UNIVERSITAT DE BARCELONA

Trabajo de Fin de Grado

GRADO DE INGENIERÍA INFORMÁTICA

Facultad de Matemáticas e Informática
Universidad de Barcelona

Desarrollo de un videojuego 2.5D con Unity

Daniel Rivero Martínez

Director: Francesc Xavier Dantí Espinasa
Realizado en: Departamento de Matemáticas e Informática
Barcelona, 26 de enero de 2017

Resumen

Con este proyecto he buscado plasmar el interés que he tenido toda la vida por los videojuegos, y los conocimientos adquiridos durante mis estudios en ingeniería informática, en el desarrollo desde cero de un videojuego.

El videojuego desarrollado pertenece al género de los juegos de plataformas y tiene una perspectiva 2.5D, es decir, los gráficos utilizados son en 3 dimensiones pero la jugabilidad se limita a las 2 dimensiones.

Para ello he utilizado el motor de videojuegos Unity y el lenguaje de programación C Sharp, ayudándome del programa de gráficos 3D Blender para el diseño de los elementos gráficos utilizados en el videojuego.

Resum

Amb aquest projecte he buscat plasmar l'interès que he tingut tota la vida pels videojocs, i els coneixements adquirits durant els meus estudis en enginyeria informàtica, en el desenvolupament des de zero d'un videojoc.

El videojoc desenvolupat pertany al gènere dels jocs de plataformes i té una perspectiva 2.5D, és a dir, els gràfics utilitzats són en 3 dimensions però la jugabilitat es limita a les 2 dimensions.

Per a això he utilitzat el motor de videojocs Unity i el llenguatge de programació C Sharp, ajudant-me del programa de gràfics 3D Blender per al disseny dels elements gràfics utilitzats en el videojoc.

Abstract

With this project I wanted to capture the interest I've had my whole life, and the knowledge acquired during my studies in computer engineering, by the development from scratch of a video game.

The developed video game belongs to the genre of platform games and has a 2.5D perspective, i.e., used graphics are 3-dimensional but the gameplay is limited to 2 dimensions.

To do this I used the Unity game engine and the C Sharp programming language, and the 3D graphics software Blender for the design of the graphic elements used in the video game.

Índice de contenido

Índice de figuras y tablas

1 Introducción

1.1 Motivación

Siempre he sido un gran aficionado al mundo de los videojuegos, desde pequeño me ha gustado jugar a juegos de todos los géneros y en todas las plataformas que he podido tener; y a día de hoy sigo manteniéndolos como una de mis principales aficiones, siempre que el tiempo me lo permite.

Por esto, cuando cursé la asignatura de Ingeniería del Software como diseñador 3D disfruté mucho del desarrollo del videojuego que realizamos en ella. Pero también me quedé con ganas de aprender más del mundo del desarrollo de videojuegos, ya que en la asignatura me centré en el rol de diseñador, y no entré en el campo de la programación y en las demás partes necesarias que te encuentras cuando tienes que hacer un proyecto completo por ti mismo.

Con lo cual, al ponerme a pensar que Trabajo de Final de Grado podría hacer, la idea de realizar como proyecto un videojuego completo desde cero con todas sus partes me pareció la forma ideal de profundizar y completar aquello que empecé a aprender con Ingeniería de Software, así como de utilizar todo lo aprendido en todos mis años como jugador de videojuegos.

1.2 Objetivos

El objetivo principal es llevar a cabo la creación de un videojuego, utilizando para ello el motor de videojuegos Unity y el lenguaje de programación C Sharp, y ayudándome de Blender para los recursos 3D que requiera el proyecto.

El género elegido para el videojuego es el género de plataformas con una temática de fantasía medieval, en el cual el protagonista tendrá mejoras y recolectables diversos para aumentar su poder que podrá recoger por el escenario, además de enemigos a vencer distribuidos por el mapa. Asimismo, la perspectiva que se utiliza es 2.5D, que se basa en que el mapa y sus elementos son diseños 3D, pero la jugabilidad para el jugador se limita al 2D (alto y ancho), con lo cual se consigue un videojuego visualmente moderno con una jugabilidad clásica.

Está claro que el resultado al que se busca llegar no es el equivalente a un videojuego comercial completo, el cual suele desarrollar un equipo completo de personas con buenos conocimientos en su campo, ya que, entre otros factores lógicos como el tiempo y los recursos que tengo, prácticamente no tengo conocimientos previos en el desarrollo de videojuegos y necesito un periodo de documentación y aprendizaje de los diferentes programas a utilizar.

Aun así, durante el proyecto siempre se buscará el cubrir unos requisitos básicos del videojuego, así como el alcanzar un nivel adecuado en todas sus áreas, como son:

- La lógica: scripts que permitan jugar e interaccionar de forma completa y permitan una buena escalabilidad del código.

- La jugabilidad: aun al no tratarse de un producto completo, que el diseño del escenario y el flujo de juego tenga una coherencia.

- La estética: un estilo visual definido y decente.

2 Estado del arte

2.1 Videojuegos

La industria del videojuego se inició en los años 70, con la aparición de las primeras máquinas recreativas. Durante esas primeras décadas, los videojuegos representaban una forma de ocio algo cara y analógica, aún muy primaria y enfocada principalmente a los niños.

Según fueron avanzando las décadas, los recursos de hardware y software disponibles para desarrollar los videojuegos fueron creciendo exponencialmente. Esto hizo que aparecieran continuamente nuevas plataformas para jugar, con más capacidad de computación y más potencia gráfica, siendo las primeras que se popularizaron la NES y la Mega Drive, continuando con la revolución que causaron la Nintendo 64 y la PlayStation, y llegando hasta día de hoy, que siguen apareciendo nuevas videoconsolas, como la Nintendo Switch, la PlayStation 4 y la Xbox One. Además de todas estas, hay que tener en cuenta la que es una de las plataformas más utilizadas a día de hoy para jugar, el ordenador personal.

Aparte de todas estas plataformas de sobremesa, estás últimas dos décadas han conseguido muchísima fuerza las consolas portátiles, plataforma que se abrió al mundo con la creación de la GameBoy por parte de Nintendo, se consolidó al público en general con la Nintendo DS, y plataforma a la que a día de hoy prácticamente todo el mundo tiene acceso, ya que los smartphones se han convertido en la nueva consola portátil de nuestra generación.

Todos estos avances en las capacidades de las videoconsolas, y en el público que las utiliza, ha hecho que la variedad de los videojuegos crezca enormemente y aparezcan nuevos géneros como los de acción, estrategia, rol, aventura, puzles, simuladores, deportes, etc., cada uno de ellos con sus subgéneros propios. Además, las nuevas tecnologías han permitido crear una nueva forma de juego muy popular estos últimos años, además de implementable en todos estos géneros, que es la opción del multijugador, tanto en versión local como en su versión online; lo cual ha llegado a crear videojuegos que permiten que miles de jugadores puedan jugar en un mismo mundo virtual simultáneamente y puedan competir entre ellos, como son, por ejemplo, los MMORPG (Massively Multiplayer Online Role-Playing Game).

Desde esos inicios de la industria, con el paso de las décadas y los avances que conllevaron, el mundo de los videojuegos ha evolucionado hasta llegar a ser una de las principales formas de entretenimiento de la sociedad actual, actualmente llegando a duplicar la facturación de la

industria del cine. La industria da trabajo a un gran número de gente de todas las disciplinas en todo el mundo y se podría desgranar en las siguientes divisiones: los inversores, los desarrolladores de videojuegos, los creadores del software empleado por los desarrolladores, los fabricantes de hardware, las distribuidoras de videojuegos, y los consumidores. Por ejemplo, el desarrollo de un videojuego puede llegar a costar desde unos miles de dólares para un pequeño juego hasta el presupuesto de un videojuego AAA como es el GTA V que fue de 167 millones de dólares.

Se podría decir que los videojuegos se han asentado en la sociedad actual como una de las principales y más habituales formas de ocio de la población, cubriendo cada vez más las diferentes generaciones de jugadores, y disminuyendo la brecha entre las diferentes edades que lo consumen.

Actualmente están surgiendo nuevas corrientes dentro de la industria del videojuego, como es la industria indie, pequeñas empresas o equipos que desarrollan videojuegos de poco presupuesto y que muchas veces consiguen resultados que no tienen nada que envidiar a las grandes compañías de videojuegos; gracias a esto, es mucho más fácil para cualquier desarrollador entrar en la industria y publicar sus creaciones. Por otro lado, se están empezando a desarrollar muchos videojuegos con fines educativos, ya que se está demostrando que la integración de los videojuegos en el ámbito educativo hace que los estudiantes estén más motivados a la hora del aprendizaje. Y, no hay que olvidarse de lo que actualmente es el futuro de los videojuegos, la realidad virtual, campo en el que las grandes empresas se están enfocando, y en el que se pueden observar grandes avances en los últimos años.

Y, por último, es necesario remarcar una nueva forma de ver los videojuegos que ha surgido está última década: los jugadores profesionales y el deporte electrónico; algo que empezó como algo secundario pero actualmente genera una gran cantidad de dinero y mueve a mucha gente en todo el mundo. Estos últimos años se ha visto como cada vez más países están considerándolos deporte oficial y ya se pueden ver retransmisiones en las televisiones de muchos países.

2.2 Unity

En el mercado de los motores de videojuegos (game engines) existen muchas opciones para elegir según tus necesidades a la hora de querer desarrollar un videojuego. Por un lado están los motores privados, que son los que algunas empresas como Sony o Ubisoft utilizan para desarrollar muchos de sus juegos y a los cuales los usuarios particulares no tienen acceso; y, por otro lado, tenemos los motores públicos como son Unity, Unreal Engine o Cry Engine, que adquiriendo la licencia necesaria cualquier usuario o empresa puede utilizar.

En el caso de Unity, existen dos tipos de licencias: una licencia gratuita personal que permite usar el programa a cualquiera que la solicita, además de dar acceso a muchas de sus funcionalidades y recursos; y, una licencia profesional, que te da acceso completo a todas las funcionalidades que contiene, la cual es obligatoria comprar si el proyecto desarrollado genera unos beneficios superiores a los 100.000 dólares.

Este sistema de licencias fomenta que cualquier persona o pequeña empresa interesada en aprender a crear videojuegos y lanzarlos al mercado pueda acceder a un programa tan amigable y completo como es Unity. Este hecho, sumado a la actual edad dorada por la que están pasando los videojuegos indie, ha propiciado que en los últimos años algunos de los grandes juegos actuales se hayan desarrollado con este motor, como pueden ser juegos para ordenador como Pillars of Eternity y Rust, así como grandes juegos para móviles como Pokemon Go y Hearthstone.

Además, es importante indicar que Unity es un motor de videojuegos multiplataforma, es decir, permite crear proyectos para la mayoría de plataformas que hay actualmente en el mercado: PC (Windows, Mac, Linux…), dispositivos móviles (Android, iOS…), consolas (PlayStation 4, Xbox One, Wii U, Nintendo DS…), WebGL, Smart TV’s y los nuevos dispositivos de realidad virtual. Haciendo uso en la mayoría de ellas del motor gráfico OpenGL, y dando en Windows la alternativa de usar Direct3D, si así lo quiere el desarrollador.

Por otro lado, en el apartado del lenguaje de programación a utilizar, Unity permite desarrollar los scripts del proyecto usando C Sharp o JavaScript. Siendo este primero el más usado y, por lo tanto, el que tiene más documentación y soporte detrás por parte de la comunidad.

2.3 Blender

Existe una gran cantidad de software dedicado a la creación de gráficos 3D, tanto de pago como gratuitos, como podrían ser 3D Studio Max (líder actual en la industria), Lightwave 3D (muy utilizado por productoras de efectos especiales), Maya (usado por Pixar) o Blender.

Aun así, aunque no sea el más utilizado, para este proyecto he decidido utilizar Blender, debido a que es un software libre que ofrece una gran potencia y todas las funcionalidades que necesito, y tiene una extensa y activa comunidad de usuarios detrás que están constantemente ampliándolo y dando soporte a sus usuarios. Además, ya poseo cierta experiencia utilizándolo para la creación de elementos 3D, adquirida cursando la asignatura de Ingeniería de Software.

Blender es un programa de creación de gráficos 3D que permite el modelado, texturizado, renderizado y la animación de elementos en 3 dimensiones con relativa facilidad; además, es un software multiplataforma, se puede utilizar en Windows, Mac y GNU/Linux; y, permite la exportación de los elementos creados a una gran variedad de formatos, muchos de los cuales compatibles con Unity.

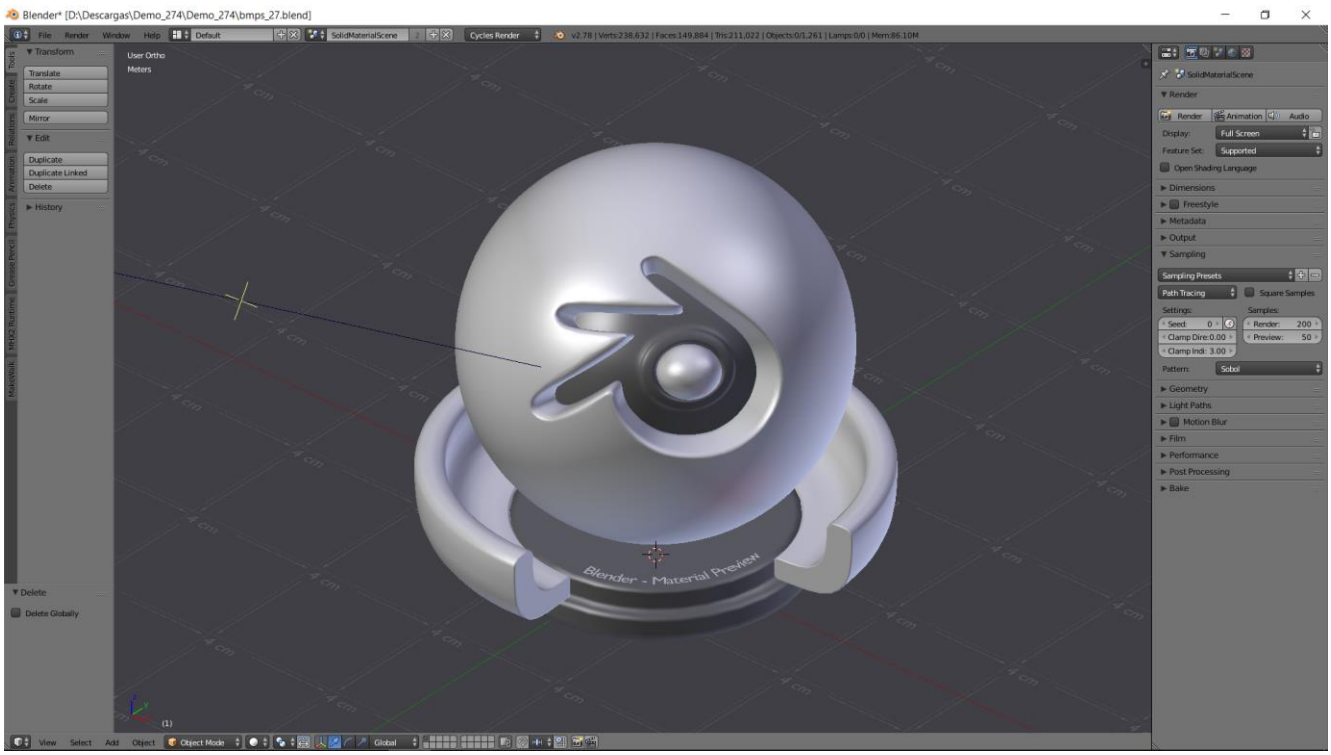

Figura 1: Ejemplo de modelado en Blender.

3 Análisis

3.1 Decisiones iniciales

Previamente al inicio del desarrollo del videojuego es necesario realizar un análisis para definir diversos temas importantes y básicos del mismo.

El primer paso a seguir para diseñar un videojuego es definir el género, la temática y la perspectiva. En este caso, el género será un juego de plataformas que, además, contendrá enemigos distribuidos por el escenario a los que el personaje principal tendrá que derrotar, así como diferentes objetos que al recogerlos reforzarán al personaje de distintas maneras.

En cuanto a la temática, este proyecto, en sus inicios, empezó a desarrollarse como un videojuego con temática militar postapocalíptica, pero al analizar en profundidad el producto que quería a nivel jugable y estético se cambió a una temática de fantasía medieval, ya que considero que me permitirá insertar elementos de combate interesantes como son las armas cuerpo a cuerpo, así como ataques mágicos a distancia, y conseguir darle variedad al sistema de combate. Asimismo, elijo esta temática porque es un campo en el que me siento cómodo y tengo experiencia, tanto como jugador por los años que llevo jugando a este tipo de videojuegos, como desde el punto de vista de diseñador ya que cuando cursé la asignatura de Ingeniería de Software me encargué de diseñar elementos para un videojuego de esta temática, lo cual me facilitará el proceso de imaginar y diseñar los diferentes elementos del videojuego.

Por último, la perspectiva elegida es el 2.5D, la cual me permitirá aprovechar y perfeccionar la experiencia que adquirí cursando la asignatura de Ingeniería de Software en el campo del diseño de elementos en tres dimensiones, pero dentro de un videojuego con una lógica y una jugabilidad en dos dimensiones. Además, esta combinación genera un videojuego visualmente llamativo por sus elementos con profundidad, a la vez que se consigue mantener la jugabilidad de un videojuego de plataformas clásico.

Figura 2: Trine, juego de plataformas en 2.5D con temática fantástica medieval.

3.2 Jugabilidad

El siguiente paso en el diseño previo del videojuego es definir la jugabilidad del mismo, en este caso el juego lo protagonizará un caballero con armadura, controlado por el jugador, el cual podrá andar, correr y saltar para avanzar por el escenario, a la vez que podrá atacar cuerpo a cuerpo con el arma de la que disponga y lanzar proyectiles mágicos, mientras disponga de puntos de magia suficientes para ello. Además, el personaje tendrá varios parámetros que definirán su cantidad de vida, velocidad de movimiento, etc.

Por otro lado, el juego tendrá diversos tipos de enemigos que detectarán, perseguirán y atacarán al personaje principal. Dichos enemigos tendrán también diversos parámetros que les definirán: sus puntos de vida, su velocidad, su poder de ataque, su velocidad de ataque o la velocidad a la que detecta al personaje principal al acercarse a ellos. La idea es tener enemigos de categoría normal y, además, un enemigo intermedio y un enemigo final, los cuales serán especiales.

En el aspecto del escenario, estará compuesto por plataformas estáticas y plataformas móviles, las cuales se moverán en bucle o será necesario activarlas mediante algún tipo de mecanismo, Asimismo, a lo largo del escenario, habrá elementos que dañaran al jugador al tocarlos, así como elementos destruibles que bloquearán el avance del jugador o le permitirán el acceso a mejoras.

A lo largo del escenario se repartirán distintas mejoras para el jugador: diversas armas, que el jugador podrá equiparse y usar para atacar, otorgando cada una de ellas diferentes parámetros de ataque (daño, velocidad de ataque, etc.).

En el aspecto de los objetos que el jugador podrá encontrar por el mapa también estarán las pociones de vida y de magia, que curarán los puntos de vida y los puntos de magia del jugador, respectivamente. Por otro lado, repartidos por el escenario también habrá tesoros que el jugador podrá recoger y así aumentar su puntuación. Este tipo de elementos recolectables (las pociones y las monedas) también podrán soltarlas los enemigos al ser derrotados por el jugador).

Asimismo, el jugador tendrá que disponer de una interfaz en la cual se muestre la información importante necesaria, como son los puntos de vida, los puntos de magia, las monedas recogidas y el arma que se está utilizando. También será necesario añadir botones para pausar y cerrar el juego, así como una pantalla que explique al jugador los controles necesarios para jugar.

Por último, se define que el objetivo del juego será llegar al final del mapa y derrotar al enemigo final, intentando alcanzar ese punto con el mayor número de tesoros posible, derrotando para ello a los enemigos del camino y recogiendo los tesoros repartidos por el mapa.

3.3 Elementos necesarios

Llegados a este punto, es necesario hacer un listado orientativo de los elementos que serán necesarios para el desarrollo del videojuego, el cual expongo a continuación, repartido en diversas categorías:

- Mapa: plataformas de terreno de diferentes tamaños (unas se usarán como terreno estándar y otras como plataformas móviles), pequeñas plataformas de madera (se utilizarán para mecanismos estilo ascensor), palancas para los mecanismos, puertas o barreras, elementos dañinos al tocarlos, cofres.

- Personaje: un personaje principal, con las animaciones necesarias.

- Enemigos: varios enemigos, dos de ellos especiales para ser el enemigo intermedio y el enemigo final, con sus animaciones correspondientes.

- Armas: varias armas de diferentes estilos.

- Pociones: dos tipos, de vida y de magia.

- Tesoros: monedas, joyas o algún tipo de elemento que represente tesoros, serán necesarios varios tipos, que darán valdrán diferente al ser recogidos.

- Interfaz del jugador: iconos para representar los diferentes elementos de la misma.

3.4 Eventos importantes

A continuación, se definen algunos de los eventos o situaciones importantes que sean necesarias de implementar en el escenario, ya sea por su función de enseñar al jugador una mecánica básica del videojuego, la utilización de algún elemento del mismo, o simplemente porque planteé un reto interesante para el jugador. A continuación algunos ejemplos pensados, con sus bocetos originales correspondientes:

- Probando la magia: el jugador empezará sin puntos de magia, llegará a un punto del escenario en el que recogerá una poción de magia y se encontrará un enemigo, el cual no podrá golpearle, pero el jugador si podrá lanzarle magias, debido a un obstáculo del camino.

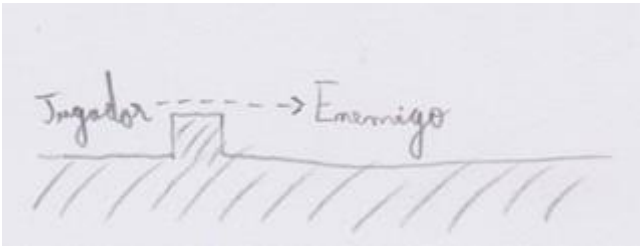

Figura 3: Boceto original, realizado durante el diseño previo, de la escena "Probando la magia".

- Objetos destruibles: el jugador encontrará un tesoro detrás de una barrera, golpeando la barrera descubrirá que es un elemento del mapa con puntos de vida, el cual puede destruir para alcanzar el tesoro.

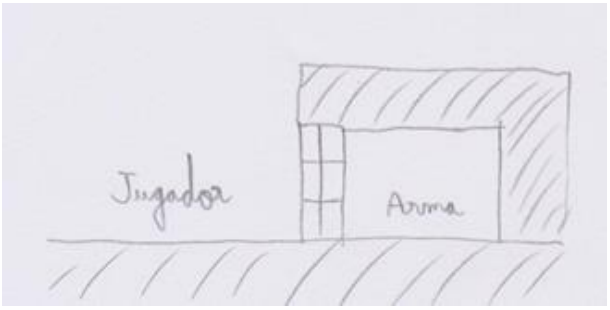

Figura 4: Boceto original, realizado durante el diseño previo, de la escena "Objetos destruibles".

- Mecanismos: el jugador se encontrará con dos plataformas estáticas y una palanca, al pulsar la palanca las plataformas se moverán, y así el jugador podrá alcanzar diferentes tipos de objetivo, como sería un tesoro o poder continuar avanzando hacia su objetivo.

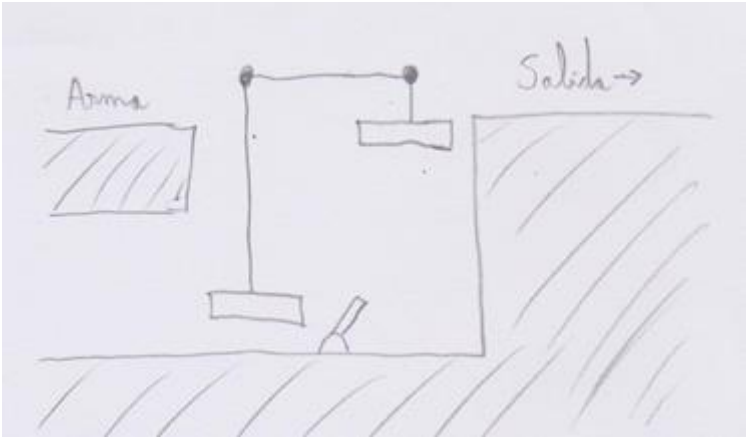

Figura 5: Boceto original, realizado durante el diseño previo, de la escena "Mecanismos".

3.4 Diseño del escenario

Llegados a este punto del análisis y diseño previo al desarrollo del videojuego, es necesario crear un boceto aproximado del mapa a implementar, con los elementos definidos anteriormente repartidos en él.

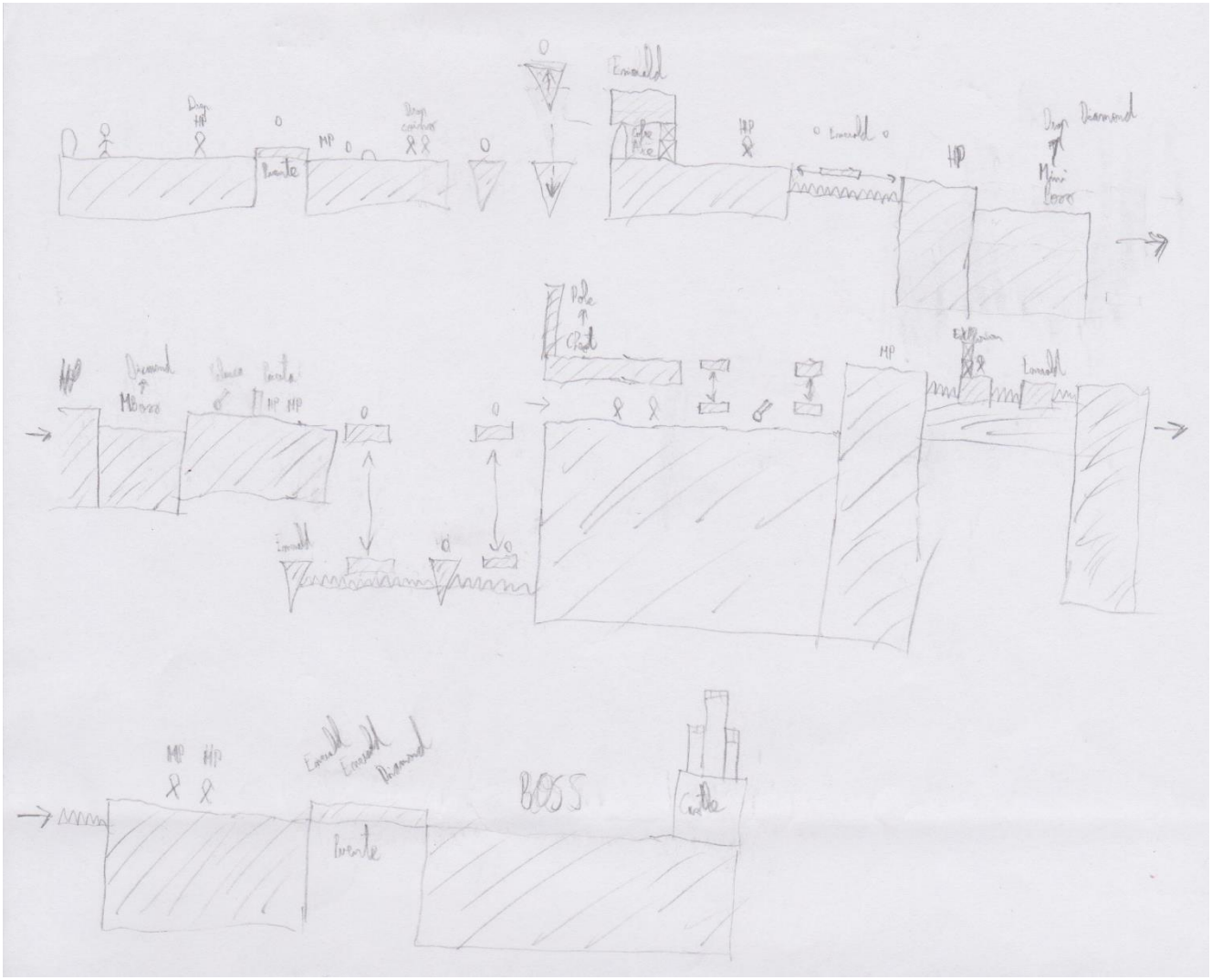

Figura 6: Boceto original, realizado durante el diseño previo, del escenario del juego.

Veo necesario aclarar que, tanto durante la implementación del mapa, como una vez implementado el mapa e iniciado su testeo, las distancias y elementos u objetos repartidos por el escenario sufrirán modificaciones, con el objetivo de mejorar y pulir la jugabilidad del videojuego de cara al jugador.

3.5 Codificación

La codificación necesaria para este videojuego se realiza, como he indicado anteriormente, mediante scripts en lenguaje C Sharp. El funcionamiento de dichos scripts en Unity se basa en que cada uno de ellos implementa una clase, con sus métodos y sus parámetros, dentro del proyecto.

Como ya aclaré en un apartado anterior, cuando empecé inicialmente este proyecto, tenía en mente un videojuego de disparos con estética militar postapocalíptica, debido a este hecho se pueden encontrar varios scripts, clases y variables con nomenclatura propia de esta temática. Más adelante, cuando analicé en profundidad el diseño que quería a nivel jugable y estético, cambié a la temática y género actuales, por los motivos ya explicados en el primer apartado de esta sección de análisis.

Dentro del listado de scripts a desarrollar se encuentran, a grandes rasgos, los siguientes:

- Jugabilidad: scripts que controlen los elementos básicos del funcionamiento del juego, como son: la cámara, el menú de pausa, la muerte del jugador, el reinicio y el final del videojuego, etc.

- Mapa: controlar los elementos dinámicos que haya en el escenario, como serían plataformas móviles, mecanismos que activen ciertos elementos (palancas y puertas), explosiones, etc.

- Personaje: controlar las acciones del personaje y las interacciones con los objetos que recoja y los enemigos que se encuentre. Así como implementar un sistema de combate tanto cuerpo a cuerpo, como a distancia. Y un sistema de inventario para gestionar las armas de las que dispone.

- Enemigos: definir la inteligencia artificial de los enemigos, así como los parámetros que los definan y su interacción con el personaje.

- Recolectables: controlar la interacción de estos elementos (pociones, tesoros y armas) con el jugador y lo que le aportan al mismo.

4 Diseño y desarrollo

4.1 Jugabilidad

Como base de funcionamiento del videojuego, he visto necesario implementar los scripts mostrados en la figura 7.

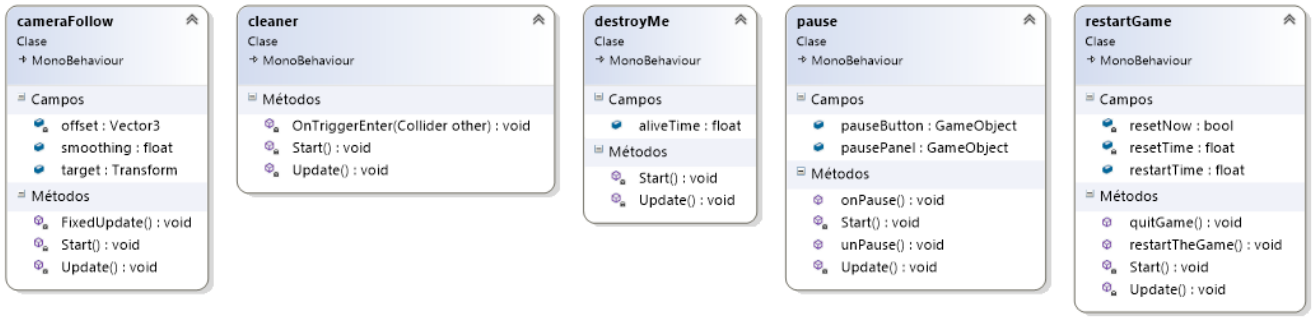

Figura 7: Scripts que afectan al juego de forma global.

La utilidad de estos scripts es la siguiente:

- cameraFollow: se encarga de que la cámara del juego siga de forma automática y fluida la posición del personaje principal, controlado por el jugador.

- cleaner: se encarga de que cuando el personaje caiga al vacío, este sea eliminado del juego.

- destroyMe: se encarga de la autodestrucción de algunos objetos pasado un tiempo determinado. Por ejemplo, se utiliza para que las bolas de fuego que no impacten con nada en un tiempo específico sean eliminadas y no vuelen infinitamente.

- pause: se encarga de pausar el juego cuando se pulsa el botón correspondiente y mostrar la ventana con los controles del juego, así como de retomar el juego al salir de dicha ventana.

- restartGame: se encarga de reiniciar el juego cuando el jugador es eliminado o cuando se completa el juego. También se utiliza para cerrar el juego cuando se pulsa el botón correspondiente en la interfaz del usuario.

4.2 Mapa

El escenario imaginado previamente, una vez diseñado, implementado y pulida su jugabilidad ha quedado con la estructura mostrada en la figura 8.

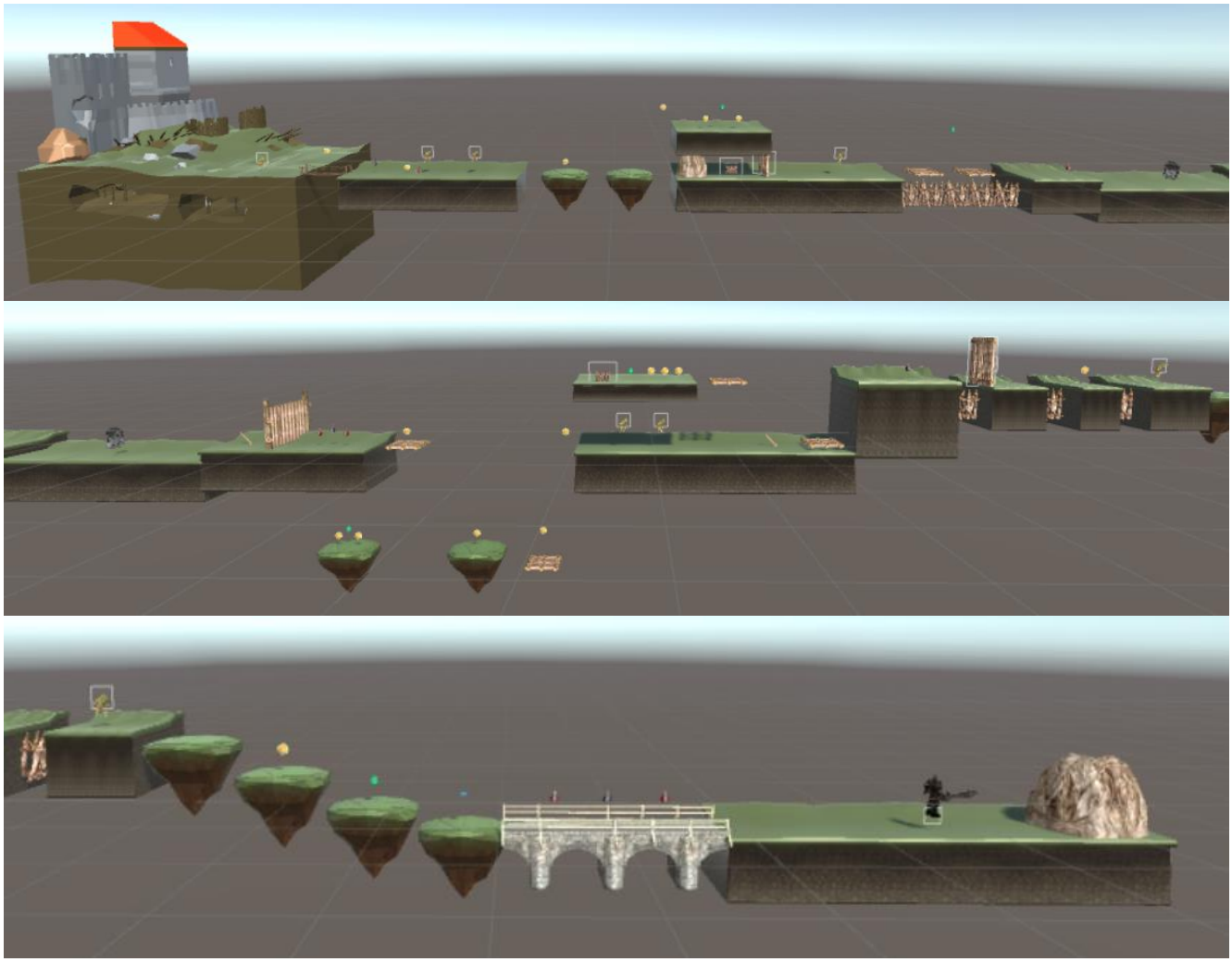

Figura 8: Escenario del juego

Se pueden observar los diferentes elementos que han sido necesarios para su creación: un terreno fijo, un pequeño terreno móvil, una plataforma móvil de madera, un cofre, unos pinchos de madera, una barrera de madera, una palanca de madera, varias rocas y dos puentes.

Para hacer funcionar la interacción del jugador con diversos elementos del mapa se han creado los scripts mostrados en la figura 9.

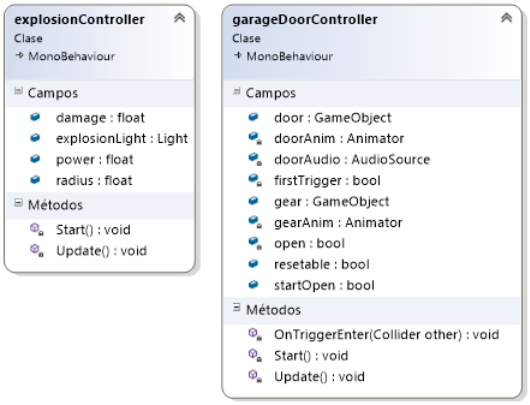

Figura 9: Scripts que afectan a elementos del mapa.

En este caso, la clase “explosionController” se encarga de activar y gestionar la explosión creada al destruir un elemento destruible del mapa. Y, por otro lado, la clase “garageDoorController” se encarga de hacer funcionar el mecanismo que hace que ciertas puertas o plataformas se activen al usar una palanca.

4.3 Personaje

El personaje controlado por el jugador es un caballero con armadura, el cual tendrá un arma equipada (explicado en el siguiente apartado) y podrá andar, correr y saltar, además de atacar físicamente con dicha arma y a distancia mediante bolas de fuego mágicas. Se puede observar el modelo implementado en la figura 10.

Figura 10: Modelo del personaje, en Blender sin texturizar y en Unity texturizado.

Todas estas funcionalidades del personaje están gestionadas por los scripts mostrados en la figura 11.

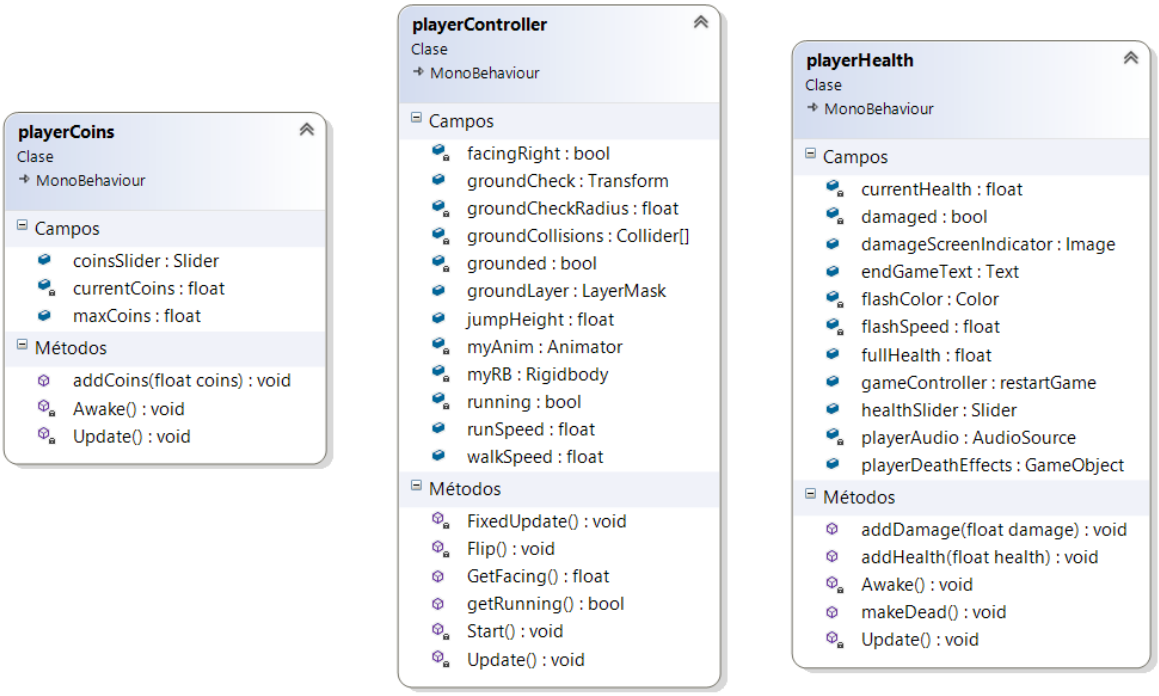

Figura 11: Scripts que afectan al personaje.

El script principal es “playerController” el cual se encarga de controlar todos los movimientos del personaje, además de sus colisiones con el escenario y las físicas del mismo. Asimismo, con esta clase se define la velocidad de andar y correr del personaje y la altura a la que podrá saltar.

Por otro lado, el script “playerHealth” controla los puntos de vida que tiene el personaje en cada momento y actualizar esos datos en la interfaz del jugador. Esta clase contiene métodos encargados de añadir vida cuando el jugador recoja pociones de vida o de restar vida cuando el jugador sea atacado por un enemigo. También será el encargado de mostrar un indicador en pantalla al recibir daño y mostrar el mensaje correspondiente de final de partida cuando el jugador muera y reiniciar el juego cuando esto ocurra.

Por último, la clase “playerCoins” gestiona los tesoros que tiene el jugador en cada momento y actualiza dicha cantidad cuando recoge más de estos elementos por el escenario.

4.4 Armas

Finalmente en el juego se han implementado tres armas diferentes, cada una de un tipo y con sus propios parámetros a nivel jugable. El jugador empezará con la espada y podrá conseguir las otras dos según avance por el escenario. Las armas implementadas se muestran en la figura 12.

Figura 12: Modelos de las armas, en Blender sin texturizar y en Unity texturizadas.

Son necesarios varios scripts para implementar el sistema de ataque cuerpo a cuerpo y los proyectiles mágicos a distancia, estos scripts están vinculados a cada arma de manera individual y permiten la inclusión de nuevas armas de forma muy fácil y rápida, sin tener que modificar el personaje; también ha sido necesario crear scripts para gestionar el sistema de inventario de armas. Todos estos scripts se muestran en la figura 13.

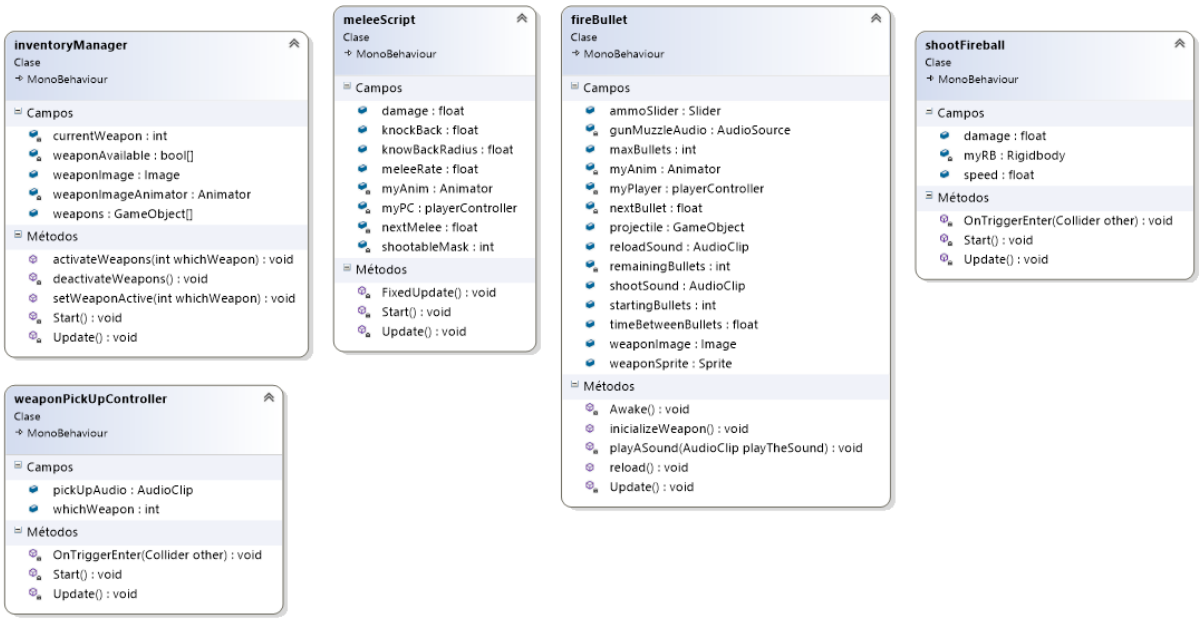

Figura 13: Scripts que afectan a lar armas.

En primer lugar, está el sistema de ataque a corta distancia, el cual es controlado por la clase “meleeScript” y gestiona desde su interacción con los enemigos u objetos destruibles del escenario al realizar el ataque, hasta los parámetros que definen cada arma.

En este aspecto, se han dado unos valores específicos a cada arma intentando que cada una tenga una potencia y funcionalidad acorde a su estilo; y así conseguir que cada arma tenga unas ventajas y desventajas, y aporten variedad a la jugabilidad dando al jugador diferentes estilos de combate. Dichos valores se muestran en la tabla 1.

Tabla 1: Atributos de las armas (script: “meleeScript”).

	Damage	Melee Rate	Knock Back	Knock Back Radius
Espada	20	0.5	10	2.5
Hacha	30	0.75	15	1.5
Lanza	25	0.66	10	4

Por otro lado, el sistema de ataques mágicos a distancia está controlado por dos scripts. Un primer script llamado “fireBullet”, encargado de gestionar los puntos de magia y sus actualizaciones en la interfaz cuando se recuperen o gasten dichos puntos, al recoger pociones y lanzar magias, respectivamente. Asimismo, también es el encargado de todos los sonidos relacionados con los ataques a distancia.

El segundo script encargado de esto es la clase “shootFireball”, vinculada a un proyectil en forma de bola de fuego y que define el daño y la velocidad de dicho proyectil.

Por último, el sistema del inventario de armas está controlado por un script llamado “inventoryManager”, con esta clase se definen la cantidad de armas que hay disponibles en el juego, con sus modelos e iconos correspondientes. A su vez, permite definir cuál es el arma inicial que está activa, se encarga de activar para su uso las armas que va recogiendo el jugador por el escenario y permite cambiar de un arma a otra dentro del juego.

Junto a esta última clase, tenemos otro script llamado “weaponPickUpController”, que es el script que está vinculado a las armas repartidas por el mapa y se encarga de activar el arma correspondiente cuando se recoja.

4.5 Enemigos

Finalmente se han integrado en el juego tres enemigos diferentes, cada uno con sus animaciones y atributos correspondientes. El primero de ellos es un goblin y es el que utilizo como enemigo estándar, el cual se muestra en la figura 14.

Figura 14: Modelo de enemigo estándar, en Blender sin texturizar y en Unity texturizado.

El segundo enemigo es un golem de piedra y representa un enemigo especial que el jugador tendrá que derrotar cuando llegue a la mitad del escenario. Dicho enemigo se puede observar en la figura 15.

Figura 15: Modelo del jefe intermedio, en Blender sin texturizar y en Unity texturizado.

Por último, el tercer enemigo implementado es un caballero oscuro que hace de enemigo final del juego, al cual el jugador debe derrotar para pasarse el juego. Este enemigo se puede ver en la figura 16.

Figura 16: Modelo del jefe final, en Blender sin texturizar y en Unity texturizado.

Para controlar el sistema de interacción de los diferentes enemigos con el escenario y el personaje principal, así como definir sus atributos y su inteligencia artificial, entre otras funcionalidades, he implementado diversos scripts, mostrados en la figura 17. Estos scripts se pueden vincular individualmente a cualquier enemigo adicional que se quiera añadir al juego de forma rápida y sencilla, lo cual permite una buena escalabilidad si el juego crece y se requiere tener una variedad mayor de enemigos.

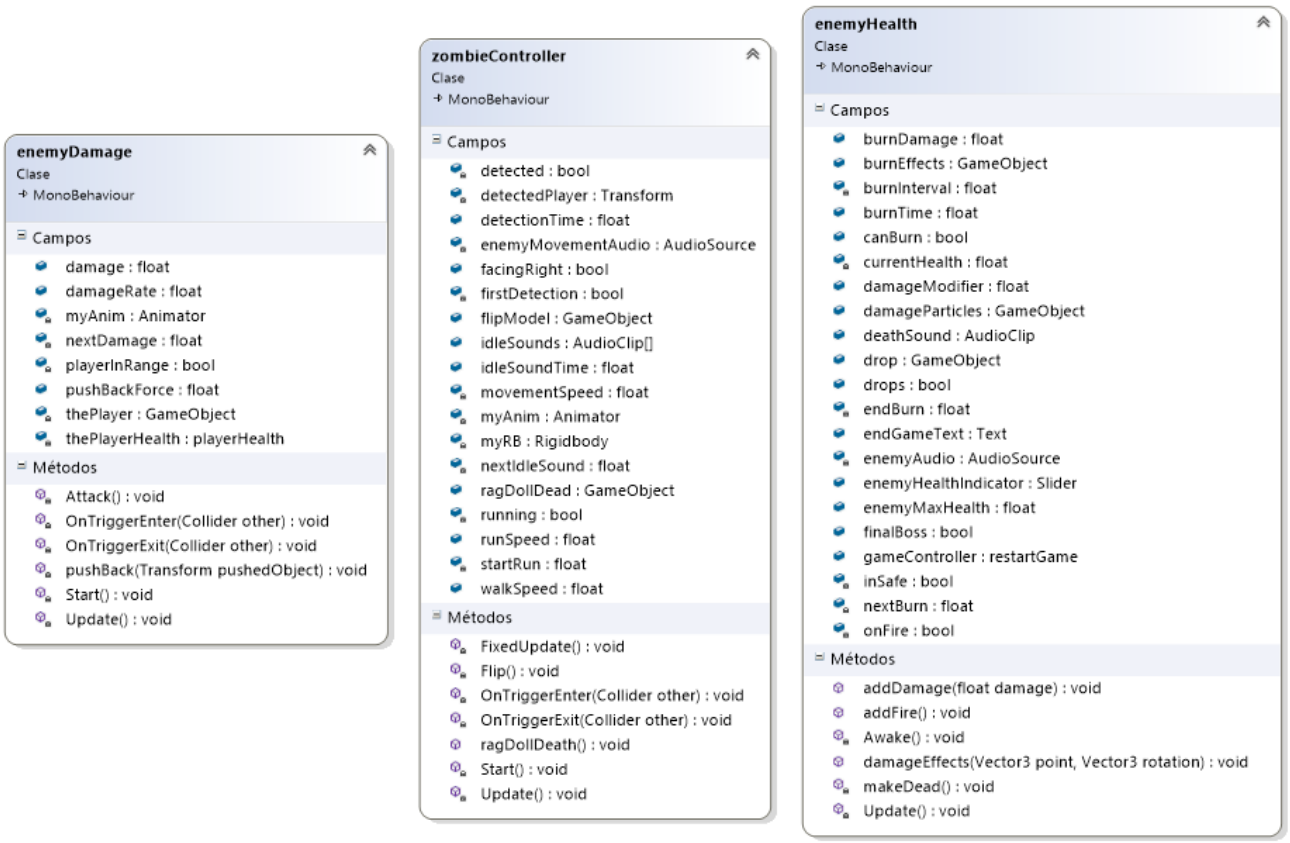

Figura 17: Scripts que afectan a los enemigos.

En primer lugar, el script “zombieController” será el encargado principal del correcto funcionamiento de los enemigos, este script define los atributos básicos del enemigo y controla los componentes básicos del mismo, como es su movimiento, sus animaciones y sus sonidos. Además, es el que gestiona la inteligencia artificial de los enemigos: el rango de detección para cuando se acerca el personaje, el tiempo de tardan en detectarlo cuando entra en dicho rango y las acciones que realizarán cuando esto ocurra.

Por otro lado, el script “enemyDamage” se encarga de definir el daño y la velocidad de ataque de los enemigos, así como de gestionar la aplicación de dicho daño a lo largo del tiempo cuando entra en contacto con el jugador.

Por último, el script “enemyHealth” controla la vida de los enemigos y se encarga de actualizarla cuando este recibe daño, incluido el indicador de vida mostrado por pantalla. Este script también define si el enemigo puede entrar en estado de quemado, lo cual dura un tiempo determinado y causa pequeños daños en ese tiempo. Asimismo, controla cuando el enemigo se queda sin vida y muere, y si dicho enemigo deja un objeto en el escenario al ser derrotado, así como de definir cuál es dicho objeto, como puede ser una moneda o una poción.

Para ofrecer una buena y coherente experiencia de juego de cara al jugador, los atributos que definen a los diferentes enemigos del escenario se han configurado con los valores mostrados en la tabla 2.

Tabla 2: Atributos de los enemigos (scripts: “zombieController”, “enemyDamage” y “enemyHealth”).

	Health	Damage	Damage Rate	PushBack Force	Detection Time	Walk Speed	Run Speed	Fire	Drop
Goblin	100	5	1	10	1	2	7	Si	Moneda
Golem	200	10	1.5	10	1.5	1.5	5.5	No	Diamante
Caballero	250	15	2	10	2	2	6	Si	Diamante

4.6 Recolectables

A lo largo del escenario hay repartidos diversos elementos que el jugador puede recoger para conseguir diferentes ventajas. Estos objetos se pueden diferenciar en dos clases: por un lado, existen las pociones de vida y magia, las cuales recuperan puntos de vida y magia, respectivamente. Dichas pociones se muestran en la figura 18.

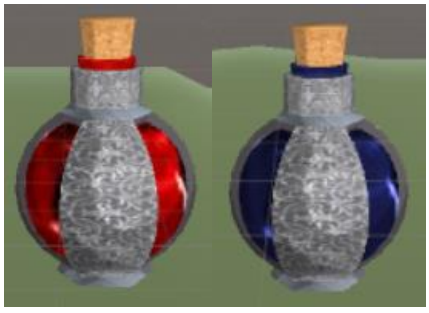

Figura 18: Pociones de vida y màgia en Unity.

La segunda clase de objetos que el jugador puede encontrar son los tesoros, dentro de estos hay tres diferentes implementados: las monedas, las esmeraldas y los diamantes. Cada uno de ellos se han definido con un valor diferente según lo difícil que sea su obtención en el juego: las monedas valen 1, las esmeraldas valen 10 y los diamantes valen 25. Estos tesoros se muestran en la figura 19.

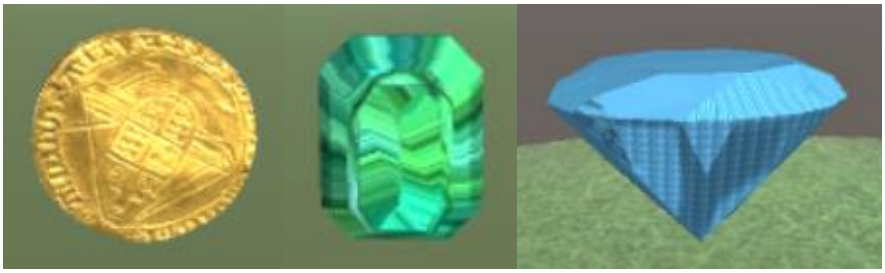

Figura 19: Tesoros del juego (moneda, esmeralda y diamante) en Unity.

Todos estos elementos recolectables por el mapa son gestionados por un script u otro, según el tipo de objeto que sea. Dichos scripts se pueden observar en la figura 20.

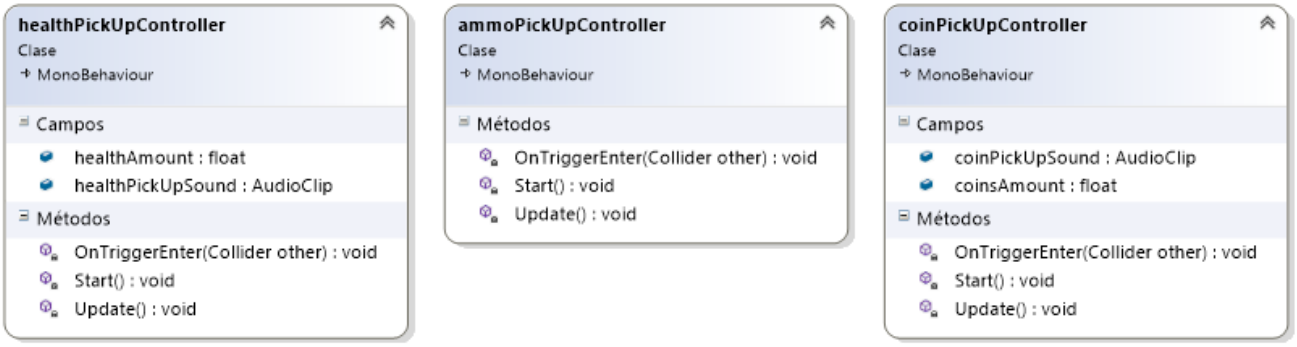

Figura 20: Scripts que afectan a los objetos recolectables.

Por un lado, las pociones de vida tienen vinculado el script “healthPickUpController”, el cual activa su efecto cuando el jugador entra en contacto con el objeto, así como define la cantidad de vida que recupera dicha poción al recogerla y su sonido de recogida correspondiente.

Por otro lado, en el caso de las pociones de magia el encargado es el script “ammoPickUpController”, que hará la misma función que el anterior script, pero en este caso regenerará todos los puntos de magia directamente.

Y, por último, existe el script “coinPickUpController” que controlará los diferentes tesoros y su interacción con el personaje al recogerlos, así como el valor de cada uno y su sonido de recogida.

4.7 Interfaz de usuario

He implementado una interfaz de usuario con el mínimo de elementos que he creído necesarios para la correcta jugabilidad del juego, así he conseguido una interfaz limpia, que permite al jugador centrarse en la acción del juego, controlando de forma rápida los parámetros importantes. Esta interfaz básica del juego se puede observar en la figura 21.

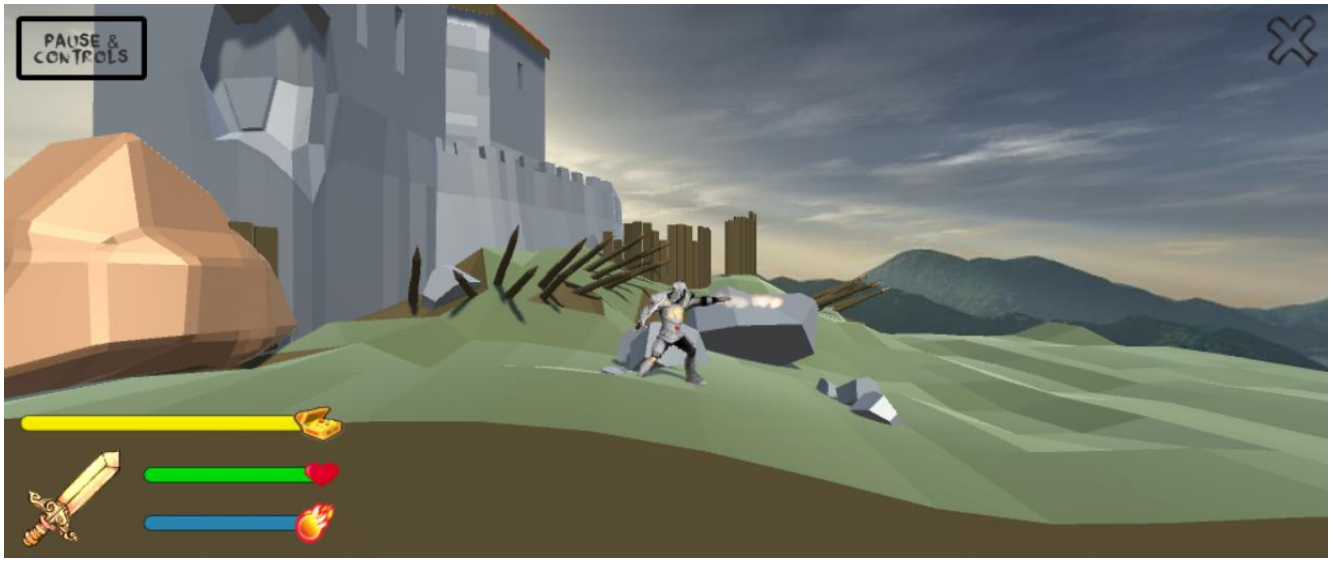

Figura 21: Interfaz de usuario.

Analizando en profundidad esta interfaz, se puede dividir en tres zonas claras:

- Estado del jugador: en la zona inferior izquierda se muestra el arma equipada en cada momento. Además, se muestran los puntos de vida y de magia, así como la puntuación obtenida mediante la recolección de tesoros, utilizando unas barras de forma clara.
- Pausa y controles: en la zona superior izquierda hay un botón para pausar el juego a la vez que se abre un panel donde se muestran los controles del juego.
- Cerrar: en la zona superior derecha hay un botón para cerrar el juego.

4.8 Lógica: elementos a destacar

Entre todos los scripts y funcionalidades desarrolladas para el juego, explicadas brevemente durante este capítulo de la memoria, veo necesario profundizar en el funcionamiento de algunas de ellas que considero especialmente importantes.

En primer lugar, hay que aclarar que el motor de videojuegos Unity funciona mediante componentes, los cuales se añaden y vinculan a cada uno de los elementos integrados en el juego para así conseguir el funcionamiento deseado. Mediante este sistema, se vinculan a los recursos implementados los componentes que se necesiten: scripts, colliders (colisionadores), fuentes de audio, controladores de animaciones…

Uno de los componentes que más he utilizado para el desarrollo han sido los colliders, estos componentes, combinados con los métodos desarrollados en los scripts, me han permitido controlar todas las colisiones que se encuentran en el juego: el personaje con el suelo y los elementos del escenario, el sistema de combate entre el personaje y los distintos enemigos, el sistema de detección de los enemigos que se activan cuando el personaje entra en su rango, la recogida de elementos recolectables del mapa, etc.

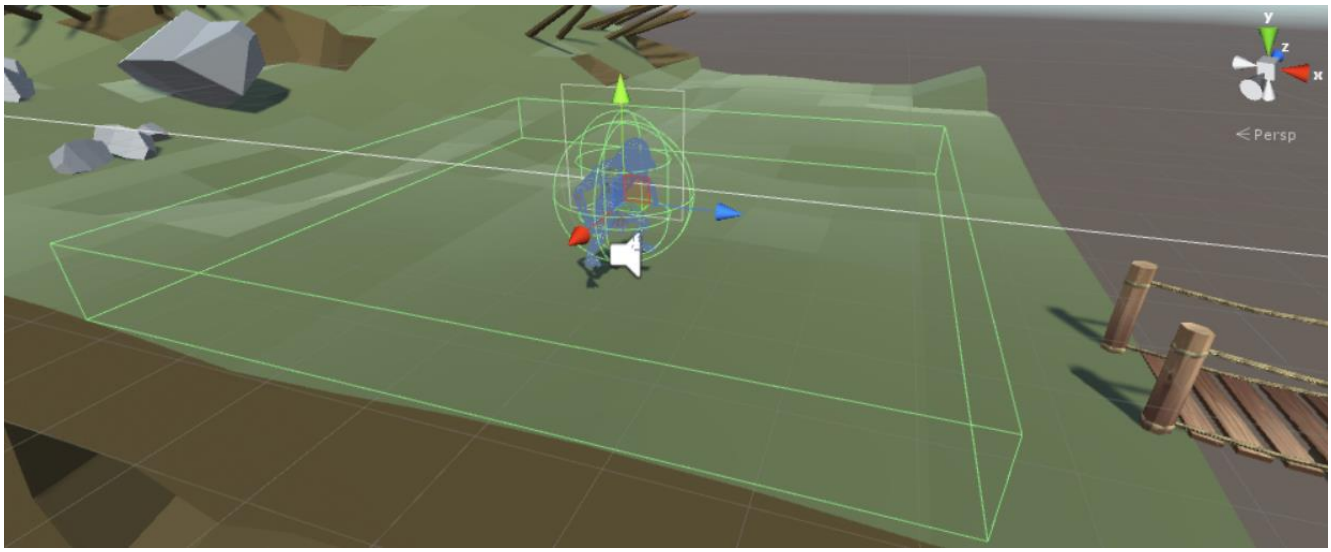

Figura 22: Colliders utilizados para gestionar las colisiones del enemigo goblin.

Un buen ejemplo de su funcionamiento es la forma en que los utilizo para gestionar las distintas colisiones de un enemigo: un enemigo cualquiera del juego tiene tres colliders implementados y configurados, observables en la figura 22.

El primero de estos colliders, es uno con forma esférica adaptado a un tamaño un poco menor al del modelo del enemigo, este collider, junto con el script que también tiene vinculado dicho modelo, permiten controlar que el enemigo se vea afectado por las colisiones con el terreno y los elementos del escenario y pueda desplazarse por él, así como ser el área a la que el jugador debe atacar para que el enemigo reciba daño.

Por otro lado, existe un segundo collider de forma esférica, que se adapta al tamaño del enemigo, utilizado para gestionar el ataque del enemigo al personaje principal, de forma que si el personaje está dentro de este collider la inteligencia artificial desarrollada para el enemigo deja de andar o correr, según el estado en que esté, y lanza un ataque hacía el.

Por último, el gran collider de forma rectangular define el área del escenario a la que debe entrar el personaje para ser detectado por este enemigo, cuando esto ocurre la inteligencia artificial del enemigo se activa y este empieza a perseguir al jugador para atacarle, primero andando y, una vez pasado el tiempo de detección configurado, corriendo.

Para dar una idea global del comportamiento definido en los enemigos, gracias a estos colliders y los scripts vinculados a ellos, en la figura 23 se muestra la IA de forma simplificada.

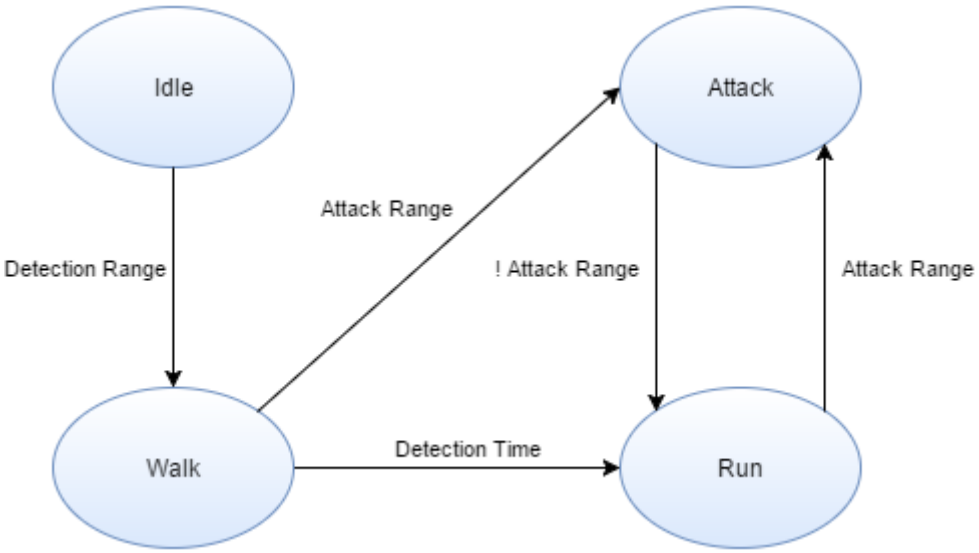

Figura 23: Diagrama simplificado de la IA implementada en los enemigos.

Todo este sistema de interacción entre los enemigos y el personaje principal está controlado mediante varios métodos definidos en los scripts vinculados a los colliders y al enemigo en sí, los principales métodos de dichos scripts se explican a continuación.

```
// FixedUpdate is called once per every fixed framerate frame
void FixedUpdate () {

    if (detected) {
        if (detectedPlayer.position.x < transform.position.x && facingRight)
            Flip ();
        else if (detectedPlayer.position.x > transform.position.x && !facingRight)
            Flip ();

        if (!firstDetection) {
            startRun = Time.time + detectionTime;
            firstDetection = true;
        }
    }

    if (detected && !facingRight)
        myRB.velocity = new Vector3 ((movementSpeed * -1), myRB.velocity.y, 0);
    else if (detected && facingRight)
        myRB.velocity = new Vector3 (movementSpeed, myRB.velocity.y, 0);

    if (!running && detected) {
        if (startRun < Time.time) {
            movementSpeed = runSpeed;
            myAnim.SetTrigger ("run");
            running = true;
        }
    }

    if (!running) {
        if ((Random.Range (0, 10) > 5) && nextIdleSound < Time.time) {
            AudioClip tempClip = idleSounds [Random.Range (0, idleSounds.Length)];
            enemyMovementAudio.clip = tempClip;
            enemyMovementAudio.Play ();
            nextIdleSound = idleSoundTime + Time.time;
        }
    }

}
void OnTriggerEnter (Collider other) {

    if (other.tag == "Player" && !detected) {
        detected = true;
        detectedPlayer = other.transform;
        myAnim.SetBool ("detected", detected);

        if (detectedPlayer.position.x < transform.position.x && facingRight)
            Flip ();
        else if (detectedPlayer.position.x > transform.position.x && !facingRight)
            Flip ();
    }

}

void OnTriggerExit (Collider other) {

    if (other.tag == "Player") {
        if (running) {
            myAnim.SetTrigger ("run");
            movementSpeed = walkSpeed;
            running = false;
        }
    }

}
```

Figura 24: Métodos implicados en la IA enemiga en el script "zombieController".

Primeramente, en la figura 24, se muestran algunos métodos utilizados para la definir la inteligencia artificial del enemigo, en este caso se trata del script “zombieController”, el cual contiene métodos encargados de controlar la detección del personaje por parte de los enemigos.

En ellos se puede ver como se varían las variables pertinentes cuando un collider con la etiqueta “Player” entra en contacto con el collider que tiene definido el enemigo, así como cuando dejan de estar en contacto dichos colliders y un método “FixedUpdate” que se ejecuta en cada frame del juego. En este caso se gestiona, principalmente, la transición entre andar y correr (y sus animaciones) del enemigo para perseguir a dicho personaje, además de algunos audios y la dirección en la que mira el personaje, girándolo cuando sea necesario.

```
// Update is called once per frame
void Update () {
    if (playerInRange)
        Attack ();
}

void OnTriggerEnter (Collider other) {
    if (other.tag == "Player") {
        playerInRange = true;
    }
}

void OnTriggerExit (Collider other) {
    if (other.tag == "Player") {
        playerInRange = false;
    }
}

void Attack () {
    if (nextDamage <= Time.time) {
        myAnim.SetTrigger ("attack");
        thePlayerHealth.addDamage (damage);
        nextDamage = Time.time + damageRate;
        pushBack (thePlayer.transform);
    }
}

void pushBack (Transform pushedObject) {
    Vector3 pushDirection = new Vector3 (0, (pushedObject.position.y - transform.position.y), 0).normalized;
    pushDirection *= pushBackForce;

    Rigidbody pushedRB = pushedObject.GetComponent<Rigidbody> ();
    pushedRB.velocity = Vector3.zero;
    pushedRB.AddForce (pushDirection, ForceMode.Impulse);
}
```

Figura 25: Métodos implicados en la IA enemiga en el script “enemyDamage”.

A continuación, en la figura 25, siguiendo el mismo funcionamiento que en el anterior script, se observan los métodos implicados en que los enemigos ataquen al jugador pertinente cuando este esté dentro de su rango de ataque, mediante sus colliders.

Cuando el personaje entre o salga del rango del enemigo se actualizará la variable correspondiente y, en la siguiente ejecución de “Update” se ejecutará el método “Attack”, encargado de activar la animación correspondiente al ataque, aplicar el daño a la vida del jugador, reiniciar el tiempo para el siguiente ataque y empujar al personaje del jugador con la fuerza que se haya definido.

Por último, el sistema de recibir daño de los enemigos utiliza el collider definido en el enemigo para ello, pero tiene un flujo de ejecución diferente.

En este caso, el encargado de controlar la condicionalidad son los scripts del collider del personaje; por ejemplo, en la situación de que el jugador realice un ataque cuerpo a cuerpo con el arma, el script “meleeScript” gestionará las variables y el estado condicional del collider del personaje con el del enemigo a golpear, y llamará a la función “addDamage” del script “enemyHealth” definido en el collider del enemigo para aplicar el daño causado correspondiente. Estos métodos se muestran en las figuras 26 y 27.

```
void FixedUpdate () {

    float melee = Input.GetAxis ("Fire1");

    if (melee > 0 && nextMelee < Time.time && !(myPC.getRunning())) {

        myAnim.SetTrigger ("gunMelee");
        nextMelee = Time.time + meleeRate;

        Collider[] attacked = Physics.OverlapSphere (transform.position, knowBackRadius, shootableMask);

        int i = 0;
        while (i < attacked.Length) {
            if (attacked[i].tag == "Enemy") {
                enemyHealth doDamage = attacked [i].GetComponent<enemyHealth> ();
                doDamage.addDamage (damage);
                doDamage.damageEffects (transform.position, transform.localEulerAngles);
            }
            i++;
        }
    }
}
```

Figura 26: Método del script “meleeScript” encargado de gestionar el ataque a un enemigo.

```
public void addDamage (float damage) {

    enemyHealthIndicator.gameObject.SetActive (true);
    damage = damage * damageModifier;

    if (damage <= 0f)
        return;

    currentHealth -= damage;
    enemyHealthIndicator.value = currentHealth;
    enemyAudio.Play ();
    if (currentHealth <= 0)
        makeDead ();

}
```

Figura 27: Método del script “enemyHealth” encargado de aplicar el daño recibido al enemigo.

5 Ejecución del proyecto

5.1 Planificación

La planificación de este proyecto se ha basado en la ejecución de las fases explicadas a continuación:

1. Concepto del videojuego: elección del motor de juegos, lenguaje de programación y programa de diseño 3D a utilizar. Asimismo, selección del género, estética y perspectiva del videojuego a desarrollar, entre otros aspectos básicos previos.

2. Unity:
   2.1. Aprendizaje: aprender el funcionamiento de Unity y del lenguaje de programación C Sharp.
   2.2. Desarrollo de la programación: implementación de la parte lógica del juego necesaria para su funcionamiento, utilizando modelos 3D básicos de Unity para ello.

3. Diseño del videojuego: definición definitiva de los aspectos básicos del juego (género, temática y perspectiva), especificación en profundidad de su jugabilidad, definición de los elementos requeridos, diseño del escenario, etc.

4. Blender:
   4.1. Aprendizaje: aprender el funcionamiento de Blender.
   4.2. Creación 3D: creación de los elementos 3D necesarios para el juego, con sus animaciones correspondientes.

5. Implementación:
   5.1. Integración: incorporación de los elementos 3D en Unity y combinación con su lógica correspondiente.
   5.2. Creación del videojuego: creación y configuración del juego diseñado utilizando los diseños 3D y la lógica.
   5.3. Testeo: pruebas y reconfiguración de los componentes del juego para optimizar su jugabilidad.

6. Memoria del proyecto: escritura de la memoria del proyecto.

Esta planificación se muestra ampliada, con la información de los periodos de tiempo planteados inicialmente y con la versión final de dichos periodos, en los diagramas de Gantt de las figuras 28 y 29.

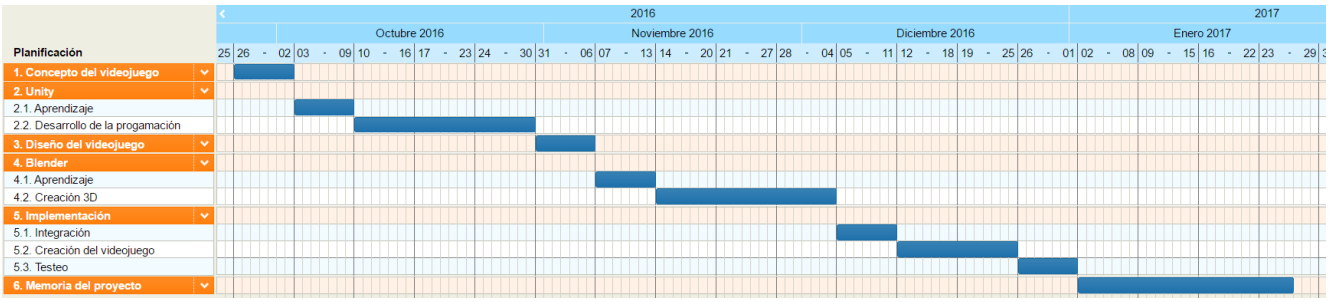

Figura 28: Diagrama de Gantt de la planificación inicial del proyecto.

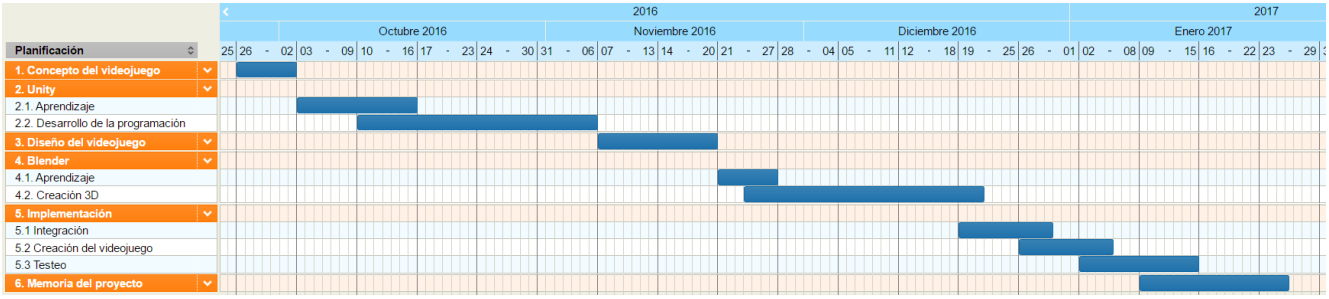

Figura 29: Diagrama de Gantt de la planificación real del proyecto.

Como se puede observar en los diagramas de Gantt mostrados, los tiempos originalmente pensados para cada uno de los puntos del desarrollo han sufrido variaciones según se avanzaba en el proyecto y surgían complicaciones.

Se puede ver como el tiempo necesario para desarrollar la lógica y los diseños 3D del juego son más largas de lo inicialmente imaginado, así como también se alarga la parte del diseño del videojuego, debido a que decidí investigar y desarrollar más ampliamente este campo para crear un producto con una mayor coherencia y jugabilidad.

Debido a esto, las fases de la implementación del videojuego y la escritura de la memoria del proyecto se vieron aceleradas y en parte paralelizadas, aunque este hecho no creó grandes problemas ya que son elementos que permiten su desarrollo simultáneo sin ningún problema.

5.2 Resultados

El producto final obtenido ha sido satisfactorio, considero que se han cubierto los objetivos y requisitos planteados inicialmente de forma correcta, siempre llegando a unos mínimos decentes en todos los aspectos.

Primeramente, la lógica implementada del videojuego permite el correcto funcionamiento del mismo, así como una buena escalabilidad de sus elementos de cara a ampliaciones futuras, como son la implementación de nuevas armas o nuevos enemigos.

Por otro lado, la estética fantástica medieval que se definió para el juego se ha alcanzado de forma correcta, utilizando para ello unos modelos de personaje, enemigos, escenarios y demás elementos adecuados para dicha temática.

Y, por último, el nivel de jugabilidad que se buscaba, siempre teniendo en cuenta que se trata de un producto no completo, limitado por mis conocimientos y por las horas definidas para desarrollar un TFG, considero que se ha cubierto adecuadamente y permite al jugador desarrollar su experiencia de juego de forma positiva.

5.3 Ampliaciones futuras

El juego obtenido como resultado del proyecto cubre la mayoría de sus campos de forma correcta, pero muchos de ellos a un nivel básico de funcionamiento; en un inicio se tenían en mente unas funcionalidades y variedad de ciertos elementos que al final han sido imposibles de alcanzar, por falta de tiempo y carencia de conocimientos sobre el funcionamiento completo del motor Unity o en el ámbito del diseño 3D.

Debido a esto, en caso de que en un futuro se quiera ampliar este proyecto, a continuación se listan varias de las funcionalidades que se pueden añadir al juego para mejorarlo, algunas de

ellas fácilmente aplicables gracias a la escalabilidad y modularidad de los componentes programados:

- Armas: implementar nuevas armas que aporten nuevos estilos de combate, como podría ser un arco.

- Magias: implementar nuevos ataques mágicos de otros elementos (agua, viento, tierra…), los cuales tendrían diferentes efectos sobre los enemigos y serían más o menos contra los enemigos según qué enemigo fuera (un enemigo de fuego se vería más afectado por una magia de agua). Se podría definir que cada arma tuviera un tipo de magia vinculada, para dar más peso a la utilización de varias armas.

- Enemigos: implementar nuevos tipos de enemigos, como enemigos voladores o enemigos estáticos fijos (modo torre defensiva) que ataque a distancia al jugador cuando este se acerque.

- Escenario: ampliar los elementos decorativos del escenario, para así sacar el máximo provecho visual de la utilización de la perspectiva 2.5D.

- Jugabilidad global: implementar puntos de guardado (checkpoints), que hagan que una vez el jugador alcance estos puntos en el mapa, si este muere vuelva a aparecer en ese último punto, en vez de otra vez al inicio del escenario. Esta implementación permitiría crear escenarios más grandes sin perder en jugabilidad.

5.4 Valoración económica

Para realizar una valoración económica aproximada de lo que sería el coste real del desarrollo del proyecto realizado, es necesario definir las horas de trabajo en cada aspecto del proyecto y el sueldo de los trabajadores necesarios.

Por un lado, las horas de trabajo realizadas se pueden extraer del diagrama de Gantt mostrado en la figura 23, definiendo que aproximadamente una semana media se compone de 20 horas de trabajo (4 horas diarias de lunes a viernes).

A continuación, es necesario dividir las diferentes tareas del desarrollo del proyecto en los roles de los trabajadores que serán necesarios para llevarlas a cabo. Para este proyecto se necesitarían: un diseñador de videojuegos (para definir el concepto del juego y su diseño), un programador (para desarrollar la lógica del juego), un diseñador 3D (para generar los modelos necesarios), un desarrollador de videojuegos (para la tarea de la implementación del juego con Unity) y una persona encargada de realizar la documentación del proyecto.

Con estos datos se puede realizar la evaluación del coste del proyecto, la cual se muestra en la tabla 3.

Tabla 3: Evaluación económica del proyecto.

Función	Horas	Precio (€) / Hora	Coste (€)
Diseño del juego	60	25	1.500
Programación	100	15	1.500
Diseño 3D	90	15	1.350
Desarrollo del juego	80	20	1.600
Documentación	50	10	500
	380		**6.450**

6 Conclusiones

El desarrollo completo de un videojuego, aunque sea a pequeña escala, requiere de una buena planificación y distribución de todas las funciones a realizar. Esto es debido a que para la creación de un buen producto son necesarios conocimientos de muchos ámbitos diferentes (programación, diseño 3D, diseño de videojuegos…) para poder crear e implementar todos los elementos necesarios para el juego, por esto mismo, si no se tiene una buena planificación del desarrollo y se tienen claramente definidos sus requisitos, la correcta combinación de estos elementos en un producto unificado y de calidad se vuelve prácticamente una misión imposible.

De esta realidad he sido víctima a lo largo del desarrollo de este proyecto, ya que, aunque yo me encargara de todas las partes del proceso, estar atento a todas las dependencias que se crean entre los distintos campos del desarrollo según este va avanzando es una tarea muy difícil si no se tiene bien clara la planificación y las especificaciones del producto que se quiere, cosa que lógicamente no estaba del todo clara en un inicio, debido a mi falta de experiencia y conocimientos en el campo del desarrollo de videojuegos.

Aun así, estoy satisfecho con el proyecto realizado y el producto desarrollado, considero que tiene un nivel aceptable para haber sido creado por una sola persona y dentro del tiempo disponible para desarrollar un TFG.

Asimismo, considero que se ha demostrado que se han adquirido unos conocimientos suficientes en los diferentes ámbitos del desarrollo de este proyecto. Además de que, en este punto, con más tiempo y recursos a mi disposición se podría mejorar el juego e intentar alcanzar el nivel de un pequeño producto a nivel comercial.

Bibliografía

[1] *Unity – Learn – Tutorials* [en línea]. Unity Technologies, 2016 [consultado en octubre de 2016]. Disponible en: https://unity3d.com/es/learn/tutorials.

[2] *Blender – Support – Tutorials* [en línea]. Blender, 2016 [consultado en noviembre de 2016]. Disponible en: https://www.blender.org/support/tutorials/.

[3] *How to design levels for a platformer* [en línea]. Diorgo Jonkers, 4 de julio de 2011 [consultado en noviembre de 2016]. Disponible en: http://devmag.org.za/2011/07/04/how-to-design-levels-for-a-platformer/.

[4] *11 tips for making a fun platformer* [en línea]. Diorgo Jonkers, 18 de enero de 2011 [consultado en diciembre de 2016]. Disponible en: http://devmag.org.za/2011/01/18/11-tips-for-making-a-fun-platformer/.

[5] *13 more tips for making a fun platformer* [en línea]. Diorgo Jonkers, 19 de julio de 2012 [consultado en diciembre de 2016]. Disponible en: http://devmag.org.za/2012/07/19/13-more-tips-for-making-a-fun-platformer/.

[6] *Sidescroller level design* [en línea]. Ken Bowen, 30 de diciembre de 2012 [consultado en diciembre de 2016]. Disponible en: http://gamedevprofessor.com/sidescroller-level-design/.

[7] *A beginner's guide to designing video game levels* [en línea]. Mike Stout, 26 de enero de 2016 [consultado en diciembre de 2016]. Disponible en: https://gamedevelopment.tutsplus.com/tutorials/a-beginners-guide-to-designing-video-game-levels--cms-25662.

[8] *Practical game design: The rule of threes* [en línea]. Lars Doucet, 3 de abril de 2016 [consultado en diciembre de 2016]. Disponible en: http://www.gamasutra.com/blogs/LarsDoucet/20100304/4562/Practical_Game_Design__The_Rule_of_Threes.php.