UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Proceso de desarrollo de un videojuego 2D

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Botella Perpiñá, Adrián

Tutor/a: Lluch Crespo, Javier

CURSO ACADÉMICO: 2023/2024

Resumen

Este Trabajo Fin de Grado (TFG) se centra principalmente en el proceso de desarrollo de un videojuego 2D aplicando una metodología ágil. Para poder aplicar la metodología correctamente se utiliza una herramienta web (Trello) que nos permite llevar una mejor organización durante el proceso. Para el desarrollo y la programación del videojuego 2D se utiliza la herramienta de desarrollo para videojuegos Unity y el lenguaje de programación C#. El videojuego 2D desarrollado es un juego del género acción-aventura con plataformas y con una temática fantástica medieval que recibe una pequeña inspiración de videojuegos como “Hollow Knight” o “Blasphemous”.

Palabras clave: proceso de desarrollo, videojuego 2D, metodología ágil, Trello, programación, Unity y lenguaje de programación.

Abstract

This Final Degree Project (FDP) focuses mainly on the development process of a 2D video game applying an agile methodology. In order to apply the methodology correctly, a web tool (Trello) is used that allows us to have a better organization during the process. For the development and programming of the 2D video game, the Unity video game development tool and the C# programming language are used. The 2D video game developed is an action-adventure genre game with platforms and a fantastic medieval theme that receives a little inspiration from video games like “Hollow Knight” or “Blasphemous”.

Keywords: development process, 2D video game, agile methodology, Trello, programming, Unity and programming language.

Resum

Aquest Treball Fi de Grau (TFG) se centra principalment en el procés de desenvolupament d'un videojoc 2D aplicant una metodologia àgil. Per a poder aplicar la metodologia correctament s'utilitza una eina web (Trello) que ens permet portar una millor organització durant el procés. Per al desenvolupament i la programació del videojoc 2D s'utilitza l'eina de desenvolupament per a videojocs Unity i el llenguatge de programació C#. El videojoc 2D desenvolupat és un joc del gènere acció-aventura amb plataformes i amb una temàtica fantàstica medieval que rep una xicoteta inspiració de videojocs com “Hollow Knight” o “Blasphemous”.

Paraules clau: procés de desenvolupament, videojoc 2D, metodologia àgil, Trello, programació, Unity i llenguatge de programació.

Índice

Índice de imágenes

Índice de tablas

1 Introducción

Este primer punto tiene por propósito servir como introducción del documento de la memoria describiendo de manera general en qué consiste el trabajo realizado para llevar a cabo el desarrollo del proyecto de este TFG. Tal y como se expone más adelante, este proyecto consiste en un videojuego 2D de género Metroidvania desarrollado de manera colaborativa junto con un compañero de la rama de ingeniería del software, Diego Ruiz Muñoz.

Este proyecto se ha realizado llevando a cabo una metodología ágil, ejecutando diferentes fases de pruebas con la finalidad de verificar y validar que el proyecto se comporta adecuadamente, a la vez que se identifican defectos, fallos o errores y se aplican patrones de diseño y programación respetando un estándar de mantenimiento del software, solucionando de esta manera varios de los problemas que hemos conseguido localizar durante las fases de pruebas del proyecto.

Para llevar a cabo la realización de este proyecto se han utilizado diversas herramientas, destacando entre ellas el motor de videojuegos multiplataforma de desarrollo Unity empleado en el transcurso de la asignatura de Desarrollo de Videojuegos 3D.

En el videojuego desarrollado para este TFG el jugador ha de recorrer el mapa enfrentándose a diferentes enemigos con el objetivo de mejorar al personaje principal y conseguir obtener un mejor equipamiento con el que derrotar a los distintos jefes del videojuego y lograr obtener de esta manera las gemas requeridas para el final del videojuego.

Todos los sistemas desarrollados a lo largo del proyecto cumplen una función importante, tanto para el apropiado desempeño como para la inmersión del jugador en el mismo.

Finalmente, el enlace que se expone a continuación permite visualizar un video donde se muestran de manera general las distintas funcionalidades del videojuego:

https://media.upv.es/#/portal/video/92ff7a60-6235-11ef-8e57-33fa480721ac

Por otro lado, en este enlace a la plataforma web Itch.io, https://diegoger222.itch.io/tfg-diego-adrian, tenemos la posibilidad de ejecutar el videojuego desarrollado para el proyecto de este TFG introduciendo la contraseña “patata”. Los controles del videojuego se pueden consultar en el apartado 6.4 Controles del videojuego de este documento.

1.1 Motivación

Tal y como se nos ha explicado en varias ocasiones a lo largo del Grado en Ingeniería Informática, la informática se encuentra constantemente en evolución abarcando una cantidad de conceptos excesivamente extensa, haciendo necesario dividir este grado en distintas ramas de especialización e imposibilitando en alguna ocasión alcanzar completamente todo lo que se desearía en algunas asignaturas.

En el transcurso de la asignatura de Desarrollo de videojuegos 3D tuvimos la oportunidad de desarrollar un videojuego 3D que recibe el nombre de Wendigo y

presentado durante la feria de proyectos el 16 de diciembre de 2021. Este videojuego se desarrolló aprovechando los conocimientos aprendidos a lo largo de la asignatura relacionados al motor de videojuegos multiplataforma Unity. Desarrollar un videojuego haciendo uso de la herramienta Unity nos resultó una experiencia interesante y única, en cierta medida provocada a causa de que el desarrollo de un videojuego presenta diferencias considerables comparado con el desarrollo de cualquier otra clase de aplicación software debido en su mayor parte a que un videojuego tiene como objetivo divertir y entretener al jugador, no obstante, debido a que esta asignatura cuenta con una duración de únicamente un cuatrimestre nos dio la sensación de no disponer del tiempo adecuado para poder investigar e intentar experimentar todo lo que se hubiera esperado.

Por este motivo, mi compañero Diego Ruiz tomó la decisión de proponerme realizar un proyecto colaborativo para desarrollar un videojuego 2D utilizando el motor multiplataforma Unity. Este proyecto colaborativo nos proporciona la posibilidad tanto de tratar de profundizar más en los conceptos que consideramos que no habíamos sido capaces de explorar y experimentar en el transcurso de la asignatura de Desarrollo de Videojuegos 3D, como de aprovechar completamente los conocimientos aprendidos en las distintas asignaturas que se han cursado a lo largo del Grado en Ingeniería Informática.

1.2 Objetivos

El proyecto realizado para este TFG cuenta con dos objetivos principales que se encuentran interrelacionados entre ellos. El primer objetivo principal consiste en desarrollar una versión jugable de un videojuego 2D de género Metroidvania con una amplia diversidad de escenarios, trampas, enemigos, jefes y objetos aleatorios que proporcionen una considerable variedad a la jugabilidad del videojuego.

El segundo objetivo principal que se planea lograr con este proyecto consiste en conseguir cumplir con el primer objetivo principal llevando a cabo una metodología ágil a lo largo del desarrollo del videojuego con el objetivo de comprobar la eficiencia de esta metodología en un ámbito distinto al que habitualmente se está acostumbrado, permitiendo llevar una mejor organización y seguimiento de las distintas tareas a realizar durante el desarrollo.

Estos objetivos se deben alcanzar utilizando todo lo aprendido en el transcurso de las distintas asignaturas que se han cursado en el Grado de Ingeniería Informática, poniendo en práctica estándares de mantenimiento del software, patrones de diseño y programación y llevando a cabo distintas clases de pruebas.

Los objetivos secundarios en los que deriva el primer objetivo principal del proyecto son:

- Desarrollar un tutorial que proporcione al jugador la posibilidad de probar y acostumbrarse a los diferentes controles del videojuego.
- Diseñar un mapa con distintas ambientaciones impulsando la inmersión del jugador y consiguiendo por lo menos una zona que fuerce al jugador a explorar enfrentándose a distintos enemigos con la finalidad de poder obtener un mejor equipo.
- Desarrollar distintos enemigos y jefes para poder enfrentar, cada uno de ellos con sus diferentes estadísticas y habilidades especiales que fomentan la diversidad del videojuego.

- Diseñar distintas trampas con diferentes patrones de activación que poder distribuir por el mapa con el objetivo de añadir dificultad del videojuego.
- Crear una considerable variedad de objetos con distintos efectos y estadísticas que favorezcan la diversidad, la jugabilidad y la experiencia de juego ayudando al jugador en el transcurso del videojuego.

1.3 Metodología

Este proyecto se ha llevado a cabo aplicando una metodología ágil [1] [2] basada en una combinación de los sistemas Scrum y Kanban y haciendo uso de la herramienta web Trello con la intención de llevar una mejor organización y seguimiento de las tareas tomando en consideración la capacidad de trabajo que puede abordar cada uno de los miembros del equipo.

Durante el desarrollo de este proyecto se han llevado a cabo seis *sprints* (desde el *sprint* 0 hasta el *sprint* 5) de dos semanas de duración cada uno. Cada una de las tareas realizadas a lo largo del proyecto se ha realizado durante alguno de estos *sprints*. A continuación, se presenta una imagen de los *sprints* en el tablero Kanban de la herramienta web Trello:

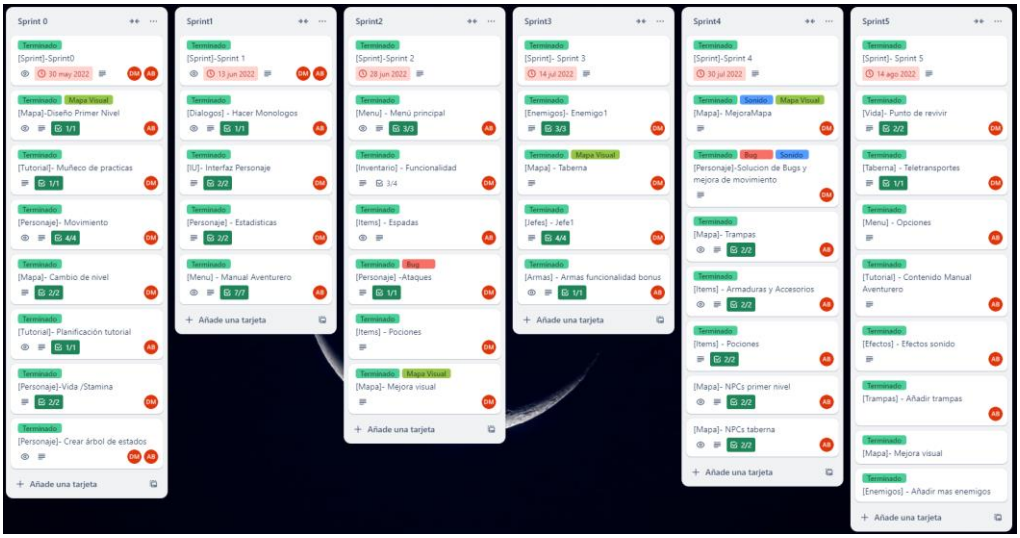

Imagen 1: Sprints

Con la intención de poder llevar una buena organización y seguimiento de estas tareas se ha dividido el tablero Kanban de la herramienta web Trello en diferentes columnas, por las que debe de pasar cada una de las tares tareas, representadas mediante tarjetas, antes de considerar la tarea como terminada. En orden, tal y como se puede apreciar en la Imagen 2 las columnas en las que se ha dividido el tablero Kanban de la herramienta web Trello son las que se exponen a continuación:

- **Backlog:** Esta columna acumula todas las tareas que se han pensado y extraído del análisis de requisitos del proyecto, pero que todavía no se han planificado, siendo posible que alguna de ellas no llegue a realizarse.
- **Next Sprint:** En esta columna se listan las tareas que se han planificado para realizar en el próximo *sprint*. Una vez que el *sprint* actual se encuentre finalizado todas las tareas de esta lista deberán de pasar a la columna Sprint Backlog.

- **Sprint Backlog:** Esta columna almacena las tareas que se encuentran planeadas para este *sprint*, pero que todavía no se ha comenzado con su desarrollo.
- **Programar:** En esta columna podemos encontrar las tareas que se están realizando en el momento actual.
- **Aplicar Pruebas:** Esta columna se utiliza para determinar que tareas se encuentran pendientes de pasar las diferentes pruebas, con el objetivo de considerar la tarea como terminada.
- **Done:** En esta columna se conservan todas las tareas que se consideran terminadas del *sprint* actual una vez que han pasado de manera satisfactoria todas las pruebas.
- **Sprints terminados:** Una vez se acaban las dos semanas de duración del *sprint* todas las tareas de la columna Done pasan a guardarse en una columna con el nombre del *sprint* con el objetivo de poder revisar las tareas que se realizaron durante ese *sprint*.

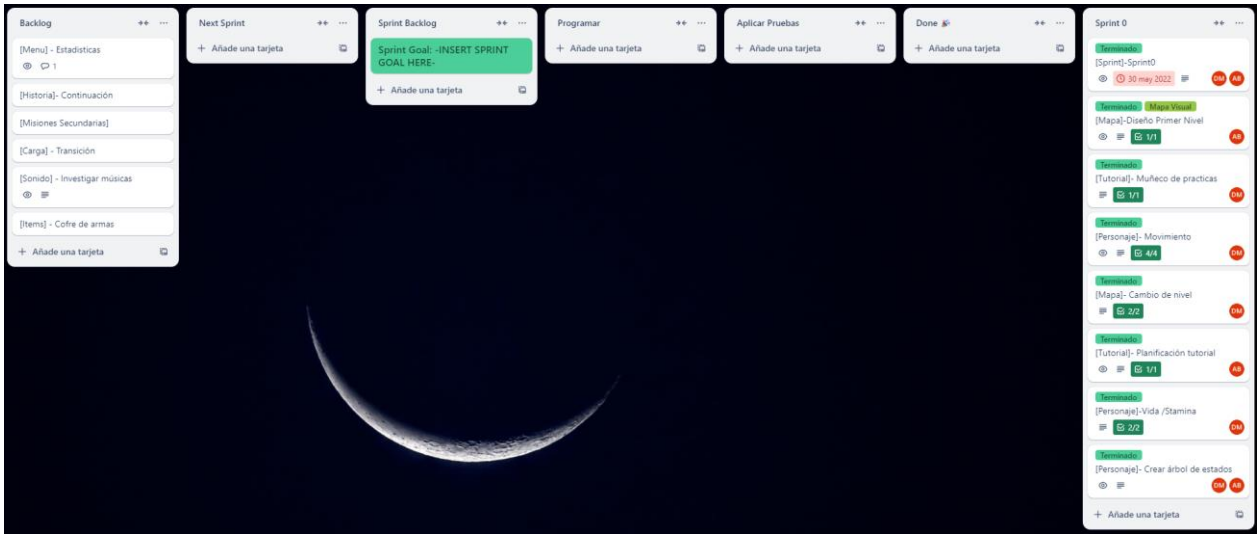

Imagen 2: Tablero Kanban

1.4 Estructura

La estructura de la memoria ha sido redactada con la intención de hacer posible comprender el propósito y funcionamiento de todas las partes del proyecto, partiendo de una base de conocimientos más generales para posteriormente detallar en profundidad conceptos relacionados más internamente con el proyecto.

Este primer punto consiste en una introducción al proyecto, exponiendo de manera general la forma en la que se ha desarrollado y explicando la metodología empleada, los objetivos que se planean cubrir con el proyecto y el motivo que incentiva la creación del mismo. Adicionalmente se ha incluido un apartado con la finalidad de mencionar al equipo completo del trabajo, así como indicar detalladamente las partes del proyecto que ha realizado cada uno de los miembros que han participado en la realización del mismo.

El segundo punto del documento contiene una aclaración del estado del arte del proyecto, al mismo tiempo que se exponen ciertas definiciones esenciales con el objetivo de comprender algunos conceptos relacionados con el proyecto, mostrando

varios ejemplos que sirvieron de inspiración para la realización del mismo y presentando la propuesta del proyecto que lo distingue de los demás.

El tercer punto de la memoria muestra las distintas herramientas que se han utilizado durante el desarrollo del proyecto justificando resumidamente el motivo por el que se han elegido utilizar.

El cuarto punto trata de explicar detalladamente una serie de aspectos relativos a la herramienta Unity profundizando en ciertos conceptos que se tratan posteriormente en el desarrollo del proyecto con el propósito de comprender mejor su funcionamiento.

El quinto punto consiste en el análisis detallado de los distintos requisitos, tanto funcionales como no funcionales, del proyecto.

El sexto punto contiene una explicación acerca de los distintos diseños que se han usado a lo largo del desarrollo del proyecto sin entrar demasiado en detalle en el desarrollo de los mismos.

El séptimo punto consiste en la descripción del desarrollo e implementación del proyecto exponiendo el funcionamiento y explicando los diferentes scripts que conforman las distintas partes y elementos del mismo.

El octavo punto contiene las distintas pruebas que se han llevado a cabo, así como los defectos, fallos y errores que se han logrado detectar durante estas pruebas, las soluciones que se han aplicado y las ideas y sugerencias que se obtuvieron por parte de los usuarios externos.

El noveno punto muestra la conclusión del documento de la memoria de este TFG en la que podemos encontrar resumidos cada uno de los objetivos alcanzados durante el proyecto y una reflexión sobre la relación del proyecto del TFG con las asignaturas cursadas a lo largo del Grado en Ingeniería Informática.

En el décimo y último punto del documento se presenta la bibliografía del documento, donde se muestran los distintos lugares que se han utilizado para obtener información adicional con la intención de mejorar la información expuesta en la memoria.

Finalmente, también podemos encontrar un anexo que trata el grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS) y muestra una reflexión sobre esta relación.

1.5 Colaboraciones

Para llevar a cabo el desarrollo de este proyecto se ha contado con el apoyo de Diego Ruiz Muñoz. Seguidamente, se muestran dos tablas con la intención de indicar el trabajo realizado por cada uno de los miembros del equipo. En la Tabla 1 se destaca que parte del proyecto ha realizado cada miembro, mientras que en la Tabla 2 se presentan los distintos scripts del videojuego indicando que miembros del equipo han participado en el desarrollo de los mismos.

Parte del proyecto	Miembros del equipo	
	Adrián Botella Perpiñá	Diego Ruiz Muñoz
Creación de objetos	X	X
Manual Aventurero	X	
Menú Principal	X	
Personaje Principal		X
Enemigos		X
Ataques Especiales	X	
Diálogos/Monólogos	X	
NPCs	X	
Estadísticas		X
Inventario		X
Misiones	X	X
Música y Efectos de sonido	X	X
Trampas	X	
Historia	X	X
Mapa	X	X
Interfaz de Usuario		X
Tutorial	X	
Opciones	X	

Tabla 1: Partes del proyecto

Scripts	Miembros del equipo	
	Adrián Botella Perpiñá	Diego Ruiz Muñoz
Objetos		
IModifier		X
ItemControlador		X
ItemObject	X	X
Manual Aventurero		
ManualAventurero	X	
Menú Principal		
MenuPrincipal	X	
Personaje Principal		
BarraDeVida		X
ControladorPersonaje		X
Dano		X
Escudo		X
Mana		X
MovimientoPersonaje		X
AtacandoBool		X
AtacandoBoolEspecial		X
AtacandoBoolEspecialSalida		X
DesactivarColisiones		X
Enemigos Normales		
ArbolCont		X
DanyoVisible		X

DanyoEnemigo		X
EstadisticasEnemigo		X
MovimientoCenti		X
Muerte		X
QuietoTime		X
SeguirJugador		X
Vida		X
Enemigos Pacíficos		
Cuervo		X
CuervoGrito		X
CuervoVuelo		X
CuervoVuelta		X
MovimientoCaw		X
Jefes		
CastEnem		X
CastHab		X
DanyoHabi		X
JefeControl		X
JefeNumAlt		X
MovimientoAlmas		X
MovimientoJefe		X
VidaJefe		X
Ataques Especiales		
AtaqueEspecial	X	
Destroy	X	
LanzarEfecto	X	
LanzarMeteorito	X	
Diálogos/Monólogos		
ControlDialogos	X	
Textos	X	
NPCs		
PersonajeInteractuable	X	
Estadísticas		
Estadisticas		X
Experiencia		X
PuntosEstadisticas		X
Inventario		
InformacionInventario		X
Inventory		X
Slot		X
SlotEquipo		X
ControladorInventario		X
Misiones		
Comprobador	X	
ComprobadorMision	X	

Música y Efectos de sonido		
ActivadorSonidos	X	
EjecutarSonido	X	X
ControladorSonido	X	
Trampas		
ActivarTrampa	X	
ActivarTrampaOso	X	
DestroyThis	X	
TrampaPinchoMovil	X	
DanyoTrampa	X	
Mapa		
OcultarZona		X
Elevador		X
Palanca		X
Portales		X
Cofres	X	X
InicioJugador		X
SalidaJefes		X
SalidaJugador		X
TorreCheck		X
TorreCuracion		X
PilarTp		X
Interfaz de usuario		
ControlIUPortales		X
ControladorEventos		X
DanyoIU		X
MovimientoIU		X
ControladorCanvas		X
Opciones		
MostrarNumero	X	
Settings	X	
Cámara		
ControladorCMvcam		X
ControladorMainCam		X

Tabla 2: Scripts

2 Estado del arte

Dentro de la industria de los videojuegos podemos encontrar muchos géneros de videojuegos diferentes, tales como videojuegos del género acción, disparos, estrategia, simulación, deporte, aventura, rol, etc. A medida que la industria de los videojuegos fue creciendo y estableciéndose en el mercado estos géneros fueron evolucionando, hasta el punto de necesitar diferenciarlos por subgéneros. Un ejemplo de estos subgéneros es el Metroidvania, género al que pertenece el videojuego desarrollado para este TFG.

El género de videojuegos Metroidvania [3] es un subgénero de los videojuegos de género acción-aventura combinado con el género de plataformas no lineal, este subgénero recibe su nombre de las sagas de videojuegos Metroid y Castlevania, que, a pesar de no ser los primeros videojuegos de este género, terminaron por acuñar el término Metroidvania debido a su gran éxito y mecánicas específicas y singulares, las cuales representarían un modelo a seguir para futuros videojuegos de este género.

A continuación, se presentan una serie de videojuegos pertenecientes a este subgénero y que resultaron de inspiración para el desarrollo y la realización del videojuego desarrollado para este TFG.

2.1 Videojuegos de género Metroidvania

Blasphemous

Blasphemous [4] es un videojuego de género Metroidvania ambientado en el folclore católico del sur de España. Blasphemous está desarrollado por el estudio español de programación The Game Kitchen y publicado por Team17. El videojuego fue lanzado para las plataformas PlayStation 4, Xbox One, Nintendo Switch y PC.

Blasphemous es un videojuego de acción y plataformas con desplazamiento horizontal que presenta una estética única y llamativa influenciada por la fantasía medieval representada mediante el pixel art clásico.

Blasphemous se trata de un videojuego desafiante con controles sencillos donde el personaje principal, el Penitente, debe enfrentarse a los diferentes enemigos y jefes haciendo uso de las habilidades especiales y objetos para avanzar en la historia. En Blasphemous la progresión del personaje principal está estrechamente relacionada con la exploración, puesto que a lo largo del videojuego el jugador puede mejorar habilidades y adquirir artefactos navegando por los diferentes escenarios del mapa que están llenos de caminos ocultos, atajos, trampas y enemigos. Al ser un videojuego de plataformas no lineal, el jugador tendrá la oportunidad de decidir qué camino tomar y explorar primero, siendo en ocasiones necesario haber obtenido alguna habilidad o algún artefacto especial para acceder a alguna zona del mapa en específico.

Imagen 3: Blasphemous

Hollow Knight

Hollow Knight [5] es un videojuego perteneciente al género Metroidvania desarrollado y publicado por Team Cherry. El videojuego ha sido lanzado para las plataformas PC, Nintendo Switch, PlayStation4 y Xbox One.

Hollow Knight es una aventura desafiante e ingeniosa donde el personaje principal, el Caballero, deberá explorar un gran mundo compuesto por diferentes zonas interconectadas haciendo uso de sus movimientos y técnicas de combate y derrotando a los distintos jefes esparcidos por las áreas con el fin de descubrir los secretos y tesoros del reino de Hollownest. A lo largo del juego el jugador consigue nuevos poderes y objetos que le permitirán avanzar en su aventura desbloqueando habilidades necesarias para acceder a nuevas zonas del reino de Hollownest.

Un dato interesante que cabe destacar es que Hollow Knight está desarrollado aprovechando las herramientas de desarrollo integradas en el motor Unity y las extensiones disponibles en la *Asset Store* para poder cumplir los objetivos técnicos del videojuego, permitiendo a los desarrolladores centrarse en otros aspectos como el arte que presenta el videojuego.

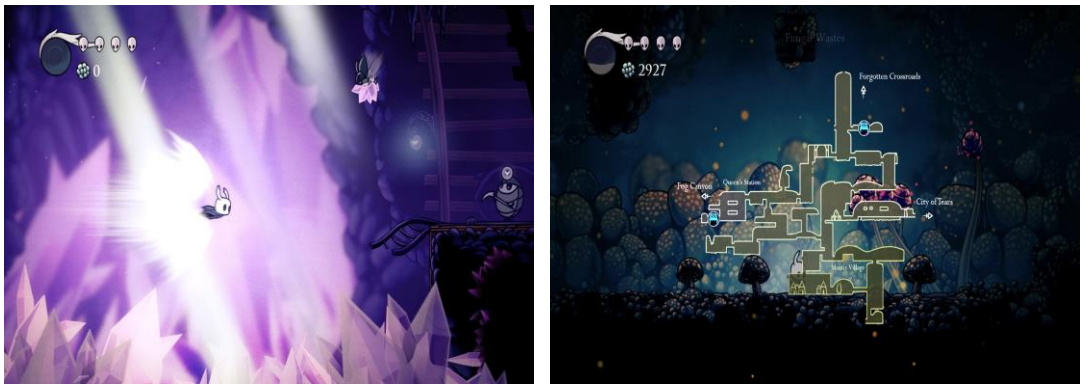

Imagen 4: Hollow Knight

Castlevania: Symphony of the Night

Castlevania: Symphony of the Night [6] es uno de los videojuegos que sirvió como punto de inflexión para que el subgénero Metroidvania se consolidara como tal, su influencia como videojuego referente en este género supuso un hito que marcaría un antes y un después para el mundo de los videojuegos.

Castlevania: Symphony of the Night es un videojuego de acción-aventura con plataformas y gráficos 2D desarrollado, publicado y distribuido por Konami. Symphony of the Night fue una entrega que revolucionó por completo la serie de Castlevania, introduciendo un nuevo estilo e incluyendo novedades en las mecánicas del juego añadiendo elementos RPG (juego de interpretación de papeles, *role-playing game*) más propios de un título de género rol, pero manteniendo la esencia del desplazamiento lateral 2D propio de los videojuegos de género acción-aventura mezclado con el género plataformas.

En Castlevania: Symphony of the Night el jugador controla al vampiro Alucard, hijo de Drácula, con el que podrá atacar con múltiples armas a los diferentes enemigos y tendrá la libertad de explorar el castillo de Drácula y las áreas adyacentes al mismo pudiendo volver a pasar por áreas ya visitadas después de ganar experiencia y haber

aprendido nuevas habilidades que proporcionan acceso a nuevas áreas. Los elementos RPG introducidos en esta entrega promueven la exploración con el fin de aumentar los atributos de Alucard y conseguir armas y artículos especiales.

Imagen 5: Castlevania: Symphony of the Night

2.2 Propuesta

El videojuego desarrollado para este TFG propone desviarse en cuanto a la jugabilidad habitual del genero Metroidvania, planteando una variación enfocada específicamente en el apartado perteneciente al equipamiento. Esta variación proporciona la posibilidad de que la adquisición de los objetos y el equipamiento sea mayoritariamente aleatoria, haciendo uso de una considerable variedad de equipamiento y objetos con estadísticas aleatorias incluidas en un rango que depende del propio objeto de equipamiento, garantizando de esta manera que, a pesar de que cuando adquieras un objeto de equipamiento y este sea repetido, en la misma partida o en una totalmente distinta, las estadísticas que otorgará este objeto de equipamiento probablemente serán diferentes. De esta forma el videojuego desarrollado para este TFG propone distinguirse de los demás Metroidvania (que comúnmente recurren a utilizar estadísticas fijas en sus objetos) agregando un factor de decisión y dejando en las manos del jugador la libertad de escoger que objetos de equipamiento desea usar en cada momento, ya sea por la situación en la que se encuentre, la habilidad especial de dicho objeto de equipamiento o por las estadísticas que le aporta al jugador, ofreciendo una dinámica distinta en las diferentes partidas.

Asimismo, otro aspecto adicional que se aparta de los demás Metroidvania, y que podemos encontrar en el videojuego desarrollado para este TFG, es la historia que se presenta para complementar a esta jugabilidad, la cual se respalda en elementos humorísticos y cómicos con la finalidad de generar una mejor y diferente experiencia de juego.

3 Herramientas utilizadas

En esta sección se introducen las diferentes herramientas utilizadas para llevar a cabo el desarrollo del proyecto, exponiendo brevemente la razón por la que se han decidido utilizar y el uso que se le ha dado a cada una de ellas.

Unity

Unity [7] es un motor de videojuegos multiplataforma de desarrollo creado por Unity Technologies y disponible para Microsoft Windows, Linux y Mac OS. Unity es la herramienta principal que se ha decidido utilizar para el desarrollo de este proyecto debido a que es una herramienta fácil de usar y que proporciona una gran versatilidad a la hora de programar gracias a las herramientas de desarrollo integradas en el motor y las compatibilidades con los editores de código y los entornos de desarrollo integrados (IDEs), por otra parte, los integrantes del grupo ya disponían de conocimientos básicos sobre la aplicación y utilización de Unity, debido a que ya habían cursado la asignatura de Desarrollo de Videojuegos 3D. En la próxima sección profundizaremos sobre los detalles del motor explicando las herramientas de desarrollo más importantes utilizadas durante el desarrollo del proyecto.

Trello

Trello [8] es un software de administración de proyectos con interfaz web y con cliente para Android y iOS para organizar proyectos desarrollado por Trello Inc. Trello hace uso del sistema Kanban, sistema de información basado en tarjetas que controla la fabricación de los productos en la cantidad y tiempo necesarios para cada uno de los procesos por los que pasa dicho producto. Para la realización del proyecto se utiliza la interfaz web de Trello, creando un tablón virtual donde añadir tarjetas con las diferentes tareas a realizar durante el desarrollo del proyecto. Para la organización durante el desarrollo del proyecto se ha decidido utilizar Trello por su accesibilidad y manejabilidad, que permite organizar de forma sencilla las distintas tareas que conforman el proyecto.

Git

Git [9] es un software de control de versiones de código abierto diseñado por Linus Torvalds cuyo propósito es llevar un registro de los cambios realizados sobre archivos en un repositorio de código compartido, coordinando de forma eficiente y confiable el trabajo que varias personas realizan sobre estos archivos compartidos. Git es el sistema de control de versiones que se ha utilizado para llevar a cabo la gestión y dirección de los archivos que constituyen el código fuente del proyecto.

GitHub: GitHub es una plataforma de desarrollo colaborativo de software utilizado para alojar proyectos utilizando el sistema de control de versiones Git. Principalmente GitHub se utiliza para la creación de código fuente de programas. En la plataforma de GitHub está alojado completamente el código fuente de este proyecto.

GitHub Desktop: GitHub Desktop es una herramienta de código abierto que extiende y simplifica el flujo de trabajo aplicando las mejoras prácticas de Git y GitHub y utilizando una interfaz visual sencilla y cómoda que permite mejorar la

productividad y fomenta el trabajo colaborativo. GitHub Desktop es la herramienta que se ha decidido utilizar para la administración de los archivos compartidos del código fuente del proyecto.

Microsoft Visual Studio

Microsoft Visual Studio [10] es un entorno de desarrollo integrado (IDE) desarrollado por Microsoft y disponible para los sistemas operativos Windows y macOS. Visual Studio dispone de compatibilidad con múltiples lenguajes de programación y entornos de desarrollo web. La programación de este proyecto se ha realizado en lenguaje C# utilizando Microsoft Visual Studio debido a su compatibilidad e integración con el motor Unity, el cual permite acoplar Microsoft Visual Studio como editor de código externo posibilitando de esta forma la programación y edición del código fuente del proyecto de forma directa y cómoda.

Itch.io

Itch.io [11] es una plataforma de distribución digital, destinada a publicar videojuegos independientes. Itch.io permite publicar y compartir cualquier tipo de contenido digital y organizar *game jams* (encuentros de desarrolladores o *hackatón* que tiene como propósito la creación de uno o más videojuegos en un corto período de tiempo). Para la elaboración de nuestro proyecto hemos decidido hacer uso de Itch.io debido a la gran cantidad de contenido disponible que posee. Además, hemos decidido subir el videojuego de forma privada a la plataforma de Itch.io una vez construido por el motor Unity.

4 Unity

Esta sección muestra una serie de aspectos sobre el motor Unity utilizado durante el desarrollo del proyecto con el objetivo de profundizar y explicar algunos conceptos básicos fundamentales para poder entender mejor el funcionamiento principal de Unity.

Una de las características o funcionalidades principales del motor Unity que merece la pena destacar es la posibilidad que ofrece de poder utilizarse junto con herramientas externas, apoyándose en ellas para realizar cambios sobre los objetos y scripts creados y actualizarlos automáticamente en todas las instancias presentes en todo el proyecto sin necesidad de importarlas nuevamente de forma manual. Además, Unity también hace uso de herramientas integradas en el motor para dar soporte y gestionar las físicas, colisiones, sombreados, texturas y efectos. Por último, comentar que Unity también contiene una ventana que nos proporciona un acceso a la *Unity Asset Store*, desde donde podemos descargar diferentes *assets* para utilizar en nuestro proyecto y que podremos administrar desde la ventana Administrador de Paquetes que proporciona el motor Unity.

Ahora se van a detallar los distintos componentes del motor Unity para poder tener una idea básica de algunos conceptos cuando se mencionen más adelante.

4.1 Escenas

Las escenas en el motor Unity son los elementos primordiales donde los desarrolladores trabajan creando, inspeccionando, modificando y/o eliminando diferentes *GameObjects* en dicha escena. Estos *GameObjects* contenidos en cada una de las escenas conforman todo o parte de la aplicación o juego desarrollado. Este juego o aplicación puede emplear una o varias escenas dependiendo de la magnitud y complejidad del proyecto que se desea producir.

En nuestro proyecto hacemos uso de las escenas interconectándolas para representar el menú principal y cada uno de los distintos niveles que forman parte de nuestro videojuego.

4.2 GameObjects

Los *GameObject* [12] son el concepto más importante y fundamental en el editor de Unity. Cada uno de los objetos y elementos que encontramos en las escenas y que conforman la aplicación o juego son *GameObjects*, que pueden representar tanto personajes como sonidos, efectos especiales, luces, cámaras, etc.

Los *GameObjects* funcionan como unos contenedores compuestos por diferentes componentes, que le proporcionan propiedades, determinan su comportamiento y establecen su funcionalidad, para transformarlos en el objeto que el desarrollador necesita. En función del tipo de objeto que se espera desarrollar, un *GameObject* puede incluir distintas combinaciones de componentes.

Los *GameObjects* se pueden almacenar junto con todos sus componentes en las carpetas del proyecto convirtiéndolos en *Prefabs*, que actúan como una plantilla a partir de la cual se pueden crear nuevas instancias del *Prefab* en la misma escena o

en una diferente, para poder reutilizar el *GameObject*. Cualquier modificación que se realice en un *Prefab* se verá incorporado automáticamente en todas las instancias de ese *Prefab*, lo que proporciona la posibilidad de llevar a cabo modificaciones en todo el proyecto sin la necesidad de tener que realizar la misma modificación reiteradamente en cada una de las instancias creadas en el proyecto.

4.2.1 Componentes

Los componentes son las piezas funcionales que contiene cada *GameObject*. Los componentes incorporan propiedades que se pueden modificar con el objetivo de definir el comportamiento y dotar de funcionalidad a un *GameObject*. Se pueden crear, inspeccionar, modificar y/o eliminar los componentes adjuntos a un *GameObject* en la ventana Inspector tras haber seleccionado el *GameObject* en la ventana Jerarquía o Escena.

Seguidamente se exponen los tipos de componentes más importantes que posee integrados el motor Unity y que se pueden agregar a los distintos *GameObjects*.

- **Transform:** Un *GameObject* siempre tiene un componente *Transform* asociado cuando se crea, dicho componente es el más importante y no se puede eliminar, debido a que este componente se utiliza para representar la posición, la rotación y la escala del objeto en la escena. En caso de que los *GameObjects* cuenten con una jerarquía padre-hijo los valores del componente *Transform* del hijo se evalúan con respecto al componente *Transform* del padre. En caso de que el *GameObject* no tenga ningún padre las propiedades del componente *Transform* de ese *GameObject* se evalúan en comparación con el *World Space* de la escena.

- **Scripts:** Un script incluido en un *GameObject* como un componente proporciona la posibilidad de imponer la funcionalidad del objeto modificando el *GameObject* y sus Componentes pudiendo activarlos, desactivarlos y manipularlos, permitiendo cambiar los distintos valores y parámetros de los que dispone cada uno de ellos recurriendo a la utilización de código. Posteriormente se detallará en más profundidad el ciclo de vida del script.

- **Collider:** Los componentes *Collider* determinan la forma y el volumen de un *GameObject* con el objetivo de calcular sus colisiones físicas. Existen *colliders* de distintos tipos y formas para usarse y ajustarse tanto a *GameObjects 2D* como a *GameObjects 3D*. Cabe destacar especialmente un parámetro de los *Colliders*, *Is Trigger*, en caso de que este parámetro esté activado el *GameObject* puede ser atravesado por otros objetos, y cuando esto sucede se genera una señal.

- **Animator:** El componente *Animator* se utiliza para conceder animaciones a un *GameObject* en las escenas. El componente *Animator* necesita una referencia a un *Animator Controller* para gestionar y organizar la máquina de estados desde la ventana Animador y determinar que clip de animación utilizar, controlando cuándo y cómo combinar y hacer las transiciones entre ellas.

- **Audio Source:** El componente *Audio Source* permite procesar un clip de audio enlazándolo a un *GameObject* en la escena. El componente *Audio Source* dispone de la capacidad para reproducir cualquier clase de clip de audio y puede configurarse para reproducirlos en 2D, 3D o como una combinación de ambos.

- **Rigidbody:** El componente *Rigidbody* sitúa un *GameObject* bajo el control del motor de física, permitiendo un comportamiento basado en las leyes de la física haciendo posible recibir fuerzas, como la gravedad, y torsiones para proporcionar que los *GameObjects* tengan colisiones y un movimiento más realista. Todo *GameObject* necesita un *Rigidbody* para ser afectado por estas fuerzas y poder desempeñar acciones bajo ellas o interactuar con otros objetos a través del motor de física.

4.3 Flujo del ciclo de vida del script

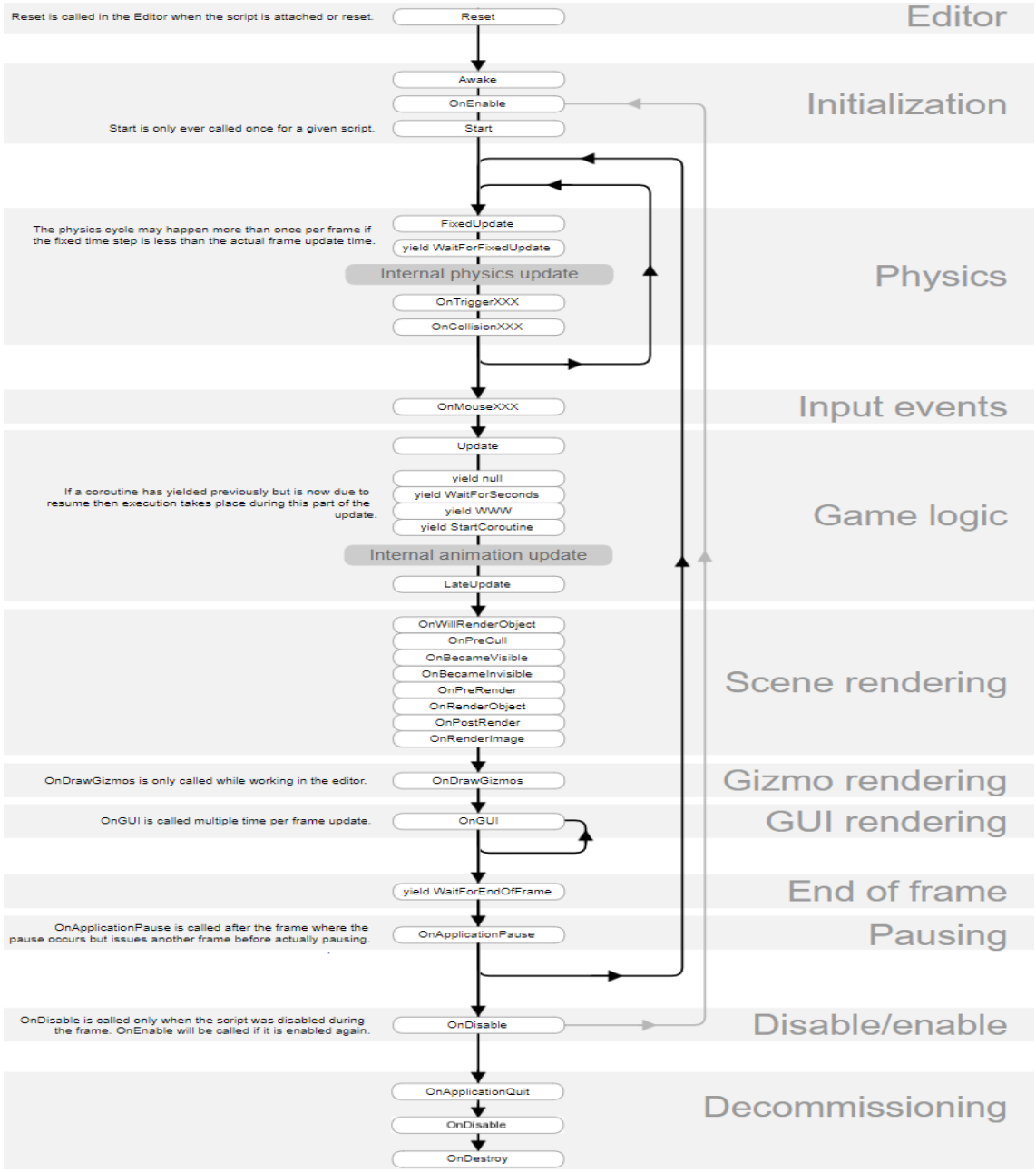

Imagen 6: Flujo del ciclo de vida del script

En este punto se resume cómo el motor Unity ejecuta periódicamente una serie de métodos implícitos y específicos del motor en un orden predeterminado. Los scripts en el motor Unity heredan todos de la clase *MonoBehaviour*. Seguidamente se exponen los métodos más utilizados de la clase *MonoBehaviour* iniciando por los primeros métodos que son ejecutados y finalizando por los últimos métodos que son ejecutados por el flujo del ciclo de vida del script que podemos observar en la Imagen 6 [13]:

- **Awake:** Es el primer método ejecutado. Este método se llama siempre antes de cualquier método *Start* en todos los objetos y también justo después de que se instancie el *Prefab* o *GameObject*. Este método es el que equivaldría como sustituto de los constructores en el motor Unity, suele utilizarse para inicializar referencias entre objetos. En caso de que un *GameObject* esté desactivado durante el inicio, el método *Awake* no se ejecuta hasta que se activa dicho *GameObject*. El método *Awake* sólo se ejecuta una única vez por cada objeto al arrancar una escena a la que pertenece.

- **OnEnable:** El método *OnEnable* solo se ejecuta si el objeto está activo. Este método se invoca justo después de que se active el objeto, esto ocurre cuando se crea una instancia de *MonoBehaviour*, cuando se carga una escena o cuando un *GameObject* con el componente de script es instanciado.

- **Start:** Es el segundo paso en la inicialización de un *GameObject*. Este método se ejecuta siempre después del método *Awake* y antes de la primera invocación al método *Update*. El método *Start* solo se invoca si el objeto está activado. El método *Start* sólo se ejecuta una única vez por cada objeto al arrancar una escena a la que pertenece.

- **FixedUpdate:** El método *FixedUpdate* se suele invocar con más frecuencia que el método *Update*. El método *FixedUpdate* se puede llamar varias veces por fotograma si la velocidad entre fotogramas es baja y es posible que no se llame entre fotogramas si la velocidad entre fotogramas es alta. Este método se invoca antes de todos los cálculos y actualizaciones de física, por esta razón *FixedUpdate* es el método ideal donde realizar modificaciones en el *Rigidbody* del *GameObject*.

- **OnTriggerEnter, OnTriggerStay y OnTriggerExit:** Estos métodos están relacionados con el componente *Collider* de los *GameObjects*. Estos métodos son invocados cuando el *GameObject* tiene activado el parámetro *Is Trigger* del componente *Collider* y es atravesado por otro *collider*. Estos métodos se invocan en la entrada, la estancia y la salida respectivamente del otro *collider*.

- **OnCollisionEnter, OnCollisionStay y OnCollisionExit:** Estos métodos están relacionados con el componente *Collider* de los *GameObjects*. Estos métodos son invocados cuando los componentes *Collider* de dos o más *GameObjects* colisionan y no se atraviesan. Estos métodos se invocan cuando se detectan las colisiones en la entrada, la estancia y la salida respectivamente de los *colliders*.

- **Update:** El método *Update* solo se ejecuta si el objeto está activado. Este método se invoca una vez por fotograma. Es el método central donde se realizan todas las actualizaciones y cálculos que no dependen de otros objetos.

- **LateUpdate:** El método *LateUpdate* se invoca una vez por fotograma, después de que finalice la invocación del método *Update* de los *GameObjects*, por lo

tanto, cualquier cálculo que se ejecute en el método *Update* ya se habrá completado cuando inicie el método *LateUpdate*. Este método suele emplearse cuando se necesitan los resultados previamente calculados en el método *Update* para realizar nuevos cálculos y actualizar los datos en el método *LateUpdate*.

- **OnDisable:** El método *OnDisable* se invoca cuando el comportamiento del *GameObject* se desactiva o antes de invocar al método *OnDestroy*. Este método se suele utilizar para realizar toda clase de limpieza y/o retirar el objeto del juego y la escena actual. Si el objeto vuelve a ser invocado en otra escena más adelante hay que volver a ejecutar de nuevo el método *OnEnable*.

- **OnDestroy:** El método *OnDestroy* sólo se puede invocar con objetos que hayan estado previamente activos. Este método se invoca después de todas las actualizaciones realizadas por los métodos *Update* y durante el último fotograma de la existencia del objeto. El objeto puede destruirse en respuesta a la llamada *Object.Destroy* o al cierre de una escena.

Una vez aclarados los métodos fundamentales del flujo del ciclo de vida del script en el motor Unity conseguimos comprender mejor como se ha trabajado en el proyecto tomando en consideración su comportamiento y funcionamiento. Adicionalmente a los métodos nombrados previamente, podemos encontrar la existencia de otros métodos que participan interviniendo en otras partes del flujo del ciclo de vida del script tales como los métodos *Yield WaitForSeconds* o *Yield StartCoroutine* que tienen que ver con corrutinas.

4.4 Interfaz de Unity

En los siguientes puntos se proporciona una introducción detallada de las distintas ventanas de edición del motor Unity, iniciando por las ventanas más comunes y seguidamente se introducirán algunas ventanas de edición que incorporan una funcionalidad más avanzada.

4.4.1 Ventanas comunes

Cuando iniciamos por primera vez un proyecto en el motor Unity se nos plantean diferentes ventanas abiertas por defecto en la interfaz general del motor Unity. El objetivo de este punto es introducir estas ventanas principales e indicar para que se utiliza cada una de ellas.

Jerarquía

La ventana Jerarquía es una representación de texto jerárquica en forma de lista que muestra cada *GameObject* en la escena. Cada *GameObject* de la escena tiene su propia representación en la jerarquía, por esta razón las dos ventanas, Jerarquía y Escena están intrínsecamente relacionadas. La jerarquía muestra la organización de los *GameObjects* y sus relaciones padre-hijo.

Esta ventana se puede utilizar para estructurar, ordenar y agrupar los *GameObjects* que se utilizan en una escena. Cuando se crean o eliminan *GameObjects* en la ventana Escena también se crean o eliminan de la ventana Jerarquía. La ventana Jerarquía puede incorporar más de una Escena, cada una de ellas con sus propios *GameObjects*.

Juego

La ventana Juego se utiliza para simular cómo se verá la aplicación o juego tras su renderizado final a través de los componentes *Camera* representando una imagen desde un punto de vista en particular de los *GameObjects* que se encuentran presentes en la escena y que contienen este componente *Camera*. Los botones de la barra de herramientas se utilizan para controlar el modo de reproducción del editor del motor Unity y ver como se reproduce el juego o aplicación ya renderizada. Durante el modo de reproducción toda modificación que se lleve a cabo es provisional y se restablece una vez se sale del modo de reproducción, por esta razón es muy importante tener presente si se está en el modo de reproducción o no. La interfaz de usuario del editor del motor Unity se oscurece para recordarlo, no obstante, el usuario puede modificar esto para que en vez de que se oscurezca cambie a otro color que se adapte mejor a sus gustos.

Escena

La ventana Escena permite editar visual e interactivamente las escenas y navegar a través de ellas. Esta ventana puede utilizarse para seleccionar, manipular, distribuir y modificar cualquier tipo de *GameObject* y/o *Prefab*. La ventana Escena permite la posibilidad de mostrar una perspectiva 2D o 3D, dependiendo de la clase de proyecto que se esté llevando a cabo.

Inspector

La ventana Inspector se utiliza para ver, configurar y editar cualquiera de las propiedades del *GameObject* seleccionado en ese momento. Puesto que existen distintos tipos de *GameObjects* que contienen distintas combinaciones de componentes y conjuntos de propiedades, el diseño y el contenido de la ventana Inspector varía cada vez que se selecciona un *GameObject* distinto.

Proyecto

La ventana Proyecto muestra todos los *Assets* y archivos que están disponibles para usar y que están relacionados con el proyecto. Esta ventana es la forma principal con la que puedes navegar y buscar *Assets* y otros archivos del proyecto. Cada vez que se importa algún *Asset* o archivo al proyecto este se subirá y aparecerá en esta ventana.

Consola

La ventana Consola proporciona notificaciones sobre los errores, advertencias y otros mensajes que genera el editor del motor Unity. Estos errores y advertencias mostrados pueden servir de ayuda para detectar, localizar e identificar problemas en el proyecto, tales como problemas en los procesos del motor Unity o errores de compilación en los scripts. Esta ventana también puede ayudar a depurar el proyecto utilizando la clase *Debug* para poder imprimir mensajes propios en la ventana Consola y poder ver el valor de alguna variable en cierto punto del script para ver cómo cambian o comprobar si un punto del script es alcanzado al realizar alguna acción. Las notificaciones que aparecen en la ventana Consola del motor Unity se pueden filtrar para facilitar la búsqueda de algún mensaje en específico.

Seguidamente mostramos una imagen donde se pueden apreciar las diferentes ventanas que acabamos de mencionar:

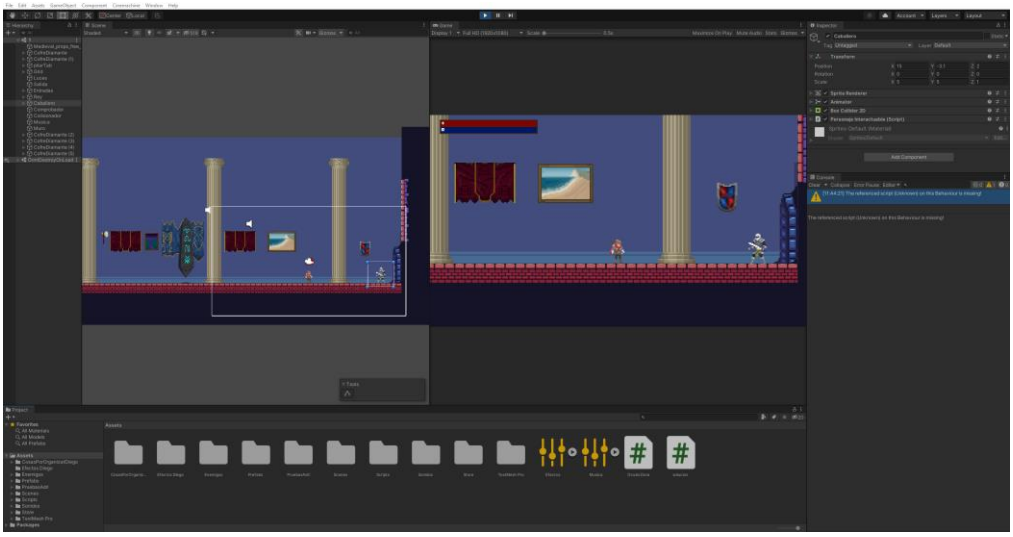

Imagen 7: Ventanas comunes

4.4.2 Ventanas avanzadas

Adicionalmente a las ventanas principales que se han expuesto en el punto anterior, el motor de Unity provee una serie de funcionalidades más avanzadas que no aparecen a primera vista. En este punto se introducen algunas de las ventanas que permiten acceder a estas funcionalidades avanzadas.

Animación

La ventana Animación se utiliza para previsualizar y editar clips de animación asociados a un *GameObject* animado en el motor Unity. Esta ventana no aparece abierta por defecto en la interfaz general del motor Unity cuando se inicia por primera vez un proyecto, por lo tanto, para abrir la ventana Animación en el motor Unity es necesario seleccionar la opción Ventana > Animación.

La ventana Animación está relacionada con las ventanas Jerarquía, Escena, Inspector y Proyecto. De la misma forma que la ventana Inspector, la ventana Animación ilustra la línea de tiempo y los fotogramas clave de la animación del *GameObject* o clip de animación seleccionado en ese momento. Se puede seleccionar un *GameObject* utilizando las ventanas de Jerarquía o Escena o seleccionando un *Asset* de clip de animación recurriendo a la ventana Proyecto.

Animador

La ventana Animador concede la posibilidad de crear, inspeccionar y modificar *Assets* de *Animator Controller*. Esta ventana tiene dos secciones principales:

- **Área de diseño:** Esta área se utiliza para crear, organizar y conectar mediante transiciones los diferentes estados en el *Animator Controller*, creando de esta forma una máquina de estados.

- **Panel de capas y parámetros:** Está compuesto por dos subventanas. La subventana de Parámetros se utiliza para crear, inspeccionar y editar los parámetros del *Animator Controller*, es decir, las variables que se definen dentro del *Animator Controller* y a los que se puede acceder y asignar valores desde scripts, de esta forma es como un script puede controlar o afectar al flujo de la máquina de estados. La subventana de Capas se utiliza para crear, inspeccionar y editar capas dentro de su *Animator Controller* para gestionar máquinas de estado complejas.

Mezclador de audio

La ventana Mezclador de Audio proporciona la capacidad de configurar y gestionar los diversos *Audio Mixers*, que se enrutarán en la salida de los *Audio Source* permitiendo aplicarles efectos, atenuaciones de volumen y correcciones de tono. Más adelante se indicará la manera en la que se ha desarrollado y configurado el sistema de sonido del videojuego desarrollado para este TFG, empleando para ello esta ventana.

Asset Store

El motor Unity cuenta con una ventana que nos da la opción de acceder a la página web de la *Unity Asset Store*. En anteriores versiones del motor Unity la *Unity Asset Store* se encontraba alojada en esta ventana, sin embargo, para poder mejorar el rendimiento en el Editor del motor Unity, ahora esta ventana solo se utiliza como redirección a la página web.

En la página web de la *Unity Asset Store* podremos navegar para descargar y/o comprar distintos paquetes de *Assets* que podremos importar en nuestro proyecto desde la ventana Administrador de Paquetes.

Administrador de Paquetes

La ventana Administrador de Paquetes nos permite ver los distintos paquetes de *Assets* descargados y/o comprados para poder administrarlos seleccionando cuántos y cuáles de ellos deseamos importar en el proyecto.

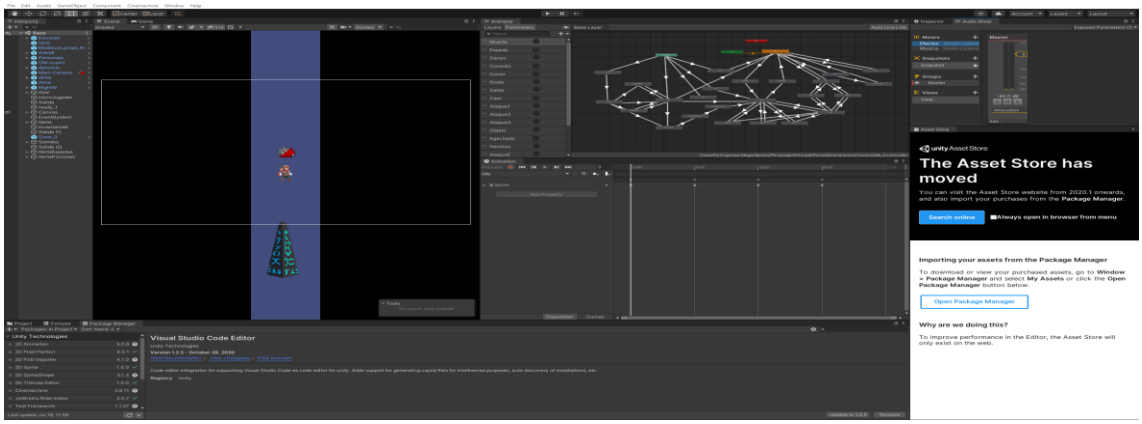

Imagen 8: Ventanas avanzadas

4.5 Construir Ejecutable

Finalmente, vale la pena resaltar que el motor de Unity proporciona una opción para construir un ejecutable seleccionando la opción Archivo > Configuraciones de compilación. Esta ventana nos proporciona la posibilidad de seleccionar las distintas escenas que compondrán nuestro ejecutable para de esta manera poder cambiar entre escenas por medio del código. Esta ventana nos ofrece diversos métodos para construir el ejecutable del proyecto. Una vez se seleccione el método deseado para la construcción del ejecutable del proyecto la primera escena que se mostrará será la que se encuentre en el lado superior de las escenas seleccionadas.

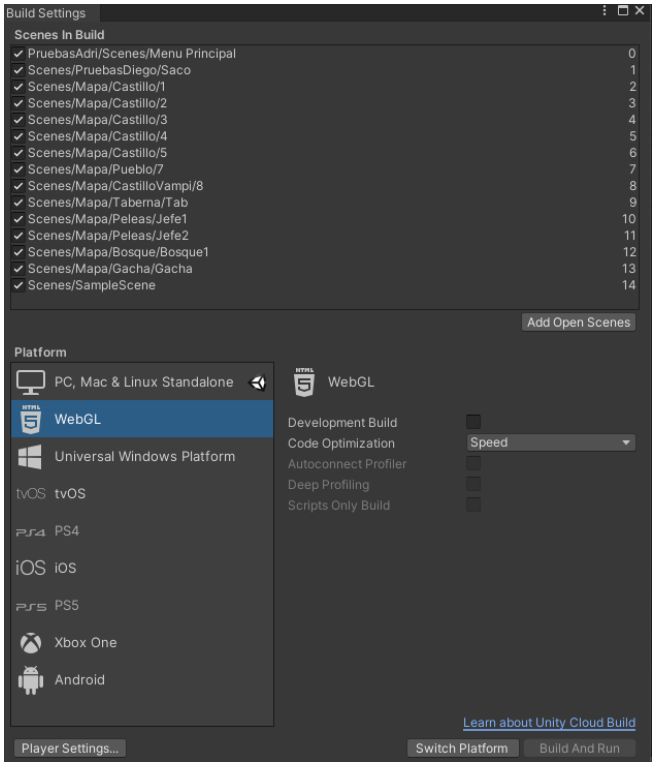

Imagen 9: Construir ejecutable

El ejecutable de este proyecto se ha construido seleccionando el método WebGL, para que, de esta manera, sea soportado por plataformas Web y poder subir el proyecto a la plataforma Itch.io.

5 Análisis de requisitos del proyecto

En este apartado se van a exponer los distintos requisitos del proyecto, tanto funcionales como no funcionales. En un apartado posterior se especificará cómo se ha llevado a cabo el desarrollo de los mismos.

5.1 Requisitos Funcionales del proyecto

Con la intención de poder entender de manera más efectiva los distintos requisitos funcionales del proyecto se han fraccionado en varias secciones:

- Menú Principal
- Sistema de diálogos y monólogos
- Sistema de misiones
- Manual del Aventurero
- Sistema de combate
- Trampas
- Cofres
- Sistema de sonidos
- Sistema del personaje
- Sistema de inventario
- Sistema de estadísticas
- Sistema de enemigos

Menú Principal

Identificador	CU-01
Nombre	Fade In
Descripción	Al comenzar el videojuego la pantalla primero aparece en negro y va progresivamente apareciendo la imagen del Menú Principal.
Actor	Jugador
Escenario	1. El jugador comienza el videojuego. 2. El sistema muestra la pantalla en negro y progresivamente va mostrando la imagen del Menú Principal.

Identificador	CU-02
Nombre	Fade Out
Descripción	Al iniciar o salir del videojuego la imagen del Menú Principal se va desvaneciendo hasta fundirse en negro.
Actor	Jugador
Escenario	1. El jugador pulsa el botón de Iniciar Juego o Salir en el Menú Principal. 2. El sistema va oscureciendo la imagen del Menú Principal hasta fundirse en negro.

Identificador	CU-03
Nombre	Salir del videojuego
Descripción	Permite cerrar el videojuego
Actor	Jugador
Escenario	1. El jugador pulsa el botón de Salir en el Menú Principal. 2. El sistema inicia la animación *Fade-Out*. 3. El sistema cierra el videojuego.

Identificador	CU-04
Nombre	Mostrar interfaz de opciones
Descripción	Permite al jugador ver la interfaz de opciones
Actor	Jugador
Escenario	1. El jugador pulsa el botón Opciones en el Menú Principal. 2. El sistema muestra al jugador la interfaz de opciones.

Identificador	CU-05
Nombre	Cerrar interfaz de opciones
Descripción	Permite al jugador cerrar la interfaz de opciones
Actor	Jugador
Escenario	1. El jugador pulsa el botón de cerrar la interfaz. 2. El sistema oculta la interfaz de opciones.

Identificador	CU-06
Nombre	Iniciar juego
Descripción	Permite al jugador iniciar el juego
Actor	Jugador
Escenario	1. El jugador pulsa el botón Iniciar Juego. 2. El sistema inicia la animación *Fade-Out*. 3. El sistema carga la primera escena del videojuego.

Sistema de diálogos y monólogos

Identificador	CU-07
Nombre	Iniciar diálogo o monologo
Descripción	Permite al jugador iniciar un diálogo o monologo en caso de que esta acción sea posible
Actor	Jugador
Escenario	1. El personaje principal entra en contacto con un personaje interactuable. 2. El sistema comprueba si este personaje tiene todavía un diálogo o monologo pendiente. 2.1. En caso afirmativo inicia una animación para indicárselo al jugador. 2.2. En caso negativo el jugador no puede interactuar con el personaje y el caso de uso termina. 3. El jugador pulsa la tecla de interactuar. 4. El sistema inicia la animación inicial del sistema de diálogos mostrando la primera frase u oración del diálogo o monólogo.

Identificador	CU-08
Nombre	Avanzar diálogo o monologo
Descripción	Permite al jugador avanzar a la siguiente frase u oración del diálogo o monologo
Actor	Jugador
Escenario	1. El jugador pulsa la tecla de avanzar diálogo o monologo. 2. El sistema comprueba que no es la última frase u oración del diálogo o monologo. 3. El sistema comprueba si es un dialogo. 3.1. En caso afirmativo el sistema actualiza la interfaz del sistema de diálogos cambiando la imagen de la persona que habla. 4. El sistema muestra la siguiente frase u oración del diálogo o monologo.

Identificador	CU-09
Nombre	Cerrar diálogo o monologo
Descripción	Cuando no quedan más frases u oraciones en el diálogo o monologo permite al jugador cerrar el diálogo o monologo
Actor	Jugador
Escenario	1. El jugador pulsa la tecla de avanzar diálogo o monologo. 2. El sistema comprueba que es la última frase u oración del diálogo o monologo. 3. El sistema inicia la animación final del sistema de diálogos. 4. El sistema comprueba si el personaje debe soltar algún objeto al terminar el diálogo o monologo. 4.1. En caso afirmativo el sistema instancia el objeto.

Sistema de misiones

Identificador	CU-10
Nombre	Terminar Misión
Descripción	Permite al sistema registrar como terminada una misión
Actor	Jugador
Escenario	1. El jugador termina una misión. 2. El sistema registra la misión como terminada

Identificador	CU-11
Nombre	Comprobar Misiones
Descripción	Permite al sistema comprobar las misiones que han sido terminadas
Actor	-
Escenario	1. El sistema comprueba que misiones han sido terminadas 1.1. En caso de que exista una misión terminada el sistema se encarga de llevar a cabo las acciones pertinentes (Desactivar *GameObjects*, marcar diálogos/monólogos como terminados, etc.)

Manual del Aventurero

Identificador	CU-12
Nombre	Abrir Manual Aventurero
Descripción	Permite al jugador abrir el manual del aventurero
Actor	Jugador
Escenario	1. El jugador utiliza el manual del aventurero. 2. El sistema inicia la animación de abrir el manual del aventurero. 3. El sistema muestra el contenido de la primer página del manual del aventurero.

Identificador	CU-13
Nombre	Siguiente página Manual Aventurero
Descripción	Permite al jugador pasar a la siguiente página del manual del aventurero
Actor	Jugador
Escenario	1. El jugador pulsa el botón de siguiente página del manual del aventurero. 2. El sistema comprueba si existe una siguiente página en el manual del aventurero. 2.1. En caso negativo el caso de uso termina. 3. El sistema inicia la animación de pasar a la siguiente página del manual del aventurero. 4. El sistema muestra el contenido de la siguiente página del manual del aventurero.

Identificador	CU-14
Nombre	Anterior página Manual Aventurero
Descripción	Permite al jugador volver a la anterior página del manual del aventurero
Actor	Jugador
Escenario	1. El jugador pulsa el botón de anterior página del manual del aventurero. 2. El sistema comprueba si existe una página anterior en el manual del aventurero. 2.1. En caso negativo el caso de uso termina 3. El sistema inicia la animación de volver a la anterior página del manual del aventurero. 4. El sistema muestra el contenido de la anterior página del manual del aventurero.

Identificador	CU-15
Nombre	Cerrar Manual Aventurero
Descripción	Permite al jugador cerrar el manual del aventurero
Actor	Jugador
Escenario	1. El jugador pulsa el botón de cerrar el manual del aventurero. 2. El sistema inicia la animación de cerrar el manual del aventurero.

Sistema de combate

Identificador	CU-16
Nombre	Ataque especial
Descripción	Permite al jugador realizar un ataque especial
Actor	Jugador
Escenario	1. El jugador presiona la tecla de ataque especial. 2. El sistema comprueba si es posible realizar el ataque especial. 3. En caso de ser posible el sistema realiza el ataque especial correspondiente.

Identificador	CU-17
Nombre	Usar poción
Descripción	Permite al jugador recuperar vida o mana utilizando una poción
Actor	Jugador
Escenario	1. El jugador usa una poción desde el inventario. 2. El sistema comprueba que tipo de poción se ha usado. 3. El sistema suma la cantidad de vida o mana correspondiente al tipo de poción.

Trampas

Identificador	CU-18
Nombre	Activar trampa
Descripción	La trampa se activará
Actor	-
Escenario	1. El sistema comprobará si el personaje principal se encuentra lo suficientemente cerca. 1.1. En caso negativo el caso de uso termina 2. El sistema activará la trampa

Identificador	CU-19
Nombre	Movimiento trampa
Descripción	La trampa se moverá
Actor	-
Escenario	1. El sistema moverá la trampa a una velocidad. 2. El sistema comprobará si la trampa se ha movido una distancia. 2.1. En caso afirmativo el sistema moverá la trampa en dirección contraria y se repite el paso 2.

Cofres

Identificador	CU-20
Nombre	Abrir Cofre
Descripción	Permite al jugador abrir un cofre y obtener un objeto
Actor	Jugador
Escenario	1. El jugador interactúa con el cofre. 2. El sistema comprueba el tipo de cofre. 3. El sistema calcula la calidad del objeto dependiendo del tipo del cofre 4. El sistema determina aleatoriamente el objeto dependiendo de la calidad calculada. 5. El sistema se encarga de calcular aleatoriamente las estadísticas del objeto, dentro de un rango de valores que dependerá del propio objeto. 6. El sistema instancia el objeto en el lugar del cofre.

Sistema de sonidos

Identificador	CU-21
Nombre	Modificar volumen de la música
Descripción	Permite al jugador modificar el volumen de la música del videojuego
Actor	Jugador
Escenario	1. El jugador modifica el valor del volumen de la música del videojuego. 2. El sistema establece el volumen de la música del videojuego al nuevo valor asignado.

Identificador	CU-22
Nombre	Modificar el volumen de los efectos
Descripción	Permite al jugador modificar el volumen de los efectos de sonido del videojuego
Actor	Jugador
Escenario	1. El jugador modifica el valor del volumen de los efectos de sonido del videojuego. 2. El sistema establece el volumen de los efectos de sonido del videojuego al nuevo valor asignado.

En lo que respecta a los sistemas de personaje, inventario, estadísticas y enemigos, estos sistemas han sido desarrollados por el compañero de equipo Diego Ruiz Muñoz, por esta razón, los requisitos funcionales relacionados con estos sistemas se explican en su correspondiente TFG.

5.2 Requisitos No Funcionales del proyecto

A continuación, se expone una tabla con los distintos requisitos no funcionales que se han planeado alcanzar durante el proyecto:

Identificador	Descripción
RNF-01	El videojuego debe de poder funcionar en plataforma web.
RNF-02	El videojuego debe de poder cargar cada uno de los niveles del mismo en menos de un minuto.
RNF-03	El videojuego debe de poder iniciarse independientemente del sistema operativo del equipo que se utilice
RNF-04	El videojuego debe de funcionar correctamente sin importar la resolución de la pantalla del equipo que se utilice.

6 Diseño del proyecto

En esta sección se describen los distintos aspectos que conforman el proyecto exponiendo la evolución del videojuego sin entrar en detalle en su desarrollo, puesto que esto se explicará en otra sección más adelante.

6.1 Concepto principal

La idea central del proyecto es desarrollar un videojuego de género Metroidvania en el cual se tiene que avanzar por las distintas escenas que forman parte del mapa del videojuego, enfrentándose a enemigos y superando trampas para subir de nivel y conseguir equipamiento, con el propósito de alcanzar las zonas donde se encuentran los jefes. El objetivo principal del juego es vencer a los jefes para conseguir las gemas que desbloquearán al jefe final del videojuego.

A medida que se avanza por las distintas zonas del mapa la cantidad de enemigos y trampas va aumentando, de esta forma la dificultad del videojuego estará relacionada con la zona en la que se encuentre el jugador, su nivel y su equipamiento. Este equipamiento se puede adquirir aleatoriamente abriendo los cofres dispersados por las distintas zonas del mapa. Por otra parte, las estadísticas de este equipamiento son diferentes oscilando en un rango, es decir, un objeto equipable cuenta con un máximo y un mínimo de estadísticas, al crearse el objeto este obtiene un valor dentro del rango, de esta forma, el mismo objeto puede tener estadísticas diferentes, con esto el videojuego busca conseguir añadir un factor de decisión dando al jugador la libertad para elegir que equipamiento desea llevar en cada momento ofreciendo cierto dinamismo en las distintas partidas con la intención de alcanzar el final del videojuego con el equipamiento que se ha ido obteniendo a lo largo del videojuego.

6.2 Ambientación del videojuego

En el siguiente punto se presenta la historia principal del videojuego, que se ha utilizado como base a la hora de crear los distintos diseños que pueden encontrarse durante el videojuego, dotándolos de sentido.

La historia de nuestro videojuego nos sitúa inicialmente en el reino de Plotkonia, donde nuestro protagonista, Hugo, es un mensajero normal y corriente al que el rey de Plotkonia, Areg II, le encarga la misión del anterior héroe del reino, tras el fallecimiento del mismo. Hugo deberá viajar por todo el continente con el objetivo de reunir las gemas legendarias custodiadas por enemigos muy poderosos para poder reparar el sello que se encuentra al borde de la destrucción y que mantiene retenida en las mazmorras del castillo una gran amenaza.

El rey Areg II, sin ningún plan en mente tras la muerte del anterior héroe, decide encomendar la misión a nuestro protagonista ignorando todos sus comentarios. De esta forma Hugo se ve obligado a aceptar la misión y emprender su aventura con el equipamiento más básico que le han concedido.

6.3 Diseños del videojuego

En este punto se presentan todos los modelos y diseños que se han utilizado durante el desarrollo del videojuego, así como una breve descripción exponiendo sus particularidades.

Mapa

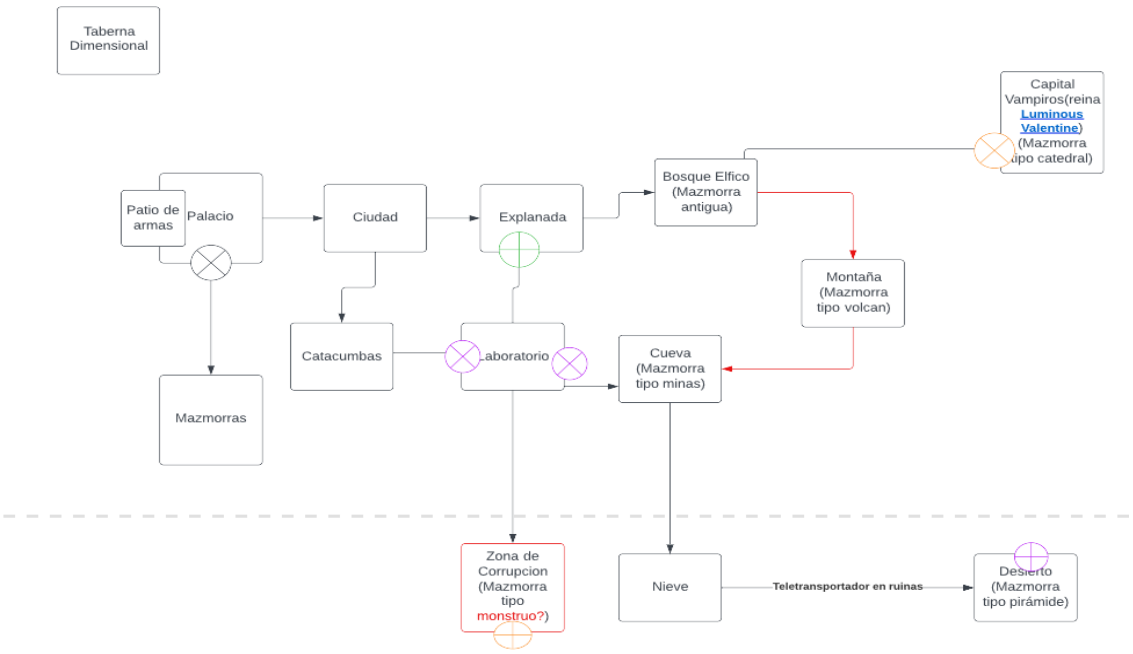

Imagen 10: Boceto básico inicial del mapa

En un principio, el concepto inicial que se pensó para la construcción del mapa consistía en distintas zonas interconectadas y ambientadas en distintos paisajes y escenarios (ver Imagen 10). Se comenzó primero produciendo un boceto básico inicial de todas las zonas que incorporaría el mapa teniendo en cuenta los distintos *assets* de los que se disponían conseguidos de diferentes localizaciones, como por ejemplo Itch.io o la *Unity Asset Store*, y la historia que se tenía pensada hasta ese momento para poder dotarlo de significado.

Una vez ya se tenía planeado el boceto básico inicial del mapa, se decidió acotar las zonas que se implementarían en el proyecto, teniendo en cuenta diversos aspectos, por ejemplo, la necesidad, la complejidad, el aspecto, la estética o lo bien definida y pensada que se tenía la zona. Finalmente se decidió implementar las siguientes zonas: Palacio, Ciudad, Bosque Élfico, Capital Vampiros y Taberna Dimensional.

Durante el transcurso del desarrollo del proyecto estas zonas fueron recibiendo modificaciones tanto en sus nombres como en sus contextos. Cada una de las zonas que se ha implementado posee un objetivo distinto. Seguidamente se introducen brevemente estas zonas:

- **Palacio:** A lo largo del desarrollo se ha decidido cambiar el nombre de esta zona por “Castillo”. Esta zona está constituida por cinco escenas sencillas y tiene como objetivo servir de tutorial, presentando las interacciones con los diferentes elementos del entorno y entregando al jugador un manual con los controles del videojuego en una de las escenas del Castillo llamada “Patio de armas”. A continuación, se muestran unas imágenes con las diferentes escenas que constituyen la zona del Castillo.

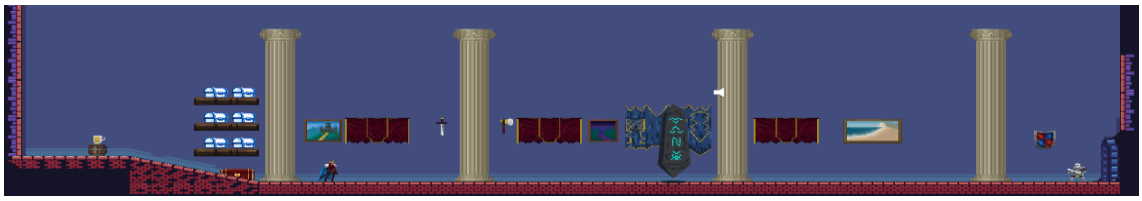

Imagen 11: Sala del trono

Imagen 12: Salón del castillo

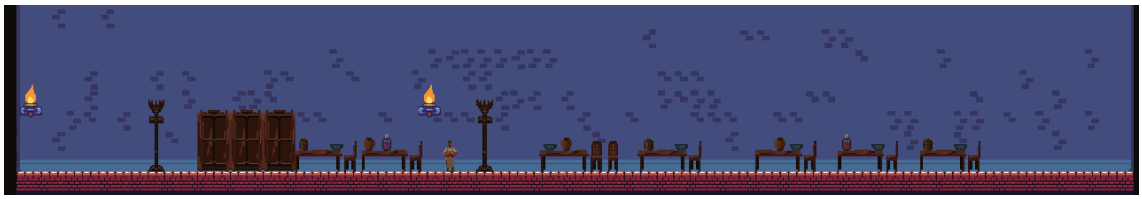

Imagen 13: Cocina del castillo

Imagen 14: Patio de armas

Imagen 15: Pasillo del castillo

- **Ciudad:** Durante el desarrollo se decidió cambiar el contexto de esta zona por el de un pueblo, cambiando también su nombre por “Pueblo”. Esta zona está formada por una escena sencilla con el objetivo de ser una escena entre el tutorial y el primer nivel con enemigos, además de tener un punto de acceso a la zona “Taberna Dimensional”.

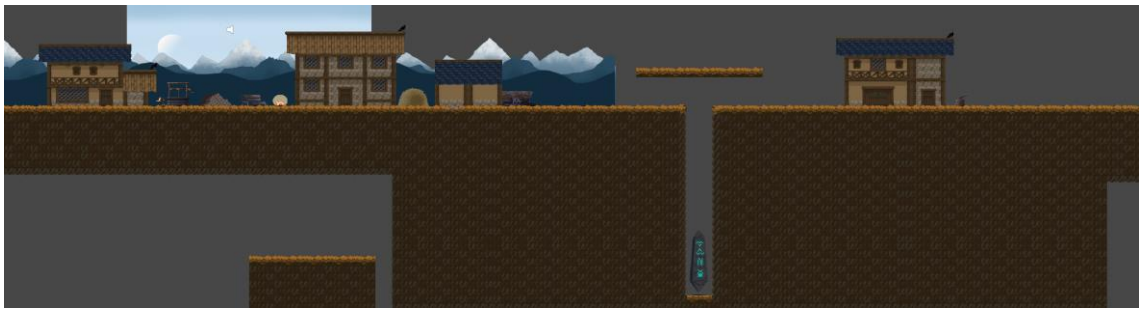

Imagen 16: Pueblo

- **Bosque Élfico:** Mientras se progresaba en el desarrollo se decidió cambiar el nombre de esta zona por “Bosque”. Esta zona consta de una escena y tiene como propósito ser el primer nivel con enemigos del videojuego, presentando de esta manera el sistema de combate.

Imagen 17: Bosque

- **Capital Vampiros:** En el transcurso del desarrollo se decidió cambiar el ambiente de esta zona por la de una mazmorra, cambiando también su nombre por “Mazmorra”. Asimismo, este cambio ocasiona que se tenga que cambiar el nombre de la zona “Mazmorra” que hay debajo de la zona “Palacio”, pero se decidió pensar en esto más adelante. Esta zona consiste en una sola escena que tiene la finalidad de ser una zona muy amplia con una cantidad abundante de enemigos, trampas y áreas ocultas a simple vista, además de ser el punto desde el que se puede acceder a las distintas escenas donde combatir con los jefes del videojuego.

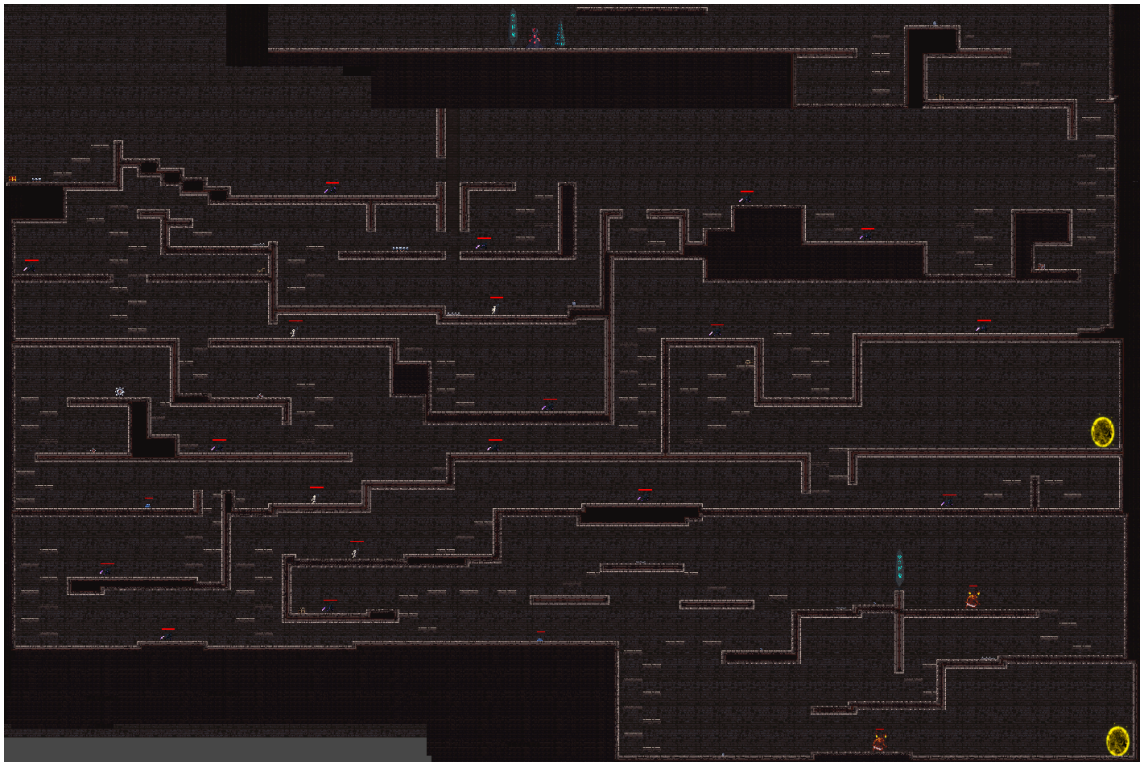

Imagen 18: Mazmorra

- **Taberna Dimensional:** Esta zona está compuesta de una escena y tiene como intención ser un área de descanso a partir de la cual poder viajar entre las distintas zonas del videojuego rápidamente sin necesidad de efectuar un trayecto ampliamente extenso. Adicionalmente, esta zona contiene diferentes NPCs (personajes no jugables, *non playable character*) con los que el jugador podrá interactuar para recibir consejos y curiosidades acerca del propio videojuego.

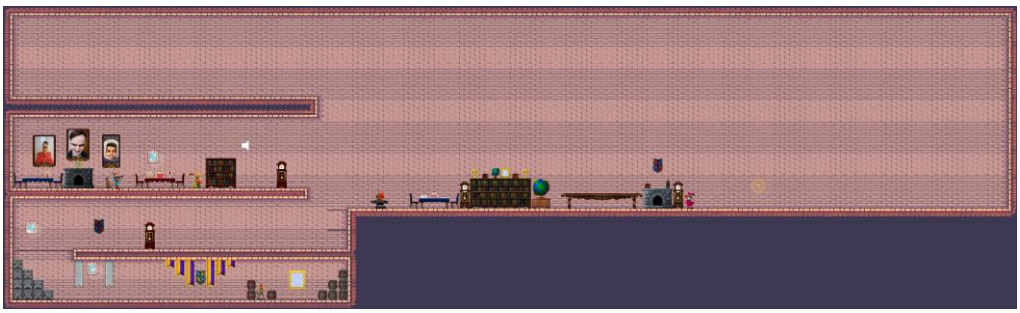

Imagen 19: Taberna Dimensional

El desafío principal que se ha encontrado a la hora de desarrollar las diferentes zonas del mapa del videojuego es diseñar las distintas zonas tomando en consideración los diferentes *assets* disponibles, ajustándolos para, de esta manera, lograr que se complementen y dándole un significado y sentido de acuerdo con la historia del videojuego.

Personaje principal

En un principio la descripción que se pensó para el personaje principal de la historia del videojuego era la siguiente: “Aventurero del rango más bajo que siempre ha evitado pelear, trabaja a tiempo parcial como mensajero para ganarse la vida. Está equipado con el equipo básico que se le ha proporcionado”. Posteriormente se decidió que el personaje principal de la historia empezaría siendo un mensajero normal y corriente llamado Hugo y después de que el rey de Plotkonia le asigne la misión de salvar el reino se convertiría en un aventurero.

El personaje principal del videojuego puede realizar distintas acciones como andar, saltar, rodar, atacar y protegerse. A continuación, se presentan unas imágenes del modelo del personaje principal y sus distintas acciones:

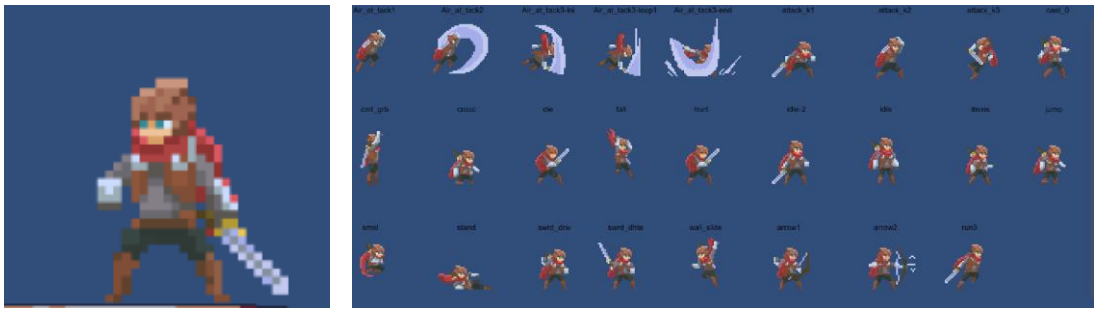

Imagen 20: Personaje Principal

Enemigos

A lo largo del desarrollo del videojuego se hicieron pruebas con distintos enemigos para poder filtrarlos y seleccionar a los enemigos que mejor se adecuaban a la zona. En el momento actual existen cinco tipos diferentes de enemigos normales y dos jefes, cada uno de ellos con estadísticas diferentes como se muestran en las tablas 3 y 4. Se ha empezado el desarrollo de un tercer jefe, no obstante, se encuentra en una etapa muy inicial.

Los enemigos normales se encuentran distribuidos por las distintas zonas del mapa del videojuego y tienen un comportamiento básico basado en un patrullaje de una región reducida de la zona hasta que detectan que el jugador se encuentra a cierta distancia, en ese momento lo perseguirán durante un tiempo para después retornar a su región de patrullaje inicial.

Por otro lado, los jefes se encuentran en unas zonas fijas del mapa del videojuego y tienen un comportamiento más complejo con mecánicas distintas dependiendo del jefe. A continuación, se muestran unas imágenes de los diseños de los enemigos junto con unas tablas de sus estadísticas:

Imagen 21: Enemigos

Enemigo	Estadísticas				
	Vida	Ataque	Defensa Física	Defensa Mágica	Experiencia
Arbol	100	24	54	20	20
Demonio	100	30	70	40	20
Esqueleto	1	5	50	50	1
Night Warrior	100	30	20	100	20
Slime	100	7	30	40	20

Tabla 3: Estadísticas de los enemigos

Imagen 22: Jefes

Jefe	Estadísticas				
	Vida	Ataque	Defensa Física	Defensa Mágica	Experiencia
Senob	150	15	125	60	250
Uyako	250	20	70	100	250

Tabla 4: Estadísticas de los jefes

Objetos

En este apartado se presentan los distintos objetos que se pueden encontrar en el videojuego. Uno de los puntos fundamentales que se desea alcanzar durante el desarrollo del videojuego es que cada partida sea diferente y variada, para lograr este objetivo se ha utilizado una amplia variedad de objetos con estadísticas aleatorias dentro de un rango, permitiendo de esta forma que, aunque en la misma partida obtengas un objeto repetido, las estadísticas de estos objetos seguramente serán

diferentes. Los objetos del videojuego se pueden conseguir abriendo cofres, pertenecen al sistema de inventario y se pueden dividir en diferentes categorías. Seguidamente se exponen estas categorías:

- **Espadas**

  Actualmente existen un total de 36 espadas diferentes en el videojuego que proporcionan estadísticas al jugador. La mayoría de estas espadas poseen un efecto distinto cuando se realiza el ataque especial. A continuación, se muestra una imagen con el diseño de las 36 diferentes espadas que podemos encontrar en el videojuego:

Imagen 23: Espadas

- **Accesorios**

  Durante el videojuego podemos hallar 3 tipos diferentes de accesorios que proporcionan una pequeña cantidad de estadísticas al jugador dependiendo de la calidad y del tipo de accesorio. Los 3 tipos de accesorios son: Anillos, Collares y Runas. Por otra parte, existen dos clases de anillos y collares, los normales y los mágicos, dependiendo de esta clase el accesorio proporcionará una estadística u otra. Seguidamente se muestra una imagen con el diseño de los distintos accesorios que existen en el videojuego:

Imagen 24: Accesorios

- **Armaduras**

En el videojuego podemos encontrar un total de 14 sets de armaduras diferentes, cada uno de estos sets de armadura está compuesto por 4 piezas de armadura: Casco, Pechera, Pantalones y Zapatos. Estas piezas de armadura proporcionan al jugador estadísticas distintas dependiendo del set al que pertenezcan. A continuación, se muestra una imagen con el diseño de los distintos sets de armadura:

Imagen 25: Armaduras

- **Pociones**

Las pociones son uno de los métodos de recuperarse sin morir en el videojuego. Al principio del desarrollo solo había 10 pociones diferentes divididas en 2 grupos, posteriormente se decidió implementar 10 pociones más, también divididas en 2 grupos nuevos, dando un total de 20 pociones diferentes divididas en 4 grupos de 5 pociones cada grupo. Los dos primeros grupos son las pociones de vida normales (roja) y las pociones de mana normales (azul) y los dos grupos que se implementaron más tarde son las pociones de vida benditas (amarilla) y las pociones de mana benditas (verde). Más adelante se entrará en detalle en los efectos de las pociones.

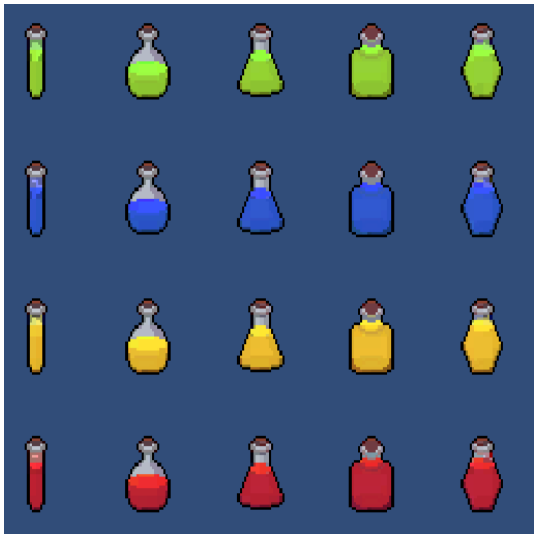

Imagen 26: Pociones

- **Gemas**

Las gemas son los objetos que el jugador debe reunir para desbloquear el final del videojuego, en otras palabras, el objetivo principal del videojuego. Esta categoría de objetos no se puede obtener en los cofres. Para poder conseguir estos objetos el jugador deberá derrotar a los distintos jefes del videojuego. Actualmente solo se pueden conseguir dos de las cinco gemas.

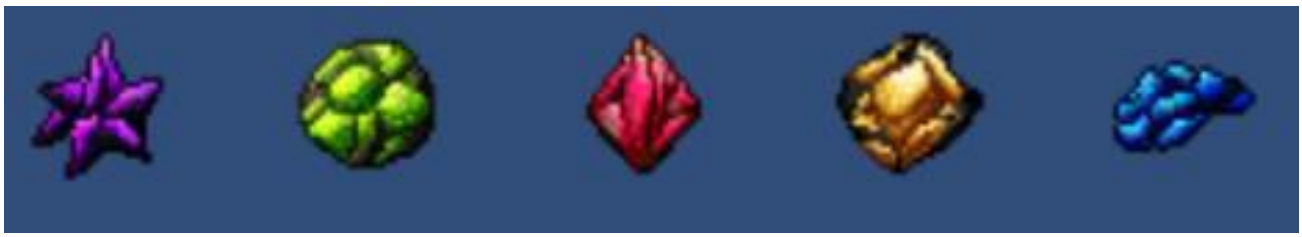

Imagen 27: Gemas

- **Manual del aventurero**

El manual del aventurero es un objeto especial que se decidió implementar como un objeto de ayuda para que el jugador pueda consultar los controles del videojuego en cualquier momento. Este objeto no se puede obtener en los cofres, la única forma de obtenerlo es terminar el tutorial en el Patio de armas del Castillo.

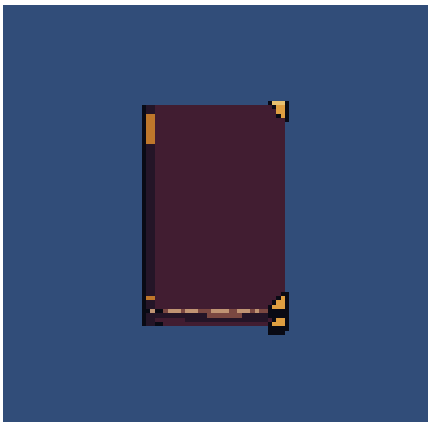

Imagen 28: Manual del aventurero

Entorno

En este apartado se van a presentar los diseños de los elementos del entorno más importantes que podemos encontrar distribuidos por el mapa del videojuego junto con una breve descripción de los mismos.

- **Trampas**

Las trampas son elementos del entorno cuyo propósito es el de obstaculizar el avance del jugador. Algunas de estas trampas se encuentran ocultas hasta que el jugador se encuentra lo suficientemente cerca como para que se activen, consiguiendo de esta forma que el jugador esté alerta en todo momento y añadiendo algo de dificultad adicional al videojuego.

La idea de tener trampas distribuidas por el mapa se pensó al inicio del proyecto, no obstante, no se pudieron implementar hasta que el proyecto estuvo bastante avanzado, pues había otros aspectos de mayor importancia. Tras un proceso de selección, y teniendo en cuenta la armonía con el mapa, las trampas que se decidieron implementar son las que se pueden observar en la siguiente imagen:

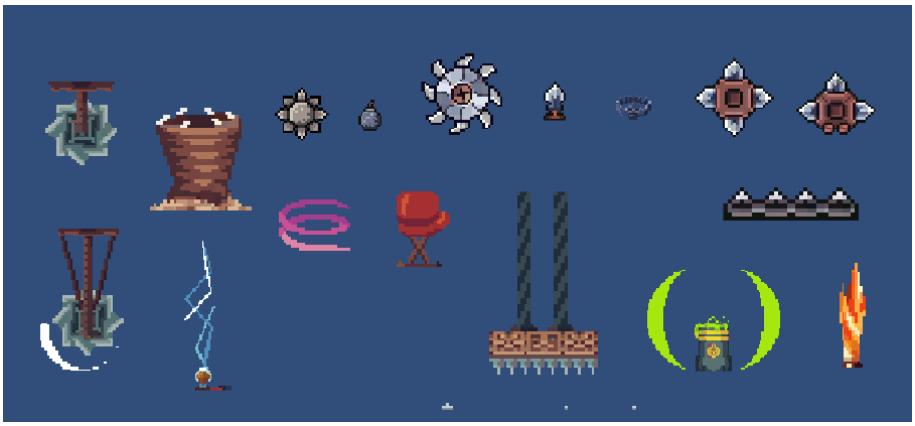

Imagen 29: Trampas

- **Cofres**

Los cofres son la forma principal de obtener objetos en el videojuego. Existen cuatro tipos diferentes de cofre y dependiendo del tipo se tendrá más o menos posibilidades de obtener un mejor objeto. En un principio los objetos se iban a obtener al derrotar a los enemigos, pero a medida que el proyecto se fue desarrollando se fueron añadiendo cada vez más cantidad de enemigos y al final se decidió implementar un sistema diferente para la obtención de los objetos, los cofres.

A continuación, se puede ver una imagen que muestra los distintos tipos de cofre que podemos encontrar distribuidos por el mapa del videojuego. Posteriormente se explicará más detalladamente el comportamiento de los cofres.

Imagen 30: Cofres

- **Torres**

Las torres son elementos del entorno que proporcionan al jugador cierta utilidad dependiendo de su tipo. Actualmente existen 3 tipos distintos de torres (ver Imagen 31). De izquierda a derecha estas torres son:

- PilarTP: Estas torres se han decidido implementar con la finalidad de solucionar una de las cuestiones que se tenían al principio del desarrollo del proyecto: “¿De qué forma podrá viajar el jugador a la Taberna Dimensional?”. Estas torres sirven como teletransportes ubicados en lugares importantes del mapa del videojuego. Al

principio al entrar en contacto con ellas te teletransportaban a la Taberna Dimensional, posteriormente se decidió que tenía más sentido dar la elección al jugador haciendo que solo se teletransporte si decide interaccionar con la torre.

- TorreCheckpoint: Estas torres se han implementado con la intención de solucionar una de las cuestiones más importantes del videojuego: “¿Qué ocurre cuando el jugador muere en el videojuego?”. Una vez el proyecto estaba avanzado se pensó en la idea de un elemento del entorno que permita guardar la posición del jugador cuando pasa por esta ubicación, haciendo que el jugador regrese a esa localización en caso de morir.

- TorreCuracion: Estas torres se decidieron implementar con el objetivo de ayudar al jugador durante su aventura. Estas torres se sitúan al inicio de algunas zonas con el propósito de recuperar la vida del jugador.

Imagen 31: Torres

- **Elevador**

El elevador es un elemento del entorno utilizado para poder conectar algunas partes en concreto del mapa del videojuego sin la necesidad de abusar de las plataformas para poder recorrer una distancia en vertical. Para poder utilizar el elevador el jugador simplemente deberá golpear la palanca que hay situada en el mismo. A continuación, se muestra una imagen con el diseño del elevador:

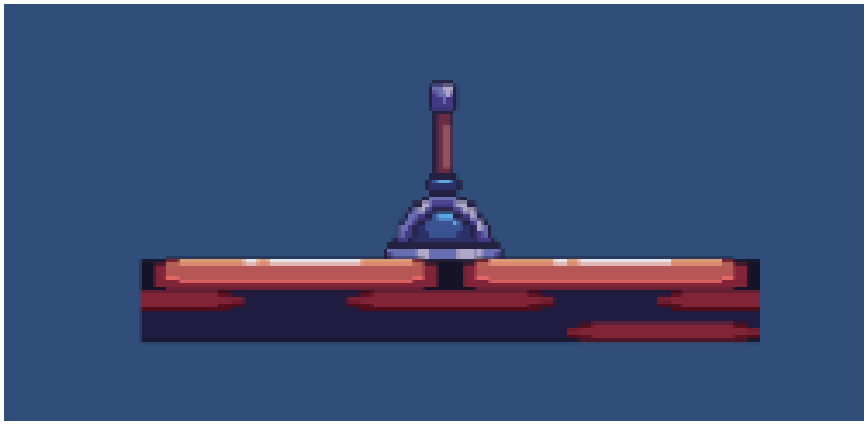

Imagen 32: Elevador

6.4 Controles del videojuego

En este punto se presentan los controles utilizados en el videojuego. Estos controles se pueden consultar dentro del videojuego utilizando el Manual del Aventurero. También se ha pensado añadir una opción en el menú principal del videojuego para poder consultarlos antes de empezar a jugar, pero esta idea no se ha llevado a cabo por el momento.

- Moverse a la izquierda: Tecla A
- Moverse a la derecha: Tecla D
- Saltar: Tecla Espacio
- Enfundar/Desenfundar Arma: Tecla E
- Ataque: Click Izquierdo
- Ataque Especial: (En el suelo) Tecla Q
- Ataque Descendente: (En el aire) Tecla Q
- Escudo: Click Derecho
- Interactuar con NPCs: Tecla F
- Avanzar Diálogos: Tecla F
- Abrir/Cerrar Inventario: Tecla I
- Ver descripción: (En el inventario) Click Derecho sobre el objeto
- Equipar/Usar objeto: (En el inventario) Click Izquierdo sobre el objeto
- Ocultar/Mostrar Inventario: (En el inventario) Presionar Botón I
- Ocultar/Mostrar Subir Estadísticas: (En el inventario) Presionar Botón +
- Cerrar Menú Portales: Tecla P
- Cerrar Manual: Tecla I

6.5 Interfaces del videojuego

Las distintas interfaces del videojuego están diseñadas para adaptarse a las pantallas de PC. En este punto se expondrán cada una de las diferentes interfaces que se pueden encontrar en el videojuego describiendo cada uno de los elementos que las constituye.

1. **Barra de vida**

   La barra de vida es el medidor que indica la vida actual de la que dispone el jugador. Cuando el jugador recibe daño esta barra se reduce, cuando el jugador se recupera la vida con alguna poción esta barra aumenta y cuando la barra se vacía el jugador muere.
2. **Barra de mana**

   La barra de mana es el medidor que indica el mana actual del que dispone el jugador. Cuando el jugador utiliza un ataque especial la barra se reduce, cuando el jugador se recupera el mana con alguna poción esta barra aumenta y cuando la barra se vacía el jugador no puede activar los efectos del ataque especial.

Imagen 33: Interfaz de vida y mana

3. **Inventario**

   El inventario es la interfaz que muestra los objetos que ha recogido el jugador durante su aventura. El inventario está compuesto por 30 slots.

4. **Interfaz de estadísticas**

   La interfaz de estadísticas es la interfaz que recoge y muestra todo lo relacionado con las estadísticas del jugador. Esta interfaz de estadísticas está compuesta por la barra de experiencia, la lista de estadísticas, el inventario de equipamiento y los botones para mostrar/ocultar el inventario y la interfaz de subir estadísticas.

5. **Barra de experiencia**

   La barra de experiencia es el medidor que indica la experiencia actual que posee el jugador. Cuando el jugador mata a un enemigo la barra aumenta, cuando la barra se llena el jugador sube de nivel, recibe un punto de estadísticas y la barra se reinicia.

6. **Lista de estadísticas**

   La lista de estadísticas muestra los puntos actuales de cada una de las diferentes estadísticas que posee el jugador.

7. **Slot de inventario**

   El slot de inventario es cada uno de los elementos que componen el inventario. En cada uno de estos slots de inventario se puede almacenar un tipo de objeto.

8. **Inventario de equipamiento**

   El inventario de equipamiento es el elemento de la interfaz de estadísticas que muestra el equipamiento actual del jugador.

9. **Botón Mostrar/Ocultar inventario**

   El botón Mostrar/Ocultar inventario es un botón que el jugador puede utilizar para ocultar o mostrar el inventario situado a la izquierda. Si el jugador presiona el botón y el inventario se está mostrando este se ocultará, pero si cuando el jugador presiona el botón el inventario está oculto este se mostrará.

10. **Botón Mostrar/Ocultar subir estadísticas**

    El botón Mostrar/Ocultar subir estadísticas es un botón que el jugador puede utilizar para mostrar u ocultar la interfaz de subir estadísticas. Si el jugador presiona el botón y la interfaz de subir estadísticas está oculta esta se

mostrará, pero si cuando el jugador presiona el botón la interfaz de subir estadísticas se está mostrando esta se ocultará.

11. **Interfaz Información Objeto**
    La interfaz Información Objeto es la interfaz que muestra la información relacionada con el objeto que se ha seleccionado.

Imagen 34: Interfaz del inventario

12. **Interfaz de subir estadísticas**
    La interfaz de subir estadísticas es la interfaz que alberga todo lo implicado con utilizar los puntos de estadísticas recibidos por subir de nivel.
13. **Botón Restar punto**
    El botón Restar punto es un botón que el jugador puede utilizar para quitar un punto de estadísticas a esa estadística si desea utilizarlo en otra. Este botón solo se mostrará si esa estadística tiene algún punto de estadísticas asociado a la misma.
14. **Imagen de estadística**
    La imagen de estadística es un icono que hace referencia a la estadística que se verá afectada al sumar o restar puntos de estadísticas.
15. **Botón Sumar punto**
    El botón Sumar punto es un botón que el jugador puede utilizar para añadir un punto de estadísticas a esa estadística. Este botón solo aparece si el jugador cuenta con algún punto de estadísticas sin utilizar.

Imagen 35: Interfaz de subir estadísticas

7 Desarrollo del proyecto

En esta sección se exponen los elementos esenciales que se han llevado a cabo durante la fase de desarrollo del proyecto. Esta sección se encuentra fraccionada en diferentes puntos que ayudan a entender mejor como se ha desarrollado el proyecto.

7.1 Menú principal

Tal y como se ha mencionado anteriormente el videojuego desarrollado para este TFG consta de diversas escenas, cada una de ellas con sus propios *GameObjects*. El menú principal es la primera escena que aparece al iniciar el videojuego. Es una escena sencilla compuesta por un *GameObject* llamado *canvas* que a su vez se encuentra formada por otros *GameObjects*: una imagen para el fondo del menú, tres botones, un sistema de partículas, una imagen negra para producir una animación de "*fade-in*/*fade-out*" y la interfaz de opciones del videojuego.

Seguidamente, se exponen los componentes más relevantes del *canvas* que compone el menú principal.

Componentes

- **Animator:** El componente *Animator* del *canvas* es el encargado de controlar las animaciones de FadeIn y FadeOut.

- **Audio Source:** El componente *Audio Source* es el encargado de reproducir la música del menú principal. Tiene enrutado en la salida el *Audio Mixer* Musica que se aclarará en detalle en un apartado posterior.

- **MenuPrincipal (Script):** El script MenuPrincipal es el componente encargado del funcionamiento del menú principal. Se encarga de definir el comportamiento de los 3 botones del menú principal, es decir, que ocurre cuando se pulsa cada botón: activar animaciones, cambiar de escena, mostrar/ocultar la interfaz de opciones y/o cerrar el juego.

- **Settings (Script):** El script Settings es el componente encargado de establecer el volumen de la música y los efectos de sonido, según lo configurado en la interfaz de opciones, interactuando con los dos *Audio Mixers*: Musica y Efectos. Si el valor de alguna de estas opciones cambia este script se encarga de establecer los nuevos valores de forma inmediata.

Imagen 36: Menú Principal

7.1.1 Interfaz de opciones

La interfaz de opciones es un *GameObject* colocado en la escena del menú principal. Se encuentra conformada por una variedad de imágenes empleadas para elaborar el fondo de la interfaz, dos *sliders* para proporcionar la posibilidad de configurar las opciones de volumen de la música y los efectos de sonido, empleando el script Settings, y dos textos que se aprovechan para mostrar el valor numérico actual del *slider* recurriendo al script MostrarNumero.

Imagen 37: Interfaz de opciones

7.2 Sistema de diálogos y monólogos

El sistema de diálogos es uno de los apartados imprescindibles del videojuego, puesto que a través de este es posible narrar la historia principal del videojuego al jugador. Este sistema es simple y fácil de comprender desde el punto de vista del jugador, aunque moderadamente más complejo en lo que respecta a su desarrollo.

El sistema de diálogos se encuentra compuesto por tres scripts fundamentales que se comentan acto seguido:

- **Textos:** El script Textos sirve para determinar las oraciones y frases que se mostrarán, especificar si se trata de un monologo y dar la oportunidad de establecer un objeto a soltar una vez finalizado ese diálogo o monologo.
- **PersonajeInteractuable:** El script PersonajeInteractuable se incluye como componente de un *GameObject* con el cual el jugador tiene la posibilidad de interactuar para iniciar un diálogo o monologo. Este script aprovecha el script Textos explicado anteriormente para determinar los diversos diálogos y/o monólogos de los que dispone asociados dicho *GameObject*, confiriendo la posibilidad de configurar el componente con la finalidad de que estos diálogos y/o monólogos se puedan volver a reproducir, retornando a iniciar desde el principio una vez que ha concluido el último o establecer que este diálogo y/o monologo que se presenta se determine de manera aleatoria escogiendo de entre los diversos diálogos y/o monólogos de los que dispone asociados dicho *GameObject*. Por otra parte, este script se interrelaciona interaccionando con

el script ComprobadorMision, el cual se explicará más adelante, con el objetivo precisar si se ha llevado a cabo una misión o no.

- **ControlDialogos:** El script ControlDialogos es el responsable de controlar todo lo que sucede en el transcurso de un diálogo y/o monólogo. Sus funciones más importantes son las siguientes:
  - **ActivarDialogo:** La función ActivarDialogo se encarga principalmente de activar la animación ActivarDialogo. Al final de esta animación hay establecido un evento para llamar a la función ActivaFrase de este script.
  - **ActivaFrase:** La función ActivaFrase se encarga de mostrar la primera oración o frase del diálogo o monologo aprovechando la función MostrarCaracteres.
  - **SiguienteFrase:** La función SiguienteFrase se encarga de intercambiar la oración o frase actual del diálogo o monologo por la nueva oración o frase siguiente que corresponde mostrándola recurriendo a la función MostrarCaracteres e intercambiando la imagen que aparece en la interfaz en caso de ser un diálogo. Cuando la oración o frase actual es la última del diálogo o monologo esta función llamará a la función CerrarDialogo.
  - **CerrarDialogo:** La función CerrarDialogo se encarga de activar la animación CerrarDialogo y, en caso de encontrarse configurado, soltar el objeto del script Textos de ese diálogo o monologo.
  - **MostrarCaracteres:** La función MostrarCaracteres es una corrutina encargada de mostrar la oración o frase carácter por carácter en lugar de mostrarla completa desde el principio.

Imagen 38: Interfaz del sistema de diálogos y monólogos

7.3 Sistema de misiones

Justo como se ha explicado anteriormente el sistema de misiones del videojuego está ampliamente relacionado con el sistema de diálogos. Este sistema de misiones se encuentra compuesto básicamente por los siguientes dos scripts:

- **ComprobadorMision:** El script ComprobadorMision es uno de los componentes que contiene el personaje principal del videojuego. Este script se encarga de guardar el estado de las misiones que ha completado el jugador. En el momento que el jugador completa una misión el script PersonajeInteractuable llama a este script con el propósito de que se guarde el estado de la misión en una variable que pueda indicar que la misión se ha realizado, de esta manera se podrá comprobar que misiones ha completado el jugador y que misiones todavía tiene pendientes aún por completar.
- **Comprobador:** El script Comprobador se encarga de consultar si se ha completado una misión realizando una llamada al script ComprobadorMision. En caso afirmativo el script se encarga de desactivar de la escena los *GameObjects* que se encuentran configurados. Este script sirve mayoritariamente para desactivar muros invisibles y/o NPCs en caso de que estos no sean necesarios una vez que se ha comprobado que cierta misión ha sido completada. Este script es un componente incorporado en un *GameObject* vacío colocado en la escena llamado Comprobador y que se ejecuta al entrar en dicha escena.

7.4 Manual del Aventurero

El Manual del Aventurero es un objeto especial que se consigue exclusivamente después de terminar el tutorial del videojuego en el Patio de armas del Castillo. Se eligió implementar este objeto en una etapa temprana del desarrollo del videojuego con la intención de tener la posibilidad de consultar todos los controles del videojuego en cualquier momento. De esta manera, se consiguió perfeccionar su funcionamiento y contenido, a medida que se progresaba en el transcurso de la etapa de desarrollo.

En el momento en que el jugador usa el objeto especial del Manual del aventurero desde el sistema de inventario, se ejecuta una llamada al script ManualAventurero, ubicado como componente de un *GameObject* con el mismo nombre que el script. Dicho *GameObject* es uno de los elementos que conforman el *GameObject* que ejerce la función de *Canvas* principal del videojuego.

El *Canvas* principal del videojuego es un *GameObject* que únicamente se instancia una vez al inicio del mismo, haciendo uso del patrón de diseño y programación Singleton, el cual se explicará más detalladamente en un apartado posterior. El script ManualAventurero, conjuntamente con el *Animator Controller* correspondiente, ubicados ambos como componentes en el *GameObject* ManualAventurero son los encargados de controlar el correcto funcionamiento del Manual del aventurero mostrando todos los controles del videojuego y permitiendo iniciar las animaciones de “Abrir” y “Cerrar” del Manual del aventurero en el momento que el jugador lo desee.

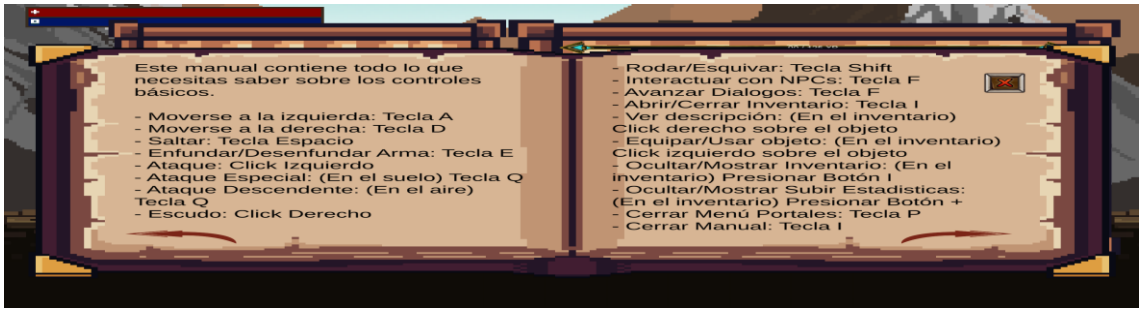

Imagen 39: Interfaz del Manual del aventurero

7.5 Sistema de combate

El sistema de combate que se ha decidido desarrollar para el videojuego no se desvía excesivamente del sistema de combate que comúnmente emplean los videojuegos del genero Metroidvania. Este sistema de combate se caracteriza por ser un sistema de combate en tiempo real, en los que el tiempo transcurre de manera continua para los jugadores forzándoles a tomar decisiones más rápidamente que un sistema de combate por turnos y proporcionando una sensación de acción más dinámica y frenética.

7.5.1 Ataques especiales

En este punto se describe una de las piezas fundamentales y más llamativas del sistema de combate, los ataques especiales. Los ataques especiales son un recurso ampliamente empleado en los videojuegos de género Metroidvania, debido a que estos ataques especiales y su funcionamiento permiten distinguirlos dotándolos de distintas estrategias y ofreciendo al jugador la oportunidad de decidir en qué ocasiones considera oportunas y convenientes utilizarlos. Debido a este motivo, la idea de desarrollar ataques especiales se ha encontrado presente desde el inicio del proyecto.

Con el objetivo de lograr poner en correcto funcionamiento este sistema de combate basado en ataques especiales se emplea el uso de cuatro scripts principalmente:

- **AtaqueEspecial:** El script AtaqueEspecial es llamado desde el script MovimientoPersonaje, indicándole el arma que posee equipada el personaje principal en el momento en que el jugador ejecuta un ataque especial con el suficiente mana. Este script se encarga principalmente de instanciar el *Prefab* “Efectos Arma” como un *GameObject* en la escena actual, teniendo en consideración la posición y dirección del personaje principal, y activar el *GameObject* del efecto, junto con sus animaciones, y/o el sonido correspondiente del arma, con la condición de que disponga de ellos, aprovechando el script ActivadorSonidos que se describirá detalladamente más tarde.
- **LanzarEfecto:** El script LanzarEfecto se puede encontrar introducido como un componente exclusivamente en los *GameObjects* relacionados a los efectos de los ataques especiales que lanzan alguna especie de proyectil hacia delante, siempre teniendo en cuenta la dirección del personaje principal en el momento en que el jugador a ejecutado el ataque especial. Este script se encarga tanto de proporcionar velocidad de movimiento al *GameObject* del efecto como de detectar el momento en que colisiona con un enemigo, suelo o pared, iniciando la animación final del efecto.
- **LanzarMeteorito:** El script LanzarMeteorito se comporta prácticamente del mismo modo que el script LanzarEfecto mencionado previamente, con las principales diferencias de que este script permite que el efecto realice un vector inclinado descendente, en lugar de únicamente hacia delante, y que al colisionar con un enemigo, suelo o pared inicia dos animaciones simultáneamente. Este script únicamente se encuentra introducido como un componente exclusivo en el *GameObject* relacionado con el efecto de un ataque especial en concreto.

- **Destroy:** El script Destroy se llama inmediatamente después de que la última animación relacionada con el efecto del ataque especial ha concluido. El script se encarga principalmente de destruir el *GameObject* “Efectos Arma” instanciado en la escena y de detener el sonido asociado al efecto, en caso de tenerlo.

7.5.2 Sistema de recuperación

En este apartado veremos en detalle el sistema de recuperación de puntos de vida y mana del personaje. En el momento actual existen tres métodos distintos que el jugador puede ser capaz de emplear para recuperar puntos de vida y/o mana del personaje principal del videojuego. Estos tres métodos son las pociones, las torres de curación y resucitar tras morir.

Seguidamente, se describe de manera detallada el funcionamiento de cada uno de estos métodos:

- **Pociones:** Tal y como se ha mostrado anteriormente, las 20 pociones diferentes que se han desarrollado actualmente en el videojuego se pueden agrupar en 4 grupos de 5 pociones cada grupo (ver Imagen 26):
  - **Pociones de vida normales (Rojas):** Las pociones de vida normales, desde el nivel 1 hasta el nivel 5, recuperan una cantidad de puntos vida prefijados de acuerdo con el nivel de la poción. Respectivamente, la cantidad de puntos de vida que recuperan es la que se muestra a continuación: 10, 20, 30, 40 y 50 puntos de vida.
  - **Pociones de mana normales (Azules):** Las pociones de mana normales, desde el nivel 1 hasta el nivel 5, recuperan una cantidad de puntos de mana prefijados de acuerdo con el nivel de la poción. Respectivamente, la cantidad de puntos de mana que recuperan es la que se muestra a continuación: 10, 20, 30, 40 y 50 puntos de mana.
  - **Pociones de vida benditas (Amarillas):** Las pociones de vida benditas, desde el nivel 1 hasta el nivel 5, recuperan un porcentaje de los puntos de vida máximos del personaje principal en el momento en el que el jugador toma la decisión de usar la poción. En función del nivel de la poción, respectivamente, los porcentajes de puntos de vida que recuperan son los que se muestran a continuación: 10%, 20%, 30%, 40% y 50% de los puntos de vida máximos del personaje principal.
  - **Pociones de mana benditas (Verdes):** Las pociones de mana benditas, desde el nivel 1 hasta el nivel 5, recuperan un porcentaje de los puntos de mana máximos del personaje principal en el momento en el que el jugador toma la decisión de usar la poción. En función del nivel de la poción, respectivamente, los porcentajes de puntos de mana que recuperan son los que se muestran a continuación: 10%, 20%, 30%, 40% y 50% de los puntos de mana máximos del personaje principal.
- **Torres de curación:** Las torres de curación se pueden encontrar situadas al inicio de algunas de las zonas del videojuego. Esta torre ayuda al jugador recuperando todos los puntos de vida del personaje principal al entrar en contacto con la misma, sin modificar los puntos de mana de los que dispone el personaje principal en ese momento, por medio del script TorreCuracion. Se han desarrollado torres de curación que disponen con la capacidad de recuperar los puntos de vida del personaje principal en infinitas ocasiones y

otras que únicamente disponen con la capacidad de recuperar los puntos de vida del personaje principal una sola vez.

- **Resurrección del personaje principal:** El último método, y generalmente el más indeseado por el jugador, para poder recuperar completamente tanto los puntos de vida como de mana del personaje principal consiste en resucitar. Este método solamente sucede en el momento en el que los puntos de vida del personaje principal se reducen a 0, teletransportándolo a la última TorreChekpoint con la que el personaje principal ha entrado en contacto.

7.6 Trampas

Tal y como se mostró anteriormente, las trampas son uno de los elementos del entorno que contribuyen tanto a complementar la ambientación del videojuego como a agregar a la dificultad un poco de nivel extra, obstaculizando el avance y manteniendo alerta al jugador en todo momento.

Es posible lograr conseguir un correcto funcionamiento de las trampas debido principalmente a la utilización de 5 scripts que se indican con más detalle a continuación:

- **ActivarTrampa:** El script ActivarTrampa, incluido como componente de la mayor parte de los *Prefabs* relacionados con las trampas, se encarga básicamente de comprobar la distancia a la que se encuentra el personaje principal en comparación con el *GameObject* relacionado con la trampa que posee comprendido este script como componente, iniciando las animaciones de este *GameObject*, solo si se cumple la condición de que el personaje principal se encuentra a una distancia lo suficientemente cerca del *GameObject* relacionado con la trampa.
- **ActivarTrampaOso:** El script ActivarTrampaOso actúa fundamentalmente del mismo modo que el script ActivarTrampa descrito previamente, con la importante distinción de que este script inicia las animaciones del *GameObject* relacionado con la trampa, únicamente en caso de que el personaje principal colisione con el *Collider* de este *GameObject*, en lugar de tomar en cuenta la distancia con el mismo.
- **TrampaPinchoMovil:** El script TrampaPinchoMovil se ha desarrollado con la intención de proporcionar movimiento a una de las trampas que cuenta con un nombre idéntico al de este script, TrampaPinchoMovil, consiguiendo lograr alcanzar un comportamiento similar al de enemigos del videojuego, con la fundamental variación de que esta trampa no perseguirá al personaje principal. El script TrampaPinchoMovil se encarga de controlar el movimiento del *GameObject* relacionado con la trampa produciendo que este se desplace una cierta distancia a una cierta velocidad configuradas, inmediatamente después de desplazarse esta distancia el script se ocupa de controlar el movimiento del *GameObject* con el propósito de que este ejecute el desplazamiento de vuelta con dirección a la posición inicial con la intención de volver a comenzar el ciclo de movimiento de nuevo.
- **DanyoTrampas:** El script DanyoTrampas, situado como componente junto con los *Colliders* de las trampas, se encarga tanto de determinar el tipo de daño de la trampa, físico o mágico, como de ejecutar una llamada al script BarraDeVida, incorporado como componente del personaje principal, para poder reducir los puntos de vida correspondientes, teniendo en consideración

la estadística de defensa respectiva del personaje principal, en el caso de que este script confirme que el personaje principal ha colisionado con la trampa.

- **DestroyThis:** El script DestroyThis se comporta de un modo similar al script Destroy de los ataques especiales destacado anteriormente, con la principal diferencia de que este script en lugar de destruir el *GameObject* "Efectos Arma" destruirá el *GameObject* que disponga asociado este script como componente. Este script se aplica particularmente siempre que se pretende destruir una trampa al finalizar la ejecución del último de sus efectos de animación, en particular, con los *GameObjects* TrampaBomba, los cuales tras finalizar sus animaciones y explotar tienen que ser destruidos y desaparecer de la escena.

7.7 Cofres

Los cofres son uno de los elementos del entorno más decisivo e influyente del videojuego, debido a que, tal y como bien se ha expuesto en un apartado previamente, son el principal método de conseguir adquirir los distintos objetos en el videojuego. Los cofres se pueden lograr encontrar distribuidos por el mapa del videojuego, algunos de ellos colocados en zonas ocultas mientras que otros se encuentran situados más a simple vista. Ordenados de peor a mejor los cuatro diferentes tipos de cofres que podemos encontrar son: Madera, Hierro, Oro y Diamante.

Con el objetivo de ser capaces de lograr conseguir comprender mejor el comportamiento de los cofres primero tenemos que entender que los distintos objetos del videojuego que se pueden conseguir obtener con los cofres se encuentran agrupados en seis grupos independientes que determinan la calidad del objeto, los distintos grupos, ordenados de peor a mejor son: Hierro, Bronce, Oro, Platino, Legendaria y Reliquia. De acuerdo con el tipo de cofre hay más o menos probabilidades de conseguir obtener un objeto de calidad superior.

El script que hace posible el apropiado funcionamiento de los cofres es el script Cofres, cuyo comportamiento se especifica en mayor detalle seguidamente:

- **Cofres:** El script Cofres se encarga sobre todo de desempeñar dos tareas. La primera tarea del script consiste tanto en comprobar cuando el personaje principal se encuentra colisionando con el *GameObject* del cofre como de comprobar si el cofre todavía no ha sido abierto, iniciando en este caso la animación de un bocadillo flotante con la intención de indicar al jugador que tiene la posibilidad de interactuar con ese cofre. En el momento en el que el jugador toma la decisión de interactuar con el cofre, el script se encarga de desempeñar su segunda tarea, que consiste tanto en iniciar la animación de abrir el cofre como de, aplicando un sistema de probabilidades basado en el tipo de cofre, computar la calidad del objeto que será instanciado, una vez calculada la calidad del objeto, se determinará, mediante otro sistema de probabilidades, el objeto en concreto a instanciar que pertenezca al grupo de dicha calidad. Por último, merece la pena resaltar que, tal y como se ha mencionado anteriormente, dependiendo de la clase de objeto instanciado poseerá estadísticas aleatorias dentro de un rango de valores, favoreciendo de esta manera la aleatoriedad y alcanzando el dinamismo que se deseaba para el proyecto, consiguiendo que a pesar de que puedas obtener objetos repetidos, en la misma partida o en una diferente, las estadísticas que posea este objeto generalmente no serán las mismas.

7.8 Sistema de sonidos

En esta sección se expone en detalle una pieza clave elemental para la ambientación y que aporta contexto al mundo que se ha empleado como fundamento base mientras se lleva a cabo el desarrollo del videojuego, creando una mayor sensación de inmersión del jugador en el mismo, el sistema de sonidos.

Con el fin de conseguir lograr una apropiada ejecución y funcionamiento del sistema de sonidos en este videojuego se han desarrollado, empleando la ventana Mezclador de Audio descrita en un apartado anterior, dos *Audio Mixers*: Musica y Efectos. Estos *Audio Mixers* se hacen cargo, respectivamente, de gestionar y controlar los sonidos de la música y los efectos integrados en el videojuego. Para ello simplemente se necesita configurar el *Audio Mixer* enrutándolo en la salida del *Audio Source* correspondiente según se encuentre asociado con la música o con algún efecto de sonido del videojuego. Merece la pena hacer hincapié en que, tal y como se ha expuesto previamente, el volumen de ambos *Audio Mixers* se puede configurar desde la interfaz de opciones del Menú Principal.

En lo que respecta a la implementación de la música en el videojuego, únicamente se necesita crear un solo *GameObject* en la escena y adjuntarle como componente un *Audio Source*, configurándolo con el clip de audio correspondiente, el *Audio Mixer* Musica e indicando que tiene que empezar a sonar en el instante en el que se carga la escena (*Play On Awake*) y tiene que repetirse el clip de audio en el momento en el que finaliza el mismo (*Loop*).

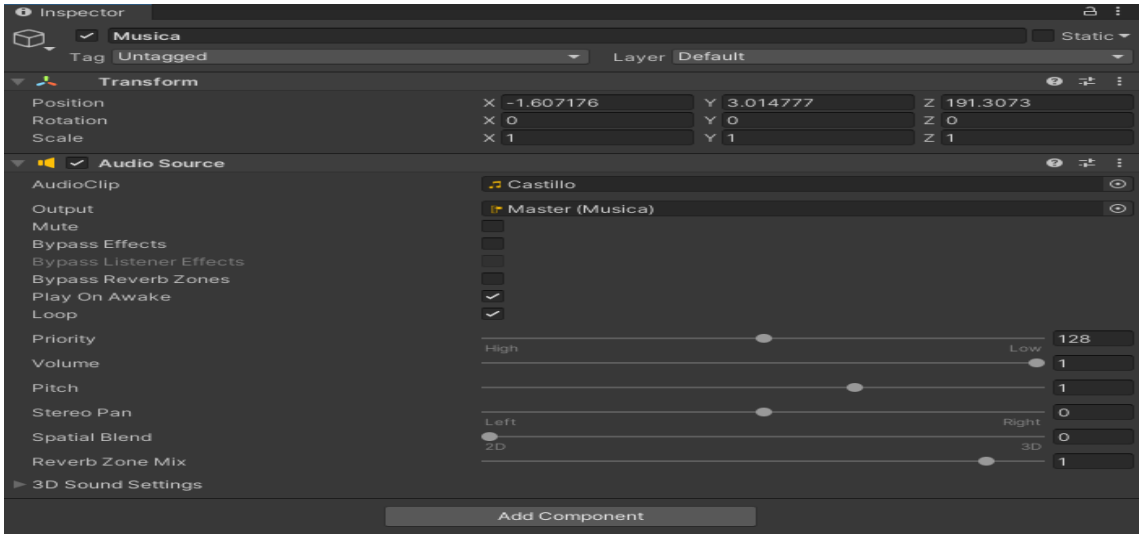

Imagen 40: Música

Por otro lado, la implementación de los efectos de sonido en el videojuego es ligeramente más complicada que la música. La principal dificultad encontrada durante la implementación de los efectos de sonido es determinar el procedimiento a utilizar para poder reproducir estos efectos de sonidos cada vez que se necesite, es decir, los efectos de sonido tienen que permanecer disponibles y preparados para poder ejecutarse en cualquier momento.

Con la finalidad de lograr conseguir esto se ha tomado la decisión de llevar a cabo la siguiente solución: Se ha creado un *GameObject* que posee adjuntado como

componente un script que guarda todos y cada uno de los efectos de sonido del videojuego. Este *GameObject* únicamente se instancia una sola vez al inicio del videojuego, aprovechando para conseguir lograr esto el patrón de diseño y programación Singleton y preservando este *GameObject* entre las distintas escenas que componen el videojuego.

Con el propósito de conseguir lograr una adecuada ejecución de los efectos de sonido del videojuego se han desarrollado tres scripts fundamentales:

- **ActivadorSonidos:** El script ActivadorSonidos es el encargado de guardar todos y cada uno de los efectos de sonido del videojuego, permitiendo que estos se encuentren disponibles y preparados para ejecutarse en todo momento. El script implementa dos métodos, que reciben como parámetro el nombre del efecto de sonido requerido, un método permite activar mientras que el otro permite desactivar el efecto de sonido con ese nombre.
- **ControladorSonido:** El script ControladorSonido se ocupa de aplicar el patrón de diseño y programación Singleton, asegurando que el *GameObject* de los efectos de sonido que tiene adjuntado este script como componente únicamente se instancie una sola vez al inicio del videojuego y permitiendo que este *GameObject* no sea destruido al cambiar de escena en el videojuego.
- **EjecutarSonido:** El script EjecutarSonido se ha desarrollado con la finalidad de poder ejecutar sonidos cada vez que un *GameObject* entra en un estado de Animación que tenga adjuntado este script como componente. Con el objetivo de lograr conseguir esto el script EjecutarSonido efectúa una llamada al script ActivadorSonidos pasando como parámetro el nombre del efecto de sonido que se encuentra configurado en este script.

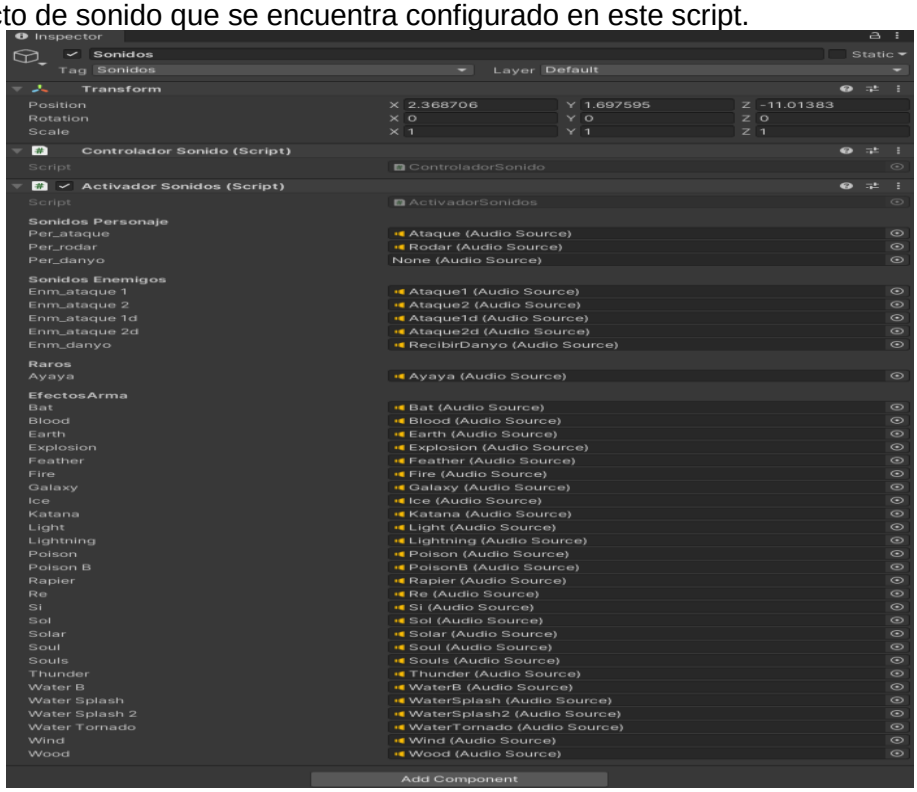

Imagen 41: Sonidos

7.9 Patrones de diseño y programación

Los patrones de diseño y programación [14] [15] son “descripciones exactas de la comunicación existente entre una serie de objetos y clases que pueden utilizarse para resolver un problema de diseño general en un contexto particular”. A continuación, se presentan en profundidad los patrones de diseño y programación que se han puesto en práctica a lo largo del desarrollo del proyecto, justificando el motivo de su utilización.

7.9.1 Singleton

El patrón de diseño y programación Singleton [16] nos permite asegurarnos que solamente dispondremos de una única instancia de una clase, proporcionando un punto de acceso general a dicha instancia.

Este patrón de diseño y programación se ha utilizado en el proyecto con el objetivo de solucionar el problema de preservar las instancias de algunos *GameObjects* entre las diferentes escenas del videojuego, sin la necesidad de instanciarlos de nuevo. Este patrón se ha implementado en diversos scripts: ControladorPersonaje, ControladorCMvcam, ControladorMainCam, ControladorCanvas, ControladorEventos, ControladorInventario y ControladorSonido. El principal motivo por el que se ha tomado la decisión de aplicar este patón de diseño y programación es conseguir garantizar que determinados *GameObjects* únicamente se instancian una sola vez preservando su estado entre las diferentes escenas que conforman el videojuego, debido a que estos *GameObjects* son esenciales para el adecuado desempeño del videojuego.

El patrón de diseño y programación Singleton se ha implementado creando en los scripts una variable estática de la propia clase llamada “Instance”. El método Awake() de los scripts que implementan este patrón de diseño y programación se encarga de verificar si la clase ya ha sido instanciada anteriormente comprobando si la variable “Instance” es *null*. En el caso de que esta clase ya se encuentre instanciada, el método Awake() destruye esta nueva instancia, asegurando que únicamente se disponga de una sola instancia de la misma. Si es la primera vez que se instancia la clase, el método Awake() se encarga de proporcionar un punto de acceso general a dicha instancia empleando el método DontDestroyOnLoad() que permite preservar el *GameObject* entre las distintas escenas del videojuego.

7.9.2 Prototype

El patrón de diseño y programación Prototype [17] se ha aplicado con el objetivo de solucionar un problema que ha aparecido durante una de las pruebas a la hora de instanciar los distintos objetos del videojuego: Siempre que se obtenían dos objetos repetidos, los valores de las estadísticas de estos objetos eran idénticos. Esto ocasionaba un conflicto con uno de los aspectos más importantes que se pensaba alcanzar en el proyecto. Este patrón de diseño y programación se ha implementado en el script ItemObject, encargado de crear el objeto que se necesita clonar e instanciar.

El patrón de diseño y programación Prototype se ha implementado creando el método getCopyItemObject(), que se encarga de crear un nuevo ItemObject de la misma clase, sin la necesidad de conocer la clase del mismo, copiando los valores de

los distintos campos del objeto que se necesita clonar, con la excepción del *array* que almacena los valores de las estadísticas del objeto, ItemBuff, el cual se volverá a generar con nuevos valores aleatorios, consiguiendo lograr producir un objeto clonado idéntico al original con valores en las estadísticas diferentes.

7.10 Estándares de mantenimiento del software

Tal y como se nos ha enseñado en la asignatura de la rama de Ingeniería del Software, MES (Mantenimiento y evolución del software), los estándares de mantenimiento del software son aspectos importantes que se deben tomar en consideración a la hora de programar y desarrollar el código de un proyecto, puesto que al respetar ciertas reglas predefinidas es considerablemente más sencillo comprender y mantener el código en un futuro.

Seguidamente, se exponen algunas de las reglas que se han respetado a la hora de desarrollar el código del proyecto:

Para empezar, el nombre de las variables es uno de los aspectos importantes a tomar en consideración, debido a que este aspecto puede afectar gravemente a la comprensión y mantenimiento del software. Un nombre de variable demasiado abreviado puede no llegar a ser lo suficientemente descriptivo provocando confusiones, sobre todo para las demás personas que no han participado en el desarrollo de esa fracción del código del proyecto, por ejemplo, en el supuesto de que se tome la decisión de escoger como nombres de variables “pi” y “pf” para hacer referencia al punto inicial y punto final respectivamente de una trampa móvil, otra persona puede llegar a encontrar dificultades a la hora de intentar comprender a qué hacen referencia estas variables, sin embargo, en el caso de que recurramos a “puntoInicial” y “puntoFinal” como nombres de las variables evitamos este problema haciendo considerablemente más sencillo que cualquier persona logre comprender a que estamos haciendo referencia en todo momento, optimizando el mantenimiento del código.

Tal y como se puede observar en el ejemplo anterior, la aplicación de las mayúsculas en el nombre de las variables o métodos es otro de los aspectos importantes que pueden llegar a influir considerablemente en el mantenimiento del software, puesto que es mucho más comprensible si el nombre de la variable es “puntoInicial” que en caso de que se llame “puntoinicial”. A lo largo del desarrollo de este proyecto se ha tomado la decisión de que los nombres de las variables empiecen por minúscula, no obstante, en el supuesto de que el nombre de la variable deba cuentan con más de una palabra se usa una mayúscula para empezar las demás palabras que pertenecen al nombre de la variable. Por otro lado, para los nombres de los métodos se ha tomado la decisión de que empiecen por mayúsculas, consiguiendo distinguirlos de las variables y siguiendo el estándar de los nombres de los demás métodos que posee el motor Unity incluidos implícitamente.

Finalmente, merece la pena destacar una de las cuestiones más controvertidas a la hora de desarrollar software, la utilización de comentarios en el propio código. Por una parte, algunas personas defienden que en caso de que el código se encuentre adecuadamente diseñado no debería de ser necesario emplear comentarios, mientras que otras personas consideran que los comentarios pueden resultar de utilidad agregando información adicional, facilitando su posterior entendimiento y optimizando el mantenimiento del software en el futuro, tomando en consideración que la persona encargada de este mantenimiento puede ser alguien que no ha tenido ninguna

relación con el desarrollo del mismo. Por nuestra parte, se ha tomado la decisión de añadir comentarios en las partes del código que no son lo suficientemente descriptivas, por ejemplo, en el script AtaqueEspecial se pueden encontrar algunos comentarios con la intención de informar la espada a la que hace referencia cada uno de los identificadores.

8 Pruebas

Se han llevado a cabo distintas pruebas con la finalidad de verificar y validar el correcto comportamiento de cada uno de los elementos desarrollados durante el proyecto. En este apartado se van a mostrar las pruebas más significativas que se han llevado a cabo, así como los defectos, fallos y errores que se han logrado encontrar y, en el caso de existir, el procedimiento que se ha empleado para solucionarlos.

8.1 Pruebas de unidad particulares

Para comenzar, en este punto se van a exponer varias de las pruebas de unidad particulares más relevantes que se han llevado a cabo:

- **Diálogos y monólogos:** En un principio, el sistema de diálogos y monólogos inició como un sistema sencillo que implementaba un sistema únicamente basado en diálogos simples. A lo largo del desarrollo ha ido evolucionando incorporando nuevas funcionalidades conforme a lo que se requería en el momento, por ejemplo, incluyendo la opción de monólogos o de soltar un objeto una vez que ha terminado el diálogo o monólogo. Por este motivo, se han llevado a cabo distintas pruebas de unidad a este sistema en varias fases diferentes del desarrollo a medida que se iban añadiendo nuevas funcionalidades al sistema. Estas pruebas de unidad se concentraron mayoritariamente en verificar y validar el apropiado funcionamiento del sistema, tomando en consideración las nuevas funcionalidades que se iban agregando y comprobando que la interfaz se actualice correctamente. Durante estas pruebas de unidad no se consiguieron encontrar defectos, fallos o errores relevantes, no obstante, mientras se llevaban a cabo la prueba general del sistema y las pruebas de aceptación se lograron identificar ciertos problemas relacionados con las interfaces y con la colisión del personaje principal con el personaje interactuable.
- **Misiones:** El sistema de misiones es un sistema relativamente simple, por esta razón no ha sido necesario realizar una gran cantidad de pruebas al respecto. Las pruebas ejecutadas a este sistema tienen como objetivo principal asegurar que al completar una misión se eliminen los *GameObjects* pertinentes de la escena y que los diálogos o monólogos asociados a la misión completada no se vuelvan a repetir. Estas pruebas se han repetido varias veces con las distintas misiones produciendo en todas ellas un resultado satisfactorio y cumpliendo completamente con todos los requisitos a la primera sin presentar ningún defecto, fallo o error significativo.
- **Manual Aventurero:** La interfaz relacionada con el Manual del Aventurero es una de las interfaces importantes del videojuego, puesto que permite la posibilidad de consultar todos los controles del videojuego en cualquier momento. Por este motivo, se llevaron a cabo distintas pruebas enfocándose en comprobar que las animaciones de la interfaz se ejecutaban debidamente, que el contenido se mostraba y actualizaba como corresponde y el adecuado comportamiento de los botones de la interfaz. Durante estas pruebas de unidad no se lograron identificar defectos, fallos o errores importantes, sin embargo, mientras se realizaban estas pruebas se consiguió determinar que el método para cerrar la interfaz no era del todo intuitivo. Por esta razón se tomó la decisión de incluir un botón para poder cerrar la interfaz. Asimismo, mientras se

llevaban a cabo las pruebas de aceptación con usuarios se consiguió localizar un problema relacionado con las interfaces del videojuego.

- **Ataques Especiales:** Los ataques especiales son una de las funcionalidades más destacables del videojuego. Por este motivo, debemos asegurarnos de que funcionan adecuadamente realizando varias pruebas minuciosas a cada uno de los efectos especiales que podemos encontrar en el videojuego. Durante estas pruebas se han conseguido localizar dos problemas:
    1. El primer problema se encuentra relacionado con los componentes Collider de los efectos especiales. Este problema consiste en que, durante determinados *frames* de la animación de los efectos especiales, el *collider* de dicho efecto no se encontraba correctamente ajustado provocando que algún enemigo recibiera daño cuando no debería o no lo recibiera cuando claramente el efecto especial había colisionado con el mismo. Para poder solucionar este error se ajustaron los *colliders* en *frames* claves de la animación y se desactivaba el *collider* en el último *frame* de animación de cada uno de los efectos especiales que existen en el videojuego.
    2. El segundo problema detectado, y uno de los más importantes, se producía a causa de que no se tenía en consideración la dirección hacia la que el personaje principal estaba mirando a la hora de utilizar un ataque especial, ocasionando que el efecto especial del ataque se instancie siempre hacia la misma dirección. Para conseguir solucionar este problema simplemente se efectuó una modificación que permite tomar en consideración la dirección hacia la que el personaje principal se encuentra mirando a la hora de utilizar un ataque especial e instanciar el efecto especial correspondiente a ese ataque.
- **Trampas:** Tal y como se ha mencionado anteriormente, las trampas son uno de los elementos del entorno más importante del videojuego. Se han llevado a cabo distintas pruebas a las diferentes trampas que se pueden encontrar en el videojuego, enfocándose principalmente en comprobar su correcto funcionamiento. Estas pruebas permitieron detectar ciertos problemas:
    1. Durante estas pruebas se ha detectado un problema con una de las trampas en concreto, la TrampaBomba, el problema consiste en que, al explotar, esta trampa no desaparecía, por el contrario, al terminar la animación la TrampaBomba continuaba existiendo, en consecuencia, cada vez que el personaje principal vuelve a pasar por la localización en la que se encontraba la TrampaBomba este recibe el daño de la misma manera que en el caso de que la explosión de la trampa hubiera colisionado con él. Para lograr solucionar este problema se ha empleado el script DestroyThis, el cual se encarga de destruir el *GameObject* de la escena una vez que ha finalizado su animación.
    2. El segundo problema identificado se encontraba en el parámetro *Is Trigger* de los componentes *Colliders* de los *Prefabs* de las trampas. Al tener este parámetro activado, con el objetivo de conseguir generar una señal cuando se detecta que la trampa ha colisionado con el personaje principal y disminuir sus puntos de vida, el personaje principal puede atravesar la trampa, lo cual provoca una incoherencia en la mayor parte de ellas, que al ser unos objetos solidos no deberían de poder ser atravesados. Para lograr solucionar este problema se ha insertado un *GameObject* con un componente *Collider* con el parámetro *Is Trigger* desactivado como hijo en el *Prefab*, con el propósito de evitar que el personaje principal pueda atravesar el *Prefab* de la trampa.

- **Sonidos:** Las últimas pruebas de unidad que se han llevado a cabo se encuentran relacionadas con los sonidos del videojuego. Inicialmente se ha realizado una prueba con el propósito de verificar y validar que los ajustes al nivel del volumen, tanto de la música como de los efectos de sonido del videojuego, configurados en la interfaz de Opciones del Menú Principal se aplican de manera apropiada, concluyendo la prueba con que, efectivamente, estos ajustes se aplican correctamente. Posteriormente se han efectuado unas pruebas con el objetivo de comprobar que cada una de las distintas músicas que podemos encontrar en el videojuego se reproducían de manera adecuada al entrar en la escena correspondiente y que al terminar el clip de audio asociado a esta música volviera a comenzar de nuevo permitiendo que el juego no se quede sin música en ningún momento. En esta segunda prueba relacionada con la música del videojuego no se ha detectado ningún problema. Finalmente, se ha llevado a cabo una tercera prueba con la intención de comprobar el funcionamiento de cada uno de los efectos de sonido del videojuego. Durante esta tercera prueba se ha conseguido identificar un problema que afecta a los ataques especiales que lanzan alguna especie de proyectil hacía adelante como efecto especial. El problema consiste en que el efecto de sonido del proyectil lanzado no se desactiva una vez que dicho proyectil colisiona con el enemigo, suelo o pared, en cambio, el clip de audio continua hasta finalizar, provocando una inconsistencia y ocasionando que varios efectos de sonido se puedan solapar arruinando completamente la experiencia de juego y la inmersión del jugador. Este error se ha solucionado deteniendo el efecto de sonido asociado al efecto especial en el script Destroy, explicado anteriormente, justo antes de destruir el *GameObject* “Efectos Arma” instanciado en la escena y que se encarga de realizar la animación correspondiente del efecto especial.

8.2 Prueba general del sistema

Para continuar, una vez que se ha logrado conseguir una versión completamente ensamblada y jugable del videojuego se ha tomado la decisión de llevar a cabo una prueba general del sistema con la finalidad de validar y verificar su desempeño conjunto, comprobar si cumple con los requisitos del proyecto y detectar posibles defectos, fallos o errores poniendo a prueba por encima todas y cada una de las funcionalidades del videojuego e intentando simular el comportamiento de un jugador cualquiera. Esta prueba se ha llevado a cabo utilizando la plataforma web de Itch.io.

Durante la prueba general del sistema se consiguieron encontrar los siguientes defectos, fallos y errores:

- **Sistema de diálogos y monólogos:** Mientras se lleva a cabo la prueba general del sistema se ha logrado detectar un problema relacionado con el sistema de diálogos y monólogos que no se ha conseguido encontrar durante las pruebas de unidad. Para poder comenzar un diálogo o monólogo se necesita que el personaje principal se encuentre en contacto con el *collider* del NPC interactuable, el problema que se ha identificado consiste en que al interactuar con NPCs que poseen varios diálogos o monólogos es necesario dejar de estar en contacto y volver a entrar en contacto con el *collider* del NPC interactuable para poder volver a interactuar con ellos. Este problema se ha intentado solucionar de distintas maneras, no obstante, todas las soluciones propuestas afectaban gravemente al apropiado comportamiento del sistema de diálogos y

monólogos. Por esta razón, al no disponer del tiempo suficiente y ser un problema que no perjudica en gran medida al adecuado funcionamiento del videojuego, se ha tomado la decisión de solucionar este problema más adelante.

- **Problema en la instanciación de los ítems:** Uno de los puntos fundamentales que se aspiran a alcanzar durante el proyecto consiste en que cada partida sea diferente y variada, por lo que los distintos ítems equipables del videojuego deben de poseer diferentes estadísticas, incluso cuando son el mismo tipo de objeto. Durante la prueba general del sistema se ha descubierto un problema en el momento de instanciar los ítems. El problema encontrado consiste en que dos ítems del mismo tipo instanciados poseen las mismas estadísticas. Para poder solucionar este problema se ha tomado la decisión de aplicar el patrón de diseño y programación Prototype explicado en detalle anteriormente, de esta manera se ha logrado conseguir que la instanciación de dos objetos del mismo tipo sean un clon copiando los valores de los distintos campos salvo por el *array* encargado de almacenar los valores de las estadísticas del ítem.
- **Problema en las plataformas:** Las plataformas son uno de los elementos del entorno más importante, puesto que permiten al personaje principal recorrer los distintos escenarios del mapa del videojuego. Durante la prueba general del sistema se han conseguido localizar dos problemas relacionados con las plataformas:
    1. El primer problema que se ha logrado encontrar consiste en que el personaje principal no consigue saltar entre plataformas adecuadamente, puesto que en el momento de intentarlo colisiona con la segunda plataforma en lugar de atravesarla. Este problema se ha solucionado modificando uno de los valores del componente *Plataform Effector 2D* con la intención cambiar el ángulo que define los lados de la plataforma, logrando de esta manera que al colisionar con los laterales de la segunda plataforma esta permita al personaje principal atravesarla.
    2. El segundo problema que se ha conseguido identificar consiste en que en algunas ocasiones el personaje principal se queda atascado en las plataformas sin poder volver a moverse nuevamente. Este problema sucede en escasas ocasiones, por lo que no se ha logrado identificar que lo provoca.

8.3 Pruebas de aceptación con usuarios

Para concluir, se han llevado a cabo un par de pruebas de aceptación con usuarios externos con la intención de recibir un *feedback* imparcial con el que conseguir identificar nuevos requisitos, tanto funcionales como no funcionales, que poder implementar más adelante, al mismo tiempo que se localizaban ciertos defectos, fallos o errores existentes que no se consiguieron encontrar durante las anteriores pruebas.

La primera prueba de aceptación con usuarios externos se ha llevado a cabo con un grupo reducido de cuatro personas. El principal objetivo de esta prueba es verificar y validar el desempeño del proyecto en la plataforma web Itch.io una vez que se encontraba completamente ensamblado. Durante esta primera prueba se ha detectado un problema relacionado con las distintas interfaces del videojuego. Este problema ha sido provocado a causa de que las interfaces del videojuego se han diseñado utilizando una resolución de pantalla de 1920x1080, por lo que, en caso de iniciar el videojuego con una resolución diferente, las interfaces del videojuego no se mostraban

ajustadas correctamente. Después de la prueba de aceptación se ha solucionado este problema modificando ciertos ajustes de los distintos *canvas* del videojuego para permitir que las distintas interfaces del videojuego se autoajusten a la resolución del equipo que está ejecutando el videojuego.

La segunda prueba de aceptación con usuarios externos se ha realizado con un grupo mucho más extenso, de catorce personas. Esta segunda prueba de aceptación tiene como objetivo tanto detectar defectos, fallos o errores que no se han podido detectar durante las pruebas anteriores como recibir ideas y sugerencias con el propósito de poder mejorar la experiencia de juego. Durante esta segunda prueba se han localizado los siguientes defectos, fallos y errores:

- **TrampaBomba:** Mientras se llevaba a cabo la prueba, los usuarios externos reportaron un problema con una de las trampas del videojuego, la TrampaBomba. Este problema consiste en que esta trampa no reducía los puntos de vida del jugador a pesar de que la explosión de la misma colisionaba con el personaje principal. Este problema se produce a causa de que en el último *frame* de la animación el *collider* de la trampa no posee activado el parámetro *Is Trigger*, provocando que el *GameObject* no pueda ser atravesado e imposibilitando la generación de la señal que permite restar los puntos de vida correspondientes al personaje principal del videojuego.
- **Paredes:** En el caso de que el personaje principal se encuentre en el aire junto a una pared, si el jugador toma la decisión de moverse en dirección hacia esta pared, el personaje principal se queda pegado en el aire, en vez de continuar cayendo hacia abajo, esto ocurre hasta que el jugador decide dejar de moverse hacia la dirección de la pared que tiene justo al lado. Con el objetivo de lograr conseguir una solución a este problema se establecieron dos detectores en los laterales del personaje principal que no permiten el desplazamiento hacia esa dirección en el caso de detectar alguna de las paredes del mapa que conforman el videojuego.
- **Vida del jugador:** Este problema consiste en que, en el caso de que el jugador tome la decisión de no abrir la interfaz de estadísticas en ningún momento previo a recibir algún daño, este daño no se actualizaba adecuadamente en los puntos de vida, provocando que el jugador se volviera inmortal. Para poder solucionar este problema se ha tomado la decisión de abrir y cerrar automáticamente la interfaz de estadísticas en la escena de carga al iniciar el videojuego, forzando de esta manera a actualizar los puntos de vida del personaje principal correctamente.
- **Vida de los jefes:** La vida de los jefes son unas interfaces especiales que se pueden encontrar únicamente en las escenas donde el personaje principal se enfrenta en combate contra los jefes del videojuego. El problema que consiguieron identificar los usuarios externos durante esta segunda prueba de aceptación es, en esencia, el mismo problema que se había detectado en la primera prueba de aceptación, es decir, un problema relacionado con la resolución de la pantalla del equipo que ejecuta el videojuego, y por lo tanto la solución que se ha aplicado es la misma que se realizó en su momento tras la primera prueba de aceptación y que, al tratarse de interfaces especiales que únicamente se encuentran presentes en dos de las escenas del videojuego, no se aplicó a las mismas.

Finalmente, merece la pena comentar las ideas y sugerencias que se han recibido como *feedback* por parte de los usuarios externos:

- **Recuperar mana tras resucitar:** Al inicio del proyecto tras resucitar el personaje principal únicamente recuperaba al completo los puntos de vida. Después de realizar la segunda prueba de aceptación se tomó la decisión de agregar la recuperación de mana con el objetivo de conseguir una experiencia de juego más justa con el jugador. Esta idea ha sido propuesta por la gran mayoría de usuarios externos que tuvieron la oportunidad de participar en las pruebas de aceptación.
- **Cofre de armas:** Debido a la amplia variedad de objetos que posee el videojuego, y que el método de obtenerlos generalmente es de manera aleatoria en algún cofre del mapa del videojuego, los usuarios externos experimentaron algunas partidas en las que no consiguieron ningún arma durante las mismas, arruinando la experiencia de juego al convertirse la misma en un desafío demasiado difícil de superar. Por este motivo sugirieron la implementación de un cofre especial que solamente pudiera contener armas. Esta sugerencia se ha tenido en cuenta y se ha tomado la decisión de implementarla más adelante.

9 Conclusiones

Tal y como se ha mencionado en el punto 1.2 Objetivos de esta memoria uno de los objetivos principales del proyecto consiste en lograr una versión jugable de un videojuego 2D de género Metroidvania. Como hemos podido comprobar a lo largo de la memoria este objetivo se ha cumplido hasta cierto punto, consiguiendo una versión del videojuego que se puede probar en el enlace facilitado en el punto 1 Introducción de esta memoria.

Esta versión del videojuego cumple completamente con todos y cada uno de los objetivos secundarios del proyecto, puesto que consta de un tutorial, varios escenarios con trampas, enemigos, dos jefes y una considerable variedad de objetos y mecánicas esenciales como, por ejemplo, los ataques especiales que ofrecen una gran diversidad y mejoran la experiencia de juego.

Por lo que se refiere al segundo objetivo principal comentado en el punto 1.2 Objetivos, este proyecto ha servido para conseguir adoptar una metodología ágil durante el desarrollo de un videojuego, llevando a cabo una buena organización y seguimiento del trabajo realizado y por realizar en todo momento, al mismo tiempo que se nos proporciona la oportunidad de aplicar dos patrones de diseño y programación y realizar distintas pruebas para verificar y validar las diferentes partes del proyecto.

Seguidamente, exponemos un enlace a la plataforma de desarrollo colaborativo de software GitHub, donde podemos encontrar y acceder al código fuente del proyecto para poder consultarlo en cualquier momento deseado:

https://github.com/diegoger222/TFG-Diego-Adrian

Para concluir, merece la pena mencionar que la realización de este proyecto ha sido posible gracias a todos los conocimientos y la experiencia adquirida en las asignaturas del Grado en Ingeniería Informática, especialmente las asignaturas de la rama de Ingeniería del Software. Seguidamente nombramos algunas de las asignaturas más importantes para la realización del proyecto y su relación con el mismo:

- **Desarrollo de Videojuegos 3D:** Durante esta asignatura se ha aprendido tanto a utilizar de manera general el motor de videojuegos multiplataforma de desarrollo Unity como el proceso de desarrollo y el ciclo de vida de un videojuego, además de servir como motivación para la realización de este proyecto.
- **Proceso de software y Proyecto de ingeniería de software:** Estas asignaturas nos han enseñado en que consiste seguir una metodología ágil, sus ventajas e inconvenientes y la manera adecuada de aplicar esta metodología durante el proceso de desarrollo de los proyectos de software.
- **Análisis y especificación de requisitos:** Los conocimientos adquiridos en esta asignatura han sido de utilidad a la hora de identificar, analizar y especificar los diferentes requisitos, tanto funcionales como no funcionales, del proyecto que podemos encontrar detallados en el punto 5 Análisis de requisitos del proyecto de esta memoria. Este análisis también ha sido de ayuda durante la organización y planificación del proyecto obteniendo los diferentes casos de uso y tareas a realizar aplicando una metodología ágil basada en una combinación entre los sistemas Scrum y Kanban.

- **Diseño de software:** En esta asignatura se nos han presentado los distintos patrones de diseño y programación, sus ventajas y desventajas, así como la importancia de saber identificar cuando y como aplicar cada uno de ellos.
- **Mantenimiento y evolución de software:** Durante esta asignatura se nos ha explicado la importancia de la utilización de un estándar de mantenimiento de software, así como los diferentes tipos de pruebas que se pueden llevar a cabo para lograr conseguir un desarrollo estructurado del código que permita una mejor comprensión y entendimiento del mismo.

Por último, destacar la importancia de las asignaturas Introducción a la informática y a la programación y Programación del primer año del Grado en Ingeniería Informática que han presentado los conceptos fundamentales de la programación y que han servido como base para la realización del proyecto.

10 Bibliografía y referencias

[1] GÓMEZ BLANES, Rafael. *El libro negro del programador*. B&t Books, 2017. ISBN: 9781496153357

[2] Manifiesto ágil | Disponibe en: https://agilemanifesto.org/principles.html

[3] Metroidvania | fandom. Disponible en: https://castlevania.fandom.com/es/wiki/Metroidvania

[4] Blasphemous | Disponible en: https://www.blasphemous2game.com

[5] Hollow Knight | Disponible en: https://www.hollowknight.com

[6] Castlevania: Symphony of the Night | fandom. Disponible en: https://castlevania.fandom.com/es/wiki/Castlevania:_Symphony_of_the_Night

[7] Unity Manual | Disponible en: https://docs.unity3d.com/es/530/Manual/UnityManual.html

[8] Trello | Disponible en: https://trello.com/es/tour

[9] Git | Disponible en: https://git-scm.com/

[10] Visual Studio | Disponible en: https://visualstudio.microsoft.com/es/#vs-section

[11] Itch.io | Disponible en: https://itch.io/docs/general/about

[12] GameObjects Unity | Disponible en: https://docs.unity3d.com/es/530/Manual/GameObjects.html

[13] Orden de Ejecución de Funciones de Evento Unity | Disponible en: https://docs.unity3d.com/es/530/Manual/ExecutionOrder.html

[14] FREEMAN, Eric; FREEMAN, Elisabeth; SIERRA, Kathy; BATES, Bert. *Head First Design Patterns*. O'REILLY, 2004. ISBN: 9780596007126

[15] GAMMA, Erich; HELM, Richard; JOHNSON, Ralph; VLISSIDES, John. *Design Patterns Elements of Reusable Object-Oriented Software*. Addison Wesley, 1994. ISBN: 0201633612

[16] Singleton | Refactoring Guru. Disponible en: https://refactoring.guru/es/design-patterns/singleton

[17] Prototype | Refactoring Guru. Disponible en: https://refactoring.guru/es/design-patterns/prototype

ANEXO

OBJETIVOS DE DESARROLLO SOSTENIBLE

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenibles	Alto	Medio	Bajo	No Procede
ODS 1. **Fin de la pobreza.**				X
ODS 2. **Hambre cero.**				X
ODS 3. **Salud y bienestar.**				X
ODS 4. **Educación de calidad.**		X		
ODS 5. **Igualdad de género.**				X
ODS 6. **Agua limpia y saneamiento.**				X
ODS 7. **Energía asequible y no contaminante.**				X
ODS 8. **Trabajo decente y crecimiento económico.**				X
ODS 9. **Industria, innovación e infraestructuras.**				X
ODS 10. **Reducción de las desigualdades**.				X
ODS 11. **Ciudades y comunidades sostenibles**.				X
ODS 12. **Producción y consumo responsables.**				X
ODS 13. **Acción por el clima.**				X
ODS 14. **Vida submarina.**				X
ODS 15. **Vida de ecosistemas terrestres.**				X
ODS 16. **Paz, justicia e instituciones sólidas.**				X
ODS 17. **Alianzas para lograr objetivos.**	X			

Reflexión sobre la relación del TFG con los ODS

Tal y como se puede observar en la tabla anterior el proyecto desarrollado para este TFG, sobre todo por ser un videojuego 2D que posee como motivación fundamental entretener y divertir al jugador, no posee ningún grado de relación con la mayor parte de los Objetivos de Desarrollo Sostenible (ODS). Tras reflexionar con respecto a los distintos ODS se ha determinado que el proyecto desarrollado para este TFG se encuentra relacionado principalmente con los siguientes ODS:

- **ODS 4. Educación de calidad:** El proyecto se encuentra relacionado con este ODS en un grado medio al poder ser aprovechado en cierta medida a modo de modelo a seguir a la hora de mostrar una posible manera de desarrollar un videojuego 2D llevando a cabo una metodología ágil, basada en una combinación entre los sistemas Scrum y Kanban, durante el desarrollo del proyecto, al mismo tiempo que se presentan las distintas maneras de proceder, los posibles defectos, fallos y errores a tomar en consideración desde el inicio del proyecto, con la intención de no tener que lidiar con estos posteriormente, y presentando ejemplos, aplicados al desarrollo de videojuegos 2D, de los distintos patrones de diseño y programación que se han utilizado durante el desarrollo del proyecto con el objetivo de solucionar algunos de estos problemas. Con el propósito de conseguir lograr hacer esto posible este proyecto se encuentra publicado de manera pública en GitHub y de forma privada en la plataforma web Itch.io protegido por una contraseña publicada en este TFG, posibilitando de este modo que cualquier persona interesada pueda acceder y consultar el código desarrollado del proyecto.
- **ODS 17. Alianzas para lograr objetivos:** Tal y como se ha mencionado previamente, este proyecto se ha llevado a cabo de manera cooperativa conjuntamente con Diego Ruiz Muñoz con la finalidad de lograr conseguir un objetivo común. Adicionalmente, tal y como se ha señalado en la reflexión sobre la relación del TFG con el ODS 4 Educación de calidad, el proyecto se puede encontrar publicado de manera pública en GitHub, permitiendo de esta manera que cualquier persona pueda acceder al código del proyecto con la intención de consultar el proyecto, extraer ideas o adquirir conocimientos aplicados en el propio proyecto, sirviendo de ayuda para otros proyectos y logrando alcanzar objetivos comunes impulsando el establecimiento y la formación de alianzas con el objetivo de resolver distintos problemas de manera indirecta. Debido a estos dos motivos se considera que el proyecto desarrollado para este TFG guarda un alto grado de relación con el ODS 17 Alianzas para lograr objetivos.

Para finalizar, tal y como se puede contemplar, el trabajo realizado durante el proyecto desarrollado para este TFG se encuentra relacionado escasamente con los ODS, encontrándose estas relaciones asociadas con el intercambio y la transmisión de información y la utilización del código como ejemplos aplicados al desarrollo de un proyecto enfocado a un videojuego 2D.