Universidad de Valladolid

Escuela de Ingeniería Informática

Trabajo Fin de Grado

Grado en Ingeniería Informática
Mención en Tecnología de la Información

Desarrollo de un videojuego multijugador

Autor:
Pablo Rodríguez García-Jiménez

Universidad de Valladolid

Escuela de Ingeniería Informática

Trabajo Fin de Grado

Grado en Ingeniería Informática
Mención en Tecnología de la Información

Desarrollo de un videojuego multijugador

Autor:
Pablo Rodríguez García-Jiménez

Tutor:
Carlos Enrique Vivaracho Pascual

Resumen

Este trabajo de fin de grado presenta el desarrollo de un videojuego multijugador online para dispositivos móviles. Tras una breve descripción del estado de la industria, se realiza un análisis de algunas de las herramientas mas utilizadas por desarrolladores independientes para la creación de videojuegos y para el despliegue del entorno multijugador. A través de un Documento de Diseño del Videojuego (GDD), se documenta las características del proyecto durante su ciclo de vida.

Para el desarrollo de la aplicación, se ha utilizado el motor de videojuegos Unity, uno de los mas reconocidos de la industria, y se describen los elementos que forman parte de un proyecto Unity. Para el despliegue del entorno multijugador se usa la base de datos NoSQL Firebase Realtime Database, utilizada a través de la API también descrita en el trabajo.

Palabras clave: Videojuegos, Entorno multijugador, Unity, Firebase Realtime Database, GDD.

Abstract

This end-of-degree project presents the development of an online multiplayer videogame for mobile devices. After a brief description of the state of the art of the industry, an analysis is done on the most used development tools by independant developers in order to create videogames and for a multiplayer environment deployment. Using a Game Design Document (GDD), the proyect features are documented through it's life cycle.

Unity, one of the most recognized videogame engines in the industry, has been used for the development of this proyect, and a description of the elements that are part of this engine has been made. For the multiplayer environment, the NoSQL database Firebase Realtime Database has been used, and his API has also been described in this proyect.

Keywords: Videogames, Multiplayer environment, Unity, Firebase Realtime Database, GDD.

Índice general

Índice de tablas

Índice de figuras

Capítulo 1 Introducción

Hace ya bastantes años que los videojuegos han dejado de ser un simple pasatiempo de unos pocas personas. Esta industria ha ido ganando cada vez más y más terreno a las demás industrias del entretenimiento, convirtiéndose en una de las más grandes. Ya son pocas las personas que no han jugado nunca a ninguno de estos videojuegos. Para muchos consumir este medio se ha convertido en parte de la rutina diaria o semanal.

Para muchísimas personas, en especial adolescentes, los videojuegos no son solo una manera de entretenerse, sino que además sirven como una oportunidad para socializar con nuevos y viejos amigos, a la vez que crear nuevos. Esta capacidad de unir a las personas hacen a los videojuegos una herramienta útil para establecer conexiones. La creación de este proyecto tiene como objetivo ser un instrumento para el estrechamiento de relaciones interpersonales [1].

En este proyecto se quiere analizar el estado actual de la industria de los videojuegos multijugadores online, descubrir herramientas de desarrollo usadas por múltiples desarrolladores independientes y las soluciones más comunes que se usan para el despliegue online. Tras este análisis, se pretende crear uno de estos videojuego multijugador online para dispositivos móviles pasando por los procesos del desarrollo de software, desde la selección de una metodología y planificación del proyecto, pasando por la elaboración del documento de diseño del videojuego, el análisis de requisitos y diseño de software y por último la implementación usando algunas de las herramientas utilizadas en la industria (Unity y Firebase Realtime Database).

1.1 Objetivos

El objetivo principal es la elaboración de un videojuego multijugador para dispositivos móviles que permita a los jugadores crear partidas para jugar con otros usuarios a través de Internet, con una sincronización de estados entre las aplicaciones de estos usuarios.

1.1.1 Objetivos Específicos

Definiendo el objetivo principal y final de este proyecto, se han considerado los siguientes objetivos:

- Estudiar el estado actual del sector de los videojuegos online.

- Estudiar y analizar algunas de las herramientas de creación de videojuegos y soluciones *Back-end* más usadas
- Realizar el diseño conceptual del videojuego a través del *Game Design Document*.
- Diseño e implementación del videojuego, utilizando las herramientas escogidas.

1.2 Organización de la memoria

Esta memoria se compone de 6 capítulos divididos en secciones.

En el 1^{er} capítulo denominado Introducción se presenta de forma generalizada la idea del proyecto, la motivación llevada a su ejecución y los objetivos a cumplir en este proyecto.

En el 2^{o} se presenta la metodología usada para la elaboración del proyecto, la planificación de las tareas a realizar y los posibles riesgos que puedan surgir durante el desarrollo del proyecto..

El 3^{o} capítulo, llamado Estado del Arte, pretende llevar un análisis bibliográfico de sobre la evolución y estado actual de la industria del videojuego. Se hará un enfoque a juegos de carácter online y los que son para dispositivos móviles. Tras ello se analizaran algunas de los motores de videojuegos más utilizados en la industria y algunas soluciones para la implementación de un *back-end*. Termina con la descripción de las herramientas que se usaran en el proyecto.

El 4^{er} capítulo es donde se documenta el Game Design Document. En este documento se describirán los todos los elementos relacionados con el videojuego que se va a crear. Esta estructurado para acoger las diferentes partes del desarrollo de un videojuego, como el Gameplay, Interfaz, elementos Multimedia o marketing y audiencia.

El 5^{o} capítulo, denominado Desarrollo del Proyecto, se describe el proceso de elaboración de la aplicación. Cuenta con análisis y diseño software del proyecto, un análisis bibliográfico de La API del motor Unity y la base de datos Realtime Database, y por último, una descripción de los elementos del videojuego.

Por último, el 6^{o} capítulo, se comentan los objetivos logrados y las tareas a realizar como parte de trabajo para el futuro.

Capítulo 2 Metodología y Planificación

Para el desarrollo de este proyecto se ha decidido usar una metodología Agile que a diferencia de las metodologías tradicionales, dan una mayor flexibilidad, puesto que permiten adaptar el trabajo a las cambiantes circunstancias que se pueden dar durante el desarrollo de un videojuego.

Estas metodologías, además, implementan el desarrollo incremental, que consiste en la construcción del proyecto de manera progresiva. En cada etapa, que consta de análisis, diseño, implementación y pruebas, se agrega una nueva funcionalidad, lo que permite un progreso visible y permite una mejor depuración de cada una de las funcionalidades. Además, el software puede ser usado a partir de las primeras etapas, pudiendo implementar otras funcionalidades después, estrategia que siguen algunos de los proyectos de la industria del videojuego.

Entre las metodologías Agiles se ha seleccionado Scrum, que consiste en la división del trabajo en tareas realizadas en bloques de tiempo fijo (semanas) para conseguir un resultado completo en cada iteración. En la metodología Scrum, se asignan roles a los diferentes miembros del equipo pero, al ser un proyecto individual, una sola persona ejercerá esos roles. Estos roles son:

- **Dueño del proyecto:** Es el representante del negocio, encargado de maximizar el valor del proyecto y hacer las decisiones sobre el trabajo a realizar. En este caso, será el encargado de planificar los sprints y tareas.
- **Scrum Master:** Actúa como líder del equipo de desarrollo y se encarga de que el proceso Scrum se haga correctamente. Traslada la visión del proyecto al equipo de desarrollo. En este caso será el encargado de documentar las herramientas que se usarán durante el desarrollo y desarrollar el GDD (apartado 4) para transmitir la visión global del proyecto.
- **Equipo de desarrollo:** Equipo con los conocimientos técnicos necesarios para llevar a cabo el desarrollo del proyecto. En este caso son los desarrolladores de la aplicación.

2.1 Análisis de riesgos

Este análisis permitirá la mitigación de los efectos introducidos por posibles contratiempos que retrasen el proyecto. Se han identificado una serie de riesgos posibles indicados en las tablas 2.1 a 2.7, en los que se asignan un nivel de impacto y se describen planes para la reducción del impacto.

Identificador	R-01
Riesgo	Calculo erróneo del tiempo de una tarea
Descripción	Calculo incorrecto del tiempo de desarrollo de un incremento, que puede retrasar el resto de tareas.
Plan	Priorización de tareas más importantes.
Impacto	Alto

Tabla 2.1: Riesgo 1

Identificador	R-02
Riesgo	Cambios de los requisitos
Descripción	Cambios en los requisitos del software
Plan	Reajustar tiempos y priorizar tareas. Reajustar dependencia de tareas
Impacto	Alto

Tabla 2.2: Riesgo 2

Identificador	R-03
Riesgo	Aparición de tareas no previstas
Descripción	Cuando nuevas tareas no planificadas en el proyecto surgen.
Plan	Reajustar tiempos y dependencias de tareas.
Impacto	Alto

Tabla 2.3: Riesgo 3

Identificador	R-04
Riesgo	Desarrollo de funciones con errores
Descripción	Desarrollo de funcionalidades que no superan las pruebas de software, haciendo perder el tiempo de desarrollo de otras tareas.
Plan	Priorización de tareas más importantes.
Impacto	Alto

Tabla 2.4: Riesgo 4

Identificador	R-05
Riesgo	Problemas en el ordenador de desarrollo
Descripción	Aparición de problemas relacionadas con el hardware o el sistema operativo del equipo informático de desarrollo.
Plan	Copias de seguridad, ajustado de tiempo de tareas.
Impacto	Medio, Alto, dependiendo del problema

Tabla 2.5: Riesgo 5

Identificador	R-06
Riesgo	Errores con software de terceros.
Descripción	Surgimiento de dependencias u obsolescencia en el software de terceros
Plan	Ajustado de tiempo tareas. Elegir otro software como ultima medida.
Impacto	Medio

Tabla 2.6: Riesgo 6

Identificador	R-07
Riesgo	No disponibilidad de la base de datos
Descripción	Servidores caídos que impiden el progreso del proyecto
Plan	Posposición de tareas.
Impacto	Medio

Tabla 2.7: Riesgo 7

2.2 Planificación

El trabajo de este proyecto se ha dividido en las siguientes tareas o incrementos:

- Sprint 1: Descubrimiento y comparación de herramientas.
- Sprint 2: Análisis de las herramientas elegidas.
- Sprint 3: Diseño del videojuego
- Sprint 4: Análisis de requisitos.
- Sprint 5: Diseño del software.
- Sprint 6: Diseño e implementación de la interfaz.
- Sprint 7: Creación de los elementos gráficos.
- Sprint 8: Conexión con la base de datos.
- Sprint 9: Implementación de la base de datos.
- Sprint 10: Desarrollo de la partida.
- Sprint 11: Desarrollo de ajustes.
- Sprint 12: Implementación de funcionalidades extra.

Siguiendo la metodología Scrum se hace un estimado de tiempo de cada una de las tareas o incrementos, que se denominaran Sprints. A cada incremento se le asigna un sprint. La duración estimada de cada una de estas iteraciones se detalla en la tabla 2.8.

Sprint	**Duración (semanas)**	**Duración (horas)**
Sprint 1	1 semana	18 horas
Sprint 2	2 semanas	36 horas
Sprint 3	2 semanas	36 horas
Sprint 4	1 semana	18 horas
Sprint 5	2 semanas	36 horas
Sprint 6	1 semana	18 horas
Sprint 7	1 semana	18 horas
Sprint 8	1 semana	18 horas
Sprint 9	2 semanas	36 horas
Sprint 10	2 semanas	36 horas
Sprint 11	1 semana	18 horas
Sprint 12	1 semana	18 horas

Tabla 2.8: Sprints previstos y su duración planteada

Sin embargo, la duración del proyecto fue mayor de la inicialmente planteada. Un videojuego, durante todo el ciclo de vida de su desarrollo, puede estar sujeto a múltiples cambios, y estos cambios pueden desajustar la planificación, como se detalla en el riesgo de la tabla [2.2]. Esto hizo que fuera necesario un reajuste del tiempo dedicado a cada tarea. En la tabla 2.9, se muestran los tiempos reales que ha durado cada sprint.

Sprint	**Duración**	**Duración (horas)**
Sprint 1	1 semana	24 horas
Sprint 2	2 semana	44 horas
Sprint 3	2 semanas	36 horas
Sprint 4	2 semana	30 horas
Sprint 5	3 semanas	62 horas
Sprint 6	1 semana	12 horas
Sprint 7	2 semana	36 horas
Sprint 8	1 semana	10 horas
Sprint 9	2 semanas	30 horas
Sprint 10	3 semanas	54 horas
Sprint 11	1 semana	18 horas
Sprint 12	1 semana	18 horas

Tabla 2.9: Sprints reales y su duración real

La mayor diferencia de tiempo surge en el Sprint 10, correspondiente al desarrollo de la partida. Esto se debe a que se trata de la mayor parte del peso del proyecto y es muy susceptible a los cambios que puede haber en ellos, teniendo que pasar la fase de pruebas cada vez que se realizaban cambios en los requisitos, que los hubo.

Los cambios en requisitos, se vieron reflejados también en el sprint 4 y 5 correspondientes a la fase de análisis y diseño del proyecto, puesto que se tuvo que revisitar y replantear los

requisitos planteados, casos de uso y diagramas de secuencia.

2.3 Presupuesto

Durante el desarrollo del proyecto se han utilizado documentos, assets y material multimedia gratuitos bajo licencia Creative Commons, es decir, libre de modificación y distribución. Por supuesto, el desarrollado este proyecto es con fines académicos, por lo tanto el equipo de desarrollo (en este caso, yo) no requiere de un sueldo. Por lo tanto, el presupuesto calculado para el desarrollo del proyecto es en base al uso del ordenador y de la conexión a Internet a través de un ISP.

El ordenador de desarrollo consume de media (aproximada) unos 0,275 kWh y se cobran a una tarifa media de 0,23 euros el kWh. Si el uso del ordenador es de 300 horas entonces el coste de su uso es de 18,97 euros. El uso de Internet, en el lugar de desarrollo, se paga una tarifa fija de 40 euros mensuales pagadas entre 4 personas. si la duración del proyecto, según la planificación establecida, dura 12 semanas, el coste de uso de internet para el desarrollo del proyecto sale a 30 euros.

El ordenador, al igual que la conexión a Internet, no es utilizado en todas las fases del proyecto y, en cada fase, puede haber operaciones que no necesite de su uso, por ejemplo, en la fase de diseño y análisis, se pueden desarrollar diagramas y bocetos usando papel, que luego serán convertidos en digital. Esto afecta al uso del ordenador, que su coste de desarrollo sera menor al inicialmente planteado, pero no a la tarifa fija de uso de Internet.

Capítulo 3 Contexto del proyecto

Este capítulo analiza el estado actual del desarrollo de videojuegos multijugador online, describe herramientas de desarrollo que se usan en esta industria y las soluciones más comunes que se usan para su despliegue online.

3.1 Estado del Arte

La industria del videojuego es una de las industrias más lucrativas del entretenimiento, siendo más grande que la industria del cine, televisión y música juntas. Ya en 2013 se alcanzaron los 76,5 mil millones de dolares y se estima que en 2023 se supere la cifra de 200 mil millones de dolares, un incremento del 167 %. La pandemia provocada por el Covid-19 también ha sido un gran factor en el crecimiento de la industria en 2020, al ser un medio que se puede consumir desde la comodidad del hogar, favorecido por la cuarentena que muchos países establecieron.

El numero de personas que jugaron a un videojuego fue cercano a los 3 mil millones en 2021, un 5,3 % más que el año pasado, siendo más de la mitad (1,615 mil millones) procedentes del continente asiático [3]. Uno de los factores más importantes causantes de este crecimiento es el aumento de usuarios de Internet. La mejora de las infraestructuras y los precios asequibles de móviles y planes de datos ha hecho que haya 5 mil millones de usuarios de Internet en abril de 2022.

Las plataformas donde se puede consumir los juegos varían entre ordenadores personales (PC), consolas (Microsoft XBOX, Sony PlayStation y Nintendo Switch las principales) y los dispositivos móviles, siendo este último el más popular tras un crecimiento masivo desde la revolución del smartphone. En 2021 el dispositivo móvil ocupaba más de la mitad del sector con un 52 % (smartphone un 45 % y tablets un 7 %), creciendo un 34 % desde el 2012. La Covid-19 ha tenido un impacto más grave en el beneficio de los videojuegos desarrollados en consola y PC, cuyas compañías de desarrollo en estas plataformas suelen tener unos equipos más grandes, altos valores de producción y una estrategia de negocio consistente en la compra de producto en “retail”, a diferencia del mercado móvil, con equipos más pequeños y compras dentro de la app; estrategia que les ha puesto como los más importantes del mercado.

De entre todos los tipos de juegos es destacable los orientados a multijugador, tanto cooperativos o competitivos. El mercado online global ha generado 21,1 mil millones de dolares en 2020, un 21,9 % más que el año anterior. Se estima que hay mil millones de jugadores online y se prevé que supere los 1.3 mil millones en 2025.[4]

3.1.1 Juegos multijugador online de la actualidad

En la tabla 3.1 se muestran los videojuegos más jugados en 2021, usando como medida el pico más alto de numero de jugadores simultáneos, según Steam, la tienda y plataforma más importante de videojuegos en PC.

Videojuego	Pico de jugadores simultáneos
Counter Strike: Global Offensive (CSGO)	1.199.610
New World	913.030
DOTA 2	757.140
Valheim	499.540
PlayerUnknown's Battlegrounds (PUBG)	465.710
World of Warships	353.410
Apex Legends	330.880
Halo Infinite	258.540
Rust	241.750
CyberPunk 2077	255.960

Tabla 3.1: Juegos más jugados ordenados por pico de jugadores simultáneos en Steam [2]

A esta lista hay que añadirle otros juegos como Fortnite, que esta en la plataforma de Epic Games, y en mayo de 2020 alcanzo los 350 millones de usuarios registrados [5]. También destaca Roblox, con 54,1 millones de usuarios activos diarios (DAU) en el primer cuarto del año 2022 [6]. Otro juego también popular es League of Legends, en la plataforma de Riot Games, que alcanza los 125 millones de usuarios activos mensualmente [7]. Además algunos juegos como PUBG, están disponibles en otras plataformas como Xbox, Playstation o Móvil, haciendo que esos números sean imprecisos (Actualmente alcanza más de 700.000 jugadores simultáneos [8]).

Sin embargo la la lista de la tabla 3.1 nos da una idea de los géneros más populares en PC. Todos son juegos multijugador online, salvo CyberPunk y Halo Infinite, aunque este último también ofrece una experiencia multijugador.

CSGO, PUBG, Fortnite, Apex Legends y el multijugador de Halo Infinite pertenece al género del shooter, uno de los más populares del medio, género que pertenece al de acción y permite a los jugadores controlar un personaje que dispone de un arma de fuego. Otro género también popular es el Multijugador Masivo en Linea (MMO),al que pertenece, New World, y World of Warships y otros muchos como World of Warcraft y Final Fantasy XIV. Los juegos de Estrategia en Arena en Tiempo Real (ARTS) al que pertenece League of Legends y Dota 2 es un genero también popular en PC. En estos juegos dos o más jugadores compiten en tiempo real, no por turnos, desarrollando una estrategia para alcanzar los objetivos del videojuego. Por último podemos cerrar los géneros con el Sandbox, que permite a un jugador deambular y cambiar un mundo virtual del que goza de su completo acceso desde el principio. Rust, Roblox y Valheim pertenecen a este último género.

Figura 3.1: CSGO, ejemplo de un juego Shooter

En los dispositivos móviles es muy común observar los géneros que aparecen en la tabla 3.2, como los plataformas que se llevan un 15 % de la monetización de los juegos Free-to-Play (F2P). Sin embargo podemos establecer que el mercado móvil es más casual viendo las estadísticas de la tabla 3.2. Los juegos casuales requieren menos del usuario, y son jugados normalmente en ratos libres. Un usuario de dispositivo móvil suele usarlo en tiempos muertos, por ejemplo en el transporte publico, en el servicio o entre actividad y actividad. Los juegos de plataformas, estrategia, Puzzles, Infinite Runner o Idle (juegos en los que se hace progreso esperando tiempo real) se adaptan perfectamente a las características de este usuario. Algunos de estos géneros, el de estrategia por ejemplo, son muy populares cuando se tratan de un multijugador online. Algunos ejemplos de este último género pueden ser Clash of Clans (mas de 500.000.000 descargas en Play Store), Clash Royale (mas de 100.000.000 descargas) o Lords Mobile: Tower Defense (mas de 100.000.000 descargas)[9].

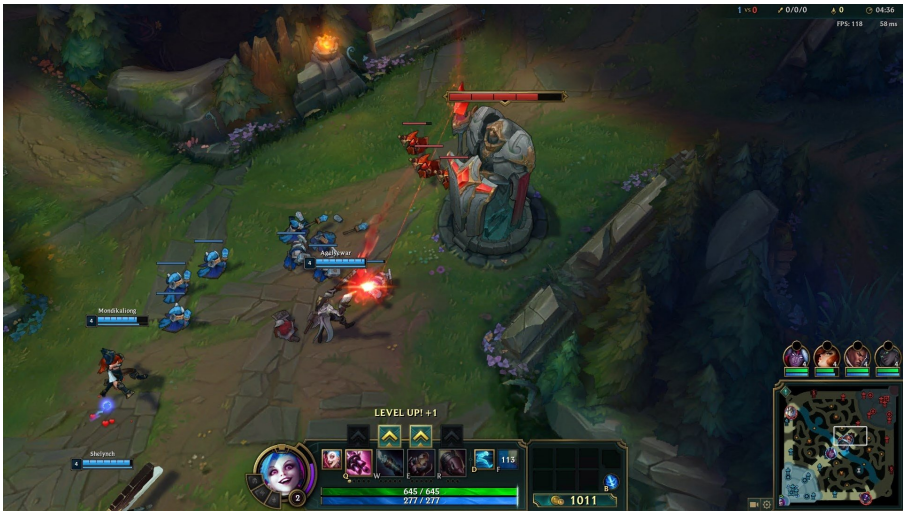

Figura 3.2: League of Legends, ejemplo de un juego ARTS

Genero	Porcentaje de monetización
Plataformas	15 %
Shooter	13 %
Estrategia	11 %
Infinite Runner	10 %
Role- playing Game (RPG)	8 %
Puzsles	8 %
MMO	5 %
Idle	5 %

Tabla 3.2: Géneros con más monetización en móvil F2P [10]

3.2 Análisis de Herramientas

Una de las primeras y más importantes decisiones a la hora de crear cualquier videojuego es elegir el motor de juegos con el que se va a desarrollar. Un motor de juegos suele tener diferentes funcionalidades como simulación de físicas, un renderizador de gráficos, animación, scripting y sonido y, al mismo tiempo, proporcionan un entorno de desarrollo integrado con un conjunto de herramientas para la creación o modificación de estas funcionalidades [11]. Existen bastantes motores de juegos disponibles en el mercado, y en esta sección analizamos unos pocos.

Aparte del motor gráfico tenemos que analizar determinadas herramientas y servicios *back-end* de manera que podamos construir un videojuego multijugador, ya sea mediante un servidor externo o mediante el protocolo *P2P*.

3.2.1 Motor de videojuego

En esta sección analizamos algunos de los motores de juegos más utilizados y conocidos de la industria, indicando algunas de sus características y las ventajas y desventajas sobre otros motores.

3.2.1.1 Unreal Engine

Unreal Engine [12] es la plataforma más completa y potente de la industria del videojuego, aunque también es usada para la realización de animaciones, simulaciones, o en la industria automóvil y arquitectónica. Desarrollado por Epic Games, es uno de los motores más utilizados de la industria.

Aunque esta orientado a entornos 3D permite también proyectos 2D de cualquier tamaño. Unreal Engine permite un diseño modular, la personalización de plugins, el control de versiones y el desarrollo multiplataforma. Esto lo ha convertido en uno de los motores más fiable de la industria, usado por múltiples empresas, ya sean grandes o pequeñas. De entre las compañías que utilizan este software en alguno de sus proyectos destacan Microsoft, Electronic Arts, o Square Enix.

Unreal Engine cuenta con una API extensa, con una documentación muy completa. basada en C++. Además, el código fuente esta accesible a todos los usuarios suscritos a Unreal Engine. Existen diversos tipos de licencias según la utilización que se le de al motor:

- **Licencia Publishing:** Para aquellos proyectos cuyo objetivo es su monetización. Aunque gratis al principio, Epic Games se llevara un 5 % del ingreso adquirido del producto una vez hay alcanzado el millón de dolares en ingresos brutos:
- **Licencia Creators:** A diferencia de la publishing es para proyectos que no se van a monetizar, de uso personal o empresarial. Es totalmente gratis.
- **Personalizada:** Ofrece todas las características y el precio es negociado. Para empresas.
- **Unreal Entreprise Program:** También para grandes empresas, ofreciendo todas las características, a un precio fijo anual de 1500$, además de quitar la comisión por "*royalties*"

Con cualquiera de estas opciones de licencia se obtiene acceso a todos los materiales de aprendizaje, entre los que hay más de 100 horas de videotutoriales y webinars regulares. Con la licencia Personalizada y Enterprise se da la opción de aprendizaje privado.

Unreal Engine también tiene una amplia comunidad de miembros subscriptores, donde se puede resolver dudas gracias a otros miembros en foros. Se puede, también, comprar la opción de soporte premium con las licencias Personalizadas y Enterprise, donde empleados de Epic Games resolverán las preguntas.

El motor permite que los proyectos se conecten a Internet, aunque no incluye servidores dedicados, teniendo que ser el cliente el que proporcione sus servidores para sus servicios en linea. Es posible la instalación de plugins que permitan la conexión a servidores dedicados proporcionados por otras empresas.

3.2.1.2 Unity

Al igual que Unreal Engine, Unity [13] es un motor utilizado en diferentes sectores, como los videojuegos (su principal mercado), la construcción o el automovilismo. Desarrollado por Unity Technologies, ofrece una plataforma de desarrollo para cualquier videojuego destinado a la mayoría de plataformas, ya sea 2D o 3D, de un jugador o multijugador.

Unity, de manera similar a Unreal Engine, ofrece una gran cantidad de vídeos, tutoriales y cursos gratis para aprender a usar la plataforma, además de un soporte basado en foros de la comunidad de usuarios donde es fácil de encontrar respuestas a dudas, ya sea sobre la interfaz, las físicas o la API que utiliza C#.

Unity se ha convertido en la plataforma de creación de videojuegos más usada actualmente [14], y de las plataformas más utilizadas por usuarios que empiezan a crear videojuegos, lo cual se refleja en el soporte de la comunidad, que es más extenso que el de Unreal Engine.

La plataforma ofrece diferentes licencias, adaptadas a la necesidad de la persona o equipo:

- **Estudiante:** Preparado para estudiantes mayores de 16 años en una institución educativa acreditada. Es gratuita y cuenta con un acceso a assets de alta calidad.
- **Personal:** Para cualquier persona con ingresos menores de 100 mil USD en los últimos 12 meses. Contara con la funciones básicas para crear y optimizar juegos de ultima generación.
- **Plus:** Opción de 399$ por año y persona que, aparte de la plataforma básica de desarrollo, ofrece herramientas de diagnostico avanzadas.
- **Pro:** opción de 1800$ por año y puesto, que ofrece todo lo de la licencia Plus pero con acceso al código fuente de la plataforma, la posibilidad de crear contenido para plataformas cerradas (Consolas de Nintendo, Xbox, Playstation, etc) y soporte técnico oficial de Unity.
- **Empresa:** Opción de 4000$ por mes y 20 puestos, que ofrece todo lo que ofrece la licencia Pro con unos conjuntos de herramientas de soluciones específicas para cada sector de la industria y servicios de entrenamiento de empleados bajo demanda, resolviendo las dudas de los trabajadores en cuanto surgen.

3.2.1.3 Godot

Godot [15] es una plataforma libre exclusivamente para el desarrollo de videojuegos. Desarrollada y utilizada por la empresa argentina OKAM, fue lanzado posteriormente al publico bajo una licencia libre permisiva, es decir, el software puede ser distribuido como software libre o propietario.

Aunque también es utilizada para entornos 3D, destaca en los 2D por su poder de tratar a los píxeles como unidades, lo que es perfecto para juegos con estética de PixelArt, (ver figura 3.3) . Además permite la elección de múltiples lenguajes para scripting, como C++, C#, GDScript(basado en Python) e incluso *scripting visual*.

Figura 3.3: Ejemplo de arte PixelArt

Debido a la multitud de posibilidades de scripting y al acceso a el código fuente en todo momento lo hace una buena plataforma para la inclusión de módulos o herramientas, desarrolladas por el usuario o cualquier otra persona o empresa, que acompañen a las ya incluidas con el software base.

A diferencia de las otras alternativas ya vistas, Godot no tiene tutoriales ni vídeos para el aprendizaje, apoyándose únicamente en el soporte de la comunidad que, aunque activo, no es tan extenso como lo es el de Unity o el de Unreal Engine, debido a la menor cantidad de usuarios que usa esta plataforma.

Una característica adicional es que en la pantalla inicial del programa podemos elegir entre una serie de plantillas de proyectos que pueden servir como tutorial o ejemplos de diversas funcionalidades del programa. Estas plantillas son bastante variadas, como por ejemplo, una calculadora simple, una implementación de una cámara de un First Person Shooter, un sistema de guardado y cargado, o un juego multijugador.

Otro de los puntos positivos de esta plataforma es que no necesita instalación, se descarga un archivo comprimido desde GitHub y el programa se puede abrir mediante un ejecutable.

3.2.1.4 GameMaker Studio

GameMaker [16] es otra plataforma exclusiva para el desarrollo de videojuegos. Desarrollado por Yoyogames, es una plataforma escogida por desarrolladores independientes por poder trabajar sin necesidad de saber de un lenguaje de programación.

La plataforma afirma ser una de las más avanzadas en el desarrollo de videojuegos 2D y, aunque esta especializado en este tipo, es posible crear entornos tridimensionales. Su principal característica es el uso de DnD (Drag and Drop), un lenguaje de scripting visual que permite a usuarios que no conocen la programación tener una alternativa fácil de aprender. Para usuarios con más experiencia en la programación también se ofrece una API de GML, un sublenguaje de XML.

YoyoGames ofrece los siguientes tipos de licencia de su plataforma, que todas permiten el acceso completo al IDE y los materiales de aprendizaje, diferenciándose solo en las plataformas a las que permiten exportar los videojuegos:

- **Gratuita:** Únicamente permite la exportación del proyecto a GXC, la tienda exclusiva del navegador Opera GX.
- **Creator:** Por 4,25 al mes puedes exportar tus juegos a plataformas de escritorio, como Windows , macOs y Linux.
- **Indie:** Por 8,19 al mes permite además la exportación de proyectos a HTML5, plataformas moviles y la Plataforma Universal de Windows, que incluye XBOX.
- **Enterprise:** Por 67,99 permite las exportaciones a consola, aunque no incluye las licencias de las propias empresas desarrolladoras de esas consolas (Nintendo switch, Playstation, etc), teniendo que adquirirlas a parte.

YoyoGames, al igual que Godot, no ofrece ningún tipo de soporte técnico, dejando estos problemas en una comunidad de usuarios, para los que existe un foro dedicado. Sin embargo,

sí que hay videotutoriales y documentación para el aprendizaje de este motor.

Al igual que Unity y Unreal Manager, Gamemaker ofrece una tienda aparte donde poder conseguir assets para los videojuegos o herramientas externas para añadir funcionalidades a la plataforma.

La instalación es sencilla y no requiere de tanto espacio de disco como Unity y Unreal Engine, solo se necesita crear una cuenta, elegir una licencia y descargarnos el instalador. Para las pruebas se ha elegido la licencia gratuita, aunque no se pueden hacer exportaciones de juegos a dispositivos móviles.

3.2.2 Soluciones Back-End

En esta sección se explora diversas soluciones *back-end* para la implementación de la opción multijugador.

3.2.2.1 Photon

Photon [17] ofrece varias soluciones para multijugador dependiendo del tipo y necesidades. Photon Realtime [18] es el producto que sirve de base y proporciona los servidores, herramientas y conexiones básicas cliente-servidor para el desarrollo multijugador. Entre sus características se encuentra el guardado en Cloud mediante *webhooks*, partidas entre diferentes plataformas (*Cross-Platform*) y la escalabilidad para alojar usuarios.

Existen otros dos conjuntos de herramientas que se construyen sobre el ya mencionado Photon Realtime. Uno de ellos es Fusion [19], un SDK para crear un sistema de juego más fluido, incluso con latencias más altas. Esto lo consigue con predicción *Client-side*, que permite al cliente elaborar una respuesta de las entradas del usuario sin necesidad de que el servidor actualice el estado.

Mediante el uso de *Snapshots*, una captura de la información del estado actual del juego, Fusión permite el interpolado. El *interpolado de Snapshots* consiste en la obtención de un estado perdido en la red, mediante el uso dos *Snapshots* de los que se calcula un valor medio entre los para calcular el valor aproximado del valor perdido. Esto, sin embargo, introduce un cierto retraso ya que no se mostrara el estado más actual nunca [20]. Ver figura 3.4

El otro conjunto de herramientas proporcionado por Photon es Quantum [21], que trata de minimizar el uso de la red realizando solo la transmisión de entradas entre los jugadores. Quantum obtiene los entradas del servidor y ejecuta una simulación en local, con las predicciones de entradas y *rollbacks*, el retroceso hacia un estado anterior. Esto hace que el peso de la ejecución recaiga sobre los usuarios, lo cual lo hace perfecto para juegos que no requieren de cálculos de físicas e interacciones con el medio y solo necesitan las entradas del jugador (juegos de lucha por ejemplo).

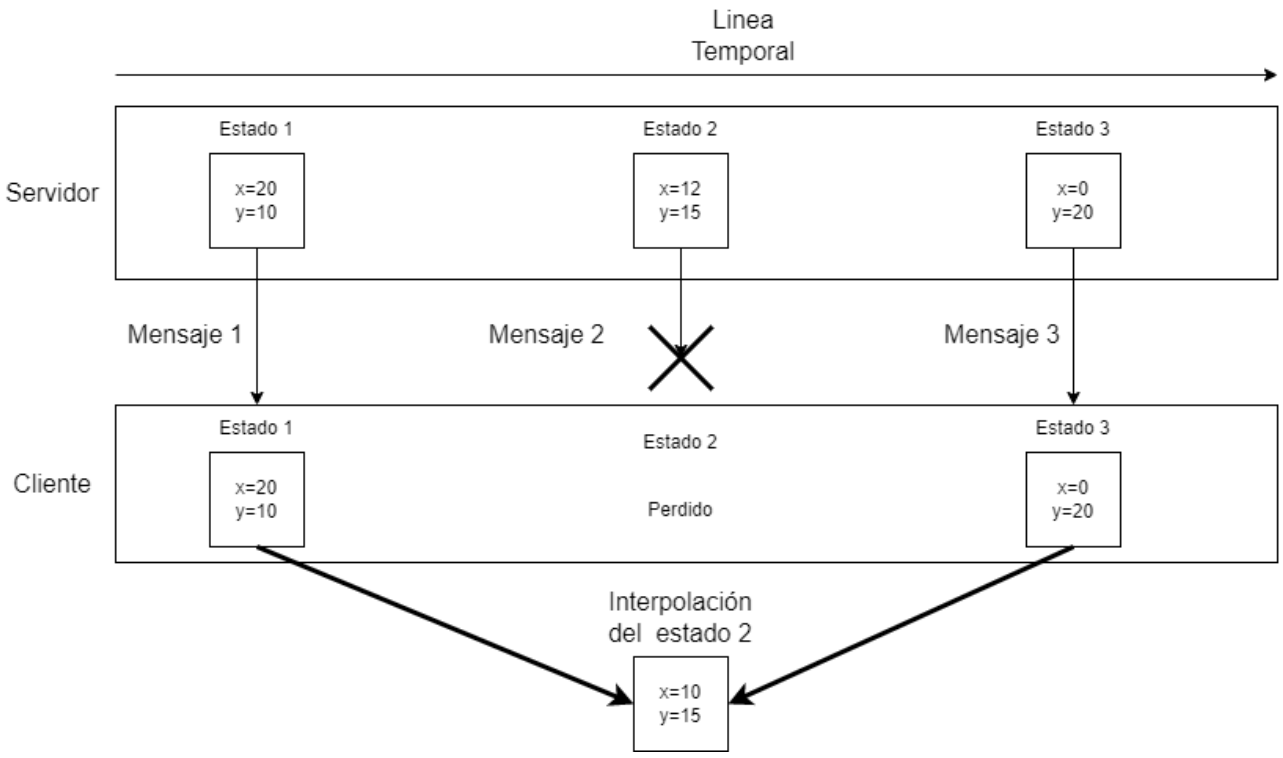

Figura 3.4: Interpolado de Snapshots

Las licencias de estos productos son todas iguales, teniendo una versión gratuita con un máximo de 20 jugadores simultáneos, 500 mensajes por segundo en cada Sala (agrupación de jugadores que juegan la misma partida) y 60GB de trafico total mensual. El siguiente nivel de licencia varía según el producto elegido, pero permitiría 100 usuarios simultáneos y 0.3 TB de trafico mensual por 95 euros por Fusion y 125 en el caso de Quantum.

3.2.2.2 Firebase

Firebase [22] es una plataforma de desarrollo de aplicaciones respaldada por Google. La plataforma se compone de una serie de productos o herramientas para construir, monitorizar y lanzar al mercado aplicaciones. Entre los servicios que proporciona están los servidores cloud, herramientas de autenticación, analíticas de Google o monitores de rendimiento.

Entre las herramientas que ofrece Firebase nos interesarían Cloud Firestore y Realtime Database. Ambas son bases de datos NoSQL, que ofrecen SDKs para usar servidores sin la necesidad de desplegarlos y mantenerlos. Ambas bases de datos mantienen a los usuarios conectados una actualización de cambios en tiempo real.

Realtime Database [23] es la base de datos original de Firebase, que permite la sincronización de estados entre usuarios con poca latencia. Estructura la base de datos mediante un árbol de elementos JSON. Con esta base de datos se puede crear Scripts que se ejecutan cuando hay cambios en algún de los nodos de este árbol.

Cloud Firestore [24] por su parte utiliza un modelo de datos más intuitivo que permite consultas más amplias que la anterior. Firestore usa colecciones y documentos y también permite estructurar los elementos similarmente a un árbol. Permite la ejecución de un código *back-end* que responder a cambios de la base de datos.

Ambas herramientas son muy similares, variando la manera de estructurar los datos. Para decidir que base de datos escoger se tienen en cuenta una serie de consideraciones presentadas en la tabla 3.3

Uso	**Realtime Database**	**Cloud Firestore**
Consultas	Para sincronizar datos, con consultas básicas	Para consultas avanzadas, ordenación y transacciones.
Operaciones en datos	pocos GB de datos que cambian frecuentemente	Muchos GB o TB de datos que se leen más que cambian
Disponibilidad	99,95 % del tiempo en linea	99.999 % de tiempo en linea
Consultas offline en local	Raramente	Raramente o Frecuentemente
Numero de instancias	Una instancia o muchas	muchas instancias

Tabla 3.3: Realtime vs Firebase [25]

Otra de las herramientas ofrecidas por firebase es la autenticación de usuarios mediante Firebase Authentication [26], que permite la identificación de usuarios usando un email y contraseña, un teléfono o una cuenta en Google, Twitter, Facebook o Github. Las bases de datos pueden apoyarse en esta herramienta para la monitorización de usuarios y permisos.

3.2.2.3 GameSparks

GameSparks [27]es una solución basada en Cloud que permite la creación de características de servidor para cualquier juego, y para muchas plataformas ya sea para Android, Steam, Playstation o Xbox y permite ser integrado en los motores Unity y Unreal Engine. Permite crear una API del lado del servidor y scripts HTML para configurar la lógica de los objetos dentro del juego. Permite la ejecución de códigos tras eventos, procesando solicitudes, mensajes o logs. Además, permite la integración de github para la sincronización de los scripts y un historial de cambios dentro de la plataforma.

GameSparks ofrece servicio de Base de datos NoSQL basado en MongoDB para trabajar con colecciones. Estas colecciones pueden usarse para guardar datos relativos a la partida como estadísticas de un jugador. La herramienta ofrece un explorador para organizar o manejar estas colecciones.

Para la comunicación entre jugadores, GameSparks permite la creación y cargado de archivos binarios con la información de cada uno de los jugadores. Dentro del juego podemos obtener una URL segura donde se encuentran almacenados los archivos para la descarga de estos. Estos

binarios se pueden administrar, permitiendo elegir los archivos qué se quieren cargar y realizar un seguimiento de su progreso.

Otros servicios que ofrece la plataforma es la autenticación de usuarios (mediante Facebook, Twitter, Google, Apple, Microsoft, Amazon y Steam), notificación y mensajería y la planificación de diferentes eventos, ya sea el envió de mensajes o la ejecución de scripts de copias de seguridad.

GameSparks ofrece los siguientes niveles de suscripción para utilizar el servicio [28]:

- **Development:** Incluye las herramientas esenciales, pero no permite publicar el juego ni realizar algún tipo de *Matchmaking* (emparejamiento de jugadores en una partida). Esta es la versión de prueba que es gratis, pero solo permite 100 usuarios simultáneamente.

- **Standard:** Por 299$ al mes permite la publicación de juego en vivo y con ello la posibilidad de tener 10.000 jugadores simultáneamente, pero sigue sin tener la posibilidad de MatchMaking.

- **Enterprise:** Solución para empresas para negociar un contrato con las características ofrecidas y el precio a pagar.

3.2.2.4 PlayFab

PlayFab [29] es la solución *back-end* propuesta por Microsoft Azure. Se trata de un plataforma completa que ofrece servidores dedicados, autenticación de jugadores, administración de datos de jugadores, chat de voz y texto, emparejamiento, y otras herramientas para la retención de audiencia, *Testing* de versiones *alphas* y *betas*, y analíticas en tiempo real. Según la plataforma que ese elija, ofrece diversos SDKs, para Java, C# o C++.

De entre los servicios más destacables se encuentran:

- **Servidores dedicados:** De poca latencia y alta fiabilidad, con alcance global y protección ante ataques *DDoS*. Se trata de una infraestructura de servidores con opciones de escalado dinámico [30].
- **Identidad y datos multiplataforma:** Permite la sincronización de estados de juego y usuarios de diferentes plataformas de juegos, ya sea Xbox, Playstation, Facebook, etc [31].
- **PlayFab Party:** Tecnología de chat de texto y de voz segura flexible y de poca latencia [32].
- **Emparejamiento:** PlayFab presenta su sistema de emparejamiento para encontrar jugadores basados en tickets y colas. Un ticket consiste en un jugador o grupo de jugadores que

quiere jugar una partida, y la cola son una colección de tickets y reglas para emparejarlos [33].

PlayFab tiene un sistema de precios *pay-as-you-go*, que te cobra según el uso que se le de a este servicio [34]. Algunos ejemplos de métricas cuantificables usadas para el calculo del precio son el numero de lecturas y escrituras en la base de datos, la cantidad total de datos almacenados y el numero de peticiones al servidor. Sin embargo tiene otros niveles mensuales que ofrecen un determinado número de métricas que si se superan se aplica el *pay-as-you-go*. Estos niveles son:

- **Free to Start:** Permite un total de 10 videojuegos conectados a PlayFab de hasta 100.000 usuarios por cada uno. Permite 150K consultas gratis de hasta 1 MB por 0 euros al mes que, si se superan, se aplican tarifas Pay-as-you-go. Cada millón de peticiones adicionales cuestan 7$.
- **Standard Plan:** Por 99$ al mes permite 150.000 peticiones que si se superan se aplican tarifas *pay-as-you-go*. Cada millón de peticiones adicionales cuestan 6,50$
- **Premiun Plan:** Por 1.999$ al mes permite 150.000 peticiones que si se superan se aplican tarifas *pay-as-you-go*. Cada millón de peticiones adicionales cuestan 6$

3.2.3 Herramientas Utilizadas

En esta sección se discuten las herramientas utilizadas para la creación del videojuego. Para la elección de estas herramientas se ha tenido en cuenta lo estudiado en los apartados anteriores y las características del videojuego que vamos a crear, que se describiría en el apartado 4.

3.2.3.1 Motor de videojuego

El videojuego planteado no requiere de capacidades gráficas avanzadas, sin embargo, necesita poder ser exportado a un dispositivo móvil. De los motores analizados, GameMaker Studio es un motor bastante simple que aunque posee las suficientes capacidades para realizar este proyecto, su version gratuita no permite la exportación a sistemas operativos Android.

Godot por su parte sí permite la exportación a dispositivos móviles, ya que al ser una plataforma libre no posee las restricciones de licencias. Es una herramienta capaz de "mover" el proyecto planteado, aunque nos limita las opciones que podríamos usar como solución *back-end*, debido a que no muchas empresas desarrollan para esta plataforma debido a que es menos conocida. El ser poco conocida también tiene la consecuencia de que aprender a usarla es más complicado debido a que la ayuda de la comunidad es más pequeño.

Las opciones restantes son Unity y Unreal Engine, plataformas con gran soporte y bastante usadas por toda la industria del videojuego. Ambas son muy potentes y se puede escoger casi cualquier solución *back-end* de las analizadas anteriormente. Las licencias gratuitas de cada plataforma permiten la exportación a Android.

Unreal Engine es una herramienta gráfica demasiado potente, que requiere de ordenadores con bastante capacidad gráfica. Cuando se hizo una instalación de prueba no se pudo iniciar el programa debido a que requería las versiones de controladores más actualizados de Direct X 11 o 12. Por desgracia la gráfica del ordenador usada para el desarrollo, Windows HD Graphics 2000, soporta solo hasta la versión 10_1, que lo hace incompatible con este programa.

Por lo tanto se ha escogido Unity. Esta es una herramienta compatible con el ordenador usado para el desarrollo y cumple los requisitos de este trabajo. Con la licencia personal que ofrece la compañía se obtiene lo suficiente para el desarrollo del proyecto. Tras crear una cuenta, se puede instalar UnityHub, una aplicación de inicio, que permite ejecutar los proyectos creados, conectar con la comunidad, acceder a los tutoriales e instalar o desinstalar las versiones de Unity que deseemos. La versión utilizada sera la 2019.4.1f1 por razones de compatibilidad con la solución *back-end* elegida en el siguiente apartado.

3.2.3.2 Back-End

Unity, además del motor, ofrece plugins, herramientas y paquetes para crear juegos multijugador. Algunas de estas herramientas se alojan en el marco multijugador de Unity llamado Netcode [35], que es open source:

- **Vivox:** Se tarta de el chat de voz o de texto que ofrece Unity, para permitir que los jugadores de la partida hablen entre ellos. Presenta tecnologías de accesibilidad como *text-to-speech* o *speech-to-text* [36].
- **Relay:** Conecta los jugadores usando *P2P* usando un entorno de cliente-servidor. Facilita, pues, el entorno multijugador sin la necesidad de un servidor dedicado [37].
- **Lobby:** Facilita el emparejamiento de jugadores y la configuración de opciones antes de cada sesión de juego, permitiendo la creación de salas publicas y privadas [38].

Netcode, Relay y Lobby, aunque se integran bien en Unity, están en fase *beta*, por lo que estas soluciones pueden contener algún *bug* que impida el correcto funcionamiento del sistema.

Con estas opciones descartadas, elegimos entre las opciones analizadas en el apartado anterior. El juego requerirá de muchos accesos al servidor para saber la posición de los jugadores que, a su vez, tendrán que ser notificados cuando alguno de estos jugadores realice una acción que suponga cambios en el servidor.

Todas las opciones analizadas anteriormente cumplen los requisitos necesarios para la construcción del entorno multijugador. Lo ideal sería utilizar cada uno de estos servicios y realizar pruebas de diagnostico y rendimiento para concretar la mejor opción. Sin embargo, esto resultaría bastante costoso tiempo y, por lo tanto, se ha decidido escoger la herramienta según las licencias disponibles gratuitas.

GameSparks no permite la publicación del juego en ninguna plataforma por lo que la opción queda descartada. De entre las otras tres se ha escogido Firebase Realtime Database, que

ofrece mayor cantidad de jugadores simultáneos que Photon, y no tiene un limite de tamaño de los mensajes como PlayFab. Esta limitado a 1GB en total de almacenamiento y a 10 GB mensuales de descarga, que no es mucho pero es más que suficiente para el prototipado del proyecto.

Se ha elegido Realtime Database a Cloud Firestore porque las características del proyecto requieren de muchas consultas pequeñas y no de pocas consultas grandes. La sincronización de datos no requerirá consultas de tamaño grande.

3.2.3.3 GIMP

GNU Image Manipulation Program (GIMP) es un programa de edición de imágenes *open source*. Aunque principalmente todos los assets gráficos del videojuego serán obtenidos de repositorios gratuitos, esta aplicación sera utilizado en ocasiones si se quiere modificar algunas de las imágenes obtenidas.

GIMP permite la modificación pixel a pixel y cuenta con herramientas de recorte selección, escalado y transparentado. Permite exportar las imágenes en formatos PNG, JPG o GIF.

3.2.3.4 Texmaker y LaTeX

LaTeX es un sistema de composición de textos muy utilizados en artículos académicos basado en instrucciones. Este documento utiliza este sistema.

Texmaker es un plataforma de edición de LaTeX distribuido bajo la licencia GPL (General Public License).La edición de LaTeX de este documento se ha llevado a cabo a través de esta plataforma.

3.2.3.5 Astah

Astah es una herramienta de modelado UML (Unified Modeling Language) para la creación de diagramas en el proceso de diseño y análisis software. Los diagramas creados en las secciones de diseño y análisis han sido creadas usando esta herramienta, con el uso de una licencia proporcionada por Escuela de Ingeniería Informática de la Universidad de Valladolid.

3.3 Unity: Descripción y Elementos Básicos

En esta sección se describen algunos de los elementos básicos que ofrece Unity.

3.3.1 Scenes

Las escenas o “Scenes” son las zonas donde se construye el proyecto siguiendo el esquema que usa Unity [39]. En estas escenas se colocan los escenarios, personajes, obstáculos o interfaz

de usuario (UI) y la posición de estos objetos están representados por una coordenada X,Y,Z. Se pueden tener más de una de ellas en cada proyecto, cada una con diferentes elementos. Cada una de estas escenas viene con un *GameObject* (ver apartado 3.3.2) con una cámara predeterminada.

La cámara [40] es la encargada de capturar la escena y hacerla plana para mostrarla al usuario. La posición de la cámara determina lo que muestra con el Z de su coordenada, definiendo la dirección de la vista dependiendo si es negativa o positiva.

Existen dos modos de proyectar la vista de la cámara, la primera es similar a como el ojo humano funciona, haciendo los objetos más lejanos en pequeño y los más cercanos más grandes. Este efecto es llamado *perspectiva*, que se usa cuando se intenta crear una escena realista para dar información al usuario de profundidad. La otra forma de proyectar ignora estas reglas y renderiza los objetos de manera uniforme. Esta forma de proyección es la *ortográfica*.

Ambas proyecciones definen unos limites de lo que pueden observar desde su posición actual conocidos como *"Clipping Planes"*. Estos limites son definidos por dos planos perpendiculares al eje Z de la cámara, el lejano y el cercano. La cámara renderiza los objetos situados dentro de estos dos planos. En una cámara ortográfica los dos planos son rectángulos de igual tamaño creando un cono de visión semejante a un paralepípedo rectangular mientras que la cámara en *perspectiva* utiliza un *clipping plane* cercano más pequeño que el lejano, haciendo que su cono de visión sea una pirámide truncada (ver figura 3.5).

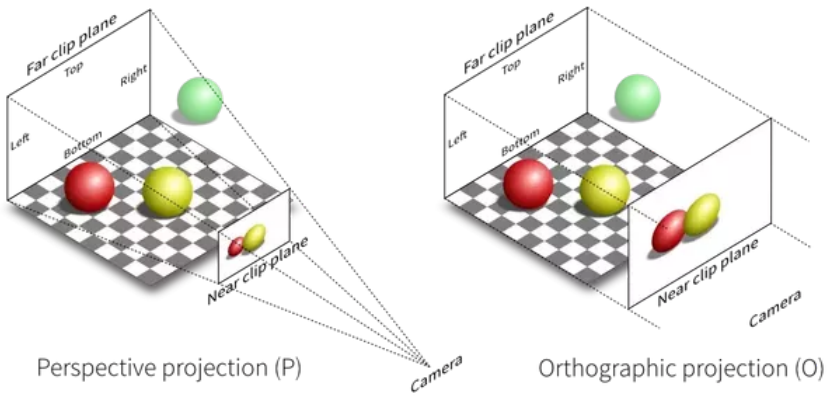

Figura 3.5: Diferencia entre las proyecciones

La cámara permite renderizar los objetos que deseamos usando capas y una mascara denominada *cullingMask* [41]. Por defecto existen unas capas predeterminadas [42] (se pueden crear más) y la cámara renderizará todas ellas. Sin embargo, podemos desactivar el renderizado de alguna de estas capas y los objetos que estén en esa capa no aparecerán para el usuario. Esto es útil para juegos con mundos abiertos que renderizan muchas elementos, pudiendo hacer invisibles elementos lejanos para mayor rendimiento gráfico.

3.3.2 GameObjects

Los *GameObjects*[43] son los elementos básicos en Unity que representan los personajes, obstáculos o el escenario. Sirven de contenedores para los componentes, que son los que implementan la funcionalidad. Todos ellos tienen por defecto el componente *Transform*, que contiene la información sobre las coordenadas, rotación y escala con respecto a la escena en la que son mostradas.

En *Transform*, la posición de un objeto esta establecido en un vector de 3 dimensiones, siendo X el eje horizontal, Y el eje vertical y Z el eje de profundidad. La orientación de un objeto esta representada por los ángulos de Euler, siendo X, Y,Z los grados del ángulo que el objeto se traslada con respecto a los planos perpendiculares a los ejes X,Y,Z. La escala representa el tamaño del objeto respecto al original siendo 1, por defecto, 0.5 la mitad, y 2 el doble de tamaño.

Los *GameObjects* pueden ser hijos de otros *GameObjects* y estos a su vez ser padres de otros. Esto supone que el *Transform* de los hijos tomara como punto de referencia el padre en vez de la escena. Esto supone una ventaja para elementos con muchas piezas que se deban mover juntas, porque moviendo el padre por la escena, mueves al hijo. Un ejemplo seria un personaje formado por dos imágenes cabeza y cuerpo, si una es hija de la otra solo es necesario mover al padre para mover el personaje por completo.

Otros componentes importantes que pueden tener un *GameObject* son los *Collider* o colisionadores [44]. Se trata de una serie de polígonos que detectan cuando entran en contacto con otros *colliders*. Unity ofrece *colliders* de diversas formas, ya sean cuadrados, círculos u otros polígonos en el caso de los juegos 2D, y cúbicos, esféricos o una malla de polígonos en el caso de los 3D. La forma y tamaño de estos polígonos se pueden modificar, ajustándose a los limites deseados para cada *GameObject*.

Para que a un *GameObject* se le apliquen las físicas del juego debe tener un componente *Rigidbody* o *Rigidbody 2D* [45] según el tipo de juego. Al hacer esto el objeto pasa bajo control de motor de físicas de Unity o del usuario. Existen tres tipos de cuerpos que definen el comportamiento del movimiento e interacciones con *Colliders*:

- **Dinámico:** A un *rigidbody* dinámico se le aplican todas las físicas del motor de Unity. Los objetos con este tipo de cuerpo tienen una masa finita y se ven afectados por las fuerzas de la gravedad, valores que se pueden modificar, junto con el coeficiente de fricción afectado al movimiento de posición y rotación.
- **Kinemático:** Un objeto kinemático esta diseñado para moverse bajo simulación pero bajo control del usuario y sus Scripts en vez del motor de físicas. No se ve afectado por masa ni gravedad.
- **Estático:** El objeto no se moverá, no se vera afectado por la gravedad ni masa y en cualquier interacción con algún objeto dinámico se comportara como un objeto inamovible

Salvo el estático, los *rigidbodys* permiten la interpolación o extrapolación de frames para salvar recursos del sistema, entre actualizaciones del motor de físicas. El interpolado suavizara

el movimiento en base a la posición del objeto en frames anteriores, mientras que el extrapolado intentara predecir el movimiento del objeto en el siguiente frame.

Por defecto, los *Collider* colisionaran con otros *Collider* y no podrán sobrelaparse el uno al otro, aplicándose físicas normales (dos objetos físicos no pueden estar en el mismo espacio). Esto sin embargo puede cambiar si uno de los *Colliders* se convierte en un *Trigger* o disparador, que envía un mensaje al *GameObject* que lo tenga, indicando que un objeto a entrado en su área, permitiéndole ejecutar un evento. Los *Collider* pueden estar asociados a un material físico, que permite ajustar el coeficiente de fricción del *Collider* o el grado en el que rebota de una superficie.

Imaginamos que creamos un *GameObject* con algunos de estos componentes. Si queremos usarlo podemos introducirlo a la escena. Pero si queremos usara varias copias de este objeto en la misma o diferentes escenas tenemos que convertirlo en un *Prefab* [46]. Un Prefab o prefabricado es un *GameObject* cuyos componentes y características son guardadas para la fácil replicación de este objeto.

3.3.3 Sprite Management

Los *Sprites* [47] son objetos gráficos de dos dimensiones, ya sean texturas en entornos 3D o los propios gráficos de un juego 2D. Para la administración de estos componentes gráficos Unity ofrece los siguientes herramientas y componentes:

- **Sprite Creator:** Es una herramienta para crear *placeholders*, que sirven como *Sprites* temporales.
- **Sprite Editor:** Permite la extracción de varios elementos gráficos dentro de un mismo sprite, puestro que a veces es más fácil importarlos todos con una sola imagen dentro de Unity.
- **Sprite Renderer:** Es un componente asociado a un *GameObject*. Permite el cambio de sprite, selección de un color de renderizado (aplicado al color del sprite), la manera en la que la imagen escala cuando se redimensiona y la capa de renderizado. Esta ultima permite elegir, cuando un objeto solapa a otro, quien sera dibujado por encima del otro, siempre y cuando estén en el mismo plano XY. Objetos en una misma capa pueden darse una prioridad entre unos y otros.

A la hora de importar una imagen a Unity y convertirla en sprite se pueden seleccionar diferentes configuraciones.

- **Sprite Mode:** Permite seleccionar cuantos sprites contienen la imagen, para seleccionar como deben extraerse en el Sprite Editor.
- **Pixeles por Unidades:** Permite seleccionar cuantos pixeles de la imagen corresponden a una unidad de espacio de Unity
- **Pivot:** Configura el punto de la imagen que se usara como referencia como su centro de posición y rotación.

- **Modo de Filtrado:** Determinara la manera que la imagen se mostrara conforme se acerque o aleje a una cámara con perspectiva.

- **Tamaño máximo:** Cambia el tamaño máximo de espacio que ocupa una imagen, útil para limitar el uso de almacenamiento. Si se excede esta cantidad un algoritmo de cambio de tamaño, comprimirá la imagen con la calidad seleccionada.

3.3.4 Animaciones

Para dar una sensación de movimiento y cambio son necesarias unas Animaciones. Unity ofrece dos herramientas para crear, modificar y administrar estas animaciones.

La ventana de Animación [48] de Unity permite la introducción de imágenes en una linea temporal para la creación de imágenes mediante superposición de imágenes. Estas imágenes serán mostradas al usuario a una velocidad marcado por la tasa de muestreo introducida en las opciones de la ventana. La tasa de muestreo marca cuantos fotogramas serán mostrados por cada segundo.

En ciertos caso, la animación viene con alguna acción del *GameObject* que lo realiza, por ejemplo, un personaje blande una espada en la animación y si esa espada contacta con algún enemigo le hace daño. Para ello a la animación se le puede añadir algún componente existente del *GameObject*, como su *Transform*, *collider* o *Rigidbody*. En cada una de los fotogramas de esa animación se pueden modificar cada uno de los valores de esos componentes, por ejemplo, haciendo aparecer un *Collider* correspondiente a la espada, o que el personaje se mueva un poco hacia adelante al blandirla y luego vuelva a su posición original.

Otra manera de animar es usando un conjunto de *Sprites* que componen un objeto animado. Esto seria mediante el cambio de posición de cada una de las partes. Usando el ejemplo anterior seria como si tuviéramos el personaje por un lado y su brazo y espada por el otro, moviéramos la posición de esa espada para animar.

Es posible que varios elementos tengan varias animaciones, por lo que Unity ofrece Animator [49] para la administración de estas (ver figura 3.6). En él se puede definir un diagrama de estados donde puedes definir cuando y en que orden estas animaciones son reproducidas. También se puede usar un parámetro definido en algún script que actué como *Trigger* de alguna animación (Si velocidad mayor que x $\rightarrow$ animación de correr).

3.3.5 Canvas

El *canvas* [50] es un *GameObject* que sirve como cuadro para colocar los elementos de la interfaz, puesto que es padre de todos los elementos de la IU. El área del *Canvas* se muestra como un rectángulo en la escena y los elementos de IU son renderizados en esta, con orden de primer hijo a último hijo dentro de la jerarquía. Existen tres tipos de modo de renderizado del *Canvas*:

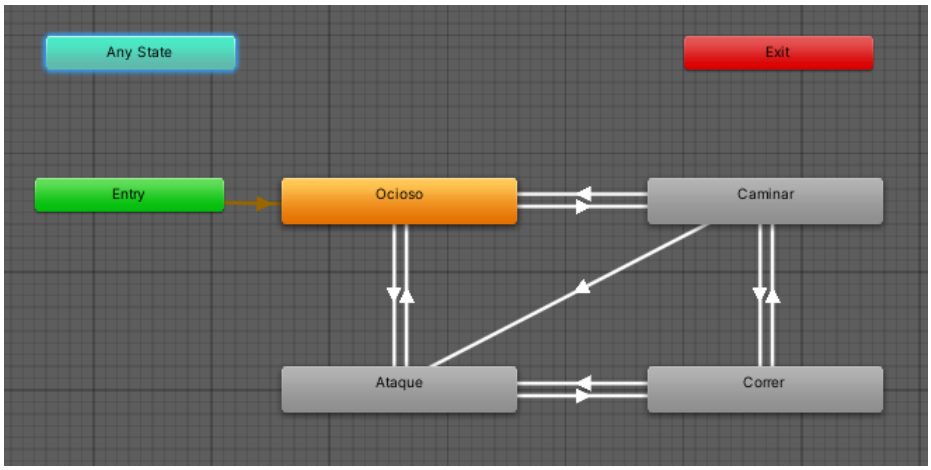

Figura 3.6: Diagrama de estados de la animaciones

- **Espacio de Pantalla - Superposición:** Los elementos de la pantalla aparecerán según como se distribuyan en el espacio del *Canvas* y se ajustara si cambia el tamaño de la pantalla o resolución.

- **Espacio de Pantalla - Camara:** Similar al anterior pero el *Canvas* se coloca a una distancia de la cámara. En el modo ortográfico de esta cámara no cambia mucho pero en el modo perspectiva la interfaz se mostrara siguiendo esta perspectiva y se puede distorsionar cambiando el Campo de Visión de la cámara o la distancia a esta.

- **Espacio del Mundo:** Los elementos del *Canvas* se comportaran como un objeto normal de la escena. La interfaz se renderizará siguiendo las reglas basadas en su colocación en la escena (si esta detrás de otro objeto, lo tapará).

El *Canvas* viene acompañado de un *EventSystem* [51], un administrador de envió de eventos basados en una entrada de algún elemento de esta interfaz. *EventSystem* controla que objeto de la interfaz es el seleccionado, qué función o llamada realizar si es seleccionado y a qué objeto esta dirigido esa llamada.

Todos los elementos de la interfaz, en vez de un *Transform*, reciben un *Rect Transform* [52], que permiten almacenar la posición, tamaño, pivotes y puntos de anclaje del elemento. Un punto de anclaje [53] es una posición que permite a la IU tener un punto de referencia en la escena del que no se moverá de posición. Viene bien para mantener la interfaz en el mismo sitio sin importar la resolución y el tamaño de la pantalla.

Los elementos que se pueden usar en la interfaz incluyen textos, imágenes, botones, chekmarks, scrollbars y dropdowns. Algunos de estos elementos usan otros como por ejemplo textos e imágenes. El elemento Texto permite, además, cambiar el tamaño de letra, la fuente, el color y la alineación. Las fuentes de texto pueden ser customizadas importando ficheros de fuente (.ttf o .otf)[54].

3.3.6 Input Actions Assets

Un *Input Action Asset* contiene información correspondiente a las acciones de cada entrada de usuario, sus correspondientes *bindings* [55] y los esquemas de control. Una acción en este contexto es cualquier cosa que quiere hacer el usuario y un *binding* lo relaciona a la entrada (pulsar tecla, mover joystick, etc) correspondiente para realizar esta acción.

En estos assets podemos crear un Mapa para agrupar diferentes acciones. Tres tipos de Acciones se pueden seleccionar [56]:

- **Botón:** Perfecto para entradas como teclas o botones y actúa como un *Trigger*. Una acción de un botón ejecuta un evento y espera hasta que vuelva a ser llamado, pulsándose otra vez.
- **Valor:** Devuelve un valor de los esquemas de control conectadas. Es usado, por ejemplo en joysticks, que devuelven un Vector de 2 dimensiones dependiendo de la posición del stick o un touchscreen con las coordenadas de la posición del dedo. Esta acción se activara cada vez que se lea un valor que no es por defecto.

  Si diferentes entradas realizan la misma acción, el sistema realizara una desambiguación, eligiendo cuál de las dos entradas hará caso y cuál ignorará. Normalmente elegirá el valor más grande dependiendo del tipo de magnitud que se está midiendo.
- **Pass Through:** funciona similarmente a Value, pero el sistema no realiza ninguna desambiguación en caso de dos inputs realizando la misma acción, haciendo las dos acciones una después de la otra.

Bindings determina la entrada del usuario que necesita usar para realizar cada acción. Para definir la entrada se debe seleccionar su path, es decir, la ruta a la dirección del archivo de la entrada. Así, para usar el joystick derecho de un mando, el path correspondiente al binding seria algo parecido a "rigth joystick [GAMEPAD]". Diferentes acciones pueden tener más de un binding.

3.3.7 Scripting

En esta sección se detallan algunos operaciones y funciones proporcionadas tanto por la API de Unity como por el API de Firebase.

Cada *GameObject* puede ser cargado con diferentes scripts de comportamiento como componentes. Estos scripts afectaran solo al *GameObject* que los aloja, aunque dentro de este Script se pueden llamara a funciones de otros *GameObjects* mediante el uso de SendMenssage().

MonoBehaviour [57] es la clase base de la cual todos los scripts Unity derivan. Para que una clase pueda ser ejecutada y simulada en el motor debe heredar esta clase. Los métodos que implementa esta clase son de invocación de llamadas como Invoke(), CancelInvoke() o InvokeRepeating(), que llama a la función pasada como parámetro, cancela las invocaciones o las va repitiendo cada cierto tiempo, respectivamente.

Sin embargo, el peso de esta clase recae en los mensajes, que son determinadas operaciones que son llamadas cada vez que se cumple cierta condición. Los mensajes más importantes de esta función son, entre otros:

- **Update():** Esta función es ejecutada en cada fotograma. Normalmente el código referido a leer los inputs de cada usuario son colocados aquí.
- **FixedUpdate():** Esta función se ejecuta independientemente a la tasa de fotogramas y se usa para el calculo de físicas. Por defecto se ejecuta 50 veces cada segundo aunque este valor puede ser cambiado accediendo a Time.fixedDeltaTime. Esta operación es útil porque permite que diferentes dispositivos con diferentes tasas de fotogramas ejecuten las físicas de manera similar.
- **Awake():** Esta operación se ejecuta cuando la instancia del script es inicializada, ya sea porque el *GameObject* que lo contiene es cargado en la escena o porque ha pasado a estar activo. Esta operación se ejecutara solo una vez en la vida del *GameObject*, a no ser que se desactive y active otra vez.
- **Start()**: similar a Awake pero es llamado cuando el componente del script es activado. Se ejecutara una sola vez a no ser que el script sea desactivado y activado otra vez. Esta función, al igual que Awake sera llamada antes que cualquier Update.
- **LateUpdate()**: Se ejecuta después de que todas las operaciones Update() de cada objeto de la escena sea llamado, util organizar para ejecuciones
- **Funciones OnCollision:** Es un conjunto de operaciones que se ejecutan en cuanto existe una interacción con *Colliders* o Rigidbodys. OnCollisionEnter se ejecuta cuando empiezan a tocarse dos colisiones, OnCollisionExit se ejecuta cuando dos colisionadores dejan de tocarse y OnCollisionStay se ejecuta cada frame en que dos colisionadores se están tocando. Estas funciones tienen una correspondiente versión para entornos 2D (OnCollisionEnter2D, OnCollisionExit2D y OnCollisionStay2D).
- **Funciones OnTrigger:**Similares a las funciones anteriores pero se ejecutan cuando el *Trigger* asociado al objeto toca otro *GameObject*.
- **OnDestroy():** Se ejecuta justo antes de que el *GameObject* asociado sea destruido.
- **OnEnable() y OnDisable():** Son ejecutados cuando el *GameObject* asociado es activado o desactivado en la escena.

Update() y FixedUpdate() son las funciones principales ya que ejecutarán el código de las físicas. Sin embargo, no pueden ser utilizadas para esperar eventos o esperar segundos, ya que pararía la simulación de físicas o la renderización de fotogramas. Para ello Unity dispone de operaciones paralelas mediante el uso de Coroutines.

Una coroutine o corrutina [58] es un método que permite la pausa de su ejecución, devolver el control a Unity y volver a ejecutar desde donde lo dejo en el siguiente fotograma. Son perfectas para funciones que ejecutan una serie de eventos a lo largo del tiempo. Estas funciones usan iteradores (IEnumerator) y la instrucción yield return para pausar la ejecución. Retornando

una instancia de Clase WaitForSeconds [59] con yield permitirá ejecutar delays. Las Coroutines se ejecutan con StartCouroutine(método()) y se pueden parar mediante StopCouroutine o StopAllCoroutines.

3.4 Scripting en Firebase Realtime Database

Para el uso de la Realtime Database, Firebase ofrece la clase *Firebase.Database* [60] para la comunicación con los elementos de esta base de datos. Dentro de esta clase existen un conjunto de subclases importantes que pasamos a describir:

La clase que permite el punto de entrada se trata de *Firebase.Database.FirebaseDatabase* [61], donde se puede encontrar una instancia por defecto de la base de datos o se puede obtener otra con GetInstance(). Desde esta clase se puede conectarse y desconectarse de la base de datos mediante GoOnline() y GoOffline(), métodos públicos de esta clase. Desde la instancia por defecto (defaultInstance) podemos obtener referencias a cualquier nodo de la base de datos introduciendo una ruta desde el nodo padre, pasando por los hijos, hasta llegar a el nodo deseado o, podemos obtener la referencia raíz (RootReference).

Una referencia de la base de datos es una clase *DatabaseReference* [62] que representa cualquier localización de la base de datos. Cada referencia tiene una Llave o Key y puede tener un valor o más hijos. Esta clase ha sido utilizada principalmente para escribir los datos en la base de datos, ya que ofrece las siguientes funciones.

- **Child(string path):** Obtiene una *DatabaseReference* a alguno de los hijos de estas referencia.
- **Pull():** Crea una referencia. Se usa en conjunto con Child() para añadir nuevos hijos, por ejemplo RootReference.child("nuevo").Push().
- **SetValueAsync(Object objeto):** Guarda los datos en la base de datos en la referencia.
- **SetRawJsonValueAsync(string json):** Guarda los datos en como un string que representa un json.
- **RemoveValueAsync():** Establece null como valor de la referencia.

En la figura 3.7 podemos observar una instancia de la base de datos. En el podemos encontrar la *RootReference*, representada con la url de más arriba de la figura. Cada uno de los nodos representa otras referencias de la base de datos, cada una con su Llave y con un valor, que pueden ser más nodos como en el caso de Nodo1, Nodo1.2 o un valor como el caso de Nodo1.1 Nodo1.2.1 y Nodo2.

Si queremos leer los datos de RealtimeDatabase podemos hacerlo mediante Eventos. La clase *Query* [63] es la encargada de ello. Ofrece una serie de escuchadores de eventos que se ejecutaran cuando se cumplan ciertas condiciones en un nodo de la base de datos o en sus hijos y devolverán una *snapshot* o instantánea de esta (*DataSnapshot*):

Figura 3.7: Estructura de Firebase

- **ChildAdded:** Se dispara el evento cuando se crean nuevos hijos dentro del nodo especificado.
- **ChildChanged:** Se dispara cuando cambia algún valor o nodo recibe cambios dentro del nodo especificado
- **ChildMoved:** Se dispara cuando en el nodo especificado se mueven los hijos.
- **ChildRemoved:** Se dispara cuando algún hijo es eliminado en el nodo especificado.
- **ValueChanged:** Se dispara cuando el valor del nodo es cambiado.

Los eventos anteriores devuelven una clase ValueChangedEventArgs [64] (en el caso de ValueChanged) y ChilChangedEventArgs[65] (en el caso de los demás). Dentro de esta clase se encuentra otra importante llamada DataSnapshot [66]. Se trata de una copia inmutable de la localización de la base de datos donde se ejecuto el evento. Ofrece los siguientes métodos:

- **Child(string path):** Proporciona otra DataSnapshot con la información de uno de los hijos del nodo que ejecuta la función.
- **GetValue():** Devuelve el valor del nodo como un tipado nativo. Estos tipados son bool, string, double, IdIctionarystring,object o Listobject
- **GetRawJsonValue():** Devuelve el valor de string del json introducido en este nodo.

Capítulo 4 Game Design Document

EL Game Design Document o GDD es un documento vivo que forma parte del ciclo de desarrollo de los videojuegos. Se trata de un documento colaborativo en el que diferentes equipos de desarrollo, ya sean programadores, diseñadores, artistas o equipos de marketing, realizan una descripción de todos los aspectos del videojuego y sirve como guía para su desarrollo. Este documento esta sujeto a constantes actualizaciones de cada una de sus secciones a lo largo del ciclo de vida del proyecto.

Aunque es un documento muy usado en la industria, debido a la alta variedad de tipos de videojuegos, no existe una estructura fija. Para este GDD se ha usado la estructura propuesta por Tracy Fullerton [67] adaptándose a los contenidos de este juego.

4.1 Introducción

El objetivo de este proyecto es la creación de un videojuego multijugador competitivo en 2D de acción que sea divertido y entretenido para un publico amplio, fácil de aprender a jugar y que jugadores con diferentes niveles de destreza puedan ganar. El juego podrá ser jugado por usuarios en cualquier parte desde el móvil.

4.1.1 Resumen del Juego

"Spaces - EX" es un juego multijugador local de 2 a 4 jugadores en el que cada jugador tendrá que conseguir la mayor puntuación durante un intervalo de tiempo para conseguir la victoria. Se trata de un juego de Acción multijugador con cámara cenital sobre el centro del mapa. Los jugadores se podrán mover en cualquier dirección del plano 2D.

El jugador controlará a una nave espacial y en cada partida los jugadores tendrán que llegar primeros a las gemas que aparecerán aleatoriamente por todo el campo de juego antes de que se acabe el tiempo. Cuanto más gemas consiga el jugador mayor sera su puntuación total y el jugador que tenga la mayor puntuación sera el vencedor. Los jugadores podrán robar la puntuación de los demás contrincantes y sera más fácil perder puntuación cuanta más se tenga.

El juego tendrá una temática espacial centrada en una zona del espacio exterior. Habrá diferentes arenas de juego con diferentes elementos para favorecer u obstaculizar el movimiento de las naves controlados por los jugadores. Estos obstáculos están relacionados con el espacio

exterior, como por ejemplo asteroides o agujeros negros.

El juego tendrá una estética que mezclara "PixelArt", propia de juegos clásicos de consolas como la Nintendo Entertainment System (NES) o la Sega Master System, con gráficos más modernos y acompañada de música con temática espacial. Esta estética, acompañada de controles simples recordaran a aquellos juegos más antiguos, pero en una plataforma más actualizada.

4.2 Audiencia, Plataforma, Marketing

4.2.1 Audiencia

Aunque el juego es de índole competitiva realmente se pretende llamar la atención de aquellos jugadores que busquen socializar con un grupo de amigos, los llamados "socialisers" del esquema de tipos de jugador de *Richard Bartle* [68]. El juego estará más centrado en partidas caóticas divertidas en el que cualquier jugador, sea cual sea su nivel de destreza con los videojuegos, pueda obtener la victoria. No obstante, jugadores un poco más competitivos también podrán ver un atractivo en las mecánicas del juego, aunque no se pretende que sean muy profundas para que la mayoría de jugadores alcancen el limite de habilidad del juego.

Figura 4.1: Tipos de Jugadores según Richard Bartle

La principal audiencia buscada sera un grupo de personas (amigos, compañeros, etc.) que se pongan a jugar unas partidas durante su tiempo libre en compañía. Una de ellas puede crear una partida y el resto se pueden unir a la sala creada.

4.2.2 Plataforma

Este juego estará disponible en un principio para dispositivos móviles con sistema ope-

rativo Android. Al ser un juego con gráficos simples no usará más de 1 GB de memoria RAM y no se requerirá una GPU muy avanzada.

4.2.3 Marketing

El propósito de la creación de este videojuego es principalmente con fines de aprendizaje, por lo que inicialmente este producto no será vendido en ninguna plataforma, aunque estará disponible el juego en Google Drive y en la pagina web Itch.io.

4.3 Gameplay

4.3.1 Descripción

Los jugadores aparecerán en cada partida cerca de cada una de las esquinas del mapa y se podrán mover en cualquier dirección. Todos los personajes se moverán a la misma velocidad y, además, los jugadores dispondrán de una serie de habilidades, ya sea básicas u obtenibles mediante cajas de objetos que aparecerán aleatoriamente por el mapa. Estas últimas se aplican al jugador directamente tras obtenerlas y son:

- **Disparo:** Ataque básico que todos los jugadores dispondrán desde el principio y sera la principal manera de sabotear a los contrincantes. Podrás disparar al frente un proyectil cada cierto periodo de tiempo corto. El proyectil se moverá a una velocidad mayor que la del personaje. El proyectil desaparece al impactar en un jugador y hace perder la mitad de la puntuación que tenia hasta ahora

- **Ataque carga:** Aparecerá en una caja de objetos. Se trata de una carga a toda velocidad en la dirección en la que el jugador se estaba moviendo. Cuando se pulse el botón del objeto el jugador se moverá muy rápido hacia donde había tocado por ultima vez, haciendo perder la mitad de la puntuación a los jugadores con los que colisione.

- **Disparo explosivo**: Aparecerá en una caja de objetos y otorgara 5 disparos explosivos. Se trata de un ataque de corto alcance, bastante lento pero potente, que dejará a tus contrincantes un tiempo sin poder moverse a parte de hacerles perder la mitad de su puntuación.

- **Supervelocidad:** Aparecerá en caja de objetos. Subirá la velocidad base del jugador durante un corto periodo de tiempo para poder esquivar mejore los ataques y coger gemas más rápido.

- **Invulnerabilidad:** Aparecerá en caja de objetos. Durante un periodo corto de tiempo seras inmune a los ataques de los contrincantes.

4.3.2 Control

El juego esta pensado para ser jugado en una dispositivo móvil y por lo tanto el esquema de control tiene en cuenta las limitaciones de esta plataforma. La pantalla táctil sera la única forma a través de la cual se podrá controlar este juego.

Habilidad	**Control**
Moverse	Tocar la pantalla táctil
Disparo	Tocar dos veces seguidas la pantalla táctil
Objetos	Tocar el icono del objeto

Tabla 4.1: Tabla de controles

4.3.3 Normas y Comportamiento

El juego consiste en una sesión en la que los jugadores se conectaran a una sala y jugarán partidas de 2 minutos. Al unirse a la sala los jugadores podrán votar en que arena jugar y el juego elegirá aleatoriamente teniendo en cuenta esos votos, de manera que si hay tres jugadores que eligen un mapa y otro usuario elige otro mapa diferente. haya un 75 % de probabilidad de que se juegue en el primer mapa y un 25 % de probabilidad para el otro. Entre cada partida los jugadores tendrán la posibilidad de seguir jugando o salir de la sesión, momento en el cual sera reemplazado por un jugador controlado por la IA.

4.3.3.1 Jugador

Cada jugador se moverá en la dirección de la pantalla donde el usuario este tocando, acelerando instantáneamente a la velocidad máxima posible. Las naves empezaran con 0 puntos y con la posibilidad de realizar disparos básicos.

Cuando un jugador es atacado, su puntuación se verá reducida siempre a la mitad, redondeando hacia lo bajo en el caso que queden decimales (de 5 puntos pasara a 2). Estará "aturdido" durante unos pocos frames donde no se podrá mover y gozara de invulnerabilidad durante unos frames más tras ese periodo, indicado por el parpadeo del personaje. En el momento de ser atacado saldrán volando la mitad de las gemas conseguidas hasta ese momento por ese jugador, aterrizando en partes de la arena para poder ser atrapados por los contrincantes.

4.3.3.2 Gemas

Los gemas son la fuente de puntuación de la partida. Aparecerán pseudoaleatoriamente en el mapa evitando los lugares donde están posicionados los jugadores para evitar ocasiones en las que un jugador este moviéndose sin objetivo y se cree una gema justo en la posición donde pasaba, dando una ventaja y haciendo que el juego dependa más de la suerte. Las gemas estarán estáticas en el punto donde aparecen y serán destruidas una vez un jugador se acerque para obtenerlas.

Los gemas aparecerán con mayor frecuencia al principio de la partida y cada vez aparecerán menos a lo largo de esta. En los últimos segundos de la partida no aparecerán más, dejando solo la posibilidad de ataque a contrincantes para conseguir puntuación. Existirá un limite de gemas no recogidas por los jugadores en pantalla, para evitar la acumulación infinita de estos objetos, pudiendo provocar un aumento no deseado del uso de la memoria.

4.3.3.3 Cajas de Objetos

Las cajas de objetos también aparecerán pseudoaletoriamente tratando de evitar aparecer justo encima de los jugadores. Aparecerá una cada 20 segundos y estará disponible hasta que un jugador la coja o pasen 10 segundos. Esta cajas concederán un *powerup* temporal al jugador que llegue antes a su lugar de aparición. Este *powerup* es aplicado en el momento de obtener la caja y el jugador la almacenará para poder elegir cuando usa el objeto de esa caja. Si el jugador ya tiene una caja almacenada y consigue otra, se sobreescribe la primera

4.3.3.4 Arenas

Las arenas son el lugar donde los jugadores competirán en cada partida. Los jugadores, al unirse a la sala, podrán votar que arena prefieren jugar en la siguiente partida, y sera elegidas teniendo en cuenta estos votos. Cada arena tendrá diferentes obstáculos que darán variedad al juego. Los mapas se describen a continuación:

- **Exosfera:** Arena estándar. No tiene ningún obstáculo, ideal para jugadores novatos.
- **Cinturón de asteroides:** En el aparecen asteroides, obstáculos que se mueven y dañan al jugador al contacto haciéndoles perder la mitad de la puntuación y esparciendo las gemas de ese jugador por la pantalla.
- **Nebulosa:** Mapa con polvo estelar en algunas zonas, que te reducirán la velocidad si entras en ellas.
- **Agujero negro:** Zona de poca visibilidad. Solo se verá a unos pocos centímetros de distancia de los jugadores y las gemas emitirán luminiscencia cada cierto tiempo, permitiendo conocer su localización.
- **Centro de la galaxia:** En él aparecerán pequeños agujeros negros que absorberán a algunos jugadores cercanos y les dejara inmovilizados por un tiempo, siendo expuesto a los ataques de los jugadores.

- **Agujeros de gusano:** Estos agujeros aparecerán de vez en cuando por el mapa permitiéndonos teletransportarnos a otros lugares. Aparecerán siempre por pares.

4.3.3.5 Menú principal

El menú principal se compondrá de una interfaz con botones y campos de introducción de Strings, correspondientes al apodo del jugador y a un código de sala que puede introducir el usuario para unirse a una sala ya existente. El código de una sala se compone de 4 caracteres alfanuméricos en mayúscula. A la hora de crear una sala se generara un código aleatorio con esas características.

Si se quiere crear o unirse a una partida, se debe introducir antes un apodo en uno de los campos proporcionados para ello. Para unirse a una partida se debe introducir un código de lobby o sala existente y correcto.

4.3.3.6 Lobby

Justo al entrar en la sala se presentara al usuario opciones de naves que podrá elegir para la partida. Si un jugador que ha entrado anteriormente elige una nave, ningún otro jugador podrá seleccionarla a no ser que abandone la sala. También se le presentara una selección de mapas de las cuales elegirá una. La selección de la arena de la partida sera de forma aleatoria de entre las elegidas por los jugadores.

En la sala el usuario podrá moverse con tranquilidad como si estuviera dentro de una partida, pero sin la posibilidad de disparar ni dañar a los contrincantes, que también se estarán moviendo por el mapa. El usuario que ha creado la sala o, en su defecto por abandono por parte de este de la sal la persona que se ha unido antes, sera el único que podrá iniciar la partida o cambiar los ajustes.

4.3.3.7 Inteligencia Artificial

La nave controlada por la IA se moverá siempre en dirección al objeto más cercano a ella, ya sea una gema o una caja de objetos. Durante su recorrido disparara en la dirección en la que se este moviendo cada 1-2 segundos. Los objetos que obtenga de las cajas serán usadas inmediatamente.

4.3.4 Condiciones de Victoria y Derrota

Al cabo de dos minutos la partida finalizara y el jugador con más puntos sera declarado el vencedor de la partida. Una pantalla final anunciara la posición final de todos los jugadores. Si los jugadores con mayor puntuación empatan en puntos, ambos serán declarados ganadores. Tras conseguir el numero de victorias dictado por las reglas del lobby se declara ganador al jugador y se vuelve a la sala de espera del lobby.

La única condición de victoria y derrota ocurre cuando la partida finaliza. Aunque el jugador no tenga ninguna gema o las pierda todas podrá continuar jugando y conseguir más puntos ya sea recogiendo gemas o atacando a los contrincantes.

4.4 Interfaz

El juego tendrá diferentes pantallas correspondientes al menú principal, al lobby y a la partida.

4.4.1 Menú Principal

Es la primera pantalla que aparece tras ejecutar el juego. En el aparece el titulo del juego, botones para crear o unirse a un Lobby, escribir un código de lobby y ponerse un apodo. Estos botones son accionados mediante toque del dedo (ver figura 4.2).

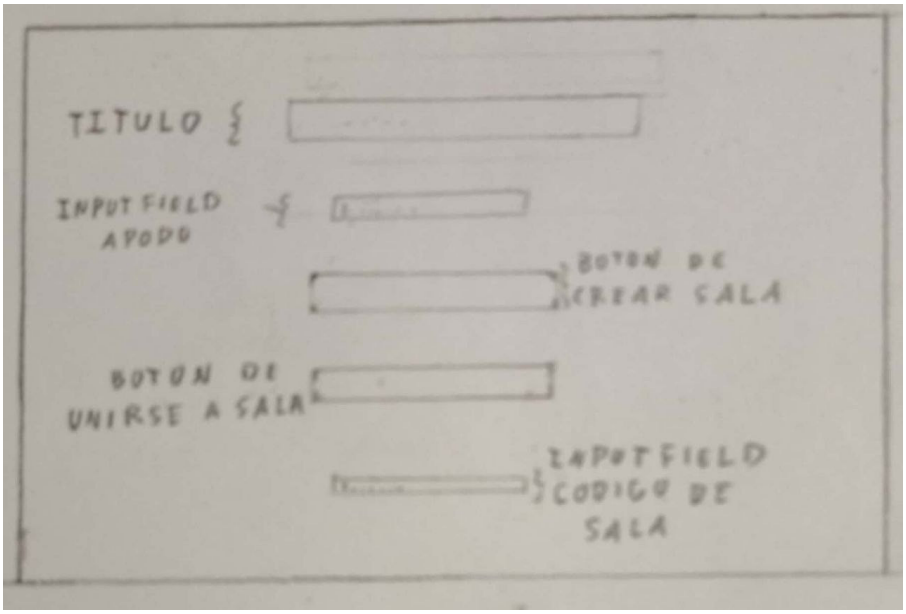

Figura 4.2: Interfaz del menú principal

Para ponerse un apodo o escribir el código de un lobby simplemente se puede tocar el campo de escritura con el dedo. Haciendo esto aparecerá el teclado del dispositivo. En ambos campos solo se podrán poner caracteres alfanuméricos, ignorando cualquier carácter otro que se escriba. En el campo del código de la sala, además se pondrá cada carácter en mayúsculas automáticamente ya que todos los caracteres del código son en mayúsculas.

4.4.2 Salas

Lo primero que se presenta al jugador cuando entra a un lobby es la selección de la nave. Esto estará conformado por un texto describiendo la acción que debe hacer el jugador y 4 botones cuyos iconos son las diferentes naves. Si algún jugador que ha entrado antes al lobby ha seleccionado una nave, esa opción aparecerá oculta para los jugadores que estén en esa pantalla.

Si se es el creador de la partida o el jugador más antiguo en su defecto, se aparecerá un botón de ajustes. Pulsando este botón se podrá cambiar algunos ajustes de la partida como:

- **Victorias:** Mediante un slider, te permite seleccionar del 1 al 10, el numero de victorias necesarias para elegir el ganador del lobby.

- **Jugadores IA:** Mediante un checkbox permite seleccionar si jugadores controlados por la IA rellenan el lobby.

- **Mapas:** Mediante un dropbox, permite seleccionar si los mapas lo elige el jugador con más tiempo en la sala, son aleatorios o por votación de los jugadores, la opción por defecto.

- **Cajas de objetos:** Con un slider se permite regular el numero de cajas de objetos que aparecen en la partida.

También, el creador o jugador más longevo sera el único que tendrá la opción de empezar la partida, mediante un botón. Todos los jugadores de la partida podrán ver los apodos y naves de los demás jugadores y las reglas de la partida actual.

4.4.3 Partida

La interfaz gráfica dentro de cada partida tiene que tener en cuenta la información de todos los jugadores. Como los jugadores empezaran en cada una de las esquinas de la pantalla, en la esquina correspondiente aparecerá la información de ese jugador. El puntaje total de cada uno tiene que ser lo más destacado, de manera que todos los jugadores puedan ver quien va ganando y por cuanto. Acompañado del puntaje se indicara la posición del jugador(1^o, 2^o, 3^o o 4^o) y si tiene un objeto de una caja (ver figura 4.3).

Para dar mayor visión y facilidad a los jugadores, en cada una de sus pantallas la interfaz correspondiente a su puntuación sera más grande, destacando sobre el resto. También en la interfaz el color de la puntuación estará coloreado por el color de la nave correspondiente, para dar mayor claridad.

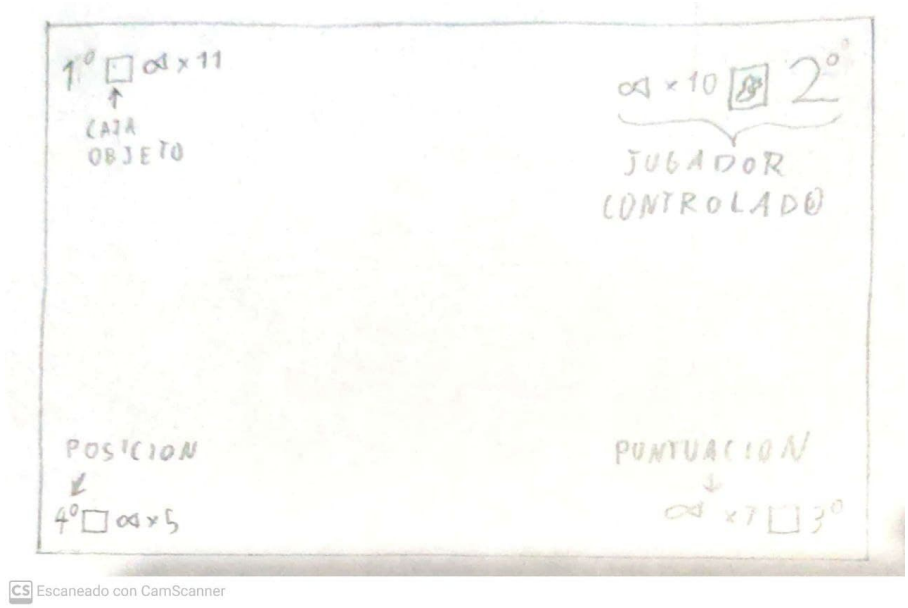

Figura 4.3: Interfaz de la Partida

4.5 Multimedia

Los *Sprites* usados para las naves, entornos e IU serán archivos en formato de imágenes como PNG o JPEG, al igual que la mayoría de las animaciones, formadas por la secuencia de estas imágenes.

4.5.1 Personajes

Cada jugador usara una nave con diseño Pixel Art de entre 4 posibles. La nave tendrá un pequeño fuego saliendo de los motores también hecho al estilo Pixel-Art.

Las naves han sido obtenidas del Pack de assets del usuario Gustavo Vituri en la pagina de Itch.io (ver figura 4.4). Ninguna licencia ha sido requerida para la utilización o modificación de estos Assets. (CC0)

4.5.2 Gemas

Las gemas serán un conjunto de piedras preciosas de diferentes colores según la arena donde se juegue la partida, para evitar confusión con paletas de colores similares al fondo de pantalla.

Estas gemas han sido diseñadas por Giacomo Magalhães - GM ilustra, usuario de la pagina web Itch.io (ver figura 4.5). Este pack de sprites no necesitaba ninguna licencia para ser usado en el juego (CC0).

Figura 4.4: Naves de los jugadores

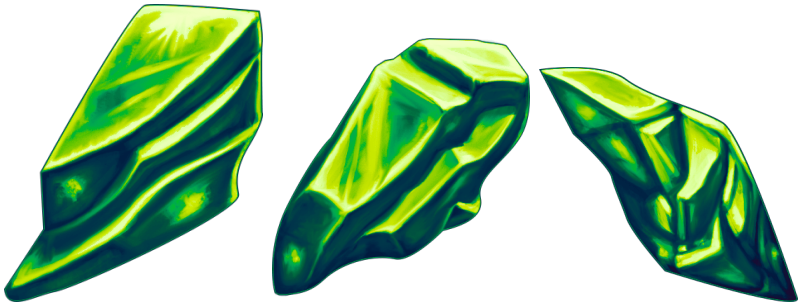

Figura 4.5: Gemas de la partida

4.5.3 Entorno

Todos los mapas tendrán un temática del espacio exterior ya que los jugadores controlan a una nave espacial. Sin embargo, cada uno de los mapas, a excepción del más básico, la exosfera, tendrá una diferente temática a mayores, descrita a continuación:

- **Cinturón de asteroides:** El fondo tendrá la imagen de un cinturón de asteroides, ya que el tema de esta arena es el de los obstáculos de los asteroides.
- **Nebulosa:** En este mapa se encuentran zonas con acumulación de polvo estelar.
- **Agujero Negro:** La oscuridad sera el tema principal de este escenario, puesto que se desarrolla en una zona donde la fuerza gravitatoria es tan grande que la luz no puede escapar.
- **Centro de la galaxia:** El escenario de esta arena sera rico en estrellas cúmulos y planetas al ser una zona plagado de estos cuerpos celestiales. No influirán en el Gameplay sino que estarán en el Background.

- **Agujeros de gusano:** Su temática sera similar al mapa más básico, sin embargo agujeros de gusano aparecerán en el.

Algunos de los *Sprites* de los fondos de las arenas han sido creados por Screaming Brain Studios, desde la siguiente pagina web: `https://screamingbrainstudios.itch.io/seamless-space-backgrounds`, como se muestra en la figura 4.6 . Ninguna licencia ha sido necesitada para el uso de estos assets.

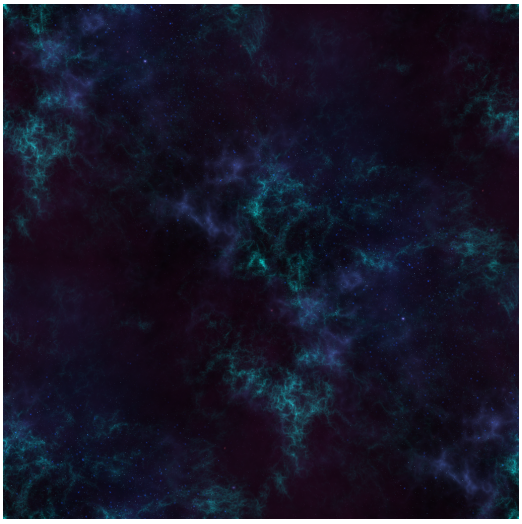

Figura 4.6: Ejemplo de un fondo usado

4.5.4 Fuentes de Texto

La fuente de texto utilizado tiene una estética PixelArt espacial. Se trata de dos archivos .ttf, uno de tamaño 5 y otro de tamaño 8, que contienen las definiciones de la tipografía de tal esta fuente. Al ser PixelArt, los tamaños de la letra a la hora de usarlo deben ser múltiplos de su tamaño original(5,10,15 para el de 5 y 8,16,24 para el de 8).

Estos archivos han sido creados por el usuario John Watson de la pagina Itch.io (ver figura 4.7). No se ha necesitado ninguna licencia para la utilización o modificación de estos Assets. (CC0)

4.5.5 Animación

Se usará primordialmente la animación mediante secuencia de *Sprites*. Las naves espaciales tendrán animación mientras se mueven y si disparan. Algunos elementos de los escenarios también tienen animaciones (asteroides, agujeros negros, agujeros de gusano, etc).

Figura 4.7: Fuente de texto

4.5.6 Música

El menú principal y todas las pantallas no relacionadas con la partida tendrá una misma melodía. En las partidas, dependiendo del mapa donde se este jugando sonara una música diferente relacionados con el entorno donde se compite.

Los movimientos y ataques de las naves serán acompañados con sus respectivos efectos sonoros así como los elementos del entorno, que sonaran a un volumen menor para minimizar su importancia, excepto los elementos que obstaculizan, que necesitaran llamar la atención.

4.6 Especificaciones Técnicas

El juego estará disponible para dispositivos móviles con sistema operativo Android. Se jugara principalmente usando la pantalla táctil.

El entorno multijugador se desplegara usando un servidor externo, Firebase Realtime Database. Los dispositivos de los jugadores podrán conectarse desde cualquier parte a Internet y al servidor y obtener datos actualizados de los usuarios de la misma partida que analizaran, procesaran y mostraran un resultado por pantalla.

El servidor se trata de una base de datos NoSQL donde se guardaran elementos JSON. Los dispositivos enviaran a la base de datos un archivo JSON con los datos de las acciones realizadas por sus usuarios y serán notificados por la base de datos cuando haya cambios en ella, mediante una escucha activa. Estos cambios serán obtenidos por todos los dispositivos para ser procesados y actualizar la información en pantalla (posición de los demás jugadores, puntuación, etc).

El entorno multijugador se compondrá de salas, cada una de ellas con un código de sala. Para que un usuario entre en una sala concreta debe conocer el código de esa sala, que introdu-

cirá en el campo correspondiente. La aplicación buscara en la base de datos esa sala y añadirá al usuario en ella a la vez que obtiene los datos de los demás usuarios.

Capítulo 5 Desarrollo del proyecto

5.1 Análisis del proyecto

5.1.1 Requisitos

5.1.1.1 Requisitos Funcionales

- **RF -01:** La aplicación deberá poder ser instalada en un dispositivo móvil Android
- **RF -02:** El sistema sera capaz de conectarse al servidor Firebase.
- **RF -03:** El sistema deberá ser capaz de enviar y recibir datos al servidor Firebase
- **RF -04:** El sistema deberá permitir a los usuarios crear o unirse a una sala.
- **RF -05:** El sistema deberá mostrar las acciones de los demás usuarios de la sala por pantalla.
- **RF -06:** El sistema deberá sincronizar las partidas de los usuarios.
- **RF -07:** El sistema determinara un ganador al final de cada partida.

5.1.1.2 Requisitos no Funcionales

- **RNF -01:** El sistema debe ser fácil de usar y comprender por un usuario nuevo
- **RNF -02:** El sistema deberá estar disponible a al menos 100 usuarios simultáneos
- **RNF -03:** El sistema deberá validar las entradas de texto del usuario asegurándose de que solo haya caracteres alfanuméricos.
- **RNF -04:** El sistema no deberá permitir al usuario introducir por su cuenta datos en el servidor Firebase.
- **RNF -05:** El sistema cancelara las transacciones con la base de datos en caso de error.

5.1.2 Casos de Uso

En este proyecto se han identificado dos actores. Usuario, que es el usuario del sistema, y Servidor Firebase, un actor auxiliar que representa a la base de datos. Se han identificado los casos de uso mostrados en la figura 5.1:

Los casos de uso se describen en las tablas 5.1 a 5.11.

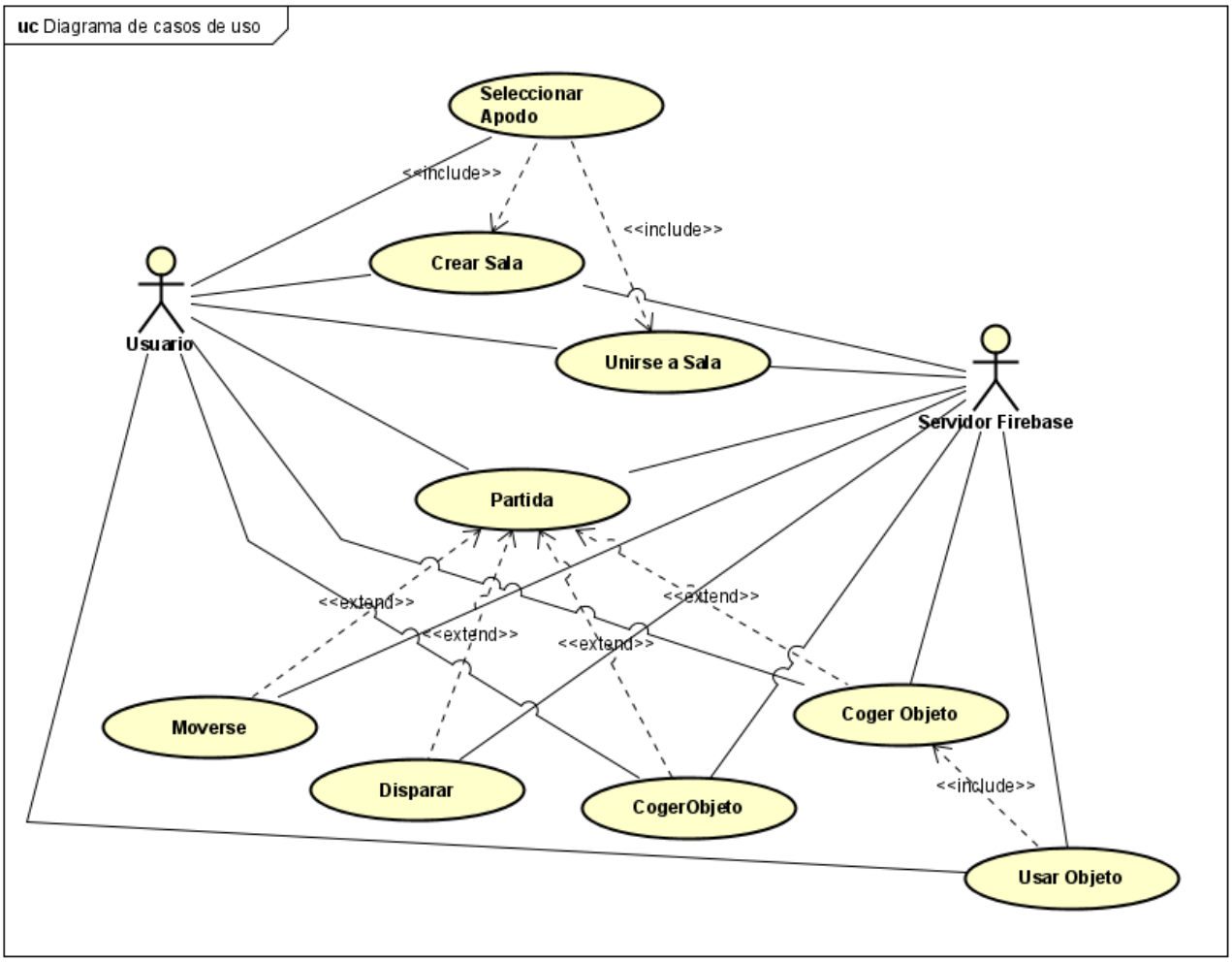

Figura 5.1: Diagramas de Casos de Uso

Caso de Uso	Introducir Apodo
Identificador	CU-01
Actores	Usuario
Pre-condiciones	Ninguna
Post-condiciones	Ninguna
Descripción	El usuario introduce un apodo en el campo de texto correspondiente al apodo
Secuencia	**1-** El usuario selecciona el campo de texto de introducir apodo
	2- *El sistema saca por pantalla el teclado del teléfono*
	3- El usuario introduce su apodo
	4- *El sistema comprueba que los caracteres introducidos son alfanuméricos y lo muestra en el campo de texto*
Secuencia Alternativa	**4.a-** *El sistema encuentra caracteres no alfanuméricos y los ignora*

Tabla 5.1: Caso de Uso CU-01

Caso de Uso	Crear Sala
Identificador	CU-02
Actores	Usuario, Servidor Firebase
Pre-condiciones	El usuario a introducido un apodo
Post-condiciones	Ninguna
Descripción	El usuario crea una sala donde podrán unirse otros usuarios para jugar una partida.
Secuencia	**1-** El usuario selecciona Crear partida
	2- *El sistema comprueba que el usuario haya introducido un apodo*
	3- *El sistema genera un código de sala*
	4- *El sistema envía un mensaje a Servidor Firebase para crear la sala*
	5- Servidor Firebase crea la sala y notifica al sistema
	6- *El sistema muestra al jugador una selección de naves a elegir*
	7- El usuario elige una nave
	8- *El sistema muestra al jugador una selección de mapas a elegir*
	9- El usuario elige un mapa
	10- *El sistema envía la información a Servidor Firebase del apodo, nave y mapa elegido por usuario*
	11- Servidor Firebase escribe la información y notifica al sistema
	12- *El sistema muestra al usuario el apodo, código del lobby, y nave*
	13- *El sistema se pone a escuchar a Servidor Firebase esperando otros usuarios*
Secuencia Alternativa	**2.a-** *El sistema no recibe apodo y notifica al usuario de que se lo debe poner*
	5.a- *El sistema recibe un error de Servidor Firebase y aborta la operación*
	11.a- *El sistema recibe un error de Servidor Firebase y aborta la operación*
	13.a.1- Servidor Firebase notifica al sistema de la entrada de un jugador **13.a.2-** *Sistema recupera la información de ese jugador.* **13.a.3-** *El sistema muestra el apodo y nave de ese jugador al usuario.*

Tabla 5.2: Caso de Uso CU-02

Caso de Uso	Unirse a sala
Identificador	CU-03
Actores	Usuario, Servidor Firebase
Pre-condiciones	El usuario a introducido un apodo
Post-condiciones	Ninguna
Descripción	El usuario se una a una sala creada por un jugador usando el código de sala
Secuencia	**1-** El usuario introduce el código de sala
	2- *El sistema que no haya caracteres alfanuméricos y pone el código en mayúsculas*
	3- El usuario pulsa el botón de unirse a sala
	4- *El sistema comprueba que el usuario haya introducido un apodo*
	5- *El sistema comprueba que se haya introducido un código de sala*
	6- *El sistema envía un mensaje a Servidor Firebase para comprobar que el código de sala existe y saber el numero de jugadores que hay en el*
	7- Servidor Firebase notifica al sistema que la sala existe y hay menos de 4 jugadores
	8- *El sistema envía un mensaje a Servidor Firebase para obtener la información de los usuarios*
	9- Servidor Firebase informa al sistema sobre los usuarios de la sala
	10- *El sistema muestra al jugador una selección de naves que puede elegir*
	11- El usuario elige una nave
	12- *El sistema muestra al jugador una selección de mapas a elegir*
	13- El usuario elige un mapa
	14- *El sistema envía la información a Servidor Firebase del apodo, nave y mapa escogidos por el usuario*
	15- Servidor Firebase escribe la información y notifica al sistema
	16- *El sistema muestra al usuario el apodo, código del lobby, y nave del usuario*
	17- *El sistema muestra al usuario el apodo, código del lobby, y nave de los demas usuarios*
	18- *El sistema se pone a escuchar a Servidor Firebase esperando otros usuarios*

Tabla 5.3: Caso de Uso CU-03 1^o Parte

Secuencia Alternativa	**2.a-** *El sistema encuentra caracteres no alfanuméricos y los ignora*
	4.a- *El sistema no recibe apodo y notifica al usuario de que se lo debe poner*
	4.a- *El sistema no encuentra el código de la sala y notifica al usuario que debe introducirlo*
	7.a.1- Servidor Firebase notifica al sistema que la sala no existe **7.a.2-** *El sistema notifica al usuario que la sala no existe*
	7.b.1- Servidor Firebase notifica al sistema que la sala tiene 4 jugadores **7.b.2-** *El sistema notifica al usuario que la sala esta llena*
	9.a- *El sistema recibe un error de Servidor Firebase y aborta la operación*
	15.a- *El sistema recibe un error de Servidor Firebase y aborta la operación*
	18.a.1- Servidor Firebase notifica al sistema de la entrada de un jugador **18.a.2-** *Sistema recupera la información de ese jugador.* **18.a.3-** *El sistema muestra el apodo y nave de ese jugador al usuario.*

Tabla 5.4: Caso de Uso CU-03 2^o Parte

Caso de Uso	Partida
Identificador	CU-04
Actores	Usuario, Servidor Firebase
Pre-condiciones	Usuario es el jugador más antiguo de la sala
Post-condiciones	La partida se iniciara para todos los jugadores de la sala
Descripción	El usuario inicia la partida en la sala y esta se desarrolla hasta el final del tiempo
Secuencia	**1-** El usuario pulsa el botón de iniciar partida
	2- *El sistema comprueba que configuración de elección de arenas hay seleccionado, la opción por defecto*
	3- *El sistema recupera de Servidor Firebase los mapas seleccionados por los usuarios*
	4- *El sistema elige un mapa de los seleccionados al azar, envía el mapa a Servidor Firebase y le indica que la partida va a comenzar*
	5- Servidor Firebase avisa a todos los jugadores de la sala que la partida empieza y en que mapa
	6- *El sistema recibe la notificación de que la partida inicia y en que mapa*
	7- *El sistema coloca las naves de los jugadores en la posición inicial, en el mapa correspondiente y prepara la interfaz de la partida*
	8- *El sistema muestra una cuenta atrás y espera a una notificación de Servidor Firebase para empezar la partida*
	9- *El sistema notifica a Servidor Firebase que la cuenta atrás ha terminado y empieza la partida*
	10- Servidor Firebase notifica a los usuarios que la cuenta atrás ha terminado y empieza la partida
	11- *El sistema recibe la notificación de que la cuenta atrás ha terminado y empieza la partida*
	12- *El sistema escucha las acciones de los jugadores y cambios de objetos y gemas en la base de datos de Servidor Firebase*
	13- *El sistema crea otra cuenta atrás para la finalizacion de la partida*
	14- *El sistema crea objetos y gemas aleatoriamente durante la partida y los envia a Servidor Firebase*
	15- Servidor Firebase confirma los cambios y notifica a los jugadores de la aparición de estos objetos
	16- *El sistema obtiene los objetos y gemas del servidor y los muestra en pantalla*
	17- *El sistema notifica de la finalizacion de la partida al Servidor Firebase*
	18- Servidor Firebase notifica a todos los usuarios de la finalizacion de la partida
	19- *El sistema notifica al usuario del ganador de la partida y vuelve al estado anterior al inicio de la partida*

Tabla 5.5: Caso de Uso CU-04 1^o Parte

Secuencia Alternativa	**2.a.1-** *El sistema comprueba que en la configuración de selección de arenas es al azar.* **2.a.2-** *El sistema elige un mapa de los seleccionados al azar, envía el mapa a Servidor Firebase y le indica que la partida va a comenzar* GO TO Paso 4
	2.b.1- *El sistema comprueba que en la configuración de selección de arenas es según la elección del jugador más longevo o creador de la partida.* **2.b.2-** *El sistema envía el mapa a Servidor Firebase y le indica que la partida va a comenzar* GO TO Paso 4
	5.a- *El sistema recibe un error de Servidor Firebase y aborta la operación*
	10.a- *El sistema recibe un error de Servidor Firebase y aborta la operación*
	12.a.1- El Servidor Firebase notifica al sistema de las acciones de los jugadores **12.a.2-** *El sistema obtiene los cambios y procesa las acciones de los jugadores para mostrarlas en pantalla* **15.a-** *El sistema recibe un error de Servidor Firebase y aborta la operación* GO TO Paso 14 **18.a-** *El sistema recibe un error de Servidor Firebase y aborta la operación* GO TO Paso 17

Tabla 5.6: Caso de Uso CU-04 2^o Parte

Caso de Uso	Moverse
Identificador	CU-05
Actores	Usuario, Servidor Firebase
Pre-condiciones	Se debe haber iniciado una partida
Post-condiciones	Los cambios serán comunicados a todos los usuarios de la partida
Descripción	El usuario mueve su nave por la arena de la partida
Secuencia	**1-** El usuario toca la posición de la pantalla a la que quiere que su nave se mueva
	2- *El sistema comienza a mover la nave al punto deseado*
	3-*El sistema envía a Servidor Firebase las coordenadas de la nave*
	4- Servidor Firebase avisa a los demás usuarios de la acción de esa nave
Secuencia Alternativa	**4.a-** *El sistema recibe un error de Servidor Firebase e ignora el movimiento*

Tabla 5.7: Caso de Uso CU-05

Caso de Uso	Disparar
Identificador	CU-06
Actores	Usuario, Servidor Firebase
Pre-condiciones	Se debe haber iniciado una partida
Post-condiciones	Los cambios serán comunicados a todos los usuarios de la partida
Descripción	El usuario dispara un láser
Secuencia	**1-** El usuario dispara desde la nave
	2- *El sistema genera un disparo*
	3-*El sistema envía a Servidor Firebase las coordenadas y orientación de ese disparo*
	4- Servidor Firebase avisa a los demás usuarios de la acción de esa nave
Secuencia Alternativa	**4.a-** *El sistema recibe un error de Servidor Firebase e ignora el disparo*

Tabla 5.8: Caso de Uso CU-06

Caso de Uso	Coger Gema
Identificador	CU-07
Actores	Usuario, Servidor Firebase
Pre-condiciones	Se debe haber iniciado una partida
Post-condiciones	Los cambios serán comunicados a todos los usuarios de la partida
Descripción	El usuario coge una gema con su nave y aumenta su puntuación
Secuencia	**1-** El usuario coge una gema con su nave
	2- *El sistema aumenta la puntuación del jugador y borra la gema*
	3-*El sistema envía a Servidor Firebase que la gema se borre y actualize la puntuacion del jugador*
	4- Servidor Firebase avisa a los demás usuarios de la acción de esa nave
Secuencia Alternativa	**3.a-** *El sistema recibe un error de Servidor Firebase y lo intenta de nuevo*

Tabla 5.9: Caso de Uso CU-07

Caso de Uso	Coger Objeto
Identificador	CU-08
Actores	Usuario, Servidor Firebase
Pre-condiciones	Se debe haber iniciado una partida
Post-condiciones	Los cambios serán comunicados a todos los usuarios de la partida
Descripción	El usuario coge un objeto con su nave y lo guarda para su posterior uso
Secuencia	**1-** El usuario coge un objeto con su nave
	2- *El sistema guarda el objeto para el uso de su jugador y lo borra de la arena*
	3-*El sistema envía a Servidor Firebase que el objeto se borre y actualize la información del jugador*
	4- Servidor Firebase avisa a los demás usuarios de la acción de esa nave
Secuencia Alternativa	**3.a-** *El sistema recibe un error de Servidor Firebase y lo intenta de nuevo*

Tabla 5.10: Caso de Uso CU-08

Caso de Uso	Usar Objeto
Identificador	CU-09
Actores	Usuario, Servidor Firebase
Pre-condiciones	Se debe haber iniciado una partida, Se debe haber cogido un objeto
Post-condiciones	Los cambios serán comunicados a todos los usuarios de la partida
Descripción	El usuario usa el objeto cogido anteriormente y lo gasta
Secuencia	**1-** El usuario usa un objeto
	2- *El sistema elimina el objeto para impedir otra vez su uso*
	3-*El sistema envía a Servidor Firebase que actualice el estado del jugador*
	4- Servidor Firebase avisa a los demás usuarios de la acción de esa nave
Secuencia Alternativa	**3.a-** *El sistema recibe un error de Servidor Firebase y lo intenta de nuevo*

Tabla 5.11: Caso de Uso CU-09

5.1.3 Modelo de Dominio

El la figura 5.2 se presentan el diagrama de clases, donde se han identificado las siguientes clases:

- Usuario: Representa el usuario del sistema.
- Sala: Representa la sala donde diversos usuarios se juntan para jugar una partida.

- Nave: Se trata de los personajes usados por cada usuario en la sala.
- Gemas: Los items de puntuación de la partida.
- Objetos: Los items que servirán de PowerUps.
- Disparo : Proyectil disparado por la nave del usuario que interactuara con las naves de los enemigos.

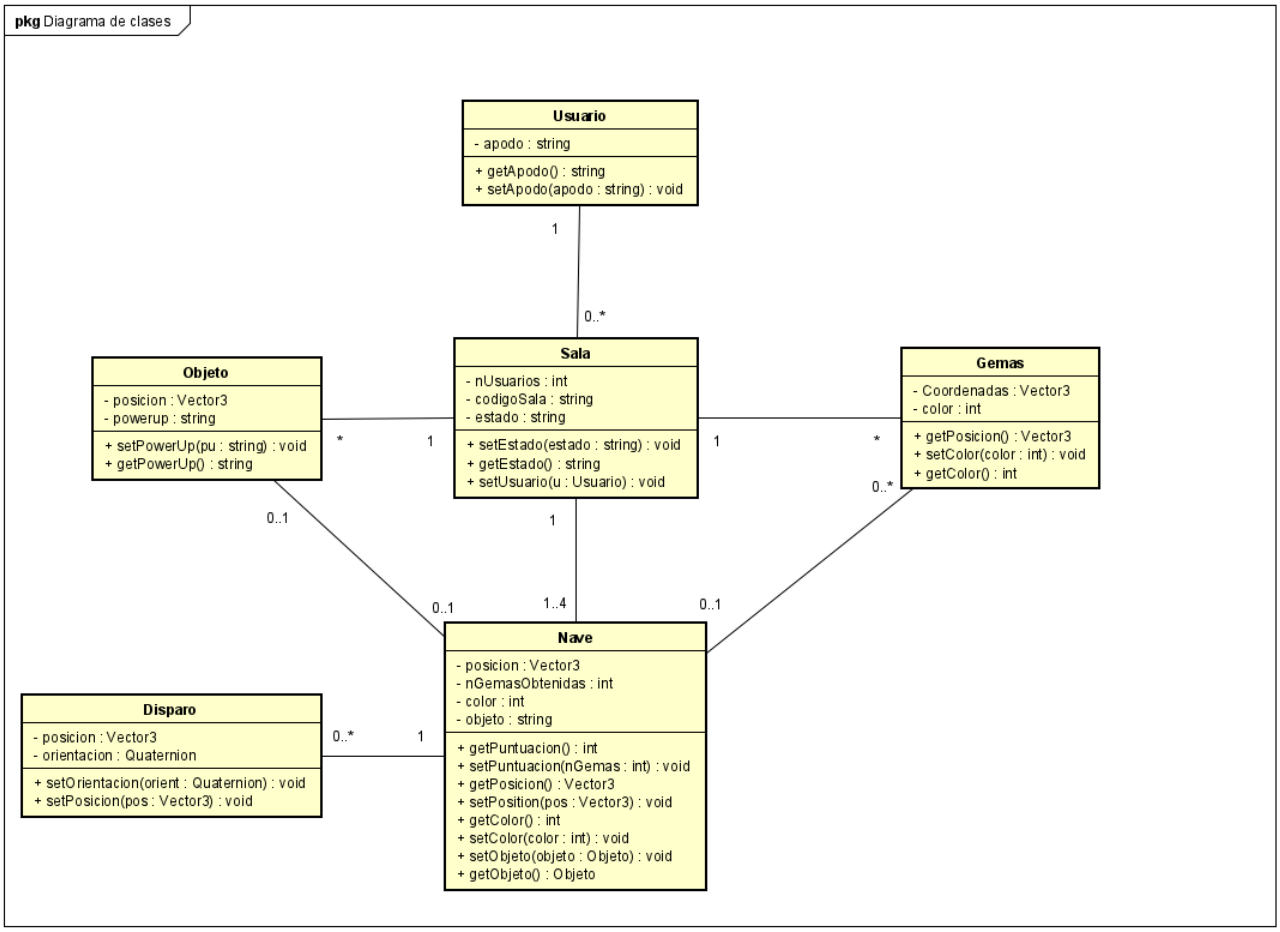

Figura 5.2: Diagrama de Clases

5.2 Diseño

5.2.1 Arquitectura de la aplicación

La arquitectura de esta aplicación utilizará el patrón de diseño Modelo-Vista-Controlador (MVC). La vista estará compuesta por la interfaz de la aplicación, que se encargara de mostrar la información al jugador y recibir sus entradas. El controlador se encargara de obtener esos entradas y ordenar la realización de las acciones correspondientes, es decir, la lógica del sistema. El modelo, se encargara de definir la estructura de los datos y se encargara de la gestión de

la base de datos. La vista será notificada de los cambios realizados por parte del modelo y del controlador para actualizar la información que se muestra al usuario en pantalla.

5.2.2 Diagramas de Secuencia

En esta sección se presentan los diagramas de secuencia correspondientes a cada uno de los casos de uso presentados en la sección de 5.1.2. Estos se muestran en las figuras 5.3 a 5.11.

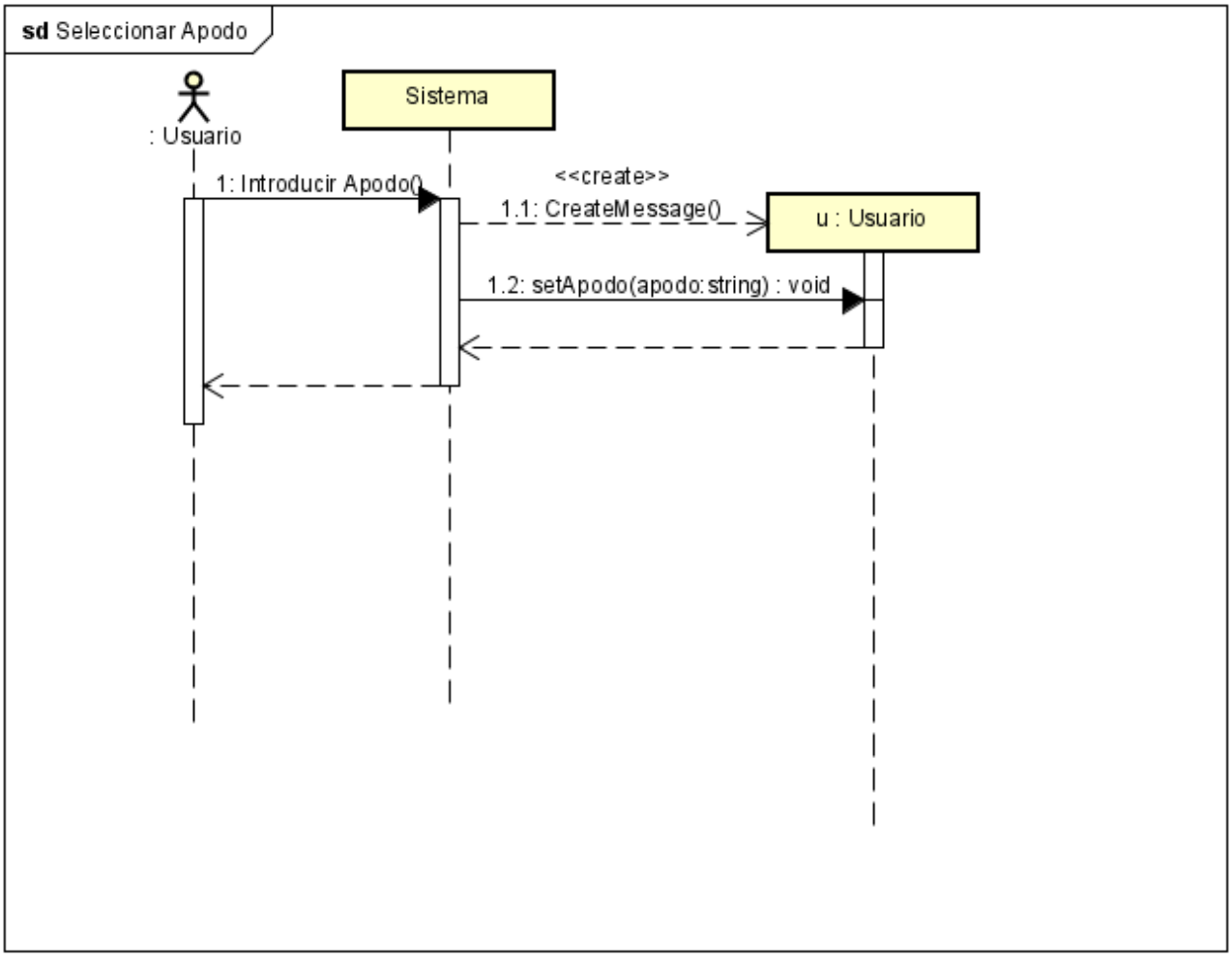

Figura 5.3: Diagrama de Secuencia de Introducir Apodo

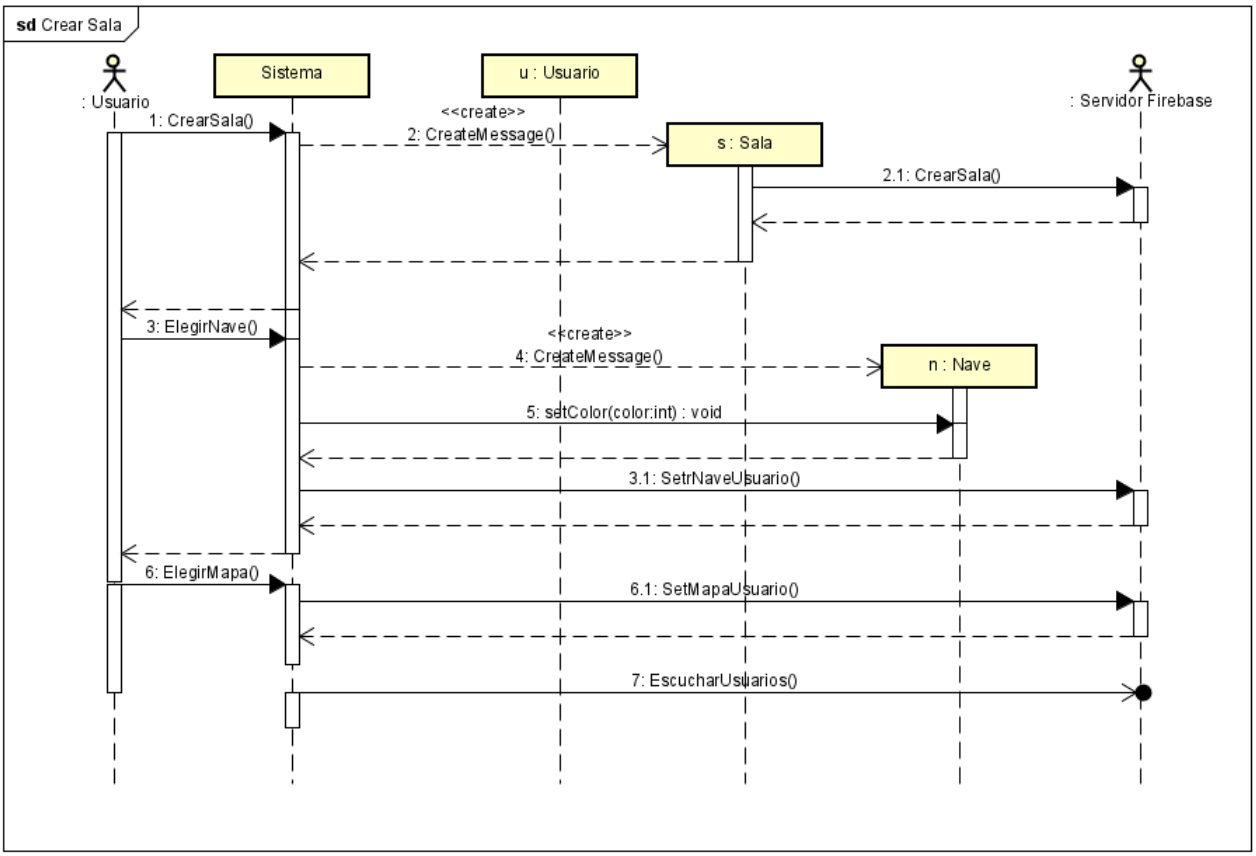

Figura 5.4: Diagrama de Secuencia de Crear Sala

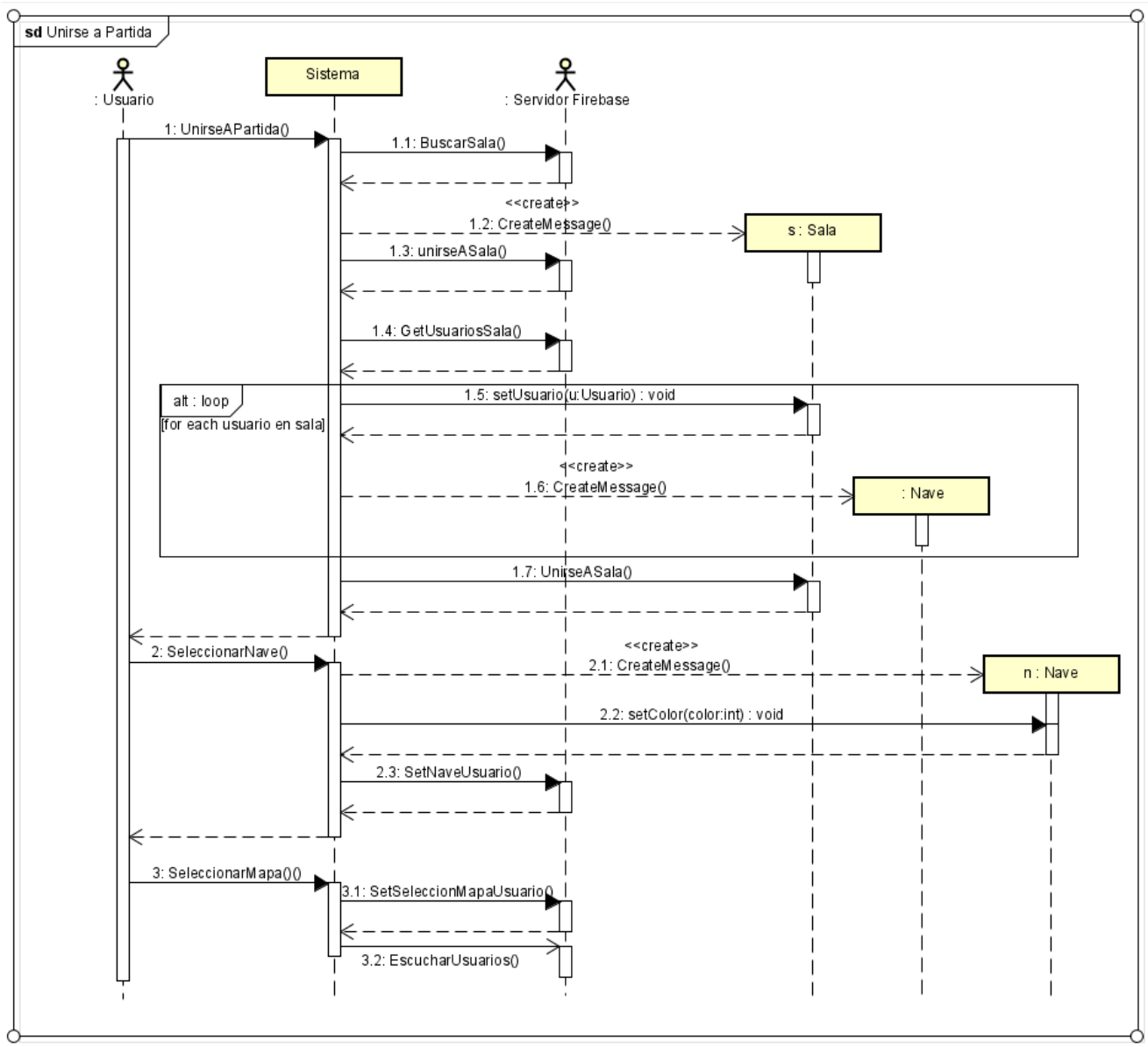

Figura 5.5: Diagrama de Secuencia de Unirse a Sala

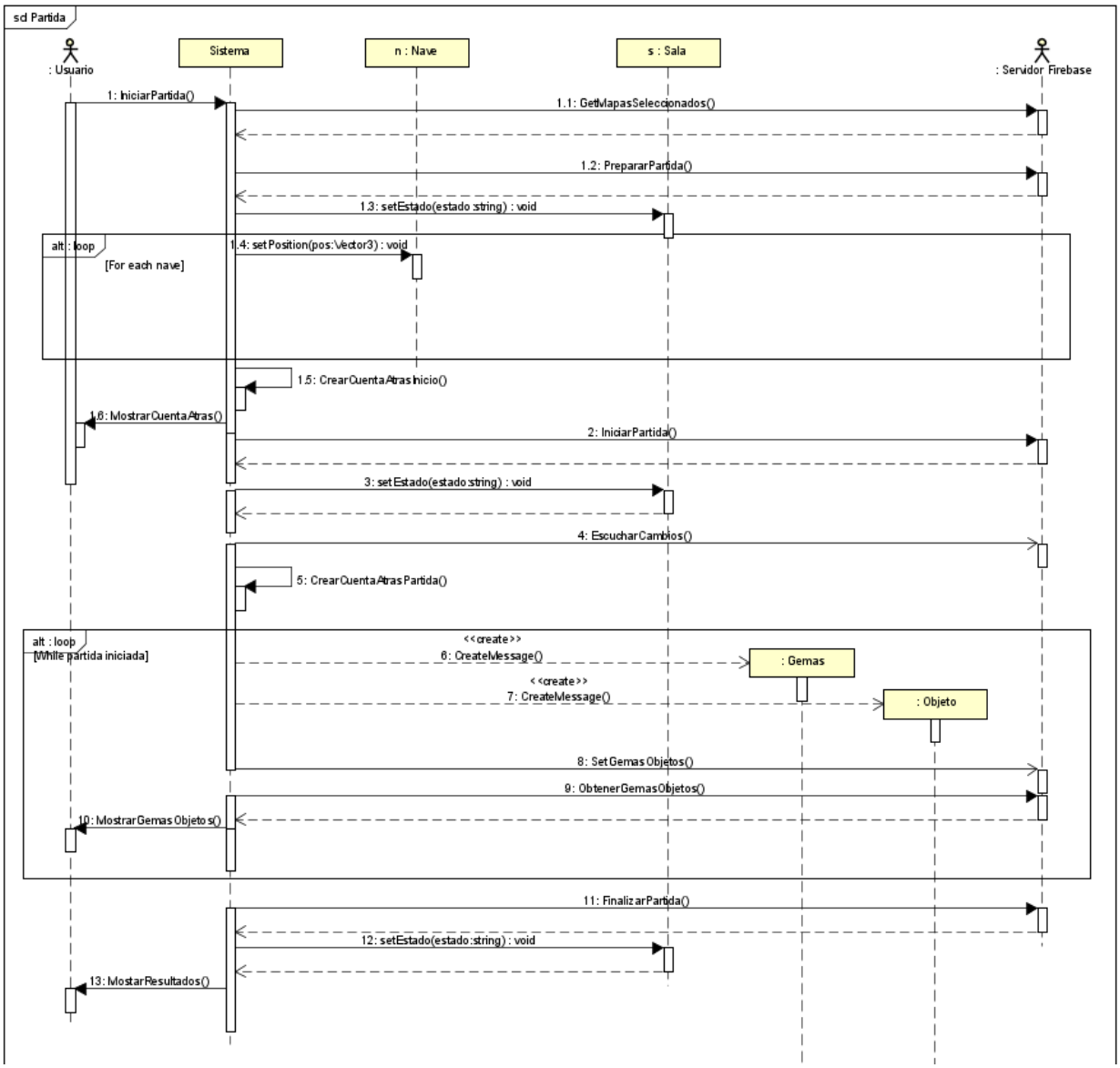

Figura 5.6: Diagrama de Secuencia de Partida

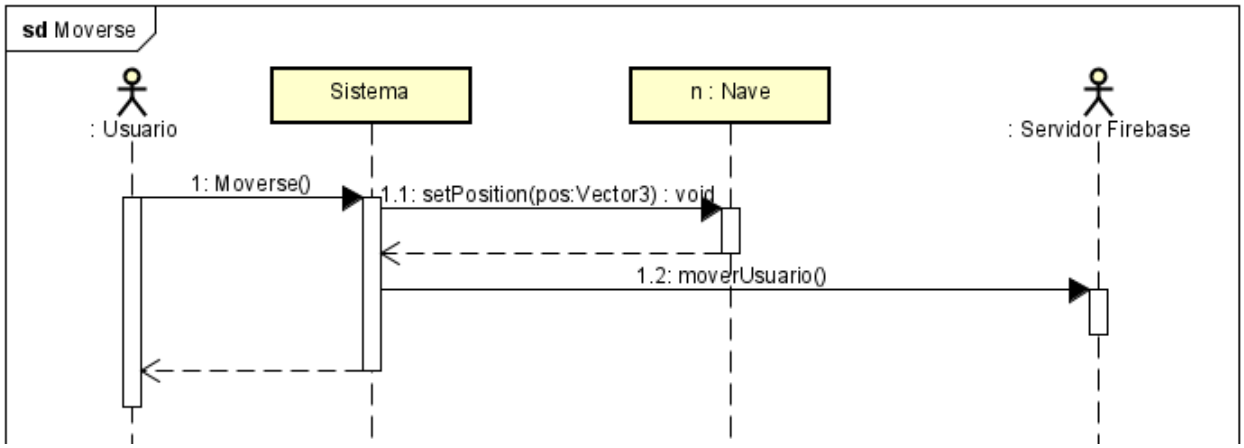

Figura 5.7: Diagrama de Secuencia de Moverse

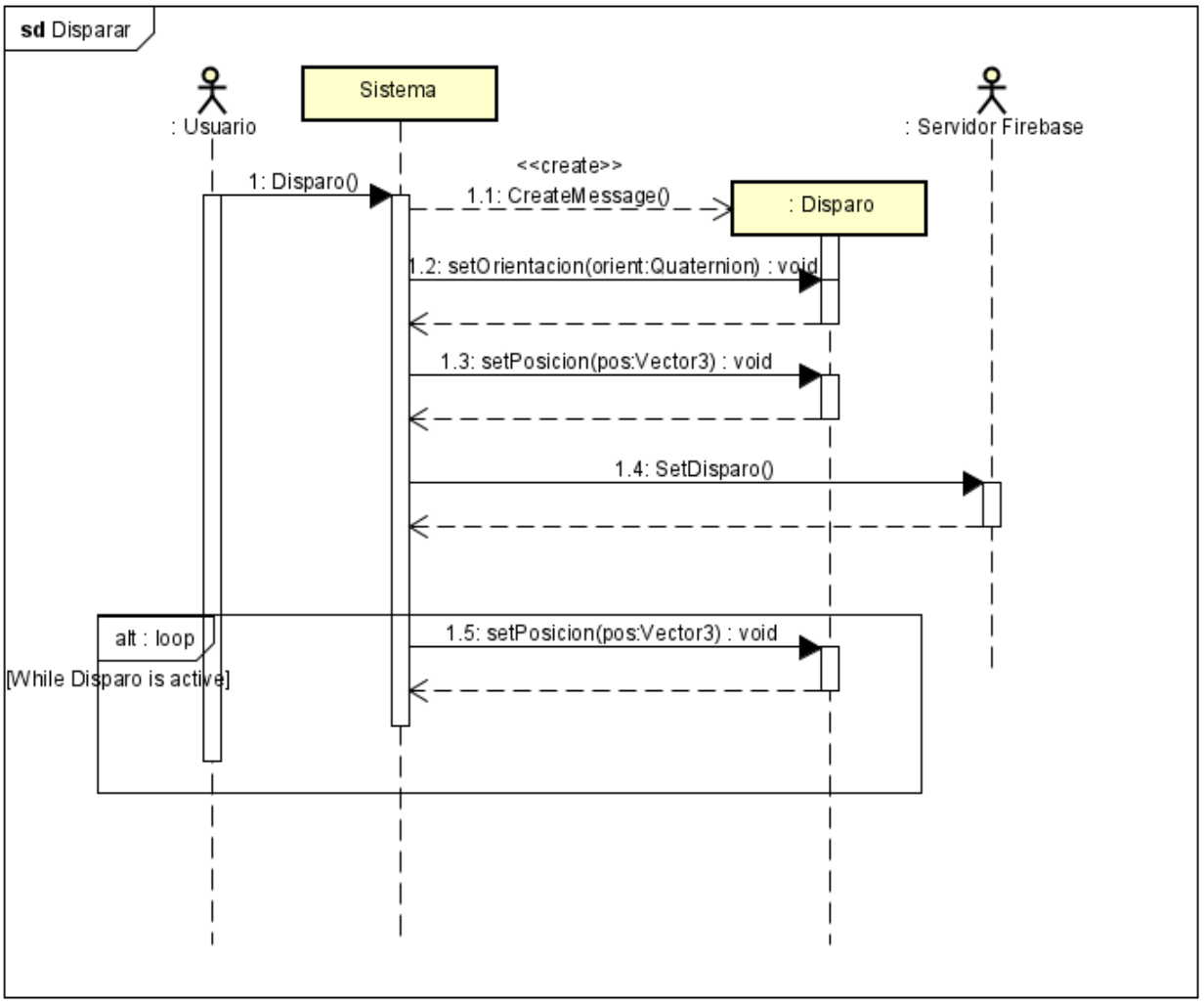

Figura 5.8: Diagrama de Secuencia de Disparar

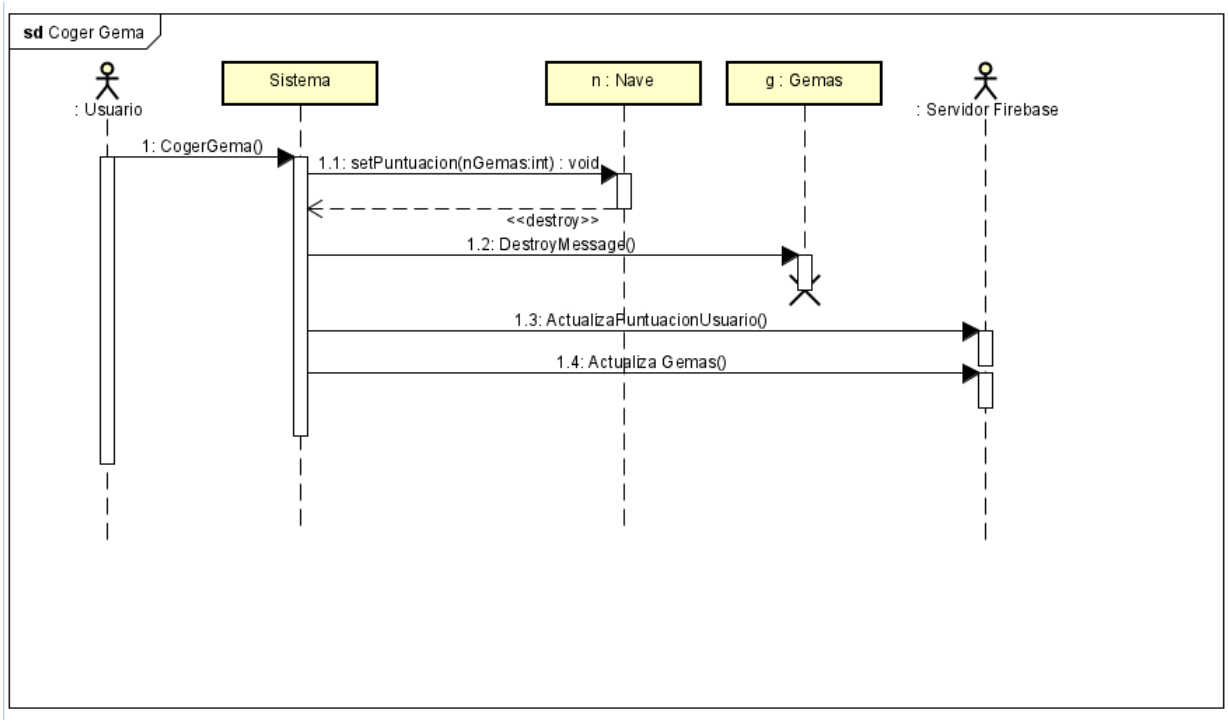

Figura 5.9: Diagrama de Secuencia de Coger Gema

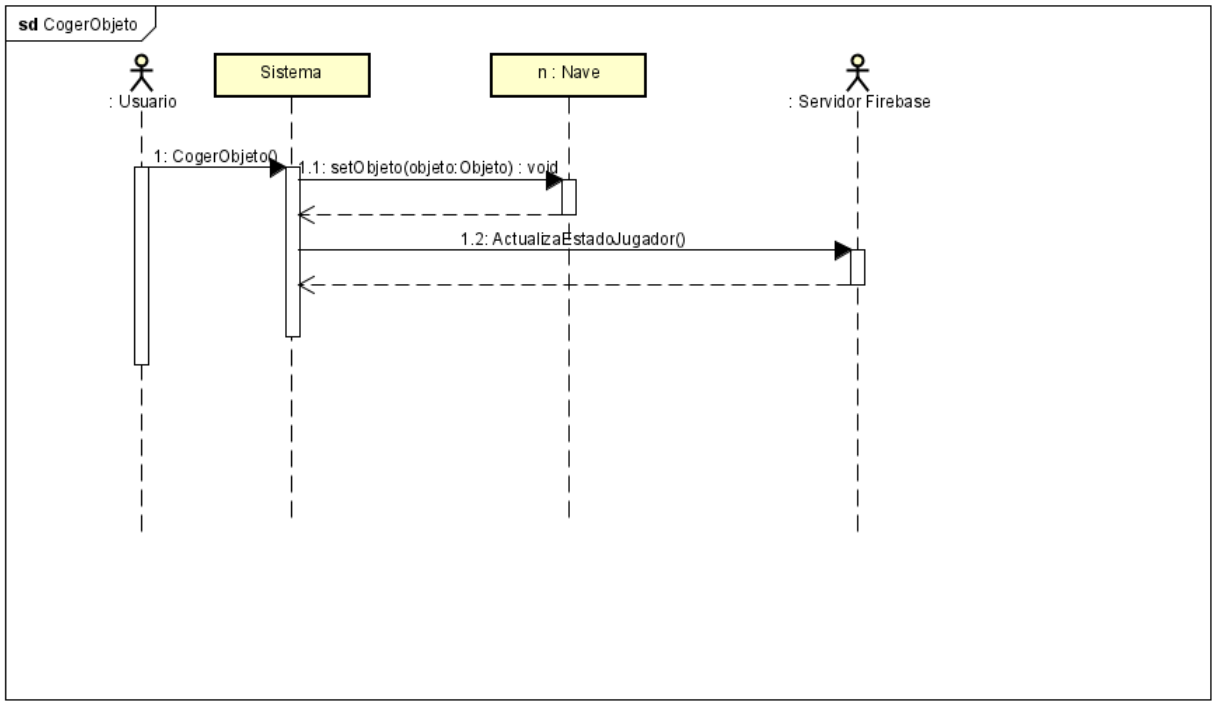

Figura 5.10: Diagrama de Secuencia de Coger Objeto

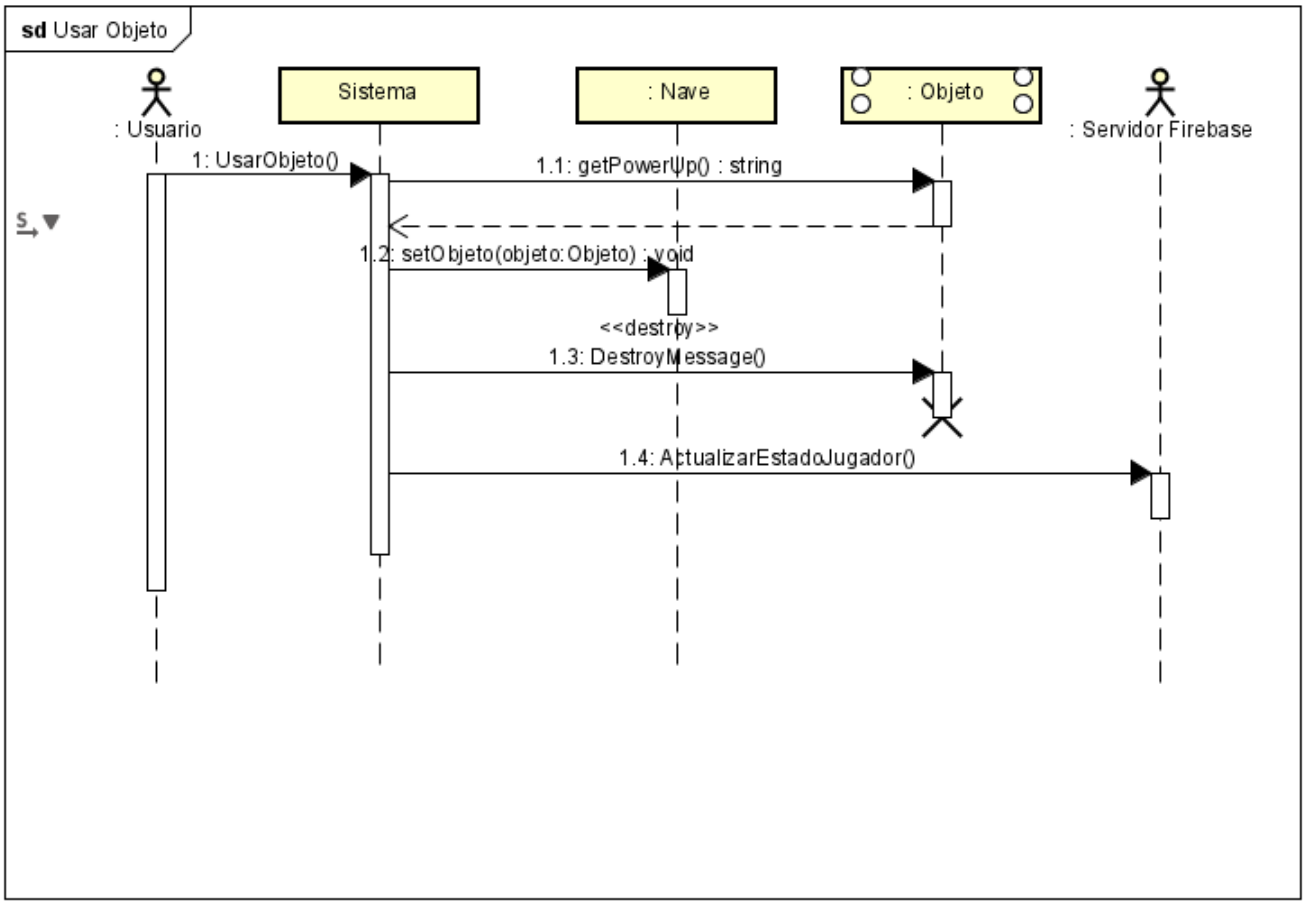

Figura 5.11: Diagrama de Secuencia de Usar Objeto

5.2.3 Diagrama de Despliegue

El diagrama de despliegue de la figura 5.12 nos muestra la arquitectura de ejecución del sistema, con los entornos de ejecución de las aplicaciones y servidores. Esto nos ayudará a modelar la topología del hardware.

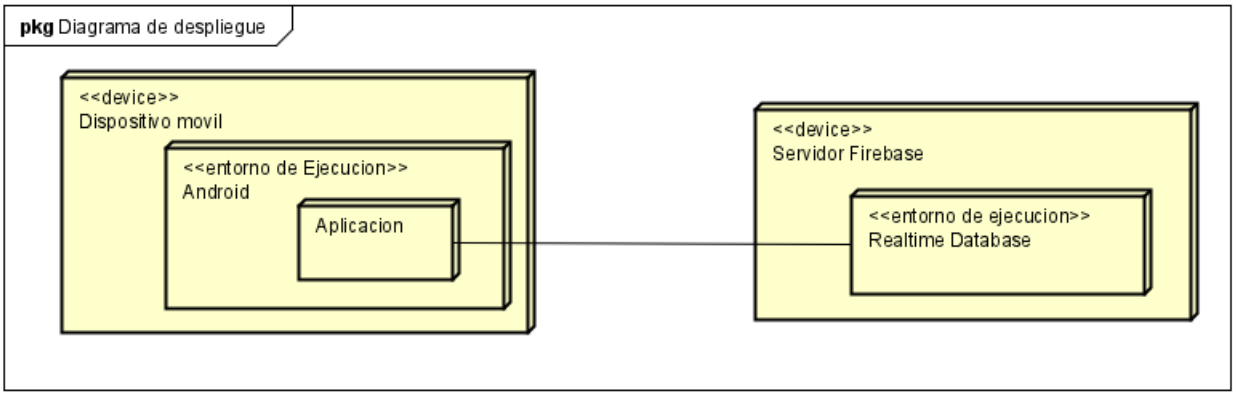

Figura 5.12: Diagrama de Despliegue

5.3 Implementación

En este apartado se describe el proceso de implementación del software, usando los elementos de Unity descritos en la sección 3.3 y la API de Firebase explicada en el apartado 3.4.

La instalación de Unity se hace a través de UnityHub, una plataforma de gestión de proyectos y versiones Unity. La versión del motor de Unity utilizada se trata de la 2019.4.1.f1. Se ha evitado las versiones más recientes de Unity (2020,2021) por su incompatibilidad con los servicios de Unity debido a que cambiaron el sistema de construcción de proyectos. A la hora de descargar la versión elegida tenemos que elegir el paquete de Android de construcción del proyecto.

Tras la descarga de Unity y la creación de un proyecto vació 2D lo primero que hacemos es introducir el SDK de Firebase [69]. Tras iniciar sesión en Firebase, agregamos nuestra aplicación a la plataforma, mediante el uso del ID de la aplicación, encontrado en los settings del proyecto. A continuación se introducen los archivos de configuración en el motor y se descarga el SDK de Firebase Unity.

Desde Unity nos vamos a Elementos >Importar Paquete >Paquete Personalizado, donde podemos elegir los paquetes de los productos que utilizaremos en el proyecto. Dentro de lo descargado elegimos los contenidos de la carpeta dotnet4, que se trata de la compatible con nuestra versiona actual de firebase. En esta carpeta elegimos firebaseDatabase.unitypackage que se corresponde al paquete con el SDK de RealtimeDatabase.

En Unity debemos cambiar la configuración de construcción. Esto es en File >Build Settings. Desde aquí cambiamos nuestra plataforma objetivo a Android para que Unity sepa que debe construir APKs con el proyecto. Una vez hecho esto podemos empezar a crear el juego.

En este proyecto se han creado dos *Scenes*, una dedicada al menú principal y al Lobby y otra dedicada a la partida principal. Al ser un juego en 2 dimensiones sin ningún tipo de perspectiva se ha elegido la opción de proyección ortográfica, que mostrara los elementos como si estuvieran en el mismo plano. Tanto el tamaño de la cámara como los *clipping planes* se han dejado con los valores por defecto, puesto que se ajustara de todas maneras el tamaño de los *GameObjects* en base al tamaño de sus *Sprites*.

5.3.1 Escena Lobby

La escena Lobby esta conformada por el menú principal y el menú del Lobby. Es la primera escena que el jugador se encuentra cuando inicia el juego. En ella un *Canvas* dibuja una interfaz sobre un fondo con temática espacial.

El *Canvas* se compone de tres *GameObjects*, cada uno correspondiente a diferentes estados de la interfaz. Estos *GameObjects* y sus interfaces se activarán y desactivarán según el usuario avance por esta, mostrándose solo uno a la vez.

El primero de estos *GameObjects* esta compuesto por los elementos de la interfaz que se aparecen al jugador nada más empezar la partida. Esta formado por el titulo del juego, un *InputField* para introducir el apodo del jugador, un botón para crear partida,un InputField para introducir un código de sala y un botón para unirse a la sala. También se encuentran por defecto oculto unos mensajes de error para el usuario. Estos saltarán durante un momento cuando el usuario no introduzca apodo o no haya introducido un código de partida correcto. El InputField del apodo comprobara que el total de caracteres introducidos sea menor que 50 y el InputField del código comprobara que sean menor que 4.

El segundo objeto aparecerá después de que el usuario se una a una sala. Se trata del menú de selección de nave. Se compone de un texto indicando las instrucciones al jugador y 4 botones, cada uno de ellos con el sprite de la nave que quiera seleccionar. El juego no deja elegir dos naves iguales por lo que leerá de la base de datos si alguien esta usando alguna de ellas y lo ocultara al jugador que acaba de entrar a la sala.

El tercer objeto del *Canvas* agrupa a los elementos de la sala de espera del Lobby. Entre ellos están un texto indicando el código de la sala, y cuatro textos, por defecto vacíos, que indica los nombres de las personas que se han introducido en la sala.

El *Canvas* ha sido configurado para que escale según el tamaño de la pantalla del dispositivo móvil que ejecuta la función. Como resolución de referencia se ha elegido la HD (1280x720) ya que la mayoría de teléfonos móviles de la actualidad presentan esta resolución o incluso mejor (Full HD por ejemplo).

La imagen del fondo de la pantalla se ha obtenido de un sprite y se ha colocado en una capa llamada Background. Esta capa se situara siempre al fondo de la escena, quedando por debajo de los demás objetos de la escena, incluyendo los elementos de la interfaz de usuario.

Por último, en la escena se encuentra un *GameObject* vacío llamado GameManager. La función de este objeto es simplemente la de contener el script principal del menú, llamado GameManagerLobby. Se trata de una clase que hereda de *MonoBehaviour* con la lógica de la escena. Debe estar en un objeto porque si no Unity no ejecutaría la clase. Esta clase es descrita en la sección 5.3.4.8.

5.3.2 Escena Juego

La escena Juego sera la escena donde se produzca la partida de la sala. Posee la misma configuración de cámara y *Canvas* que la escena Lobby. En esta escena las naves de los jugadores e moverán libremente por la pantalla recogiendo gemas hasta el final de la partida.

La interfaz de usuario es bastante más simple, teniendo solo cuatro textos indicando los nombres de los jugadores y su puntuación. También hay un texto centrado en la pantalla que hará de cuenta atrás, actualizándose cada segundo al inicio de la partida para luego ser desactivado.

La imagen del fondo es otro Sprite que, de manera similar a lo indicado en la escena Lobby, se ubica en la capa BackGround para que se muestre por debajo de la IU y de las naves de los jugadores.

Por último se encuentra otro GameObject vació llamado GameManager, que sera el encargado de sostener el script de la lógica de la escena. Este script contiene la clase homónima GameManager, heredera de MonoBehaviour y que es descrita en la sección.
5.3.4.9.

5.3.3 Prefabs

En las dos escenas existen objetos que se usan más de una vez. Para evitar la creación manual de estos objetos se han creado *Prefabs* que serán creados mediante un script.

Representando al usuario tenemos la nave espacial PlayerShip. Sus principales componentes son un RigidBody2D, un Box *Collider* y un Sprite Renderer. El RigidBody2D esta configurado como kinemático, de manera que el motor de físicas de Unity no afectara al movimiento de este personaje, siendo únicamente controlado por Scripts. El SpriteRenderer se encargara de asignar el Sprite de la nave elegida por el jugador al *GameObject*, al igual que situarlo por delante de la capa de Background.

Para obtener las interacciones entre nave y objetos, gemas y otras naves, se utiliza el *Collider* con forma de caja. Este collider rodea la nave con un cuadrado en vez de ser ajustado al Sprite. Esto se hace porque las naves tienen diferentes formas por lo que podría ser una ventaja o desventaja según la nave elegida del jugador. En su lugar la colisión esta formada por un cuadrado de tamaño similar a todos los *Sprites*.

Los otros componentes son Scripts que controlaran el movimiento de la nave. Uno, PlayerController (ver apartado 5.3.4.5, es utilizada por la nave que controla el usuario del dispositivo. El otro, EnemigoController (ver apartado 5.3.4.6), es el script para las naves de los demás jugadores que aparecen en la pantalla. Según el *Prefab* sea del usuario o de los contrincantes, se desactivara un script u otro. Ambos script son descritos en el apartado 5.3.4.

PlayerShip tiene dos *GameObjects* hijos, cuyo Sprite es el fuego del cohete de la nave, uno de ellos contiene un Sprite más pequeño que se mostrará cuando la nave esta quieta y otro contiene un Sprite más grande para cuando la nave este en movimientos. Ambos *GameObjects* tienen una pequeña animación para dar más dinamismo al fuego.

Otro *Prefab* es el disparo del jugador que al igual que la nave tiene como componentes un SpriteRenderer, un *Box Collider* ajustado al *Sprite*, y un *RigidBody2D* kinemático. Para calcular el movimiento de este disparo se utiliza el script Disparo, descrito en 5.3.4.7.

El último *Prefab* es la gema que los jugadores cogerán para subir su puntuación. Esta gema esta compuesta por solo un Script que controlara la colisión con las naves, un colisionador y el SpriteRenderer que contiene su Sprite. No necesita ningún Rigidbody2D puesto que este objeto es estático.

5.3.4 Clases

En esta sección se describen la clases creadas para el proyecto. Algunas de esta clases implementan las funcionalidades descritas en los diagramas de secuencia de la sección 5.2.2

5.3.4.1 User

Esta clase representa el usuario del modelo de dominio [5.2]. Es una clase para representar datos en vez de para la ejecución del juego por lo tanto no hereda de *MonoBehaviour*. Cada usuario tiene una instancia de esta clase, usada, además de para guardar su apodo, para guardar la nave elegida, los proyectiles que lanzan y la puntuación conseguida. Los valores, salvo el de usuario se resetean al acabar la partida. Sus atributos son:

- **username (string):** El apodo del jugador.
- **playerNumber (int):** El numero del jugador en la partida.
- **playerObject (GameObject):** El *GameObject* de la nave del usuario.
- **ship (Sprite):** El Sprite de la nave utilizado por playerObject.
- **proyectil (GameObject):** el proyectil lanzado por la nave.

Aparte de las correspondientes operaciones get y set de cada uno de los atributos y su constructor, se ha definido el método SetSpriteShip(Sprite), encargada de precargar un Sprite al componente SpriteRenderer del playerObject.

5.3.4.2 Gema

Esta clase representa la clase Gema del modelo de dominio [5.2]. Gema es una clase simple que representa el objeto gema. Esta clase ha sido necesaria para llevar un control de la información en la base de datos, ya que asigna el objeto gema a un numero identificativo, que permitirá a los jugadores saber que gema tiene cambios en la base de datos. (ver apartado5.3.4.9). Implementa dos atributos con sus respectivas operaciones get y set de cada una.

- **go (GameObject):** El *GameObject* de la gema
- **nGema (int):** Identificador de Gema

5.3.4.3 Lobby

Esta clase representa la sala de la partida [5.2]. En principio iba a ser una clase que se puede instanciar como objeto, pero al final se ha decidido que fuera por ser una clase estática. La razón de este cambio es por una característica de la plataforma de Unity basada en que una escena es independiente de otra. Esto supone que ninguna de las instancias de los objetos creados en una escena se guardan y no se pueden utilizar en otra escena. La excepción a esta norma es la clase estática, cuyos datos permanecen intactos tras los cambios de escena.

Otra forma de haber podido resolver este conflicto es mediante el uso de un fichero Json donde se podrían guardar los datos, cargar la nueva escena y, volver a leer los datos de ese

fichero. Sin embargo Json no puede usar la clase *GameObject*, que es necesario tener en cuenta entre escenas.

Los atributos implementados son:

- **userList (static List<User>):** Una lista (array) de los usuarios del lobby.
- **code (static string):** El código de la sala
- **thisPlayerNumber (static int):** Es el numero de usuario del usuario del dispositivo. Este numero se usa como control debido a que se ejecutaran métodos adicionales según si el usuario es el creador de la partida o el que lleva más tiempo, representado usualmente con el numero 1. Ver apartado 5.3.4.9.

5.3.4.4 InputManager

InputManager es una clase encargada de obtener las acciones de un usuario sobre la pantalla táctil. Esta clase crea un sistema de subscripción de eventos que permite a otras clases subscribirse para ser notificadas de alguna entrada del usuario, para poder ejecutar métodos tras esas entradas. Esto lo logra mediante el uso de *delegados*.

Un Delegate es un tipo en C# que representa una referencia a un método con unos parámetros y tipo de valor devuelto determinado. Los controladores de eventos no son más que métodos que se invocan a través de delegados [70].

InputManager hereda de *MonoBehaviour* porque necesita ejecutarse antes de todos los scripts que lo utilizan. Para ello podemos poner la clausula [DefaultExecutionOrder(-1)] al principio de la clase. Se ejecuta la funcion Awake() para obtener una referencia a la cámara, que se usara para calcular la posición del dedo.

InputManager se subscribe a dos eventos que se activaran con las entradas del usuario, uno cuando se ponga el dedo por pantalla y otro cuando se mueva el dedo. Cuando se produzcan estas situaciones, se convertirá la posición del dedo en coordenadas dentro de la escena. Otras clase pueden utilizar esta posición, junto con el punto en el tiempo que se produjo el evento, para realizar otros métodos. Por ejemplo, la clase PlayerController (ver apartado 5.3.4.5), utiliza esta posición para calcular el movimiento de la nave del jugador.

5.3.4.5 PlayerController

PlayerController es la clase principal que controlara el movimiento y acciones de la nave del usuario. Esta situada como componente del *Prefab* PlayerShip y hereda de *MonoBehaviour* ya que se ejecutara en simulación. Esta ocupada del movimiento del personaje, como se indica en el diagrama de secuencia 5.7, y de la obtención de gemas, como se detalla en el diagrama 5.9.

Awake inicializa la clase, añadiendo el InputManager como un componente de este *GameObject* y obteniendo referencias a los *GameObjects* hijos de PlayerShip, los fuegos del cohete, así

como una referencia a el objeto GameManager.

Utilizando los métodos de InputManager, se utiliza la posición del dedo del jugador para calcular la posición en la escena a la que debe moverse PlayerShip. La coordenada del dedo se guarda en una variable global, de manera que en cada fotograma, se calcula la siguiente posición de la nave. Update moverá frame a frame la nave hacia el último punto donde se ha tocado la pantalla, guardado en la variable global ("coordMov"). Se moverá a una velocidad determinada no afectada por la tasa de fotogramas gracias al uso de Time.deltaTime, que contiene la información del tiempo de que ha tardado el último frame en completarse. También calcula la rotación que debe tener el objeto calculando el ángulo entre el eje x y el segmento que une la nave con la posición a la que se debe mover (ver figura 5.13).

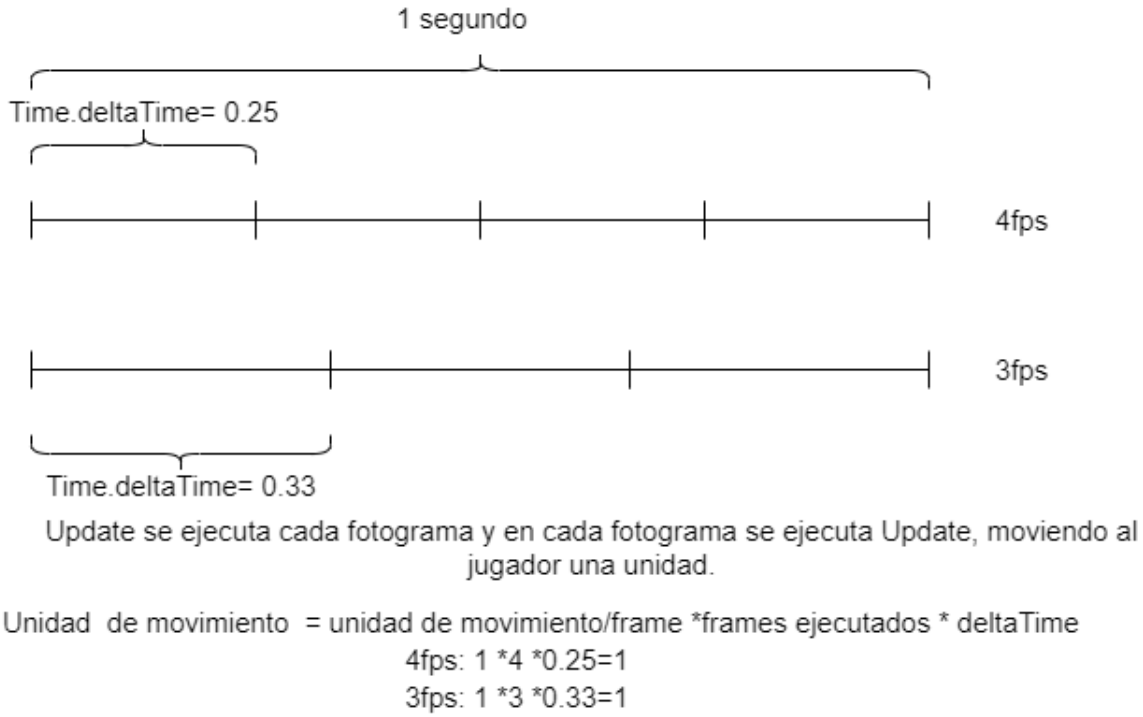

Figura 5.13: Delta time y la ejecución de físicas independientemente de los fotogramas por segundo.

Si se detecta dos pulsaciones del dedo seguidas, se creara un *GameObject* a partir del prefab Disparo en la posición y rotación actual de la nave. También se enviara un mensaje a GameManager para que este escriba en la base de datos que el disparo ha sido creado (ver apartado 5.3.4.9).

5.3.4.6 EnemigoController

EnemigoController es otro Script situado en el *Prefab* PlayerShip, que se ejecutará si la nave pertenece a un contrincante y no esta controlada por el jugador. Es muy similar a PlayerController pero no esta atada a ningún evento recibido por input del jugador. Contiene una

función Update que hace lo mismo que la función Update de PlayerController, pero esta vez la variable global "coordMov" es actualizada mediante otro método que sera llamado por el GameManager, con la posición adecuada sacada de la base de datos.

Para el movimiento de las naves enemigas se pensó en usar solo la clase GameManager para que actualizara el movimiento de estas. GameManager en un principio iba a obtener las coordenadas de la base de datos correspondientes a las naves contrincantes y se las iba a colocar directamente en los PlayerShip. Esto sin embargo, si se producía cada fotograma iba a consumir muchos recursos de red. Si se ejecutaba cada pocos frames se consumirían menos recursos pero produciría un movimiento no fluido de las naves contrincantes, como si se teletransportaran.

Al final se ha usado el método llamado por GameManager, para que el motor de físicas calcule el movimiento a partir de unas coordenadas que no tiene porque ser actualizadas en cada frame. Cuantos más frames pasen entre actualización más fluido será el movimiento de la nave, pero menor los recursos en red que se utilizaran.

5.3.4.7 Disparo

Disparo es un simple Script que controla el movimiento rectilíneo del disparo cuando es creado. La función Start obtiene el componente Rigidbody2D y ejecuta la función OnDestroy con un delay, el tiempo de vida del disparo. Esta encargado de la implementación del disparo, como se indica en el diagrama 5.8.

Update mueve en cada fotograma el disparo aun cierta velocidad y hacia el frente del disparo, obtenido usando el ángulo de rotación del disparo, originalmente el ángulo en el que la nave estaba situada en el momento de la acción.

Disparo también ejecuta OnTriggerEnter2D ejecutado cuando el *Collider* del *Prefab* Disparo entra en contacto con otro *Collider*. Dentro de esta función se observa que objeto ha sido el causante y si es una nave enemiga, envía un mensaje a esta indicándola que ha sido golpeada, mensaje aun no implementado.

5.3.4.8 GameManagerLobby

Es la clase principal encargada de la lógica de la escena Lobby. Hereda de *MonoBehaviour* y se conectara a la base de datos para introducir al usuario en una partida. Al inicio de la aplicación, se recupera una instancia de la base de datos y creará una referencia al nodo raíz de esta.

Si el usuario pulsa el botón de Crear Partida, se ejecutara la función CrearPartida, descrita en la figura 5.4. Esta función primero comprueba si se ha introducido un apodo. Si no se ha introducido se avisa al jugador. Con el apodo, la función crea una instancia de User para el usuario del dispositivo, asignándole el numero 1 de la partida. El usuario con ID 1 es el más importante de la sala, debido a que se encargara de enviar *flags* a la base de datos, necesarios para la sincronización de estados entre jugadores.

CrearPartida es la encargada de crear una sala, generando un código de cuatro letras alfanuméricas, y ponerla en la base de datos como nodo hijo del nodo raíz. Dentro del nodo de la sala, se colocan flags de control, unos indicando las naves que han sido seleccionadas y otro que indica si la partida ha sido empezada, con valores booleanos (ver figura 5.14). Tras esto, mediante una corrutina, se ejecuta paralelamente al juego un método para cambiar la interfaz a la elección de naves.

Figura 5.14: Estado de la base de datos tras CrearPartida

En la interfaz de selección de naves (ver figura 5.15), se crea una referencia al nodo "Colores" de la base de datos y pasará a escuchar cuando se produzca un cambio en ese nodo, que ejecutará una función encargada de leer las naves disponibles y cambiará la interfaz en función de si alguna está seleccionada, impidiendo al jugador escogerla. La base de datos notificará si se produce algún cambio en este nodo y actualizará la interfaz de selección de la nave.

Figura 5.15: Menú de las Naves

Según el botón o nave que haya pulsado, se ejecutará una función que cambiará el valor correspondiente al color en la base de datos y se creará el *GameObject* PlayerShip para el control

del usuario. Esta nave tendrá el control del Script PlayerController en vez de EnemigoController. Una vez hecho esto se cambia la interfaz de usuario nuevamente para mostrar el nombre de usuario de los jugadores que entren en la partida y el código de la sala.

Una vez cambiada la interfaz de nuevo a la correspondiente a la figura 5.16, se termina introduciendo los datos del jugador en la base de datos, entre ellos el nombre de usuario y un lugar otra poner la puntuación. Esto lo hace de la forma indicada en la figura 5.17. De la base de datos se crea una referencia al nodo "Jugadores" y se pone a escuchar cambios en ese nodo, para ejecutar un método de actualización de la sala, cuando, por ejemplo, se introduzcan jugadores. También se crea una referencia al nodo Empezado, un "flag" que se escucha para saber cuando empieza la partida.

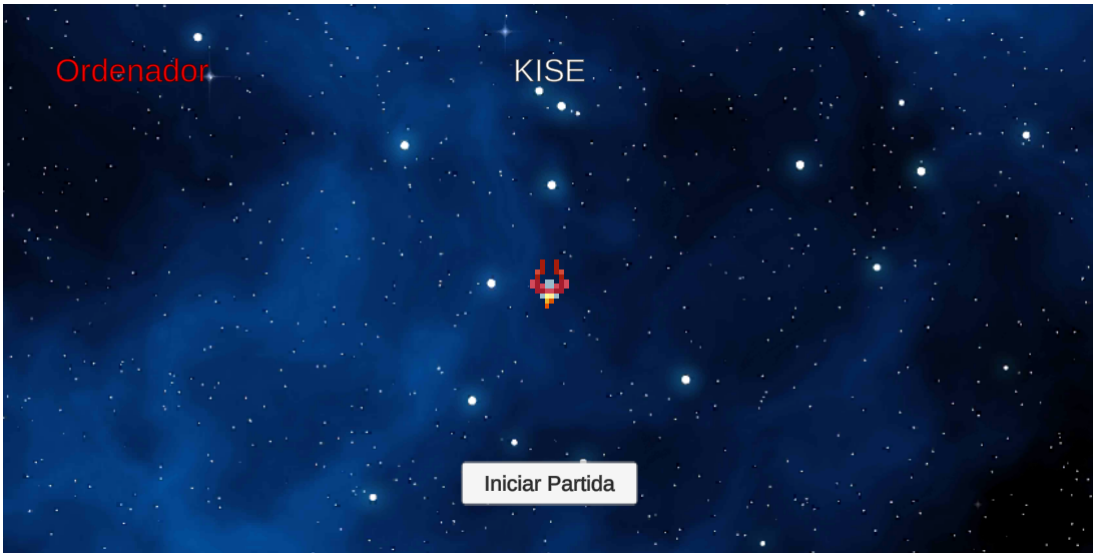

Figura 5.16: Sala de espera tras crear partida

Si un jugador se une a la sala, actualizara la base de datos, que mandara un mensaje a los demás usuarios, y se encargarán de mostrar el PlayerShip del contrincante así como su nombre en la interfaz. Este PlayerShip, para los demás usuarios, tendrá como componente el script EnemigoController en vez de PlayerController. Esta función esta también encargada de actualizar el movimiento de las naves enemigas enviando un mensaje a EnemigoController con las nuevas coordenadas del usuario, recogidas del nodo "coord" de la base de datos (ver figura 5.17).

Si en el menú Principal el usuario pulsa Unirse a Partida entonces se ejecutara la función UnirsePartida, descrita en el diagrama de secuencia 5.5. Esta función lo primero que hace es comprobar primero, que el usuario se haya puesto un apodo, segundo, que haya introducido un código de sala correcto y tercero, que ese código de sala sea de una sala creada en la base de datos. Si falla alguna de estas comprobaciones el usuario sera notificado.

UnirsePartida lee la referencia correspondiente a la sala y por cada jugador que encuentra crea un objeto PlayerShip con la información encontrada en la base de datos. Este objeto PlayerShip es el correspondiente a los contrincantes, por lo que contiene EnemigoController. Tras ello crea una instancia de User con el nombre del usuario y un numero correspondiente al

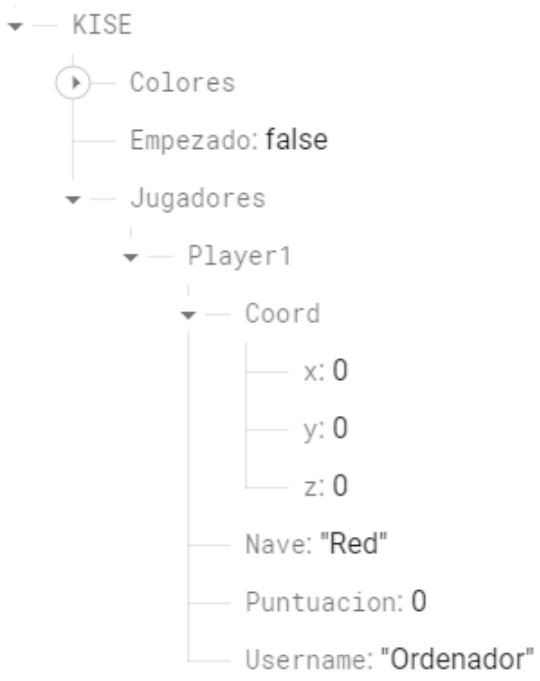

Figura 5.17: Estado de la base de datos tras crear la sala

orden de entrada del usuario (si el usuario entra el tercero, su numero de User es el 3). A partir de aquí se ejecuta el método que cambia la interfaz al menú de selección de naves, espera la entrada del jugador y lo introduce en la partida, actualizando la base de datos.

Cuando el flag "Empezado" de la base de datos cambie de valor, debido a que el jugador creador de la partida le de al botón Empezar Partida (solo le aparece a el), se cargará la escena de Juego y se procederá a empezar la partida.

OnApllicattionQuit, por último desconectara a los jugadores de la base de datos en caso de que se salga de la aplicación.

Por último, cuando el jugador está en la sala, en Update(), cada cierto número de frames se actualiza el valor de las coordenadas de la posición del playerShip del usuario a la base de datos, al nodo "coordtt correspondiente al usuario.

5.3.4.9 GameManager

GameManager es el Script principal de la escena Juego. Al igual que GameManagerLobby se encarga de la comunicación con la base de datos, obteniendo una instancia de esta y la referencia a el nodo de la sala cuando se carga la escena. Esta también encargada de la comunicación con la base de datos para sincronizar las acciones de los jugadores.

IniciarPartida es el primer método que se ejecuta y sigue alguno de los pasos descritos en el diagrama de secuencia 5.6. Es llamada por Start() y su función es volver a crear los objetos de las naves tanto del usuario como él de los contrincantes y ponerlos en la posición inicial. Se

encarga de añadir un evento en la base de datos que esperará a que los usuarios activen cuando sean completamente cargados en escena. Esto se hace para sincronizar estados.

Si el identificador del User de la partida tiene el numero 1, el sistema de su dispositivo se encargará de crear dos nodos en la base de datos, un "flag" para la sincronización de la cuenta atrás de los jugadores y otro nodo donde se introducirán las gemas en la partida, como se indica en la figura 5.18. Al final de IniciarPartida, los jugadores se considerarán cargados en escena, por lo que envían a la base de datos su estado. Una vez todos los jugadores hallan sido cargados, se inicia la cuenta atrás.

El método encargado de crear una cuenta atrás en la interfaz del usuario empieza a escuchar cambios en el nodo "CuentaAtras", que indicará que la partida comienza. Tras ello, ejecuta una cuenta atrás, mostrando en cada segundo los restantes para el comienzo de la partida. Una vez hecho esto, el usuario con Id 1 envía el mensaje a la base de datos cambiando "CuentaAtras". Cuando la base de datos notifique a los jugadores de que la partida ha comenzado se ejecuta otro método, encargado de crear las escuchas de los diferentes nodos de control de la partida ,"Jugadores" y "Gemas".

Cuando se produzcan cambios en el nodo "Jugadores", se ejecuta una función que actualiza las posiciones de los jugadores y sus disparos. Si se producen cambios en el nodo "Gemas", el sistema se encargará de gestionar las gemas que aparecen en pantalla, de manera que estén en las mismas posiciones para todos los jugadores, y si alguno de ellos coge la gema, desaparezca para todos. Al finalizar el método, los jugadores empezaran la partida y se crearan las gemas. Tras el inicio de la partida la base de datos tendrá una estructura familiar a la figura 5.18.

Las gemas serán creadas en una corrutina ejecutada en el dispositivo del usuario con UserID 1. El método encargado creará los objetos gemas pero no los mostrara al usuario, si no que lo enviará a la Base de datos, para que luego se actualice cuando sea notificado del cambio en el nodo de "Gemas" de la base de datos.

Update y OnApplicationQuit hacen lo mismo que en GameManagerLobby, una envía cada ciertos fotogramas la posición del objeto del usuario y la otra desconecta al usuario d la base de datos.

5.3.4.10 GemsController

Es una clase simple encargada del control del *Collider* de la gema. Cuando un objeto entra en su colisionador se ejecuta OnTriggerEnter2D, que comprueba que objeto ha sido el que colisiona y si es la nave del jugador, envía un mensaje a la GameManager para que suba la puntuación del jugador y se autodestruye.

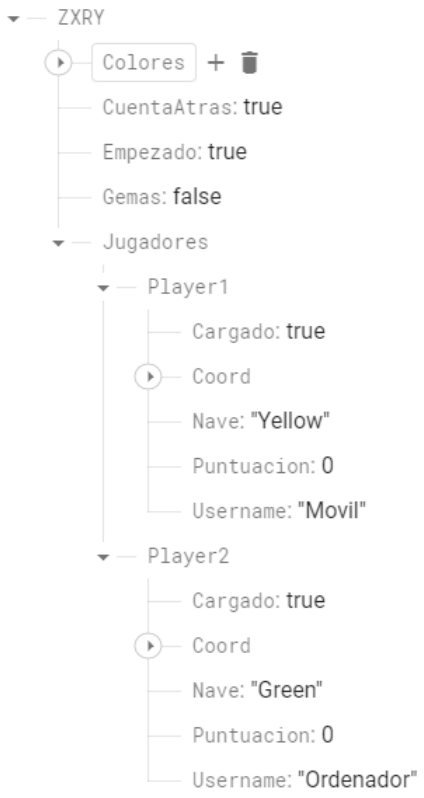

Figura 5.18: Contenido de la base de datos tras el inicio de la partida

5.4 Pruebas

En esta sección se detallan las pruebas realizadas sobre la implementación realizada.

Identificador	CP-01
Descripción	Se intenta crear una sala sin poner nada en el InputField de apodo
Entradas	Botón de Crear Partida, sin texto en InputField de apodo
Salida Esperada	Mensaje de error, indicando al usuario que debe poner un apodo.

Tabla 5.12: Caso de Prueba 1

Identificador	CP-02
Descripción	Se intenta unirse a una sala sin poner nada en el InputField de apodo.
Entradas	Botón de Unirse Partida, sin texto en InputField de apodo.
Salida Esperada	Mensaje de error, indicando al usuario que debe poner un apodo.

Tabla 5.13: Caso de Prueba 2

Identificador	CP-03
Descripción	Se intenta unirse a una sala sin poner nada en el InputField del código de sala
Entradas	Botón de Unirse Partida, sin texto en InputField de sala.
Salida Esperada	Mensaje de error, indicando al usuario que debe poner un código de sala correcto.

Tabla 5.14: Caso de Prueba 3

Identificador	CP-04
Descripción	Se intenta unirse a una sala que ya contiene cuatro jugadores
Entradas	Botón de Unirse Partida, 4 jugadores en la sala.
Salida Esperada	Mensaje de error, indicando al usuario que la sala tiene 4 jugadores

Tabla 5.15: Caso de Prueba 4

Identificador	CP-05
Descripción	Se intenta unirse a una partida sin poner nada en el InputField de apodo.
Entradas	Botón de Unirse Partida, sin texto en InputField de apodo.
Salida Esperada	Mensaje de error, indicando al usuario que debe poner un apodo.

Tabla 5.16: Caso de Prueba 5

Identificador	CP-06
Descripción	El usuario crea y se una a una sala.
Entradas	Botón de crear partida, con un apodo introducido.
Salida Esperada	El usuario se introduce en la sala. Se crea la interfaz de selección de naves.

Tabla 5.17: Caso de Prueba 6

Identificador	CP-07
Descripción	El usuario se une a una sala.
Entradas	Botón de unirse a partida, con un apodo introducido y un código de sala existente.
Salida Esperada	El usuario se introduce en la sala. Se crea la interfaz de selección de naves, ocultando las naves ya elegidas por otros jugadores.

Tabla 5.18: Caso de Prueba 7

Identificador	CP-08
Descripción	El usuario elige una nave.
Entradas	En la pantalla de selección de naves, elegir una tocando en la pantalla.
Salida Esperada	Se crea la interfaz, que indica el nombre del usuario y el código de la sala. Se crea la nave elegida por el usuario. Se actualiza en otros dispositivos la interfaz de selección de la nave, ocultando la elegida por el usuario

Tabla 5.19: Caso de Prueba 8

Identificador	CP-09
Descripción	El usuario se mueve por la sala.
Entradas	En una sala, tocar la pantalla.
Salida Esperada	La nave del jugador se mueve a hacia la posición del dedo. Si existen otros jugadores en la sala, se actualizara el movimiento en sus pantallas.

Tabla 5.20: Caso de Prueba 9

Identificador	CP-10
Descripción	El usuario creador de la sala, inicia la partida.
Entradas	En la sala,el usuario que crea la sala tocar el botón de crear partida.
Salida Esperada	Todos los usuarios en la sala se colocan en posición de inicio, se actualiza sus interfaces con los elementos de la partida, se hace una cuenta atrás y, cuando esta finalice, se deja mover a los jugadores.

Tabla 5.21: Caso de Prueba 10

Identificador	CP-11
Descripción	El usuario dispara una bala desde su nave.
Entradas	En la partida, el usuario toca dos veces la pantalla.
Salida Esperada	Desde la nave del usuario, se crea un disparo que viajará en la dirección que la nave encaraba y, tras dos segundos, el disparo desaparecerá. Este disparo será visto por todos los usuarios de la partida.

Tabla 5.22: Caso de Prueba 11

Identificador	CP-12
Descripción	El usuario no dispara una bala, puesto que ha disparado recientemente.
Entradas	En la partida, el usuario toca dos veces la pantalla, cuando existe un disparo suyo en la pantalla.
Salida Esperada	No ocurre nada

Tabla 5.23: Caso de Prueba 12

Capítulo 6 Conclusiones y trabajo futuro

Tras todo los mostrado, se han logrado todos los objetivos marcados en la sección 1.1, se ha estudiado el estado actual del sector de los videojuegos, se han analizado herramientas de creación de videojuegos, se ha realizado el diseño conceptual del videojuego y se ha diseñado y implementado el software.

En este TFG se ha desarrollado un prototipo de videojuego multijugador online para dispositivos móviles, usando el motor de videojuegos Unity y desarrollando el entorno multijugador a través de la base de datos Realtime Database de Firebase, una base de datos NoSQL. Una vez acabado el proyecto podemos decir que se han hecho las siguientes aportaciones:

- Se ha analizado e investigado diferentes herramientas destinadas al desarrollo de videojuegos y se ha comparado sus características y funciones.
- Se han investigado diferentes herramientas y servicios destinados al desarrollo de un *back-end* para la creación de un entorno multijugador o de sincronización de estados en aplicaciones, y se ha comparada sus características y funciones.
- Se ha detallado la estructura y el funcionamiento del motor gráfico Unity, los elementos que lo componen y las principales funciones y métodos de su API.
- Se ha descrito el funcionamiento y estructura de la base de datos Realtime Database, su sistema de notificaciones de eventos, y los métodos y funciones de su API para leer y escribir en la base de datos.
- Se ha investigado sobre los documentos de diseño de videojuego que detallan las características del proyecto durante todo el ciclo de su desarrollo, y se ha aportado un esquema propio de documento.
- Se ha descrito el funcionamiento de la aplicación y todos los elementos que lo componen, desde la instalación del software necesario, hasta el proceso que sigue el videojuego para sincronizar los usuarios de una partida.

6.1 Trabajo futuro

Como parte del trabajo a hacer en el futuro, se proponen las siguientes tareas:

- Terminar el prototipo y lanzar la versión completa de la aplicación, al igual que la creación de nuevo contenido, como el desarrollo de nuevos mapas, la posibilidad de unirse a salas publicas y a la implementación de nuevas mecánicas.
- Optimizar y mejorar las funcionalidades de la aplicación, usando herramientas de medición de rendimiento.
- Analizar la retroalimentación de jugadores para mejorar aspectos de la interfaz o el control y garantizar la usabilidad.
- Crear un sistema de autenticación que permita una administración eficaz de los usuarios de la aplicación y de la base de datos.

Bibliografía

[1] Lendhart, Amanda. 6 de agosto 2015. Vídeo Games are Key Elements in Friendship for Many Boys. Pew Research Center. https://www.pewresearch.org/internet/2015/08/06/chapter-3-video-games-are-key-elements-in-friendships-for-many-boys/ (Visitado el 01-06-2022)

[2] Clement, J. 24 de febrero de 2022. Most played games on Steam in 2021, by peak number of concurrent players. Statista. https://www.statista.com/statistics/656278/steam-most-played-games-peak-concurrent-player/ (Visitado el 12-03-2022)

[3] Wijman, Tom. 1 de julio de 2021. Global Games Market Report The VR & Metaverse Edition. Newzoo. Obtenido de https://newzoo.com/insights/trend-reports/newzoo-global-games-market-report-2021-free-version

[4] Clement, J. 9 de mayo 2022. Online gaming - statistics & facts. Statista https://www.statista.com/topics/1551/online-gaming/ (Visitado el 15-03-2022)

[5] Clement, J. 21 de febrero de 2022. Registered users of Fortnite worldwide from August 2017 to May 2020. Statista. https://www.statista.com/statistics/746230/fortnite-players/ (Visitado el 15-03-2022)

[6] Clement, J. 16 de mayo de 2022. Daily active users (DAU) of Roblox games worldwide from 4th quarter 2018 to 1st quarter 2022. Statista. https://www.statista.com/statistics/1192573/daily-active-users-global-roblox/ (Visitado el 14-06-2022)

[7] Galov, Nick. 06 de abril 2022. What's the Most Popular MOBA - League of Legends Player Count. Web Tribunal. https://webtribunal.net/blog/league-of-legends-player-count/ (Visitado el 14-06-2022)

[8] Anónimo. PUBG Live Player Count - How Many People Are Playing Now?. PlayerCounter. https://playercounter.com/pubg/ (Visitado el 15-03-2022)

[9] Anónimo. 13 de julio de 2022. The 20 most popular Strategy Games for Android ever. AppBrain. https://www.appbrain.com/apps/popular/strategy/ (Visitado el 13-07-2022)

[10] Clement, J. 13 de junio de 2022. Highest monetizing free-to-play (F2P) mobile gaming genres according to F2P developers worldwide in 2020. Statista. https://www.statista.com/statistics/1215450/highest-monetizing-f2p-mobile-game-genres-developer/ (Visitado el 14-06-2022)

[11] Varios Autores. Motor de videojuegos. Wikipedia https://es.wikipedia.org/wiki/Motor_de_videojuego (Visitado el 18-03-2022)

[12] Unreal Engine. (n.d). The most powerful real-time 3D creation tool - Unreal Engine. https://www.unrealengine.com/en-US (Visitado el 25-03-2022)

[13] Unity. (n.d). Plataforma de desarrollo en tiempo real de Unity. https://unity.com/es (Visitado el 25-03-2022)

[14] Toftedahl, Marcus. 30 de septiembre de 2019. Which are the most commonly used Game Engines? GameDeveloper. https://www.gamedeveloper.com/production/which-are-the-most-commonly-used-game-engines- (Visitado el 26-03-2022)

[15] Godot. (n.d). Godot Engine - Free and open source 2D and 3D game engine. https://godotengine.org/ (Visitado el 26-03-2022)

[16] YoYoGames GameMaker. (n.d). Crea fácilmente videojuegos con GameMaker. https://gamemaker.io/es/gamemaker (Visitado el 27-03-2022)

[17] Photon. (n.d). Multiplayer Game Development Made Easy | Photon Engine. https://www.photonengine.com/ (Visitado el 2-04-2022)

[18] Photon. (n.d). Global Cross Platform Realtime Multiplayer Game Framework. https://www.photonengine.com/en-US/Realtime (Visitado el 2-04-2022)

[19] Photon. (n.d). Setting the Benchmark for Multiplayer games | Photon Engine. https://www.photonengine.com/en-US/Fusion (Visitado el 2-04-2022)

[20] Fiedler, Glenn. 30 de noviembre de 2014. Snapshot Interpolation. Gaffer on Games. https://gafferongames.com/post/snapshot_interpolation/ (Visitado el 4-07-2022)

[21] Photon. (n.d). Blazing Fast Deterministic Networking Engine | Photon engine. https://www.photonengine.com/en-US/Quantum (Visitado el 2-04-2022)

[22] Google Firebase. (n.d). Firebase. https://firebase.google.com/ (Visitado el 3-04-2022)

[23] Google Firebase. (n.d). Firebase Realtime Database. https://firebase.google.com/products/realtime-database?hl=es-419 (Visitado el 3-04-2022)

[24] Google Firebase (n.d). Cloud Firestore. https://firebase.google.com/products/firestore?hl=es-419 (Visitado el 3-04-2022)

[25] Google Firebase. (n.d). Choose a Database: Cloud Firestore or Realtime Database. https://firebase.google.com/docs/database/rtdb-vs-firestore (Visitado el 3-04-2022)

[26] Google Firebase. (n.d). Firebase Authentication. https://firebase.google.com/products/auth (Visitado el 3-04-2022)

[27] GameSparks. (n.d). Product. https://www.gamesparks.com/product (Visitado el 3-04-2022)

[28] GameSparks. 19 de febrero de 2019. Pricing. https://www.gamesparks.com/pricing/ (Visitado el 3-04-2022)

[29] Microsoft Azure Playfab. (n.d). Microsoft Azure PlayFab | Full Stack LiveOps, Real-time Control. https://playfab.com/ (Visitado el 4-04-2022)

[30] Microsft Azure PlayFab. (n.d). Servers - PlayFab | Microsoft Docs. https://docs.microsoft.com/en-us/gaming/playfab/features/multiplayer/servers/ (Visitado el 4-04-2022)

[31] Microsoft Azure PlayFab. (n.d). Multiplayer Sevices. https://playfab.com/multiplayer/ (Visitado el 4-04-2022)

[32] Microsoft Azure PlayFab. (n.d). Azure PlayFab Party overview - PlayFab | Microsoft Docs. https://docs.microsoft.com/en-us/gaming/playfab/features/multiplayer/networking/ (Visitado el 4-04-2022)

[33] Microsoft Azure PlayFab. (n.d). Matchmaking - PlayFab | Microsoft Docs. https://docs.microsoft.com/en-us/gaming/playfab/features/multiplayer/matchmaking/ (Visitado el 4-04-2022)

[34] Microsoft Azure PlayFab. (n.d). Pricing. https://playfab.com/pricing/ (Visitado el 4-04-2022)

[35] Unity. (n.d). 1st Party Netcode Solution | Unity. https://unity.com/products/netcode (Visitado el 7-04-2022)

[36] Unity. (n.d). In game Voice and Text Chat with Vivox | Unity Gaming Services. https://unity.com/products/vivox (Visitado el 7-04-2022)

[37] Unity. (n.d). Relay -Reduce your Gaming Networking Cost. https://unity.com/products/relay (Visitado el 7-04-2022)

[38] Unity. (n.d). Lobby - The Best Multiplayer Gamimg Experience | Unity. https://unity.com/products/lobby (Visitado el 7-04-2022)

[39] Unity. (n.d.). Unity - Manual: Scenes. https://docs.unity3d.com/Manual/CreatingScenes.html (Visitado el 21-05-2022)

[40] Unity. (n.d.). Unity - Manual: Cameras. https://docs.unity3d.com/Manual/CamerasOverview.html (Visitado el 21-05-2022)

[41] Unity. (n.d.). Unity- Scripting API: Camera.cullingMask. https://docs.unity3d.com/ScriptReference/Camera-cullingMask.html (Visitado el 21-05-2022)

[42] Unity. (n.d.). Unity - Manual: Layers. https://docs.unity3d.com/Manual/Layers.html (Visitado el 21-05-2022)

[43] Unity. (n.d.). GameObject - Unity Manual. https://docs.unity3d.com/es/2018.4/Manual/class-GameObject.html (Visitado el 21-05-2022)

[44] Unity. (n.d.). Unity- Scripting API: Collider. https://docs.unity3d.com/ScriptReference/Collider.html (Visitado el 21-05-2022)

[45] Unity. (n.d.). Rigidbody 2D - Unity Manual. https://docs.unity3d.com/es/2019.4/Manual/class-Rigidbody2D.html (Visitado el 21-05-2022)

[46] Unity. 2016. Unity Manual: Prefabs. https://docs.unity3d.com/es/530/Manual/Prefabs.html (Visitado el 21-05-2022)

[47] Unity. 25 de abril de 2018. Unity - Manual: Sprites. https://docs.unity3d.com/Manual/Sprites.html (Visitado el 23-05-2022)

[48] Unity. (n.d.). Unity - Manual: Using the Animation View. https://docs.unity3d.com/Manual/animeditor-UsingAnimationEditor.html (Visitado el 23-05-2022)

[49] Unity. (n.d.). Unity - Scripting API: Animator. https://docs.unity3d.com/ScriptReference/Animator.html (Visitado el 23-05-2022)

[50] Unity. (n.d.). Canvas - Unity: Manual. https://docs.unity3d.com/es/2019.4/Manual/UICanvas.html (Visitado el 22-05-2022)

[51] Unity. (n.d.). EventSystem (Sistema de Eventos) - Unity: Manual. https://docs.unity3d.com/es/2018.4/Manual/EventSystem.html (Visitado el 22-05-2022)

[52] Unity. (n.d.). Unity - Scripting API: RectTransform. https://docs.unity3d.com/ScriptReference/RectTransform.html (Visitado el 22-05-2022)

[53] Unity. (n.d.). Unity - Manual: Anchors. https://docs.unity3d.com/560/Documentation/Manual/windowsholographic-anchors.html (Visitado el 23-05-2022)

[54] Unity. (n.d.). Unity - Manual: Font Assets. https://docs.unity3d.com/Manual/class-Font.html (Visitado el 22-05-2022)

[55] Unity. (n.d.). Input Bindings | Package Manager UI Website. https://docs.unity3d.com/Packages/com.unity.inputsystem@0.9/manual/ActionBindings.html (Visitado el 24-05-2022)

[56] Unity. (n.d.). Enum InputActionType | Input system | 1.0.2. https://docs.unity3d.com/Packages/com.unity.inputsystem@1.0/api/UnityEngine.InputSystem.InputActionType.html (Visitado el 24-05-2022)

[57] Unity. (n.d.). Unity - Scripting API: Monobehaviour. https://docs.unity3d.com/ScriptReference/MonoBehaviour.html (Visitado el 25-05-2022)

[58] Unity. (n.d.). Unity - Manual: Coroutines. https://docs.unity3d.com/Manual/Coroutines.html (Visitado el 25-05-2022)

[59] Unity. (n.d.). Unity - Scripting API: WaitForSeconds. https://docs.unity3d.com/ScriptReference/WaitForSeconds.html (Visitado el 25-05-2022)

[60] Google Firebase. 17-06-2021. Firebase.Database Namespace. https://firebase.google.com/docs/reference/unity/namespace/firebase/database (Visitado el 26-05-2022)

[61] Google Firebase. 17-06-2021. Firebase.Database.FirebaseDatabase Class Reference. https://firebase.google.com/docs/reference/unity/class/firebase/database/firebase-database (Visitado el 26-05-2022)

[62] Google Firebase. 17-06-2021. Firebase.Database.DatabaseReference Class Reference. https://firebase.google.com/docs/reference/unity/class/firebase/database/database-reference (Visitado el 26-05-2022)

[63] Google Firebase. 17-06-2021. Firebase.Database.Query Class Reference. https://firebase.google.com/docs/reference/unity/class/firebase/database/query (Visitado el 26-05-2022)

[64] Google Firebase. 17-06-2021. Firebase.Database.ValueChangedEventArgs Class Reference. https://firebase.google.com/docs/reference/unity/class/firebase/database/value-changed-event-args (Visitado el 26-05-2022)

[65] Google Firebase. 17-06-2021. Firebase.Database.ChildChangedEventArgs Class Reference. https://firebase.google.com/docs/reference/unity/class/firebase/database/child-changed-event-args (Visitado el 26-05-2022)

[66] Google Firebase. 17-06-2021. Firebase.Database.DatabaseSnapshot. https://firebase.google.com/docs/reference/unity/class/firebase/database/data-snapshot (Visitado el 26-05-2022)

[67] Tracy Fullerton, Game Design Workshop, Editorial Elsevier, segunda edición, 2008

[68] Varios Autores. (n.d.). Bartles Player Types for Gamification. Interaction DEesign Foundation. https://www.interaction-design.org/literature/article/bartle-s-player-types-for-gamification (Visitado el 14-04-2022)

[69] Google Firebase. 03-06-2022. Agregar Firebase a tu proyecto de Unity | Firebase Documentation. https://firebase.google.com/docs/unity/setup?hl=es-419 (Visitado el 14-06-2022)

[70] Varios Autores. 08-27-2022. Delegados en C#. https://docs.microsoft.com/es-es/dotnet/csharp/programming-guide/delegates/ (Visitado el 13-07-2022)