TRABAJOS FIN DE GRADO

Universidad de Jaén

Escuela Politécnica Superior de Jaén

Remake del videojuego DOOM

Autor: David López Martínez

Grado: Ingeniería Informática

Director: Carlos Javier Ogayar Anguita
Departamento del director: Informática

Fecha: 11/09/2024

(Página intencionalmente en blanco)

Universidad de Jaén

Departamento de Informática

Don Carlos Javier Ogayar Anguita, tutor del Trabajo Fin de Grado titulado: '**Remake del videojuego DOOM**', que presenta Don David López Martínez, otorga el visto bueno para su entrega y defensa en la Escuela Politécnica Superior de Jaén.

Jaén, Septiembre de 2024

El alumno:

LOPEZ MARTINEZ DAVID - 77387513B

Firmado digitalmente por LOPEZ MARTINEZ DAVID - 77387513B
Fecha: 2024.09.15 15:17:19 +02'00'

David López Martínez

El tutor:

OGAYAR ANGUITA CARLOS JAVIER - 26039170L

Firmado digitalmente por OGAYAR ANGUITA CARLOS JAVIER - 26039170L
Fecha: 2024.09.13 12:45:16 +02'00'

Carlos Javier Ogayar Anguita

(Página intencionalmente en blanco)

Agradecimientos

A lo largo de estos años de estudio, he contado con el apoyo de mi familia y amigos, quienes siempre estuvieron presentes en los momentos más difíciles y en los logros alcanzados. Su compañía ha sido fundamental para llegar a este punto, y por ello les estaré eternamente agradecido.

A mi pareja, por ser un pilar constante, que siempre está ahí para apoyarme en cualquier situación haciendo que la vida sea un poco más sencilla gracias a su cariño.

A mi tutor por darme la oportunidad de realizar este proyecto y su ayuda en los momentos que me sentía perdido.

Quiero dedicar este trabajo en especial a la memoria de mi padre y mis abuelos, quienes, aunque ya no están, continúan siendo una inspiración y una guía en mi vida.

FICHA DEL TRABAJO FIN DE TÍTULO	
Titulación	Grado en Ingeniería Informática
Modalidad	Proyecto de Ingeniería
Especialidad (solo TFG)	Tecnologías de la Información
Mención (solo TFG)	Sistemas Gráficos
Idioma	Español
Tipo	General
TFT en equipo	No
Autor/a	David López Martínez
Fecha de asignación	02/11/2022
Descripción corta	Se ha realizado un remake del videojuego clásico DOOM de 1993. Creando desde cero y con el intento de ser fiel al videojuego original, pero mejorando su motor gráfico y su jugabilidad.

NORMAS APLICADAS EN ESTE DOCUMENTO	
LOCALES	
TFT-UJA:2017	Normativa de Trabajos Fin de Grado, Fin de Máster y otros Trabajos Fin de Título de la Universidad de Jaén (Normativa marco UJA aprobada en Consejo de Gobierno)
TFT-EPSJ:2017	Normativa sobre Trabajos Fin de Grado y Fin de Máster en la Escuela Politécnica Superior de Jaén (Normativa EPSJ aprobada en Junta de Escuela)
TFT-EPSJ	Criterios de evaluación y normas de estilo para TFG y TFM de la Escuela Politécnica Superior de Jaén
NACIONALES E INTERNACIONALES	
ISO 2145:1978	Documentación - Numeración de divisiones y subdivisiones en documentos escritos
UNE 50132:1994	Traducción de la ISO 2145
APA 6ª edición	Estilo de referencias y citas de APA (American Psychological Association)

NORMAS UTILIZADAS COMO BASE O REFERENCIA	
NACIONALES	
UNE 157001:2014	Criterios generales para la elaboración formal de los documentos que constituyen un proyecto técnico
UNE 157801:2007	Criterios generales para la elaboración de proyectos de sistemas de información
*Estas normas se han utilizado **como base o referencia** para la inclusión de algunos contenidos y definiciones sobre elaboración de proyectos, entendiendo como **proyecto** la documentación consensuada entre una empresa y un cliente, que da lugar al perfeccionamiento de un contrato para la elaboración de una obra o la prestación de un servicio. Por consiguiente, no debe esperarse la aplicación de estas normas en cuanto a la completitud de los contenidos ni a la organización de los mismos.*	

Contenido

Índice de ilustraciones

Índice de tablas

1 ESPECIFICACIÓN DEL TRABAJO

En este capítulo se presenta la especificación del trabajo, con una estructura y contenidos **inspirados** en los criterios y recomendaciones que establece la norma UNE 157801:2007 - "*Criterios Generales para la elaboración de proyectos de Sistemas de Información*".

A lo largo del documento se utilizarán términos y acrónimos cuya descripción aparecen en el apartado 6 (Definiciones y abreviaturas).

1.1 Introducción

En esta memoria se describirá de forma detallada el diseño y desarrollo de un "remake" de un videojuego clásico.

Lo primero que hay que saber es diferenciar los conceptos de remakes, remasters y reboots:

- **Remake**: Un remake es una reinterpretación completa de un juego existente, donde se reconstruye desde cero para adaptarse a las tecnologías y estándares actuales. Los remakes suelen conservar la historia y las mecánicas centrales del juego original, pero se rediseñan gráficos, sonidos y a menudo se mejoran o amplían aspectos de la jugabilidad. La finalidad es crear una experiencia moderna que capture la esencia del juego original. Ejemplos notables incluyen el remake de "Resident Evil 2" y "Final Fantasy VII Remake".
- **Remaster**: Un remaster es una versión actualizada de un juego que preserva su contenido esencial, pero mejora los aspectos técnicos. Aquí, los gráficos, el audio y, a veces, las mecánicas se mejoran para adaptarse a las resoluciones y estándares de calidad actuales. A diferencia de los remakes, los remasters no alteran sustancialmente la estructura del juego original ni sus mecánicas principales. Ejemplos famosos de remasters incluyen la "BioShock: The Collection" y "The Elder Scrolls V: Skyrim Special Edition".

- **Reboot**: Un reboot es un reinicio completo de una franquicia. En lugar de continuar la historia o el diseño de los juegos anteriores, un reboot toma elementos clave y los reinterpreta con una visión fresca y actualizada. Los reboots pueden cambiar significativamente la dirección de la trama, personajes y jugabilidad. La finalidad es atraer a nuevos públicos y revitalizar el interés en la serie. Ejemplos notables son el reboot de "Tomb Raider" y "DOOM" de 2016.

Estas estrategias de la industria de los videojuegos no solo permiten que los juegos clásicos se mantengan relevantes, sino que también brindan nuevas oportunidades para atraer a una audiencia más amplia, aprovechando la nostalgia y actualizando la experiencia para las generaciones actuales.

Al videojuego al que se le va a realizar el remake es “DOOM”, lanzado originalmente para PC el 10 de diciembre de 1993. Se estableció como un pionero en el género de los first person shooter (FPS). La combinación de su icónica banda sonora original compuesto por Robert Prince y su distintiva estética, a cargo del talentoso diseñador gráfico John Romero, consolidaron su popularidad. Este impacto se vio aún más amplificado en las versiones posteriores para diversas plataformas, donde se abordaron algunas de las carencias previamente mencionadas.

Dada su relevancia en la historia de los videojuegos y el contexto en el que fue creado, consideramos pertinente llevar a cabo este proyecto. No solo servirá para mostrar los conocimientos que hemos adquirido durante nuestros años de estudio en el campo de la ingeniería informática, al recrear el juego desde sus cimientos y añadir nuevas funcionalidades, sino que también nos brindará la oportunidad de rendir homenaje a uno de los títulos que ha influido en numerosos juegos contemporáneos y del pasado.

Este esfuerzo no solo constituye un tributo al juego que ha dejado su huella en innumerables títulos que le siguieron, sino que también es un homenaje personal al primer juego de estilo FPS al que jugué, el cual despertó mi profundo interés por la tecnología y me motivó a emprender una carrera en informática.

1.2 Objetivos del trabajo

El objetivo principal del trabajo será el diseño y desarrollo de un remake del título “DOOM” juego original de 1993. Para ello se deberá de incidir en varios aspectos a tener en cuenta durante la realización:

- Seleccionar el motor gráfico en el que se llevará a cabo el remake y adquirir las habilidades necesarias para utilizar sus herramientas.
- Realizar un análisis detallado de las mecánicas, niveles y objetivos del juego original para poder trasladarlos fielmente al remake.
- Desarrollar las mecánicas fundamentales del juego, enfocándose en aspectos como la jugabilidad, el combate y la exploración.
- Diseñar y modelar los niveles que va a contener el juego.
- Buscar y diseñar modelos 3D, efectos de sonido y música que se adecuen al análisis del videojuego original.
- Modelar y animar al personaje principal, enemigos y algunos objetos del juego utilizando el software auxiliar necesario para ello.
- Diseñar y desarrollar la interfaz de usuario y su interacción.

Una vez que se cumplan estos objetivos, se logrará el resultado de la adaptación del juego original DOOM ofreciendo una experiencia más renovada para los jugadores.

Este proyecto no solo representará un tributo al influyente juego que sentó las bases de los FPS, sino que también proporcionará la oportunidad de aplicar y expandir nuestros conocimientos en diseño de videojuegos y tecnología.

1.3 Antecedentes y estado del arte

En los primeros años de los videojuegos, los remakes eran conocidos como "conversiones" y rara vez se asociaban con la nostalgia. Debido a las limitaciones y las diferencias de hardware, los juegos que aparecían en múltiples plataformas a menudo tenían que ser completamente rehechos. Estas conversiones a menudo implicaban cambios significativos en los gráficos y la jugabilidad, y podrían ser

consideradas como remakes, aunque se distinguen de los remakes posteriores en gran medida por su intención. Una conversión se creaba con el objetivo principal de adaptar un juego a una pieza específica de hardware, generalmente contemporánea o casi contemporánea al lanzamiento original.

A medida que avanzaba la tecnología, los remakes comenzaron a tomar una forma más ambiciosa. Los juegos clásicos vieron oportunidades para ser recreados con gráficos mejorados, sonido envolvente y mecánicas pulidas. Sin embargo, el enfoque en la modernización y la mejora técnica comenzó a diferenciar a los remakes de las conversiones. En este proceso, los juegos originales a menudo eran reimaginados para ajustarse a los estándares y capacidades de los juegos más recientes en la serie, llevando a reinterpretaciones más modernas.

El renacimiento contemporáneo de los remakes ocurrió con el auge del fenómeno del retrogaming. Las compañías vieron en los remakes una forma de revivir marcas nostálgicas y revitalizar franquicias envejecidas. Esta época también presenció remakes exitosos como "Super Mario All-Stars", que revitalizó la serie Mario de NES con gráficos mejorados. Los remakes también capitalizaron la evolución tecnológica, permitiendo que juegos de generaciones anteriores fueran reinterpretados con mejoras visuales y jugables. A medida que las plataformas de descarga como Xbox Live Arcade y PlayStation Network ganaron popularidad, los remakes se volvieron aún más accesibles para los jugadores, con precios más bajos y oportunidades para revivir juegos clásicos a precios adecuados para el mercado retro. [2]

El juego en el que se basa este remake, DOOM, fue lanzado el 10 de diciembre de 1993, para el sistema operativo DOS. Fue uno de los juegos más importantes de 1993 que la mayoría de los PC tenían instalado este juego.

En ese momento, el género de los FPS estaba en sus primeras etapas. DOOM se destacó por su enfoque en la acción rápida y la perspectiva en primera persona, que crearon una experiencia de juego nueva y emocionante. El juego también innovó en la tecnología al presentar el "motor DOOM", una herramienta que permitía la

representación de entornos tridimensionales a través de mapas de texturas. Esta tecnología cambió el juego, allanando el camino para futuros títulos en 3D.

En este juego encarnamos a un marine espacial, conocido popularmente como Doomguy, que se encuentra en una misión rutinaria en una estación de Fobos, una de las lunas de Marte, cuando de repente se produce un fallo en un experimento abriendo las puertas del infierno, liberando a demonios y espíritus que se apoderan de los cuerpos de los marines caídos convirtiéndolos en zombies, infestando las instalaciones. Nuestro protagonista tiene que abrirse paso entre las hordas de no muertos y demonios para conseguir cerrar el portal. [3]

Ilustración 1.1 DOOM 1993

DOOM fue un éxito crítico y comercial tras su lanzamiento, obteniendo una reputación como uno de los mejores y más influyentes juegos de todos los tiempos. Se estima que vendió alrededor de 3.5 millones de copias para 1999, y se estima que hasta 20 millones de personas lo habían jugado en los dos años posteriores a su lanzamiento. Se le ha denominado el "padre" del género de los FPS y se considera uno de los juegos más importantes en su categoría. Dio lugar a una variedad de imitadores y clones, así como una comunidad de **modders** y de **speedrunners**. DOOM ha sido portado oficial y no oficialmente a una amplia variedad de plataformas

desde entonces y ha sido seguido por varios juegos en la serie, incluyendo DOOM II (1994), DOOM 3 (2004), DOOM (2016) y DOOM Eternal (2020).

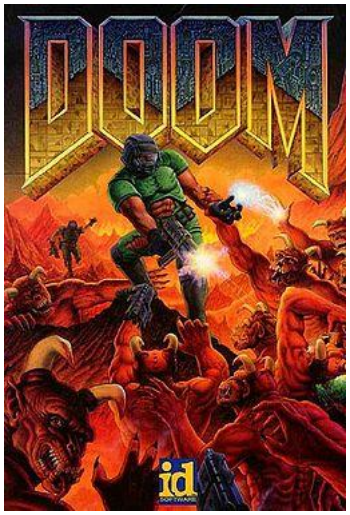

Ilustración 1.2 Portada DOOM

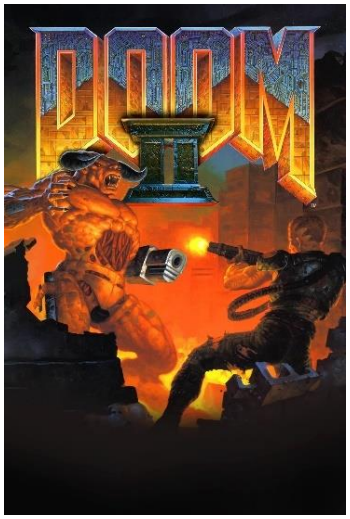

Ilustración 1.3 Portada DOOM 2

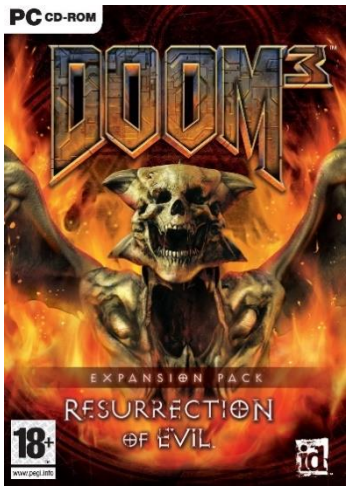

Ilustración 1.4 Portada DOOM 3

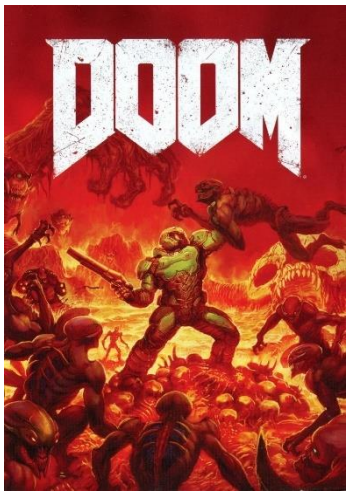

Ilustración 1.5 Portada DOOM 2016

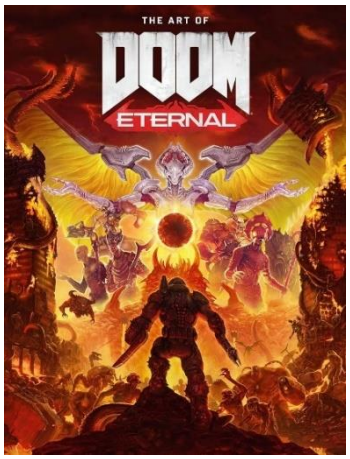

Ilustración 1.6 Portada DOOM eternal

1.4 Descripción de la situación de partida

1.4.1 Descripción del entorno actual

Después del éxito del primer DOOM se lanzaron otros dos juegos continuando con la historia de este: “DOOM II: Hell on Earth” en 1994 y “DOOM 3” en 2004.

Ilustración 1.7 Gameplay DOOM 2

Ilustración 1.8 Gameplay DOOM 3

Tras esto siguieron realizando remasters de estos videojuegos durante el tiempo hasta que en 2016 decidieron realizar un reboot de la saga con el videojuego “DOOM” comúnmente conocido como DOOM 2016 para evitar confusiones con el

videojuego original. Fue bien recibido por críticos y jugadores. La campaña para un jugador, los gráficos, la banda sonora y la jugabilidad recibieron considerable elogio, con los críticos reconociendo al juego por recapturar el espíritu de los clásicos juegos de Doom y los FPS de la década de 1990. Fue el segundo videojuego más vendido en América del Norte y el Reino Unido en el mes de su lanzamiento, y vendió más de 500,000 copias para PC en el mismo período de tiempo.

Ilustración 1.9 Gameplay DOOM 2016

Una secuela titulada “DOOM Eternal” fue lanzada en marzo de 2020 junto con su expansión “The Ancient Gods” divida en 2 partes siendo lanzadas la primera en 2020 y la segunda en 2021.

Ilustración 1.10 Gameplay DOOM eternal

1.4.2 Resumen de las deficiencias y carencias identificadas

Como hemos visto no existe ningún remake oficial de este juego de 1993 por lo tanto las deficiencias y carencias siguen siendo las mismas que tenía cuando el juego fue lanzado.

1.4.2.1 Deficiencias

- Una de las principales deficiencias que nos encontramos es el apartado gráfico ya que las texturas son de baja calidad y no son atractivo a día de hoy.

Ilustración 1.11 Mapa original E1M1

- El movimiento del personaje es algo tosco, ya que no sé permite un apuntado realista ni un movimiento libre del ratón.

1.4.2.2 Carencias

- Modelado 3D real de los **assets**, DOOM tenía un modelado de niveles en 3D pero los assets eran **sprites** en 2.5D.

Ilustración 1.12 Enemigo DOOM

- Sistema de apuntado y punto de mira del arma, DOOM no tenía un sistema de apuntado o alguna forma de indicar donde estas disparando.

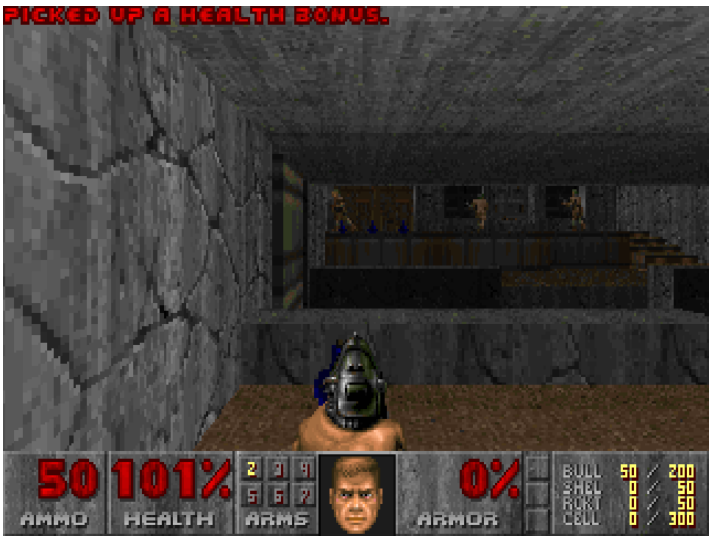

Ilustración 1.13 HUD DOOM

1.5 Requisitos iniciales

Los requisitos principales de este proyecto son:

- La apariencia debe de ser parecida a la del videojuego original, por lo que se deberán buscar o modelar objetos 3D que sean similares a los del original.

- Las mecánicas del videojuego deben de estar basadas en el videojuego original, intentando que sean mecánicas mejoradas del título clásico.
- Se debería de realizar una interfaz básica para poder comenzar una nueva partida, cargar una partida anterior, pausar la partida, modificar opciones gráficas o de sonido.
- Se debe de incluir banda sonora, cada acción debe de tener un sonido asociado.

1.6 Alcance

Este proyecto se entregará cuando su desarrollo haya sido completado.

Se entregará el ejecutable que permitirá correr una versión del remake del DOOM. Que incluirá dos niveles jugables, los objetos, armas y enemigos correspondientes a esos niveles, además incluirá también un menú principal y un menú de pausa.

También se entregará el proyecto donde se incluirá todo el código, assets y elementos de audio necesarios para la realización del mismo.

Por último, también se hará entrega de la documentación final del proyecto donde se explica detalladamente el desarrollo del mismo.

1.7 Hipótesis y restricciones

El TFG se define como una asignatura de 12 créditos, lo que supone que la duración total del proyecto será de 300 horas, incluyendo todas las etapas del ciclo de vida, con la excepción del mantenimiento. Por consiguiente, la principal restricción aplicable es la limitación de la duración del trabajo.

Teniendo en cuenta esta restricción de tiempo y que se debe de dividir tanto en el desarrollo del proyecto como en la documentación y que este es realizado por una sola persona es posible que la calidad del juego sea menor a la esperada.

También hay que tener en consideración que durante el desarrollo del proyecto se puedan encontrar **bugs** o fallos de programación que retrasen el desarrollo del

mismo causando que algunos elementos anteriormente planteados tengan que ser reducidos o eliminados.

Por último, se debe de considerar los conocimientos básicos en modelado y animación en 3D del desarrollador, por lo que se depende en gran medida de modelados gratuitos.

1.8 Estudio de alternativas y viabilidad

Para el desarrollo de este proyecto se ha realizado un estudio de distintas alternativas para realizar el mismo.

1.8.1 Motores Gráficos

Esta es posiblemente la elección más importante a la hora de desarrollar el proyecto, ya que, es un software que proporciona una variedad de herramientas y funcionalidades para facilitar el desarrollo de videojuegos y aplicaciones interactivas. Este tipo de software se encarga de manejar aspectos técnicos como el renderizado de gráficos, la física, la animación, la inteligencia artificial y otros componentes esenciales para crear mundos virtuales inmersivos. Los motores gráficos permiten a los desarrolladores enfocarse en la creación de contenido y jugabilidad en lugar de tener que construir todos los componentes desde cero.

Podemos realizar una gran diferenciación en los motores gráficos:

- **Motores Propios**: Los motores propios son desarrollados internamente por un estudio de videojuegos específico para sus proyectos. Estos motores pueden ser altamente personalizados y adaptados para satisfacer las necesidades y visión creativa de un juego en particular. Aunque ofrecen un alto grado de control y flexibilidad, también pueden ser más intensivos en términos de tiempo y recursos, ya que requieren construir y mantener todas las funcionalidades desde cero.

  Por ejemplo, Frostbite es un motor propio desarrollado por EA DICE que se utiliza en juegos como la serie "Battlefield" y "FIFA". Es conocido por

su capacidad para crear gráficos impresionantes y efectos visuales realistas.

- **Motores Globales (Third-Party Engines)**: Los motores globales son plataformas de desarrollo creadas por empresas especializadas en software de juego. Estos motores se licencian para ser utilizados por diferentes estudios y desarrolladores de todo el mundo. Ofrecen un equilibrio entre eficiencia y versatilidad, ya que proporcionan una base sólida de herramientas y características preconstruidas.

Ambos enfoques tienen sus ventajas y desafíos. Los motores propios ofrecen un control total sobre el proceso de desarrollo, pero pueden ser más costosos y requerir más tiempo. Los motores globales son más eficientes y brindan acceso a una comunidad y recursos más grandes, pero pueden limitar cierto grado de personalización.

Debido a las características de este proyecto elegiremos un motor global dentro de las distintas posibilidades que existen

1.8.1.1 Unity

Unity es uno de los motores más populares y accesibles del mercado. Se lanzó en 2005 y ha crecido de manera significativa en cuanto a capacidades y versatilidad. Unity es conocido por ser una herramienta fácil de usar para principiantes y profesionales, y permite desarrollar para múltiples plataformas, como consolas, PC, móviles, y realidad virtual. [4]

Ilustración 1.14 Logo Unity

Unity cuenta con un gran número de ventajas que lo han convertido en una de las herramientas más utilizadas de la industria de videojuegos:

- **Facilidad de uso:** La curva de aprendizaje es más amigable en comparación con otros motores. Su interfaz de usuario intuitiva y documentación extensa hacen que sea ideal para desarrolladores novatos.
- **Gran comunidad:** Tiene una comunidad muy activa y recursos en línea abundantes (tutoriales, foros, assets gratuitos y de pago).
- **Asset Store:** Unity cuenta con una tienda de recursos que permite a los desarrolladores acceder a modelos 3D, scripts, animaciones, texturas y más.
- **Soporte para Realidad Virtual y Realidad Aumentada:** Unity es ideal para el desarrollo de aplicaciones y juegos en VR y AR, con soporte nativo para dispositivos como Oculus, HTC Vive, y HoloLens.
- **Personalización:** El motor es altamente personalizable a través de C#, permitiendo una gran flexibilidad en cuanto a lógica de juego y mecánicas.

Aunque también cuenta con algunas limitaciones como:

- **Rendimiento:** A menudo, Unity es menos eficiente que otros motores (como Unreal Engine) en cuanto a rendimiento gráfico y capacidad de manejar proyectos de gran escala con gráficos muy detallados.
- **Gráficos menos avanzados:** Aunque ha mejorado con el tiempo, la calidad gráfica nativa de Unity está por debajo de motores como Unreal Engine.
- **Licencias:** La versión gratuita tiene limitaciones, y las licencias de empresa son costosas.

1.8.1.2 Unreal Engine

Unreal Engine, desarrollado por Epic Games, es uno de los motores más potentes en cuanto a gráficos, y es ampliamente utilizado para desarrollar juegos AAA. Es conocido por su impresionante capacidad de renderizado y sus herramientas avanzadas, lo que lo hace ideal para proyectos con gráficos de alta calidad.

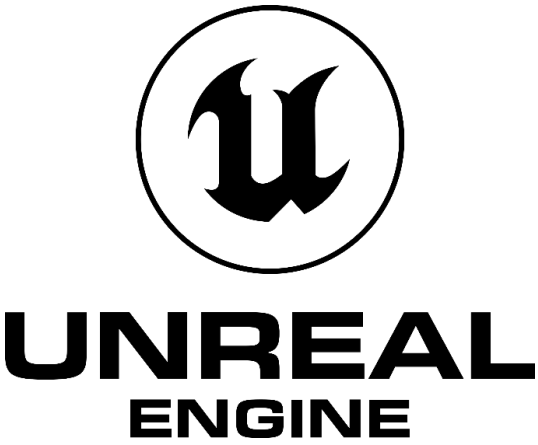

Ilustración 1.15 Logo Unreal Engine

Unreal engine, desde el lanzamiento de Unreal Engine 5 ha implementado una serie de ventajas que ha hecho que crezca exponencialmente, como son: [5]

- **Licencias:** La versión gratuita tiene limitaciones, y las licencias de empresa son costosas.
- **Gráficos de alta calidad:** Unreal Engine es famoso por ofrecer gráficos fotorrealistas gracias a su motor de renderizado avanzado. Utiliza tecnologías como el trazado de rayos en tiempo real (ray tracing) que ofrece una calidad visual impresionante.
- **Motor de física robusto:** Su motor de física es uno de los más avanzados, lo que facilita la creación de simulaciones físicas realistas.
- **Blueprints (Sistema de scripting visual):** Permite a los desarrolladores crear complejos sistemas de juego sin necesidad de programar, lo cual es ideal para artistas y diseñadores que no son expertos en programación.
- **Popularidad en juegos AAA:** Es el motor preferido para estudios grandes y proyectos de alto presupuesto, usado en juegos como "Fortnite", "Gears of War" y "Final Fantasy VII Remake".
- **Comunidad y recursos:** Tiene una gran comunidad y una extensa biblioteca de tutoriales, ejemplos y assets gratuitos (incluyendo el acceso a algunos recursos del propio Epic Games).
- **Gratis con regalías:** Motor gratuito, Unreal cobra el 5% a partir de que el proyecto genere más de 1.000.000€ en ventas.

Aunque, esta potencia gráfica y capacidad avanzada tienen un precio:

- **Curva de aprendizaje:** Es más difícil de aprender que Unity, especialmente si no se tiene experiencia en programación o desarrollo de videojuegos.
- **Requisitos de hardware:** Unreal Engine requiere un equipo más potente para funcionar sin problemas, especialmente en proyectos grandes o con gráficos muy complejos.
- **Peso del motor y compilación:** Los proyectos creados en Unreal tienden a ser más pesados, tanto en términos de almacenamiento como de memoria RAM y consumo de recursos.

1.8.1.3 CryEngine

CryEngine, desarrollado por Crytek, es conocido por su capacidad de ofrecer gráficos de altísima calidad, similar a Unreal Engine. Fue popularizado con títulos como "Crysis", y se ha destacado por sus herramientas de creación de mundos grandes y realistas. [6]

Ilustración 1.16 Logo CryEngine

Ofrece ciertas ventajas como son:

- **Gráficos impresionantes:** Como Unreal, CryEngine es capaz de ofrecer gráficos fotorrealistas con iluminación dinámica, sombreado y renderizado de alta calidad.
- **Potente motor de física:** CryEngine tiene un avanzado motor de física que soporta simulaciones complejas y realistas, como fluidos, partículas y destrucción en tiempo real.
- **Herramientas de creación de mundos:** Es ideal para crear entornos abiertos y detallados, lo que lo hace adecuado para juegos de tipo "sandbox" o de exploración masiva.

- **Gratis con regalías:** El motor es gratuito de utilizar, con regalías opcionales, lo que significa que solo pagas a Crytek si el proyecto genera ingresos.

Aunque en cambio, ofrece una serie de grandes desventajas:

- **Curva de aprendizaje:** CryEngine es más difícil de aprender que Unity y, en algunos casos, incluso que Unreal. No es tan accesible para principiantes y su documentación es menos extensa.
- **Soporte y comunidad:** La comunidad de CryEngine es más pequeña en comparación con Unity o Unreal, lo que puede hacer más difícil encontrar recursos y soporte.
- **Complejidad de optimización:** Requiere conocimientos avanzados para optimizar adecuadamente los juegos, especialmente en proyectos grandes, lo que puede resultar complicado para pequeños equipos de desarrollo.

1.8.1.4 Amazon Lumberyard

Amazon Lumberyard es un motor de juego basado en una versión modificada de CryEngine. Amazon lo lanzó en 2016, con el objetivo de integrar el desarrollo de videojuegos con los servicios en la nube de Amazon Web Services (AWS) y Twitch, lo que lo hace una opción interesante para proyectos con grandes ambiciones de conectividad. [7]

Ilustración 1.17 Logo Lumberyard

Como está basado en CryEngine tiene las ventajas del mismo, pero además tiene otras ventajas como:

- **Integración con AWS:** Si planeas desarrollar juegos multijugador o con características en línea, Lumberyard ofrece una integración nativa con los servicios de la nube de Amazon, lo que facilita la creación de juegos escalables y robustos en términos de servidores.

- **Integración con Twitch:** Ofrece funciones nativas para Twitch, lo que lo hace atractivo para juegos con funciones interactivas para transmisiones en vivo y comunidades de jugadores.
- **Gratis sin regalías:** El motor es completamente gratuito y no tiene regalías, lo que lo hace atractivo desde una perspectiva de costos. Amazon solo cobra por el uso de sus servicios AWS.

Y como podemos esperar tiene las mismas desventajas que CryEngine añadiendo algunas:

- **Curva de aprendizaje:** Similar a CryEngine, puede ser difícil de aprender debido a la falta de documentación extensa y una comunidad más pequeña en comparación con Unity o Unreal.
- **Dependencia de AWS:** Aunque es gratuito, está diseñado para funcionar mejor con AWS, lo que puede hacer que los costos se acumulen rápidamente si el proyecto depende de los servicios en la nube de Amazon.
- **Comunidad y soporte limitado:** En comparación con otros motores, la comunidad es más pequeña y el número de recursos disponibles (tutoriales, assets) es mucho más limitado.

1.8.2 Software de modelado 3D y animación

El software de modelado y 3D es esencial en industrias como el desarrollo de videojuegos, la animación, el cine y la realidad virtual. Estos programas permiten a los artistas crear, texturizar, animar y renderizar modelos tridimensionales que pueden usarse en una amplia gama de aplicaciones. Existen muchas herramientas de modelado 3D en el mercado, cada una con sus propias características y fortalezas.

1.8.2.1 Blender

Blender es un software de código abierto y gratuito que ha ganado una inmensa popularidad gracias a su capacidad para ofrecer una gama completa de herramientas para la creación 3D. Con Blender, puedes realizar tareas de modelado, texturizado, animación, esculpido digital, simulación, composición y renderizado. [8]

Ilustración 1.18 Logo Blender

Gracias a su naturaleza de código abierto, cuenta con una comunidad muy activa que constantemente mejora el software y produce tutoriales, complementos y recursos.

1.8.2.2 Maya

Maya, desarrollado por Autodesk, es uno de los programas más utilizados en la industria del cine, la televisión y los videojuegos para la creación de contenido 3D. Es famoso por sus potentes herramientas de animación y modelado, lo que lo convierte en una opción favorita para estudios profesionales que trabajan en proyectos a gran escala. [9]

Ilustración 1.19 Logo Maya

Maya se ha utilizado en la producción de películas de animación y efectos visuales ganadoras de premios.

Maya es un software de pago, y su licencia es de un coste muy elevado. Por lo que es prohibitivo para pequeños artistas y estudios.

1.9 Descripción de la solución propuesta

1.9.1 Justificación del motor gráfico

Teniendo en cuenta las ventajas y desventajas de los distintos motores gráficos considerados anteriormente, y teniendo en consideración el juego que se va a desarrollar, se ha tomado la decisión de utilizar Unreal Engine ya que permitirá obtener unos resultados muy buenos.

1.9.2 Justificación del software de modelado 3D y animación

Una vez expuestos ambas opciones, la opción más lógica es Blender debido a ser totalmente gratuito y su gran comunidad que proporciona distintos tutoriales para su aprendizaje.

1.10 Tecnologías utilizadas

En resumen, las tecnologías utilizadas en este proyecto son:

- **Unreal Engine 5.2:** Motor gráfico en el que se realizará el videojuego.
- **Blender 3.6:** Software de modelado 3D y animación.
- **Visual Studio 2019:** Es el IDE que utiliza Unreal Engine para la programación en C++.
- **Programación en C++ y blueprints:** Además de C++ se usará los blueprints para crear lógicas complejas de juego sin necesidad de escribir grandes cantidades de código.

1.11 Metodología de desarrollo de software

Para el desarrollo de este proyecto se ha hecho uso de la metodología ágil Scrum ya que permite un enfoque flexible, basado en entregas iterativas de incrementos funcionales. Esto facilita la implementación de mejoras constantes, como el desarrollo de niveles, modelos 3D, IA y mecánicas de combate, y permite adaptar el proyecto a los cambios que puedan surgir en el camino.

1.12 Estimación del tamaño y esfuerzo

Ya que el presente proyecto es un TFT, no existen restricciones de tipo económico, sino de tipo temporal (un número aproximado de horas). Por consiguiente, los cálculos de tamaño del proyecto están supeditados el tiempo disponible. En cuanto al esfuerzo, se dispone de tan un solo efectivo (la persona autora del trabajo).

1.13 Planificación temporal

A continuación, se muestra un diagrama de Gantt donde podemos observar la planificación de las distintas iteraciones durante el tiempo.

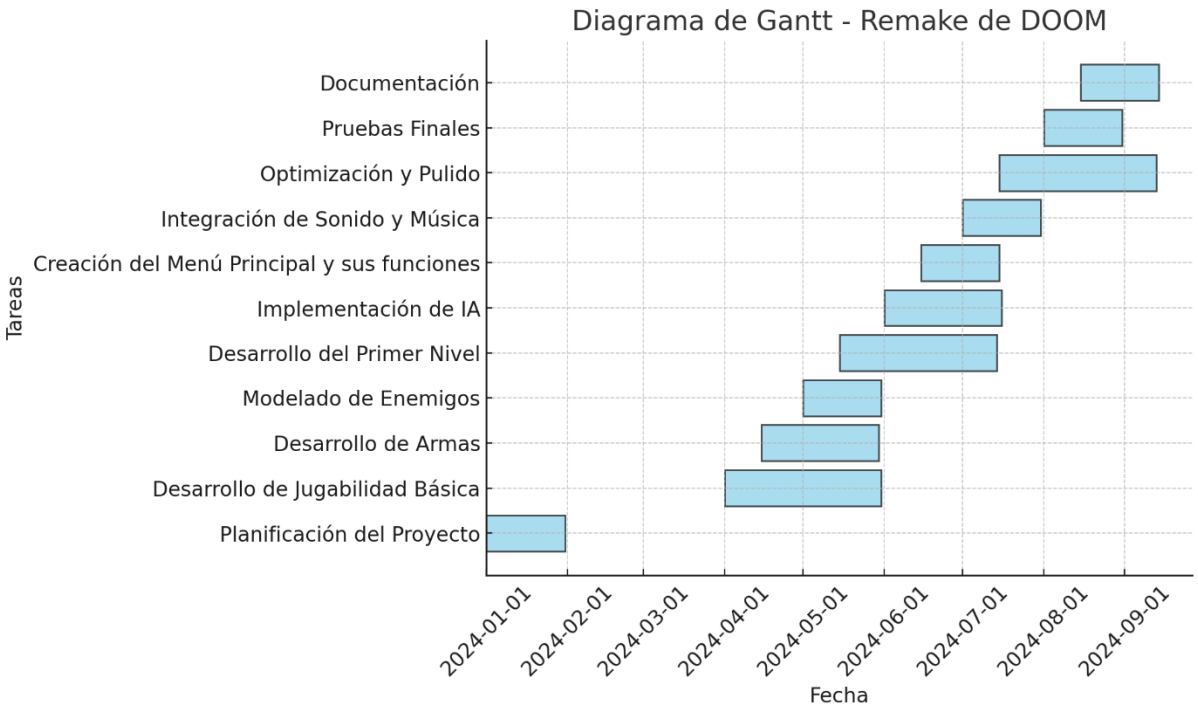

Ilustración 1.20 Diagrama de Gantt

1.14 Presupuesto

A la hora de estimar un presupuesto hay que tener en cuenta distintos argumentos como los recursos humanos y los medios técnicos necesarios para la realización del proyecto.

1.14.1 Recursos Humanos

Si el proyecto que se ha planteado se realizase con un equipo profesional, como mínimo serían necesarios:

- **Un diseñador de videojuegos.**
- **Un gestor de proyectos.**
- **Un artista 3D/Animador/Diseñador de niveles.**
- **Un ingeniero de sonido.**
- **Un programador de videojuegos.**
- **Un tester de videojuegos.**

Para este trabajo todas las tareas las realiza una misma persona, por lo tanto, para el presupuesto tendremos en cuenta esto mismo. El sueldo medio en España para un programador de videojuegos junior, con experiencia de 0-3 años, es de 22.500€ al año. Como la etapa de desarrollo del proyecto nos ocupa unos 6 meses y el mismo ha sido realizado por 1 sola persona. [4]

Recursos humanos	Coste(€)/Mes	Duración	Total
Programador	1875€/mes	6 meses	11250€
Coste total:			11250€

Tabla 1.1.1 Desglose del gasto en recursos humanos

1.14.2 Software

Para el cálculo de los costes se tiene que tener en cuenta que de los softwares utilizados para este proyecto sólo Microsoft Office Word, es de pago, con un coste de 7€ al mes.

Software	Coste(€)/Mes	Duración	Total
Unreal	0,00€/mes	6 meses	0€
Blender	0,00€/mes	6 meses	0€
Microsoft Office Word	7€	2 meses	14€
Coste total:			14€

Tabla 1.1.2 Desglose del gasto en software

1.14.3 Hardware

A la hora de tener en cuenta el hardware hay que tener en cuenta de que se ha trabajado en dos equipos para el desarrollo del proyecto, un ordenador de sobremesa que tiene un coste de unos 1000€ y un portátil que tiene un coste de unos 1200€. Asumiremos la amortización de los equipos a 5 año.

Hardware	Coste(€)/Mes	Duración	Total
Sobremesa	16,67€/mes	6 meses	100,02€
Portátil	20€/Mes	6 meses	120€
Coste total			220,02€

Tabla 1.1.3 Desglose presupuesto total

1.14.4 Presupuesto total

Para el cálculo total del presupuesto sumaremos los gastos anteriormente desglosados y además se le añadirá un coste derivado del uso de instalaciones, facturas de servicios y gastos de gestión, con lo que añadiremos un 10% de sobrecoste al proyecto.

Concepto	Coste(€)/Mes	Duración	Total
Recursos humanos	1875€/mes	6 meses	11250€
Software	7€/Mes	2 meses	14€
Hardware	36.67€/Mes	6 meses	220,02€
Coste total			11484,02€

Tabla 1.1.4 Desglose presupuesto total

Concepto	Coste(€)
Coste del proyecto	11484,02€
Gastos derivados	1148,40€
Coste total del proyecto	12632,42€

Tabla 1.1.5 Desglose importe total del proyecto

2 DISEÑO INICIAL

Al hacer uso de una metodología ágil en este apartado se dará una descripción general del diseño previo del proyecto, que ampliaremos en la parte del desarrollo del mismo.

2.1 Especificaciones del sistema

En este apartado hablaremos de los requisitos funcionales y no funcionales del proyecto.

2.1.1 Requisitos Funcionales

Los requisitos funcionales son los requisitos que nos permiten realizar el proyecto.

- **Iniciar una nueva partida**: El jugador debe poder comenzar una nueva campaña desde el principio.
- **Cargar partida**: El jugador debe poder cargar una partida guardada previamente.
- **Guardar partida**: Debe haber opciones para guardar el progreso del jugador en cualquier momento o en puntos específicos.
- **Explorar los mapas**: El jugador debe poder moverse libremente por los niveles y mapas del juego, con la posibilidad de explorar distintas áreas.
- **Interacción con enemigos**: Los enemigos deben reaccionar al jugador y atacarlo, proporcionando combates dinámicos.
- **Sistema de combate dinámico**: El jugador debe poder enfrentarse a enemigos en tiempo real, esquivar ataques y moverse rápidamente.

- **Obtención de armas y municiones**: El jugador debe poder recoger armas, munición y objetos de los escenarios.
- **Uso de objetos consumibles**: El jugador debe poder recoger y usar objetos como kits médicos y armaduras para restaurar salud o mejorar su protección.
- **Acceso a menú de pausa**: El jugador debe poder acceder a un menú de pausa para revisar armas, configuraciones y estado del personaje.
- **Progresión de nivel**: El jugador debe poder completar los niveles y avanzar a niveles más difíciles.
- **Interacción con puertas y dispositivos**: El jugador debe poder interactuar con puertas, llaves y otros dispositivos para progresar en el nivel.
- **Desbloqueo de áreas ocultas**: El jugador debe poder encontrar áreas secretas escondidas en los niveles.
- **Usar armas especiales**: El jugador debe poder acceder a armas especiales y explosivos para enfrentar enemigos más poderosos.

2.1.2 Requisitos No Funcionales

Los requisitos no funcionales son aquellos que definen que características debe tener un sistema más allá de las funcionales. En nuestro caso definiremos las siguientes:

- **Interfaz intuitiva**: La interfaz del juego debe ser simple e intuitiva, para que el jugador pueda aprender rápidamente cómo moverse, disparar y usar objetos.
- **Gráficos actualizados**: El remake debe contar con gráficos mejorados en comparación con el juego original, aprovechando las capacidades gráficas modernas.
- **Optimización del rendimiento**: El juego debe ser capaz de correr de forma fluida en diferentes configuraciones de hardware, optimizando el uso de CPU y GPU.

- **Fiabilidad del sistema**: El juego debe ser estable, reduciendo al máximo la posibilidad de errores, pérdidas de progreso o caídas durante la ejecución.
- **Fácil mantenimiento del código**: El código debe estar bien documentado y estructurado, lo que facilitará su mantenimiento y futuras actualizaciones.
- **Diseño de sonido inmersivo**: El sonido y la música del juego deben ser envolventes, creando una atmósfera adecuada para la experiencia de DOOM.

2.2 Análisis y diseño del sistema

En esta parte vamos a ver los distintos casos de uso que podemos tener en nuestro proyecto además de imágenes de la interfaz del juego original en el que nos basamos para realizar la interfaz de proyecto. Utilizaremos el formato que

2.2.1 Casos de uso

Vamos a presentar los casos de uso para nuestro proyecto.

Casos de Uso			
CU-01	Crear una partida nueva	CU-09	Equipar distintas armas
CU-02	Cargar una partida	CU-10	Recoger vida
CU-03	Guardar partida	CU-11	Recoger armadura
CU-04	Moverse	CU-12	Recibir daño
CU-05	Disparar	CU-13	Infligir daño
CU-06	Recargar	CU-14	Menú de pausa
CU-07	Recoger munición	CU-15	Modificar opciones gráficas
CU-08	Recoger distintas armas	CU-16	Modificar opciones de sonido

Tabla 2.1 Casos de uso

2.2.1.1 CU-01 Crear una partida nueva

CU-01	Crear una partida nueva
Descripción	El Usuario puede crear una partida nueva, borrando la anterior si existiese y comenzar el juego desde el primer nivel.
Precondición	Encontrarse en el menú de inicio.
Secuencia normal	1. El usuario elige “Nueva Partida” 2. El sistema revisa si existe alguna partida guardada anterior. 3. El usuario comienza su nueva partida.
Postcondición	El usuario comienza el juego desde el principio, el sistema habrá creado un nuevo archivo de guardado eliminando, si existiese, el anterior
Excepciones	Si existen datos de guardado se borran.
Comentarios	-

Tabla 2.2 CU-01 Crear una partida nueva

2.2.1.2 CU-02 Cargar una partida

CU-02	Cargar una partida
Descripción	El Usuario puede crear una partida nueva, borrando la anterior si existiese y comenzar el juego desde el primer nivel.
Precondición	Debe existir al menos una partida guardada.
Secuencia normal	1. El jugador selecciona "Cargar Partida". 2. El sistema muestra las partidas guardadas disponibles. 3. El jugador selecciona una partida guardada. 4. El sistema carga el nivel y estado correspondiente.
Postcondición	El jugador continúa la partida desde el punto guardado.
Excepciones	Si no hay partidas guardadas, el sistema muestra un mensaje de error.
Comentarios	-

Tabla 2.3 CU-02 Cargar una partida

2.2.1.3 CU-03 Guardar partida

CU-03	Crear una partida
Descripción	El jugador guarda su progreso en una ranura disponible.
Precondición	El jugador debe estar en un punto del juego donde se permite guardar.
Secuencia normal	1. El jugador selecciona "Guardar Partida". 2. El sistema muestra las ranuras de guardado disponibles. 3. El jugador selecciona una ranura para guardar. 4. El sistema guarda el estado del juego.
Postcondición	El progreso se guarda en la ranura seleccionada.
Excepciones	Si existen datos de guardado se sobrescriben.
Comentarios	-

Tabla 2.4 CU-03 Guardar partida

2.2.1.4 CU-04 Moverse

CU-04	Moverse
Descripción	El jugador puede moverse en el entorno utilizando las teclas de dirección.
Precondición	El jugador debe estar en un nivel jugable.
Secuencia normal	1. El jugador presiona las teclas o usa el joystick para moverse. 2. El sistema actualiza la posición del jugador en el mapa. 3. El sistema detecta colisiones con el entorno.
Postcondición	El jugador se mueve libremente por el entorno.
Excepciones	Si el jugador choca con un obstáculo, el movimiento se detiene.
Comentarios	-

Tabla 2.5 CU-04 Moverse

2.2.1.5 CU-05 Disparar

CU-05	Disparar
Descripción	El jugador dispara el arma equipada.
Precondición	El jugador debe tener un arma equipada y munición disponible.
Secuencia normal	1. El jugador presiona el botón de disparo. 2. El sistema verifica si hay munición disponible. 3. El sistema dispara el arma y genera un proyectil.
Postcondición	El proyectil impacta el entorno o un enemigo.
Excepciones	Si no hay munición, no se realiza el disparo.
Comentarios	-

Tabla 2.6 CU-05 Disparar

2.2.1.6 CU-06 Recargar

CU-06	Recargar
Descripción	El jugador recarga el arma equipada
Precondición	El jugador debe tener munición adicional.
Secuencia normal	1. El jugador presiona el botón de recarga. 2. El sistema recarga el arma hasta su capacidad máxima.
Postcondición	El arma del jugador está completamente cargada.
Excepciones	Si no hay munición adicional, la recarga no se realiza.
Comentarios	-

Tabla 2.7 CU-06 Recargar

2.2.1.7 CU-07 Recoger Munición

CU-07	Recoger Munición
Descripción	El jugador recoge munición del entorno.
Precondición	Debe haber un paquete de munición disponible.
Secuencia normal	1. El jugador se acerca a un paquete de munición. 2. El sistema permite al jugador recoger la munición. 3. El sistema actualiza la cantidad de munición del jugador.
Postcondición	El jugador recibe munición
Excepciones	Si el jugador ya tiene munición completa, no puede recoger más.
Comentarios	-

Tabla 2.8 CU-07 Recoger Munición

2.2.1.8 CU-08 Recoger distintas armas

CU-08	Recoger distintas armas
Descripción	El jugador puede recoger diferentes armas del entorno.
Precondición	Debe haber un arma disponible en el entorno.
Secuencia normal	1. El jugador se acerca al arma. 2. El sistema permite al jugador recoger el arma. 3. El arma se equipa automáticamente o se agrega al inventario.
Postcondición	El jugador adquiere un arma nueva.
Excepciones	Si el jugador ya tiene esa arma, sólo recoge munición.
Comentarios	

Tabla 2.9 CU-08 Recoger distintas armas

2.2.1.9 CU-09 Equipar distintas armas

CU-09	Equipar distintas armas
Descripción	El jugador puede equipar las armas que tiene en su inventario.
Precondición	El jugador debe tener más de una arma en su inventario.
Secuencia normal	1. El jugador abre el menú de armas. 2. El sistema muestra las armas disponibles. 3. El jugador selecciona un arma. 4. El sistema equipa el arma seleccionada.
Postcondición	El jugador tiene equipada una nueva arma.
Excepciones	Si el arma elegida no está en posesión del jugador, no se realiza el cambio de arma.
Comentarios	-

Tabla 2.10 CU-09 Equipar distintas armas

2.2.1.10 CU-10 Recoger vida

CU-10	Recoger vida
Descripción	El jugador recoge paquetes de vida para recuperar salud.
Precondición	Debe haber un paquete de vida en el entorno.
Secuencia normal	1. El jugador se acerca al paquete de vida. 2. El sistema permite al jugador recoger el paquete. 3. La salud del jugador se incrementa.
Postcondición	El jugador recupera salud.
Excepciones	Si la salud del jugador está al máximo, no puede recoger más vida.
Comentarios	-

Tabla 2.11 CU-10 Recoger vida

2.2.1.11 CU-11 Recoger armadura

CU-11	Recoger armadura
Descripción	El jugador recoge paquetes de armadura para aumentar su protección.
Precondición	Debe haber un paquete de armadura en el entorno.
Secuencia normal	1. El jugador se acerca al paquete de armadura. 2. El sistema permite recoger el paquete. 3. La armadura del jugador aumenta.
Postcondición	El jugador incrementa su protección.
Excepciones	Si la armadura del jugador está al máximo, no puede recoger más.
Comentarios	-

Tabla 2.12 CU-11 Recoger armadura

2.2.1.12 CU-12 Recibir daño

CU-12	Recibir daño
Descripción	El jugador recibe daño de un enemigo o el entorno.
Precondición	El jugador debe estar expuesto a un ataque o peligro.
Secuencia normal	1. Un enemigo o el entorno inflige daño al jugador. 2. El sistema reduce la salud o armadura del jugador.
Postcondición	El jugador pierde salud o armadura.
Excepciones	Si el jugador tiene invulnerabilidad, no recibe daño.
Comentarios	-

Tabla 2.13 CU-12 Recibir daño

2.2.1.13 CU-13 Infligir daño

CU-13	Infligir daño
Descripción	El jugador inflige daño a un enemigo mediante ataques o armas.
Precondición	El jugador debe tener un arma o ataque disponible.
Secuencia normal	1. El jugador ataca a un enemigo. 2. El sistema verifica el impacto y aplica el daño al enemigo.
Postcondición	La salud del enemigo disminuye.
Excepciones	Si el enemigo es inmune temporalmente, no recibe daño.
Comentarios	-

Tabla 2.14 CU-13 Infligir daño

2.2.1.14 CU-14 Menú de pausa

CU-14	Menú de pausa
Descripción	El jugador accede al menú de pausa para detener el juego y acceder a las opciones.
Precondición	El jugador debe estar en medio de una partida.
Secuencia normal	1. El jugador presiona el botón de pausa. 2. El sistema detiene el juego y muestra el menú de pausa.
Postcondición	El jugador puede acceder a opciones o reanudar la partida.
Excepciones	No se puede pausar durante cambios de nivel.
Comentarios	-

Tabla 2.15 CU-01 Menú de pausa

2.2.1.15 CU-15 Modificar opciones gráficas

CU-15	Modificar opciones gráficas
Descripción	El jugador ajusta las opciones gráficas del juego.
Precondición	El jugador debe estar en el menú de opciones o en el menú de pausa.
Secuencia normal	1. El jugador accede a las opciones gráficas. 2. El sistema permite modificar la calidad gráfica. 3. El jugador guarda los cambios.
Postcondición	Los ajustes gráficos se aplican de inmediato.
Excepciones	Si los cambios no son compatibles con el sistema, se restauran los ajustes predeterminados.
Comentarios	-

Tabla 2.16 CU-15 Modificar opciones gráficas

2.2.1.16 CU-16 Modificar opciones de sonido

CU-16	Modificar opciones de sonido
Descripción	El jugador ajusta las opciones de sonido del juego.
Precondición	El jugador debe estar en el menú de opciones o en el menú de pausa.
Secuencia normal	1. El jugador accede al menú de opciones de sonido. 2. El sistema permite ajustar el volumen del juego, los efectos de sonido y la música. 3. El jugador guarda los cambios.
Postcondición	Los nuevos ajustes de sonido se aplican inmediatamente.
Excepciones	Si los cambios no se aplican correctamente, el sistema restaura los valores predeterminados.
Comentarios	-

Tabla 2.17 CU-16 Modificar opciones de sonido

2.2.2 Diseño de la interfaz

A la hora de diseñar la interfaz nos hemos basado en la interfaz del juego original, aunque actualizándola a las interfaces más modernas. [11]

Ilustración 2.1 HUD original

También nos hemos basado en el menú original del juego. [12]

Ilustración 2.2 Menú DOOM original

Ilustración 2.3 Opciones DOOM Original

3 DESARROLLO

Como hemos visto, ya hemos establecido la estructura del proyecto, a continuación, veremos cómo se ha avanzado en el desarrollo del videojuego de forma iterativa. En cada una de estas iteraciones detallaremos la implementación y evolución de las distintas tareas asignadas, también se podrá ver los diagramas asociados a cada etapa del proceso.

3.1 Primera Iteración

En la primera iteración del proyecto de remake de Doom, el principal objetivo fue desarrollar una escena de pruebas en la que pudieran implementarse las mecánicas básicas del jugador y probar el flujo de la jugabilidad. Aunque esta fase fue introductoria, resultó fundamental para establecer las bases del comportamiento del jugador y las interacciones con el entorno.

3.1.1 Creación de la escena de pruebas

Una vez creado el proyecto en Unreal, lo primero que haremos será crear una escena sencilla con el propósito de probar las mecánicas esenciales del jugador, como el movimiento y la interacción con objetos en un entorno controlado.

El diseño de la escena incluía paredes simples y áreas delimitadas para pruebas de desplazamiento, combate y recogida de objetos.

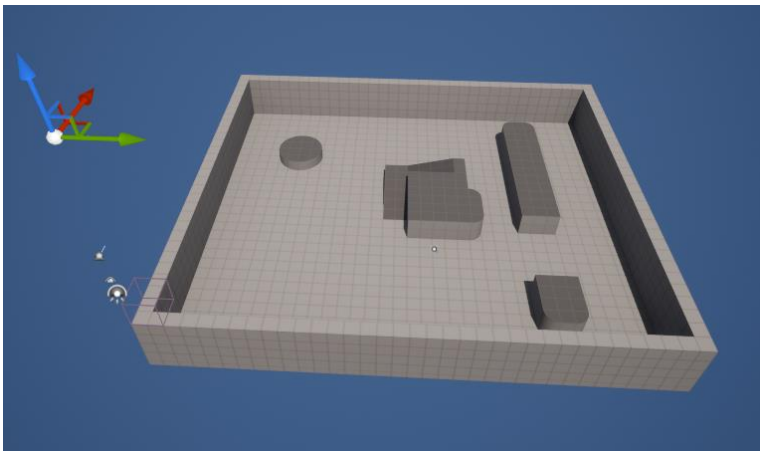

Ilustración 3.1 Escena de prueba

3.1.2 Jugador

El diseño del personaje principal lo hemos obtenido en la web Sketchfab.com, que contiene un gran número de modelos de personajes y objetos en 3D.

Ilustración 3.2 Modelo jugador

A la hora de implementar nuestro personaje al descargarlo de Sketchfab.com no tenía un esqueleto que permitiera una animación del mismo. Por lo tanto, hemos utilizado Blender para añadirle un esqueleto al modelo 3D, para simplificar la animación del personaje se ha utilizado el esqueleto de uno de los personajes que ofrece Unreal Engine por defecto, Manny.

Ilustración 3.3 Modelo por defecto Unreal Engine

Una vez añadido el esqueleto, hemos procedido a importar el modelo a Nuestro proyecto. Dando como resultado el modelo del jugador con las animaciones por defecto de Manny. Una vez obtenido esto hemos comenzado a crear animaciones de forma procedural basándonos en las animaciones que vienen para Manny en el asset StarterPack.

Ilustración 3.4 Jugador ejecutando una animación

Para gestionar estas animaciones se ha modificado el Animator BluePrint del personaje por defecto de Unreal, para adecuar las animaciones a nuestras necesidades.

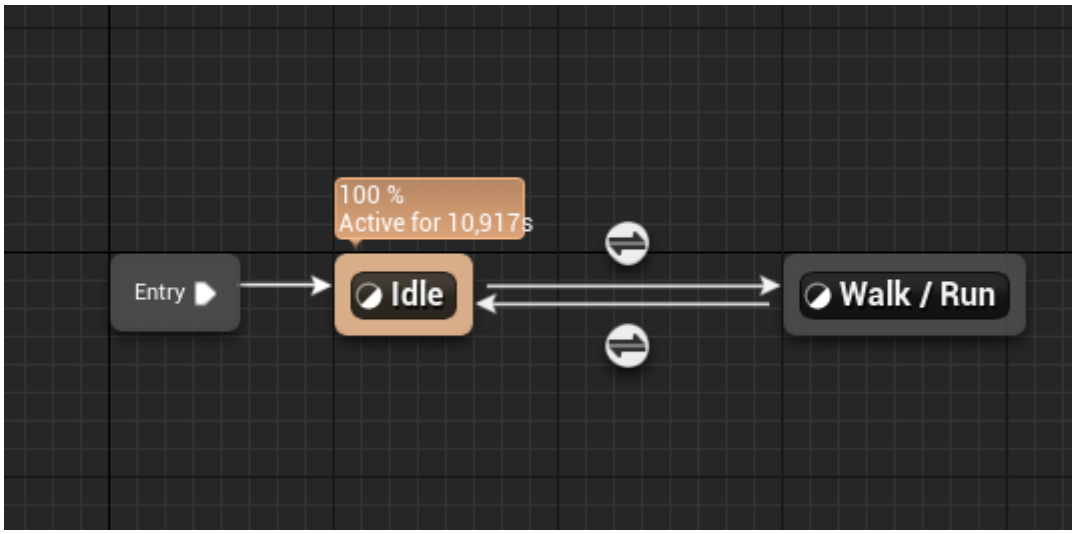

Ilustración 3.5 Estados animación ABP jugador

Hemos creado la clase BP_Player donde se implementará toda la lógica del jugador y sus interacciones. Para la utilización del jugador, esta clase cuenta con un skeletal mesh, un capsule collider y una cámara. El skeletal mesh es el esqueleto del jugador que hemos creado antes en Blender, el capsule component se utiliza para saber donde con que choca el jugador, disparos, pickups, etc, y la cámara es la cámara principal del juego.

Ilustración 3.5 Jugador

3.1.3 Disparo

Para implementar la funcionalidad de disparo, primero se importó un conjunto de armas desde el Marketplace de Unreal Engine. Una vez importadas, se pasó a crear el BP_WeaponBase, que será la clase base de todas nuestras armas, ya que el juego tiene distintos tipos desde armas cuerpo a cuerpo como escopetas o ametralladoras. Una vez creada, se crea un “hijo” de esta clase, en ella se programa el disparo del arma. Nosotros utilizaremos un raycast para el disparo, este raycast nace desde la punta del arma y va hacia donde esté apuntando el jugador. Si el rayo colisiona con otro actor se realiza una comprobación de si es un personaje o es un elemento de la escena y se aplica el daño que tenga definida el arma por disparo.

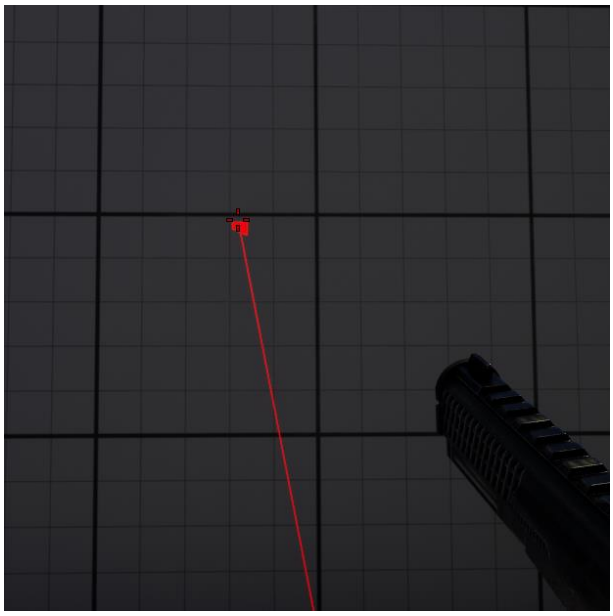

Ilustración 3.6 Raycast arma

Como arma base creamos un hijo de la clase BP_WeaponRayCast e implementamos la pistola.

Ilustración 3.6 Modelo pistola

Una vez creada el blueprints para el arma, pasamos a añadirlas al BP_Player, para ello utilizaremos un socket en el esqueleto del jugador en la mano derecha donde adjuntaremos el arma para que siempre esté en esa posición. Además, para una animación procedural se utilizará otro socket en el esqueleto del arma para que

la mano izquierda del jugador siempre esté posicionada donde queramos y se mueva con las animaciones de disparo del propio arma.

Desde el jugador, cada vez que se realiza el input del botón izquierdo del ratón se realiza un disparo, o si se deja pulsado y el arma es automática se realizan disparos hasta que se suelte.

3.1.4 Enemigo

Para su diseño recurrimos de nuevo al personaje que viene por defecto en Unreal Engine, creamos un nuevo Animator BluePrint(ABP) para los enemigos que sea diferente del ABP del jugador.

Como al jugador, se implementa la lógica para recibir daño utilizando el evento AnyDamage, este evento se activa cada vez que el enemigo recibe algún daño. A su vez se implementa un sistema de salud simple, tiene una vida máxima y una vida actual que se va modificando según el daño que reciba, una vez su vida llegue a 0 se activa el evento de la muerte, este evento para al enemigo, activa sus físicas para que se cree un ragdoll y además destruye su capsule component ya que sino esta nos seguiría persiguiendo por todo el mapa y nos chocaríamos constantemente con ella.

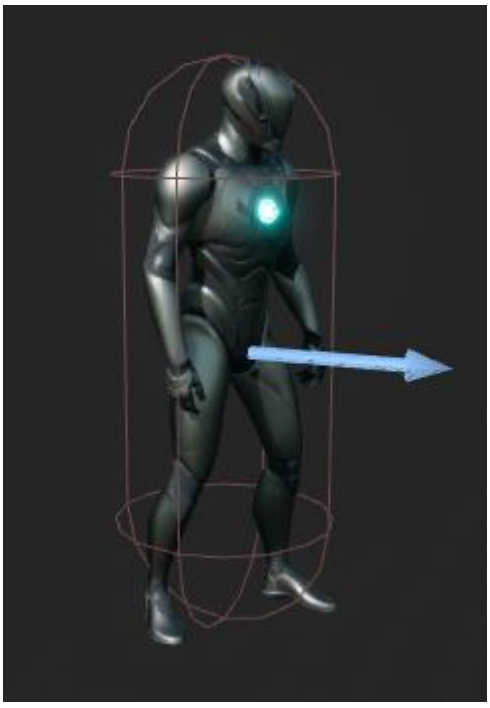

Ilustración 3.7 Enemigo Base

Una vez hecho esto creamos una inteligencia artificial sencilla para el enemigo para que cuando vea al jugador, vaya hasta él y le ataque que es una función que dispara un raycast hacia el jugador y si le impacta aplica daño.

3.1.5 Recarga

Dentro del jugador establecemos distintas variables para llevar la munición de las armas, tanto la del cargador actual, como la de reserva que tiene el jugador.

En el sistema de disparo se añade una función que haga que cada vez que se dispare el arma se reduzca la munición del cargador en 1.

Una vez el jugador presiona la tecla R se inicializa la función que permite la recarga del arma, siguiendo la siguiente lógica. Se calcula el número de balas que faltan en el cargador, una vez comprobado que hay ese número de balas se recarga el arma en su totalidad, pero si en cambio el número de balas en la reserva es inferior, se recargan las balas que queden en la reserva.

Cuando comprobamos que la funcionalidad es correcta, se añaden efectos como sonido y animación para la recarga.

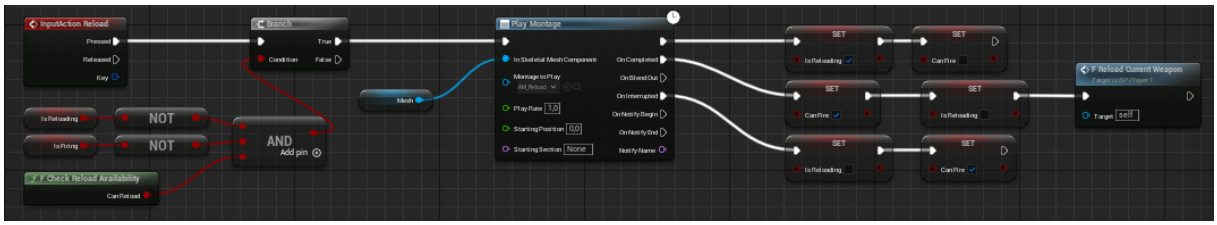

Ilustración 3.8 Evento de recarga

3.1.6 Recogida de elementos

El siguiente paso en nuestro desarrollo es el pickup de elementos, por ejemplo, la munición que será nuestro primer elemento que desarrollaremos.

Para su implementación haremos como con el arma, crearemos una clase base desde la cual crearemos el resto y así tendremos un escalado sencillo del proyecto. En esta clase, hay un sphere collision, es decir, una esfera de colisión, que es invisible mientras se está jugando, y el mesh que querramos mostrar en el juego.

Ilustración 3.8 Pick up base

Cuando el jugador pasa sobre esa esfera de colisión se realiza una comprobación dependiendo del tipo de elemento que se esté recogiendo. Si es munición, se comprueba que el jugador no tiene ya la munición máxima para ese tipo de arma y si es así se añade la cantidad de munición establecida en la clase. Es exactamente igual tanto para la salud como para la armadura, se realizan las comprobaciones necesarias y se suma el valor establecido al campo correspondiente.

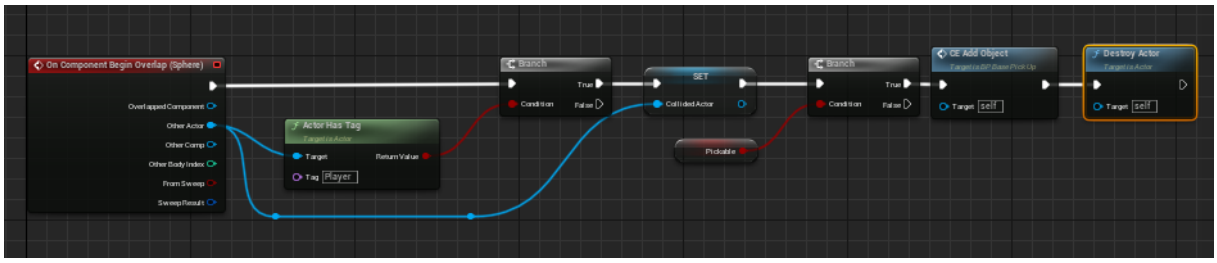

Ilustración 3.9 Lógica pick up

El objetivo principal de esta primera iteración fue desarrollar un entorno donde se pudieran probar y refinar las interacciones básicas del jugador con el entorno y los elementos. Este sprint, resultó ser crucial para sentar las bases del proyecto, ya que permitía un enfoque iterativo en futuras mecánicas más avanzadas y la ampliación de escenarios más complejos.

Esta primera fase nos proporcionó una plataforma sólida para avanzar en las siguientes iteraciones, donde se implementarían aspectos más detallados del combate y la interacción con enemigos y escenarios.

3.2 Segunda Iteración

Una vez hemos completado la primera iteración, ya tenemos una base para comenzar a diseñar el primer nivel e ir implementando nuevas armas, interacciones con objetos del nivel como las puertas o las plataformas además de el modelado de los distintos enemigos y desarrollar su inteligencia artificial para que realicen tareas más complejas.

3.2.1 Añadimos armas

Partiendo de la clase de BP_WeaponRaycast que creamos en la iteración anterior vamos a crear el resto de armas, para ello creamos hijos de la clase mencionada anteriormente. Dentro de estos hijos podemos establecer distintos parámetros que harán que el funcionamiento de cada arma sea distinto.

- **Escopeta:** Para el funcionamiento de una escopeta hemos modificado la función de disparo dentro de la clase base. En el disparo original se lanza un raycast solo, en cambio si el parámetro de escopeta está activo ya no se lanza un solo raycast sino que se lanzan 8 a los que se les modifica un poco la trayectoria mediante un parámetro llamado spread.

Ilustración 3.10 Modelo escopeta

- **Chaingun:** La chaingun es una ametralladora pesada, que dispara de forma automática, es decir, no dispara 1 vez por cada click en el ratón, sino que dispara sin parar hasta que se deja de presionar el click.

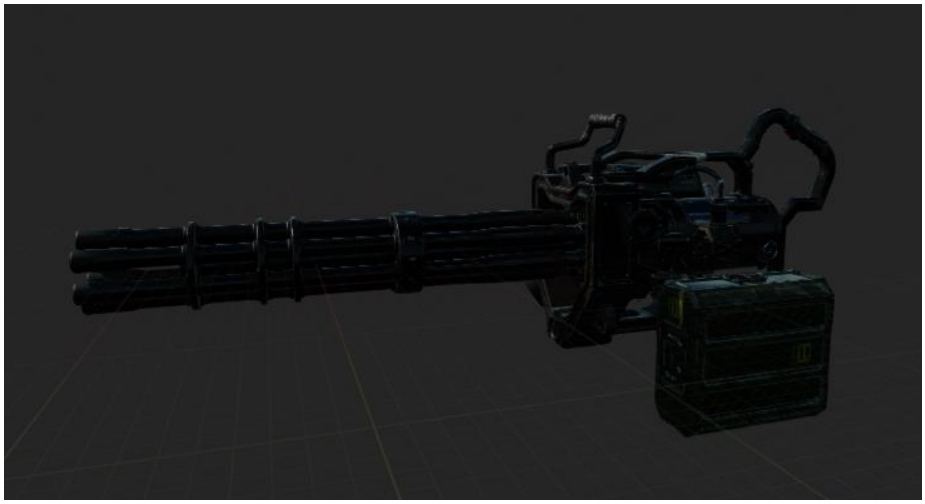

Ilustración 3.11 Modelo chaingun

- **Puño:** El jugador no lleva ningún arma y pega puñetazos. Para conseguir esto lo que hemos hecho es utilizar la misma función que la de un disparo de la pistola, pero esta arma no tiene ningún mesh y se le ha modificado el rango al que llega el raycast para que sea mínimo.

Ilustración 3.13 Imagen puño

- **Motosierra:** Posiblemente sea el arma más icónica de este videojuego. Para implementarla lo que se ha realizado es lo mismo que con el puño, pero añadiéndole la funcionalidad del disparo automático. Al revés que con

el puño esta arma si consta de efectos y animaciones que son agregados a la función de disparo.

Ilustración 3.13 Modelo motosiella

Una vez implementadas todas las armas hemos dado paso a modificar las animaciones del jugador para que se muestre en pantalla una animación lo más realista posible. Para ello se han modificado en el ABP las animaciones de estado pasivo del jugador y de movimiento.

Además, como ya hay bastantes armas se ha creado una función que realiza el cambio entre las distintas armas disponibles por el jugador.

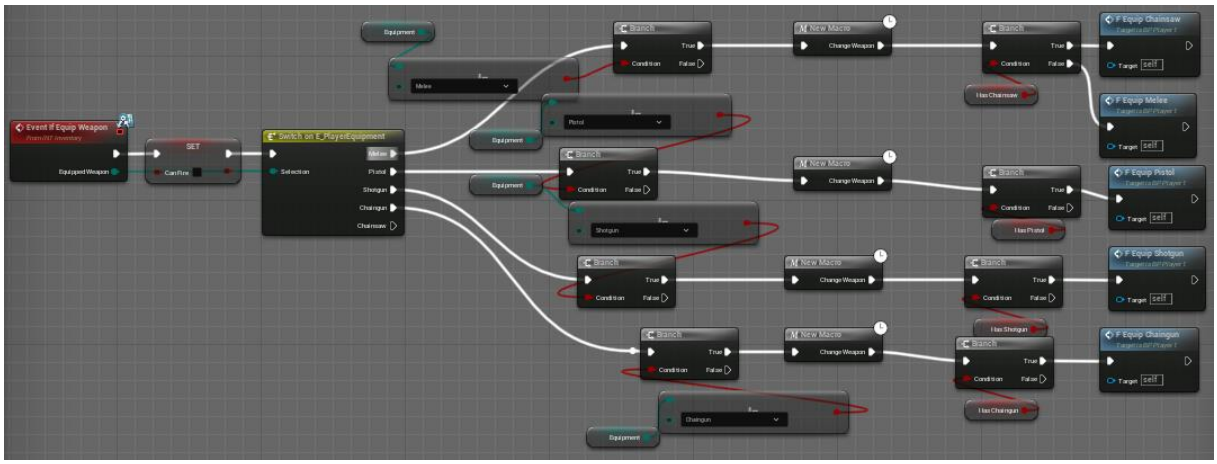

Ilustración 3.14 Evento cambio de arma

3.2.2 Añadimos más enemigos

Comenzamos esta iteración añadiendo los enemigos que habrá en el juego que son tres:

- **Soldado:** El soldado es el enemigo más fácil del juego tiene poca vida y su daño es bajo.

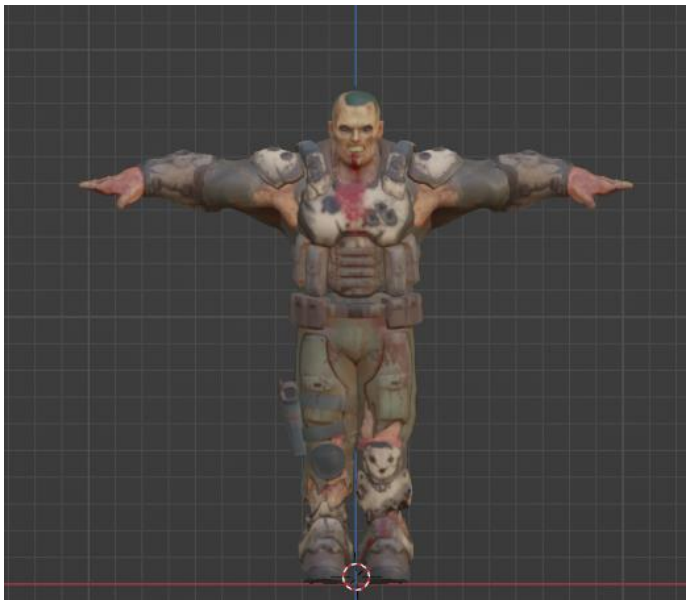

Ilustración 3.15 Modelo soldado

- **Sargento:** Tiene más vida que el soldado y además su arma es una escopeta que hace bastante daño.

Ilustración 3.16 Modelo sargento

- **IMP:** Es el enemigo con más vida que implementaremos y su ataque es una bola de fuego que realiza un gran daño al impactar.

Ilustración 3.17 Modelo IMP

Para ello recurrimos a Sketchfab.com para descargar los modelos 3D de los enemigos Sargento e IMP, una vez descargados utilizaremos Blender para modificar el modelo del Sargento y así crear al soldado, ya que no se ha encontrado un modelo similar de nuestro gusto. Una vez creados los modelos, se realiza el mismo proceso que con el jugador, se crea un esqueleto basándonos en el esqueleto de unreal engine y este se importa junto con sus animaciones que se han obtenido de Mixamo.com a Unreal Engine.

A la hora de implementarlos modificaremos el enemigo base para añadirle un AIController lo que será su cerebro a la hora de la toma de decisiones. Estas decisiones están representadas en un Behavior Tree, en él se representan las posibles decisiones que puede tomar el enemigo para realizar distintas acciones, para ello es necesario la implementación de un Blackboard, que es donde se almacenan las variables compartidas entre el AIController y el Behavior Tree.

Creamos el Behavior Tree y se establecen los distintos estados que puede tener el enemigo:

- **Pasivo:** El enemigo busca un si hay un elemento del tipo BP_Patrol, que es una línea de puntos que tiene que seguir el enemigo para realizar una patrulla en una zona en concreto.

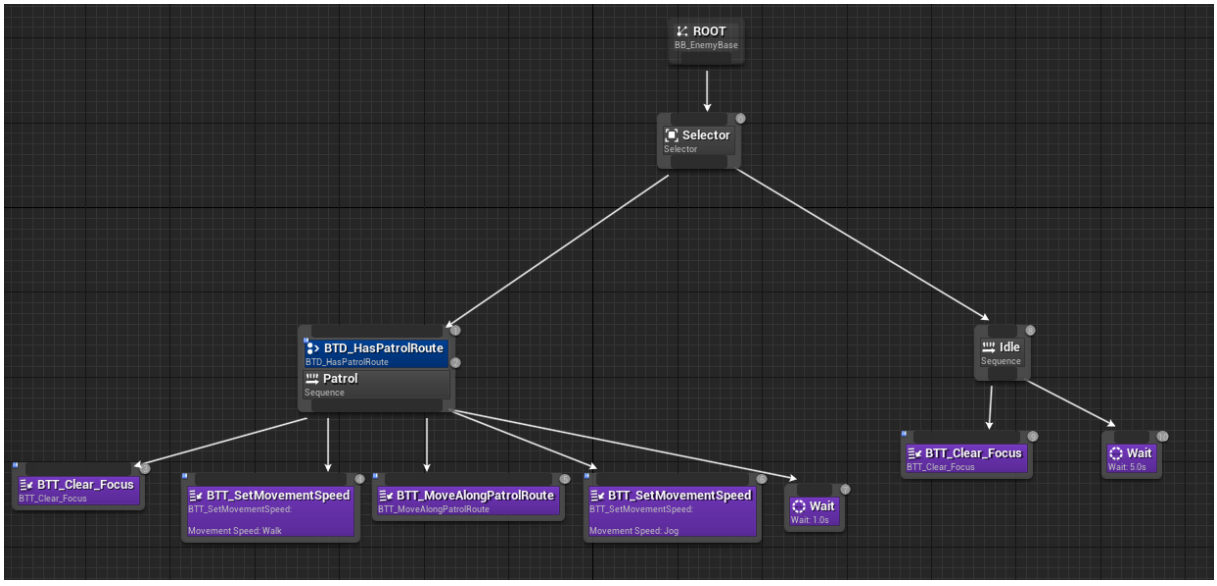

Ilustración 3.18 Subárbol de comportamiento pasivo

- **Ataque:** En el momento en el que el enemigo recibe un estimulo ya sea visual o por daño del jugador este sale corriendo hacia el hasta situarse dentro de su rango de ataque y comienza a atacar al jugador.

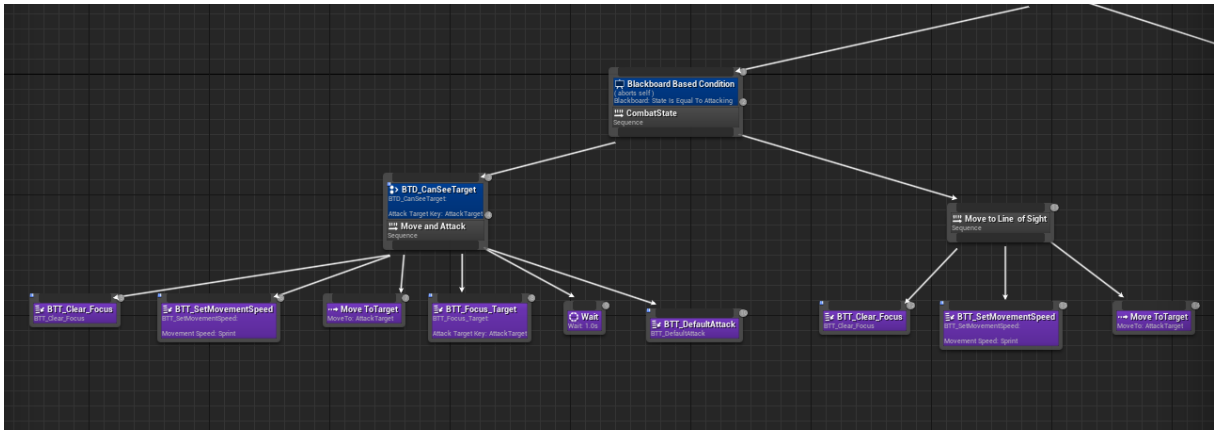

Ilustración 3.19 Subárbol de comportamiento de ataque

3.2.3 Interacciones con objetos

Dentro de los niveles hay distintos objetos que son interactuables, como son las puertas, los botones o palancas y las plataformas.

Creamos como para el resto de clases una clase base para las puertas y las plataformas BP_DoorBase, en esta clase se utiliza un mesh que será la puerta y 2 Box Collider, uno que será el que comprobará si el jugador se encuentra dentro de ella para poder interactuar con la puerta y el otro box collider se utiliza para comprobar que no hay ningún actor debajo de la puerta y así no se cierre sobre el jugador o un enemigo.

Para realizar la apertura de puerta se utiliza un timeline para el movimiento de la puerta que se realiza en el eje Z y tarda un segundo, así conseguimos el efecto de movimiento de apertura y cierre, además de que se utilizan efectos sonoros durante el movimiento.

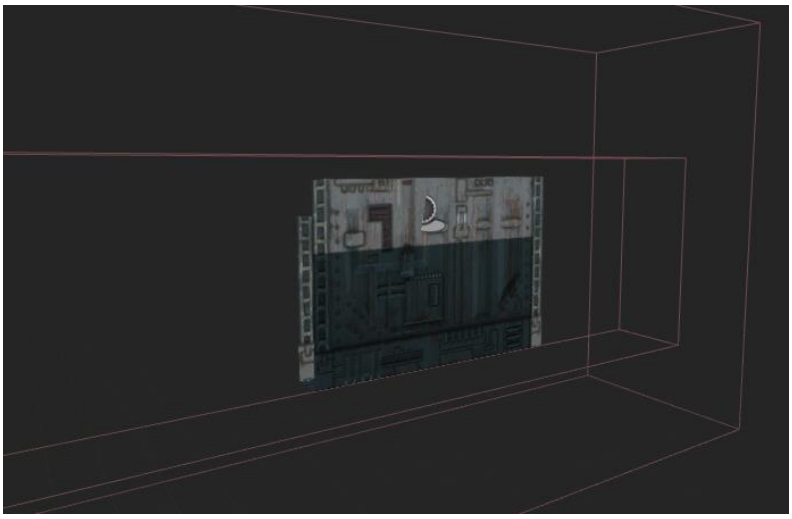

Ilustración 3.20 Modelo puerta

A la hora de crear las plataformas observamos que la lógica es igual que en la puerta, pero sólo es necesario el collider para la interacción por lo tanto reutilizaremos la lógica y las funciones.

Y para la implementación de los botones lo que hemos hecho es crear la clase BP_Button que consta de un plano y un box collider, para crear el efecto de pulsación en el botón lo que se procede a hacer es un cambio de material del plano. Es decir, el plano original tiene el material con el botón desactivado y cuando el jugador está dentro del box collider e interactua con el botón este cambia su material por otro en el que si está activo y además se reproduce un sonido.

Ilustración 3.21 Modelo botón

3.2.4 Diseño e implementación primer nivel

Realizamos el diseño del primer nivel del juego, este nivel es una réplica exacta del primer nivel del juego original, pero se han modificado los materiales para que sean materiales PBR y dar un aspecto mucho más realista que el del mapa original, para esto hemos usado Blender, usando una imgen del mapa del juego original se ha creado una réplica mediante el uso de planos y a estos se les ha añadido y ajustado su material dentro del nivel. Separamos las puertas y plataformas que forman parte del nivel para que se importen de manera individual.

(

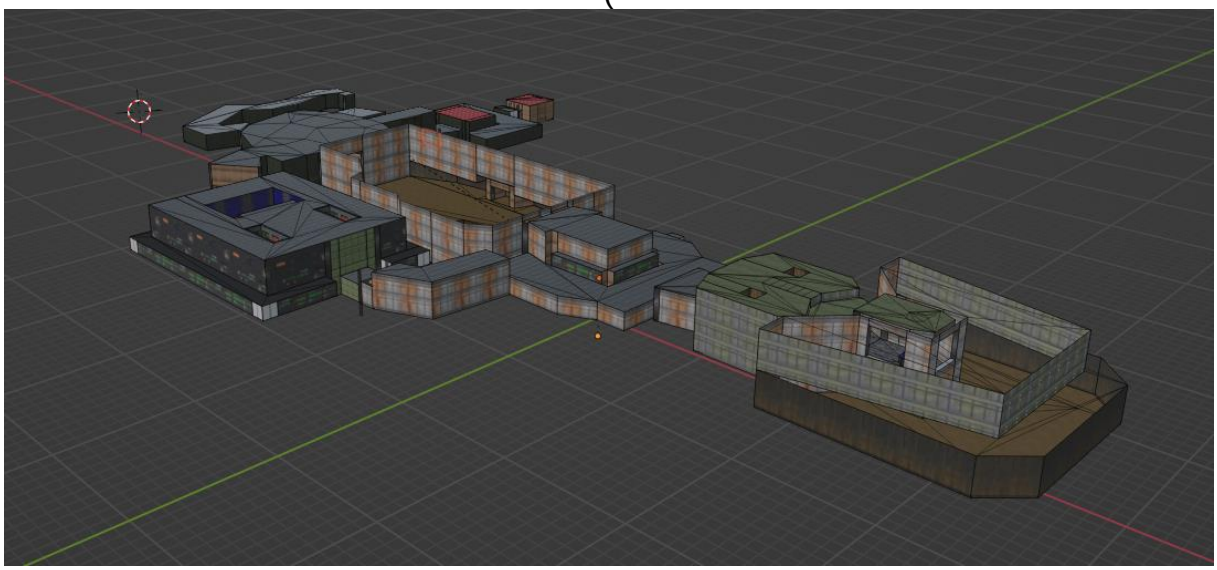

Ilustración 3.22 Primer nivel

Para su implementación importamos el nivel al proyecto, y vemos si se han desajustado algún material y lo corregimos. Una vez hecho esto, creamos un nuevo nivel dentro de unreal, y creamos un sky box para la luz ambiental que habrá en el nivel. Ajustamos los valores del sky box hasta nuestro gusto dentro del nivel, y comenzamos a colocar iluminaciones dentro del nivel.

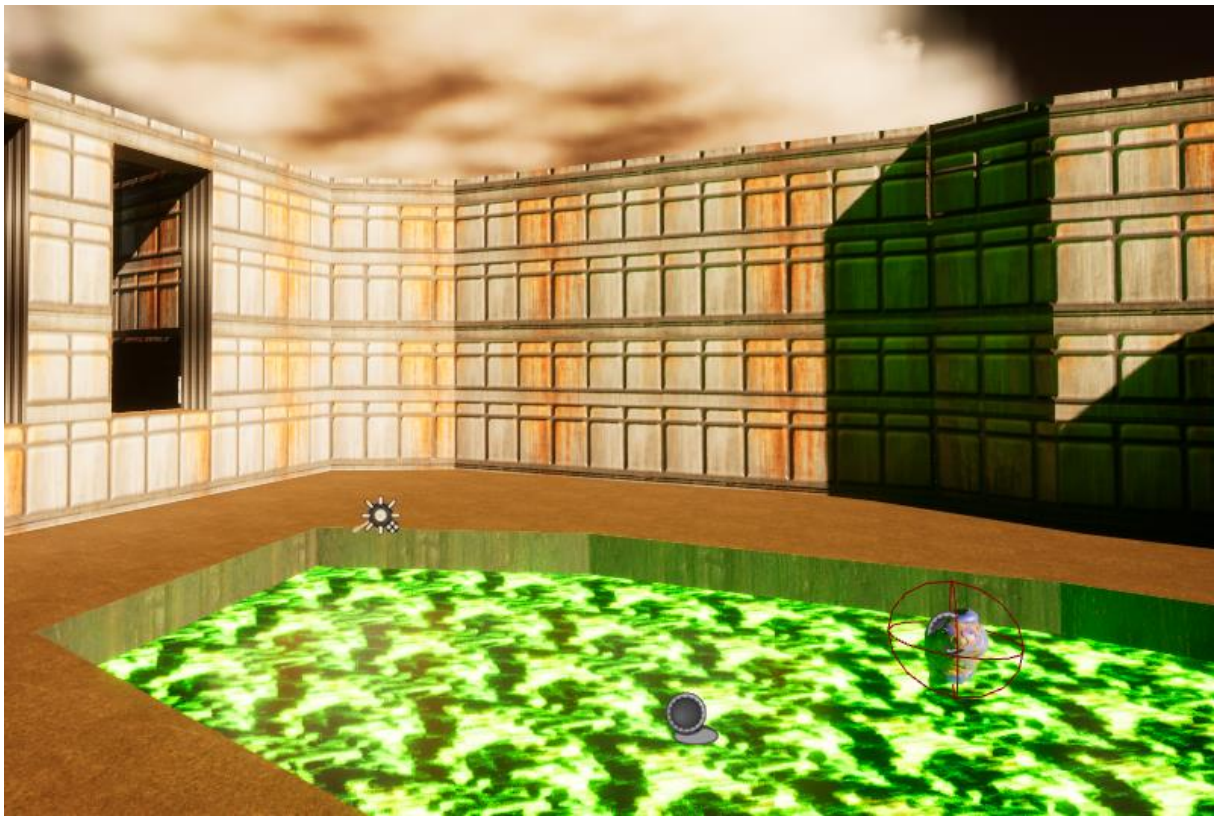

Ilustración 3.23 Nivel importado en Unreal Engine

Una vez colocadas toda la iluminación dentro del nivel comenzamos a añadir assets que son de decoración del nivel. Como son las torretas electrónicas, el poste de luz y los candelabros.

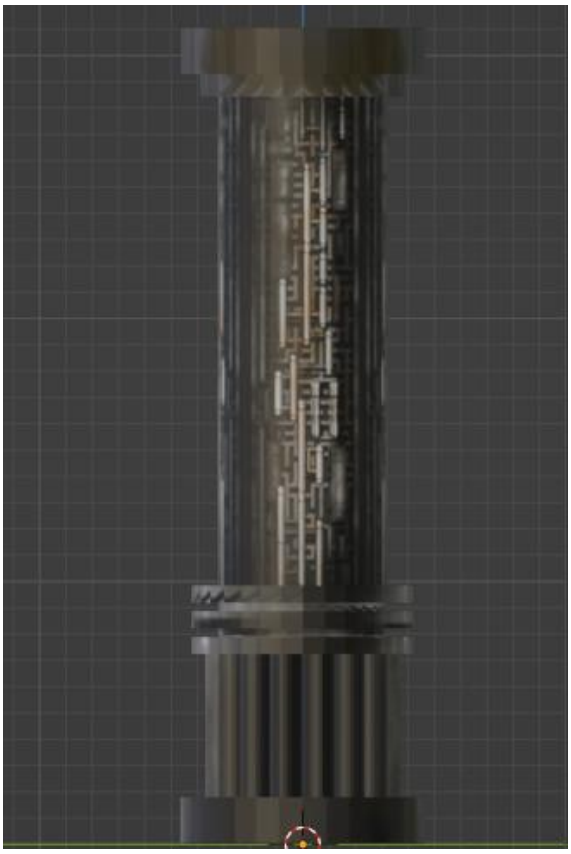

Ilustración 3.24 Modelo torre

Ilustración 3.25 Modelo candelabro

Ilustración 3.26 Modelo poste de luz

Pues una vez terminado esto procedemos a añadir el punto de partida del jugador y comprobamos que está todo correcto.

3.3 Tercera Iteración

Una vez tenemos la jugabilidad prácticamente realizada comenzaremos a implementar los distintos detalles que hay en el primer nivel, como son los distintos elementos pick up, los barriles explosivos, el HUD además del menú principal, el menú de pausa.

3.3.1 Añadimos pick ups

Como hemos explicado anteriormente dentro del juego existen varios elementos que son del estilo pickup como pueden ser los bonus de armadura, los bonus de salud, las municiones, las propias armas y llaves.

Creamos una clase base para la creación de pickups dentro del juego y comenzamos a implementar en ella las funcionalidades comunes de los pickups que

son el mesh, el sphere collider y las funciones que comprueban si el jugador a colisionado con el pickup.

Una vez realizado esto comenzamos a implementar las distintas subclases que habrá por cada uno de los pickups.

Comenzamos implementando los pickups de munición, esta clase recibe un enumerador que será el que determine que tipo de munición es la que dará ese asset en el nivel, además este enumerador determina el mesh que se verá y sólo hay que añadir la variable de cuanta munición va a dar.

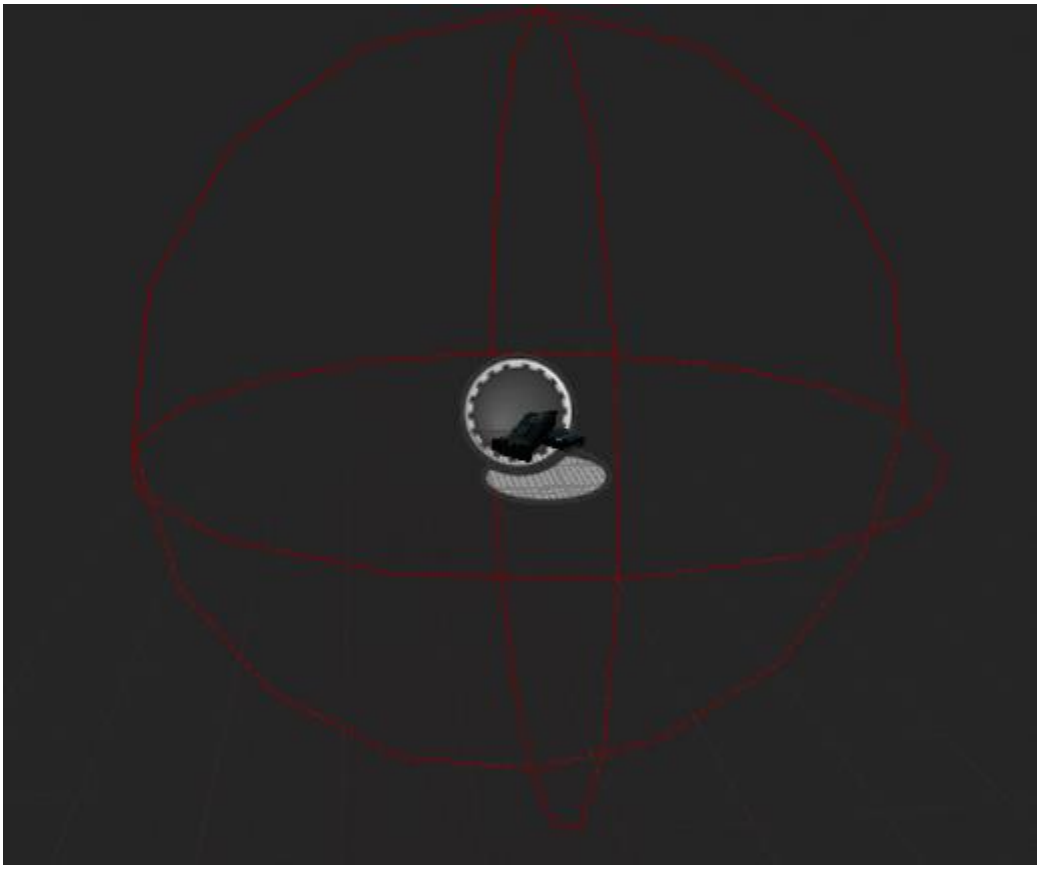

Ilustración 3.27 Pick up munición

Continuamos creando los pick ups de salud, estos sumaran una cantidad determinada por el tipo de cura que se haya establecido por defecto, existen 2 tipos de curas:

- **Normales** que solo curan hasta un máximo establecido que es 100.
    - **Medikit**: cura 25 de vida.

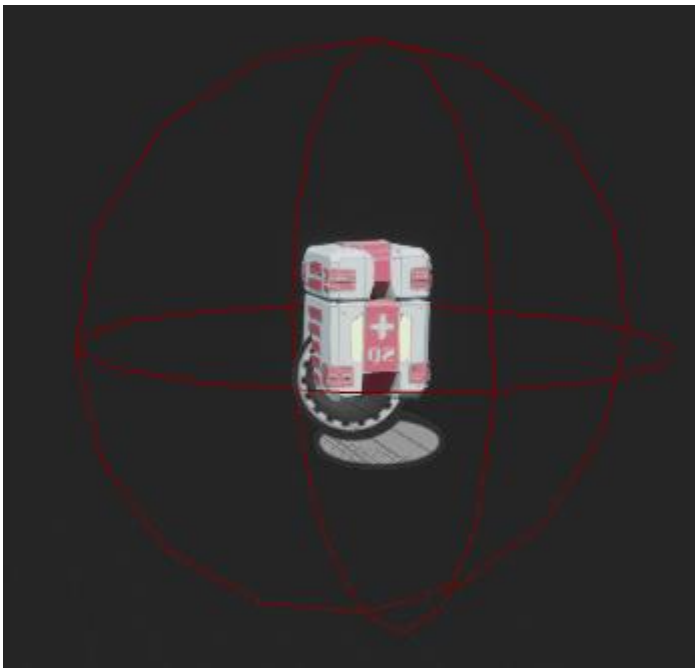

Ilustración 3.28 Pick up medikit

- **Stimpack:** cura 10 de vida.

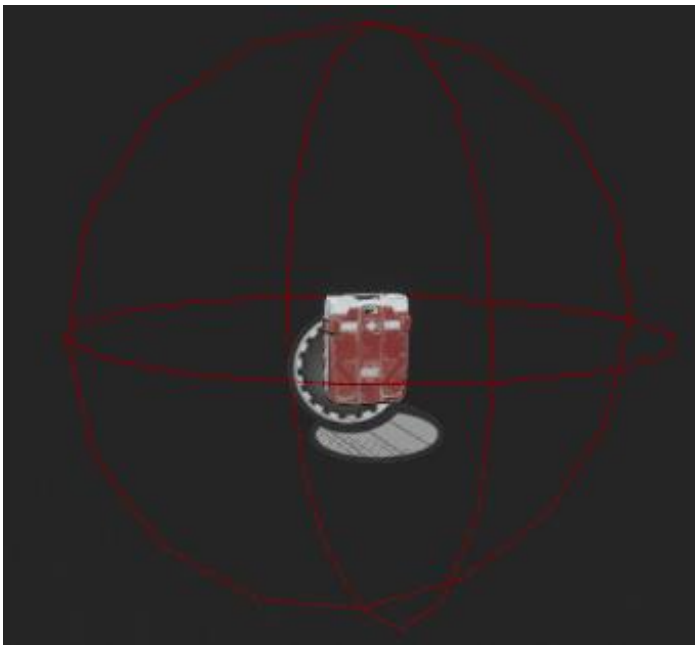

Ilustración 3.29 Pick up stimpack

- Bonus de salud: estos permiten que el jugador alcance hasta un 200 de vida máxima

- **Soul Sphere:** Nos da un bonus de salud de 100 y otro 100 de armadura.

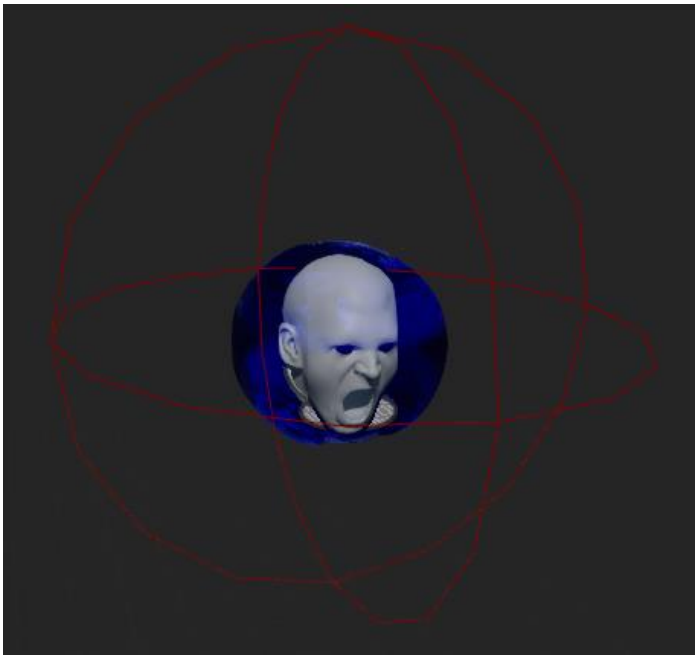

Ilustración 3.30 Pick up soul sphere

- **Poción:** Nos da un bonus de salud de 1.

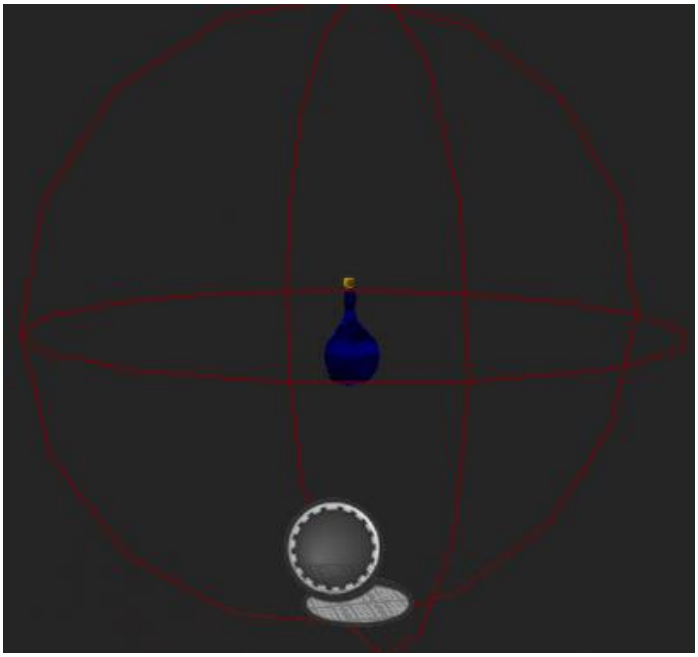

Ilustración 3.31 Pick up poción

También hemos creado los pick up de armadura, su implementación es igual que los de salud, sólo cambian los modelos y que añaden armadura no salud. Existen 2 tipos de armadura:

- **Normal:** Solo llega hasta 100 de armadura.
    - **Armadura verde:** Nos da 100 de armadura máxima.

Ilustración 3.32 Pick up armadura verde

- **Bonus de armadura:** Llega hasta los 200 de armadura.
    - **Armadura azul:** Da 200 de armadura.

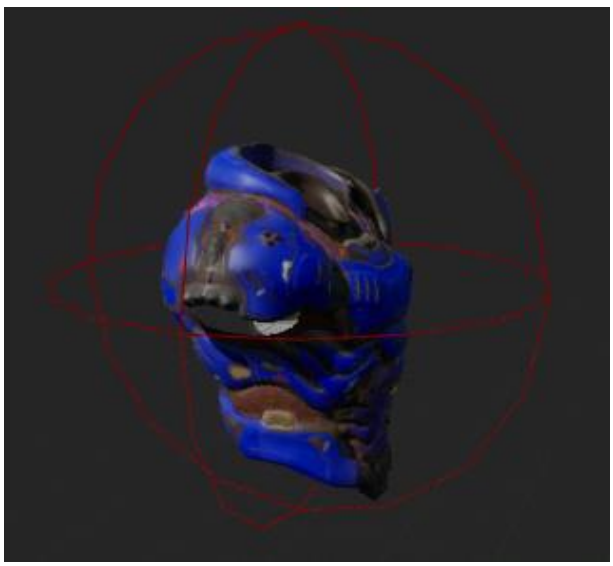

Ilustración 3.33 Pick up armadura azul

- **Casco:** Nos da un bonus de armadura de 1 hasta un máximo de 200.

Ilustración 3.34 Pick up casco

Continuamos creando hijos de la clase base y creamos el pick up de la mochila, este pickup nos permite aumentar la capacidad de munición máxima al doble y además nos da una pequeña cantidad de munición para todas las armas.

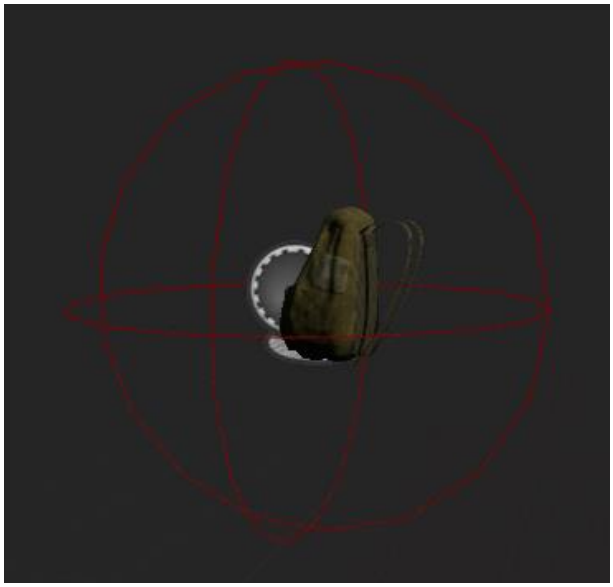

Ilustración 3.35 Pick up mochila

Y, por último, creamos el pickup de la llave la cual cuando es recogida activa un flag en el jugador para que así pueda interactuar con puertas que requieran de esta.

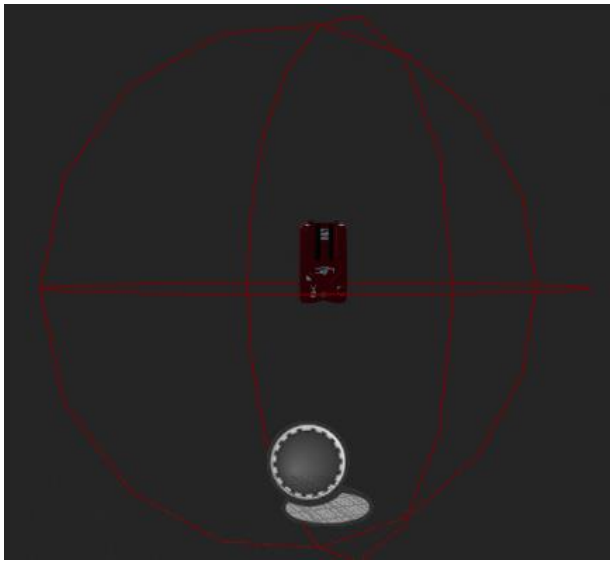

Ilustración 3.36 Pick up llave

3.3.2 Barril explosivo

Para su diseño acudimos a Sketchfab.com y extraemos un modelo de un barril que nos guste y lo importamos a Unreal. Dentro de Unreal fracturamos ese modelo y para que cuando explote el barril de la sensación de desintegrarse.

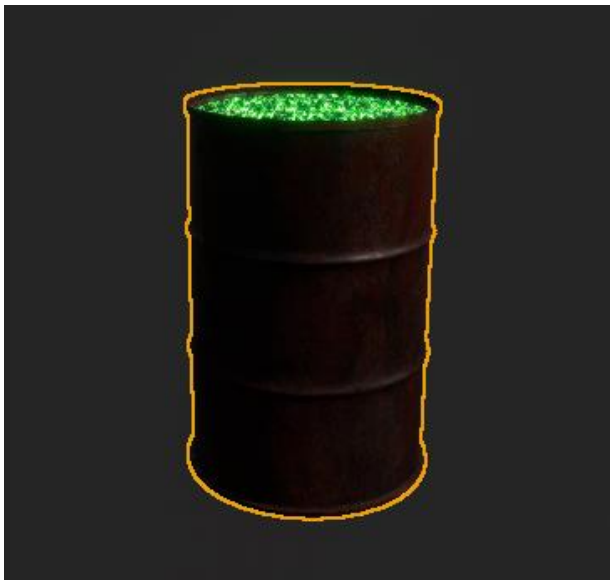

Ilustración 3.37 Barril explosivo

Para su implementación usamos el modelo que acabamos de fracturar y un conjunto de partículas que nos da un efecto de ebullición del barril. Para su explosión, cuando el barril reciba algún daño al llegar su vida a 0 explotará aplicando una fuerza radial para que el barril salga volando en distintos pedazos.

Imagen implementación

3.3.3 HUD

Creamos un widget para la clase Player llamado PlayerUI. La interfaz es una interfaz 2D que siempre está en pantalla mientras estamos jugando, basándonos en el HUD original, pero lo actualizamos quitando el material de la parte inferior, por lo demás se mantiene la cara que nos da un feedback de cuánto daño ha recibido el jugador, esta se sitúa en la parte izquierda justo encima de la caja donde se muestra las barras de vida y de armadura. En la parte derecha se muestra la información sobre el arma, nombre y munición.

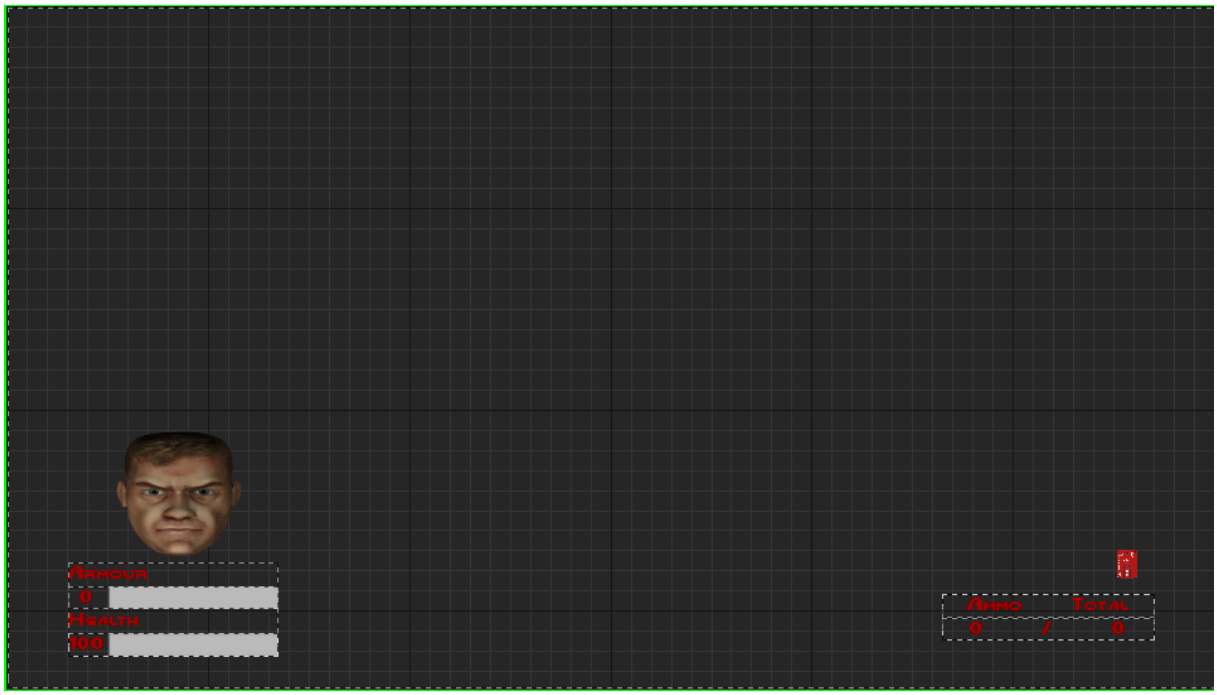

Ilustración 3.38 Widget playerUI

Además, en la esquina superior derecha se ha implementado un cuadro de texto en el que se muestra información sobre lo que va ocurriendo en el nivel, que pick up has recogido, etc.

3.3.4 Menús

Comenzamos a implementar el menú de inicio y de pausa, para el menú de inicio creamos un nuevo nivel ya que queremos que el menú sea a tiempo real.

Para el menú principal se ha importado una calavera 3D a la cual se le han añadido unas llamas de forma procedural dentro del propio nivel y esta representa el selector dentro del menú.

Ilustración 3.39 Menú principal

Además, se implementa las funcionalidades para las distintas opciones:

- **Nueva partida:** Se carga el primer nivel del juego con los parámetros del jugador por defecto y comienza la partida.
- **Cargar partida:** Se carga el último nivel en el que se haya realizado un guardado con los parámetros que tenía el jugador en ese momento.
- **Opciones:** Si es pulsada, nos abre un nuevo widget que nos permite ver las distintas opciones tanto gráficas, sonido, sensibilidad del ratón. Si se pulsa la opción de sonido nos llevará a un nuevo widget donde hay 3 sliders que controlan el nivel de volumen general, de la música y de los efectos.

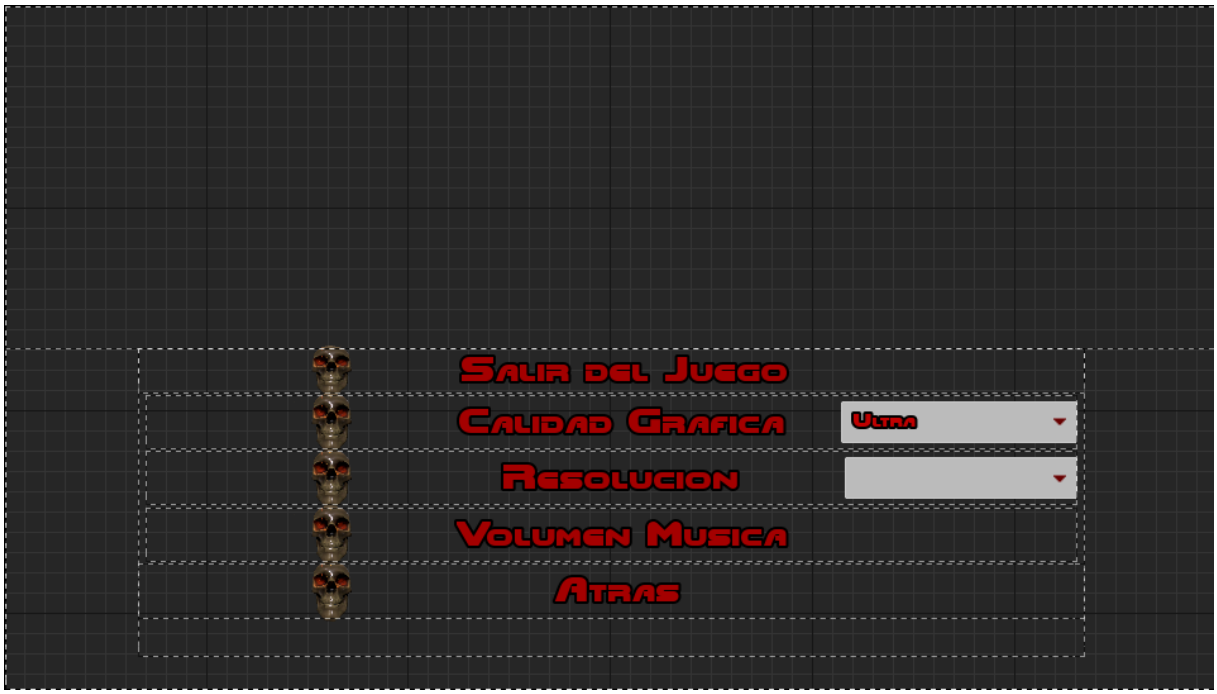

Ilustración 3.40 Menú de opciones del menú principal

- **Salir:** Se ha implementado una función en la que si se presiona se cierra el juego.

Para el menú de pausa se utiliza la misma calavera que en el menú principal y se le añade un par de botones nuevos:

- **Guardar Partida:** Se realiza un guardado en el punto donde se encuentra el jugador.
- **Atrás:** Nos devuelve a la partida.

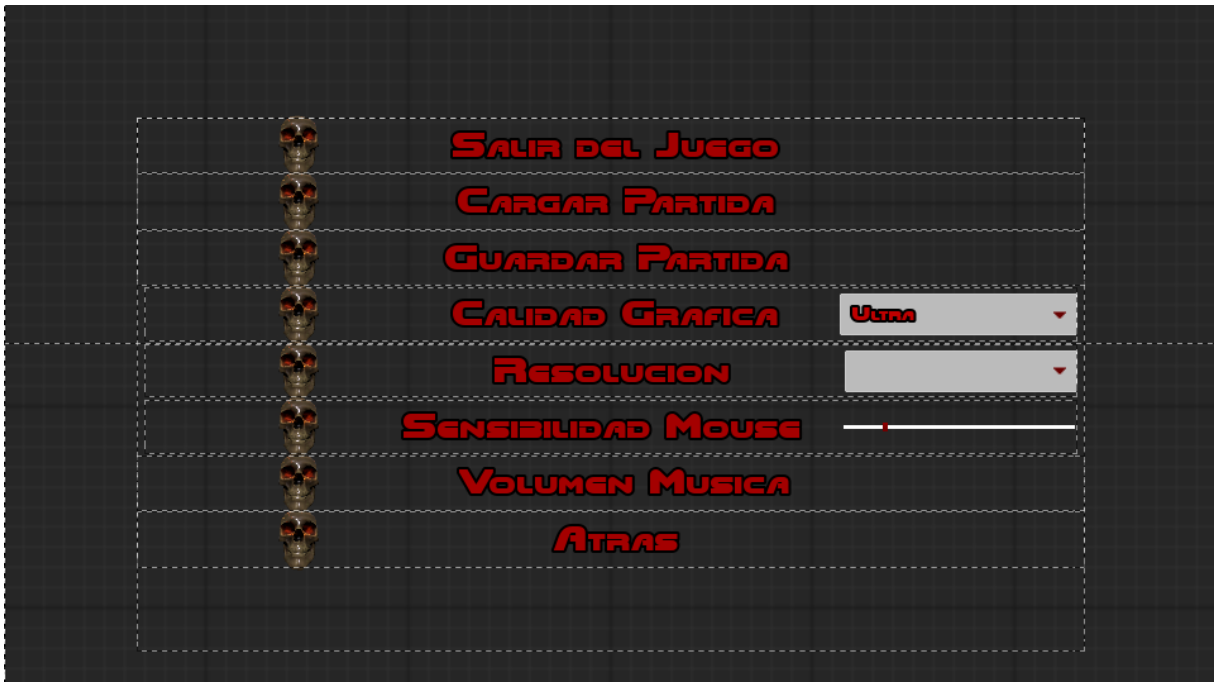

Ilustración 3.41 Menú de pausa

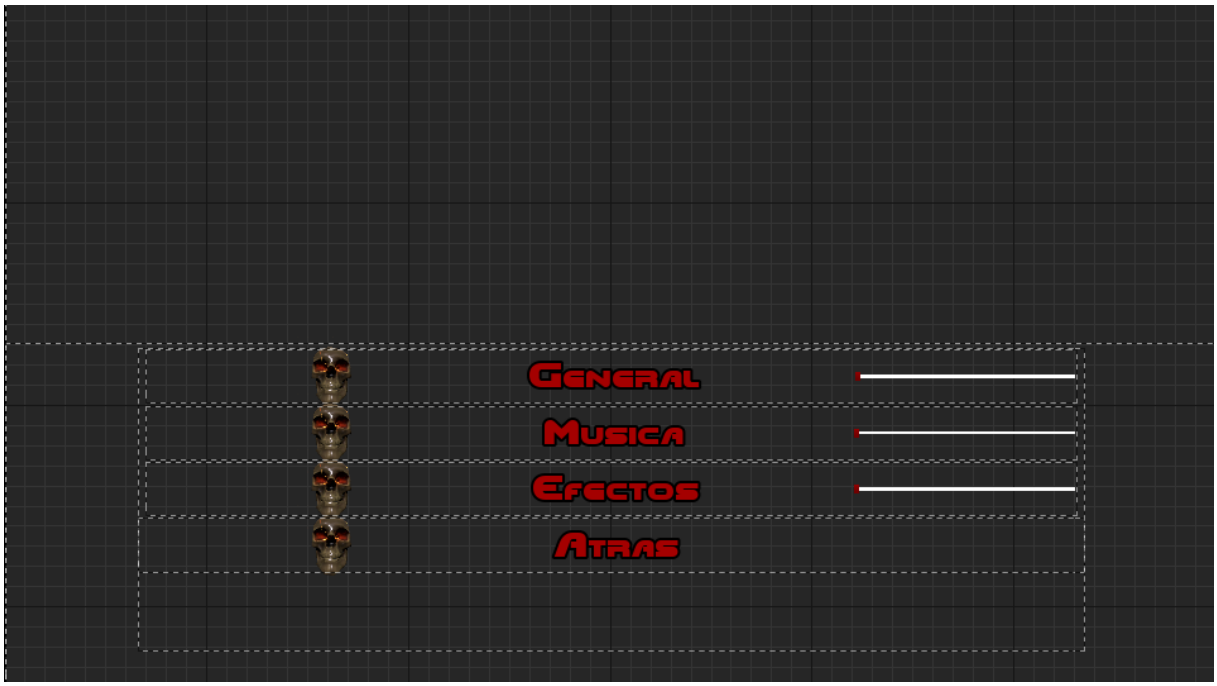

Ilustración 3.42 Menú opciones de sonido

3.4 Cuarta Iteración

Esta es la última iteración del proyecto, implementaremos el sistema de carga y de guardado, el de perdida de la partida y el de victoria de la partida, sistema de sonido.

3.4.1 Sistema de carga y guardado

Comenzamos la implementación del sistema de guardado creando una clase de tipo SaveGame que nos brinda Unreal, en la cual crearemos todas las variables que queramos guardar, como son la munición, la vida, la armadura, el nombre del nivel en el que nos encontramos, el volumen de la música o de los efectos, etc.

Ahora somos capaces de guardar o cargar una partida desde cualquier punto gracias a la creación de los métodos:

- **SaveGame:** este método crea un objeto del tipo SaveGame al que se le asignan todas las variables que hemos mencionado antes y se guardan en un fichero.
- **LoadGame:** este método comprueba si existe un fichero de guardado y si es así, lo carga asignando todas las variables guardadas en él.

Para poder mantener el nivel de volumen o de sensibilidad del ratón ajustada a lo que ha designado el jugador, cada vez que se realiza una modificación de las mismas estas se guardan o cada vez que se inicia un nuevo nivel se guarda la partida.

3.4.2 Pantalla perdida y victoria de partida

Creamos un widget que se mostrará tanto si el jugador ha llegado al final del juego como si muere, lo único que cambia es el texto que se muestra sobre un fondo difuminado en el que se mostrará “Has perdido” en el caso de morir o “Has ganado” en el caso de llegar al final del juego. También se muestran 3 botones, uno de reinicio, el cual reinicia el juego por completo, otro de cargar partida y por último el de salir que cortará la ejecución del juego.

Ilustración3.43 Pantalla victoria

3.4.3 Sistema de sonido

Hemos añadido sonido a todos los niveles y a todas las interacciones que se realizan en el videojuego, incluyendo sonido en los menús.

Los sonidos se dividen en 2 tipos, música que es la música que suena de fondo durante el juego y efectos estos son los sonidos que suenan cuando se realizan las interacciones. El volumen de estos será definido por el jugador en el menú de opciones de sonido que hay en el menú principal y en el de pausa.

3.5 Pruebas finales

La realización de pruebas finales es de vital importancia en el desarrollo de un software. En Unreal Engine, esta etapa se puede realizar en cualquier momento del desarrollo ya que nos permite le ejecución del juego, para así poder evaluar, ajustar las nuevas implementaciones, así las pruebas se han ido realizando a la misma vez que el proyecto:

- **Mecánicas de movilidad:** Hemos comprobado que el jugador se mueve de forma fluida, el resultado de la prueba ha sido OK.

- **Mecánica de apuntado:** Prueba en la que se revisa si el funcionamiento del apuntado es correcto, el resultado de la prueba ha sido OK.
- **Mecánica de disparo:** En esta prueba se ha comprobado la acción de disparo, durante esta prueba comprobamos que el disparo no procedía de la punta del arma sino del centro del jugador, se ha corregido este error cambiando el punto de partida del raycast a la punta del arma.
- **Mecánica de recarga:** Durante esta prueba se ha detectado que el funcionamiento de la recarga no era correcto, ya que se recargaba la totalidad de la munición de reserva, corregimos este error añadiendo una nueva variable que controla la munición actual y se cambia la función para añadir esta variable.
- **Mecánica de cambio de arma:** Hemos comprobado que el cambio de arma no se realiza correctamente ya que al cambiar de arma se pierde la munición que tiene el arma, esto se corrige controlando la munición de cada arma de forma individual, creando variables de munición actual y de reserva por arma, además se ha detectado que la animación ocurre antes de que se realice el cambio del arma, se corrige cambiando el orden en el que se llama a la animación dentro de la función.
- **Mecánica de pick up salud:** Se ha comprobado que el pick up de salud es correcto, aunque tiene un bug en el que permite que se sume la salud por encima del máximo, se modifica la función realizando nuevas comprobaciones dependiendo del tipo de pick up de salud para que no ocurra este bug.
- **Mecánica de pick up munición:** Prueba en la que se revisa el funcionamiento del pick up de munición. El resultado de esta prueba es OK.
- **Mecánica de pick up de armadura:** Durante esta prueba se ha revisado el funcionamiento de esta mecánica. El resultado de la prueba es OK.
- **Mecánica de pick up de arma:** Durante esta prueba se ha comprobado el pick up de las distintas armas que hay en el proyecto. Hemos detectado un bug en el que el arma que se recoge no tiene munición, se modifica la

función que añade el arma al jugador para que sume la munición del arma a la munición de reserva del arma que tiene el jugador.

- **Disparo motosierra y puño:** Durante la prueba del pick up de arma se ha detectado que cuando se dispara la motosierra o el puño aparece un error dentro de Unreal Engine, este error ocurría porque no se estaba controlando lo que ocurría si el raycast no colisionaba con ningún objeto del nivel, se ha corregido comprobando si golpea o no y si no lo hace se termina la ejecución del disparo.
- **Animaciones de arma:** En esta prueba se ha visto si las animaciones propias que tienen las armas para dar un efecto más realista al juego. El resultado de esta prueba ha sido OK.
- **Actualización del HUD:** Hemos comprobado que el HUD se modifica a tiempo real con las acciones que se realizan en el juego. El resultado ha sido satisfactorio.
- **Pruebas de la IA:** Se ha comprobado si el funcionamiento de la IA era correcto y se han detectado algunos fallos. Cuando el jugador daña a los enemigos estos no reaccionan, esto se ha corregido modificando la función de disparo ya que en Unreal Engine para que la IA detecte daño no sirve la función Apply Damage, hay que llamar a la función Report Damage, se ha cambiado esta función y ya reacciona correctamente. También se ha detectado que la IA reacciona ante cualquier actor que ve, por lo tanto, se ha añadido la comprobación de que para que reaccione el actor que ve tiene que ser del tipo jugador.
- **Funcionalidad del menú:** Durante esta prueba hemos comprobado el funcionamiento del menú principal ha sido correcto.
- **Funcionalidad del menú de pausa:** Hemos comprobado que durante la partida si se pulsa la tecla “escape” se consigue la funcionalidad deseada ya que se para el juego, se puede ver el ratón y interactuar con los distintos botones.
- **Mecánica de guardado:** Se ha comprobado que ocurría un error ya que no se estaban guardando correctamente los datos, esto se ha corregido

añadiendo una variable de guardado en el jugador donde añade todas las variables que se quieren guardar y esta variable es accesible por la clase BP_SaveGame y ya puede realizar el guardado en un archivo de todas estas variables.

- **Mecánica de carga:** Durante esta prueba se ha comprobado el funcionamiento de esta mecánica es correcto.
- **Funcionalidad del menú de opciones:** Durante esta prueba se han detectado, que el cambio en las opciones de sonido no se veía reflejados en el juego, esto se ha corregido realizando un guardado siempre que se cambie algo de este menú.

4 CONCLUSIONES Y TRABAJOS FUTUROS

El desarrollo de un remake de Doom en Unreal Engine ha sido un proyecto desafiante y gratificante. A lo largo del proceso, se ha logrado capturar la esencia clásica del juego, manteniendo la acción rápida y frenética, mientras se aprovechaban las ventajas de un motor moderno como Unreal Engine para mejorar gráficos, físicas, y jugabilidad.

El proyecto ha demostrado la capacidad de utilizar motores de juegos modernos para revitalizar clásicos, ofreciendo una experiencia de juego más pulida y optimizada sin perder el alma del título original.

Debido a las limitaciones de tiempo, este juego sólo se puede considerar un prototipo o demo del mismo, al que le faltaría añadirle funcionalidades que no ha sido posible realizar, como:

- Implementación de un mayor número de niveles.
- Implementación de un modo multijugador.
- Aumentar el número de enemigos.
- Aumentar el número de armas.

Estas mejoras son sólo algunas de las que se podrían añadir al proyecto para mejorar el producto final. Realizar un proyecto de esta envergadura es demasiado

grande para realizarla en solitario, aunque esto nos ha permitido ampliar nuestras capacidades y conocimientos tanto en modelado 3D como en programación de videojuegos.

5 APÉNDICES

5.1 Guía original del Trabajo Fin de Título

Este trabajo consiste en la realización de un remake de algún videojuego clásico utilizando un motor de videojuegos actual, como Unreal Engine, CryEngine o Unity. El videojuego original, así como el motor a utilizar, quedan a elección del alumnado. Hay que describir las mecánicas de juego y la experiencia de usuario perseguida, incluyendo el sistema de recompensas, los mecanismos de progreso del usuario, los recursos gráficos y sonoros empleados, etc. Para la elaboración de escenas, personajes, NPCs, objetos, animaciones y todo tipo de material gráfico y sonoro, se podrán utilizar elementos descargables de la red, incluyendo sprites originales o cualquier otro recurso. Se recomienda elegir un videojuego de la década de los 80s y 90s, buscando la sencillez.

5.1.1 Objetivos del TFG

• Diseño del remake del videojuego clásico elegido, especificando las características y el sistema/entorno objetivo. • Descripción de las dinámicas, mecánicas y componentes de juego, que deberían ser las mismas del juego original. Se permite añadir contenido adicional o variaciones. • Diseño de la interfaz de usuario y los mecanismos de interacción. • Diseño del entorno virtual 2D/3D, incluyendo todo el material gráfico y sonoro. • Desarrollo del juego propuesto, incluyendo la programación de los componentes necesarios. • Realización de pruebas de evaluación del sistema y análisis de resultados.

5.1.2 Metodología a desarrollar

- Describir el videojuego clásico elegido para realizar el remake. Describir las dinámicas, mecánicas y componentes de juego, lo que incluye narrativa, objetivos, recompensas, etc.

- Realizar un breve estado del arte sobre motores de videojuegos y seleccionar uno de ellos. Justificar las decisiones tomadas.
- Seleccionar y utilizar una metodología basada en Ingeniería del Software para el análisis de requisitos, diseño, implementación, prueba y documentación. Se recomienda utilizar una metodología incremental.
- Detallar la estimación temporal y costes de desarrollo.
- Realizar el desarrollo del remake, incluyendo el modelado del entorno virtual 2D/3D.
- Realizar pruebas para el prototipo y documentar los resultados.
- Redacción de la documentación final requerida.

5.1.3 Documentos y Formatos de Entrega

- Documentación generada durante el proceso, de acuerdo a las buenas prácticas de la ingeniería del software. Esto se aplicará tanto en los proyectos de Ingeniería como en los trabajos teóricos o experimentales, es decir, siempre que se genere algún tipo de desarrollo, tanto software como webs, scripts, etc.
- Para el formato de documentación de los Proyectos de Ingeniería se utilizará la norma UNE 157801. Tal y como especifican las normas de estilo de la EPSJ: 'Los Proyectos de Ingeniería deberán presentar una estructura y contenido adaptados a la norma UNE 157001 - Criterios generales para la elaboración de proyectos - u otra derivada de ésta, en función de la naturaleza del proyecto'. '…para el caso de los sistemas de información informatizados (sistemas informáticos, sistemas de información geográfica, etc.) está la norma derivada de la anterior UNE 157801 - Criterios generales para la elaboración de proyectos de sistemas de información'. Para ello, se puede utilizar también la Norma CCII-N2016-02 (Norma Técnica para la realización de la Documentación de Proyectos en Ingeniería Informática) del Consejo General de Colegios Profesionales de Ingeniería en Informática, que incluye e implementa la norma UNE 157801.
- Memoria del trabajo, incluyendo manuales y anexos, en formato PDF.

- Código fuente completo y ejecutables del prototipo o software desarrollado.
- Documentos y archivos adicionales a los que se haga mención expresa en la memoria.
- Vídeo de demostración de la aplicación o de la experimentación realizada.
- Anexos para la presentación del trabajo:
  - Informe favorable del tutor.
  - Autorización de publicación, si procede.

5.1.4 Modificación del TFG

Se realizó un cambio de título de TFG de “Remake de un videojuego clásico” a “Remake de un Shooter Clásico”.

5.2 Manuales de usuario

5.2.1 Controles

- Movimiento del jugador a la izquierda: “A”.
- Movimiento del jugador a la derecha: “B”.
- Movimiento del jugador al frente: “W”.
- Movimiento del jugador hacía atrás: “S”.
- Apuntado: Movimiento del ratón.
- Disparar: “Click izquierdo” del ratón.
- Recargar: “R”
- Cambiar arma: Teclas “1,2,3,4”
- Pausar el juego: Tecla “Escape”.
- Interactuar: “E”

6 DEFINICIONES Y ABREVIATURAS

- **Reinicio (Reboot)**: En el contexto de videojuegos, un reinicio se refiere al desarrollo de un nuevo juego dentro de una franquicia, donde se comienza desde cero, rediseñando personajes, historia y mecánicas. Se hace para actualizar una serie o cambiar su dirección, dejando de lado lo establecido en títulos anteriores.
- **Franquicia**: Una serie de videojuegos relacionados entre sí, normalmente con personajes, mundos o mecánicas comunes. Un ejemplo de franquicia sería *The Legend of Zelda*, que incluye varios juegos dentro del mismo universo, aunque cada título puede variar en estilo y mecánicas.
- **Motor gráfico (Game Engine)**: El software que facilita la creación de videojuegos al proporcionar herramientas como renderizado de gráficos, físicas, simulaciones y gestión de assets.
- **Mecánicas**: Son las reglas y sistemas que definen la jugabilidad en un videojuego. Incluyen acciones que el jugador puede realizar, como moverse, saltar, atacar o resolver puzles, y cómo esas acciones interactúan con el entorno y otros personajes.
- **Software auxiliar**: Herramientas adicionales utilizadas junto con Unreal Engine para crear contenido o mejorar el proceso de desarrollo. Estos pueden incluir programas de modelado 3D como Blender.
- •**Interfaz (UI - User Interface)**: La disposición gráfica con la que el jugador interactúa, incluyendo menús, HUD (Heads-Up Display) y otros elementos visuales que proporcionan información y permiten navegar por el juego.
- **Modders**: Personas que modifican un videojuego existente, creando nuevo contenido o alterando el ya presente.
- **Speedrunner**: Jugadores que buscan completar un videojuego lo más rápido posible, utilizando técnicas que a menudo explotan bugs, glitches, o mecánicas avanzadas del juego.

- **Texturas**: Imágenes bidimensionales que se aplican a los modelos tridimensionales (meshes) para darles detalles visuales como colores, sombras y relieves.
- **Assets**: Son todos los recursos que componen un videojuego, como modelos 3D, texturas, sonidos, animaciones, y efectos especiales.
- **Sprites**: Imágenes bidimensionales que representan objetos, personajes o efectos en juegos 2D o en juegos 3D con elementos bidimensionales.
- **Apuntado (Aiming)**: Mecánica que permite al jugador controlar hacia dónde apunta su personaje, generalmente utilizando el ratón o un joystick para dirigir la vista de la cámara o una mira.
- **Bugs**: Errores o fallos en el código o diseño de un videojuego que causan comportamientos inesperados o no deseados.
- **Pickup**: Objetos interactivos que el jugador puede recoger dentro del juego, como armas, municiones, salud u otros ítems.
- **Mesh**: Un modelo tridimensional formado por vértices, aristas y caras. Los meshes son usados para construir los personajes, objetos y ambientes en un juego. En Unreal Engine, los meshes se importan y luego se les aplican texturas, animaciones y colisiones.
- **Timeline**: Herramienta utilizada en Unreal Engine para crear animaciones o controlar la evolución de ciertos eventos en el tiempo. Se utiliza para manejar cambios en variables como posición, rotación o intensidad de efectos visuales a lo largo de un período determinado.
- **FPS (First-Person Shooter)**: Género de videojuegos en el que el jugador experimenta la acción a través de los ojos del protagonista, con un enfoque en disparos y combates.
- **3D (Three-Dimensional)**: Representación gráfica que utiliza tres dimensiones (alto, ancho y profundidad) para crear objetos y entornos realistas o estilizados en un videojuego.

- **PC (Personal Computer)**: Plataforma de hardware utilizada para ejecutar videojuegos, entre otras aplicaciones. En el contexto de desarrollo, los juegos para PC son una de las plataformas más comunes y flexibles.
- **2.5D (Two-and-a-half-Dimensional)**: Estilo gráfico que mezcla elementos en 2D y 3D. Los juegos 2.5D tienen gráficos tridimensionales, pero con jugabilidad en dos dimensiones (plano horizontal y vertical).
- **TFG (Trabajo de Fin de Grado)**: Documento o proyecto final que los estudiantes deben realizar al concluir sus estudios universitarios de grado.
- •**VR (Virtual Reality)**: Tecnología que permite a los usuarios interactuar con entornos digitales tridimensionales inmersivos utilizando dispositivos especializados como gafas de realidad virtual.
- **AR (Augmented Reality)**: Tecnología que superpone objetos o información digital sobre el mundo real a través de dispositivos como móviles o gafas.
- **C#**: Lenguaje de programación desarrollado por Microsoft, usado en la creación de videojuegos, especialmente en el motor Unity. Es conocido por su simplicidad y flexibilidad en la creación de sistemas complejos.
- **AAA (Triple-A)**: Término que se refiere a videojuegos con un presupuesto elevado de desarrollo y marketing, producidos por grandes estudios.
- **AWS (Amazon Web Services)**: Plataforma de servicios en la nube ofrecida por Amazon, utilizada en desarrollo de videojuegos para servicios en línea, servidores, almacenamiento, inteligencia artificial y más.
- **IDE (Integrated Development Environment)**: Entorno de desarrollo integrado que facilita la programación al combinar herramientas de edición de código, depuración, y compilación.
- **C++**: Lenguaje de programación de alto rendimiento ampliamente utilizado en el desarrollo de videojuegos debido a su capacidad para manejar grandes cantidades de procesamiento y memoria.
- **IA (Inteligencia Artificial)**: En videojuegos, se refiere a la simulación de comportamientos inteligentes en NPCs (personajes no jugadores) o

sistemas del juego para hacerlos interactuar de manera realista con el entorno y el jugador.

- **€ (Euro)**: La moneda oficial de la zona euro, utilizada en muchos países de la Unión Europea.
- **CPU (Central Processing Unit)**: El procesador central de un ordenador, encargado de ejecutar las instrucciones básicas de los programas, incluyendo videojuegos. Es responsable del rendimiento general del sistema.
- **GPU (Graphics Processing Unit)**: Unidad de procesamiento gráfico, especializada en manejar y renderizar gráficos 3D en tiempo real en videojuegos.
- **Etc (Etcétera)**: Abreviatura utilizada para indicar que la lista mencionada continúa con elementos similares o relacionados, pero que no se detalla completamente.
- **ABP (Animation Blueprint)**: Sistema en Unreal Engine que permite crear animaciones complejas para personajes u objetos utilizando un gráfico de nodos para definir el comportamiento de las animaciones.
- **PBR (Physically Based Rendering)**: Técnica de renderizado en motores gráficos como Unreal Engine que simula cómo la luz interactúa físicamente con los materiales para producir imágenes más realistas.
- **NPCs (Non-Playable Characters)**: Personajes dentro de un videojuego que no son controlados por los jugadores, sino por la IA. Interactúan con el entorno y con el jugador en función de las reglas y programación establecidas.
- **PDF (Portable Document Format)**: Formato de documento utilizado para compartir archivos de texto e imágenes de manera que se mantengan consistentes en cualquier dispositivo.

7 BIBLIOGRAFÍA

[1] Diferencias entre remake y remasterización en los videojuegos () Consultado 15 de agosto de 2024, de https://www.hobbyconsolas.com/reportajes/diferencias-remake-remasterizacion-videojuegos-177394

[2] Remake de videojuegos. Consultado el 15 de agosto de 2024, de https://en.wikipedia.org/wiki/Video_game_remake

[3] Doom (franquicia). Consultado el 16 de agosto de 2024, de

https://en.wikipedia.org/wiki/Doom_(franchise)

[4] Unity (motor de videojuego). Consultado el 20 de agosto de 2024, de https://es.wikipedia.org/wiki/Unity_(motor_de_videojuego)

[5] Unreal Engine. Consultado el 7 de agosto de 2024, de https://es.wikipedia.org/wiki/Unreal_Engine

[6] CryEngine (motor de videojuego). Consultado el 20 de agosto de 2024, de https://es.wikipedia.org/wiki/CryEngine

[7] Amazon Lumberyard. Consultado el 20 de agosto de 2024, de https://es.wikipedia.org/wiki/Amazon_Lumberyard

[8] Blender (software). Consultado el 20 de agosto de 2024, de https://es.wikipedia.org/wiki/Blender

[9] Autodesk Maya (software). Consultado el 20 de agosto de 2024, de https://es.wikipedia.org/wiki/Autodesk_Maya

[10] Sueldo promedio de un programador de Unreal Engine. Consultado el 30 de agosto de 2024, de https://www.tokioschool.com/formaciones/cursos-videojuegos/programacion-unreal-engine/sueldo/

[11] Heads-up display de Doom. Consultado el 5 de septiembre de 2024, de https://doom.fandom.com/wiki/Heads-up_display

[12] Menú de Doom. Consultado el 5 de septiembre de 2024, de

https://doomwiki.org/wiki/Menu#Doom_menu