UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

Graduado en Ingeniería del Software. Plan 2010

Desarrollo de un Juego Multijugador de Disparos Táctico

Development of a Multiplayer Tactical Shooter Game

Realizado por
Francisco Matas Moreno

Tutorizado por
Antonio José Fernández Leiva

Departamento
Lenguajes y Ciencias de la Computación

MÁLAGA, junio de 2025

Agradecimientos

Dedicado a todas aquellas personas que me han acompañado en este camino hasta mi sueño. Por los que ya no están, los que están y los que llegarán.

En especial a mi padre, por ser mi mayor referencia en la vida demostrándome cada día que hay que luchar por lo que queremos.

A mi madre, por ser un pilar incondicional de apoyo que siempre ha confiado en mí hasta cuando más hundido estaba.

A mi hermana, por ser esa guía que siempre necesitaré, la que me marca el camino con sus huellas para que camine seguro.

A mis abuelos, que siempre han creído en mí y prometí por ellos todo esto.

A mis amigos de Herrera, que me han apoyado en este camino aportando momentos de descanso y muchas risas.

A mi nueva familia de Málaga, por ser un descubrimiento inolvidable como aquellos días por Riviera Maya que me hicieron recordar lo que significaba la felicidad.

A mis compañeros de piso, en especial a Miwel por ser un hermano mayor para mí.

A Flis Lerni, Recuperación POO, Yellow Beavers y la Triple Entente por estar siempre ahí, en las buenas y en las malas. Gracias por apoyarme siempre.

Y por último, agradecer a Antonio Fernández Leiva por su implicación, por su guía y por compartir este amor por los videojuegos.

Abstract

The main objective of the Final Degree Project is focused on the implementation of a video game belonging to the tactical first-person shooter genre, commonly known by its acronym *FPS* (*First Person Shooter*). This type of game is characterized by offering an intense action experience from the perspective of the controlled character, promoting both precision and strategic combat. For the development of the project, the video game development engine *Unreal Engine*, developed by *Epic Games*, has been used. This engine provides powerful tools for designing interactive environments. One of the most relevant tools used has been the visual scripting system known as *Blueprints*, which has enabled the development of complex game logic without the need to write traditional code.

Throughout this document, the entire planning, design, implementation, and testing process of the video game is presented in detail. It also includes conceptual and technical references that served as inspiration, as well as the methodology applied to ensure a structured development. Ultimately, this report reflects the work carried out, the knowledge acquired, and the evolution of the video game from its initial ideas to its final implementation.

Keywords: Video game, FPS, Blueprints, Unreal Engine

Resumen

El objetivo principal del Trabajo de Fin de Grado desarrollado se centra en la implementación de un videojuego perteneciente al género de disparos táctico en primera persona, comúnmente conocido por sus siglas en inglés como *FPS* (*First Person Shooter*). Este tipo de juegos se caracteriza por ofrecer una experiencia de acción intensa desde la perspectiva del personaje controlado, promoviendo tanto la precisión como la estrategia en el combate. Para la realización del proyecto se ha utilizado el motor de desarrollo de videojuegos *Unreal Engine*, desarrollado por la empresa *Epic Games*, el cual proporciona potentes herramientas para el diseño de entornos interactivos. Una de las herramientas más relevantes empleadas ha sido el sistema visual de programación denominado *Blueprints*, que ha permitido desarrollar lógicas de juego complejas sin necesidad de escribir código tradicional.

A lo largo del presente documento se expone detalladamente todo el proceso de planificación, diseño, implementación y pruebas del videojuego. También se incluyen referencias conceptuales y técnicas que han servido de inspiración, así como la metodología aplicada para asegurar un desarrollo estructurado. En definitiva, la memoria refleja el trabajo realizado, los conocimientos adquiridos y la evolución del videojuego desde sus primeras ideas hasta su implementación final.

Palabras clave: **Videojuego, FPS, Blueprints, Unreal Engine**

Índice

1. Introducción **1**

1.1. Estructura de la memoria . . . 1

1.2. Motivación . . . 2

1.3. Objetivos . . . 2

1.4. Tecnologías usadas . . . 2

2. Contexto **5**

2.1. Conceptos generales . . . 5

2.2. Género *FPS* . . . 8

2.3. Referencias . . . 8

2.3.1. *Call of Duty* . . . 8

2.3.2. *Valorant* . . . 9

2.3.3. *Counter Strike* . . . 10

3. Análisis y Diseño de la Solución **13**

3.1. Metodología . . . 13

3.1.1. *Scrum* . . . 14

3.1.2. Planificación cronológica . . . 16

3.2. Primer concepto del proyecto . . . 18

3.3. Requisitos . . . 18

3.3.1. Funcionales . . . 18

3.3.2. No funcionales . . . 19

3.4. Diagramas . . . 20

3.4.1. Diagramas de clases . . . 20

3.4.2. Diagramas de flujo . . . 23

3.5. Bocetos de interfaz . . . 26

3.5.1. Menú principal . . . 26

3.5.2. Menú de ajustes . . . 27

3.5.3. Menú de compra . . . 28
3.5.4. HUD . . . 29
3.6. Diseño del escenario . . . 30
3.7. *High Concept* . . . 31
3.7.1. Concepto . . . 32
3.7.2. Género . . . 32
3.7.3. Público objetivo . . . 32
3.7.4. Historia . . . 32
3.7.5. Estilo artístico y desarrollo . . . 32
3.7.6. Gameplay . . . 33

4. Implementación y Desarrollo **35**
4.1. Personaje jugable . . . 35
4.1.1. *Assets* del personaje . . . 36
4.1.2. *Blueprint* de funcionalidad . . . 37
4.1.3. *Blueprint* de animación . . . 40
4.1.4. *Blueprint* de *player controller* . . . 41
4.2. Armas . . . 42
4.2.1. Sonidos y *assets* del arma . . . 42
4.2.2. *Blueprint* de la clase abstracta . . . 43
4.2.3. *Blueprints* de armas instanciables . . . 48
4.2.4. Estadísticas y datos . . . 49
4.3. Granadas . . . 50
4.3.1. *Assets* de las granadas . . . 51
4.3.2. *Blueprint* de la clase abstracta . . . 51
4.3.3. *Blueprints* de granadas instanciables . . . 52
4.4. Modo de juego . . . 53
4.4.1. Cambios con la idea inicial . . . 53
4.4.2. Desarrollo . . . 54
4.5. Menús . . . 57
4.5.1. Menú principal . . . 57

4.5.2. Menú de compra 59
4.5.3. Menú de ajustes 60
4.5.4. *HUD* 61
4.6. Multijugador 62
4.7. Desarrollo del mapa 64

5. Pruebas **67**
5.1. Pruebas realizadas dentro del inspector 67
5.2. Pruebas con otros usuarios 69

6. Conclusiones y Líneas Futuras **71**
6.1. Conclusiones 71
6.2. Líneas Futuras 72
6.3. Aprendizaje personal 73

Apéndice A. Game Design Document **79**

Apéndice B. Manual de Instalación **99**
B.1. Pasos para la instalación 99

Apéndice C. Blueprint del personaje completo **101**

Apéndice D. Blueprint de animación del personaje completo **105**

Índice de figuras

1. Portada del juego *Call of Duty* en la plataforma de *Steam* 9
2. Portada del juego *Valorant* del aniversario del quinto año 10
3. Imagen de *Counter Strike* 11
4. *Product Backlog* listado en Trello 15
5. Planificación del Sprint 3 y 4 usando el sistema de etiquetas de Trello 16
6. Diagrama de clase del jugador 21
7. Diagrama de clase de las armas del juego 22
8. Diagrama de flujo de la partida 24
9. Diagrama de flujo del menú principal 25
10. Boceto del menú principal 26
11. Boceto de confirmación de salir 27
12. Boceto del menú de ajustes 28
13. Boceto del menú de compra 29
14. Boceto del HUD 30
15. Boceto del mapa 31
16. Boceto del mapa del *High Concept* 33
17. Árbol de componentes del *blueprint* del jugador 36
18. *Asset* del personaje 37
19. Función de apuntar (*EnhancedInputActionIA_Aim*) en *blueprints* 39
20. *Blueprint* de animación 40
21. Máquina de estados de *LocomotionMT* 41
22. Situación específica de la máquina de estados *LocomotionMT* 41
23. Primer *asset* de armas [26] 42
24. Segundo *asset* de armas [25] 43
25. Clase abstracta en el inspector 44
26. Salida del *Line Trace* 45
27. Implementación en *blueprint* de la dirección del *Line Trace* 46
28. Función de recargar. Retroalimentación al jugador 47

29. Función de recargar. Aplicar recarga 47
30. Variables en el inspector para el *blueprint* del arma M4 49
31. Smoke *asset* [21] 51
32. *Projectile Movement Component* 52
33. Granada de humo en el inspector 53
34. Aparición del personaje al conectarse a una partida 55
35. *Custom Event: Respawnear* Jugador 56
36. *Custom Event*: Fin de la partida 57
37. Menú asociado al *lobby* en el inspector 58
38. Menú de compra en el inspector 60
39. Menú de ajustes con esquema de controles 61
40. Imagen del *HUD* en el juego 62
41. Secuencia de pasos internos para crear un *host* 63
42. Secuencia de pasos internos para unirse a un *host* 63
43. Estructura de carpetas del mapa en el editor de Unreal Engine 64
44. Vista general del mapa de juego 65
45. Carril central del mapa de juego 65
46. Zona de aparición del equipo atacante en el juego 66
47. *Blueprint* de la función de creación del personaje 101
48. *Blueprint* de la función de movimiento de la cámara 101
49. *Blueprint* de la función de movimiento del personaje 102
50. *Blueprint* de la función de salto del personaje 102
51. *Blueprint* de la función de correr 102
52. *Blueprint* de la función de agacharse 103
53. *Blueprint* de la función de disparar (automático y único) 103
54. *Blueprint* de la función de recargar arma 103
55. *Blueprint* de la función de recibir daño 104
56. *Blueprint* de la máquina de estado para cuando el personaje esta agachado . . 105
57. *Blueprint* de la máquina de estado utilizada cuando el personaje tiene un arma 106

1 Introducción

En esta primera sección se realiza un resumen de la estructura del documento, la motivación de la realización de proyecto, los objetivos y las tecnologías empleadas para todo el proceso de desarrollo.

1.1. Estructura de la memoria

1. *Introducción*: se describe la motivación, objetivos y tecnologías usadas en el proyecto.
2. *Contexto*: en este apartado se describe la base de la idea, así como, inspiraciones en otros juegos.
3. *Análisis y diseño de la solución*: en esta sección se muestra el desarrollo del proyecto entre lo que podemos destacar los bocetos iniciales y la metodología empleada. Además, se muestran los diagramas de clase y otros esquemas propios de la metodología utilizada.
4. *Implementación*: se explican detalles de la implementación del proyecto.
5. *Pruebas*: se muestran las pruebas hechas al proyecto con el objetivo de evitar errores y subsanarlos.
6. *Conclusiones y lineas futuras*: como su nombre indica, incluye un pequeño resumen final y se mencionan algunas actualizaciones futuras que son posibles para realizarse.
7. *Anexos*: en general es información útil para entender el contenido de la memoria, donde se muestran algunos documentos importantes como el *Game Design Document* o el Manual de Usuario.

1.2. Motivación

Como todo el mundo sabe, los videojuegos constituyen una industria que ha crecido exponencialmente en los últimos años. Estos se caracterizan por tener múltiples objetivos desde el más básico como puede ser el entretenimiento hasta algunos más profundos como la ayuda a la educación, en este caso conocidos como *videojuegos serios*. Sin embargo, todos cumplen una función social de desconexión en la que permiten al jugador a desentenderse de sus problemas diarios y disfrutar de momentos inolvidables solos o acompañados con amigos o desconocidos de la red que comparten las mismas aficiones.

Con todo esta idea, mi principal motivación para realizar este proyecto fue conectar a personas a través de un videojuego sencillo que permite retarse entre los jugadores convirtiendo el momento de juego en unos instantes de diversión máxima.

Además, quiero destacar la admiración por los videojuegos que me han acompañado desde mis inicios y que ahora gracias a este proyecto he podido realizar un videojuego propio con todas las ideas que tenía en mi cabeza desde que era un niño.

1.3. Objetivos

El principal objetivo de este Trabajo de Fin de Grado es el desarrollo de un juego multijugador que conecta a dos personas que pueden retarse en un duelo de puntería en un escenario. Además, como otros objetivos también importantes son la aplicación de los conocimientos adquiridos en los cuatro años de carrera como podrían ser el análisis de requisitos, los diagramas de clase o flujo o la planificación y gestión de un proyecto software, incluyendo el diseño y la implementación de software avanzado.

1.4. Tecnologías usadas

A continuación listo las diferentes tecnologías utilizadas en el proyecto y con que objetivo se han utilizado.

1. *Unreal Engine (v5.4.4)*: motor de desarrollo más potente y utilizado actualmente en juegos AAA. Desarrollado y distribuido gratuitamente por la empresa Epic Games. En este proyecto, se ha utilizado para el desarrollo integro del videojuego [16].

2. *Visual Paradigm*: utilizado para la creación de los diagramas de clases entre otros [33].

3. Trello: útil para la planificación del proyecto y tener una visión temporal de las tareas a realizar, realizadas y por hacer.

4. Github: utilizado para el control de versiones del código. Permite una interacción sencilla entre cambios realizados en el programa [29].

5. Excel: usado para el desarrollo de algunos diagramas como el de *Gantt*.

6. Overleaf: herramienta *online* con la que he desarrollado la memoria en Latex [30].

7. FAB.com: biblioteca de *assets* donde he encontrado todos los *assets* incluidos en el programa [19].

8. Udemy: herramienta *online* que vende y distribuye cursos de pago con el que he podido aprender sobre la herramienta de Unreal Engine [32].

9. Paint: utilizada para edición de imágenes.

2
Contexto

En esta segunda sección se explican algunos conceptos claves para entender la memoria, asi como, el género del juego desarrollado y algunas referencias a otros juegos en los que he basado la idea de este proyecto.

2.1. Conceptos generales

Los conceptos a tener en cuenta para la memoria son:

1. *Blueprint*: es un sistema de programación *no-code* que permite crear funciones complejas sin necesidad de tener conocimientos avanzados de programación. En resumen, se trata de una forma de programación visual en la que un usuario puede arrastrar nodos y conectarlos entre sí para ejecutar una función. En *Unreal Engine*, un *Blueprint* es un sistema visual de programación que permite a los desarrolladores crear lógica de juego y comportamientos de objetos sin necesidad de escribir código en lenguajes de programación como C++.

   Los *Blueprints* son una forma gráfica de programar que utiliza un sistema de bloques y conexiones para crear flujos de lógica. Estos bloques pueden representar acciones, condiciones, variables y otros elementos que se utilizan para crear comportamientos complejos en el juego.

   Algunas de las características clave de los *Blueprints* en *Unreal Engine* son:

   - **Programación visual:** los *Blueprints* permiten a los desarrolladores crear lógica de juego de manera visual, sin necesidad de escribir código.
   - **Arrastrar y soltar:** los bloques de *Blueprint* se pueden arrastrar y soltar en un área de trabajo para crear flujos de lógica.

- **Conexiones:** los bloques se pueden conectar entre sí para crear relaciones lógicas.
- **Variables y parámetros:** los *Blueprints* permiten la creación de variables y parámetros que se pueden utilizar para personalizar el comportamiento de los objetos.

Los *Blueprints* son una herramienta poderosa en *Unreal Engine* que permiten a los desarrolladores crear juegos complejos y emocionantes sin necesidad de tener experiencia en programación. Existen algunos tipos según el objetivo que tengamos, entre los que podemos destacar [5]:

a) *Level blueprint*: es el *blueprint* principal del nivel que estamos jugando.

b) *Actor blueprint*: es un tipo de *blueprint* que permite instanciar objetos en el mundo y darles un comportamiento. Similar a una clase de Java.

c) *Player controller*: se refiere a un tipo específico de *blueprint* diseñado para controlar el comportamiento y la lógica de un personaje jugable (*player character*) en el juego. Estos *blueprints* suelen ser usados para implementar movimientos, acciones, interacciones y otros aspectos relacionados con la experiencia del jugador.

d) *GameMode Base*: es un *blueprint* en el que se definen modos de juego, es decir sus reglas, puntuajes y otros aspectos relacionados.

e) *Animation Blueprint*: es un tipo de *blueprint* en el que se define las animaciones de jugadores enemigos u otros elementos animados.

2. *Input Mapping Context*: es una colección de asignaciones entre *Input Actions* y dispositivos de entrada físicos (teclas, botones, ejes analógicos, etc.). Actúa como una *tabla de traducción* que indica qué tecla o botón activa cada acción definida. Este sistema permite crear distintos contextos de entrada (por ejemplo, uno para el menú y otro para el *gameplay*) y activarlos o desactivarlos dinámicamente según la situación del juego. Forma parte del sistema mejorado de entrada (*Enhanced Input System*) introducido en Unreal Engine 5 [9].

3. *Input Action*: es un objeto que representa una acción lógica que puede ser ejecutada por el jugador, como saltar, disparar, correr o abrir un menú. No está vinculada directamente a una tecla o botón, sino que actúa como una abstracción que facilita la vinculación

posterior a múltiples dispositivos de entrada (teclado, ratón, mando, etc.). Este sistema permite separar la lógica del juego de la configuración física del control, mejorando la flexibilidad y escalabilidad del input en el proyecto.

4. *Asset*: es cualquier tipo de archivo que permite representar a un elemento de manera individual, ya sea visual o sonoro, por ejemplo, texturas, animaciones, audio o efectos especiales.

5. *FPS*: De las siglas en inglés *First Person Shooter* o, en español, tirador en primera persona. Se refiere a los juegos en los que la cámara está colocada de manera que simula la vista real de una persona, a diferencia de los juegos en tercera persona en los que el usuario ve todo el cuerpo del jugador desde una visión trasera.

6. *HUD* (*Head-Up Display*): es la información que se muestra en todo momento en la pantalla para tener un vistazo rápido de las estadísticas más importantes para el jugador. Se suele mostrar el número de vidas, puntos de experiencia, balas de un arma, entre otros [34].

7. *GUI* (*Grafical User Interface*): En español, interfaz gráfica del usuario, es el conjunto de elementos visuales e interactivos que permiten al usuario comunicarse con el juego de forma intuitiva. Algunos de los elementos más importantes son los botones o las cajas de texto. Suele ser confundida con la idea de *HUD*.

8. *Static Mesh*: es un tipo de recurso tridimensional utilizado en los motores de videojuegos, como Unreal Engine, que representa una malla 3D no animada [15].

9. *Máquina de estados finita* (*Finite State Machine*): es un modelo computacional que representa el comportamiento de un sistema mediante un conjunto de estados definidos y transiciones entre ellos, activadas por eventos o condiciones específicas. En videojuegos, se utiliza comúnmente para controlar la lógica de personajes, menús, animaciones o cualquier sistema que deba responder de forma estructurada a distintos escenarios o entradas del jugador.

10. *Host* (Anfitrión): es el dispositivo o jugador que inicia y gestiona la sesión de juego, actuando como servidor para los demás participantes (clientes). El *host* se encarga de

controlar la lógica del juego, sincronizar los eventos entre los jugadores y mantener el estado global de la partida.

2.2. Género *FPS*

Este género es uno de los géneros de videojuegos más famosos e influyentes en la industria. Su principal característica es que el jugador experimenta la aventura desde los ojos del personaje, lo que proporciona una inmersión mucho mayor.

Desde sus orígenes, en los años 90, con títulos con *DOOM*, el género ha evolucionado de manera exponencial, con entregas que han marcado un antes y un después en la industria de los videojuegos como *Call Of Duty*, de la empresa Activision [1], entre otros. Gracias al desorbitado desarrollo cada vez se perfecciona más la experiencia de usuario incluyendo mecánicas nuevas o creando modos de juego con innumerables temáticas. Actualmente, existen múltiples subgéneros como el táctico, el *battle royale* o temática *hero*.

La popularidad del género se mantiene gracias a su dinamismo, su competitividad y su constante innovación tecnológica, lo que lo hace ideal para explorar mecánicas de interacción, diseño de niveles y técnicas de programación avanzadas en motores como Unreal Engine.

2.3. Referencias

En este apartado, se comentan algunas referencias de juegos en los cuales se ha basado la idea para la definición de mi proyecto. Cabe destacar, que dichas referencias han servido para conseguir un punto común entre ellos, seleccionando para el proyecto aquellos aspectos más convenientes, los cuales se indican en cada subsección.

2.3.1. *Call of Duty*

Call of Duty es uno de los *FPS* más jugados y vendidos de la historia de los videojuegos. Su idea se basa en múltiples modos de juego que cuentan con gran frenetismo como el duelo por equipos, atrapa la bandera o baja confirmada.

La saga cuenta con un total de cuarenta y cuatro entregas para la mayoría de las consolas o dispositivos móviles. Se estima que se han vendido unas quinientas millones de copias de toda la saga convirtiéndola así en una de las sagas de juegos más exitosa de toda la historia. En

2020, lanzaron su primer *battle royale*, *Warzone*, que fue realmente exitoso al ser una entrega gratuita.

Este videojuego ha servido para inspirar la selección del armamento del personaje y además para replicar, con ciertos cambios, el modo de juego.

Figura 1: Portada del juego *Call of Duty* en la plataforma de *Steam*

2.3.2. *Valorant*

Valorant es un juego de disparos táctico en primera persona, que fue lanzado por la empresa Riot Games en 2020. El juego ha experimentado un gran crecimiento en todos los aspectos técnicos y de comunidad desarrollando múltiples eventos durante todo el año. El evento más prestigioso es el *Valorant Champions* que es culmen de todo un año de competiciones de los mejores jugadores del mundo. Se desarrolla una vez al año, dando premios que ascienden a varios millones de euros para el club ganador.

Por su parte, cuenta con un modo de juego por rondas por equipos de cinco personas, en la que cada uno elige un personaje al principio de la partida que cuenta con ciertas habilidades con las que debe coordinarse con el resto de su equipo para ganar cada ronda.

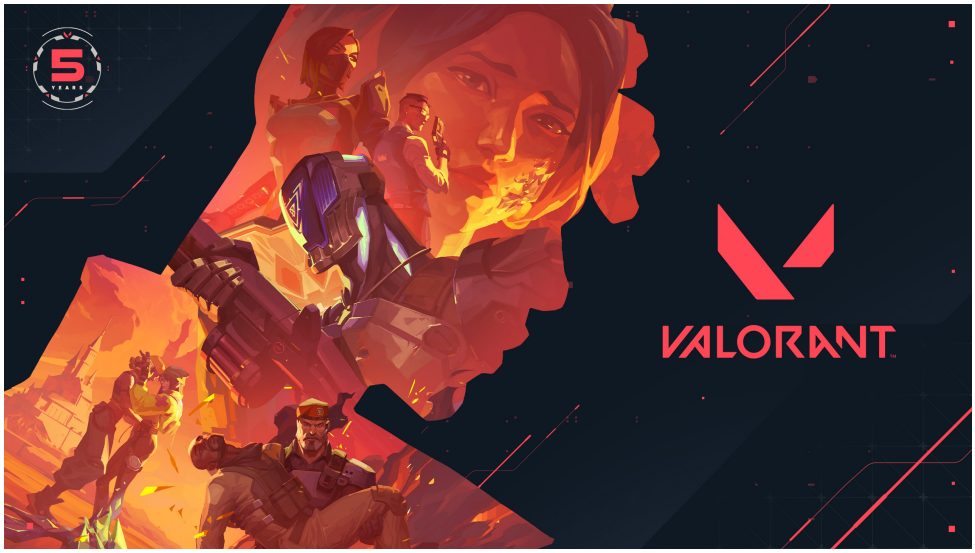


Figura 2: Portada del juego *Valorant* del aniversario del quinto año

En cuanto a los aspectos comunes entre *Valorant* y el proyecto los principales puntos compartidos son:

- Modo de disparo táctico
- Compra de armas

La principal idea que se comenta en el apéndice A, estaba basado en *Valorant* que finalmente tras algunos cambios se asemeja más al modo de juego de *Call Of Duty*.

2.3.3. *Counter Strike*

Counter Strike es un juego de disparos táctico realista, con un sistema de juego similar a *Valorant*. Su principal diferencia es que en *Valorant* los personajes tienen habilidades no realistas mientras que en *Counter Strike* si lo son.

La saga ha contado con múltiples actualizaciones sobre todo en cuanto al motor utilizado siendo la última de ellas realizada en el año 2023. Utiliza el motor *Source 2*, desarrollado por la misma empresa que desarrolla el juego, Valve.

Como *Valorant*, tiene una gran comunidad competitiva que realiza grandes torneos durante todo el año.

Figura 3: Imagen de *Counter Strike*

El modo de juego es igual que en *Valorant*, de hecho, se considera que *Valorant* copió a este juego aunque cambiando pequeños aspectos como el número de rondas a ganar o la fase de prórrogas.

En cuanto a los aspectos comunes entre *Counter Strike* y el proyecto los principales puntos compartidos son:

- Modo de disparo táctico
- Compra de armas
- Carácter realista del juego

3

Análisis y Diseño de la Solución

En este apartado se comentan distintos aspectos sobre el análisis y diseño del proyecto. En primer lugar, se habla de la metodología elegida y cómo se ha llevado a cabo durante el proceso de desarrollo. Seguidamente, se muestran aspectos relacionados con el proceso de ingeniería del software como los requisitos o diagramas. Finalmente, se muestran algunos bocetos de la interfaz que se propuso al inicio del diseño, además de un boceto a mano alzada del escenario principal.

3.1. Metodología

Existen numerosas metodologías para el desarrollo de proyectos software. Sin embargo, en este desarrollo se ha elegido una metodología ágil, la cual se basa en los siguientes aspectos:

1. Las personas somos lo primero
2. Equipos multifuncionales
3. Entregar valor de manera iterativa e incremental
4. El cambio es una oportunidad de mejora
5. La retrospectiva como forma de reflexión sobre el desarrollo realizado previamente
6. El cliente está presente durante todo el proceso y es el centro de ejecución

Dentro de las metodologías ágiles, existen muchos tipos, en concreto se ha elegido *Scrum* que se explica a continuación.

3.1.1. *Scrum*

Scrum se basa en ciclos cortos y repetitivos de trabajo llamados *sprints*, que suelen durar entre una y cuatro semanas. Al finalizar cada *sprint*, se entrega un incremento funcional del producto, lo que permite obtener retroalimentación temprana y continua del cliente o usuarios finales. Para el proyecto se han elegido *sprints* de dos semanas. En cada *sprint* se han realizado las siguientes tareas:

- *Sprint* 0: Formación y aprobación del proyecto con el tutor.
- *Sprint* 1: Desarrollo del documento *High Concept* y *Game Design Document*. Búsqueda de *assets* para el proyecto.
- *Sprint* 2: Análisis de requisitos y creación del proyecto.
- *Sprint* 3: Diagramas de clase y desarrollo del escenario.
- *Sprint* 4: Desarrollo de la funcionalidad principal.
- *Sprint* 5: Desarrollo de los menús del juego. Resolución de errores del *sprint* 4.
- *Sprint* 6: Pruebas y correcciones.
- *Sprint* 7: Desarrollo de la memoria del proyecto.
- *Sprint* 8: Cierre del proyecto. Finalización de la memoria.

Antes de iniciar con la definición y encajar las tareas dentro de un *sprint* u otro, se define el *product backlog*, es decir, la pila de producto o lo que es lo mismo todas las tareas a realizar durante el proyecto.

Para llevar a cabo esta organización se ha utilizado la herramienta de Trello, ya que su fácil interpretación y usabilidad hacen que sea la aplicación idónea para lo que el proyecto requería. Trello permite marcar las tareas creadas con etiquetas personalizadas con las que indica claramente en que fase del desarrollo se encuentra cada una de ellas. A continuación, se muestran algunos ejemplos de como he organizado el proyecto dentro de la herramienta.

Figura 4: *Product Backlog* listado en Trello

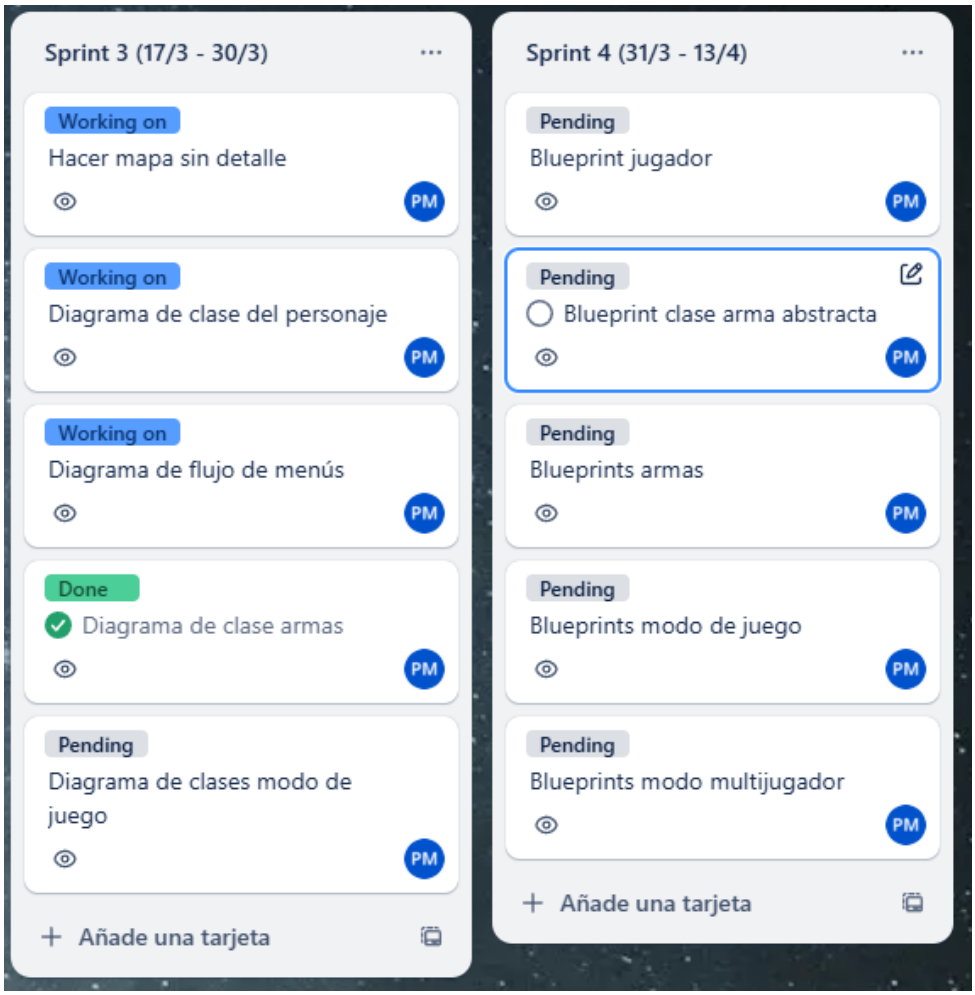


Figura 5: Planificación del Sprint 3 y 4 usando el sistema de etiquetas de Trello

3.1.2. Planificación cronológica

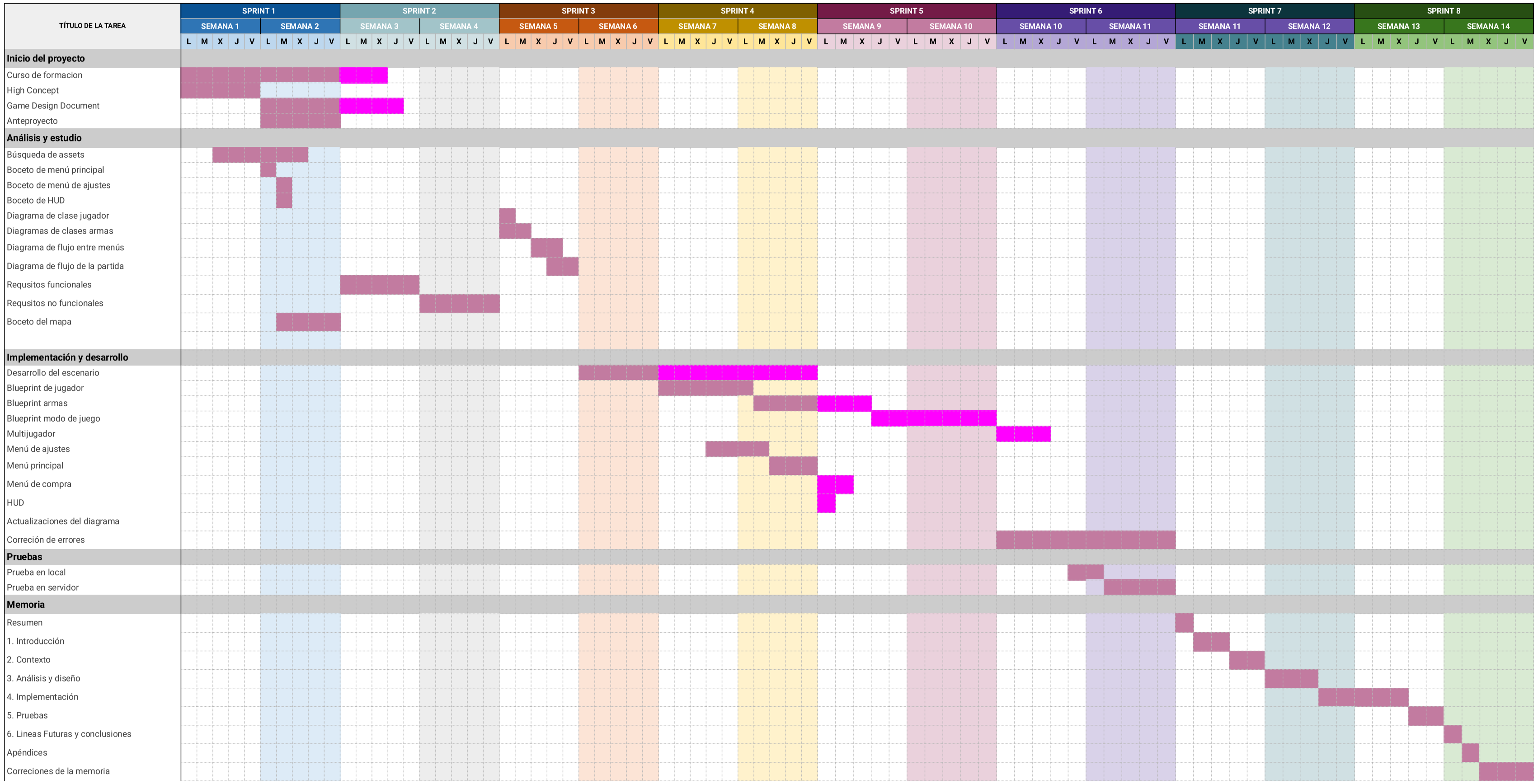
Para realizar la planificación cronológica más exacta, además de realizar la planificación en Trello como ya hemos visto en el apartado anterior, se ha realizado un diagrama de *Gantt* con la herramienta de Excel.

En el diagrama se muestra como se han ido realizando las tareas de manera diaria, indicando con una leyenda de colores si se han realizado las tareas en el plazo previsto o cuanto he excedido ese tiempo. Por ello, durante el *sprint* seis la carga de trabajo es menor se ha utilizado también como *sprint* para recuperar el tiempo que se había retrasado hasta el momento mientras se realizaban las pruebas pertinentes.

A continuación, se muestra el diagrama en una tabla dispuesta de forma horizontal para facilitar su lectura.

DIAGRAMA DE GANTT

TÍTULO DEL PROYECTO	SILENT CHAOS	FECHA INICIO	03/02/25
RESPONSABLE DEL PROYECTO	Francisco Matas Moreno	FECHA FIN	08/06/25



3.2. Primer concepto del proyecto

La primera idea para el proyecto fue realizar un *FPS* táctico con el que jugar con tu equipo y enfrentarte a otro equipo en un mismo servidor. Además, la idea era recrear un ambiente bélico para introducir aún más al jugador en el ambiente.

Como en los videojuegos a los que se intenta hacer referencia, la idea era tener una variedad de armas para poder elegir distintas armas o equipamiento según la situación de la partida. Por otro lado, se iba a realizar por un modo de juego por rondas con atacantes y defensores. Los atacantes debían llevar un artefacto a una zona del mapa y los defensores evitarlo.

Además, el desarrollo debía ser realizado en Unreal Engine 5, utilizando las facilidades que ofrece, con el objetivo de darle un carácter realista y aprender una nueva herramienta.

Para profundizar en el primer concepto del proyecto ir al apartado 3.7

3.3. Requisitos

En todo proyecto software es necesario una captación de requisitos para entender las necesidades del cliente y llevar a cabo el proyecto de manera organizada. Para el análisis de requisitos se llevó a cabo una reunión con el cliente, que definió los necesarios para el proyecto, los cuales se detallan en las próximas subsecciones.

3.3.1. Funcionales

Los requisitos funcionales son aquellas especificaciones que describen las funciones, comportamientos y servicios que el sistema debe ofrecer. Representan lo que el software debe ser capaz de hacer en términos de interacción con el usuario, procesamiento de datos, comunicación con otros sistemas o respuesta a determinadas entradas.

En reunión con el cliente se definieron los siguientes requisitos funcionales:

RF1.- El jugador debe poder moverse en cuatro direcciones.

RF2.- El personaje debe poder disparar con clic izquierdo.

RF3.- El jugador debe poder elegir entre todas las armas disponibles desde un menú propio.

RF4.- El jugador debe poder comprar granadas de mano desde el mismo menú de armas.

RF5.- El jugador debe poder ver los controles desde el menú principal.

RF6.- El jugador debe poder elegir entre crear una partida o unirse a una.

RF7.- El jugador debe poder moverse libremente por el escenario durante la partida.

RF8.- El juego debe ser compatible con resoluciones 1280x720 píxeles y 1920x1080.

RF9.- El juego debe poder ser ejecutado en sistemas operativos Windows 10 o versiones posteriores.

Tras el desarrollo del producto se han podido implementar todos ellos de forma exitosa. Se indica y referencia en el próximo apartado número 4 qué implementación hace uso de cada requisito funcional.

3.3.2. No funcionales

Los requisitos no funcionales son especificaciones que definen las cualidades, restricciones o atributos del sistema, más allá de las funciones específicas que debe realizar. No describen qué hace el sistema, sino cómo debe hacerlo. Incluyen aspectos como el rendimiento, la escalabilidad, la seguridad, la usabilidad, la disponibilidad, la eficiencia energética o la compatibilidad con distintos dispositivos o plataformas.

Este tipo de requisitos condicionan de manera directa la arquitectura del programa, así como, su diseño y las soluciones de programación elegidas. Por tanto, en la reunión con el cliente se definieron inicialmente los siguientes requisitos:

RNF1.- El sistema debe responder en un tiempo de respuesta o latencia de la partida menor a 100 milisegundos en condiciones de red estables.

RNF2.- El sistema no debe reducir la tasa de fotogramas por segundo a una inferior a 60 durante la partida.

RNF3.- El tiempo de carga entre el menú principal y la partida no debe ser superior a quince segundos.

RNF4.- La interfaz debe ser intuitiva, permitiendo ser entendida toda funcionalidad sin necesidad de un tutorial.

RNF5.- El sistema de armas debe ser modular con el objetivo de añadir nuevas armas en actualizaciones futuras.

RNF6.- Los botones deben poderse clicar y el usuario debe recibir una retroalimentación por parte de ellos de manera visual o auditiva.

3.4. Diagramas

En esta subsección se muestran algunos diagramas de diseño con los que luego se han desarrollado los *blueprints* del juego. Estos esquemas ayudan a tener una visión general del funcionamiento de las clases o de los flujos del programa.

Por un lado, los diagramas de flujo reflejan la lógica de ejecución de determinadas mecánicas del juego, como la compra de armas, el sistema de daño o el control de las rondas. Este tipo de diagrama permite representar de forma clara y secuencial las decisiones y ramificaciones que ocurren durante la partida, siendo especialmente útil para trasladar esa lógica directamente a *blueprints* en Unreal Engine.

Por otro lado, los diagramas de clases permiten identificar los distintos actores o componentes principales del proyecto (jugadores, armas, interfaz, lógica de ronda, etc.) y las relaciones entre ellos. Estos diagramas han servido como base para organizar la estructura modular del juego, facilitar la reutilización de código y definir las responsabilidades de cada clase dentro del sistema.

Ambos tipos de diagramas han sido fundamentales en la fase de planificación y han facilitado una implementación ordenada y coherente del sistema completo del juego dentro del motor Unreal Engine 5, mediante el uso intensivo de *blueprints* visuales.

3.4.1. Diagramas de clases

A continuación se muestran algunos de los diagramas de clase desarrollados para el análisis y diseño del programa.

- Diagrama de clase del jugador: este diagrama de clase muestra los atributos y métodos que el jugador puede realizar. En este diagrama se muestra el *blueprint* de primera persona del jugador donde se muestran todas las variables que tienen, junto al *blueprint* de animación que contiene para animar el personaje y le de un toque de viveza a la partida.

En cuanto a los *IMC* (*Input Mapping Context*) se muestran los dos que tiene el personaje divididos en las acciones por defecto y las acciones que el personaje puede realizar cuando tiene una granada o un arma equipada.

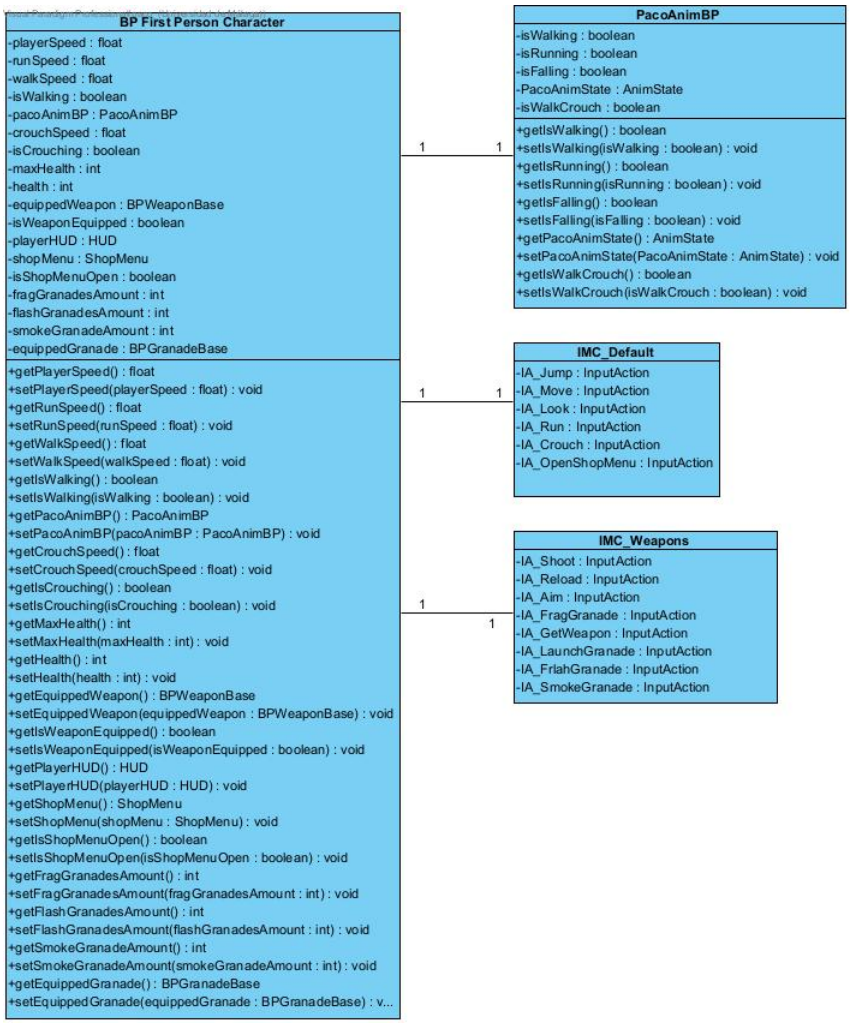


Figura 6: Diagrama de clase del jugador

- Diagrama de clases de arma: este diagrama muestra como se ha realizado la lógica de armas, para ello se ha creado una clase abstracta que contiene todos atributos y métodos que tienen las clases que heredan y si son instanciables. Se define la clase abstracta como *BP_WeaponBase* y dos enumerados que indican el tipo de arma (*WeaponEnum*) y otro con el tipo de disparo, automático o semiautomático (*FirerateEnum*).

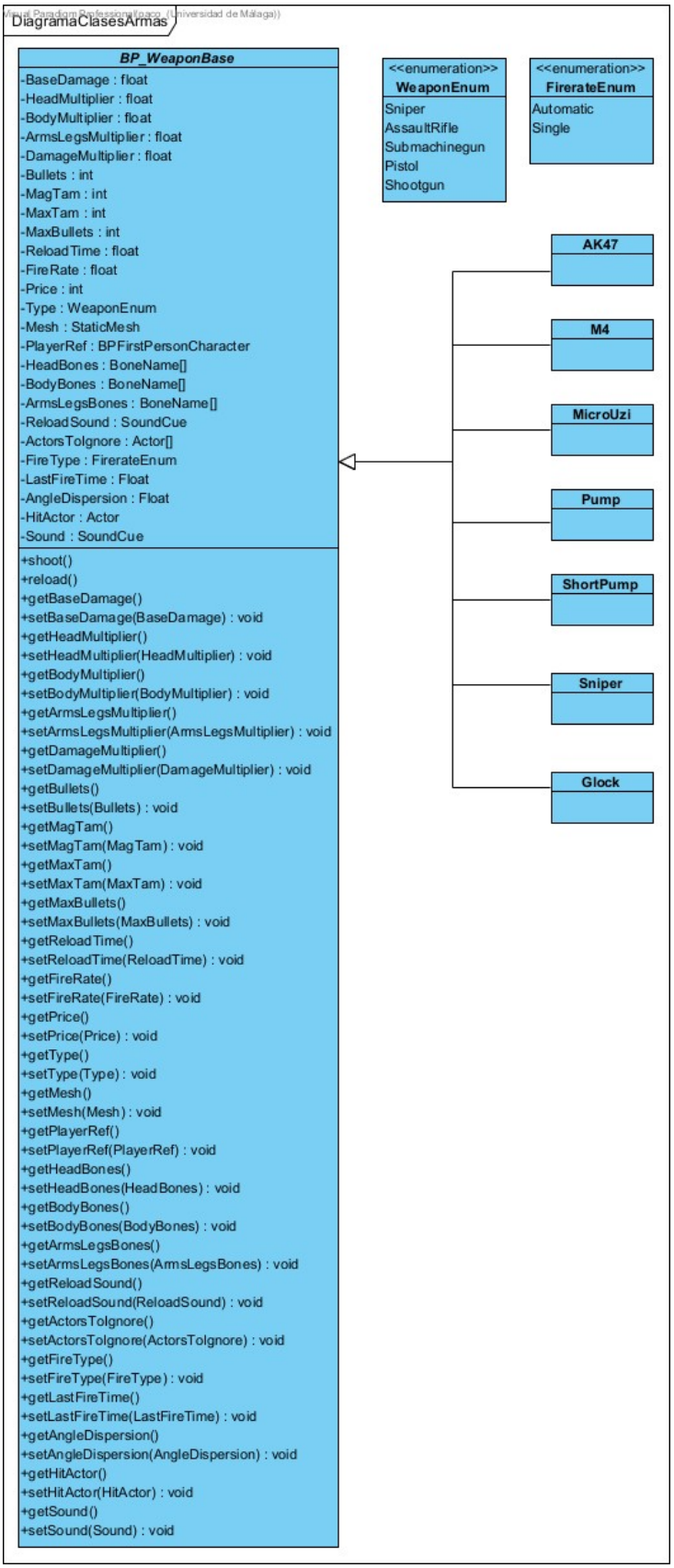


Figura 7: Diagrama de clase de las armas del juego

3.4.2. Diagramas de flujo

Los diagramas de flujo son una herramienta visual utilizada para representar procesos, decisiones y secuencias de acciones de manera clara y estructurada. Mediante el uso de símbolos gráficos estandarizados (como rectángulos para procesos, rombos para decisiones y flechas para indicar el flujo), estos diagramas permiten comprender de forma rápida la lógica y el funcionamiento de un sistema sin necesidad de revisar directamente el código o la implementación técnica.

En el contexto de este proyecto, los diagramas de flujo han sido utilizados para planificar y documentar la lógica de distintas funcionalidades del videojuego, como el sistema de disparo, la recarga de armas, el flujo del modo de juego o la reaparición de los jugadores. Gracias a su capacidad para simplificar procesos complejos, han resultado especialmente útiles durante las fases de diseño y desarrollo, facilitando la organización del trabajo y la comunicación de ideas.

La estructura y uso de estos diagramas sigue el enfoque descrito en [2], donde se destaca su utilidad para visualizar de forma ordenada cualquier flujo de trabajo.

- **Diagrama de flujo de la partida:** El siguiente diagrama de flujo representa la lógica principal del modo de juego desarrollado. El flujo comienza con el inicio de la partida, momento en el que se ejecuta la acción de ***spawnear* al jugador**. Una vez el personaje ha aparecido en el escenario, se verifica si tiene un arma equipada.

  En caso negativo, se abre el **menú de compra**, donde el jugador debe seleccionar un arma para poder continuar. Una vez el arma está equipada, el jugador puede comenzar a **disparar al rival**.

  Cada disparo realizado se evalúa mediante una condición que determina si el disparo ha sido acertado. Si no impacta, se permite seguir intentando. En caso de que el disparo acierte, se procede a **decrementar la vida** del jugador objetivo.

  Tras esta reducción de vida, se verifica si el jugador sigue con vida. Si la vida es mayor a cero, el ciclo continúa normalmente y se permite seguir combatiendo. Si, por el contrario, la vida es igual o inferior a cero, se considera que el jugador ha sido eliminado.

  A continuación, se **incrementa el número de *kills* del jugador atacante** y se comprueba si ha alcanzado el número de eliminaciones necesarias para ganar (en este caso,

cinco). Si no se ha alcanzado el límite, el flujo regresa al punto de aparición para continuar el enfrentamiento. En caso afirmativo, la partida finaliza.

Este flujo resume de forma clara y visual la lógica de ciclo de combate básico del juego, desde el *spawn* del jugador hasta la condición de victoria.

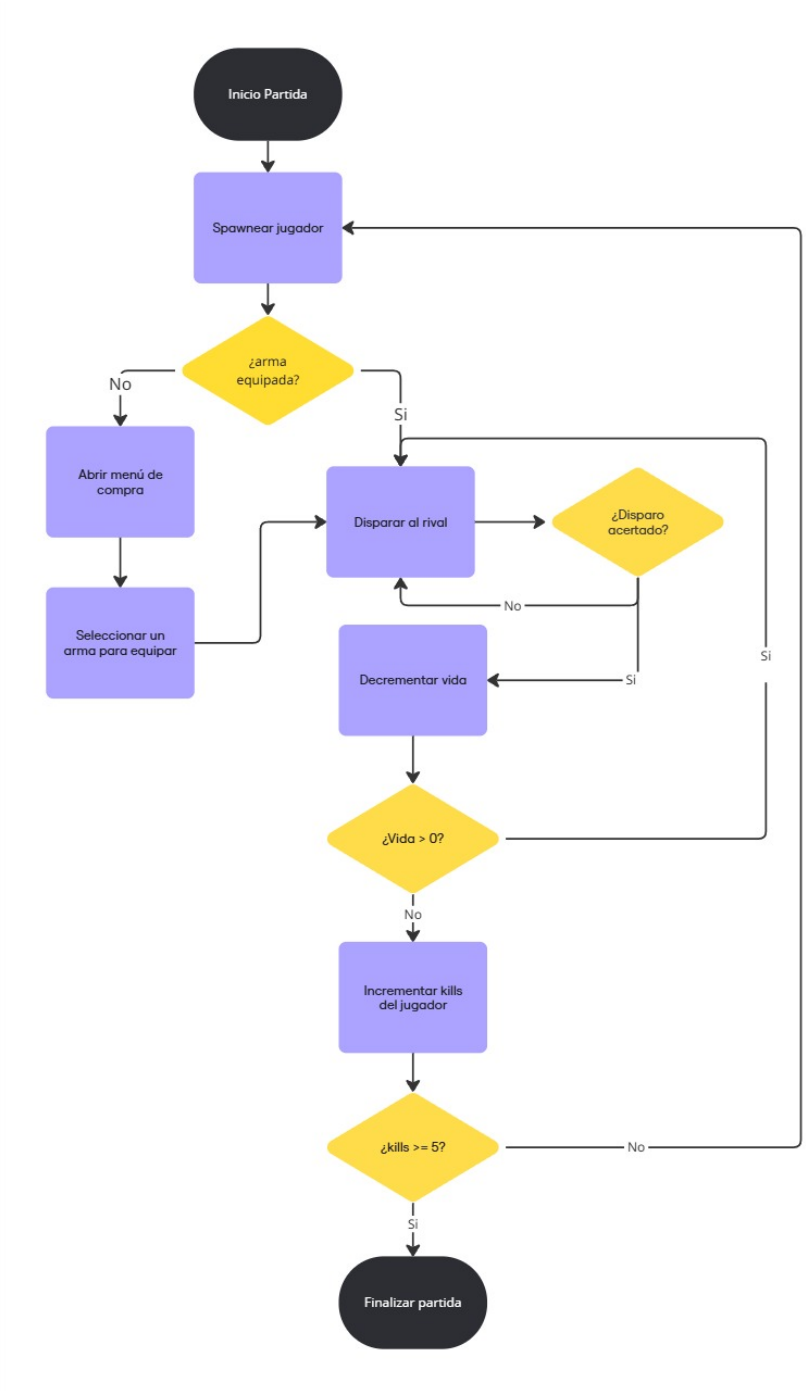


Figura 8: Diagrama de flujo de la partida

- **Diagrama de flujo del menú principal:** El siguiente diagrama de flujo representa el comportamiento del **menú principal** del videojuego, que actúa como punto de entrada para el sistema multijugador. Desde este menú, el jugador puede realizar tres acciones principales: crear una sesión como host, unirse a una sesión existente o acceder al menú de ajustes.

  El flujo comienza en el nodo **Menú principal** y, a continuación, evalúa qué botón ha sido pulsado por el jugador. Si se pulsa el botón **Crear *Host***, se inicia una sesión en modo escucha (*listen server*), lo que convierte al jugador en servidor y le permite albergar una partida multijugador. Si se pulsa el botón **Unirse**, el sistema intenta conectar con un servidor existente utilizando la dirección IP introducida en el campo correspondiente del menú.

  Por otro lado, si el jugador pulsa el botón de **Ajustes**, se abre un menú adicional donde puede configurar diferentes parámetros del juego. Este diagrama resume de forma clara la lógica inicial que determina cómo se accede y se gestiona la conexión al modo multijugador desde la interfaz principal.

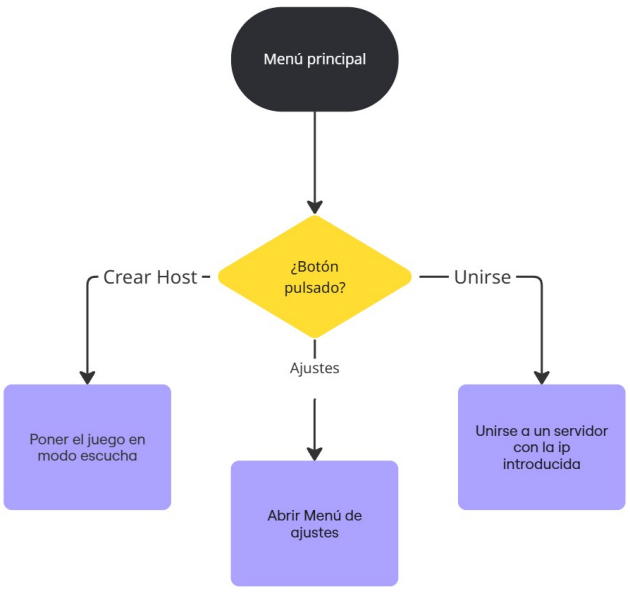


Figura 9: Diagrama de flujo del menú principal

3.5. Bocetos de interfaz

En este subapartado se muestra los bocetos que se realizaron al inicio para entender y tener una visión previa de la aplicación. Estos bocetos inspiraron la interfaz final de la aplicación. A continuación, se describen todos los bocetos definidos para cada uno de los menús.

3.5.1. Menú principal

Es el menú principal del juego, el que el jugador visualiza cuando inicia la aplicación. El usuario debe poder iniciar una partida, unirse a ella o abrir el menú de ajustes.

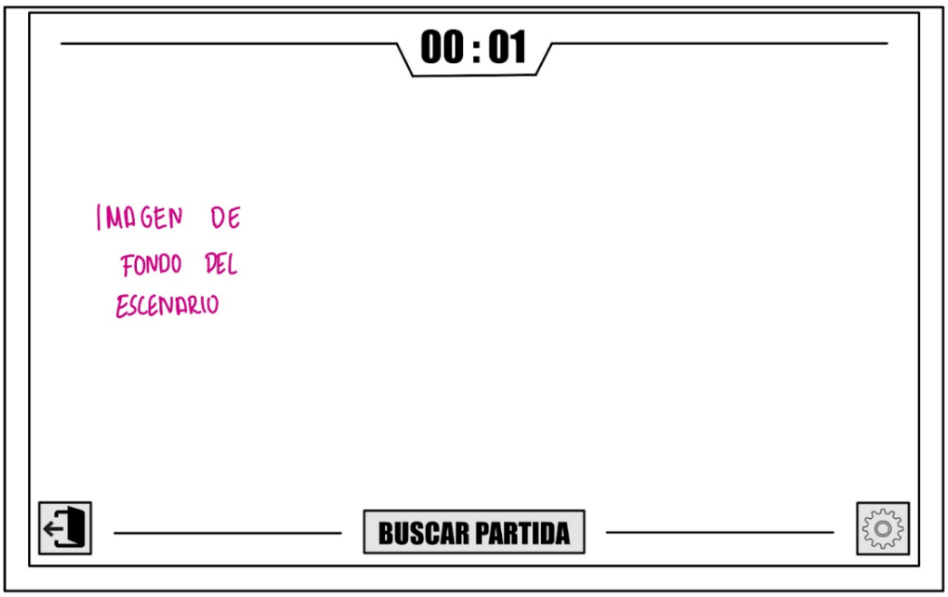


Figura 10: Boceto del menú principal

Además al pulsar el botón de salir, indicado con una puerta en la parte abajo izquierda de la pantalla, se muestra una confirmación para no cerrar el juego sin intención por clicar aleatoriamente por la pantalla.

Figura 11: Boceto de confirmación de salir

Cuando el usuario pulse la acción de ”SALIR”, el juego se cerrará. En caso de pulsar en el botón rojo ”NO”, el usuario volverá al menú principal.

3.5.2. Menú de ajustes

El menú de ajustes muestra todas las configuraciones posibles que el jugador puede modificar para adaptar su experiencia de juego. Además, contendrá una sección de controles donde el usuario podrá leer los controles de una manera gráfica para facilitar una rápida visualización.

Al pulsar en cada una de las secciones se muestran ajustes del tipo que se indica con el objetivo de facilitar su fácil visualización. Por su parte, pulsando el botón ”SALIR”, el jugador vuelve al menú principal.

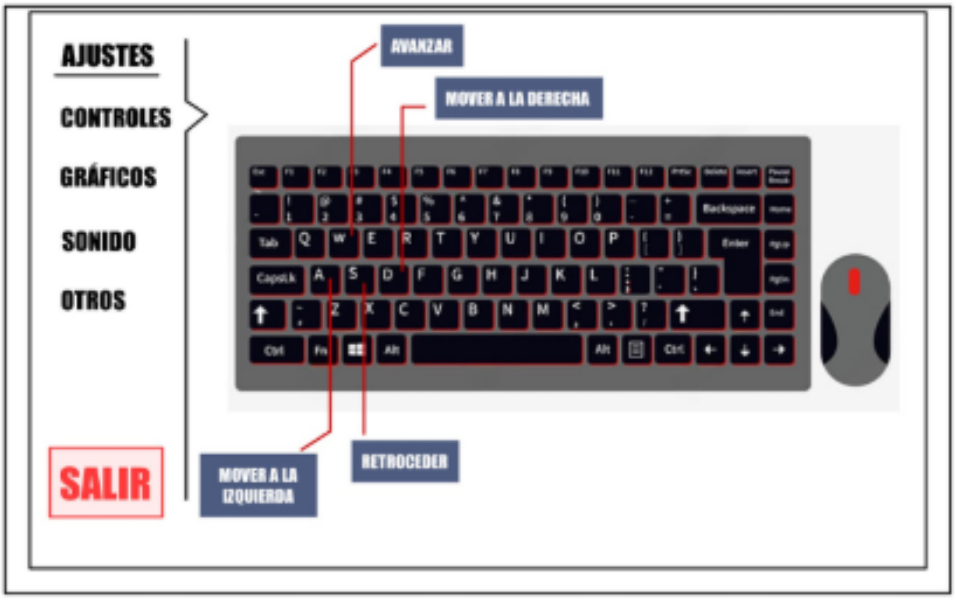


Figura 12: Boceto del menú de ajustes

3.5.3. Menú de compra

Este menú solo es accesible desde dentro de la propia partida. Se puede activar y desactivar durante la fase de compra.

Este menú consiste en unos botones con los iconos de cada arma, que al clicar en uno de ellos, si el jugador tiene créditos suficientes para adquirirla, obtendrá el arma y se le restará el coste del arma al total de sus créditos.

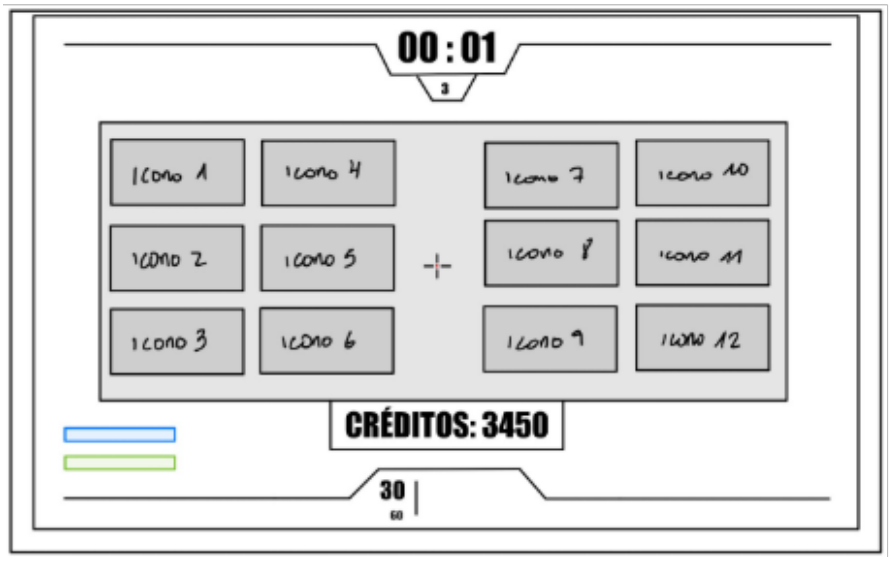


Figura 13: Boceto del menú de compra

3.5.4. HUD

Para facilitar esta característica se ha realizado una versión preliminar de esta. La idea es tener un HUD limpio y lo más visual posible. Por ello, el boceto inicial es:

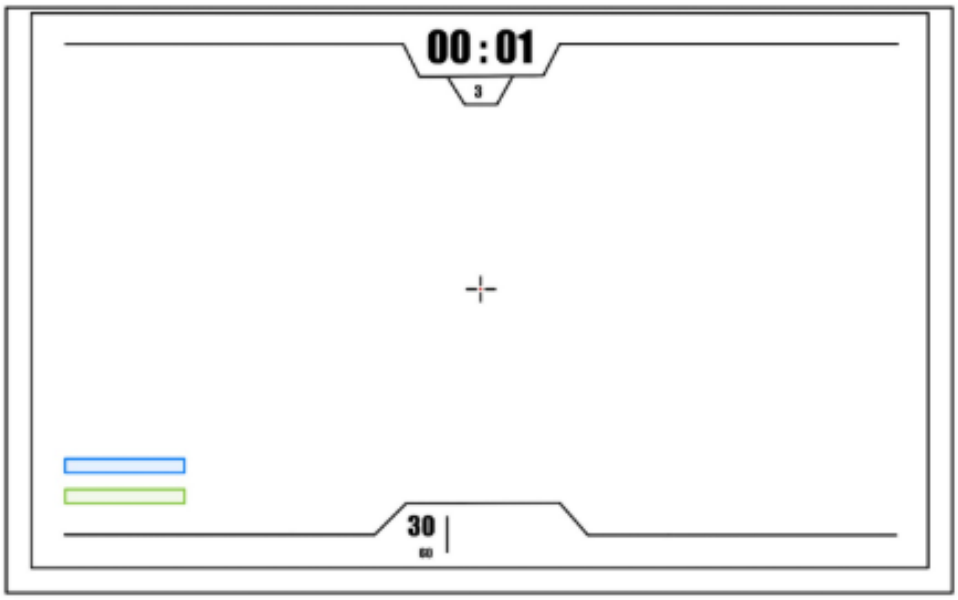


Figura 14: Boceto del HUD

Por su parte, en la parte inferior, se encuentra el número de balas restantes del cargador y del total. También hay dos barras de vidas que indica la vida del jugador y el total de escudo que tiene.

En el centro, existe una mirilla que indica la dirección y sentido de las balas disparadas por el jugador si este está completamente parado, si no se aplicará un error en función de la velocidad del jugador.

3.6. Diseño del escenario

El diseño del escenario comenzó a partir de los requisitos definidos en el documento de diseño del juego (Apéndice A). Con el objetivo inicial de implementar el modo de juego original planteado en el que se enfrentan dos equipos de entre cuatro y seis jugadores, se desarrolló un mapa de tamaño considerable, capaz de albergar partidas de hasta doce jugadores.

Una de las principales ventajas de haber diseñado un entorno amplio y bien estructurado es su versatilidad. Este escenario puede adaptarse fácilmente a distintos modos de juego clásicos del género de disparos, como *duelo por equipos*, *atrapa la bandera* o *punto caliente*, entre otros. Sin embargo, para el modo de juego finalmente implementado en esta versión, basado en enfrentamientos individuales entre dos jugadores, el tamaño del mapa puede resultar ex-

cesivo y, por tanto, afectar negativamente al ritmo de las partidas, reduciendo su intensidad y frenetismo.

En cuanto al diseño visual, se han utilizado ***assets* procedentes de la biblioteca de Quixel**, un proveedor integrado en la plataforma Fab que distribuye sus recursos bajo licencia de uso personal. Estos recursos han permitido construir un entorno realista y detallado sin necesidad de desarrollar modelos desde cero [31].

La estructura espacial del mapa se basa en un enfoque comúnmente utilizado en los juegos de disparos multijugador: el **diseño de tres carriles**. Esta disposición ofrece múltiples rutas y puntos de enfrentamiento, fomentando tanto el juego estratégico como el dinamismo. Para guiar el proceso de construcción del escenario, se elaboró previamente un **boceto a mano alzada** que sirvió como referencia durante la implementación en el motor de juego.

Figura 15: Boceto del mapa

3.7. *High Concept*

En esta sección se presenta el *High Concept* del videojuego, un elemento clave en la fase de diseño que resume de forma clara y concisa la idea central del proyecto. El *High Concept* tiene como objetivo definir la esencia del juego, su propuesta de valor y los aspectos que lo hacen único o atractivo para los jugadores. Esta descripción sirve como punto de partida tanto para la planificación del desarrollo como para comunicar la visión del juego a posibles colaboradores, tutores o interesados en el proyecto. En definitiva, actúa como una carta de presentación que

condensa en pocas líneas la identidad del videojuego.

3.7.1. Concepto

Es un juego multijugador online donde dos equipos, compuestos por cuatro jugadores, se enfrentan en un escenario para intentar eliminar al equipo contrario y alzarse con la victoria.

El juego estaría ambientado en un mundo apocalíptico donde cada personaje cuenta con un arsenal que le servirá de medio para poder abatir a los oponentes.

3.7.2. Género

Este estaría incluido en el género de videojuegos *tactical shooter* en primera persona, donde la colaboración, coordinación y tu propia agilidad mental son de vital importancia para trazar estrategias que tus adversarios no esperen y poder así conseguir derrotarlos.

3.7.3. Público objetivo

Puesto que se trata de un juego de disparos, el público objetivo será de una edad mínima de adolescente ya que la idea tampoco consiste en juego hiperrealista que pueda herir sensibilidad.

3.7.4. Historia

Por el estilo de juego, la historia no es importante.

3.7.5. Estilo artístico y desarrollo

El desarrollo sería con la herramienta de *Unreal Engine 5*, siendo un videojuego 3D.

En cuanto al estilo artístico, el objetivo es desarrollar un escenario con un estilo apocalíptico usando *assets* gratis de *Fab*, entre otros. Además, es importante desarrollar un escenario que permita al jugador reinventarse en cada jugada y utilizar el entorno a su favor para ejecutar jugadas conjuntas con sus compañeros.

Una posible idea de escenario es:

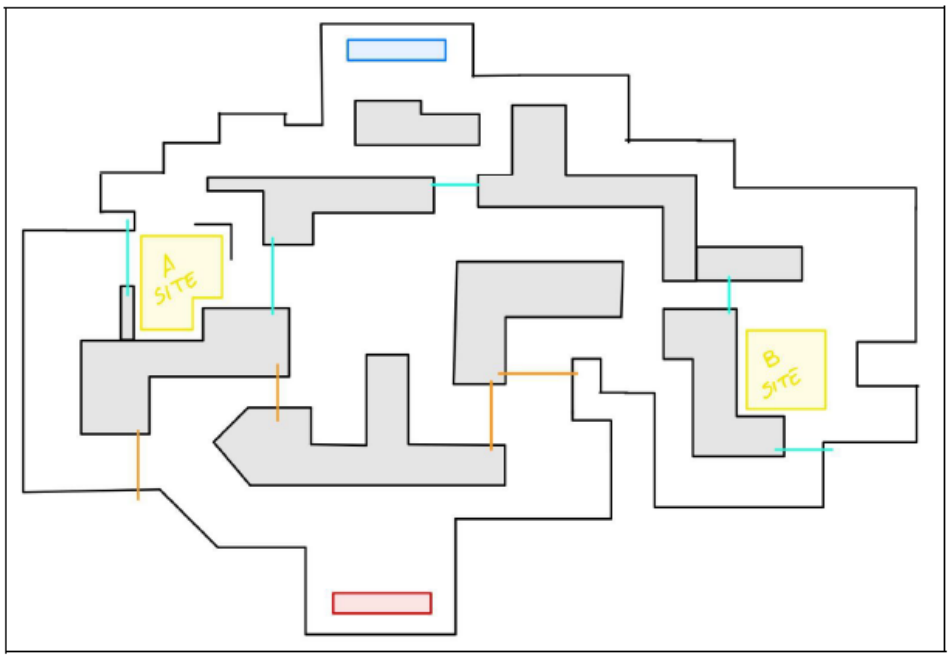


Figura 16: Boceto del mapa del *High Concept*

Vista superior del mapa

3.7.6. Gameplay

A pesar de que existen numerosas entregas de videojuegos con este estilo, por ejemplo, *Counter Strike*, *Valorant*, *Rainbow Six Siege*, entre otros, ninguno cuenta con una ambientación apocalíptica o cuyos equipos estén conformados únicamente por cuatro miembros, haciendo así que los jugadores tengan que elegir más cuidadosamente qué arsenal y utensilios llevar.

Profundizando un poco más en el *gameplay*, lo principal, es conseguir la victoria. Esta se conseguirá cuando un equipo haya conseguido ganar un total de nueve rondas. Cada equipo tendrá un rol de atacante o defensor que se elegirá al azar al comienzo de la partida y que continuará en dicho rol durante un total de 8 rondas. En cada ronda, al inicio, habrá una fase de compra de arsenal en la que los jugadores podrán elegir qué comprar en función de la economía que tengan (tras cada ronda el equipo recibe un dinero ficticio en función del desempeño), tras esto el equipo atacante contará con un total de noventa segundos para poder desplegar un dispositivo que detonará en treinta segundos tras ser desplegado. Si el equipo atacante consigue desplegar el dispositivo a tiempo y hacer tiempo para que este detone o eliminar al otro equipo por completo, ganarán la ronda, en otro caso, el equipo defensor habrá

ganado. Tras las 8 rondas, los roles se invierten para continuar con la partida.

Además, el juego dispondrá de un único modo de juego. Siempre habrá que conseguir la victoria por dos rondas de diferencia, es decir, en caso de llegar a un resultado de ocho rondas para cada equipo, será necesario decantar la balanza ganando dos rondas seguidas. Esta parte de la partida se considera como una "prórroga" en la que se invertirán los roles tras cada ronda y contarán con los mismos recursos para no beneficiar a ningún equipo.

4

Implementación y Desarrollo

En esta nueva sección se describe todo lo relacionado con la implementación del programa, es decir, todo lo que se ha realizado en la plataforma de Unreal. Para facilitar y estructurar la memoria se ha dividido este apartado en varios subpuntos en los que se trata de manera detallada todos los aspectos más importantes del desarrollo.

4.1. Personaje jugable

El personaje está controlado por el jugador que, con los controles de teclado y ratón, puede realizar las acciones que están disponibles.

En Unreal Engine, cada *blueprint* puede tener varios componentes dentro del mismo para realizar una clase con la que interactuar. En este caso, el personaje está conformado de los siguientes componentes:

- *Capsule Component*: Es un componente con forma de cápsula que envuelve al resto de elementos y que actúa como colisión con el objetivo de evitar que el personaje atraviese paredes [6].
- *Arrow Component*: Es un componente que sirve para indicar hacia donde mira el objeto [4].
- *Mesh*: En *Unreal Engine*, un *mesh* es una representación tridimensional de un objeto, compuesta por una colección de vértices, aristas y caras que definen su forma. Los *meshes* se utilizan para construir elementos visibles en el mundo del juego, como personajes, armas, edificios o decoraciones. Existen dos tipos principales: los *Static Meshes*,

que no tienen animaciones, y los *Skeletal Meshes*, que están compuestos por un esqueleto interno que permite su animación mediante huesos.

- *First Person Camera*: es la cámara por la que ve el jugador. Tiene un *mesh* como hijo que solo sirve para el inspector ya que luego está oculta en el juego.

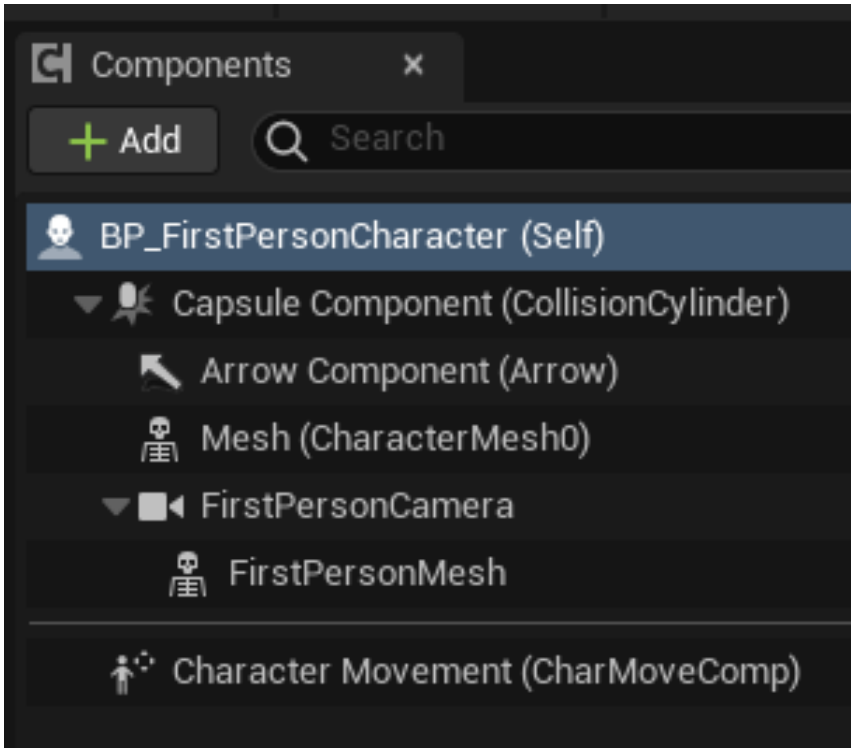


Figura 17: Árbol de componentes del *blueprint* del jugador

4.1.1. *Assets* del personaje

En cuanto al *asset* elegido para el personaje, se ha elegido un personaje con estilo militar que se adecúa a la temática del juego. Para ello, dentro de la biblioteca de *assets* de FAB.com se ha escogido el siguiente[22]:

Figura 18: *Asset* del personaje

El *asset* se distribuye bajo la licencia CC-BY, la cual permite el uso libre del *asset* siempre y cuando se reconozca la autoría de la obra [3].

En Unreal Engine, el *assets* se importa de forma automática y crea varios archivos entre ellos un static *mesh* que es el que utilizaremos y asignaremos al *Mesh* de la figura 17

4.1.2. *Blueprint* de funcionalidad

Este *blueprint* es el que incluye todas las funciones que el jugador puede realizar. Para simplificar su resumen en esta memoria, he realizado una tabla con cada función disponible y una descripción de cada una de ellas para facilitar al lector su entendimiento. Las funciones que se pueden realizar por el jugador son las que están disponibles en los *input mappings context*

Función	**Descripción**
EnhancedInputActionIA_Look	Hace que la cámara esté controlada por el ratón.
EnhancedInputActionIA_Move	Hace que el jugador se mueva con las teclas que se hayan introducido en el *Input Action* (las teclas son A, W, S, D).
EnhancedInputActionIA_Jump	Hace que el jugador realice un salto.
EnhancedInputActionIA_Run	Incrementa la velocidad del jugador.
EnhancedInputActionIA_Crouch	Decrementa la velocidad del jugador y hace que el jugador se agache ocupando menos espacio.
EnhancedInputActionIA_Shoot	Si el jugador tiene un arma equipada realiza un disparo llamando a la función que contiene el *blueprint* del arma.
EnhancedInputActionIA_Reload	Si el jugador tiene un arma equipada y no tiene el cargador al máximo, recarga el arma equipada.
EnhancedInputActionIA_Aim	Si el jugador tiene un arma equipada, esta acción permite realizar un zoom en la cámara para que sea más fácil apuntar en largas distancias.
EnhancedInputActionIA_GetWeapon	Si la variable *equippedWeapon* no es nula, alterna entre mostrar y ocultar el arma.

EnhancedInputActionIA_FragGranade	Si el número de granadas de fragmentación es mayor que cero el jugador se equipa con una granada de fragmentación que puede lanzar para hacer daño al rival.
EnhancedInputActionIA_FlashGranade	Si el número de granadas de *flash* es mayor que cero el jugador se equipa con una granada de *flash* que puede lanzar para hacer cegar al rival.
EnhancedInputActionIA_SmokeGranade	Si el número de granadas de humo es mayor que cero el jugador equipa se equipa con una granada de humo que puede lanzar para generar una nube de humo que impide la visión del personaje a través de él.
EnhancedInputActionIA_LaunchGranade	Si el personaje tiene equipada un arma al pulsar lanza la granada en dirección a donde este apuntando con una fuerza predeterminada.
EnhancedInputActionIA_OpenShopMenu	Permite al jugador abrir el menú de compra.

Cuadro 1: Funciones del *blueprint* del personaje

Como ejemplo, vamos a incluir una imagen de la función *EnhancedInputActionIA_Aim*:

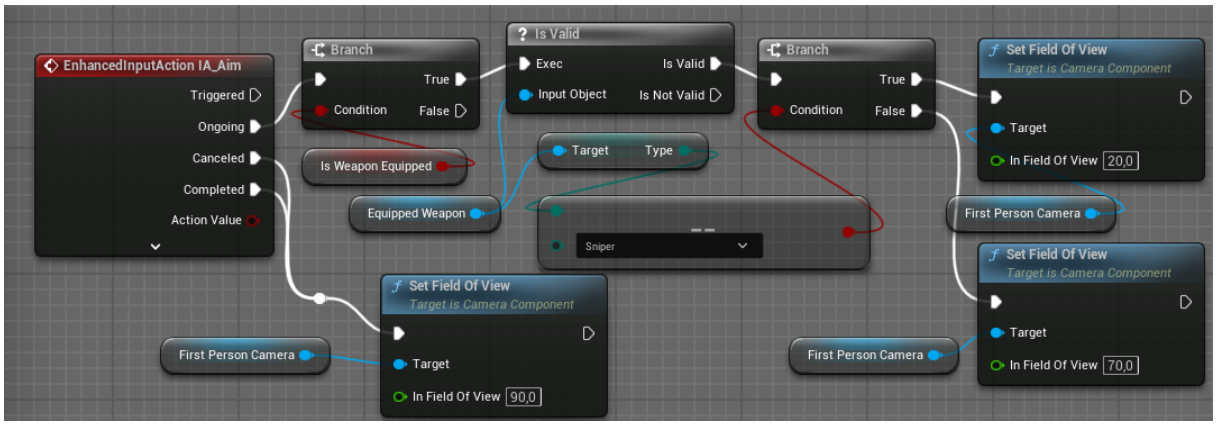


Figura 19: Función de apuntar (*EnhancedInputActionIA_Aim*) en *blueprints*

En la imagen 19 podemos ver como la función tiene varios parámetros que configuran el modo de entrada de teclado, por ejemplo, en este caso es *Ongoing*, lo que significa que realiza la función mientras detecte la entrada de teclado. Tras ello, verifica si el personaje tiene un arma equipada y si esta no es nula. Tras ello comprueba el tipo de arma que es, y en caso de ser un *sniper*, el zoom de la cámara sera mucho mayor que se ajusta con el parámetro *field of view*.

4.1.3. *Blueprint* de animación

El jugador tiene un *blueprint* de animación propio que ajusta la animación realizada en cada momento según unos parámetros definidos. Unreal Engine facilita esta implementación con los *blueprints* de animación. Estos *blueprints* permiten crear máquinas de estados con las que realizar transiciones entre animaciones, es decir, cada estado de la máquina realiza una animación y cuando se dan ciertas condiciones de transición, la máquina cambia el estado y, por tanto, cambiando la animación realizada. Además, también permite la transición entre varias máquinas de estado para facilitar la escalabilidad de la misma.

En este *blueprint* se han definido tres máquinas de estado:

- *LocomotionMT*: define la animación por defecto del jugador.
- *CrouchMT*: define la animación cuando el jugador esta agachado.
- *AimingMT*: define la animación cuando el jugador tiene un arma equipada.

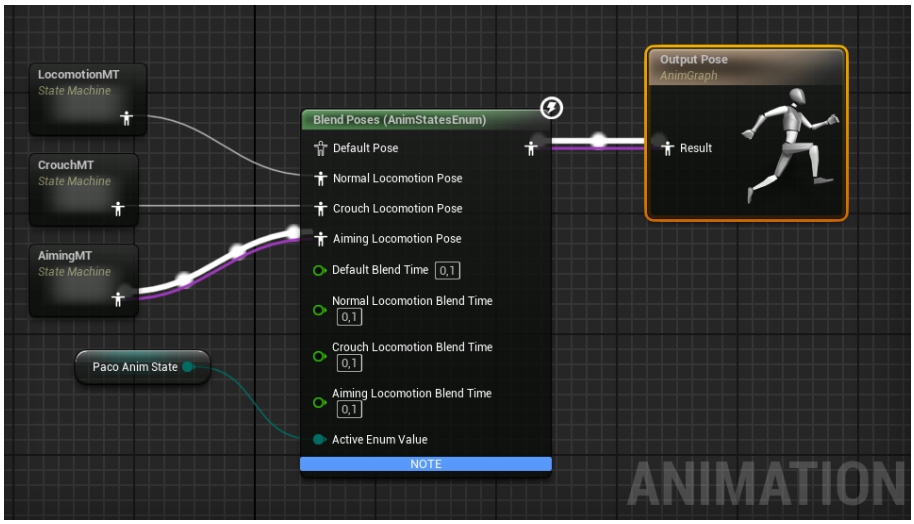


Figura 20: *Blueprint* de animación

Dentro de cada nodo denominado como *State Machine* se encuentra una máquina de estados que según las variables definidas para controlar las transiciones entre estados. Para cambiar entre máquina de estados se utiliza una variable de tipo enumerado *AnimStateEnum* en la que guardo la máquina de estados a utilizar en cada momento.

Por ejemplo, cuando la variable enumerada es igual a *NormalLocomotion* la máquina de estados utilizada es *LocomotionMT*. Cada máquina de estado se dibuja de la siguiente forma:

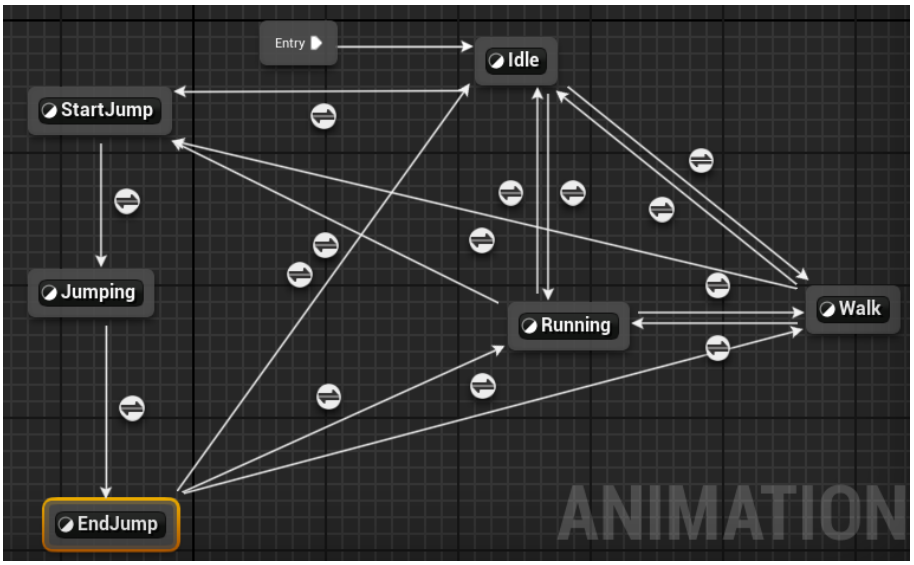


Figura 21: Máquina de estados de *LocomotionMT*

Donde en un estado se realiza:

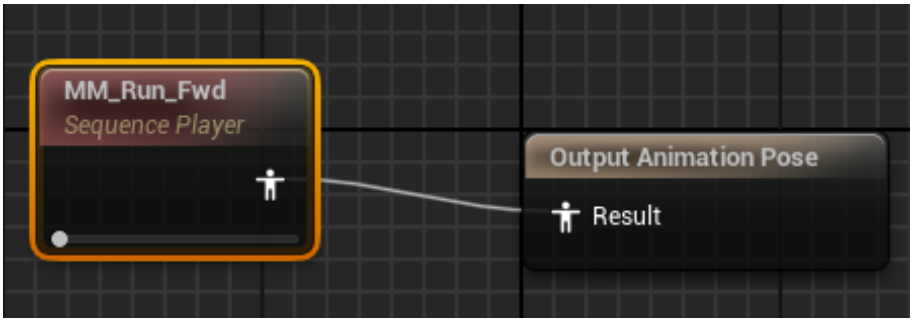


Figura 22: Situación específica de la máquina de estados *LocomotionMT*

4.1.4. *Blueprint* de *player controller*

Un *PlayerController* es la interfaz entre el *Pawn* (personaje o entidad controlable) y el jugador humano que lo maneja. Representa esencialmente la voluntad del jugador, transmitiendo

sus acciones (entradas de teclado, ratón o *gamepad*) al personaje en el mundo del juego [12]. Su principal función es interpretar las entradas del usuario y traducirlas en comportamientos dentro del juego, como moverse, disparar o interactuar.

4.2. Armas

En esta sección se detalla el sistema de armas implementado en el videojuego, tanto desde el punto de vista funcional como estructural. Se explica la lógica seguida en el diseño de *Blueprints* para gestionar la selección, disparo, recarga y comportamiento general de cada arma. Además, se describe la organización modular empleada para facilitar la reutilización y ampliación del sistema, permitiendo añadir nuevas armas sin modificar la estructura base.

El desarrollo del sistema de armamento se ha realizado íntegramente mediante *Blueprints* de Unreal Engine 5, priorizando una lógica clara, escalable y fácilmente mantenible. Para ello, se ha creado una estructura de herencia.

A lo largo de esta sección se analizan tanto los componentes técnicos como las decisiones de diseño que han guiado la creación de este subsistema clave en la jugabilidad del proyecto.

4.2.1. Sonidos y *assets* del arma

En cuanto al *asset* elegido para las armas, he elegido unos *assets* con estilo realista de la bibliteca de FAB.com. Para ello he utilizado los siguientes *assets*:

Figura 23: Primer *asset* de armas [26]

Figura 24: Segundo *asset* de armas [25]

Todos los *assets* adicionales empleados en el desarrollo de las armas se encuentran referenciados en la bibliografía al final de este documento [24] [23].

Los efectos de sonido utilizados en el proyecto han sido obtenidos de recursos con licencia libre, lo que permite su uso, modificación y distribución sin restricciones comerciales. Estos sonidos han sido empleados para representar acciones clave dentro del juego, como disparos, recargas o explosiones, mejorando así la inmersión del jugador. Todos los sonidos han sido descargados desde una plataforma especializada en recursos de audio libres de derechos, la cual se detalla en la bibliografía del presente documento [27].

Cada arma del juego cuenta con un sonido característico de disparo, diseñado para reflejar su tipo y potencia (por ejemplo, un sonido seco para una pistola o un disparo potente para un rifle de francotirador). Además, todas las armas comparten un sonido común de recarga fallida o ”sin balas”, que se activa cuando el jugador intenta disparar sin munición disponible. Estos sonidos han sido implementados mediante el sistema de *Sound Cue* de Unreal Engine, que permite combinar y controlar múltiples archivos de audio de forma visual, ofreciendo un sistema flexible para gestionar variaciones, prioridades o condiciones de reproducción [14].

4.2.2. *Blueprint* de la clase abstracta

En primer lugar, se ha diseñado una clase base abstracta para representar el comportamiento general de las armas del juego. Esta clase no puede ser instanciada directamente, ya

que su propósito es servir como estructura común para todas las armas específicas. En ella se definen tanto las variables compartidas (como el daño, la cadencia de disparo, la capacidad del cargador o el tiempo de recarga) como las funciones fundamentales, tales como disparar, recargar o reproducir sonidos. Esta implementación permite que cada subclase (por ejemplo, una escopeta, un rifle o una pistola) herede esta lógica base y pueda sobrescribir o modificar los valores y comportamientos necesarios sin duplicar código, facilitando la escalabilidad y el mantenimiento del sistema de armamento.

En Unreal Engine, una clase se puede indicar como abstracta desde el propio inspector indicando el siguiente *checkbox*:

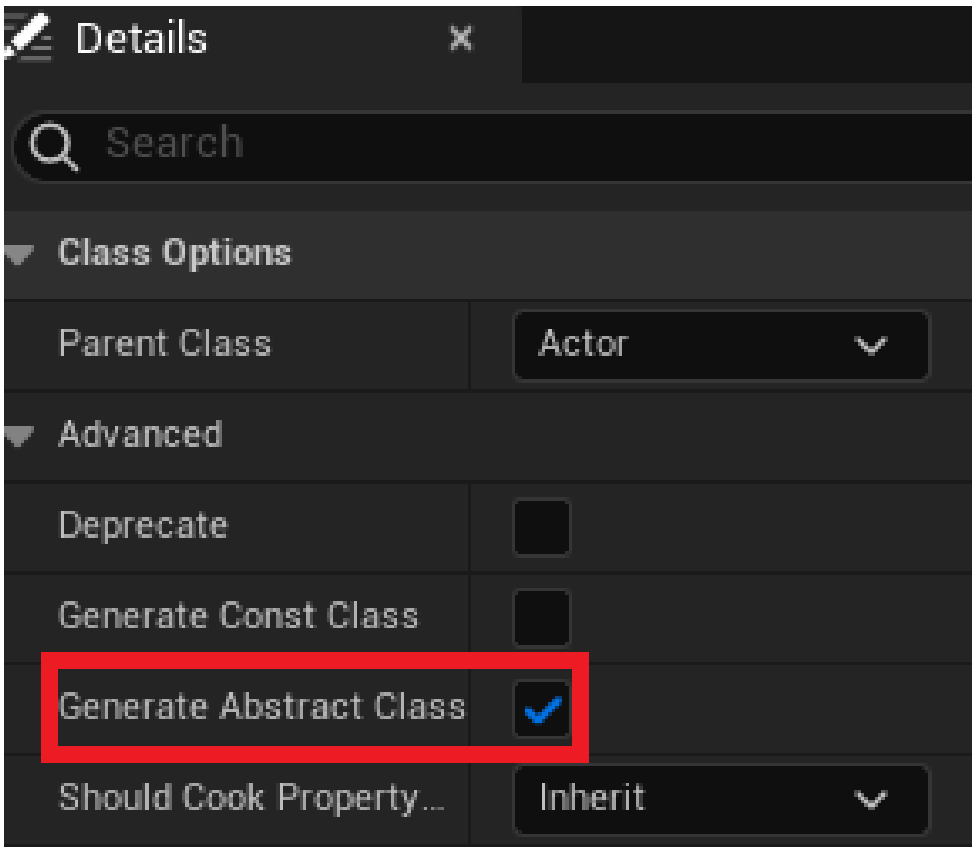


Figura 25: Clase abstracta en el inspector

En cuanto a las funciones implementadas en el *blueprint* son las funciones que cumple un arma principalmente: disparar y recargar.

La función de disparo ha sido implementada utilizando *Line Trace By Channel* , una función que genera un vector entre dos puntos definidos por coordenadas espaciales y devuelve el primer objeto con el que dicho vector colisiona. Esto permite, por ejemplo, detectar impactos

en partes específicas del cuerpo del enemigo (como un hueso determinado) y aplicar un daño ajustado en función del área alcanzada[17].

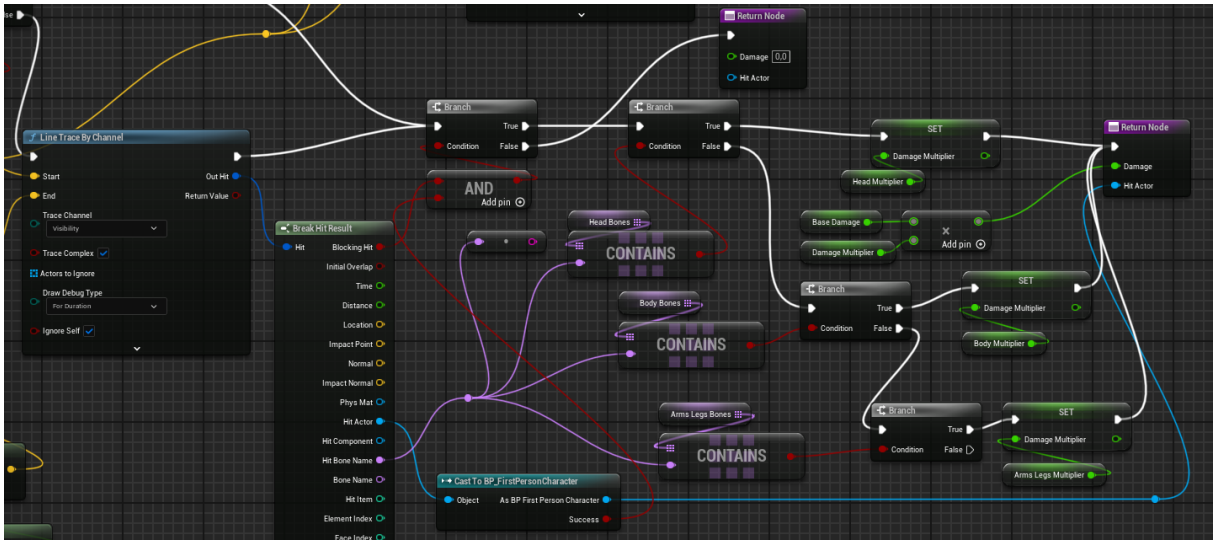


Figura 26: Salida del *Line Trace*

Además, para añadir un componente táctico y realista al sistema de disparo, se ha incorporado un margen de error mediante la función *Random Unit Vector in Cone Degrees* [13]. Esta función genera un vector aleatorio de longitud unitaria dentro de un cono definido, utilizando una distribución uniforme. Gracias a esto, se simula la dispersión del disparo, introduciendo variabilidad en la dirección de los proyectiles y evitando trayectorias perfectamente rectas, lo cual mejora la jugabilidad y realismo del combate.

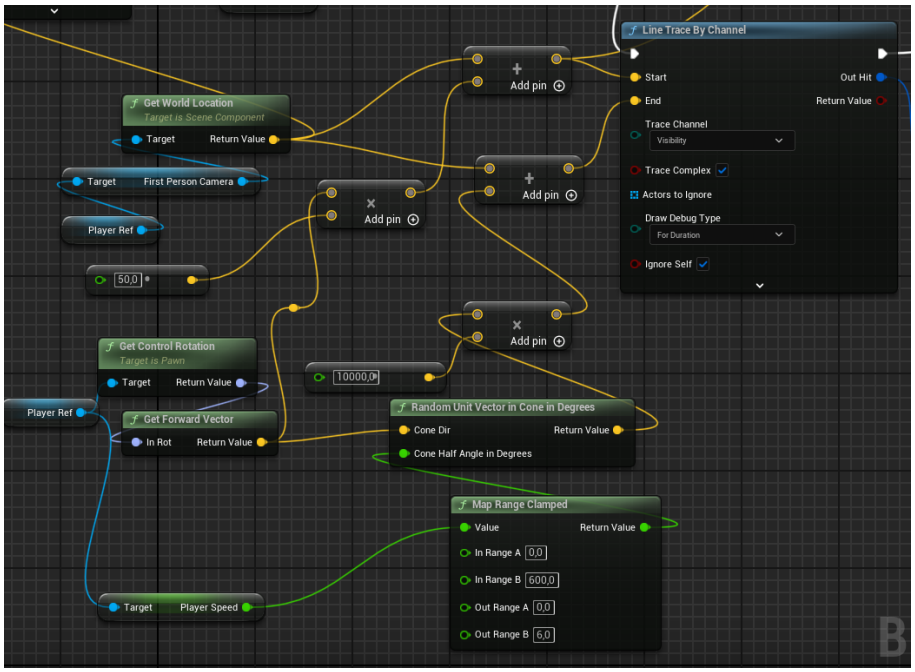


Figura 27: Implementación en *blueprint* de la dirección del *Line Trace*

El sistema de recarga de armas ha sido implementado mediante *Blueprints* y se activa al ejecutar la función *Reload*. Este evento evalúa dos condiciones clave antes de permitir la recarga: primero, comprueba si el número de balas en reserva (*Max Bullets*) es mayor que cero, y segundo, si el cargador actual (*Bullets*) tiene menos balas que su capacidad máxima (*Mag Tam*). Ambas condiciones se combinan con un nodo lógico *AND*; si ambas se cumplen, el flujo pasa por el nodo *Branch* (condicional) y se considera que la recarga es válida.

Una vez verificada la condición, se reproduce un sonido asociado a la recarga mediante la función *Play Sound at Location*, utilizando el sonido almacenado en *Reload Sound* y la posición del jugador obtenida mediante *Get Actor Location*. Esto proporciona un *feedback* auditivo inmediato que refuerza la acción y mejora la experiencia del jugador.

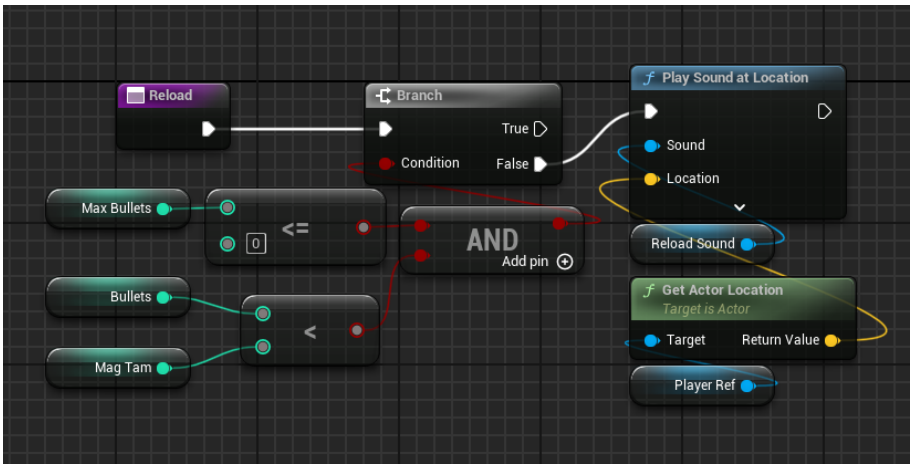


Figura 28: Función de recargar. Retroalimentación al jugador

La lógica encargada de actualizar las balas tras la recarga se encuentra en la función *Apply Reload*. Esta se ejecuta una vez se ha confirmado que el jugador puede recargar el arma. En primer lugar, se calcula la cantidad de balas necesarias para llenar el cargador (*Need Bullets*), restando las balas actuales (*Bullets*) de la capacidad máxima del cargador (*Mag Tam*). A continuación, se evalúa cuántas balas pueden ser realmente recargadas (*To Reload Bullets*), tomando el valor mínimo entre las necesarias y las disponibles en la reserva (*Max Bullets*). Esto evita que se sobrecargue el arma o que se usen más balas de las disponibles.

Una vez determinada la cantidad que se va a recargar, se actualizan las variables del sistema: se suma *To Reload Bullets* a las balas del cargador (*Bullets*) y se resta esa misma cantidad de la reserva (*Max Bullets*). Esta estructura garantiza una gestión coherente de la munición y previene errores como recargas infinitas o inconsistencias entre el cargador y la reserva.

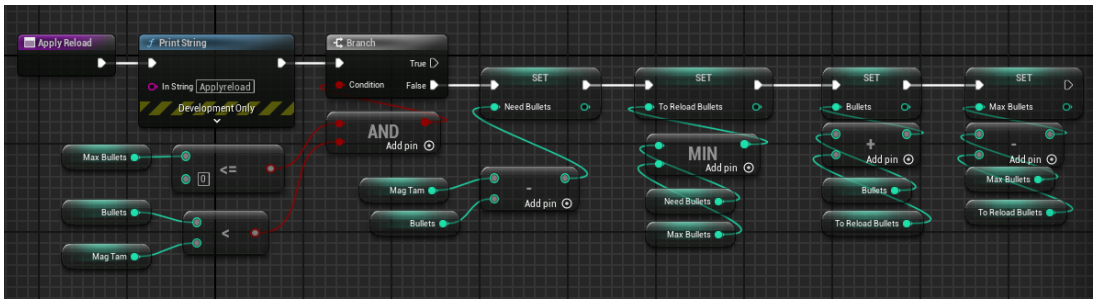


Figura 29: Función de recargar. Aplicar recarga

4.2.3. *Blueprints* de armas instanciables

Las armas instanciables del proyecto se han desarrollado a partir de la clase base abstracta de arma. Gracias a la herencia, cada arma específica únicamente necesita definir sus propios valores (como el daño, la cadencia de disparo, el tamaño del cargador o el tiempo de recarga), ya que toda la lógica común está ya implementada en la clase padre. Para crear una nueva arma, basta con duplicar una clase existente e introducir las modificaciones necesarias en sus variables, sin necesidad de reescribir funciones ni alterar el comportamiento base. Este enfoque basado en herencia y modularidad favorece la escalabilidad del sistema, permitiendo incorporar fácilmente nuevas armas, y mejora el control y mantenimiento de estadísticas, ya que los parámetros son fácilmente editables.

Además, se han desarrollado diferentes tipos de armas con características diferenciadas según su función en combate. Entre ellas se incluyen **fusiles de asalto** (como el M4 y el AK47), **escopetas** (tanto de postas como recortadas), **subfusiles** (como la Micro-Uzi), una **pistola** (Glock) y un **rifle de francotirador** (*sniper*). Esta variedad permite ofrecer al jugador estilos de juego distintos según sus preferencias y necesidades tácticas durante la partida.

A continuación, se resumen las características principales de cada tipo de arma:

- **Fusiles de asalto (M4, AK47):** eficaces en distancias medias y largas, con una cadencia de disparo moderada que permite mantener el control y la precisión.
- **Subfusiles (Micro-Uzi):** óptimos para combates a corta distancia, con una **cadencia de disparo muy alta**, pero menor precisión a largas distancias.
- **Escopetas (postas y recortada):** disparan un total de **8 perdigones** por disparo, con una **dispersión significativa**, lo que las hace efectivas a **muy corta distancia** y en espacios cerrados.
- **Pistola (Glock):** arma secundaria equilibrada, útil en situaciones de emergencia o cuando no se dispone de otra arma.
- **Rifle de francotirador:** diseñado para combates a **larga distancia**, con gran potencia por disparo y baja cadencia, ideal para eliminar enemigos a distancia con precisión.

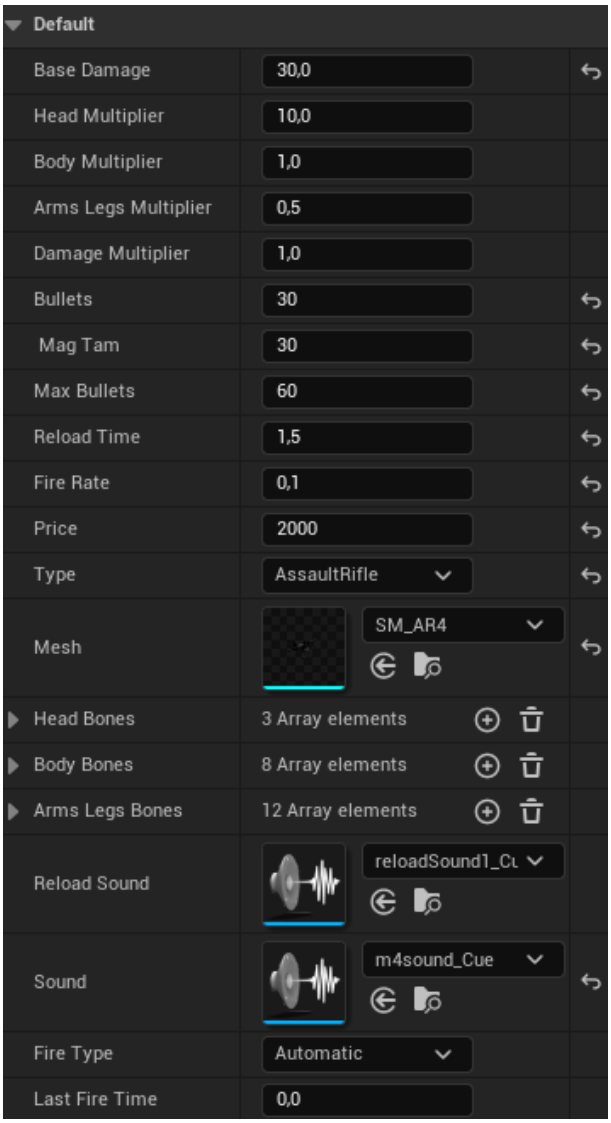


Figura 30: Variables en el inspector para el *blueprint* del arma M4

4.2.4. Estadísticas y datos

En esta sección se muestran los datos de cada arma, en los que se muestra en otras cosas su nombre, su daño y su cadencia de disparo. Para facilitar la lectura se disponen en forma de tabla:

Nombre	**Tipo**	**Cadencia de disparo**	**Tamaño del cargador**	**Tiempo de recarga**	**Daño (cabeza/cuerpo/piernas)**
M4	Fusil de asalto	Alta	30	1.5	300/30/15
AK47	Fusil de asalto	Media	25	1.3	400/40/20
Micro Uzi	Subfusil	Muy alta	40	1.0	54/18/9
Recortada	Escopeta	Baja	2	2	100/10/5*
Escopeta de postas	Escopeta	Baja	4	2	160/16/8*
Barret	Rifle de francotirador	Baja	5	4	999/100/50
Glock	Pistola	Media	10	1	60/20/10

Cuadro 2: Tabla de armas y características principales

** El daño indicado para las escopetas representa el daño por cada perdigón.*

4.3. Granadas

En este apartado se detalla la implementación del sistema de granadas dentro del videojuego, siguiendo una estructura similar a la utilizada en el sistema de armas. Para ello, se ha creado una clase base abstracta que contiene las propiedades y comportamientos comunes a todos los tipos de granadas, como el temporizador de detonación, el modelo visual o las referencias a efectos de sonido y partículas. A partir de esta clase, se han generado distintos *Blueprints* que heredan de la clase abstracta, los cuales implementan su propia lógica específica en la función de explosión.

Cada tipo de granada sobrescribe o amplía el comportamiento del método de explosión (*Explode*), permitiendo así definir efectos particulares como daño en área, ceguera temporal o bloqueo de visión con humo. Esta estructura basada en herencia permite mantener el código organizado y facilita tanto la reutilización como la escalabilidad del sistema, permitiendo

añadir nuevos tipos de granadas de forma sencilla.

4.3.1. *Assets* de las granadas

En cuanto al *asset* elegido para las granadas, he elegido unos *assets* con estilo realista de la bibliteca de FAB.com. Para ello he utilizado los siguientes *assets*:

Figura 31: Smoke *asset* [21]

El resto de *assets* están referenciados en la bibliografía final [20]

4.3.2. *Blueprint* de la clase abstracta

En primer lugar, se ha diseñado una clase base abstracta para representar el funcionamiento general de las granadas del juego. Esta clase no puede ser instanciada directamente, ya que su propósito es servir como interfaz común para todas las granadas.

En ella, se ha definido un único parámetro, *Fuse Time*, que es una variable de tipo *float* que indica el tiempo que tarda en explotar la granada.

Por su parte, se le ha añadido un elemento llamado *Projectile Movement Component* , se encarga de actualizar la posición de otro componente durante cada *tick*. Este tipo de componente admite comportamientos avanzados como rebotes tras colisiones o movimiento guiado hacia un objetivo (*homing*) [11]. Por defecto, se aplica el movimiento al componente raíz del actor propietario, aunque es posible seleccionar otro componente específico mediante la función *SetUpdatedComponent*. Si el componente actualizado está simulando físicas, únicamente se ten-

drán en cuenta los parámetros iniciales de lanzamiento (por ejemplo, la velocidad inicial) y, a partir de ese momento, el comportamiento del proyectil quedará gestionado completamente por el sistema de física.

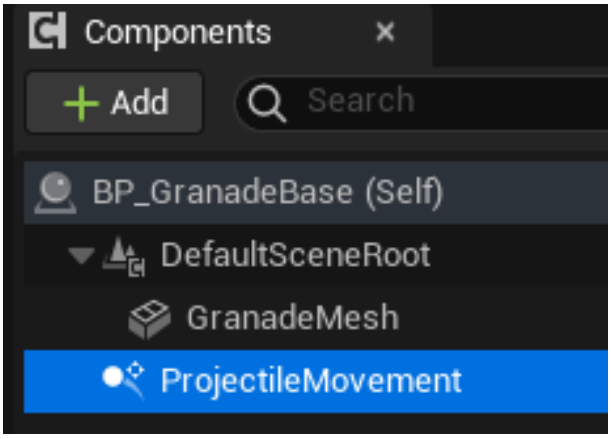


Figura 32: *Projectile Movement Component*

4.3.3. *Blueprints* de granadas instanciables

A continuación, se enumeran los tipos de granadas implementados en el proyecto:

- **Granada de fragmentación:** provoca daño en área al explotar, siendo efectiva para eliminar enemigos en espacios reducidos o tras coberturas.
- **Granada cegadora (*flash*):** genera un destello de alta intensidad que desorienta visualmente al jugador afectado durante unos segundos, simulando una ceguera temporal.
- **Granada de humo:** al detonar, crea una nube de humo que bloquea la visibilidad en una zona determinada, permitiendo ocultarse o cruzar áreas expuestas sin ser detectado. Esta granada cuenta con un sistema de partículas *niagara* [7] y un punto de aparición desde el cual aparecen.

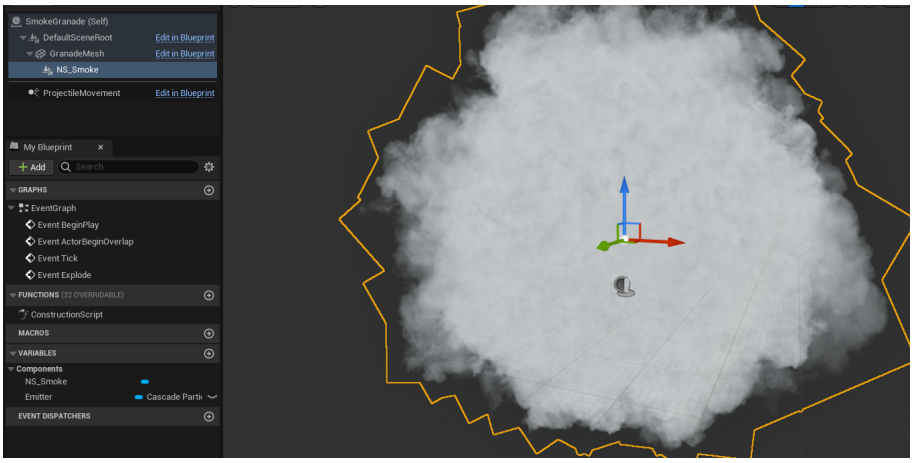


Figura 33: Granada de humo en el inspector

4.4. Modo de juego

El modo de juego implementado en este proyecto ha sido estructurado a través del componente *GameMode*, una clase central en Unreal Engine que define las reglas, condiciones de victoria, lógica de inicio y gestión de jugadores dentro de una partida [10]. Esta clase actúa como el núcleo del comportamiento del juego a nivel global, y en este caso ha sido personalizada para adaptar las condiciones de victoria, el reinicio de partidas y otras mecánicas específicas del modo multijugador competitivo desarrollado.

En esta sección se detalla el funcionamiento del modo de juego configurado, así como su interacción con otros componentes del sistema, incluyendo el control del marcador, reinicio de personajes al morir y finalización de la partida al cumplirse las condiciones de victoria.

4.4.1. Cambios con la idea inicial

A lo largo del desarrollo se han explorado y aplicado distintas configuraciones del *GameMode*, adaptadas a los requisitos de cada tipo de partida. Inicialmente, el modo de juego principal estaba orientado a un sistema por rondas, en el que dos equipos se enfrentaban con roles de atacante y defensor, alternando posiciones tras cada conjunto de rondas. Este sistema se basa en condiciones estrictas de victoria (alcanzar 9 rondas con una diferencia mínima de dos), lo que implica un control más complejo del flujo de juego, reinicio de personajes y gestión del marcador. Todo este comportamiento ha sido gestionado desde el *GameMode* personalizado, en combinación con el *GameState* y el *PlayerController*, que coordinan la lógica del equipo,

las condiciones de victoria y el reinicio entre rondas [10].

Como alternativa a este sistema estructurado por rondas, se ha desarrollado un modo de juego adicional más ágil y dinámico, enfocado en el combate individual. En esta modalidad, el mapa cuenta con múltiples puntos de aparición (*spawns*) distribuidos estratégicamente. Cada vez que un jugador reaparece, puede seleccionar un arma desde un menú de compra simplificado y se enfrenta al otro jugador con el objetivo de eliminarlo. Cuando un jugador muere, resurge en un punto diferente del mapa y tiene la oportunidad de cambiar de arma, fomentando la experimentación y la adaptación táctica. El ciclo se repite hasta que uno de los dos jugadores alcanza cinco eliminaciones, condición que define el final de la partida.

Este segundo modo se centra más en la acción directa y en el desarrollo de habilidades individuales, con una lógica de control más simple pero igualmente gestionada desde el *GameMode*. A diferencia del modo por rondas, no requiere reinicios globales ni control por equipos, sino una lógica basada en eventos individuales, lo que demuestra la versatilidad del sistema de *GameMode* en Unreal Engine para adaptar distintos estilos de juego dentro del mismo proyecto.

4.4.2. Desarrollo

Para el desarrollo del modo de juego se ha creado un *blueprint* de *GameMode* de Unreal Engine, FiveKillsGM, en el cual se han definido las reglas descritas en el apartado anterior. Dentro del *blueprint* se han definido varios *Customs Events* [8].

Los *customs events* definidos dentro del *blueprint* son los siguientes:

- ***OnPostLogin*:** se ejecuta automáticamente cada vez que un nuevo jugador se conecta al servidor. Este evento forma parte del ciclo de inicialización gestionado por el *GameMode* y se encarga de crear y posicionar el personaje controlable del jugador recién conectado.

  En primer lugar, se obtiene una lista de todos los puntos de aparición del tipo *Target Point* en el nivel mediante la función *Get All Actors Of Class*. Estos puntos se almacenan en una variable (*Respawns*) para poder reutilizarlos durante el resto de la partida. A continuación, se selecciona aleatoriamente uno de estos puntos y se obtiene su ubicación con la función *Get Actor Location*.

  Con esa información, se ejecuta el nodo *Spawn Actor*, que instancia un nuevo personaje

(en este caso, del tipo *BP First Person Character*) en la posición seleccionada. Finalmente, se asigna el control del personaje al jugador que se ha conectado mediante el nodo *Possess*, vinculando al controlador (*New Player*) con el personaje recién creado.

Esta lógica permite que cada jugador entre en la partida de forma independiente y automática, apareciendo en una posición válida y comenzando a jugar sin intervención manual, lo que es fundamental para el funcionamiento de partidas multijugador en tiempo real.

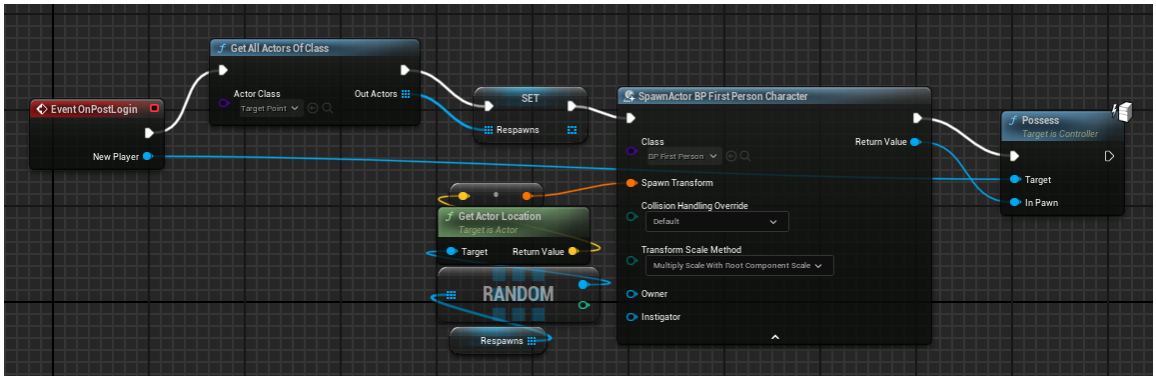


Figura 34: Aparición del personaje al conectarse a una partida

- ***RespawnearJugador*:** se ejecuta cuando un jugador muere. Esta función recibe como parámetros el personaje eliminado (*Dead Character*) y el jugador que ha realizado la eliminación (*Killer*). En primer lugar, se incrementa el contador de eliminaciones (*Kills*) del jugador que ha realizado la baja. A continuación, se comprueba si el número de eliminaciones de dicho jugador ha alcanzado el valor objetivo establecido en la variable *Kills to Win*. Si se cumple esta condición, se llama a la función Fin Partida, que finaliza la partida.

  Si aún no se ha alcanzado la condición de victoria, el jugador eliminado se reposiciona en el mapa mediante la función *Set Actor Location*, utilizando una ubicación aleatoria elegida desde una lista de puntos de *respawn* (*Respawns*). Además, se restablecen sus estadísticas iniciales: su salud se pone al máximo (M*ax Health*) y se le equipa el arma por defecto (*Equipped Weapon*).

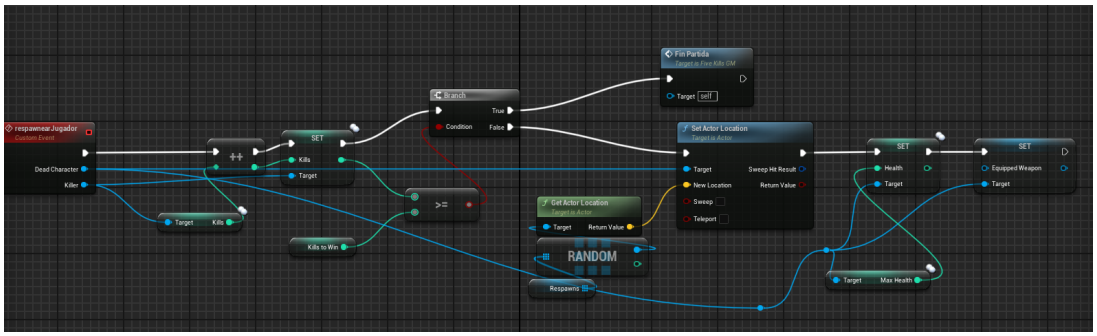


Figura 35: *Custom Event: Respawnear* Jugador

- **finPartida:** diseñado para gestionar el cierre de la partida cuando uno de los jugadores alcanza el número de eliminaciones requerido para ganar. En primer lugar, se obtiene una referencia a todos los actores del tipo *BP First Person Character* presentes en el nivel mediante la función *Get All Actors Of Class*. A continuación, se recorre este *array* con un *For Each Loop* y se evalúa si alguno de ellos ha alcanzado o superado el umbral de eliminaciones (en este caso, cinco *kills*).

  Si se cumple esta condición, se muestra una pantalla final personalizada al jugador correspondiente mediante el nodo Mostrar Pantalla Final, indicando que ha ganado. Al resto de jugadores también se les muestra la misma pantalla, pero sin marcar el estado de victoria.

  Una vez gestionado este *feedback*, se aplica un retraso de cinco segundos (*Delay*) antes de ejecutar la transición al siguiente nivel. Pasado ese tiempo, se vuelve a recorrer el *array* de jugadores y se cambia de nivel mediante el nodo *Open Level (by Name)*, que redirige a todos los jugadores al nivel llamado *lobby*, donde pueden iniciar una nueva partida o abandonar la sesión.

  Esta lógica centralizada dentro del *GameMode* permite un cierre controlado, sincronizado y visualmente claro de cada partida, asegurando que todos los jugadores reciban *feedback* del resultado y que la transición hacia el siguiente estado del juego sea fluida y automática.

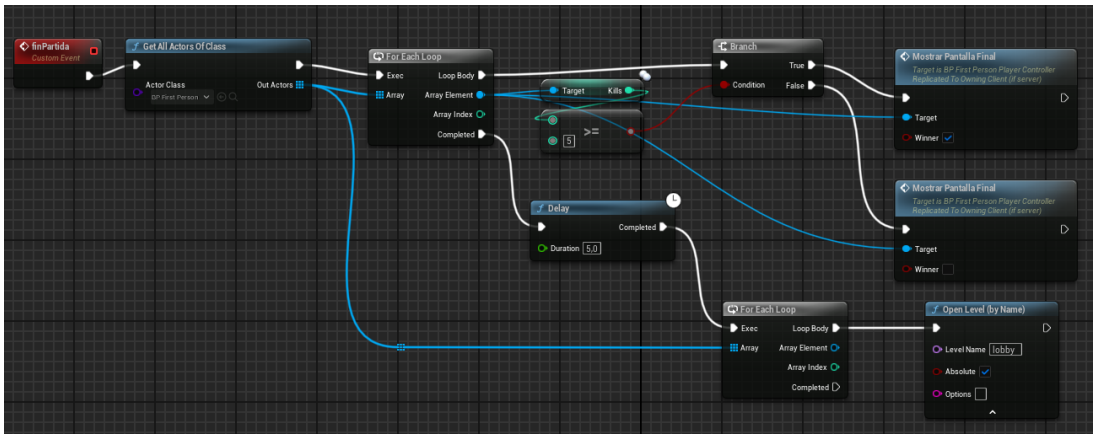


Figura 36: *Custom Event*: Fin de la partida

4.5. Menús

En este apartado se detalla la implementación de los distintos menús del videojuego, desarrollados mediante el sistema de *Widget Blueprints* de Unreal Engine [18]. Esta herramienta permite crear interfaces gráficas de usuario (*UI*) de forma visual e intuitiva, combinando diseño y lógica mediante nodos en *Blueprint*. A través de los *widgets*, se ha construido una interfaz funcional, clara y coherente con el estilo del juego.

Los menús creados incluyen el menú principal, desde el que el jugador puede iniciar la partida o salir del juego; el menú de selección de armas, que permite al jugador elegir su equipamiento antes de reaparecer y la pantalla final, que muestra el resultado de la partida indicando el ganador. Todos estos elementos han sido implementados con lógica propia en cada *widget*, y se comunican con el *PlayerController* y el *GameMode* para activar comportamientos concretos como cargar un nuevo nivel, asignar un arma o mostrar el resultado al terminar la partida.

En los siguientes apartados se explican los pasos seguidos para la construcción de cada uno de estos menús, así como las decisiones de diseño tomadas y los eventos utilizados para su correcta integración con el resto del sistema de juego.

4.5.1. Menú principal

El menú principal del videojuego ha sido desarrollado mediante un *Widget Blueprint* llamado *menu_lobby*, estructurado visualmente a través de un *Canvas Panel*, que permite posicionar

libremente todos los elementos de la interfaz. Este menú actúa como punto de entrada para el jugador en las partidas multijugador, permitiéndole iniciar una sesión como *host* o unirse a una partida existente.

Dentro del *canvas*, se han definido dos botones principales:

- **Botón CREAR *HOST*:** este botón permite al jugador iniciar una nueva partida como servidor (*host*). Al pulsarlo, se lanza la lógica de creación del servidor y se carga el nivel correspondiente. Internamente, este botón está vinculado a un evento que ejecuta la función de anfitrión de partida.

- **Botón UNIRSE:** permite al jugador conectarse a una partida ya iniciada por otro jugador. Este botón está vinculado a un campo de texto (*ipserver*), donde el usuario debe introducir la dirección IP del servidor al que desea conectarse. Una vez completado el campo y pulsado el botón, se ejecuta la lógica de conexión al servidor.

Adicionalmente, se han incluido elementos visuales como una imagen que actúa como fondo del menú, generando una ambientación coherente con el estilo del juego, y un botón de engranaje reservado para abrir el menú de ajustes. Además, se ha incluido una música ambiental para incrementar la experiencia de usuario.

Figura 37: Menú asociado al *lobby* en el inspector

En la parte superior derecha del menú principal se ha añadido un **botón en forma de cruz** con la funcionalidad de cerrar el juego. Este botón permite al jugador salir de la aplicación de forma directa desde la interfaz principal, sin necesidad de utilizar combinaciones externas del sistema operativo.

Al pulsarlo, se ejecuta la función integrada *Quit Game* de Unreal Engine, que finaliza correctamente la sesión del juego cuando se encuentra empaquetado. De este modo, se mejora la usabilidad y se ofrece una salida clara y accesible desde el primer menú del juego, alineándose con los estándares de diseño de videojuegos actuales.

4.5.2. Menú de compra

Durante la partida, el jugador tiene acceso a un **menú de compra** diseñado mediante *Widget Blueprints*, el cual se activa cuando el personaje reaparece y aún no tiene un arma equipada. Este menú, llamado *ShopMenu*, está compuesto por un *Canvas Panel* que contiene tres columnas de botones organizadas mediante contenedores del tipo *VerticalBox*, los cuales agrupan las armas disponibles por categorías.

- En la primera columna (*VerticalBoxLeft*) se encuentran botones para seleccionar armas secundarias o de corto alcance como la **Glock**, la **Shorty** y la **Micro Uzi**.
- La segunda columna (*VerticalBoxCenter*) agrupa las armas principales como la **AK47**, **M4**, la escopeta (**Pump**) y el **rifle de francotirador (Sniper)**.
- Por último, la tercera columna (*VerticalBoxCenter_1*) contiene las opciones de compra de granadas: **Frag Grenade**, **Flash Grenade** y **Smoke Grenade**.

Este sistema permite al jugador **elegir libremente su equipamiento antes de reincorporarse al combate**, reforzando la libertad táctica y el dinamismo del juego. Además, cada botón está vinculado a su correspondiente clase de arma o granada, de modo que al pulsarlo se instancia automáticamente el objeto seleccionado y se equipa al personaje. El diseño modular de este menú facilita tanto su extensión como su mantenimiento, y permite limitar la compra de armas solo en momentos clave (como al reaparecer), lo que evita su uso abusivo durante el combate activo.

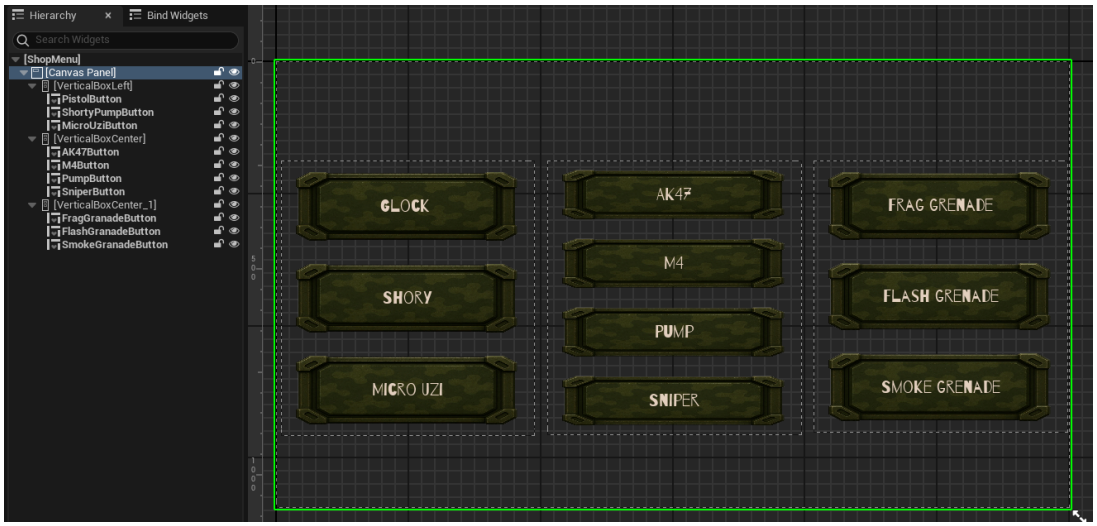


Figura 38: Menú de compra en el inspector

4.5.3. Menú de ajustes

El menú de ajustes permite al jugador consultar los controles del juego de manera visual e intuitiva. Está compuesto por una pantalla de fondo con el nombre del juego y dos botones principales: **CONTROLES** y **VOLVER**. Al acceder a la sección de controles, se muestra una imagen representativa de un teclado y un ratón donde se detallan todas las acciones que el jugador puede realizar, junto con la tecla o botón correspondiente.

- En el teclado se destacan las teclas usadas para el **movimiento** (W, A, S, D), así como otras teclas asignadas a funciones específicas como lanzar granadas (C), saltar (barra espaciadora), recargar (R), agacharse (Shift), caminar (Ctrl), y equipar diferentes tipos de granadas (Q, E, X).
- En el lado derecho se incluye un esquema del ratón, indicando que el **clic izquierdo** permite disparar y el **clic derecho** sirve para apuntar.

Este menú tiene como objetivo ofrecer una referencia rápida y clara al jugador, facilitando el aprendizaje de los controles sin necesidad de consultar documentación externa. Además, su integración visual con el resto del menú mantiene la coherencia estética del juego.

Figura 39: Menú de ajustes con esquema de controles

4.5.4. *HUD*

El ***HUD*** (*Heads-Up Display*) implementado en el videojuego proporciona al jugador la información esencial durante la partida sin interferir con la inmersión ni con la visibilidad del entorno. Ha sido desarrollado utilizando *Widget Blueprints* y se actualiza dinámicamente a través de referencias al *PlayerController* y al *GameState*.

En la parte superior de la pantalla se muestra el número de eliminaciones (***KILLS***), que se incrementa cada vez que el jugador consigue abatir a un oponente. En la parte inferior central se presenta el **estado de munición**, dividido en dos valores: el número de balas en el cargador actual y la cantidad total de balas disponibles en la reserva. Estos valores se actualizan en tiempo real cuando el jugador dispara o recarga, ya que en este sistema no existe la posibilidad de recoger munición adicional durante la partida.

Además, en la parte inferior derecha se muestran las **granadas adquiridas**, que se indican mediante un texto descriptivo junto al icono correspondiente, el cual aparece únicamente cuando el jugador ha comprado ese tipo de granada en el menú de compra. Esto permite al jugador identificar qué utilidades tiene disponibles en cada momento de forma clara y rápida.

El diseño del *HUD* ha sido concebido para ser funcional, minimalista y coherente con la estética general del juego, permitiendo al jugador acceder a información clave sin distraerlo

del combate.

Figura 40: Imagen del *HUD* en el juego

4.6. Multijugador

El sistema multijugador del proyecto ha sido implementado utilizando la funcionalidad de conexión en red que ofrece Unreal Engine 5. La conexión entre jugadores se gestiona desde el propio menú principal mediante dos botones: **Crear *Host*** y **Unirse**.

Cuando un jugador pulsa el botón **Crear *Host***, se crea una **sesión de escucha** (*listen server*), la cual permite que el jugador actúe como servidor y, al mismo tiempo, participe en la partida. A continuación, se carga el nivel multijugador configurado con las opciones necesarias para aceptar conexiones entrantes.

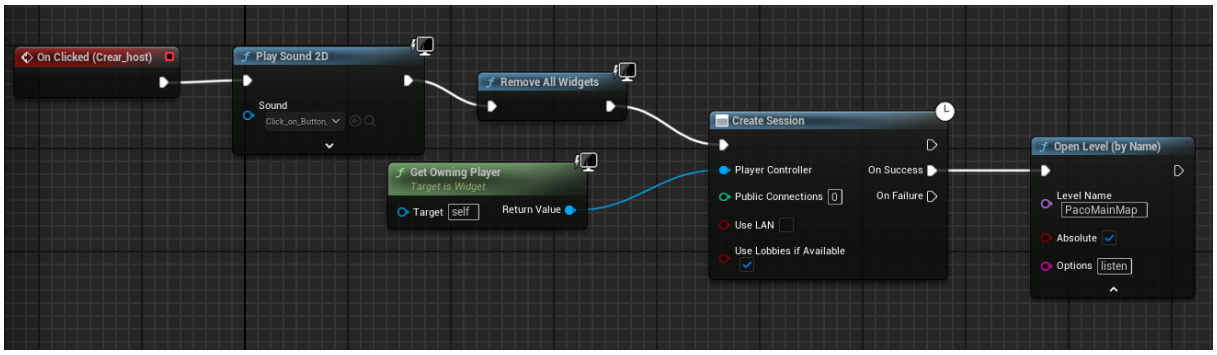


Figura 41: Secuencia de pasos internos para crear un *host*

Por otro lado, si otro jugador introduce una dirección IP válida en el campo de texto y pulsa el botón **Unirse**, el sistema intenta conectarse a esa dirección como cliente. Si la conexión tiene éxito, el jugador accede a la misma sesión y se sincroniza con el estado actual de la partida. En caso de fallo (por ejemplo, si la IP es incorrecta o no hay una sesión activa en esa dirección), el sistema devuelve al jugador al menú principal, permitiéndole intentarlo nuevamente.

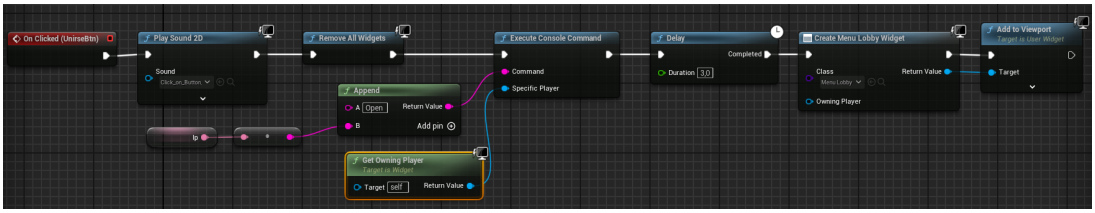


Figura 42: Secuencia de pasos internos para unirse a un *host*

El comando *open <IP>* en Unreal Engine 5 permite al cliente iniciar una conexión directa con un servidor multijugador mediante una dirección IP específica. Al ejecutarse este comando, el motor intenta conectarse a una sesión activa en el *host* indicado, que debe estar ejecutando el juego en modo de escucha (*listen server*). Si la conexión es exitosa, el cliente se une a la partida y comienza a recibir toda la información sincronizada del mundo, incluyendo el estado del nivel, los jugadores y sus posiciones.

Este sistema proporciona una base sencilla pero efectiva para partidas multijugador entre dos jugadores, sin necesidad de servidores dedicados, aprovechando las herramientas integradas de Unreal Engine.

La referencia utilizada es guiada de un tutorial de YouTube [28]

4.7. Desarrollo del mapa

El diseño del escenario se ha llevado a cabo siguiendo la estructura clásica de los videojuegos de disparos más conocidos, basada en un sistema de **tres carriles**. Esta organización espacial facilita la creación de rutas alternativas y zonas de enfrentamiento controladas, fomentando tanto el juego estratégico como el dinamismo durante las partidas.

Para la construcción del nivel se han utilizado principalmente *assets* gratuitos disponibles en la plataforma Fab, lo que ha permitido desarrollar un entorno visualmente coherente y detallado sin necesidad de modelado 3D personalizado. La mayoría de estos *assets* son de tipo *modular*, lo que ha facilitado la creación de estructuras con distintas formas y alturas reutilizando las mismas piezas base.

El mapa se ha organizado internamente utilizando una estructura de carpetas clara y jerárquica dentro del editor de Unreal Engine, lo que ha simplificado tanto la navegación por el nivel como el proceso de edición. En la Figura 43 puede observarse un ejemplo de dicha organización, donde cada conjunto de elementos del mapa se agrupa bajo una carpeta específica que representa su zona o funcionalidad (por ejemplo, *Casa1*, *Containers* o *DecoracionSpawnAtacante*).

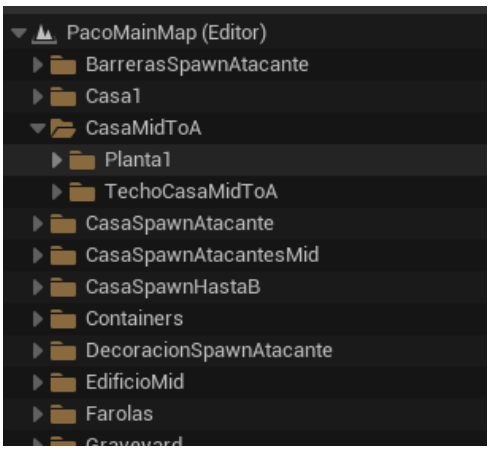


Figura 43: Estructura de carpetas del mapa en el editor de Unreal Engine

Para el diseño visual y jugable se ha priorizado la distribución de coberturas, accesos y líneas de visión equilibradas entre las distintas zonas del mapa. En las siguientes figuras se muestran diferentes capturas del entorno resultante desde distintos ángulos:

Figura 44: Vista general del mapa de juego

Figura 45: Carril central del mapa de juego

Figura 46: Zona de aparición del equipo atacante en el juego

Este enfoque ha permitido crear un entorno funcional, escalable y visualmente atractivo, listo para adaptarse tanto a modos de juego individuales como por equipos en futuras versiones del proyecto.

5
Pruebas

En esta sección se detallan las pruebas realizadas para verificar el correcto funcionamiento del videojuego desarrollado. El objetivo de estas pruebas es detectar errores, comprobar que todas las funcionalidades se comportan como se espera, y recoger impresiones sobre la experiencia de usuario. Para ello, se ha optado por una estrategia de pruebas en entorno local, permitiendo simular partidas multijugador y recopilar comentarios directos de distintos usuarios. Estas pruebas no solo han servido para validar técnicamente el sistema, sino también para valorar aspectos relacionados con la jugabilidad, la usabilidad de los menús y la claridad de la interfaz.

5.1. Pruebas realizadas dentro del inspector

Durante el proceso de desarrollo del proyecto, también se han llevado a cabo diversas pruebas personales orientadas a detectar errores de funcionamiento y mejorar la jugabilidad general. Estas pruebas se realizaron en entornos locales y permitieron identificar y corregir múltiples fallos que afectaban a la experiencia del usuario. A continuación, se detallan algunos de los errores encontrados y las soluciones implementadas:

- **Granadas atravesando superficies:** al lanzar una granada hacia el suelo o muy cerca de una pared, esta atravesaba la geometría del nivel, quedando fuera del espacio jugable. Este error se debía a que la colisión no estaba correctamente habilitada al momento del lanzamiento. Se solucionó activando primero la colisión del proyectil antes de aplicar la fuerza de lanzamiento, asegurando así que interactúe correctamente con el entorno desde el inicio.

- **Cambio de granada incorrecto:** si el jugador tenía una granada en la mano, el sistema no permitía cambiar correctamente a otro tipo de granada. Este comportamiento se debía

a un error en el flujo de control del código que gestionaba la selección de granadas. Fue corregido ajustando la lógica condicional del *blueprint*, permitiendo cambiar de tipo de granada sin importar si ya se tenía una equipada.

- **Error visual en la cámara al agacharse:** se detectó un fallo en el que, al alternar entre el estado agachado y de pie, la cámara del jugador se adelantaba levemente, permitiendo ver a través de las paredes. Esto afectaba negativamente al realismo del juego. La solución consistió en ajustar manualmente la posición de la cámara cada vez que se producía un cambio entre estar de pie y agachado, evitando así desplazamientos no deseados del punto de vista del jugador.

- **Transiciones incorrectas entre animaciones de caminar y correr:** se produjo un fallo en la transición entre estas dos animaciones, lo que causaba que el personaje no respondiera adecuadamente al cambiar de velocidad. Para corregirlo, se introdujo una variable booleana que indica si el jugador está corriendo o caminando, lo cual permite gestionar correctamente las condiciones de transición en el *blueprint* de animación.

- **Mal posicionamiento del arma en las manos del personaje:** inicialmente el arma no se mostraba alineada correctamente con la mano del personaje, lo que generaba una apariencia poco realista. Este problema se solucionó añadiendo un *socket* en la mano del esqueleto del personaje y ajustando la rotación del *mesh* del arma para que encajara correctamente, logrando así una integración visual más precisa.

- **Animación de salto incorrecta:** al realizar un salto, la animación del personaje no se interrumpía al aterrizar, lo que provocaba que este permaneciera en el aire visualmente durante unos segundos. Este error afectaba a la coherencia visual del movimiento. Se resolvió ajustando las condiciones de transición en el *blueprint* de animación para que se detectara correctamente el contacto con el suelo mediante una variable booleana que indica si el personaje está en el aire o no, forzando así la transición a la animación correspondiente en tierra.

5.2. Pruebas con otros usuarios

Las pruebas han sido realizadas dejando jugar a distintos amigos y familiares con distintas experiencias para que puedan aportar sus opiniones sobre el juego.

A continuación, muestro los comentarios más recogidos en los que he podido corregir ciertos fallos y ajustar aspectos que facilitan la experiencia de juego:

- Menús intuitivos para crear una partida y el menú de controles debería ser ampliado para poder ajustar los controles (futura mejora).
- Dificultad para unirse a una partida, sobre todo, los usuarios con menor conocimiento al no conocer como conectarse a un dirección IP o como obtener la del otro usuario. Como mejora se propone construir un servidor de *matchmaking* automático.
- Menú de compra y *HUD* intuitivo.
- Mapa grande adaptado para muchos modos de juego y variado visualmente.
- Buena variedad de armas y granadas útiles.

Con estos comentarios se realizaron los ajustes pertinentes y se ha apuntado los comentarios de mejora para futuras versiones del videojuego.

6

Conclusiones y Líneas Futuras

En esta última sección se presentan las **conclusiones generales** obtenidas tras el desarrollo del proyecto, valorando los objetivos alcanzados, los retos técnicos superados y las decisiones clave que han determinado el resultado final. Asimismo, se ofrece una reflexión crítica sobre el proceso de diseño e implementación, evaluando tanto los aspectos positivos como aquellos susceptibles de mejora.

Por otro lado, se exponen una serie de **líneas futuras de trabajo** que podrían incorporarse en desarrollos posteriores. Estas propuestas tienen como finalidad ampliar, optimizar o enriquecer las funcionalidades existentes, ya sea desde el punto de vista técnico, visual o jugable, y sientan las bases para una posible evolución del proyecto más allá de su versión actual.

6.1. Conclusiones

A pesar de los muchos impedimentos y cambios de rumbo durante el transcurso del proyecto, finalmente se ha podido realizar un proyecto jugable, con los requisitos que se propusieron en el anteproyecto de este Trabajo de Fin de Grado.

Podemos destacar que se ha implementado un juego multijugador en primera persona del género FPS, con sus bases para ser fiel a lo que se propuso, aunque luego hayan existido cambios como el modo de juego, entre otros. Al final, se pensó para dos jugadores debido a la facilidad de implementación y, además, al ser únicamente dos jugadores permite crear partidas entre amigos y retarse entre ellos.

Cabe destacar, el gran esfuerzo realizado tanto a nivel de diseño como de desarrollo del nivel, interfaces, mecanismos de interacción o lógica de videojuego. Todo esto influye en gran manera en el diseño y desarrollo del proyecto software

Por otra parte, el uso de las metodologías aprendidas durante estos cuatro años de carrera, más concretamente *Scrum*, que es la que más se adaptaba a las necesidades del proyecto.

6.2. Líneas Futuras

Aunque el proyecto ha alcanzado un resultado funcional y jugable, se identifican diversas áreas de mejora y expansión que podrían abordarse en futuras versiones. Estas líneas futuras permitirían elevar la calidad del juego, enriquecer la experiencia del usuario y acercarse aún más al diseño original planteado. A continuación, se detallan las principales propuestas de mejora:

- **Desarrollar un sistema de servidor y matchmaking:** permitir a los jugadores conectarse a partidas públicas mediante búsqueda automatizada.
- **Implementar el modo de juego original:** desarrollar el modo de juego inicialmente definido, con rondas estructuradas, cambio de equipo y lógica competitiva.
- **Extender el modo actual a partidas por equipos:** adaptar el sistema multijugador actual para permitir combates entre dos equipos, manteniendo también la opción de partida individual.
- **Incluir nuevas armas:** ampliar el repertorio de armas disponibles para ofrecer más variedad y estilos de juego, incluyendo nuevos modelos y estadísticas.
- **Añadir dispersión al disparo:** implementar un sistema de error por dispersión para hacer los disparos menos predecibles y más realistas, especialmente en armas automáticas.
- **Incorporar nuevas granadas:** añadir, entre otras, una granada aturdidora (*stun*) con efectos visuales y de gameplay, ampliando las opciones tácticas del jugador.
- **Ampliar los ajustes de usuario:** ofrecer más opciones de configuración en el menú, incluyendo parámetros de audio, vídeo, controles y accesibilidad.
- **Aumentar el detalle del nivel actual:** enriquecer el entorno del mapa principal con más elementos decorativos, iluminación y ambientación.

- **Desarrollar nuevos mapas jugables:** diseñar y añadir más escenarios para mejorar la rejugabilidad y adaptarse a distintos estilos de partida.
- **Mejorar efectos de sonido:** refinar los sonidos actuales e incorporar nuevos efectos para armas, pasos, explosiones y ambiente.
- **Mejorar los efectos visuales:** aplicar mejoras con partículas, efectos de impacto, post-procesado y animaciones, especialmente usando Niagara.
- **Optimizar los menús:** rediseñar los menús con una interfaz más profesional y opciones más intuitivas para el usuario.
- **Implementación de inteligencia artificial en caso de desconexión:** consiste en desarrollar un sistema de inteligencia artificial que permita simular el comportamiento de un jugador real en situaciones de desconexión. Esta IA estaría diseñada para tomar decisiones en función del estado actual de la partida, como moverse, atacar, defender o utilizar habilidades, garantizando así la continuidad y el equilibrio del juego incluso ante imprevistos de red. Para ello, se integraría un mecanismo de toma de decisiones basado en el entorno y el comportamiento esperado del jugador reemplazado.

6.3. Aprendizaje personal

El desarrollo de este Trabajo de Fin de Grado ha supuesto una experiencia enriquecedora a nivel personal y académico, permitiéndome adquirir nuevos conocimientos y consolidar habilidades en diversas áreas relacionadas con el desarrollo de videojuegos.

En primer lugar, este proyecto me ha brindado la oportunidad de profundizar en el uso de **Unreal Engine 5**, una herramienta profesional ampliamente utilizada en la industria. A lo largo del proceso he aprendido a manejar su sistema de *Blueprints*, los *Widget Blueprints* para la interfaz de usuario, el sistema de físicas, la creación de lógica de juego mediante eventos y el sistema de partículas *Niagara*, entre otros aspectos clave del motor.

Además, este trabajo me ha permitido adentrarme en el mundo del desarrollo de videojuegos desde una perspectiva completa, abarcando desde el diseño inicial de mecánicas y mapas hasta la implementación de funcionalidades técnicas, pasando por aspectos visuales, sonoros y de jugabilidad.

Otro punto relevante ha sido el refuerzo de los conocimientos adquiridos sobre **metodologías ágiles**, aplicando principios de planificación iterativa y organización por sprints. Esta forma de trabajo me ha ayudado a dividir el proyecto en etapas asumibles, priorizar tareas y adaptarme mejor a los tiempos disponibles.

Asimismo, el desarrollo del TFG ha implicado afrontar cambios imprevistos respecto al planteamiento inicial del juego, lo que me ha permitido trabajar la **capacidad de adaptación** y entender que todo cambio representa también una oportunidad de mejora y aprendizaje.

Por último, uno de los logros técnicos más destacables ha sido la **implementación de un sistema multijugador** funcional, utilizando la arquitectura cliente-servidor que ofrece Unreal Engine. Gracias a ello, he comprendido los fundamentos del trabajo en red aplicado a videojuegos, la gestión de sesiones, el control de conexiones y la sincronización entre jugadores, aspectos fundamentales para el desarrollo de experiencias multijugador.

En conjunto, este proyecto ha representado un reto técnico y creativo que ha contribuido de forma significativa a mi formación como desarrollador.

Referencias

[1] Activision. *Sitio oficial de Activision*. Último acceso: mayo de 2025. 2025. URL: https://www.activision.com/es.

[2] Asana. *¿Qué es un diagrama de flujo?* Último acceso: mayo de 2025. 2024. URL: https://asana.com/es/resources/what-is-a-flowchart.

[3] Creative Commons. *Attribution 4.0 International (CC BY 4.0)*. Último acceso: mayo de 2025. 2013. URL: https://creativecommons.org/licenses/by/4.0/.

[4] Epic Games. *ArrowComponent | Unreal Engine Python API*. Último acceso: mayo de 2025. 2024. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/python-api/class/ArrowComponent?application_version=5.0.

[5] Epic Games. *Blueprints: Visual Scripting in Unreal Engine*. Último acceso: mayo de 2025. 2024. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/blueprints-visual-scripting-in-unreal-engine.

[6] Epic Games. *Capsule Component | Unreal Engine Documentation*. Último acceso: mayo de 2025. 2024. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/BlueprintAPI/Components/Capsule.

[7] Epic Games. *Creating Visual Effects in Niagara for Unreal Engine*. Último acceso: mayo de 2025. 2024. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/creating-visual-effects-in-niagara-for-unreal-engine.

[8] Epic Games. *Custom Events in Unreal Engine*. Último acceso: mayo de 2025. 2024. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/custom-events-in-unreal-engine.

[9] Epic Games. *Enhanced Input in Unreal Engine*. Último acceso: mayo de 2025. 2024. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/enhanced-input-in-unreal-engine.

[10] Epic Games. *Game Mode and Game State in Unreal Engine*. Último acceso: mayo de 2025. 2024. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/game-mode-and-game-state-in-unreal-engine.

[11] Epic Games. *Movement Components in Unreal Engine*. Último acceso: mayo de 2025. 2024. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/movement-components-in-unreal-engine.

[12] Epic Games. *Player Controllers in Unreal Engine*. Último acceso: mayo de 2025. 2024. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/player-controllers-in-unreal-engine.

[13] Epic Games. *RandomUnitVectorInConeInDegrees | Unreal Engine Documentation*. Último acceso: mayo de 2025. 2024. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/BlueprintAPI/Math/Random/RandomUnitVectorinConeinDegrees.

[14] Epic Games. *Sound Cue Reference for Unreal Engine*. Último acceso: mayo de 2025. 2024. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/sound-cue-reference-for-unreal-engine.

[15] Epic Games. *Static Meshes | Unreal Engine Documentation*. Último acceso: mayo de 2025. 2024. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/static-meshes.

[16] Epic Games. *Unreal Engine 5*. Último acceso: mayo de 2025. 2024. URL: https://www.unrealengine.com/en-US/unreal-engine-5.

[17] Epic Games. *Using a Single Line Trace (Raycast) by Channel in Unreal Engine*. Último acceso: mayo de 2025. 2024. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/using-a-single-line-trace-raycast-by-channel-in-unreal-engine.

[18] Epic Games. *Widget Blueprints in UMG for Unreal Engine*. Último acceso: mayo de 2025. 2024. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/widget-blueprints-in-umg-for-unreal-engine.

[19] Fab. *Fab - Asset Marketplace for Unreal Engine and More*. Último acceso: mayo de 2025. 2024. URL: https://www.fab.com/.

[20] Fab - Unreal Engine Marketplace. *Flash Grenade Asset - Fab Listing*. Último acceso: mayo de 2025. 2024. URL: https://www.fab.com/listings/871fb7d6-b89f-4d62-96dc-7584cec278bb.

[21] Fab - Unreal Engine Marketplace. *Grenade Asset - Fab Listing*. Último acceso: mayo de 2025. 2024. URL: https://www.fab.com/listings/0214ea5b-2b6d-4726-83de-5c6cee49935d.

[22] Fab - Unreal Engine Marketplace. *Military Soldier Character - Fab Listing*. Último acceso: mayo de 2025. 2024. URL: https://www.fab.com/listings/8e200050-3158-4762-b297-f785b5b1533d.

[23] Fab - Unreal Engine Marketplace. *Recortada (Shotgun) Asset - Fab Listing*. Último acceso: mayo de 2025. 2024. URL: https://www.fab.com/listings/406e003f-1a3b-4c9f-9588-6f16baf079d5.

[24] Fab - Unreal Engine Marketplace. *Shotgun Asset - Fab Listing*. Último acceso: mayo de 2025. 2024. URL: https://www.fab.com/listings/cb46c575-7c6a-460e-82ef-8ef532df50c3.

[25] Fab - Unreal Engine Marketplace. *Sniper Rifle Asset - Fab Listing*. Último acceso: mayo de 2025. 2024. URL: https://www.fab.com/listings/aea49220-2699-4bca-9d42-d7c845d61c68.

[26] Fab - Unreal Engine Marketplace. *Weapon Asset - Fab Listing*. Último acceso: mayo de 2025. 2024. URL: https://www.fab.com/listings/8aeb9c48-b404-4dcd-9e56-1d0ecedba7f5.

[27] Freesound Contributors. *Freesound - Free sound effects and samples*. Último acceso: mayo de 2025. 2025. URL: https://freesound.org/.

[28] GaTu. *Cómo crear un sistema de multijugador con lobby - Unreal Engine 5.1*. YouTube. Último acceso: mayo de 2025. 2023. URL: https://youtu.be/QFq6knjA9C4.

[29] GitHub. *GitHub - Collaborative Development Platform*. Último acceso: mayo de 2025. 2024. URL: https://github.com/.

[30] Overleaf. *Overleaf - Online LaTeX Editor*. Último acceso: mayo de 2025. 2024. URL: https://www.overleaf.com/.

[31] Quixel. *Quixel - Fab Creator Profile*. Último acceso: mayo de 2025. 2024. URL: https://www.fab.com/sellers/Quixel.

[32] Udemy. *Udemy - Online Courses and Learning*. Último acceso: mayo de 2025. 2024. URL: https://www.udemy.com/.

[33] Visual Paradigm International. *Visual Paradigm - Visual Modeling and Diagramming Tool*. Último acceso: mayo de 2025. 2024. URL: https://www.visual-paradigm.com/.

[34] Wikipedia contributors. *HUD (videojuegos) — Wikipedia, la enciclopedia libre*. Último acceso: mayo de 2025. 2025. URL: https://es.wikipedia.org/wiki/HUD_(videojuegos).

Apéndice A
Game Design Document

DOCUMENTO DE DISEÑO DE VIDEOJUEGO

GAME DESIGN DOCUMENT

Trabajo Fin de Grado de Francisco Matas Moreno

Tutorizado por Antonio José Fernández Leiva

Universidad de Málaga

Índice

1. Información general....3
1.1. Título del juego....3
1.2. Género....3
1.3. Plataforma de juego....3
1.4. Motor de desarrollo....3
1.5. Equipo de desarrollo....3
2. Resumen del juego....4
2.1. Descripción general....4
2.2. Público objetivo....4
3. Historia....4
3.1. Sinopsis....4
3.2. Ambientación y mundo....5
3.3. Personajes....5
3.3.1. Grupo Foxtrot (Equipo defensor)....5
3.3.2. Grupo Echo (Equipo atacante)....6
3.4. Progresión de la historia....6
4. Mecánicas de juego....6
4.1. Gameplay....6
4.2. Armas y utensilios....8
4.2.1. Armas....8
4.2.1.1. Armas principales....8
4.2.1.2. Armas secundarias....9
4.2.2. Utensilios....9
4.2. Controles....9
4.2.1. Controles con teclado y ratón....9
4.2.2. Controles con gamepad....10
4.3. Sistema de progresión....10
5. Arte y diseño visual....10
5.1. Estilo gráfico....10
5.2. Diseño de personajes....11
5.3. Diseño de niveles....11
5.4. Diseño de armamento....11
5.4.1. Armas....11
5.4.2. Utilidades....11
5.5. Animaciones y efectos....12
6. Sonido y música....12
6.1. Música....12
6.2. Efectos de sonido....12
7. Interfaz y experiencia de usuario....13
7.1. Menús y HUD....13
7.1.1. Menú principal....13

7.1.2. Menú de ajustes..... 15
7.1.3. Menú de fase de compra de armamento..... 16
7.1.4. HUD..... 16
8. Monetización..... 17
8.1. Precio de venta al público..... 17
8.2. Microtransacciones..... 18
9. Distribución..... 18
9.1. Plataformas de lanzamiento..... 18
9.2. Actualizaciones futuras..... 18
10. Anexos..... 18

1. Información general

En esta sección, se realizará una introducción general al documento en sí y, a todo lo relacionado con detalles fundamentales del juego como su título o género.

1.1. Título del juego

El título del juego es “Silent Chaos″. Este nombre hace referencia al caos que se da en el silencio de la guerra. Dos equipos intentando sorprenderse entre sí, haciendo movimientos silenciosos que en cualquier momento puede estallar en un absoluto caos de disparos y explosiones.

1.2. Género

El juego estaría encajado en el género de disparos en primera persona, más conocido en inglés como First-Person Shooter o, simplemente, FPS (de sus siglas en inglés). Además, tiene un componente táctico por lo que podemos incluirlo en el subgénero de disparos táctico o Tactical Shooter, en inglés.

1.3. Plataforma de juego

El juego está desarrollado para ser utilizado en ordenador con teclado y ratón o mando de consola conectado por cable.

1.4. Motor de desarrollo

Se utiliza para el desarrollo el motor Unreal Engine de Epic Games. La versión utilizada es la versión 5.4.4 ya que es actualmente una de las últimas versiones, aunque no la última, lo que conlleva dos grandes ventajas: la primera, es una versión estable y, la segunda, es una versión con la mayoría de nuevas características introducidas.

1.5. Equipo de desarrollo

El equipo de desarrollo está compuesto por Francisco Matas Moreno, único desarrollador del proyecto. El tutor, por su parte, cumple la función de cliente que irá guiando durante todo el proceso en los sprints de desarrollo.

2. Resumen del juego

En esta sección, se realiza una introducción más detallada al juego en sí, sin entrar en detalles de historia o de personajes que son detallados en la sección número tres.

2.1. Descripción general

"Silent Chaos" es un juego de disparos en primera persona (FPS) con un fuerte enfoque táctico y cooperativo. Enfrenta a dos equipos con objetivos opuestos en intensos combates estratégicos dentro de un mundo postapocalíptico.

El equipo defensor, Foxtrot, tiene la misión de proteger su base y continuar las investigaciones sobre un misterioso néctar con propiedades potenciadoras. Por otro lado, el equipo atacante, Echo, compuesto por soldados altamente entrenados, busca infiltrarse y sabotear las investigaciones para evitar que Foxtrot adquiera demasiado poder.

Las partidas se desarrollan en un entorno urbano devastado por una guerra nuclear, con zonas reconstruidas estratégicamente utilizadas por ambos equipos. La comunicación y la cooperación entre jugadores serán clave para la victoria, ya que cada enfrentamiento puede pasar de un tenso sigilo a un explosivo combate en cuestión de segundos.

2.2. Público objetivo

El juego está dirigido a todas aquellas personas interesadas en los "shooters" y en experiencias cooperativas. Debido a la naturaleza del juego, que incluye elementos de disparos y contenido potencialmente sensible, se recomienda para un público adolescente en adelante.

3. Historia

En esta sección, se muestra la historia que existe detrás de este juego. Además, se comentará la ambientación y la historia de cada personaje jugable del juego.

3.1. Sinopsis

Tras una gran guerra nuclear, el mundo quedó devastado por la fuerza de los constantes bombardeos. Diez años después, cuando algunas zonas volvieron a ser habitables, diversos grupos radicales empezaron a tomar

edificios en ruinas y a reconstruir algunos puntos para poder sobrevivir en la nueva realidad.

Meses más tarde de que varios grupos se asentasen de nuevo en la superficie, la sección de exploración del grupo “Foxtrot” descubrió un nuevo material líquido, similar al agua, que potenciaba las habilidades de los humanos que la tomaban.

Considerado como un néctar divino, los investigadores de “Foxtrot” aún consideraban que dicha bebida podía mejorarse haciendo que se multiplicasen sus efectos positivos.

Debido al reciente descubrimiento, y al ser el único poseedor, el resto intenta evitar que las investigaciones continuen y evitar así un control absoluto. El grupo más destacado es el “Echo”, quienes intentan obstaculizar las investigaciones, por contar con los soldados mejor preparados.

3.2. Ambientación y mundo

El escenario principal del juego está ambientado en una ciudad en ruinas, con algunas zonas reconstruidas y otras aún sumergidas en la fauna y flora que sobrevivió a la guerra nuclear.

El escenario contará con tres zonas principales por donde moverte junto con tus compañeros, además de la zona base de cada grupo, donde será el objetivo de cada uno de ellos.

3.3. Personajes

Los grupos o secciones que son jugables en los roles de atacante y defensor son:

3.3.1. Grupo Foxtrot (Equipo defensor)

Este equipo es el portador del néctar y que defiende su base. Es considerado un grupo paramilitar con roles definidos que ha encontrado una gran oportunidad para controlar al resto de la población.

Para los caracteres de este grupo se usa el paquete de assets: link al asset

3.3.2. Grupo Echo (Equipo atacante)

Este grupo está formado por un conjunto de personas que eran antiguos militares o civiles que tomaron alguna formación militar. Debido a esto cuentan con gran habilidad y están totalmente preparados para evitar las investigaciones del grupo "Foxtrot".

Para los caracteres de este grupo se usa el paquete de assets: link al asset

3.4. Progresión de la historia

4. Mecánicas de juego

En esta sección se explica el modo de juego y las posibilidades que tiene el jugador dentro de la partida. Se muestra el formato, las armas y utensilios obtenibles por el jugador y todas las mecánicas posibles de movimiento.

4.1. Gameplay

El juego consiste en un multijugador de disparos táctico en el que dos equipos intentan conseguir derrotarse entre sí. Para vencer, ambos se enfrentan en un formato por rondas. Cada ronda, uno de los equipos defenderá un "punto caliente" o "zona de plante de la bomba" y el otro intentará llevar un artefacto explosivo a dicho punto y esperar un cierto tiempo hasta que este explote.

Para vencer, es necesario ganar un total de nueve rondas. Con el objetivo de evitar favorecer a un equipo concreto, se jugarán ocho rondas de forma consecutiva con el rol que comience la partida. Tras estas ocho rondas, los roles son invertidos.

En cada ronda, existen varias fases:

- **Fase de compra de armamento:** cada ronda comienza por esta fase, donde los jugadores tienen un tiempo para elegir el armamento con el que disputar la nueva ronda. Si el jugador ha sido eliminado solo tendrá una pistola, sin embargo, si ha sobrevivido, mantiene el arma que adquirió la última ronda. El jugador podrá adquirir cierto armamento en función del coste y de la reserva económica que haya hecho hasta el momento.
- **Fase de combate:** comienza automáticamente tras el fin de la fase de compra de armamento. En esta fase, hay un tiempo límite para

el equipo atacante, es decir, el que porta el explosivo hasta la zona de plante, de un total de noventa segundos. Si el equipo atacante consigue llevar el artefacto y desplegarlo a tiempo, tendrán que esperar hasta que este detone. Por su parte, el equipo defensor debe evitar que los atacantes desplieguen la bomba en los primeros noventa segundos.

- **Fase de post plante de la bomba:** Esta fase comienza únicamente si los atacantes consiguen desplegar la bomba a tiempo en la zona de plante en el tiempo de la fase de combate. En caso de ser así, comienza una cuenta atrás donde los atacantes tienen que defender la bomba y evitar que los defensores la desactiven. Por su parte los defensores deben evitar que la bomba explote y, por tanto, desactivarla.

Tras estas rondas, cada jugador recibe una suma de dinero para invertir en la siguiente ronda.

Además, hay que destacar varias rondas más relevantes de la partida:

- Ronda nº 1: Es la primera ronda de la partida, en la que los jugadores comienzan con el mismo equipamiento y el mismo dinero para armamento.
- Ronda nº 8: Es la última ronda antes de invertir los roles, en la que todos los jugadores pierden todo su armamento y dinero, independientemente del resultado.
- Ronda nº 9: Ídem a la primera ronda, con roles invertidos.

Por otra parte, si existe un empate, comienza las rondas de prórroga que funciona de la siguiente forma:

- Ronda nº 1 de la prórroga: Esta ronda solo ocurre si, y solo si, los equipos han empatado en las primeras dieciséis rondas, es decir, si en el cómputo global de rondas ambos equipos tienen ocho rondas ganadas. En esta ronda, los equipos dispondrán de la misma cantidad de dinero y los roles son invertidos.
- Ronda nº 2 de la prórroga: Ídem a la primera ronda de la prórroga, con roles invertidos.

Si alguno de los equipos consigue ganar ambas rondas de forma consecutiva gana la partida. En caso contrario, vuelve a empezar otra prórroga.

4.2. Armas y utensilios

Para poder eliminar a los enemigos, cada jugador puede obtener un armamento que se clasifica en: armas, principales y secundarias, y utensilios.

4.2.1. Armas

Cada jugador puede portar una de ellas únicamente. Además, el jugador siempre porta un arma de estilo cuerpo a cuerpo.

4.2.1.1. Armas principales

Existen cuatro tipos de armas principales: fusil de asalto, subfusil, escopeta y rifle de francotirador.

Para facilitar la lectura se muestra una tabla con las estadísticas de cada arma:

Nombre	Tipo	Cadencia de disparo	Tamaño del cargador	Coste	Tiempo de recarga	Daño (cabeza/ cuerpo/ piernas)
M16	Fusil de asalto	Alta	30	(por definir)	(por definir)	
AK47	Fusil de asalto	Media	25	(por definir)	(por definir)	
Micro Uzi	Subfusil	Muy alta	20	(por definir)	(por definir)	
AK74	Subfusil	Alta	30	(por definir)	(por definir)	
Recortada	Escopeta	Baja	2	(por definir)	(por definir)	
Escopeta de postas	Escopeta	Baja	5	(por definir)	(por definir)	
Barret	Rifle de francotirador	Baja	5	(por definir)	(por definir)	
Glock	Pistola	Media	10	(por definir)	(por definir)	

4.2.1.2. Armas secundarias

Nombre	Tipo	Cadencia de disparo	Tamaño del cargador	Coste	Tiempo de recarga	Daño (cabeza/ cuerpo/ piernas)
Cuchillo táctico	Cuerpo a cuerpo	Baja	0	gratis	(por definir)	

4.2.2. Utensilios

Cada jugador podrá utilizar sus créditos para comprar herramientas como:

- Granada de fragmentación
- Granada de humo
- Granada cegadora

4.2. Controles

4.2.1. Controles con teclado y ratón

Con el teclado se realizan las siguientes acciones:

- A: Moverse a la izquierda
- D: Moverse a la derecha
- W: Moverse hacia delante
- S: Moverse hacia atrás
- Shift: Agacharse/Levantarse
- Ctrl izquierdo: Andar
- R: Recargar el arma
- Q: Granada de mano
- E: Interactuar
- Tabulador: Mostrar/Ocultar estadísticas de la partida
- C: Arma principal
- X: Arma secundaria
- F: Arma cuerpo a cuerpo
- M: Mostrar/Ocultar mapa
- Espacio: saltar

Con el ratón se realizan las siguientes acciones:

- Clic principal: Disparar arma
- Clic secundario: Activar/desactivar mira

4.2.2. Controles con gamepad

Los controles con mando conectado al ordenador (ps4, ps5, xBox) son:

- Botones de dirección:
  - Izquierdo: Mostrar/Ocultar mapa
  - Derecho: Mostrar/Ocultar estadísticas de la partida
  - Superior: Mostrar/Ocultar mapa
  - Inferior:
- Botones de acción:
  - Izquierdo: Recargar el arma
  - Derecho: Agacharse/Levantarse
  - Superior: Interactuar
  - Inferior:saltar
- Joystick:
  - Izquierdo: Mover personaje
  - Derecho: Mover cámara
  - Pulsar izquierdo: Equipar granada de mano
  - Pulsar derecho: Correr/Andar
- Botones traseros:
  - Izquierdo 1: Arma anterior
  - Izquierdo 2: Activar/desactivar mira
  - Derecho 1: Arma siguiente
  - Derecho 2: Disparar

4.3. Sistema de progresión

El juego tiene un sistema de experiencia por jugador que representa una media entre las partidas jugadas y su desempeño en cada una de ellas. Se mide con niveles que escalan de forma exponencial, es decir, a mayor nivel, mayor cantidad de experiencia necesaria para subir de nivel.

5. Arte y diseño visual

En esta sección, se muestra como se realiza el diseño del nivel, así como el estilo de los personajes, armas y cualquier otro aspecto relacionado con la experiencia visual del jugador.

5.1. Estilo gráfico

El estilo utilizado es de carácter realista usando assets gratuitos de Fab, el mercado de Unreal Engine. En este se pueden encontrar numerosos artículos gratuitos que son utilizados en el nivel. La principal biblioteca

de recursos utilizada es Quixel que cuenta con diversos assets para múltiples entornos.

Con estos recursos, se realiza un nivel realista que se asimile a la realidad mostrada por la historia del juego.

5.2. Diseño de personajes

Para el diseño de los personajes, como se comenta en la sección 3.3, para los personajes se utilizan dos estilos diferentes para cada equipo. El equipo atacante o "Echo" utiliza un asset (link al asset) que asimila a un equipo de contrabando que realiza emboscadas a otras personas. En cuanto al equipo defensor o "Foxtrot", se utiliza un asset (link al asset) con un carácter más militar con el objetivo de representar un equipo que defiende su base o territorio de ataques enemigos.

5.3. Diseño de niveles

El nivel principal cuenta con varias zonas diferenciadas:

- Zona de aparición: es la parte del mapa donde cada equipo aparece al principio de cada ronda. Existe, por tanto, la zona de aparición "Foxtrot" y "Echo".
- Zona de plante de la bomba: son dos y está destacada en el mapa. Es el objetivo de los atacantes. Se denominan con los nombres en clave "A" y "B".
- Zonas de tránsito: son los caminos que hay entre las zonas de aparición y las zonas de plante. Son las zonas más grandes del nivel donde se libran la mayoría de combates.

5.4. Diseño de armamento

5.4.1. Armas

Para el diseño de las armas de fuego y el cuchillo también se utilizan varios assets (link 1, link 2, link 3) gratuitos del mercado de Unreal Engine. Tienen un estilo realista.

5.4.2. Utilidades

Para las granadas de mano se utilizan otros assets gratuitos (link 1, link 2, link 3).

Estas utilidades realizarán efectos en el nivel que modifican el transcurso de la partida, debido a los efectos realizados que son:

pantallas de humo, efecto cegador (la pantalla del jugador que mire a la granada se pondrá de un color blanco brillante, simulando una ceguera temporal por una luz de alta intensidad) y granada de fragmentación (daña al jugador y hace una pequeña vibración a la cámara)

5.5. Animaciones y efectos

El personaje realiza las siguientes animaciones:

- Agacharse
- Levantarse
- Saltar
- Andar en todas las direcciones
- Correr en todas direcciones
- Apuntar con el arma
- Animación de descanso (se realiza cuando el personaje está parado)

En cuanto a las armas las animaciones son:

- Disparo
- Recarga
- Sin munición

6. Sonido y música

En esta sección se comenta que música se ha creado para los menús y ambientación. Además de los efectos de sonido elegidos para los pasos, disparos y explosiones, entre otros.

6.1. Música

Cuando el jugador se encuentre en el menú principal o de ajustes se reproducirá una melodía musical que introduzca al jugador en el ambiente de guerra.

Para realizar la música se utiliza un programa gratuito de edición de sonido.

6.2. Efectos de sonido

Los efectos de sonido empleados son gratuitos y extraidos de la página web Freesound.org

Los efectos de sonido seleccionados son:

- Disparos
  - Rifle francotirador: https://freesound.org/people/EvanBoyerman/sounds/393651/
  - M16: https://freesound.org/people/SuperPhat/sounds/416417/
  - AK47:https://freesound.org/people/Franki-01234/sounds/201669/
  - Micro Uzi: https://freesound.org/people/Franki-01234/sounds/201672/
  - AK74:https://freesound.org/people/morganpurkis/sounds/390663/
  - Recortada: https://freesound.org/people/TB0Y298/sounds/718175/
  - Escopeta de postas: https://freesound.org/people/Marregheriti/sounds/266105/
  - Glock: https://freesound.org/people/seanmorrissey96/sounds/509435/
  - Cuchillo táctico:https://freesound.org/people/1histori/sounds/412430/
  - Arma sin municion: https://freesound.org/people/JoseIgnacioTriay/sounds/515194/
  - Recarga:
- Granadas de mano:
  - Fragmentación: https://freesound.org/people/Artninja/sounds/750825/
  - Cegadora: https://freesound.org/people/TheMistrzuni/sounds/733587/
  - Humo:https://freesound.org/people/pfranzen/sounds/583262/?
- Otros:
  - Pasos: https://freesound.org/people/whothefuckisjojo/sounds/673305/

7. Interfaz y experiencia de usuario

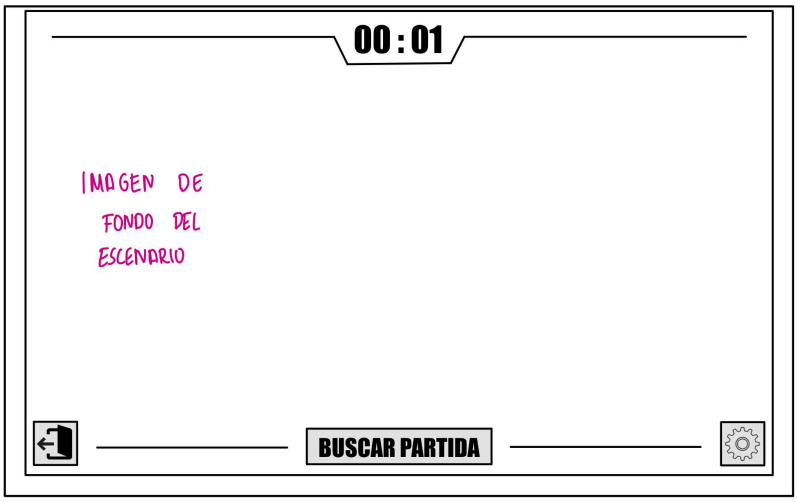
En esta sección, se muestran los menús que se ven en el juego. Observaremos, los que se diseñan y el resultado final, así como la navegación entre ellos.

7.1. Menús y HUD

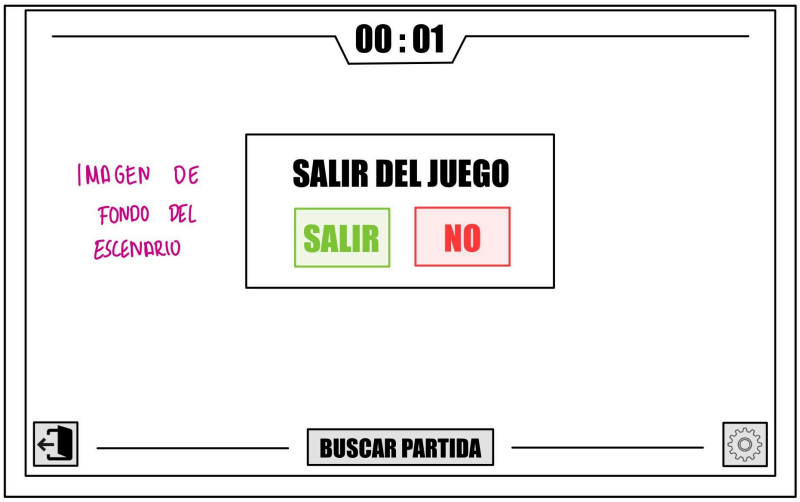
7.1.1. Menú principal

El menú principal es la primera vista que tiene el jugador cuando entra al juego. En el podemos observar:

- Un cronómetro, que se inicia cuando pulsamos el botón “Buscar Partida” que encontramos en la parte inferior. Tiene un carácter informativo para el usuario, es decir, muestra el estado del sistema y hace entender al usuario que realmente está buscando una nueva partida.
- El botón “Buscar Partida” que inicia el cronómetro y busca una nueva partida
- El botón de salir, ubicado en la esquina inferior izquierda, que está representado con la metáfora de una puerta y una flecha, un icono reconocible que expresa a la perfección el significado.
- El botón de ajustes, ubicado en la esquina inferior derecha, que está representado con la metáfora de una tuerca, un icono generalizado que se utiliza en cualquier ámbito digital para representar los ajustes o configuración.

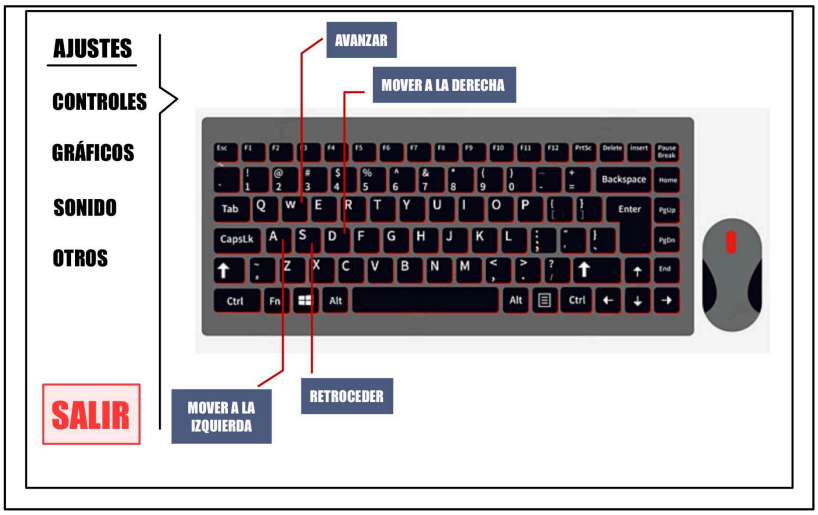


Al pulsar el botón de salir, se mostrará el mismo menú con una pregunta de confirmación para evitar que el juego sea cerrado sin intención por parte del jugador. Entonces, al pulsar en el botón de salir, se muestra otra pantalla superpuesta en el mismo menú principal en la que puedes confirmar salir o no.



7.1.2. Menú de ajustes

El menú de ajustes muestra todas las configuraciones posibles que el jugador puede modificar para adaptar su experiencia de juego. Además, contendrá una sección de controles donde el usuario podrá leer los controles de una manera gráfica para facilitar una rápida visualización.



En este menú, los ajustes están clasificados según su campo. Esto ayuda al usuario a encontrar el ajuste que quiera mucho más rápido y de una manera más eficaz.

Este menú es accesible en cualquier momento, es decir, en el menú principal, independientemente de que se esté buscando partida, o durante el transcurso de la partida.

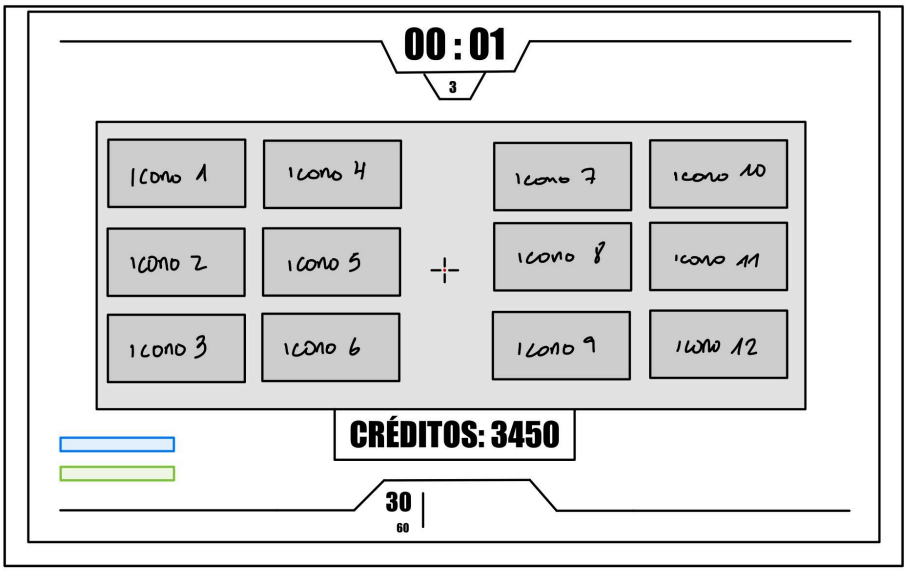
Al pulsar el botón de salir (identificado con el texto “SALIR”) vuelve al lugar donde estuviese anteriormente, es decir, si el

jugador se encontraba en el menú principal volverá allí y si se encontraba en una partida volverá a mostrar el HUD del jugador.

7.1.3. Menú de fase de compra de armamento

Este menú solo es accesible desde dentro de la propia partida. Se puede activar y desactivar durante la fase de compra.

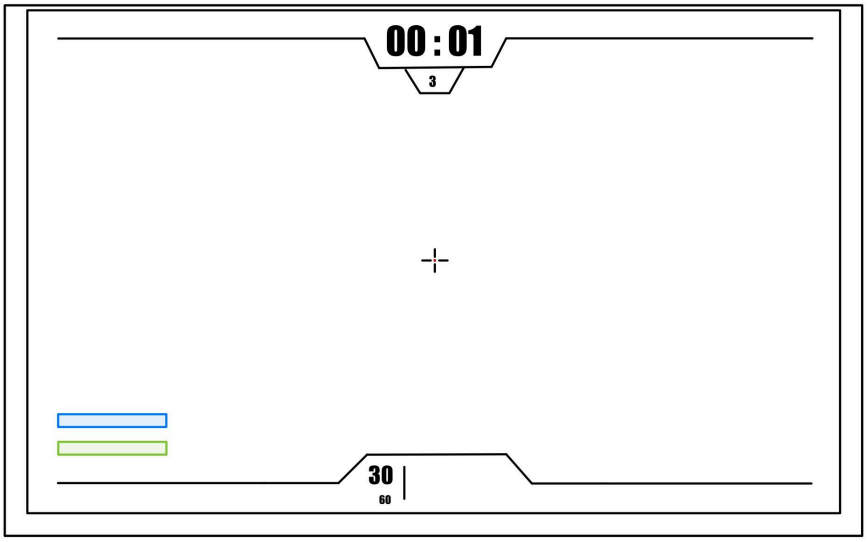
Este menú consiste en unos botones con los iconos de cada arma, que al clicar en uno de ellos, si el jugador tiene créditos suficientes para adquirirla, obtendrá el arma y se le restará el coste del arma al total de sus créditos.



https://freesound.org/people/BaggoNotes/sounds/721502/

7.1.4. HUD

El HUD (Head-Up Display) es la interfaz que muestra la información importante al jugador de un solo vistazo a la pantalla. Para facilitar esta característica se ha realizado una versión preliminar de esta. La idea es tener un HUD limpio y lo más visual posible. Por ello, el boceto inicial es:



Se puede observar un cronómetro que indica el tiempo de la ronda o de la fase de compra. Debajo de este, hay un número que indica la ronda en la que está la partida.

Por su parte, en la parte inferior, se encuentra el número de balas restantes del cargador y del total. También hay dos barras de vidas que indica la vida del jugador y el total de escudo que tiene.

En el centro, existe una mirilla que indica la dirección y sentido de las balas disparadas por el jugador si este está completamente parado, si no se aplicará un error en función de la velocidad del jugador.

8. Monetización

En esta sección se indica cómo se realiza la recaudación de fondos del juego. Se indica el precio de venta al público, así como, las microtransacciones que existen.

8.1. Precio de venta al público

El juego se distribuye de forma gratuita al público para aumentar el número de jugadores. Así mismo, se permite probar a todos los usuarios de la red la experiencia de juego y poder tener otra opción para jugar con sus amigos.

8.2. Microtransacciones

El juego no contará con microtransacciones para adquirir ningún tipo de contenido. De tal forma, no se descarta incluirlas en un futuro cuando el desarrollo principal haya concluido y se centre en incluir mejoras estéticas y visuales para los jugadores, siendo siempre opcionales para ellos y no teniendo un carácter del estilo “pagar para ganar” o “pay to win”, en inglés.

9. Distribución

En esta sección se comenta cómo se distribuye el archivo ejecutable para todo el público

9.1. Plataformas de lanzamiento

La principal plataforma de lanzamiento es el ordenador. Dentro de este se estudia donde se puede distribuir.

9.2. Actualizaciones futuras

Tras completar una versión jugable y con el mínimo de errores posibles, el desarrollo se centrará en implementar más escenarios y armas con el objetivo de que los jugadores tengan nuevas experiencias y no dejen de explorar nuevas formas de entretenimiento.

Apéndice B
Manual de Instalación

B.1. Pasos para la instalación

1. **Descargar el proyecto**

   Accede al siguiente enlace de Google Drive para descargar el archivo ejecutable del juego:

   https://drive.google.com/drive/folders/1srkVEvzAluJJcw2_avK4Bm6X7sgg7pv2?usp=sharing

   Nota: Si el enlace está protegido, asegúrate de tener permisos para visualizar o descargar el archivo.

2. **Extraer el contenido**

   Si el archivo descargado está comprimido (.zip), haz clic derecho sobre él y selecciona "Extraer todo...". Elige una carpeta de destino, por ejemplo: Escritorio o Documentos.

3. **Ejecutar el juego**

   Una vez extraído, localiza el archivo SilentChaos.exe dentro de la carpeta. Haz doble clic sobre él para iniciar el juego.

Consejos adicionales

- Ejecuta el juego como administrador si tienes problemas de permisos (*clic derecho >Ejecutar como administrador*).
- Si tu antivirus bloquea el archivo, añade una excepción o marca el ejecutable como seguro.

Apéndice C
Blueprint del personaje completo

Para complementar toda la información del *blueprint* de funcionalidad principal se añade en este anexo imágenes sobre algunas de las funciones implementadas.

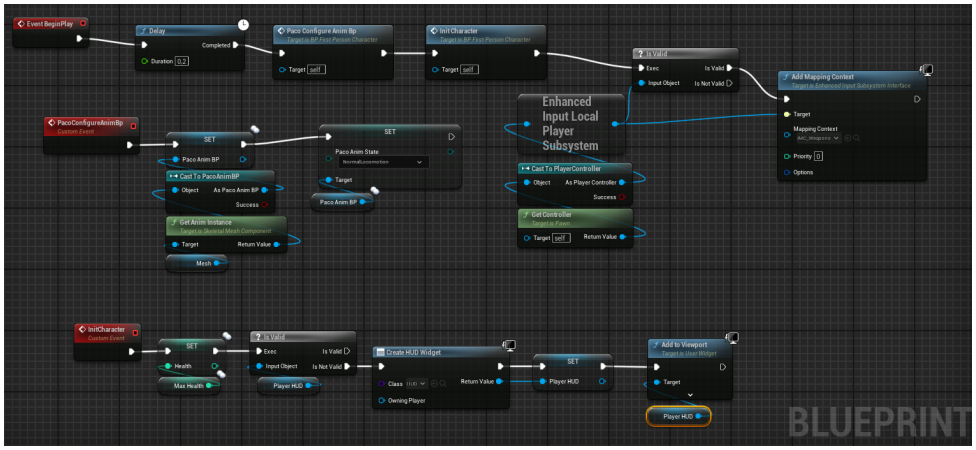


Figura 47: *Blueprint* de la función de creación del personaje

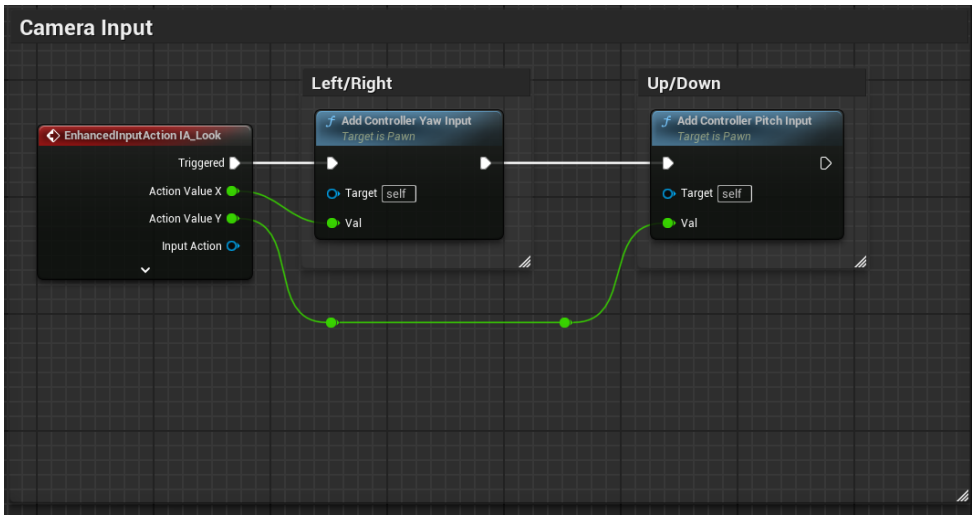


Figura 48: *Blueprint* de la función de movimiento de la cámara

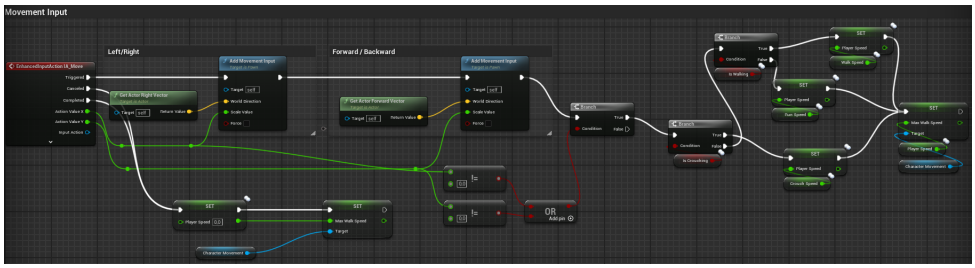


Figura 49: *Blueprint* de la función de movimiento del personaje

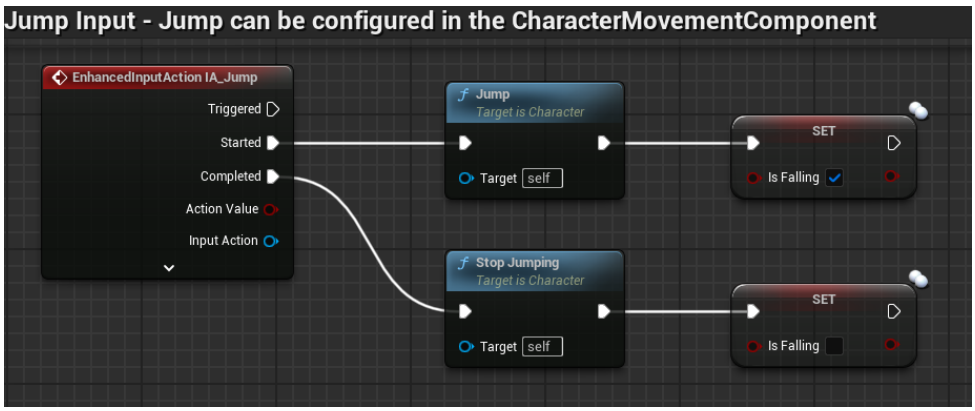


Figura 50: *Blueprint* de la función de salto del personaje

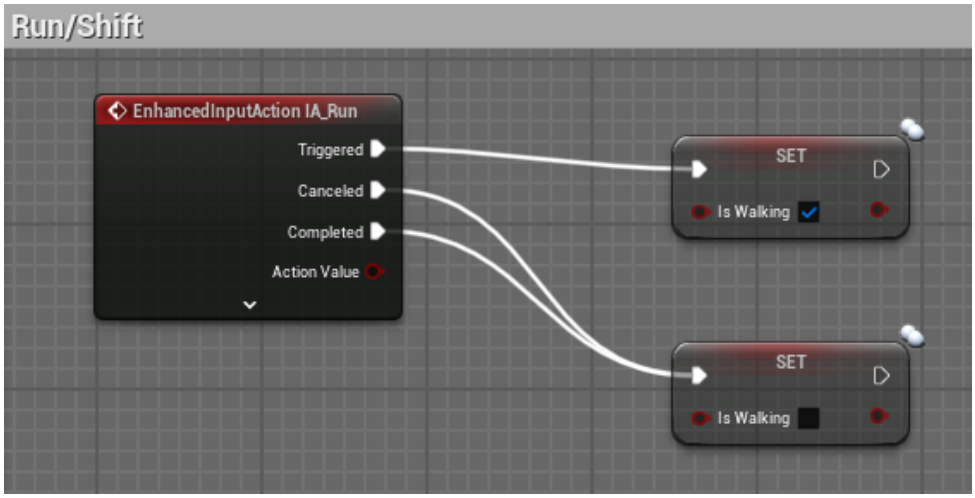


Figura 51: *Blueprint* de la función de correr

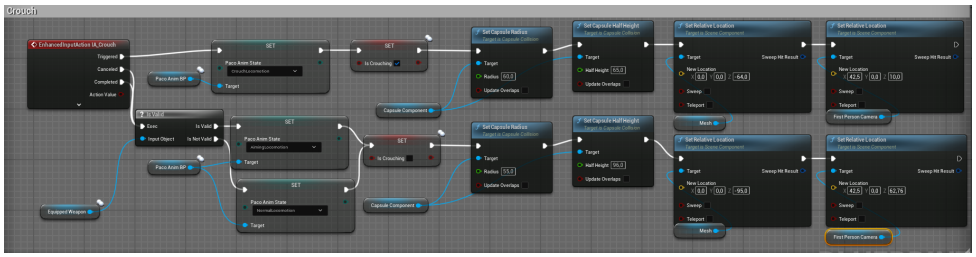


Figura 52: *Blueprint* de la función de agacharse

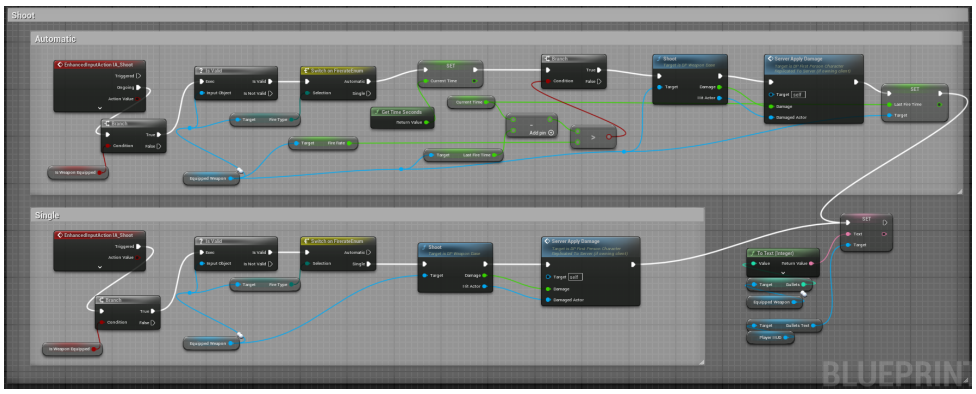


Figura 53: *Blueprint* de la función de disparar (automático y único)

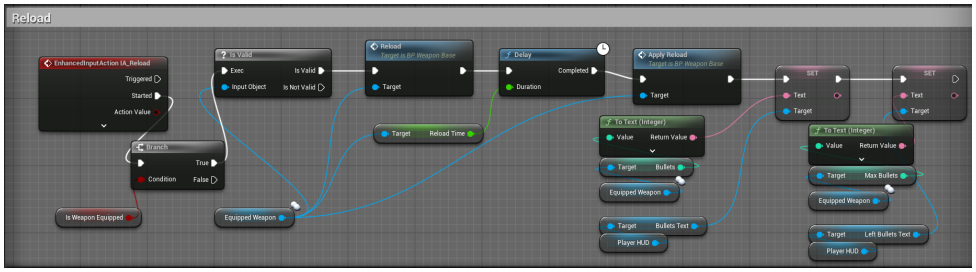


Figura 54: *Blueprint* de la función de recargar arma

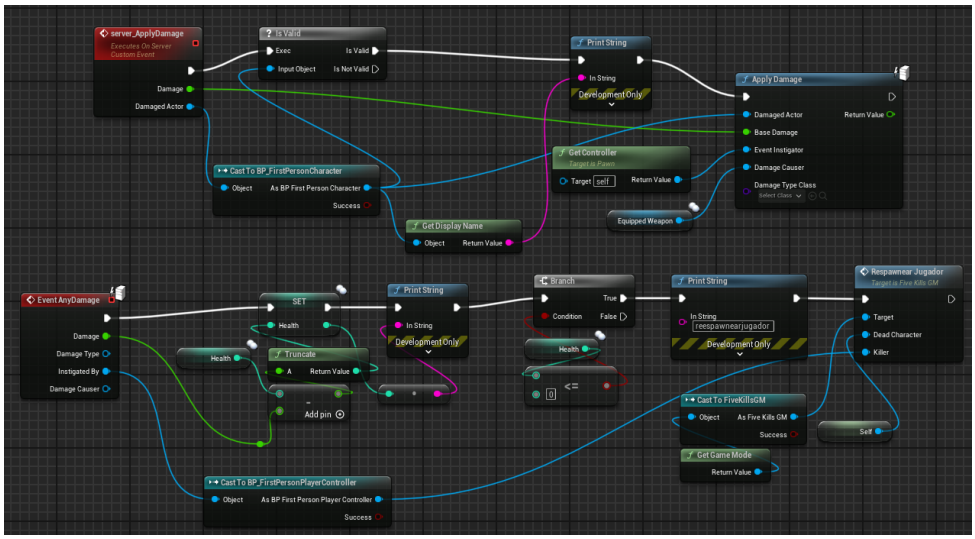


Figura 55: *Blueprint* de la función de recibir daño

Apéndice D
Blueprint de animación del personaje completo

Para complementar toda la información del *blueprint* de animación del personaje principal se añade en este anexo imágenes sobre las máquinas de estado implementadas.

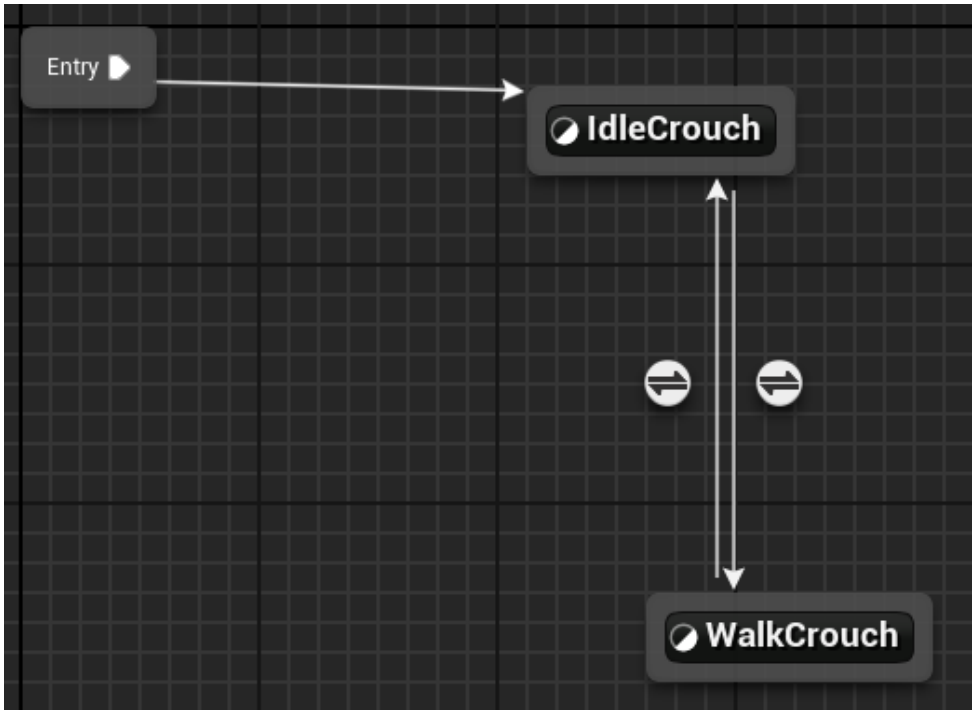


Figura 56: *Blueprint* de la máquina de estado para cuando el personaje esta agachado

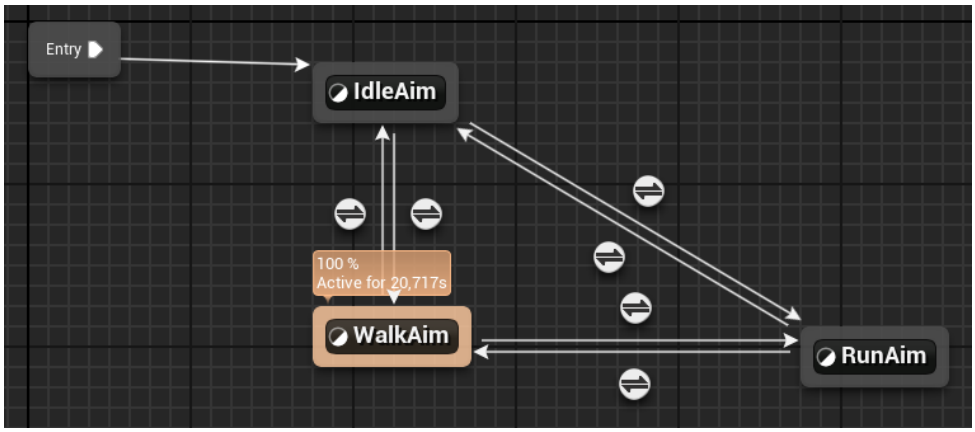


Figura 57: *Blueprint* de la máquina de estado utilizada cuando el personaje tiene un arma

UNIVERSIDAD DE MÁLAGA | uma.es

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga

E.T.S. DE INGENIERÍA INFORMÁTICA