las conexiones a Internet, se desarrollaron y adoptaron soluciones que permitían jugar a través de TCP/IP, el protocolo de Internet estándar.

La capacidad de jugar *Doom* en modo multijugador impulsó la popularidad de las LAN parties (fiestas de red local). Los jugadores podían conectarse a través de una red local y competir entre sí en mapas personalizados y *mods* creados por la comunidad. Su éxito contribuyó a la creciente popularidad de los videojuegos y atrajo a una nueva audiencia. La cultura multijugador que rodeaba a *Doom* estableció un precedente para futuros juegos en línea, desempeñando un papel crucial en la evolución de los juegos multijugador y de los FPS.

Debido al éxito de *Doom*, id Software solo tardó 3 años en lanzar un nuevo FPS multijugador llamado *Quake* (figura 5). Este videojuego, marcó un avance significativo en los videojuegos en 3 dimensiones, siendo uno de los primeros en usar un motor gráfico completamente tridimensional. A diferencia de *Doom*, *Quake* fue desarrollado teniendo en cuenta desde el principio el multijugador. Esto coincidió con el crecimiento de la popularidad de las conexiones a Internet, lo que permitió a los jugadores de todo el mundo competir entre sí en partidas multijugador. *Quake* usaba una arquitectura cliente-servidor, usando servidores dedicados para el manejo de las partidas. Debido a las conexiones de la época, la latencia y la pérdida de paquetes influían negativamente en la jugabilidad. Por eso, en una actualización lanzaron *QuakeWorld*, donde entre otras cosas, se introdujo la predicción del cliente (ver apartado 2.3.2.1).

Figura 4: Videojuego *Doom*. Imagen obtenida de wikipedia.comFigura 5: Videojuego *Quake*. Imagen obtenida de engadget.com

## 2.2. Desafíos de los videojuegos multijugador

Debido a la complejidad adicional de gestionar la interacción en tiempo real entre múltiples jugadores, los videojuegos multijugador presentan una serie de desafíos únicos, entre los que se encuentran los siguientes:

- **Latencia y sincronización.** La latencia es el tiempo que tarda la información en viajar desde un jugador hasta el servidor, este tiempo es conocido como *ping*. Gestionar la latencia y sincronizar las acciones de los jugadores en tiempo real puede ser complicado. La predicción del cliente y otras técnicas se utilizan para hacer que la experiencia sea más fluida, pero aún así, minimizar los efectos de la latencia sigue siendo un desafío.
- **Jitter.** El jitter es la variabilidad en la latencia de la conexión, es decir, los cambios impredecibles en el tiempo de viaje de los paquetes de datos. El jitter

puede causar fluctuaciones en la suavidad de la experiencia de juego. Si la latencia varía constantemente, los jugadores pueden experimentar movimientos irregulares de otros jugadores, lo que puede dificultar la predicción de sus acciones y también afectar la jugabilidad.

- **Pérdida de paquetes.** Como su nombre indica, la pérdida de paquetes sucede cuando un paquete de datos no llega a su destino. La pérdida de paquetes también puede introducir movimientos irregulares o acciones que no son registradas correctamente.
- **Seguridad y trampas.** Proteger el juego contra trampas y *hacks* es esencial para mantener una experiencia de juego justa. Los desarrolladores deben implementar medidas de seguridad sólidas para evitar que los jugadores manipulen el juego de manera injusta, lo que puede incluir sistemas antitrampas y monitoreo constante.
- **Interoperabilidad de plataformas.** La creciente diversidad de plataformas, como PC, consolas y dispositivos móviles, plantea desafíos para lograr la interoperabilidad entre ellas. Los desarrolladores deben abordar cuestiones relacionadas con la compatibilidad cruzada y garantizar que los jugadores de diferentes plataformas puedan interactuar entre sí.
- **Diversidad de conexiones.** Ya que los jugadores pueden tener diferentes niveles de conexión a Internet y ancho de banda, los desarrolladores deben diseñar sistemas que se adapten a una variedad de conexiones para garantizar una experiencia de juego justa y accesible para todos.
- **Escalabilidad.** Garantizar que el juego funcione de manera eficiente y sin problemas a medida que aumenta el número de jugadores es un desafío importante. Los desarrolladores deben diseñar la arquitectura del servidor y de la red para ser escalable y capaz de manejar grandes cantidades de usuarios simultáneos.
- **Gestión de servidores.** Mantener los servidores en funcionamiento, administrar la carga, aplicar parches y realizar actualizaciones de manera eficiente son aspectos muy importantes en el desarrollo y mantenimiento de los juegos multijugador. La infraestructura del servidor debe ser sólida para evitar interrupciones del servicio.

### 2.3. Técnicas usadas en los videojuegos multijugador

Como ya hemos visto en el apartado anterior, en el transcurso de los años han surgido multitud de técnicas para tratar de mitigar los problemas emergentes de los juegos en red. Debido a otros factores, como las limitaciones de la red, estas técnicas pueden no representar una solución definitiva, pero suelen mejorar notablemente la experiencia de usuario.

#### 2.3.1. Enfoques utilizados para la sincronización de estado

Una correcta actualización del estado del juego en todos los clientes es imprescindible para que todos los jugadores experimenten una experiencia de juego sincronizada. Esto se consigue mediante el envío de mensajes, los cuales contienen

información relevante del estado del juego. Cada una de estas técnicas trata este paso de mensajes de forma diferente.

### 2.3.1.1 Deterministic Lockstep

Esta técnica utiliza los *inputs* de los jugadores para crear las mismas simulaciones tanto en el servidor como en todos los clientes. Haciendo posible ejecutar todas las acciones en el mismo orden y condiciones iniciales, y garantizando que todos lleguen al mismo estado del juego después de ejecutar un conjunto de acciones específico.

Debido a este funcionamiento, *Deterministic Lockstep* otorga grandes ventajas. La primera de estas es que debido a que la secuencia de acciones y eventos es reproducible, se puede reproducir el mismo estado del juego entre los clientes y el servidor, garantizando la coherencia del estado.

La ejecución determinista hace que sea más difícil para los jugadores realizar trampas, ya que la única manera de alterar el estado del juego es mediante *inputs*. Cualquier intento de alterar el estado del juego de manera no autorizada se detectará fácilmente al comparar los resultados con otros clientes.

En *Deterministic Lockstep* los paquetes enviados son muy ligeros ya que solo se envía el input de los jugadores y no es necesario enviar información detallada en cada actualización. Esto conlleva una menor carga de la red.

Pese a todo esto, *Deterministic Lockstep* tiene ciertas limitaciones. La primera de estas limitaciones es que este enfoque solo es posible si el juego es determinista, es decir el juego debe tener siempre una lógica predecible y consistente. Esta característica impide su implementación en multitud de juegos, los cuales usan físicas no deterministas. Estas físicas pueden conducir a resultados diferentes debido al time step, el cual puede variar en los diferentes jugadores.

La ejecución determinista hace compleja la introducción de contenido dinámico generado aleatoriamente.

La ejecución determinista también puede introducir latencia percibida, ya que los jugadores deben esperar a que todos los participantes terminen un frame en un paso de la simulación antes de ver el resultado.

En el caso de una pérdida de paquetes, el determinismo del juego puede verse comprometido ya que no todos los clientes ejecutarían la acción o input del paquete perdido, pudiendo llegar a tener consecuencias significativas en la sincronización del juego. Por eso es necesario el uso conjunto de otras técnicas como *reconciliación* 2.3.2.4 u otras técnicas para el manejo de pérdidas, como retransmisión de los datos o detección de pérdidas de paquetes para garantizar la integridad y sincronización del juego.

Debido a que en *Deterministic Lockstep* solo se envía el input de los jugadores, en el caso de desconexiones haría falta medidas complementarias para que el jugador desconectado sea actualizado hasta el estado actual de la partida.

### 2.3.1.2 Snapshot Interpolation

En este enfoque, el servidor envía instantáneas o *snapshots* del estado del juego de forma regular y ajustable. Cuando los clientes reciben estos *snapshots* interpolan esta información para llegar al estado actual.

La interpolación suaviza las transiciones entre actualizaciones, creando una apariencia más fluida y natural del movimiento y mitigando a su vez la percepción de la latencia. Debido a que en este enfoque los clientes conocen de forma regular el estado

del juego, aunque exista una pérdida de paquetes, los jugadores pueden interpolar hasta llegar al estado más reciente.

*Snapshot Interpolation* se puede usar en situaciones no deterministas, ya que toda la información necesaria se comunica de forma constante. La simulación de físicas tampoco tiene los problemas de *Deterministic Lockstep* dado que estas pueden ser interpoladas de manera efectiva.

Aunque la interpolación mejora la suavidad del juego y la experiencia del jugador, existen ocasiones en las que puede llevar a una representación inexacta del juego, mostrando los objetos o jugadores en lugares donde, en realidad, ya no se encuentran.

Esta técnica requiere el envío de snapshots sobre el estado del juego, por esto *Snapshot Interpolation* requiere una transmisión de datos mayor incluyendo posiciones, rotaciones, velocidades o escalados. Esto conlleva un aumento en la carga de la red. Por otra parte, el uso de movimientos no lineales, como saltos o movimientos que implican cambios abruptos en la velocidad, pueden resultar difíciles de manejar con la interpolación. Esto puede conducir a representaciones visuales inexactas.

### 2.3.1.3 State Synchronization

En este enfoque, al igual que en *Deterministic Lockstep*, la simulación del juego se ejecuta tanto en el servidor como en el cliente, enviando el *input* de los jugadores, sin embargo el estado del juego también es enviado, de forma similar a *Snapshot Interpolation*. Estas dos características propias de los enfoques mencionados anteriormente, permite ser empleada en los casos no deterministas y al ser ejecutado en ambas partes, la lógica del juego sigue siendo ejecutada entre actualizaciones.

*State Synchronization* también permite el ajuste en el intervalo de tiempo entre los envíos de los paquetes de estado, permitiendo equilibrar la precisión y el ancho de banda en función de las necesidades del juego.

En *Snapshot Interpolation*, el envío del estado del juego puede suponer un uso elevado de ancho de banda. Sin embargo, en este caso, además de enviar el estado del juego de una manera regular y ajustable, podemos controlar qué elementos estamos enviando y cuales no, reduciendo así la carga de la red.

*State Synchronization* hace que los clientes reciban actualizaciones regulares de estado y apliquen los cambios de forma local. Esto garantiza que todos los participantes tengan una visión consistente del juego.

En los casos de pérdida de paquetes, los clientes pueden seguir manteniendo un estado coherente basada en la última información recibida.

En la tabla 1 se presenta una comparativa de las diferentes técnicas de sincronización comentadas, junto con sus ventajas y desventajas:

Técnica	Descripción	Ventajas	Desventajas
DL	Deterministic Lockstep: Sincroniza el estado de todos los clientes en intervalos de tiempo fijos. Cada cliente ejecuta las mismas acciones en el mismo orden y al mismo tiempo.	<ul style="list-style-type: none"> <li>■ Consistencia del estado entre clientes.</li> <li>■ Dificulta las trampas de los jugadores.</li> <li>■ Reducción del ancho de banda.</li> </ul>	<ul style="list-style-type: none"> <li>■ Requiere sincronización precisa del reloj.</li> <li>■ Requiere que el juego sea determinista.</li> <li>■ Requiere medidas especiales para manejar desconexiones.</li> <li>■ Sensible a la latencia de red.</li> </ul>
SI	Snapshot Interpolation: Utiliza instantáneas del estado del juego y realiza interpolación entre ellas para suavizar el movimiento de los objetos.	<ul style="list-style-type: none"> <li>■ Mayor fluidez en la visualización del juego.</li> <li>■ Los jugadores pueden interpolar en caso de pérdida de paquetes.</li> <li>■ Reducción de la sensibilidad a la latencia.</li> </ul>	<ul style="list-style-type: none"> <li>■ Posible desajuste entre estado del juego y visualización.</li> <li>■ Dificultades con movimientos no lineales.</li> <li>■ Mayor consumo de ancho de banda.</li> </ul>
SS	State Synchronization: Comparte y sincroniza constantemente los estados de todos los objetos relevantes del juego entre los clientes.	<ul style="list-style-type: none"> <li>■ Flexibilidad para manejar diferentes tipos de objetos.</li> <li>■ Mayor precisión en la sincronización.</li> <li>■ Permite el control sobre el envío de datos para reducir la carga de la red.</li> <li>■ Visión consistente del juego entre todos los participantes.</li> </ul>	<ul style="list-style-type: none"> <li>■ Mayor carga en el ancho de banda.</li> <li>■ Requiere manejo de conflictos.</li> </ul>

Cuadro 1: Comparativa de técnicas de sincronización en juegos multijugador

### 2.3.2. Técnicas de compensación de latencia

En los juegos multijugador, el proceso general es el siguiente: primero, el jugador genera comandos, que se empaquetan y envían al servidor. El servidor recibe y deserializa estos comandos, calcula el estado del juego y luego empaqueta y envía este estado a los clientes. Finalmente, los clientes reciben el estado del juego y lo actualizan para mostrar los cambios a los jugadores.

En resumen, los clientes deben esperar a que el servidor calcule el estado correcto y lo envíe de vuelta para que se muestre en la pantalla, lo que introduce una latencia notable y empeora la experiencia del usuario.

### 2.3.2.1 Predicción del cliente

En *Predicción del cliente*, en lugar de esperar a recibir actualizaciones del servidor, los clientes realizan predicciones locales sobre el estado del juego en función de los *inputs* del jugador, asumiendo que dichos *inputs* sean aceptados por el servidor. Cuando se reciben actualizaciones del servidor, estas predicciones se corrigen o confirman. Esto ayuda a reducir la percepción de latencia y mejora la *responsividad* del videojuego. El uso de esta técnica puede comprometer la coherencia en el estado si la predicción local del cliente no es la misma que la simulación del servidor o si las entradas de usuario no llegan al servidor por una pérdida de paquetes. Esto se puede resolver con una técnica de *reconciliación* 2.3.2.4.

### 2.3.2.2 Interpolación del cliente

Esta técnica trata de solucionar la apariencia de saltos o movimientos bruscos que suceden al recibir actualizaciones de posición en un ratio menor a la tasa de actualización del juego. Cuando este se produce, los jugadores ven a los objetos transportándose a su nueva posición o estado instantáneamente. Si la latencia o el jitter es alto, la información también puede llegar de manera intermitente afectando de nuevo a la suavidad del juego.

*Interpolación del cliente* Se centra en la interpolación suave en el cliente de las posiciones y estados de los objetos del juego entre actualizaciones. Al recibir una actualización, en lugar de mostrarse al jugador al instante, se realiza una transición gradual, desde el estado anterior hasta el nuevo estado calculando las posiciones intermedias, creando una apariencia más suave y natural. Esto mejora la visualización del movimiento, especialmente en situaciones de latencia de red.

### 2.3.2.3 Extrapolación del cliente y Dead Reckoning

La extrapolación es una técnica que trata de prever valores futuros basados en patrones o datos anteriores. En el contexto de los videojuegos multijugador a esta técnica se le denomina *Dead Reckoning*.

En el caso de que el cliente se quede momentáneamente sin información del servidor, aparece una discrepancia entre el estado actual del cliente y el estado real del juego. Para solventar este problema y con el objetivo de predecir el siguiente estado, se asume que los comportamientos actuales, como la dirección en la que se están moviendo determinados objetos, seguirá siendo la misma.

*Dead Reckoning* es utilizada para prever la posición futura de objetos y jugadores. Esta técnica calcula una estimación de la posición futura utilizando información actual, como la posición, velocidad y aceleración asumiendo que sus comportamientos actuales seguirán invariantes. En caso en el que la predicción de la posición no sea correcta, habrá una desviación que habrá que corregir, usando por ejemplo *reconciliación* 2.3.2.4.

Dependiendo del tipo de movimiento que realice el objeto y la cantidad de información que se posea, *Dead Reckoning* utilizará un algoritmo u otro. De forma simple, se emplea la ecuación del movimiento rectilíneo uniforme, prediciendo la

siguiente posición en base a la posición y velocidad actuales y un incremento en el tiempo. De esta forma con la siguiente información:

- Posición en el instante  $i$ :  $x(i)$
- Velocidad en el instante  $i$ :  $v(i)$
- Tiempo entre actualizaciones:  $t$

Se puede obtener la posición en un instante de tiempo futuro, por ejemplo en  $i+1$ .

$$x[i + 1] = x[i] + v[i] \cdot t$$

Se puede aumentar la precisión de esta fórmula aumentando la información: añadiendo velocidad angular, rotación angular o aceleración si afectan al movimiento.

La precisión en la predicción está muy ligada al tipo de movimiento que estemos calculando. La extrapolación tiende a ser más exitosa en juegos con físicas, como simuladores de coches, en comparación con juegos donde el jugador puede cambiar su dirección de movimiento de manera instantánea. La razón principal está relacionada con la naturaleza de los movimientos y la previsibilidad en ambos tipos de juegos. Por ejemplo, en un juego de carreras, si un coche se está moviendo a 108 kilómetros hora o 30 metros por segundo, después de un segundo, lo más probable es que este coche esté a 30 metros de distancia. Aunque el jugador realice alguna maniobra, en función de su velocidad y dirección actuales, se puede estimar la posición futura con una precisión alta. Sin embargo, en juegos donde los jugadores pueden cambiar instantáneamente su dirección de movimiento, como en juegos de acción rápida, la extrapolación puede ser menos efectiva. La imprevisibilidad de los cambios repentinos en la dirección hace que sea más difícil prever con precisión el estado futuro de las entidades.

#### 2.3.2.4 Reconciliación

Cuando se reciben actualizaciones de red, se utilizan para corregir las estimaciones locales. Esto ayuda a mantener la coherencia entre todos los participantes del juego y reduce la posibilidad de desviaciones significativas.

En un entorno multijugador, cada cliente tiene su propio estado local del juego, mientras que el servidor también mantiene su propia versión del estado del juego. Sin embargo, debido a la latencia de red y otros factores, la información sobre el estado del juego puede llegar a los clientes en momentos diferentes. La reconciliación se utiliza para corregir cualquier desviación o discrepancia que pueda surgir entre el estado local del cliente y el estado del juego en el servidor.

#### 2.3.3. Delay Netcode y Rollback Netcode

Los juegos de lucha tienen sus propios retos. A diferencia de muchos otros géneros de juegos, una latencia baja y constante es extremadamente importante porque la memoria muscular y las reacciones son el núcleo de prácticamente todos los juegos de lucha. Como resultado, han surgido dos estrategias destacadas para los juegos de lucha en línea: *Delay Netcode* y *Rollback Netcode*.

*Delay Netcode* asegura que ambos jugadores estén perfectamente sincronizados. Sin embargo, dadas las dificultades naturales de Internet, esto no siempre funciona,

sobre todo si la latencia es elevada. En el caso de problemas de red, para mantener la sincronización, *Delay Netcode* detiene brevemente la acción, esperando a que la información entre los dos jugadores coincida, antes de continuar con el juego. Esto puede llevar a un resultado no deseado, dependiendo de la potencia de la conexión en línea de cada jugador. Puede dar lugar a una latencia deficiente entre el mando y la acción del juego, así como a congelaciones y tartamudeos en el juego, y a entradas completamente perdidas.

Cuando no hay información del jugador remoto, *Delay Netcode* necesita hacer una pausa y esperar. Sin embargo, en *Rollback Netcode* nunca se espera a que falte información del oponente. En su lugar, se sigue ejecutando el juego con normalidad. Todas las entradas del jugador se procesan inmediatamente, como si estuviera jugando en local. Luego, cuando la entrada del jugador remoto llega unos fotogramas más tarde, *rollback* corrige sus errores. Esto se lleva a cabo rebobinando la simulación, aplicando el nuevo input del jugador remoto y actualizando y mostrando el nuevo resultado inmediatamente.

Este rebobinado ocurre de forma instantánea, en un solo fotograma, por lo que el jugador local simplemente ve que el estado del juego que creía correcto (pero que era falso) se sustituye inmediatamente por el estado del juego realmente correcto. Es posible que los personajes pueden dar un pequeño salto y, en general, las animaciones de los movimientos del jugador remoto no serán reproducidas desde su inicio cuando se muestren al jugador.

Gracias a este método, el jugador puede que ni siquiera note la inestabilidad de la red y confiará que sus entradas son tratadas de forma instantánea y consistente.

En la tabla 2 se presenta un resumen de las diversas técnicas de compensación de latencia en juegos multijugador.

<b>Técnica</b>	<b>Problema abordado</b>	<b>Descripción</b>
Predicción del cliente	Latencia en la actualización del estado del juego.	Los clientes realizan predicciones locales sobre el estado del juego en función de los <i>inputs</i> del jugador, corrigiendo estas predicciones cuando se reciben actualizaciones del servidor.
Interpolación del cliente	Apariencia de saltos o movimientos bruscos.	Se interpola suavemente en el cliente las posiciones y estados de los objetos del juego entre actualizaciones, creando una transición gradual y suave.
Extrapolación del cliente y <i>Dead Reckoning</i>	Desviaciones entre el estado local y el estado real del juego.	Se predice la posición futura de objetos y jugadores basándose en información actual, asumiendo comportamientos invariables. Se corrigen las desviaciones mediante el uso de la técnica de reconciliación.
Reconciliación	Desviaciones entre el estado local y el estado del juego en el servidor.	Se utiliza para corregir cualquier discrepancia que pueda surgir entre el estado local del cliente y el estado del juego en el servidor, manteniendo la coherencia entre todos los participantes del juego.
<i>Delay Netcode</i> y <i>Rollback Netcode</i>	Sincronización en juegos de lucha.	<i>Delay Netcode</i> pausa la acción para mantener la sincronización entre jugadores, mientras que <i>Rollback Netcode</i> sigue ejecutando el juego y corrige errores de sincronización de forma instantánea.

Cuadro 2: Resumen de técnicas de compensación de latencia en juegos multijugador

## 2.4. Arquitecturas

Las arquitecturas en juegos multijugador se refieren a las estructuras que determinan cómo se gestionan, comparten y sincronizan los datos entre los distintos jugadores para permitir a estos jugar de manera simultánea.

Existen diversas arquitecturas para implementar videojuegos multijugador, la elección de la arquitectura dependerá de varios factores como el tipo de juego, la cantidad de jugadores, los requisitos de rendimiento y las limitaciones de red.

### 2.4.1. Cliente-servidor

En la arquitectura cliente-servidor, un servidor central gestiona el estado del juego y la lógica y envía información sobre el estado a los clientes. Los clientes son responsables

de representar gráficamente el juego y enviar sus entradas al servidor. Esta arquitectura facilita la gestión del estado del juego y la lógica centralizada y permite detectar trampas de los jugadores. Es escalable para juegos con un gran número de jugadores. Sin embargo es una arquitectura costosa al tener que proveer servidores externos para las partidas.

#### **2.4.2. Peer-to-Peer (P2P)**

En esta arquitectura cada cliente actúa como un nodo que se comunica directamente con otros clientes. No hay un servidor centralizado, la lógica del juego y el estado son compartidos entre los jugadores. Es una arquitectura menos costosa al no depender de infraestructura adicional. Sin embargo, es menos escalable y más difícil de gestionar en términos de sincronización y seguridad.

#### **2.4.3. Arquitectura híbrida**

Combina elementos de las arquitecturas cliente-servidor y P2P. En esta arquitectura, existe un servidor central que gestiona ciertos aspectos del juego, mientras que los jugadores también se comunican directamente entre ellos para algunas interacciones. Puede aprovechar las ventajas de ambas arquitecturas, como la centralización de ciertas operaciones y la reducción de costes.

#### **2.4.4. Modelo de cliente host**

Similar al modelo cliente-servidor, pero uno de los clientes actúa como el anfitrión (*host*). Este anfitrión, además de actuar como cliente y poder jugar como un jugador más, comparte parte de la carga de procesamiento y manejo del estado del juego como haría un servidor. Este modelo puede implicar un ahorro de costes y evitar el uso de trampas. Pero la calidad de la experiencia y la escalabilidad depende en gran medida de la conexión y capacidad del jugador anfitrión.

En la tabla 3 se presenta una comparativa de las diferentes arquitecturas en juegos multijugador, junto con sus ventajas y desventajas:

Arquitectura	Ventajas	Desventajas
Cliente-servidor	<ul style="list-style-type: none"> <li>■ Facilita la gestión del estado del juego y la lógica centralizada.</li> <li>■ Permite detectar trampas de los jugadores.</li> <li>■ Escalable para juegos con un gran número de jugadores.</li> </ul>	<ul style="list-style-type: none"> <li>■ Costosa al tener que proveer servidores externos.</li> </ul>
Peer-to-Peer (P2P)	<ul style="list-style-type: none"> <li>■ Menos costosa al no depender de infraestructura adicional.</li> </ul>	<ul style="list-style-type: none"> <li>■ Menos escalable.</li> <li>■ Más difícil de gestionar en términos de sincronización y seguridad.</li> </ul>
Arquitectura híbrida	<ul style="list-style-type: none"> <li>■ Combina ventajas de cliente-servidor y P2P.</li> <li>■ Puede reducir costes y aprovechar la centralización de ciertas operaciones.</li> </ul>	<ul style="list-style-type: none"> <li>■ Complicada de implementar y gestionar.</li> </ul>
Modelo de cliente host	<ul style="list-style-type: none"> <li>■ Puede implicar ahorro de costes.</li> <li>■ Evita el uso de trampas.</li> </ul>	<ul style="list-style-type: none"> <li>■ La calidad de la experiencia y la escalabilidad dependen de la conexión y capacidad del jugador "host".</li> </ul>

Cuadro 3: Ventajas y desventajas de arquitecturas en juegos multijugador

## 2.5. Protocolos de transporte

Los protocolos de transporte juegan un papel fundamental en la gestión de la comunicación entre dispositivos. Estos protocolos, ubicados en la capa de transporte del modelo OSI (Open Systems Interconnection), se encargan de asegurar que los datos enviados desde una aplicación en un dispositivo lleguen de manera correcta y eficiente a una aplicación en otro dispositivo. Los dos principales protocolos de transporte utilizados en los juegos multijugador son TCP (Transmission Control Protocol) y UDP (User Datagram Protocol), cada uno con características y ventajas distintas en términos de fiabilidad y latencia.

TCP (Transmission Control Protocol) : Este protocolo se basa en la creación de conexiones entre terminales previo al intercambio de datos, esto es llevado a cabo mediante un intercambio de mensajes conocido como "handshake" de tres vías, donde se negocian parámetros y se confirma la disposición para la transmisión de datos. *TCP* garantiza el envío con éxito de los datos sin errores y en orden de envío. Utiliza mecanismos de retransmisión para recuperarse de la pérdida de paquetes y números de secuencia para ordenar los paquetes en su recepción. Además *TCP* utiliza mecanismos

para verificar la integridad del paquete denominados *checksums*. Sin embargo, aunque este protocolo es muy fiable, introduce una mayor latencia en comparación con otros protocolos.

UDP (User Datagram Protocol): Este protocolo está basado en la transmisión sin conexión previa de datagramas. Esto significa que cada paquete UDP se envía de manera independiente y no se mantiene la conexión. Al contrario que *TCP*, *UDP* no garantiza el envío con éxito de los datos, ni el orden en el que son recibidos. Sin embargo, al prescindir de los mecanismos que permiten esto en *TCP*, *UDP* tiene una menor latencia, haciéndolo adecuado para aplicaciones en las que la velocidad de transmisión es crítica, como en juegos multijugador en tiempo real.

## 2.6. Estado del arte

En este apartado, se analizarán algunos videojuegos multijugador que han alcanzado un éxito significativo, y se explorarán alguna de las arquitecturas y técnicas más innovadoras que han adoptado para mejorar la conexión y la experiencia de los usuarios.

### 2.6.0.1 League of Legends

League of Legends es un popular juego en línea de tipo MOBA (Multiplayer Online Battle Arena) desarrollado y publicado por Riot Games. En este juego, los jugadores forman equipos para enfrentarse entre sí en partidas estratégicas en las que cada jugador controla un campeón con habilidades únicas.

Respecto a las técnicas multijugador que utiliza, League of Legends permite activar la predicción del cliente para el movimiento de los personajes, pero no para las habilidades. Esto significa que el cliente asume que las entradas que envía al servidor se ejecutarán correctamente, y actualiza la posición del personaje en consecuencia, mientras espera a que el servidor confirme el nuevo estado del juego.

En *Riot Games*, la empresa creadora de múltiples juegos multijugador muy conocidos y jugados mundialmente, se dieron cuenta que la latencia y el jitter son aspectos muy importantes en la experiencia de los jugadores, según Rodrigo "Rocofu" Fuentesvilla, Gerente de Infraestructura y Servicios de Riot Games Latinoamérica: "*El ping alto y/o variable ha sido siempre una espina clavada en nuestros jugadores por años, y siempre hemos hecho muchos esfuerzos para evitarles esas ganas de voltear la mesa o destruir teclados que a veces nos genera el mal ping.*" [2] Sin embargo, se dieron cuenta que el enrutamiento de los proveedores de servicios de internet o (ISP) trata que el tráfico de los clientes salga de su red lo antes posible utilizando el punto de salida más corto, no el más rápido [3].

Es por esto que decidieron crear su propia infraestructura de red llamada *Riot Direct* diseñada específicamente para optimizar la conexión de los jugadores a los servidores de Riot Games, mejorando así la calidad de la experiencia de juego. Esta infraestructura se suma a la infraestructura de *AWS* usando un servicio llamado *AWS Global Accelerator* que mejora la latencia al dirigir el tráfico al punto de acceso más cercano, conectado a su vez a los servidores de Riot Games de manera directa [4].

### 2.6.0.2 Valorant

*Valorant*, otro de los juegos con mayor éxito de Riot Games, es un FPS *tactical shooter* donde los jugadores además de tener armas, poseen habilidades que usarán para derrotar a sus enemigos. En este videojuego al ser un juego de disparos táctico muy jugado y altamente competitivo, la sincronización y la prevención de trampas son dos elementos muy importantes. Por esto, usa una arquitectura cliente-servidor, con servidores autoritativos. Estos servidores controlan el estado de la partida, previendo el uso de trampas ya que es el servidor el que controla las decisiones más importantes.

En entornos competitivos, cada milisegundo cuenta, y una baja latencia es esencial para asegurar que las acciones del jugador se reflejen de manera inmediata en el juego. La latencia también influye en la consistencia de la experiencia de juego entre jugadores ya que es fundamental que todos los participantes vean el mismo estado del juego en tiempo real. Es por esto que además de usar la infraestructura de Riot Direct y AWS Global Accelerator y aplicar técnicas de compensación de latencia como predicción del cliente, interpolación o extrapolación, Valorant también aplica otras soluciones. Algunas de estas son:

- *Tick rate* elevado. Los servidores de Valorant han sido diseñados para poder ejecutar cada instancia del juego 128 veces por segundo, actualizando el estado del juego cada 7,8 milisegundos aproximadamente. Con este *tick rate* elevado, el servidor proporciona actualizaciones más frecuentes sobre las acciones de los jugadores y el estado del juego, lo que resulta en una menor latencia, una interpolación más suave, una detección de impactos mejorada y una experiencia multijugador más consistente.
- Filtro de buffer de red. Valorant proporciona la opción a los jugadores para ajustar la velocidad a la que el cliente envía actualizaciones al servidor.
- Registro de impactos. Para el registro de impactos, característica vital en un shooter táctico, se usa una técnica de retroceso, similar a *Rollback Netcode* (ver apartado 2.3.3). Debido a que el sistema que usa Valorant es determinista, únicamente guardando los *inputs* relacionados con las animaciones, se puede recrear cualquier estado. con este sistema, se evita tener que guardar todos los *colliders* de cada estado del juego, si no que estos son calculados en la simulación.[5]

La ventaja del que asoma o *Peeker's advantage* en inglés, es un artefacto del juego en red que suele ser el centro de las discusiones sobre la integridad competitiva en los *shooters* tácticos. Se refiere a la ventaja que tiene alguien que se asoma a una esquina sobre un oponente que está al otro lado. La optimización de la red, así como el *tick rate* elevado proporcionando actualizaciones más frecuentes y la optimización en el cliente, permitiendo la ejecución del juego a fotogramas elevados incluso en dispositivos menos potentes hace que esta ventaja sea también mitigada.[6]

### 2.6.0.3 Counter strike 2

Counter Strike 2, es un videojuego de disparos táctico en primera persona multijugador de 2023 desarrollado y publicado por Valve. Ha sido desarrollado como una actualización del famoso videojuego *Counter Strike, Global Offensive* (2012).

Al igual que Valorant, debido a que CS2 es un juego de disparos táctico muy jugado y altamente competitivo tiene una arquitectura cliente-servidor con servidores distribuidos geográficamente. También aplica técnicas de compensación de latencia como la predicción del cliente, activada por defecto pero que los jugadores pueden desactivarla.

Sin embargo, en Counter Strike 2 en lugar de aumentar la frecuencia de actualización de los servidores, o *tick rate*, a 128Hz, necesitando mayores recursos y por lo tanto mayor inversión y mantenimiento, se decidió implementar un sistema en el que la información sobre las entradas del jugador incluye también la marca de tiempo en la que la entrada ha ocurrido. Este sistema hace que aunque dos acciones, por ejemplo dos disparos de jugadores, se ejecuten en el mismo tick del servidor, este tendrá la información precisa para resolver qué jugador disparó antes, limitando los efectos negativos de un *tick rate* menor. [7]

#### 2.6.0.4 Conclusiones

Después de haber analizado los videojuegos League of Legends, Valorant y Counter Strike 2, se pueden sacar varias conclusiones sobre algunas características y técnicas que usan para mitigar los efectos negativos de la red en videojuegos multijugador.

Todos estos casos de estudio usan una arquitectura cliente-servidor, esto es debido a múltiples factores, algunos de estos son:

- **Escalabilidad.** Debido a la cantidad de usuarios que juegan este tipo de videojuegos, en la arquitectura cliente-servidor los requisitos de red y computación no son tan elevados para los jugadores como en un modelo P2P, ya que una gran parte del procesamiento la realiza el servidor. Esta arquitectura puede ampliarse más fácilmente para abarcar a muchos jugadores, los servidores pueden distribuirse en varias máquinas y localizaciones, permitiendo ampliar la disponibilidad de partidas sin sacrificar rendimiento. [8]
- **Reducción de la carga de red.** Como el servidor maneja la mayoría del coste computacional y solo envía la información esencial a los clientes, se reduce la cantidad de datos que se necesita transmitir por la red, optimizando el uso del ancho de banda y reduciendo la latencia respecto a una estructura P2P, en la cual cada jugador necesita comunicarse directamente con los demás jugadores. Las configuraciones cliente-servidor tienen acceso a una mejor infraestructura de red mediante conexiones a centros de datos, que poseen una infraestructura de red muy potente, pero su latencia depende en última instancia de la proximidad del servidor a los usuarios. Por eso estos servidores están distribuidos por muchos centros de datos distribuidos geográficamente (ver figura 6).

Figura 6: Servidores dedicados de Valorant. Imagen obtenida de netduma.com



- Seguridad.** La seguridad y la prevención de trampas es vital para videojuegos tan jugados y competitivos, esto se consigue gracias a que los servidores son autoritativos, por lo que los clientes no pueden manipular directamente el estado del juego. El código fuente de los servidores es normalmente privado y no está disponible públicamente, por lo que los atacantes tienen más difícil encontrar vulnerabilidades, esto también es llamado *Seguridad por oscuridad*

Para prevenir el uso de trampas muchos de estos juegos también usan técnicas del lado del cliente, como el uso de la aplicación *Easy Anti-Cheat*.

Debido a la naturaleza de esta arquitectura, las IP de los clientes se mantienen ocultas entre sí, evitando un ataque de denegación de servicio hacia estos. Aunque la IP de los servidores es pública, estos usan técnicas para para detectar y mitigar los ataques de denegación de servicio del lado del servidor.

Estos juegos también comparten que la importancia de la sincronización del juego de una manera rápida es vital. Por eso intentan minimizarla de diversas formas para mantener una sincronización rápida y precisa del estado del juego en todos los clientes. También implementan técnicas de compensación de latencia llevando a cabo estimaciones locales en los clientes con el fin de mitigar la latencia. No obstante, si estas estimaciones difieren del estado que el servidor envía, el estado del cliente se ajusta para garantizar su consistencia con el estado del servidor.

---

## 3. Diseño del juego

### 3.1. Sinopsis

Shooter Stars es un videojuego multijugador 3D con vista cenital para PC y dispositivos móviles, al estilo Brawl Stars. El objetivo principal es eliminar a tus enemigos para conseguir la victoria. Los jugadores podrán jugar en diferentes modos de juego con diferentes personajes, también podrán jugar contra gente en línea o con sus amigos.

### 3.2. Controles

Acción	PC	Móvil
Moverse	A,W,S,D	Joystick Izquierdo
Apuntar	Click derecho	Mantener Joystick derecho
Disparar	Click izquierdo	Soltar Joystick derecho

### 3.3. Mecánicas

Shooter Stars utiliza mecánicas simples y dinámicas que permiten un manejo fluido en dispositivos móviles y PC. Los controles para cada jugador son diferentes dependiendo de la plataforma en la que estén jugando.

Pese a que existen tres diferentes tipos de personajes, cada uno con una serie de características, arma y manera de apuntar distintas, las mecánicas son similares.

Las principales mecánicas son las siguientes:

- **Moverse.** Los jugadores en ordenador pueden moverse usando las teclas "A", "W", "S" y "D" del teclado.  
Los jugadores en dispositivos móviles pueden moverse utilizando el *joystick* izquierdo visible en pantalla y deslizándolo hacia la dirección que deseen moverse, para dejar de moverse los jugadores pueden dejar el joystick en la zona central.
- **Apuntar.** Para apuntar, los jugadores en ordenador pueden usar el click derecho del ratón en la dirección en la que deseen apuntar.  
Para dispositivos móviles, los jugadores pueden usar el *joystick* derecho visible en pantalla y deslizarlo en la dirección en la que deseen apuntar, para dejar de apuntar pueden dejar el joystick en la zona central.
- **Disparar.** Los jugadores en ordenador pueden disparar mientras apuntan o sin apuntar, haciendo click en el botón izquierdo del ratón.  
Los jugadores en dispositivos móviles pueden disparar soltando el joystick derecho, y dejando de apuntar, en la dirección en la que quieren disparar.

- **Modos de juego.** Dependiendo del modo de juego, existen elementos que pueden cambiar. En la actualidad existen dos modos de juego.  
En el modo de juego "Team Deathmatch", los jugadores pueden elegir en qué equipo están. Esta mecánica afecta a la mecánica de daño y los puntos de respawn. En el modo de juego "Free for All", todos los jugadores se enfrentarán entre ellos.
- **Daño.** Los jugadores pueden recibir daño de los jugadores enemigos al recibir un disparo o impacto de estos. Cuando esto ocurre, la barra de vida de estos disminuirá. Esta barra de vida se muestra en rojo para los jugadores enemigos y en verde para los jugadores aliados.
- **Muerte y fin de partida.** Los jugadores mueren cuando reciben daño y su vida es igual o inferior a cero, esta baja incrementará el contador de muertes del jugador y equipo. Si el contador alcanza el máximo, definido por el host en la pantalla de lobby, aparecerá la pantalla de fin de partida. De no ser así, el jugador muerto, deberá esperar unos segundos para respawnear y seguir jugando.
- **Chat de voz.** Los jugadores pueden comunicarse entre ellos durante la totalidad de la partida. Pueden ajustar el volumen de su micrófono y de los demás jugadores en la pantalla de lobby, o pulsando la tecla "tabulador" o el botón "Show Statistics" en dispositivos móviles, durante la partida.

### 3.4. Arte

Para el arte 3D del juego se ha elegido un estilo cartoon. Los personajes y escenarios están diseñados con colores vibrantes, lo que les da un aspecto atractivo y llamativo. Todos los elementos 3D se han diseñado con la técnica de Low Poly, las animaciones han seguido un estilo sencillo y cartoon y los sistemas de partículas de las armas han sido creados con pocas instancias de partículas. Todo esto ha contribuido en la creación de una estética minimalista y en la mejora del rendimiento del juego para dispositivos móviles.

Para el arte 2D se ha elegido un estilo sencillo y minimalista evitando la saturación de las interfaces. Estos recursos 2D son utilizados para las interfaces, iconos e ilustraciones.

Todo esto crea una estética sencilla y amigable para los jugadores, como se puede ver en la figura 7.

Aunque el arte no es el objetivo principal de este TFG, es relevante destacar que la mayoría de los assets del juego han sido creados específicamente para el videojuego, con la excepción de los modelos de árboles y piedras.

#### 3.4.1. Interfaces de Usuario

Para las interfaces de usuario se ha seguido un enfoque intuitivo y funcional, con un diseño limpio y fácil de navegar. Se han utilizado elementos visuales coherentes con la estética general del juego, manteniendo la simplicidad y la claridad como prioridades.

- **Facilidad de uso:** Se ha priorizado la accesibilidad y la comprensión rápida de las interfaces por parte de los jugadores. Se han evitado elementos innecesarios o confusos, asegurando una experiencia fluida y sin obstáculos.

Figura 7: Estética del videojuego



- **Estilo visual:** Las interfaces de usuario siguen el mismo estilo visual que el resto del juego, con colores vibrantes y elementos gráficos que reflejan la estética cartoon y minimalista. Esto ayuda a mantener la coherencia visual y a reforzar la identidad del juego.
- **Funcionalidad:** Cada elemento de la interfaz se ha diseñado con un propósito claro y una funcionalidad específica. Se han incluido características como menús intuitivos, indicadores visuales y controles táctiles optimizados para dispositivos móviles.
- **Adaptabilidad:** Se ha tenido en cuenta la adaptabilidad de las interfaces a diferentes plataformas y tamaños de pantalla. Se han realizado pruebas exhaustivas para garantizar que las interfaces se vean y funcionen correctamente en una variedad de dispositivos y resoluciones.

Las interfaces de usuario se han diseñado con el objetivo de ofrecer una experiencia de juego cómoda y agradable, complementando la estética general del juego y facilitando la interacción del jugador con el mundo virtual.

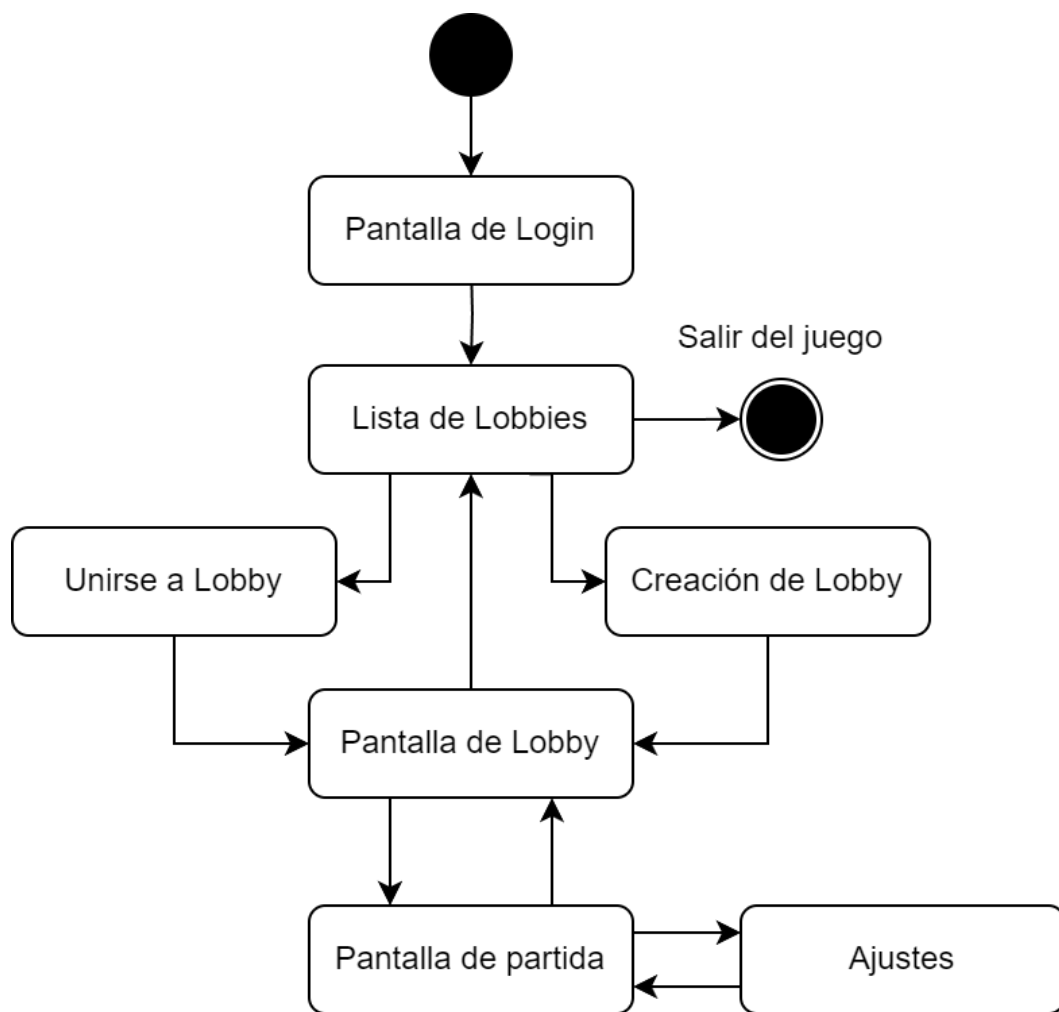
#### 3.4.1.1 Diagrama de navegación

A continuación se procede a explicar el diagrama de navegación 8, mostrando una visión sistemática de cómo los jugadores interactúan con las diversas partes del juego. En este apartado, se analiza el diseño de las rutas de navegación y la organización de las pantallas, diseñado con el objetivo de facilitar la interacción del usuario. Se ha tratado de ofrecer a los jugadores una experiencia de usuario intuitiva y coherente, guiándolos de manera eficiente a través de las distintas secciones y funcionalidades del juego.

Cuando un jugador ejecuta el videojuego se encontrará un menú donde puede iniciar sesión y elegir el nombre de usuario. Una vez ha iniciado sesión, el jugador verá una

lista de los lobbies abiertos disponibles. En esta pantalla el jugador podrá salir del juego o bien unirse a uno de estos lobbies abiertos, unirse a un lobby privado por medio de un código privado o crear su propio lobby, todas estas opciones llevan a la pantalla de lobby. En la pantalla de lobby el jugador podrá elegir su equipo, si el modo de juego lo permite, y elegir el personaje con el que quiere jugar, el jugador también puede volver a la lista de lobbies si lo desea. Una vez todos los jugadores están listos, el host iniciará la partida, lo que llevará a todos los jugadores a la partida en sí. Una vez en la partida los jugadores pueden abrir un pequeño menú de opciones. Cuando termina la partida o en cualquier momento de la partida, los jugadores podrán volver al lobby.

Figura 8: Diagrama de navegación - Máquina de estados



---

## 4. Análisis y diseño

### 4.1. Metodología empleada

Después de la planificación del diseño del juego, se procedió a la elaboración de los requisitos del proyecto. Estos requisitos, que surgieron de la planificación inicial, se organizaron y detallaron para guiar el desarrollo del juego de manera efectiva. Una vez establecidos, los requisitos se trasladaron a la plataforma Trello 5.1.4.2, donde se utilizaron como base para la gestión del proyecto.

Para la gestión del proceso de desarrollo se ha empleado una metodología ágil. El uso de esta metodología ha aportado una gran flexibilidad a la hora de elegir las funcionalidades a implementar, facilitando la mejora continua y la adaptación del desarrollo a los nuevos problemas emergentes y/o nuevas características

Dentro de las metodologías ágiles, se ha decidido usar la metodología Scrum, permitiendo dividir el trabajo en diferentes Sprints, cada uno de ellos con la intención de resolver una serie de objetivos. En el desarrollo del proyecto, los sprints han tenido una duración de tres semanas.

Para la gestión de los sprints se ha usado la herramienta Trello 5.1.4.2, creando un nuevo tablero por cada nuevo sprint y organizando las tareas según su estado (Backlog, Nuevo, En curso, Cerrado)

Antes del desarrollo de la programación del proyecto se añadieron todos los requisitos o tareas enumeradas en la columna de *Backlog*. Al empezar el primer sprint, las tareas correspondientes a este han sido movidas a la columna *To Do*. Conforme dichas tareas se ponían en marcha se pasaban a la columna *Doing* y al finalizarlas a la columna *Done*. Aunque se ha tratado de finalizar todas las tareas calculadas para un sprint esto no ha sido siempre posible, quedando en algunos sprint alguna tarea en las columnas de *To Do* o *Doing* y pasando al siguiente sprint. Esto ha sido posible gracias a la metodología ágil, que permite gran flexibilidad y la posibilidad de añadir requisitos emergentes conforme a la evolución del proyecto.

Durante el desarrollo del prototipo, se ha realizado un testeo y evaluación constante, combinada con la participación activa de usuarios reales. Se han tenido en cuenta las opiniones y comentarios de los usuarios para realizar mejoras y optimizaciones continuas. La flexibilidad que ha aportado la metodología ágil, ha permitido realizar ajustes continuos a medida que se identificaban áreas de mejora, bugs y feedback de los usuarios. Este enfoque centrado en la validación e iteración constante ha contribuido a la entrega de un producto final robusto y atractivo.

### 4.2. Requisitos

Los requisitos funcionales son las especificaciones de las acciones y funciones que se esperan del videojuego. Estos requisitos se centran en describir las operaciones específicas que el videojuego debe llevar a cabo en respuesta a ciertas entradas o eventos, definiendo qué debe hacer el sistema.

A continuación, se procede a listar los requisitos funcionales “RF”:

RF1. Creación proyecto Unity y repositorio Github

RF2. Implementación del jugador con movimiento autoritativo.

- RF3. Creación de la estructura hereditaria para diferentes armas
- RF4. Implementación de vida de los jugadores y puntos de spawn
- RF5. Creación de escena de Lobby.
- RF6. Configuración de servicios de Unity : Relay, Lobby y Vivox
- RF7. Autenticación anónima y campo para cambio de nombre.
- RF8. Implementación creación de lobby con personalización.
- RF9. Implementación actualización de lobbies existentes.
- RF10. Implementación unirse a lobby y sincronización de clientes.
- RF11. Implementación des-sincronización, expulsión y salida de jugadores del lobby.
- RF12. Implementación de audio entre jugadores con Vivox.
- RF13. Creación modelos Low Poly.
- RF14. Creación animaciones.
- RF15. Implementación modelos y animaciones.
- RF16. Implementación del cuadro de estadísticas.
- RF17. Implementación modo de juego "Contra todos".
- RF18. Implementación de sonido y feedback.

Los requisitos no funcionales se refieren a los criterios que describen las cualidades y características del sistema que no están relacionadas directamente con sus funciones específicas. En lugar de definir qué debe hacer el sistema, los requisitos no funcionales se centran en cómo debe hacerlo.

A continuación, se procede a listar los requisitos no funcionales "RNF":

- RNF1. Investigación del funcionamiento netcode, Relay, Vivox y Lobby
- RNF2. Implementación de dependencias (Netcode, Lobby, Vivox y ParelSync)
- RNF3. Investigación predicción del cliente.
- RNF4. Implementación predicción del cliente.
- RNF5. Implementación extrapolación y reconciliación.

### 4.2.1. Casos de uso

En el contexto del desarrollo de videojuegos, los casos de uso nos proporcionan una visión detallada de cómo los jugadores interactúan con el juego y qué acciones pueden realizar en diferentes escenarios. En esta sección, se presentan una serie de casos de uso que describen las principales funcionalidades y características del juego desde la perspectiva del jugador. Cada caso de uso identifica los actores involucrados, las acciones que pueden llevar a cabo y los resultados esperados, proporcionando así una guía clara para el desarrollo y la evaluación del juego.

#### UC1: Ejecución del videojuego

- **Actor principal:** Usuario.
- **Precondiciones:**
  1. PC: Tener el ejecutable.
  2. Android: Tener el APK instalado.
- **Garantías de éxito:** El usuario puede ejecutar el juego.
- **Escenario principal de éxito:**
  1. Abrir la carpeta que contiene el juego.
  2. Ejecutar el archivo.

#### UC2: Iniciar sesión

- **Actor principal:** Usuario.
- **Precondiciones:** UC1.
- **Garantías de éxito:** Tener acceso a la lista de salas públicas.
- **Escenario principal de éxito:**
  1. Escribir un nombre en el campo donde pone "Player Name".
  2. Pulsar en el botón de "Authenticate".
- **Imagen:** 9

#### UC3: Entrar en una sala pública

- **Actor principal:** Usuario.
- **Precondiciones:** UC2.
- **Garantías de éxito:** El usuario puede entrar en una sala pública.
- **Escenario principal de éxito:**
  1. Seleccionar una sala de la lista de salas.
- **Imagen:** 10

**UC4: Entrar en una sala privada**

- **Actor principal:** Usuario.
- **Precondiciones:**
  1. UC2.
  2. El usuario conoce el código privado de una sala.
- **Garantías de éxito:** El usuario puede entrar en una sala privada.
- **Escenario principal de éxito:**
  1. Introducir el código de la sala privada en el campo "Enter Lobby Code..."
  2. Pulsar en el botón "Join"
- **Imagen:** 10

**UC5: Crear una sala**

- **Actor principal:** Usuario host.
- **Precondiciones:** UC2
- **Garantías de éxito:** El usuario puede crear una sala.
- **Escenario principal de éxito:**
  1. Pulsar en el botón "+".
  2. Seleccionar los parámetros de la partida
  3. Pulsar en el botón "CREATE"
- **Imagen:** 10 y 11

**UC6: Empezar una partida**

- **Actor principal:** Usuario host.
- **Precondiciones:** UC5
- **Garantías de éxito:** El usuario host puede empezar la partida.
- **Escenario principal de éxito:**
  1. El usuario host debe esperar a que desaparezca el texto "Syncing players" y aparezca el botón "Start".
  2. Pulsar en el botón "Start"
- **Imagen:** 12

**UC7: Cambiar volumen de un jugador en lobby**

- **Actor principal:** Usuario
- **Precondiciones:** UC3 o UC4 o UC5 y al menos un usuario presente en la sala.

- **Garantías de éxito:** El usuario puede cambiar el volumen a cualquier otro usuario.
- **Escenario principal de éxito:**
  1. El usuario debe mover el "scrollbar" de volumen de otro usuario.
- **Imagen:** 12

#### UC8: Silenciar o dejar de silenciar a un jugador en lobby

- **Actor principal:** Usuario
- **Precondiciones:** UC3 o UC4 o UC5 y al menos un usuario presente en la sala.
- **Garantías de éxito:** El usuario puede silenciar o dejar de silenciar a cualquier otro usuario.
- **Escenario principal de éxito:**
  1. El usuario debe pulsar la imagen de altavoz de un usuario.
- **Imagen:** 12

#### UC9: Abrir estadísticas de la partida

- **Actor principal:** Usuario
- **Precondiciones:** UC3 o UC4 y UC6.
- **Garantías de éxito:** El usuario puede abrir el panel de estadísticas de la partida
- **Escenario principal de éxito:**
  1. PC: El usuario debe presionar la tecla tabulador
  2. Móviles: El usuario debe pulsar el botón "Show Stats"
- **Imagen:** 14

#### UC10: Cambiar volumen de un jugador en partida

- **Actor principal:** Usuario
- **Precondiciones:** UC9.
- **Garantías de éxito:** El usuario puede silenciar o dejar de silenciar a cualquier otro usuario.
- **Escenario principal de éxito:**
  1. El usuario debe mover el "scrollbar" de volumen de otro usuario.
- **Imagen:** 14

#### UC11: Silenciar o dejar de silenciar a un jugador en partida

- **Actor principal:** Usuario

- **Precondiciones:** UC9.
- **Garantías de éxito:** El usuario puede cambiar el volumen a cualquier otro usuario.
- **Escenario principal de éxito:**
  1. El usuario debe pulsar la imagen de altavoz de un usuario.
- **Imagen:** 14

#### UC12: Jugar partida

- **Actor principal:** Usuario
- **Precondiciones:** UC3 o UC4 y UC6.
- **Garantías de éxito:** El usuario puede jugar una partida.
- **Escenario principal de éxito:** PC:
  1. Usar las teclas "A", "W", "S", "D" para moverte.
  2. Usar click izquierdo para disparar.
  3. Usar click derecho para saltar

Móvil:

1. Usar el Joystick izquierdo para moverte.
  2. Mantener el Joystick derecho para apuntar.
  3. Soltar el Joystick derecho para disparar.
- **Imagen:** 13

#### UC13: Volver a sala

- **Actor principal:** Usuario
- **Precondiciones:** UC9.
- **Garantías de éxito:** El usuario puede volver a la sala.
- **Escenario principal de éxito:**
  1. El usuario debe pulsar el botón "Back to Lobby".
- **Imagen:** 14

#### UC14: Volver a lista de salas

- **Actor principal:** Usuario
- **Precondiciones:** UC3, UC4, UC5 o UC13.
- **Garantías de éxito:** El usuario puede volver a la lista de salas.
- **Escenario principal de éxito:**
  1. El usuario debe pulsar el botón de atrás " "
- **Imagen:** 12

### 4.3. Ejemplos de uso

A continuación se presentan los ejemplos de uso de la aplicación. La aplicación se encuentra operativa, por lo que también se puede probar en la siguiente url: <https://misterzzeta.itch.io/shooter-stars>

Figura 9: Autenticación del usuario

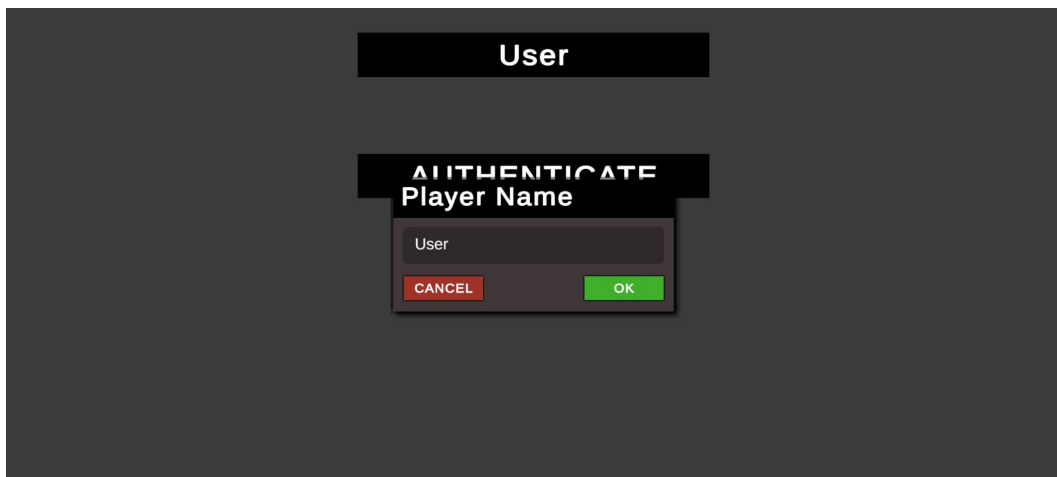


Figura 10: Lista de salas



Figura 11: Creación de Lobby

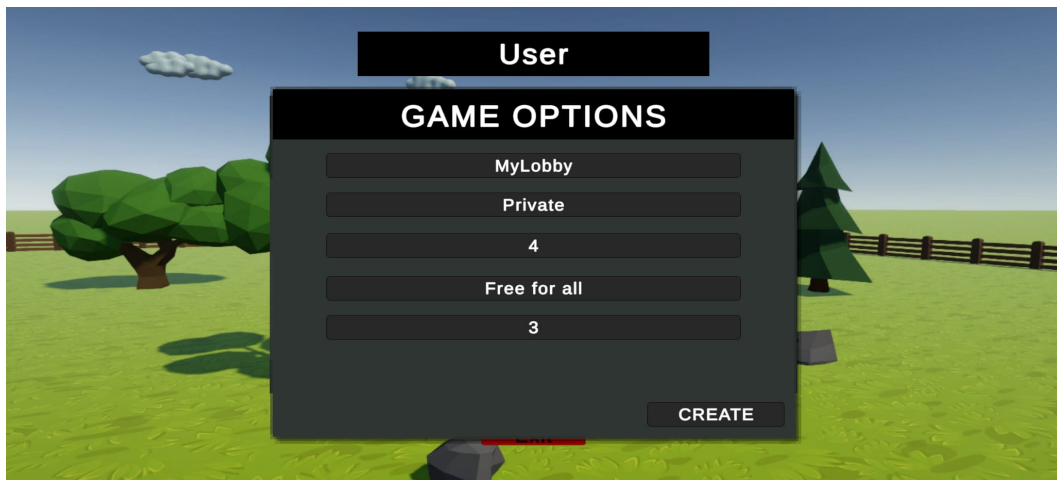


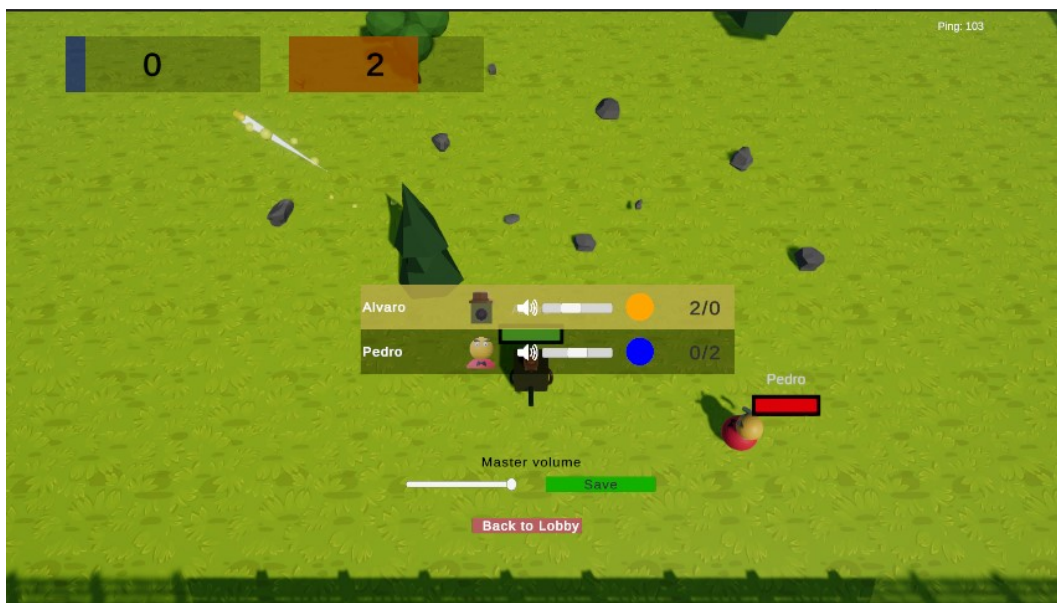
Figura 12: Sala de la partida



Figura 13: Captura de la partida



Figura 14: Menú de estadísticas de la partida

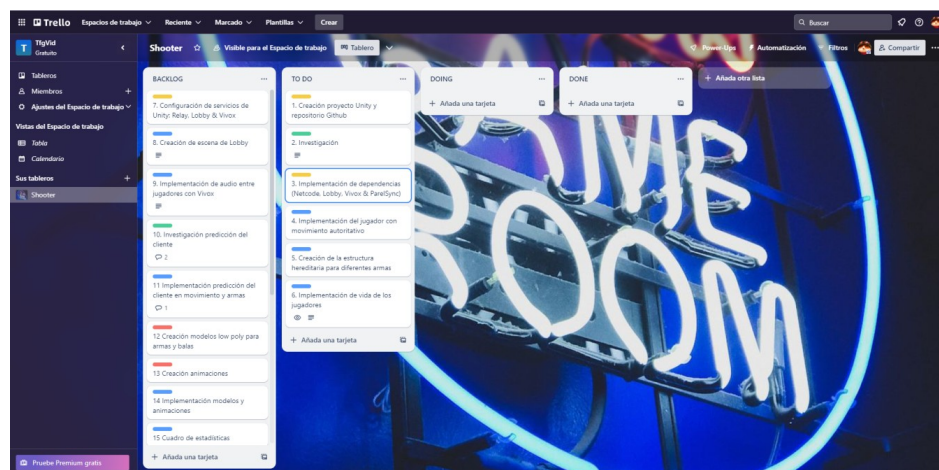


## 4.4. Proceso de desarrollo

A continuación, se procede a explicar los objetivos principales de los cuatro diferentes sprints que se han llevado a cabo dentro de la metodología ágil SCRUM.

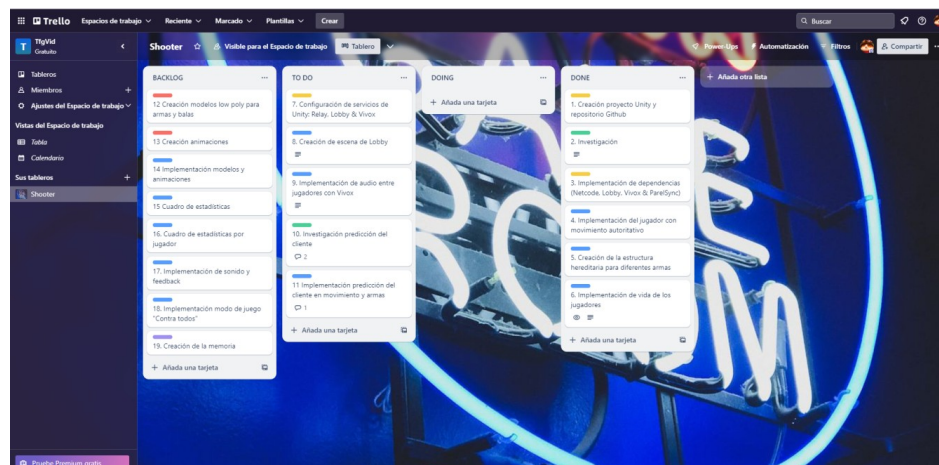
- Sprint 1** El principal objetivo de este sprint ha sido la investigación del desarrollo del proyecto y el estudio de las diferentes opciones de las tecnologías usadas en el proyecto. También se han instalado las dependencias y herramientas necesarias y se han implementado las mecánicas y elementos visuales básicos tanto para dispositivos móviles como para ordenador. Los requisitos de este sprint han sido los siguientes: RF1., RF2., RF3., RF4., RNF1. y RNF2.. El tablero de este sprint se puede ver en la figura 15.

Figura 15: Comienzo Sprint 1



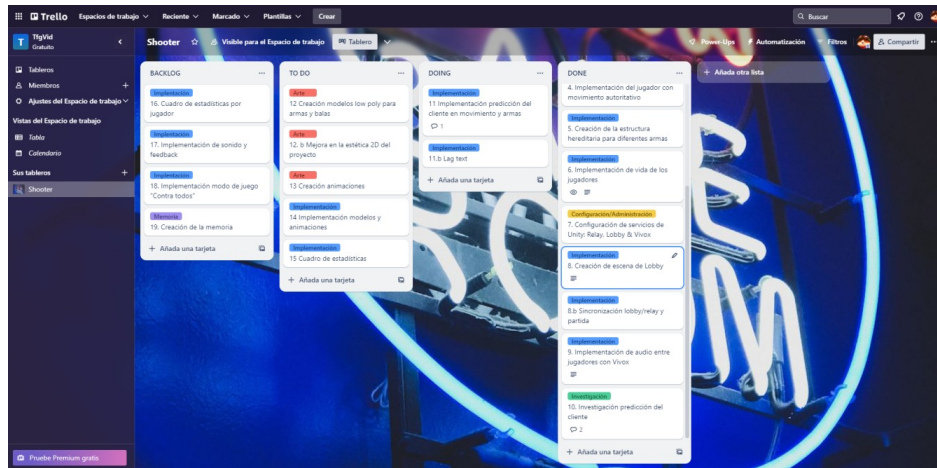
- Sprint 2** En este sprint se han implementado los servicios de Lobby 5.1.3 y de Vivox 5.1.3. También se ha estudiado las diferentes técnicas de red para juegos multijugador y se ha implementado alguna de estas. Los requisitos de este sprint han sido los siguientes: RF5., RF6., RF7., RF8., RF8., RF9., RF10., RF11., RF12., RNF3. y RNF4.. El tablero de este sprint se puede ver en la figura 16.

Figura 16: Comienzo Sprint 2



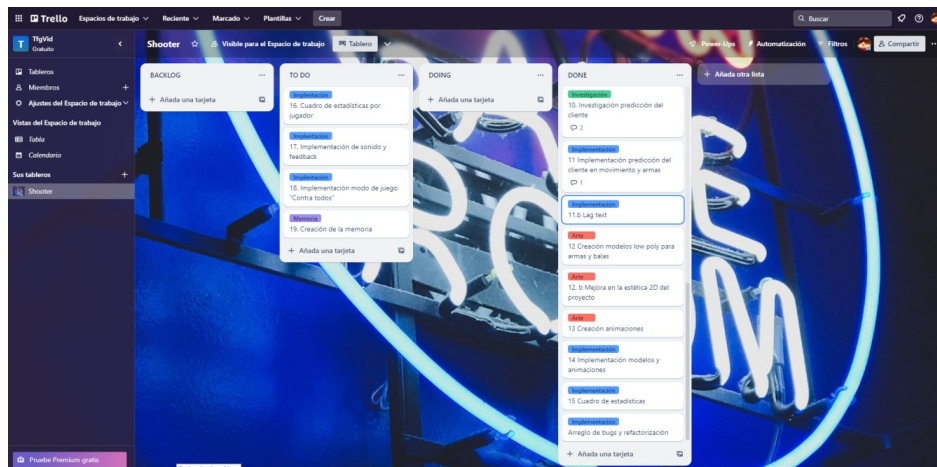
- Sprint 3** En el sprint 3 se terminó la implementación de las técnicas de red de predicción, extrapolación, reconciliación y Deterministic Lockstep para los jugadores. También se diseñó e implementó los apartados visuales y estéticos del videojuego. Los requisitos de este sprint han sido los siguientes: RF13., RF14., RF15., RF16. y RNF5.. El tablero de este sprint se puede ver en la figura 17.

Figura 17: Comienzo Sprint 3



- Sprint 4** Este último sprint se ha utilizado principalmente para la implantación de pequeños mejores para la mejora del videojuego y el desarrollo de la memoria. También se ha añadido un nuevo modo de juego y sonido. Los requisitos de este sprint han sido los siguientes: RF17. y RF18.. El tablero de este sprint se puede ver en la figura 18.

Figura 18: Comienzo Sprint 4



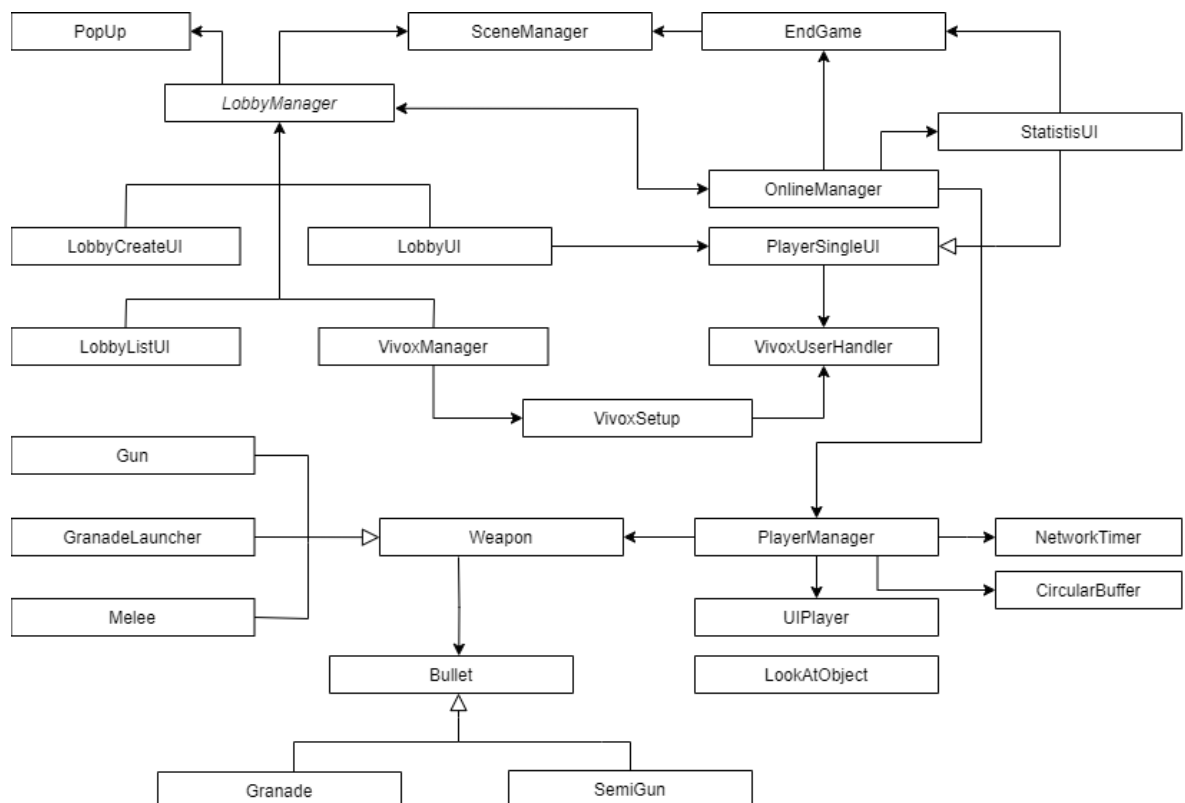
## 4.5. Diagrama de clases

El diagrama de clases es una representación fundamental del diseño técnico del juego, reflejando la estructura y las relaciones entre las distintas clases del sistema. Este diagrama final es el resultado del desarrollo iterativo y la interacción continua empleando la metodología ágil, la cual ha permitido adaptar y refinar el diseño del software a lo largo del proceso de desarrollo. Para simplificar y evitar que el diagrama se haga excesivamente grande, no se han incluido los detalles sobre atributos y métodos. En su lugar, se presenta una visión general que destaca las interacciones y dependencias clave entre las clases, proporcionando una comprensión general y estructurada del código subyacente del juego.

En la ilustración 19 se pueden observar las clases más importantes del proyecto. Podemos dividir las clases según su área de funcionamiento:

- **Lobby** (Parte superior derecha) Esta sección del diagrama se encarga de todas las funciones del lobby. Podemos destacar las siguientes clases:
  - **LobbyManager**. Esta clase es la encargada de manejar la lógica principal del lobby. La principal función de esta clase es la de crear, unirse o abandonar un lobby. Para lograr esto, debe encargarse de diferentes aspectos, como manejar los *Heartbeat*, crear o unirse a un relay o manejar el código del lobby. Para realizar estas funciones se ha utilizado el servicio de 5.1.3
  - **LobbyCreateUI**. Se encarga de mostrar y manejar las opciones existentes a la hora de la creación del lobby.
  - **LobbyUI**. Se encarga de crear y mostrar a los jugadores en la pantalla de lobby mediante el uso de un *prefab* que contiene la clase *PlayerSingleUI*. También se encarga de la selección de personajes, modo de juego, equipos y de mostrar mensajes informativos.
  - **LobbyListUI**. Su principal función es la de la actualización de la lista de los lobbies existentes.
- **Vivox** Esta sección abarca las clases de *VivoxManager*, *VivoxSetup* y *VivoxUserHandler* y se encarga de la gestión del chat de audio del juego mediante los servicios de 5.1.3. Esta gestión trata elementos como subir el volumen, bajar el volumen, mutearse, entrar a "salas de audio." salirse de las salas de audio.
- **Juego**. Esta sección, ubicada en la parte inferior y derecha de la imagen, se encarga de la lógica del juego. En esta sección destaca la clase *PlayerManager*, la cual se encarga del control de los jugadores, implementando *state sincronitation* y técnicas de compensación de latencia. Esta clase implementa la clase *weapon*, la cual puede derivar a las clases *Gun*, *GranadeLauncher* y *MeleeLogic*, estas armas se encargan de las funciones de disparo y de apuntado. La clase *Bullet* es también la clase padre de *Melee* y de *Granade*, estos scripts se encargan del movimiento de la bala y del daño que causan, así como del área de impacto.

Figura 19: Diagrama de clases reducido



---

## 5. Descripción informática

En este capítulo se explican las diferentes tecnologías utilizadas, así como los motivos por los que han sido elegidas. También se explica el funcionamiento del proyecto desde un punto de vista técnico, entrando en detalle sobre la arquitectura del mismo y sobre la implementación y técnicas utilizadas.

### 5.1. Tecnologías y herramientas utilizadas

A la hora de empezar con el desarrollo, es vital hacer un estudio previo sobre las diferentes opciones y alternativas existentes. Para ello es necesario hacer un estudio evaluando las ventajas e inconvenientes de las diferentes posibilidades para las elecciones más importantes. En el caso de este proyecto, al tratarse de un videojuego multijugador, las principales decisiones han sido: la elección de un motor, la elección de un framework multijugador y la elección de un proveedor de servicios para albergar la capacidad multijugador. Para esto se ha decidido usar matrices de decisión, comparando las diferentes opciones de cada aspecto según múltiples criterios. Después cada criterio recibe una puntuación y se asigna un peso según su importancia relativa.

#### 5.1.1. Motor de videojuegos

Para la elección del motor, se han elegido los siguientes candidatos:

- **Unity.** Motor de videojuegos multiplataforma creado por *Unity Technologies*, para su desarrollo permite la programación en C#. Unity es uno de los motores más utilizados en la industria y gracias a esto cuenta con una extensa comunidad de desarrolladores y una amplia variedad de assets en la Asset Store, así como numerosos recursos en línea.
- **Unreal Engine 5.** También es un motor de videojuegos multiplataforma, fue creado por *Epic Games*. *Unreal Engine* es otro de los motores más utilizados en la industria, destacando por su calidad de gráficos y efectos visuales y por su sistema de desarrollo por *Blueprints* y desarrollo mediante C++. Unreal Engine permite el desarrollo en videojuegos 2D, pero está más orientado para juegos 3D con una alta calidad gráfica.
- **Godot.** Es un motor de juegos multiplataforma de código abierto y gratuito, por lo que los desarrolladores pueden distribuir y comercializar sus juegos sin tener que pagar royalties. Permite el desarrollo en su propio lenguaje, *GDScript*, aunque también se ha añadido soporte para C++, C# y VisualScript.
- **GameMaker Studio.** Es un motor de videojuegos creado originalmente por *Mark Overmars*. *GameMaker Studio* destaca por su enfoque intuitivo y amigable para el desarrollo de juegos 2D, siendo una elección popular para juegos en este estilo. Permite el desarrollo de videojuegos mediante su propio lenguaje, *Gamaker Language (GML)*

Para la elección del motor del videojuego se han usado diferentes criterios, a los cuales se les ha asignado un peso en función de su relevancia:

- **Soporte multiplataforma** Este criterio evalúa la capacidad y facilidad del motor para el desarrollo de videojuegos tanto en dispositivos móviles como en PC. Dado que la accesibilidad del juego en ambas plataformas es un factor muy importante en el proyecto, se asigna un peso de 3/10.
- **Soporte multijugador** Este criterio considera la facilidad y la eficacia con la que el motor del videojuego permite la implementación de características multijugador. Dado que el modo multijugador es una característica vital en el desarrollo de este proyecto, se asigna un peso de 3/10.
- **Documentación** La calidad y la exhaustividad de la documentación asociada al motor del videojuego son críticas para el desarrollo eficiente y sin problemas del juego. Una documentación clara y bien organizada puede facilitar el aprendizaje y la resolución de problemas para los desarrolladores. Por lo tanto, se asigna un peso de 2/10 a este criterio.
- **Conocimiento previo** Este criterio evalúa la familiaridad y la experiencia previa del equipo de desarrollo con el motor del videojuego. Un mayor nivel de experiencia previa puede acelerar el proceso de desarrollo y reducir los riesgos asociados con la curva de aprendizaje. Sin embargo, no es el único factor determinante, por lo que se asigna un peso del 2/10 a este criterio para equilibrar su influencia con respecto a los otros criterios.

Nombre	Soporte multiplataforma (Peso: 3)	Soporte multijugador (Peso: 3)	Documentación (Peso: 2)	Conocimiento previo (Peso: 2)	Total (/100)
Unreal Engine	9	9	7	7	79
<b>Unity</b>	<b>9</b>	<b>9</b>	<b>8</b>	<b>9</b>	<b>88</b>
Godot	9	9	7	4	76
GameMaker	9	7	7	5	72

Como se puede observar, el motor de videojuegos Unity obtuvo la puntuación más alta. En consecuencia, se determinó que es el motor más adecuado para este proyecto, y por lo tanto, ha sido seleccionado para el desarrollo.

Para facilitar el desarrollo multijugador del proyecto con Unity, se ha decidido usar *ParrelSync*. Parrelsync es una herramienta de código abierto para Unity 5.1.1, la cual permite a los desarrolladores el testing de videojuegos multijugador sin tener que hacer una build del proyecto. Esto es llevado a cabo clonando los elementos esenciales del proyecto, mediante links simbólicos, en otro editor de Unity. Después, ParrelSync aplica estos cambios del proyecto original en todas las copias del proyecto.

### 5.1.2. Elección de la librería multijugador

Debido a la anterior elección del motor Unity Engine, para la elección de la librería multijugador se han elegido los siguientes candidatos:

- **Netcode for GameObjects (NGO)**. *Netcode for GameObjects* es una librería de red de alto nivel para Unity basada en la tecnología *Transport Layer API*. *NGO* permite enviar GameObjects y datos a través de una sesión de red a muchos

jugadores a la vez, destacando por su capacidad para manejar grandes cantidades de objetos en red con una sobrecarga mínima en el rendimiento del juego. Esto permite abstraerse de la lógica de red, permitiendo a los desarrolladores centrarse en la construcción del juego en vez de en protocolos de bajo nivel.

Aunque Netcode for Gameobjects sea una solución relativamente nueva, es una solución oficial de Unity , lo que puede significar una mayor estabilidad y soporte y ha sido diseñada para integrarse perfectamente con Unity y su ecosistema. Además NGO está diseñado para trabajar con objetos, lo que significa que estos objetos pueden estar sincronizados de manera más fácil y eficiente.

- **Mirror**. Mirror [9] es una librería de red Open Source para Unity también basada en la tecnología *Transport Layer API*. Está construido sobre la capa de comunicación en tiempo real de transporte de nivel inferior y se encarga de muchas de las tareas habituales que requieren los juegos multijugador. Destacando la sencillez y flexibilidad de su API de programación. Aunque *Mirror* no sea una solución oficial de Unity , también cuenta con una comunidad activa y comprometida de desarrolladores.
- **Photon Unity Networking (PUN)**. Es una librería para el desarrollo de juegos multijugador para Unity . Proporciona una API sencilla para sincronizar el estado del juego entre los jugadores y manejar la comunicación en tiempo real. Esta librería está ligada a los servicios en la nube que ofrece Photon Cloud [10], usándolos como backend. Además, Photon ha sido utilizado en muchos proyectos exitosos, lo que demuestra su capacidad para manejar cargas de trabajo exigentes.
- **Coherence**. Coherence [11] es una solución de red publicada en 2022, que proporciona una gran variedad de herramientas para el desarrollo de videojuegos multijugador. Coherence proporciona una integración sencilla, una buena integración con Unity y APIs sencillas, aunque es una solución relativamente nueva y menos usadas que las demás alternativas.

Para la elección de la librería multijugador también se ha usado una matriz de decisión, los criterios han sido los siguientes:

- **Facilidad de uso** Este criterio evalúa la facilidad con la que el equipo de desarrollo puede integrar y utilizar la librería multijugador en el proyecto del videojuego. Dado que la facilidad de uso es un factor crucial para el desarrollo eficiente y sin problemas, se le asigna un peso alto de 3/10.
- **Soporte de la librería** Este criterio considera el nivel de soporte proporcionado por la librería multijugador, incluyendo actualizaciones frecuentes, corrección de errores, y disponibilidad de recursos adicionales como foros de comunidad o soporte técnico. Un buen soporte puede ser crucial para resolver problemas rápidamente y mantener el desarrollo en marcha sin contratiempos, por lo que se le asigna un peso alto de 3/10.
- **Documentación** La documentación asociada a la librería multijugador es importante para comprender su funcionamiento y utilizar sus funciones de manera efectiva. Aunque la documentación es valiosa, se le asigna un peso ligeramente menor de 2/10 en comparación con la facilidad de uso y el soporte de la librería, ya que estos dos últimos criterios pueden tener un impacto más directo en el proceso de desarrollo.

- **Conocimiento previo** Este criterio evalúa la experiencia previa del equipo de desarrollo con la librería multijugador. Si el equipo ya tiene experiencia previa con la librería, puede trabajar de manera más eficiente y evitar posibles obstáculos durante el desarrollo. Sin embargo, se le asigna un peso moderado de 2/10, ya que otros factores como la facilidad de uso y el soporte de la librería también son importantes y pueden compensar la falta de experiencia previa.

Nombre	Facilidad		Documentación	Conocimiento	Total (/100)
	de uso (Peso: 3)	Soporte (Peso: 3)	(Peso: 2)	previo (Peso: 2)	
<b>NGO</b>	<b>9</b>	<b>9</b>	<b>7</b>	<b>7</b>	<b>82</b>
Mirror	8	7	8	9	79
Photon	8	7	7	1	61
Coherence	7	7	7	1	58

Como se puede observar, el framework de NGO fue el que mayor puntuación obtuvo. En consecuencia, se determinó que es el mejor framework multijugador para este proyecto, y por lo tanto, ha sido elegido para el desarrollo.

### 5.1.3. Elección del proveedor de servicios

Tareas como el emparejamiento inicial o la búsqueda de partidas requieren el uso de servidores dedicados. Esto sucede incluso cuando la arquitectura del videojuego no requiera servidores que realicen la función de host.

Es una práctica habitual que muchos videojuegos opten por usar un proveedor de servicios independiente de la plataforma en la que se ejecuta el videojuego. Esta decisión, aunque puede tener un coste adicional, incrementa la flexibilidad y control sobre el backend, además como ocurre en este proyecto, es vital para el funcionamiento en diversas plataformas.

Debido a la elección previa de NGO, algunos proveedores de servicios para videojuegos multijugador como Photon Cloud han sido descartados.

Existen una gran cantidad de proveedores de servicios, la mayoría proporcionan hosting de servidores, los cuales no son necesarios debido a la arquitectura de nuestro juego. Sin embargo, muchos de ellos no proporcionan herramientas específicas para videojuegos multijugador, por lo que también han sido descartados.

A continuación se muestran algunos de los principales candidatos:

- **Unity Services** , con herramientas como Relay Unity 5.1.3 o Lobby Unity 5.1.3. *Unity Services* es un conjunto de herramientas y servicios en la nube que están basados para funcionar como complementos para videojuegos desarrollados en cualquier motor, pero especialmente en Unity . Por esto es una alternativa ideal para desarrolladores que ya utilizan Unity como motor de juego y quieren aprovechar su ecosistema integrado. Sin embargo, Unity Services puede no ofrecer el mismo nivel de flexibilidad y personalización que otros proveedores en la nube.
- **Google Cloud**, con herramientas como *OpenMatch*. Google Cloud es una plataforma integral en la nube que ofrece una amplia gama de productos y soluciones para diversos sectores y casos de uso, incluidos los juegos. Algunas

de las características incluyen Google Cloud Game Servers, que es un servicio totalmente gestionado que le permite implementar y gestionar clústeres de servidores de juegos.

Aunque Google Cloud permite la integración de diversos servicios, como OpenMatch, un sistema flexible de emparejamiento [12], o Firebase Leaderboard, una tabla de clasificación multiplataforma para Unity [13], es importante señalar que estos servicios pueden presentar una mayor complejidad en la integración en comparación con los servicios ofrecidos por Unity .

- **AWS.** *Amazon Web Services (AWS)* es una colección de servicios en la nube que en conjunto forman una plataforma en la nube. AWS es el proveedor de servicios más usado debido a diversos factores como la gran cantidad de servicios que ofrecen, la alta disponibilidad que garantiza una baja latencia y su reducción de costes debido a su política de pagar solo por lo que usas o *pay as you go* [14]. Esta plataforma tiene un conjunto de servicios dedicados a videojuegos, con herramientas como *AWS GameLift*, orientada al alojamiento de servidores con posibilidad de integración con otros servicios y *AWS GameLift FlexMatch*, un servicio de matchmaking personalizable para partidas multijugador. Pese a la experiencia anterior del equipo de desarrollo con este proveedor y aunque al igual que en *Google Cloud* AWS permite integrar diversos servicios para videojuegos tanto suyos como de terceros, esta integración resulta más compleja en comparación con los servicios de Unity .

En general, la elección del proveedor de servicios depende de las necesidades específicas del desarrollador y del proyecto. Siendo las tres opciones soluciones sólidas y ofreciendo una amplia gama de características. Debido a que Shooter Stars es un proyecto independiente desarrollado por un solo desarrollador, la facilidad de uso a tenido un peso importante.

Se han usado los siguientes criterios para la evaluación:

- **Facilidad de uso** Este criterio evalúa la facilidad con la que el equipo de desarrollo puede integrar y utilizar los servicios del proveedor en el proyecto del videojuego. Dado que la facilidad de uso es crucial para un desarrollo eficiente y sin problemas, se le asigna un peso alto de 3/9
- **Documentación** La calidad y accesibilidad de la documentación asociada a los servicios del proveedor es importante para comprender su funcionamiento y utilizar sus herramientas de manera efectiva. Aunque la documentación es valiosa, se le asigna un peso moderado de 2/9 en comparación con la facilidad de uso, ya que esta última puede tener un impacto más directo en el proceso de desarrollo.
- **Conocimiento previo** Este criterio evalúa la experiencia previa del equipo de desarrollo con los servicios del proveedor. Si el equipo ya tiene experiencia previa con el proveedor, puede trabajar de manera más eficiente y evitar posibles obstáculos durante el desarrollo. Sin embargo, se le asigna un peso moderado de 2/9, ya que otros factores como la facilidad de uso y la documentación también son importantes y pueden compensar la falta de experiencia previa.
- **Herramientas específicas** Este criterio considera la disponibilidad de herramientas específicas proporcionadas por el proveedor que pueden ser útiles

para el desarrollo del videojuego. La presencia de herramientas específicas puede facilitar ciertas tareas o agregar funcionalidades adicionales al proyecto. Se le asigna un peso moderado de 2/9, ya que estas herramientas pueden ser beneficiosas pero no son imprescindibles para la elección del proveedor.

Nombre	Facilidad de uso (Peso: 3)	Documentación (Peso: 2)	Conocimiento previo (Peso: 2)	Herramientas específicas (Peso 2)	Total (/90)
<b>Unity Services</b>	<b>8</b>	<b>9</b>	<b>1</b>	<b>9</b>	<b>62</b>
AWS GameLift	3	7	7	6	49
Google Cloud	3	7	1	4	33

Como se puede observar, el proveedor Unity Services obtuvo la puntuación más alta. En consecuencia, se determinó que es el proveedor más adecuado para este proyecto, y por lo tanto, ha sido elegido para el desarrollo.

Entre los servicios más populares de Unity Services, se encuentran:

- Servicios de análisis que ayudan en la recopilación de datos sobre el rendimiento y comportamiento de los usuarios en el videojuego y tomar decisiones informadas en el desarrollo, como Unity Analytics,
- Servicios de anuncios que ayudan en la integración de anuncios, generando ingresos mediante publicidad.
- Servicios de red que permiten la creación de plataformas multiusuario.

Para el desarrollo del proyecto se han utilizado diversos servicios de red de Unity , entre los que se encuentran los servicios de Relay, Lobby, Vivox y Autenticación.

- *Lobby* es un servicio de Unity Gaming Services el cual permite a los jugadores conectarse a diferentes partidas. Este servicio permite a los jugadores crear *lobbies* públicos que otros pueden ver y unirse. También permite crear *lobbies* privados los cuáles requieren de un código o invitación.
- *Player Authentication* es un servicio de Unity Gaming Services el cual permite a los jugadores iniciar sesión anónimamente o mediante distintos proveedores. Esto es útil para conocer la identidad de los usuarios y poder proveer de seguridad y consistencia a los desarrolladores y usuarios.
- *Relay* es un servicio que permite conectar diferentes usuarios sin la necesidad de un servidor dedicado. Relay permite comunicaciones *P2P* y *UDP* fáciles y seguras. Este servicio facilita la conexión de los usuarios sin la necesidad de controlar las direcciones IP y puertos de los diferentes usuarios. En lugar de utilizar un servidor dedicado, el servicio de Relay proporciona conectividad a través de un servidor Relay universal que actúa como un proxy. Basado en la tecnología de red WebRTC, el servicio de Relay de Unity garantiza una comunicación de baja latencia, siendo fundamental para aplicaciones en tiempo real como juegos multijugador.
- *Vivox* este servicio permite a los diferentes jugadores comunicarse entre ellos mediante el uso de un chat de voz o texto dentro del juego.

#### 5.1.4. Otras herramientas

##### 5.1.4.1 Git

Git [15] es la herramienta para el control de versiones más utilizada a nivel mundial. Ha sido utilizado para controlar las versiones de la aplicación.

##### 5.1.4.2 Trello

Trello [16] es una aplicación para web y dispositivos móviles orientada a la gestión de proyectos. Esta herramienta facilita el desarrollo de diferentes tipos de productos mediante el uso de tableros organizativos donde se pueden organizar los diferentes eventos de un proyecto. Se ha usado Trello en el proyecto junto a la metodología ágil Scrum para gestionar el progreso del desarrollo del proyecto.

##### 5.1.4.3 Visual Studio

Visual Studio Code [17] es un editor de código de Microsoft. Este editor ha sido utilizado para el desarrollo en C# debido a su fácil integración con Unity, su importación automática de archivos, su facilidad para hacer *debug* y conocimiento previo.

##### 5.1.4.4 Github

Github [18] es un servicio basado en la nube el cual permite el alojamiento de repositorios Git de código.

##### 5.1.4.5 Blender

Aplicación informática para la creación de modelos 3D, iluminación renderizado y animación. Ha sido utilizada para la creación de los modelos y animaciones del videojuego.

##### 5.1.4.6 Photoshop

Aplicación para la creación, edición y retoque de fotografías e imágenes. Ha sido utilizada para la edición de algunos de los sprites del videojuego.

## 5.2. Arquitectura de red

En esta sección se explica el funcionamiento de red del videojuego y cómo funciona la comunicación entre los distintos usuarios, así como las principales tecnologías y técnicas utilizadas en el proyecto.

Este proyecto ha seguido un modelo de arquitectura de cliente host (ver sección 2.4.4).

### 5.2.0.1 Netcode for GameObjects

*Netcode for GameObjects* permite la comunicación y sincronización de objetos entre distintos dispositivos en tiempo real. Para ello se utilizan "Network Variables", estas variables son compartidas entre todos los usuarios conectados en la misma sala y son actualizadas en tiempo real cuando cambian de valor. Estas variables han sido usadas

en el proyecto para la sincronización de la información que debe ser compartida, como por ejemplo la vida de los personajes o la puntuación de los jugadores. Todas estas variables han sido declaradas con permiso de escritura para el servidor y permiso de lectura para todos, manteniendo así la autoridad del servidor. Esto ha permitido que estas variables estén actualizadas con los valores correctos, asegurando la integridad de los datos y evitando conflictos entre usuarios añadiendo una capa de seguridad extra.

Por otra lado, existen las *Remote Procedure Call (RPC)*, estas llamadas son un mecanismo que permite invocar una función de un objeto de red y ejecutarlo en todos o alguno de los clientes conectados. Esto permite a los diferentes clientes comunicarse y sincronizar acciones mediante el envío de mensajes que llaman a funciones específicas.

Existen dos tipos de llamadas a procedimientos remotos:

- **ServerRpc:** esta llamada puede ser llamada por cualquier cliente para invocar a una función en el servidor. Estas llamadas han sido utilizadas para enviar información al servidor. Son especialmente útiles para validar acciones en el servidor antes de aplicar cambios en el estado del juego. También sirven para evitar usos indebidos, como por ejemplo, validar si un usuario ha intentado realizar una acción que no debería ser posible, como disparar cuando en verdad todavía está recargando.
- **ClientRpc:** esta llamada puede ser llamada desde el servidor para ejecutarse en todos o alguno de los clientes. Estas llamadas son muy útiles para sincronizar acciones que ocurren en diferentes clientes. Han sido utilizadas para enviar información a los clientes y sincronizar acciones. Estas funciones se han utilizado una gran cantidad de veces, como por ejemplo para reproducir un sonido de disparo, actualizar el texto de las puntuaciones o desactivar y activar un jugador cuando este muere o respawnea.

Mientras que ClientRPC se utiliza para sincronizar acciones en los clientes, ServerRPC se utiliza para realizar validaciones y tomar decisiones importantes en el servidor.

### 5.2.0.2 Comunicación entre cliente y servidor

La comunicación entre cliente y servidor se realiza a 60 Hz,  $\frac{1}{60 \times 1000} = 16,66\text{ms}$ . Esto quiere decir que cada 16,6 ms los clientes envían información sobre el jugador al servidor si han generado algún *input*. A su vez, el servidor también ejecuta la lógica del juego a una velocidad máxima de 16,6 ms, después de esto empaqueta la información de los jugadores y la envía a los clientes. Esto quiere decir, que las acciones en la partida ocurren 60 veces por segundo

Cuando el servidor recibe un *input* de un cliente conectado, en lugar de procesarlo y calcular la lógica correspondiente al instante, el servidor almacena dicho *input* en un buffer circular, creando así una especie de cola de espera para procesarlo más tarde. Esta estrategia se implementa con el objetivo de mitigar los posibles efectos negativos del jitter que podría experimentar la conexión.

El buffer garantiza un proceso ordenado: en cada "tick" de la simulación, el servidor se encarga de procesar únicamente un *input*. Esto significa que, aunque se reciban varios *inputs* consecutivos de un mismo cliente antes de ser ejecutados, estos serán almacenados en el buffer y se retiran en orden uno a uno a medida que el estado del

juego progresa. Este método contribuye a mantener la coherencia en la simulación del juego, incluso en situaciones donde la información llega de manera simultánea desde el cliente.

### 5.2.0.3 Estructuras de datos de red

El envío de datos por parte del cliente al servidor se realiza mediante una llamada *ServerRpc*, estos datos se componen principalmente por los *inputs* del jugador y son usados para el cálculo de la posición en el servidor. En el caso de que el jugador no realice ninguna acción, ningún paquete es enviado para ahorrar ancho de banda. Estos *inputs* simplemente contienen la dirección en la que el jugador desea moverse. En el paquete también se envía una marca de tiempo y un *tick*. Este "tick" funciona como un marcador único asociado al *input* enviado desde el cliente hacia el servidor, elemento necesario ya que la información es guardada en un buffer circular. Su función principal es servir como un identificador que facilita la comparación entre la posición del jugador local, y las posiciones provenientes del servidor. Este proceso de comparación tiene como objetivo determinar si se requiere llevar a cabo una reconciliación para ajustar la posición local del jugador. La marca de tiempo es utilizada para calcular la latencia de los jugadores y determinar si una extrapolación es necesaria. La estructura de este paquete se puede ver en el Anexo 1

El paquete de datos que envía el servidor a los clientes es enviado mediante una llamada *ClientRpc*. Este paquete tiene una estructura similar, en la que también se envían tanto el tick del servidor al que pertenece el paquete, como la marca de tiempo. Además el *input* es vuelto a ser enviado, esto es debido a que es necesario en caso de tener que calcular una extrapolación o una reconciliación, funcionando como si fuera una rotación. Como parámetro extra se envía la posición del jugador para el cálculo de la reconciliación.

### 5.2.0.4 NetworkTransform

Se ha decidido delegar la actualización de la posición y rotación de los objetos al componente *NetworkTransform* de la librería *NGO*. Este componente se encarga de actualizar de manera automática en los clientes el "transform", osea la posición, rotación y escala de los *GameObjects*. Para la implementación en el proyecto, estos componentes han sido optimizados, enviando solo las posiciones en el eje "X" e "Y" y la rotación en el eje "Z", minimizando la cantidad de datos a enviar, lo que mejora la eficiencia en la transmisión y reduce la latencia. También se han aumentado los umbrales de posición y rotación, reduciendo la frecuencia de las actualizaciones de sincronización, al sincronizar sólo los cambios por encima o igual a los valores umbral. Por último se ha decidido usar el formato en coma flotante de media precisión o *half float precision* aumentando la compresión de la rotación en el eje "Z", usando solo 8 bytes en vez de 16 para cada actualización de este valor. Por último se ha incorporado la interpolación para mejorar la suavidad entre las actualizaciones de las transformaciones en las instancias no autoritativas, este apartado se explica con más detalle en 5.2.0.5. Estas mejoras han logrado minimizar el ancho de banda utilizado y reducir la congestión en la red, sin comprometer la precisión y la calidad de la sincronización del juego

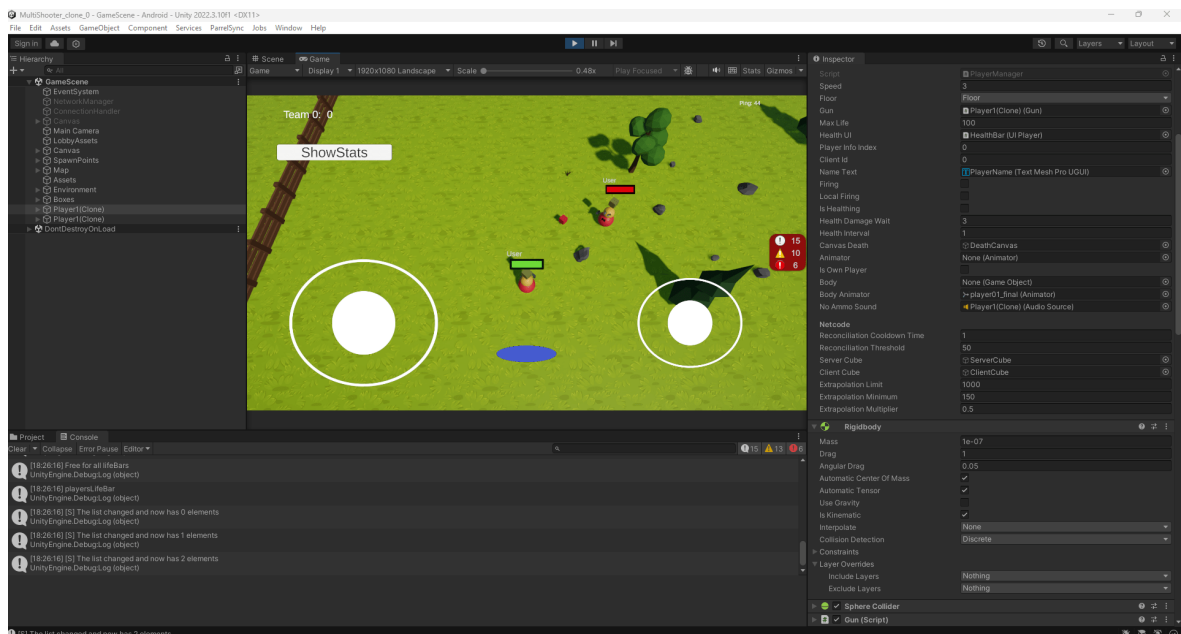
### 5.2.0.5 Implementación de técnicas de red

Se ha empleado la técnica de State Synchronization para el desarrollo de la aplicación, delegando la actualización de las posiciones de los jugadores en un componente de la biblioteca de red *NGO*, denominado *NetworkTransform*.

Dado que el movimiento de los jugadores no usa físicas, se implementó la técnica de Deterministic Lockstep, actualizando en los clientes las posiciones de los jugadores solo mediante los *input* recibidos y removiendo los *NetworkTransform*. Sin embargo, surgieron pequeños problemas como la detección de colisiones, la cual no era del todo precisa y las posiciones. Estas últimas no estaban del todo sincronizadas debido a pequeñas variaciones en el cálculo de la posición que se iban acumulando, resultando en diferencias notables entre los clientes y el servidor. En la Ilustración 20, se puede apreciar el desfase de la posición entre la posición del cliente y la posición del servidor, representada esta última por el cubo rojo.

Debido a este problema, se decidió seguir con el desarrollo usando State Synchronization, confiando en los componentes de *NGO*.

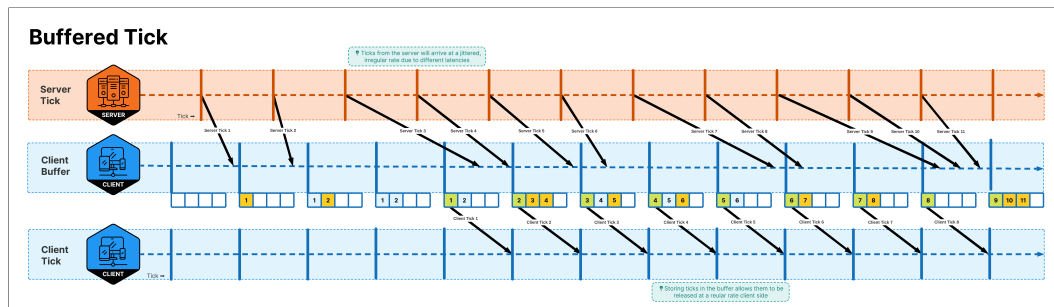
Figura 20: Desfase de posición



Las técnicas de compensación de la latencia empleadas han sido las siguientes:

- Interpolación** Se ha implementado la técnica de interpolación gestionada por el componente de *NetworkTransform*. Esta interpolación se encarga de almacenar en un buffer las actualizaciones de estado entrantes y actualiza estos estados de forma gradual para mostrar al jugador no autoritativo las transiciones de una forma suave (Ver imagen 21). Cuando esta opción está desactivada, los cambios en la transformación se aplican inmediatamente, lo que puede dar lugar a saltos aparentes en las actualizaciones de estado si la latencia es alta.
- Predicción del cliente** Para la implementación de predicción del cliente, además de enviar la información al servidor, los *inputs* del jugador son también ejecutados

Figura 21: Buffer de interpolación. Imagen obtenida de unity3d.com



inmediatamente en su máquina local. De esta forma, las acciones del jugador son inmediatamente representadas en pantalla, asumiendo que estos *inputs* serán aceptados y procesados de igual forma por el servidor. Si esto no fuera así, las posiciones se corregirán mediante *reconciliación*. De esta forma los jugadores no tienen que esperar a que los *inputs* lleguen al servidor y este los procese y mande el estado de vuelta, mostrando la respuesta de los *inputs* del usuario de una forma inmediata y reduciendo en gran medida la percepción de latencia.

Al usar los componentes de *NetworkTransform* se han tenido que sobrescribir algunos de sus métodos, cambiando la titularidad del objeto jugador a modo "owner-client" o cliente propietario. Sin esto, los jugadores no tendrían permiso para cambiar la posición de sus personajes y solo el servidor podría hacerlo, por lo que la predicción no podría llevarse a cabo. Esto conlleva a que el servidor no sea autoritativo, ya que los clientes podrían cambiar de posición de manera instantánea y el servidor aceptaría estos cambios. Por eso, además de la reconciliación, se ha desarrollado un sencillo sistema anti-trampas, en la que el servidor examina los paquetes de los *NetworkTransform* de los clientes y calcula si el cambio en la última posición recibida es posible o no. Si el cambio no es posible, el servidor cambia de forma momentánea la autoridad del personaje al servidor para corregir su estado y posteriormente devuelve la autoridad al jugador.

- **Extrapolación** Esta técnica consiste en la predicción del movimiento de los personajes.

Debido a la naturaleza del juego, en el que los jugadores pueden cambiar de dirección y rotación de manera instantánea, la predicción del movimiento es muy difícil. Por eso, la extrapolación en este tipo de juegos tiene una alta probabilidad de fallar y tener que ser corregida mediante reconciliación, debido a esto se ha establecido un umbral de latencia para que solo se ejecute en el caso de que el cliente tenga una latencia mayor a 150ms. Si un jugador tiene una latencia alta, se ejecutará la extrapolación con la ecuación del movimiento rectilíneo uniforme ajustada según la latencia, un ejemplo de este código se puede ver en el Anexo 1. En la mayoría de las actualizaciones se prevé que el jugador seguirá moviéndose recto, si esto es así, la predicción acertará y las posiciones estarán mejor sincronizadas, compensando un poco la latencia. Esto suele funcionar, ya que existen muchas más actualizaciones que cambios en la dirección. En caso de que el jugador no tenga una latencia alta, se ha decidido omitir esta predicción ya que el desfase de las actualizaciones no será muy alto y si la extrapolación está activada, cuando haya un fallo en la predicción, se muestran saltos visuales

en la representación.

- **Reconciliación** Esta técnica consiste en la corrección local de las estimaciones ejecutadas por técnicas como la interpolación o la extrapolación u otros factores .

Para su implementación se evalúa si el paquete recibido es nuevo y si no se está ejecutando reconciliación o extrapolación. De ser así, se compara las posiciones del último paquete recibido con la posición actual del cliente. Si la diferencia de posiciones es mayor a un umbral, el desajuste de posiciones se considera importante y se actualiza la posición del cliente con la posición del servidor.

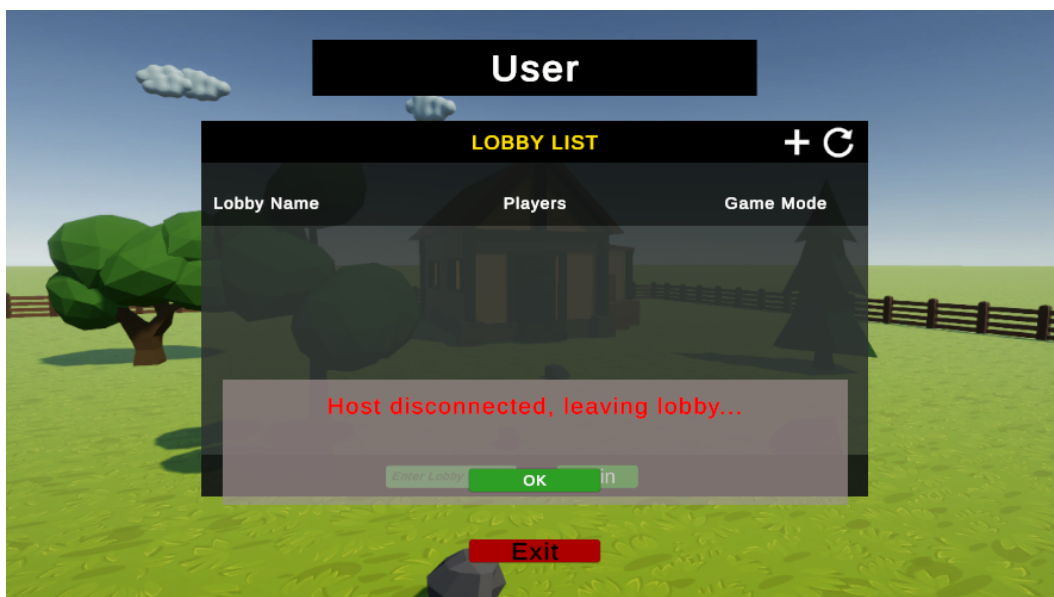
### 5.3. Manejo y propagación de errores

En cualquier aplicación de red, la gestión de la propagación de errores es un aspecto esencial, ya que los fallos en la conexión pueden ocurrir con bastante frecuencia y de forma inesperada. Estos errores pueden propiciar más errores al jugador o incluso a otros jugadores si no son gestionados correctamente.

Los fallos en la red, ya sea por problemas de conectividad, servidores o hardware, son comunes en juegos en línea, generando comportamientos impredecibles y pérdida de datos de sesión. Es esencial implementar un sistema eficaz de manejo de errores que informe a los usuarios sobre la naturaleza de los fallos y proporcione orientación para recuperar la conexión. La gestión de errores es crucial para mantener una experiencia de juego estable y satisfactoria en entornos multijugador.

Un ejemplo de como se manejan los errores en *Shooter Stars* es cuando un usuario *host* cierra la aplicación, pierde la conexión, o se desconecta de forma repentina. En este caso, los demás clientes son redirigidos a la pantalla de lobbies y se le muestra un mensaje informando que el *host* se ha desconectado, como se puede ver en la imagen 22. Con este mensaje de error, los usuarios son informados de que el fallo no ha sido debido a su dispositivo ni su conexión, sino ha sido un fallo con el usuario *host*.

Figura 22: Host Disconnected, mensaje de error



Otro caso, como se puede ver en la imagen 23, es cuando un usuario pierde la conexión con alguno de los servicios básicos de Unity y por lo tanto con la partida, ya sea por una pérdida de conexión suya, o porque algún servicio de Unity no esté disponible. En este tipo de caso se muestra al usuario el error recibido.

Figura 23: Error de conexión, mensaje de error



## 5.4. Integración Unity Services

Para la implementación de los servicios de Unity 5.1.3, primero se tuvo que crear una cuenta, activar los servicios requeridos y configurarlos en el proyecto. En la figura 24, se puede ver el *dashboard* del servicio de Lobby.

### 5.4.0.1 Relay

En el caso de Shooter Stars, Relay 5.1.3 se utiliza para establecer la comunicación en tiempo real entre los usuarios y el usuario host, así como para permitir el envío de datos, como el estado de la partida, entre estos.

El servicio de relay se inicia al crear un *lobby* por el anfitrión de la partida. Para la unión a una sala, se ha obtenido la información sobre las salas abiertas existentes junto a sus códigos de relay mediante los servicios de Lobby. Cuando un jugador elige la sala a unirse se envía una petición para unirse al servicio de relay con dicho código.

En la comunicación mediante relay se ha usado DTLs (Datagram Transport Layer Security) sobre UDP para mejorar la seguridad, evitando las escuchas no deseadas, ataques de intermediario, o modificación de mensajes.

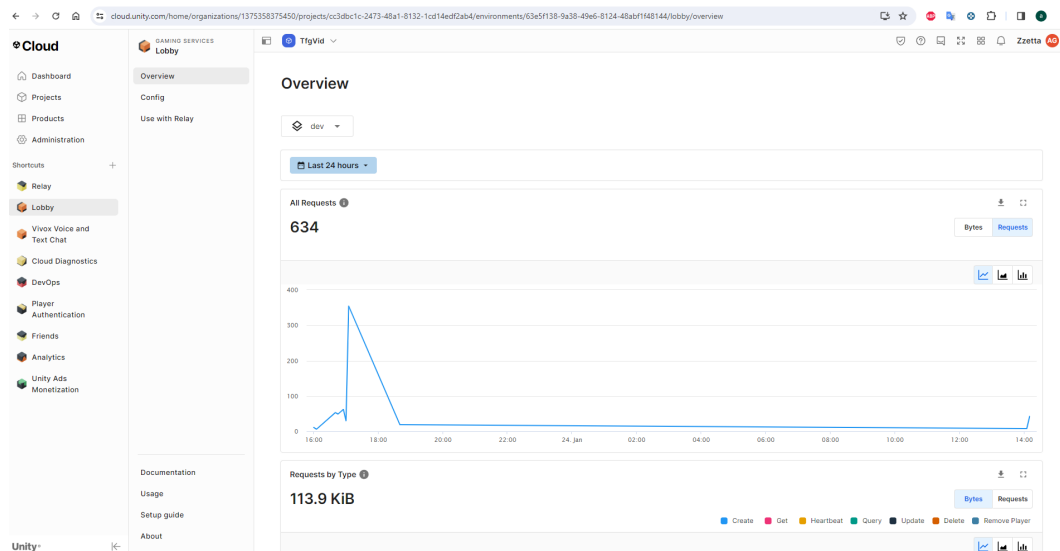
### 5.4.0.2 Lobby

En el caso de Shooter Stars, el servicio de Lobby 5.1.3 se utiliza para permitir a los usuarios unirse a una sala y jugar con otros usuarios.

Inicialmente, se usó el sistema de envío de mensajes de Lobby, utilizando un enfoque centralizado en el código del relay para facilitar la comunicación entre los jugadores. Sin embargo, se tomó la decisión de trasladar la inicialización de *NGO* y Relay tanto en el host como en los clientes a esta pantalla de *lobby*. Esta modificación permitió lograr un envío instantáneo de información entre los participantes y una integración más sencilla con los demás componentes del juego.

Durante la partida, con el propósito de mantener la conexión del *lobby* se ha creado una función encargada de mandar *heartbeats* al servicio de Lobby para que dicho *lobby* no quede inactivo y, por lo tanto sea eliminado. Este servicio también ha sido el encargado de actualizar la lista de *lobbies* existentes, así como la creación de estos o la unión a un *lobby* abierto o por código. El usuario *host* del *lobby*, el creador de la sala, puede elegir parámetros de este, como si la sala es abierta o privada, el número máximo de usuarios de la sala o si desea eliminar a un usuario de esta.

Figura 24: Unity Dashboard - Lobby



### 5.4.0.3 Vivox

Para la integración del chat de voz con el servicio de *Vivox* 5.1.3, se han creado varios controladores que manejaban el chat de manera independiente al resto del juego. Uno de estos controladores, se encargaba de establecer la conexión con este servicio y de finalizar las conexiones. El segundo controlador, se encargaba de la gestión de las salas de audio, estableciendo y finalizando la conexión con el canal de audio según la sala o *lobby* correspondiente. El último componente se encargaba de configurar y manejar los componentes de audio de los jugadores, como las barras de sonido y las opciones de silenciar al usuario o a otros participantes.

## 5.5. Adapatación multiplataforma

El desarrollo del videojuego se decidió enfocar en PC, dado que proporcionaba un entorno más cómodo para el proceso de desarrollo en un entorno de escritorio. Posteriormente, para ofrecer una experiencia de juego más versátil, se realizó una

adaptación del juego para dispositivos móviles. Esta adaptación implicó ajustes significativos en la interfaz y los controles, como el cambio de la tecla de menú de tabulador a un botón en pantalla, así como la implementación de dos *joysticks* para el apuntado y disparo en lugar del uso de las teclas AWSD y el ratón. Se usó compilación condicional para la adaptación entre ambas plataformas, permitiendo que el código se ajustara al compilar según la plataforma de implementación, garantizando la funcionalidad de los ejecutables tanto en PC como en dispositivos móviles.

## 5.6. Retos técnicos

A continuación se describirán las partes de la aplicación que han supuesto mayor complejidad en el desarrollo.

En el desarrollo de videojuegos es usual que los desarrolladores encuentren dificultades que tengan que ser superadas. Estas dificultades y los posibles puntos de fallo aumentan en los juegos multijugador debido a la necesidad de sincronización, comunicación y medidas de seguridad. Además, se incrementan aún más en videojuegos multiplataforma, ya que es necesario garantizar la compatibilidad de todos los elementos entre diferentes dispositivos.

Un ejemplo de estas dificultades se dio a la hora de adaptar el videojuego para dispositivos móviles, para lo que se usaron *joysticks* con los cuales obteníamos vectores normalizados. Sin embargo, el movimiento en jugadores en PC involucraba la posición en 3D, por lo que existían dos maneras distintas de calcular el movimiento. No fue hasta la validación que se vio que los movimientos no eran igual de rápidos debido a la forma de cálculo, por lo que los movimientos en dispositivos móviles eran más rápidos que en pc.

Aunque el desarrollo del videojuego no ha requerido de funciones de renderización o de simulación de físicas complejas, y la trayectoria de las armas en general no ha sido difícil de implementar, algunas de estas cómo la del lanzagranadas, sí que ha necesitado de una simulación de físicas propia a la hora de trazar la proyección al apuntar, como se puede ver en el Anexo 1

El proceso de desarrollo mediante los servicios de Unity Services, sobre todo del servicio de Lobby fue un desarrollo que involucró un largo periodo de aprendizaje y mejora que requirió mucha dedicación. La sincronización de los clientes y el funcionamiento interno del *lobby* fue un proceso largo y tedioso que necesitó mucho aprendizaje e intentos a base de prueba y error. Además, la implementación de un *lobby*, necesita de muchas casuísticas que deben ser manejadas, como:

- Qué pasa cuando un cliente deja la sala
- Qué pasa si el *host* deja la sala.
- Cómo volvemos a la sala y que valores debemos resetear cuando se acaba la partida.
- Cómo esperar a que los clientes estén sincronizados para poder empezar la partida.

La implementación de las técnicas de compensación de latencia y su integración con *NGO*, ha requerido un entendimiento profundo de ambas entidades. Esta implementación también ha necesitado de varias refactorizaciones y cambios en la estructura de la lógica de los jugadores.

La sincronización de variables entre distintos jugadores ha resultado ser una tarea tediosa y compleja, debido a la necesidad de sincronizar meticulosamente todos los elementos del juego y establecer valores iniciales de manera precisa. Algunos de estos elementos han sido la sincronización de jugadores, equipos, *spawns*, posiciones, animaciones, estados y estadísticas.

La depuración o *debug* también ha sido un desafío significativo, y en muchas ocasiones complejo y frustrante, esto ha sido debido a la naturaleza de la asincronía y concurrencia en las operaciones típica de videojuegos multijugador. La complejidad de rastrear eventos que ocurren en paralelo entre múltiples jugadores y la dificultad para reproducir consistentemente ciertas condiciones específicas han hecho del proceso de *debugging* un desafío constante. La distribución de datos entre clientes y *host*, junto con la necesidad de analizar el flujo de datos a través de la red, y las diferencias entre plataformas han añadido una capa adicional de complejidad. Por último, la complejidad estructural y la interdependencia de los módulos del proyecto han hecho que seguir la ejecución y entender el flujo de control haya sido particularmente difícil durante el proceso de *debugging*.

---

## 6. Validación

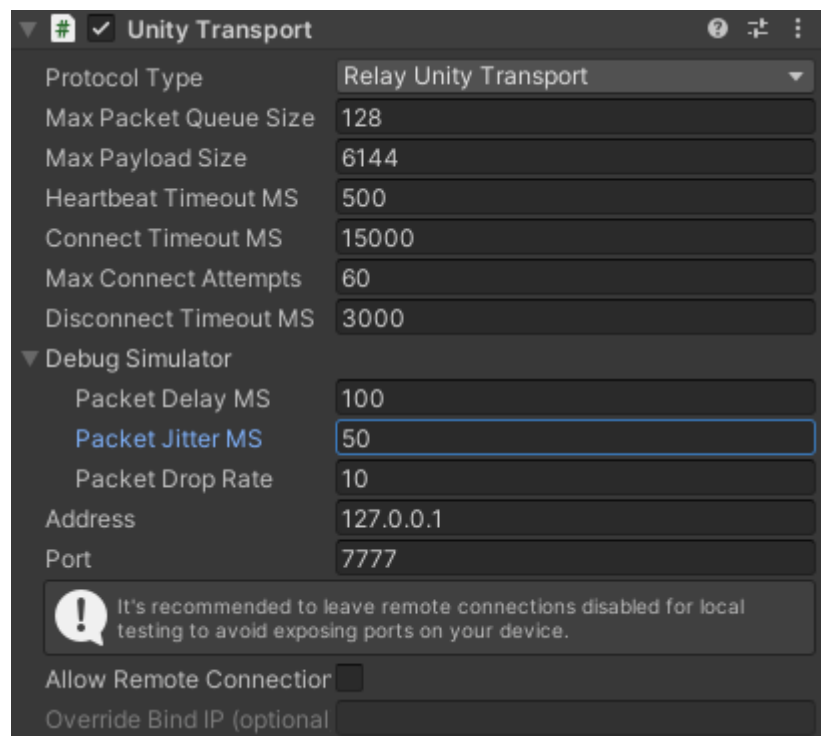
Durante el proceso de desarrollo del videojuego, se ha llevado a cabo una fase de validación para garantizar la calidad y el correcto funcionamiento de las implementaciones. La validación se ha realizado mediante pruebas manuales, así como pruebas con usuarios reales, permitiendo la identificación de posibles problemas.

### 6.1. Experimentos y pruebas

Durante el desarrollo, se han realizado pruebas de las implementaciones para asegurarse de que cumplen con los objetivos establecidos. Incluyendo muchos aspectos como la jugabilidad, la interactividad, el rendimiento y la estabilidad en diversas plataformas.

Las pruebas relacionadas con las funciones de red y estabilidad se han llevado a cabo mediante el componente de *Netcode "Unity Transport"*, el cual posee de un simulador de estabilidad de red, este componente se puede observar en la figura 25. Con este componente se han realizado pruebas de retraso de paquetes, fluctuación de paquetes y tasa de pérdida de paquetes. Debido a estas pruebas se han reajustado y mejorado las técnicas de red, también ha permitido identificar y tomar decisiones en el diseño, como el descarte de la técnica de *Deterministick Lockstep* en el videojuego 5.2.0.5.

Figura 25: Unity Transform



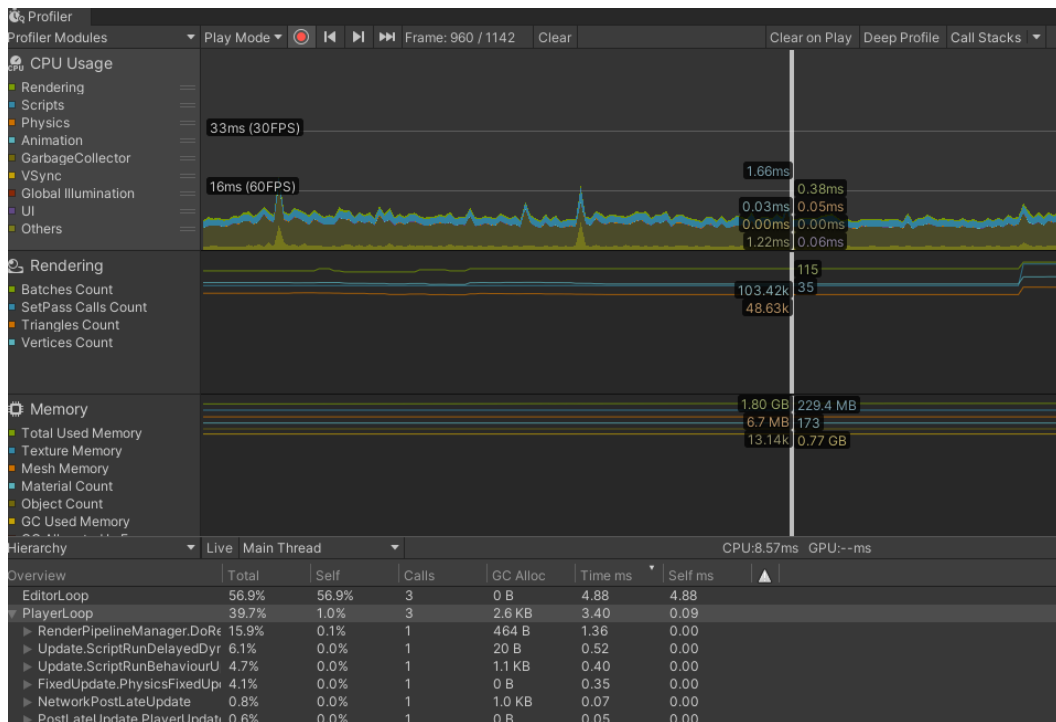
Como podemos observar en las figuras 26, 27, 28 y 29 realizadas mediante el *pro ler* de Unity (tomadas usando el mismo entorno), gracias a las pruebas tanto propias como

con usuarios reales y a optimizaciones continuas se ha conseguido una mejora en el rendimiento.

En la figura 26, obtenida en una versión temprana del desarrollo, podemos ver que el ciclo promedio tarda 8.57 ms, lo que se traduce en aproximadamente 116.73 FPS. En comparación, la figura 27, obtenida en la versión actual, muestra una disminución del tiempo del ciclo promedio a 6.38 ms, lo que equivale a aproximadamente 156.73 FPS. Esto representa una mejora significativa del rendimiento del juego, con un incremento aproximado del 34.37% en los FPS obtenidos. Esta mejora ha sido conseguida principalmente mediante la reducción de la complejidad en los modelos 3D y en la mejora de la lógica del juego, optimizando *scripts* y cálculos innecesarios para reducir la carga de la CPU.

De igual manera, como se puede observar en las figuras 28 y 29 se ha conseguido una mejora sustancial en cuanto a términos de red. Esta mejora se ha conseguido reduciendo la cantidad de información enviada de los objetos de red, como se explica en la sección 5.2.0.4. También se ha reducido la frecuencia en las llamadas RPC, sincronizando a los jugadores solamente cuando ocurre un cambio relevante.

Figura 26: Rendimiento en versión temprana del desarrollo



La validación se ha ampliado a través de pruebas con usuarios reales, quienes han experimentado el juego mediante *builds* para las plataformas específicas. La participación de estos usuarios ha permitido obtener valioso *feedback* sobre la jugabilidad, la interfaz de usuario y otros aspectos relevantes, como *bugs* o errores. Se han tenido en cuenta las opiniones y comentarios de los usuarios para realizar mejoras y optimizaciones continuas.

La validación constante, combinada con la participación activa de usuarios reales, ha sido fundamental para perfeccionar el videojuego y garantizar que cumple con las expectativas de los jugadores.

Figura 27: Rendimiento en última versión

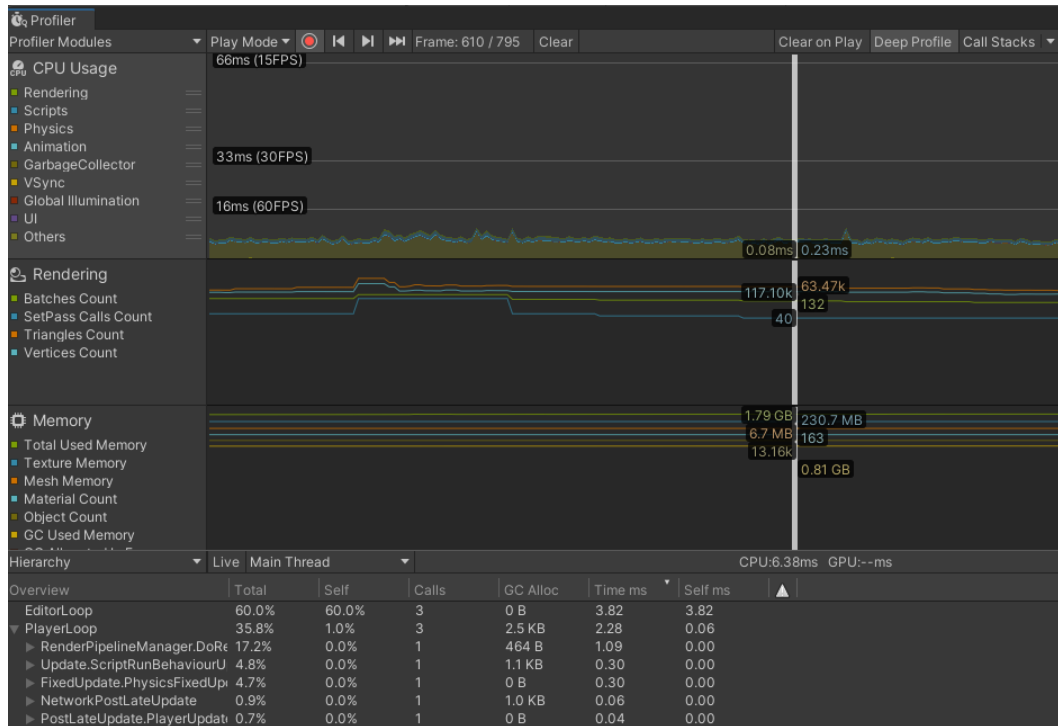


Figura 28: Ancho de banda en versión temprana del desarrollo

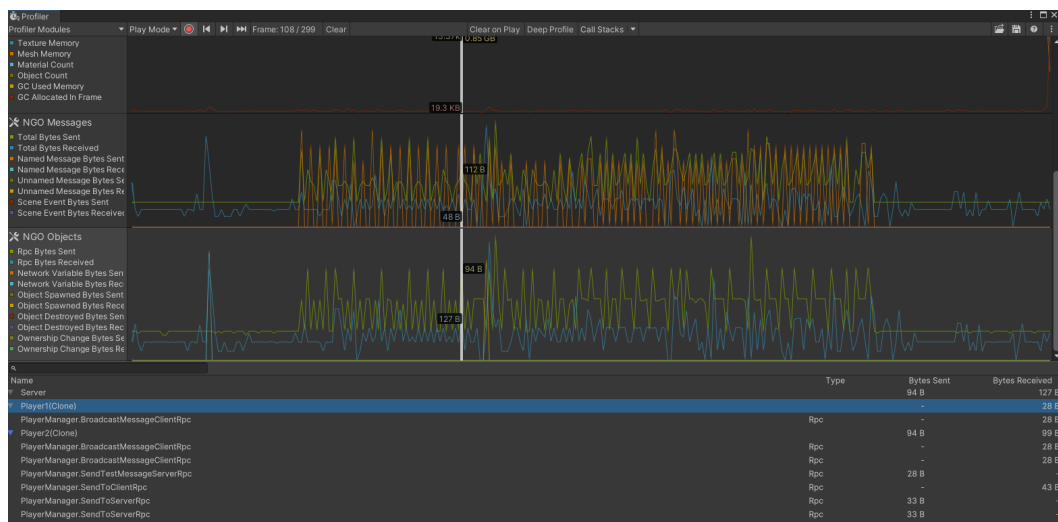
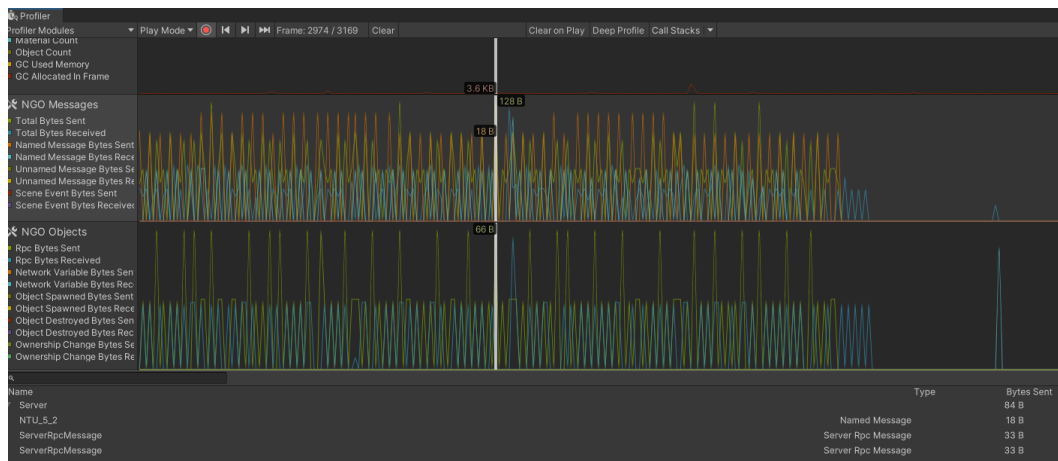


Figura 29: Ancho de banda en última versión



---

## 7. Conclusiones y trabajo futuro

### 7.1. Conclusiones

Este proyecto me ha resultado tanto a nivel personal, como sobre todo, a nivel profesional, un gran reto, al que he tenido que dedicar mucho tiempo y dedicación. Sin embargo, gracias a este proyecto, he aprendido mucho sobre diferentes tecnologías y metodologías de desarrollo, lo cual ha ampliado mis conocimientos y habilidades en el ámbito profesional. Adquiriendo una experiencia muy valiosa e interesante en el desarrollo de videojuegos multijugador y metodologías ágiles como SCRUM.

Además, he tenido la oportunidad de enfrentarme a diversos desafíos técnicos, como la integración de diferentes servicios de Unity , y la implementación de funcionalidades complejas. Estos desafíos me han permitido desarrollar mi capacidad de resolución de problemas y mejorar mis habilidades de programación.

#### 7.1.1. Logros alcanzados

Después de varios meses de desarrollo, con gran satisfacción, puedo concluir que el proyecto ha alcanzado el nivel de madurez necesario para afirmar que cumple con los objetivos establecidos.

En este apartado se va a evaluar el grado de cumplimiento de los objetivos propuestos 1.2, los cuales han sido todos superados.

- El primer objetivo, "Investigar la evolución de los videojuegos multijugador así como las posibilidades y dificultades que representan", ha sido superado. Se ha llevado a cabo una investigación sobre la evolución de los videojuegos multijugador 2, así como los diferentes problemas existentes a la hora del desarrollo de este tipo de juegos 2.2.
- El segundo objetivo, "Investigar las diferentes técnicas y procesos para hacer frente a los retos del desarrollo de los videojuegos multijugador", ha sido superado. Después de la investigación sobre los desafíos en el desarrollo de los videojuegos multijugador, se llevó a cabo una investigación sobre las diferentes técnicas existentes para solventar estos problemas 2.3. También se analizó en profundidad las posibilidades y oportunidades existentes a la hora del desarrollo de un videojuego multijugador 5.1.
- El tercer objetivo, "Desarrollar un prototipo de videojuego multijugador aplicando el conocimiento adquirido e implementando técnicas de compensación de latencia", ha sido superado. Tras la investigación previa realizada, se ha llevado a cabo el desarrollo del prototipo, este es jugable por varios jugadores simultáneamente, permitiendo la elección de varios personajes a los jugadores así como la elección de equipo, si el modo de juego lo permite. El *host* de la partida también puede configurar elementos extra, como el número de jugadores, la cantidad de *kills* para terminar la partida o el modo de juego.

También se ha conseguido la implementación de varias técnicas de compensación de latencia 5.2.0.5, capaces de hacer frente a los problemas de red y minimizar el impacto que la latencia puede tener en la experiencia de los jugadores. Entre estas técnicas implementadas, se pueden destacar las técnicas de interpolación, extrapolación, predicción del cliente y reconciliación. También se consiguió implementar seguridad en el videojuego, utilizando un sistema de autoridad del servidor para muchos de los objetos y desarrollando un sencillo sistema anti-trampas para los personajes, los cuales son controlados por los jugadores para la implementación de la predicción.

- El cuarto objetivo "Implementar la capacidad multiplataforma (móvil y PC) para el videojuego", ha sido superado. Se ha conseguido la adaptación del videojuego tanto para PC como para dispositivos móviles Android 5.5.
- Por último, el quinto objetivo "Validar el proyecto mediante diferentes pruebas para asegurar el correcto funcionamiento del videojuego", también ha sido superado. Con el objetivo de asegurar el correcto funcionamiento del videojuego, se han llevado a cabo numerosas pruebas, entre las cuales podemos destacar, pruebas manuales durante el desarrollo, pruebas de red, pruebas de rendimiento y pruebas con usuarios reales 6.

## 7.2. Trabajos futuros

La aplicación desarrollada es un proyecto muy ambicioso que necesitaría de abundantes meses o incluso años de desarrollo para ser completada. Este proyecto solamente ha entablado unas bases sólidas desde las cuales se puede seguir desarrollando la aplicación, sirviendo a su vez, como prueba de concepto de todo lo que podría llegar a ser.

Ya que la finalidad de este proyecto es la de abarcar la funcionalidad de un videojuego multijugador, la personalización y posibles características que se pueden incluir son muy numerosas. Algunas de las características más relevantes que podrían mejorar la aplicación son las siguientes:

- **Filtrado de partidas** El filtrado de partidas sería un elemento fundamental para el uso de la aplicación al elegir el tipo de partida que un jugador desea jugar. Esta característica estaba planteada para llevarse a cabo y ya almacena diferentes filtros, pero no se ha llegado a finalizar debido a falta de tiempo.
- **Autenticación con proveedores.** Google, Iphone, Steam...
- **Sistema de amigos.** Aunque puedes jugar y hablar con tus amigos uniéndote a la misma sala por medio de un código, un sistema de amigos el cual informe al jugador si un amigo está jugando, sería más cómodo para los usuarios. Una posible implementación podría implicar el uso del servicio de Unity llamado *Friends*.
- **Introducción de nuevas habilidades, campeones y mapas.** Como en el juego de Brawl Stars, introducir una habilidad extra por personaje e incluir más personajes y mapas, podría añadir más dinamismo en el juego y hacerlo más entretenido.

- **Reconexión.** La introducción a la partida de un mecanismo para que los jugadores puedan reunirse a la partida sería un aspecto importante para mejorar la experiencia de los jugadores.
- **Servidores.** Otra opción sería convertir la arquitectura del juego a una arquitectura de cliente-servidor. Aunque esta decisión pudiera acarrear mayores costes, una arquitectura cliente-servidor mejoraría los servicios, añadiendo más escalabilidad para un número mayor de jugadores y detectando con mayor precisión posibles trampas de los jugadores.
- **Publicación** en tiendas como la App Store, Play Store y Steam.
- **Level Up.** Al modo del *Brawl Stars*, se puede añadir la posibilidad de mejorar las características de los personajes, permitiendo a los jugadores subir de nivel a sus personajes favoritos, mejorando sus atributos y permitiendo desbloquear nuevas armas.
- **Tienda.** Después de añadir autenticación y cambiar la arquitectura a un modelo cliente-servidor, un modelo *freemium*, donde los jugadores pudieran comprar *skins* o nuevos campeones, podría amortizar el gasto en servidores e incluso generar algún ingreso.

## Glosario

- AWS** Amazon Web Services es una colección de servicios en la nube pública que en conjunto forman una plataforma de computación en la nube, ofrecidas a través de Internet por Amazon.com. 17
- AWS GameLift** Amazon GameLift Amazon GameLift implementa y administra servidores de juegos alojados en la nube, en las instalaciones o en implementaciones híbridas. GameLift proporciona una solución de bajo costo y de baja latencia que fluctúa en función de la demanda de los jugadores.. 42
- AWS GameLift FlexMatch** Amazon GameLift FlexMatch es un servicio de emparejamiento personalizable para juegos multijugador. Con FlexMatch, se puede crear un conjunto personalizado de reglas para determinar cómo evaluar y seleccionar a los jugadores compatibles para cada partida.. 42
- Battle Royale** Género de videojuego multijugador en línea en el cual los jugadores han de matarse entre ellos para ganar. Normalmente también combina los elementos de supervivencia y exploración de un juego de supervivencia. III, IV
- cliente-servidor** Cliente-servidor, es una arquitectura de red compuesta por dos componentes, el servidor y los clientes. En esta arquitectura el servidor brinda una serie de servicios los cuales son consumidos a los clientes.. 4
- Easy Anti-Cheat** Aplicación vinculada a numeros videojuegos multijugador usada para la prevención de trampas en el lado del cliente.. 20
- FPS** First Person Shooter es un género de videojuegos basado en disparar a otros jugadores con una vista en primera persona.. 5
- input** Información aportada por un usuario o jugador sobre las acciones que realiza este, esta información es recogida a través de dispositivos como el ratón, teclado, pantalla táctil o mando.. 8
- mods** Extensión de software creado por usuarios que modifica el contenido original, a menudo aportando nuevos contenidos, características o correcciones.. 5
- Netcode for GameObjects** Librería de red de alto nivel construida para Unity que permite abstraer la lógica de red. Se pueden enviar GameObjects y datos del mundo a través de una sesión de red a muchos jugadores a la vez.. 39
- NetworkTransform** Componente de Netcode for GameObjects que sincroniza la posición, rotación y escala de los objetos de juego conectados en red.. 46
- OpenMatch Framework** Open Source creado por Google y Unity para el emparejamiento de partidas.. 41
- P2P** Modelo de ejecución en la nube, en el que el proveedor de los servicios se encarga de la administración y levanta los servicios bajo demanda.. 43

**Remote Procedure Call (RPC)** Es un programa que utiliza un dispositivo para ejecutar código en otra máquina remota sin tener que preocuparse por las comunicaciones entre ambas, de forma que parezca que se ejecuta en local.. 45

**Seguridad por oscuridad** Es un controvertido principio de ingeniería de la seguridad, que intenta utilizar el secreto para garantizar la seguridad. 20

**UDP** User Data Protocol, es un protocolo de transporte por Internet sin conexiones, siendo menos fiable pero más rápido que otros protocolos como TCP. . 43

## Referencias

- [1] F. d'Informàtica de Barcelona, "Historia de los videojuegos." [Online]. Available: <https://www.fib.upc.edu/retro-informatica/historia/videojocs.html#:~:text=Los%20inicios,Alexander%20S.Douglas%20en%201952>
- [2] R. R. Fuentevilla, "Riot direct latam." [Online]. Available: <https://www.riotgames.com/es-419/noticias/riot-direct-y-como-funciona-en-latinoamerica>
- [3] C. Haas and I. Vidal, "Riot direct funcionamiento." [Online]. Available: <https://technology.riotgames.com/news/leveling-networking-multi-game-future#:~:text=Riot%20Direct%20is%20basically%20a,for%20critical%20game%20Drunning%20processes>.
- [4] A. R. David Press, "Riot games at the edge: Launching valorant with aws outposts." [Online]. Available: [https://d1.awsstatic.com/events/reinvent/2021/Riot\\_Games\\_at\\_the\\_edge\\_Launching\\_VALORANT\\_with\\_AWS\\_Outposts\\_GAM302.pdf](https://d1.awsstatic.com/events/reinvent/2021/Riot_Games_at_the_edge_Launching_VALORANT_with_AWS_Outposts_GAM302.pdf)
- [5] M. Reid, "Valorant's foundation is unreal engine." [Online]. Available: <https://www.unrealengine.com/en-US/tech-blog/valorant-s-foundation-is-unreal-engine>
- [6] A. D. . J. Z. E. P. . G. Director, "04: On peeker's advantage and ranked." [Online]. Available: <https://playvalorant.com/en-us/news/game-updates/04-on-peeker-s-advantage-ranked/>
- [7] N. Gentile, "¿cómo funciona counter strike 2 por dentro?" [Online]. Available: <https://www.youtube.com/watch?v=ryRa61jNiso>
- [8] H. Pandey, "Peer-to-peer vs client-server architecture for multiplayer games." [Online]. Available: <https://blog.hathora.dev/peer-to-peer-vs-client-server-architecture/>
- [9] Mirror, "Mirror." [Online]. Available: <https://mirror-networking.com/>
- [10] Photon, "Photon cloud." [Online]. Available: <https://www.photonengine.com/pun>
- [11] Coherence, "Coherence." [Online]. Available: <https://coherence.io/>
- [12] O. Match, "Open match." [Online]. Available: <https://open-match.dev/site/>
- [13] Firebase, "Firebase leaderboard." [Online]. Available: [https://github.com/FirebaseExtended/unity-solutions/tree/master/Firebase\\_Leaderboard](https://github.com/FirebaseExtended/unity-solutions/tree/master/Firebase_Leaderboard)
- [14] V. Page, "What is amazon web services and why is it so successful?" [Online]. Available: <https://www.investopedia.com/articles/investing/011316/what-amazon-web-services-and-why-it-so-successful.asp>
- [15] Git, "Git." [Online]. Available: <https://git-scm.com/>
- [16] Trello, "Trello." [Online]. Available: <https://trello.com/>
- [17] Microsoft, "Visual studio." [Online]. Available: <https://visualstudio.microsoft.com/es/>
- [18] I. GitHub, "Github." [Online]. Available: <https://github.com/>

## Anexo

### Ejemplos de código

Anexo 1

---

```
private void DrawProjection(Vector3 finalPos){
[... ]
int i = 0;
lineRenderer.SetPosition(i, startPos);
for(float time = 0; time < LinePoints; time += timeBetweenPoint)
{
    i++;
    Vector3 point = startPos + time * newVel;
    point.y = startPos.y + newVel.y * time + (Physics.gravity.y
        / 2f * time * time); //  $y = v_i * t + 1/2 * a * t^2$ 
    lineRenderer.SetPosition(i, point);
    Vector3 lastPos = lineRenderer.GetPosition(i - 1);
    if (Physics.Raycast(lastPos, (point - lastPos).normalized,
        out RaycastHit hit, (point - lastPos).magnitude,
        grenadeCollisionMask))
    {
        lineRenderer.SetPosition(i, hit.point);
        lineRenderer.positionCount = i + 1;
        return;
    }
}
// [... ]
```

---

Código 1: Función de cálculo de trazado de la granada

---

```
public struct InputPayload : INetworkSerializable
{
    public int tick;
    public DateTime timestamp;
    public Vector3 inputVector;

    public void NetworkSerialize<T>(BufferSerializer<T> serializer)
        where T : IReaderWriter
    {
        serializer.SerializeValue(ref tick);
        serializer.SerializeValue(ref timestamp);
        serializer.SerializeValue(ref inputVector);
    }
}
// [... ]
```

---

Código 2: Estructura del input del usuario enviado al servidor

---

```
void Extrapolate()
{
    if (IsServer && extrapolationTimer.IsRunning)
    {
        if (extrapolationState.inputVector != Vector3.zero)
        {
            transform.position += extrapolationState.inputVector *
                Time.deltaTime * speed * ping / 10000 *
                extrapolationMultiplier;
        }
    }
}
// [...]
```

---

Código 3: Función de extrapolación