



**Universidad  
Europea**

**UNIVERSIDAD EUROPEA DE MADRID**

**ESCUELA DE ARQUITECTURA, INGENIERÍA Y DISEÑO**

**GRADO EN INGENIERÍA INFORMÁTICA**

**PROYECTO FIN DE GRADO**

**PISTOLERO VR**

**MARCOS ELADIO SOMOZA CORRAL**

**Dirigido por**

**Dr. ANTONIO BARBA SALVADOR**

**CURSO 2022-2023**

PistoleroVR

Marcos Eladio Somoza Corral

**TÍTULO:** Pistolero VR

**AUTOR:** MARCOS ELADIO SOMOZA CORRAL

**TITULACIÓN:** GRADO EN INGENIERÍA INFORMÁTICA

**DIRECTOR/ES DEL PROYECTO:** Dr. ANTONIO BARBA SALVADOR

**FECHA:** JULIO 2023

PistoleroVR

Marcos Eladio Somoza Corral

# 1. RESUMEN

Este documento relata el trabajo fin de grado realizado. Este consiste en el desarrollo de un pmv de un videojuego VR de género fps. En este trabajo destaca tanto la investigación como el desarrollo de espacios tridimensionales inmersivos, de carácter interactuable que transmiten emociones como ninguna otra plataforma puede. Adicionalmente, se ha investigado e implementado sobre técnicas de inteligencia artificial aplicable a videojuegos, técnicas de optimización y creación de sistemas jugables que favorecen a la experiencia de usuario.

En definitiva, se trata de un producto mínimo viable que responde a la pregunta de cómo desarrollar un videojuego en realidad virtual, demostrando su viabilidad económica, los beneficios que aporta con respecto los videojuegos en plataformas más convencionales y cómo se ha llevado a cabo.

**Palabras clave:** Videojuegos, VR, FPS, optimización, árboles de comportamiento, LiveOps UGS

## 2. ABSTRACT

This document recounts the final undergraduate project carried out. It involves the development of an MVP (Minimum Viable Product) of a first-person shooter (FPS) genre VR video game. This work stands out for both the research and the development of immersive three-dimensional interactive spaces that convey emotions like no other platform can. Additionally, research has been conducted and implemented on artificial intelligence techniques applicable to video games, optimization techniques, and the creation of playable systems that enhance the user experience.

In conclusion, this is a minimum viable product that addresses the question of how to develop a virtual reality video game, demonstrating its economic viability, the benefits it brings compared to games on more conventional platforms, and how it has been carried out.

**Keywords:** Videogames, VR, FPS, optimization, behaviour trees, LiveOps UGS

## AGRADECIMIENTOS

Gracias en primer lugar a mi tutor el Dr. Antonio Barba, por darme absoluta libertad para desarrollar un videojuego como TFG y tanta ayuda al momento de escribir esta memoria.

Gracia mi hermano Bosco, por ser el mayor *playtester* de todos. Gracias a Alex y Alberto por escucharme hablar de los bugs e implementaciones innecesarias que ocurren muy de vez en cuando. Y gracias también a todas las personas que han probado el pmv, así como respondido las encuestas de satisfacción de experiencia de usuario.

### 3. TABLA RESUMEN

	DATOS
Nombre y apellidos:	Marcos Eladio Somoza Corral
Título del proyecto:	PistoleroVR
Directores del proyecto:	Dr. Antonio Barba Salvador
El proyecto se ha realizado en colaboración de una empresa o a petición de una empresa:	NO
El proyecto ha implementado un producto: (esta entrada se puede marcar junto a la siguiente)	SI
El proyecto ha consistido en el desarrollo de una investigación o innovación: (esta entrada se puede marcar junto a la anterior)	SI
Objetivo general del proyecto:	Desarrollo de un videojuego en VR

# Índice

1. RESUMEN .....	4
2. ABSTRACT .....	5
3. TABLA RESUMEN .....	7
Capítulo 1. RESUMEN DEL PROYECTO .....	15
1.1 Contexto y justificación .....	15
1.2 Planteamiento del problema .....	15
1.3 Objetivos del proyecto .....	15
1.4 Resultados obtenidos .....	15
1.5 Estructura de la memoria .....	16
Capítulo 2. ANTECEDENTES / ESTADO DEL ARTE .....	17
2.1 Estado del arte .....	17
2.2 Contexto y justificación .....	22
2.3 Planteamiento del problema .....	23
Capítulo 3. OBJETIVOS .....	24
3.1 Objetivos generales .....	24
3.2 Objetivos específicos .....	24
3.3 Beneficios del proyecto .....	25
Capítulo 4. DISEÑO DEL PROYECTO .....	26
1. Arquitectura de código .....	29
2. SDKs .....	38
3. Unity dashboard .....	40
4. Análisis UML .....	45
Capítulo 5. DESARROLLO DEL PROYECTO .....	66
5.1 Planificación del proyecto .....	66
5.2 Descripción de la solución, metodologías y herramientas empleadas .....	68
5.3 Recursos requeridos .....	70
5.4 Presupuesto .....	71
5.5 Viabilidad .....	72



5.6	Resultados del proyecto .....	74
Capítulo 6.	CONCLUSIONES .....	76
6.1	Conclusiones del trabajo .....	76
6.2	Conclusiones personales .....	76
Capítulo 7.	FUTURAS LÍNEAS DE TRABAJO .....	77
Capítulo 8.	REFERENCIAS .....	78
Capítulo 9.	ANEXOS .....	81
9.1	Glosario .....	81
9.2	Manual de usuario .....	84
9.3	Manual de instalación .....	86
9.4	Resultados de encuestas .....	88

## Índice de Figuras

Figura 1 - Estimación de ventas en la industria del videojuego. Recuperado de: <a href="https://es.weforum.org/agenda/2022/09/el-juego-esta-en-auge-y-se-espera-que-siga-creciendo-este-grafico-le-dice-todo-lo-que-necesita-saber/">https://es.weforum.org/agenda/2022/09/el-juego-esta-en-auge-y-se-espera-que-siga-creciendo-este-grafico-le-dice-todo-lo-que-necesita-saber/</a> .....	17
Figura 2 - Virtual Boy (1995). Recuperado de: <a href="https://nintendo.fandom.com/wiki/List_of_Nintendo_home_consoleshttps://nintendo.fandom.com/wiki/List_of_Nintendo_home_consoles">https://nintendo.fandom.com/wiki/List_of_Nintendo_home_consoleshttps://nintendo.fandom.com/wiki/List_of_Nintendo_home_consoles</a> .....	18
Figura 3 - Sega VR (1993). Recuperado de: <a href="https://www.elmundo.es/tecnologia/videojuegos/2020/12/12/5fc8db1afc6c83123f8b45a7.html">https://www.elmundo.es/tecnologia/videojuegos/2020/12/12/5fc8db1afc6c83123f8b45a7.html</a> .....	18
Figura 4 - Oculus Rift (2010). Recuperado de: <a href="https://www.meta.com/">https://www.meta.com/</a> .....	19
Figura 5- Oculus Quest 2 (2018). Recuperado de: <a href="https://www.meta.com/">https://www.meta.com/</a> .....	19
Figura 6 - Oculus Quest 3(2023). Recuperado de: <a href="https://www.meta.com/">https://www.meta.com/</a> .....	19
Figura 7 - Oculus Quest Pro (2021). Recuperado de: <a href="https://www.meta.com/">https://www.meta.com/</a> .....	19
Figura 8 - Cómo cuadrarían 24 fps en 60 Hz. Recuperado de: <a href="https://www.masgamers.com/los-dichosos-fps-mitos-verdades">https://www.masgamers.com/los-dichosos-fps-mitos-verdades</a> .....	20
Figura 9 - Requerimientos (1/4) de rendimiento de Meta Quest apps. Recuperado de: <a href="https://developer.oculus.com/resources/vrc-quest-performance-1/">https://developer.oculus.com/resources/vrc-quest-performance-1/</a> .....	21
Figura 10 - Zona de aparición en PistoleroVR. Elaboración propia. ....	26
Figura 11 - Zona de tienda de Pistolero. Elaboración propia. ....	27
Figura 12 - Zona de estadísticas de pistoleroVR. Elaboración propia. ....	27
Figura 13 - Zona de juego en solitario de pistoleroVR. Elaboración propia.....	28
Figura 14 - Tiendas de armas. Elaboración propia. ....	28
Figura 15 - Barreras desbloqueables. Elaboración propia. ....	28
Figura 16 - Diagrama de clases de los Managers principales del proyecto. Elaboración propia.29	
Figura 17 - Diagrama de clases del enemigo y sus componentes. Elaboración propia. ....	30
Figura 18 - Diagrama de clases del jugador y sus componentes. Elaboración propia. ....	30
Figura 19 - Diagrama de clases de los ítems de la tienda. Elaboración propia. ....	31
Figura 20 - GOAP definido según los creadores de dicha técnica para videojuegos (usada en el juego F.E.A.R). Recuperado de: <a href="https://slideplayer.com/slide/7726380/">https://slideplayer.com/slide/7726380/</a> .....	32
Figura 21 - Diagrama de clases del árbol de decisión. Elaboración propia.....	33
Figura 22 - Código del árbol de la IA. Elaboración propia. ....	33
Figura 23 - Diagrama del árbol de la IA. Elaboración propia.....	33

Figura 24 - Gráfica de la progresión exponencial de oleadas en PistoleroVR. Elaboración propia.	34
Figura 25 - Gráfica de la progresión exponencial de la vida de los enemigos por oleada en PistoleroVR. Elaboración propia.	35
Figura 26 - Diagrama de clases de los PowerUps (potenciadores) en PistoleroVR. Elaboración propia.	36
Figura 27 – Navmesh y offmesh links creados para representar las zonas caminables por los enemigos en PistoleroVR. Elaboración propia.	37
Figura 28 - Gestor de paquetes de Unity Engine. Elaboración propia. - Gestor de paquetes de Unity Engine. Elaboración propia.	38
Figura 29 - Lista de jugadores guardados en el dashboard de PistoleroVR. Elaboración propia.	40
Figura 30 - Configuración de Economy en PistoleroVR. Elaboración propia.	41
Figura 31 - Pestaña de economy en base de datos de un usuario dado en PistoleroVR. Elaboración propia.	41
Figura 32 - Base de datos de un usuario dado en PistoleroVR. Elaboración propia.	42
Figura 33 - Métodos de subida y bajada de datos en pistoleroVR. Elaboración propia.	43
Figura 34 - Método de subida de datos para recompensas en PistoleroVR. Elaboración propia.	43
Figura 35 - Scripts creados en Cloud Code en PistoleroVR. Elaboración propia.	44
Figura 36 - Script de AddMoneyToPlayer en Cloud Script en PistoleroVR. Elaboración propia.	44
Figura 37 - Script DailyReward en Cloud Code en PistoleroVR. Elaboración propia.	45
Figura 38 - Diagrama de casos de uso del jugador con el mundo. Elaboración propia.	46
Figura 39 - Diagrama de Gantt de PistoleroVR. Elaboración propia.	66
Figura 40 - Kanban del proyecto en Trello. Elaboración propia.	67
Figura 41 - Compilaciones incrementales de PistoleroVR. Elaboración propia.	68
Figura 42 - Visualización en editor del object pooling. Elaboración propia.	70
Figura 43 - Jerarquía en escena del object pooling. Elaboración propia.	70
Figura 44 - Esquema de controles de PistoleroVR. Elaboración propia.	84
Figura 45 - Contador de comienzo de partida en PistoleroVR. Elaboración propia.	85
Figura 46 - Enemigo atacando al jugador en pistolero VR. Elaboración propia.	85
Figura 47 - Aplicación de escritorio sidequest. Elaboración propia.	87
Figura 48 - Pregunta 1 de encuesta de UX. Elaboración propia.	88

Figura 49 - Pregunta 2 de encuesta de UX. Elaboración propia.....	88
Figura 50 - Pregunta 3 de encuesta de UX. Elaboración propia.....	89
Figura 51 - Pregunta 4 de encuesta de UX. Elaboración propia.....	89
Figura 52 - Pregunta 5 de encuesta de UX. Elaboración propia.....	90
Figura 53 - Pregunta 6 de encuesta de UX. Elaboración propia.....	90
Figura 54 - Pregunta 7 de encuesta de UX. Elaboración propia.....	91
Figura 55 - Pregunta 8 de encuesta de UX. Elaboración propia.....	91

## Índice de Tablas

Tabla 1- ACT-01 Jugador. Elaboración propia.	46
Tabla 2 - CU-01. Elaboración propia.	47
Tabla 3 - CU-02. Elaboración propia.	48
Tabla 4 - CU-03. Elaboración propia.	48
Tabla 5 - CU-04. Elaboración propia.	49
Tabla 6 - CU-05. Elaboración propia.	50
Tabla 7 - CU-06. Elaboración propia.	50
Tabla 8 - CU-07. Elaboración propia.	51
Tabla 9 - CU-08. Elaboración propia.	52
Tabla 10 - CU-09. Elaboración propia.	52
Tabla 11 - CU-10. Elaboración propia.	53
Tabla 12 - CU-11. Elaboración propia.	53
Tabla 13 - CU-12. Elaboración propia.	54
Tabla 14 - CU-13. Elaboración propia.	55
Tabla 15 - CU-14. Elaboración propia.	55
Tabla 16 - CU-15. Elaboración propia.	56
Tabla 17 - CU-16. Elaboración propia.	57
Tabla 18 - CU-17. Elaboración propia.	57
Tabla 19 - CU-18. Elaboración propia.	58
Tabla 20 - CU-19. Elaboración propia.	59
Tabla 21 - CU-20. Elaboración propia.	59
Tabla 22 - CU-21. Elaboración propia.	60
Tabla 23 - CU-22. Elaboración propia.	61
Tabla 24 - CU-23. Elaboración propia.	61
Tabla 25 - CU-24. Elaboración propia.	62
Tabla 26 - CU-25. Elaboración propia.	63
Tabla 27 - CU-26. Elaboración propia.	63
Tabla 28 - CU-27. Elaboración propia.	64
Tabla 29 - CU-27. Elaboración propia.	65

PistoleroVR

Marcos Eladio Somoza Corral

## Capítulo 1. RESUMEN DEL PROYECTO

El desarrollo del proyecto se basa en la implementación de diferentes sistemas de programación para lograr un producto mínimo viable de un videojuego en realidad virtual. Dichos sistemas deberán ser lo suficientemente óptimos (así como el resto de los elementos que conforman el videojuego; como los modelos 3d, el sonido, la iluminación, la inteligencia artificial, los efectos visuales...) como para que el videojuego se ejecute a 72 fotogramas por segundo de la forma más constante posible [1]. Ofreciendo a su vez una experiencia de usuario satisfactoria.

### 1.1 Contexto y justificación

Con el avance de las nuevas tecnologías se conoce la creciente popularidad de la realidad virtual, así como la implicación que tienen los videojuegos en ella. Cada vez más usuarios cuentan con el hardware necesario para vivir experiencias inmersivas de realidad virtual, pero la cantidad de videojuegos que cumplen con sus expectativas en el mercado no crece al mismo ritmo ni con la calidad óptima.

Al desarrollar este proyecto, se pretende contribuir al avance de la tecnología en realidad virtual y ofrecer un producto que cumpla con las expectativas que los usuarios traen consigo de sus experiencias con videojuegos en otras plataformas (consolas, pc, móvil...).

### 1.2 Planteamiento del problema

El problema por resolver radica en la falta de videojuegos en realidad virtual que cumplan con las expectativas de los usuarios en términos de calidad y experiencia, a un precio razonable. El ritmo de nuevos lanzamientos es extremadamente lento si se compara con otras plataformas [2]. Aunque la del VR está aumentando y más usuarios tienen acceso al hardware necesario,[3], la oferta de videojuegos no crece al mismo ritmo, habiendo nuevos títulos cada dos años que cumplan con las expectativas de calidad (Beat saber [4], Half-life: Alyx [5], Amongus VR[6] o Gorilla Tag[7]).

### 1.3 Objetivos del proyecto

Desarrollar un producto mínimo viable de un videojuego en VR que cumpla con dichas expectativas ofreciendo una experiencia de usuario adecuada. Para ello, se investigará sobre las mejores técnicas para el desarrollo en VR, desde arquitectura de código, SDK's, APIs o métodos de optimización.

### 1.4 Resultados obtenidos

En cuanto a los resultados se refiere, se ha alcanzado un producto mínimo viable (pmv) de un videojuego VR de género de disparos en primera persona compilado para Oculus Quest 2 [8]. Este hardware desarrollado por la empresa Oculus (posteriormente adquirida por Meta) es el dispositivo líder en el mercado de realidad virtual en la actualidad, con más del 50% de la base de usuarios de realidad virtual [9].

Además de tratarse de un género fps [10], más en concreto se trata de un fps por oleadas donde el jugador deberá sobrevivir a oleadas de enemigos con aspecto de zombis. La dificultad de este tipo de videojuegos, al igual que la de este pmv, es de carácter exponencial, habiendo cada vez más enemigos por oleada y mayor resistencia de los enemigos a los ataques del jugador.

El pmv del videojuego incluye, entre otros sistemas: implementación de SDK's, autenticación de usuarios, guardado de datos en la nube, ejecución de scripts en la nube, sistemas de movimiento e interacción, inteligencia artificial, reacciones hápticas, diferentes tipos de armas, sistema de progresión de dificultad, técnicas de optimización, modelado 3D, animación procedural mediante código, sonido espacial 3D, iluminación y efectos visuales.

En conjunto de todos los sistemas, escenarios y efectos sumado al género seleccionado ofrece al jugador una experiencia inmersiva, llena de emociones intensas [11]. De entre estas emociones, es destacable la del disfrute y el terror, tanto por la satisfacción de sobrevivir a las oleadas como el terror que implican los zombis en un entorno oscuro [12].

## 1.5 Estructura de la memoria

A continuación, se muestra de forma resumida la estructura de esta memoria:

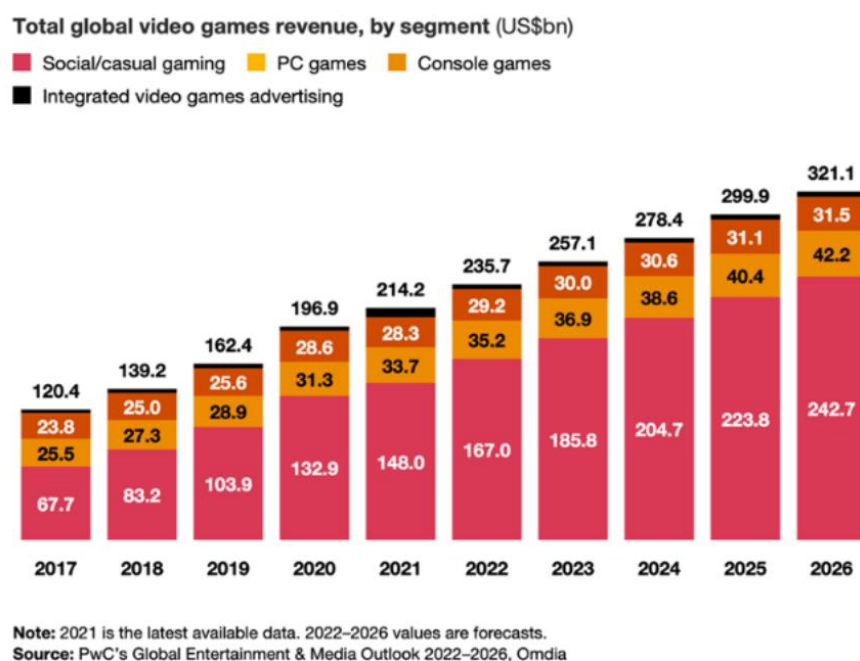
- Estado del arte: Se describe el problema, contexto, justificación y solución propuesta conforme a este proyecto.
- Objetivos: Cada elemento que conforma la solución del proyecto.
- Diseño del proyecto: Diseño del videojuego, arquitectura de código, SDKs, Unity dashboard y análisis UML del proyecto.
- Diseño del proyecto: Herramientas utilizadas, sistemas creados, optimización realizada y funcionalidad del proyecto. Además del análisis de viabilidad y resultados obtenidos.
- Conclusión: Conclusiones tanto del proyecto como personales.
- Futuras líneas de trabajo: En base a la investigación y sesiones de testeo, aquellas tecnologías que se acabarán implementando en el futuro para el proyecto, así como mejoras posibles de cara a su venta
- Referencias: Enumeración mediante IEE de las referencias de esta memoria.
- Anexo: Contiene el glosario de definiciones, el manual de usuario, así como el de instalación y los resultados de las encuestas de experiencia de usuario realizadas sobre el proyecto.



## Capítulo 2. ANTECEDENTES / ESTADO DEL ARTE

### 2.1 Estado del arte

En los últimos años, la sociedad ha sido testigo de un fenómeno en constante crecimiento que ha revolucionado la forma en la que las personas se entretienen y se sumergen en experiencias interactivas: los videojuegos. Con el exponencial avance tecnológico y la masificación de las plataformas digitales, cada vez más personas de todas las edades, géneros y procedencias están en contacto con este mercado. Tanto es así que la industria del videojuego ha experimentado un crecimiento exponencial, convirtiéndose en un sector extremadamente competitivo. Desde el punto de vista digital, se trata de la industria con mayor facturación a nivel mundial, con 148.000 millones de dólares en 2021 de facturación y una estimación de 321.000 millones de dólares para 2026.[13]



*Figura 1 - Estimación de ventas en la industria del videojuego. Recuperado de:  
<https://es.weforum.org/agenda/2022/09/el-juego-esta-en-auge-y-se-espera-que-siga-creciendo-este-grafico-le-dice-todo-lo-que-necesita-saber/>*

Este crecimiento se ha visto reflejado durante la reciente pandemia global del COVID-19, donde según Bartosz Skwarczek, cofundador y director general del mercado de videojuegos online G2A.com afirma que “La gente buscaba formas de entretenerse y mantener sus contactos sociales”. “El videojuego se ha pintado a menudo con un pincel equivocado: se le ha tachado de aislante y poco sociable. Sin embargo, la pandemia ha demostrado que esto no puede estar más lejos de la realidad” [14].

Estos conceptos de videojuegos como herramienta beneficiosa para las personas. Generando emociones positivas y estabilidad emocional, así como funciones de alivio del estrés y relajación. En cuanto al hecho de generar felicidad, los videojuegos son capaces de crear una conexión con los jugadores, quienes desarrollan un compromiso con cada tarea que se les propone. Además, los videojuegos ofrecen una vía de conexión entre diversos jugadores, permitiendo a las personas crear relaciones positivas de forma remota. Finalmente, los videojuegos generan en las personas una sensación de logro, autorrealización y recompensa cuando los jugadores terminan las tareas u objetivos que se les imponen, haciendo así comprender a las personas de la relación esfuerzo-recompensa. [15]

De hecho, se han planteado ideas sobre la implementación de los videojuegos como parte de la educación en las personas, ayudándolas a su desarrollo cognitivo desde temprana edad. Y más recientemente, con el desarrollo de nuevas tecnologías como lo es la realidad virtual, se han discutido formas de crear espacios educativos en entornos virtuales, apoyándose en el nivel de inmersión para la formación educativa. Esta idea se ha visto potenciada en los últimos años debido a la necesidad de las clases a distancia, teniendo que evolucionar a la fuerza las metodologías educativas convencionales a otra más modernas y adaptadas al mundo actual [16].

Aunque estas implementaciones sobre la realidad virtual sean modernas y den imagen de innovadoras, esta tecnología se remonta a la era precursora (1940 a 1968), y se caracterizaba por el inmenso equipo de cómputo utilizado para interactuar con la realidad virtual, sus costos de producción altamente caros y la prioridad militar y privada de los proyectos son otras características consideradas.

Siguiendo a esta era se encuentra la era prototipo (1970 a 1995), caracterizada por el equipo de cómputo más accesible, reduciendo sus costes y facilitando su acceso a instituciones académicas y grandes empresas. Durante este periodo, las grandes compañías de videojuegos comenzaron a desarrollar sus propios HMD diseñados exclusivamente para videojuegos, como Sega VR en 1993[17] o Nintendo Virtual Boy en 1995[18].

Finalmente, y en la era en la que nos encontramos ahora, está la era del consumidor (2010 a la actualidad). Donde los avances tecnológicos han permitido la distribución de hardware especializado al gran público. [19]



Figura 2 - Virtual Boy (1995). Recuperado de: [https://nintendo.fandom.com/wiki/List\\_of\\_Nintendo\\_home\\_consoles](https://nintendo.fandom.com/wiki/List_of_Nintendo_home_consoles)[https://nintendo.fandom.com/wiki/List\\_of\\_Nintendo\\_home\\_consoles](https://nintendo.fandom.com/wiki/List_of_Nintendo_home_consoles)



Figura 3 - Sega VR (1993). Recuperado de: <https://www.elmundo.es/tecnologia/videojuegos/2020/12/12/5fc8db1afc6c83123f8b45a7.html>

Fue en 2010 también cuando surgió el primer prototipo de Oculus Rift, dispositivo que en sus posteriores versiones catapultó el mercado de videojuegos para esta tecnología. Aun así, desde un punto de vista lucrativo, ha resultado más eficiente desarrollar para plataformas más convencionales (consolas, móviles y ordenadores), tanto por haber más cantidad de usuarios como por haber menos limitaciones técnicas del dispositivo durante el desarrollo.



Figura 4 - Oculus Rift (2010). Recuperado de: <https://www.meta.com/>



Figura 5- Oculus Quest 2 (2018). Recuperado de: <https://www.meta.com/>



Figura 6 - Oculus Quest 3(2023). Recuperado de: <https://www.meta.com/>



Figura 7 - Oculus Quest Pro (2021). Recuperado de: <https://www.meta.com/>

Junto con el desarrollo de videojuegos para realidad virtual aparecieron nuevos problemas para los desarrolladores que debían resolver. El déficit de la función visomotora de los jugadores (la capacidad de coordinar la visión con los movimientos del cuerpo) o la estabilidad postural [20](la capacidad de mantener el centro de masa corporal dentro del a base de sustentación) así como la cinetosis (o mareo por movimiento, es un problema común en medios de transporte, especialmente en barcos) eran los mayores problemas que generaban los videojuegos en estas plataformas. [21]

En cuanto a las causas de estas afecciones, se deben a que el aparato vestibular sufre continuas alteraciones que el cerebro no llega a entender; le está llegando una sensación de movimiento continuo pero el cuerpo está parado. Sumado a esto, la fluidez con la que la imagen se actualiza (la tasa de fotogramas por segundo, fps) puede agravar o disminuir las afecciones, si se encuentra por debajo de 60 fps los síntomas tienden a presentarse con mayor facilidad.

Es sabido que el ojo humano no es capaz de distinguir la sucesión imágenes pasado el rango de entre 30 – 60 fps (con carácter general). De hecho, los formatos de cine fijan su tasa de fotogramas por segundo en 24 como estándar, y los canales de televisión rondan los 25 – 30 fps.

Sin embargo, en los videojuegos no se concibe tan poca cantidad de fps. Debido a la gran cantidad de movimiento rápido que ocurre en los videojuegos, se precisan de más fotogramas para tener sensación de fluidez en los movimientos. De ahí que en los videojuegos convencionales tiendan a tasas de fps de entre 30 y 60.

Si bien los fps representan las veces que se actualiza el videojuego por segundo (sus cálculos de físicas, vectores, renderización...), el jugador percibe dichas actualizaciones en una pantalla. Las pantallas digitales funcionan en base a los hercios (Hz). De igual forma que los videojuegos se actualizan una cantidad  $n$  de veces por segundo, medido en fps, las pantallas actualizan la imagen a renderizar una cantidad  $m$  de veces por segundo, medido en Hz. La relación entre los fps y los Hz es de 1:1, lo que implica que, como se ve en la figura 8, si una pantalla se actualiza a 60Hz, por mucho que el videojuego se actualice a 90 fps el jugador sólo podrá ver la renderización de este a 60fps. [22]

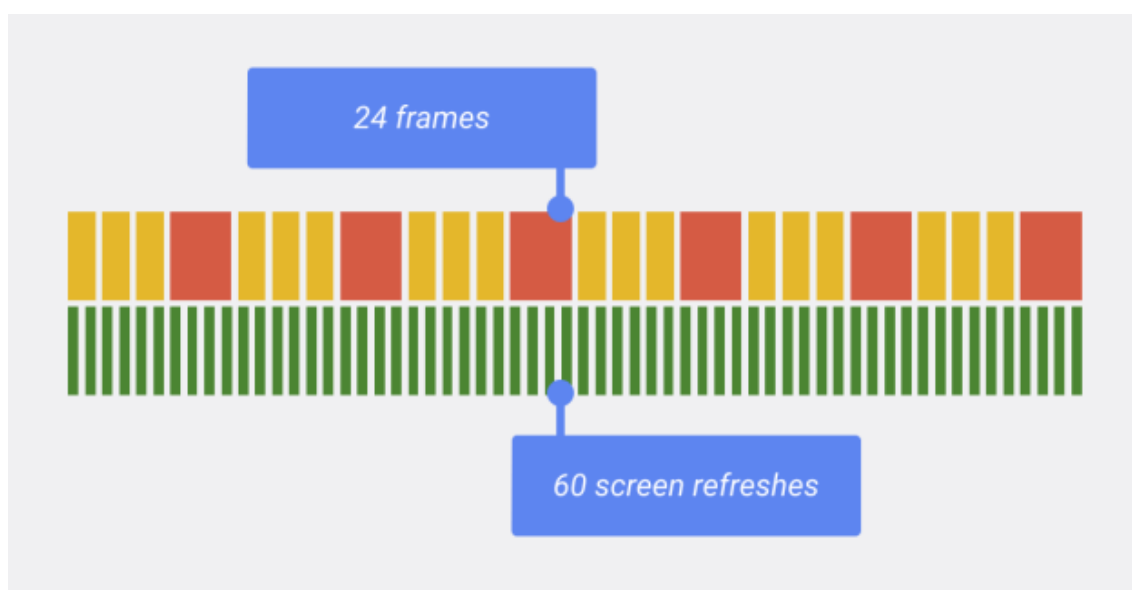


Figura 8 - Cómo cuadrarían 24 fps en 60 Hz. Recuperado de: <https://www.masgamers.com/los-dichosos-fps-mitos-verdades>

Pero estas tasas no son suficientes si se habla de realidad virtual. Al encontrarse en un entorno tan inmersivo y lleno de movimiento, sumado al hecho de que se deben renderizar dos veces todos los elementos (una vez en cada ojo, es decir, una vez en la pantalla izquierda y otra en la derecha), se necesitan más de 60 fps para mantener la percepción de continuidad de los movimientos al ojo humano.

Si bien la tasa de refresco recomendada para VR en general es de 90 fps, muchos HMD no son capaces de mantener esa tasa, debido principalmente especialmente a las limitaciones de su unidad de procesamiento gráfico (GPU) u otras limitaciones de hardware. Para el caso concreto de las Oculus Quest 2, están limitadas a 72 Hz, pudiendo incrementarlo en opciones de desarrollador a 90Hz y 120Hz. Como puede verse en la figura 9, para aplicaciones listas para su venta, oculus obliga a que se ejecuten a un mínimo de 72 Hz, pero para aplicaciones publicadas en su App lab (plataforma de Oculus para poder distribuir el videojuego para sus testeos A/B) un mínimo de 60.[23]

## VRC.Quest.Performance.1

### Criteria

The app must run at a specified refresh rate allowed for Meta Quest apps.

#### VRC platform requirement

	Required ✓	Recommended +
App Lab	✓	
Meta Quest Store	✓	
2D App Lab		+
2D Meta Quest Store		+
For App Lab		For Meta Quest Store
<ul style="list-style-type: none"> <li>Apps must use a refresh of at least 60 Hz on all supported devices.</li> </ul>		<ul style="list-style-type: none"> <li>Interactive applications on all supported devices must use a refresh of 72 Hz, 80 Hz, or 90 Hz<sup>1</sup>.</li> <li>Media applications may use a refresh of 60 Hz on all supported devices.</li> </ul>

<sup>1</sup>Application SpaceWarp (ASW) Exception: Interactive applications may use a rendering rate (FPS) that is half their target refresh rate when utilizing Engine ASW for portions of their experience.

*Figura 9 - Requerimientos (1/4) de rendimiento de Meta Quest apps. Recuperado de: <https://developer.oculus.com/resources/vrc-quest-performance-1/>*

A fin de cuentas, para lograr dichos niveles de rendimiento en videojuegos teniendo en cuenta las limitaciones del hardware, se requiere de técnicas de optimización en todos los aspectos posibles del producto. Sin embargo, estas plataformas de requerimientos de rendimiento no te informan sobre las técnicas necesarias para lograr dicha optimización, por lo que, como nueva empresa o desarrollador que desee crear para estas plataformas, tendrá tiempos de desarrollo mayores, así como costes [24]. También contarán con problemas para lograr un producto de calidad visual, buena experiencia de usuario y nivel de inmersión óptimo, debido a la falta de optimización.

## 2.2 Contexto y justificación

En la actualidad, cada vez más personas cuentan con dispositivos de realidad virtual, teniendo en cuenta la gran variedad de gafas existentes en el mercado y sus rangos de precio. Esta accesibilidad ha derivado en un aumento de usuarios, lo que a su vez deriva en una significativa inyección de capital por parte de grandes compañías como Facebook, Apple, Valve, Sony, HP o HTC, entre otros [25], para mejorar e implementar nuevas funcionalidades y productos dentro de sus gafas.

Teniendo en cuenta este contexto, también grandes desarrolladoras de videojuegos (así como pequeñas) han desarrollado videojuegos para VR, los cuales han recibido tanto buena aceptación como buenos números en ventas. Esto es debido a que los videojuegos en espacios virtuales ofrecen experiencias que los videojuegos convencionales no son capaces de ofrecer:

1. **Inmersión total:** Los juegos en VR son capaces de sumergir al jugador en un entorno virtual, haciendo sentir a éste que está realmente dentro del juego. Esto se logra a través de la visualización en 360 grados y el seguimiento de cabeza y manos, lo que permite mirar y explorar mundos virtuales de forma más natural que los videojuegos en plataformas convencionales.[26]
2. **Interacciones realistas:** En los juegos VR, el jugador puede interactuar con el entorno de una forma más natural y realista. Puede utilizar sus manos, sus gestos y movimientos corporales para manipular tanto objetos como su misma posición. Esto añade un nivel de interacción física que no puede lograrse con los controles convencionales en otras plataformas.[27]
3. **Sensación real de escalado y profundidad:** Los juegos VR ofrecen una sensación de escalado y profundidad tremendamente realista, al hacer uso de la altura real de los ojos del jugador. El usuario puede apreciar las distancias y perspectivas de forma realista, lo que incrementa el nivel de inmersión.[28]
4. **Experiencias sensoriales:** Además de la vista y las interacciones, los juegos VR se aprovechan también de otras interacciones sensoriales como el sonido espacial 3D. Esto genera tanto una experiencia aún más inmersiva como la capacidad de localizar elementos en el espacio sin verlos.[29]
5. **Emociones intensas:** El conjunto de elementos inmersivos que ofrece esta plataforma deriva en una experiencia de usuario tan realista que genera emociones muy intensas a los jugadores. Especialmente, sensaciones de miedo y euforia son las más notorias en el ámbito de los videojuegos VR. [30]

## 2.3 Planteamiento del problema

Sin embargo, con respecto a otras plataformas, la realidad virtual cuenta con una ínfima parte de oferta en videojuegos. Esto es principalmente debido a los problemas presentes a la hora de desarrollar para VR frente a los inexistentes para las otras plataformas. De entre dichos problemas, los más destacables son:

1. **Costes de producción:** Desarrollar para una plataforma específica requiere tener el hardware necesario. Desarrollar un videojuego para entornos de realidad virtual requiere adquirir los HMD donde se vaya a lanzar el producto para poder realizar sesiones de pruebas en ellos, lo que incrementa significativamente los costes de producción frente al desarrollo convencional.
2. **Potencia del HMD:** Aunque el hardware especializado para realidad virtual es cada vez más potente, sigue estando muy por debajo de la potencia de los ordenadores y consolas modernas. Esto implica que no soporta videojuegos pesados, en ningún sentido. No es capaz de soportar técnicas modernas de iluminación en tiempo real, geometrías de gran cantidad de polígonos, texturas de alta definición, efectos visuales complejos, post procesos de cámara y un largo etcétera que resultan en productos poco atractivos visualmente. Lo mismo ocurre desde el punto de vista programático, donde no se pueden realizar operaciones con exceso de complejidad, donde es recomendable no hacer uso de llamadas recurrentes.
3. **Fotogramas por segundo (fps):** Normalmente, los videojuegos en plataformas convencionales (pc, móvil o consolas) se ejecutan a 30 o 60 fps; la cantidad de veces que se actualiza la imagen en pantalla por segundo. Sin embargo, en VR se recomiendan de 72 a 90 fps para no generar problemas al usuario.
4. **Motion sickness:** El mayor problema en videojuegos para VR. Se genera cuando el usuario se mareja jugando al videojuego. Las causas más comunes son la baja tasa de fps (inferior a 60 fps lo puede causar, inferior a 40 fps lo causa de forma segura), los movimientos rápidos de elementos del videojuego (enemigos, escenarios, textos...) o la falta de sincronización entre los movimientos del usuario con el HMD y los mandos y su reflejo virtual en el videojuego (tal como el personaje virtual).

## Capítulo 3. OBJETIVOS

### 3.1 Objetivos generales

El objetivo de este proyecto es el desarrollo de un producto mínimo viable de un videojuego en realidad virtual, con un aspecto visual atractivo y una experiencia de usuario agradable, que cada una de sus partes esté todo lo optimizada posible para lograr la mejor tasa de fotogramas por segundo alcanzable.

Para ello, se deberá realizar una investigación complementaria sobre arquitectura de código legible y escalable, orientada al ahorro de recursos y optimización en entornos de tiempo real, con especial énfasis en realidad virtual, que permita su extrapolación para otros proyectos. Esta investigación se centrará en arquitectura de software, patrones de diseño y su implementación en el producto para que persista a lo largo del tiempo. El producto se desarrollará en un motor de videojuegos; un conjunto de herramientas de software o API creadas para optimizar el desarrollo de un videojuego [31]. Y más concretamente, en Unity Engine con el lenguaje de programación orientado a objetos C#.

### 3.2 Objetivos específicos

Dado que un videojuego es, en esencia, un producto de software que engloba una gran cantidad de elementos desde programáticos hasta artísticos o de diseño, y como se pretende desarrollar un producto mínimo viable, se deberán tener en cuenta todos estos aspectos. Por lo tanto, se han separado los objetivos en cuatro categorías distintivas: Arquitectura de software, realidad virtual, aspectos del videojuego y optimización.

#### Arquitectura de software

- Patrones de diseño creacionales, estructurales y de comportamiento.
- Ejecución de código en local y Cloud mediante APIs.
- Guardado de datos.
- Autenticación de usuarios.
- Test cases.

#### Realidad virtual

- Implementación de SDK's para el desarrollo VR en Unity.
- Implementación de controlador de personaje en VR.
- Detección de manos y controles con el *Input System* de Unity.
- Sonido 3D orientativo para VR.
- Sistema de interacción con el mundo.



#### Aspectos del videojuego

- Sistema de desbloqueables y economía.
- Partículas, shaders y efectos visuales.
- IA con dificultad adaptativa.
- Interfaces de usuario.
- *Environments* tridimensionales inmersivos.

#### Optimización

- Optimización de texturas y geometría.
- Optimización de renderización en tiempo real.
- Optimización en instancias de clases en tiempo real con *object pooling*.
- Optimización en la ejecución del código.
- Otros tipos de optimización.

### 3.3 Beneficios del proyecto

Este proyecto junto con su documentación ofrece una base técnica y teórica de donde partir para poder desarrollar videojuegos optimizados para realidad virtual, ofreciendo unos estándares para dar soluciones a los mayores problemas del sector del videojuego en la realidad virtual. Se agruparán tanto buenas prácticas para el desarrollo de videojuegos en plataformas de VR con carácter general como buenas prácticas específicamente requeridas para este proyecto.

En cuanto al proyecto en sí, además de su aportación como base para futuros desarrollos, ofrecerá a los jugadores un mundo inmersivo e interactivo, donde disfrutar de uno de los géneros de videojuegos más gustados por los jugadores (disparos en primera persona o *first person shooters*, fps). En él, podrán hacer uso de los múltiples sistemas de interacción con el mundo, recibir respuestas cognitivas inmediatas a sus acciones que desencadenarán en sentimientos intensos, creando una experiencia inmersiva placentera de experimentar.

## Capítulo 4. DISEÑO DEL PROYECTO

En cuanto al proyecto se refiere, consta de dos escenas diferenciadas. La primera, donde comienza el jugador. En ésta el jugador aprenderá a controlar al personaje, e interactuar con el entorno. Este primer escenario contiene las siguientes secciones:

**Zona de aparición:** El jugador aparecerá al lado del cubo de la figura 10, donde verá todas las armas disponibles en el juego, lo que incita a que intente cogerlas y aprenda su sistema de interacción. La visión del jugador se alinearán, una vez coja las armas, con las botellas de las maderas del fondo. Estas botellas sirven para que comprendan qué disparo hace cada arma, y puedan practicar con ellas.

Por otro lado, verán un caldero de oro sobre la caja. Si interactúan con él igual que con las armas, recibirán una recompensa diaria (una cantidad de dinero conforme a una fórmula dada. Una vez recogido, comenzará un contador de 24 horas hasta que pueda volver a reclamar una recompensa.



*Figura 10 - Zona de aparición en PistoleroVR. Elaboración propia.*

**Zona de tienda:** El jugador tendrá la posibilidad de gastar sus monedas en tres tipos de sombreros que actúan como cosméticos. También podrá seleccionar cuál equiparse de los que tiene desbloqueados. Para comprar un nuevo cosmético, tendrá que seleccionar el que quiera adquirir y hacer uso del *PhysicsInteractor.cs*, donde deberá golpear con el puño el botón de comprar.

Además, he de mencionar que todos los elementos comprueban en la base de datos si el usuario los ha desbloqueado con anterioridad, cuál tiene actualmente seleccionado y de qué economía dispone. Esta tienda se encuentra en la caseta al fondo izquierda de la figura 10, viéndose por dentro tal que la figura 11.



Figura 11 - Zona de tienda de Pistolero. Elaboración propia.

**Zona de estadísticas:** Esta zona (figura 12) es meramente visual, se irá actualizando en base a las partidas del jugador y albergará la cantidad de partidas que ha jugado, así como la cantidad de bajas totales. He de destacar el uso de guardado de datos en la nube para estas estadísticas, al igual que en la tienda.



Figura 12 - Zona de estadísticas de pistoleroVR. Elaboración propia.

**Zona de juego en solitario:** Finalmente, un botón UI gigante permitirá al jugador acceder a una nueva partida. Aquí aprenderá (el botón de la figura 13), si no lo ha hecho ya con el resto de las interfaces de las que dispone, a interactuar con los elementos de la UI haciendo uso del *PokeInteractor.cs*, donde deberá dar un ligero toque los elementos que desee utilizar.



*Figura 13 - Zona de juego en solitario de pistoleroVR. Elaboración propia.*

La segunda escena es donde se desarrollará la partida. El jugador deberá sobrevivir a rondas recurrentes de enemigos el máximo número de rondas posibles. Esta escena cuenta con cuatro zonas diferentes bloqueadas por barreras, como se ve en la figura 15, las cuales el jugador deberá desbloquear con dinero. El dinero lo ganará disparando a los enemigos que le perseguirán y atacarán constantemente. Además, hay colocada una tienda de armas en cada una de las subzonas, como se ve en la figura 14, donde el usuario podrá comprar tanto el arma como la munición.



*Figura 14 - Tiendas de armas. Elaboración propia.*



*Figura 15 - Barreras desbloqueables. Elaboración propia.*

## 1. Arquitectura de código

Se han alcanzado los objetivos de implementar patrones de diseño en el proyecto.

**Patrón Singleton:** para el *GameManager.cs*, el *AudioManager.cs*, el *CloudCodeManager.cs* y el *EconomyManager.cs* mediante su instancia estática. Gracias a este patrón de diseño, el resto de las clases del proyecto pueden acceder a las propiedades y métodos públicos de estos *Singletons* sin necesidad de inyectarlos como dependencias.

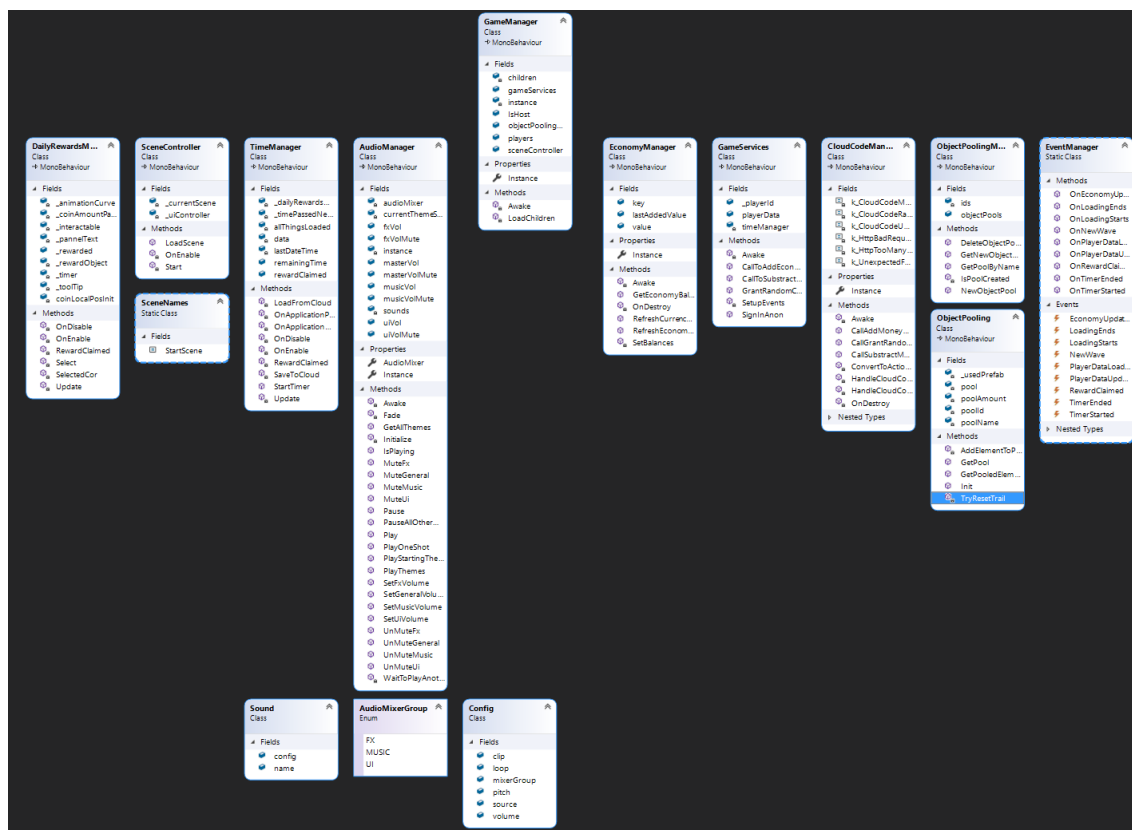


Figura 16 - Diagrama de clases de los Managers principales del proyecto. Elaboración propia.

Además de estos scripts con patrón *singleton*, como se puede ver en la figura 16, existen muchos otros controladores principales necesarios para el proyecto pero que no precisan de una instancia estática. Esto es debido a que su uso es requerido de forma más aislada.

**Patrón Observer:** Sin embargo, existe una clase estática de esta figura 16 llamada *EventsManager.cs*, que hace uso del patrón *observer*. Se trata de una clase estática ajena a Unity (es decir, que no hereda de *Monobehaviour*) y es encargada de la gestión de eventos en el código. Permite que cuando una acción se ejecute, haya otros objetos que estén “escuchando” y actúen en consecuencia mediante el uso de *delegates* y *events* nativos del lenguaje C#.



**Patrón Component:** Lo cierto es que el motor de videojuegos Unity, así como muchos otros, están basados en la arquitectura ECS (*entity component system*), donde se dividen las funcionalidades por componentes. Esta filosofía se ha extendido también para el desarrollo de diversas funcionalidades del proyecto, como para el jugador o los enemigos (figuras 17 y 18).

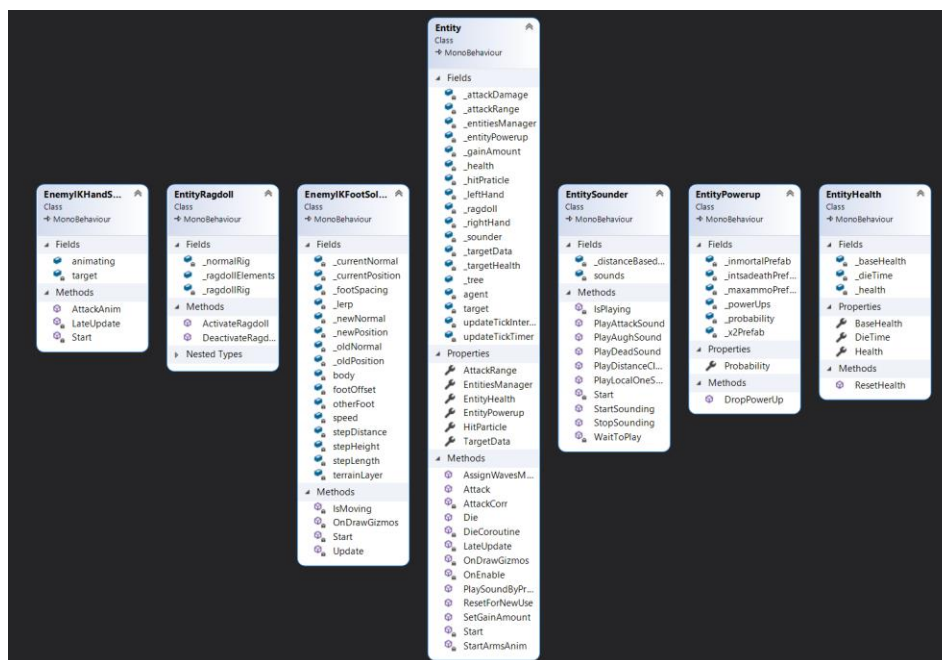


Figura 17 - Diagrama de clases del enemigo y sus componentes. Elaboración propia.

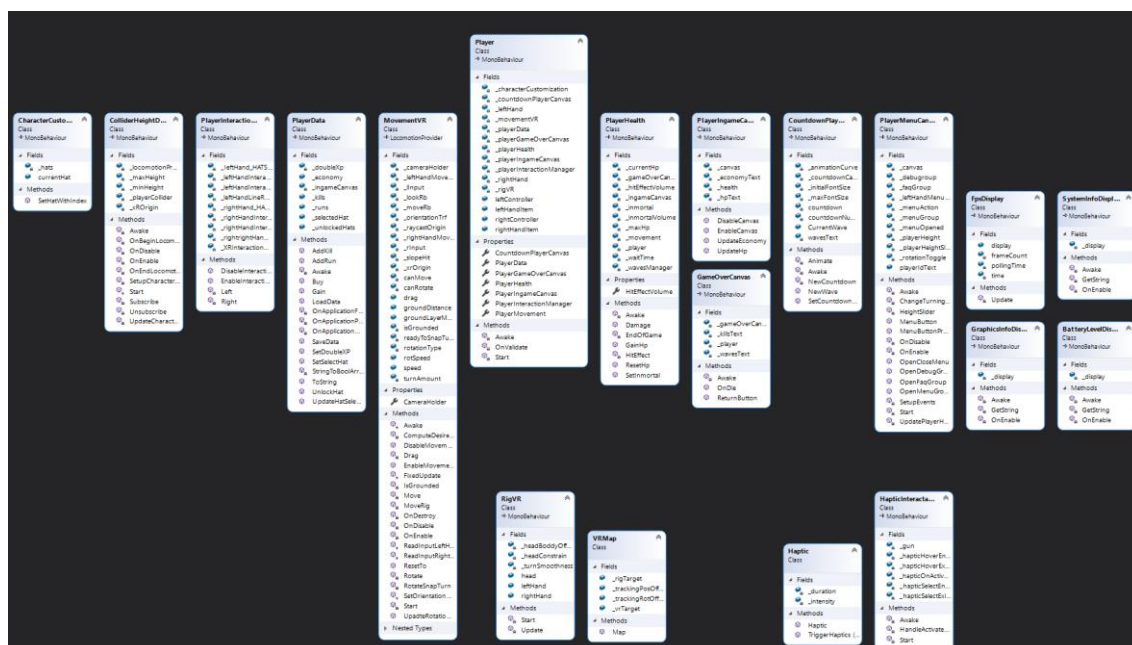


Figura 18 - Diagrama de clases del jugador y sus componentes. Elaboración propia.

**Patrón Scriptable Object:** Este patrón es propietario de Unity, ofrece una forma de serializar data de instancias de scripts. Permite crear diseños modulares e iterativos, es por ello por lo que se ha utilizado para venta de las distintas armas del juego en las tiendas.

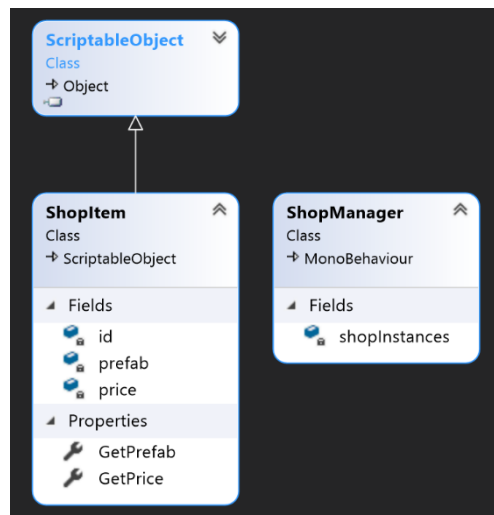


Figura 19 - Diagrama de clases de los ítems de la tienda. Elaboración propia.

De esta forma, como se ve en la figura 19 con un único objeto *ShopItem* se pueden crear ítems de venta en la tienda de todas las armas del juego actuales y futuras, dado que aloja un id, el precio y el *prefab* del arma (un *prefab* actúa como una plantilla de objeto que puede ser instanciado numerosas veces en una escena, en este caso, varía dependiendo del arma del que se trate).

Además de estos patrones de diseño, es inevitable mencionar el uso de otras técnicas propias de la programación orientada a objetos, siguiendo siempre con los principios S.O.L.I.D [32] (*Single responsibility*, *Open-closed*, *Liskov substitution*, *Interface segregation* y *Dependency inversion*) como:

- Encapsulación de variables y métodos en todas las clases del código necesarias, limitando su acceso y mejorando tanto la protección como la legibilidad del código.
- Herencia para diferentes sistemas que lo requerían, como los diferentes tipos de armas, los diferentes elementos de detección de distancias para con el jugador, los diferentes tipos de elementos que pueden recibir daño del jugador o los tipos de nodos utilizados para la IA de los enemigos.
- Polimorfismo para aquellos métodos que lo requiriesen, así como el uso de la abstracción para todos los scripts.

Por otro lado, en cuanto a la inteligencia artificial de los enemigos se refiere, se han estudiado diferentes diseños para implementar el más conveniente para el proyecto, destacando

máquinas de estados [33], GOAP (*Goal Oriented Action Planning*) [34], Árboles de decisión [35] y redes neuronales [36]. De entre todos ellos, se ha optado por implementar los árboles de decisión.

Esto se debe a que el uso de máquinas de estados para comportamientos relativamente complejos (más allá de dos o tres estados) es poco escalable, dado que se deben definir tanto cada estado como su transición con el resto.

Por otro lado, de igual forma que las máquinas de estados son insuficientes para este propósito, tanto el uso de GOAP como de redes neuronales son excesivas. Es por eso por lo que se ha decidido implementar árboles de decisión, un punto medio entre las máquinas de estados y GOAP (la cual, entre otras cuestiones, interpreta los nodos como elementos de un grafo).

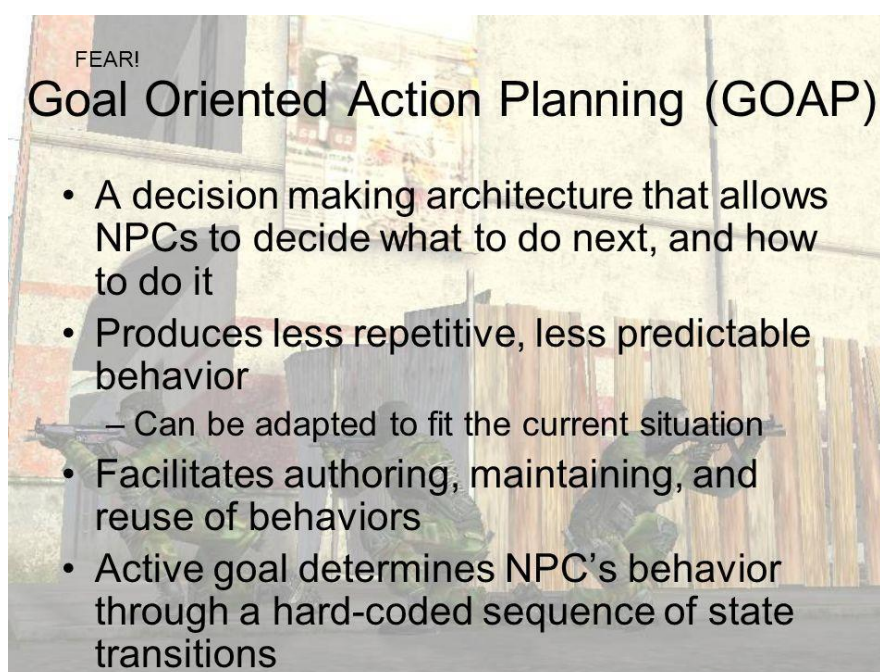


Figura 20 - GOAP definido según los creadores de dicha técnica para videojuegos (usada en el juego F.E.A.R).  
Recuperado de: <https://slideplayer.com/slide/7726380/>

Volviendo a los árboles de decisión, estos están compuestos por diferentes nodos, y es el agente quien, en base a la información que recibe del entorno, elige cuál de las ramas del árbol es la que contiene la acción (el nodo) más adecuada, como se ve en la figura 21. Cada nodo del árbol proviene de su clase padre *Node.cs*, y contiene entre otras cosas un valor de su estado en forma de *enum* (ejecutando, fallo, acierto).

En base a dicha clase padre nodo se crean dos nodos “condicionales” para formar la lógica de las ramas del árbol: Nodos selectores (OR) y nodos de secuencia (AND) (*SelectorNode.cs* y *SequenceNode.cs* respectivamente). El primero es encargado de bifurcar el árbol en *n* subramas hijas suya, donde el agente podrá elegir de entre las opciones disponibles de ese nodo. El segundo es encargado de crear una secuencia de nodos cuyas condiciones deberán ser de tipo



acierto para poder realizar la acción final de la secuencia. Estos nodos se formarán en la clase *Tree.cs* como listas de nodos, creando así un árbol de decisión.

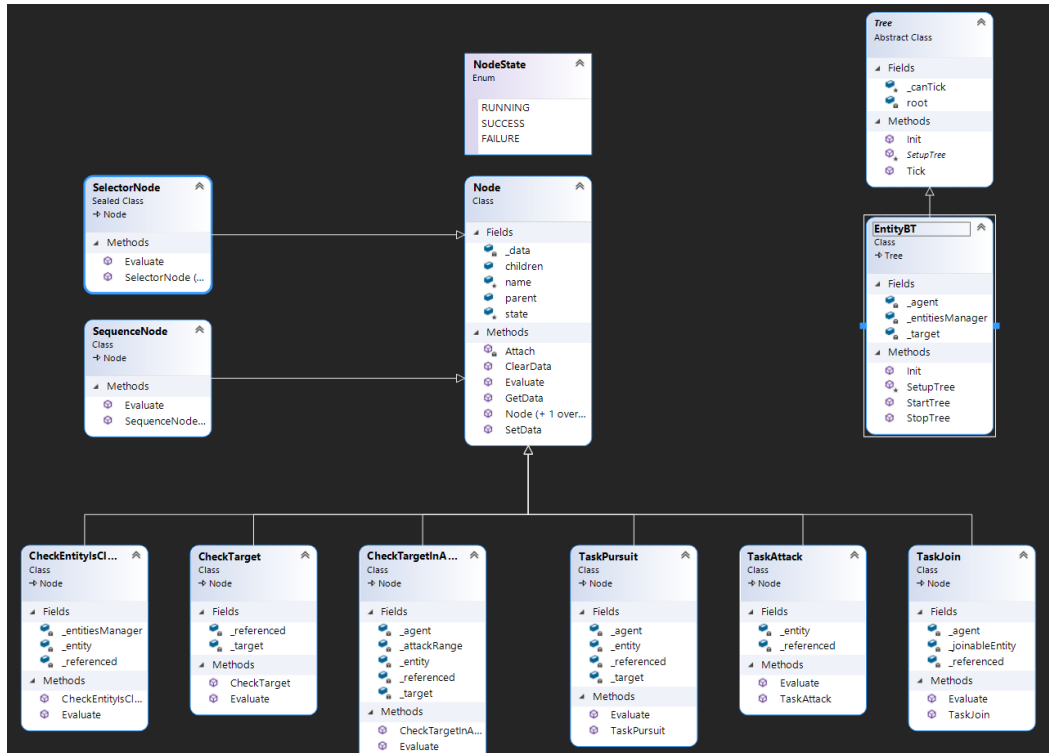


Figura 21 - Diagrama de clases del árbol de decisión. Elaboración propia

Para el diseño de la inteligencia de los enemigos, se plantean tres ramas distintas con prioridades sucesivas. La primera rama será la que el agente primará en realizar antes que comprobar si las otras son viables. Así pues, los enemigos cuentan con una estructura de árbol como se aprecia en las siguientes figuras 22 y 23.

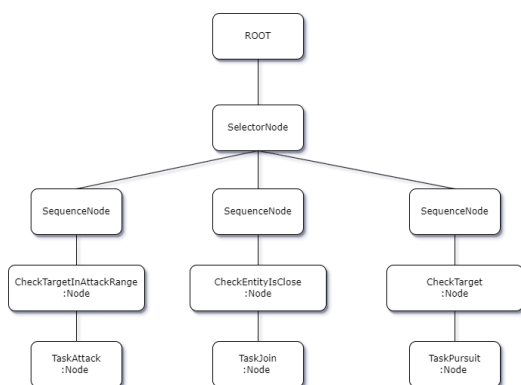


Figura 23 - Diagrama del árbol de la IA. Elaboración propia.

```
new List<Node>
{
    new SequenceNode("AttackSequence", new List<Node>
    {
        new CheckTargetInAttackRange(),
        new TaskAttack()
    }),
    new SequenceNode("JoinSequence", new List<Node>
    {
        new CheckEntityIsClose(),
        new TaskJoin()
    }),
    new SequenceNode("PursuitSequence", new List<Node>
    {
        new CheckTarget(),
        new TaskPursuit()
    })
};
```

Figura 22 - Código del árbol de la IA. Elaboración propia.

Además del nodo secuencia y el nodo selector, se han creado varios tipos de nodo adicionales para gestionar cada rama del agente. Para identificar estos tipos en dos categorías, se ha utilizado la nomenclatura donde la acción final del agente se llamará *task* (tarea en castellano) y será responsable del acto en sí mismo (atacar al jugador, mover el agente hacia otro agente o perseguir al jugador). Por otro lado, se encuentran aquellos llamados *check* (comprobación en castellano) y se encargarán de las comprobaciones previas a que el agente pueda realizar la acción.

Continuando con el diseño del proyecto y de la mano del comportamiento de la IA se encuentra la gestión de oleadas. A fin de cuentas, el *gameplay* de este videojuego se centra en las oleadas. El diseño de este sistema se basa en el número de enemigos por oleada, así como su vida incremental.

Para crear esta progresión, se ha utilizado la fórmula  $f(x) = x \cdot 1,231 + 2$ , creando así una progresión exponencial, donde  $x$  es el número de oleadas  $e$  y el número de enemigos, como se aprecia en la figura 24.

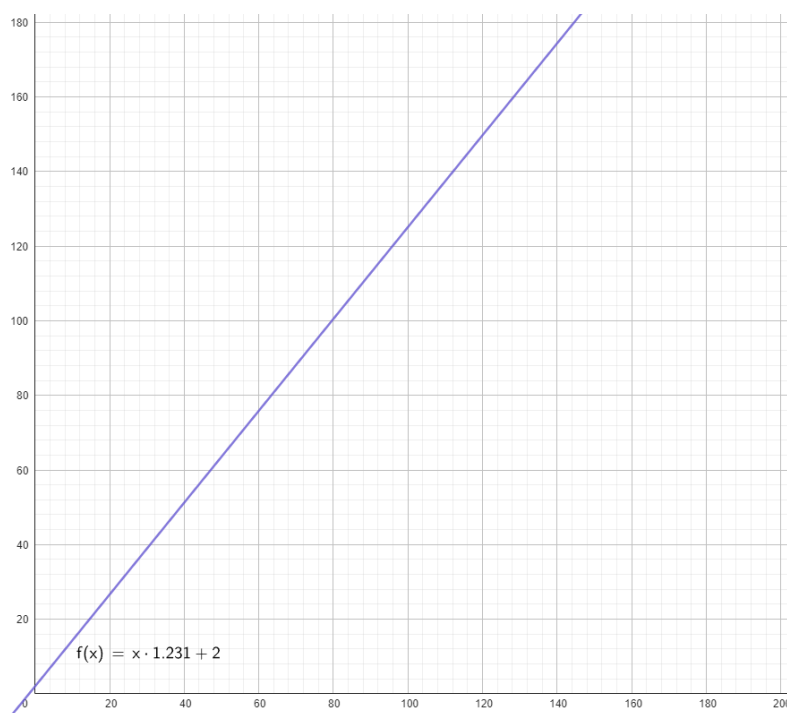
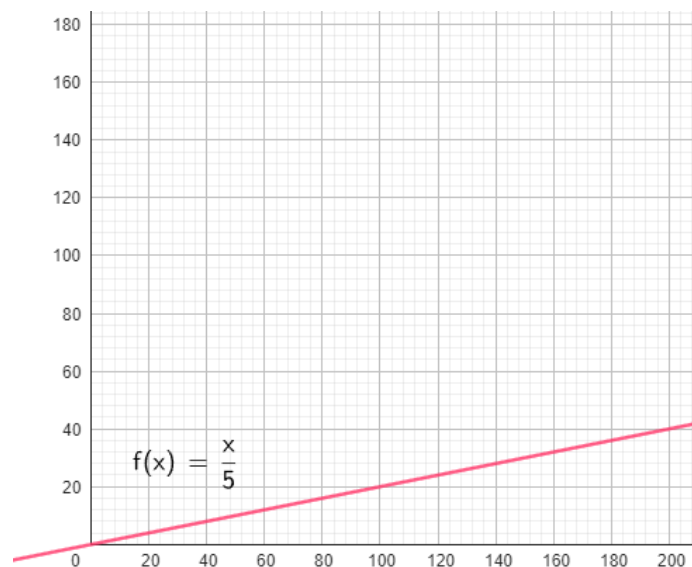


Figura 24 - Gráfica de la progresión exponencial de oleadas en PistoleroVR. Elaboración propia.

De esta forma, como creadores del videojuego se asegura que, aunque el jugador tenga un margen teóricamente infinito de oleadas jugables, llegará un momento que no sea capaz de

sobrevivir a tantos enemigos. Este hecho ayuda a que el jugador sienta una buena experiencia de usuario a la vez que no se aburra de jugar por un nivel de dificultad pobre.

Como se ha mencionado, la vida de cada enemigo también aumenta de forma progresiva, siguiendo la fórmula  $f(x) = x / 2$  (a diferencia del daño que hace el jugador con sus disparos). De esta forma, la dificultad progresiva de las partidas se verá también influenciada por la cantidad de disparos que necesitarán los enemigos para ser abatidos, como se aprecia en la figura 25.



*Figura 25 - Gráfica de la progresión exponencial de la vida de los enemigos por oleada en PistoleroVR. Elaboración propia.*

Debido a que los enemigos y las oleadas se vuelven más difíciles de sobrellevar a lo largo del tiempo por esta dificultad exponencial, sumado al hecho de que el jugador no cuenta con una subida de daño en sus ataques, la rampa de dificultad escala demasiado rápidamente para ofrecer una experiencia de juego satisfactoria. Es por eso por lo que se ha implementado un sistema adicional de *powerups* (potenciadores). El sistema en cuestión se basa en diferentes objetos que el jugador puede recoger para recibir una bonificación de las que hay disponibles.

Existen un total de cuatro potenciadores distintos, cada uno con unos efectos únicos. Está el potenciador Inmortal, que da al jugador un escudo de 10 segundos donde ningún enemigo puede alcanzarle. El potenciador de doble puntuación, donde durante 10 segundos el jugador recibe el doble de dinero por disparo dado al enemigo. El de munición máxima, que recarga toda la munición de las armas que posea el jugador en el momento de cogerlo. Finalmente, el de muerte instantánea, que derriba a todos los enemigos presentes en el escenario en el momento de recoger el potenciador.

Cabe destacar que éstos potenciadores provienen de una clase padre *PowerUp* (véase la figura 26), que a su vez proviene de otra clase *PlayerRangeDetection*. Esta última es utilizada también para las barreras desbloqueables del escenario y las tiendas de armas, pues indica mediante eventos cuándo el jugador entra o sale de la zona de detección.

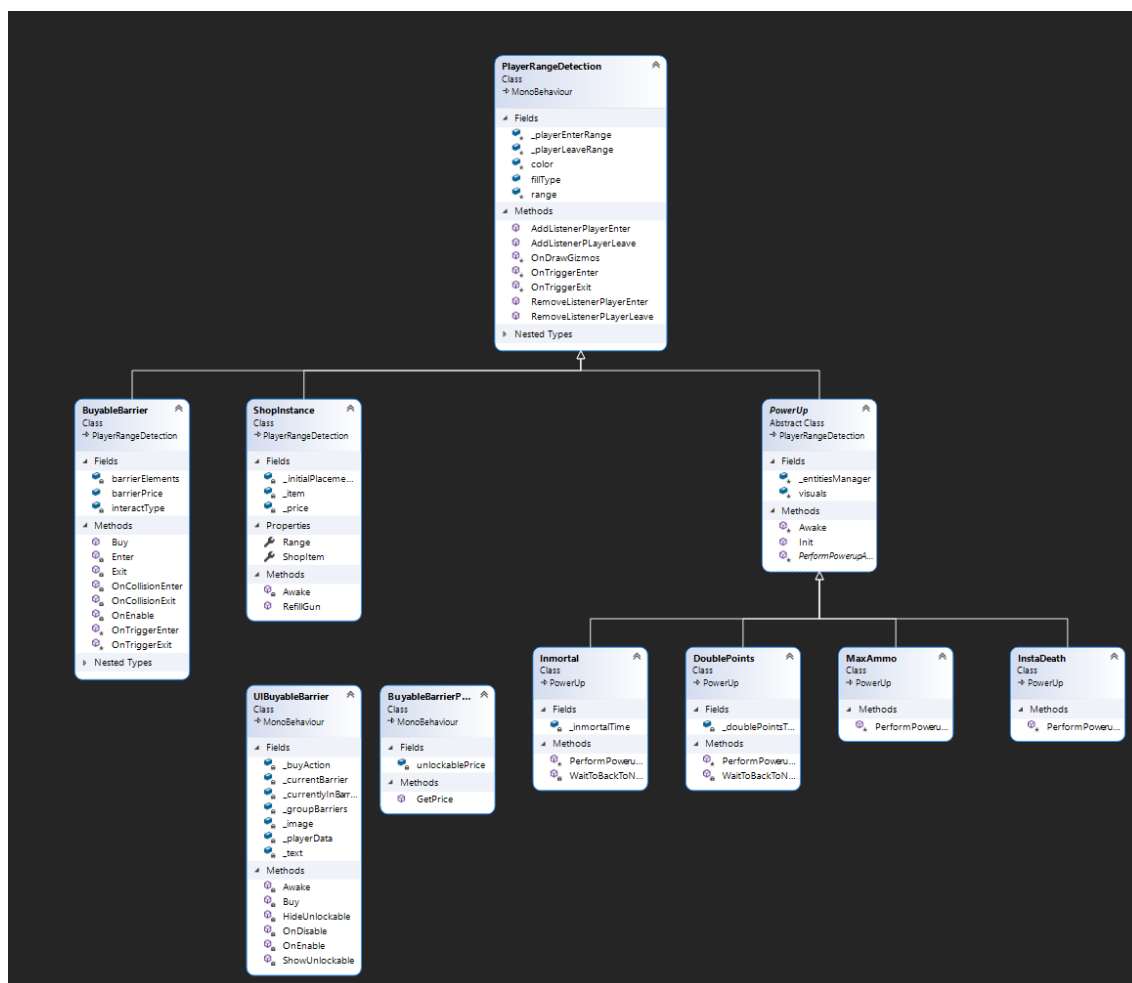


Figura 26 - Diagrama de clases de los PowerUps (potenciadores) en PistoleroVR. Elaboración propia.

Para finalizar con el apartado de los enemigos, se debe analizar qué método utilizar para el cálculo de caminos. A fin de cuentas, los enemigos estarán buscando el camino más óptimo por donde moverse para unirse a otros enemigos o perseguir al jugador.

Existen algoritmos ampliamente utilizados en la industria de los videojuegos para lograr este propósito, tal como A\*, Dijkstra o Floyd-Warshall[37], siendo el primero el más utilizado. Tanto es así, que Unity Engine ofrece un sdk propietario de navegación de IA completo y de rápida implementación, que por debajo utiliza el algoritmo A\*.

Por ende, las búsquedas de caminos de los enemigos se realizarán mediante este paquete ofrecido por Unity, donde, a grandes rasgos, permite crear un mapa caminable en base a una geometría dada. Así, una vez esté finalizado el diseño del escenario de juego, se creará una malla 3D que represente los espacios caminables por los enemigos, y se utilizará para dicho propósito.

A estas geometrías caminables por la IA se las conoce en la industria como *navmesh*, cuya traducción literal es malla navegable, a la vez que a los agentes que caminan por ella se les

conoce como *navmesh agents*. Los obstáculos dinámicos que se encuentren en el *navmesh* como *navmesh obstacles* y aquellos puntos de unión entre dos *navmesh* donde el *navmesh agent* pueda navegar, se les conoce como *offmesh links*. Se puede ver en la figura 27 la *navmesh* creada con sus *offmesh links*.

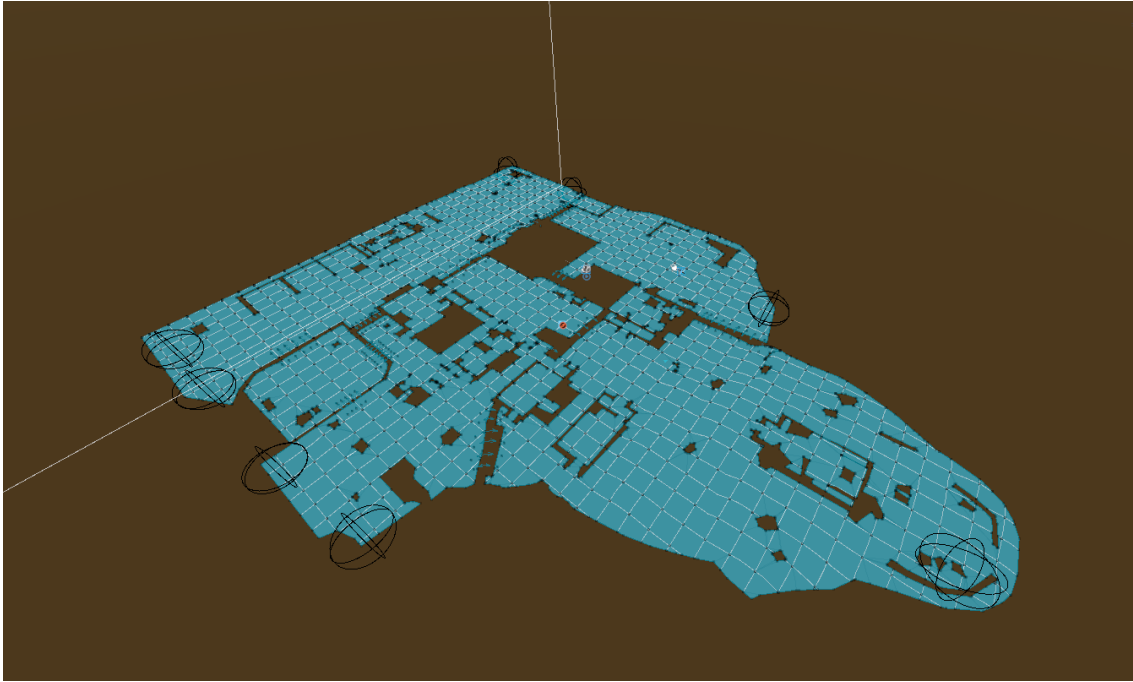


Figura 27 – Navmesh y offmesh links creados para representar las zonas caminables por los enemigos en PistoleroVR. Elaboración propia.

## 2. SDKs

Todos los sdks implementados en el proyecto están preparados para su correcta instalación en *Unity Engine*, por lo que todos se instalan de la misma forma: desde el gestor de paquetes de *Unity*, apreciable en la figura 28.

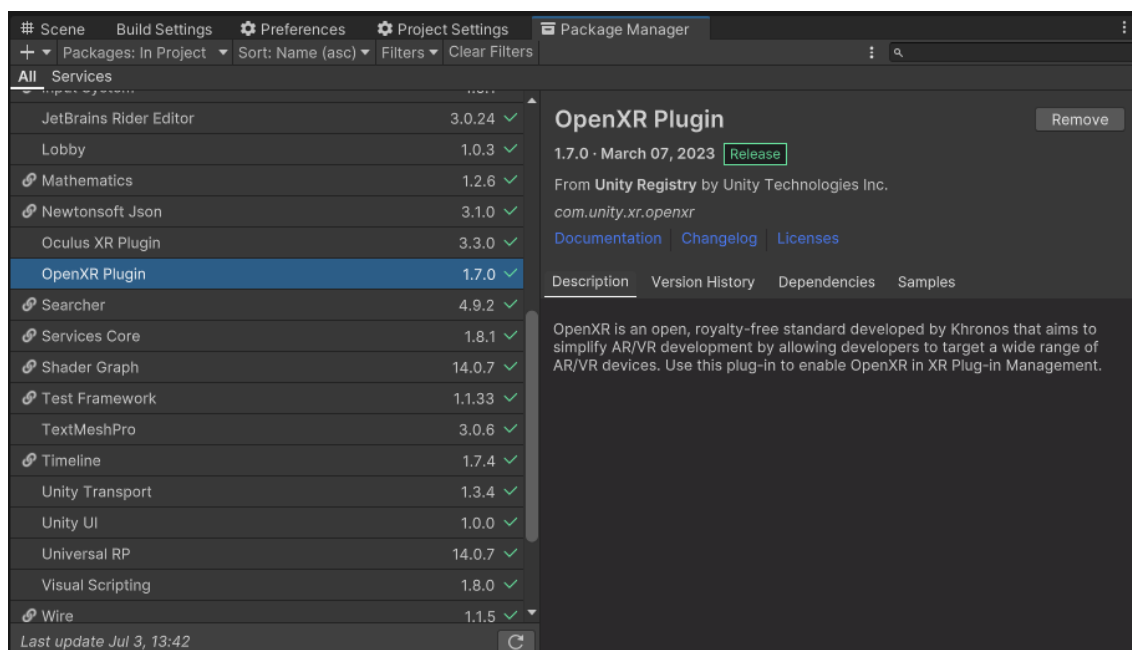


Figura 28 - Gestor de paquetes de Unity Engine. Elaboración propia. - Gestor de paquetes de Unity Engine. Elaboración propia.

Dentro de Unity, se han importado diferentes paquetes necesarios para el desarrollo:

- **XR Interaction Toolkit:** Un sistema de interacción de alto nivel basado en componentes para crear experiencias de realidad virtual (VR) y realidad aumentada (AR). Proporciona un marco de trabajo que hace que las interacciones en 3D y la interfaz de usuario estén disponibles a partir de eventos de entrada de Unity. El núcleo de este sistema es un conjunto de componentes base, y un administrador de Interacción que vincula estos componentes. También contiene componentes que se pueden utilizar para la locomoción y la representación visual.
- **Universal Render Pipeline:** proporciona flujos de trabajo amigables para los artistas que te permiten crear de manera rápida y sencilla gráficos optimizados en una variedad de plataformas, desde dispositivos móviles hasta consolas de alta gama y PC.
- **TextMesh Pro:** Utiliza técnicas avanzadas de representación de textos junto con un conjunto de shaders personalizados, lo que proporciona importantes mejoras en la calidad visual y ofrece a los usuarios una increíble flexibilidad en cuanto a estilos y texturas de texto.

- **Economy:** Servicio en la nube que permite a los desarrolladores de juegos crear la economía dentro de sus juegos y les brinda la posibilidad de realizar compras sin problemas, convertir divisas, gestionar el inventario de los jugadores y más.
- **Cloud Save:** Permite almacenar datos de forma segura y sin interrupciones en la nube.
- **Cloud Code:** Permite ejecutar lógica de juego en la nube como funciones sin servidor e interactuar con otros servicios de backend.
- **Authentication:** Ofrece gestión de identidad de jugador para los Servicios de Juegos de Unity. Cuenta con diferentes proveedores de identidad que pueden implementarse. La gestión de usuarios está disponible en el Panel de control de Unity (Unity dashboard).
- **Animation Rigging:** El Animation Rigging Toolkit permite crear y controlar sistemas de rigging avanzados en Unity utilizando jobs de animación de C#. Esto ofrece un rendimiento optimizado y una mayor flexibilidad en la manipulación de animaciones.
- **AI Navigation:** Proporcionan herramientas para construir y utilizar mallas de navegación en tiempo de ejecución y en tiempo de edición. Estos componentes permiten generar y configurar estas mallas para permitir que los objetos en el juego naveguen de manera inteligente por ellas. Estas mallas de navegación se resumen en *navmesh*, *navmesh agents*, *navmesh obstacles* y *offmesh links*. Para sus cálculos de caminos implementan el algoritmo de búsqueda de caminos A\*.
- **Input System:** Proporciona una forma más flexible de manejar la entrada del usuario en los juegos de Unity. Permite asignar acciones a diferentes eventos de entrada, facilitando la detección de inputs para multitud de plataformas.
- **OpenXR:** Para desarrollar en Unity para una plataforma de VR, es esencial implementar los SDKs adecuados de VR. Dependiendo de la plataforma específica se recomienda el uso de unos u otros (Oculus SDK para desarrollar sólo en Oculus, *OpenVR* SDK para desarrollar para Valve Index o Vive SDK para HTC Vive), sin embargo, Unity ofrece un SDK con compatibilidad múltiple: OpenXR. Se ha decidido trabajar con ese por la flexibilidad que ofrece por su soporte multiplataforma, así como por la gran cantidad de documentación oficial disponible. Es un estándar abierto y libre de derechos de autor desarrollado por Khronos que tiene como objetivo simplificar el desarrollo de realidad aumentada (RA) y realidad virtual (RV) al permitir que los desarrolladores se dirijan a una amplia gama de dispositivos RA/RV.  
Entre otros beneficios, este SDK ofrece una base para sistemas de interacción y locomoción, así como seguimiento de posición y rotación del hardware de las gafas y mandos a posiciones y rotaciones dentro del videojuego.

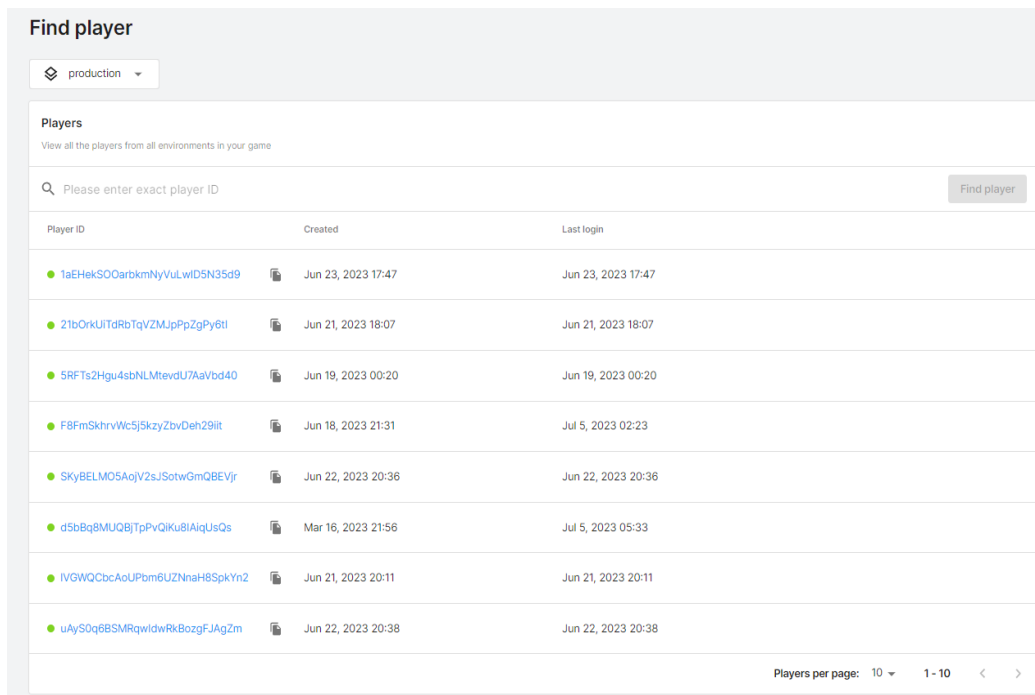
### 3. Unity dashboard

Sin embargo, se han utilizado más SDKS (paquetes de unity) además de OpenXR. De entre ellos, son esenciales los paquetes de autenticación, economía, código en la nube y guardado de datos en la nube (*Authentication, Economy, Cloud Code* y *Cloud Save*, respectivamente).

Unity facilita la visualización de la implementación de dichos paquetes de forma amigable, mediante su portal web de panel de control. A través de éste, se puede tanto analizar la cantidad de llamadas, conexiones, carga y descarga de datos como visualizar las bases de datos, editar los scripts en la nube o gestionar los diferentes usuarios del proyecto. Cabe destacar que el grupo de *Unity Game Services* (aquel que gestiona *Unity Dashboard*) ofrece muchos más servicios (multijugador, *analytics*, *devOps*, crecimiento o monetización) además de los utilizados en este proyecto.

Comenzando por describir la autenticación, se ha utilizado la llamada *AnonymousSignIn* nativo de este sdk de unity, que como se ve en la figura 29, crea id únicos para cada usuario. Ofrece un id único para cada usuario sin que éste tenga que iniciar sesión de ningún tipo, y se enlaza a la instancia del proyecto (por lo que, si el usuario desinstala la aplicación, se perderá la referencia a este id).

Para el desarrollo del pmv, es suficiente para poder gestionar las cuentas de los usuarios, sin embargo, se deberá implementar en un futuro alguno de los SDKs compatibles ofrecidos por Unity, tales como Google, Google Play Games, Apple Game Center, Facebook, Steam, Oculus y OpenID Connect [38]. Esto permitirá a los usuarios instalar y desinstalar el videojuego, así como jugarlo en otro dispositivo que no sea el suyo, sin temor a perder sus avances.



**Find player**

production

**Players**  
View all the players from all environments in your game

Search: Please enter exact player ID Find player

Player ID	Created	Last login
1aEHeKSOOarbkmNyVuLwID5N35d9	Jun 23, 2023 17:47	Jun 23, 2023 17:47
21bOrkUITdRbTqVZMJpPpZgPy8tl	Jun 21, 2023 18:07	Jun 21, 2023 18:07
5RFTs2Hgu4sbNLmtevdU7AaVbd40	Jun 19, 2023 00:20	Jun 19, 2023 00:20
F8FmSkhrvWc5j5kzyZbvDeh29lit	Jun 18, 2023 21:31	Jul 5, 2023 02:23
SKyBELMO5AojV2sJSoTwmGmQBEVjr	Jun 22, 2023 20:36	Jun 22, 2023 20:36
d5bBq8MUQBjTpPvQiku8IAiqUsQs	Mar 16, 2023 21:56	Jul 5, 2023 05:33
IVGWQCbcAoUPbm6UZnNah8SpkYn2	Jun 21, 2023 20:11	Jun 21, 2023 20:11
uAy5QqBBSMRqwidwRkBozgFJAgZm	Jun 22, 2023 20:38	Jun 22, 2023 20:38

Players per page: 10 1 - 10

Figura 29 - Lista de jugadores guardados en el dashboard de PistoleroVR. Elaboración propia.



En cuanto al paquete *Economy* (figura 30), se utiliza para asegurar la cantidad de dinero que cada jugador tiene de forma remota. Para ello, *Economy* permite crear diferentes tipos de monedas para los proyectos (muchos videojuegos hacen uso de varios tipos de monedas para diferentes cosas, como gemas, monedas, diamantes...).

Para este proyecto, se ha creado una única moneda, llamada MONEY, que se utilizará para desbloquear cosméticos. El valor de esta moneda en el cliente se lee de forma diferente a la carga de datos, pues es más variable y siempre comprobada en la nube.

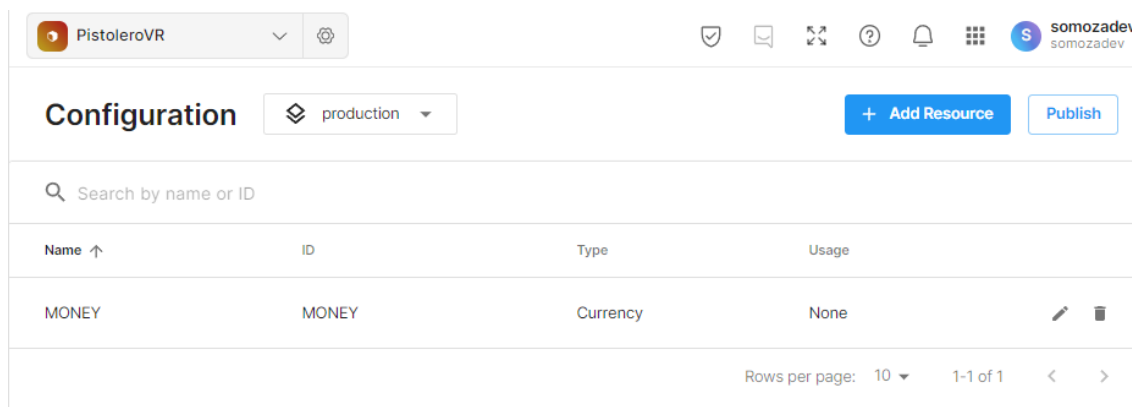


Figura 30 - Configuración de Economy en PistoleroVR. Elaboración propia.

Por otro lado, se ha hecho uso del paquete Cloud Save a modo de bases de datos no relacionales remota. Esta base de datos divide de forma automática los datos a guardar por usuarios. Es decir, crea una “instancia de base de datos” para cada jugador. Además, si existe una configuración previa de *Economy* con una moneda creada (figura 31), une la cantidad que tenga el jugador de dicha moneda también en la base de datos automáticamente.

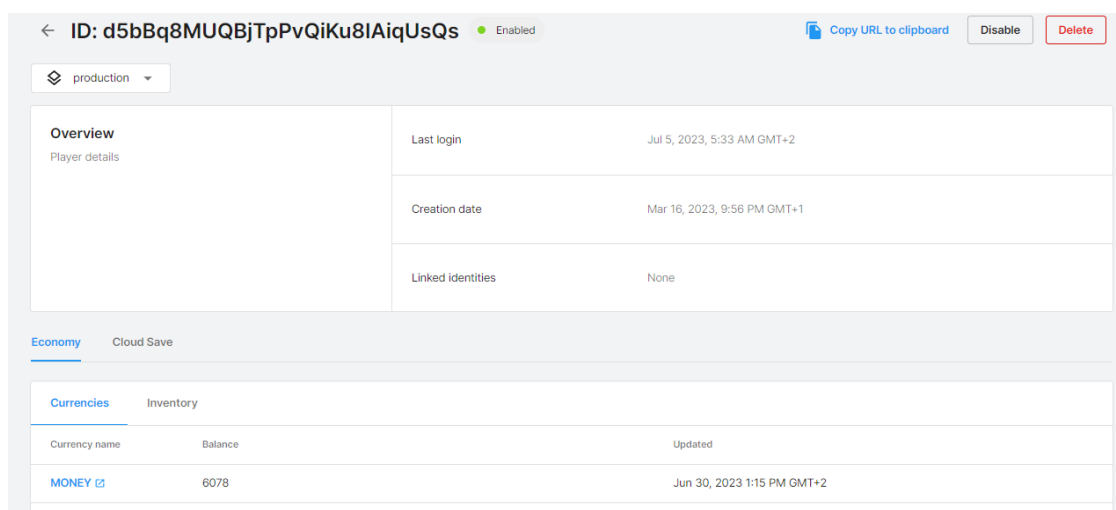
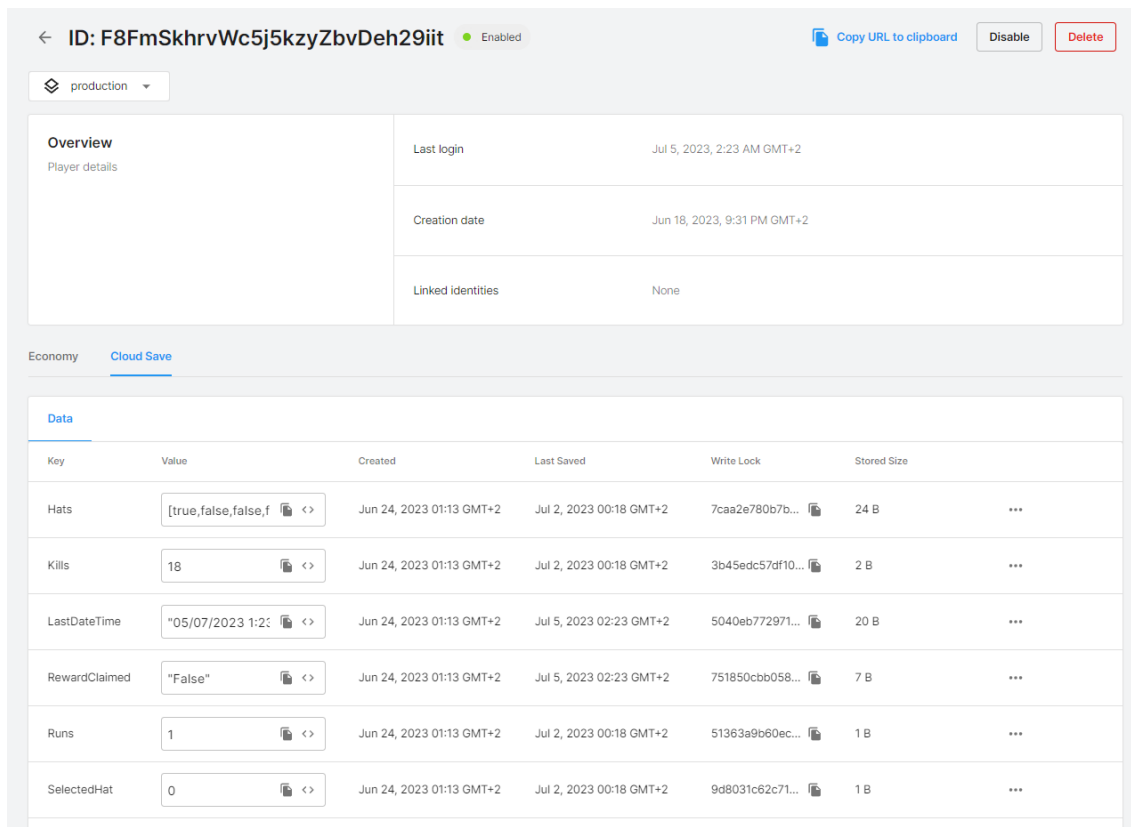


Figura 31 - Pestaña de economy en base de datos de un usuario dado en PistoleroVR. Elaboración propia.

En cuanto a los datos en sí se refieren (figura 32), se han guardado en formato json. Los datos para guardar son: las bajas totales que haya hecho el jugador, el número total de partidas que haya jugado, la fecha exacta de la última vez que se reclamó la recompensa diaria, si dicha recompensa se encuentra disponible para volver a reclamarla, el id del cosmético actualmente seleccionado y cuáles de los cosméticos existentes están adquiridos por el usuario.



Key	Value	Created	Last Saved	Write Lock	Stored Size
Hats	[true,false,false,f <>]	Jun 24, 2023 01:13 GMT+2	Jul 2, 2023 00:18 GMT+2	7caa2e780b7b... <>	24 B
Kills	18 <>	Jun 24, 2023 01:13 GMT+2	Jul 2, 2023 00:18 GMT+2	3b45edc57df10... <>	2 B
LastDateTime	"05/07/2023 1:20" <>	Jun 24, 2023 01:13 GMT+2	Jul 5, 2023 02:23 GMT+2	5040eb772971... <>	20 B
RewardClaimed	"False" <>	Jun 24, 2023 01:13 GMT+2	Jul 5, 2023 02:23 GMT+2	751850cbb058... <>	7 B
Runs	1 <>	Jun 24, 2023 01:13 GMT+2	Jul 2, 2023 00:18 GMT+2	51363a9b60ec... <>	1 B
SelectedHat	0 <>	Jun 24, 2023 01:13 GMT+2	Jul 2, 2023 00:18 GMT+2	9d8031c62c71... <>	1 B

Figura 32 - Base de datos de un usuario dado en PistoleroVR. Elaboración propia.

La ventaja de guardar los datos de esta forma es que es más eficiente serializar la clase de estadísticas a guardar de los clientes (las instancias de juego) a json y realizar peticiones de subida o bajada con ese formato. Para lograr esto se ha dividido la subida de datos en dos secciones, por un lado, lo relacionado a la recompensa diaria y por el otro el resto. Como se puede ver en la siguiente figura, se trata de dos métodos asíncronos de subida y bajada, de esta forma, se asegura la espera necesaria para que se realicen las operaciones.

Hay que destacar en la figura 33 la llamada de carga de la base de datos cuando la aplicación se abre, y la subida de datos actuales cuando el jugador salga del juego. De esta forma, se asegura mantener los datos actualizados en todo momento.

```

6 references
public async Task SaveData()
{
    _unlockedHats ??= new bool[] { true, false, false, false };
    var data = new Dictionary<string, object>
    {
        { "Kills", _kills }, { "Runs", _runs }, { "Hats", _unlockedHats }, { "SelectedHat", _selectedHat }
    };
    await CloudSaveService.Instance.Data.ForceSaveAsync(data);
}

1 reference
public async Task LoadData()
{
    var data = await CloudSaveService.Instance.Data.LoadAllAsync();

    // Debug.Log("DATA: " + data);
    foreach (var (key, value) in data)
    {
        if (key == "Kills")
            _kills = int.Parse(value);
        if (key == "Runs")
            _runs = int.Parse(value);
        if (key == "Hats")
            _unlockedHats = StringToBoolArray(value);
        if (key == "SelectedHat")
            _selectedHat = int.Parse(value);
    }

    UpdateHatSelected();
    EventManager.OnPlayerDataLoaded();
}

```

Figura 33 - Métodos de subida y bajada de datos en pistoleroVR. Elaboración propia.

Por otro lado, los datos de la recompensa diaria funcionan en su propia clase, como se ve en la figura 34. Esto se debe a su relación con el siguiente sdk del que se indagará a continuación (*cloud code*) y el principio de responsabilidad única. En líneas generales, si bien estas dos variables se actualizan también al abrir y cerrar la aplicación, también se actualiza si el tiempo de espera pasa o el jugador reclama la recompensa.

```

private async void RewardClaimed()
{
    rewardClaimed = true;
    lastDateTime = DateTime.Now;
    await SaveToCloud(data, "LastDateTime", DateTime.Now.ToString());
    await SaveToCloud(data, "RewardClaimed", rewardClaimed.ToString());
}

```

Figura 34 - Método de subida de datos para recompensas en PistoleroVR. Elaboración propia.

En referencia al Cloud Code, es utilizado para modificar tanto la moneda del jugador (*economy*) como para recibir el valor de la recompensa diaria. Así pues, se han creado tres scripts distintos para cada una de las acciones requeridas, como se puede visualizar en la figura 35.

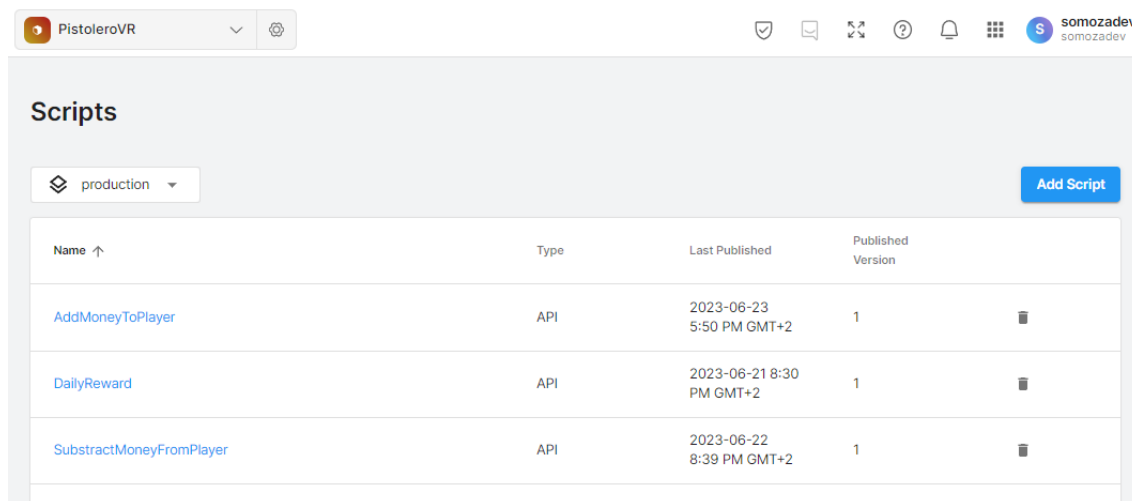


Figura 35 - Scripts creados en Cloud Code en PistoleroVR. Elaboración propia.

Todos estos scripts han sido escritos en JavaScript, y actúan como API privada para los clientes. En cuanto a los scripts, *AddMoneyToPlayer* y *SubstractMoneyFromPlayer* funcionan de forma muy similar. Ambos reciben un valor (ya sea para añadirlo o para substraerlo) y lo operan con el valor de la moneda actual del usuario que haga la *request*, como se puede ver en la figura 36.

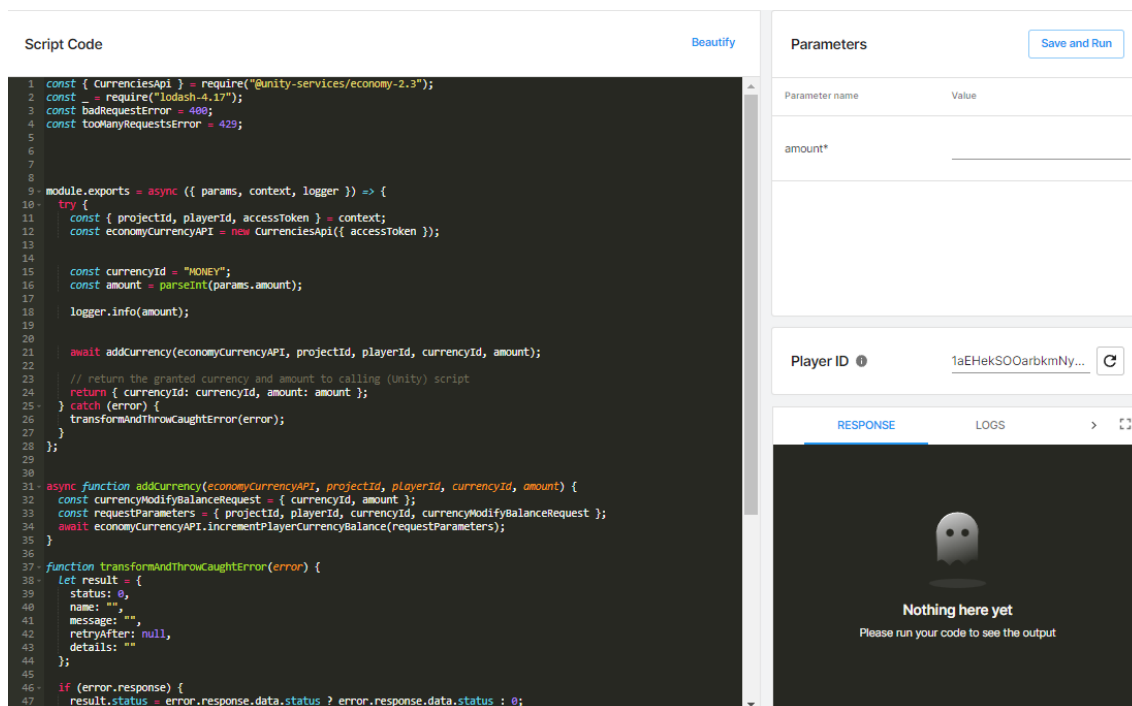
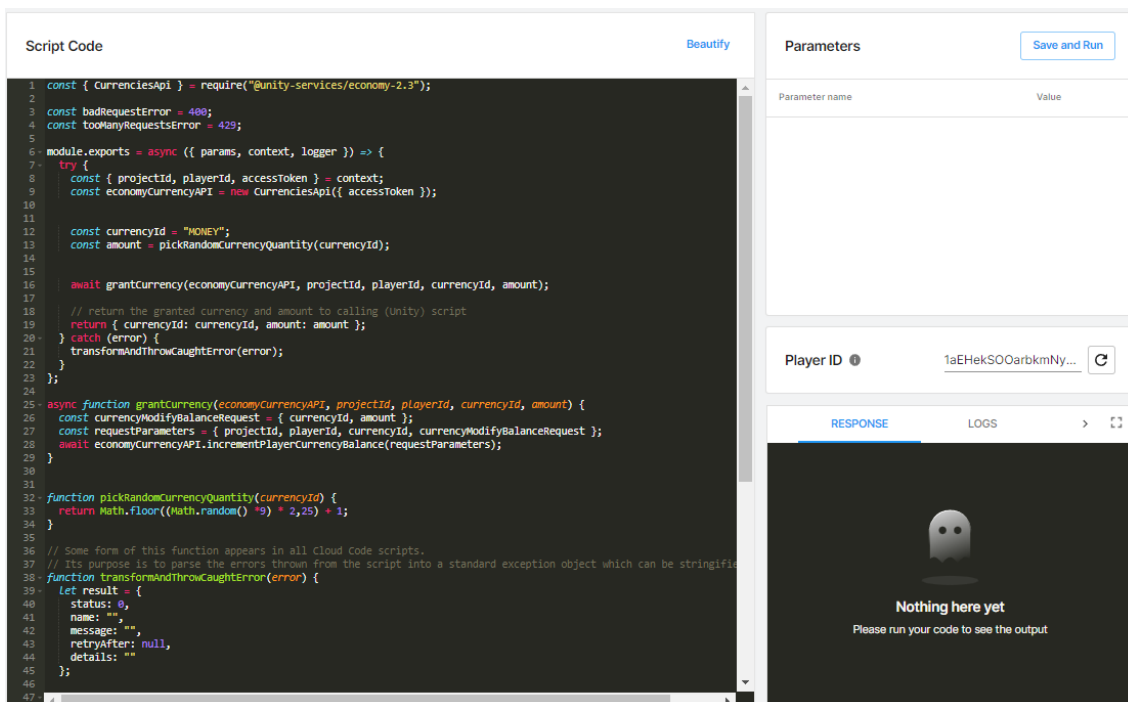


Figura 36 - Script de AddMoneyToPlayer en Cloud Script en PistoleroVR. Elaboración propia.

La única diferencia entre ambos scripts es que *AddMoney* hace uso del método de la API de *economy* "*economyCurrencyAPI.incrementPlayerCurrencyByBalance*", mientras que el script

*SubstractMoneyFromPlayer* hace uso de “*economyCurrencyAPI.decrementPlayerCurrencyBalance*”, uno añade y otro resta. Estos métodos son utilizados por los clientes cuando compra un cosmético o cuando se quiere incrementar la economía desde el mismo *dashboard* para las pruebas.

Por otro lado, se encuentra el script *DailyReward*, en la figura 37. Este hace uso de un método que devuelve un número a bonificar al jugador en base a  $\text{Math.floor}((\text{Math.random}() * 9) * 2,25) + 1$  llamando al mismo método de la API de *economy* al que llama *AddMoney*.



```

1  const { CurrenciesApi } = require("@unity-services/economy-2.3");
2
3  const badRequestError = 400;
4  const tooManyRequestsError = 429;
5
6  module.exports = async ({ params, context, logger }) => {
7    try {
8      const { projectId, playerId, accessToken } = context;
9      const economyCurrencyAPI = new CurrenciesApi({ accessToken });
10
11
12      const currencyId = "MONEY";
13      const amount = pickRandomCurrencyQuantity(currencyId);
14
15      await grantCurrency(economyCurrencyAPI, projectId, playerId, currencyId, amount);
16
17      // return the granted currency and amount to calling (unity) script
18      return { currencyId: currencyId, amount: amount };
19    } catch (error) {
20      transformAndThrowCaughtError(error);
21    }
22  };
23
24
25  async function grantCurrency(economyCurrencyAPI, projectId, playerId, currencyId, amount) {
26    const currencyModifyBalanceRequest = { currencyId, amount };
27    const requestParameters = { projectId, playerId, currencyId, currencyModifyBalanceRequest };
28    await economyCurrencyAPI.incrementPlayerCurrencyBalance(requestParameters);
29  }
30
31  function pickRandomCurrencyQuantity(currencyId) {
32    return Math.floor((Math.random() * 9) * 2,25) + 1;
33  }
34
35  // Some form of this function appears in all Cloud Code scripts.
36  // Its purpose is to parse the errors thrown from the script into a standard exception object which can be stringified
37  function transformAndThrowCaughtError(error) {
38    let result = {
39      status: 0,
40      name: "",
41      message: "",
42      retryAfter: null,
43      details: ""
44    };
45  }
46
47

```

Parameters

Parameter name	Value
Player ID	1aEHekSO0arbkMny...

RESPONSE LOGS

Nothing here yet  
Please run your code to see the output

Figura 37 - Script *DailyReward* en Cloud Code en PistoleroVR. Elaboración propia.

## 4. Análisis UML

### 1. Actores

Lo primero que debe identificarse son los actores. Debido a que este videojuego es de un sólo jugador, y los únicos usuarios reales son los jugadores, solamente existe un actor.

<b>Id y Nombre</b>	ACT-01 Jugador
<b>Versión</b>	V1
<b>Descripción</b>	Usuario / jugador activo en el proyecto

Tabla 1- ACT-01 Jugador. Elaboración propia.

## 2. Casos de uso

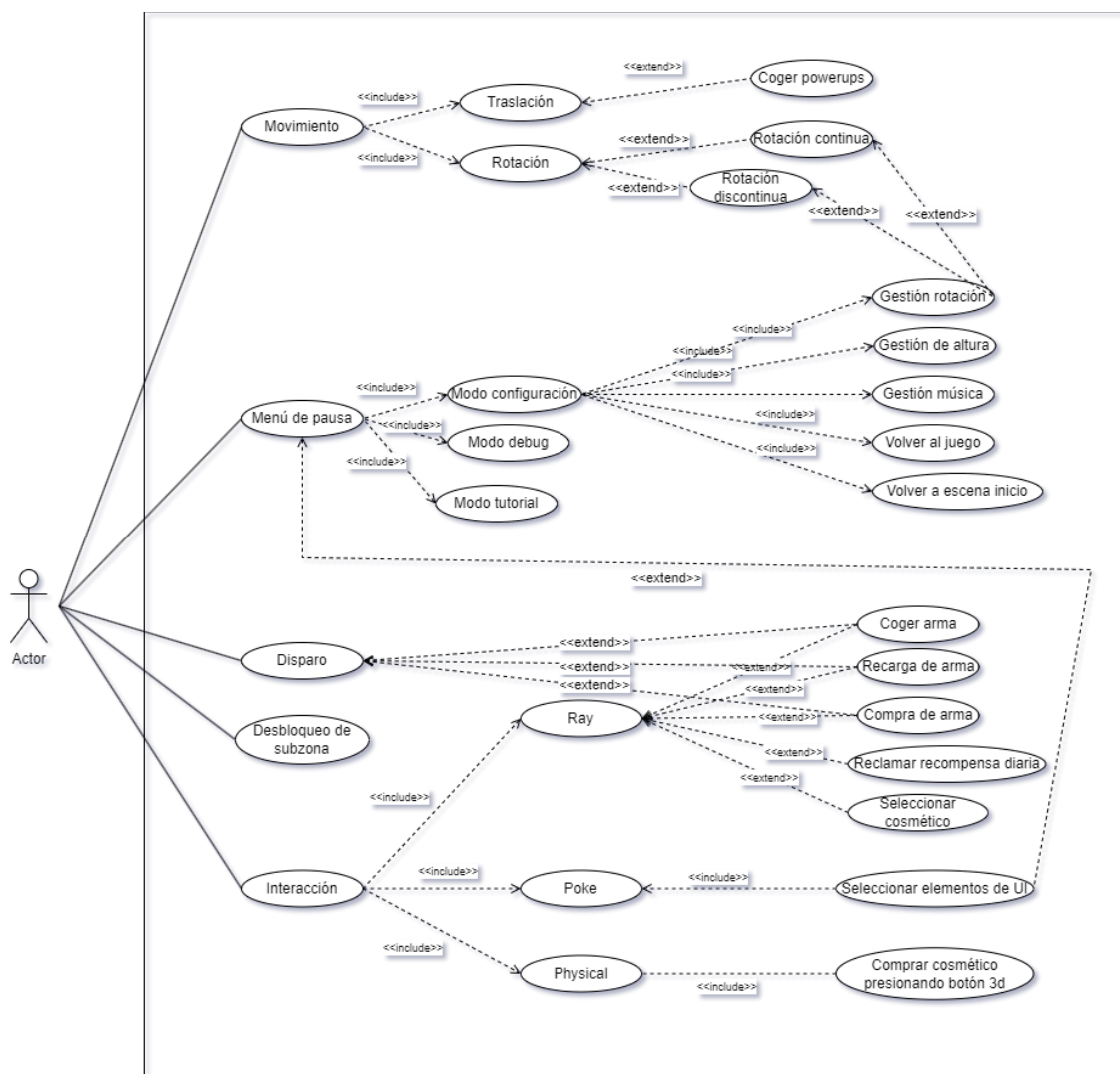


Figura 38 - Diagrama de casos de uso del jugador con el mundo. Elaboración propia.

<b>Id</b>	CU-01	
<b>Caso de uso</b>	Movimiento	
<b>Actor</b>	Usuario/Jugador	
<b>Resumen</b>	El jugador puede mover al personaje de diversas formas.	
<b>Precondiciones</b>	El jugador usa algún <i>input</i> ( <i>joystick</i> , botón, movimiento del HMD...).	
<b>Postcondiciones</b>	El personaje del jugador responde a dicho <i>input</i> , generando movimiento.	
<b>Incluye</b>	Traslación y Rotación	
<b>Extiende</b>	-	
<b>Hereda de</b>	-	
<b>Flujo de eventos</b>		
<b>Actor</b>	<b>Sistema</b>	
1. El jugador hace uso de un input de sus controladores, aquellos que estén correspondidos en el videojuego (botones, <i>triggers</i> , <i>joysticks</i> , posición física de los mandos o el HMD...)	2. El sistema detecta el input y realiza los movimientos correspondientes en el personaje del videojuego.	

Tabla 2 - CU-01. Elaboración propia.

<b>Id</b>	CU-02	
<b>Caso de uso</b>	Traslación	
<b>Actor</b>	Usuario/Jugador	
<b>Resumen</b>	El personaje se moverá en base a las acciones del jugador.	
<b>Precondiciones</b>	El jugador usa <i>joystick</i> izquierdo, o mueve su HMD, mando de la mano izquierda o mando de la mano derecha.	
<b>Postcondiciones</b>	El personaje del jugador se mueve (traslación).	
<b>Incluye</b>	-	
<b>Extiende</b>	Coger powerups	
<b>Hereda de</b>	-	

Flujo de eventos	
Actor	Sistema
1. El jugador hace uso de algún <i>input</i> de movimiento.	2. El sistema lo detecta y traslada al personaje del videojuego en la dirección del <i>input</i> .

Tabla 3 - CU-02. Elaboración propia.

<b>Id</b>	CU-03	
<b>Caso de uso</b>	Coger <i>powerup</i>	
<b>Actor</b>	Usuario/Jugador	
<b>Resumen</b>	El personaje recibirá un potenciador dependiendo de qué <i>powerup</i> sea; munición máxima, invencibilidad, <i>instakill</i> o puntos dobles.	
<b>Precondiciones</b>	El personaje colisiona con el <i>powerup</i> en el espacio 3D.	
<b>Postcondiciones</b>	El efecto del <i>powerup</i> se aplica.	
<b>Incluye</b>	-	
<b>Extiende</b>	-	
<b>Hereda de</b>	-	
<b>Flujo de eventos</b>		
<b>Actor</b>		<b>Sistema</b>
1. El jugador mueve al personaje para que colisione con un <i>powerup</i> .		2. El <i>powerup</i> aplica sus efectos y se destruye.

Tabla 4 - CU-03. Elaboración propia.

<b>Id</b>	CU-04	
<b>Caso de uso</b>	Rotación	
<b>Actor</b>	Usuario/Jugador	



<b>Resumen</b>	El personaje rotará en base a las acciones del jugador.
<b>Precondiciones</b>	El jugador usa <i>joystick</i> derecho, o rota su HMD, mando de la mano izquierda o mando de la mano derecha.
<b>Postcondiciones</b>	El personaje del jugador rota (rotación).
<b>Incluye</b>	-
<b>Extiende</b>	Rotación continua y rotación discontinua
<b>Hereda de</b>	-
<b>Flujo de eventos</b>	
<b>Actor</b>	<b>Sistema</b>
1. El jugador hace uso de algún <i>input</i> de rotación.	2. El sistema lo detecta y rota al personaje del videojuego en la dirección del <i>input</i> .

Tabla 5 - CU-04. Elaboración propia.

<b>Id</b>	CU-05
<b>Caso de uso</b>	Rotación continua
<b>Actor</b>	Usuario/Jugador
<b>Resumen</b>	El personaje rotará en base a las acciones del jugador de forma continua.
<b>Precondiciones</b>	El jugador usa <i>joystick</i> derecho y tiene seleccionado este método de rotación.
<b>Postcondiciones</b>	El personaje del jugador rota (rotación) de forma continua.
<b>Incluye</b>	-
<b>Extiende</b>	-
<b>Hereda de</b>	-
<b>Flujo de eventos</b>	
<b>Actor</b>	<b>Sistema</b>
1. El jugador activa la rotación continua. 3. El jugador usa los <i>inputs</i> de rotación.	2. El sistema lo detecta y cambia el método de rotación. 4. El personaje rota en base a los <i>inputs</i> .

Tabla 6 - CU-05. Elaboración propia.

<b>Id</b>	CU-06
<b>Caso de uso</b>	Rotación discontinua
<b>Actor</b>	Usuario/Jugador
<b>Resumen</b>	El personaje rotará en base a las acciones del jugador de forma discontinua (cada vez que rote, se rotarán 60 grados).
<b>Precondiciones</b>	El jugador usa <i>joystick</i> derecho y tiene seleccionado este método de rotación.
<b>Postcondiciones</b>	El personaje del jugador rota (rotación) de forma discontinua.
<b>Incluye</b>	-
<b>Extiende</b>	-
<b>Hereda de</b>	-
<b>Flujo de eventos</b>	
<b>Actor</b>	<b>Sistema</b>
1. El jugador activa la rotación discontinua. 3. El jugador usa los <i>inputs</i> de rotación.	2. El sistema lo detecta y cambia el método de rotación. 4. El personaje rota en base a los <i>inputs</i> .

Tabla 7 - CU-06. Elaboración propia.

<b>Id</b>	CU-07	
<b>Caso de uso</b>	Menú de pausa	
<b>Actor</b>	Usuario/Jugador	
<b>Resumen</b>	El jugador despliega un menú donde tendrá diferentes opciones e información, cuando lo haga, el resto del juego y su movimiento y rotación por <i>joysticks</i> se congelará hasta que lo cierre.	

<b>Precondiciones</b>	El jugador presiona el botón de menú situado en su mano izquierda.
<b>Postcondiciones</b>	El menú se abrirá si no lo estaba antes, o cerrará si sí.
<b>Incluye</b>	Modo configuración, modo <i>debug</i> y modo tutorial.
<b>Extiende</b>	Seleccionar elementos de UI
<b>Hereda de</b>	-
<b>Flujo de eventos</b>	
<b>Actor</b>	<b>Sistema</b>
1. El jugador presiona el botón de menú. 3. El jugador navegará por los diferentes submenús, interactuando mediante el <i>poke interactor</i> . 5. El jugador volverá a presionar el botón de menú.	2. El menú se desplegará en su muñeca izquierda, su navegación se divide en 3, config, debug y tutorial. El juego se pausa. 4. El menú responderá a los <i>poke interactions</i> del jugador. 6. Se cerrará el menú y el juego continuará.

Tabla 8 - CU-07. Elaboración propia.

<b>Id</b>	CU-08
<b>Caso de uso</b>	Modo configuración
<b>Actor</b>	Usuario/Jugador
<b>Resumen</b>	En este modo el jugador podrá modificar diferentes elementos, como la altura del HMD para con el juego, el volumen del juego, el tipo de rotación, volver al juego o volver a la escena de inicio.
<b>Precondiciones</b>	Se deberá haber abierto el menú y presionar en la pestaña de configuración.
<b>Postcondiciones</b>	Se abrirá la pestaña de configuración.
<b>Incluye</b>	Gestión rotación, gestión altura, gestión música, volver al juego y volver a escena inicio.
<b>Extiende</b>	-
<b>Hereda de</b>	-
<b>Flujo de eventos</b>	

Actor	Sistema
1. El jugador presionará el botón de menú. 3. El jugador puede presionar el botón de este submenú.	2. Este se abrirá, que de forma determinada despliega el menú de configuración. 3. El menú pasará al menú de configuración.

Tabla 9 - CU-08. Elaboración propia.

<b>Id</b>	CU-09
<b>Caso de uso</b>	Gestión de rotación
<b>Actor</b>	Usuario/Jugador
<b>Resumen</b>	Se cambiará de un sistema de rotación continua a otro discontinua y viceversa cada vez que se presione este <i>toggle</i> .
<b>Precondiciones</b>	Se deberá haber abierto el menú y estar en la pestaña de configuración, así como presionar el <i>toggle</i> .
<b>Postcondiciones</b>	El método de rotación se alternará en base al valor del <i>toggle</i> .
<b>Incluye</b>	-
<b>Extiende</b>	Rotación continua y rotación discontinua.
<b>Hereda de</b>	-
<b>Flujo de eventos</b>	
<b>Actor</b>	<b>Sistema</b>
1. El jugador presionará el toggle.	2. El tipo de rotación cambiará.

Tabla 10 - CU-09. Elaboración propia.

<b>Id</b>	CU-10	
<b>Caso de uso</b>	Gestión de Altura	
<b>Actor</b>	Usuario/Jugador	
<b>Resumen</b>	Se podrá regular el nivel de volumen del juego mediante el <i>poke interactor</i> , moviendo un <i>slider</i> .	
<b>Precondiciones</b>	Se deberá haber abierto el menú y estar en la pestaña de configuración,	

	así como mover el valor del <i>slider</i> .
<b>Postcondiciones</b>	La altura del personaje variará en base al valor del <i>slider</i> .
<b>Incluye</b>	-
<b>Extiende</b>	-
<b>Hereda de</b>	-
<b>Flujo de eventos</b>	
<b>Actor</b>	<b>Sistema</b>
1. El jugador moverá el <i>slider</i> para cambiar la altura.	2. La altura del personaje responderá al valor de dicho <i>slider</i> .

Tabla 11 - CU-10. Elaboración propia.

<b>Id</b>	CU-11		
<b>Caso de uso</b>	Gestión de música		
<b>Actor</b>	Usuario/Jugador		
<b>Resumen</b>	Se podrá regular el nivel de volumen del juego mediante el <i>poke interactor</i> , moviendo un <i>slider</i> .		
<b>Precondiciones</b>	Se deberá haber abierto el menú y estar en la pestaña de configuración, así como mover el valor del <i>slider</i> .		
<b>Postcondiciones</b>	El volumen del juego variará en base al valor del <i>slider</i> .		
<b>Incluye</b>	-		
<b>Extiende</b>	-		
<b>Hereda de</b>	-		
<b>Flujo de eventos</b>			
<b>Actor</b>		<b>Sistema</b>	
1. El jugador moverá el <i>slider</i> para cambiar el volumen.		2. El volumen del juego responderá al valor de dicho <i>slider</i> .	

Tabla 12 - CU-11. Elaboración propia.

<b>Id</b>	CU-12	
<b>Caso de uso</b>	Volver al juego	
<b>Actor</b>	Usuario/Jugador	
<b>Resumen</b>	Se volverá a la partida, volviendo a “descongelar” los elementos de esta.	
<b>Precondiciones</b>	Tener abierto el menú en configuración y presionar el botón de <i>Resume</i> o volver a presionar el botón de menú del controlador que abre/cierra el menú de pausa (CU7).	
<b>Postcondiciones</b>	Se volverá a la partida.	
<b>Incluye</b>	-	
<b>Extiende</b>	-	
<b>Hereda de</b>	-	
<b>Flujo de eventos</b>		
<b>Actor</b>	<b>Sistema</b>	
1. El jugador presionará el botón de <i>Resume</i> o el botón de su mando izquierdo de menú.	2. Se volverá a la partida.	

*Tabla 13 - CU-12. Elaboración propia.*

<b>Id</b>	CU-13	
<b>Caso de uso</b>	Volver a escena inicio	
<b>Actor</b>	Usuario/Jugador	
<b>Resumen</b>	Se volverá a la escena de inicio, en la posición inicial donde se empezó en el juego.	
<b>Precondiciones</b>	Tener abierto el menú en configuración y presionar el botón de <i>Menu</i> .	
<b>Postcondiciones</b>	Se volverá a la escena de inicio.	
<b>Incluye</b>	-	
<b>Extiende</b>	-	
<b>Hereda de</b>	-	

Flujo de eventos	
Actor	Sistema
1. El jugador presionará el botón de <i>Home</i> .	2. Se volverá a la escena de inicio.

Tabla 14 - CU-13. Elaboración propia.

<b>Id</b>	CU-14	
<b>Caso de uso</b>	Modo <i>debug</i>	
<b>Actor</b>	Usuario/Jugador	
<b>Resumen</b>	Este menú muestra información de <i>debug</i> conveniente para el desarrollo, como el id del usuario en la base de datos, los fps del juego, las especificaciones técnicas del dispositivo, la batería del dispositivo...	
<b>Precondiciones</b>	Tener abierto el menú y presionar el botón para pasar a menú de debug.	
<b>Postcondiciones</b>	Se mostrará el menú de <i>debug</i> con la información pertinente.	
<b>Incluye</b>	-	
<b>Extiende</b>	-	
<b>Hereda de</b>	-	
<b>Flujo de eventos</b>		
<b>Actor</b>	<b>Sistema</b>	
1. El jugador presionará el botón para abrir el submenú de <i>debug</i> .	2. Se cerrará el menú actual y se abrirá el de <i>debug</i> .	

Tabla 15 - CU-14. Elaboración propia.

<b>Id</b>	CU-15	
<b>Caso de uso</b>	Modo tutorial	
<b>Actor</b>	Usuario/Jugador	
<b>Resumen</b>	Un submenú donde se muestran tres párrafos explicando los sistemas de interacción.	

<b>Precondiciones</b>	Tener abierto el menú y presionar el botón para pasar a menú de tutorial.
<b>Postcondiciones</b>	Se mostrará el menú de tutorial con la información pertinente.
<b>Incluye</b>	-
<b>Extiende</b>	-
<b>Hereda de</b>	-
<b>Flujo de eventos</b>	
<b>Actor</b>	<b>Sistema</b>
1. El jugador presionará el botón para abrir el submenú de tutorial.	2. Se cerrará el menú actual y se abrirá el de tutorial.

*Tabla 16 - CU-15. Elaboración propia.*

<b>Id</b>	CU-16
<b>Caso de uso</b>	Disparo
<b>Actor</b>	Usuario/Jugador
<b>Resumen</b>	El jugador podrá disparar un arma.
<b>Precondiciones</b>	Deberá tener tanto un arma en la mano como munición en la misma y presionar el <i>trigger</i> del mando de la mano correspondiente a la que tiene sujetando el arma dentro del videojuego.
<b>Postcondiciones</b>	Se disparará una bala que si colisiona con un objeto rígido dejará una marca ( <i>decal</i> ) y si colisiona con un objeto móvil, le aplicará fuerza y daño.
<b>Incluye</b>	-
<b>Extiende</b>	Coger arma, Recarga de arma, Compra de arma
<b>Hereda de</b>	-
<b>Flujo de eventos</b>	
<b>Actor</b>	<b>Sistema</b>
1. El jugador coge un arma con el <i>ray interactor</i> . 2. El jugador presiona el <i>trigger</i> de la mano correspondiente a la que está agarrando el	2. El arma se coloca en su mano. 3. El arma se dispara. Si es la última bala, se recarga sola y dispara.



arma.	
-------	--

*Tabla 17 - CU-16. Elaboración propia.*

<b>Id</b>	CU-17		
<b>Caso de uso</b>	Desbloqueo de subzona		
<b>Actor</b>	Usuario/Jugador		
<b>Resumen</b>	El jugador podrá desbloquear una zona del mapa de juego.		
<b>Precondiciones</b>	Deberá estar suficientemente cerca de dicha zona para poder desbloquearla. Una vez cerca y con dinero suficiente, deberá presionar el botón “a” de su mando derecho.		
<b>Postcondiciones</b>	Se quitará el coste de la zona a la cantidad de dinero que tenga el jugador y se liberará el camino.		
<b>Incluye</b>	-		
<b>Extiende</b>	-		
<b>Hereda de</b>	-		
<b>Flujo de eventos</b>			
<b>Actor</b>		<b>Sistema</b>	
1. El jugador se acercará a la zona desbloqueable. 2. El jugador presionará el botón a.		2. La zona mostrará una UI con el precio a pagar y la forma de pagar ( <i>press a</i> ). 3. Se retirará el coste de la zona del dinero del jugador y se abrirá el camino.	

*Tabla 18 - CU-17. Elaboración propia.*

<b>Id</b>	CU-18	
<b>Caso de uso</b>	Interacción	
<b>Actor</b>	Usuario/Jugador	
<b>Resumen</b>	Las diferentes maneras que tiene el jugador para interaccionar con el mundo.	
<b>Precondiciones</b>	Deberá tener el HMD y los mandos conectados.	

<b>Postcondiciones</b>	Tendrá a su disposición las tres formas de interactuar.
<b>Incluye</b>	-
<b>Extiende</b>	<i>Ray, Poke y Physical.</i>
<b>Hereda de</b>	-
<b>Flujo de eventos</b>	
<b>Actor</b>	<b>Sistema</b>
1. El jugador estará en el juego con el HMD y los controles conectados.	2. El sistema permitirá el uso de los diferentes interactuadores.

*Tabla 19 - CU-18. Elaboración propia.*

<b>Id</b>	CU-19		
<b>Caso de uso</b>	Ray		
<b>Actor</b>	Usuario/Jugador		
<b>Resumen</b>	El jugador podrá interactuar con determinados elementos del entorno mediante un “rayo” con origen en cada mano.		
<b>Precondiciones</b>	Que el objeto con el que quiera interactuar no esté demasiado lejos y sea apto para el <i>ray interactor</i> . El jugador deberá presionar el <i>trigger</i> interno de la mano con la que apunte al objeto.		
<b>Postcondiciones</b>	El objeto se teletransportará a su mano si es un arma, en otro caso realizará su interacción (p.e, seleccionarse).		
<b>Incluye</b>	-		
<b>Extiende</b>	Coger arma, Recarga de arma, Compra de arma, reclamar recompensa diaria y seleccionar cosmético.		
<b>Hereda de</b>	-		
<b>Flujo de eventos</b>			
<b>Actor</b>		<b>Sistema</b>	
1. El jugador apunta a con lo que quiera interactuar. 2. El jugador presiona el <i>trigger</i> interno de la mano correspondiente a la que está		2. El objeto es interactuable con el rayo. 3. El objeto realiza su interacción, si es un arma, se teletransportará a su mano.	

apuntando.	
------------	--

Tabla 20 - CU-19. Elaboración propia.

<b>Id</b>	CU-20	
<b>Caso de uso</b>	Coger arma	
<b>Actor</b>	Usuario/Jugador	
<b>Resumen</b>	El jugador podrá interactuar con el arma del entorno mediante el <i>ray interactor</i> con cualquier mano.	
<b>Precondiciones</b>	Que el arma no esté demasiado lejos y que el jugador no tenga algo ya en la mano. El jugador deberá presionar el <i>trigger</i> interno de la mano con la que apunte al arma.	
<b>Postcondiciones</b>	El arma se teletransportará a su mano si no tenía nada. Si sí tuviera algo, este algo se soltaría y caería al suelo.	
<b>Incluye</b>	-	
<b>Extiende</b>	-	
<b>Hereda de</b>	-	
<b>Flujo de eventos</b>		
<b>Actor</b>	<b>Sistema</b>	
1. El jugador apunta al arma y presiona el <i>trigger interno</i> de la mano con la que apunta.	2. Si no tiene nada en la mano, el arma se teletransporta a esta. Si sí tiene, soltará lo que tenga en la mano.	

Tabla 21 - CU-20. Elaboración propia.

<b>Id</b>	CU-21	
<b>Caso de uso</b>	Recargar arma	
<b>Actor</b>	Usuario/Jugador	
<b>Resumen</b>	El jugador puede comprar una recarga completa de su arma si necesita más munición.	
<b>Precondiciones</b>	Que el arma no esté llena. Que el jugador tenga el arma en la mano y esté	

	suficientemente cerca de la tienda que venda dicha arma. Que en la otra mano no tenga otra arma. Que el jugador tenga al menos 1/3 del coste del arma para poder comprar su recarga de munición.
<b>Postcondiciones</b>	El arma que tenga se llenará de munición, igual que en su modo predeterminado.
<b>Incluye</b>	-
<b>Extiende</b>	-
<b>Hereda de</b>	-
<b>Flujo de eventos</b>	
<b>Actor</b>	<b>Sistema</b>
1. El jugador tiene un arma que no esté cargada al máximo de munición. 2. El jugador apunta a la tienda que venda dicha arma y presiona el <i>trigger</i> de la mano con la que apunta.	3. Si el jugador tiene suficiente dinero para comprar la recarga de munición, se le restará el coste de la recarga a su economía. 4. Su arma descargada se cargará por completo.

*Tabla 22 - CU-21. Elaboración propia.*

<b>Id</b>	CU-22
<b>Caso de uso</b>	Compra de arma
<b>Actor</b>	Usuario/Jugador
<b>Resumen</b>	El jugador puede comprar una armas en diferentes tiendas del mapa si cuenta con el dinero suficiente.
<b>Precondiciones</b>	Que tenga dinero suficiente, que no tenga otra arma en la mano que esté usando para interactuar y que esté suficientemente cerca.
<b>Postcondiciones</b>	El jugador tendrá el arma en su mano lista para su uso.
<b>Incluye</b>	-
<b>Extiende</b>	-
<b>Hereda de</b>	-
<b>Flujo de eventos</b>	

Actor	Sistema
1. El jugador apunta a la tienda y presiona el <i>trigger</i> de la mano con la que apunta.	2. Si el jugador tiene suficiente dinero para comprar, se le restará el coste de la recarga a su economía. 3. El arma descargada se colocará en su mano.

Tabla 23 - CU-22. Elaboración propia.

<b>Id</b>	CU-23
<b>Caso de uso</b>	Reclamar recompensa diaria
<b>Actor</b>	Usuario/Jugador
<b>Resumen</b>	El jugador puede interactuar con una olla de oro en el escenario inicial cada 24 horas, que le brindará una cantidad de dinero aleatoria (dentro de unos límites) para poder comprar cosméticos con ello.
<b>Precondiciones</b>	Que hayan pasado al menos 24 horas desde que se reclamó por última vez.
<b>Postcondiciones</b>	Se reinicia el contador y el jugador recibe una cantidad de dinero.
<b>Incluye</b>	-
<b>Extiende</b>	-
<b>Hereda de</b>	-
<b>Flujo de eventos</b>	
<b>Actor</b>	<b>Sistema</b>
1. El jugador apunta a la olla y presiona el <i>trigger</i> de la mano con la que apunta.	2. Si ha pasado al menos 24 horas, recibe una recompensa y reinicia el contador.

Tabla 24 - CU-23. Elaboración propia.

<b>Id</b>	CU-24	
<b>Caso de uso</b>	Seleccionar cosmético	
<b>Actor</b>	Usuario/Jugador	

<b>Resumen</b>	El usuario podrá seleccionar aquel cosmético que quiera comprar, o equipar un cosmético que ya tenga adquirido.
<b>Precondiciones</b>	El jugador debe encontrarse a una distancia suficientemente cercana de los cosméticos en el escenario principal.
<b>Postcondiciones</b>	Se reinicia el contador y el jugador recibe una cantidad de dinero.
<b>Incluye</b>	-
<b>Extiende</b>	-
<b>Hereda de</b>	-
<b>Flujo de eventos</b>	
<b>Actor</b>	<b>Sistema</b>
1. El jugador apunta al cosmético que quiera seleccionar para comprar si no lo tiene adquirido aún, o para equipar uno que ya tenga adquirido. 2. Presionará el trigger de la mano con la que apunta.	3. Si es un cosmético para comprar, éste se marcará para que pulse el botón de comprar. Si el precio de este cosmético es inferior al dinero del jugador, se colorará de verde, sino de rojo. Si es un cosmético ya comprado, éste se seleccionará y equipará en el jugador.

*Tabla 25 - CU-24. Elaboración propia.*

<b>Id</b>	CU-25
<b>Caso de uso</b>	Poke
<b>Actor</b>	Usuario/Jugador
<b>Resumen</b>	El sistema de interacción donde el jugador puede interactuar mediante toques con el “puño” de su personaje, concretamente con elementos de la interfaz gráfica.
<b>Precondiciones</b>	Que el elemento con el que quiera interactuar esté en contacto con el puño del personaje y sea un elemento de la interfaz gráfica.
<b>Postcondiciones</b>	El elemento responde como interactuado.
<b>Incluye</b>	Seleccionar elementos de UI
<b>Extiende</b>	-

<b>Hereda de</b>	-
<b>Flujo de eventos</b>	
<b>Actor</b>	<b>Sistema</b>
1. El jugador tocará con el puño un elemento.	2. Si el elemento puede ser interactuado mediante el “poke”, reaccionará.

Tabla 26 - CU-25. Elaboración propia.

<b>Id</b>	CU-26
<b>Caso de uso</b>	Seleccionar elementos de UI
<b>Actor</b>	Usuario/Jugador
<b>Resumen</b>	El usuario podrá seleccionar e interactuar con los elementos de la UI mediante el <i>poke interactor</i> .
<b>Precondiciones</b>	Que el elemento con el que quiera interactuar este a rango de su puño y sea apto para el <i>poke interactor</i> .
<b>Postcondiciones</b>	El elemento con el que interactúe reaccionará. (si es un botón, presionado, si es un <i>slider</i> , su valor cambiado, si es un <i>toggle</i> , verdadero o falso, etc....)
<b>Incluye</b>	-
<b>Extiende</b>	Menú de pausa
<b>Hereda de</b>	-
<b>Flujo de eventos</b>	
<b>Actor</b>	<b>Sistema</b>
1. El jugador tocará con el puño un elemento de la UI.	2. El elemento reaccionará dependiendo del tipo de UI que sea.

Tabla 27 - CU-26. Elaboración propia.

<b>Id</b>	CU-27	
<b>Caso de uso</b>	Physical	
<b>Actor</b>	Usuario/Jugador	
<b>Resumen</b>	El usuario podrá interactuar con elementos físicos del espacio tridimensional mediante el <i>physical interactor</i> .	
<b>Precondiciones</b>	Que el elemento con el que quiera interactuar este a rango de su puño, y sea apto para el <i>physical interactor</i> .	
<b>Postcondiciones</b>	El elemento reaccionará a la interacción.	
<b>Incluye</b>	Comprar cosméticos presionando botón 3d	
<b>Extiende</b>	-	
<b>Hereda de</b>	-	
<b>Flujo de eventos</b>		
<b>Actor</b>	<b>Sistema</b>	
1. El jugador golpeará con el puño un elemento físico.	2. El elemento físico reaccionará si es apto.	

Tabla 28 - CU-27. Elaboración propia.

<b>Id</b>	CU-28	
<b>Caso de uso</b>	Comprar cosmético presionando botón 3D	
<b>Actor</b>	Usuario/Jugador	
<b>Resumen</b>	Para que el jugador pueda completar la compra del cosmético seleccionado mediante el <i>ray interactor</i> , presionará un botón 3D con su puño para adquirirlo.	
<b>Precondiciones</b>	El jugador deberá estar suficientemente cerca al botón para poder golpearlo.	
<b>Postcondiciones</b>	El botón comprará el cosmético seleccionado si hay dinero suficiente (enviando un <i>request</i> al código en <i>cloud</i> y la base de datos para lograrlo).	
<b>Incluye</b>	-	



<b>Extiende</b>	-
<b>Hereda de</b>	-
<b>Flujo de eventos</b>	
<b>Actor</b>	<b>Sistema</b>
1. El jugador golpeará el botón 3D una vez seleccionado el cosmético.	2. El botón realizará la compra mediante llamadas en la nube y restará su coste del dinero del jugador. 3. El cosmético pasará a formar parte de los adquiridos por el jugador.

*Tabla 29 - CU-27. Elaboración propia.*

## Capítulo 5. DESARROLLO DEL PROYECTO

### 5.1 Planificación del proyecto

Para planificar cada apartado del proyecto, se ha hecho uso de un diagrama de Gantt (figura 39), diferenciando en tres distintos bloques: preproducción, producción y postproducción. Dentro de esos bloques, se han dividido los sistemas primordiales implementados durante el desarrollo, así como la creación de esta memoria.

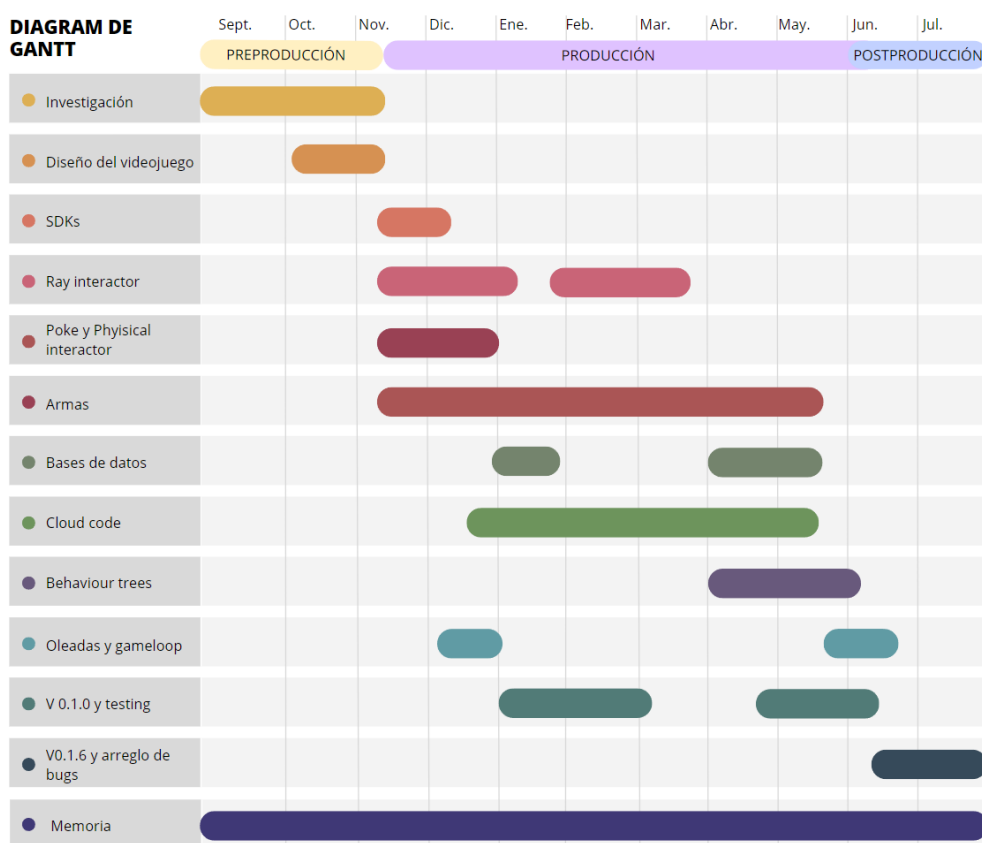


Figura 39 - Diagrama de Gantt de PistoleroVR. Elaboración propia.

Por otro lado, a fin de cumplir con los tiempos y objetivos del desarrollo, se ha aplicado la metodología ágil Kanban mediante Trello (figura 40). En él, se han añadido dos columnas adicionales, *features* y *further implementations*. Éstas son para indicar los diferentes grupos a los que pertenecen cada *task* y las implementaciones futuras deseadas tras el pmv respectivamente.

Además, se ha hecho uso continuo de github como control de versiones, donde se encuentra el proyecto. Es de acceso público (licencia Apache Licence 2.0) y se encuentra disponible en [https://github.com/somozadev/TFG\\_Informatica](https://github.com/somozadev/TFG_Informatica).

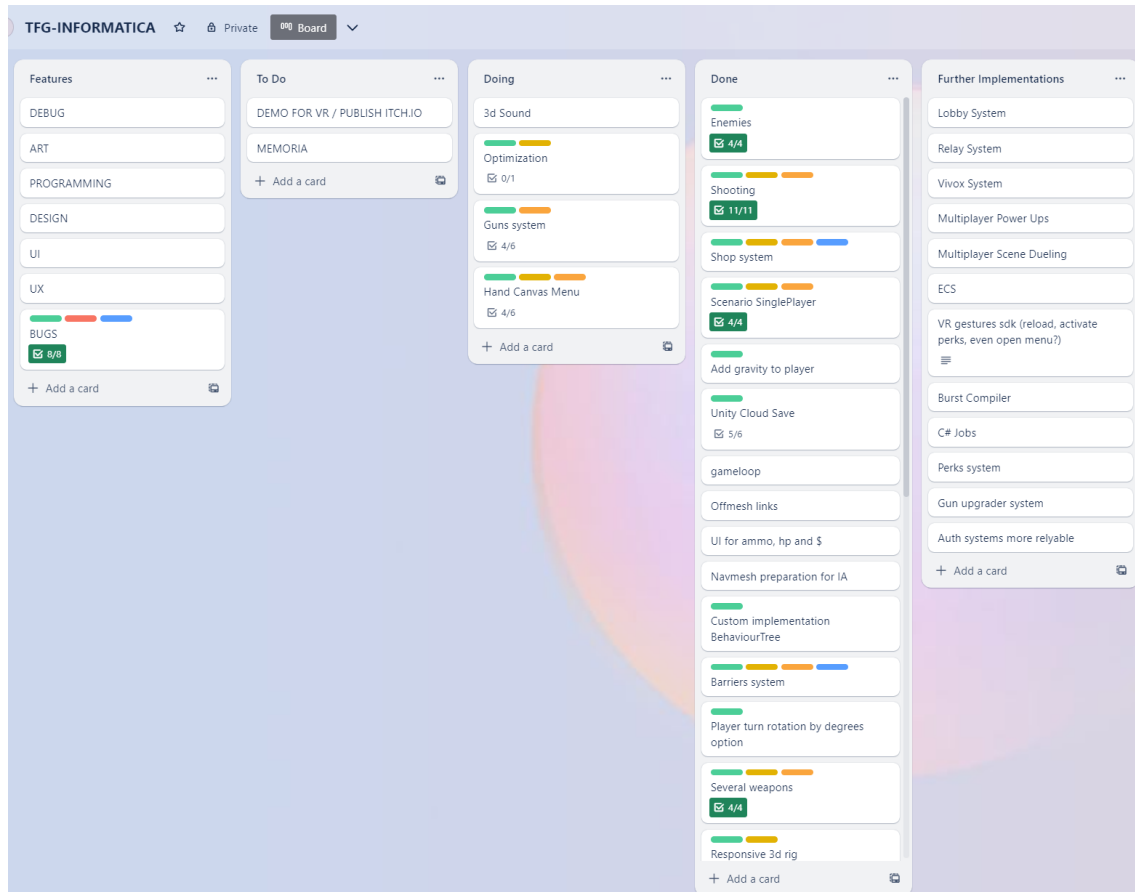
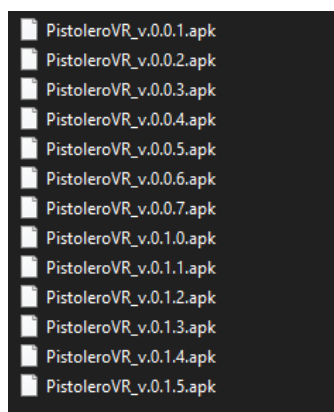


Figura 40 - Kanban del proyecto en Trello. Elaboración propia.

También mencionar el uso de la compilación por avances en la figura 41. Es decir, cada vez que se ha implementado un sistema nuevo o el proyecto ha sufrido una mejora considerable, se ha compilado con su versión correspondiente y probado sus niveles de inmersión, emoción, experiencia de usuario y jugabilidad, entre otros.



*Figura 41 - Compilaciones incrementales de PistoleroVR.  
Elaboración propia.*

## 5.2 Descripción de la solución, metodologías y herramientas empleadas

La solución que se propone es un pmv de un videojuego de género de disparos en primera persona con una experiencia de usuario satisfactoria, mediante diversos sistemas de interacción con el mundo, guardado de datos en la nube, sistema de economía y compras, y una jugabilidad dinámica e interesante para los jugadores mediante el uso de inteligencia artificial en los enemigos.

**Movimiento inmersivo:** OpenXR ofrece un *prefab* inicial de controlador para realidad virtual. Se ha optado por diseñar un sistema similar al ofrecido, pero modificando su estructura y código fuente. Además, se ha diseñado un script de movimiento para el jugador que parte del *LocomotionProvider.cs* originario de este SDK. Éste otorga la posibilidad de mover al jugador mediante una condición de llamada, es decir, si el jugador se está moviendo mediante una forma A (por ejemplo, movimiento continuo) y se quiere mover de una forma B (por ejemplo, teletransporte), la segunda deberá esperar a que la primera acabe.

La modificación se centra en el movimiento basado por velocidad de objetos físicos (rigidbody en unity) en lugar de movimiento vectorial (transforms en Unity). Este cambio no sólo otorga un movimiento más realista y por ende inmersivo, sino que también facilita la detección de colisiones con objetos. Cabe destacar la coexistencia de dos sistemas de movimiento y rotación concurrentes, uno basado en el *LocomotionProvider*, que mueve al personaje en base a los controles de los mandos del jugador. Pero el otro está basado en la posición relativa del jugador en el mundo (sus manos y HMD), que es trasladada al videojuego. Por lo tanto, se permite tanto que la persona camine y gire para que lo haga el personaje como que utilicen los controles, o ambas a la vez.

Mencionar también el uso de animación de cinemática inversa[39], donde se le dotan de “huesos” al modelo 3D del personaje del videojuego para que imiten los movimientos de los brazos cabeza y cuerpo del jugador en base a la posición de los mandos y el HMD. De esta forma se logra una inmersión total en base al movimiento real de la persona.

**Interacción inmersiva con el mundo:** Para lograr una interacción inmersiva desde todas formas posibles, se han implementado tres tipos de interacción distintas: *Ray* interactor, *poke* interactor y *physical* interactor. Cada uno se aplica a unas situaciones concretas y únicas para su tipo de interacción, lo que en conjunto crea un espacio inmersivo donde el jugador puede interactuar prácticamente con lo que quiera y de varias formas. Todas ellas implican el movimiento de las manos, el *ray* permite agarrar elementos a distancia, seleccionar elementos y interactuar con tiendas. El *poke* permite interacciones con la UI del juego, y el *physical* presionar botones 3D, así como colisionar y mover elementos con físicas (sillas, armas, vasos, botellas...). Hay que destacar que cada arma cuenta con su tipo de disparo, estadísticas y cantidad de munición propia, así como efectos de sonido y partículas.

Finalmente, para crear una mayor inmersión con el mundo, se han implementado fuentes de audio 3D, cuyo volumen y dirección sonora son determinados por el vector director entre la fuente y el jugador.

**Ejecución de código en la nube y guardado de datos:** Para realizar el guardado de datos de cada usuario único del juego, se ha utilizado tanto bases de datos en la nube como APIs o código en la nube, así como sistemas de economía. Esta solución si bien es mejorable, sienta unas buenas bases para la protección de datos (al no guardarlos de forma local), la prevención de modificaciones indebidas de los mismos y la posibilidad de persistir a lo largo del tiempo.

**Inteligencia artificial:** Desarrollando el diseño de la inteligencia artificial basada en árboles de decisión, el enemigo priorizará atacar al jugador, y si esta acción falla, intentará unirse a otros enemigos para formar una horda, y si ésta falla, intentará perseguir al jugador. Así, los enemigos están dotados de inteligencia con capacidad de decisión, una inteligencia que además de funcional es modular y puede ser incrementada en futuras mejoras o enemigos.

Es importante destacar que se han creado más nodos con clase padre *Node.cs* para gestionar cada condición y acción que se realice en el árbol. Aquellos con la palabra *Task* en el nombre son las acciones finales, y el resto las comprobaciones para llegar a ellas. De esta forma, se crea una jerarquía dentro del propio árbol que resulta más legible y sencilla de seguir.

Sin embargo, se debe tener en cuenta el entorno del que se trata, un software de ejecución en tiempo real donde priman los fps. Los árboles de decisión están constantemente revisando el estado de sus nodos, si hubiera un número considerable de enemigos actualizando sus árboles cada *frame*, el rendimiento se vería afectado. Es por eso por lo que, de momento, se ha limitado tanto el número de enemigos que pueden coexistir simultáneamente en el mundo como la forma en la que el árbol se actualiza, además de no crear instancias de enemigos (cada uno con su árbol de decisión) en tiempo real, sino hacer uso de la técnica de optimización *object pooling*.

**Optimización:** La técnica de optimización implementada que afecte al código del producto es el *object pooling*. Esta técnica crea una lista de elementos de tamaño *n* al iniciarse la aplicación. A medida que se vayan necesitando, se irán activando sucesivamente hasta que no queden elementos. Si se requieren más pero no hay elementos inactivos, se creará otro nuevo en tiempo de ejecución. Una vez el elemento activado deje de ser útil, se desactivará y volverá a la lista de

elementos activables. De esta forma, se optimiza el uso de memoria, limitando tanto la cantidad de nuevas instancias como la destrucción de éstas, obligando al recolector de basura a limpiar su espacio de memoria reduciendo así el rendimiento.

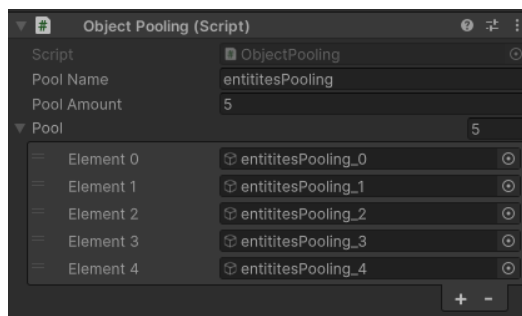


Figura 42 - Visualización en editor del object pooling.  
Elaboración propia.

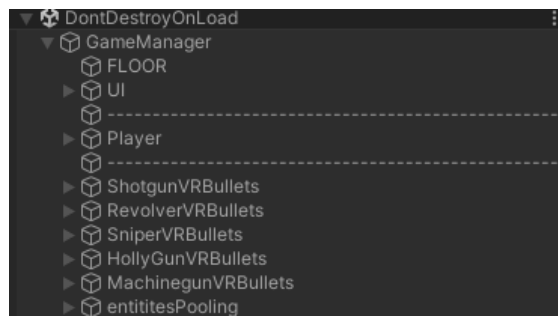


Figura 43 - Jerarquía en escena del object pooling.  
Elaboración propia.

Además, se ha creado una clase adicional para la gestión de los diferentes object poolings del proyecto; *ObjectPoolingManager*. Éste crea un diccionario donde se almacena tanto el nombre del pool como su objeto *ObjectPooling*. Al estar este gestor referenciado desde el Singleton del *GameManager*, es accesible para todas las partes del código que lo necesiten; las balas para cada tipo de arma cuentan con su propio pool, así como otro encargado de los enemigos. Tanto la visualización de la clase en editor como la jerarquía de objetos son visibles en las figuras 42 y 43 respectivamente.

En cuanto a los enemigos se refiere, además de su *object pooling* y para limitar el número de llamadas de actualización de sus árboles de decisión, se ha creado un contador que los actualiza cada 500 milisegundos, en lugar de cada *frame*. De esta forma, se reduce el número de llamadas a la mitad de su original, creando reacciones más lentas de los enemigos, pero imperceptibles para los jugadores a costa de duplicar el rendimiento.

### 5.3 Recursos requeridos

Para el desarrollo del proyecto se ha utilizado el motor de videojuegos Unity Engine, primordialmente con el lenguaje de programación C#. Dado que se trata de un videojuego en realidad virtual, también se ha hecho uso de unas gafas VR para el desarrollo y las pruebas, las Oculus Quest 2.

Gracias a algunos de los paquetes enumerados en el apartado de sdks de diseño, junto con la plataforma de Unity dashboard para la gestión del proyecto, se han realizado en ella los procesos de autenticación de usuarios, ejecución de scripts en la nube, bases de datos noSql y gestión de la economía del juego.

Finalmente, se ha hecho uso de git para el control de versiones, principalmente git bash y github desktop. Mencionar también el uso de javascript para los scripts ejecutados en la nube, y el formato de texto JSON para el guardado de datos en la nube.

## 5.4 Presupuesto

Concepto	ud (h)	Coste (€)	T
Recursos humanos			
salario trabajador	720	30,00 €	21.600,00 €
salario playtesting	5	50,00 €	250,00 €
Costes de investigación			
suscripciones a prensa especializada (medium.com)	8760	0,00 €	28,58 €
suscripciones a cursos especializados (udemy.com)	8	6,42 €	51,34 €
Costes en servidores			
costes de la base de datos	-	0,00 €	0,00 €
costes del servicio de autenticación	-	0,00 €	0,00 €
costes del servicio de economía	-	0,00 €	0,00 €
costes del servicio de cloud code	-	0,00 €	0,00 €
Costes generales de producción			
Electricidad	720	0,18 €	101,33 €
Conexión a internet	4032	0,06 €	194,10 €
Materiales directos			
modelos de escenario 3d (POLYGON Western - Low Poly 3D Art by Synty)	-	23,64 €	29,93 €
música	-	0,00 €	0,00 €
ordenador de sobremesa	-	2.765,00 €	691,25 €
oculus quest 2	-	275,71 €	275,71 €
TOTAL BRUTO			23.222,25 €
IVA		21%	4.876,67 €
<b>TOTAL NETO</b>			<b>29.471,17 €</b>

Teniendo en cuenta que:

- El salario del trabajador asciende a 30 € la hora más IVA, trabajando a media jornada durante seis meses. Teniendo en cuenta que este trabajador estará a cargo del análisis, investigación, diseño, programación, documentación e implementación de todos los aspectos del videojuego.
- Las suscripciones de prensa son de coste anual fijo. Las de cursos es un pago único.
- No existen costes de los servidores debido a su método de cobro, basado en uso. Únicamente comienzan a generar costes cuando sobrepasan las 750 mil llamadas mensuales, y dado el alcance de este proyecto, no se llegará en ningún caso para ningún servicio.
- La electricidad consta de únicamente el consumo de los elementos electrónicos utilizados para el desarrollo con un precio medio KWh de 0.12725, tales como el ordenador y sus pantallas (1.2 KWh diarios), así como el router y las oculus quest 2 (0.2KWh diarios).
- Los modelos 3D adquiridos son también de pago único. El resto de los elementos visuales y sonoros del proyecto se han desarrollado de forma interna, por lo que no incurrir en gastos de materiales directos.
- Finalmente, indicar la amortización del ordenador de sobremesa al 15% con 5 años de recorrido, lo que resulta en un coste del 25% frente a su precio original.

Y a su vez teniendo en consideración el hecho de no usar el modelo COCOMO para la estimación de costes y el cálculo de presupuesto, debido a que el único trabajador es aquel que realiza todas las tareas y este método de presupuesto basado en costes es el más realista en la industria del videojuego (especialmente a la hora de buscar financiación). Se ha interpretado de la forma en la que un trabajador autónomo (ya sea autónomo o autónomo societario dueño de una sociedad) lo facturará en la industria del videojuego. A menos costes, mayor valor de retorno y mayor facilidad de financiación.

## 5.5 Viabilidad

Teniendo en cuenta tanto los resultados obtenidos con un producto mínimo viable funcional y atractivo, el cumplimiento de los objetivos propuestos y el gran potencial tanto de mercado como de las mejoras aplicables al producto se puede afirmar que se trata de un producto viable. Si bien es cierto que un presupuesto de casi 24 mil euros parece elevado, en la industria la mayoría de los presupuestos para videojuegos independientes como lo es éste oscilan entre los 50 y los 200 mil euros. Esto indica que, al terminar y pulir el proyecto para su distribución y venta, acabará incurriendo en más gastos, ascendiendo a 40 o 50 mil euros fácilmente. Además, se debe tener en cuenta que las tiendas donde se distribuyen los productos suelen cobrar el 30%



de cada venta, por lo que en realidad como beneficios contaría el setenta por ciento de las ventas por el precio.

Además, este proyecto cuenta con una viabilidad financiera muy favorable, teniendo en consideración que el precio medio de productos del mismo género para la misma plataforma es de 25 euros. Es decir, este producto recuperará la inversión con 2000 ventas, un número realista de alcanzar sabiendo que se trata de un mercado internacional, y los efectos de márketing son extremadamente beneficiosos en esta industria.

En comparación, otros títulos como Gorilla Tag o Beat Saber, con misma experiencia de usuario satisfactoria (para su género específico de videojuegos), un estilo artístico minimalista (como lo es éste) y distribuidos en la misma plataforma donde éste se va a distribuir, han sobrepasado con creces sus expectativas de ventas, logrando niveles de ROI (retorno de inversión) estratosféricos.

Beat Saber hoy cuenta con más de 4 millones de copias vendidas a un precio de 29.99 euros. Sin haber informes oficiales, se estima que los costes de producción para este título oscilaron entre los 80 y los 100 mil euros. Asumiendo, para realizar un análisis más conservador, que la inversión ascendió a 100 mil euros, se puede observar que obtuvieron un ROI casi del cien mil por ciento.

$$ROI \rightarrow \frac{((4.000.000 \times 29,99) \times 0,7) - 100.000}{100.000} \times 100 = 83872.00\%$$

Gorilla Tag hoy en día cuenta con más de 760 mil copias vendidas a un precio de 19.50 euros. Sin tener tampoco informes oficiales, se estima que los costes de producción de este título fueron inferiores a los de Beat Saber, rondando los 60 mil euros. Asumiendo esta estimación como cierta, se puede observar que obtuvieron un ROI de casi veinte mil por ciento.

$$ROI \rightarrow \frac{((760.000 \times 19,50) \times 0,7) - 60.000}{60.000} \times 100 = 17190.00\%$$

Si bien estos ejemplos son casos extremadamente exitosos, con lograr un ROI del 400-600 % es más que aceptable y llamativo para inversores y empresas. Así pues, con una inversión de 40 mil euros, un precio de venta de 19.99 y 20 mil unidades vendidas, estaríamos hablando de un ROI de casi el seiscientos por ciento.

$$ROI \rightarrow \frac{((20.000 \times 19,99) \times 0,7) - 40.000}{40.000} \times 100 = 599.65\%$$

## 5.6 Resultados del proyecto

El resultado final de este proyecto es un fichero .apk 130 Mb, compatible con el sistema operativo Android (aquel usado por las gafas Oculus Quest 2). Además del proyecto previo a compilación, permitiendo así futuras compilaciones para otras plataformas de realidad virtual. Para con respecto los objetivos iniciales planteados, se han alcanzado los siguientes resultados:

### Arquitectura de software

Todos los objetivos de arquitectura de software definidos en la lista de objetivos se han cumplido de forma satisfactoria, destacando especialmente el uso de *Test Cases* para detectar la necesidad de que el script *GameManager.cs* se tuviera que ejecutar en primer lugar en la pila de ejecución de scripts.

### Realidad virtual

Todos los objetivos iniciales relacionados con realidad virtual, implementación de SDK's y sistemas de interacción y movimiento han sido alcanzados con éxito. Destacando la implementación de sistemas de interacción adicionales para crear un mayor nivel de inmersión (además del *RayInteractor*, se implementó el *PokeInteractor* y el *PhysicalInteractor*).

### Aspectos del videojuego

Todos los objetivos iniciales han sido implementados con éxito.

- Sobre la creación de partículas, se han creado para los disparos de las armas, los *decals* de los impactos y el sangrado de los enemigos. Además, se han implementado los sistemas de desbloqueables haciendo uso de la filosofía de la programación orientada a datos, mediante los *scriptable objects* de Unity.
- Gracias a los paquetes de modelos 3D adquiridos, se ha creado un *environment* tridimensional inmersivo en las escenas de inicio y juego.
- En cuanto a la inteligencia artificial de los enemigos, se han investigado diferentes técnicas (se hablará más de ellas en el apartado de discusión), pero se ha acabado implementando la de árboles de decisión, donde los enemigos deciden realizar una u otra rama en base a su entorno. Además de haber sido animados programáticamente, haciendo uso de cálculos de cinemática inversa para el movimiento de las piernas, y de corrutinas para los ataques de los brazos, y utilizar algoritmos de caminos basados en geometría de malla (*unity navmesh*) para moverse por el entorno.

### Optimización

En cuanto a la optimización se refiere, se han cumplido los objetivos en su mayoría.

En cuanto a la optimización de texturas y geometría, sí se ha aplicado para aquellos creados de forma propia (los enemigos, el personaje, las armas y las interfaces de usuario), pero para

aquellos modelos 3D adquiridos para el diseño de los escenarios, no se han aplicado optimización de geometría y texturas. Esto es debido a que, para un pmv donde no prima el arte, no se ha considerado. Sin embargo, sí se considera para las futuras líneas de trabajo.

Por otro lado, desde el aspecto técnico del código del videojuego, sí se han cumplido todos los objetivos, desde el *object pooling* implementado para las balas de las armas y los enemigos hasta la reducción de llamadas recurrentes cada *frame* (en el método *Update*).

Finalmente, se han de forma exitosa realizado técnicas de optimización para la renderización en tiempo real del videojuego, tales como *baking* de luces (generar las texturas de la iluminación de las escenas durante el desarrollo para suplantar las generadas en tiempo real, que requieren de más cálculos y consumo) o el *occlusion culling* (técnica que desactiva la renderización de aquellos objetos de la escena que no estén en el ángulo de visión de la cámara).

## Capítulo 6. CONCLUSIONES

### 6.1 Conclusiones del trabajo

Se ha conseguido desarrollar desde cero un producto mínimo viable de un videojuego de disparos en primera persona, con oleadas progresivas de enemigos inteligentes, sistema de cosméticos y guardado de datos completo. Que ofrece a los jugadores una experiencia inmersiva, satisfactoria de jugar y evocadora de emociones.

Además de esto, se trata de una serie de métodos y técnicas de desarrollo y optimización para videojuegos VR documentadas y de libre acceso, a fin de promover y fomentar el desarrollo de videojuegos (especialmente aquellos considerados independientes) para realidad virtual.

### 6.2 Conclusiones personales

El desarrollo de este proyecto ha supuesto un nivel de esfuerzo personal desmesurado. A fin de cuentas, los videojuegos son productos software extremadamente complejos, que requieren de equipos multidisciplinares, talentosos y numerosos para poder crear un producto digno.

Cabe destacar que ya contaba con extenso conocimiento previo sobre el desarrollo de videojuegos (especialmente sobre la programación de estos, aunque también nociones de diseño, sonido, animación, modelado 3d...), pero nunca había desarrollado para VR. Ha requerido una extensa investigación sobre el funcionamiento de los sistemas de seguimiento de posición de los mandos y gafas como la implementación de controles cómodos para mover al personaje, lo cual ahora me brinda tanto a mí como a quien desee aprender sobre ello mediante este escrito y este proyecto, el conocimiento necesario. Sumado claro a la optimización que requieren videojuegos en esta plataforma. Acostumbrado a desarrollar para PC, donde los requerimientos mínimos del sistema sobrepasan con creces los que disponen los HMD, no habían requerido de técnicas de optimización de ningún tipo.

También hay que destacar la utilización de *Unity Gaming Services*, nunca utilizado. Ha resultado no sólo ser un medio rico en documentación del que poder aprovechar funcionalidades, como las bases de datos, el sistema de economía, la autenticación de usuario o el código *cloud* (además del resto de utilidades ofertadas ahí), sino también un medio eficiente y útil, aplicable a infinidad de proyectos de videojuegos, ya no sólo de VR.

## Capítulo 7. FUTURAS LÍNEAS DE TRABAJO

En cuanto a las futuras líneas de trabajo, predominan tres aspectos clave. El apartado artístico, la optimización y el contenido.

Para el apartado visual, se deberán cambiar los modelos de los enemigos a unos modelados por un artista 3D, que cree unos modelos que estén alineados con el estilo artístico del producto como conjunto. También, y de la mano de optimizar el juego, se deberán remodelar los elementos del escenario para reducir en número de geometría de los modelos, así como reasignar sus texturas para utilizar menos cantidad y resolución, mientras se otorga el estilo artístico *lowpoly* que se intenta lograr.

En cuanto a la optimización se refiere, hay varias mejoras. Empezando por el arte, como se ha mencionado en el apartado artístico, se debería reducir la cantidad de polígonos y texturas de los escenarios. Además, al hacer dicha optimización se abren las puertas a aplicar *static* y *dynamic batching*, reducir *draw calls* y realizar instancias de GPU. Por otro lado, y lo más importante después de la optimización visual, el uso de Unity Dots, Jobs y burst compiler. Básicamente, una implementación funcional de programación multihilo (jobs) basada en la programación orientada a datos (dots), gracias a compilador burst (compilador de alto rendimiento diseñado para Unity que optimiza el código C# para un procesamiento más rápido en CPUs). No se ha hecho uso de estas tecnologías porque se encontraban en estado de “experimental”, y Unity no recomienda el uso de tecnologías con dicha etiqueta para productos dispuestos a la venta. Esta etiqueta pasó a “release” cuando el desarrollo del pmv ya había terminado (19/6/2023) y era imposible migrar a dicha tecnología por falta de tiempo.

Por otro lado, sobre la experiencia de juego se refiere, se han diseñado elementos que deberían encontrarse en el proyecto finalizado previo venta. Desde la implementación de multijugador en línea para jugar con hasta 4 jugadores simultáneos por partida, implementación de más cosméticos, armas y escenarios o más tipos de enemigos y comportamientos. Además, la implementación de inicio de sesión con Google, play games y oculus será necesaria. Mencionar también como pendiente la mejora e implementación de efectos sonoros, como los pasos de los enemigos, o ajustar mejor las distancias de propagación de los sonidos 3D (aquellos cuya fuente es una distinta al jugador).

Finalmente, migrar la publicación del proyecto de donde se encuentra ahora a una tienda oficial, como meta store o steam, con su correspondiente *landing page*, vídeos promocionales, márketing y precio. Para ello, se deberá formalizar la constitución de una sociedad con personalidad jurídica capaz de recibir pagos por el producto.

## Capítulo 8. REFERENCIAS

- [1] Meta Quest, “Testing and Performance Analysis | Oculus Developers,” 2022. <https://developer.oculus.com/documentation/unity/unity-perf/> (accessed Jul. 05, 2023).
- [2] Valve, “Títulos de realidad virtual,” 2023. <https://store.steampowered.com/vr?l=spanish&flavor=popularcomingsoon&facets13268=3%3A26> (accessed Jul. 05, 2023).
- [3] Redacción, “Steam: el número de usuarios activos con VR logra nuevo máximo histórico y Quest 2 tiene ya más del 50% de cuota,” 2022. <https://www.realovirtual.com/noticias/11691/steam-numero-usuarios-activos-vr-logra-nuevo-maximo-historico-quest-2-tiene-ya-mas-del-50-cuota> (accessed Jul. 05, 2023).
- [4] Beat Games, “Beat Saber en Oculus Rift | Oculus,” May 01, 2018. <https://www.oculus.com/experiences/rift/1304877726278670/> (accessed Jul. 05, 2023).
- [5] Valve, “Half-Life: Alyx en Steam,” Mar. 23, 2020. [https://store.steampowered.com/app/546560/HalfLife\\_Alyx/](https://store.steampowered.com/app/546560/HalfLife_Alyx/) (accessed Jul. 05, 2023).
- [6] Innersloth LLC, “Among Us VR en Oculus Quest 2 | Oculus,” Nov. 10, 2022. [https://www.oculus.com/experiences/quest/4948428055244413/?locale=es\\_ES](https://www.oculus.com/experiences/quest/4948428055244413/?locale=es_ES) (accessed Jul. 05, 2023).
- [7] Another Axiom, “Gorilla Tag en Steam,” Jan. 01, 2023. [https://store.steampowered.com/app/1533390/Gorilla\\_Tag/](https://store.steampowered.com/app/1533390/Gorilla_Tag/) (accessed Jul. 05, 2023).
- [8] Meta, “Meta Quest 2: gafas inmersivas de realidad virtual todo en uno | Meta Store | Meta Store,” 2023. <https://www.meta.com/es/quest/products/quest-2/> (accessed Jul. 05, 2023).
- [9] Alejandro VR, “Quest 2 tiene más de 6 millones de usuarios activos mensualmente,” 2023. <https://alehandorovr.com/quest-2-tiene-mas-de-6-millones-de-usuarios-activos-mensualmente/> (accessed Jul. 05, 2023).
- [10] José Ariel Román, “CLASIFICACIÓN de GÉNEROS de VIDEOJUEGOS - GameOverLA.com,” Jun. 04, 2020. <https://www.gameoverla.com/clasificacion-de-generos-de-videojuegos.html> (accessed Jul. 05, 2023).
- [11] D. T. Fernández, E. B. Moya, and R. P. Sánchez, “Immersion and emotional arousal with virtual reality videogames,” *Revista de Psicología (Peru)*, vol. 39, no. 2, 2021, doi: 10.18800/PSICO.202102.002.
- [12] S. M. Bender and B. Sung, “Fear, attention, and joy while killing zombies in Virtual Reality: A psychophysiological analysis of VR user experience,” *Psychol Mark*, vol. 38, no. 6, 2021, doi: 10.1002/mar.21444.
- [13] Simon Read, “Los videojuegos están en auge y se espera que la industria siga creciendo. | Foro Económico Mundial,” Sep. 2022. <https://es.weforum.org/agenda/2022/09/el-juego-esta-en->

auge-y-se-espera-que-siga-creciendo-este-grafico-le-dice-todo-lo-que-necesita-saber/  
(accessed Jul. 04, 2023).

- [14] IOCA Group, “Unos audífonos de calidad son el accesorio indispensable para un ‘gamer.’” <https://iocagroup.com/unos-audifonos-de-calidad-son-el-accesorio-indispensable-para-un-gamer/ioca-group> (accessed Jul. 05, 2023).
- [15] C. M. Jones, L. Scholes, D. Johnson, M. Katsikitis, and M. C. Carras, “Gaming well: Links between videogames and flourishing mental health,” *Front Psychol*, vol. 5, no. MAR, 2014, doi: 10.3389/fpsyg.2014.00260.
- [16] A. Martínez Cuello, B. Serra Soriano, E. Piquer Maño, P. García Romero, C. Ribes Vallés, and M. I. Lloria Benet, “Optimización y mejora del aprendizaje mediante la utilización de la realidad virtual en las prácticas de grados y ciclos formativos,” 2020. doi: 10.4995/inred2020.2020.11975.
- [17] Claudio Servin, “Sega VR | IDIS,” 1994. <https://proyectoidis.org/sega-vr/> (accessed Jul. 05, 2023).
- [18] Nintendo fandom, “Virtual Boy | Nintendo Wiki | Fandom,” 2018. [https://nintendo.fandom.com/es/wiki/Virtual\\_Boy](https://nintendo.fandom.com/es/wiki/Virtual_Boy) (accessed Jul. 05, 2023).
- [19] G. Lara, A. Santana, A. Lira, and A. Peña, “El Desarrollo del Hardware para la Realidad Virtual,” *RISTI - Revista Ibérica de Sistemas e Tecnologías de Informação*, no. 31, pp. 106–117, Jan. 2019, doi: 10.17013/risti.31.106-117.
- [20] A. da Silva Marinho, U. Terton, and C. M. Jones, “Cybersickness and postural stability of first time VR users playing VR videogames,” *Appl Ergon*, vol. 101, 2022, doi: 10.1016/j.apergo.2022.103698.
- [21] G. Geršak, H. Lu, and J. Guna, “Effect of VR technology matureness on VR sickness,” *Multimed Tools Appl*, vol. 79, no. 21–22, 2020, doi: 10.1007/s11042-018-6969-2.
- [22] Carlos Quevedo, “¿Qué es FPS? | PcPedia | Blog de PcComponentes,” May 04, 2023. <https://www.pccomponentes.com/que-es-fps> (accessed Jul. 05, 2023).
- [23] Meta quest, “VRC.Quest.Performance.1 | Oculus Developers,” 2022. <https://developer.oculus.com/resources/vrc-quest-performance-1/> (accessed Jul. 05, 2023).
- [24] Christian Fernández Suárez, “EXTERNAL ANALYSIS OF VIDEOGAMES INDUSTRY AND INTERNAL ANALYSIS OF COMPANY ‘RIOT GAMES,’” 2016.
- [25] Nuria Estruga, “Conoce las mejores gafas de Realidad Virtual | Blog de PcComponentes,” Feb. 24, 2022. <https://www.pccomponentes.com/mejores-gafas-de-realidad-virtual> (accessed Jul. 05, 2023).
- [26] BBVA, “Realidad virtual o cómo se cumplen los sueños de la ciencia-ficción,” 2015.
- [27] Alejandro VR, “Meta Reality: marca los inicios para la Realidad Mixta de Meta,” Dec. 19, 2022. <https://alehandorovr.com/meta-marca-sus-inicios-para-la-realidad-mixta/> (accessed Jul. 05, 2023).

- [28] Sergio Hidalgo, “Profundidad y Riqueza Visual,” 2017. <https://www.realovirtual.com/articulos/5029/profundidad-riqueza-visual> (accessed Jul. 05, 2023).
- [29] Olorama Technology, “Ver, oír, sentir y última novedad en realidad virtual: oler. | Olorama Technology,” 2023. <https://www.olorama.com/es/ver-oir-y-oler> (accessed Jul. 05, 2023).
- [30] Psico Smart Apps S.L, “Cómo la realidad virtual puede ayudar a pacientes a conectar y expresar sus emociones en terapia - Amelia Virtual Care,” 2022. <https://ameliavirtualcare.com/es/como-la-realidad-virtual-puede-ayudar-a-pacientes-a-conectar-y-expresar-sus-emociones-en-terapia/> (accessed Jul. 05, 2023).
- [31] Universidad Europea de Madrid, “¿Qué es un motor de juegos? | Blog UE,” 2022. <https://universidadeuropea.com/blog/motor-juegos/> (accessed Jul. 05, 2023).
- [32] Yiğit Kemal Erinc, “Los principios SOLID de programación orientada a objetos explicados en Español sencillo,” Nov. 28, 2022. <https://www.freecodecamp.org/espanol/news/los-principios-solid-explicados-en-espanol/> (accessed Jul. 05, 2023).
- [33] Jiménez A. and Luque J., “Máquinas de Estado y Videojuegos,” Universidad de Sevilla, Sevilla, 1994.
- [34] Miguel Arturo Martínez Gutiérrez, “Juego de Estrategia en Tiempo Real con Agentes Inteligentes y Planificación,” 2021. <https://riunet.upv.es/bitstream/handle/10251/173699/Martinez%20-%20Juego%20de%20estrategia%20en%20tiempo%20real%20con%20agentes%20inteligentes%20y%20planificacion.pdf?sequence=1> (accessed Jul. 05, 2023).
- [35] Policonomics, “Teoría de juegos I: Forma extensiva - Policonomics,” 2017. <https://policonomics.com/es/lp-teoria-juegos1-forma-extensiva/> (accessed Jul. 05, 2023).
- [36] V. I. Musat, “Redes neuronales aplicadas a videojuegos y estudio de sus mecánicas,” 2022, Accessed: Jul. 05, 2023. [Online]. Available: <https://oa.upm.es/71282/>
- [37] Samir Awad Núñez, “Técnicas utilizadas en la planificación (3). Algoritmos de caminos mínimos - Urbanismo y Transporte,” Apr. 14, 2016. <http://urbanismoytransporte.com/tecnicas-utilizadas-la-planificacion-3-algoritmos-caminos-minimos/> (accessed Jul. 06, 2023).
- [38] Unity Game Services, “PistoleroVR | Descripción general | Unity Gaming Services,” 2023. <https://dashboard.unity3d.com/gaming/organizations/1470729/projects/6f1b61b2-7389-4d7e-bb36-1161b64e5381/overview?viewMode=project> (accessed Jul. 05, 2023).
- [39] Markus Buchholz, “Inverse Kinematics Solver in C++. In this article I will demonstrate you... | by Markus Buchholz | Geek Culture | Medium,” Jan. 13, 2022. <https://medium.com/geekculture/inverse-kinematics-solver-in-c-e999f1b7f353> (accessed Jul. 07, 2023).



## Capítulo 9. ANEXOS

### 9.1 Glosario

- **Prefab:** En el contexto de los videojuegos, un prefab (prefabricado) se refiere a un objeto o conjunto de elementos predefinidos que se pueden utilizar para construir escenas o niveles. Los prefabs son una forma eficiente de reutilizar contenido y agilizar el desarrollo, ya que permiten crear instancias de objetos ya configurados y listos para usar en el juego.
- **Frame:** En los videojuegos, el término "frame" se refiere a un cuadro individual de animación en una secuencia de imágenes en movimiento. Los frames se utilizan para representar la progresión de la animación y la interacción visual en el juego, lo que crea la ilusión de movimiento fluido.
- **Task:** una representación de una operación asincrónica que puede ejecutarse en segundo plano sin bloquear el hilo principal de ejecución. Se utiliza para realizar tareas intensivas en términos de procesamiento o bloqueantes, como llamadas a servicios web o acceso a bases de datos. Las tareas se crean con el constructor Task y se pueden controlar y monitorear utilizando métodos de continuación. Proporcionan una forma estructurada de trabajar con operaciones asincrónicas, permitiendo que otras partes del programa sigan ejecutándose mientras se realiza la tarea en segundo plano.
- **Xr Origin:** El XR Origin (Origen de XR) se refiere al punto de referencia o posición base en la realidad extendida (XR), que engloba tecnologías como la realidad virtual (VR) y la realidad aumentada (AR). Este origen establece el punto de partida en el espacio virtual y es utilizado para rastrear y ubicar la posición y movimiento del jugador en el mundo virtual.
- **Tracking (Seguimiento en realidad virtual):** En el contexto de la realidad virtual (VR) y los videojuegos, el tracking se refiere al proceso de capturar y registrar la posición y movimiento del jugador dentro del entorno virtual. Utilizando sensores y dispositivos de rastreo, como cámaras, sensores de movimiento o controladores especializados, se puede realizar un seguimiento preciso de la ubicación y orientación del jugador en el mundo virtual. Esto permite una inmersión más realista y precisa, ya que los movimientos del jugador se reflejan en el entorno virtual.
- **Ray Interactor (Interactuador de rayos):** En el contexto de la realidad virtual y los videojuegos, un *ray interactor* es una herramienta o componente que permite al jugador interactuar con objetos virtuales mediante rayos virtuales. Al apuntar con un controlador o dispositivo de entrada, se emite un rayo que puede detectar colisiones con objetos en el entorno virtual. Esto se utiliza para realizar acciones como seleccionar, agarrar o manipular objetos virtuales en el juego, brindando una experiencia de interacción más inmersiva y precisa.
- **Poke Interactor (Interactuador de toque):** En la realidad virtual y los videojuegos, un poke interactor es un componente o función que permite al jugador interactuar táctilmente con el entorno virtual. A través de controladores o dispositivos de entrada,

el jugador puede tocar o hacer gestos táctiles en objetos virtuales para activar respuestas o realizar acciones específicas en el juego. Esto puede incluir tocar objetos para obtener información adicional, activar interruptores o interactuar con elementos interactivos del entorno virtual.

- **Physical Interactor (Interactuador físico):** En el contexto de la realidad virtual y los videojuegos, un *physical interactor* es un componente o sistema que simula y gestiona las interacciones físicas entre los objetos virtuales en el entorno del juego. Esto implica simular propiedades físicas como la gravedad, colisiones, fricción y movimiento realista. Los *physical interactors* permiten al jugador interactuar con los objetos virtuales de manera similar a como lo haría en el mundo real, brindando una experiencia de juego más inmersiva y realista.
- **Delegates:** los delegados son estructuras o mecanismos utilizados para manejar y comunicar eventos y acciones entre diferentes objetos o sistemas. Permiten que un objeto transmita la responsabilidad de una acción o evento a otro objeto, lo que facilita la modularidad y la interacción entre diferentes componentes del juego.
- **Gameplay:** El gameplay (jugabilidad) es el término que se utiliza para describir la experiencia de juego en general, incluyendo la mecánica, los controles, las reglas y la interacción del jugador con el juego. Se refiere a la forma en que el jugador se involucra y participa activamente en el juego, experimentando sus elementos y sistemas.
- **Gameloop:** El gameloop (bucle de juego) se refiere al ciclo principal de un juego, donde se ejecutan las diversas funciones y se actualiza el estado del juego. Esto incluye la actualización de la lógica del juego, la representación gráfica, la gestión de eventos y la interacción con el jugador. El gameloop es fundamental para el funcionamiento y flujo continuo del juego.
- **Poke:** En el contexto de la realidad virtual y los videojuegos, "*poke*" se refiere a una acción o gesto realizado por el jugador al tocar o interactuar físicamente con objetos virtuales utilizando dispositivos de entrada, como controladores o dispositivos táctiles. Puede implicar tocar un objeto con el controlador, empujar un objeto virtual o realizar un gesto específico para activar una respuesta en el juego. El "*poke*" se utiliza para brindar una experiencia de interacción táctil y realista en el entorno virtual, permitiendo al jugador explorar y manipular objetos virtuales de manera más inmersiva.
- **Sideloadig:** La práctica de instalar y ejecutar aplicaciones o juegos en dispositivos móviles o consolas de videojuegos sin utilizar las tiendas oficiales de aplicaciones. Esto permite a los usuarios acceder a contenido de terceros o no disponible en las tiendas oficiales, generalmente mediante la instalación manual de archivos APK (Android) o archivos IPA (iOS) en el dispositivo.
- **Corutina:** Una funcionalidad que permite ejecutar secuencias de código de manera controlada y con pausas en el proceso. Se utilizan para manejar tareas asíncronas, animaciones, efectos visuales, esperas o cualquier acción que requiera un flujo no lineal. Las corutinas se definen como métodos que se marcan con la palabra clave "*yield return*" y pueden ser pausadas o reanudadas en puntos específicos mediante la utilización de comandos como "*yield return WaitForSeconds*" para esperar un tiempo determinado. Esto permite implementar comportamientos más complejos y temporizados en los

juegos de Unity sin bloquear el hilo principal de ejecución y sin necesidad de estructuras más complejas como tareas o hilos.

- **Lowpoly:** Estilo artístico o técnica de representación visual que se caracteriza por el uso de polígonos de baja cantidad y detalles simplificados en la creación de modelos 3D. Los objetos y entornos en un juego de estilo *lowpoly* están diseñados con una cantidad limitada de polígonos, lo que resulta en formas geométricas más básicas y suaves en lugar de detalles realistas. Este enfoque estilizado y minimalista busca crear una estética visual distintiva, a menudo con colores planos y contrastantes, que evoca una sensación retro o nostálgica. El estilo *lowpoly* es popular en juegos indie, juegos móviles y juegos con temáticas simples, y puede ofrecer un rendimiento más eficiente en términos de recursos de hardware.
- **Static batching:** Técnica de optimización que combina varios objetos estáticos en una sola malla en tiempo de compilación. Cuando se activa el *static batching* para un conjunto de objetos estáticos en una escena, Unity los agrupa y los fusiona en una sola malla durante el proceso de compilación. Esto reduce la cantidad de llamadas de renderizado y permite que el motor de renderizado procese los objetos combinados de manera más eficiente, lo que mejora el rendimiento general del juego. El *static batching* es especialmente útil para objetos que no cambian de posición, rotación o escala durante la ejecución del juego, como edificios, paisajes o elementos de nivel estáticos. Sin embargo, no es efectivo para objetos dinámicos o que se mueven constantemente, ya que el proceso de combinación debe repetirse cada vez que se realiza un cambio en dichos objetos.
- **Dynamic batching:** Técnica de optimización que combina múltiples objetos con el mismo material y configuración en una sola llamada de renderizado, reduciendo así la cantidad de *draw calls* necesarias. Esto ayuda a mejorar el rendimiento al minimizar la carga en la CPU y la comunicación con la GPU durante el proceso de renderizado.
- **Draw calls:** Instrucciones enviadas a la GPU para dibujar un objeto o una malla en la pantalla. Cada *draw call* representa una llamada individual a la GPU, y un alto número de *draw calls* puede impactar negativamente el rendimiento del juego. Por lo tanto, reducir la cantidad de *draw calls* es esencial para mejorar la eficiencia del rendimiento.
- **GPU instantiation (instanciación en la GPU):** Técnica que permite generar múltiples instancias de un objeto o una malla directamente en la GPU, en lugar de realizar copias en la CPU. Esto es especialmente útil cuando se necesitan crear múltiples copias idénticas de un objeto, como instancias de árboles o partículas, ya que reduce la carga en la CPU y permite un procesamiento más rápido en la GPU.
- **Unity DOTS (Data-Oriented Technology Stack):** Conjunto de herramientas y tecnologías que permiten aprovechar al máximo el rendimiento y la eficiencia de los sistemas modernos, como CPUs multihilo y GPUs. DOTS se centra en la programación orientada a datos y utiliza el ECS (Entity Component System) para organizar y gestionar los datos del juego, mejorando así el rendimiento y facilitando la escalabilidad de los sistemas.
- **Jobs:** Concepto en Unity que se refiere a la ejecución paralela de tareas en hilos de CPU separados. Permite realizar cálculos intensivos de manera eficiente distribuyendo la

carga de trabajo en varios núcleos de CPU, lo que mejora el rendimiento y la velocidad de procesamiento en comparación con un enfoque secuencial.

- **Burst compiler:** Compilador de alto rendimiento diseñado para Unity que optimiza el código C# para un procesamiento más rápido en CPUs. Utiliza técnicas de compilación just-in-time (JIT) y de optimización de bajo nivel para generar código altamente eficiente, lo que puede resultar en mejoras significativas de rendimiento en comparación con la ejecución estándar de scripts en C#. El *Burst Compiler* se utiliza especialmente en combinación con Jobs para lograr un rendimiento óptimo en tareas de procesamiento intensivo.
- **Landing page:** Página web diseñada específicamente para captar la atención de los visitantes y convertirlos en clientes potenciales o realizar una acción deseada. Es una página de aterrizaje donde los usuarios son redirigidos desde anuncios, campañas de marketing o enlaces específicos. Las *landing pages* están diseñadas de manera concisa y enfocadas en un objetivo particular, generalmente con un formulario de contacto o llamado a la acción prominente.

## 9.2 Manual de usuario

Para entender cómo jugar a este videojuego, se deben conocer sus controles básicos, véase la siguiente figura. Donde los *triggers* de *shoot/select* y *grab* solamente aplicarán con el *ray interactor* del jugador.

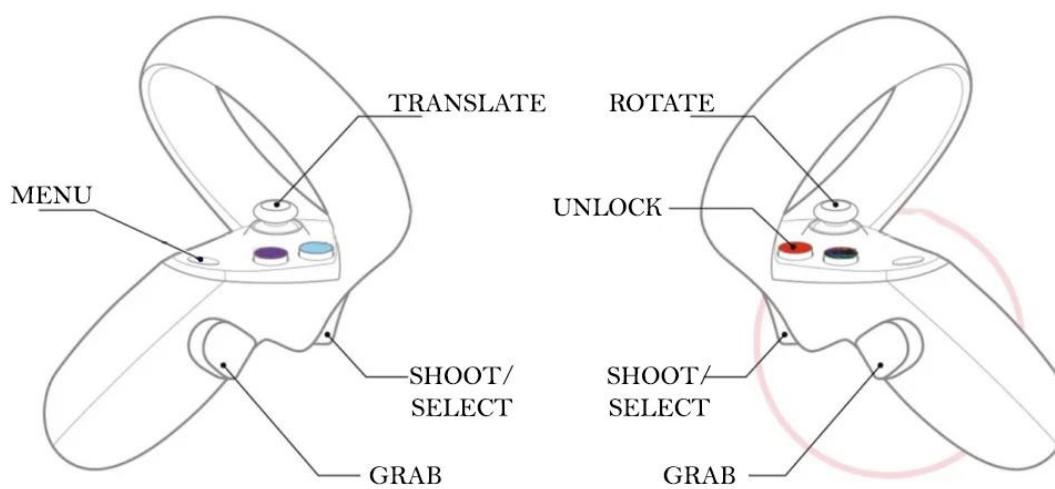


Figura 44 - Esquema de controles de PistoleroVR. Elaboración propia.

En cuanto al uso de las interfaces de usuario y los elementos físicos interactivos, se basa en movimientos de toque o golpeo con la mano física, son intuitivos. De hecho, el juego está diseñado para aprender a controlar al personaje y conocer sus armas previo al ciclo jugable en sí. En cuanto a este, una vez el jugador de un toque al botón de “singleplayer” se teletransportará a la escena de juego. En el momento en el que recoge el arma inicial del suelo (con el botón *grab* de cualquiera de sus mandos) comenzará una cuenta atrás de 10 segundos para que comience la primera oleada, como se puede ver en la figura 45. Irán apareciendo enemigos que deberá disparar (shoot/select) (figura 46), una vez elimine a todos los enemigos, avanzará a la siguiente oleada y así sucesivamente hasta que lo abatan.



Figura 45 - Contador de comienzo de partida en PistoleroVR. Elaboración propia.



Figura 46 - Enemigo atacando al jugador en pistolero VR. Elaboración propia.

Cabe destacar que el jugador cuenta en el brazo derecho de su personaje con una UI no interactuable que le informará de su vida actual, así como de su economía de esa partida. La economía la deberá usar para comprar munición, comprar diferentes armas y desbloquear zonas del mapa. Una vez el jugador muera, verá una UI con las estadísticas de la partida que estaba jugando y la opción de volver al escenario inicial, se le recompensará con la cantidad de oleadas que haya superado en formato de economía para cosméticos.

Y en cuanto al manual de usuario se refiere, debido a que el juego está diseñado para ser explicado por sí solo al jugarlo, no queda mucho más que añadir.

### 9.3 Manual de instalación

Para que el fichero .apk se encuentre disponible con su propia página de tienda en Oculus meta store y Steam, primero debería dejar de ser un pmv y evolucionar a un producto final. Es por ello por lo que se ha subido el fichero para su descarga gratuita en itch.io. Esta página web cuenta con el mismo concepto de Marketplace de videojuegos, donde la publicación de títulos es gratuita y se pueden publicar cualquier tipo de archivos; es comúnmente utilizada por desarrolladores independientes.

Es por ello por lo que el fichero se ha publicado en la dirección <https://somozadev.itch.io/pistolerovr> y es desde ahí desde donde se debe descargar el fichero apk.

Una con unas gafas Oculus en posesión (si bien el fichero está compilado para poder ser utilizado por el resto de los dispositivos VR existentes, este manual de instalación se centrará en *Oculus*. Para instalar en otros dispositivos se recomienda seguir los manuales de instalación de programas de orígenes desconocidos propios de cada hardware) se deberán seguir los siguientes pasos para instalar el fichero, proceso conocido como *sideloading*:

1. Activar la cuenta de modo desarrollador en el dispositivo. Para ello, se deberá iniciar sesión con la cuenta de *oculus* en <https://dashboard.oculus.com/>, y crear una nueva organización dándole click en el portal de *myApps*.
2. Abrir la aplicación de *oculus* móvil, acceder a las gafas donde se quiera instalar desde la pestaña de dispositivos y activar en él el modo desarrollador.
3. Instalar en el PC el programa *sidequest*, desde <https://sidequestvr.com/>.
4. Descargar el fichero .apk desde la página <https://somozadev.itch.io/pistolerovr>.
5. Conectar las gafas VR al pc mediante un cable usb-c y abrir el programa de *sidequest*.
6. Aceptar la advertencia de confiar en el dispositivo desde las gafas VR.



7. Finalmente, arrastrar el fichero. apk a *sidequest*, éste se encargará automáticamente de instalar el programa en el dispositivo. Una vez instalado, se puede cerrar y desconectar las gafas.

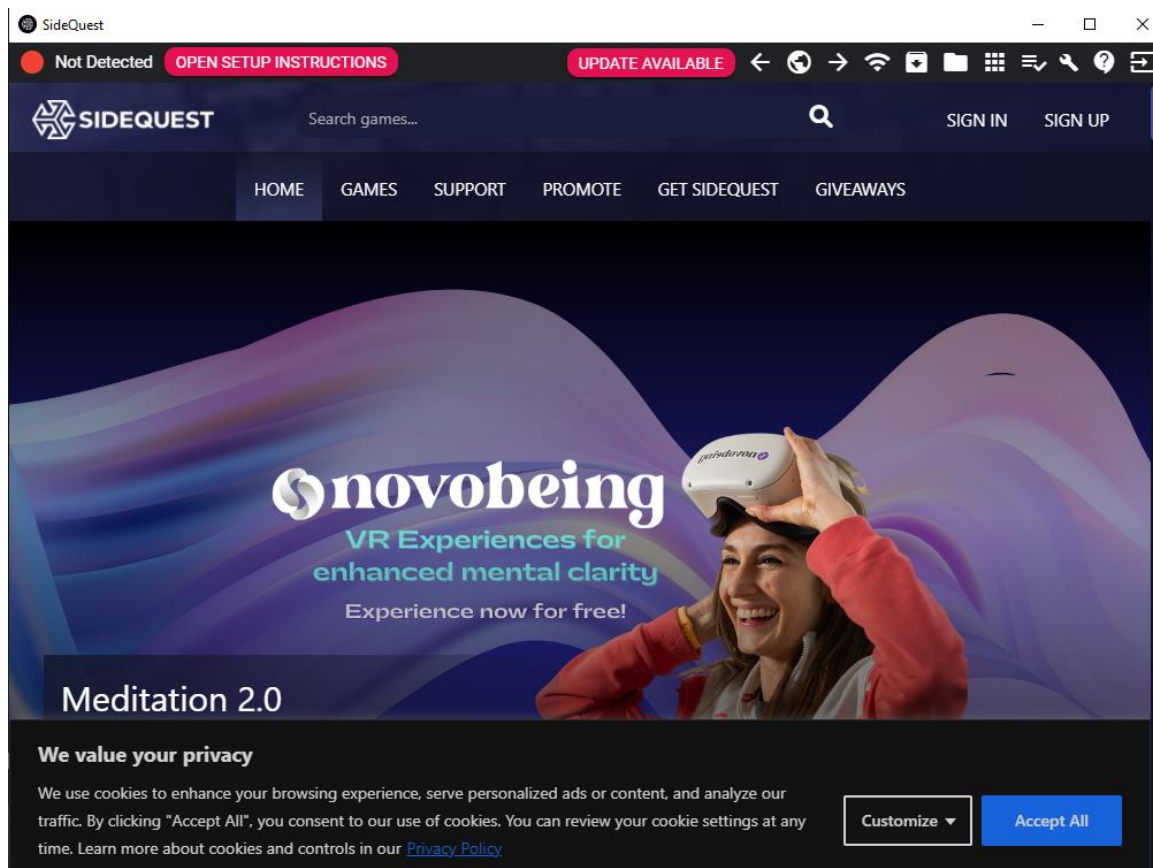


Figura 47 - Aplicación de escritorio sidequest. Elaboración propia

Finalmente, para poder jugar al juego, es importante dentro de las gafas en el apartado de aplicaciones, filtrar por orígenes desconocidos. Una vez hecho, el nombre del apk PistoleroVR debería aparecer, al abrirlo se abrirá el videojuego.

## 9.4 Resultados de encuestas

Para probar la experiencia de usuario en el videojuego, se realizaron sesiones de juego de prueba (más conocidos como sesiones de playtesting), donde además de tomar nota de las respuestas de los jugadores, estos rellenaron formularios con preguntas específicas sobre el videojuego. Si bien las respuestas o se han implementado o están ya pendientes como futuras líneas de trabajo, el contenido general con el pmv es positivo, como se ve reflejado en las encuestas:

Crees que el movimiento del personaje es adecuado?

 Copiar

4 respuestas



Figura 48 - Pregunta 1 de encuesta de UX. Elaboración propia.

Da tu opinión sincera sobre el sistema de interacción con el mundo (ray, poke, press)

4 respuestas

Al nunca haber usado VR, me resultaba confuso al principio que botones presionar para coger los objetos. A veces tiraba las armas sin querer, hasta que me he acostumbrado probándolas en el escenario principal.

Bien integrado, me gustas el color del rayo

Creo que es correcta, me gustaría más respuesta cuando estoy suficientemente cerca.

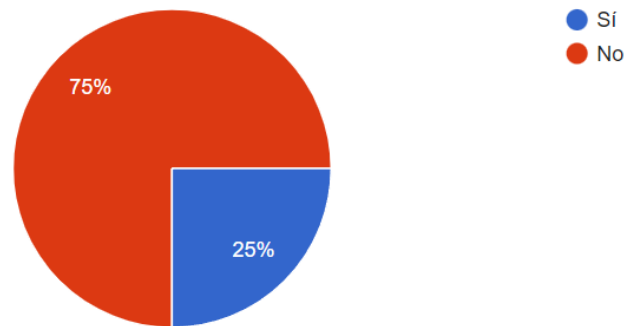
Preferiría que los menús se manejaran con el rayo en vez de con puñetazos, pero es algo más personal

Figura 49 - Pregunta 2 de encuesta de UX. Elaboración propia.



Crees que la progresión de dificultad en los enemigos es demasiado pronunciada?

4 respuestas



*Figura 50 - Pregunta 3 de encuesta de UX. Elaboración propia.*

En caso de no gustar la forma de funcionamiento de compra de armas y desbloqueo de caminos, qué harías para mejorarlo ?

3 respuestas

Que los costes suben demasiado si comparas los caminos

Simplemente indicaría de alguna forma cuando se compra armas y cuando se compra munición

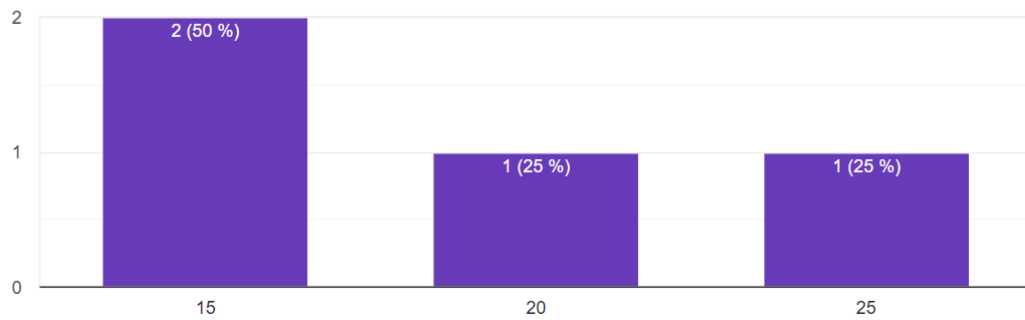
Añadir un tutorial de cómo rellenar munición

*Figura 51 - Pregunta 4 de encuesta de UX. Elaboración propia.*

Si tuvieras que decidir el precio de este juego terminado para la oculus store y steam, (recordando que ahora es un pmv), qué precio pondrías ?

 Copiar

4 respuestas



*Figura 52 - Pregunta 5 de encuesta de UX. Elaboración propia.*

Qué opinas sobre los sonidos del juego? Detectas mediante el sonido dónde están los enemigos?  
Crees que es apropiada la elección de music themes?

4 respuestas

- Creo que le añadiría inmersión sonidos de pisadas de los enemigos
- La música es increíblemente frenética, me encanta
- Me cuesta mucho saber por dónde vienen
- A veces los enemigos parece que están más cerca pero el resto genial

*Figura 53 - Pregunta 6 de encuesta de UX. Elaboración propia.*

En cuanto a la UX, consideras que este pmv es "crunchy enough" ?

4 respuestas

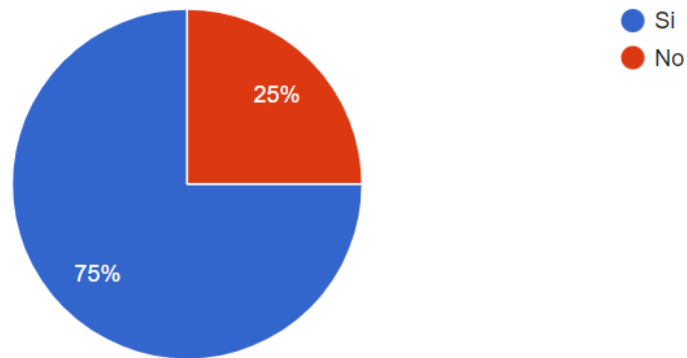


Figura 54 - Pregunta 7 de encuesta de UX. Elaboración propia.

En cuanto a la UI (los canvas de tu muñeca de menu, ingame stats, canvas de muerte, botones world canvas), consideras que este pmv es "crunchy enough" ?

 Copiar

4 respuestas



Figura 55 - Pregunta 8 de encuesta de UX. Elaboración propia.

PistoleroVR

Marcos Eladio Somoza Corral