Universitat Oberta de Catalunya

uoc.edu

Last Resurrection

Sergio Riquelme Palazón
Master de Diseño y Programación de Videojuegos
TFM – Diseño de experiencias de juego

Jordi Duch Gavaldà
Joan Arnedo Moreno

Enero de 2022

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Last Resurrection*
Nombre del autor:	*Sergio Riquelme Palazón*
Nombre del consultor/a:	*Jordi Duch Gavaldà*
Nombre del PRA:	*Joan Arnedo Moreno*
Fecha de entrega (mm/aaaa):	01/2022
Titulación:	*Master de Diseño y Programación de Videojuegos*
Área del Trabajo Final:	*TFM – Diseño de experiencias de juego*
Idioma del trabajo:	*Castellano*
Palabras clave	*Unity, First-person, Mazmorra*

Resumen del Trabajo (máximo 250 palabras): *Con la finalidad, contexto de aplicación, metodología, resultados i conclusiones del trabajo.*

El trabajo descrito en este documento tiene como objetivo el desarrollo de un juego completo y autosuficiente, aplicando para ello las diferentes metodologías aprendidas en el itinerario de master. Para ello se ha modelado elementos de juego, como los brazos del personaje principal, creado animaciones diversas para articular las acciones del jugador, diseñado escenarios de temática medieval y codificado la interacción completa entre todos los elementos de juego, usando para ello programas como Unity y Blender.

El resultado de este desarrollo es un juego 3D de acción en primera persona en el que el jugador debe hacerse paso a través de múltiples habitaciones y mazmorras, resolviendo puzles o exterminando hordas de enemigos haciendo uso de un amplio arsenal de armas y habilidades, las cuales le permitirán al jugador personalizar su experiencia de juego en adición a un sistema de movimiento fluido, así como varias habitaciones de juego muy distintas entre sí para mantener la curiosidad del jugador hasta el escenario final, donde finalmente podrá enfrentarse al rey demonio.

Abstract (in English, 250 words or less):

The work described in this document aims for the objective of developing a full, self-sufficient game, applying the diffent methodologies learned in the itinerary of the master's degree. To accomplish this, we have modeled game elements, such as the arms of the main character, created various animation to articulate the player's input, design medieval-themed scenarios and coded the complete interaction between all game elements, utilizing for this purpose software such as Unity and Blender.

The result of this development is a first-person, action 3D videogame, in which the player must make his way through multiple rooms and dungeons, solving puzles or exterminating hordes of enemies using a wide arsenal of weapons and magic skills, which will allow the players to personalize their gaming experience int addition to a fluid movement system, as well as several very different game rooms to keep the players curious until the final stage, where they can finally face the demon king.

Índice

Lista de figuras

1. Introducción

1.1. Contexto y justificación del Trabajo

Desde las primeras civilizaciones del ser humano, siempre han existido juegos y pasatiempos que nos han permitido liberarnos del estrés diario de nuestras vidas. En sus inicios, todos estos pasatiempos eran únicamente analógicos, desde jugar con palos y piedras, hasta leer un cuento o ver una obra de teatro.

Este concepto de entretenimiento evolucionó a partir de la segunda mitad del siglo pasando, correspondiéndose con la aparición de los **ordenadores**. Inicialmente, estos aparatos eran increíblemente caros y de enorme tamaño, por lo que únicamente podían encontrarse en universidades o grandes compañías, además de ser usadas casi en exclusiva para la resolución de problemas y ecuaciones matemáticas.

El primer programa que fue desarrollado únicamente con propósito lúdico es el juego *Tennis For Two,* desarrollado por *William Higinbotham* en 1958, el cual consiguió llamar la atención del público general a la hora de considerar a los ordenadores como máquinas de entretenimiento. [1]

Ilustración 1: Tennis For Two. [2]

El siguiente paso en la evolución de los videojuegos fue la aparición de máquinas especializadas para uso lúdico, como fue la *Atari* original, a partir de las cuales se afianzó este nuevo medio de entretenimiento hasta la actualidad.

Desde entonces, la evolución de los videojuegos no ha dejado de incrementar exponencialmente, fuertemente ligada al desarrollo del hardware sobre el que operan. Mientras que en sus inicios solo se podían encontrar juegos cuya representación gráfica se limitaba a unos pocos bits, actualmente podemos encontrar juegos que simulan de manera casi realista escenarios de la vida real, y en los que incluso podemos introducirnos de manera inmersiva haciendo uso de las tecnologías de realidad virtual.

Hoy en día, la gran mayoría de los hogares dispone de algún tipo de dispositivo que les permite jugar videojuegos, ya sea un ordenador de mesa o un teléfono

móvil, lo que ha permitido que cada vez un mayor número de personas se beneficie de los videojuegos para su entretenimiento, ayudando así a decrementar el estrés diario que todos experimentamos.

Con el juego ***Last Resurrection*** se ha querido recrear un estilo de juego acción en primera persona, popularizado por juegos como *Wolfenstein3D* y *DOOM (1993)*, en los que el jugador debe hacerse paso entre una serie de habitaciones, en las cuales deberá eliminar a varios enemigos para poder continuar. Este estilo de juego rápido y fluido es lo que se ha querido conseguir con este proyecto, una modalidad de juego que ayuda al jugador a desprenderse del estrés del día a día para concentrarse plenamente en el entretenimiento que se le presenta delante.

Ilustración 2: Wolfestein 3D. [3]

1.2. Objetivos del Trabajo

Como se ha comentado en el apartado anterior, el objetivo principal de este proyecto es desarrollar un prototipo completo de juego de acción en primera persona, con unos sistemas de combate y movimientos fluidos, sustituyendo los elementos de jugabilidad basada en armas de fuego que caracterizaban a los juegos en los nos basamos para desarrollarlo por un arsenal de estilo medieval, centrado en el combate cuerpo a cuerpo y el uso de armas mágicas.

Además, se ha querido aprovechar la realización de este proyecto para profundizar en las técnicas de desarrollo de videojuegos aprendidas a lo largo de todo el itinerario de master, como por ejemplo el uso de inteligencia artificial en enemigos o las pautas de diseño de niveles para la creación de las habitaciones de juego.

Por otra parte, se quiere conseguir una mayor familiaridad con el uso de plataformas de desarrollo de videojuegos como *Unity*, un software que ha conseguido colocarse como uno de los líderes del mercado a la hora de producir videojuegos profesionales y de alta sostenibilidad comercial.

Finalmente, con este proyecto final de desarrollo de videojuego se ha querido poner en práctica las múltiples etapas que conllevan realizar un videojuego completo, desde su fase inicial de diseño y conceptualización, hasta sus etapas finales de *testing* y despliegue.

1.3. Enfoque y método seguido

Como se ha comentado al final del apartado anterior, el enfoque que se ha elegido para este proyecto es comenzar desde cero un juego completo. Partiendo desde una idea inicial, el diseño de mecánicas de juego, conceptualización y modelado de elementos de juego, desarrollo de mecánicas principales y construcción del escenario de juego, hasta llegar a las fase de postproducción, *testing* y despliegue de un juego completo.

Ilustración 3: Fases del desarrollo de videojuegos. Fuente: [4]

Esta metodología nos permite abarcar las principales fases que se llevan a cabo en la producción de un juego completo, adquiriendo así experiencia de cara a la posibilidad de poder participar en el desarrollo comercial de un juego de estas características.

1.4. Planificación del Trabajo

La planificación de las asignatura asociada a este proyecto de fin de master comprendía desde el 15 de septiembre hasta el 16 de enero, siendo el 2 de enero el tiempo límite de realización del trabajo principal de desarrollo. Dentro de este periodo, se realizó el planteamiento inicial de un tiempo total de desarrollo de 300 horas, correspondientes a la cuantificación total de los 12 créditos de las asignatura. Teniendo en cuenta eso, en la planificación original del proyecto presentado en la primera PEC se planteó el siguiente desglose de hitos parciales a realizar:

1. Evaluación de assets y librerías a utilizar.
2. Creación del personaje con cámara en primera persona, movimiento simple, sprint y salto.
3. UI inicial.
4. Añadir brazos al personaje en primera persona y uso de armas de combate.
5. Introducir enemigos con lógica simple de salud y muerte.
6. Interacción de los ataques del jugador con los enemigos.
7. Movimiento y animación de los enemigos.
8. Inteligencia artificial de los enemigos para perseguir al jugador.

9. Ataques de los enemigos al personaje principal y sistema de vida y muerte del jugador.
10. Bolsa de dinero, recogida de dinero y drop de dinero de los enemigos al morir.
11. Sistema de mejora del personaje y de las armas en la tienda goblin.
12. Trampas y elementos de sala y su interacción con jugador y enemigos.
13. Creación de una sala y sistema de spawn de la horda.
14. Continuación de juego al completar una sala y sistema de respawn al morir.
15. Sistema de niveles de dificultad de un nido de horda.
16. Diseño y desarrollo del resto de salas y estructura de juego, incluyendo salas de plataformas.
17. Boss final.
18. Sonidos y música.
19. Testing de estructura de juego principal.
20. Iluminación y post-procesado.

Pero como en todo proyecto de desarrollo de videojuegos, la evaluación inicial de objetivos a completar en el tiempo de producción estipulado no ha sido completamente realista, por lo que a continuación daremos un mejor desglose de la planificación realizada en cada una de las PEC parciales asociadas con las asignatura de TFM haciendo uso de diagramas de Gantt.

1.4.1. PEC1

En esta primera entrega se debía presentar un documento ligero de diseño del videojuego, por lo que los hitos alcanzados fueron los siguientes

- Conceptualización inicial.
- Estudio de tecnologías.
- Diseño de mecánicas principales.

1.4.2. PEC2

Para la segunda entrega se nos marcaba el objetivo de tener un primer prototipo del juego en el que haber desarrollado las mecánicas principales de juego. Las tareas realizadas durante este segundo sprint son las siguientes:

- Recopilación de *assets* de terceros.
- Modelado de brazos de personaje principal.
- Creación de animaciones para el personaje principal.
- Sistema de vida de los actores de juego.
- Movimiento del personaje principal.
- Combate básico del personaje con espada y hacha de dos manos.
- IA inicial de enemigos humanoides.
- Interacción entre enemigos y personajes.

1.4.3. PEC3

Llegando a estar tercera PEC ya se nos pedía tener una versión completa y jugable del proyecto, por lo que las tareas realizadas se centran en terminar las mecánicas principales de juego, incluyendo las hordas de enemigos, y tener un nivel completo de juego, así como el desarrollo de juego completo desde la escena de inicio hasta la pantalla de fin según las condiciones de victoria. Las tareas desarrolladas en este periodo de desarrollo son las siguientes:

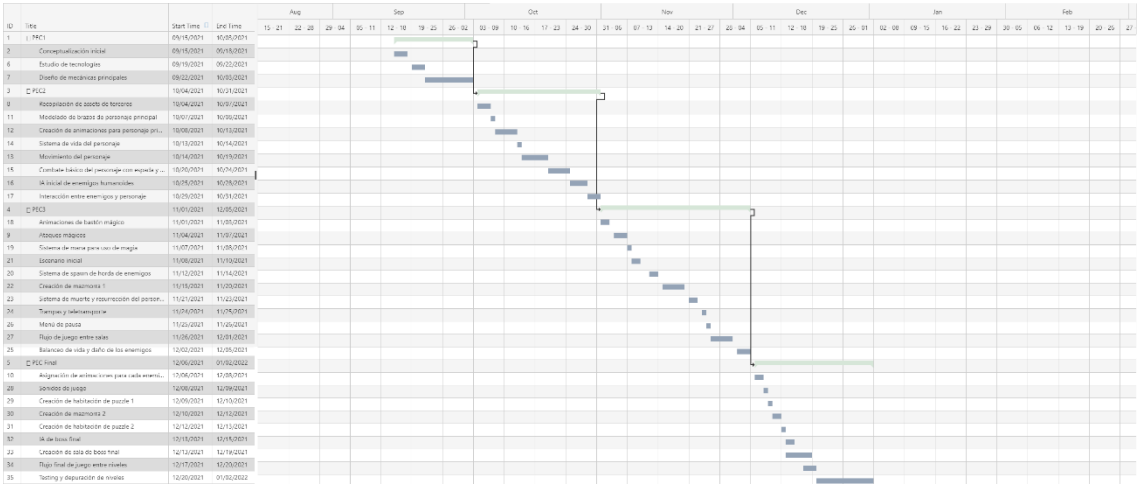
- Animaciones de bastón mágico.
- Ataques mágicos.
- Sistema de mana para uso de magia.
- Creación del escenario de inicio.
- Sistema de *spawn* hordas de enemigos.
- Creación de la primera mazmorra de horda.
- Sistema de muerte y reaparición del personaje principal.
- Creación de trampas de pinchos y teletransporte.
- Menú de pausa simple.
- Flujo de juego entre salas y condiciones de victoria.
- Balanceo de vida y daño de los enemigos.

1.4.4. PEC4

En esta última PEC, al tener menos de un mes de tiempo de desarrollo, el cual se debía compartir con la escritura de la memoria, se han centrado las tareas en pulir la experiencia de juego, limitando la adición de nuevas mecánicas de juego fuera de lo posible. Por ello, los hitos finales del desarrollo del proyecto han sido los siguientes:

- Asignación de animaciones personalizadas para cada enemigo del juego.
- Adición de sonidos a todo el juego.
- Creación de la primera habitación de puzle.
- Creación de la segunda habitación de horda de enemigos.
- Creación de una segunda habitación de puzle.
- IA asociada al *boss* final.
- Creación de la sala de *boss* y su lógica asociada.
- Definición del flujo final entre escenas de juego desde el principio hasta la sala de *boss* con sus condiciones de victoria.
- Testing y depuración del juego, arreglando y retocando aquellos errores y posibles mejoras que han ido apareciendo.

Para visualizar mejor cada uno de estos hitos conseguidos en su línea temporal correspondiente, se añade a continuación el diagrama de *Gantt* correspondiente:



1.5. Breve sumario de productos obtenidos

- Un **personaje principal** con perspectiva en **primera persona**, movimiento rápido y fluido, y un sistema de combate diverso con dos tipos de armas cuerpo a cuerpo: espada y hacha a dos manos, y un bastón mágico con dos tipos de ataques: carámbano de hielo y bola de fuego.
- Sistema básico de **inteligencia artificial para enemigos** de distintos tipos implementados mediante **máquina de estados finita** basada en persecución y ataque al personaje principal, incluyendo tanto enemigos cuerpo a cuerpo (zombis, esqueletos, etc.…) como un mago a distancia.
- Mazmorras de sala de enemigos con sistema completo de aparición de **oleadas de enemigos** desde portales colocados en el mapa.
- Habitaciones de **puzle** en las que el jugador debe coger llaves para desbloquear la puerta a la siguiente habitación superando ciertas
- **Pociones** para recuperar vida y maná.
- Elementos de sala para personalizar aún más las salas o ser usados para ciertos puzles, como las **trampas de pinchos**, **teletransportes** y las **plataformas de salto**.
- Un **boss completo** con varias fases de combate diferenciadas para proporcionar un desafío final al jugador.

1.6. Breve descripción de los capítulos de la memoria

Vamos a dar una pequeña descripción de los capítulos que se van a presentar a continuación en el documento:

- **Estado del arte**: Haremos un repaso del estado actual del género asociado al tipo de juego que se quiere desarrollar, enumerando referenciar utilizadas.
- **Definición del juego**: Se dará una revisión del documento de diseño del videojuego que se dio en la primera entrega parcial.
- **Diseño técnico**: En este apartado se desarrollará el grueso del trabajo realizado, incluyendo descripción de componentes utilizados y estructura de la funcionalidad de escenas y actores de juego, incluyendo la inteligencia artificial de los enemigos.
- **Diseño de niveles**: Describiremos las decisiones de diseño para una habitación de juego de cada tipo.
- **Manual de usuario**: Controles de juego y requerimientos técnicos.
- **Conclusiones**: Análisis de los objetivos cumplidos durante el proyecto, así como las lecciones aprendidas y posibles mejoras a realizar en el futuro.

2. Estado del arte

2.1. Género de juego

Como hemos comentado durante la introducción de este documento, podemos categorizar el juego desarrollado en este proyecto de fin de master como **acción aventura en primera persona**.

Este estilo de juegos se caracteriza principalmente por tener una progresión **lineal** con un sistema de combate **rápido y fluido,** en los que el jugador debe hacerse paso en una serie de habitaciones y salas con diseños de nivel especializados para el combate contra hordas de enemigos de características muy variadas, haciendo uso de un amplio arsenal de armas y habilidades para mejorar la experiencia de juego.

Como hemos comentado, este género de combate de hordas de enemigos se inició originalmente con juego como *DOOM o Wolfenstein 3D*, con una progresión únicamente lineal, pero el género ha ido evolucionando con los años, dando lugar a subgéneros como los *"**rogelike**"*, en los que el jugador también debe adentrarse en una serie de salas y derrotar a una horda de enemigos para continuar, con la diferencia de que no se centran en una historia con progresión lineal, sino que las habitaciones suelen crearse de manera aleatoria cada vez que el jugador empieza una partida.

A continuación vamos a comentar algunos juegos de este género y de otros similares en los que más nos hemos fijado a la hora de desarrollar este proyecto.

2.1.2. DOOM (2016)

Como comentamos durante la introducción de este documento, *DOOM* es uno de los precursores del género de acción en primera persona. En 2016 ***ID Software*** resucitó a esta franquicia sin perder la esencia original: ser un *shooter* en primera persona con un estilo de juego **frenético** y realmente violento.

Esta entrega del juego ha sido una de las mayores inspiraciones a la hora de realizar el proyecto descrito en este documento. Nuestro jugador se pone en la piel de un personaje para acabar con oleadas de enemigos a gran velocidad y usando un amplio arsenal de armas.

Ilustración 4: Doom (2016), captura de juego. Fuente: [5]

2.1.2. Skyrim

Este juego, desarrollado por *Bethesda* en 2011, ha sido uno de los juegos más influyentes de las últimas décadas en lo que respecta a **alta fantasía**. En él, el jugador encarna el papel de un *Dovahkinn* o "Sangre de dragón", lo que le permite tener habilidades especiales de un descendiente de dragones, habilidades que pondrá en uso para acabar con la amenaza de los dragones resucitados que asolan la tierra de *Skyrim.*
Al igual que los juegos comentados anteriormente, *Skyrim* es un juego de acción en primera persona, pero a diferencia de estos se trata de un **RPG** puro en el que el jugador mejorará sus habilidades y equipamiento conforme vaya avanzando en el juego para hacerse más fuerte, además de un estilo de mundo de tipo **mundo abierto**.

Ilustración 5: Captura de juego de Skyrim. Fuente: [6]

Una de las primeras decisiones de diseño que se tuvieron en cuenta a la hora conceptualizar nuestro proyecto, es que se quería crear un estilo de juego similar a *shooters* en primera persona como *DOOM*, pero utilizando un estilo de combate y ambientación basado en la **fantasía medieval**, y ahí en donde entra el papel de *Skyrim* como principal inspiración para el **sistema de combate** de nuestro proyecto.

2.2. Revisión sobre la tecnología

En los últimos años, el desarrollo de videojuegos ha crecido exponencialmente, parcialmente gracias a la aparición de **múltiples plataformas y motores gráficos** para producción de videojuegos, facilitando enormemente su desarrollo y permitiendo que incluso equipos de pocas personas o individuos particulares puedan acceder a la industria.

Primero podemos comentar algunas plataformas de desarrollo como ***RPG Maker***, que engloba a una amplia serie de programas que incluyen editores de mapas y scripts para crear juegos RPG en 2D. Esta limitación hacía imposible utilizarlo para nuestro proyecto 3D.

Por otro lado, en lo que respecta a creación de juegos 2D de manera más libre y personalizada que RPG Maker tenemos **Godot**, un motor de videojuegos especializado en 2D, pero que también cuenta actualmente con la posibilidad de crear juegos en 3D.

Si pasamos a comentar los principales motores de videojuegos especializados para entornos 3D, podemos comentar tanto **Unreal Engine** como **Unity**.

Unreal Engine fue creado inicialmente por **Epic Games** para el juego *Unreal*, un shooter en primera persona, y aunque inicialmente su uso se centró en el desarrollo de juegos de este estilo, con el tiempo ha ido evolucionando su uso a todo tipo de géneros. Además, se ha afianzado como el motor gráfico que mejores resultados gráficos consigue entre sus competidores.

Ilustración 6. Captura de pantalla de Unreal Engine 5. Fuente: [7]

Pero ya que el tiempo de desarrollo de este proyecto fin de master es bastante limitado y no se tenía experiencia previa en ninguna de las plataformas de desarrollo que se acaban de comentar, finalmente se decidió por usar **Unity**, un motor gráfico **multiplataforma** que proporciona gran flexibilidad en cuanto a los tipos de juego que se pueden desarrollar, ya sean 2D o 3D.

Además, *Unity* cuenta con una **comunidad de desarrolladores muy amplia**, lo que facilita mucho el encontrar recursos y tutoriales, facilitando así el desarrollo. Y por último, esta ha sido la plataforma que hemos ido utilizando durante todas las asignaturas cursadas en el itinerario de master, por lo que era la mejor opción a la hora de realizar un proyecto con tiempo de desarrollo tan limitado.

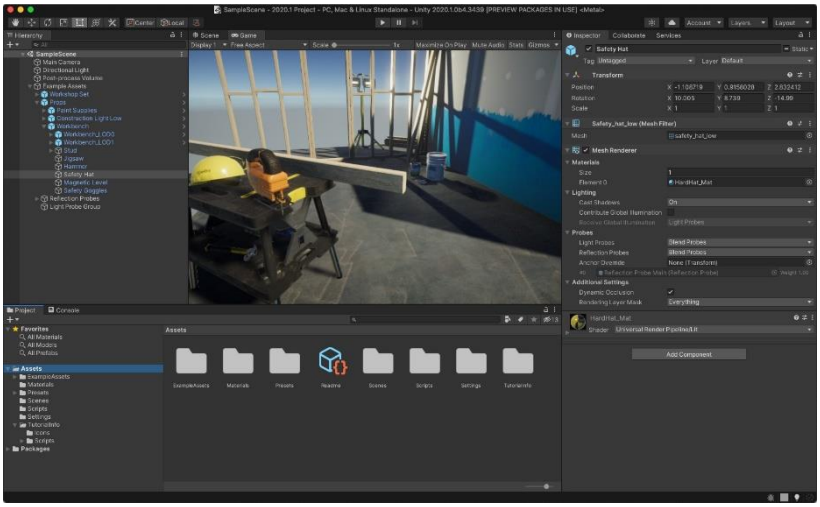


Ilustración 7. Plataforma de desarrollo Unity. Fuente: [8]

3. Definición del juego

3.1. Descripción breve de juego

Last Resurrection es un juego de acción en primera persona en el que el jugador deberá superar una serie de salas y mazmorras, cada una de ellas habitada por una **horda** de distintos enemigos o con un puzle a resolver, hasta llegar a la sala del trono y derrotar al **rey demonio**. Para conseguirlo, el jugador hará uso de **varios tipos de armas**, incluyendo armas cuerpo a cuerpo y bastones mágicos para acabar con las hordas de enemigos.

3.2. Historia y ambientación

La ambientación del juego estará basada principalmente en la que podemos ver en historias **medievales**, en concreto con temática de **fantasía oscura.** Por ello se pretende introducir estructuras típicas de la época, entre las que podemos destacar **mazmorras** y **cementerios**, en las que podremos destacar el uso de la iluminación para mejorar la sensación de un mundo desolado.

Ilustración 8. Ambientación dark fantasy.

En cuanto a la historia, se ha optado por crear una pequeña línea narrita en la cual nuestro protagonista, **Berric**, es un aprendiz de herrero en una gran ciudad de época estilo medieval. Un día, mientras Berric salía de la ciudad para comprar material en un pueblo cercano, un rey demonio aparece en el cielo, consumiendo de manera inmediata la energía vital de todos los humanos de la ciudad con el objetivo de absorber todas las almas mortales.
Al encontrarse en los bordes de la ciudad, Berric se desploma moribundo en el suelo mientras el rey demonio intenta absorber las almas, pero algo sale mal y las almas entran en el cuerpo del protagonista, lo que permite que pueda **resucitar**.
Usando la capacidad de resurrección que le proporcionan las almas, Berric se propone **acabar con el rey demonio** mientras le quede una última resurrección.

3.3. Definición de los personajes y elementos

- **Berric**: Aprendiz de herrero y protagonista del juego. Al absorber las almas de los habitantes de la ciudad consigue la habilidad de resucitar agotando una de las almas contenidas dentro de él.
- **Rey demonio**: Exterminador de vida humana en planeta. Ha tomado el trono del antiguo rey del reino.
- **Súbditos**: Todos los siervos del mal convocados por el rey demonio, incluyendo esqueletos, zombis, ogros, vikingos o hechiceros.
- **Portales de horda**. Elementos de las salas de las mazmorras de nido de horda desde las que irán apareciendo progresivamente oleadas de enemigos.
- **Trampas y elementos de sala.** A lo largo de las salas y mazmorras de la ciudad, habrán colocados múltiples tipos de trampas y otros elementos que pueden perjudicar o ayudar tanto al jugador como a enemigos.
- **Pociones**. Brebajes alquímicos que permiten recuperar vida o maná según su color.
- **Llaves**. Objetos esenciales a la hora de abrir las puertas de ciertos niveles de puzle.
- **Puertas de salida**. Puertas de **limitan el progreso** del usuario en el juego. Se desbloquean tanto derrotando por completo a la horda de enemigos de una sala, o cogiendo todas las llamas de una sala de puzle.

3.4. Interacción entre los actores de juego

- **Interacciones del jugador:**
    - El jugador podrá usar las armas de las que dispone para atacar y matar a los **súbditos** del rey demonio.
    - Si el jugador consigue llegar a la etapa final del juego, podrá enfrentarse y derrotar al **rey demonio** usando las armas comentadas anteriormente.
    - Los nidos de horda son habitaciones especiales en las cual el jugador debe **derrotar a una horda de enemigos** para continuar en la historia.
    - Cada vez que el jugador acabe por completo con la horda de una habitación, se actualizará su **punto de reaparición**, en función de la sala que acaba de completar.
    - Otro tipo de sala en las que tendrá que entrar el jugador son las salas de **puzle**, en las que deberá completar ciertos desafíos de plataformas o pequeños rompecabezas para poder recoger llaves que se le permitirán abrir las puertas hacia la siguiente sala.
    - En las habitaciones, el jugador podrá encontrarse con **trampas** como pinchos y zonas de lava con las que perderá vida o morirá si interacciona con ellas. También podrá usar elementos como **portales, plataformas** de salto o **escaleras** para facilitar su movimiento por las salas.
    - El jugador deberá eliminar a todos los enemigos de una horda o coger todas las llaves de una sala de puzle para abrir **la puerta de salida** y continuar al siguiente nivel o mazmorra.

    - Por último, el jugador también podrá recoger **pociones** de vida y maná para recuperar cierta cantidad de estos recursos
- **Interacciones de los enemigos**:
    - Los súbditos del rey demonio intentarán siempre matar al jugador cuando lo vean, **persiguiéndole** hasta estar a rango para **atacarle**.
    - Al igual que el jugador, los **enemigos** pueden recibir daño si interaccionan con las **trampas** de las salas de horda.
    - Los enemigos irán apareciendo en oleadas de los **portales de horda**.

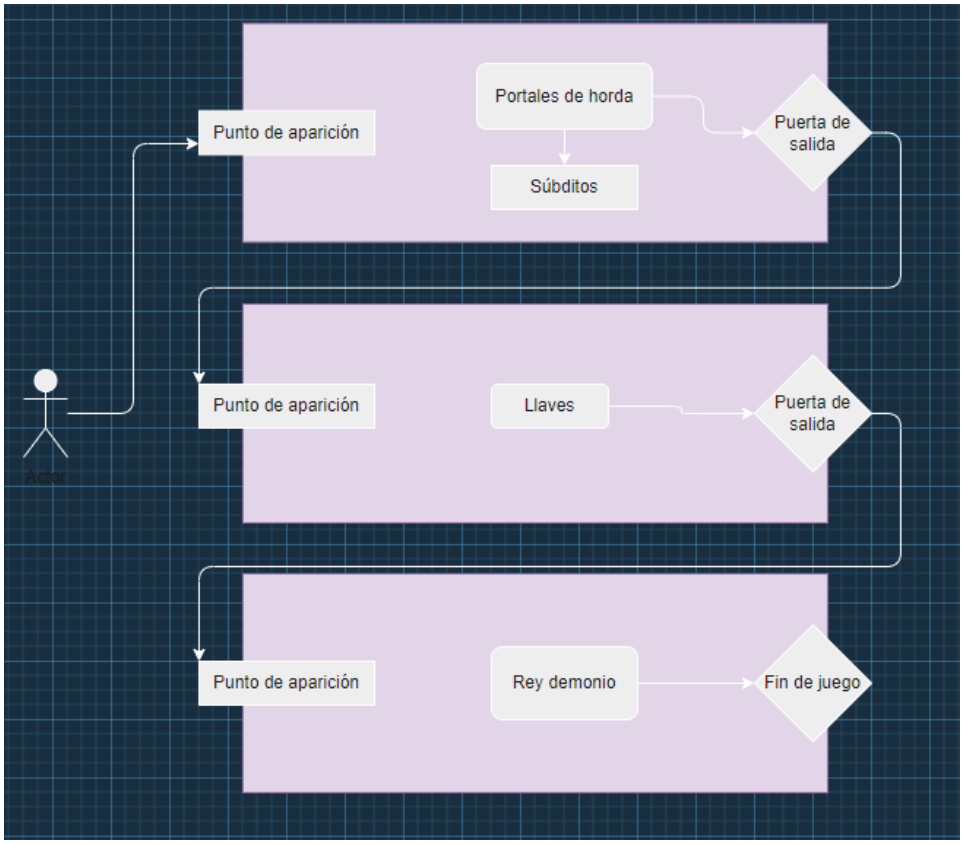


Ilustración 9: Estructura general de juego.

3.5. Objetivos planteados al jugador

El desarrollo de la historia y del juego seguirá una progresión principalmente lineal. El jugador tendrá que **avanzar de sala en sala** hasta llegar al rey demonio y acabar con él para terminar el juego.
Cuando nos referimos a una sala consideramos tres tipos posibles de sala:

- **Nido de horda**: Habitación en la que el jugador tendrá que **derrotar** a una horda de enemigos para poder **continuar**. Cada sala tendrá ciertos tipos de enemigos concretos así como una **estructura distinta** al resto en cuanto a trampas y elementos de sala.
- **Sala de puzle**: Salas en las que se deben completar desafíos de plataformas o pequeños puzles para coger cierto número de llaves y abrir la puerta de salida.
- **Sala de boss**: Último escenario del juego en el que el jugador deberá derrotar al rey demonio, el cual cuenta con distintas fases de combate, para terminar la partida

3.6. Plataforma destino

Como hemos comentado en el apartado anterior, principalmente nos centraremos en producir el juego para **ordenador**, principalmente debido al corto tiempo de desarrollo del que se dispone para completarlo, y al hecho de que no disponemos de experiencia previa en desarrollo para otros tipos de plataformas, lo que aumentaría significativamente el tiempo necesario para completarlo.

4. Diseño técnico

4.1. Entorno de desarrollo

Como hemos comentado en secciones anteriores, el entorno de desarrollo elegido para este proyecto es **Unity**. El motivo principal por el cual se ha elegido este entorno antes que otros motores gráficos es por la familiaridad que tenemos en su uso, lo que facilitó la posibilidad de desarrollar un juego por completo en el **limitado tiempo** del que se disponía para terminarlo.

En concreto, el videojuego se ha desarrollado en la versión **2020.3.17f1**, al ser la versión más estable de la plataforma en el momento en que se empezó la producción

4.2. Requerimientos técnicos de desarrollo

El videojuego se ha desarrollado en un ordenador con las siguientes características técnicas:

- **Sistema operativo**: Windows 10 Pro.
- **Procesador**: AMD Ryzen 7 3700X 8-Core Processor, 3.60 GHz.
- **Memoria RAM**: 32 GB.
- **Tarjeta gráfica**: Radeon RX 580 Series

4.3. Herramientas utilizadas

- **Unity**: Como hemos comentado, plataforma principal de desarrollo de nuestro proyecto.
- **Blender**: Software gratuito de modelado 3D, usado principalmente para creación de brazos del jugador y creación de animaciones para el personaje.
- **SourceTree**: Programa de control de versiones para mantener un repositorio local de nuestro proyecto y poder subirlo con facilidad a **GitHub**.
- **Audacity**: Aplicación de software libre, usada principalmente para grabación y edición de audio. Su uso ha sido simplemente para retocar alguno de los sonidos usados en el juego.
- **Microsoft Word:** Procesador de texto utilizado para escribir la memoria del proyecto de fin de master.

4.4. Inventario de *assets* y recursos de juego

En este apartado daremos listaremos cada uno de los ***assets*** usados para dar forma a nuestro videojuego, incluyendo tanto paquetes y modelos descargados de terceros, como elementos de creación propia

4.4.1. *Assets* de terceros

Empezaremos enumerando aquellos elementos obtenidos de terceros, así como de donde proceden.

4.4.1.1. ARPG Effects

Este paquete de *assets* ha sido comprado en la *Unity Asset Store*, y contiene números **sistemas de partículas** de uso general, especialmente para juegos de temática RPG fantásticos. [9]

En nuestro caso hemos utilizado principalmente los efectos de **portal rojo**, para representar a los portales de horda**,** los efectos de **fuego** para las antorchas y otros efectos extra por ejemplo para representar los **teletransportes** o las zonas de **salida a la siguiente habitación**.

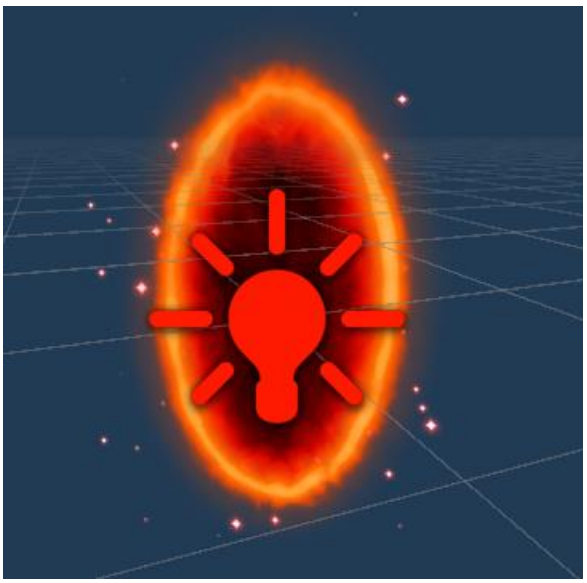


Ilustración 10. VFX de portal rojo.

4.4.1.2. Unique Projectiles Volume 2

Paquete de efectos de partículas centrados en **proyectiles mágicos**, creado por **GabrielAguiarProductions** y obtenido en la *Unity Asset Store.* [10]

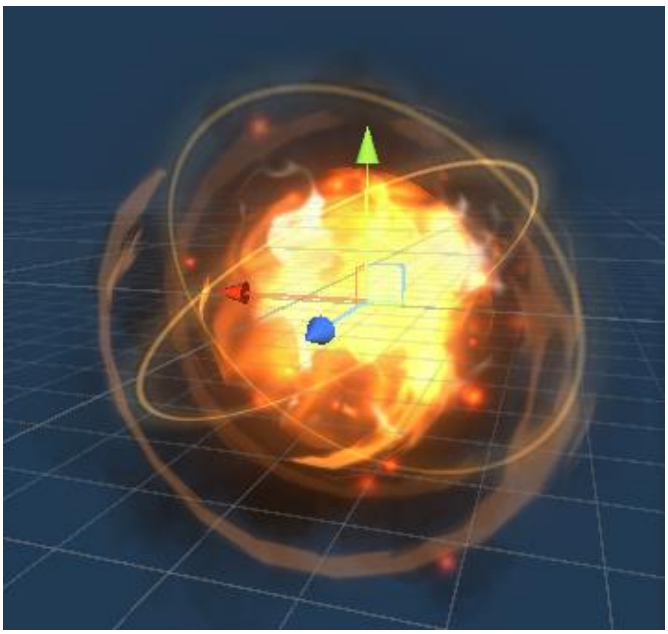


Ilustración 11. Bola de fuego.

De este paquete se han utilizado principalmente los *prefabs* para los dos ataques mágicos del jugador: **bola de fuego** y **carámbano de hielo**, y para los ataques mágicos **algunos enemigos**.

4.4.1.3. GUI PRO Kit – Fantasy RPG

Otro de los paquetes de *assets* obtenidos de la *Unity Asset Store*, en el cual podemos encontrar elementos para creación de la UI de juego, especializado también en temática fantástica RPG. [11]

Por ello, todos los elementos de UI introducidos en nuestro juego pertenecen a este paquete: **barras de vida y maná**, **botones**, **paneles** y **fondos de menú.**

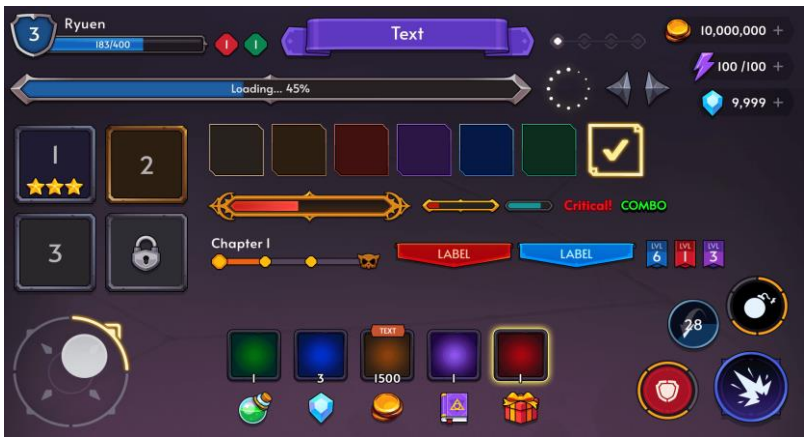


Ilustración 12. Elementos UI del paquete GUI PRO Kit

4.4.1.4. Infinity PBR – Demons

Pequeño paquete de *assets* de la *Unity Asset Store*, el cual cuenta con un modelo de demonio con varios tipos de elementos modulares para personalizarlo, así como una serie de animaciones para sus movimientos y ataques. [12]

De este paquete hemos obtenido un modelo completo para el **rey demonio** de la última escena de juego, así como las **animaciones** que tiene asociadas.

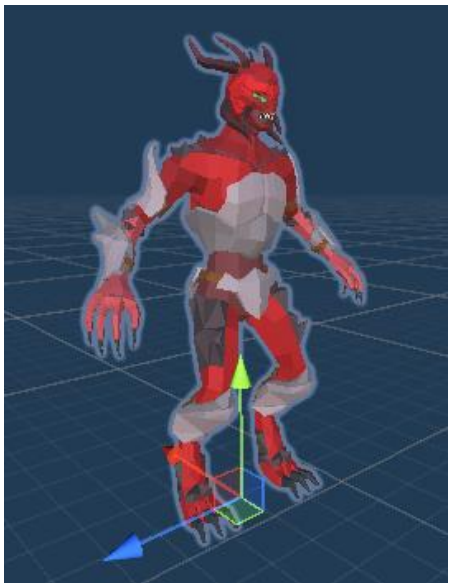


Ilustración 13. Modelo de demonio rojo

4.4.1.5. Low Poly Dungeons

Paquete de la *Unity Asset Store* que incluye una colección de 304 *prefabs* de estilo *low poly* para creación de escenas con temática de mazmorra medieval. [13]

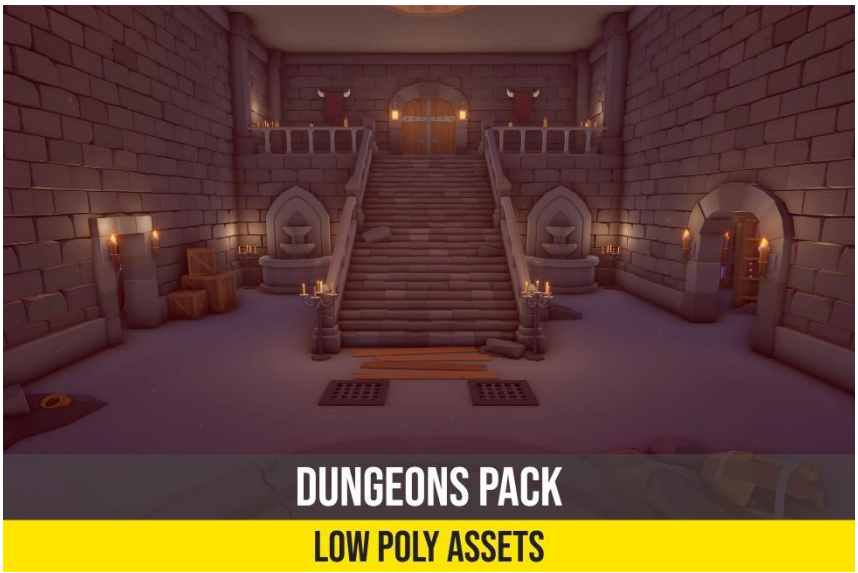


Ilustración 14. Paquete Low Poly Dungeons.

De este paquete se han obtenido casi todos los modelos utilizados para la construcción de las escenas de juego, incluyendo **muros**, **escaleras**, **puertas**, **elementos misceláneos** usados para decorar las habitaciones y mazmorras o **armas** de enemigos y del personaje.

También se han utilizado algunos de los efectos de partículas que incluye el paquete, como el sistema de partículas de **fuego** usado en el nivel de **forja.**

4.4.1.6. Low Poly Terrain – Polaris

Polaris es una herramienta de terraformación que nos permite editar superficies, de manera muy similar a los terrenos incluidos con *Unity*, pero para crear **paisajes de estilo *low poly*.** [14]

Ya que en el escenario inicial de juego se quería diseñar un **terreno exterior**, se ha utilizado esta herramienta para modificar el terreno con montañas y paisajes de estilo *low poly.*

Ilustración 15. Polaris – Low Poly Terrain

4.4.1.7. Low Poly Ultimate Pack

Uno de los paquetes con mayor número de *assets* de estilo *low poly* que hay actualmente en la *Unity Asset Store*, incluyendo tanto **modelos** completamente texturizados de **múltiples tipos** de **estilos** distintos, hasta decenas de ***rigs*** de personajes humanoides completamente preparados para ser animados.

Aunque de este paquete hemos usado algún elemento de **decoración**, como por ejemplo uno de los candelabros, principalmente hemos extraído del cada uno de los ***rigs*** usados para los **enemigos**, incluyendo zombis, esqueletos, vikingo, caballero, mago y ogro. [15]

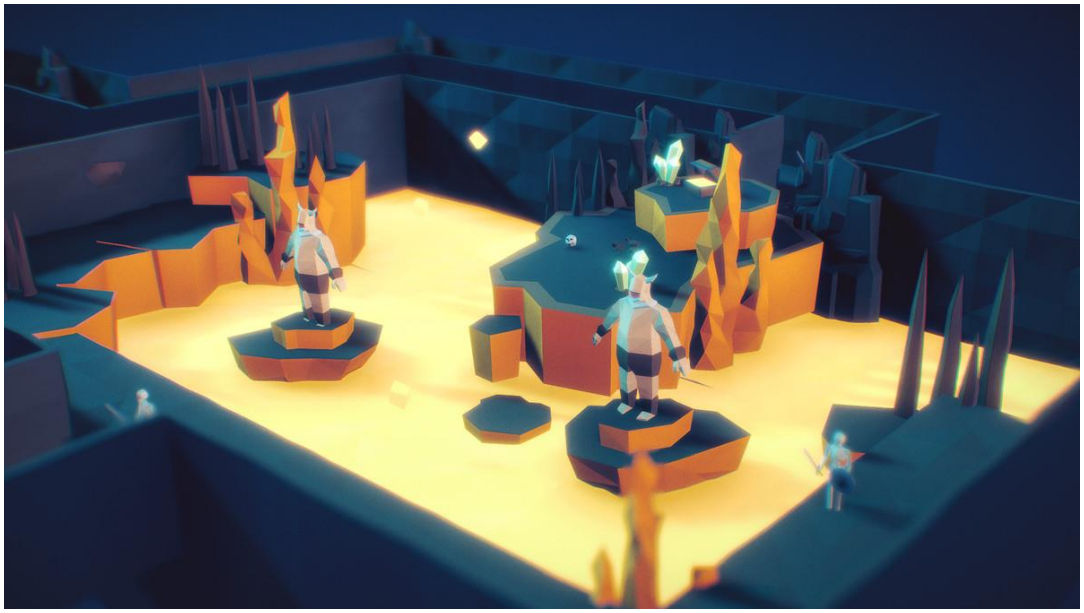


Ilustración 16. Paquete de *assets* Low Poly Ultimate Pack

4.4.1.8. Sonidos

Todos los sonidos usados para este proyecto han sido obtenidos de la página ***freesoung.org*** [16], la cual incluye una amplia colección de sonidos de todo tipo y de uso libre.

En concreto, hemos usado un sonido para cada **ataque** de los enemigos, para cuando **mueren** y sonidos cuando el jugador **recibe daño**, y **ataca** con cada una de sus **armas.**

4.4.1.9. Animaciones

Todas las animaciones usadas en los **enemigos humanoides** han sido obtenidas de la página ***Mixamo*** [17] la cual cuenta con una gran librería de animaciones de uso libre para modelos *rigs* humanoides.

Estas incluyen animaciones para el **movimiento** de los enemigos, acción de **atacar**, **recibir daño** y animación de **muerte.**

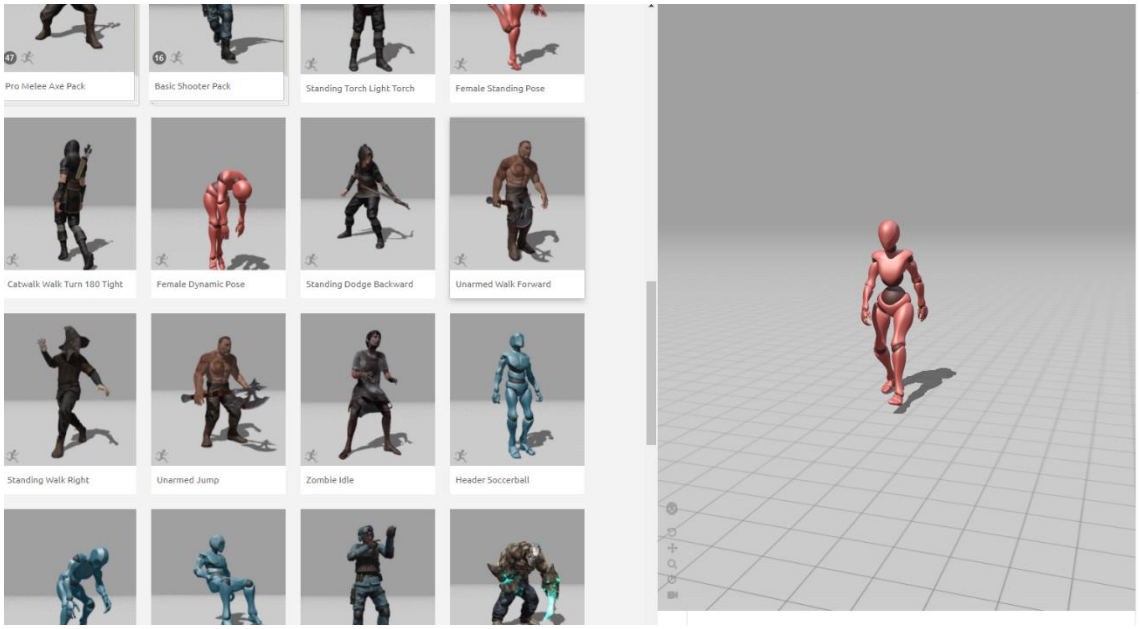


Ilustración 17. Plataforma de animaciones de Mixamo.

4.4.1.10. Sketchfab

Adicionalmente, se han descargado de las página ***Sketchfab*** modelos de uso libre para ciertos elementos de sala que no se encontraban en los paquetes comentados anteriormente, como la **trampa de pinchos** y el **bastón de mago** que usa el jugador. [18]

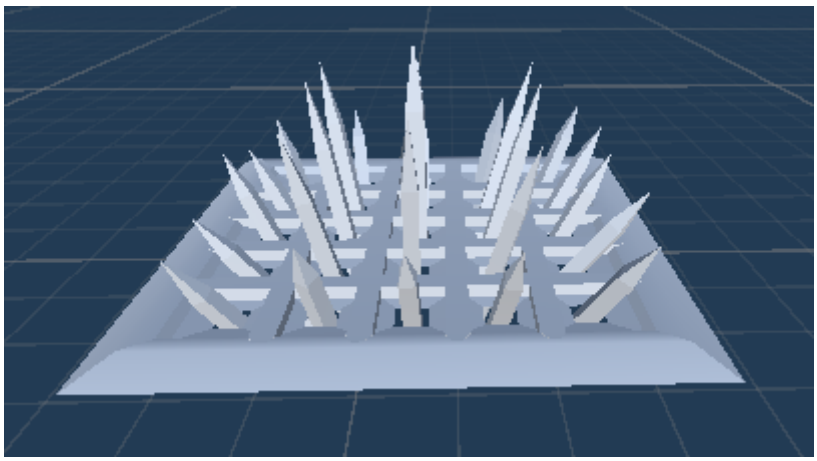


Ilustración 18. Modelo de trampa de pichos

4.4.2. *Assets* de creación propia

4.4.2.1. Modelo del personaje

Para acompañar al personaje en primera persona, se quería añadir el **modelo** de los **brazos** para mejorar la estética de juego. Pero ya que no se encontraba ningún modelo *low poly* que se ajustase a nuestras necesidades, se decidió crearlo desde cero haciendo uso del software ***Blender***, creando también el esqueleto para añadirle animaciones personalizadas.

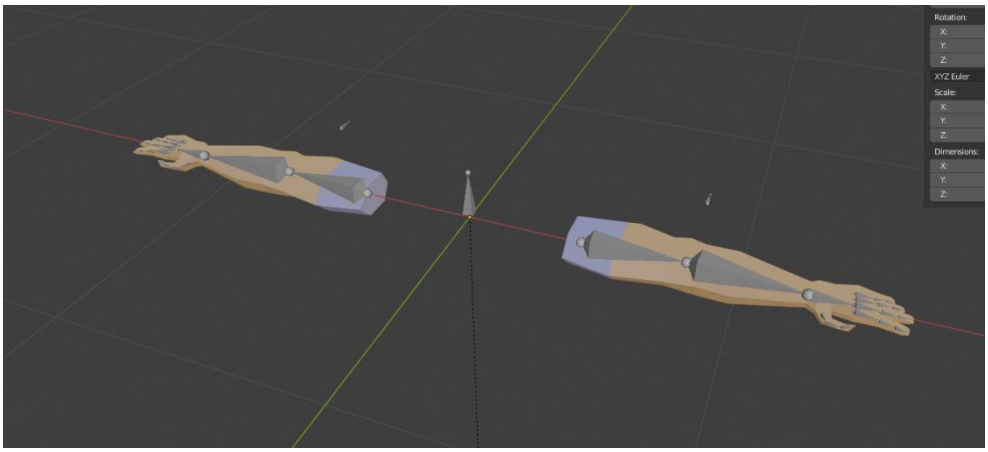


Ilustración 19. Modelo de brazos del jugador

4.4.2.2. Animaciones

Por último, al haber creado el modelo de los brazos desde cero, se necesitaba crear también animaciones de **movimiento** y **ataque** para cada una de las armas utilizadas en el juego. Para realizar esto se ha utilizado también el software ***Blender*** usando el esqueleto de brazos creado para los brazos e importando los modelos de las armas para tener una referencia en su creación.

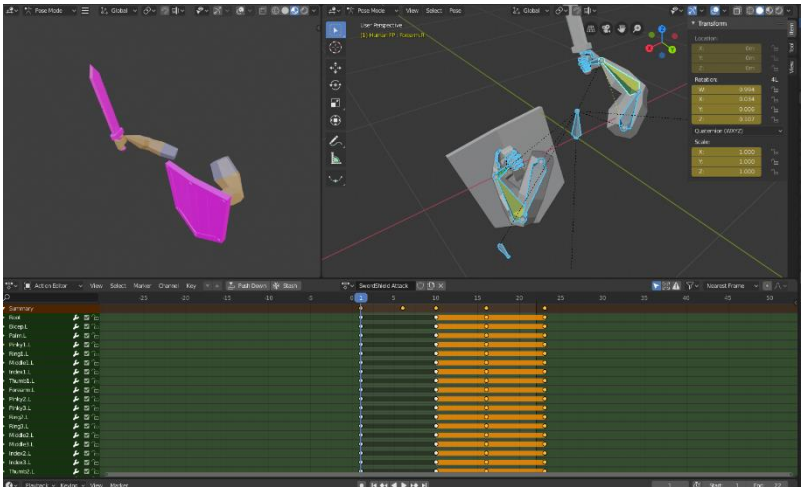


Ilustración 20. Creación de animaciones en Blender

4.5. Esquema de arquitectura del juego y componentes

En este apartado, se intentará hacer un desglose más en detalle de todos y cada uno de los componentes y elementos de juego creados para el juego. Para ello, se hará un recorrido **top-down**, empezando por comentar los elementos de juego más **generales** y amplios hasta llegar a comentar los componentes individuales

4.5.1. Estructura general

Como hemos comentado, la estructura general del juego se basará en una progresión de nuestro personaje por los siguientes tipos de salas:

- **Sala de horda**: En la que, al entrar el jugador, se activarán una serie de portales de horda, de los cuales irán apareciendo oleadas de enemigos. Si el jugador **derrota** a todos los **enemigos** de la horda, se abrirá la puerta de salida, permitiéndole continuar.
- **Sala de puzle**: Habitación sin enemigos, en las que se el jugador deberá superar ciertos **desafíos** de plataformas o **rompecabezas**, al mismo tiempo que va recogiendo **llaves** para abrir la puerta de salida y continuar.
- **Sala de boss**: Habitación final del recorrido del personaje. En ella, se deberá enfrentar a un **poderoso enemigo**, el cual contará con varias **fases de combate** en función de su porcentaje de vida. Al acabar con el *boss* el jugador ganará la partida y terminará el juego.

Cada una de estas salas contará con un script ***manager*** que controla el flujo de juego de cada sala individual, los cuales comentaremos con mayor detalle en apartados posteriores.

Primero, hay que comentar que la comunicación entre salas y entre el funcionamiento de las salas y el jugador se da a través del script "***GameController"***. Este script se asocia a un *GameObject* llamado *"GameManager*" creado en la escena inicial, en el cual se ha especificado en su método *Awake()* que **no se destruya** al **cambiar entre escenas**, manteniéndolo intacto entre salas distintas.

Lo primero que hace el *GameManager* cuando el jugador cambia de escena, es preguntarle al manager de la sala actual cual es el **punto de aparición** del jugador, cambiando la posición de este según corresponda.

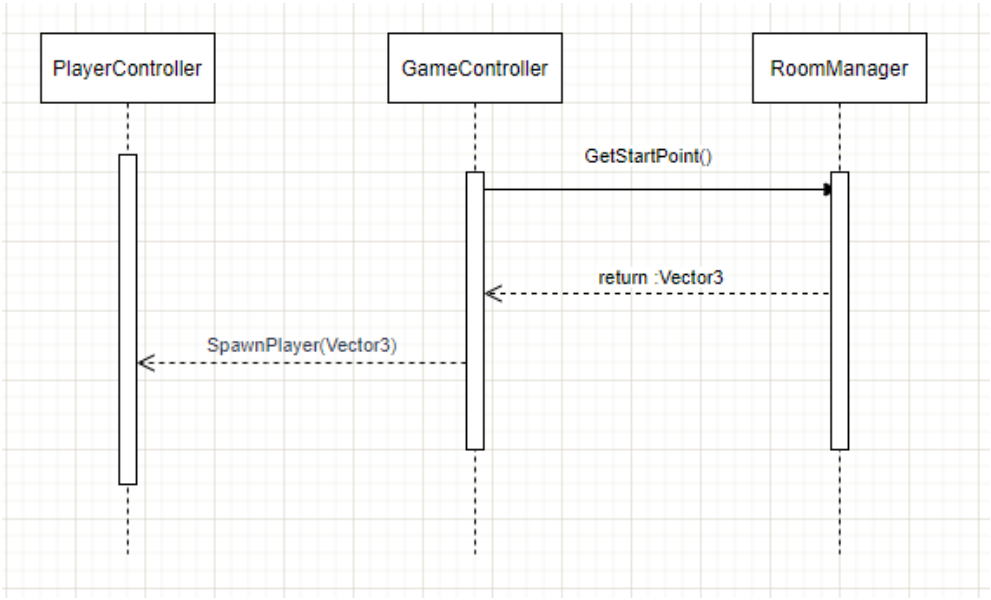


Ilustración 21. Aparición inicial del jugador

Además, si el jugador muere durante la partida, el *GameManager* debe preguntarle al *RoomManager* actual cual es la posición de ***checkpoint***, ya que si una escena cuenta con varias salas, esta posición puede **variar**. El funcionamiento sería similar al caso anterior, pero también le comunicamos al *RoomManager* que debe **reiniciarse**, que por ejemplo en el caso de las habitaciones de horda sería abrir la puerta de entrada y reiniciar los portales de horda.

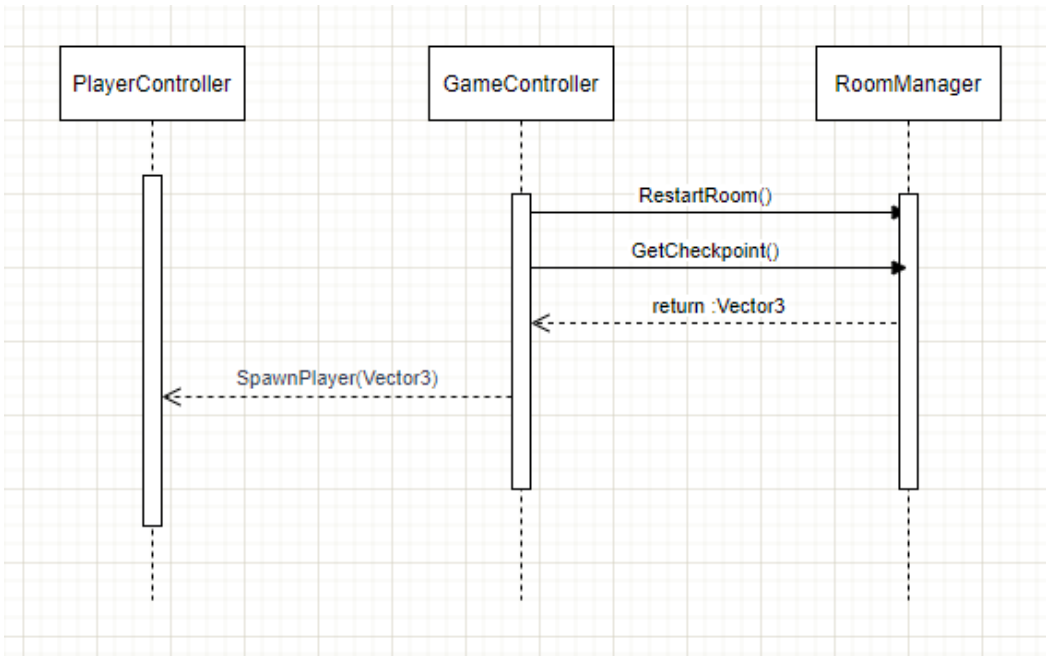


Ilustración 22. Reaparición del personaje y reinicio de sala

Por último, al final de cada escena de juego, tras la última puerta de salida asociada a un tipo de *RoomManager*, habrá un *GameObject* con un script asociado "***NextDungeon"***, que al entrar en contacto con el jugador, comunica al *GameManager* que debe pasar a la siguiente escena

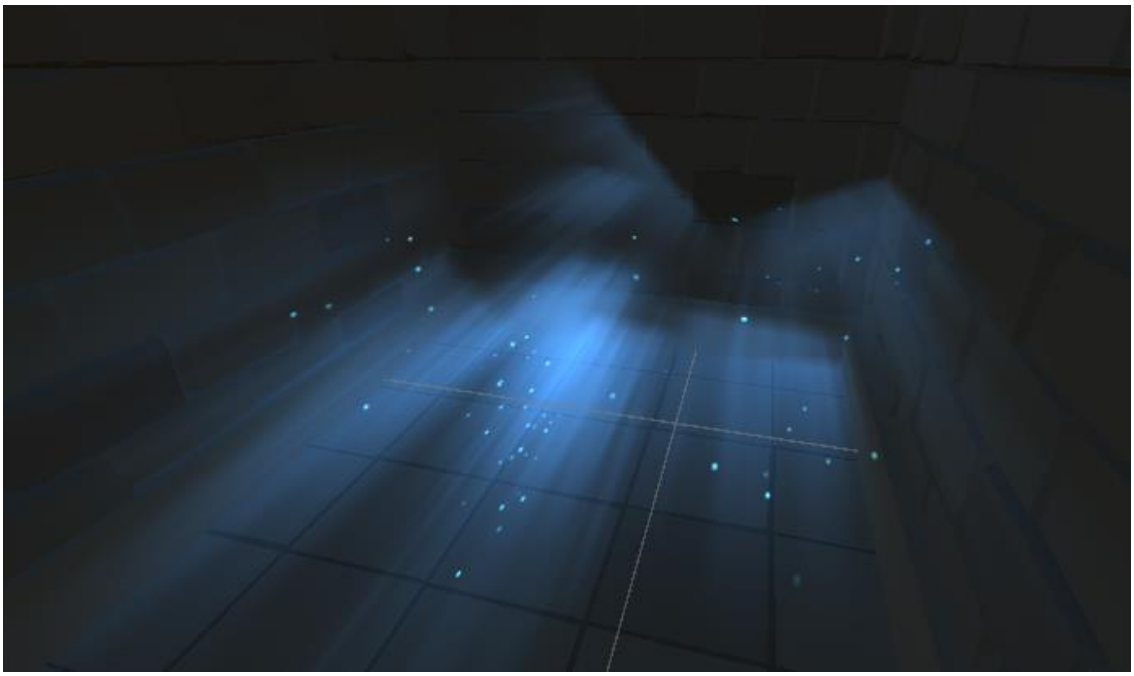


Ilustración 23. Camino a la siguiente escena.

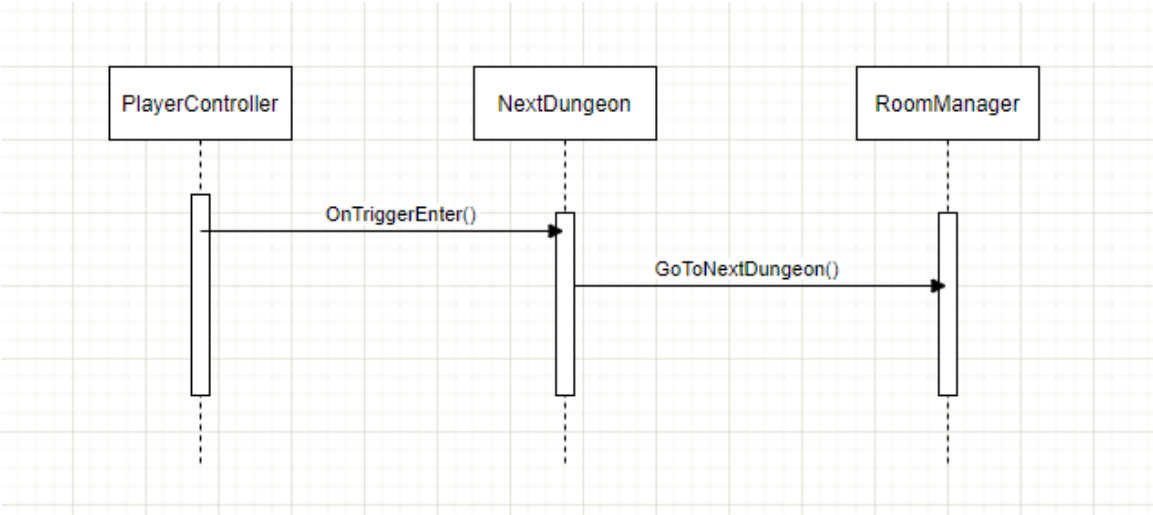


Ilustración 24. Proceso de cambio de escena

Este sería la **capa** de funcionamiento **más externa** del juego, coordinada completamente a partir de este *GameManager*. En el siguiente apartado pasaremos a comentar en detalle el siguiente nivel de estructura de juego: **las salas**.

4.5.2. Estructura de salas de juego

En este apartado vamos a comentar en mayor detalle el funcionamiento específico de cada una de las salas de juego que hemos comentado en el apartado anterior.

Primero, vamos a **detallar** cada una de las **salas** que se han añadido para el prototipo de juego presentado con este proyecto:

Nombre de la escena	Tipo de sala	Detalles
StartingZone	Null	Esta sala representa el principio de juego, por lo que no tiene asociado ningún tipo de RoomManager. Su único comportamiento característico es mostrarle al jugador un panel con los **controles de juego**
Dungeon1	DungeonManager	Primera sala de juego en la que el jugador deberá derrotar **hordas de enemigos**. Se caracteriza por ser una escena de tipo *Dungeon* con dos salas.distintas, cada una son su **propia** horda de enemigos y funcionamiento **separado**.
Puzzle1	PuzzleManager	Sala de puzle en la que el jugador deberá saltar sobre una serie de **plataformas** flotantes sin caer a la lava**,** la cual **matará** al jugador en contacto, al mismo tiempo que coge **llaves** para abrir la puerta final y pasar de escena.
Dungeon2	DungeonManager	Segunda habitación de **mazmorra** en la que habrá que eliminar oleadas de enemigos, introduciendo **nuevos enemigos** con respecto a la primera mazmorra. Además, **solo** contará

		con **una sala** (*RoomManager)* para terminar la partida y continuar.
Puzzle2	PuzzleManager	Última habitación de puzle. En ella, el jugador deberá entrar la **llave** que abre la puerta final haciendo uso de nuevos elementos como **plataformas de salto** y **teletransportes**.
Boss1	BossManager	En esta última escena del juego, el personaje deberá enfrentarse a un único **jefe final**, cuya vida aparecerá en la parte superior de la pantalla cuando entremos a la sala. Al reducir la vida del enemigo por debajo de **ciertos porcentajes**, este realizará una **fase especial de combate** en el centro de la sala, como por ejemplo lanzar magia en todas las direcciones o hacer aparecer enemigos aleatorios.

Sabiendo cómo se ha estructurado nuestro juego en los diferentes tipos de salas, vamos a pasar a continuación a explicar en mayor detalle el **comportamiento específico** de cada una de ellas.

4.5.2.1. Dungeon Manager

Esta primera sala representa la encapsulación de la mecánica principal de juego: **las hordas de enemigos**. En ellas, el jugador deberá acabar con todos los enemigos que aparezcan en oleadas para poder continuar a las siguientes escenas.

Ya que tal y como se ha creado el juego, se da la posibilidad de que en una misma escena de **mazmorra de horda** haya **varias salas**, diferenciamos entre dos controladores distintos, el ***DungeonManager***, que expone el funcionamiento de todas las salas de la escena al *GameManager*, y las diferentes ***RoomManager***, script que representan a cada una de las salas individuales de la escena.

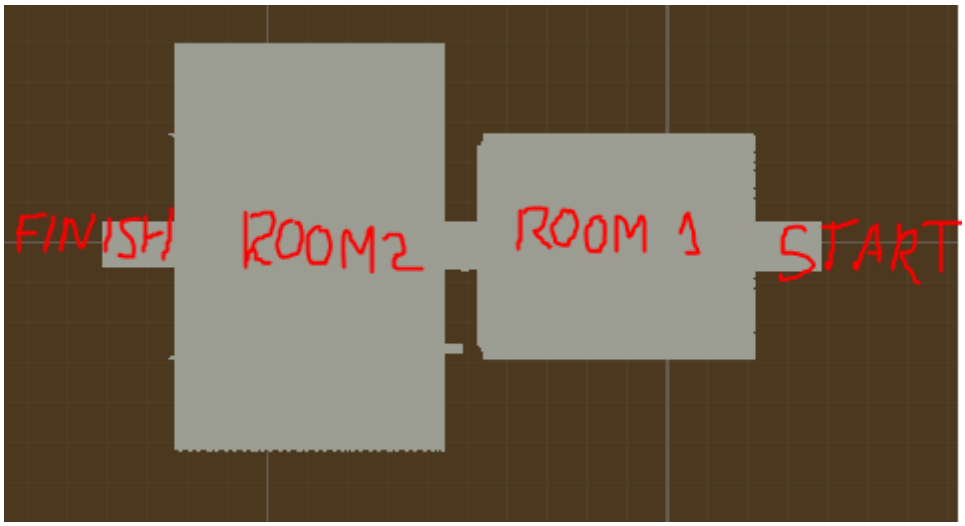


Ilustración 25. DungeonManager con dos RoomManager distintas

El funcionamiento principal del *DungeonManager* es tal y como hemos comentado, exponer al *GameManager* los elementos de juego generales que necesita este para el funcionamiento fluido de juego. Esto incluye guardar en cada momento **el *checkpoint* actual**, debido a que cuando el jugador complete una habitación entera, su checkpoint de aparición se actualizará a esta nueva sala. También se encargará de notificar al *GameManager* la posición de **aparición inicial** del personaje al entrar en la escena.

A continuación tenemos el funcionamiento específico de cada habitación asociada a un script "***RoomManager"***. Cada habitación de este tipo estará formada por los siguientes elementos clave:

- **Puerta de entrada**: Acceso principal a la habitación. Puede cerrarse o abrirse.
- **Puerta de salida**: Camino hacia la siguiente habitación **o escena**. Permanecerá cerrada hasta que completemos la sala.
- **Comienzo de habitación**: Zona de ***collider*** colocada al inicio de una sala, y por la cual debe entrar en contacto siempre el jugador al entrar a la sala. Cuando ocurre esto, le comunica al *RoomManager* que debe **comenzar** la **horda** de enemigos de la sala.

- **Portales de enemigos**: Se encargan de la funcionalidad de **instanciar enemigos** en oleadas mientras dure una oleada. Cuando han terminado de invocar a todas las hordas, desaparecen.

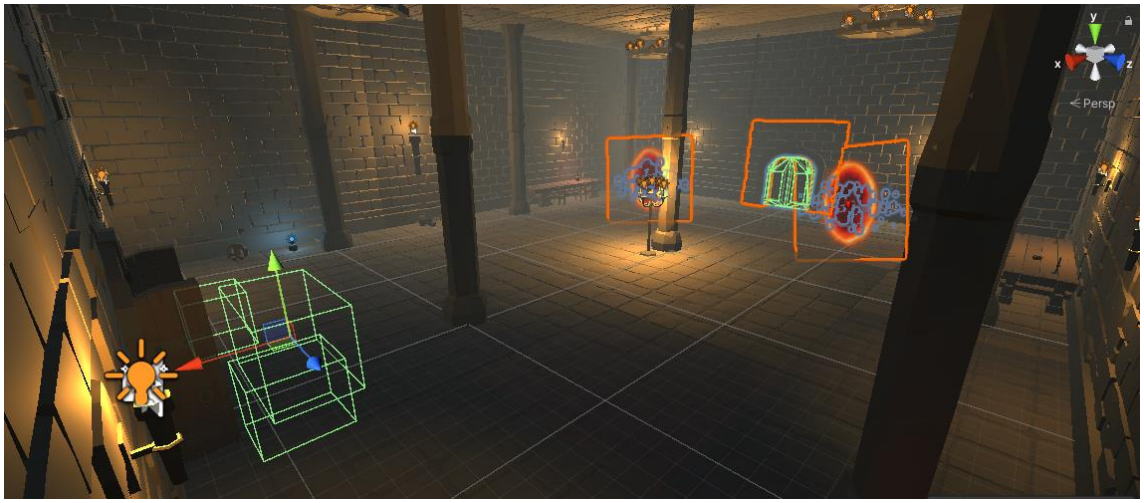


Ilustración 26. Elementos principales de una sala de horda.

Teniendo en cuenta estos elementos, el flujo de funcionamiento de este tipo de salas es el siguiente:

1. El jugador entra en la sala y hace **contacto** con el collider del **comienzo de habitación**. Este "*RoomStart*" le comunica al *RoomManager* que hay que comenzar la **horda de enemigos**.
2. El *RoomManager* **cierra** la **puerta de entrada** para limitar al jugador a la sala actual hasta que acabe con todos los enemigos o muera.
3. Se le comunica desde el manager a cada uno de los **portales de horda** que tiene asociados que debe comenzar a **invocarse** a las **oleadas de enemigos.**
4. Los **portales** comienzan a **invocar** a cada una de las hordas en bucle. Cuando terminan de invocar la **última** de las oleadas, desaparecen.
5. Para cada oleada, almacenamos en una **colección** del *RoomManager* los **enemigos que se han invocado**. Si el jugador ha acabado con todos los enemigos de una oleada, **se invoca la siguiente**.
6. Si llegados a la **última oleada** se detecta que todos los enemigos han sido eliminados se habrá completado la sala, abriendo la puerta inicial y la puerta final para que el jugador pueda continuar con la partida.

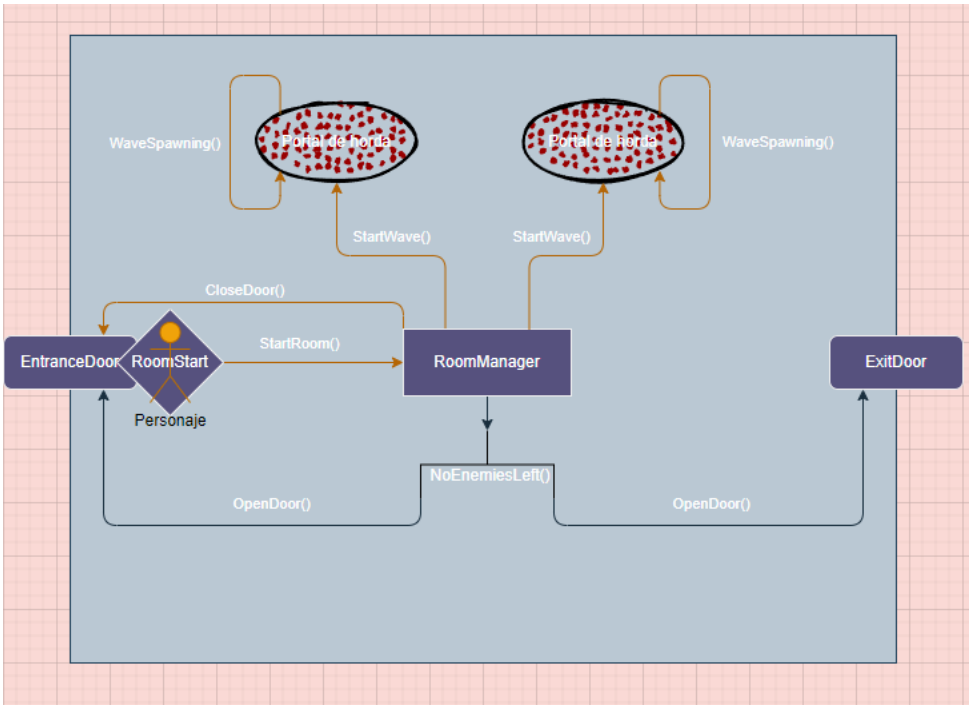


Ilustración 27. Flujo de juego de habitación de horda.

Como hemos comentado, cada portal de horda, representado por el script "*HordePortal*", tendrá asociado cierto número de **oleadas de enemigos**. Cada una de estas oleadas se almacena en una lista de cada portal mediante su entidad "***WaveConfiguration***". Esta clase tiene el propósito de indicar para cada oleada de un portal, cuántos enemigos van a aparecer, y qué **tipo de enemigos** van a aparecer. De esta manera podemos personalizar fácilmente qué enemigos queremos que aparezcan en cada **sala de horda**, además de configurar cuantas oleadas queremos que el jugador tenga que superar **para completar la habitación.**

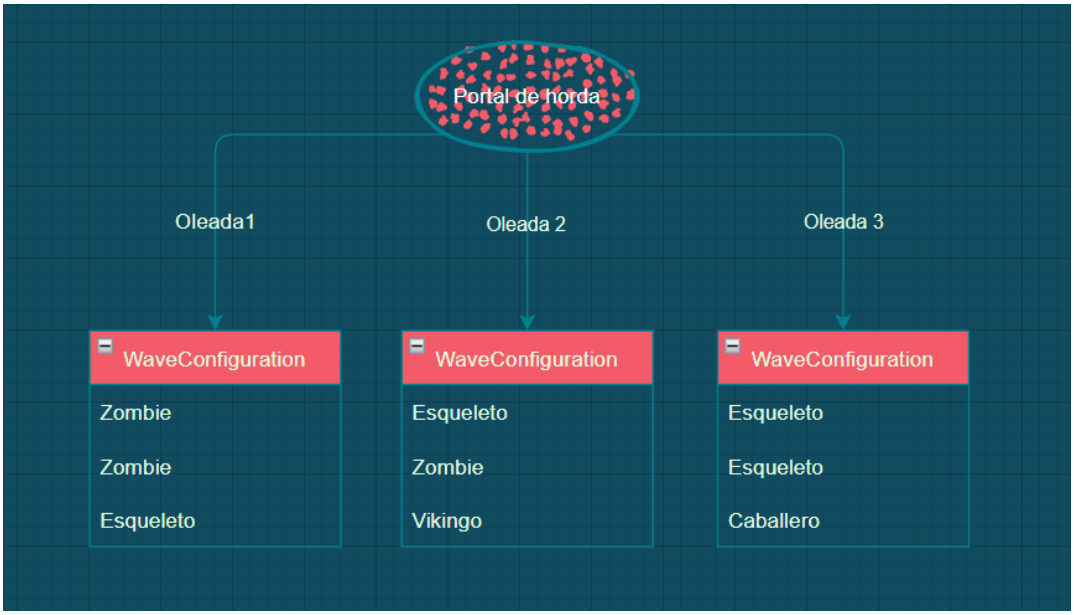


Ilustración 28. Configuración de oleadas de un portal de horda.

Ya que durante el funcionamiento de una sala de horda puede darse la posibilidad de que el jugador muera, debe establecerse un mecanismo para que, además de hacer reaparecer al jugador en el último *checkpoint* tal y como hemos comentado anteriormente, se debe **reiniciar la sala actual**, incluyendo cada una de las oleadas de enemigos.

Para ello, cuando se detecta desde el *GameManager* que el jugador ha muerto, este se lo comunica al *DungeonManager*, que a su vez accederá al *RoomManager* de la sala actual para indicarle que **reinicie las oleadas** de enemigos. Además, ya que hemos ido guardando en una colección los enemigos que se han instanciado en cada oleada, podemos eliminar los enemigos que estaban presenten en la escena cuando ha muerto el usuario.

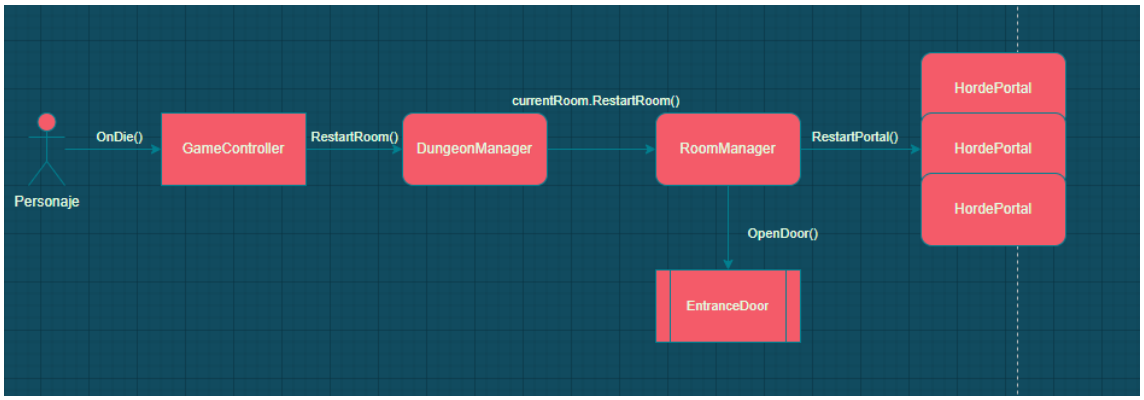


Ilustración 29. Reinicio de una sala de horda.

4.5.2.2. Puzzle Manager

La tipo de sala que se ha creado a modo de **descanso** entre salas de horda, las cuales pueden cansar al jugador si se presentan de manera continua una detrás de otra, son las salas de puzle, habitaciones con pequeños rompecabezas o desafíos de plataformas a resolver hasta llegar a la siguiente sala.

Ilustración 30. Sala de puzle con plataformas flotantes.

El funcionamiento de estos niveles de puzle es simple:

- No hay **puerta de entrada**, ya que no es necesario limitar al jugador a la sala actual.
- La **puerta de salida** estará cerrada inicialmente hasta que el jugador obtenga todas las **llaves** que hay en la habitación.
- Obtenidas las llaves, el jugador simplemente tendrá que llegar a la puerta de salida para continuar a la siguiente sala.

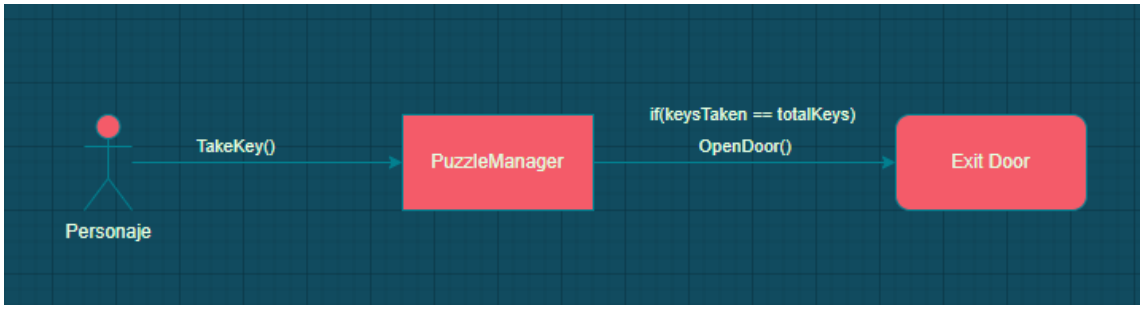


Ilustración 31. Flujo sencillo de un nivel de puzle.

Por lo tanto, la única dificultad que se le propone al jugador en estos niveles es alcanzar todas las llaves y llegar a la puerta de salida, haciendo uso del sistema de movimiento y de los elementos de sala que puedan ser necesarios.

Como añadido a la dificultad, se añaden zonas de muerte, como **pozos de lava**, en los que el jugador morirá al entrar caer sobre ellas. Si el jugador muere en estos niveles, simplemente aparecerá en la posición inicial guardada en el *PuzzleManager*, el cual se la comunicará al *GameManager.*

4.5.2.3. Boss Manager

Finalmente comentados la habitación de jefe final, la cual repasaremos brevemente ya que la funcionalidad más específica de esta sala se concentra en la IA del boss, mientras que el funcionamiento del *BossManager* es muy parecido al que ya hemos comentado para el *DungeonManager.*

Al entrar en la sala, el jugador hará contacto con el *collider* del *GameObject* ***"RoomStart"*****,** lo que dará comienzo al funcionamiento de la sala siguiendo estos pasos:

1. Se **cerrará** la puerta de entrada limitando al jugador a la sala actual.
2. Se activará la interfaz de juego **barra de vida** del boss en la parte superior de la pantalla.
3. Le indicaremos al boss, el cual permanecerá colocado en la escena desde un principio, que **comienza la pelea**, indicándole que su objetivo es el jugador y debe disponerse a atacarlo.

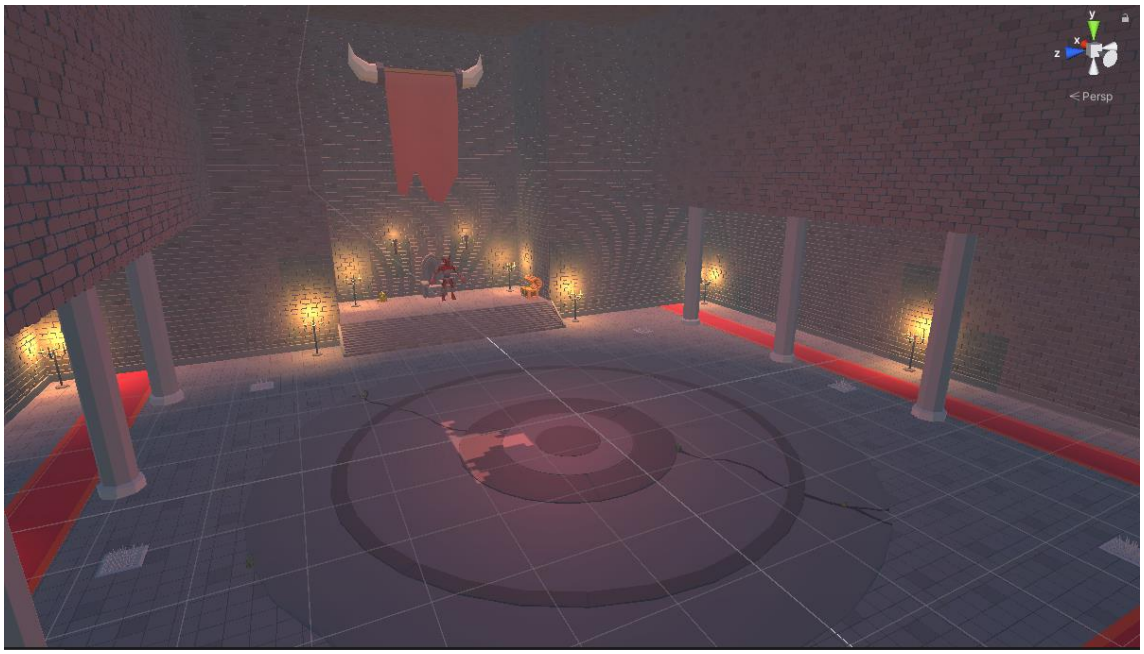


Ilustración 32. Sala de boss

Otro de los añadidos característicos de esta sala en comparación con la sala de horda, es el hecho de que cuando la pelea está en curso, irán apareciendo **pociones** de maná y vida de manera aleatoria entre una serie de posiciones colocadas a lo largo del escenario. De esta manera, al ser una pelea larga, el jugador tiene mayor posibilidad de salir victorioso.

Finalmente, si el jugador consigue derrotar al boss **decrementando su vida a 0**, tras un cierto tiempo de *delay* el *GameManager* dará por finalizada la partida cambiando a la **escena de final de juego**.

El resto de funcionalidad específica de esta sala corresponde a la IA del boss, por lo que la explicaremos en mayor detalle en los apartados siguientes.

4.5.3. Jugador principal

A la hora de comentar los elementos más individuales de nuestro proyecto vamos a comenzar por el personaje, elemento principal de interacción del jugador con el videojuego.

Este personaje estará representado mediante un *CharacterController*, componente de *Unity* que nos permitirá moverlo con facilidad por los escenario, además de controlar la colisión con otros objetos, el suelo o indicar la altura de escalones de escalera que puede subir. A continuación comentaremos brevemente los elementos más importantes de funcionamiento del jugador.

Ilustración 33. Modelo del personaje

4.5.3.1. Cámara

Como hemos comentado en el diseño del juego, la cámara del personaje será completamente en **primera persona**, lo que hará que el jugador experimente los escenarios desde la perspectiva de los ojos del personaje.

Para conseguir este efecto, se ha colocado un componente de tipo *Camera* en la jerarquía de *gameobject* de nuestro persona, asegurándonos de colocarla a suficiente altura por encima del suelo para dar el efecto de vista en primera persona.

Ya que el estilo del juego es combate de acción en primera persona, necesitábamos un control de la cámara rápido y libre, asociado a un script "*MouseLook*", que nos permite controlar la rotación de la cámara en función del movimiento que introduzca el jugador mediante los *axis* vertical y horizontal del ratón, cuya velocidad de rotación se controla mediante una variable de sensibilidad.

Además, se ha limitado los ángulos de rotación vertical para que el jugador no pueda dar una vuelta de 360 grados hacia arriba o hacia abajo.

4.5.3.2. Movimiento

Otro de los elemento más importantes del jugador es el movimiento, ya que como hemos comentado queremos que este sea muy fluido, sobre todo en combinación con el combate con enemigos y al moverse por los niveles de plataformas.

Esta funcionalidad se da en el script *PlayerController*, donde moveremos al jugador en función de su vector forward, por lo tanto controlado por la dirección de la cámara, y según el *input* del jugador con los axis horizontal y vertical. También podremos realizar un *sprint* manteniendo pulsada la tecla *shift* del teclado.

Por otro lado, se controla también la posibilidad de saltar pulsando la barra espaciadora, movimiento que solo se podrá realizar cuando el jugador esté parado sobre una superficie. El jugador mantendrá cierta inercia de la velocidad que tenga antes de hacer el salto, por lo que si haces un salto mientras esprintamos conseguiremos llegar más lejos que simplemente caminando.

Este script controlador servirá también para controlar otras funcionalidades básicas del jugador, como la muerte y reaparición del personaje, o el efecto que tiene en el movimiento del jugador elementos de sala como la plataforma de salto

4.5.3.3. Recursos y recibir daño

Un elemento muy importante sobre la funcionalidad son los recursos de los actores de juego. En el jugador tenemos dos tipos de recursos distintos: salud y maná.

El primero de estos recursos es la vida, representada por el script *"Health",* componente que tendrá asociado cualquier personaje o enemigo de juego que pueda perder vida en combate y morir. Si este recurso llega a 0, el personaje morirá, teniendo que reaparecer en el último *checkpoint*.

Todo actor de juego que pueda ser dañado, como pasa con este personaje principal, tendrá asociado un *gameobject* formado por un *collider* y un componente de tipo "*Damageable*". Ese *collider* será el que "recibirá" el impacto de ataques que reciba el persona u otro enemigo, comunicándose al script

asociado, que a su vez se encargará de reducir la vida, hasta que finalmente llegue a cero y muera.

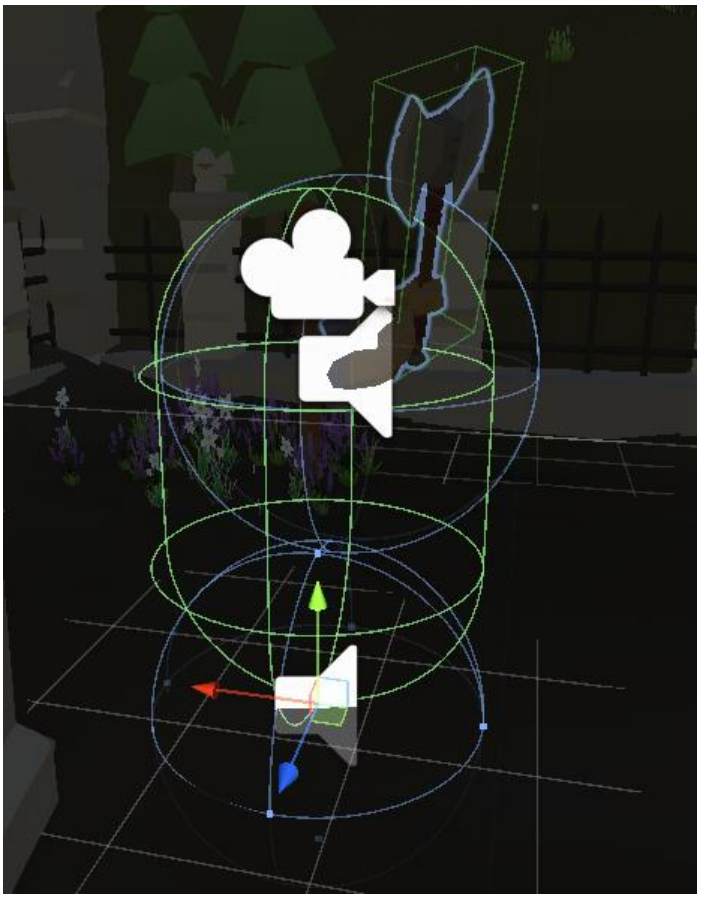


Ilustración 34. Collider del *CharacterController*.

El segundo recurso importante por comentar para el personaje principal es el maná, el cual si es exclusivo para este actor de juego. Su funcionamiento es exactamente igual que la salud del persona, solo que su decremento se producirá cuando el jugador realice un ataque mágico, los cuales tendrán cada uno un coste especifico de este recurso.

Si el maná del jugador llega a cero, no podrá realizar ningún ataque mágico. El propósito de haber introducido este recurso para el jugador es entonces **limitar** su uso de habilidades mágicas, las cuales pueden ser consideradas mucho más fuertes que los ataques cuerpo a cuerpo, sobre todo por la seguridad que proporciona atacar a distancia.

4.5.3.4. Sistema de combate

Finalmente, comentamos el sistema de combate del personaje, controlado por el usuario a partir del script "*PlayerCombat"* asociado al *GameObject* general del personaje.

En primer lugar, comentamos que se han creado los siguientes tipos de armas para el jugador:

- **Espada y escudo:** Permite al jugador atacar a corta distancia, pudiendo dañar únicamente a un solo enemigo por ataque
- **Hacha de dos manos:** Arma de gran tamaño que hace un ataque de barrido horizontal para atacar a varios enemigos al mismo tiempo.
- **Bastón mágico:** Usado para canalizar dos tipos de ataques mágicos, bola de fuego y carámbano.

Unas de las funcionalidades que da este script de control del sistema de combate, es la posibilidad de cambiar de arma utilizando la rueda del ratón o los números del teclado. Para conseguir cambiar fácilmente entre armas, se ha decidido por crear un *prefab* para cada tipo de arma, incluyendo el modelo de los brazos y del arma. De esta manera, guardamos en el script una referencia a cada uno de estos prefabs, pudiendo cambiar entre ellos fácilmente desactivando o activando su *gameobject*.

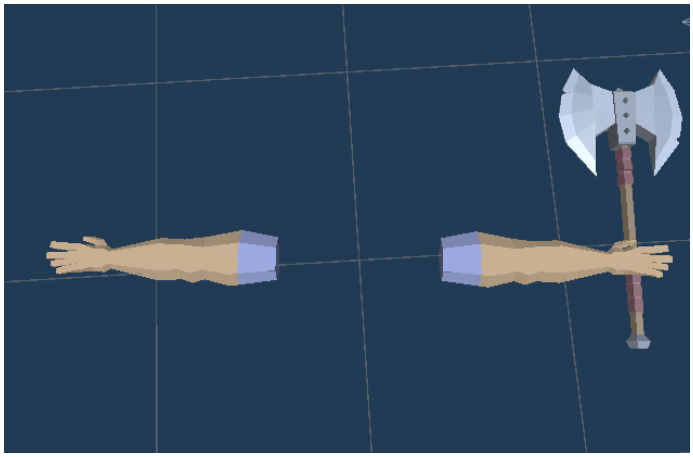


Ilustración 35. Modelo para uso de hacha de dos manos.

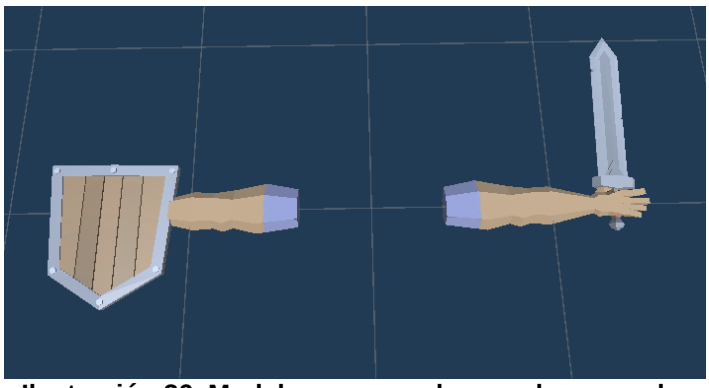


Ilustración 36. Modelo para uso de espada y escudo.

Por lo tanto, la funcionalidad especifica de cada tipo de arma individual estará encapsulada en su *gameobject.* Por ejemplo, para cada *gameobject* de arma tendremos asociado un componente *Animator* en el que indicaremos las animaciones que hemos creado para nuestro jugador en primera persona, incluyendo animaciones de movimiento en función de la velocidad del jugador o animación de uso del arma.

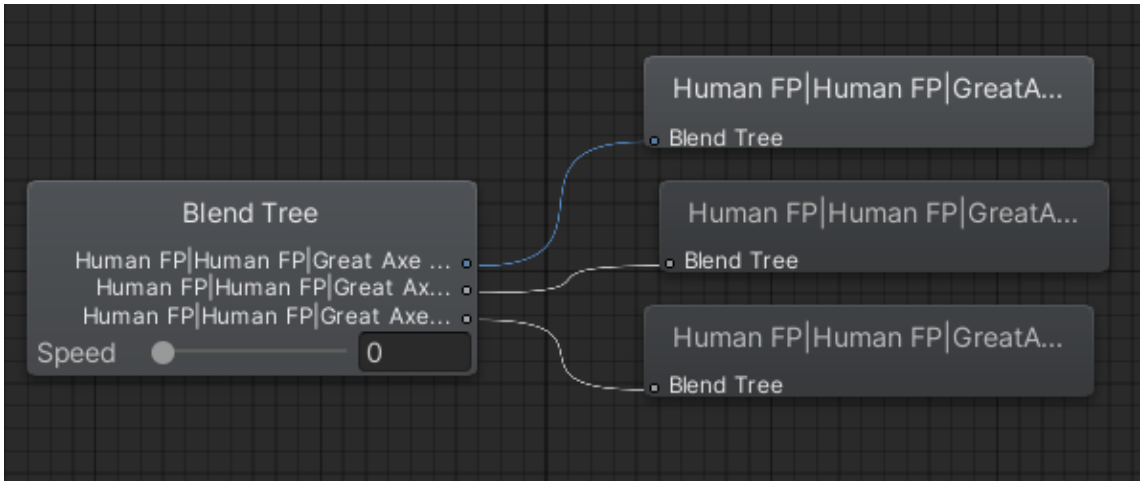


Ilustración 37. BlendTree de animaciones de movimiento de hacha a dos manos.

Por otro lado, tenemos la lógica asociada a los ataques de cada tipo de arma. Estos ataques se iniciarán desde el script "*PlayerCombat"* a partir del *input* del jugador para los clic izquierdo y derecho del ratón. Una vez se ha pulsado uno de estos botones, se comprobará qué tipo de arma tiene equipada ahora mismo el jugador, usando para ellos una referencia al script "*PlayerWeapon"* actual, clase de la que heredan los tres tipos de armas para implementar su funcionalidad concreta.

Lo que se hace en ese momento, en función del clic del ratón que se haya pulsado, es reproducir la **animación** de la acción primaria o de la acción secundaria. En nuestro caso, únicamente el bastón mágico tiene implementada una acción secundaria.

Entonces, lo que se quería es que la acción específica de cada arma estuviese perfectamente sincronizada con su animación, como por ejemplo, que la espada haga daño únicamente en los *frames* de animación en los que se encuentra hacia delante. Esto se ha conseguido introduciendo un *Animation Event* específico en la animación de ataque de cada arma, activando su funcionalidad en el momento justo mediante un método *StartAction()*. De la misma forma, usamos otro *Animation Event* para indicar, en el caso de la espada y el hacha de dos manos, cuando deben dejar de hacer daño.

A continuación, comentamos brevemente la funcionalidad específica de cada acción de combate realizada por cada tipo de arma:

- **Espada de una mano**. Cuando se inicia la acción de combate de la espada, se activa un *collider* asociado al modelo de la espada, el cual, al impactar con un enemigo, le hará daño. Para que solo haga daño a un enemigo, se controla que en una misma acción del arma, si ya se ha golpeado a alguien, no pueda volver a hacer daño.
- **Hacha de dos manos**. Al igual que para la espada de una mano, se activa un *collider* asociado al hacha mientras está activa la acción. A diferencia de la espada de una mano, como se quiere poder atacar a varios enemigos en un mismo barrido del arma, se ha introducido una colección de aquellos enemigos que se ha golpeado en el ataque actual.
  Esto nos permite poder atacar a todos los enemigos que golpea, pero comprobando que a cada enemigo solo se le haya golpeado una vez en este ataque.
- **Bastón mágico**. Para este último arma, cuando se inicia la acción de combate, se lanza un proyectil concreto en función de si se ha pulsado la acción primaria o la acción secundaria:
  - **Bola de fuego (acción primaria)**. Proyectil de baja velocidad y con caída mediante el efecto de la gravedad que, al golpear con un objeto, explota provocando daño a todos los enemigos que haya en su rango de efecto.
  - **Carámbano de hielo**. Proyectil de mayor velocidad que se desplaza en un recorrido rectilíneo sin caída. Al golpear con un objeto, se rompe en varios trozos con el mismo comportamiento, los cuales se instancian en direcciones en un círculo en torno al carámbano original. Si los trozos golpean otra superficie volverán a propagarse en más trozos.

Una diferencia fundamental entre las armas cuerpo a cuerpo y el bastón mágico, es que este último utiliza maná como recurso. Tanto la bola de fuego como el carámbano de hielo tendrán un coste específico de maná para poder ser utilizados, por lo que si el maná del jugador no llega a los niveles requeridos, no se podrá utiliza la magia.

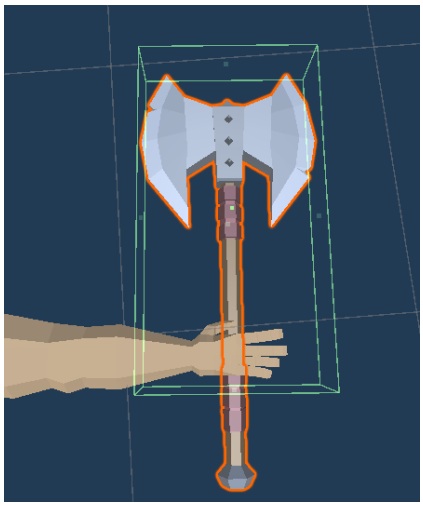


Ilustración 38. *Collider* de impacto del hacha de dos manos.

4.5.4. Enemigos de horda

Pasamos a continuación a comentar a los actores de juego contra los que tendrá que enfrentarse nuestro jugador en las salas de horda. Daremos primero una pequeña descripción de los componentes que los forman y posteriormente explicaremos la inteligencia artificial que tienen asociada.

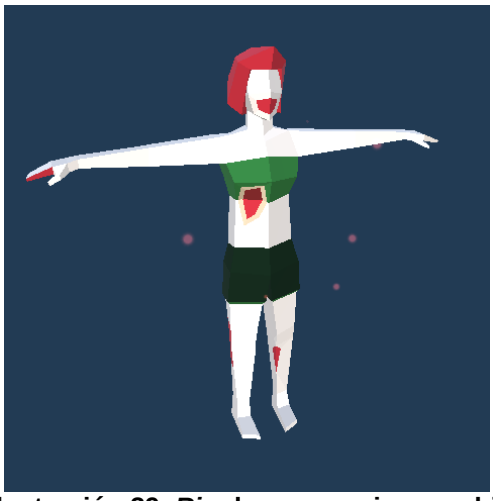


Ilustración 39. *Rig* de un enemigo zombi.

Ya que estos enemigos son controlados completamente mediante inteligencia artificial, llevarán asociado un componente *NavMeshAgent*, el cual se encargará de controlar su movimiento por la escena. Además, en este componente podremos indicar parámetros como la velocidad de movimiento, el radio o la altura del agente. Para que estos enemigos puedan moverse por la escena haciendo uso de *pathfinding* habremos generado previamente un *NavMeshSurface*, indicando por qué zonas pueden moverse y por cuáles no.

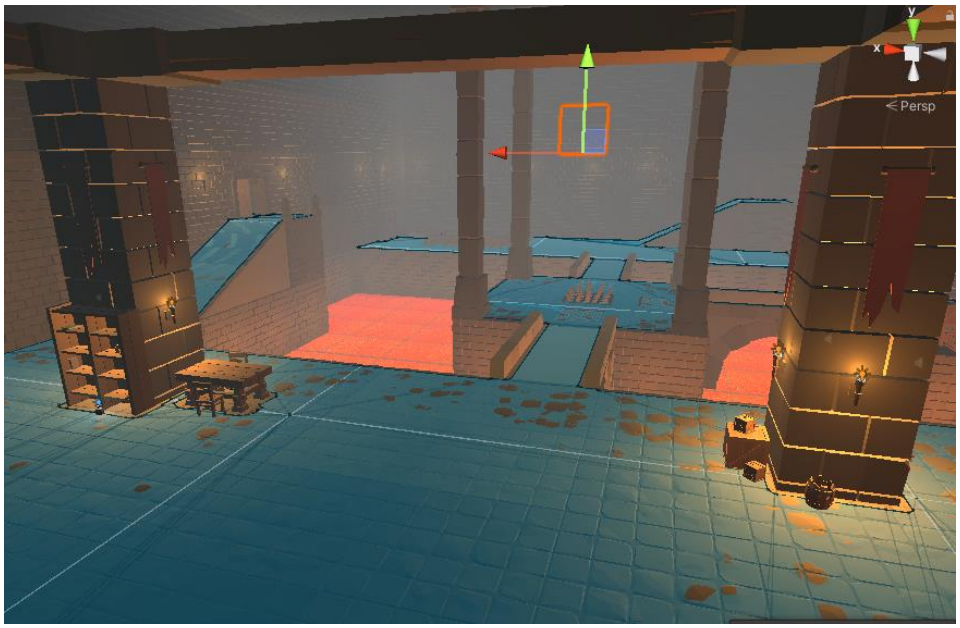


Ilustración 40. NavMeshArea de una escena,

4.5.4.1. Controlador de enemigo.

Todos los enemigos tendrán también asignado un script controlador *EnemyController*, donde se encapsula la mayor parte de su funcionalidad básica. Esto incluye actualización de la dirección de movimiento, en qué dirección debe rotar, recibir daño o el comportamiento específico de muerte, además del tratamiento de las animaciones.

Por otro lado, se implementa en este script los diferentes tipos de ataques que pueden realizar los enemigos. Al igual que para el sistema de combate del jugador, para sincronizar la animación de ataque a la perfección con el momento en el que deben realizar daño, se ha incluido en cada animación de ataque un *Animation Event* llamado EnemyAttack(), el cual indica el momento justo de la animación en la que deben ejecutarse los ataques.

Los ataques implementados para los enemigos se resumen en estos dos tipo:

- **Ataque a corta distancia**: Este será el ataque el que utilicen por defecto la gran mayoría de enemigos del juego. En el momento de atacar, se realizará un *SphereCast* hacia delante con un ratio proporcional a la distancia de ataque de cada enemigo. Si este *cast* detecta a nuestro jugador, le hará daño.
- **Ataque mágico**: Principalmente usado por los acólitos enemigos. Al realizarse el ataque, se instanciará un proyectil mágico desde la posición de la mano del enemigo con la que realiza el ataque, y en dirección al jugador objetivo. El proyectil lanzado seguirá una dirección rectilínea hasta que golpee un objeto o se disipe.

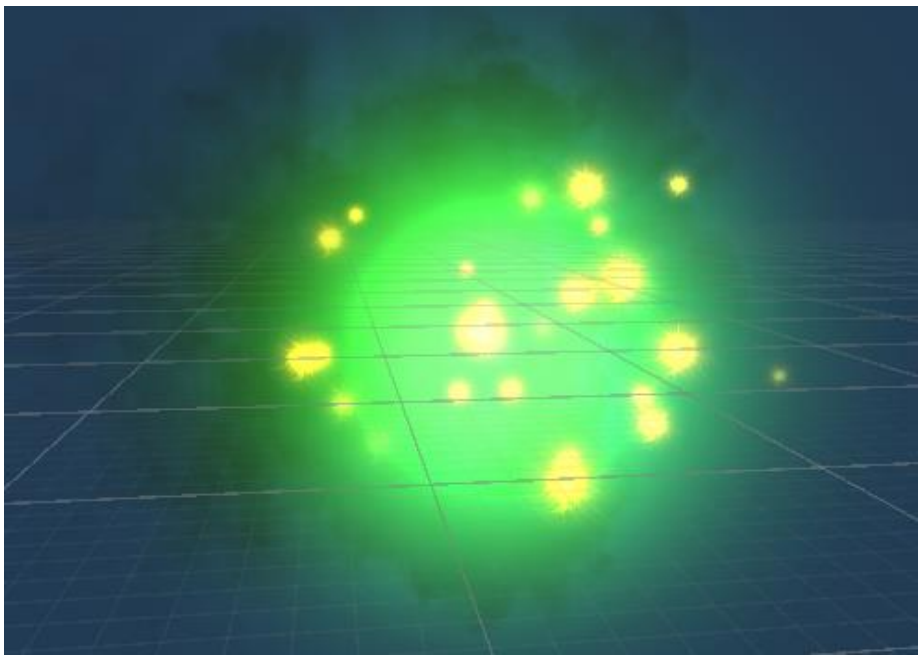


Ilustración 41. Proyectil mágico enemigo.

4.5.4.2. Vida del enemigo

Como hemos comentado anteriormente, todos los actores de juego utilizan el mismo script *Health* para almacenar la salud actual y detectar cuando ha muerto. De igual manera, se ha creado un *gameobject* en los enemigos que actúa como *hitbox* a la hora de ser golpeados por el jugador u otros elementos de juego que puedan dañarlos. Este elemento tendrá asociado el mismo script *Damageable* que el jugador, desde donde se detectarán los golpes recibidos para decrementar la salud o reproducir alguna animación al ser dañado.

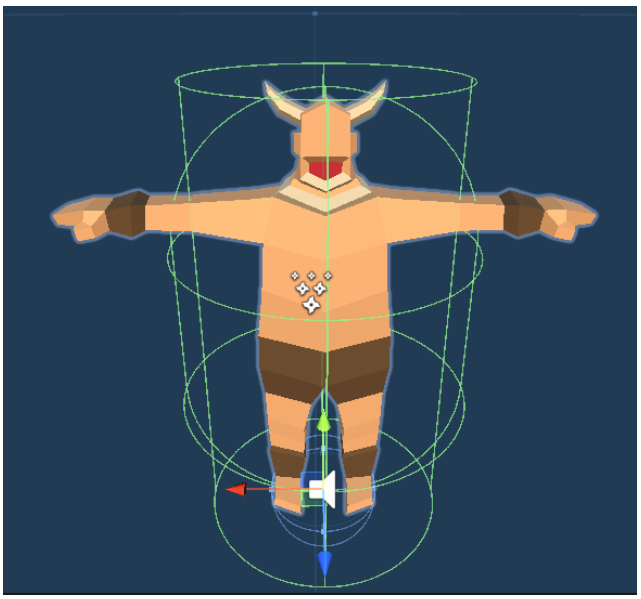


Ilustración 42. Hitbox de un enemigo.

4.5.4.3. Animaciones

Con respecto a las animaciones de los enemigos, debemos comentar brevemente cierta comportamiento específico que se ha añadido para mejorar su funcionamiento.

Ya que queríamos que el modelo de los personajes pudiese desplazarse y atacar al mismo tiempo, se han utilizado varias *layer* en los *Animator* de los enemigos, una para el movimiento, otra para atacar y una última para recibir un ataque.

Esto nos ha permitido aplicar una máscara de avatar en las capas de atacar y recibir daño para que únicamente se muevan las **partes superiores** del cuerpo, lo que permite combinar el movimiento de los enemigos con atacar o recibir daño, ya que cuando se realiza por ejemplo una animación de ataque, se moverá únicamente la parte superior del cuerpo, mientras que las piernas seguirán utilizando la animación de movimiento.

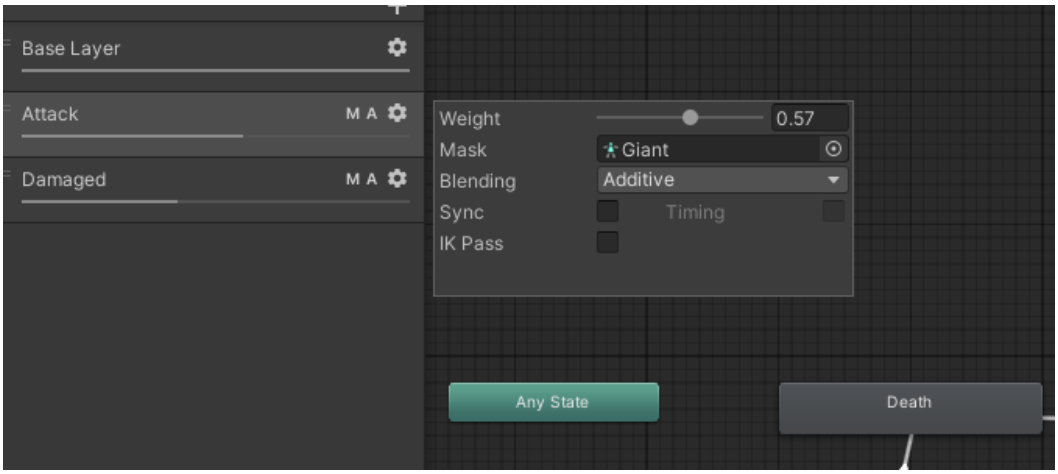


Ilustración 43. Capas de animación de los enemigos.

4.5.4.4. Tipos de enemigos

Finalmente, haremos un desglose de todos los tipos de enemigos implementados y sus características especiales:

Tipo	*Daño*	*Vida*	*Rango*	*Velocidad*
Cultist	10	60	40	3
Giant	30	100	6	1
Knight	10	70	4	2
Skeleton	2	30	4	6
Viking	15	50	4	2
Zombie male	5	30	4	7
Zombie female	5	30	4	8
Demon Boss	5	500	10	4

4.5.5. Jefe final

Entre los tipos de enemigos que se han implementado, el jefe demonio final es el más característico, y del cual vamos a explicar en mayor detalle algunas de sus fases de combate.

Al igual que para los enemigos normales cuerpo a cuerpo, el *boss* intentará acercarse al jugador para atacarlo físicamente. Pero para hacer más interesante el combate, se han añadido un par de fases especiales, las cuales se activan en función del porcentaje de vida restante del jefe. Comentamos estas dos fases a continuación:

- **Fase especial 1:** En esta fase, el enemigo, tras moverse al centro de la sala, comenzará a canalizar proyectiles mágicos en un círculo en torno a si mismo. Para conseguir esto, se instancian los proyectiles según su índice, teniendo en cuenta que cada canalización instancia 32 bolas mágicas en torno al enemigo, haciendo uso de los radianes de una circunferencia dividido entre el número de proyectiles que se van a instanciar.
  Para decrementar el efecto del hueco libre que queda entre proyectiles, en los cuales el jugador se puede quedar parado para esquivarlos, en cada bucle de canalización se aplica cierto *offset* de rotación en la posición de instanciación de las bolas mágicas.
- **Fase especial 2:** Funciona de manera similar a la fase anterior, pero en vez de instanciar proyectiles, se instanciarán **enemigos** con IA, los cuales perseguirán al jugador e intentarán atacarlo.

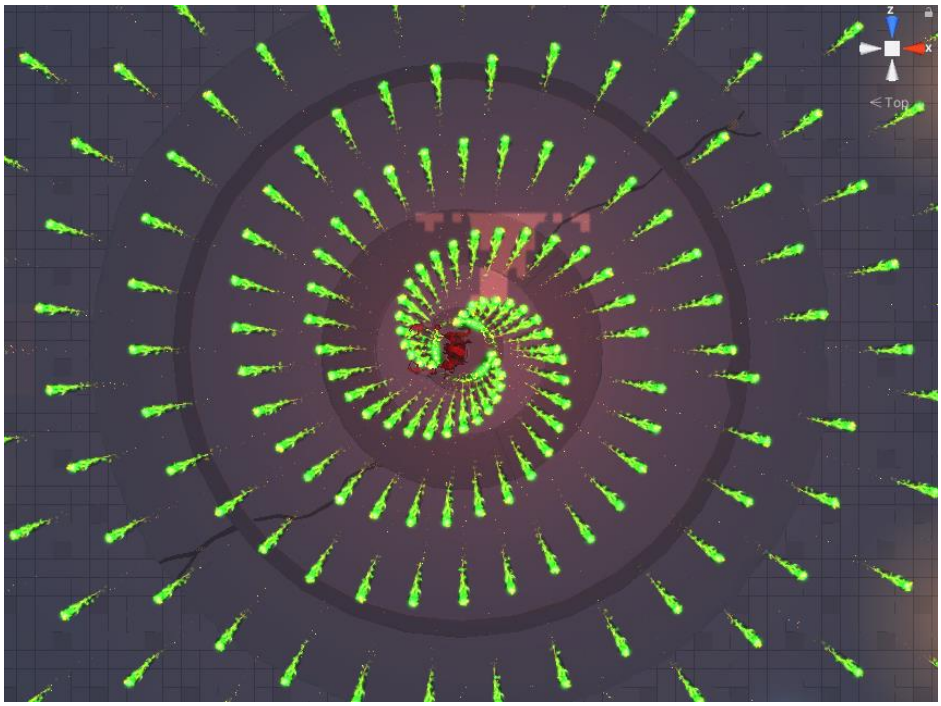


Ilustración 44. Vista aérea de la primera fase especial del boss final.

4.5.6. Elementos de sala.

En este apartado comentaremos el funcionamiento del resto de elementos que podemos ver en la sala con funcionalidad propia.

4.5.6.1. Teletransportes

Elementos de sala utilizados para transportar rápidamente al jugador entre dos puntos separados del mapa.

Su funcionamiento es simple. Si el jugador entra dentro del *collider* de uno de ellos, se moverá instantáneamente a la posición del teletransportador pareja.

Para evitar que se transporte continuamente entre ellos al estar constantemente dentro de sus *colliders*, cuando un jugador se teletransporta, se activará una variable de *cooldown* que impedirá que vuelva a usarlos. Esta variable se desactivará cuando el jugador se aleje del *collider* del teletransportador al que se ha movido.

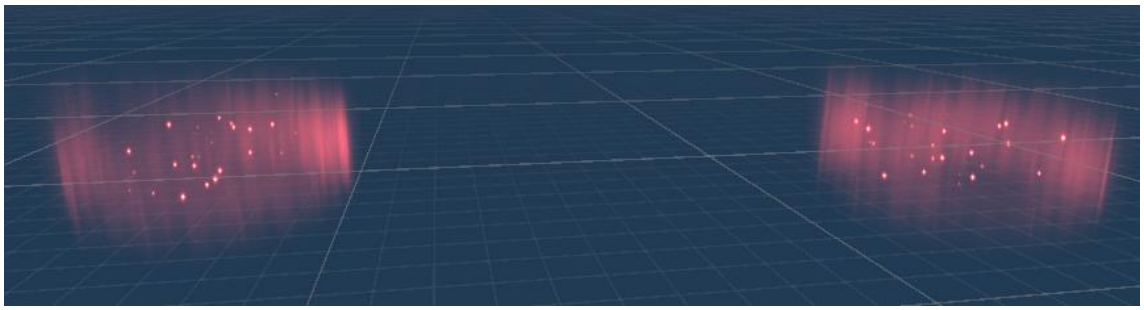


Ilustración 45. Par de teletransportadores.

4.5.6.2. Trampas de pinchos

Este elemento será utilizado en las salas como trampa que puede hacer daño tanto al jugador, como a los enemigos que puedan aparecer en una habitación de horda.

Se activará cuando algún actor de juego se coloque sobre ella, momento en el que activará su animación para levantar los pinchos del suelo. Esta animación tiene asignado un *Animation Event* para que empiece a hacer daño cuando los pinchos hayan ascendido lo suficiente, sincronizando mejor el efecto.

En cada animación de levantamiento de los pinchos, hará daño a cada actor de juego que se encuentre dentro de su *collider* de acción una sola vez, evitando que haga daño repetidamente.

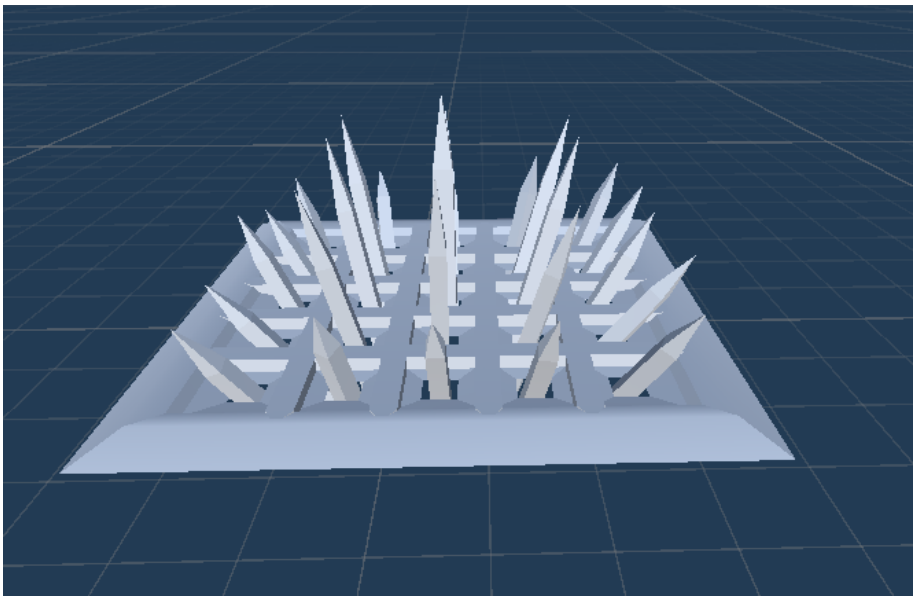


Ilustración 46. Modelo de trampa de pinchos.

4.5.6.3. Plataformas de salto

Otro elemento añadido para mejorar la navegación del personaje por los escenarios es la plataforma de salto. Este componente, cuando el jugador entra en contacto con su *collider*, le indica al personaje que debe aplicársele cierta cantidad de velocidad vertical. Cada plataforma podrá tener una fuerza específica de salto, que ayudará al personaje a llevar a zonas elevadas a las que no podría llevar con su salto normal.

Además, la dirección de velocidad que se le aplica al jugador dependerá del vector *Up* de la plataforma, lo que nos permitirá crear plataformas que empujen al jugador horizontal o diagonalmente.

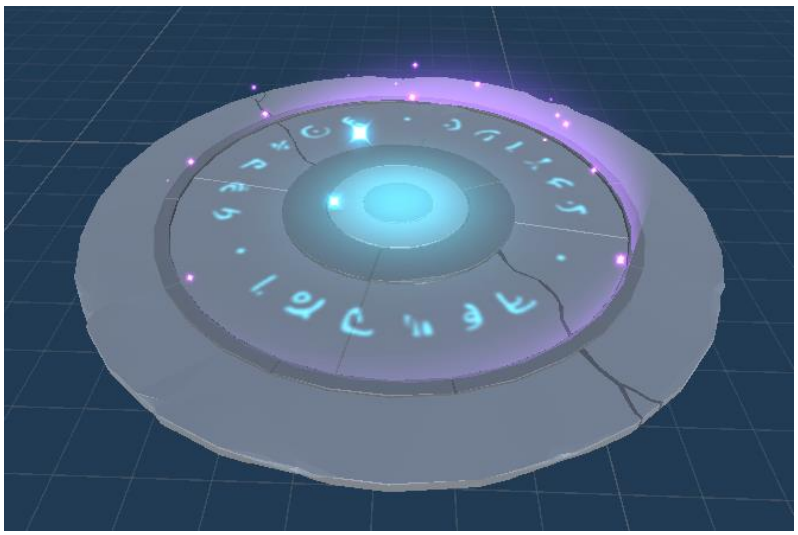


Ilustración 47. Modelo de plataforma de salto.

4.5.6.4. Pociones

Consumibles que el jugador puede recoger por los escenario para recuperar los recursos de salud y maná.

Cuando el jugador entra en contacto los *colliders* de estos *gameobject*, se les comunicará a los script de vida o maná que deben recuperar una cantidad determinada de ese recurso.

Ya que según el jugador vaya consumiendo objetos, irá perdiendo recursos para aguantar en las salas sin morir, se ha incluido una pequeño funcionalidad en los enemigos por la cual, al morir, tienen cierta probabilidad de soltar un tipo de poción aleatoria.

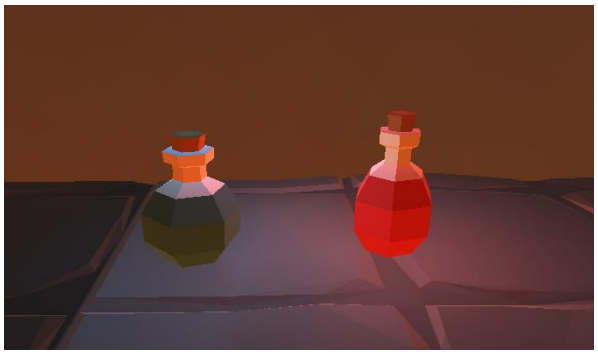


Ilustración 48. Poción de vida y maná.

4.5.6.5. Zonas de lava

Finalmente, en ciertos escenarios del juego se han introducido suelos o pozos de lava en los que, si el jugador cae y entra en contacto con ellos, directamente perderá toda su vida y morirá.

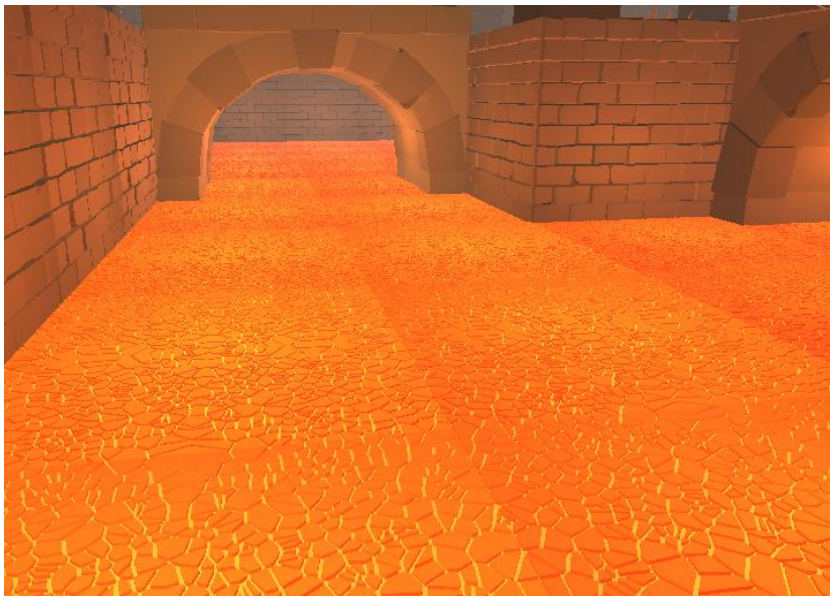


Ilustración 49. Zona de lava.

4.6. Inteligencia artificial

Como hemos comentado, para el comportamiento de los enemigos se han implementado unos sencillos scripts de inteligencia artificial a partir de máquinas de estados finitas (FMS). A continuación explicaremos con mayor detalle los tres tipos de comportamiento inteligente desarrollados.

4.6.1. Enemigos de horda cuerpo a cuerpo

Este tipo incluye a todos los enemigos que atacan cuerpo a cuerpo al jugador. Su comportamiento se basa en perseguir al jugador hasta que están a rango suficiente para poder atacar. Esto se resume en los dos estados siguientes:

- **Persecución:** Le indicamos en todo momento al enemigo la posición actual del jugador objetivo, a partir de la cual su *NavMeshAgent* buscará un camino hasta este haciendo uso de *pathfinding*.
- **Ataque**: En este estado, si nos encontramos a rango de ataque del personaje, dejará de moverse e intentará realizar un ataque activando la animación de ataque y su *AnimationEvent* correspondiente.
  Para facilitar el ataque del enemigo hacia el jugador aunque este último se haya movido, se establece un ratio de distancia de ataque por la cual el enemigo sigue moviéndose mientras realiza el ataque.

Por otro lado, tendríamos las transiciones entre estados de este tipo de enemigos:

- **Persecución -> Ataque:** El jugador está en rango de ataque.
- **Ataque -> Persecución**: El jugador no está en rango de ataque.

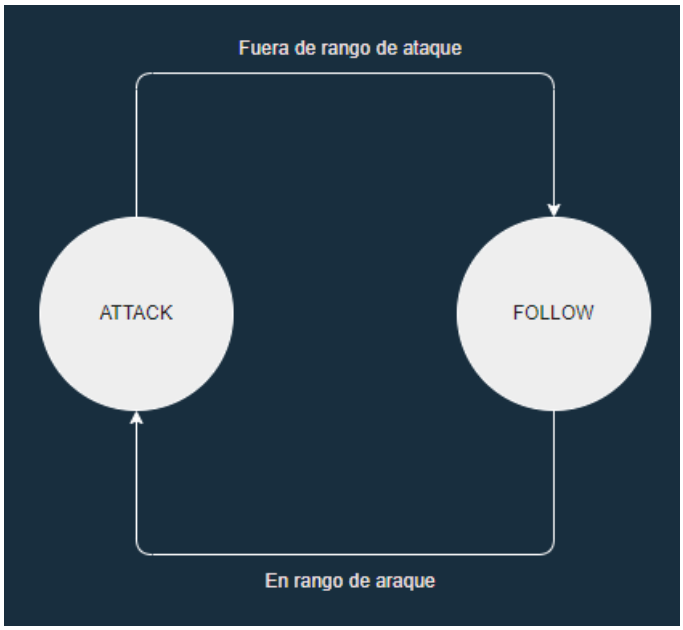


Ilustración 50. FMS de enemigo cuerpo a cuerpo.

4.6.2. Enemigos a distancia

Este tipo de comportamiento se utiliza exclusivamente para los enemigos acólitos, los cuales disparan proyectiles mágicos en la dirección del jugador como hemos comentado previamente. Comentamos a continuación sus estados y transiciones:

- **Persecución**: Funciona igual que para los enemigos cuerpo a cuerpo, modificando el objetivo de movimiento del *NavMeshAgent* a la posición actual del jugador.
- **Ataque**: El enemigo deja de moverse y calcula mediante un *BoxCast* si hay algún obstáculo en la trayectoria entre el proyectil y el jugador. En caso contrario, intenta disparar al jugador.

En cuanto a las nuevas transiciones entre ambos estamos podemos comentar los siguientes cambios:

- **Persecución -> Ataque:** Al igual que para los enemigos cuerpo a cuerpo, pasaremos a atacar al jugador cuando estemos a rango de ataque, con el añadido de que, si el enemigo detecta mediante un *BoxCast* que el jugador está tras un obstáculo y por lo tanto el proyectil lo podrá golpearle, permanecerá en estado de seguimiento hasta conseguir tener **"línea de visión**" para disparar.
- **Ataque -> Persecución**: De la misma forma, si el enemigo a distancia está muy lejos del jugador pasará automáticamente al estado de seguimiento hasta acercarse lo suficiente. Si por el contrario está a rango de ataque, primero deberá comprobar si no hay ningún obstáculo en la trayectoria de disparo, ya que en caso contrario deberá seguir en persecución del personaje.
-

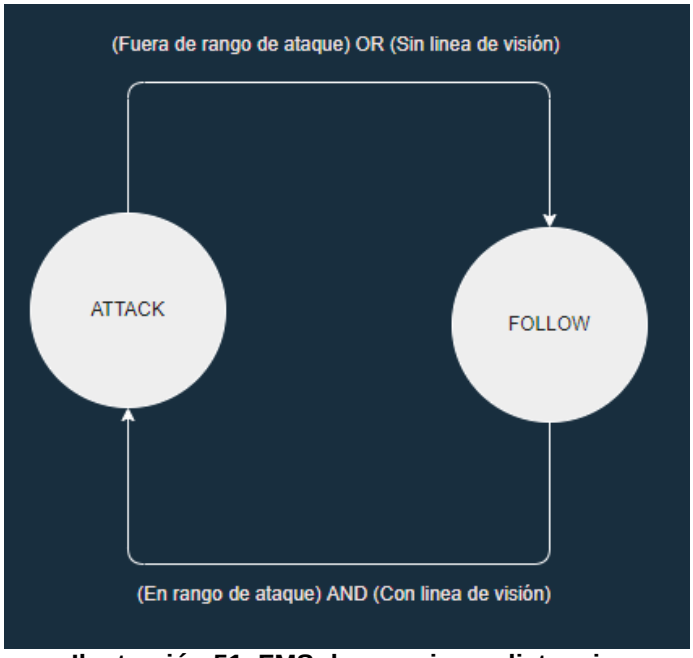


Ilustración 51. FMS de enemigo a distancia.

4.6.3. Jefe final

Finalmente, comentaremos el comportamiento inteligente del enemigo final del juego, en el cual, tal y como hemos mencionado anteriormente, se han desarrollado dos fases especiales de combate para mejorar la experiencia del encuentro, cada una de ellas incluyendo un estado adicional a la máquina de estados de los enemigos cuerpo a cuerpo:

- **Especial 1:** Como hemos comentado anteriormente, en este estado se quiere que el jefe final se desplace al centro de la sala y comienza a canalizar bolas mágicas entorno a sí mismo.
  Para ello, primero le indicamos al *NavMeshAgent* que se desplace a la posición que tenemos almacenada para el centro de la sala.
  Una vez detectamos que el enemigo ha llegado al dentro de la sala, empezará a lanzar un número concreto de oleadas de bolas mágicas, con cierto tiempo entre ellas.
- **Especial 2**: Mismo funcionamiento que para el estado anterior, pero en este caso, cuando el jefe llega al centro de la habitación, comenzará a invocar enemigos aleatorios que perseguirán al jugador.

Por último, comentaremos las transiciones añadidas para el caso de uso de las fases especiales:

- **Persecución -> Especial1**: Cuando la vida del jefe final baja del 66% y aun no se ha utilizado la habilidad especial 1.
- **Ataque -> Especial1:** Cuando la vida del jefe final baja del 66% y aun no se ha utilizado la habilidad especial 1.
- **Persecución -> Especial2**: Cuando la vida del jefe final baja del 33% y aun no se ha utilizado la habilidad especial 2.
- **Ataque -> Especial2:** Cuando la vida del jefe final baja del 33% y aun no se ha utilizado la habilidad especial 2.
- **Especial1 -> Persecución:** El jefe final termina de canalizar la habilidad especial 1.
- **Especial1 -> Especial2**: Si durante la canalización del especial 1 la vida del jefe baja del 33%
- **Especial2 -> Persecución:** El jefe termina de canalizar la habilidad especial 2.

El resto de los estados de ataque y persecución funcionan igual que para los enemigos normales cuerpo a cuerpo, por lo que su comportamiento en estos estados sería igual que para este tipo de enemigos.

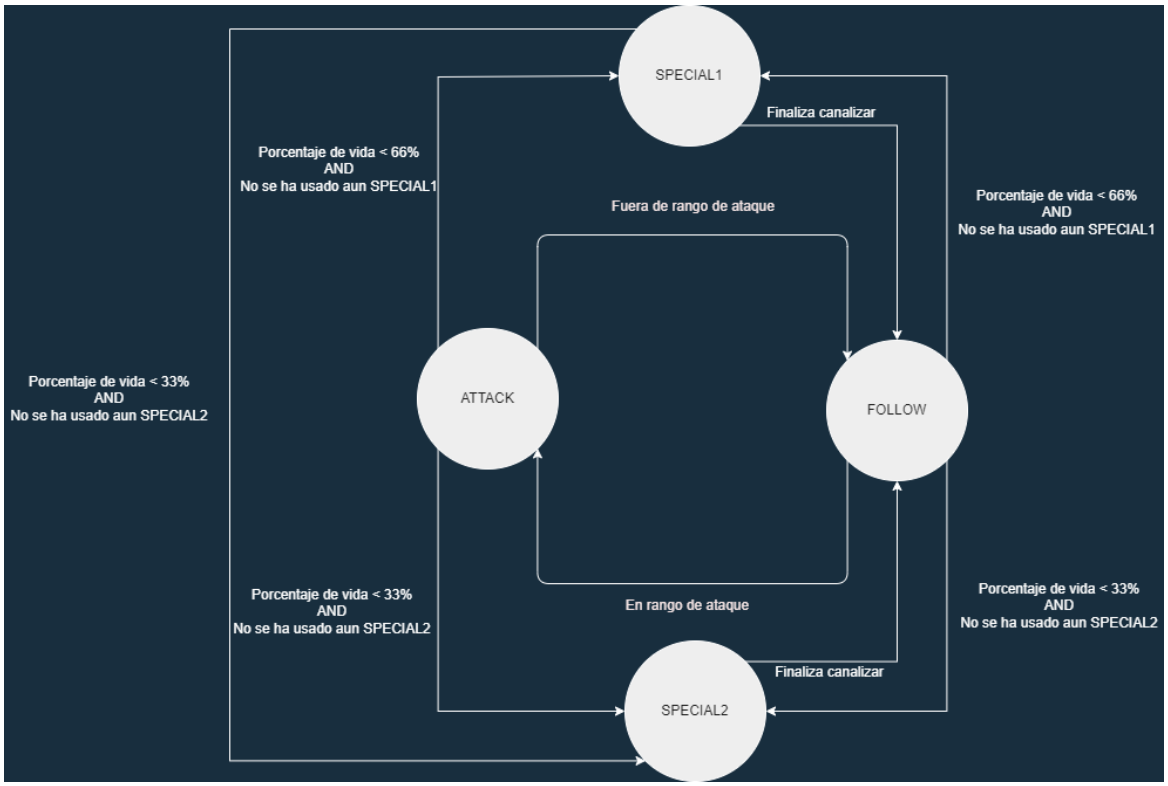


Ilustración 52. FMS del jefe final.

5. Diseño de niveles

En este último apartado de desarrollo, explicaremos brevemente el diseño aplicado para una habitación de cada tipo.

5.1. Habitación de horda

Esta habitación corresponde a la segunda sala que podemos encontrar en la primera escena de mazmorra de horda.

En ella, se quiso añadir al diseño la existencia de dos **niveles de altura**. Este tipo de diseño, para un juego de acción en primera persona como el que se intenta desarrollar, mejora mucho la navegabilidad del jugador, sobre todo cuando se ve abrumado por muchos enemigos, permitiéndole saltar desde el puente que vemos en la parte superior del mapa hacia abajo para esquivar a los enemigos si está rodeado.

De manera similar a lo que acabamos de comentar, se incluyen dos pares de **teletransportadores** para conectar la parte inferior y superior del escenario, permitiendo al jugador moverse instantáneamente entre ellas, con la peculiaridad de que ya que el teletransportador de la parte superior del mapa se encuentra al lado de un portal de horda, deberá tener cuidado de que no haya enemigos cerca al usarlo.

Finalmente, se han incluido **trampas** de pinchos por la parte inferior del mapa, con el objetivo tanto de dificultar el movimiento del jugador por una parte del mapa que es más amplia, al mismo tiempo que para dar una herramienta al jugador que le permite atraer a los enemigos a estas trampas para hacerles daño.

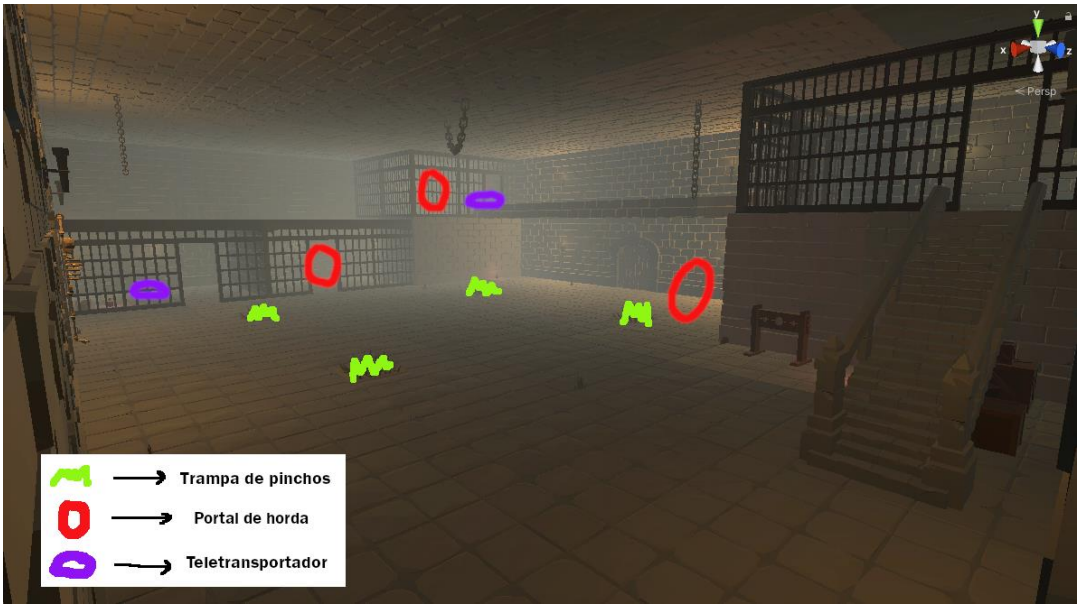


Ilustración 53. Diseño de habitación de horda.

5.2. Habitación de puzle

Por otro lado tenemos las habitaciones de puzle, las cuales ya hemos comentado anteriormente que su funcionalidad es muy simple, por lo que su uso se centra en el diseño de juego que se le aplica para superar desafíos o rompecabezas.

En la habitación que mostramos a continuación, el diseño principal se vaya en el uso de las mecánicas de movimiento y salto para superar un desafío de plataformas, al mismo tiempo que el jugador va recogiendo las llaves para poder abrir la puerta que lleva a la siguiente escena.

Para las decisiones de diseño de este mapa, se ha querido que el usuario aprenda con precisión su altura de salto y la diferencia entre un salto con o sin sprint. Por ello, las plataformas han sido colocadas a la distancia exacta para tener que usar un salto con sprint o salto sin sprint, lo que llevará al usuario a aprender gradualmente a controlar las limitaciones y posibilidades de su sistema de movimiento.

Para incrementar la sensación de peligro si no se consigue alcanzar las plataformas, en el fondo del mapa se ha colocado un pozo de lava, en el cual nuestro personaje morirá al caer dentro y tendrá que empezar desde el principio del mapa.

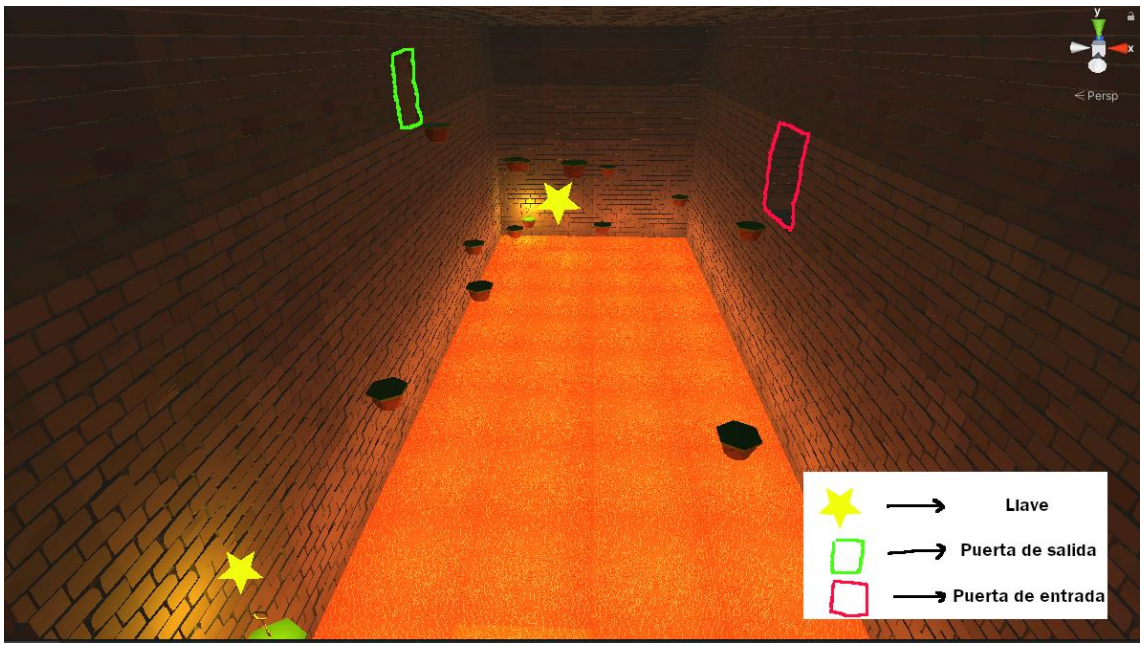


Ilustración 54. Diseño de habitación de puzle.

5.3. Habitación de jefe final

Por último, comentamos brevemente el diseño que se ha aplicado para la habitación donde el jugador se enfrenta al jefe final.

Como se puede observar, la sala es bastante amplia y sin apenas componentes que estorben el movimiento durante el combate.

Durante la primera fase de combate especial del jefe, sabemos que se moverá al **centro** de la sala, desde donde empezará a canalizar bolas mágicas. Debido al número de bolas que se instancian en cada canalización, el jugador tendrá que esquivarlas si se queda en la parte centrar de la sala. Pero si prevé este ataque, podrá moverse detrás de una de las **columnas** que hay en los laterales de la habitación para protegerse.

Además, ya que durante la segunda fase de ataque especial el jefe invocará muchos enemigos que podrán acorralarle fácilmente, se han incluido un par de **teletransportadores** que conectan ambos laterales de la sala, los cuales puede usar el jugador para alejarse de los enemigos.

Finalmente, ya que el número de enemigos que se invocan en la última fase puede ser abrumador, se incluyen **trampas** de pinchos en ciertas zonas del mapa para ayudar al jugador a bajar la vida a los enemigos.

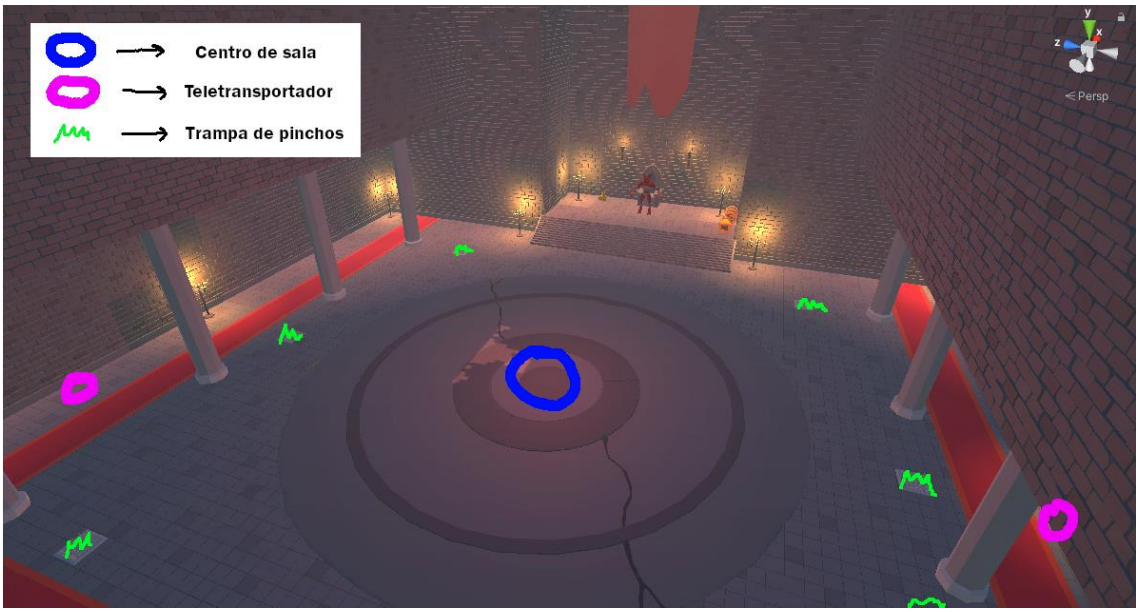


Ilustración 55. Diseño de habitación de jefe final.

6. Análisis de costes

En este apartado adicional se va a intentar hacer un análisis de costes del proyecto, tanto en costes inferidos de sueldos equivalentes durante el tiempo de desarrollo, equipamiento utilizado y finalmente el coste energético aproximado.

Se tendrá en cuenta que el proyecto ha sido desarrollado por una única persona con el cargo de *desarrollador de videojuegos* en España, con un tiempo de desarrollo equivalente a los 12 créditos asociados a la asignatura de TFM, un total de 300 horas.

El desglose de los costes sería:

- Teniendo en cuenta un sueldo medio de 1740€ para programador de videojuegos en España [19], que equivaldría a 10.04€ a la hora, el **sueldo total** sería de: **3012**€.
- Coste del **equipamiento**, incluyendo ordenador de escritorio, teclado, ratón y dos monitores: **2000**€.
- Para su uso en el proyecto se han obtenido **licencias** de *assets* en la Unity Asset Store que ascienden a un coste total de: **64.02**€

En **total**, los costes derivados de este proyecto ascenderían a: **5076.02** €.

6.1 Coste energético

Ya que el único equipamiento de desarrollo que se ha utilizado es un ordenador de escritorio, daremos el coste energético derivado de su uso.

De media un ordenador de sobremesa puede llegar a consumir desde 180W hasta 220W, lo que dependiendo del coste energético del kW podría suponer un coste por hora de 0.14€/kWh. [20]

Aplicado este precio a las horas de desarrollo que hemos fijado, esto supone un coste energético total de: **42**€.

7. Manual de usuario

Ya que principalmente nos centraremos en el desarrollo para ordenador, mostramos a continuación los controles principales del usuario con respecto a la funcionalidad de juego.

- **W:** Caminar hacia delante.
- **A**: Caminar hacia la izquierda.
- **S**: Caminar hacia atrás.
- **D**: Caminar hacia la derecha.
- **Barra Espaciadora**: Saltar
- **Mayus**: Correr
- **Clic Izquierdo**: Ataque principal
- **Clic Derecho**: Ataque secundario
- **Mover ratón:** Control de cámara

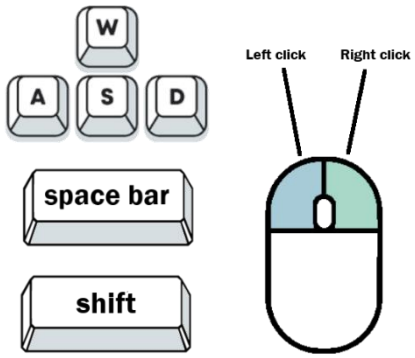


Ilustración 56. Controles de juego.

- Requerimientos **máximos** probados para el juego:
  - **Sistema operativo**: Windows 10 Pro.
  - **Procesador**: AMD Ryzen 7 3700X 8-Core Processor, 3.60 GHz.
  - **Memoria RAM**: 32 GB.
  - **Tarjeta gráfica**: Radeon RX 580 Series

- Requerimientos **mínimos** probados para el juego:
  - **Sistema operativo**: Windows 10 Pro.
  - **Procesador**: Intel® Core™ i5-4210U CPU @ 1.70GHz 2.40GHz.
  - **Memoria RAM**: 8 GB.
  - **Tarjeta gráfica**: GeForce GT 820M

8. Conclusiones

En este último capítulo vamos a recapitular los objetivos que se han cumplido para este proyecto de fin de master, así como una reflexión sobre las lecciones aprendidas y posibles mejoras aplicables a nuestro trabajo en un futuro.

La lección más importante que se ha aprendido para este trabajo es la gran importancia de una buena planificación de objetivos y metas de cara a afrontar proyectos de gran envergadura. Esto incluye la necesidad de dividir el trabajo en tareas pequeñas y abordables en vez de empezar con conceptos muy generales, los cuales acaban llevando a confusiones y problemas en la producción.

Después, ya que para el proyecto de master no se contaba con material de refuerzo como para el resto de las asignaturas del master, ha sido muy importante aprender a buscar recursos en línea, sabiendo filtrar la información que mejor se adaptaba a nuestro proyecto y sus necesidades de cara a desarrollar un videojuego sólido.

En cuanto a las lecciones aprendidas durante el propio desarrollo técnico del proyecto, podemos destacar el uso de *layers* para mejorar las animaciones de *rigs* humanoides, la optimización de escenarios muy grandes cuando se utilizan muchos efectos de partículas o se instancian muchos *gameobjects* y principalmente el conocimiento adquirido a la hora de desarrollar modelos, animaciones y funcionalidad completa para un personaje con perspectiva en primera persona.

Comentando los objetivos que se plantearon al principio del desarrollo, se subestimó el tiempo que requeriría desarrollar algunas de las funcionalidades de juego, así como los distintos tipos de salas que se tenían planeadas, lo que llevó a que varias de estas metas que nos propusimos, como el uso de dinero para mejorar las características del persona, finalmente no hayan sido posibles.

En lo que a esto último respecta, se considera que en futuros proyectos que se deban afrontar de desarrollo de videojuegos, se intentará dar una estimación más al alza del número de horas necesarias para terminar los objetivos planteados, para así poder alcanzar una previsión más realista del producto final que se quiere conseguir.

Finalmente, se enumerarán algunas de las líneas de trabajo futuro que no se han podido añadir al trabajo final por falta de tiempo o porque no se tuvieron en cuenta en la planificación inicial:

- Implementar un sistema de **dinero**, el cual podrán ir soltando los enemigos o ser obtenidos de cofres que encontremos por el mapa.
- Entre habitación y habitación podrán aparecer **tiendas** goblin donde podremos utilizar el dinero para comprar mejoras de armas o para incrementar las características del jugador como la vida máxima o la capacidad de maná máxima.
- Para poder dar libertad al usuario en cuanto a mejorar sus características o para proporcionarle un mayor desafío, se plantea la posibilidad de poder repetir salas de horda que se haya completado anteriormente en **dificultades superiores**. Superar por primera vez una dificultad concreta de una sala proporcionará un cofre de dinero de cantidad proporcional a la dificultad.
- Múltiples nuevas salas de los tres tipos de habitación que se incluyen en el juego.
- Nuevas armas y magia, incluyendo **ballesta**, posibilidad de usar el **escudo** con la espada de una mano o ataque mágico de **terremoto**.
- Mejorar la ambientación del juego con una inclusión directa de la historia que se ha comentado en el diseño del juego, con cinemáticas y diálogos con personajes del juego.

Bibliografía

[1] Smithsonian, «Video Game History,» [En línea]. Available: https://www.si.edu/spotlight/the-father-of-the-video-game-the-ralph-baer-prototypes-and-electronic-games/video-game-history.

[2] B. N. Laboratory, «Tennis For Two».

[3] P. Checkpoint, «¿Recuerdas Wolfenstein 3D? El primer FPS de tu infancia regresa.,» [En línea]. Available: https://predatorcheckpoint.com/wolfenstein-3d-el-primer-fps-de-id-software-para-el-mundo/.

[4] Innovecs, «Game Development Process,» [En línea]. Available: https://innovecs.com/blog/game-development-process/.

[5] MeriStation, «Análisis de DOOM (2016),» [En línea]. Available: https://as.com/meristation/2016/05/18/analisis/1463533200_155504.html.

[6] J. WOJNAR, «Every Elder Scrolls Game Ranked From Shortest To Longest (According To How Long To Beat),» [En línea]. Available: https://www.thegamer.com/every-elder-scrolls-game-ranked-from-shortest-longest-according-how-long-beat/.

[7] E. Games, «A first look at Unreal Engine 5,» [En línea]. Available: https://www.unrealengine.com/en-US/blog/a-first-look-at-unreal-engine-5.

[8] Unity, «https://unity3d.com/es/beta/2020.1b,» [En línea]. Available: https://unity3d.com/es/beta/2020.1b.

[9] «Unity Asset Store, Action RPG FX, Archanor VFX,» [En línea]. Available: https://assetstore.unity.com/packages/vfx/particles/action-rpg-fx-38222.

[10] «Unity Asset Store, Unique Projectiles Vol. 2,» [En línea]. Available: https://assetstore.unity.com/packages/vfx/particles/unique-projectiles-vol-2-156067.

[11] «Unity Asset Store, GUI PRO Kit - Fantasy RPG,» [En línea]. Available: https://assetstore.unity.com/packages/2d/gui/gui-pro-kit-fantasy-rpg-170168.

[12] «Unity Asset Store, Low Poly Character - Demons, Infinity PBR,» [En línea]. Available: https://assetstore.unity.com/packages/3d/characters/creatures/low-poly-character-demons-183691.

[13] «Unity Asset Store, Low Poly Dungeons, Just Create,» [En línea]. Available: https://assetstore.unity.com/packages/3d/environments/dungeons/low-poly-dungeons-176350.

[14] «Unity Asset Store, Low Poly Terrain - Polaris 2020, Pinwheel Studio,» [En línea]. Available: https://assetstore.unity.com/packages/tools/terrain/low-poly-terrain-polaris-2020-170400.

[15] «Unity Asset Store, Low Poly Ultimate Pack, polyperfect,» [En línea]. Available: https://assetstore.unity.com/publishers/19123.

[16] «FreeSound,» [En línea]. Available: https://freesound.org/.

[17] «Mixamo,» [En línea]. Available: https://www.mixamo.com/#/.

[18] «Sketchfab,» [En línea]. Available: https://sketchfab.com/feed.

[19] Jobted, «Sueldo del Programador de Videojuegos en España,» [En línea]. Available: https://www.jobted.es/salario/programador-videojuegos.

[20] Tarify, «Cuál es el consumo de un ordenador de sobremesa,» [En línea]. Available: https://tarify.es/noticias/cual-consumo-ordenador-sobremesa.

Anexos

Trailer del videojuego:

Trailer Last Resurrection - YouTube

Defensa del proyecto TFM:

Defensa TFM - Last Resurrection - YouTube