ESCUELA TÉCNICA SUPERIOR DE INGERIERÍA INFORMÁTICA

GRADO EN DISEÑO Y DESARROLLO DE VIDEOJUEGOS

Curso Académico 2022/2023

Trabajo Fin de Grado

DISEÑO Y DESARROLLO DE UN VIDEOJUEGO MULTIJUGADOR APLICANDO TÉCNICAS DE MEJORA Y OPTIMIZACIÓN A TRAVÉS DE INVESTIGACIÓN DEL CAMPO PARA SU CORRECTO FUNCIONAMIENTO

Autor: David Pozo Sánchez

Director: Julio Guillén García

Usted es libre de:

- Compartir — copiar y redistribuir el material en cualquier medio o formato.
- Adaptar — remezclar, transformar y construir a partir del material.

Bajo los siguientes términos:

- Atribución — Usted debe dar crédito de manera adecuada, brindar un enlace a la licencia, e indicar si se han realizado cambios. Puede hacerlo en cualquier forma razonable, pero no de forma tal que sugiera que usted o su uso tienen el apoyo de la licenciante.
- NoComercial — Usted no puede hacer uso del material con propósitos comerciales.
- CompartirIgual — Si remezcla, transforma o crea a partir del material, debe distribuir su contribución bajo la lamisma licencia del original.

A mis amigos y compañeros de carrera
por haber estado ahí todo el tiempo
y por las aventuras que hemos vivido.

A mi tutor Julio por haberme ayudado siempre.

A mi hermano.

A mi madre y mi padre.

"No hay imposibles, solo improbables"

Resumen

Este documento muestra el diseño y desarrollo de un videojuego multijugador desarrollado en Unity con Netcode for GameObjects entre otras herramientas.

Para ello se realiza una breve introducción y descripción del problema que se enfrenta junto a los objetivos que se proponen y la metodología que se ha llevado a cabo.

Posteriormente se introduce la parte teórica de este proyecto, la cual incluye los conceptos teóricos relevantes y necesarios para el desarrollo de este proyecto, la historia destrás del campo de los videojuegos multijugador y algunos casos de estudio que ayudan a entender todo el contexto de este proyecto.

Después de la parte teórica se presenta la parte práctica con un breve resumen de diseño del juego. Más adelante se especifica y desarrolla que el videojuego se ha dividido en las etapas de análisis, diseño e implementación del mismo.

Tras realizar todo el proyecto se realizó un proceso de validación que consistió en someter el código a diferentes pruebas que verifican su correcto funcionamiento.

Por último, se ha realizado una discusión del cumplimiento de los objetivos, un postmortem del videojuego y unas líneas futuras a modo de resumen de todo el desarrollo del proyecto.

Palabras Clave

Videojuego

Multijugador

Unity

Netcode for GameObjects

Abstract

This project consists of the development of a multiplayer video game using Unity with its Unity for GameObjects library, along with other Unity or third-party tools. To carry out this development, a study has been previously conducted on the evolution of multiplayer video games and their advancements throughout history, as well as analyzing and studying the current state of multiplayer video game development in order to create one.

The video game that will be developed to later become multiplayer will be a **Brawl Stars** style game with shooting and movement that will need to be synchronized over the network.

The objective of this project is to create a fully functional multiplayer video game, and above all, that the multiplayer part works correctly. To achieve this, research will be conducted on current techniques for multiplayer development, and the best ones will be selected to be implemented and to test the advantages and disadvantages of each compared to the others. Once this is done, the technique that best fits the video game will be further explored. All of this process will be accompanied by validation to ensure that everything works correctly.

Keywords

Videogame

Multiplayer

Unity

Netcode for GameObjects

Índice de contenido

Índice de ilustraciones

1. Introducción

1.1. Descripción del proyecto

Se propone un videojuego multijugador desarrollado en el motor de videojuegos Unity apoyándose en herramientas externas, ya sean proporcionadas por Unity o por terceros.

Para el desarrollo de este videojuego multijugador se hará uso de diferentes técnicas multijugador para el correcto funcionamiento de las partidas online. Antes de empezar el desarrollo se realizará una investigación previa donde se recopilará información sobre cómo funciona un videojuego multijugador, sus técnicas, como se aplican y los problemas que pueden surgir durante su desarrollo, todo esto, junto a cómo está la situación del multijugador en el motor de videojuegos Unity que es donde se llevará a cabo el proyecto.

La principal herramienta que se va a usar para la parte de multijugador va a ser *Netcode for GameObjects* (NGO), una librería de alto nivel de Unity dedicada a la comunicación multijugador en dicho motor. Cuando se inició el proyecto esta librería aún estaba en versión 0.1.0, pero actualmente la versión más reciente es la 1.2.0. Esta librería proporciona herramientas para el paso de mensajes en línea (en inglés, *online)* y la sincronización de datos, pero no es suficiente por lo que se hará uso de otras herramientas externas como Photon o *Unity Gaming Services (UGS)*, en las cuales se profundizará más adelante.

Una vez terminado el proyecto, este será un videojuego multijugador totalmente funcional en el que se habrán aplicado técnicas multijugador para el correcto funcionamiento del mismo, a través de NGO. En este videojuego se podrá crear salas, unirse a ellas y poder jugar partidas con otros jugadores en tiempo real.

Ilustración 1 Logo Netcode for GameObjects

1.2. Motivación

Los videojuegos multijugador son cada vez más populares y están en auge debido a que las personas cada vez están más conectadas, y una de las herramientas que usan para

interactuar son los videojuegos, ya que les permite estar en un entorno audiovisual e interactivo, algo que solo los videojuegos multijugador pueden ofrecer.

Por otro lado, la ya no tan reciente salida de la librería NGO para Unity es una gran motivación para crear un videojuego multijugador ya que es la primera librería oficial de Unity dedicada especialmente a los videojuegos multijugador ofreciendo multitud de herramientas como paso de mensajes o sincronización de datos entre otras funcionalidades.

Debido al gran abanico de posibilidades que existen en cuanto a tipos de videojuegos, diferentes mecánicas y diferentes formas de implementarlas entre otros factores, a la hora de diseñar y crear la parte multijugador de un videojuego, en cada uno de ellos es completamente diferente debido a que hay que adaptarse a diversas condiciones para que funcione correctamente y aplicar diferentes técnicas y optimizaciones dependiendo de lo que requiera el videojuego.

Como consecuencia de todo esto, otra gran motivación de este proyecto es aprender y explorar las diferentes soluciones para las diferentes situaciones que pueden ocurrir. Como ya se ha comentado hay un gran número de situaciones por lo que se hará una discretización para centrarse en las que más se repiten para posteriormente usar la que mejor se adapte al videojuego de este trabajo de fin de grado (TFG).

1.3. Objetivos

El objetivo final de este proyecto es crear un videojuego multijugador completamente funcional con Unity y su librería NGO, haciendo que este funcione bien, especialmente la parte *online*, para la que hará falta investigar y estudiar todo el ámbito multijugador. Pero para llegar a este objetivo hay otros objetivos que cumplir:

- Analizar la historia y evolución de los videojuegos multijugador para comprender su funcionamiento y las técnicas utilizadas en su desarrollo.
- Investigar la situación actual de la creación de videojuegos multijugador, examinando las técnicas utilizadas y su aplicación para resolver problemas específicos.
- Desarrollar un videojuego multijugador aplicando el conocimiento adquirido y empleando diferentes técnicas para evaluar su efectividad.

- Validar el proyecto mediante pruebas exhaustivas para garantizar su correcto funcionamiento y éxito.

1.4. Metodología

Para la realización de este proyecto, se ha empleado una metodología de desarrollo iterativo e incremental. Desde el inicio, se ha realizado una planificación exhaustiva que incluye todas las tareas necesarias para su ejecución.

Utilizando esta metodología, se han llevado a cabo ciclos repetitivos de desarrollo, lo que ha permitido una mayor flexibilidad y adaptabilidad a medida que se avanza en el proyecto.

Esta aproximación ha brindado la oportunidad de obtener retroalimentación temprana, validar el progreso y realizar mejoras incrementales en el trabajo realizado. Además, ha contribuido a minimizar riesgos y maximizar la eficiencia en el desarrollo de este proyecto.

El reparto de estas tareas se ha dividido en iteraciones, donde el objetivo es que al terminar cada iteración se obtenga un producto cerrado y mejor que en la iteración anterior, lo que significa que al final de cada iteración se tiene un producto funcional y ausente de bugs, pero no completo hasta la última iteración.

A pesar de que el sistema utilizado tiene algunas similitudes con otros enfoques ágiles como SCRUM, no se ha requerido enfatizar en la coordinación y comunicación que comúnmente se asocia con estas metodologías de desarrollo de software, ya que el proyecto ha sido llevado a cabo por un único desarrollador.

1.4.1. Planificación

Para crear, organizar y llevar un seguimiento de las tareas e iteraciones se ha usado la herramienta *online* Trello. Debido a ser una sola persona la responsable del proyecto no ha hecho falta hacer un panel demasiado complejo ya que al solo tener que organizarse una sola persona no hace falta crear exceso de etiquetas, roles y columnas.

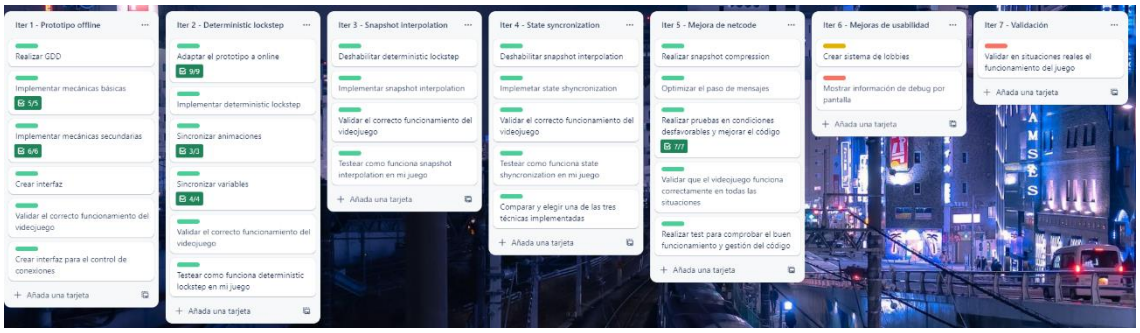


Ilustración 2 Trello - Parte 1

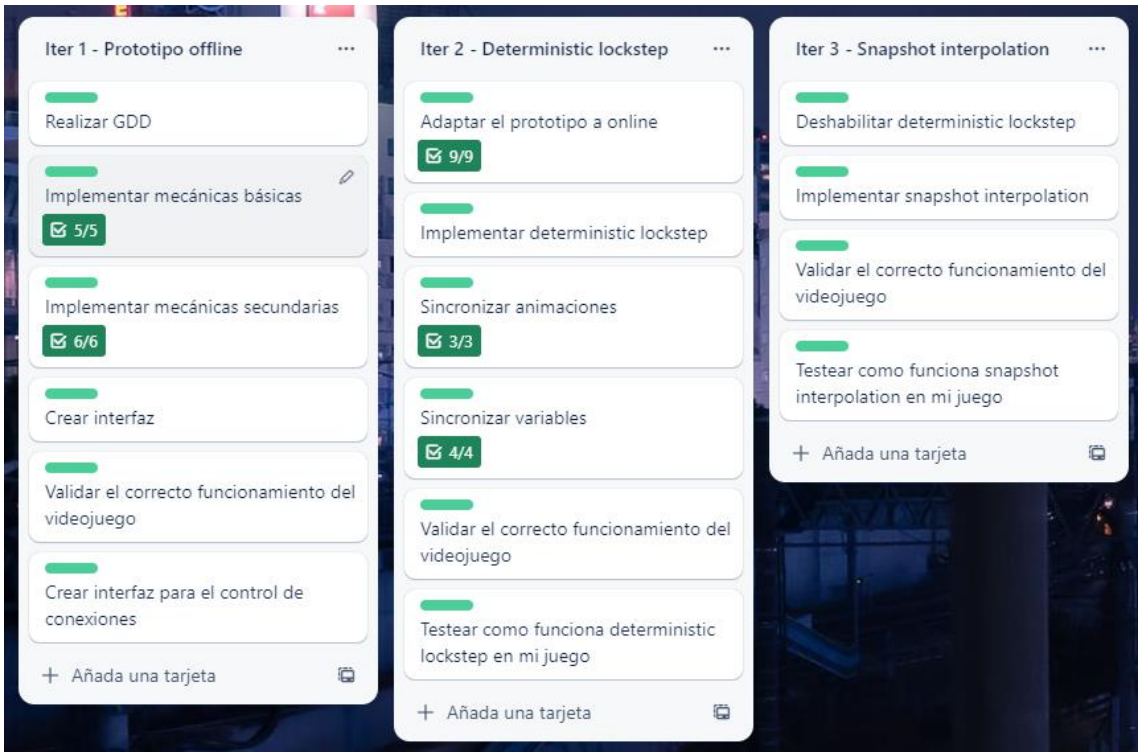


Ilustración 3 Trello - Parte 2

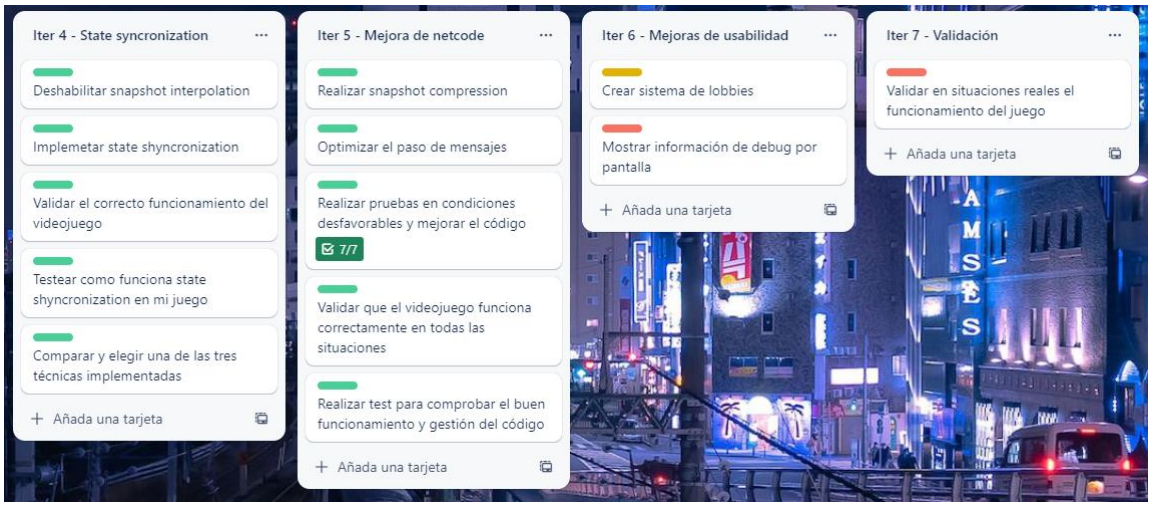


Ilustración 4 Trello - Parte 3

Para llevar una mayor organización las tareas están marcados con un color, el verde significa que ya se ha completado la tarea, el amarillo significa que está en desarrollo y el rojo es que aún no se ha hecho.

1.4.2. Desarrollo de la planificación

Para realizar la planificación de cómo se iba a desarrollar el proyecto se ha usado la herramienta Microsoft Project donde se han marcado unas etapas en la que cada una tiene un objetivo.

Al final de cada etapa tiene que haber una nueva versión del proyecto funcional y ausente de *bugs*. Las etapas de desarrollo tienen una duración de aproximadamente 2 semanas cada una ya que son objetivos de dificultad similar por lo que se ha empleado un tiempo similar en su desarrollo, por otro lado, las partes teóricas han durado más ya que es un proceso más lento y que se hace en paralelo con las etapas de desarrollo. Durante cada etapa se hace un planteamiento, desarrollo y *testing* del objetivo que se tiene en esa etapa.

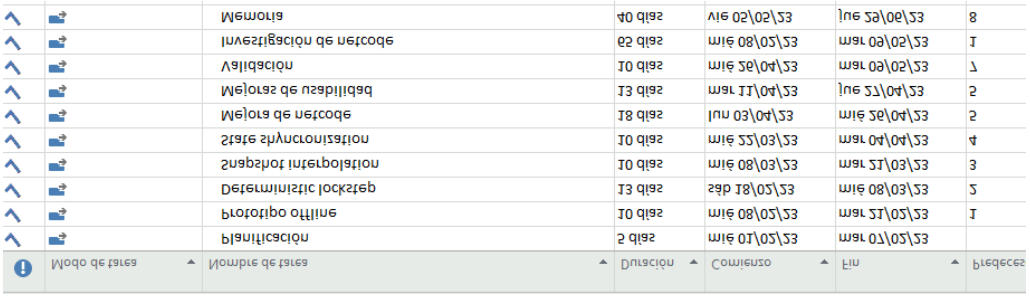


Ilustración 5 Microsoft Project - Parte 1 (pequeño)

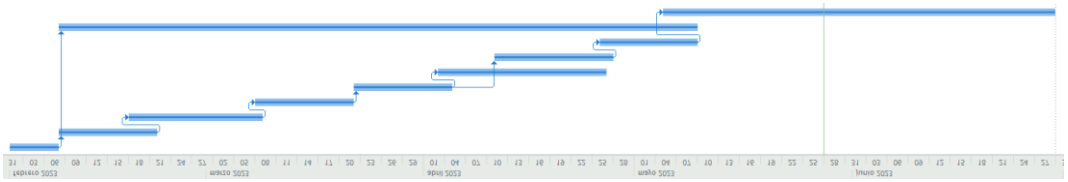


Ilustración 6 Microsoft Project – Parte 2 (pequeño)

Para ver las imágenes con más detalles consultar el Anexo IV

2. Marco teórico de los videojuegos multijugador

2.1. Historia de los videojuegos multijugador

Para empezar a hablar de los videojuegos multijugador y su historia primero tenemos que definir lo que es un videojuego multijugador.

Un videojuego multijugador es un tipo de entretenimiento digital que permite a varios jugadores interactuar en un mismo entorno virtual, ya sea cooperando para lograr un objetivo común o compitiendo entre sí en una variedad de juegos y modalidades. Estos juegos permiten a los usuarios conectarse en línea o jugar juntos en una misma pantalla y, por lo general, emplean diversas técnicas multijugador para asegurar el correcto funcionamiento del juego en red.

Los videojuegos multijugador son aquellos que poseen cualquier modalidad de juego que permita la interacción de dos o más jugadores al mismo tiempo, ya sea de manera física en una misma consola, en 2 o más portátiles (incluyendo teléfonos móviles) mediante cables o conexión inalámbrica, o mediante servicios en línea u otro tipo de red con personas conectadas a la misma.

Entre las diferentes formas que hay de jugar un videojuego multijugador están el juego en línea, donde cada jugador está en un dispositivo y en una red diferente del resto de jugadores, juego en red de área local (en inglés, *Local Area Network* (LAN)), donde los jugadores están en una misma red, pero en diferentes dispositivos, y por último está el juego local, donde los jugadores juegan en el mismo dispositivo, más en concreto en la misma pantalla, y por lo tanto en la misma red.

Para mantener la coherencia entre los diferentes jugadores se utilizan técnicas de sincronización de estado de juego para mantener la coherencia del juego en tiempo real y evitar retrasos o interrupciones.

A continuación se van a ver los videojuegos multijugador más importantes y que más relevancia han tenido de la historia.

2.1.1. Tennis for Two

Ahora que sabemos que es un videojuego multijugador podemos empezar a hablar de su historia, la cual comienza con el que es considerado por muchos el primer videojuego de la historia, **Tennis for Two** (1958), creado por William Higinbotham en el Laboratorio Nacional de Brookhaven. **Tennis for Two** era un videojuego multijugador local, el cual consistía en un simulador de tenis en vista lateral compuesto por una línea horizontal que simulaba el suelo, otra vertical que simulaba la red y un punto que se movía que era la pelota.

Para jugar a **Tennis for Two**, al ser un videojuego local, ambos jugadores jugaban en la misma pantalla, aunque en este caso no se jugaba en una pantalla, sino en un osciloscopio, el cual era un instrumento electrónico que se utilizaba para medir señales eléctricas en la época (1958). El dispositivo de entrada que usaba este osciloscopio para poder jugar a **Tennis For Two** era un dispositivo analógico hecho a mano compuesto por un botón para hacer que la pelota rebotase y una rueda para cambiar el ángulo con el que la pelota iba a rebotar.

Tennis for Two marcó un antes y un después en la historia de los videojuegos multijugador ya que no solo fue el primer videojuego de la historia, sino que también fue el primero en el que podían jugar más de una persona de forma simultánea en una misma partida, para lo que fue necesario gestionar el input de más de un mando de forma simultánea.

Definitivamente este videojuego fue el inicio de la historia de los videojuegos multijugador.

Ilustración 7 Tennis for Two

2.1.2. Spacewar!

El siguiente paso en la historia de los videojuegos multijugador lo dio **Spacewar!** (1961), un videojuego desarrollado por Steve Russell, Martin Graetz y Wayne Wiitanen, estudiantes del MIT (Instituto Tecnológico de Massachusetts). Este videojuego se creó para demostrar las capacidades del recién salido en aquella época, el Procesador de datos programado (en inglés, *Programmed Data Processor-1* (PDP-1)), el cual ya usaba una pantalla en vez de un osciloscopio.

Spacewar! es un videojuego de naves espaciales donde cada jugador controla una nave y tienen que dispararse para destruir la nave rival mientras un pozo gravitatorio lo arrastra. Este juego tiene algunas peculiaridades como que el fondo de estrellas se generaba de forma aleatoria en cada partida y que contaba con número limitado de combustible y balas. Para poder jugar **Spacewar!** inicialmente se usaban los interruptores del PDP-1, pero posteriormente se creó un mando para poder jugarlo de forma más cómoda.

Aunque parece que **Spacewar!** no tuvo ningún avance significativo respecto a **Tennis for Two** ya que parecen similares, realmente **Spacewar!** tuvo ciertos logros relevantes como que se ejecutaba en un ordenador y usaba pantallas, haciéndolo más accesible, lo que provocó que llegará a más personas y gracias a que era de código abierto, es decir, cualquier persona puede acceder al código y modificarlo, muchas personas conocieron el

potencial de los ordenadores y se introdujeron en el mundo de los videojuegos multijugador mejorando la versión original de **Spacewar!**, llegando incluso a poder jugarlo en 2 pantallas diferentes, es decir, pasó de ser un videojuego local a LAN. Esto provocó un auge de los videojuegos, que al ser **Spacewar!** un videojuego multijugador, se fortaleció este sector. Este auge también fue gracias a que el ordenador donde se desarrolló, el PDP-1, empezó a traer **Spacewar!** instalado de fábrica para demostrar el gran potencial que podía llegar a ofrecer el ordenador, lo que hizo que se hiciese aún más popular.

Ilustración 8 Spacewar!

2.1.3. Computer Space y Pong

Con **Computer Space** (1971) los videojuegos dieron un gran paso en la historia ya que este estaba desarrollado por una empresa (Nutting Associates) y no por un grupo de estudiantes a modo de hobby, y no solo eso, sino que también fue el primer videojuego que se vendió de forma comercial en máquinas arcades con un funcionamiento de monedas.

Computer Space era un juego de naves espaciales en el que los jugadores controlaban una nave en una pantalla en blanco y negro, evitando obstáculos y disparando a enemigos en una carrera contra el tiempo.

En la historia de los videojuegos multijugador **Computer Space** no destacó ya que su modo de 2 jugadores era por turnos, es decir, primero una persona juega un nivel y después otra jugaba el mismo nivel para posteriormente comparar la puntuación, pero a

pesar de no destacar en el ámbito multijugador, fue una gran base para los videojuegos para el resto de la historia por ser el primero en comercializarse y por ser la base de un videojuego mítico, el **Pong**, ya que los creadores de **Computer Space**, Nolan Bushnell y Ted Dadney, posteriormente fundaron Atari y lanzaron el videojuego **Pong**.

Pong es considerado como uno de los videojuegos más influyentes de la historia y un hito en la evolución de los videojuegos multijugador. Desarrollado por Atari en 1972, **Pong** fue el primer juego arcade comercialmente exitoso y el primer videojuego en alcanzar la cultura popular en masa.

Pong es un juego de tenis de mesa electrónico para dos jugadores. Los jugadores controlan las paletas en la pantalla y tratan de golpear una pelota virtual hacia el campo del oponente sin dejar que la pelota caiga en su propia área. El juego es muy simple en su mecánica de juego, pero su impacto en la industria de los videojuegos fue enorme.

Pong fue el primer videojuego en ofrecer una experiencia multijugador a gran escala, ya que fue el primer videojuego multijugador que llegó a una gran cantidad de público debido a su popularidad e hizo que los jugadores amasen los videojuegos multijugador debido al factor competitivo que ofrecían.

Ilustración 9 Máquina Computer Space

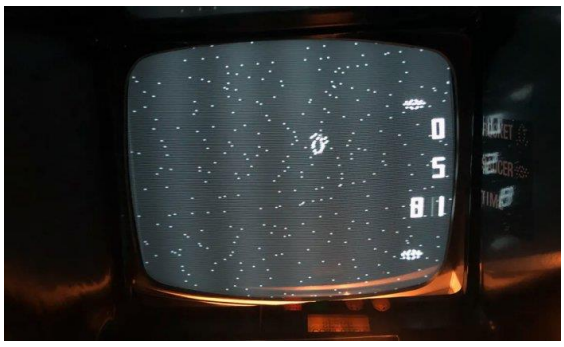


Ilustración 10 Computer Space

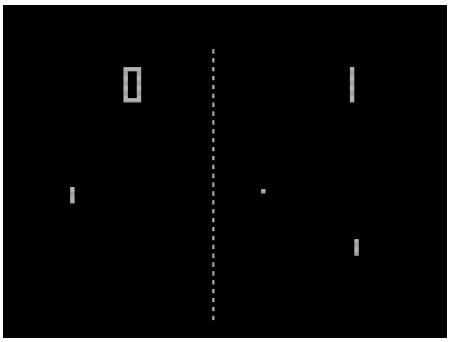


Ilustración 11 Pong

2.1.4. La plataforma PLATO

Hasta ahora todos los videojuegos que hemos visto han sido multijugador, pero jugando en la misma pantalla, pero a partir de la década de los 70 empezaron a aparecer videojuegos multijugador en los que cada persona jugaba en un dispositivo distinto pero que debían estar conectados a la misma red, es decir, jugaban en LAN. Este gran avance fue gracias a PLATO el cual fue un sistema informático desarrollado en la Universidad de Illinois en la década de 1960, que a pesar de que estaba diseñado principalmente para la educación, se convirtió en un importante precursor de los videojuegos multijugador en línea. La plataforma PLATO ofrecía un abanico de videojuegos, entre los que había alguno multijugador como **`Empire`** (1973), el cual es considerado el primer videojuego multijugador (sin contar los que se jugaban en la misma pantalla).

En **`Empire`** el objetivo del juego era conquistar y controlar un territorio mientras se defendía de los ataques de otros jugadores. Los jugadores se enfrentaban en un mapa dividido en hexágonos, cada uno representando un territorio, y podían mover sus unidades a través de ellos. El juego se desarrollaba en tiempo real, lo que significa que los jugadores podían tomar decisiones y dar órdenes en cualquier momento, lo que les daba una sensación de libertad y control. **`Empire`** fue un éxito inmediato en la red PLATO y fue uno de los primeros juegos en permitir partidas con un gran número de jugadores simultáneos.

Como ya hemos dicho, gracias a PLATO, para jugar videojuegos multijugador no era necesario estar en el mismo sitio usando la misma pantalla ya que ahora solamente hacía falta estar conectado a la misma red, esto se lograba a un sistema de un ordenador central (en inglés, *mainframe)* (servidor) y "terminales tontos" (clientes). Para que estos juegos funcionasen cada jugador se conectaba a un "terminal tonto", llamados así porque no tenían capacidad de cómputo ya que lo único que hacían era pintar en pantalla el estado

del juego que le llegaba del servidor y enviar las entradas de control (en inglés, *inputs)* al servidor, y cada uno de estos terminales tontos estaba conectado por red a un *mainframe*, un ordenador con capacidad de cómputo que llevaba la lógica del videojuego a través de los *inputs* que le llegaban de los jugadores, para después enviarle el estado del juego a todos los terminales tontos para que estos lo pinten por pantalla. Como ya hemos hablado los clientes era terminales tontos que no tenían capacidad de cómputo por lo que era el *mainframe* quien le daba ciclos de la Unidad Central de Procesamiento (en inglés, *Central Processing Unit* (CPU)) para que tuviera mayor tasa de refresco.

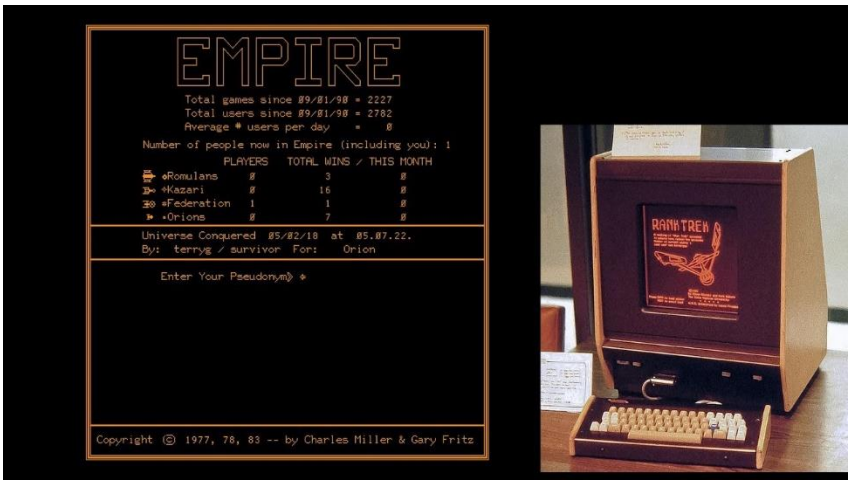


Ilustración 12 Empire - PLATO

2.1.5. Multi-User Dungeon (MUD)

A finales de la década de los 70, tras la gran popularidad de PLATO, en 1978 aparece **Multi-User Dungeons** (**MUD**), que después de 20 años desde la aparición del primer videojuego de la historia (**Tennis for Two**), es el primer videojuego *online*, donde cada jugador puede estar en un dispositivo y red completamente diferentes.

MUD permitía a los jugadores interactuar en un mundo virtual en línea en tiempo real, usando texto para comunicarse y tomar decisiones que afectaban el mundo virtual y la experiencia de juego de otros jugadores. El juego fue diseñado para ser jugado por varios jugadores a la vez, y permitía que los jugadores se unieran en grupos para explorar el mundo virtual y luchar contra monstruos.

MUD fue uno de los primeros juegos en ofrecer un ambiente multijugador persistente *online*, donde los jugadores podían entrar y salir del juego en cualquier momento y las acciones de los jugadores continuaban teniendo efecto en el mundo virtual incluso cuando no estaban conectados.

Para conseguir este logro de poder jugar a través de red se usó un sistema cliente servidor, en el que por un lado hay un servidor central que alojaba el mundo virtual para tener persistencia de datos aparte de para mantener sincronizados todos los clientes, y por otro lado hay terminales que se conectaban al servidor como clientes. El protocolo de comunicación que usaba **MUD** era Telnet, el cual usaba método de envío de paquetes y fue muy usado en las décadas de 1970 y 1980 y permitía a los usuarios conectarse a un servidor remoto a través de internet. Debido a las limitaciones de la época hacía falta usar técnicas para optimizar la comunicación *online*, y una de esas técnicas era la compresión de datos LZ77 la cual utiliza un diccionario de coincidencias para buscar y reemplazar secuencias de datos que ya se han transmitido previamente. Al reemplazar estas secuencias repetitivas con un código más corto, se puede reducir significativamente el tamaño total de los datos que se transmiten a través de la red.

El éxito de **MUD** llevó al desarrollo de otros juegos de rol multijugador en línea y sentó las bases para los juegos en línea modernos. Además, **MUD** ayudó a popularizar la idea de que los juegos en línea podían ser experiencias sociales, y no solo juegos solitarios jugados en una pantalla.

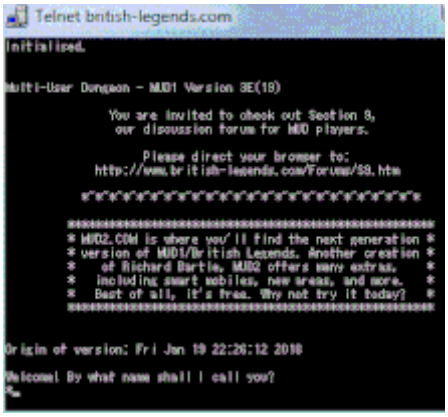


Ilustración 13 Multi User Dungeon

2.1.6. Hosted Online Games

En la década de los 80, una nueva forma de disfrutar los videojuegos surgió con la idea del pagar para jugar (en inglés, *pay for play)*. Las compañías empezaron a ofrecer juegos *online* en los que los usuarios pagaban una cuota mensual por jugar a ellos. Un ejemplo de esto es el **MegaWars 1** de Compuserv. Para que esto funcionase las compañías ofrecían ciclos de CPU cuando las computadoras estaban inactivas para permitir a los jugadores acceder a estos juegos mediante pago.

MegaWars 1 fue uno de los primeros juegos en línea que utilizó el modelo de pagar para jugar (en inglés, *pay for play*), en el que los jugadores debían pagar una tarifa mensual para acceder al juego. La empresa CompuServe, que desarrolló y hospedó el juego, ofrecía ciclos de CPU cuando las computadoras estaban inactivas, lo que permitía a los jugadores jugar en línea sin tener que comprar su propio hardware. Para jugar a **MegaWars 1**, los usuarios debían suscribirse a CompuServe y luego pagar una tarifa adicional para acceder al juego.

Sin embargo, debido a los altos precios de estas plataformas, surgieron alternativas como las BBS (*Bulletin Board Systems*), los cuales eran sistemas informáticos que permiten a los usuarios comunicarse entre sí a través de mensajes y compartir archivos en una red de ordenadores. Estas permitían a los usuarios conectarse a través de un módem y acceder a una amplia variedad de juegos, incluyendo **MUDs** (**Multi-User Dungeons**). Esto permitió sacar los MUD de las redes universitarias y llevarlos al público en general.

El sistema que usaban estas formas de jugar en línea era similar a la de MUD, era un sistema cliente-servidor donde era el servidor quien guardaba el mundo y ejecutaba las instrucciones de los clientes para después pasarles el estado del juego. Sin embargo, debido a la inminente llegada de Internet y la popularización de la *World Wide Web*, estas formas de jugar videojuegos multijugador fueron perdiendo su relevancia y la mayoría cerraron.

2.1.7. Doom

El siguiente gran juego que marcó una diferencia en el mundo de los videojuegos multijugador fue **Doom**, lanzado en 1993 por idSoftware. Este videojuego es considerado uno de los más influyentes en la historia de los videojuegos, especialmente en lo que se refiere a la experiencia multijugador ya que a pesar de que en aquellos días los videojuegos *online* eran bastante rudimentarios debido a que la mayoría de los jugadores se conectaban a través de módems de baja velocidad y el juego se ejecutaba a través de conexiones de texto, **Doom** gracias al protocolo IPX de Novell (entre otras técnicas), conseguía una buena experiencia de juego. El protocolo IPX (*InterPacket Exchange*) es un protocolo de red a nivel de enlace de datos que se utilizaba para establecer conexiones de red en juegos multijugador mediante paquetes de datos llamados datagramas en una arquitectura red de pares (en inglés, *peer-to-peer* (P2P)).

La arquitectura P2P es una forma de comunicación descentralizada entre equipos o dispositivos, en la que no hay un servidor centralizado que controle todo el tráfico de datos. En cambio, cada dispositivo en la red, también conocido como nodo, puede actuar como cliente y servidor al mismo tiempo, lo que permite que cada nodo se comunique directamente con otros nodos sin pasar por un servidor central.

Doom no solo revolucionó el mundo de los videojuegos multijugador por conseguir una buena conexión cuando los módems ofrecían baja velocidad, sino que también de forma similar a **Spacewar!** que al ser código abierto (en inglés, *open source)* consiguió llegar a más personas y que crearan modificaciones del juego original, **Doom** fue uno de los primeros juegos en permitir a los jugadores crear y compartir sus propios niveles y modificaciones a través del uso de herramientas como el **Doom Editing Utility** (**DEU**). Esto ayudó a fomentar una comunidad de jugadores y *modders*, lo que llevó a la creación de numerosas modificaciones (en inglés, *mods)* y mapas personalizados que ampliaron la vida útil del juego.

Ilustración 14 Doom

2.1.8. Quake

Después del éxito de **Doom**, idSoftware (los creadores de **Doom**) lanza **Quake** solo 3 años más tarte (1996), el cual mejoró la experiencia multijugador respecto a su predecesor.

Quake es un videojuego de disparos en primera persona, el cual fue uno de los primeros juegos en utilizar gráficos en 3D. **Quake** se convirtió en un éxito y estableció el estándar

para los juegos de disparos en primera persona en línea. Su código fuente fue liberado en 1999 y ha sido ampliamente utilizado por la comunidad de modificación de juegos.

Respecto a su apartado multijugador, **Quake** dio un gran paso que marcó el futuro de como funcionaban los videojuegos multijugador ya que su arquitectura era cliente-servidor (al contrario de **Doom** que era *P2P*), acompañado de servidores que funcionaban 24 horas al día, por lo que ya no era necesario que una persona fuera el host y conocieses su IP para poder conectarte a su partida, con este sistema era mucho más fácil jugar al multijugador ya que solo tenías que conectarte a un servidor público y podías jugar con más personas, esto produjo un gran salto en la historia de los videojuegos multijugador ya que marcó como funcionarían las bases hasta día de hoy.

Pero este sistema también tenía algunas desventajas como la latencia debido a las lentas conexiones de la época, el *ping*, que es el tiempo que tarda en viajar la información del cliente al servidor y volver y la pérdida de paquetes, que es la información que se pierde. Debido a estas inconveniencias a veces la experiencia de juego no era tan buena, pero poco después del lanzamiento lanzaron la actualización gratuita llamada QuakeWorld la cual introdujo optimizaciones para el multijugador como la predicción en el lado del cliente (técnica que se sigue usando hoy en día), lo que permitía que los clientes predijesen el estado del juego en caso de que tuvieran mala conexión.

Ilustración 15 Quake

2.1.9. Age of Empires

Age of Empires fue lanzado en 1997 por Ensemble Studios como un juego de estrategia en tiempo real (RTS) que permitía a los jugadores controlar una civilización y competir

con otras de forma *online*. El juego ofrecía soporte para hasta 8 jugadores, donde cada uno controlaba 50 unidades, es decir, había que sincronizar 400 unidades en tiempo real de forma *online*, algo que para los módems de la época de 28.8kbps era difícil de gestionar.

Para superar estas limitaciones, **Age of Empires** utilizó una arquitectura *P2P*, donde los datos se enviaban directamente entre los clientes en lugar de pasar a través de un servidor centralizado. Además, empleaba un protocolo de datos fiables para asegurar que los datos llegaran correctamente y en el orden adecuado. Pero la estrategia más importante que usaron para conseguir superar este desafío fue usar *deterministic lockstep*. El *deterministic lockstep* es una técnica (que se sigue usando hoy en día) donde solo se enviaba la información de control de cada jugador y cada equipo realizaba una simulación determinista para cada unidad, es decir, en vez de enviar el estado de cada unidad, solamente se enviaba el input del jugador para posteriormente replicar ese input en todos los clientes, logrando así sincronizar cientos de tropas aún con las limitaciones de la época. Hay que destacar que el *deterministic lockstep* aunque es muy bueno para sincronizar un gran número de entidades tiene dos desventajas, la primera es que no es fácil de implementar, y la segunda es que no siempre se puede usar ya que solo funciona cuando el estado de un videojuego o una entidad es determinista, es decir, cuando a través de un mismo input siempre va a suceder exactamente lo mismo, lo que deja fuera del alcance juegos o entidades que usan fuerzas, aceleraciones y velocidades entre otros.

Ilustración 16 Age of Empires

2.1.10. Ultima Online

Ultima Online es considerado el primer juego de rol multijugador masivo en línea (en inglés *Massively Multiplayer Online Role-Playing Game* (MMORPG), lanzado en 1997 por Origin Systems. El juego permitía a miles de jugadores de todo el mundo conectarse a un mismo mundo virtual en línea y explorar, interactuar y jugar juntos.

Para lograr gestionar el multijugador con todos los posibles jugadores que puede haber de forma simultánea al ser un MMO, **Ultima Online** utilizaba una arquitectura cliente-servidor y un protocolo basado en TCP/IP donde el servidor controlaba todos los aspectos del mundo virtual y cada cliente se conectaba al servidor para recibir actualizaciones y enviar comandos.

El paso de mensajes se realizaba a través de un sistema de eventos que se ejecutaban en el servidor. Cuando un jugador realizaba una acción, como mover su personaje o atacar a un enemigo, el cliente enviaba un mensaje al servidor con la información de la acción. El servidor procesaba el evento y enviaba la información actualizada a todos los clientes que le hiciese falta esa información, por ejemplo, si un jugador da un golpe solo se les notifica a los jugadores que están cerca.

Para mantener la sincronización entre los clientes, **Ultima Online** utilizaba un sistema de autoridad del servidor. El servidor controlaba todos los aspectos del mundo virtual, como la posición de los objetos, la salud de los personajes y los eventos que sucedían en el mundo. Los clientes solo tenían acceso a la información que el servidor les proporcionaba.

Ilustración 17 Ultima Online

2.1.11. Starsiege: Tribes

Starsiege: Tribes es un videojuego de disparos en primera persona (en inglés, *first person shooter* (FPS)) desarrollado en 1998 por Dynamix y publicado por Sierra On-Line. Este videojuego implemento una serie de técnicas para lograr un correcto funcionamiento en sus partidas multijugador de hasta 128 jugadores simultáneos, a pesar de las limitaciones de la época con módems que iban a 55.6kbps.

Para conseguir este logro usaban una arquitectura cliente-servidor como ya hemos visto pero con un protocolo de transmisión de datos no fiable, algo poco visto. Para lograr que el multijugador funcionase a pesar de tener datos que no llegaban dividieron los mensajes en 4 tipos:

- Datos no garantizados: Este tipo de datos como su nombre indica no estaban garantizados que llegasen ya que no eran tan importantes. Por ejemplo, transmitir datos que no afecten al combate.
- Datos garantizados: Estos datos son los que hay que asegurarse de que llegan debido a que son muy importantes para el desarrollo de la partida. Por ejemplo, la acción de disparar, algo crucial para un juego de disparos.
- Estado de datos más reciente: Estos son aquellos en los que importa solamente el último valor, como por ejemplo la vida del jugador. En este caso la posición no estaría ya que la posición es importante en cada momento para la colisión de balas.

- Datos garantizados urgentes: Estos son los más importantes y de los que hay que asegurarse de que llegan seguro y lo más rápido posible como por ejemplo el movimiento.

Otra técnica que usaron fue el manejador fantasma (en inglés, *ghost manager)*, el cual consistía en que el servidor decide qué información necesita saber el jugador para darle hacerle llegar los paquetes más importantes para ellos.

Ilustración 18 Starsiege: Tribes

2.1.12. Servicios multijugador

Después de más de más de 40 años del inicio del mundo de los videojuegos con **`Tennis for Two`** en 1958, a partir del año 2000, los videojuegos multijugador son conocidos y queridos por todos y ya tenía una gigantesca comunidad. En este punto grandes empresas como Sony o Microsoft, aprovechando las nuevas tecnologías y el avance de la conexión a internet, empezaron a crear servicios que permitiesen jugar y crear videojuegos multijugador con más facilidad y comodidad. Algunos de esos son los siguientes

2.1.12.1. PS2 con soporte multijugador online (2000)

La PlayStation 2 fue la primera consola de videojuegos en ofrecer soporte para jugar en línea. Utilizó tecnologías como la conexión a internet de banda ancha y servidores dedicados para permitir que los jugadores jugaran en línea con amigos y otros jugadores de todo el mundo. Esto abrió la puerta para que otras consolas ofrecieran experiencias de juego en línea similares en el futuro.

2.1.12.2. Xbox & Xbox Live (2002)

La consola Xbox de Microsoft fue la primera en ofrecer soporte para jugar en línea con una suscripción a Xbox Live. Utilizó tecnologías como la conexión a internet de banda

ancha y los servidores dedicados para permitir que los jugadores se conectaran y jugaran juntos en línea. También implementó características como la mensajería instantánea y los logros en línea, lo que mejoró la experiencia del jugador y promovió una mayor interacción social.

Ilustración 19 Xbox Live

2.1.12.3. WiiConnect24 (2006)

Nintendo WiiConnect24 fue una característica en línea de la consola Wii que permitía a los jugadores conectarse en línea y descargar actualizaciones y contenido adicional mientras la consola estaba en modo de espera. También permitió a los jugadores conectarse con amigos y enviar mensajes. Aunque no fue tan influyente como algunas de las otras tecnologías mencionadas en esta lista, fue un paso importante para Nintendo hacia la conectividad en línea.

2.1.12.4. Play Station Plus (2010)

PlayStation Plus es un servicio de suscripción de PlayStation que ofrece juegos gratuitos, descuentos exclusivos y funciones multijugador en línea. Esto incluye la capacidad de jugar en línea con amigos y otros jugadores de todo el mundo. Además, ofreció almacenamiento en la nube para guardar juegos y datos de juego en línea, lo que permitió a los jugadores acceder a su progreso y estadísticas desde cualquier consola de PlayStation.

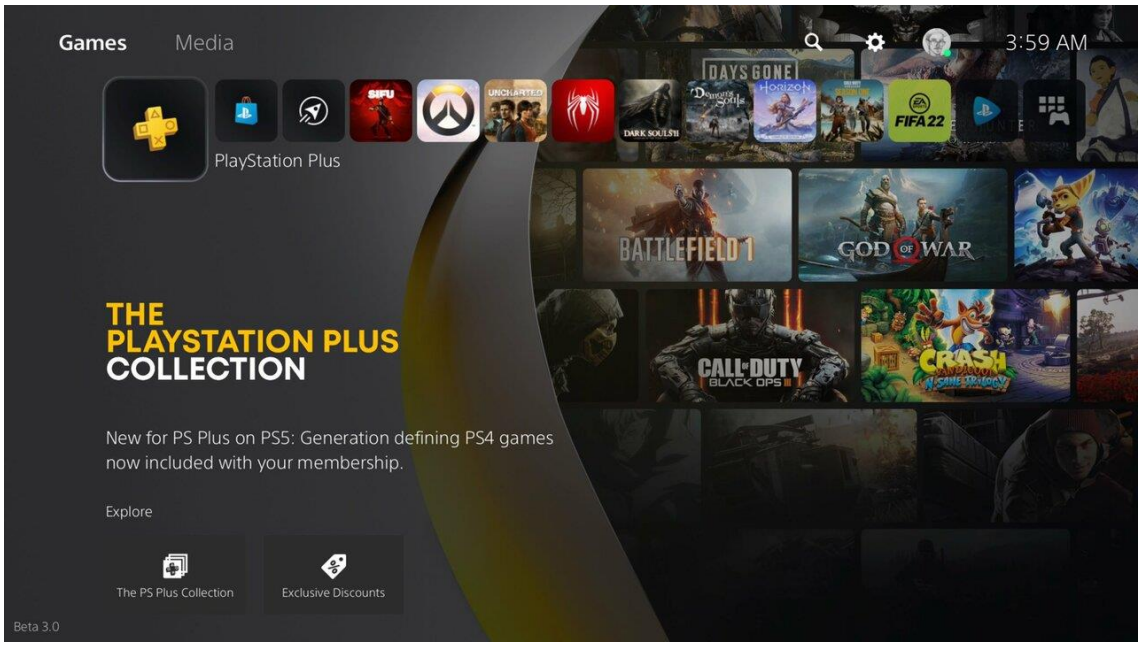


Ilustración 20 Play Station Plus

2.1.12.5. Nintendo Network (2012)

Nintendo Network es un servicio permitió a los usuarios jugar juegos multijugador en línea en la consola Nintendo Wii U y Nintendo 3DS. Además, el servicio permitió a los usuarios conectarse con amigos y otros jugadores en línea, así como comprar y descargar juegos digitales. También ofreció una función de chat de voz en línea que permitió a los usuarios comunicarse mientras jugaban. Nintendo Network utilizó una tecnología de red mejorada para mejorar la estabilidad de la conexión en línea y la experiencia multijugador.

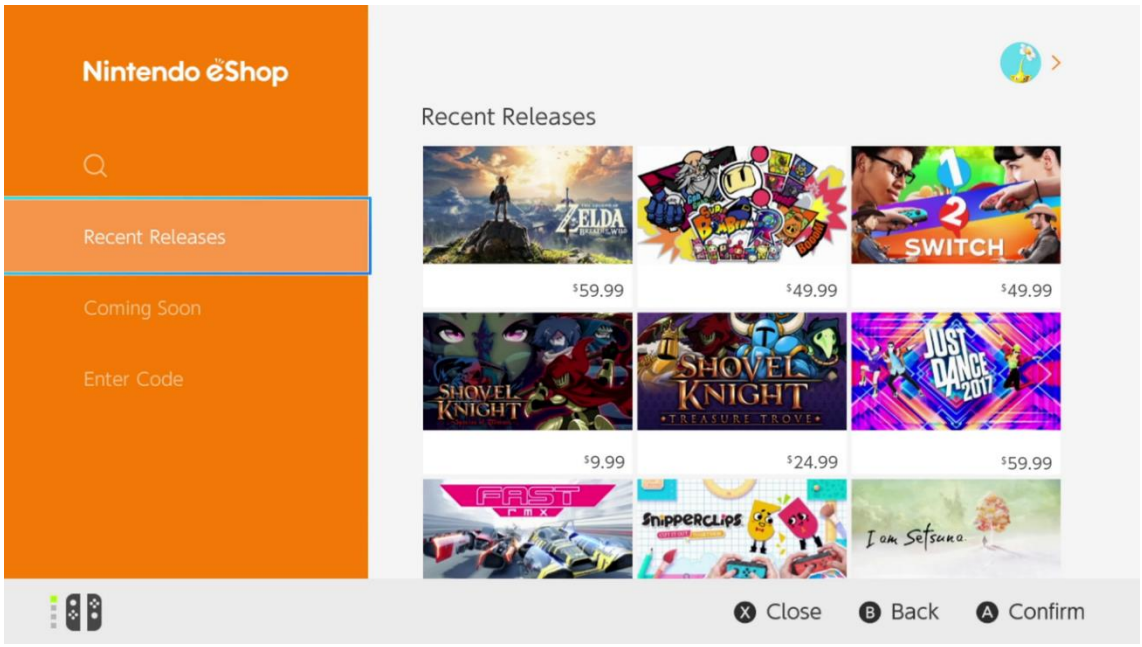


Ilustración 21 Nintendo Network

2.1.12.6. Conclusión

En general, todos estos avances han mejorado significativamente la accesibilidad y calidad del multijugador en línea en los videojuegos, permitiendo a los jugadores conectarse y jugar con otros usuarios de todo el mundo en tiempo real. Además, han permitido a los desarrolladores de juegos crear experiencias de juego más inmersivas y conectadas, que aprovechan al máximo el potencial del multijugador en línea.

2.1.13. Onlive

Para acabar con la historia de los videojuegos multijugador vamos a hablar de OnLive, una plataforma de videojuegos en la nube lanzada en 2010. Ofrecía una experiencia de juego en línea en tiempo real, sin necesidad de descargar o instalar los juegos en el ordenador o consola del usuario. La idea detrás de OnLive era permitir a los jugadores acceder a los juegos de alta calidad sin la necesidad de invertir en hardware costoso.

El funcionamiento de OnLive se basaba en la tecnología de *streaming* de video. Los juegos se ejecutaban en los servidores de OnLive y se transmitían en tiempo real al dispositivo del usuario, que podía ser una PC, Mac, TV o dispositivo móvil. Los usuarios controlaban el juego a través de una aplicación cliente que enviaba los comandos al servidor de OnLive, que a su vez procesaba el juego y enviaba los gráficos de vuelta al usuario.

Ilustración 22 Onlive

2.2. Mercado actual

Una vez llegamos a la década de 2010 las bases de los videojuegos multijugador y su *netcode* ya están bien cimentadas y estandarizadas. A lo largo de más de 50 años desde la aparición del primer videojuego multijugador, **Tennis for Two** (1958), se ha ido mejorando la experiencia multijugador a través de nuevas técnicas, mejoras del código, mejoras externas a los videojuegos como la velocidad de internet, nuevos protocolos, entre otras mejoras como ya hemos visto en el punto anterior hasta el punto de poder jugar un videojuego sin necesidad de descargarlo. Por lo que en este punto donde los videojuegos multijugador están bien asentados y todo el mundo los conoce, desde 2010 hasta la actualidad se invertido mucho esfuerzo en mejorar aún más la experiencia con nuevas técnicas, nuevos protocolos, con inversiones de dinero para servidores dedicados, entre otras mejoras que veremos más adelante para poder conseguir la mejor experiencia posible hasta el punto de que no parezca que estás jugando en tiempo real con alguien que está a cientos de kilómetros.

En cuanto a los videojuegos multijugador más destacados de este periodo, cabe mencionar algunos títulos como **Fortnite**, **Apex Legends**, **PlayerUnknown's Battlegrounds (PUBG)**, **Overwatch**, **GTA Online**, **Minecraft** o **ROBLOX**. Estos juegos han sido líderes en su género y han conseguido una gran popularidad gracias a su enfoque en el multijugador, así como a su innovación en cuanto a la jugabilidad y la interacción con los jugadores.

En resumen, desde el año 2010 la industria de los videojuegos multijugador ha experimentado una evolución constante, mejorando la gestión del *netcode*, la sincronización de los clientes, la reducción de la latencia y la implementación de nuevas técnicas para la gestión del multijugador. Todo esto ha permitido una experiencia más fluida y una mayor precisión en los juegos multijugador, lo que ha llevado a la popularidad de títulos como **Fortnite**, **Apex Legends**, **PUBG** y **Overwatch**.

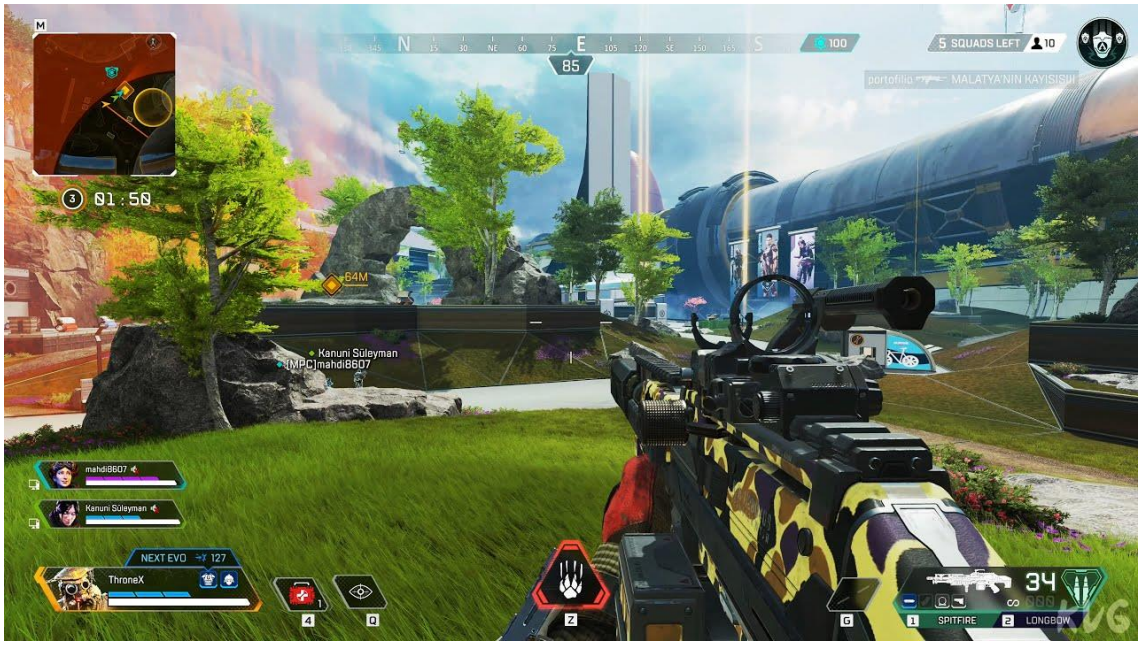


Ilustración 23 Apex Legends

2.3. Desafíos y problemáticas en los juegos multijugador

A pesar de que los avances que ha habido durante más de 60 años, en especial en los últimos 10 que son donde más esfuerzo y dinero se ha invertido en mejorar la experiencia multijugador, aún sigue habiendo problemas que afectan a dicha experiencia. Algunos de estos problemas son los siguientes:

- Latencia: La latencia es el tiempo que tarda un paquete de datos en llegar desde el emisor al receptor. Un alto nivel de latencia puede provocar retrasos en la transmisión de información, lo que puede afectar a la experiencia de juego y hacer que los movimientos de los personajes sean lentos y poco precisos.

  Para medir la latencia se usa el tiempo de ida y vuelta (en inglés, *round-trip-time (RTT)*), el cual es el tiempo que tarda un mensaje en ir al servidor y volver. Este problema no afecta igual a todos los juegos ya que por ejemplo en un **Call of Duty** es muy importante que el estado del juego esté perfectamente sincronizado

lo más rápido posible debido a que es necesario para tener una buena sensación de juego, mientras que por ejemplo en el **Monopoly Online** no es tan necesario ya que es juego por turnos y si un mensaje te llega tarde no afecta a la experiencia de juego.

- *Jitter*: El *jitter* se refiere a las variaciones en la latencia entre un paquete y el siguiente que pueden producirse durante la transmisión de datos. Un alto nivel de *jitter* puede provocar que los paquetes de datos lleguen desordenados, lo que puede afectar a la sincronización entre los jugadores y provocar fallos en la ejecución de acciones.

  El problema del *jitter* es una prolongación del problema de la latencia ya que no solo llegan los paquetes tarde, sino que también desordenados, lo que puede provocar situaciones extrañas donde por ejemplo un jugador de repente se mueve rápido hacia adelante y hacia atrás, ya que el jugador solamente se ha movido hacia adelante, pero los paquetes con su posición han llegado desordenados.
- Pérdida de paquete: La pérdida de paquetes se produce cuando un paquete de datos no llega a su destino. Esto puede deberse a diferentes factores, como la congestión de la red, la mala calidad de la conexión o la falta de ancho de banda. Si se pierden demasiados paquetes, la información recibida por los jugadores puede ser incompleta, lo que puede afectar a la experiencia de juego.

  Perder un paquete importante puede ser muy perjudicial para la experiencia ya que si solo se pierde un paquete de la posición no afecta significativamente porque al tener la siguiente posición ya se arregla mediante interpolación, pero en el caso del input de disparar afecta a la experiencia ya que supone que gane un jugador u otro.
- Fiabilidad de red: Cuando hablamos de fiabilidad de red esta se refiere a la capacidad de la red para mantener una conexión estable y fiable entre los jugadores. Si la conexión es inestable, pueden producirse fallos en la transmisión de datos, lo que puede afectar a la experiencia de juego y provocar la desconexión de los jugadores.

  Este problema es la principal consecuencia de los problemas que hemos visto anteriormente ya que una mala conexión de red produce diversos fallos en la comunicación de los videojuegos multijugador. Al ser uno de los principales problemas, las compañías introducen métodos y técnicas para disminuir el efecto

de este problema como numeración de paquetes, colas de procesamiento con garantías, separar claramente datos prioritarios de datos desechables, entre otras técnicas.

- Desincronización: La desincronización se produce cuando la información recibida por los jugadores no coincide entre sí. Esto puede deberse a diferentes factores, como una mala gestión del *netcode* o problemas de latencia, *jitter* o pérdida de paquete. Si los jugadores están desincronizados, la experiencia de juego puede verse afectada y puede ser difícil para los jugadores interactuar entre sí.

  Este es un problema muy grave ya que no es que un jugador vaya con desfasado respecto a otro, sino que a cada uno le ha llegado una información diferente por lo que cada uno ve una realidad diferente.
- Carga del servidor: La carga de servidor se refiere a la capacidad del servidor para procesar la información de los jugadores. Si el servidor está sobrecargado, puede haber retrasos en la transmisión de información y puede afectar a la experiencia de juego de los jugadores.

  Este problema no es tan habitual ya que los servidores suelen estar preparados para el tráfico normal de personas que suele tener, pero en ocasiones donde hay evento *online* como los de `Fortnite` por ejemplo, se conectan más personas de lo habitual. Esto es algo común es los lanzamientos de los videojuegos del género MMO, donde durante las primeras semanas suele haber colas de hasta horas para poder jugar debido al gran número de personas que están intentando entrar.
- Falta de soporte para multiplataforma: Se refiere a la dificultad de hacer que los jugadores de diferentes plataformas (como PC y consolas) jueguen juntos en el mismo servidor. Si no hay soporte para plataformas cruzadas, puede haber problemas de conectividad y desincronización entre los jugadores.

  Este es uno de los problemas que la comunidad lleva pidiendo años que se arregle para poder jugar con cualquier persona.

Como hemos visto aún hay problemas que perjudican la experiencia multijugador, pero durante los últimos años se han desarrollado soluciones para combatirlos y reducir su repercusión negativa en los videojuegos.

2.4. Soluciones y técnicas específicas para el desarrollo de juegos multijugador

Como ya se ha visto a pesar de todos los avances que se han hecho en el mundo de los videojuegos multijugador y de *netcode* aún existen problemas que dificultan el desarrollo de los mismos, pero para hacer más fácil esta labor se han desarrollado una serie de técnicas para agilizar el proceso. Estas técnicas son soluciones genéricas para problemas genéricos y que a su vez por cómo están construidas evitan la aparición de posibles problemas. Estas soluciones se agrupan en diferentes secciones dependiendo del tipo de problema que abarca.

2.4.1. Sincronización de estado

La sincronización de estado tiene el objetivo de que todos los clientes tengan una réplica exacta del estado del juego. Esto se consigue a través del paso de mensajes, los cuales contienen información del estado del juego, y cada técnica usa estos mensajes de una forma diferente ya que cada técnica se adapta mejor a unas situaciones que otras ya que no son genéricas para todos los casos. Los principales tipos de sincronización son los siguientes:

- *Deterministic Lockstep*: *Desterministic lockstep* es una técnica de sincronización en la que para sincronizar el estado del juego solo se envía el input que ha realizado el jugador y este se replica en todas las instancias del juego para que realice la misma acción.

  La principal ventaja que tiene esta técnica es que el paquete donde se envía la información es muy ligero y no depende de lo complejo que sea el juego ya que con solo ese input ya se sincroniza todo el juego.

  Aunque parece que esta técnica es muy buena y eficiente no siempre se puede usar ya que es necesario que los inputs y el estado del juego sea determinista. Esto significa que cuando se realiza un input en el juego, siempre va a generar exactamente el mismo resultado, por lo que deja fuera de las posibilidades un gran abanico de videojuegos ya que cualquier videojuego que use físicas, velocidades, fuerzas o aceleraciones (entre otros factores) no generará exactamente el mismo resultado debido al time step, que puede variar entre diferentes instancias del juego.

  Pero a pesar de que muchos videojuegos no pueden optar a poder usar esta técnica, hay muchos otros que si pueden, y le beneficia mucho ya que en juegos como **`Age of empires`** se simulan cientos de tropas, que serían muy costoso de sincronizar

si no fuera porque es un videojuego determinista y solamente con saber el input ya se puede sincronizar el juego.

- *Snapshot Interpolation*: La técnica de *snapshot interpolation* consiste en capturar el estado del jugador y lo relacionado con él, y enviar esa información en el paquete para después hacerla llegar a todos los clientes e interpolar el último estado recibido con el nuevo. Esta técnica se puede usar en situaciones no deterministas, al contrario que la técnica de *deterministic lockstep*, pero como se puede intuir el peso de estos paquetes es mayor ya que hace falta pasar toda la información (posición, rotación, interacciones…) constantemente, al contrario que antes que solo se enviaba el input cuando había.

  La principal ventaja de esta técnica es que está todo perfectamente sincronizado dando igual la situación ya que si se pierde un paquete no pasa nada porque se interpolan el estado con el siguiente que llegue.

  Por otro lado, el gran inconveniente es la gran cantidad de datos que hay que sincronizar, que hace que se genere latencia (en inglés, *lag)* porque se congestiona la red. Pero para arreglar esto existe un concepto llamado *snapshot compression*, el cual consiste en un conjunto de técnicas que se usan para reducir la cantidad de datos que se envían en los paquetes para no saturar la red. Algunas de estas técnicas son evitar enviar datos innecesarios como por ejemplo si no ha cambiado el estado no enviar información, reducir el tamaño de las variables que se envían, entre otras técnicas.

- *State Synchronization*: La última técnica de sincronización es *state synchronization*, la cual es una mezcla de las dos anteriores ya que en esta enviamos tanto el input como el estado.

  Como ya hemos visto, enviar solo el input no funciona en todas las situaciones ya que hace falta que el videojuego sea determinista, pero es muy eficiente ya que el tamaño del paquete es muy ligero, y por otro lado hemos visto que pasar todo el estado es muy exacto pero muy pesado, por lo que *state synchronization* envía siempre el input (aunque el juego no sea 100% determinista) y de forma regular pero no continua el estado, lo que hace enviar paquetes ligeros la mayoría del tiempo con lo que el juego está casi sincronizado y cada X tiempo envía un paquete pesado para sincronizar el desfase que se ha creado replicando los *inputs*.

State synchronization usa la mejor parte de las técnicas que hemos visto anteriormente por lo que la hace una forma de sincronizar a los jugadores muy sólida.

2.4.2. Predicción de estado

Este conjunto de técnicas sirve para intentar predecir el siguiente estado evitando esperar el mensaje del servir, consiguiendo así una experiencia de juego más fluida.

- *Rollback Netcode*: *Rollback Netcode* es una técnica que es muy usada en los videojuegos de pelea como por ejemplo **`Street Fighter V`**, **`Dragon Ball FighterZ`** o **`Super Smash Bros`** debido a que en este tipo de juegos es muy importante el orden y momento en el que se ejecutan las acciones. Esta técnica consiste en predecir cual es la siguiente acción del oponente para que el *gameplay* sea más fluido y exacto. En el caso de que la predicción falle se hace un *rollback* unas fracciones de segundos atrás antes de la predicción para posteriormente realizar la acción correcta.
- *Client-side predicticon*: Esta técnica es similar a la anterior, pero se usa de una forma más genérica, es decir, el cliente predice cual va a ser el siguiente estado del resto de clientes, pero en el caso de que no haga el correcto hace una interpolación al estado correcto que ha recibido del servidor.

2.4.3. Protocolos

El siguiente conjunto de técnicas se centra en la forma en la que se envían los paquetes entre clientes y como se manejan. Hay diversas formas de enviar paquetes, a continuación, se encuentran las dos más importantes:

- TCP/IP: TCP/IP (*Transmission Control Protocol/Internet Protocol*) es un protocolo de comunicación orientado a conexión, lo que significa que establece una conexión entre dos dispositivos antes de transferir datos. TCP/IP garantiza la entrega de datos sin errores, ya que utiliza un sistema de confirmación de paquetes y retransmisión de paquetes perdidos. Esto hace que TCP/IP sea una buena opción para aplicaciones que requieren alta confiabilidad, pero también significa que puede haber una latencia más alta debido al tiempo que lleva establecer y confirmar la conexión.

- UDP: Por otro lado, UDP (*User Datagram Protocol*) es un protocolo de comunicación sin conexión, lo que significa que no se establece una conexión antes de transferir datos, en cambio, los datos se envían como "datagramas" a través de la red. UDP es más rápido que TCP/IP, ya que no requiere confirmación de paquetes ni retransmisión de paquetes perdidos, lo que significa que hay menos latencia. Sin embargo, esto también significa que los datos pueden perderse o llegar en un orden diferente al que fueron enviados, lo que puede causar problemas en aplicaciones que requieren una alta fiabilidad.

En resumen, TCP/IP es más lento, pero más confiable, mientras que UDP es más rápido, pero menos confiable. En los videojuegos multijugador, a menudo se utiliza una combinación de ambos protocolos, con UDP utilizado para la transmisión en tiempo real de datos de posición de jugadores, disparos y otros eventos rápidos, y TCP/IP utilizado para la comunicación más lenta pero más confiable de información como el inicio de sesión y otros datos del juego.

2.4.4. Arquitecturas

Las arquitecturas en *netcode* es la estructura de red que se usa para gestionar la comunicación entre los diferentes clientes de un videojuego. Se trata de un conjunto de patrones y técnicas utilizados para dividir la carga de trabajo y distribuir la información necesaria para mantener una experiencia de juego fluida y sincronizada para todos los jugadores. Existen diversas arquitecturas de *netcode*, cada una con sus propias ventajas y desventajas. Algunas de las arquitecturas más usadas son:

- Cliente-servidor: La arquitectura cliente-servidor se utiliza en juegos en los que los jugadores se conectan a un servidor centralizado que gestiona la partida. El servidor es el encargado de sincronizar el estado del juego y de enviar la información a los clientes, y estos de enviarle al servidor los inputs que han realizado. Esta arquitectura suele ser más estable, ya que el servidor actúa como árbitro y puede detectar trampas o comportamientos maliciosos de los jugadores.
- P2P: En la arquitectura *P2P* los jugadores se conectan directamente entre ellos sin la necesidad de un servidor centralizado. Cada jugador es responsable de sincronizar su estado de juego con el de los demás jugadores. Esta arquitectura suele ser más escalable y menos costosa, pero también es más susceptible a problemas de seguridad y puede ser difícil de gestionar si hay muchos jugadores.

- Híbrida: Esta arquitectura combina elementos de las dos anteriores. Por ejemplo, puede haber un servidor centralizado que gestione la partida, pero que delegue la sincronización del estado del juego en los propios clientes. O puede haber un grupo de jugadores que actúen como servidores entre ellos, mientras que otros se conectan directamente a un servidor centralizado.

2.5. Casos de estudio

Una vez ya hemos visto toda la historia de los videojuegos multijugador y de su estado actual vamos a analizar brevemente algunos ejemplos de videojuegos multijugador actuales y como aplican las técnicas que hemos visto para el correcto funcionamiento del apartado multijugador:

2.5.1. League of Legends

League of Legends es uno de los videojuegos *online* más jugados de la actualidad contando con más de 10 años de trayectoria y millones de jugadores. Riot Games, empresa desarrolladora de **League of Legends**, en 2015 quiso poner freno a los problemas de red ya que el multijugador funcionaba correctamente, pero debido a la latencia no era tan bueno como podía ser, por lo que se pusieron a investigar para intentar resolver el problema. Después de esta investigación concluyeron que internet no estaba preparado para aplicaciones en tiempo real porque si tienes que esperar 1 segundo para que cargue una página web no pasa nada, pero en cambio en un videojuego en 1 segundo suceden muchas acciones por lo que es importante sincronizar el estado lo más rápido posible. Por cómo funciona internet le da prioridad al enrutador (en inglés, *router)* que tiene el camino más corto, no al que tiene el camino más rápido, esta fue uno de los problemas que encontraron en cómo funciona internet, entre otros factores.

Debido a todos estos inconvenientes su solución fue crear su propio internet, en el que no haya retrasos, se calcule la ruta más óptima para cada momento, menor pérdida de paquetes y sin depender de factores externos.

Ilustración 24 League of Legends

2.5.2. Valorant

Valorant es uno de los FPS más famosos de la actualidad, entrando dentro del subgénero de los *tactical shooters*. **Valorant** también pertenece a la empresa Riot Games, por lo que debido a que la salida de **Valorant** fue posterior a la de **League of Legends**, **Valorant** ya parte con el sistema que tiene **League of Legends** que se ha explicado antes.

El reto que ha tenido **Valorant** ha sido que, al ser un juego de disparos táctico, (en inglés, *taltical shooter*) es extremadamente importante la sincronización, para la cual han usado una arquitectura cliente-servidor con autoridad de servidor (en inglés, *server-authority)*, lo que significa que es el servidor quien maneja el estado de la partida y quien decide lo que sucede en la misma. Con esta técnica mitigan el problema que tiene este tipo de juegos, que son los hackers, ya que al ser el servidor quien toma las decisiones es más difícil hacer trampas. Por otro lado, usan la técnica de predicción en el lado del cliente (en inglés, *client side prediction)*, la cual consiste en que los clientes predicen el estado siguiente de la partida para disimular la latencia y en caso de que la predicción sea errónea porque no coincide con el servidor, debido a que este último es quien tiene la autoridad, se ajusta el estado del cliente para que sea el mismo que el servidor.

Esto junto al sistema de internet propio que hemos explicado anteriormente **Valorant** consigue una muy buena experiencia de juego y sincronización multijugador.

Ilustración 25 Valorant

2.5.3. Call of Duty

Call of Duty es una de las franquicias más grandes de videojuegos, donde anualmente lanzan una nueva entrega, pero en este caso vamos a hablar de la entrega llamada **Call of Duty Mordern Warfare** (la versión de 2019).

El funcionamiento de *netcode* en este videojuego es similar al de **Valorant**, teniendo una arquitectura cliente-servidor donde el servidor tiene la autoridad, pero los clientes hacer predicciones, al igual que en **Valorant**. En el caso de **Call of Duty** la predicción es diferente ya que tiene un buffer de acciones y en base a esa información predice cual va a ser el siguiente estado del juego, pero al igual que en el **Valorant**, si la predicción no es igual al estado que recibe del servidor, el estado del cliente es corregido.

Ilustración 26 Call of Duty

2.5.4. Conclusiones

Los casos de estudio analizados, **League of Legends**, **Valorant** y **Call of Duty Modern Warfare**, comparten algunas características comunes en cuanto a la aplicación de técnicas para el correcto funcionamiento del multijugador de los videojuegos:

- **Sincronización del estado:** Todos los juegos comprenden la importancia de la sincronización del estado del juego entre los diferentes jugadores. Reconocen que la latencia y los retrasos en la transmisión de datos pueden afectar negativamente la experiencia del juego. Por lo tanto, implementan estrategias para minimizar la latencia y asegurar una sincronización rápida y precisa del estado del juego en todos los clientes.
- **Arquitectura cliente-servidor:** Los juegos utilizan una arquitectura cliente-servidor, donde el servidor tiene la autoridad sobre el estado del juego y toma decisiones importantes. Esto ayuda a prevenir trampas y hackeos, ya que los clientes no pueden manipular directamente el estado del juego. Los clientes realizan predicciones locales para disimular la latencia, pero si estas predicciones difieren del estado recibido del servidor, el estado del cliente se corrige para mantener la coherencia con el servidor.

2.6. Conclusiones

Después de haber analizado todo el marco teórico que engloba el mundo de los videojuegos multijugador podemos sacar varias conclusiones:

- Progreso: Analizando la historia y desarrollo que han tenido los videojuegos multijugador en tan solo 65 años, se puede decir que es un ámbito de la tecnología que cada vez avanza más rápido y desde que nació solo ha mejorado, pasando de dos rayas y una pelota en un osciloscopio con mandos analógicos (**Tennis for Two**) a videojuegos *online* donde se pueden conectar miles de personas con una sincronización excepcional. El mundo de *netcode* no ha parado de crecer y hoy en día sigue haciéndolo y cada vez más rápido ya que cada vez se aplica a más campos como los profesionales para ayudar a la realización de otros trabajos como medicina.
- Potencial: Como ya se ha hablado, el crecimiento de *netcode* ha sido exponencial, y hoy en día lo sigue siendo. Debido a que cada vez se invierte más esfuerzo y dinero en este sector y más sectores están interesados, el futuro de *netcode* y los videojuegos multijugador es prometedor y repleto de avances. Gracias a todo el esfuerzo que se está poniendo en este sector en el futuro aparecerán nuevas técnicas y recursos que mejoraran el mundo de los videojuegos multijugador y su experiencia, y más teniendo en cuenta el reciente boom de las inteligencias artificiales, las cuales ayudan a mejorar software y a crearlo más rápido.

3. Diseño de videojuego Cubes Battle

3.1. Sinopsis

Cubes Battle es un trepidante videojuego del género arena de batalla en línea del estilo **Brawl Stars** que combina acción, estrategia y elementos de combate en un emocionante entorno de batalla en línea. Los jugadores se sumergirán en intensas y vertiginosas peleas, compitiendo contra otros jugadores en una lucha por la victoria.

3.2. Arte

El estilo visual de nuestro juego se basará en una combinación de arte 3D con *voxel art* y arte 2D simple y limpio. Esta combinación creará una estética distintiva y atractiva que se destacará en el género de juegos.

El entorno y los personajes en el juego se representarán con *voxel art* en 3D. El *voxel art* utiliza cubos en lugar de polígonos tradicionales para formar objetos y personajes. Esto le dará al juego un aspecto único y retro, recordando la estética de los juegos clásicos, al tiempo que aprovecha las capacidades y la flexibilidad del arte en 3D.

Además del arte en 3D con *voxel art*, el juego también contará con elementos visuales en 2D que seguirán un estilo simple y limpio. Estos elementos se utilizarán para la interfaz de usuario, los iconos, las ilustraciones y otros elementos visuales complementarios.

3.3. Controles

Acción	Input
Movimiento	W, A, S, D
Saltar	Tecla espaciadora
Esquivar	Control izquierdo
Interactuar con objetos del suelo	E
Atacar	Click izquierdo
Lanzar granada	Click derecho
Recargar arma	R
Cambiar a arma cuerpo a cuerpo	1
Cambiar a arma a distancia	2

3.4. Mecánicas del juego

Cubes Battle incorpora una serie de mecánicas que permiten a los jugadores enfrentarse entre sí en emocionantes combates. Algunas de las principales mecánicas incluyen:

Movimiento:

- Los jugadores podrán moverse utilizando las teclas “W”, “A”, “S” y “D” en el teclado.
- El personaje se desplazará en la dirección correspondiente a las teclas presionadas.

Salto:

- Los jugadores podrán hacer que su personaje salte presionando la barra espaciadora.
- El salto permitirá a los jugadores sortear obstáculos, alcanzar plataformas elevadas y evadir ataques enemigos.

Interacción con objetos en el suelo:

- Al acercarse a objetos en el suelo, los jugadores podrán interactuar con ellos presionando la tecla "E".
- Los objetos serán principalmente armas, y los jugadores podrán recogerlos para sustituir a los actuales.

Ataque:

- Los jugadores podrán atacar a los enemigos haciendo clic izquierdo del ratón.
- Al hacerlo, el personaje utilizará el arma equipada para dañar a los enemigos.

Recarga de armas:

- Los jugadores podrán recargar sus armas presionando la tecla "R".
- La recarga será necesaria cuando el cargador de un arma se haya vaciado por completo.

Cambio de arma:

- Los jugadores podrán cambiar entre el arma cuerpo a cuerpo y el arma a distancia presionando las teclas "1" y "2", respectivamente.
- Estas teclas permitirán una rápida adaptación a diferentes situaciones de combate.

Deslizamiento:

- Al presionar el control izquierdo mientras el personaje está en movimiento, podrá deslizarse por el suelo.
- Durante el deslizamiento, el personaje se moverá más rápido, lo que permitirá evadir ataques y desplazarse ágilmente por el entorno.

Granadas:

- Los jugadores podrán lanzar granadas presionando el clic izquierdo del ratón.
- Cada jugador podrá llevar hasta un máximo de 4 granadas a la vez.

Recolectar corazones y munición:

- Los jugadores podrán recuperar vida pasando por encima de corazones que se encuentren en el suelo.
- La munición necesaria para las armas podrá ser recolectada de la misma manera.

Sistema de vida y muerte:

- Si la vida del jugador llega a 0, el personaje morirá.
- Después de unos segundos, el jugador revivirá y podrá continuar jugando desde un punto de control.

Daño por balas y golpes enemigos:

- Cada vez que el jugador reciba un disparo o un golpe de un enemigo, su barra de vida disminuirá.
- Es importante evitar el daño enemigo y utilizar estratégicamente las armas y habilidades para mantenerse con vida.

3.5. Objetivos

Algunos de los objetivos a la hora de jugar **`Cubes Battle`** son los siguientes

1. Sobrevivir y acumular bajas: El objetivo principal del juego es sobrevivir y ser el jugador con más bajas en el campo de batalla. Los jugadores deberán enfrentarse a otros competidores y utilizar sus habilidades, estrategias y recursos disponibles para asegurarse de que son los que más bajas tienen.

2. Ganar experiencia: Otro objetivo es jugar mucho para para ser el jugador que mejor sabe usar el mapa a su favor para así tener más garantía de ganar una partida.
3. Mejorar: Como en cualquier videojuego competitivo es muy importante mejorar como jugador sabiendo usar mejor las armas y sabiendo como administrar los recursos disponibles.

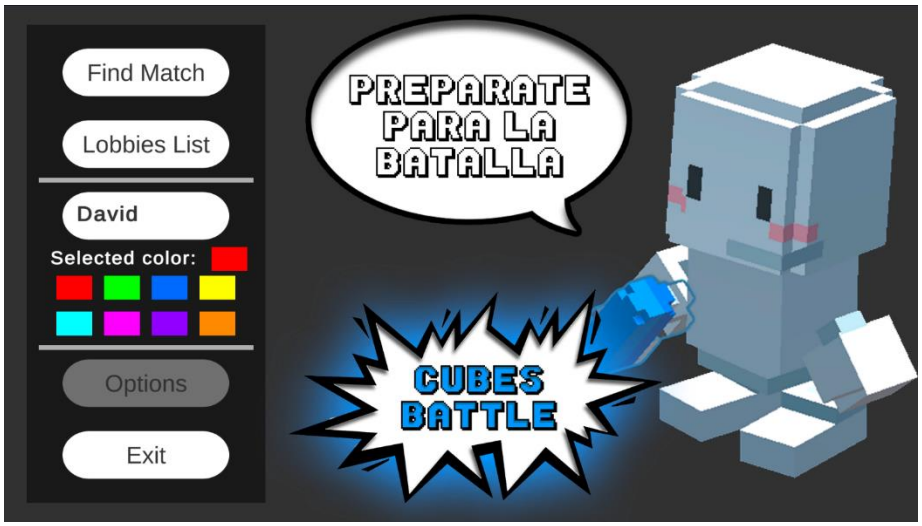


Ilustración 27 Cubes Battle - Menú de inicio

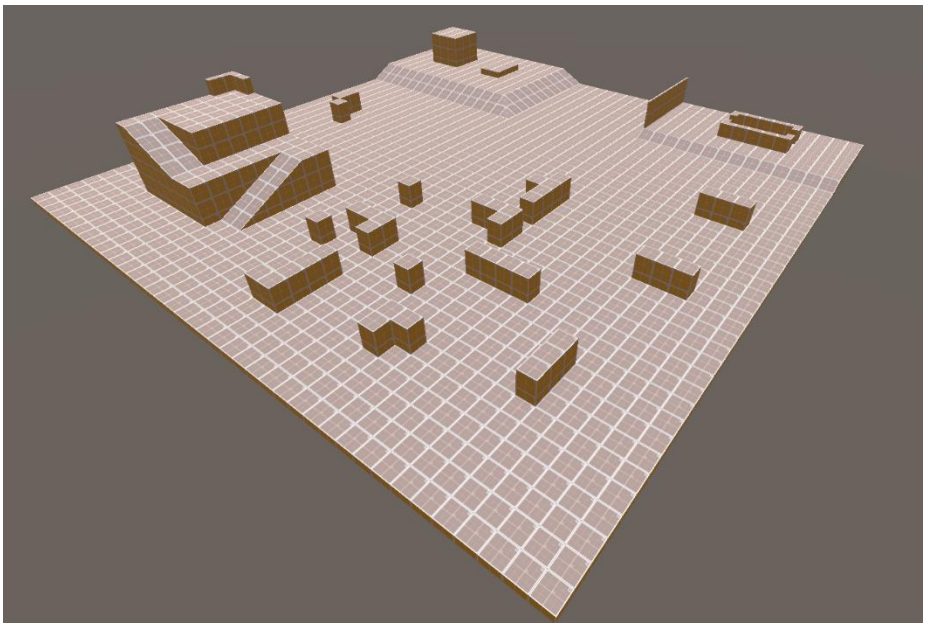


Ilustración 28 Cubes Battle - Esceanrio

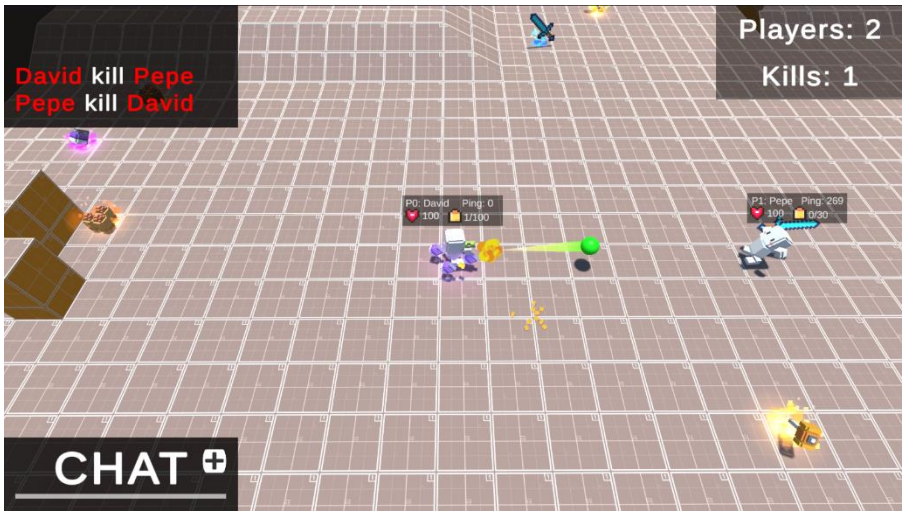


Ilustración 29 Cubes Battle - Ingame 1

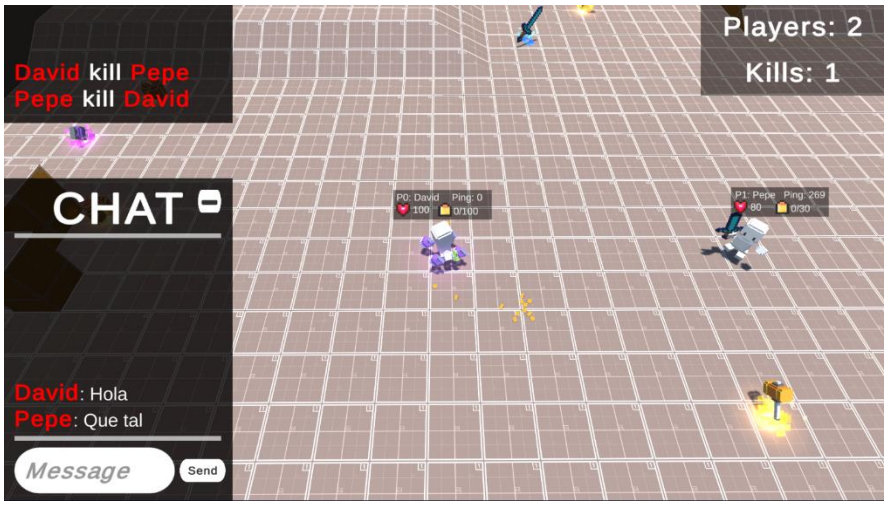


Ilustración 30 Cubes Battle - Ingame 2

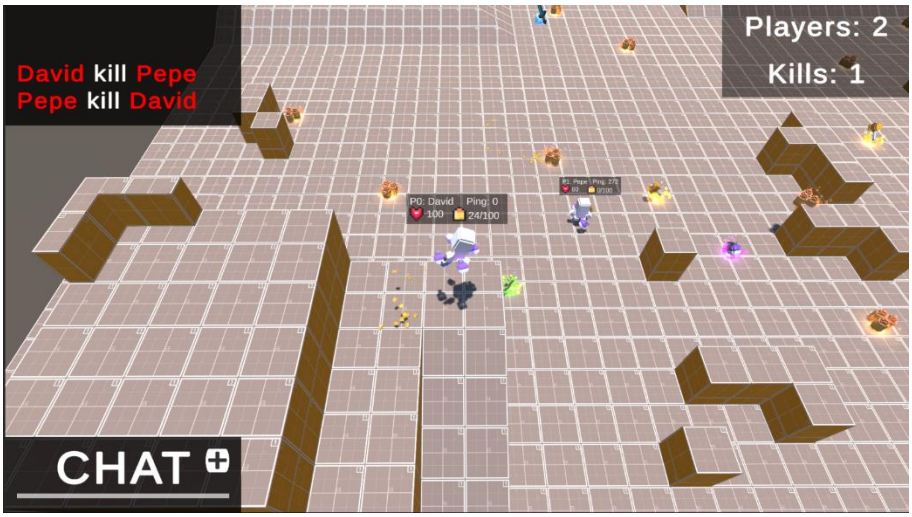


Ilustración 31 Cubes Battle - Ingame 3

3.6. Resumen

Cubes Battle es un videojuego multijugador en 3D con estética *voxel art* donde cada jugador puede moverse, saltar, esquivar, disparar, recargar o atacar con un arma de cuerpo a cuerpo. Cuando un jugador entra en partida puede moverse, saltar y esquivar, pero aún no puede interactuar con otros jugadores ya que no tienen armas, pero en el momento en que entra el mínimo de jugadores necesarios para iniciar la partida empiezan a aparecer armas tanto de cuerpo a cuerpo como a distancia que los jugadores podrán usar y empezar a combatir. Aparte de aparecer armas también aparecen objetos como munición, o vida extra. Cada jugador puede tener 1 arma cuerpo a cuerpo y un 1 arma a distancia.

Este videojuego al ser un prototipo no tiene el flujo completo de partidas por lo que cuando alguien muere está en modo fantasma durante unos segundos y después revive. Este juego cuenta con historial de muerte, recuento de muertes, lista de jugadores en partida y chat de texto durante la partida.

4. Descripción informática

4.1. Análisis general

NGO es una librería de alto nivel para el desarrollo de aplicaciones en entornos multijugador, abstrayendo al programador de la lógica *networking*. Esta herramienta te permite enviar mensajes a través de la red a una gran cantidad de usuarios. Esta herramienta ha sido la base de este proyecto, cuyo objetivo ha tenido diseñar y desarrollar un videojuego multijugador completamente funcional usando como base para el apartado del multijugador NGO, y por lo tanto Unity como motor de videojuegos, ya que NGO es una librería exclusiva de Unity. Teniendo esta base como objetivo, para desarrollar el proyecto se va a seguir una serie de pasos:

- Investigación: El primer paso para desarrollar este proyecto es investigar *netcode*, en que consiste, que herramientas existen, que técnicas se usan y que tipos de soluciones hay para cada tipo de videojuego. El objetivo de esta investigación es tener un amplio conocimiento sobre los videojuegos multijugador y *netcode* para así poder empezar a diseñar el resto de proyecto y como va a funcionar. Con esta investigación no solo se busca saber el estado actual de *netcode*, sino también saber cuáles son las técnicas para cuando se tenga el prototipo offline del videojuego saber cuáles son las mejores, implementarlas, compararlas y quedarse con la mejor.
- Prototipo: Después de haber realizado la investigación, el siguiente paso ha sido diseñar el videojuego base, es decir, aunque el tema del proyecto es el ámbito multijugador y más en específico *netcode*, primero hace falta un videojuego base offline para poder adaptarlo a *online* y explorar el mundo *netcode*. Para elegir como va a ser el videojuego base se buscará un tipo de videojuego donde el multijugador sea importante y *netcode* tenga mucho peso en él, por lo que se elegirá un tipo de videojuego que tenga mucho movimiento y diferentes elementos que sincronizar para que la implementación de *netcode* suponga un reto. La elección de las mecánicas del videojuego base se hará basándose en el conocimiento obtenido anteriormente para conseguir un proyecto donde *netcode* sea el principal factor.
- Implementación: El siguiente paso después de tener el prototipo offline es convertirlo a *online* usando NGO. Para hacer esta implementación de la parte multijugador se van a elegir las 3 técnicas que mejor se adapten al prototipo para

implementarlas, compararlas y elegir cual es la que mejor se adapta al videojuego, para posteriormente desarrollarla en profundidad y mejorarla para que la experiencia de usuario sea la mejor posible incluso en condiciones desfavorables.

- Mejorar *netcode*: Una vez tenemos implementada la técnica que mejor que adapte al videojuego, hay que adaptarla y optimizarla para el caso específico del videojuego para que funcione correctamente. En esta fase no solo se va a adaptar al videojuego, sino que también se va a llevar al límite la implementación de *netcode* forzando situaciones desfavorables como añadir latencia o pérdida de paquetes para comprobar que el videojuego va a funcionar correctamente incluso en las peores situaciones.
- Validación: Después de tener el proyecto completamente funcional y optimizado se llega a una fase en la que hay que validar que todo funcione correctamente a través de *tests*. Los *tests* son una buena práctica de programación ya que sirven para comprobar el correcto funcionamiento del código en diferentes situaciones que puedan suceder durante el uso del programa. En el caso de este proyecto algunos *tests* son validar que el programa detecta correctamente cuando se conecta un nuevo jugador y bajo qué condiciones.

  Al terminar esta fase el proyecto ya estará completo será completamente funcional y estará ausente de errores.
- Usabilidad: Aunque esta fase no está relacionada con *netcode*, que es el centro del proyecto, es importante que cualquier tipo de software cumpla unos mínimos de usabilidad, es decir, que sea fácil e intuitivo de usar. La usabilidad en un proyecto software se refiere a la facilidad de uso de la aplicación o software por parte de los usuarios, es decir, se busca que el diseño y la funcionalidad del software sean intuitivos, sencillos y eficientes para que los usuarios puedan interactuar con él de manera cómoda y sin dificultades.

4.1.1. Requisitos funcionales

RF1. Los jugadores deben poder crear un *lobby*.

RF2. Los jugadores deben poder acceder a una lista de los lobbies públicos que hay.

RF3. Los jugadores deben poder unirse a un *lobby* pública a través de la lista de *lobbies*.

RF4. Los jugadores deben poder unirse a un *lobby* mediante el código de lobby, ya sea pública o privada.

RF5. Los jugadores deben poder elegir un nombre asociado a su personaje.

RF6. Los jugadores deben poder elegir un color asociado a su personaje.

RF7. Los jugadores deben poder salir de un *lobby* y unirse a otro.

RF8. Los personajes deben poder interactuar con el resto de los personajes de la misma partida.

RF9. Los jugadores deben poder ver una lista de los jugadores de la partida.

RF10. Los personajes deben poder interactuar con los elementos del entorno.

RF11. Los personajes deben poder moverse.

RF12. Los personajes deben poder saltar.

RF13. Los personajes deben poder atacar.

RF14. Los personajes deben poder disparar.

RF15. Los personajes deben poder recargar.

RF16. Los personajes deben poder cambiar de arma.

RF17. Los jugadores deben poder comunicarse a través de un chat de texto dentro de la partida y durante la partida.

3.1.2. Requisitos no funcionales

RNF1. El estado de la partida debe estar sincronizada.

RNF2. El juego debe estar ausente de *lag*, y en caso de que no dependa del juego debe mitigar su efecto lo máximo posible.

RNF3. Las acciones de cada jugador deben ser ejecutaba en todos los clientes.

RNF4. Los jugadores deben revivir automáticamente después de morir al cabo de un tiempo.

RNF5. El juego debe tener sonido.

RNF6. El juego debe tener una interfaz simple y fácil de usar.

RNF7. Los jugadores deben poder saber el número de bajas que llevan.

RNF8. El juego debe tener una arquitectura escalable que permita agregar nuevas características, personajes o mapas en futuras actualizaciones.

RNF9. El juego debe tener una respuesta rápida a las acciones de los jugadores, minimizando la latencia y proporcionando una jugabilidad fluida.

4.1.3. Herramientas y tecnologías usadas

A continuación, se va a explicar las herramientas que se han usado para el desarrollo de este proyecto. Para cada una se va a explicar que es y como se ha usado en el proyecto:

- Unity 2020.3.33f1: Este motor de videojuegos multiplataforma desarrollado por Unity Technologies. Es una herramienta muy popular utilizada por programadores, artistas y diseñadores para crear videojuegos y experiencias interactivas en 2D y 3D. Unity ofrece una amplia gama de herramientas y características, incluyendo una interfaz de usuario intuitiva, un motor de física, un sistema de animación y un lenguaje de scripting basado en C#. Admite la creación de videojuegos para varias plataformas, como ordenador, consolas, dispositivos móviles y realidad virtual.

  Unity ha sido la herramienta base sobre la que se ha cimentado el proyecto ya que es el motor de videojuegos donde se ha desarrollado el videojuego. Algunas de las herramientas que se explican a continuación son librerías, recursos (en inglés, *assets)* o *plugins* de Unity, que se han añadido al mismo para cubrir algunas faltas que tiene Unity.

  La versión de Unity que se ha usado ha sido la 2020.3.33f1.

Ilustración 32 Logo Unity

- Visual Studio 16.10.3: Este entorno de desarrollo integrado (IDE, por sus siglas en inglés) creado por Microsoft, es una herramienta de software que permite a los programadores y desarrolladores crear, depurar y mantener aplicaciones de software en múltiples lenguajes de programación, como C++, C#, Visual Basic, JavaScript y otros. Es ampliamente utilizado en la industria de software y es considerado uno de los IDE más populares y completos del mercado.

  Visual Studio ha sido el editor de código que se ha usado para toda la parte de programación, la más importante de todo el proyecto.

La versión de Visual Studio que se ha usado ha sido la 16.10.3.

Ilustración 33 Logo Visual Studio

- NGO 1.0.0: Esta ha sido la herramienta fundamental para este desarrollo, debido a que esta es una biblioteca de red de alto nivel diseñada para Unity que permite abstraerte de la lógica de la red. Te permite enviar *GameObjects* y datos del mundo a través de la red a muchos jugadores al mismo tiempo. Con *Netcode*, puedes enfocarte en construir tu juego en lugar de en los protocolos de bajo nivel y los marco de trabajo (en inglés, *frameworks)* de red.

  Como ya se ha comentado anteriormente NGO es el centro de todo el proyecto ya que es el que se ha usado para hacer toda la parte de paso de mensajes y sincronización de estados entre otras tareas.

  La versión de NGO que se ha usado ha sido la 1.0.0.

Ilustración 34 Logo Netcode for GameObjects

- *Probuilder* 4.5.2: Este *asset* es un conjunto de herramientas de modelado 3D integradas en Unity que permite a los desarrolladores crear rápidamente prototipos, diseños y niveles de juego de manera eficiente. Con *ProBuilder*, los desarrolladores pueden crear, editar y personalizar objetos 3D directamente en la vista de la escena de Unity, sin tener que cambiar a otra herramienta de modelado 3D.

Esta herramienta se ha usado para crear el escenario del videojuego ya que al no ser una parte importante del proyecto simplemente se ha creado un escenario simple con formas simples como cubos y rampas.

La versión de ProBuilder que se ha usado ha sido la 4.5.2.

- *ProGrids* 3.0.3-preview.6: Este *asset* es un conjunto de herramientas de Unity que proporciona una rejilla de escena personalizable y herramientas de manipulación de objetos. Con *ProGrids*, los desarrolladores pueden alinear fácilmente objetos en la escena de Unity y editarlos con precisión. *ProGrids* incluye una variedad de opciones de configuración, como el tamaño y la división de la rejilla, y también ofrece la posibilidad de mostrar u ocultar la rejilla según sea necesario. Esto facilita el trabajo de diseño y la creación de niveles para juegos y aplicaciones en Unity.

  ProGrids es una herramienta que se complementa con *ProBuilder* ya que permite mover más fácilmente los elementos del escenario ofreciendo herramientas que agilizan el trabajo.

  La versión de ProGrids que se ha usado ha sido la 3.0.3-preview.6.

- *In-game Debug Console*: Gracias a este *asset* de Unity que está disponible en la *Unity Asset Store* que proporciona una consola de depuración en tiempo real dentro del juego. Esta herramienta permite a los desarrolladores y diseñadores depurar y ver la consola en tiempo real sin necesidad de estar en el editor de Unity o de salir del juego o de volver a compilar el código. La consola de depuración se puede personalizar para mostrar la información específica del juego que se desea. Esto hace que la depuración del juego sea más rápida y eficiente, lo que a su vez puede acelerar el proceso de desarrollo.

  In-game Debug Console es un *asset* de la *Unity Asset Store* que permite ver la consola de Unity en la *build* del juego (ejecutable del juego). De normal la consola solo se puede ver en el editor de Unity pero se ha usado este *asset* para poder verla en todas las instancias del juego y poder encontrar los errores más fácilmente.

- Photon 2.0.1: Este servicio en la nube ofrece una solución de red para juegos multijugador en tiempo real. Photon permite a los desarrolladores de juegos crear juegos multijugador en línea con una conexión sólida y sin problemas a través de una red de servidores dedicados. Con el plugin de Photon Unity, los desarrolladores de juegos pueden integrar fácilmente la solución de red en sus

juegos Unity, lo que les permite crear juegos multijugador en línea escalables y personalizables sin la necesidad de desarrollar su propia infraestructura de red. Además, Photon también proporciona herramientas para la administración del servidor y análisis de datos en tiempo real para los desarrolladores de juegos.

Photon es un *asset* de Unity que facilita el desarrollo de videojuegos multijugador, pero en este proyecto solo se ha usado la parte de la infraestructura en red que ofrece ya que el resto se ha hecho con NGO.

La versión de Photon que se ha usado ha sido la 2.0.1.

Ilustración 35 Logo Photon

- *Unity Gaming Services*: Este conjunto de herramientas es una plataforma en línea que proporciona herramientas y servicios para desarrolladores de juegos de Unity. Ofrece soluciones para la gestión de jugadores, la monetización de juegos, el análisis de datos y la integración de redes sociales, todo dentro del ecosistema de Unity. Con *Unity Gaming Service*, los desarrolladores pueden acceder a herramientas y recursos para crear y lanzar sus juegos, así como para mejorar la experiencia del usuario y aumentar su alcance. También ofrece servicios en la nube, lo que permite a los desarrolladores crear juegos en línea y multijugador sin tener que preocuparse por la administración de servidores y el escalado de infraestructura.

  Este servicio ofrece un abanico de herramientas, pero para este proyecto se han usado solo dos:

  - Unity *Relay* 1.0.5: Este servicio proporcionado por Unity que permite la conexión de jugadores en red de manera segura y confiable mediante un túnel de conexión en la nube. Funciona como un intermediario entre los jugadores para ayudar a superar los problemas de conectividad, tales como problemas de red o cortafuegos, y para mantener las conexiones de red seguras y estables. El servicio de Unity *Relay* es especialmente útil en situaciones en las que la conexión directa de jugador a jugador es difícil de establecer, como en juegos multijugador en línea que involucran a

jugadores de todo el mundo con diferentes tipos de conexión a internet. Unity *Relay* es una función disponible en el paquete de *UGS*, que se puede utilizar en cualquier plataforma de juego que utilice Unity, incluyendo PC, consolas y dispositivos móviles.

Unity *Relay* se ha usado para la gestión de conexiones. Este es parecido a Photon pero por un lado es oficial de Unity y por otro tiene sinergia con la otra herramienta que sea ha usado de *UGS*.

La versión de Unity Relay que se ha usado ha sido la 1.0.5.

- Unity Lobby 1.0.3: Esta herramienta permite a los desarrolladores crear fácilmente lobbies y salas de juego *online* para sus juegos multijugador. Con Unity Lobby, los jugadores pueden buscar y unirse a partidas *online*, y los desarrolladores pueden controlar el flujo de jugadores y administrar las partidas en línea. Unity Lobby también proporciona herramientas para la integración de búsqueda de pareja (en inglés, *matchmaking)* y la creación de partidas personalizadas. Unity Lobby es una función disponible en el paquete de *UGS*, para facilitar el desarrollo de juegos multijugador en línea.

  Unity Lobby se ha usado para permitir tener más de un *lobby* y poder gestionarlos fácilmente, aparte de permitir crear salas y unirse a ellas muy fácilmente.

  La versión de Unity Lobby que se ha usado ha sido la 1.0.3.

Ilustración 36 Logo Unity Gaming Services

- Trello: Para la gestión de proyectos y tareas se ha usado Trello. Permite crear tableros virtuales donde se pueden agregar tarjetas con tareas, listas de control, comentarios, adjuntos y otros elementos relacionados con el proyecto. Los usuarios pueden asignar tareas a otros miembros del equipo, establecer plazos,

etiquetas y otros metadatos para organizar y priorizar el trabajo. Trello es ampliamente utilizado para el desarrollo de software, la gestión de proyectos, la planificación de eventos y otras tareas de colaboración.

Trello se ha usado para seguir el desarrollo de tareas y avance del proyecto.

Ilustración 37 Logo Trello

- Microsoft Project: Esta herramienta es un software de gestión de proyectos que permite a los usuarios planificar, programar, asignar recursos, hacer un seguimiento del progreso y gestionar el presupuesto de un proyecto. Con Microsoft Project, los usuarios pueden crear diagramas de Gantt, asignar tareas, establecer dependencias y colaborar con otros miembros del equipo. Se utiliza comúnmente en una variedad de industrias, desde la construcción y la ingeniería hasta la tecnología de la información y el desarrollo de software.

  Esta herramienta se ha usado para la gestión del proyecto, permitiendo ver el avance de las tareas y como se relacionan entre sí.

Ilustración 38 Logo Microsoft Project

- Clumsy 0.3 RC4: Esta herramienta es gratuita y de código abierto, y se utiliza para simular diferentes condiciones de red y probar la capacidad de un programa o aplicación para manejarlas. Con Clumsy, los desarrolladores pueden emular

conexiones de red lentas, pérdida de paquetes, latencia y otros problemas comunes de la red para verificar como su software responde en situaciones de la vida real.

Como ya se ha comentado una de las fases del proyecto es comprobar que funcione en situaciones desfavorables y mejorarlo, y para poder hacerlo se ha usado esta herramienta que permite crear estas situaciones.

La versión de Clumsy que se ha usado ha sido la 0.3 RC4.

- Github y github destkop: Esta plataforma en línea proporciona alojamiento para repositorios de control de versiones basados en Git. Es una herramienta muy popular para colaborar en proyectos de software y para el control de versiones de código fuente. Los usuarios pueden crear repositorios para sus proyectos, subir y compartir su código con otros desarrolladores, y colaborar en proyectos con otras personas.

  GitHub Desktop es una aplicación de escritorio que facilita el uso de GitHub sin la necesidad de usar la línea de comandos. Permite a los usuarios clonar repositorios, crear ramas, fusionar cambios y publicar su trabajo en la plataforma de GitHub.

  Ambos, GitHub y GitHub Desktop, funcionan en conjunto para facilitar la colaboración y el control de versiones en proyectos de software. Los usuarios pueden usar GitHub para alojar su código fuente y colaborar con otros desarrolladores en el mismo proyecto, y utilizar GitHub Desktop para clonar el repositorio, realizar cambios en el código, crear y fusionar ramas, y publicar el trabajo realizado en la plataforma de GitHub.

  Estas dos herramientas se han usado en conjunto para tener un control sobre las versiones del juego.

  La versión de Github Destkop que se ha usado ha sido la 3.2.3.

Ilustración 39 Logo Github

Ilustración 40 Logo Github Desktop

4.2. Diseño

4.2.1. Diagramas

4.2.1.1. Diagrama de flujo

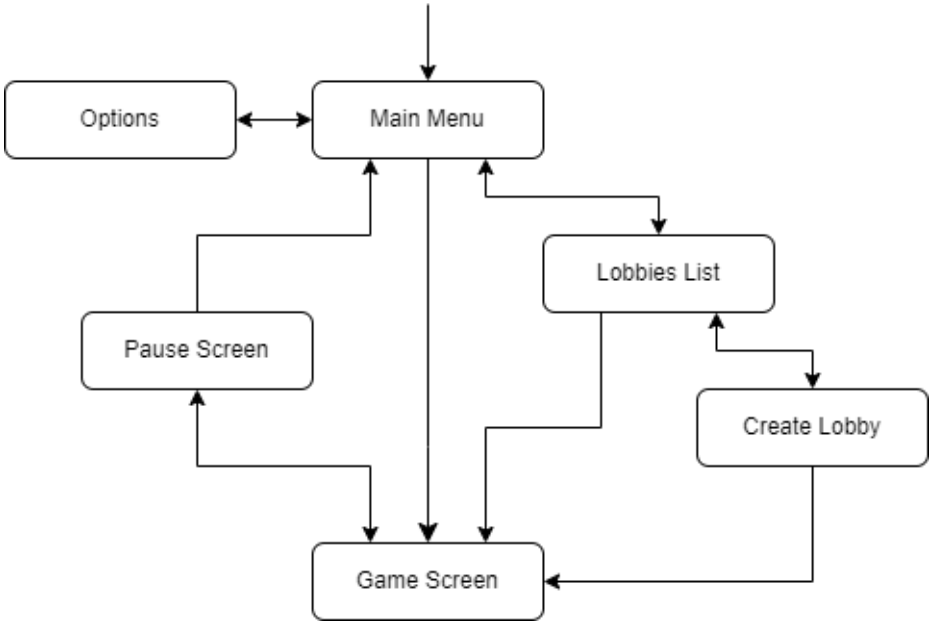


Ilustración 41 Diagrama de flujo

En la ilustración 36 se puede ver cómo se puede navegar por las diferentes pantallas del videojuego. La pantalla que aparece al entrar al juego es la que se llama “Main Menu” desde la cual se puede por un lado introducir el nombre y el color del personaje, y por otro lado se puede acceder al menú de opciones para configurar las preferencias del juego, a la pantalla de Lobbies llamada “Lobbies List” y también se puede acceder a la pantalla de juego llamada “GameScreen” en la cual se desarrolla la partida.

Durante la partida se podrá acceder a la pantalla de pausa en la que se podrá ver el código de la sala, los jugadores en la partida y te permitirá salir de la partida.

Por último, desde la pantalla de "Lobbies List", a la cual se ha accedido de la pantalla "MainMenu", se podrá acceder a una partida a través de la lista de lobbies disponibles en pantalla o a través del código de la sala. Otra funcionalidad que tiene esta pantalla es poder acceder a la pantalla de crear lobbies, la cual se llama "Create Lobby" y desde la cual se puede configurar y crear un nuevo *lobby*.

4.2.1.2. Diagrama de clases

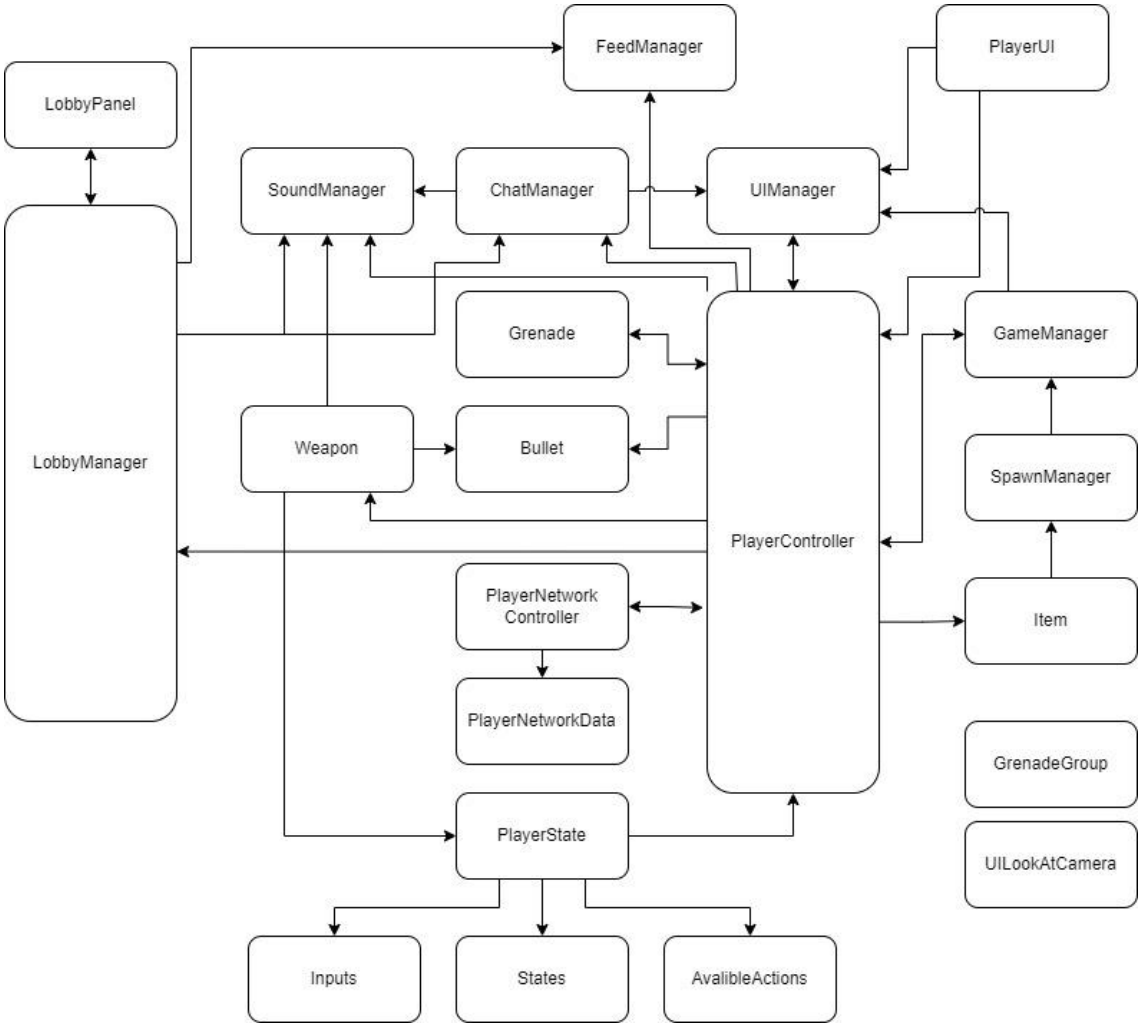


Ilustración 42 Diagrama de clases reducido

En la ilustración 37 se puede ver el diagrama de clases que tiene este proyecto donde cada clase es un script. Gracias a las flechas que conectan las diferentes clases podemos ver las dependencias que hay entre ellas.

Para ver más en detalle el diagrama ir al ANEXO III

Estas clases se pueden dividir en 3 grupos dependiendo de su función:

- *Gameplay*: Estas clases son las que se encuentran en la columna central (en la parte media y baja de la ilustración 37), y se encargan de las mecánicas del juego. Estas clases son: `PlayerController`, `Weapon`, `Bullet`, `Grenade`, `PlayerNetworkController`, `PlayerNetworkData`, `PlayerState`, `Inputs`, `States`, `AvalibleActions`, `Items`, `GrenadeGroup` y `UILookAtCamera`.
- Gestión: Estas clases se encargan de gestión el flujo del juego y la partida y se encuentran en la parte derecha y superior de la ilustración 37. Estas son: `GameManager`, `SpawnManager`, `PlayerUI`, `UIManager`, `FeedManager`, `ChatManager` y `SoundManager`.
- Lobby: Estas clases se encuentran a la izquierda de las imágenes y se encargan de la gestión de los *lobbies*. Estas clases son `LobbyManager` y `LobbyPanel`.

4.2.1.3. Diagrama de secuencia

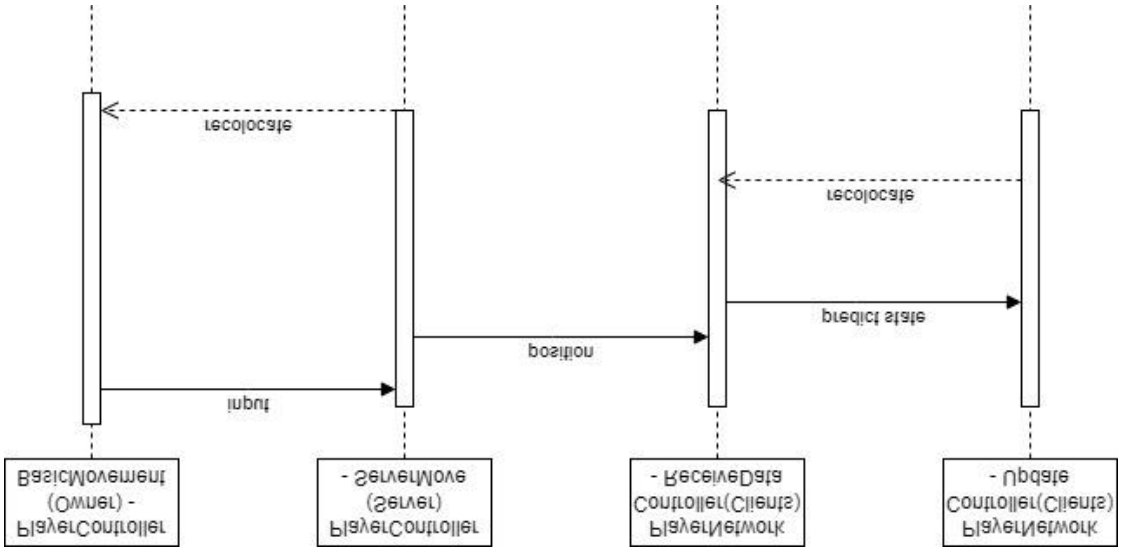


Ilustración 43 Diagrama de secuencia

Como se puede ver en la ilustración 38 la secuencia que se lleva a cabo para sincronizar a los jugadores (de forma simplificada) es que el jugador realiza una acción la cual es ejecutada y también es enviada al servidor, que se encarga de realizarla, y enviarla el estado resultante al resto de clientes para que actualicen el estado y al también se envía al propio propietario (en inglés, *owner*) para que corrija su estado.

4.1.1.4. Casos de uso

En este apartado se presentan una serie de casos de uso y el procedimiento que el usuario seguiría para llevarlos a cabo, las condiciones previas que deben cumplirse y el resultado

en caso de éxito, así como algunas consecuencias adicionales en caso de cometer un error durante el proceso.

UC1: Abrir el videojuego

Actor principal: Usuario.

Precondiciones: Tener la carpeta del juego en el ordenador.

Garantías de éxito: Abrir el juego.

Escenario principal de éxito:

1. Abrir la carpeta que contiene el juego.
2. Ejecutar el archivo Cubes Battle.exe.

UC2: Poder entrar en partida.

Actor principal: Usuario.

Precondiciones: UC1.

Garantías de éxito: Tener acceso a poder jugar.

Escenario principal de éxito:

1. Escribir un nombre en el campo donde pone “Player Name”.

UC3: Entrar en una partida de forma rápida.

Actor principal: Usuario.

Precondiciones: UC2.

Garantías de éxito: Entrar en una partida.

Escenario principal de éxito:

1. Hacer click en el botón que pone “Find Match”.

UC4: Entrar en una partida a un lobby específica.

Actor principal: Usuario.

Precondiciones: UC2.

Garantías de éxito: Entrar en una partida.

Escenario principal de éxito:

1. Hacer click en el botón que pone “Lobbies List”.
2. Hacer click en el botón que pone “Join” de alguno de los lobbies que aparece en la lista.

UC5: Entrar en una partida mediante código.

Actor principal: Usuario.

Precondiciones: UC2.

Garantías de éxito: Entrar en una partida.

Escenario principal de éxito:

1. Hacer click en el botón que pone “Lobbies List”.
2. Introducir el código de un lobby existente en el campo que pone “Lobby Code”.
3. Hacer click en el botón que pone “Join By Code”.

UC6: Crear un lobby.

Actor principal: Usuario.

Precondiciones: UC2.

Garantías de éxito: Entrar en una partida.

Escenario principal de éxito:

1. Hacer click en el botón que pone “Lobbies List”.
2. Hacer click en el botón que pone “Create Lobby”.
3. Introducir un nombre en el campo que pone “Lobby Name”.
4. Hacer click en el botón que pone “Create Lobby”.

UC7: Interactuar con otros jugadores.

Actor principal: Usuario

Precondiciones: UC3 o UC4 o UC5 o UC5.

Garantías de éxito: Jugar una partida.

Escenario principal de éxito:

1. Usar las teclas “W”, “A”, “W” y “D” para moverte.
2. Usar la tecla “Space” para saltar.
3. Usar el click izquierdo para disparar o atacar.

4.2.2. Técnicas usadas

Una de las partes más importantes del proyecto consiste en analizar y estudiar netcode y lo que ofrece actualmente para elegir las herramientas y técnicas más adecuadas para el videojuego. Antes de empezar dicha investigación hay que tener claro que se busca en estás técnicas ya que dependiendo del tipo de juego hay que realizar la investigación desde un punto de vista u otro, por lo que antes de empezarla se dejó claro que hacía falta un conjunto de técnicas que permitan la máxima precisión con el menor retraso (en inglés, *delay)* posible, por lo que teniendo en cuenta que el videojuego que se va a desarrollar va a trabajar con pocas entidades en red pero que hace falta que sea muy precisa su sincronización ya que en el videojuego al ser un *soother* con partidas de máximo 10 personas, hay que encontrar técnicas que nos ofrezcan eso. Después de dicha investigación se han elegido 3 técnicas de sincronización para probar en el proyecto y analizar sus ventajas y desventajas entre ellas para finalmente elegir la que mejor se adapta al juego. Estas tres técnicas son las siguientes:

- *Deterministic Lockstep*: La técnica de sincronización llamada *Deterministic Lockstep* es una forma muy eficiente de sincronizar el estado del juego. Con esta técnica, sólo se envía el input que ha realizado el jugador, y este se replica en todas las instancias del juego, asegurando que se realice la misma acción en cada una de ellas. Esta técnica es particularmente útil para juegos con un gran número de elementos y jugadores, ya que los paquetes que se envían son muy ligeros y no dependen de la complejidad del juego.

  Sin embargo, esta técnica no es viable para todos los tipos de juegos. Es necesario que tanto los inputs como el estado del juego sean deterministas, lo que significa que cada vez que se realiza una acción, se obtiene exactamente el mismo resultado. Esto hace que muchos juegos que utilizan físicas, velocidades, fuerzas o aceleraciones no puedan utilizar esta técnica, ya que el time step puede variar entre diferentes instancias del juego.

Aunque hay muchos juegos que no pueden utilizar la técnica de *Deterministic Lockstep*, hay otros que sí pueden aprovecharla enormemente. Por ejemplo, juegos como **Age of Empires**, donde se simulan cientos de tropas, pueden beneficiarse enormemente de esta técnica, ya que es determinista y, por lo tanto, se puede sincronizar el juego únicamente conociendo los inputs de los jugadores. En resumen, la técnica de *Deterministic Lockstep* es una herramienta muy útil para los desarrolladores de juegos que tienen la suerte de poder aplicarla.

Esta técnica es buena, pero en este proyecto no se puede aplicar ya que cuenta con elementos incompatibles con la misma como fuerzas.

- *Snapshot Interpolation*: La técnica de *Snapshot Interpolation* es una técnica de sincronización en la que se captura el estado actual del jugador, incluyendo su posición, rotación y otras interacciones, y se envía esa información en paquetes de red. Después, los clientes interpolan el último estado recibido con el nuevo estado para mantener la sincronización del juego. A diferencia de la técnica de *Deterministic Lockstep*, esta técnica se puede usar en situaciones no deterministas y es especialmente útil para juegos en los que los objetos tienen físicas complejas, lo que hace que la sincronización basada en input no sea una opción viable.

  Aunque la *Snapshot Interpolation* permite la sincronización perfecta del juego, tiene un inconveniente importante: la gran cantidad de datos que se deben sincronizar en cada paquete. Esto puede hacer que el juego experimente retrasos, conocidos como *lag*, debido a la congestión de la red. Para solucionar este problema, existe la técnica de *snapshot compression*, que incluye un conjunto de técnicas para reducir la cantidad de datos que se envían en cada paquete. Algunas de estas técnicas incluyen evitar enviar datos innecesarios, reducir el tamaño de las variables que se envían y otras medidas similares.

  Esta técnica es compatible con este proyecto, pero debido a que en ciertas situaciones estas interpolaciones no ofrecían una imagen real como atravesar paredes.

- *State Synchronization*: La técnica de *State Synchronization* es muy efectiva en juegos donde la precisión es importante pero el tamaño del paquete también es un factor crítico. Esta técnica combina lo mejor de las dos técnicas de sincronización anteriores: envía tanto el input del jugador como el estado del juego, pero de una forma más inteligente.

En lugar de enviar constantemente todo el estado del juego, *State Synchronization* solo envía actualizaciones de estado periódicas, lo que reduce significativamente el tamaño de los paquetes. Esto significa que el juego está casi sincronizado la mayor parte del tiempo, lo que mejora significativamente la experiencia del usuario, mientras que también reduce la carga en la red.

Sin embargo, enviar solo el input no funciona en todas las situaciones ya que hace falta que el videojuego sea determinista, por lo que *State Synchronization* envía el input y el estado para garantizar que el juego sea preciso y justo para todos los jugadores.

Además, esta técnica permite ajustar el intervalo de tiempo entre las actualizaciones de estado, lo que permite equilibrar la precisión y el ancho de banda en función de las necesidades del juego. En resumen, *State Synchronization* es una técnica de sincronización muy sólida que se adapta perfectamente a muchos tipos de juegos, proporcionando un equilibrio entre la eficiencia y la precisión en la sincronización del juego.

Después de implementar y testear estas tres técnicas la elegida fue *State Synchronization* ya que es la que mejor se adapta al juego debido a que lo mantiene completamente sincronizado sin consumir una gran cantidad de ancho de banda. Las otras dos opciones fueron desechadas ya que por un lado la técnica de *deterministic lockstep* al ser un juego no determinista no sincronizaba correctamente los clientes, pero se vio que apenas consume ancho de banda, y por otro lado, la técnica de *Snapshot Interpolation*, funcionaba correctamente con las mejoras aplicando *Snapshot Compression* pero después de implementar *State Synchronization* junto con las mejoras de *Snapshot Compression* se pudo observar un mejor rendimiento ya que se consigue un perfecta sincronización a la vez que se consigue ocupar poco ancho de banda.

4.3. Desarrollo

Durante este apartado se va a explicar cómo ha sido el desarrollo del proyecto, como ha avanzado a lo largo del tiempo y que problemas han surgido y con que soluciones se han sobrepasado. El desarrollo se divide en varios puntos dependiendo de la fase en la que el proyecto se encontrase:

4.3.1. Prototipo offline

El primer paso para desarrollar este proyecto ha sido tener un prototipo de videojuego sobre el cual se pueda trabajar para poder hacer pruebas de *netcode* y aplicar las diferentes técnicas que sean necesarias. Para realizar este prototipo primero hay que tener en cuenta que tiene que ofrecer suficientes oportunidades y mecánicas como para que permita aplicarle diferentes técnicas y que realmente sean relevante y se note su implementación a nivel de sincronización, por lo que antes de hacer cualquier prototipo es necesario informarse de que tipo de videojuego explotan más las capacidades de *netcode*.

Por lo tanto, antes de realizar el prototipo es necesario hacer una investigación previa sobre el mundo de *netcode* y como se encuentra actualmente para elegir el tipo de videojuego y sus mecánicas. Para conseguir esto se ha realizado una investigación cuyo objetivo ha sido recopilar información sobre las técnicas más usados en que tipo de videojuegos se aplican, para esto se ha explorado diversos géneros de los videojuegos, analizando que tipos técnicas usan y como las aplican. Después de analizar varios géneros se ha elegido el género *shooter* ya que usa las técnicas más populares (*Deterministic lockstep, Snapshot interpolation, State shyncronization*) y es un género que tiene un gran abanico de mecánicas útiles para explotar *netcode*. Dentro del género *shooter* se ha elegido el subgénero arena de batalla (estilo **Brawl Starts**) ya que por un lado es más accesible para más personas y por otro lado es más fácil comprobar el correcto funcionamiento de *netcode*.

Una vez decidido el tipo de juego se diseñaron las mecánicas para posteriormente implementar el prototipo en Unity. Este prototipo no solo son las mecánicas sino que para una mejor experiencia de usuario también cuenta con un sistema de sonido, interfaz, menús, entre otras funcionalidades. Una vez creado el prototipo con todos sus apartados lo único que faltaba era validar que todo funcionase correctamente para poder centrarse en la parte de *netcode*. Durante este proceso de validación se detectaron algunas mecánicas que se podían mejorar para ser más útiles aparte de encontrar y solucionar bugs.

4.3.1.1. El videojuego

El videojuego realizado se llama **Cubes Battle** y como ya se ha mencionado del género de videojuegos multijugador de arena de batalla en línea y se basa en el videojuego **Brawl Stars**.

Cubes Battle es un videojuego multijugador en 3D con estética *voxel* art donde cada jugador puede moverse, saltar, esquivar, disparar, recargar o atacar con un arma de cuerpo a cuerpo. Cuando un jugador entra en partida puede moverse, saltar y esquivar, pero aún no puede interactuar con otros jugadores ya que no tienen armas, pero en el momento en que entra el mínimo de jugadores necesarios para iniciar la partida empiezan a aparecer armas tanto de cuerpo a cuerpo como a distancia que los jugadores podrán usar y empezar a combatir. Aparte de aparecer armas también aparecen objetos como munición, o vida extra. Cada jugador puede tener 1 arma cuerpo a cuerpo y un 1 arma a distancia.

Este videojuego al ser un prototipo no tiene el flujo completo de partidas por lo que cuando alguien muere está en modo fantasma durante unos segundos y después revive. Este juego cuenta con historial de muerte, recuento de muertes, lista de jugadores en partida y chat de texto durante la partida.

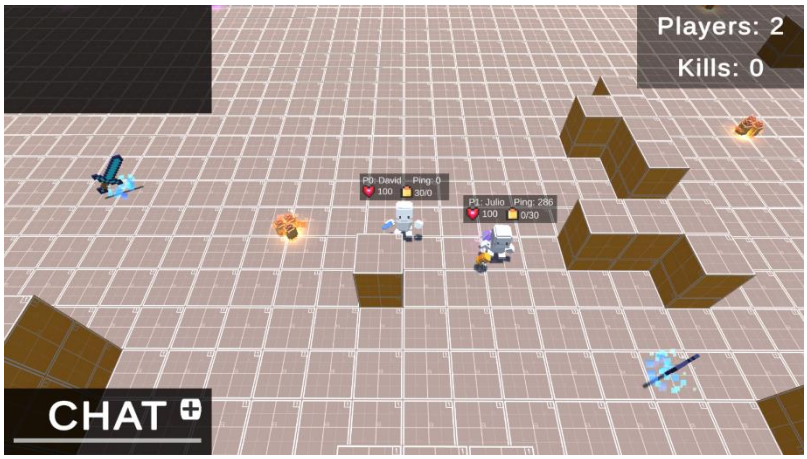


Ilustración 44 Cubes Battle

4.3.2. Deterministic lockstep

Una vez ya se tenía el prototipo al completo se empezó a implementar la parte de *netcode*. La primera técnica que se implementó fue *deterministic lockstep* (recordemos que se van a implementar las tres técnicas para compararlas y elegir la mejor).

La técnica de *Deterministic Lockstep* es un método de sincronización de juegos multijugador en tiempo real. Consiste en enviar solo la entrada del jugador al servidor, que la distribuye a todos los demás clientes. En lugar de enviar constantemente la información de estado del juego, el cliente ejecuta la entrada recibida de manera local, lo que hace que el juego sea completamente sincronizado entre todos los jugadores.

La principal ventaja de esta técnica es que el paquete de información que se envía es muy ligero y no depende de la complejidad del juego, lo que significa que se puede usar para juegos con muchos jugadores y eventos simultáneos. Además, dado que la entrada del jugador es la única información que se envía, la técnica es muy resistente a la pérdida de paquetes de datos.

Sin embargo, el uso de la técnica de *Deterministic Lockstep* requiere que el juego sea determinista. Esto significa que, para una entrada dada, el estado del juego debe ser el mismo en todos los clientes. Los juegos que implican física, velocidad, fuerza, aceleración u otros factores que pueden generar resultados ligeramente diferentes no son compatibles con esta técnica.

En cuanto al uso de esta técnica, se utiliza comúnmente en juegos de estrategia en tiempo real (RTS) como **`Age of Empires`** o **`Starcraft`**, que no implican una física compleja o resultados no deterministas. En estos juegos, el servidor de juego sincroniza los inputs de los jugadores y distribuye estos inputs a todos los clientes para que puedan replicar la misma acción y mantener la sincronización entre todos los jugadores.

Debido a que era la primera técnica que se implementó hizo falta crear la base aparte de la sincronización de personaje, la cual consistía en sincronizar variables y animaciones entre otras cosas.

Como ya se ha dicho anteriormente el videojuego es estilo **`Brawl Starts`**, por lo que hay disparos, movimientos, físicas, velocidades y otros elementos que depende del time step de cada instancia del juego, lo que lo hace que el juego sea no determinista, que provoca que esta técnica no se pueda aplicar a este juego ya que al ser un juego con tanto movimiento es muy fácil que se desincronice. Durante las pruebas que se realizaron después de la implementación se confirmó lo anteriormente dicho, se desincronizaba muy fácilmente, pero gracias a la posterior implementación de algunas técnicas para "sincronizar" el time step, se mejoró la experiencia, pero no era suficiente ya que después de un tiempo jugando se desincronizaba.

Después de pruebas y mejoras las conclusiones que se sacaron de esta técnica es que es muy ligera ya que no hubo ningún problema con la red debido a la ligereza de los paquetes, pero al ser un juego no determinista, incluso con las optimizaciones no fue posible hacerlo funcionar.

Como se puede ver en la Ilustración 45, el desfase de la posición entre un cliente y otro es considerable.

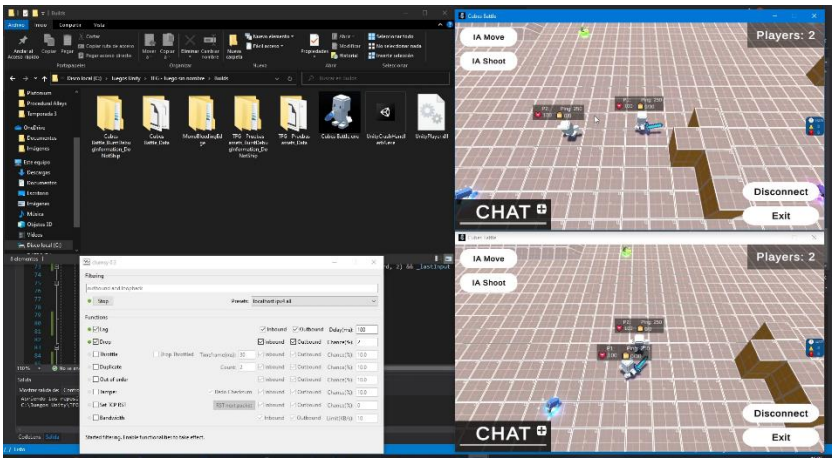


Ilustración 45 Cubes Battle - Deterministic Lockstep

4.3.3. Snapshot interpolation

Para esta parte se desactivo todo el código relacionado con *deterministic lockstep* pero se mantuvo el código de sincronización de variables, animaciones…

La técnica de *Snapshot Interpolation* es una técnica utilizada en programación de videojuegos y aplicaciones en red que se utiliza para sincronizar estados en tiempo real en diferentes dispositivos de juego. La técnica funciona mediante la captura de un estado del juego (en inglés, *snapshot*) y la interpolación entre dos *snapshots* para crear un estado más suave.

En esta técnica, el servidor captura snapshots y envía esta información a los clientes. Los clientes luego interpolan entre los *snapshots* recibidos para crear una transición suave entre los diferentes estados del juego.

La principal ventaja de esta técnica es que es útil en situaciones no deterministas, como en juegos en los que el comportamiento del personaje del jugador está influenciado por la física y otros factores que no se pueden predecir.

Sin embargo, la técnica también tiene algunas desventajas. Dado que los *snapshots* se toman en momentos regulares, puede haber una pequeña latencia entre los cambios en el juego y cuando se envía la información del *snapshot* al cliente. Esto puede hacer que los clientes experimenten pequeños retrasos o desincronizaciones en la acción del juego. Además, la técnica puede consumir una cantidad significativa de ancho de banda si se

utilizan *snapshots* de alta calidad, si se deben enviar *snapshots* con frecuencia o si se tienen que sincronizar muchos elementos.

Debido a que esta técnica está preparada para videojuegos no deterministas, los resultados al implementarla fueron mejor que con *determinstec lockstep* ya que los jugadores estaban perfectamente sincronizados en todo momento gracias a la interpolación. En las situaciones donde solo se tenían que sincronizar los jugadores el videojuego funcionaba perfectamente, sin embargo, cuando se tenían que sincronizar más elementos la red se congestionaba debido a la cantidad de paquetes y su peso por lo que había dos problemas, el primero era la congestión de red, la cual se solucionó usando *snapshot compression* reduciendo el peso de los paquetes además de reducir la frecuencia de envío, lo que provocó el segundo problema, al no enviar tantos paquetes había que mejorar la interpolación ya que no servía cualquiera. Después de hacer estas mejoras el resultado fue bastante bueno y ya se podía jugar sin problemas.

Como se puede ver en la Ilustración 46 el juego se sincroniza bien pero gastaba mucho recursos.

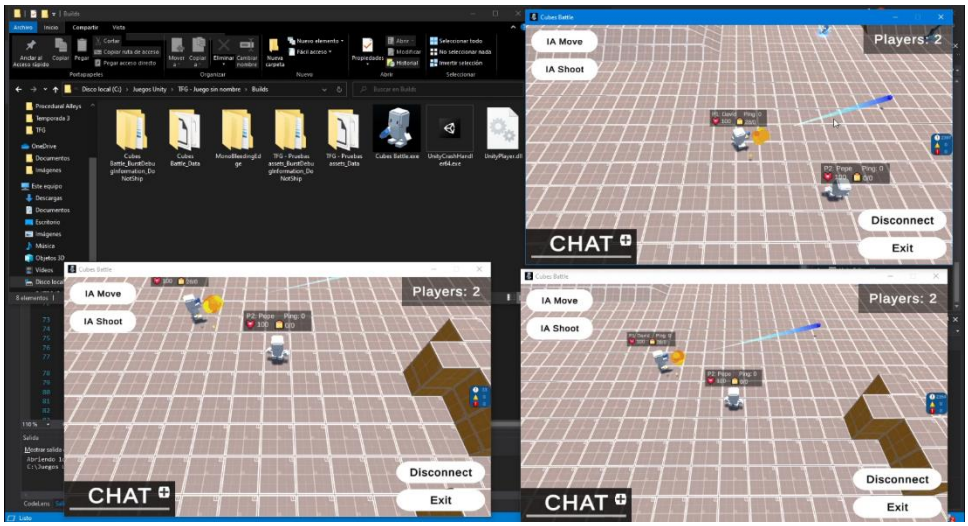


Ilustración 46 Cubes Battle - Snapshop interpolation

4.3.4. State shyncronization

Para esta parte se desactivo todo el código relacionado con *snapshot interpolation* pero se mantuvo el código de sincronización de variables, animaciones…

La técnica de *State Synchronization* es muy efectiva en juegos donde la precisión es importante pero el tamaño del paquete también es un factor crítico. Esta técnica combina

lo mejor de las dos técnicas de sincronización anteriores: envía tanto el input del jugador como el estado del juego, pero de una forma más inteligente.

En lugar de enviar constantemente todo el estado del juego, *State Synchronization* solo envía actualizaciones de estado periódicas, lo que reduce significativamente el tamaño de los paquetes. Esto significa que el juego está casi sincronizado la mayor parte del tiempo, lo que mejora significativamente la experiencia del usuario, mientras que también reduce la carga en la red.

Sin embargo, enviar solo el input no funciona en todas las situaciones ya que hace falta que el videojuego sea determinista, por lo que *State Synchronization* envía el input y el estado para garantizar que el juego sea preciso y justo para todos los jugadores.

Además, esta técnica permite ajustar el intervalo de tiempo entre las actualizaciones de estado, lo que permite equilibrar la precisión y el ancho de banda en función de las necesidades del juego. En resumen, *State Synchronization* es una técnica de sincronización muy sólida que se adapta perfectamente a muchos tipos de juegos, proporcionando un equilibrio entre la eficiencia y la precisión en la sincronización del juego.

Esta técnica al tener lo mejor de las técnicas vistas anteriormente es la que mejor resultado ha dado ya que la red no se congestiona y aunque el juego es no determinista siempre está perfectamente sincronizado. El buen funcionamiento de esta técnica no solo se debe a lo sólida que es, sino que también se ha reutilizado las técnicas de *snapshot compression* e interpolación que se usaron en la implementación de *snapshot interpolation*, lo que ha dado un muy buen resultado, teniendo una sincronización perfecta sin saturar la red.

Después de implementar las tres técnicas, ver sus ventajas y desventajas y compararlas, la técnica seleccionada ha sido *State shyncronization*, debido a su robustez y buen desempeño.

Como se puede ver en la Ilustración 47 El juego está correctamente sincronizado y no gasta excesivos recursos en red.

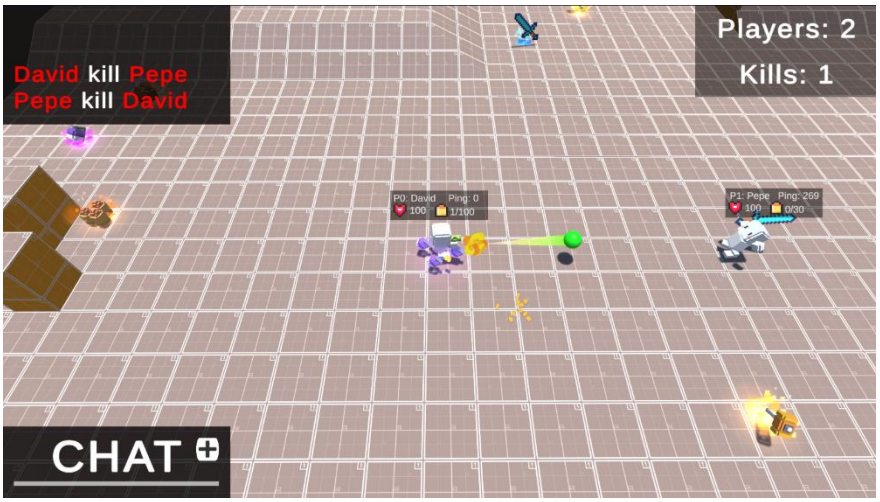


Ilustración 47 Cubes Battle - State Shyncronization

4.3.5. Mejora de netcode

Una vez realizada la implementación de las tres técnicas y elegida la que mejor se adapta al videojuego, el siguiente paso fue mejorar todo lo posible la implementación para explotar todas las capacidades que ofrece *netcode* y *state shyncronization*. Las mejoras más relevantes que se realizaron fueron las siguientes:

- *Snapshot compression*: El objetivo principal de la *Snapshot Compression* es 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. Al reducir la cantidad de datos enviados, se mejora la eficiencia de la transmisión y se reduce la latencia, lo que resulta en una experiencia de juego más fluida. En el caso de este proyecto, algunas de las técnicas que se han usado ha sido que si el estado del jugador no ha cambiado no se envíe el paquete ya que no al ser idéntico al anterior no va a aportar nada a la sincronización y evita enviar un paquete entero. Otra técnica ha sido sustituir la rotación, la cual es un *Quaternion* (Vector de 4 elementos) por una variable de tipo short ya que la única rotación que se aplica es en eje Y, lo que significo reducir el tamaño de la información de la rotación más de 4 veces.
- *Client-side prediction*: Esta técnica consiste en permitir que el cliente del juego realice predicciones locales sobre los movimientos y acciones del resto de jugadores, en lugar de esperar a recibir actualizaciones del servidor.

En esta técnica, el cliente del juego realiza una simulación local basada en la información recibida del servidor, asumiendo que las acciones del jugador y los eventos del juego seguirán una secuencia predecible. De esta manera, el cliente puede mostrar inmediatamente los resultados de las acciones del jugador sin tener que esperar la confirmación del servidor.

Esta técnica es realmente útil en situaciones desfavorables de red ya que permite predecir donde están el resto de los jugadores para que la experiencia de usuario sea como la de un juego local. Aunque se realiza esta predicción, cuando llega el paquete del servidor con la posición correcta se realiza una interpolación para corregir la predicción si fuera necesario. Esta técnica ha resultado realmente útil en situaciones de lag ya que en situaciones normales no es tan decisiva. Para que esta técnica sea más efectiva se ha usado la extrapolación, la cual consisten en enviar diferentes datos para permitir predecir correctamente al resto de clientes la posición futura.

- Sincronizar información de una forma óptima: En NGO existen 2 formas de sincronizar datos, a través de llamadas de procedimiento remoto (en inglés, Remote Procedure Call (RPC)), las cuales consisten en llamadas directas de un cliente al servidor o del servidor a todos los clientes, y la otra forma es con *NetworkVariables*, las cuales son un tipo de variables propias de NGO que cuando se cambia su valor automáticamente en todas las instancias de la partida.

  Hasta ahora casi todas las variables se sincronizaban mediante llamadas RPC, pero en algunos casos se hacían llamadas que no hacían falta y saturaban la red sin necesidad, por lo que todas estas variables pasaron de sincronizarse a través de *NetworkVariables*.

Aparte de aplicar técnicas que mejoran la experiencia de usuario y mejoras la experiencia en situaciones desfavorables de red, también se usado varias herramientas para facilitar la gestión de partidas:

- Lobby: Lobby es una herramienta del paquete de *Unity Gaming Service* la cual facilita la creación de lobbies junto con la gestión de estas. Gracias a esta herramienta el proyecto cuenta con sistema de creación de lobby, tanto

privadas como públicas, y pudiendo variar la cantidad de jugadores. Una vez creado el *lobby* se genera un código para poder unirse a esta para que así desde el menú, puedas unirte a una partida desde la lista de lobbies o uniéndote, introduciendo el código.

- *Relay*: *Relay* es una herramienta del paquete de *Unity Gaming Service* la cual permite la conexión en red de las partidas. Hasta ahora las partidas se jugaban de forma local simulando que estaba online, y apoyándose en Clumsy para poder simular las situaciones de *lag*. Gracias al *Relay* de Unity las partidas se pueden realizar en red.

4.3.6. Mejoras de usabilidad

En busca de optimizar la experiencia del usuario, se ha puesto especial atención en el aspecto de la usabilidad de la aplicación. Se ha trabajado arduamente para simplificar y hacer más accesible el uso de la plataforma a todos los usuarios, independientemente de su nivel de experiencia o conocimientos técnicos.

Una de las principales mejoras implementadas se relaciona con la capacidad de ofrecer una mayor cantidad de información en pantalla sin comprometer la limpieza y la organización de la interfaz. Se ha buscado encontrar el equilibrio perfecto entre la presentación de datos relevantes y la simplicidad visual, evitando sobrecargar al usuario con una excesiva cantidad de información o elementos confusos.

Para lograr esto, se ha trabajado en la jerarquización de la información, priorizando aquellos datos que son más relevantes para el usuario y colocándolos en lugares estratégicos dentro de la interfaz. Se ha realizado un estudio exhaustivo de las necesidades y preferencias de los usuarios, lo que ha permitido identificar qué información es esencial y cómo presentarla de manera clara y concisa.

4.4. Conclusiones

Tras el desarrollo de este proyecto, un videojuego *shooter* multijugador en Unity C# utilizando NGO, se pueden sacar conclusiones sobre *netcode* y más en específico las técnicas que se han usado: *deterministic lockstep, snapshot interpolation y state synchronization*.

Deterministic lockstep ha demostrado ser una técnica eficiente en juegos donde la precisión y la consistencia son cruciales. Sin embargo, su limitación de requerir un juego determinista excluye muchos escenarios en los que las físicas y las interacciones

complejas están presentes. En este caso, al tratarse de un *shooter* multijugador con elementos de física y velocidad variables, esta técnica no se ajustaba a las necesidades.

Snapshot interpolation, por otro lado, ofrece una buena solución para juegos no deterministas al capturar y enviar actualizaciones de estado periódicas. Esto permite una sincronización adecuada y una experiencia de juego fluida. Sin embargo, el gran tamaño de los paquetes de datos puede generar congestión en la red y crear *lag* en la experiencia de juego. Afortunadamente, la técnica de *snapshot compression* ha ayudado a mitigar este problema al reducir la cantidad de datos enviados, evitando la transmisión de información innecesaria.

Finalmente, después de evaluar las opciones, se ha usado la técnica más apropiada para este videojuego, *state synchronization*. Esta técnica combina lo mejor de las dos anteriores, enviando tanto el input del jugador como actualizaciones de estado periódicas. Esto ha permitido equilibrar la precisión y el ancho de banda de acuerdo con las necesidades del juego. El juego está casi sincronizado la mayor parte del tiempo, proporcionando una experiencia fluida para los jugadores, y periódicamente se envían paquetes más pesados para sincronizar cualquier desfase. Además, se ha utilizado *snapshot compression* para reducir el tamaño de los paquetes y minimizar la congestión en la red.

En resumen, la implementación de *state synchronization* en este videojuego multijugador ha demostrado ser la elección acertada. Ha permitido lograr una sincronización precisa del estado del juego, garantizando una experiencia de juego justa y consistente para todos los jugadores. A través de la combinación de técnicas y la optimización de los paquetes de datos, se han logrado resultados satisfactorios en términos de rendimiento y jugabilidad.

5. Validación

5.1. Tests

Para comprobar el correcto funcionamiento de las ciertas funcionalidades del proyecto se han usado *tests*, los cuales consisten en verificar y validar el funcionamiento del videojuego. Los *tests* son utilizados para detectar errores, verificar el cumplimiento de los requisitos y asegurar que el videojuego se comporte de acuerdo a lo esperado.

Los *tests* tienen diferentes tipos, entre los que se encuentras los *tests* unitarios que son los que se han usado en este proyecto. Los test unitarios son pruebas que se centran en verificar el comportamiento de componentes individuales del videojuego, como funciones, métodos o clases. Estos *tests* se ejecutan de manera automatizada para asegurar que el código funcione correctamente en aislamiento.

Los *tests* unitarios que se han implementado en este proyecto han sido para verificar que las conexiones a las partidas funcionan correctamente en diferentes situaciones. Los *tests* que se han implementado han sido los siguientes:

- Conexión de clientes.
    - Cliente que se conecta a una nueva sesión de juego.
    - Cliente que se conecta a una nueva sesión de juego tras abandonar una sesión de juego anterior.
        - En el caso de un juego alojado por un cliente, después de finalizar un juego anterior como anfitrión.
    - Cliente que se conecta a una sesión de juego en curso (*late-joining*).
    - Cliente que se vuelve a conectar a una sesión de juego en curso.
    - Cliente que no puede conectarse debido a una aprobación denegada.
- Desconexión de clientes.
    - Desconexión del cliente mediante el cierre de *NetworkManager*.
    - Desconexión del cliente al cerrar la aplicación.
    - Desconexión del cliente al perder la conexión con el host/servidor.
        - Deshabilitando Internet en el cliente.
        - Deshabilitándolo en el host/servidor.
- Host / Servidor iniciando la sesión.
    - Host/Servidor iniciando una nueva sesión de juego.

  - Host/Servidor iniciando una nueva sesión de juego después de cerrar una sesión de juego anterior.
    - En el caso de un juego alojado en un cliente, después de desconectarse de un juego anterior como cliente.

- Apagado del host/servidor.
  - Desconexión correcta del host/servidor al cerrar *NetworkManager*.
  - Host/Servidor desconectándose, cerrando la aplicación.
  - Si se necesitan servicios para funcionar:
    - El Host/Servidor se desconecta al perder la conexión con los servicios.

6. Conclusiones

6.1. Logros alcanzados

Para empezar a hablar de los logros alcanzados primeros vamos a revisar los objetivos que se plantearon al inicio del documento:

- Estudiar la historia y evolución de los videojuegos multijugador: Como se ha podido ver en el punto 2.1, se ha realizado un análisis de toda la historia de los videojuegos multijugador, junto con las técnicas que se han ido usando y los avances que se iban haciendo en cada generación. De todo este análisis se ha podido sacar mucha información y contexto del mundo de los videojuegos multijugador que ha ayudado a entender cómo funciona y como enfocar este proyecto. Toda esta información ha sido plasmada en el punto 2.1 de este documento de una forma estructurada y ordenada, por lo que se puede decir que este objetivo se ha cumplido.
- Analizar la situación actual en la que se encuentra la creación de videojuegos multijugador, las técnicas que se usan y como solucionan problemas: Después de todo el análisis histórico se ha realizado un profundo estudio de la situación actual de los videojuegos multijugador y de *netcode* ya que este análisis ha sido los cimientos del proyecto porque ha proporcionado un gran conocimiento que ha servido para desarrollar el proyecto. Al igual que el marco histórico todo este conocimiento ha sido reflejado en los puntos 2.2, 2.3 y 2.4 de este documento de una forma estructurada y ordenada, enfocándose en el tipo de videojuego que se ha realizado en este proyecto, por lo que este objetivo también ha sido cumplido.
- Creación de un videojuego multijugador: Este ha sido el principal objetivo del proyecto, pero para llegar a este ha sido necesario cumplir los dos anteriores ya que son los cimientos de este. Como se puede ver el videojuego ha sido desarrollado con éxito y cumpliendo con todos los estándares y requisitos de debía tener por lo que se puede decir que este objetivo se ha cumplido.
- Validación del proyecto: Este es el último objetivo que se ha propuesto al principio del documento, y es uno importante ya que es que se asegura de que todo el proyecto funcione correctamente y no tenga fallos. Como se ha visto en el punto 4 de este documento ha habido una labor para asegurarse de que este objetivo se cumpla correctamente.

Más allá de estos objetivos que se propusieron al principio del documento, se ha conseguido otros logros como todos los requisitos funcionales que hay en el punto 4.1 y 4.2, ya que todos ellos están implementados en el videojuego.

6.2. Lecciones aprendidas

El desarrollo de este proyecto, tanto de la parte más teórica de investigación como la parte más práctica de desarrollo ha supuesto un desafío debido a la magnitud del mismo, teniendo en cuenta que todo ha sido realizado por una sola persona.

Con este proyecto he aprendido a cómo gestionar un proyecto tan grande y como sobrepasar las dificultades que han ido surgiendo ya que a pesar de que desde el principio ha habido una planificación y un orden, no siempre ha ido como se esperaba y ha habido que improvisar para poder mejorar y seguir adelante con el proyecto.

Por otro lado, todo el conocimiento teórico que he obtenido con este proyecto es otra lección aprendida ya que es conocimiento nuevo que he conseguido y que voy a poder aplicar en un futuro en diversos proyectos. Por la parte teórica también he conseguido agilizar el proceso de análisis y aprendizaje tanto de documentos de historia como de documentación de las diferentes herramientas que se han usado en este proyecto.

Por la parte práctica, el hecho de enfrentarse en una situación real a este tipo de proyecto me ha ayudado a aprender mucho más sobre las tecnologías usadas y a los problemas que surgen y como sobrepasarlos.

A continuación, se realizará un análisis del desarrollo del proyecto, dividiéndolo en aspectos positivos y aspectos a mejorar:

- Aspectos positivos identificados:
    - Ritmo de trabajo constante: Se destaca que se ha mantenido un ritmo de trabajo constante, lo cual ha permitido avanzar de manera continua en el proyecto. Esto es un aspecto positivo, ya que asegura un progreso constante y evita retrasos innecesarios.
    - Cumplimiento de metas: Se destaca que el proyecto se ha completado en su totalidad, sin necesidad de eliminar ninguna de las metas planteadas inicialmente. Esto indica que se ha logrado alcanzar los objetivos establecidos, lo cual es un logro significativo y demuestra la efectividad de la planificación y ejecución del proyecto.

  - Investigación exhaustiva: Se destaca que se ha realizado una exhaustiva investigación previa al inicio del desarrollo. Esto demuestra una dedicación para adquirir los conocimientos necesarios y comprender a fondo los aspectos relevantes del proyecto. Una investigación adecuada sienta las bases para un desarrollo sólido y bien fundamentado.
  - Pausas periódicas y finalización de tareas: Se menciona que se han realizado pausas periódicas para analizar el estado del proyecto y finalizar tareas antes de comenzar otras. Estas pausas permiten una evaluación regular del progreso y aseguran que las tareas se completen antes de pasar a la siguiente etapa. Esto contribuye a mantener un enfoque organizado y evitar posibles problemas futuros.
  - Refactorización del proyecto: Se destaca que se ha llevado a cabo la refactorización del proyecto, lo que implica mejorar su eficiencia y facilitar su escalabilidad. La refactorización es un proceso importante para optimizar el código y la estructura del proyecto, lo que puede resultar en un mejor rendimiento, mantenibilidad y capacidad de adaptación a futuras necesidades.

- Aspectos que mejorar identificados:
  - Organización de los scripts: Se destaca que la organización de los scripts podría haber sido mejor, ya que algunos de ellos resultaron ser muy grandes y difíciles de manejar. Una mejor organización y estructuración de los scripts puede facilitar la comprensión, el mantenimiento y la colaboración en el proyecto, lo que conduce a un desarrollo más eficiente y menos propenso a errores.
  - Falta de experiencia con proyectos grandes: Se reconoce que la falta de experiencia con proyectos de gran envergadura ha llevado a que no se cumplan ciertas fechas y se haya tenido que replanificar aspectos debido a problemas no contemplados. La falta de experiencia puede ser un desafío en proyectos complejos, pero también brinda oportunidades de aprendizaje y mejora continua. Es importante reflexionar sobre las lecciones aprendidas y considerar cómo aplicar esos conocimientos en proyectos futuros.

Es fundamental tener en cuenta estos aspectos a mejorar para proyectos futuros, ya que permitirán optimizar la organización y abordar los desafíos de manera más efectiva.

6.3. Líneas futuras

Este proyecto tiene diferentes posibilidades para seguir expandiéndolo:

- Nuevas mecánicas: Este proyecto es un prototipo de videojuego, lo que significa que no es un juego completo, por lo que se le puede añadir fácilmente más mecánicas para crear un videojuego más complejo. Debido a que actualmente las mecánicas que hay implementadas son genéricas de un *shooter*, por lo que se puede orientar a un tipo de juego en específico y crear mecánicas en base a eso.
- Guardado de datos: Actualmente no existe un sistema de guardado de datos por lo que una posible mejora puede ser implementar un sistema de guardado y carga de datos para todos los datos que lo necesiten, como por ejemplo nivel, muertes (en inglés, *kills)*, horas jugadas…
- Sistema de amigos: Actualmente puedes jugar con tus amigos si este te da el código de la sala y hay hueco en la misma, pero una posible mejora puede ser crear un sistema de amigos que te diga si están jugando y te permita unirte a su partida de una forma más cómoda.
- Adaptación a dispositivos móviles: El videojuego en este momento se puede jugar en ordenador pero es una buena idea adaptarlo a dispositivos móviles ya que este tipo de juegos triunfan más en esa plataforma, aparte de expandir los límites del videojuego.
- Modos de juego: El videojuego al ser un prototipo solo cuenta con un modo de juego, pero lo ideal sería que contase con varios modos de juegos y que al crear un *lobby* te permitiese elegir el modo de juego que se va a jugar en ese lobby.

Bibliografía

Armitage, G. (2006). *Networking and Online Games: Understanding and Engineering Multiplayer Internet Games.* Wiley.

Bartle, R. A. (s.f.). *Multi-User Dungeons*. Obtenido de Multi-User Dungeons: https://mud.co.uk/richard/ifan294.htm

Fiedler, G. (1 de Octubre de 2008). *UDP vs. TCP*. Obtenido de Gaffer On Games: https://gafferongames.com/post/udp_vs_tcp/

Fiedler, G. (29 de Noviembre de 2014). *Deterministic Lockstep*. Obtenido de Gaffer On Games: https://gafferongames.com/post/deterministic_lockstep/

Fiedler, G. (28 de Noviembre de 2014). *Introduction to Networked Physics*. Obtenido de Gaffer On Games: https://gafferongames.com/post/introduction_to_networked_physics/

Fiedler, G. (30 de Noviembre de 2014). *Snapshot Interpolation*. Obtenido de Gaffer On Games: https://gafferongames.com/post/snapshot_interpolation/

Fiedler, G. (4 de Enero de 2015). *Snapshot Compression*. Obtenido de Gaffer On Games: https://gafferongames.com/post/snapshot_compression/

Fiedler, G. (5 de Enero de 2015). *State Synchronization*. Obtenido de Gaffer On Games: https://gafferongames.com/post/state_synchronization/

Fiedler, G. (24 de Marzo de 2019). *Fixing the Internet for Games*. Obtenido de Gaffer On Games: https://gafferongames.com/post/fixing_the_internet_for_games/

Fiedler, G. (s.f.). *Gaffer On Games*. Obtenido de Gaffer On Games: https://gafferongames.com/

InfinityWard. (25 de Octubre de 2019). *Call of Duty: Modern Warfare Netcode Explained!* Obtenido de Youtube: https://www.youtube.com/watch?v=tCpYV4k_izE

Madhav, J. G. (2015). *Multiplayer Game Programming: Architecting Networked Games.* Addison-Wesley Professional.

Stagner, A. (2013). *Unity Multiplayer Games.* Packt Publishing.

Technology, R. G. (6 de Noviembre de 2015). *Fixing the Internet for Real Time Applications: Part I*. Obtenido de Riot Games: https://technology.riotgames.com/news/fixing-internet-real-time-applications-part-i

Technology, R. G. (11 de Febrero de 2016). *Fixing the Internet for Real Time Applications: Part II.* Obtenido de Riot Games: https://technology.riotgames.com/news/fixing-internet-real-time-applications-part-ii?_gl=1*1w5xvj1*_ga*NTc1MTAwMTc1LjE2ODMxMDIwNTY.*_ga_7VLGVTHBTW*MTY4MzE4MzcwMy4zLjEuMTY4MzE4NTA5NC42MC4wLjA.

Technology, R. G. (24 de Julio de 2016). *Fixing the Internet for Real-Time Applications: Part III.* Obtenido de Riot Games: https://technology.riotgames.com/news/fixing-internet-real-time-applications-part-iii?_gl=1*fvhf2m*_ga*NTc1MTAwMTc1LjE2ODMxMDIwNTY.*_ga_7VLGVTHBTW*MTY4MzE4MzcwMy4zLjEuMTY4MzE4NTEwNi40OC4wLjA.

Technology., R. G. (28 de Julio de 2020). *Peeking into VALORANT's Netcode.* Obtenido de Riot Games: https://technology.riotgames.com/news/peeking-valorants-netcode

Unity. (24 de Abril de 2023). *About Netcode for GameObjects*. Obtenido de Unity Multiplayer Networking: https://docs-multiplayer.unity3d.com/netcode/current/about/index.html

Ludografía

Nombre	Año	Desarrolladora	Distribuidora
Age of Empires	1997	Ensemble Studios	Microsoft Studios
Apex Legends	2019	Respawn Entertainment	Electronic Arts
Call of Duty: Modern Warfare	2019	Infinity Ward	Activision
Computer Space	1971	Nutting Associates	Nutting Associates
Doom	1993	id Software	id Software
Dragon Ball FighterZ	2018	Arc System Works	Bandai Namco
Empire	1971	Walter Bright	Walter Bright
Fortnite	2017	Epic Games	Epic Games
League of Legends	2009	Riot Games	Riot Games
MegaWars 1	1983	Doug Carlston	Automated Simulations
Multi-User Dungeon (MUD)	1978	Richard Bartle	N/A
Overwatch	2016	Blizzard Entertainment	Blizzard Entertainment
Pong	1972	Atari	Atari
PlayerUnknown's Battlegrounds	2017	PUBG Corporation	PUBG Corporation
Quake	1996	id Software	id Software
Spacewar!	1962	Steve Russell	N/A
Starsiege: Tribes	1998	Dynamix	Sierra On-Line
Street Fighter V	2016	Capcom	Capcom
Super Smash Bros	1999	HAL Laboratory	Nintendo
Tennis for Two	1958	William Higinbotham	N/A
Ultima Online	1997	Origin Systems	Electronic Arts
Valorant	2020	Riot Games	Riot Games

Manual de uso

ANEXO I

Glosario

ANEXO II

Anexos

Anexo I – Manual de uso

El sistema de uso del proyecto es muy sencillo:

1. Una vez abres el videojuego tienes que escribir un nombre en el lugar indicado y también puedes elegir un color entre los disponibles, aunque esto es opcional.

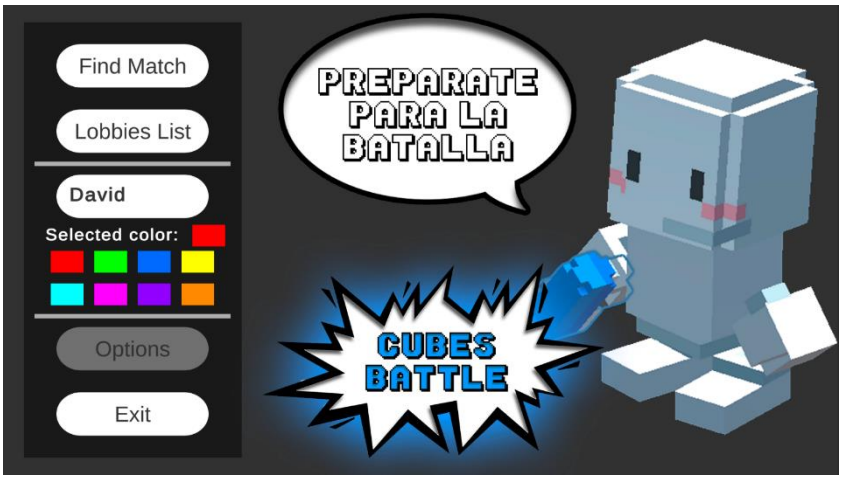


Ilustración 48 Manual de uso - Parte 1

2. Una vez escribas el nombre ya podrás acceder a los botones de "Find Match" y Lobbies List:
    a. Find Match te meterá automáticamente en una partida, si no hay ninguna disponible la creará y si hay alguna disponibles te unirá a esa.
    b. Lobbies List te lleva a una pantalla donde podrás crear un lobby a tu gusto o unirte a uno ya existente.
3. En la pantalla de la lista de lobbies puedes hacer varias acciones:

Ilustración 49 Manual de uso - Parte 2

a. Volver al menú principal mediante el botón “Back”
b. Actualizar la lista de lobbies mediante el botón “Refresh List” para que también aparezcan los nuevos lobbies que se han creado.
c. Puedes introducir el código de un lobby existente y pulsar el botón “Join By Code” para unirte a ese lobby.
d. Si hay lobbies en la lista puedes unirte mediante el botón “Join” que hay en cada lobby.
e. Puedes ir a la pantalla de creación de lobby mediante el botón “Create Lobby”.

4. En la pantalla de creación de lobby debes introducir un nombre y puedes configurar el número de jugadores máximos y si es privada o no (si es privada no aparece en la lista de lobbies y solo puedes acceder mediante código). Una vez configurado si le das al botón “Create Lobby”, crea el lobby con las características indicadas y te unes automáticamente.

Ilustración 50 Manual de uso - Parte 3

Controles:

- Moverse: Para moverse se usan las teclas “W”, “A”, “S” y “D”
- Saltar: Para saltar se usa la tecla de “Espacio”
- Deslizarte: Para deslizarte, tienes que pulsar la tecla “Left Control” mientras te estás moviendo.
- Cambiar de arma: Para cambiar de arma tienes que pulsar la tecla “1” o “2” dependiendo de arma que tengas equipada actualmente.

- Recargar: Para recargar mientras mientras tienes un arma de balar tienes que pulsar la tecla “R”, siempre y cuando tengas balas.
- Atacar: Para atacar hay que pulsar el click izquierdo del ratón.
- Recoger objetos: Algunos objetos se recogen automáticamente al pasar por encima de ellos mientras que otros es necesario pulsar la tecla “E” para recogerlos.
- Lanzar granada. Para lanzar la granada hay que pulsar el botón derecho del ratón siempre y cuando tengas granadas disponibles.

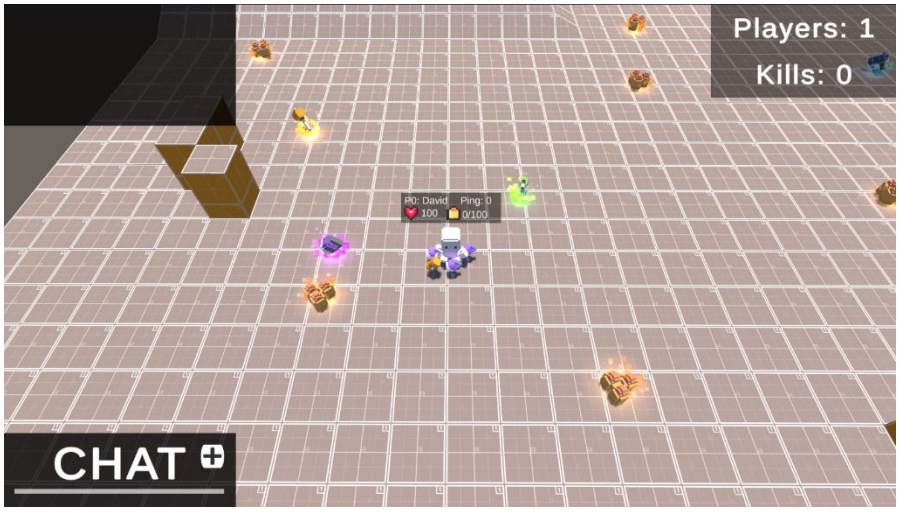


Ilustración 51 Manual de uso - Parte 4

Anexo II – Glosario

Término	**Definición**
Ancho de banda	La capacidad máxima de transferencia de datos de una red.
API	Conjunto de reglas y protocolos que permiten la interacción entre diferentes componentes de software.
Arquitectura de red	Estructura y diseño de una red de computadoras.
Asset	Recurso utilizado en el desarrollo de un videojuego, como modelos 3D, texturas o sonidos.
Bug	Error o fallo en el software que produce un comportamiento no deseado.
Cliente-Servidor	Modelo de arquitectura de red donde un cliente solicita servicios a un servidor.
Código abierto	Software cuyo código fuente está disponible y puede ser modificado y distribuido libremente.
CPU	Unidad central de procesamiento, encargada de ejecutar las instrucciones de un programa.
Debug	Proceso de identificar y corregir errores en el software.
Deterministic Lockstep	Técnica de sincronización en la que los clientes avanzan en el juego de forma determinista.
Diagrama de Gantt	Representación gráfica de las tareas y plazos de un proyecto.
FPS	Género de videojuegos que se centra en disparos.
Frameworks	Conjunto de herramientas y librerías que facilitan el desarrollo de software.
Git	Sistema de control de versiones utilizado para gestionar y rastrear cambios en el código fuente.
IDE	Entorno de desarrollo integrado, que proporciona herramientas para escribir, compilar y depurar código.
Input	Datos proporcionados por el usuario a través de dispositivos como teclado, ratón o controlador.
IP	Protocolo de comunicación utilizado para la transmisión de datos en Internet.
Jitter	Variación en el retardo de la transmisión de datos.

Lag	Retraso perceptible en la respuesta de un videojuego debido a la latencia de red.
LAN	Red de computadoras o dispositivos interconectados en un espacio limitado para compartir recursos y comunicarse.
Librería	Conjunto de funciones y recursos reutilizables que facilitan el desarrollo de software.
Librería de alto nivel	Librería que proporciona abstracciones y funciones simplificadas para facilitar el desarrollo de aplicaciones.
Lógica networking	La lógica detrás de la comunicación y sincronización de datos en una red.
Lobby	Espacio virtual donde los jugadores pueden reunirse antes de comenzar una partida.
Mainframe	Unidad central de procesamiento de alto rendimiento utilizada en entornos empresariales.
Mecánica	Conjunto de reglas y acciones que rigen el funcionamiento de un videojuego.
Mods	Modificaciones o extensiones creadas por la comunidad para modificar o añadir contenido a un videojuego.
Modders	Personas que crean mods o modificaciones para videojuegos.
Módem	Dispositivo que permite la conexión a Internet a través de una línea telefónica.
Motor de videojuegos	Plataforma o software utilizado para el desarrollo de videojuegos.
Netcode	Conjunto de técnicas y algoritmos utilizados para gestionar la comunicación en juegos en línea.
Netcode for GameObjects	Herramienta de Unity que facilita la sincronización de objetos en juegos multijugador.
Offline	Modo de juego que no requiere conexión a Internet.
Online	Modo de juego que requiere una conexión a Internet.
Peer-to-Peer	Modelo de arquitectura de red donde los dispositivos se conectan directamente entre sí sin un servidor central.

Pérdida de paquetes	Situación en la que algunos paquetes de datos no llegan correctamente a su destino en una red.
Ping	Medida del tiempo de respuesta entre un cliente y un servidor en una red.
Plugins	Componentes adicionales que se pueden agregar a un software para ampliar su funcionalidad.
Protocolo de red	Conjunto de reglas y estándares que definen la comunicación entre dispositivos en una red.
PDP-1	Computadora de los años 60, conocida por ser la primera en ejecutar un videojuego.
Quaternion	Representación matemática de la orientación y rotación en tres dimensiones.
Relay	Servidor intermedio que retransmite datos entre dispositivos en una red.
Rollback Netcode	Técnica de sincronización en la que se revierten acciones para mantener la consistencia entre los clientes.
Round-trip-Time	Tiempo que tarda un paquete de datos en ser enviado desde un origen y recibir una respuesta de destino.
Router	Dispositivo de red que dirige el tráfico entre diferentes redes.
RTS	Género de videojuegos de estrategia en tiempo real.
SCRUM	Metodología ágil de gestión de proyectos.
Script	Fragmento de código que contiene instrucciones para realizar una tarea específica.
Server-authority	Modelo de arquitectura de red donde el servidor tiene autoridad sobre la simulación del juego.
Snapshot Interpolation	Técnica de interpolación que suaviza los cambios entre instantáneas de estado en un juego multijugador.
Software	Conjunto de programas, instrucciones y datos utilizados por un sistema informático.
State Synchronization	Técnica de sincronización que garantiza que todos los clientes tengan el mismo estado del juego.

Tactical shooters	Género de videojuegos de disparos que enfatizan el trabajo en equipo y la estrategia.
TCP/IP	Conjunto de protocolos utilizados para la transmisión de datos en redes.
Terminal tonto	Dispositivo de entrada y salida básico utilizado para acceder a un sistema remoto.
Test	Proceso de verificación y validación de software para detectar errores y asegurar su funcionamiento correcto.
UDP	Protocolo de transporte que proporciona una comunicación no confiable y sin conexión.
Vector3	Estructura de datos que representa una posición o dirección en tres dimensiones.
World Wide Web	Sistema de información global que permite el acceso a documentos e información a través de Internet.

Anexo III- Diagrama de clases extendido

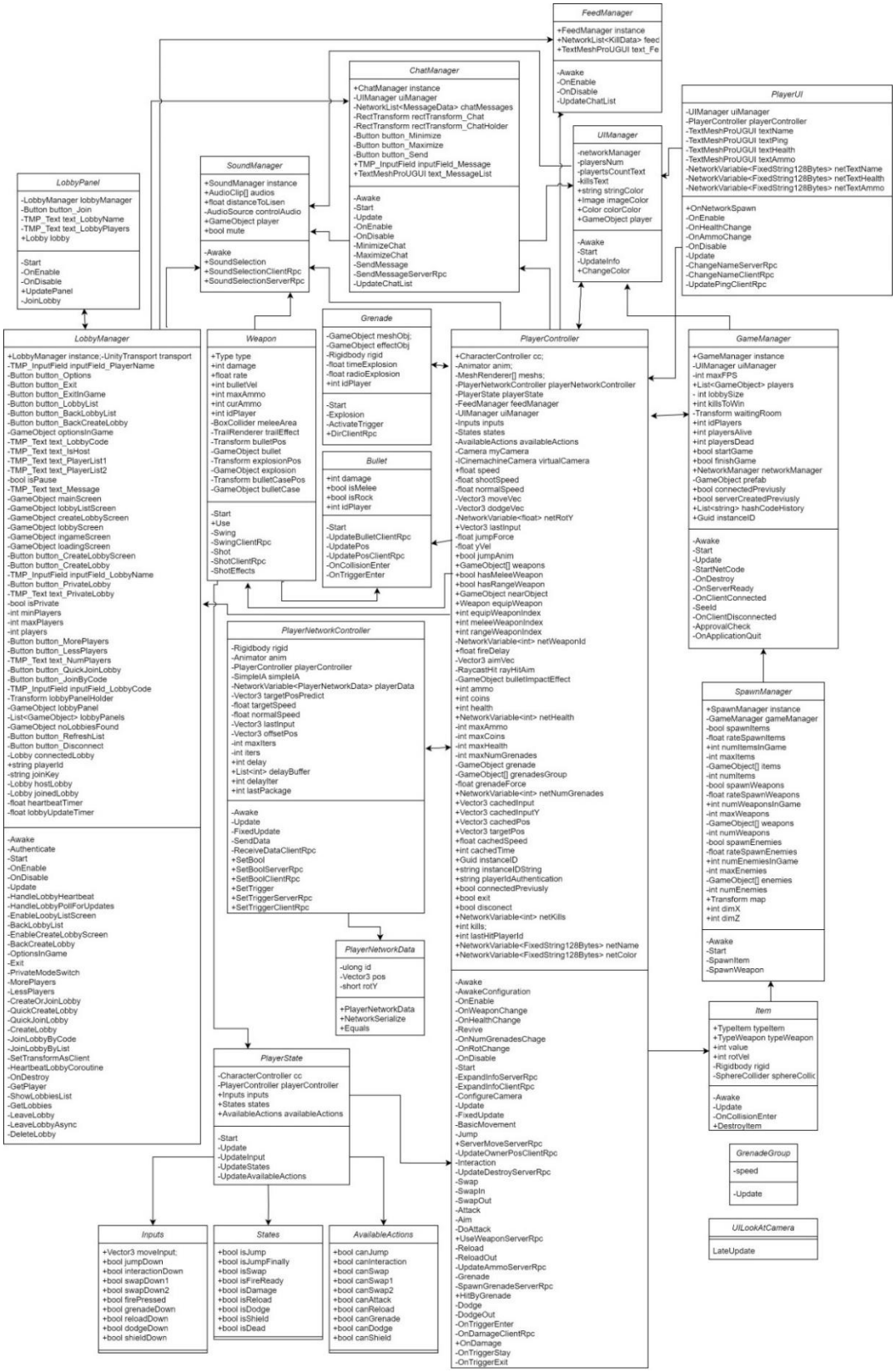
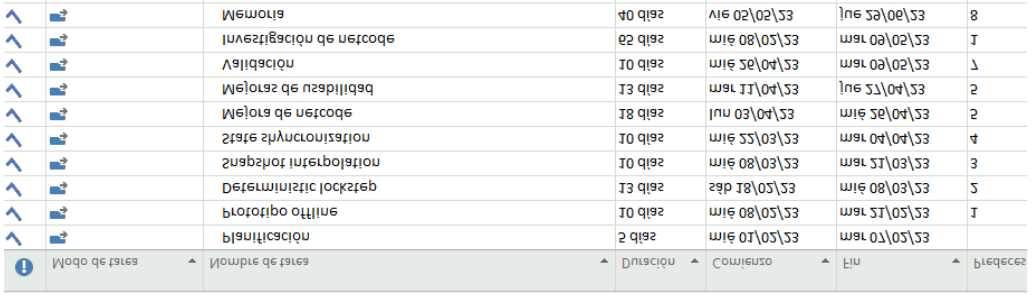


Ilustración 52 Diagrama de clases extendido

Anexo IV – Desarrollo de la planificación



Modo de tarea	Nombre de tarea	Duración	Comienzo	Fin	Predeces
	Planificación	2 días	mié 01/05/13	mar 07/05/13	
	Prototipo offline	10 días	mié 08/05/13	mar 21/05/13	1
	Deformización lockstep	13 días	sáb 18/05/13	mié 08/06/13	5
	Snapshot interpolation	10 días	mié 08/05/13	mar 21/05/13	3
	State synchronization	10 días	mié 22/05/13	mar 04/06/13	4
	Mejora de vértices	18 días	lun 03/06/13	mié 26/06/13	2
	Mejoras de usabilidad	13 días	mar 11/06/13	lun 24/06/13	2
	Validación	10 días	mié 26/06/13	mar 09/07/13	7
	Investigación de vértices	62 días	mié 08/05/13	mar 06/08/13	1
	Memoria	40 días	vie 02/08/13	lun 26/09/13	8

Ilustración 53 Microsoft Project - Parte 1 (grande)

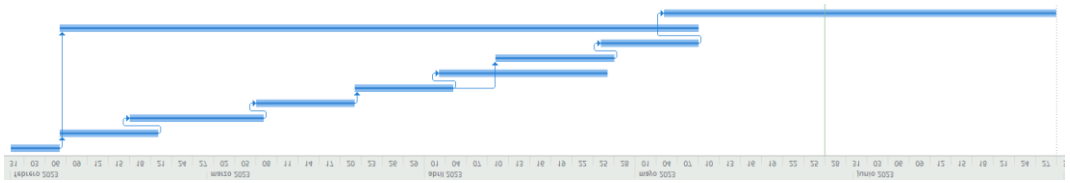


Ilustración 54 Microsoft Project - Parte 2 (grande)