UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Diseño y desarrollo de un videojuego 2D en Unity usando metodología ágil

Trabajo Fin de Máster

Máster Universitario en Ingeniería Informática

AUTOR/A: Serra Almenar, Ignacio José

Tutor/a: Abad Cerdá, Francisco José

CURSO ACADÉMICO: 2022/2023

Resumen

Este trabajo de fin de máster describe el proceso de desarrollo de un videojuego de plataformas en 2D utilizando la metodología ágil.

El juego se enmarca dentro del género *metroidvania*, donde el jugador debe explorar un mundo interconectado lleno de peligros y desafíos, superando obstáculos y derrotando enemigos para avanzar en la historia. El arte del juego está hecho a mano, de tipo digital, y se plasma en la pantalla con una cámara 2D de tipo "desplazamiento lateral".

El primer nivel comienza con el jugador atrapado en una prisión, y su objetivo es escapar de ella a lo largo de tres niveles, cada uno con su propio diseño de trampas y jefes de nivel.

A medida que el jugador avanza, adquirirá nuevas habilidades que le permitirán acceder a nuevas áreas y vencer a enemigos más poderosos.

El juego se ha desarrollado utilizando el motor Unity, y el enfoque ágil ha permitido trabajar de manera más eficiente y responder a los cambios y retroalimentación de los usuarios con mayor rapidez. El trabajo detalla las distintas etapas del proceso de desarrollo, desde la planificación inicial hasta la iteración constante y el resultado final.

También se muestran las herramientas utilizadas a lo largo del proyecto, así como los desafíos y las soluciones encontradas durante el proceso de desarrollo.

Palabras clave: metodología ágil, videojuego, plataformas, metroidvania, Unity, Scrum, C#,2D

Abstract

This master's thesis describes the development process of a 2D platformer video game using agile methodology.

The game falls within the metroidvania genre, where the player must explore an interconnected world filled with dangers and challenges, overcoming obstacles and defeating enemies to progress in the story. The game's art is handcrafted, of the digital type, and is displayed on screen with a 2D "side-scrolling" camera.

The first level begins with the player trapped in a prison, and their objective is to escape from it over the course of three levels, each with its unique trap design and level bosses.

As the player progresses, they will acquire new abilities allowing them to access new areas and defeat more powerful enemies.

The game was developed using the Unity engine, and the agile approach allowed for more efficient work and a quicker response to user feedback and changes. The work details the different stages of the development process, from initial planning to constant iteration and the final outcome.

Tools used throughout the project are also showcased, as well as the challenges and solutions found during the development process.

Keywords: agile methodology, videogame, platforms, metroidvania, Unity, Scrum, C#,2D

Tabla de contenidos

1. Introducción

Este trabajo de fin de máster describe el proceso de desarrollo de un videojuego de plataformas en 2D utilizando la metodología ágil.

Desde sus inicios, los videojuegos se han destacado como herramientas multifacéticas, desempeñando un papel más allá de la simple diversión. Con el paso de los años, su influencia en la cultura, la economía y la tecnología es innegable, posicionándose como un pilar en la sociedad contemporánea a la vez que se compagina con otras disciplinas como la música, el cine, el arte y la literatura.

El trabajo tiene como adjunto un anexo del documento de diseño, el cual describe el videojuego desarrollado.

1.1 Motivación

La industria de los videojuegos ha vivido una transformación monumental desde sus primeras apariciones. En este contexto, surge la motivación para aportar a esta evolución y continuar el legado. La posibilidad de fusionar narrativas, arte y técnica ofrece un campo vasto para la creatividad y el desarrollo técnico.

Dentro de este panorama, los videojuegos 2D en particular, evocan una sensación nostálgica, recordando épocas donde la simplicidad gráfica se combinaba con historias y mecánicas profundas. Se identifica, por tanto, una oportunidad de revisitar este formato y aportar una propuesta fresca y contemporánea.

El sector del videojuego se puede considerar una de las principales herramientas de entretenimiento de los últimos tiempos, por lo que ha desembocado en un desarrollo más complejo, haciéndolo un reto, basándose en grandes proyectos con diferentes departamentos trabajando y cooperando entre ellos.

Es por ello, por lo que la principal motivación de este proyecto se basa en el desafío del desarrollo de un videojuego que obligue a una planificación minuciosa haciendo uso de metodologías ágiles, tales como Scrum [4], a la vez que se mejoran y evolucionan las habilidades de desarrollo.

1.2 Objetivos

La intención principal de este trabajo ha sido concebir y desarrollar un videojuego 2D de plataformas en Unity. Pero más allá de este desarrollo, se ha buscado la planificación de un proyecto desde su inicio hasta el final. A través de *sprints* semanales, se ha organizado un producto mínimo viable que podía ser entregado y probado cada semana, cada vez con una serie de mecánicas nuevas, enemigos y escenarios.

A nivel de desarrollo, el juego dispone de distintos niveles que el jugador podrá explorar. Se ha intentado que cada nivel posea una identidad propia, con obstáculos y plataformas que requieran del jugador habilidad y estrategia.

Cada nivel, además de enemigos básicos, tiene un jefe final. Estos adversarios no son meros obstáculos, se les ha dotado de mecánicas diferentes en los que cada enfrentamiento lleva al jugador a tomar tácticas diferentes para poder derrotarlos.

Se ha procurado que cada elemento artístico cuente una parte de la historia, desde los fondos hasta los *sprites* de los personajes. El juego tiene un estilo *cartoon*, con una combinación de colores que sean agradables a la vista del jugador.

En cuanto al apartado musical, se ha seleccionado para que cada nivel tenga su propio estilo, así como cada jefe final, que tienen un tema diferente cada uno.

En resumen, los objetivos a desarrollar en este trabajo son los siguientes:

- Planificación del proyecto usando la metodología ágil Scrum, organizando entregas por sprint semanal.
- Diseño y desarrollo de mecánicas del personaje principal.
- Animación de diferentes armas del personaje principal.
- Diseño de las mecánicas de los enemigos.
- Desarrollo de los enemigos.
- Diseño y desarrollo de los niveles.
- Diseño y desarrollo de la interfaz del juego.
- Diseño e implementación del sistema de sonido.
- Diseño y desarrollo de las diferentes pruebas de habilidad y trampas

1.3 Impacto

Aunque la naturaleza indie del proyecto puede sugerir un alcance limitado, se reconoce que en la actualidad muchos títulos de este estilo han alcanzado renombre y éxito. Por lo tanto, aunque las expectativas deben ser realistas, se aspira a que el juego resuene en una comunidad apasionada que valora retos y desafíos.

Mediante publicidad en redes sociales y la publicación en plataformas dedicadas a videojuegos, se busca aumentar el impacto y abarcar un nicho de jugadores más amplio.

1.4 Metodología

El desarrollo ágil, y en particular Scrum, ha proporcionado un marco estructurado pero flexible para el desarrollo del juego. Esta elección se basa en la naturaleza cambiante y evolutiva del desarrollo de videojuegos, donde la retroalimentación y la iteración son clave.

En cuanto a la herramienta utilizada para la organización del proyecto, "Hack and Plan", ha sido indispensable para todo el proceso de desarrollo. Aporta una serie de características específicas para la gestión del proyecto de un videojuego, facilitando la organización de sprints [17] por tareas, etiquetas y seguimiento, así como la facilidad de exportar un informe de desarrollo.

En cuanto al enfoque de desarrollo iterativo ha permitido una adaptación constante. A través de sprints se ha ido construyendo el videojuego, permitiendo ajustes de la dificultad, resolución de errores y *feedback* de la comunidad, enfocando siempre lograr un PMV que sirva como base para las iteraciones siguientes.

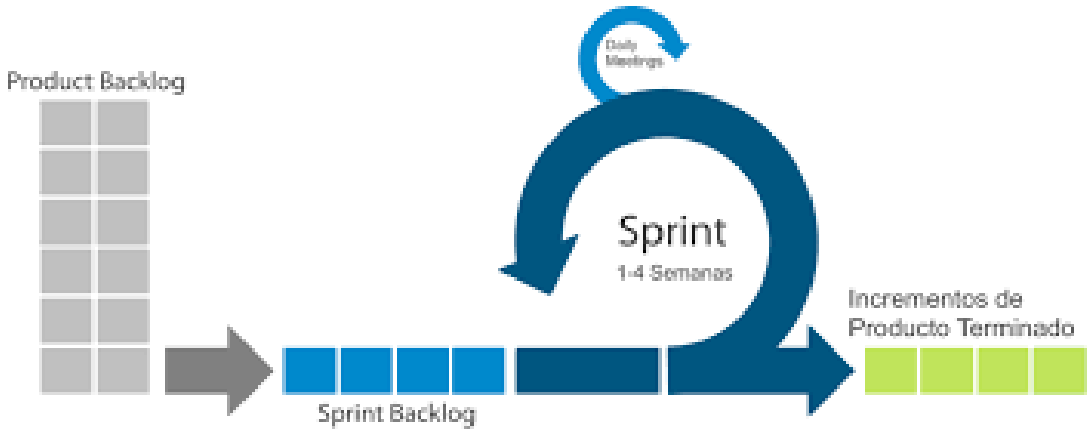

Figura 1. Detalle de sprints

1.5 Estructura de la memoria

La memoria se va a estructurar en diferentes partes y va a documentar todo el proceso del desarrollo.

- **Introducción:** Presenta de manera general la motivación, el objetivo y la metodología aplicada en el proyecto.

- **Estado del arte:** Se basa en una comparación de soluciones similares al proyecto, lo que ayuda a identificar los requisitos de una forma más fácil.

- **Análisis de requisitos:** Se define lo que se espera del producto final y como responderá a las necesidades y expectativas del usuario.

- **Diseño:** se explica la tecnología utilizada en el proyecto, así como las diferentes clases, controladores y animaciones que dan vida al proyecto.

- **Implementación:** en este apartado se habla de las librerías externas utilizadas, assets y *sprites* que se han utilizado para la implementación de este desarrollo.

- **Resultados:** el apartado consta de los diferentes resultados que ha dado el proyecto en redes sociales como itch.io, encuestas en Google Forms y familiares que lo han probado.

- **Conclusiones:** reflexión sobre los resultados y conclusiones que se han obtenido, comparándolos con los objetivos que se tenían prefijados.

2. Estado del arte

2.1 Crítica al estado del arte

En el amplio y diverso mundo de los videojuegos, existen géneros que han perdurado y evolucionado a lo largo del tiempo, adaptándose a las tecnologías emergentes y las preferencias cambiantes de los jugadores. Uno de estos géneros destacados es el de los videojuegos de plataformas 2D, en el que los jugadores navegan por niveles bidimensionales, superando obstáculos, enemigos y desafíos a medida que avanzan. Dentro de este ámbito, se encuentra el subgénero conocido como *metroidvania*, una fusión de las mecánicas y estilos de dos titanes del gaming: "Metroid" y "Castlevania". Estos juegos se caracterizan por ofrecer mundos interconectados, que el jugador puede explorar libremente. A medida que se avanza en la historia, el jugador adquiere nuevas habilidades que le permiten acceder a áreas previamente inaccesibles.

Más allá de la mecánica de juego, también es importante considerar la naturaleza de la experiencia del jugador. En este caso, el juego se ha diseñado como una experiencia "offline" para un solo jugador. En una era donde los juegos multijugador en línea dominan gran parte del mercado, los juegos para un solo jugador siguen siendo cruciales ya que ofrecen narrativas más profundas y experiencias más inmersivas que son exclusivamente del jugador, sin interrupciones de otros jugadores.

Para contextualizar adecuadamente el trabajo desarrollado en este proyecto, es esencial examinar soluciones similares existentes en el mercado. A continuación, se presenta una tabla que destaca juegos tanto contemporáneos como clásicos del género *metroidvania*, evaluando sus características, mecánicas y enfoques para proporcionar una visión amplia de cómo este proyecto se sitúa en el panorama actual de videojuegos.

2.2 Análisis del mercado

En este apartado se van a analizar productos similares al planteado en el proyecto y que han servido de inspiración en la realización de este, tanto visualmente como en mecánicas.

La idea es obtener diferentes puntos en común entre los diferentes videojuegos del mismo género para poder implementarlas en el proyecto de manera adecuada.

2.3 Super Metroid

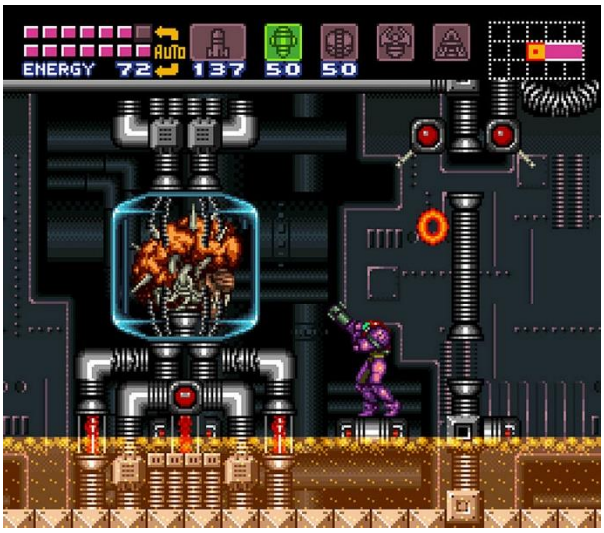

Figura 2. Super Metroid

Lanzado en 1994 para la Super Nintendo, este juego es uno de los pilares fundamentales del género *metroidvania*.

Super Metroid es un juego del género acción y aventuras 2D, el movimiento de la cámara es de tipo desplazamiento lateral. El argumento tiene lugar en el planeta ficticio Zebes. Es un mundo abierto con áreas conectadas por puertas y ascensores. El jugador controla a la protagonista Samus Aran, la cual está buscando un Metroid robado por Ridley, líder de los piratas espaciales. El jugador es capaz de correr, saltar, agacharse y disparar el arma en un total de ocho direcciones.

A lo largo de los diferentes niveles, el jugador puede comprar potenciadores que mejoran las características de Samus, así como la obtención de habilidades especiales que permiten acceder a lugares anteriormente inaccesibles.

2.4 Castlevania: Symphony of the Night

Figura 3. Castlevania

Esta entrega de 1997 para PlayStation revolucionó la serie *Castlevania* al introducir elementos de RPG y un castillo interconectado para explorar.

La jugabilidad en *Symphony of the Night* es la típica de cualquier juego de plataformas 2D de la época. El protagonista se llama Alucard, un vampiro. Sus movimientos son saltar y atacar con diferentes armas. La exploración del castillo otorga experiencia para aprender nuevas habilidades. El lugar en el que se desarrolla el juego, es un mundo abierto muy poco común para la época.

Los elementos RPG permiten a Alucard mejorar sus atributos.

2.5 Ori and the Blind Forest

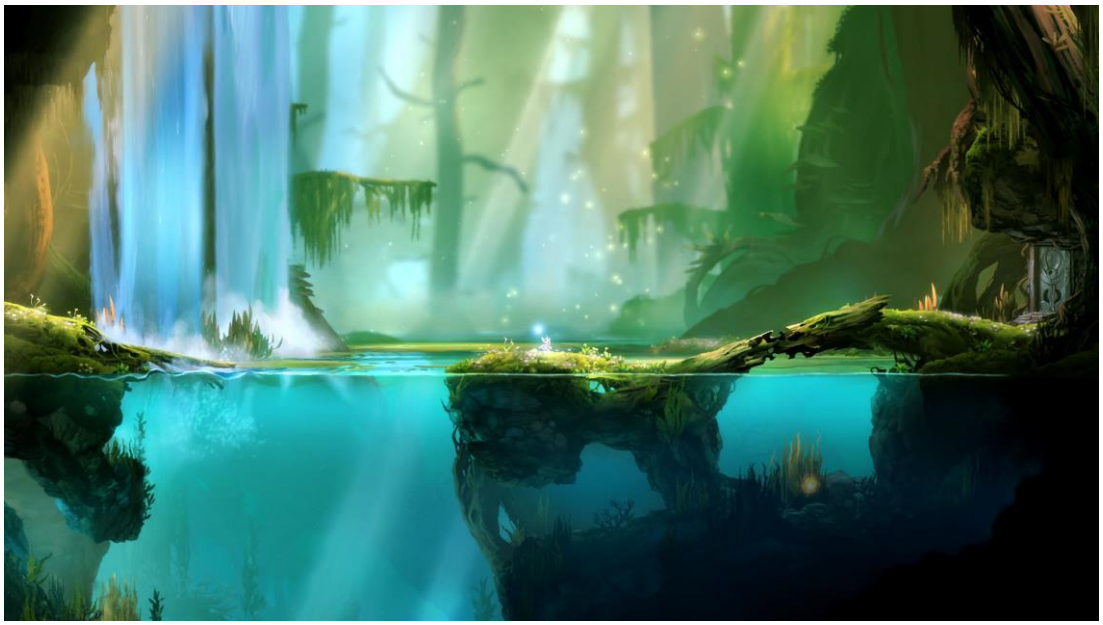

Figura 4. Ori and the blind forest

Lanzado en 2015, destaca por su belleza visual y sonora. Lo que lo hace único es su emotiva narrativa, que se entrelaza con desafíos de plataformas precisos y mecánicas innovadoras.

Ori and the Blind Forest es un juego de plataformas en el que el jugador controla a Ori, un espíritu guardían blanco, y a Sein.

Ori tiene las habilidades de correr, saltar y trepar, mientras que Sein puede disparar llamas de a los enemigos.

El jugador deberá interactuar, resolver puzles y combatir diferentes enemigos para restaurar el bosque.

La recolección de Ori de los diferentes fragmentos de vida y energía le otorgará nuevas habilidades. El mundo se presenta al jugador al estilo *Metroidvania*, es decir, mientras se obtienen habilidades nuevas, se desbloquean pasajes que anteriormente estaban bloqueados.

2.6 Hollow Knight

Figura 5. Hollow Knight

Publicado en 2017, este juego independiente destacó rápidamente por su profunda mecánica, estética atractiva y mundo vasto y misterioso.

Hollow Knight nos pone en la piel del Caballero en la búsqueda de los secretos que habitan en Hallownest.

El juego tiene como enfoque principal la exploración, los saltos de plataformas y el. El jugador explorará un mundo interconectado usando diferentes movimientos como esquivar, salto doble y rebote entre enemigos.

El combate está centrado en el aguijón del Caballero, el cual es un arma muy parecida a una espada y que puede golpear en cuatro direcciones. Este aguijón puede ser mejorado para hacer más daño a los enemigos.

Al final de cada zona, un jefe espera al jugador, el cual deberá derrotar para continuar la aventura. Al acabar con ellos, éstos otorgan al jugador una habilidad especial que permitirá al jugador explorar nuevas zonas anteriormente.

2.7 Blasphemous

Figura 6. Blasphemous ingame

Publicado en 2019, este título se diferencia de otros juegos *metroidvania* por su intensa temática religiosa y su estética inspirada en el arte y la cultura española.

El juego empieza con una maldición llamada El Milagro, la cual ha caído sobre la tierra de Cvstodia. El protagonista es el único superviviente de la hermandad y se encuentra atrapado en un ciclo de penitencia donde muere y resucita constantemente, por lo que debe encontrar el origen de la maldición para salvarse.

A lo largo del juego, el protagonista adquiere habilidades especiales que permiten luchar de maneras diferentes contra los enemigos.

Como se puede observar, las características más importantes del videojuego es el género de plataformas, más concretamente *metroidvania,* el tipo (2D), modo de un jugador, dificultad desafiante.

Para analizar a la competencia, se ha desglosado la siguiente tabla comparativa que presenta los siguientes puntos:

- **Videojuego plataformas 2D:** como se ha comentado, el videojuego se ha realizado en 2D con *sprites* disponibles en internet, con un estilo de gráficos de alta resolución.
- **Aprendizaje rápido:** adaptación de la habilidad del jugador según avanza en el juego.
- **Mecánicas complejas:** se trata de las diferentes habilidades que puede realizar el jugador principal.
- **Dificultad**: dificultad alta que resulte en un reto para el jugador.
- **Gráficos de alta resolución**: densidad de píxeles superior al *pixel art*
- **Registro de puntuación**: pese a ser un juego offline, permite el registro de puntuaciones en una tabla puntuaciones.

2.8 Comparativa

Nombre	Plataformas	Aprendizaje rápido	Mecánicas	Dificultad	Alta resolución	Registro
Dandelion	✓	✓	✓	✓	✓	✓
Super Metroid	✓	✓	✗	✗	✗	✓
Castlevania	✓	✗	✓	✓	✗	✗
Hollow Knight	✓	✓	✓	✓	✓	✗
Blasphemous	✓	✓	✗	✓	✗	✗

A modo de resumen de la table, se puede comprobar que el género del videojuego realizado es de plataformas y que presenta una dificultad adaptativa según el nivel en el que nos encontramos.

La dificultad en los primeros niveles es baja para conseguir captar la atención del jugador, sin embargo, cuanto más avanza, las mecánicas se vuelven más complejas para conseguir que el jugador sienta una progresión escalonada. De esta forma, la dificultad tiene que subir para contrarrestar la obtención de estas nuevas mecánicas más poderosas.

El arte está dibujado con una alta resolución de píxeles, lo que produce un efecto *cartoon* en el videojuego.

Por último, el videojuego presenta la posibilidad de registrar la puntuación por nivel para medir la habilidad del jugador contra otros jugadores.

2.9 Motores de videojuegos

En este apartado se va a analizar los motores de videojuegos más populares para comparar las distintas características y comprender cuáles han motivos los motivos para elegir Unity en la realización de este proyecto.

2.9.1 Unreal Engine

Figura 7. Unreal Engine logo

Origen: Desarrollado por Epic Games, Unreal Engine ha sido uno de los pilares de la industria desde su introducción en 1998.

Características: Unreal es conocido por su avanzado sistema de gráficos, que ha sido utilizado en títulos triple A con alta demanda visual. Su lenguaje de programación principal es el C++, pero también dispone de un sistema visual llamado "Blueprint" que permite a los desarrolladores crear lógica de juego sin necesidad de escribir código.

Ventajas: Su alto grado de personalización y rendimiento. Tiene una amplia comunidad y numerosos recursos didácticos disponibles.

Desventajas: La curva de aprendizaje puede ser más pronunciada que otros motores, y para proyectos comerciales, hay una estructura de regalías asociada con su uso.

2.9.2 Unity

Figura 8. Unity logo

Origen: Desarrollado por Unity Technologies [1], este motor se lanzó en 2005 y rápidamente ganó popularidad, especialmente entre desarrolladores independientes.

Características: Unity utiliza C# como su lenguaje principal y es conocido por su flexibilidad y versatilidad, permitiendo a los desarrolladores crear juegos tanto en 2D como en 3D.

Ventajas: Es altamente accesible para principiantes y tiene una amplia base de usuarios, lo que se traduce en una gran comunidad y una vasta cantidad de recursos educativos. Su modelo de licencia es amigable para desarrolladores independientes y pequeños estudios.

Desventajas: Aunque es altamente versátil, algunos desarrolladores opinan que para proyectos muy específicos o con demandas técnicas extremas, otros motores podrían ser más adecuados.

2.9.3 Godot

Figura 9. Godot logo

Origen: Es un motor de código abierto que ha ganado tracción en los últimos años gracias a su modelo libre y su activa comunidad.

Características: Godot tiene su propio lenguaje de script llamado GDScript, aunque también soporta C# entre otros. Ofrece herramientas tanto para 2D como para 3D, y tiene un sistema único de "nodos" que los desarrolladores utilizan para construir escenas y mecánicas de juego.

Ventajas: Siendo de código abierto, es completamente gratuito sin importar el tamaño o el éxito del proyecto. Es ligero, y su diseño modular permite a los desarrolladores usar solo las características que necesitan.

Desventajas: Al ser más nuevo que Unity o Unreal, puede carecer de algunas características avanzadas o tener una menor cantidad de recursos educativos en comparación.

Una vez conocidos los puntos fuertes y débiles de cada motor, la elección ha sido Unity por los siguientes motivos:

1. **Versatilidad 2D**: Dado que el proyecto es un videojuego 2D de plataformas, Unity es particularmente fuerte en este ámbito, ofreciendo herramientas específicas para el desarrollo 2D.

2. **Lenguaje de Programación**: C# es un lenguaje intuitivo y robusto. Su estructura orientada a objetos es ideal para el desarrollo de videojuegos y es más accesible que el C++ de *Unreal*.

3. **Licencia**: La licencia gratuita de Unity permite desarrollar y publicar el juego sin costes.

4. **Recursos**: La amplia base de usuarios de Unity significa que hay una enorme cantidad de tutoriales, foros, y recursos disponibles. De esta manera se pueden resolver multitud de problemas, conflictos específicos y aprender de manera fluida el entorno de trabajo de Unity.

5. **Integración**: Unity ofrece una amplia gama de *plugins* y herramientas integradas, tanto por defecto como en su tienda, que han sido cruciales para el desarrollo del juego.

6. **Exportación multiplataforma**: El juego tiene previsto publicarse para todo tipo de plataformas, y en este caso Unity pone muy fácil la exportación a los diferentes sistemas.

En resumen, la elección de Unity se ha basado en su combinación de versatilidad, accesibilidad, y una rica base de recursos, haciendo que sea una elección ideal para el proyecto en cuestión.

3. Análisis de requisitos

Como en cualquier desarrollo de software, el análisis de requisitos es una fase crucial. Es en esta etapa donde se define y comprende qué es lo que se espera del producto final, y cómo este responderá a las necesidades y expectativas del usuario.

El análisis de requisitos no solo se centra en la funcionalidad y características del videojuego, sino también en aspectos externos que pueden influir en su desarrollo y distribución, como el marco legal y ético, la protección de datos y la propiedad intelectual. Una definición adecuada de estos requisitos permitirá un desarrollo más fluido, evitará retrabajos innecesarios y asegurará que el juego cumpla con todas las normativas y expectativas relevantes.

3.1 Análisis del marco legal y ético

Todo videojuego, independientemente de su escala o propósito, se encuentra sujeto a un conjunto de leyes y regulaciones que varían según el país o región. Estas leyes pueden abordar desde la clasificación por edades hasta la representación de ciertos temas o contenidos. Además, más allá de lo legal, es esencial considerar las implicaciones éticas de las decisiones tomadas durante el desarrollo, asegurando que el producto sea respetuoso y considerado con la diversidad y sensibilidad de su público objetivo.

3.1.1 Análisis de protección de datos

Si bien este juego es offline, hace uso y guarda los datos de preferencia de las opciones, como por ejemplo el volumen del audio y la puntuación obtenida en cada nivel junto al nombre que quiera registrar. Aunque no son datos sensibles, en un futuro se podría cifrar para que los datos no pudieran ser modificados de forma directa.

3.1.2 Propiedad intelectual

El aspecto visual y sonoro del videojuego, es decir, tiles, assets, sprites y pistas de audio han sido obtenidos a través de la tienda de Unity, por lo que se han obtenido de manera legal.

La licencia que usan estos assets es *Standard Unity Asset Store EULA* [2], la cual permite el uso de los recursos a nivel comercial.

3.1.3 Clasificación PEGI

La clasificación PEGI pensada para el juego es PEGI 12. Es una clasificación que presenta un lenguaje más adulto y escenas de violencia gráfica en contextos de fantasía.

3.2 Identificación de actores

En esta sección se van a identificar los actores que interactúan con el producto.

Identificador	**ACT1**
Nombre	Jugador
Descripción	Usuario principal de la aplicación

3.3 Diagrama de contexto

El diagrama de contexto es una herramienta utilizada en el análisis de sistemas para representar de manera gráfica cómo un sistema interactúa con su entorno. Este diagrama es el nivel más alto en una serie de diagramas que describen el sistema en cuestión y sus interacciones. Es una representación simplificada que muestra el sistema como un único proceso, sin entrar en los detalles de su funcionamiento interno.

En este caso, el diagrama de contexto sería el siguiente:

- **El Sistema**: El videojuego desarrollado en Unity.
- **Entidades Externas**:
  - **Jugador**: La persona que juega e interactúa con el juego.
  - **Servidor**: Si hay actualizaciones, contenido descargable o alguna interacción online.
  - **Base de datos**: Si el juego guarda puntuaciones, progreso, configuraciones u otros datos relevantes.

- **Flujos de Información**:
  - Entre el juego y el jugador: acciones de juego, decisiones, entradas (teclado, ratón, controlador).
  - Entre el juego y el servidor: actualizaciones, puntuaciones.
  - Entre el juego y la base de datos: guardar/cargar progreso, configuraciones.

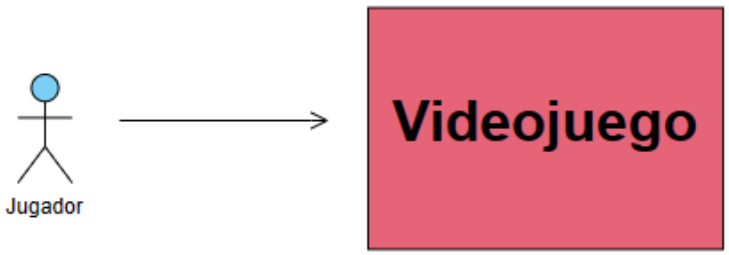

Figura 10. Diagrama de contexto

3.4 Diagrama de casos de uso

3.4.1 Menú principal

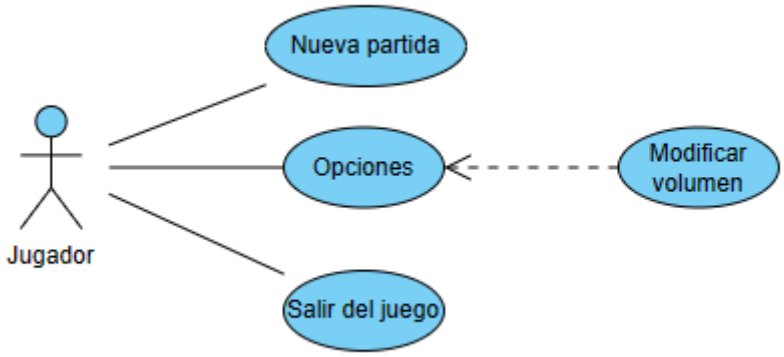

Figura 11. Diagrama de casos de uso del menú

En esta imagen podemos ver como el jugador puede interactuar con el menú principal y qué opciones tiene.

3.4.2 Juego

Figura 12. Diagrama de casos de uso del juego

En la Figura 12 se observan todas las interacciones que el jugador puede realizar.

3.5 Definición de requisitos

En este apartado se va a identificar, documentar y analizar las necesidades y especificaciones que el videojuego debe cumplir para satisfacer las expectativas del jugador y garantizar una experiencia de juego coherente y fluida. Estos requisitos se van a dividir en dos categorías: requisitos funcionales y no funcionales.

3.5.1 Requisitos funcionales

Identificador	RF1
Nombre	Movimiento
Descripción	El jugador podrá moverse de manera horizontal por el terreno. No habrá aceleración ni deceleración, la velocidad será constante.
Prioridad	Alta

Identificador	RF2
Nombre	Salto
Descripción	El jugador podrá saltar. La altura del salto será x2 a la del personaje. Si el jugador suelta la tecla antes, la altura del salto será menor.
Prioridad	Alta

Identificador	RF3
Nombre	Doble salto
Descripción	El jugador podrá realizar otro salto adicional. Se reiniciará cuando el jugador toque el suelo.
Prioridad	Alta

Identificador	RF4
Nombre	Esquivar
Descripción	El jugador podrá realizar un movimiento de esquiva. Obtiene 0.5 segundos de invulnerabilidad. Permite traspasar enemigos.
Prioridad	Media

Identificador	RF5
Nombre	Saltar hacia abajo
Descripción	Según la plataforma, el jugador podrá saltar hacia abajo.
Prioridad	Media

Identificador	RF6
Nombre	Ataque rápido con cualquier arma
Descripción	El jugador tendrá acceso a un ataque rápido con cualquier tipo de arma.
Prioridad	Alta

Identificador	RF7
Nombre	Ataque fuerte con cualquier arma
Descripción	El jugador tendrá acceso a un ataque fuerte con cualquier tipo de arma
Prioridad	Media

Identificador	RF8
Nombre	Recoger objeto
Descripción	El jugador podrá recoger cualquier tipo de objeto (monedas, pociones, armas) que estén por el terreno o que un enemigo haya dejado
Prioridad	Media

Identificador	RF9
Nombre	Escalar
Descripción	El jugador podrá utilizar escaleras para subir o bajar por ellas
Prioridad	Media

Identificador	RF10
Nombre	Mirar arriba/abajo
Descripción	El jugador puede desplazar la cámara hacia arriba o hacia abajo
Prioridad	Baja

Identificador	RF11
Nombre	Empezar partida
Descripción	El jugador puede empezar partida
Prioridad	Alta

Identificador	RF12
Nombre	Opciones
Descripción	El jugador tendrá al menú de opciones desde cualquier escena
Prioridad	Media

Identificador	RF13
Nombre	Pausar juego
Descripción	El jugador puede pausar el juego. Todos los elementos de la escena se quedarán paralizados
Prioridad	Media

Identificador	RF14
Nombre	Salir del juego
Descripción	El jugador podrá salir del juego en cualquier momento
Prioridad	Media

Identificador	RF15
Nombre	Mecánicas de enemigos básicos
Descripción	Los enemigos se moverán y atacarán al jugador cuando estén en su rango de visión
Prioridad	Alta

Identificador	RF16
Nombre	Mecánicas de jefes finales
Descripción	Los jefes finales tendrán diferentes fases según su vida y realizarán diferentes ataques al jugador
Prioridad	Media

Identificador	RF17
Nombre	Diseño del sistema de sonido
Descripción	Existirá un sistema de sonido que reproduzca sonido ambiental y efectos sonoros para procurar *feedback* al jugador
Prioridad	Media

Identificador	RF18
Nombre	Eventos
Descripción	El jugador podrá activar diferentes objetos que realizarán una serie de eventos especiales, tales como abrir puertas o desactivar trampas.
Prioridad	Baja

Identificador	RF19
Nombre	Plataformas especiales
Descripción	El jugador podrá utilizar plataformas que realizarán un determinado movimiento para desplazarse
Prioridad	Media

Identificador	RF20
Nombre	Modificar volumen
Descripción	El jugador podrá modificar el volumen del juego.
Prioridad	Baja

3.5.2 Requisitos no funcionales

Identificador	RNF1
Nombre	Sistema de interfaz
Descripción	La interfaz se adaptará a cualquier dispositivo en el que se ejecute el juego
Prioridad	Alta

Identificador	RNF2
Nombre	Disponibilidad
Descripción	El juego debe garantizar al usuario que el 99% de las veces iniciará el juego de manera correcta
Prioridad	Alta

Identificador	RNF3
Nombre	Duración de carga
Descripción	La carga entre escenarios no deberá superar los 4000 milisegundos
Prioridad	Alta

Identificador	RNF4
Nombre	Rendimiento
Descripción	El juego deberá ofrecer como mínimo una tasa de 60 fotogramas por segundo
Prioridad	Alta

Identificador	RNF5
Nombre	Capacidad
Descripción	El juego no deberá de ocupar más de 1 GB en el disco duro
Prioridad	Alta

Identificador	RNF6
Nombre	Mantenibilidad
Descripción	El código debe de ser mantenible de manera fácil
Prioridad	Alta

Identificador	RNF7
Nombre	Tolerancia a fallos
Descripción	El sistema no deberá salir de manera abrupta mientras se juega y sin consentimiento del jugador
Prioridad	Alta

Identificador	RNF7
Nombre	Inicio del juego
Descripción	El inicio del juego desde que se ejecuta no debe de tardar más de 10 segundos
Prioridad	Alta

3.6 Solución propuesta

Según los requisitos obtenidos, la solución que se ha llevado a cabo para nuestro proyecto es la siguiente:

3.6.1 Sistema de niveles

Para mantener al jugador interesado y desafiado, se ha decidido dividir el juego en tres niveles diferentes, cada uno con una temática distinta. Esta variación temática no solo da la sensación de progreso, sino que también evita la monotonía al introducir nuevos elementos visuales y desafíos en cada etapa. Además, al estructurar el juego de esta manera, se facilita la posibilidad de expansión futura, añadiendo más niveles si fuera necesario.

3.6.2 Enemigos

La diversidad y escalabilidad de los enemigos son fundamentales para mantener el interés y el desafío a lo largo del juego. Por lo tanto, en cada nivel se han incluido diferentes tipos de enemigos, cada uno con un nivel de dificultad superior al anterior. Además, los enemigos se han tematizado acorde a la escena en la que aparecen, reforzando la coherencia estética y narrativa del juego.

3.6.3 Menús

Los elementos de los menús se han modularizado para facilitar su reutilización en diferentes escenas del juego. Esto no solo optimiza el tiempo de desarrollo, sino que también asegura la consistencia visual y funcional de los menús en todo el juego.

3.6.4 Historia

La narrativa del juego se diseñó antes del desarrollo para asegurar una cohesión total entre la historia, la estética y las mecánicas del juego. Esto implicó la creación de un guión detallado que incluyera los eventos principales, los diálogos de los personajes y las transiciones entre escenas.

3.6.5 Eventos

Los eventos y las trampas se han distribuido cuidadosamente a lo largo de cada nivel para evitar la repetición y mantener al jugador alerta. Esto incluye la variación en los tipos de trampas y eventos, así como su frecuencia y ubicación.

3.6.6 Sistema de sonido

El sonido es un elemento crucial para la inmersión y la jugabilidad. Por tanto, se ha desarrollado un sistema de sonido que incluye efectos ambientales, sonidos de personajes y música de fondo. Excepto los sonidos que dependen de la distancia del jugador, como el ruido de un río o un enemigo cercano, todos los demás sonidos están gestionados por un único *prefab* que persiste a través de todas las escenas. Esto asegura una transición suave entre escenas y evita la duplicación de objetos de sonido.

3.7 Plan de trabajo

Para el diseño, desarrollo e implementación del proyecto se ha utilizado la metodología ágil, SCRUM, apoyada con herramientas de trabajo como *HacknPlan* y *Github*.

3.7.1 Metodología Ágil: SCRUM

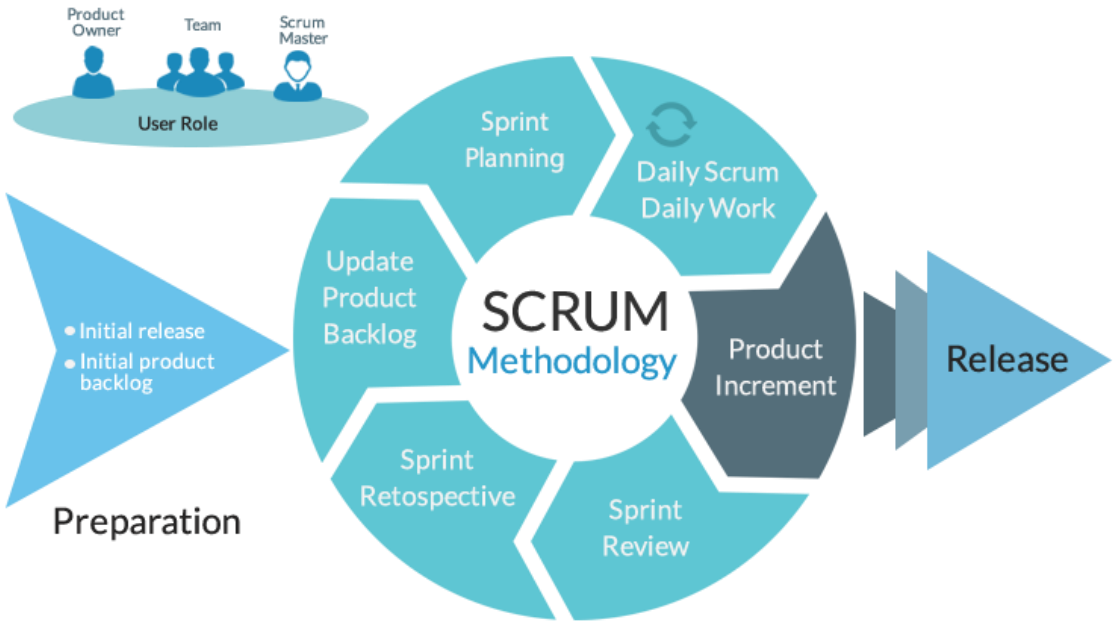

Figura 13. Metodología ágil SCRUM

Para el diseño, desarrollo e implementación del videojuego, se ha optado por usar la metodología ágil SCRUM, que es un marco de trabajo para el desarrollo y mantenimiento de productos complejos, generalmente software. Esta metodología se basa en el manejo iterativo e incremental del proyecto, lo que permite una mayor flexibilidad y capacidad de respuesta a los cambios. En SCRUM, el trabajo se divide en 'Sprints', que son períodos de tiempo determinados (generalmente de dos semanas) en los que se deben completar ciertas tareas.

Roles en SCRUM:

- **Product Owner**: Es el responsable de definir las características del producto y decidir sobre las funcionalidades a incluir en cada Sprint. En este caso, el producto owner seríamos mi tutor y yo
- **Scrum Master**: Facilita el trabajo del equipo, asegurándose de que se sigan las reglas de SCRUM y se eliminen los obstáculos que impidan el avance del proyecto. En el contexto del proyecto, al ser solo una persona, he sido yo.
- **Equipo de Desarrollo**: Son los encargados de realizar el trabajo de diseño, implementación y pruebas del producto. En este caso, yo.

Eventos en SCRUM:

- **Sprint**: Periodo de tiempo durante el cual se realiza el trabajo necesario para entregar una versión incrementada del producto.
- **Reunión** de Planificación de Sprint: Al principio de cada Sprint, se realiza una reunión para decidir qué tareas se van a realizar durante ese Sprint.
- **Revisión** de Sprint: Al final de cada Sprint, se realiza una revisión del trabajo realizado y se presenta al *Product Owner*.
- **Retrospectiva** de Sprint: Tras la revisión, se realiza una reunión para analizar cómo ha ido el Sprint y cómo se puede mejorar en el futuro.

En el contexto del proyecto las reuniones, revisiones y retrospectiva las realizaba junto con mi tutor al finalizar el sprint, donde discutíamos los puntos que se habían realizado, lo que había dado problemas y lo que se iba a realizar en el sprint siguiente.

3.7.2 HacknPlan

Figura 14. HacknPlan logo

Es una herramienta de planificación de proyectos especialmente diseñada para desarrollo de videojuegos [10]. Permite desglosar el proyecto en tareas más pequeñas, asignar esas tareas a miembros del equipo y realizar un seguimiento del progreso. En este proyecto, HacknPlan se utilizó para organizar y planificar las tareas de cada Sprint, asignar responsabilidades y hacer un seguimiento del progreso.

3.7.3 Github

Figura 15. Github logo

Es una plataforma de desarrollo colaborativo que utiliza el sistema de control de versiones Git [9]. Permite a los equipos de desarrollo colaborar en el código fuente, gestionar cambios, realizar revisiones de código y mantener un historial de versiones.

En este proyecto, Github se utilizó para almacenar el código fuente del juego, gestionar los cambios en el código y facilitar la colaboración entre los miembros del equipo.

3.7.4 Estimación del proyecto

La estimación del proyecto se ha realizado a partir de una serie de fases:

- Mecánicas de personaje
- Mecánicas de enemigos
- Diseño de niveles
- Elementos interactuables
- Menús
- Selección de *sprites*
- Selección de sonidos

Aproximadamente la estimación que se ha realizado es de 6 meses de desarrollo.

Fase	Estimación
Mecánicas de personaje	**30h**
Mecánicas de enemigos	**50h**
Diseño de niveles	**120h**
Elementos interactuables	**15h**
Menús	**5h**
Selección de *sprites*	**5h**
Selección de sonidos	**5h**

4. Diseño

4.1 Arquitectura software

La arquitectura utilizada ha sido de tres capas:

- **Presentación**: clases pertenecientes a las interfaces del usuario, cuya finalidad es la de interactuar con el jugador.
- **Lógica**: clases que procesan los datos de la aplicación.
- **Persistencia**: clases que pertenecen a los datos de la aplicación que tienen que guardarse. Tales como la puntuación y la vida del jugador.

4.2 Capa de presentación

En el proyecto, la capa de presentación incluye varios componentes clave:

- **Menú principal**: es la primera pantalla con la que el usuario va a interactuar. Incluye tres botones principales: "Empezar Juego", "Opciones" y "Salir". El botón "Empezar Juego" inicia el juego propiamente dicho, "Opciones" lleva a un menú donde el jugador puede ajustar ciertos parámetros como el volumen del audio, y "Salir" cierra la aplicación.
- **Opciones**: en este menú, el jugador puede ajustar ciertos parámetros del juego. Por ahora, este menú incluye un 'slider' para ajustar el volumen del audio.
- **Interfaz del juego**: esta es la interfaz que el jugador ve mientras juega. Incluye la vida del personaje, representada gráficamente, el avatar del personaje y el contador de monedas que el jugador ha recogido.

4.2.1 Menú principal

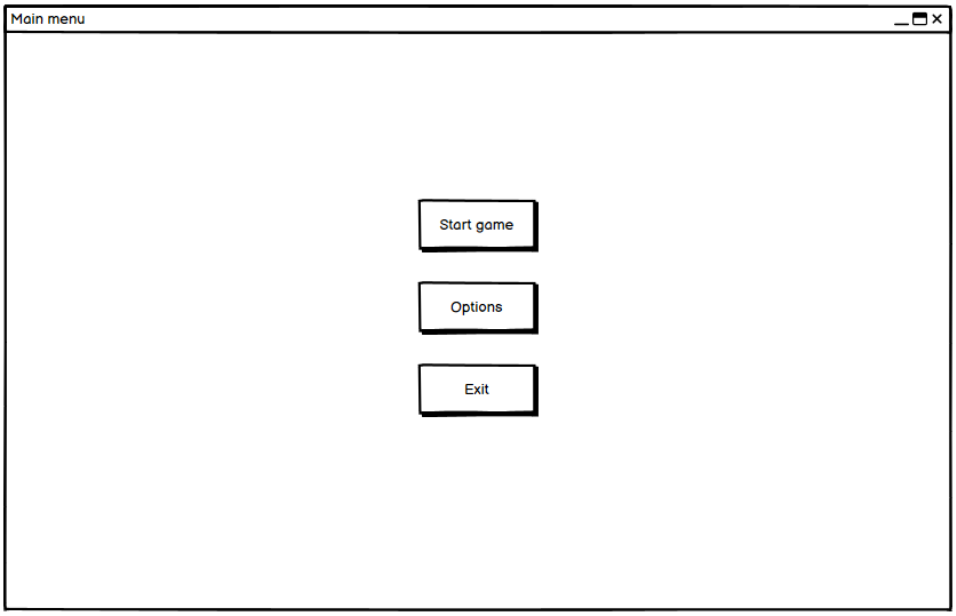

Figura 16. Mockup menú principal

4.2.2 Opciones

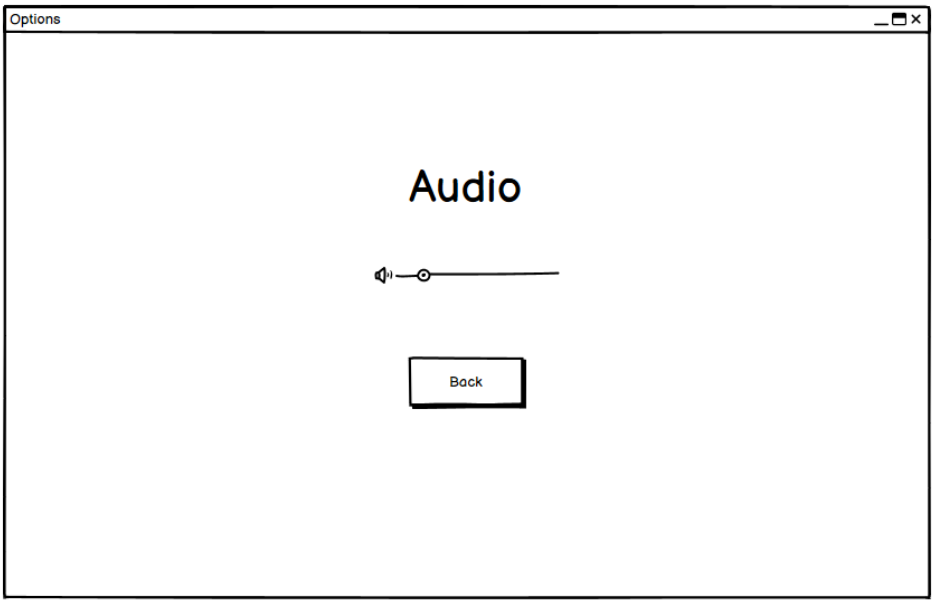

Figura 17. Mockup opciones

4.2.3 Juego

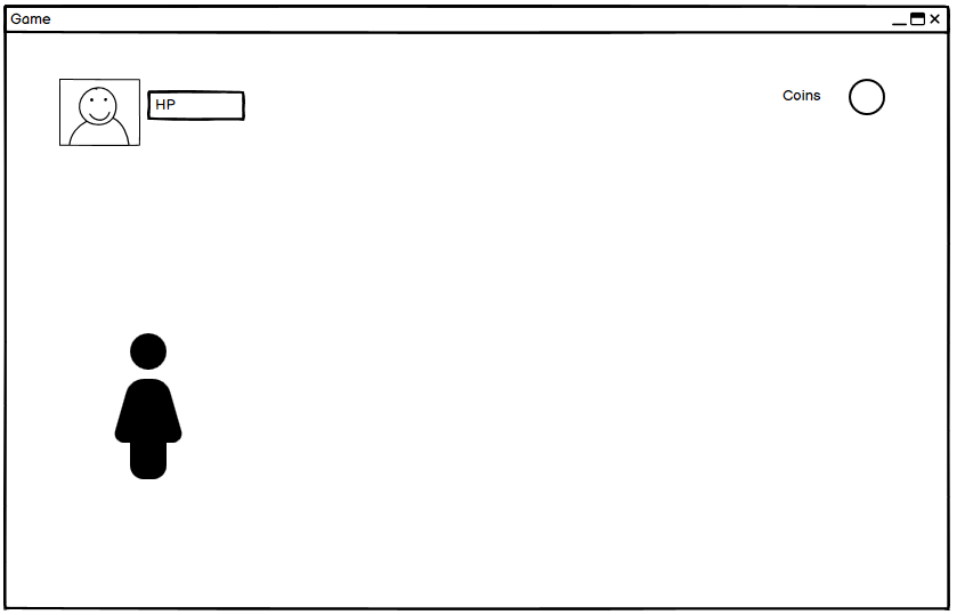

Figura 18. Mockup UI juego

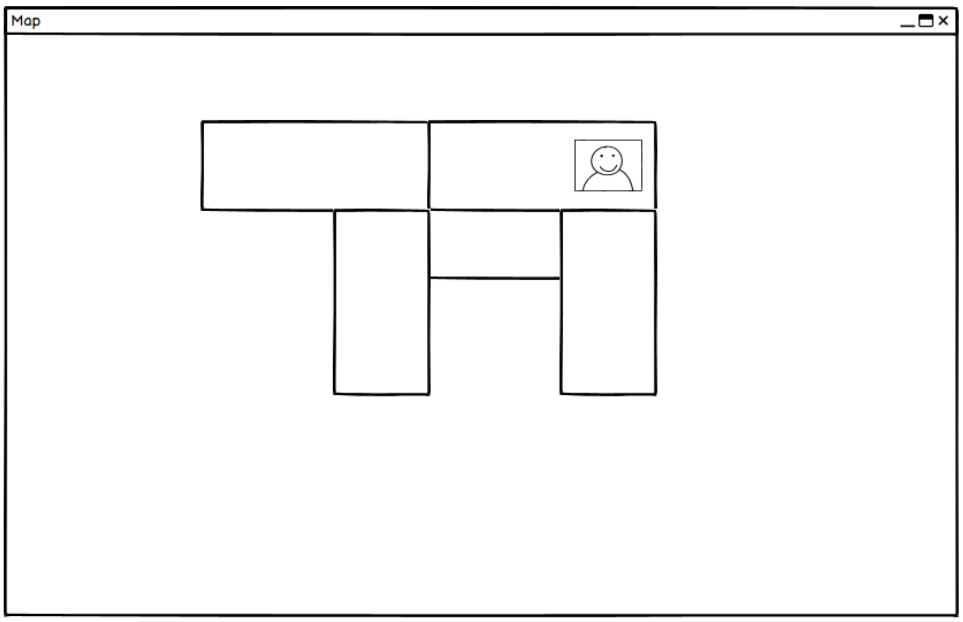

Figura 19. Mockup mapa

4.3 Capa de lógica de negocio

En esta capa se ha cubierto la manera de controlar el flujo del juego, gestionar el estado actual, y coordinar las interacciones entre el jugador, los enemigos, y el entorno. Esto incluye la gestión de la física del juego, como las colisiones y los movimientos de los personajes.

- **Gestión de personajes**: Este componente se encarga de controlar los personajes del juego, tanto el personaje principal controlado por el jugador, como los enemigos. Esto incluye la gestión de la vida, su posición, sus movimientos, y sus interacciones con otros elementos del juego, como las trampas u objetos.

- **Gestión de niveles**: Este componente se encarga de gestionar los distintos niveles del juego. Esto incluye la carga de los niveles, la gestión de los objetos y enemigos presentes en cada nivel, y la transición entre los diferentes niveles.

- **Gestión de interacciones**: Este componente se encarga de gestionar las interacciones entre los diferentes elementos del juego. Esto incluye las colisiones entre los personajes y los objetos, las interacciones entre el jugador y los enemigos, y las interacciones entre el jugador y el entorno, como recoger objetos o activar trampas.

- **Gestión de puntuación**: Este componente se encarga de gestionar la puntuación del jugador. Incluye la recogida de monedas, la puntuación obtenida al derrotar enemigos, y la puntuación final al completar un nivel.

Cada uno de estos componentes es fundamental para el funcionamiento del juego y ha sido diseñado e implementado cuidadosamente para asegurarse de que el juego sea divertido, desafiante y atractivo para el jugador.

4.4 Capa de persistencia

En cualquier videojuego, es esencial poder guardar y cargar datos, como el progreso del jugador, las configuraciones, las puntuaciones, entre otros. En este proyecto, la capa de persistencia se encarga de manejar el almacenamiento y recuperación de estos datos.

Se han utilizado *ScriptableObjects* para mantener la información de los objetos. El *ScriptableObject* es una clase proporcionada por Unity que permite almacenar grandes cantidades de datos compartidos independientes de la instancia del script, por lo que son persistentes en sus estados. Esto es útil para almacenar datos como las estadísticas de los enemigos, las configuraciones del juego, y otros datos que necesitan ser compartidos entre diferentes objetos y escenas.

Los *ScriptableObjects* son especialmente útiles para este proyecto porque permiten una fácil edición y manejo de datos en el editor de Unity, y también ayudan a reducir la carga de la memoria durante el tiempo de ejecución, ya que se pueden compartir entre diferentes objetos.

Además de los *ScriptableObjects*, también se utilizan archivos de guardado para almacenar el progreso del jugador, como los niveles completados, la puntuación obtenida, y los objetos recogidos. Estos datos se almacenan en el disco en un formato serializado y se cargan en el juego cuando es necesario.

La gestión de la persistencia de datos es crucial para la experiencia del jugador, ya que permite al jugador retomar el juego desde donde lo dejó, y también ayuda a mantener el juego justo y equitativo, al asegurarse de que todas las acciones y logros del jugador se registren y reconozcan adecuadamente.

4.5 Planificación del proyecto

Como se ha comentado en apartados anteriores, la gestión del proyecto se ha llevado a cabo mediante un desarrollo iterativo a partir de *sprints*, utilizando la metodología ágil SCRUM.

Esta metodología ha permitido tener un producto mínimo viable o MVP desde la primera iteración. A partir de cada iteración o sprint, se ha ido añadiendo nueva funcionalidad, nuevos niveles y corrección de errores que han aparecido en anteriores *sprints*.

La estimación inicial que se realizó fue de aproximadamente 6 meses, sin embargo, el primer sprint empieza el 15/06/2022 y el último sprint acaba el 08/04/2023.

El desarrollo se alarga 10 meses debido a nueva funcionalidad añadida de la que se había previsto y errores que han ido surgiendo.

Se han realizado 10 *sprints* a lo largo del proceso de desarrollo y en la tabla siguiente se resumen con su duración, las historias de usuario y las tareas que ha tenido cada uno de ellos.

Sprint	Duración	Historias de usuario	Tareas
1	**2 semanas**	**4**	**7**
2	**2 semanas**	**4**	**18**
3	**2 semanas**	**4**	**19**
4	**2 semanas**	**2**	**13**
5	**2 semanas**	**5**	**9**
6	**2 semanas**	**9**	**25**
7	**2 semanas**	**9**	**16**
8	**2 semanas**	**7**	**11**
9	**2 semanas**	**10**	**17**
10	**2 semanas**	**5**	**8**

Gracias al plugin de *HacknPlan*, se ha podido obtener una vista Gantt de cada sprint.

A continuación, se van a mostrar que tareas se han realizado en cada uno de estos *sprints*, el tiempo que ha tomado para cada una de ellas y una breve explicación de lo que ha consistido el sprint.

Sprint 1

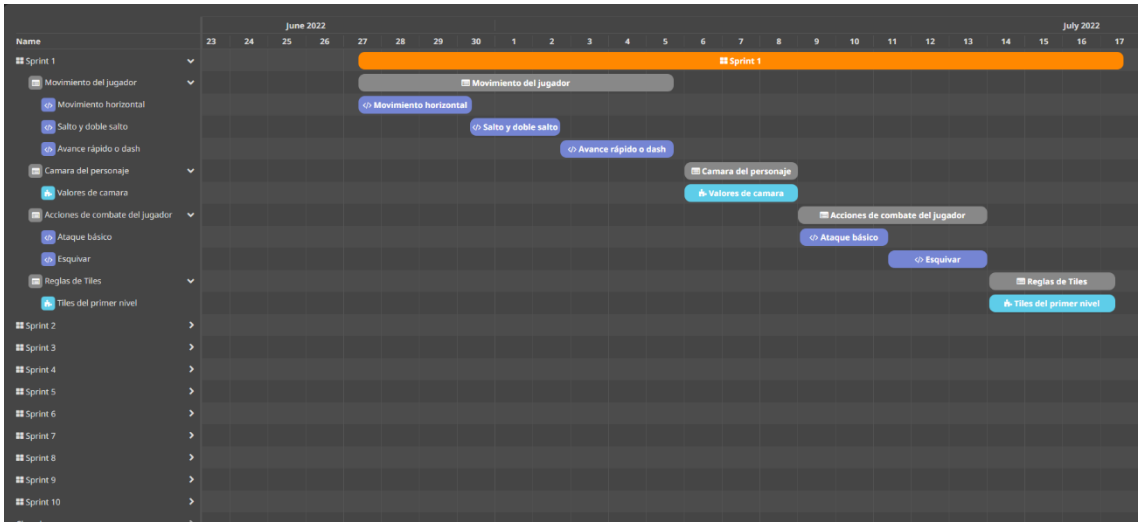

Figura 20. Planificación Sprint 1

El primer sprint se centró en desarrollar las mecánicas fundamentales del personaje principal, como el movimiento horizontal, el salto y doble salto, el avance rápido o *dash*, y las acciones de combate. Además, se trabajó en la implementación y ajuste de la cámara del personaje, se establecieron las reglas para los tiles del entorno y se inició el diseño de los enemigos del primer nivel. Este sprint sentó las bases para la interacción del personaje principal con el entorno y los enemigos.

Sprint 2

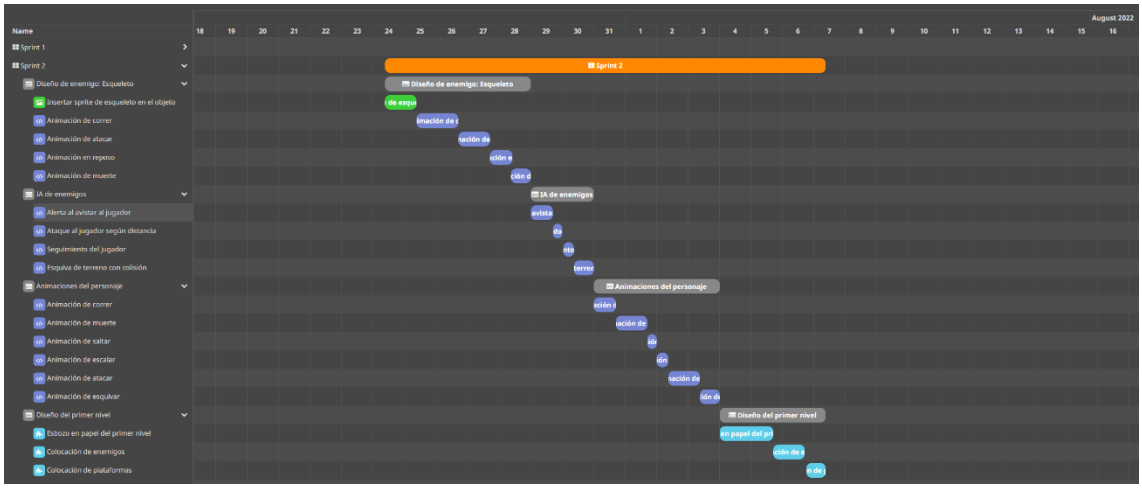

Figura 21. Planificación Sprint 2

La segunda iteración consistió en el diseño de un enemigo específico, el esqueleto. Esto implicó la inserción del *sprite* del esqueleto en el objeto, así como el desarrollo de diversas animaciones para el enemigo, como correr, atacar, estar en reposo y morir. Además, se implementaron diferentes IA (Inteligencia Artificial) de enemigos, incluida la alerta al avistar al jugador, ataque al jugador según la distancia, seguimiento del jugador y esquiva de terreno con colisión. También se trabajó en las animaciones del personaje principal, incluidas las de correr, morir, saltar, escalar, atacar y esquivar. Por último, se inició el diseño del primer nivel del juego, incluyendo un esbozo en papel, la colocación de enemigos y la colocación de plataformas. Este sprint se centró en el diseño de enemigos y la mejora de las animaciones tanto del personaje principal como de los enemigos.

Sprint 3

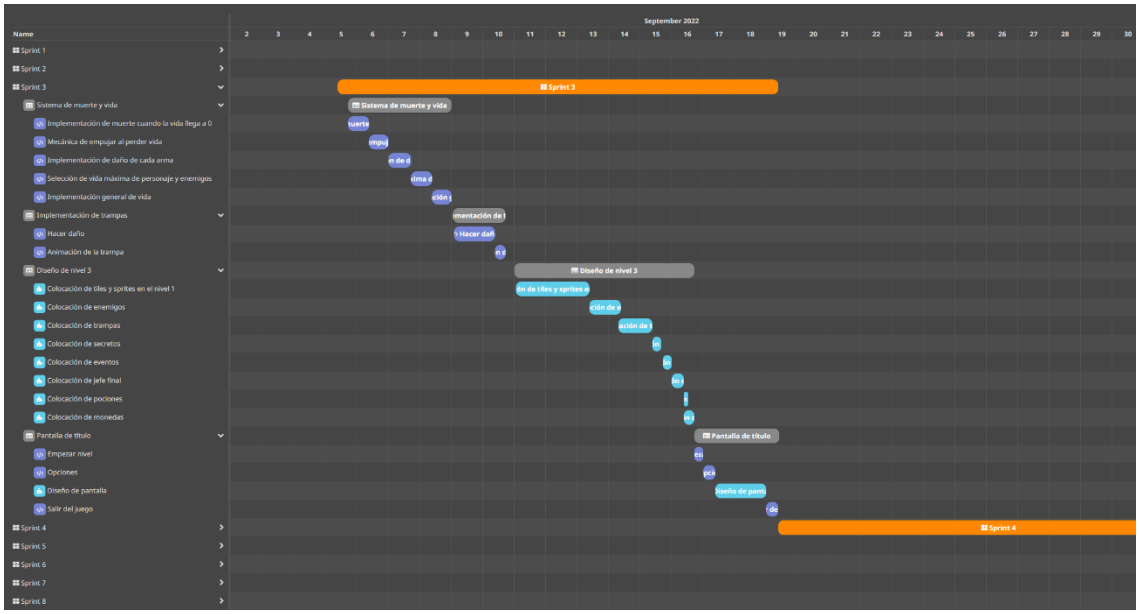

Figura 22. Planificación Sprint 3

En el tercer sprint se implementó el sistema de vida y muerte, la cual incluye la mecánica de morir cuando la vida llega a 0, la mecánica de empujar al personaje cuando pierde vida y la implementación del daño causado por cada arma. También se realizó la implementación general de la vida, estableciendo la vida máxima del personaje principal y de los enemigos. Además, se trabajó en la implementación de trampas, incluyendo su capacidad para infligir daño y las animaciones asociadas. Se continuó con el diseño de niveles, específicamente en la colocación de tiles, *sprites* y enemigos en el primer nivel y el diseño del tercer nivel. Por último, se desarrolló la pantalla de título, incluidas las opciones para comenzar el nivel, acceder a otras opciones y salir del juego. Este sprint se centró en la implementación de sistemas vitales del juego, como el sistema de vida y muerte, y el diseño de niveles.

Sprint 4

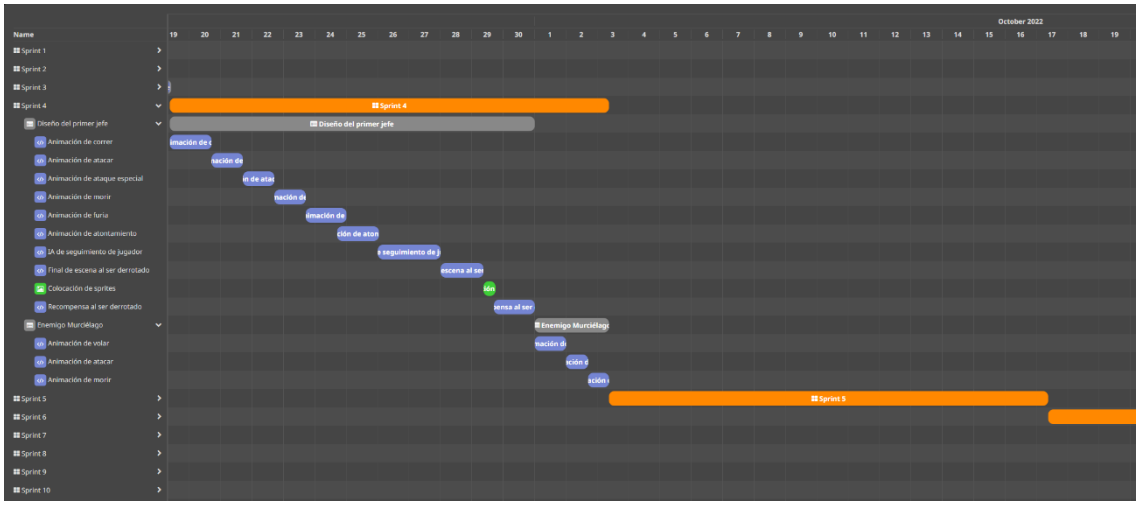

Figura 23. Planificación Sprint 4

El enfoque principal del cuarto sprint fue el diseño del primer jefe del juego. Esto implicó la creación de una IA para el seguimiento del jugador por parte del jefe, la implementación de una escena final cuando el jefe es derrotado, la colocación de sprites y la implementación de una recompensa al derrotar al jefe. Además, se inició el desarrollo de otro enemigo, el murciélago. Este sprint se centró en el diseño y desarrollo de elementos cruciales para el progreso del jugador en el juego.

Sprint 5

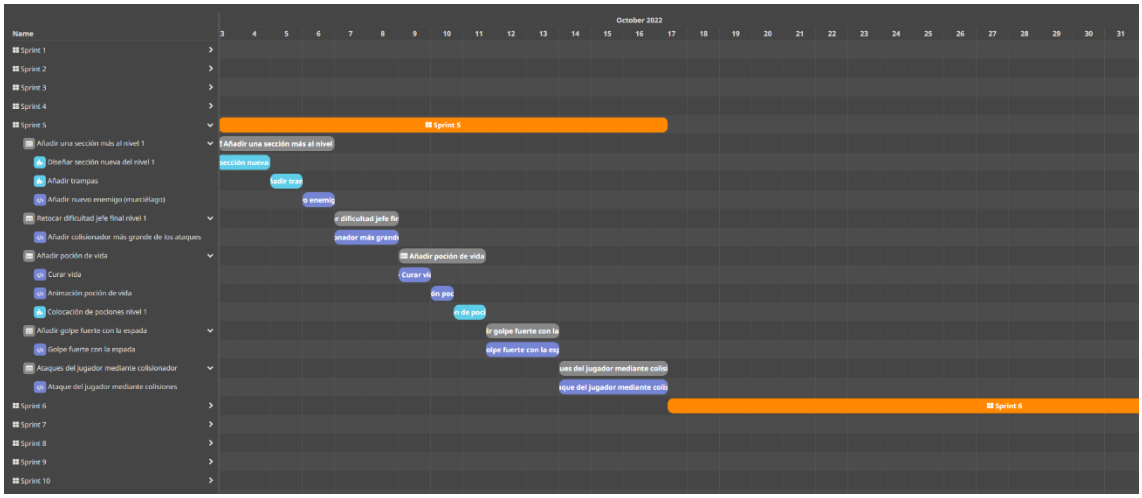

Figura 24. Planificación Sprint 5

Durante este sprint, se realizó una ampliación del primer nivel, añadiendo una nueva sección que incluía más trampas. Se ajustó la dificultad del jefe final del nivel 1, retocando el colisionador de ataque del jefe y del jugador. Además, se introdujo un nuevo elemento, la poción de vida, que permite al jugador curarse, junto con su respectiva animación y colocación en el primer nivel. Por último, se implementó un nuevo ataque para el jugador, un golpe fuerte con la espada, y se ajustó la mecánica de ataque del jugador utilizando colisionadores. Este sprint implicó ajustes significativos en el nivel 1, los cuales mejoraron la jugabilidad y la experiencia del jugador, incluidas nuevas mecánicas de juego y ajustes de dificultad.

Sprint 6

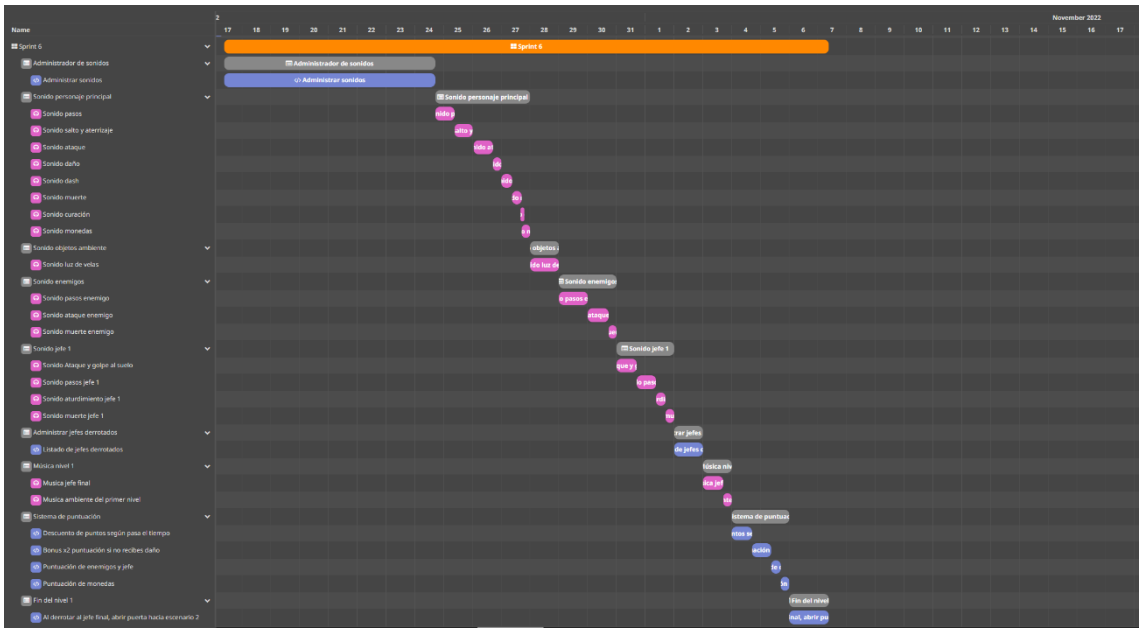

Figura 25. Planificación Sprint 6

En el sexto sprint, la mayor parte se dedicó a la administración de sonidos. Esto implicó la implementación de sonidos para varios elementos del juego, incluidos los pasos, saltos, aterrizajes, ataques, muertes y curaciones del personaje principal, así como los sonidos ambientales, los sonidos de los enemigos y los sonidos relacionados con el primer jefe. Se desarrolló un sistema para administrar los jefes derrotados y se implementó música ambiental para el primer nivel. También se introdujo un sistema de puntuación que incluía descuentos en puntos con el paso del tiempo, bonificaciones por no recibir daño y puntuaciones para derrotar enemigos y jefes. Finalmente, se implementó una función para hacer más lento al jefe final y abrir la puerta hacia el escenario 2 al derrotarlo. Este sprint, por lo tanto, implicó una considerable mejora en la experiencia del usuario al añadir sonidos y música, así como un ajuste necesario a las mecánicas del juego y a la progresión del nivel.

Sprint 7

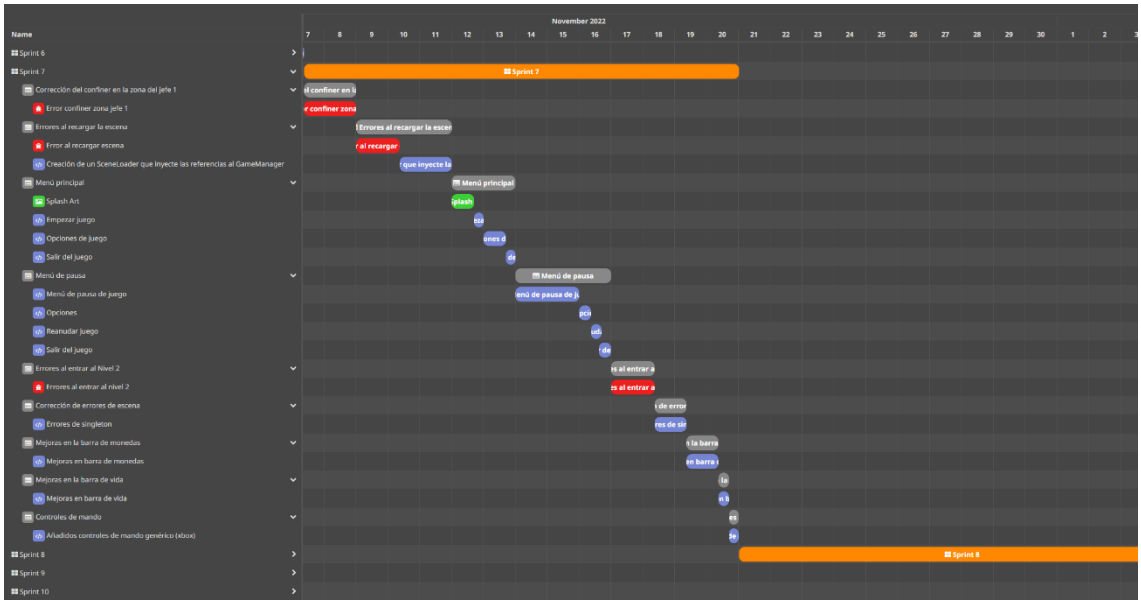

Figura 26. Planificación Sprint 7

Durante este sprint, se corrigieron varios errores críticos, incluido un problema con el *confiner* en la zona del primer jefe y errores que surgían al recargar la escena. Para resolver estos problemas, se creó un *SceneLoader* que inyecta las referencias al *GameManager*. Además, se implementaron las funciones de menú, entre ellos, un menú de pausa con opciones para reanudar o salir del juego, se corrigieron errores que surgían al entrar al nivel 2, se realizaron mejoras en la interfaz de usuario, incluidas las barras de monedas y vida, y se añadieron controles del mando genérico de Xbox. Este sprint se centró en corregir errores, mejorar la interfaz y la experiencia del usuario, lo cual es crucial para asegurar que el juego funcione sin problemas y sea accesible y agradable.

Sprint 8

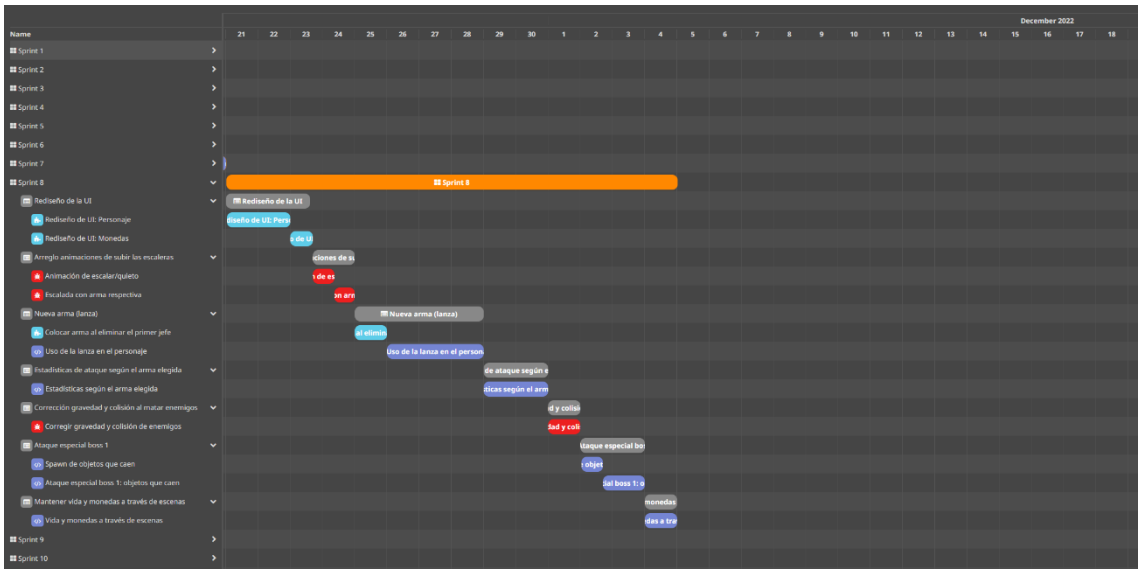

Figura 27. Planificación Sprint 8

El octavo sprint implicó una revisión significativa de la interfaz de usuario (UI), incluidos los elementos de UI del personaje y de las monedas. Además, se corrigieron las animaciones de subir escaleras para el personaje y se introdujo una nueva arma, la lanza, que se otorga al jugador después de eliminar al primer jefe. También se implementaron estadísticas de ataque específicas para cada arma, lo que agregó una capa adicional de estrategia al juego. Se corrigieron problemas con la gravedad y la colisión de los enemigos al ser eliminados, y se implementó un ataque especial para el primer jefe. Finalmente, se añadió una función para mantener la cantidad de vida y monedas del jugador constante a través de diferentes escenas, lo que mejoró la cohesión y fluidez del juego. Este sprint fue vital para pulir varios aspectos del juego y asegurar que la experiencia del jugador sea coherente y gratificante.

Sprint 9

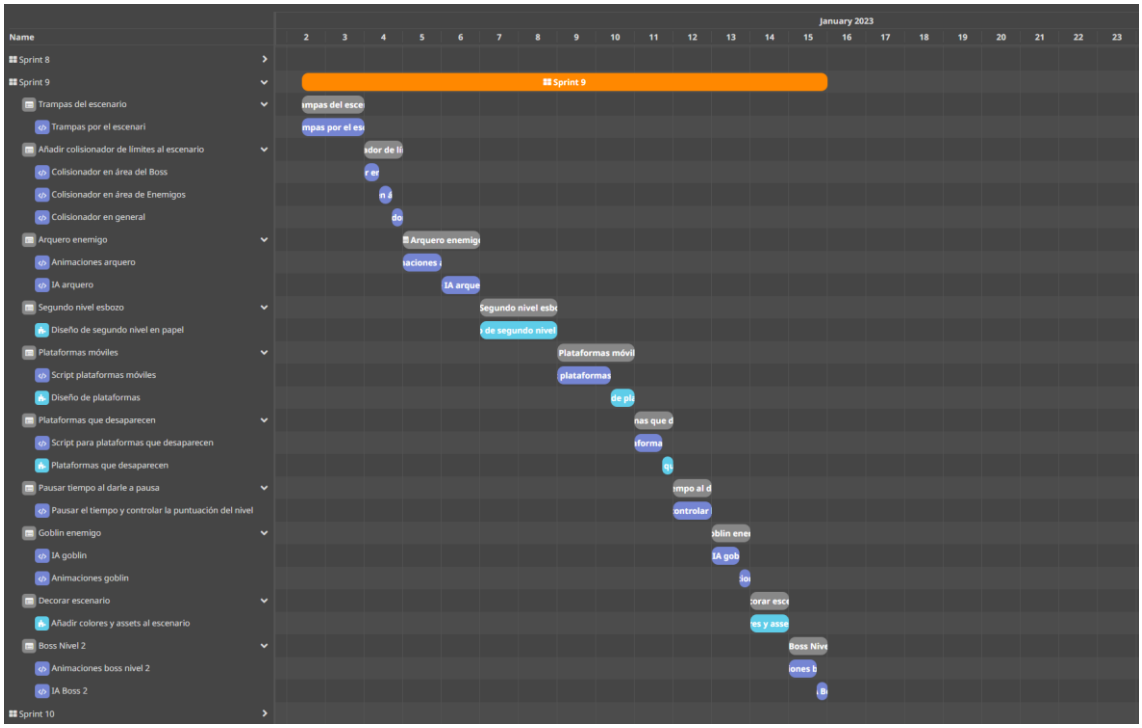

Figura 28. Planificación Sprint 9

El noveno sprint se dedicó a la revisión de las mecánicas del juego. Se introdujeron trampas en el escenario y se añadieron colisionadores de límites para definir las áreas accesibles. Se creó un nuevo enemigo, el arquero, con sus respectivas animaciones y IA. Se esbozó el diseño del segundo nivel en papel, y se implementaron plataformas móviles y plataformas invisibles, cada una con su propio script y diseño. Se añadió la funcionalidad de pausar el tiempo y controlar la puntuación del nivel al pausar el juego. Además, se creó otro enemigo, el goblin, con su IA y animaciones correspondientes. También se dedicó tiempo a decorar el escenario, añadiendo colores y *assets* para mejorar la estética del juego. Por último, se trabajó en el jefe del segundo nivel, creando sus animaciones y su comportamiento.

Sprint 10

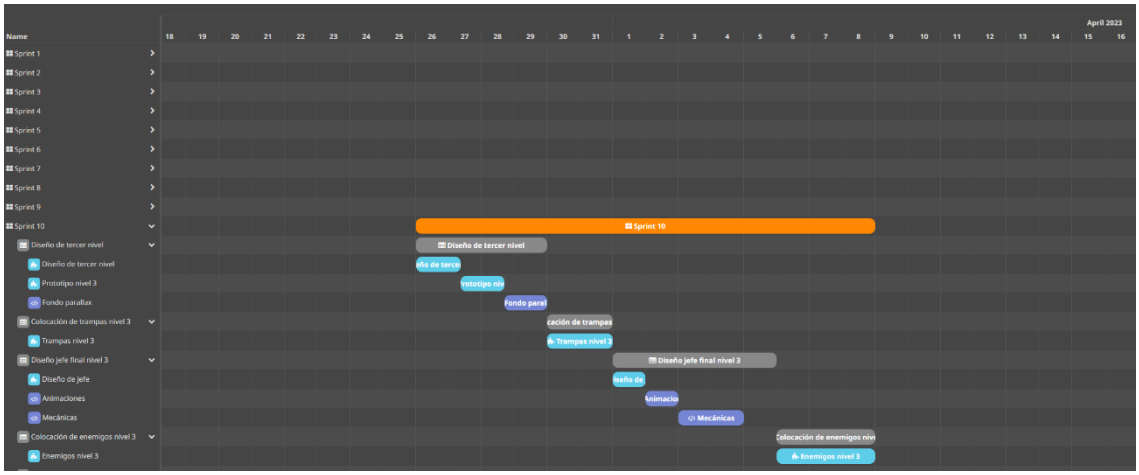

Figura 29. Planificación Sprint 10

El último sprint se centró en el diseño y desarrollo del tercer y último nivel del juego. Se creó el diseño y se desarrolló un prototipo inicial. Para mejorar la estética, se implementó un fondo *parallax*, que crea un efecto de profundidad moviendo el fondo a una velocidad diferente que el primer plano. También se colocaron trampas en el tercer nivel, añadiendo obstáculos y desafíos para el jugador. Además, se diseñó el jefe final del tercer nivel, lo que implicó crear su apariencia, movimientos y ataques. Por último, se colocaron los enemigos, asegurando que estuvieran distribuidos de manera que desafiaran, pero no abrumaran al jugador, dando una conclusión satisfactoria para el jugador.

4.6 Diseño detallado

En esta sección, se van a describir detalladamente los tres sistemas más importantes implementados en el videojuego: el sistema del jugador, el sistema de enemigos y el sistema de niveles.

También se va a mostrar los diagramas de clases pertenecientes a cada uno de estos sistemas.

Como en el proyecto existen más de 60 clases, se van a seleccionar solo aquellas más importantes para cada sistema.

4.6.1 Sistema del Jugador

El objetivo principal del sistema del jugador es gestionar todas las acciones e interacciones que puede realizar el jugador durante el juego. Esto incluye, pero no se limita a, el movimiento del personaje, la interacción con objetos, el uso de armas, y la gestión de la vida y monedas del personaje.

4.6.1.1 Diagrama de clases

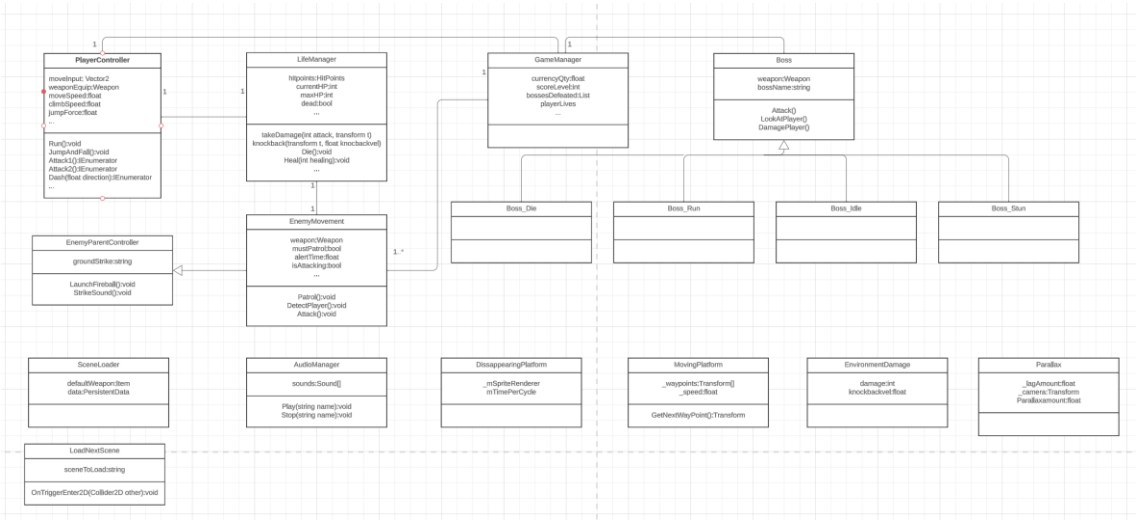

Figura 30. Diagrama de clases UML

Haciendo referencia a la figura 30, encontramos las siguientes clases principales:

La clase **PlayerController** es la encargada de recoger los datos de entrada de un dispositivo, ya sea mando o teclado. A partir de este dato y de una velocidad elegida como variable, el jugador se moverá horizontalmente por el escenario.

En cuanto al salto y al doble salto, según la fuerza determinada y el tiempo que el jugador mantenga el botón, la fuerza será mayor o menor.

También se gestiona el movimiento de esquivar del jugador, el cual tiene unos momentos de invulnerabilidad y atraviesa enemigos.

Si el jugador pulsa los botones de ataque, la clase realizará el ataque débil o fuerte, dependiendo del botón utilizado.

La clase **LifeManager** va asociada no solo al jugador, también a los enemigos, y se encarga, según sus atributos, a gestionar la vida, la curación, el knockback y la muerte de éstos.

Tiene asociada una clase hitPoint, un scriptableObjecto que contiene la vida inicial, actual y máxima. De este modo, gestiona también la muerte de enemigos y del jugador, así como la puntuación de cada enemigo al ser derrotado

4.6.2 Sistema de Enemigos

En el videojuego se pueden distinguir dos tipos principales de enemigos: enemigos básicos y jefes finales. Los enemigos básicos siguen una estructura similar en términos de patrullaje, detección del jugador, ataque cuando están en rango de visión y persecución del jugador si este sale de su rango.

Sin embargo, los jefes finales cuentan con una máquina de estados más avanzada que les permite realizar una variedad de ataques y habilidades especiales, lo que los convierte en desafíos únicos y difíciles para el jugador.

La clase **EnemyMovement** gestiona el daño del enemigo, la velocidad de patrullaje y la velocidad al detectar al jugador y la distancia de detección:

La clase **EnemyParentController** se encarga de gestionar ataques especiales de la clase enemigo a la que está asociada.

A diferencia de los enemigos básicos, los jefes finales tienen una manera diferente de comportarse con el jugador:

La clase **Boss** se encarga de añadir nuevos atributos al enemigo, como los distintos patrones de ataque especial

Las últimas cuatro clases del jefe final en las que se incluyen **Boss_Idle, Boss_Run, Boss_Stun** y **Boss_Die** están adjuntadas al controlador de animaciones [14] y actúan dependiendo el estado en el que se encuentre el jefe [15].

Las cuatro clases tiene dos métodos comunes que consisten en:

- **OnStateEnter**(): se ejecuta la primera vez que se produce el estado en concreto y serviría para gestionar el estado de reposo, de correr, de atontado y de muerte respectivamente del jefe final.

- **OnStateUpdate**(): se ejecuta cada vez que se encuentra en ese estado. Por ejemplo, en el estado de correr se ejecutaría varias veces para perseguir al jugador e intentar atacarle.

4.6.3 Sistema de Niveles

El sistema de niveles es responsable de la estructura y el progreso del juego. Esto incluye la disposición de los enemigos, trampas, plataformas y otros objetos en cada nivel, así como la transición entre diferentes niveles y escenas. Además, este sistema también se encarga de la gestión de la dificultad del juego, asegurando que cada nivel presente un desafío adecuado para el jugador, basándose en su progreso y habilidades adquiridas hasta ese momento. También es responsable de guardar el progreso del jugador, permitiéndole retomar el juego desde el último punto guardado.

La clase **SceneLoader** se encarga de gestionar los datos persistentes en cada escena. Hace uso de un *ScriptableObject* de tipo *PersistentData* que gestiona todos los datos persistentes que se deben de mantener en todos los niveles.

La clase **LoadNextScene** se encarga de gestionar la transición entre los diferentes niveles/escenas del juego, pasando como argumento el nombre de la escena en cuestión.

La clase **Parallax** se encarga de producir el efecto del mismo nombre en el fondo cuando el jugador se mueve.

El daño por entorno también entra dentro del sistema de niveles, y la clase **EnvironmentDamage** es la que se encarga de ello

Una de las clases más importantes del proyecto es el **GameManager**.

Está en contacto con casi todas las demás clases, tales como **PlayerController, EnemyMovement, LifeManager, AudioManager.**

Se encarga de gestionar las monedas del jugador, la puntuación, los jefes derrotados, las armas conseguidas, y las vidas actuales del jugador.

Del sistema de sonido se encarga prácticamente la clase **AudioManager**.

Contiene como atributos una lista de clases de tipo Sound a las cuales se les puede asociar una pista de audio.

Como métodos podemos utilizar **Play()** y **Stop()** pasando como argumento el nombre de la pista para empezar o parar el sonido.

Como parte de los niveles 2 y 3, se añaden diferentes tipos de plataforma. La clase **DissapearingPlatform** añade la posibilidad de hacer un circuito de plataformas que desaparecen y reaparecen en un tiempo configurable.

El segundo tipo de plataforma añadida es la clase **MovingPlatform**, que gestiona el movimiento de manera automática y horizontalmente de la plataforma mediante el uso de caminos predefinidos y una velocidad marcada en el editor.

4.7 Tecnología utilizada

4.7.1 Unity

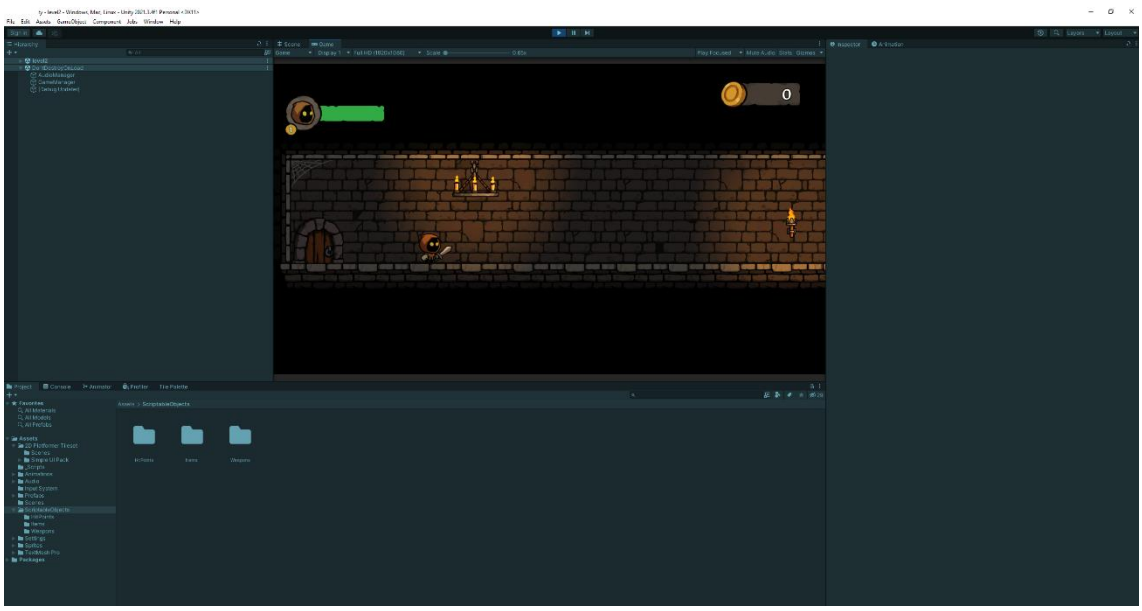

Figura 31. Motor Unity en modo Juego

Unity es uno de los motores de juego más populares y utilizados en la industria de los videojuegos. Es un motor multiplataforma que permite a los desarrolladores crear juegos en 2D y 3D para una variedad de plataformas, incluidos móviles, consolas, PC y web. Unity proporciona una amplia gama de herramientas y recursos que facilitan el desarrollo del juego, desde un editor visual que permite a los desarrolladores crear y organizar escenas, hasta un sistema de físicas

robusto y un potente motor de renderizado. Además, Unity cuenta con una amplia comunidad de desarrolladores y un mercado de *assets*, lo que permite a los desarrolladores acceder a una gran cantidad de recursos y compartir sus propios trabajos.

Algunas de las características clave de Unity incluyen:

- **Motor de Renderizado**: Unity cuenta con un motor de renderizado avanzado que soporta gráficos en 2D y 3D, iluminación dinámica, sombras en tiempo real, y una variedad de efectos especiales.
- **Sistema de Físicas**: Unity incluye un sistema de físicas completo que permite a los desarrolladores simular el comportamiento físico de objetos y personajes en el juego.
- **Scripting**: Unity utiliza C# como lenguaje de programación, lo que permite a los desarrolladores escribir scripts para controlar el comportamiento de los objetos y personajes en el juego.
- **Editor visual**: Unity proporciona un editor visual que permite a los desarrolladores crear y organizar escenas, ajustar propiedades de objetos, y visualizar el juego mientras lo desarrollan.
- **Soporte Multiplataforma**: Unity permite a los desarrolladores crear juegos para una amplia variedad de plataformas, incluidos móviles, consolas, PC y web, con la capacidad de exportar el juego a diferentes plataformas con poco o ningún cambio en el código.

Uno de los factores más importantes para decantarte sobre un motor u otro, es la Asset Store de Unity.

La tienda da la posibilidad de conseguir de manera gratuita o a precios muy asequibles arte y componentes que facilitarán mucho el trabajo en el desarrollo del videojuego.

4.7.2 Rider

Figura 32. Rider logo

Rider [5] es un IDE (Entorno de Desarrollo Integrado) desarrollado por JetBrains, diseñado específicamente para trabajar con el lenguaje de programación C#. Es ampliamente utilizado por los desarrolladores de Unity debido a su integración profunda con el motor y sus características avanzadas de edición de código, depuración y análisis de rendimiento.

Algunas de las características clave de Rider incluyen:

- **Integración con Unity**: Rider se integra profundamente con Unity, proporcionando características específicas de éste, como la edición de shaders, la visualización de logs y la depuración de scripts directamente desde el IDE.
- **Análisis de código**: incluye un potente analizador de código que ayuda a los desarrolladores a escribir código de alta calidad, detectando errores y sugiriendo mejoras en tiempo real.
- **Depuración**: también proporciona herramientas avanzadas de depuración que permite inspeccionar y modificar el estado de su juego mientras se ejecuta, tanto en el editor de Unity como en el juego en sí.
- **Navegación y refactorización**: incluye una serie de herramientas [6] de navegación y refactorización que permite explorar y modificar el código de manera eficiente.
- **Control de versiones**: tiene soporte para todos los sistemas de control de versiones populares, como Git, lo que facilita el trabajo en equipo y la gestión de cambios en el proyecto.

4.8 Desarrollo de la solución propuesta

En esta sección se va a desglosar el resultado de la solución propuesta, de manera que se explique cómo se ha pasado de la propuesta inicial a la solución.

Se muestra el resultado del juego, el desarrollo de niveles, los elementos interactuables del entorno, el desarrollo de los enemigos, las mecánicas básicas del jugador, la interfaz visual y el sistema de sonido.

4.9 Arte

Se barajaron diferentes estilos artísticos 2D para este videojuego, como el *pixel art*, sin embargo, se buscaba un dibujo más definido por lo que se ha optado por el dibujo digital a mano, con una resolución de 256x256.

Figura 33 Ejemplo de tiles

Los *assets* del arte, fueron comprados en Unity con el nombre de *2D Platformer Tileset* [13], el cual contiene 15 enemigos, 1 personaje principal, 311 objetos decorativos y 3 fondos parallax de 5 capas cada uno.

4.9.1 Desarrollo de escenarios

El desarrollo de los escenarios se ha realizado utilizando una herramienta llamada *Tilemap*, la cual está integrada en el motor Unity.

Con esta herramienta se puede pintar cada celda individualmente por diferentes capas, otorgándolas colisiones y diferentes efectos.

Se ha dividido en las siguientes secciones:

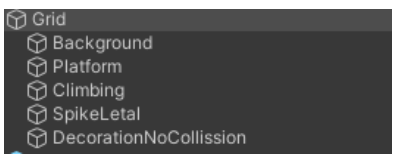

Figura 34. Grid de tiles presentes en el juego

- **Background**: Esta capa se utiliza para añadir elementos que van más al fondo, los cuales no colisionan con el jugador y sirven para detallar el mapeado.
- **Platform**: La capa de plataformas, el jugador puede colisionar con ella y se utiliza para ofrecer al jugador plataformas de salto o para impedir el paso.
- **Climbing**: Capa orientada a la escalada. Todo *tile* que esté pintado en esta capa podrá ser utilizado por el jugador para escalar.
- **SpikeLetal**: Esta capa tiene como efecto hacer daño al jugador. Si éste colisiona con ella, dependiendo de los atributos que tenga, hará daño y empujará al jugador. La mayor parte de esta capa ha sido utilizada para pintar pinchos y trampas.
- **DecorationNoCollission**: esta capa se ha utilizado para aquellas celdas que necesiten ser pintadas para decorar, que estén por delante del personaje, pero que sin embargo no tenga ningún tipo de colisión.

El juego se divide en tres niveles que se van desbloqueando cuando se derrota al jefe del nivel.

El jugador comienza en una prisión situada en lo más profundo de una mazmorra.

Su objetivo será subir a la superficie para poder escapar, pero en su camino se encontrará enemigos y trampas mortales que tendrá que superar.

El primer nivel [16] es el más accesible de todos y los primeros compases de éste sirven para que el jugador se familiarice con los controles, enfrentándose por separado, primero a las plataformas, al primer enemigo esqueleto, las primeras trampas y por último al jefe final [8]. Se han distribuido caminos secretos con recompensas como monedas y pociones para dar sentido a la exploración.

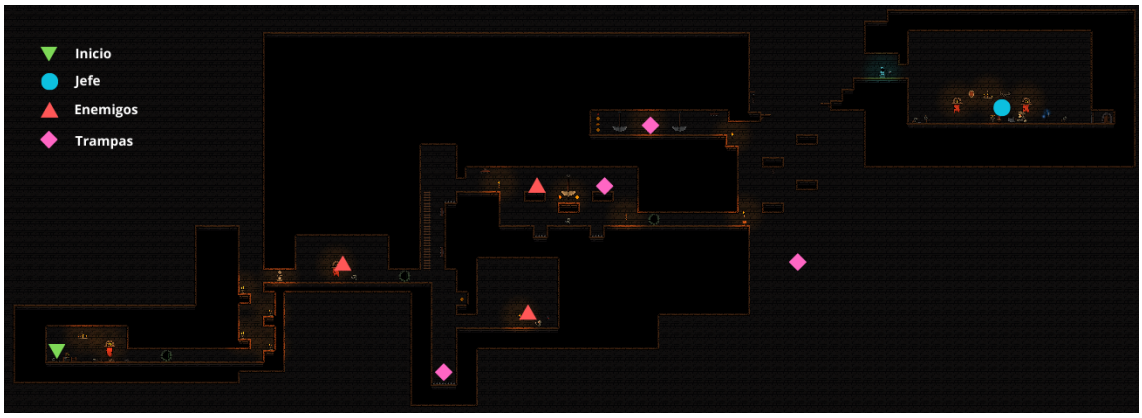

Figura 35. Diseño del nivel 1

Cada nivel está compuesto por diferentes enemigos y trampas, siendo el primer nivel más fácil e incrementando la dificultad a medida que avanzas.

En el caso del segundo nivel, se ha incrementado la dificultad tanto en la parte de las plataformas, como con dos nuevos enemigos, el arquero y el esqueleto mago.

Se han añadido trampas más mortíferas, plataformas móviles y plataformas que desaparecen

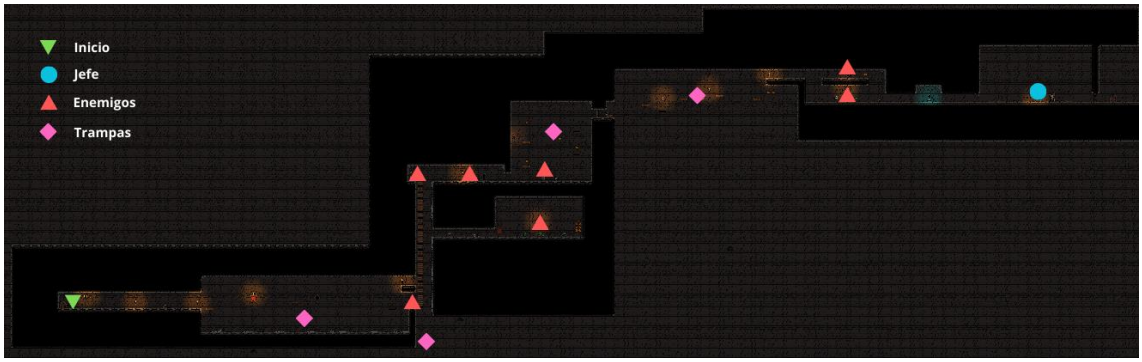

Figura 36. Diseño del nivel 2

El último nivel está ambientado en una cueva, el último paso para conseguir salir de la mazmorra. Está compuesto por un fondo *parallax* que engloba 5 capas de fondo, las cuales se solapan y mediante un *script*, cuando la cámara sigue al jugador, consigue que cada fondo vaya a una velocidad diferente, siendo la capa más cercana al jugador la más rápida, y las capas más lejanas más lentas.

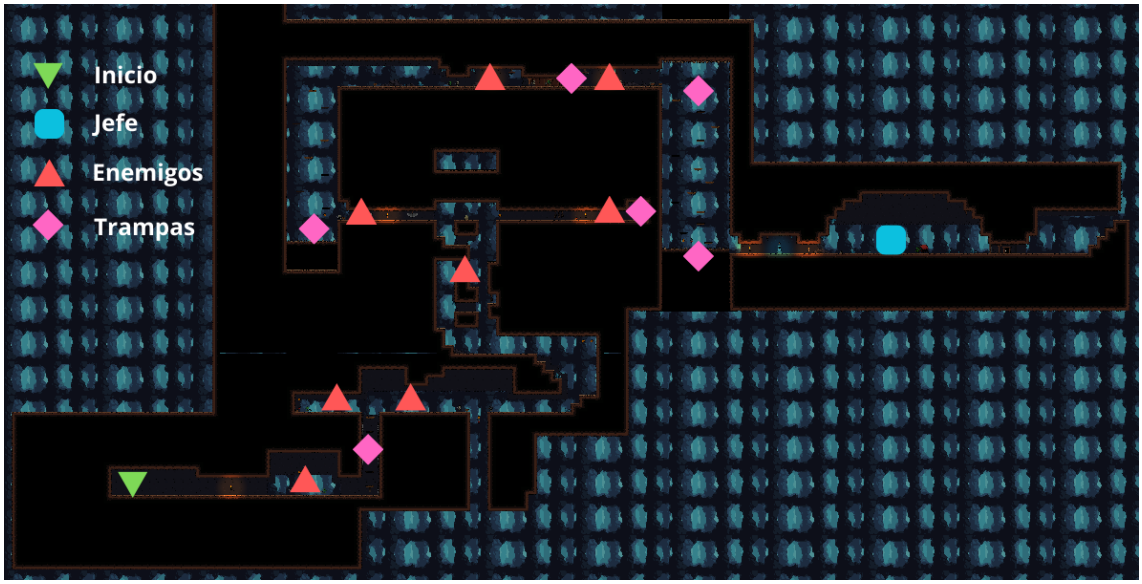

Figura 37. Diseño del nivel 3

Como se ha comentado en la sección anterior, los elementos para la creación del mapa se han dibujado mediante la herramienta de Unity *Tilemap*, sin embargo, para pintar el grueso de todos los niveles se ha utilizado un componente llamado *Rule Tile*.

La *Rule Tile* es un tipo especial de *tile* que se utiliza para cambiar automáticamente el *sprite* de un *tile* en función de sus vecinos. Esto es especialmente útil para crear terrenos o caminos que deben cambiar su apariencia en función de los *tiles* circundantes, lo que ha ayudado a crear un entorno más cohesivo y visualmente agradable con menos esfuerzo.

Por ejemplo, para pintar un camino se puede usar una de estas reglas para definir qué *sprite* se debe usar para cada tile en función de si tiene un *tile* vecino a la izquierda, a la derecha, arriba o abajo. Entonces, en lugar de colocar manualmente cada variante se puede colocar una sola regla y dejar que Unity elija automáticamente el *sprite* correcto en función de su contexto.

Esto ha permitido ahorrar mucho tiempo y esfuerzo. Además, estas reglas son bastante personalizables, por lo que ha permitido definir reglas propias y adaptarlas a las necesidades del juego.

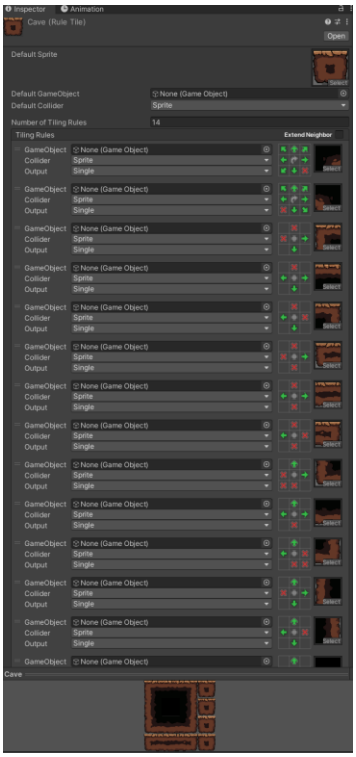

Figura 38. Reglas de Tile de la cueva

En la imagen anterior, se puede ver como se ha añadido un total de 15 sprites. Se elige uno como base y a continuación se van añadiendo sus vecinos. Para cada uno de ellos es necesario configurar si va a haber un tile arriba, abajo, derecha, izquierda o no lo va a haber.

Por ejemplo en la primera regla, las flechas están en verde menos abajo a la derecha, esto quiere decir que en el único sitio donde no se va a poder pintar nada es abajo a la derecha. Esto es así porque ese *sprite* pertenece a una esquina inferior derecha.

Para darle más detalle e inmersión, se ha llenado el escenario de elementos que tienen luz propia, tales como velas, antorchas y otros focos de luz de diferentes tonalidades.

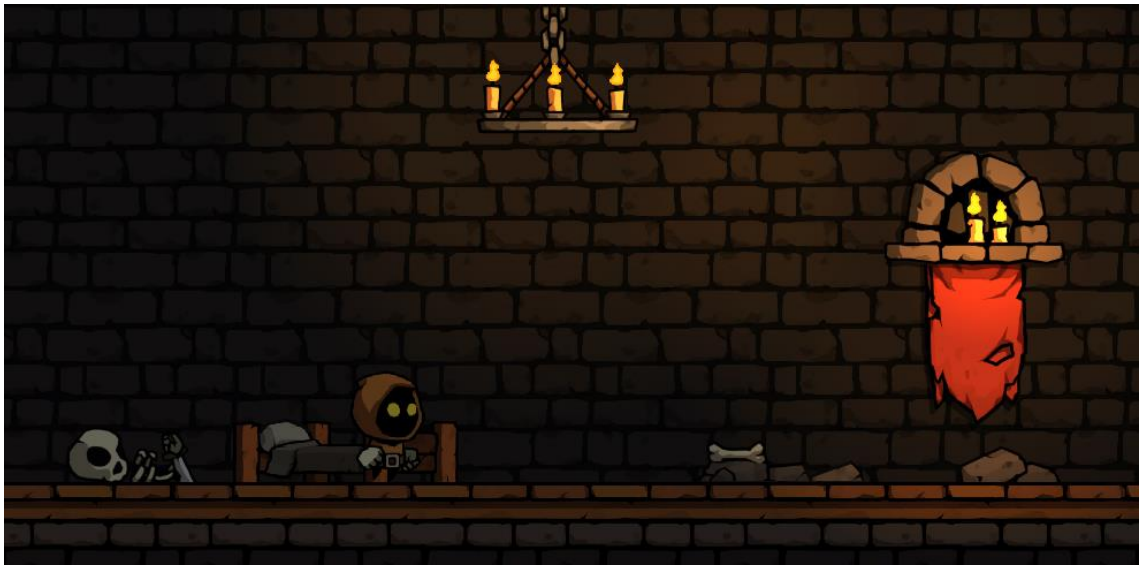

Figura 39. Iluminación del nivel 1

Este tipo de luces se llaman luces dinámicas. Lo que se puede conseguir con este tipo de luces es que los demás objetos tengan luces y sombras dependiendo de la distancia a la que se encuentran con respecto a la luz de la escena.

Por último, la posibilidad de moverse entre niveles se ha resuelto con la adición de una puerta.

Figura 40. Puerta para el siguiente nivel

Esta puerta se activa cuando el jugador entra dentro del rango de su *collider*, de esta forma, automáticamente el jugador aparece en el siguiente nivel.

4.9.2 Elementos interactuables

A lo largo del juego, el jugador se va a encontrar con una serie de elementos con los que puede interactuar. La manera de interactuar con la mayoría de los objetos se resuelve con un *Collider2D*. Es decir que, si el jugador se acerca a ellos, el objeto lanzará un evento a la clase encargada de procesarlo.

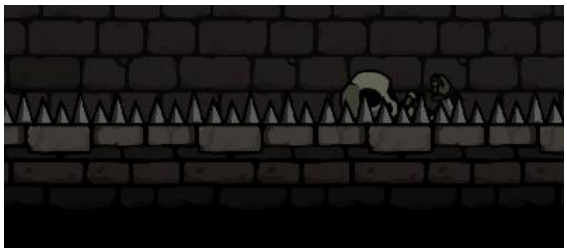

Figura 41. Trampa de pinchos

Pinchos: Una de las primeras trampas que nos podemos encontrar en el juego. Están dibujados en diferentes posiciones y aunque la mayoría de ellos solo quitan una parte de vida del jugador, existen otro tipo de pinchos que pueden quitarte la vida entera.

Figura 42. Trampa de guadaña giratoria

Guadaña giratoria: Una guadaña que gira 360º de manera continua y que realiza daño cuando el jugador es golpeado por el filo. El giro se ha resuelto utilizando un *AnimatorController* y creando una animación que gire, ajustando el pivote a la punta de la rueda.

Figura 43. Trampa de pinchos aplastantes

Pinchos aplastantes: Pinchos que se mueven verticalmente de manera indefinida. Para esquivarlos hay que esperar a que los pinchos estén en la parte superior. El movimiento se ha

resuelto de la misma manera que la guadaña, con un *AnimatorController* y una animación que se mueve hacia arriba y hacia abajo, dejando un margen para que el jugador pueda esquivarla.

Figura 44. Poción de curación

Poción de curación: Se pueden encontrar tres tipos de pociones, pequeñas, medianas y grandes. Cuanto mayor sea la poción curativa, más vida recuperará el jugador al recogerla. La recogida se resuelve mediante el método *onTriggerCollider2D*(), el cual detecta si la persona que ha entrado en su radio de colisión es el jugador. De ser así, la poción llama al *LifeManager* del jugador, pasándole como argumento la curación del objeto. Este valor está predefinido en el *ScriptableObject* de tipo *Healing*

Figura 45. Monedas

Monedas: A lo largo de los escenarios, el jugador encontrará muchas monedas repartidas por el mapa, algunas a simple vista y otras más escondidas. Proporcionan puntuación adicional al finalizar el nivel. La recogida se resuelve de la misma forma que la poción de curación, a excepción de que al *GameManager* se le pasa como argumento la puntuación de la moneda y éste la registra en la variable correspondiente.

Figura 46. Armas

Armas: El jugador puede utilizar diferentes armas para atacar a los enemigos. Algunas de éstas se pueden encontrar al derrotar un jefe final. La recogida se resuelve también con un *trigger* al entrar en su radio de colisión. Al ser recogida se guarda en una lista del *GameManager*. A partir de este momento el jugador podrá cambiar de arma y utilizar sus propiedades como por ejemplo, empujón o lanzamiento, las cuales se determinan por el *ScriptableObject* asociado y que guarda la información.

Figura 47. Palanca activable

Palanca: En diferentes escenarios, el jugador deberá de activar palancas para que una puerta cerrada se abra y permita al jugador avanzar en el nivel. Se activan al ser golpeadas. Estas palancas tienen un temporizador. Si pasan *x* segundos (determinado en el inspector de Unity), la palanca vuelve al estado natural y las puertas se cierran.

4.9.3 Mecánicas del jugador

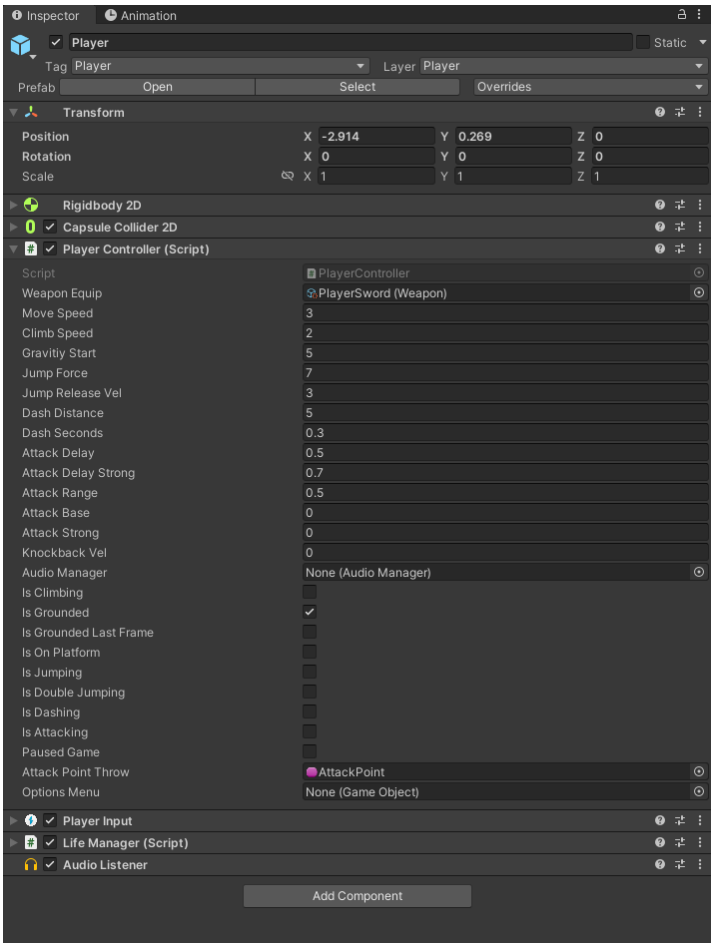

Figura 48. Prefab del jugador con todos sus componentes

En la imagen anterior podemos ver los componentes asociados al jugador.

En este caso, se ha querido hacer uso de las físicas que proporciona Unity, es por ello que tiene asociado un componente *Rigidbody2D*, el cual va a permitir el uso de fuerzas y gravedad para dotar de movimiento al jugador.

Estos posibles movimientos que puede realizar el jugador son los siguientes:

- **Movimiento horizontal**: El jugador va a poder moverse a la izquierda o a la derecha a una velocidad determinada.
- **Salto**: El jugador, si está en el suelo, va a poder realizar un salto. La fuerza de éste dependerá de la presión que realice el jugador del botón en cuestión.
- **Doble salto**: Igual que el salto, pero con la diferencia de que solo se puede realizar después de un salto inicial.

- **Esquivar**: El jugador avanza rápidamente rodando hacia la posición que haya elegido. Se puede realizar también en el aire, permite traspasar colisiones enemigas y otorga unos milisegundos de invulnerabilidad.
- **Ataque básico**: Golpe básico del jugador que realiza un daño normal y un empujón medio al enemigo. Puede lanzarse de manera más rápida.
- **Ataque fuerte**: Golpe fuerte del jugador que realiza un daño fuerte y un empujón grande al enemigo. Tarda más en lanzarse y tiene un tiempo de recarga superior.

Por otra parte, se ha asociado también al jugador el componente *Capsule Collider2D*, el cual se ajusta al personaje principal de manera que el resto de los elementos del escenario puedan interactuar con él.

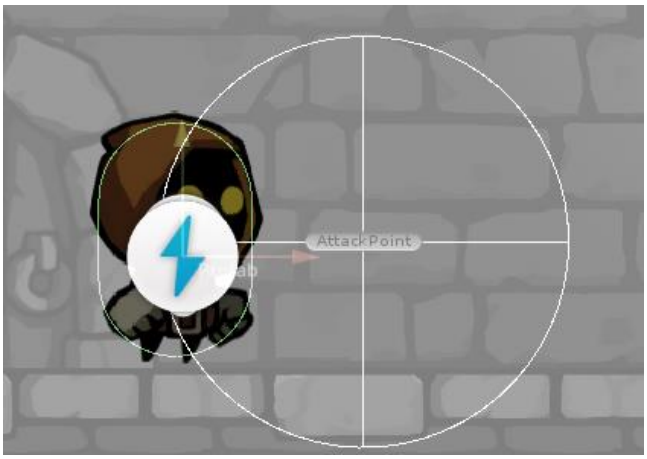

Figura 49. Radio de colisión y de ataque

El componente *PlayerController* se asocia al personaje principal y como se ha comentado anteriormente, es el *script* que va a otorgar al jugador todos los atributos y métodos necesarios para su control.

El componente *PlayerInput* de Unity ha facilitado el uso botones de entrada para el manejo del personaje principal. Proporciona *triggers* necesarios para poder interceptar los botones que pulsa el jugador. Este componente facilita enormemente el uso de cualquier dispositivo de entrada, ya sea un teclado o un mando.

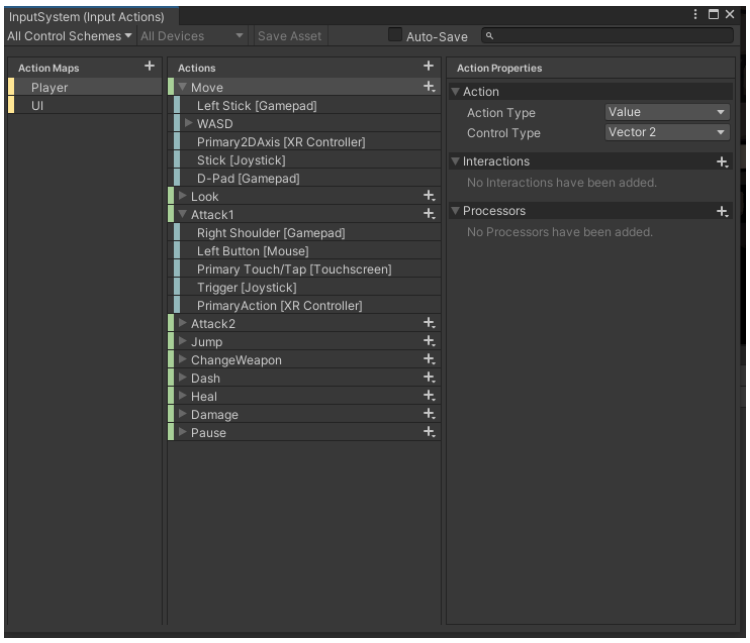

Figura 50. Configuración de dispositivos de entrada

El componente *AudioListener* permite que el jugador pueda escuchar los sonidos que provienen de otros objetos. Por ejemplo, los pasos de los esqueletos se escucharían según el lugar de donde proviene el sonido.

Por último, el componente *Animator*, el cual está asociado a un objeto hijo del jugador, es el que va a permitir transicionar entre las diferentes animaciones que tiene el personaje.

A continuación, muestro como quedaría el árbol de transiciones:

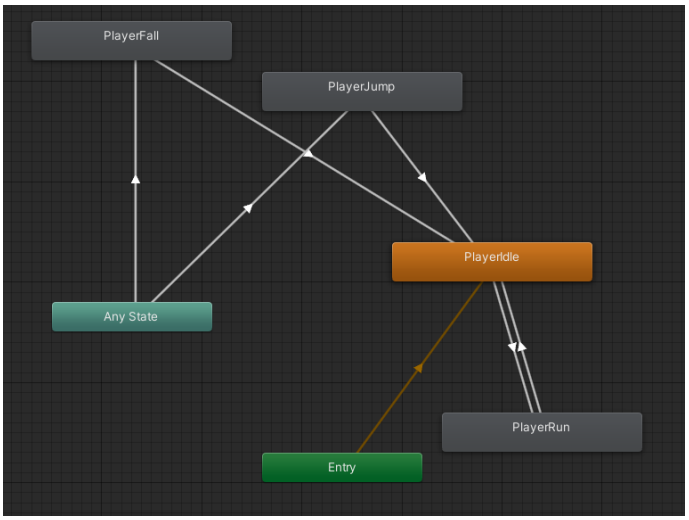

Figura 51. Controlador de animaciones sin armas

Esta transición básica se utiliza cuando el jugador no tiene armas, por lo que desde reposo el jugador puede correr. Y desde cualquier otro estado el jugador puede saltar.

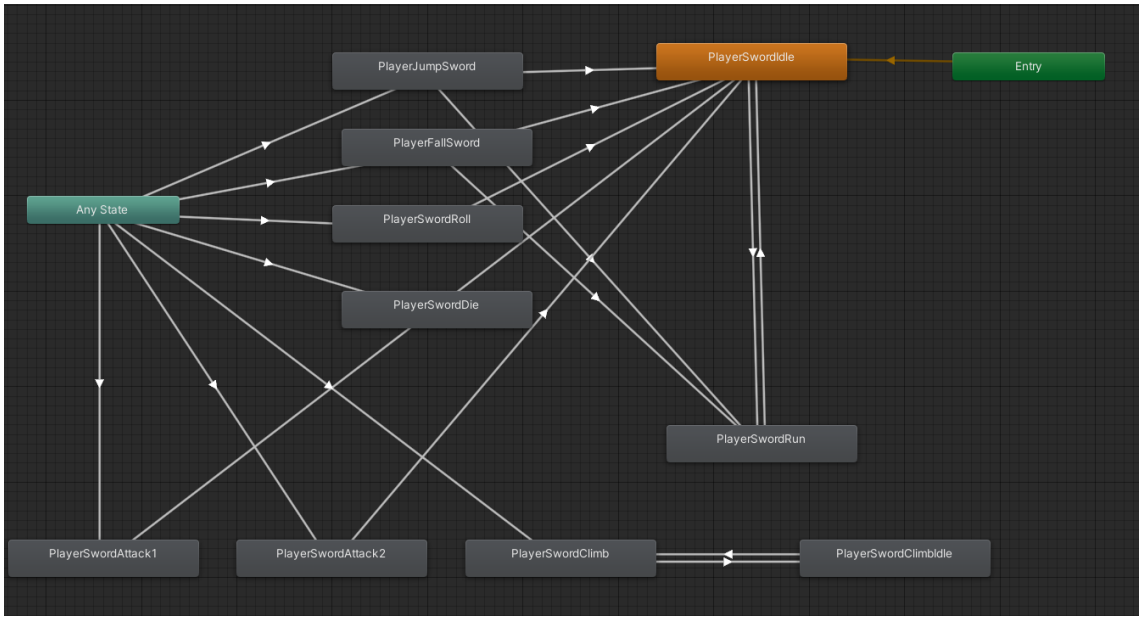

Figura 52. Controlador de animaciones con la espada

En este caso, el árbol de animaciones es para el uso de la espada. El jugador puede saltar, esquivar, atacar, escalar y morir desde cualquier estado, y en reposo, el jugador puede correr.

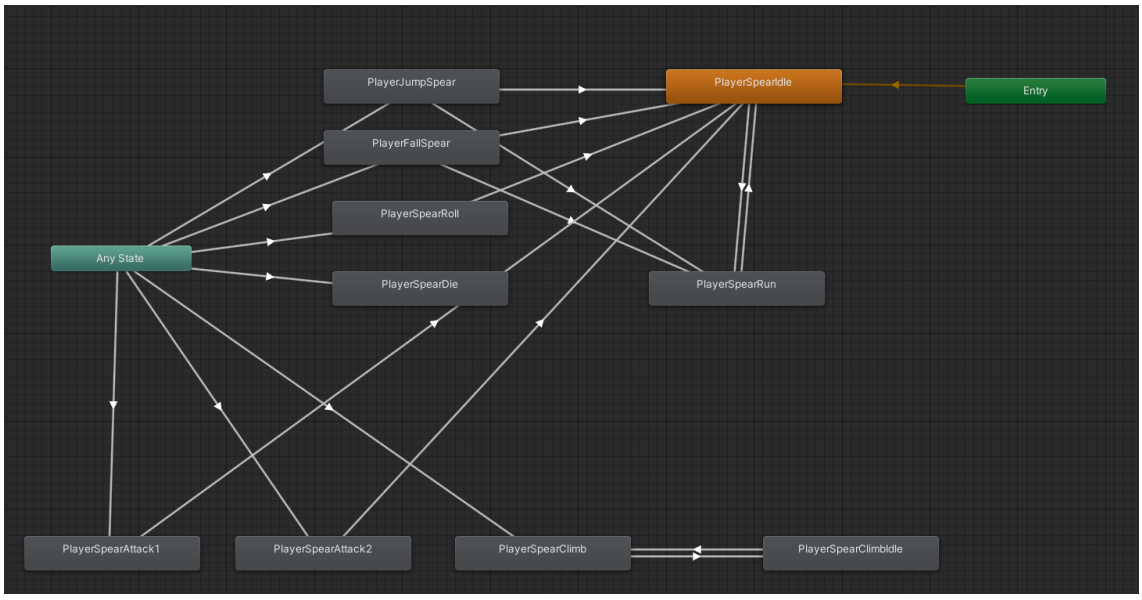

Figura 53. Controlador de animaciones con la lanza

En este caso el jugador empuña una lanza y se haría de la misma forma.

Figura 54. Parámetros de transición entre animaciones

En esta imagen se puede comprobar una serie de parámetros que se utilizan en las transiciones de cada estado, permitiendo pasar de uno a otro o impidiendo el acceso a animaciones no deseadas.

Por último, para cada transición se ha creado una animación.

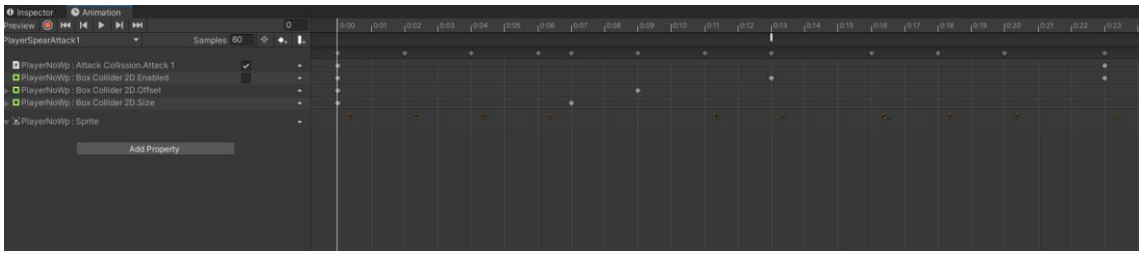

Figura 55. Ejemplo de animación de ataque, puntos clave y atributos

Esta ventana de Unity permite adjuntar una serie de *sprites*, que unidos, componen una animación. Como si de un editor de video se tratara, se pueden añadir puntos clave, y a partir de estos puntos, modificar algunos atributos del jugador. En el ejemplo anterior tenemos la animación de ataque básico y se puede comprobar que en algunos *frames* de la animación, se activa y desactiva el *Collider* que permitirá hacer daño a los enemigos.

4.9.4 Desarrollo de enemigos

A la hora de realizar los enemigos básicos, se ha intentado abstraer la programación para que todos ellos pudieran utilizar el mismo *script* y añadir alguna variación si fuera necesario.

El uso del *prefab* de Unity ha facilitado este proceso.

En Unity, un *prefab* es un objeto o un conjunto de objetos preconfigurados que se puede guardar y reutilizar múltiples veces.

Crear enemigos en un videojuego puede ser un proceso laborioso, especialmente si hay muchos tipos diferentes de enemigos, cada uno con sus propias características, comportamientos y habilidades. Aquí es donde los *prefabs* han sido realmente de ayuda.

En lugar de crear manualmente cada enemigo, se ha creado un *prefab* para cada tipo.

Una vez creado, se pueden instanciar nuevas copias de ese enemigo en cualquier parte del escenario. Además, si se necesita hacer un cambio en un tipo particular de enemigo, solo es necesario hacer el cambio en el *prefab* y se reflejará automáticamente en todas sus instancias.

A continuación, se van a mostrar los enemigos del juego, el árbol de animación y las mecánicas que ofrecen.

4.9.5 Enemigos básicos

Los componentes asociados al enemigo son muy parecidos a los que están asociados al personaje principal.

RigidBody2D para las físicas y el movimiento, *BoxCollider2D* que actúa como límite para que el enemigo no se salga del camino y el método que se define en el propio *script*, en el cual se hace uso del método *RayCast* proporcionado por Unity para detectar al jugador.

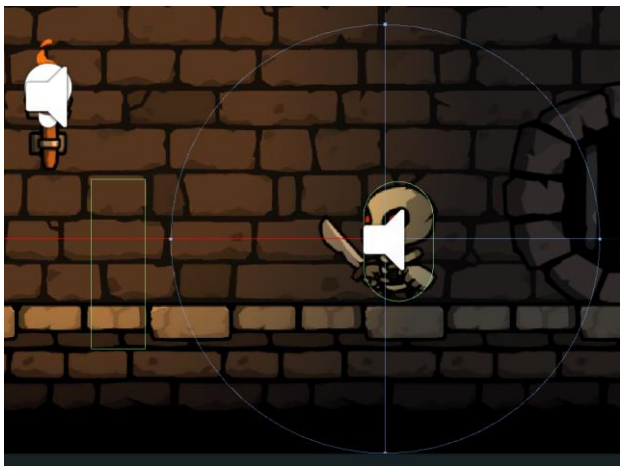

Figura 56. Radio de colisión, de ataque y visión de los enemigos

En la imagen se pueden ver los *Gizmos* de Unity, que son herramientas visuales para identificar los elementos que se han comentado. El rayo rojo es el *RayCast*, cuando éste choca con el jugador, el enemigo advierte su presencia y le persigue. El radio azul se utiliza para el sistema de alerta. Cuando el jugador sale del radio azul varios segundos, el enemigo vuelve al estado normal y posteriormente a la patrulla.

El ataque y daño del enemigo se resuelve activando y desacivando otro *BoxCollider2D*, que está posicionado en el arma del enemigo.

Figura 57. Colisión del enemigo al atacar

Como se observa en la imagen, cuando el enemigo efectúa el ataque, el *collider* aparece, y si este detecta al jugador, se produce el daño.

En cuanto al árbol de animaciones, tenemos el siguiente:

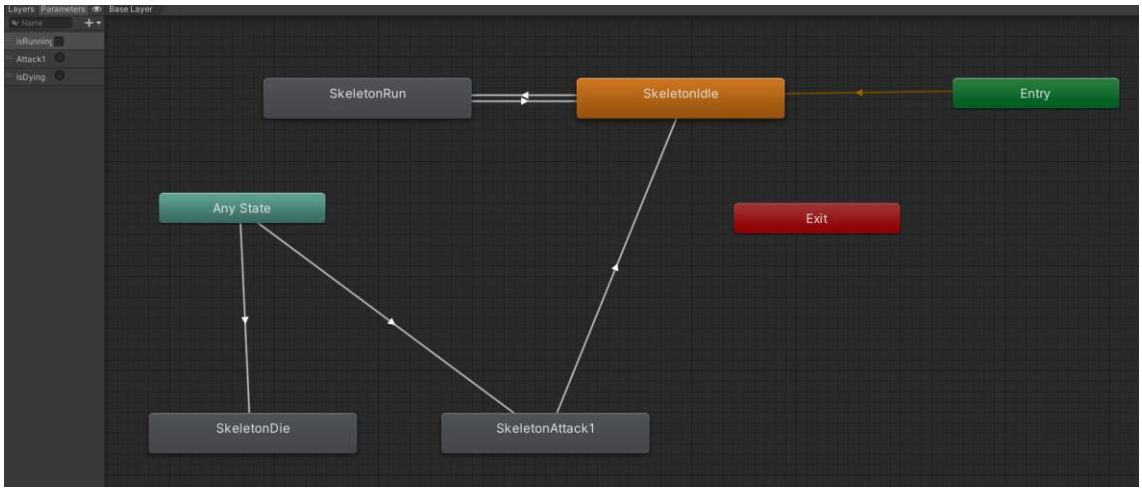

Figura 58. Controlador de animaciones del esqueleto

El esqueleto comienza en su estado de reposo. Cuando el *script EnemyMovement* considera que el enemigo tiene que patrullar, se llama a una función que pone el parámetro isRunning a *True*, por lo que el esqueleto pasa de estar en reposo a correr. Esto también se activa en el momento en el que el enemigo detecta al jugador. Cuando esto último ocurre, el enemigo se acerca y cuando el personaje entra en el rango, se activa el parámetro de tipo *trigger Attack1*. Esto resulta en el ataque al jugador.

Por último, cuando el *LifeManager* del enemigo detecta que sus puntos de vida bajan a 0, se aciva el *trigger isDying*, que resulta en la animación de muerte del enemigo y todas sus componentes activas se desactivan.

4.9.6 Jefes finales

Los jefes finales tienen un comportamiento diferente al de los enemigos básicos.

Por una parte, el sistema de detección es diferente. La sala del jefe tiene un *trigger* que se activa cuando el jugador entra en la zona. De esta forma, el jefe y la música de éste se activan y empieza la batalla.

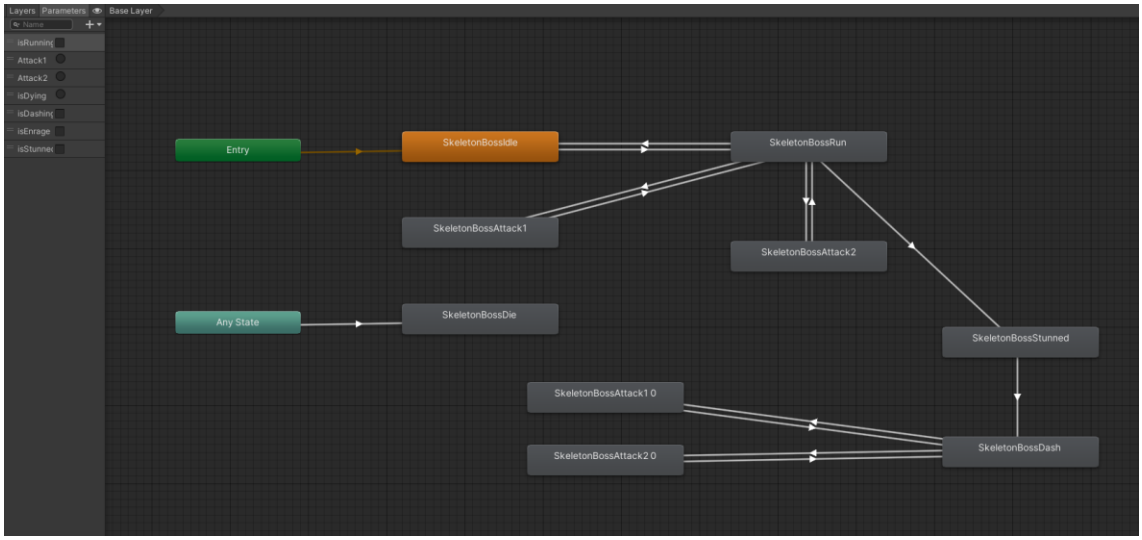

Figura 59. Controlador de animaciones del Esqueleto jefe

A diferencia de los enemigos comunes, los jefes finales están desarrollados a partir de máquinas de estado. En Unity se les llama *StateMachineBehaviour*.

Esta clase se usa para adjuntar comportamientos a los estados en un *AnimatorController*. Lo que permite es que cada uno de los estados del jefe tenga una lógica y comportamiento diferentes.

El árbol de animaciones actúa de la siguiente forma:

SkeletonBossIdle – El jefe está en estado reposo hasta que el jugador activa el *trigger* del escenario. En este momento el jefe pasa al estado *SkeletonBossRun*.

SkeletonBossRun – Desde este estado se calcula la posición del jugador restando la posición del enemigo y la del jugador. Una vez el jefe ha posicionado al jugador, le persigue e intenta colocarse a rango para atacarle. Si lo consigue, el enemigo tiene 50% de posibilidades de realizar el ataque1 o el ataque2, pasando así a *SkeletonBossAttack1* o *SkeletonBossAttack2.*

Si el enemigo está al 50% de vida o inferior, se activa el estado *SkeletonBossStunned.*

El enemigo queda atontado unos segundos y se enfurece, entrando en el estado *SkeletonBoshDash*, en el cual el enemigo adquiere más daño y velocidad de movimiento.

A partir de aquí puede realizar dos ataques especiales nuevos, *SkeletonBossAttack10 y SkeletonBossAttack20,* el primero consiste en un golpe en arco que abarca más radio que el golpe normal y el segundo es igual que el anterior, con la diferencia que al golpear el suelo caen objetos desde arriba.

Como último estado tenemos el estado de muerte del enemigo *SkeletonBossDie*. Cuando el jugador consigue que los puntos de vida del enemigo lleguen a 0, el enemigo muere y se activa el

trigger correspondiente para reproducir la música de vencedor, abrir la puerta del siguiente nivel y dejar el arma del jefe lista para ser recogida por el jugador.

4.9.7 Sistema de sonido

El sistema de sonido que se ha implementado en el juego se basa en un objeto llamado *AudioManager*.

Este objeto se encuentra en todos los niveles del juego y solo puede existir uno de ellos.

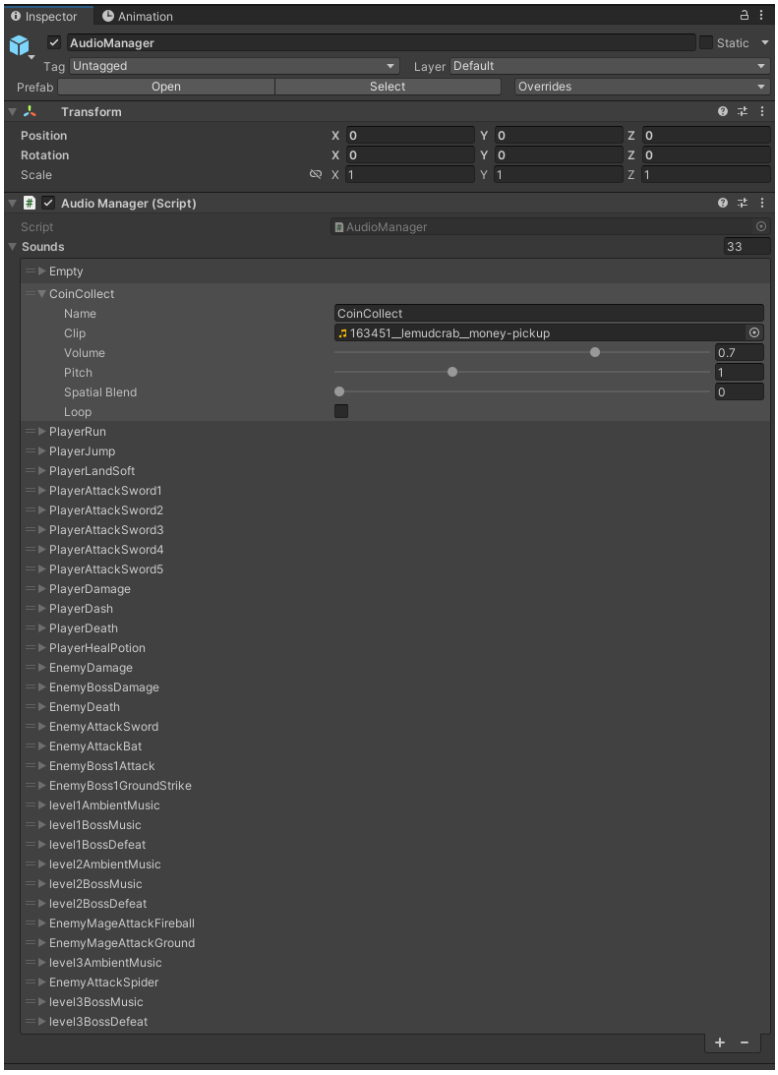

Figura 60. Sistema y gestión audio

Está resuelto de la siguiente manera.

Al script AudioManager se le ha añadido una variable de tipo lista. Esta lista va a contener todos los audios del juego que se deban reproducir independientemente de la posición del jugador. Es decir, todos aquellos que no tienen que ser Audio 3D, como, por ejemplo, el sonido de recogida de pociones, monedas, pisadas del jugador, ataque con el arma, muerte, música de ambiente, etc.

Cada sonido de la lista puede ser configurado para que tenga un volumen diferente y/o se pueda poner en bucle.

El *script* también se encarga de reproducir y parar cada sonido en el momento adecuado. Es decir, hay algunos audios que no se deben de superponer y otros que no se deben de reproducir indefinidamente antes de que acabe el primero.

Por último, para el sonido espacial se ha hecho uso del componente *AudioSource*, el cual se ha añadido a los enemigos, antorchas y todos los objetos que debieran tener un sonido espacial en 3D.

4.9.8 Cámaras

Para el uso de las cámaras se ha utilizado *CineMachine*.

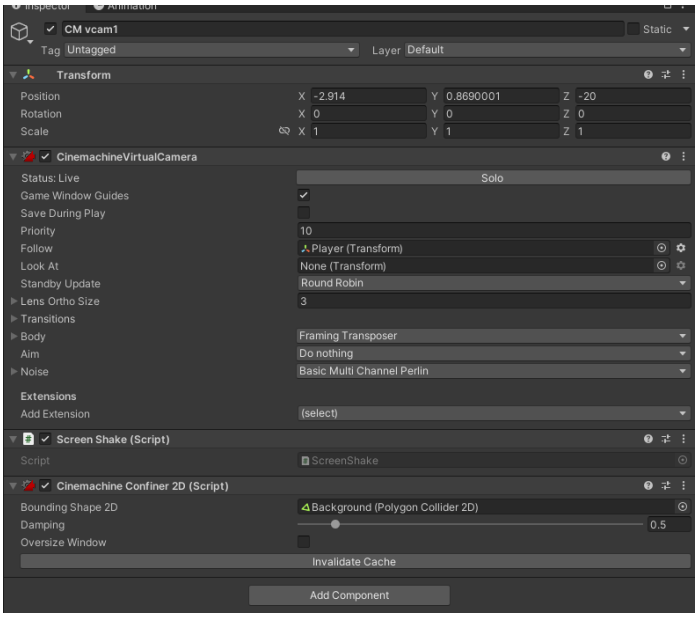

Figura 61. Sistema de cámaras de CineMachine

CineMachine es un paquete de Unity que proporciona una serie de herramientas de cámara muy robusta y flexible. Las características que ofrece este paquete y que han servido para el diseño del videojuego son las siguientes:

- **Cámaras virtuales**: Cinemachine permite cambiar fácilmente entre diferentes cámaras virtuales dependiendo de la acción del juego.
- Esto ha resultado muy útil a la hora de enfocar la atención del jugador en puntos clave.
- **Suavidad y estabilidad**: Cinemachine ofrece una variedad de opciones para suavizar el movimiento de la cámara y eliminar sacudidas no deseadas.
- ***Confiner***: El componente *Confiner* se ha utilizado para marcar los límites del juego y aquellas zonas en las que no queremos que el jugador pueda interactuar.

Por último, asociado a la cámara también se ha añadido un script *ScreenShake*, el cual consigue un efecto de temblor en la cámara cuando el jugador golpea al enemigo.

4.9.9 Menú del videojuego

4.9.10

Este es el primer menú con el que el jugador interactuará al iniciar el juego:

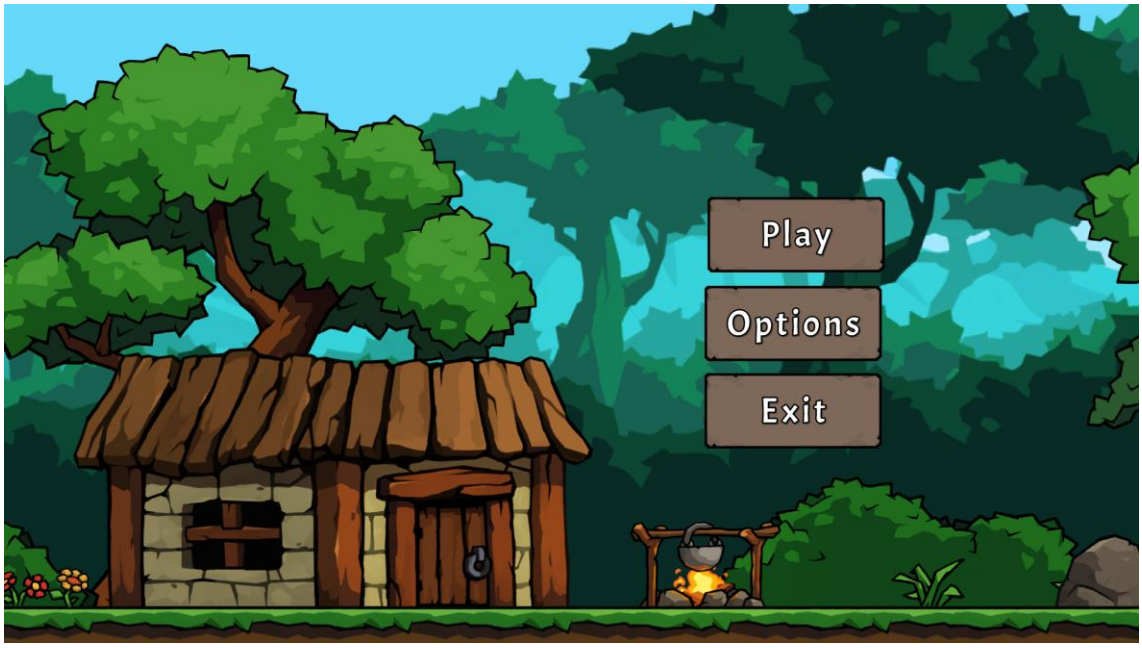

Figura 62. Menú principal del juego

- **Jugar**: Al seleccionar esta opción, el jugador comenzará el juego desde el principio o desde un punto guardado anteriormente.
- **Opciones**: Esta selección llevará al jugador al menú de opciones, donde se podrá modificar algunos ajustes como el volumen.
- **Salir**: Al elegir esta opción, el juego se cerrará.

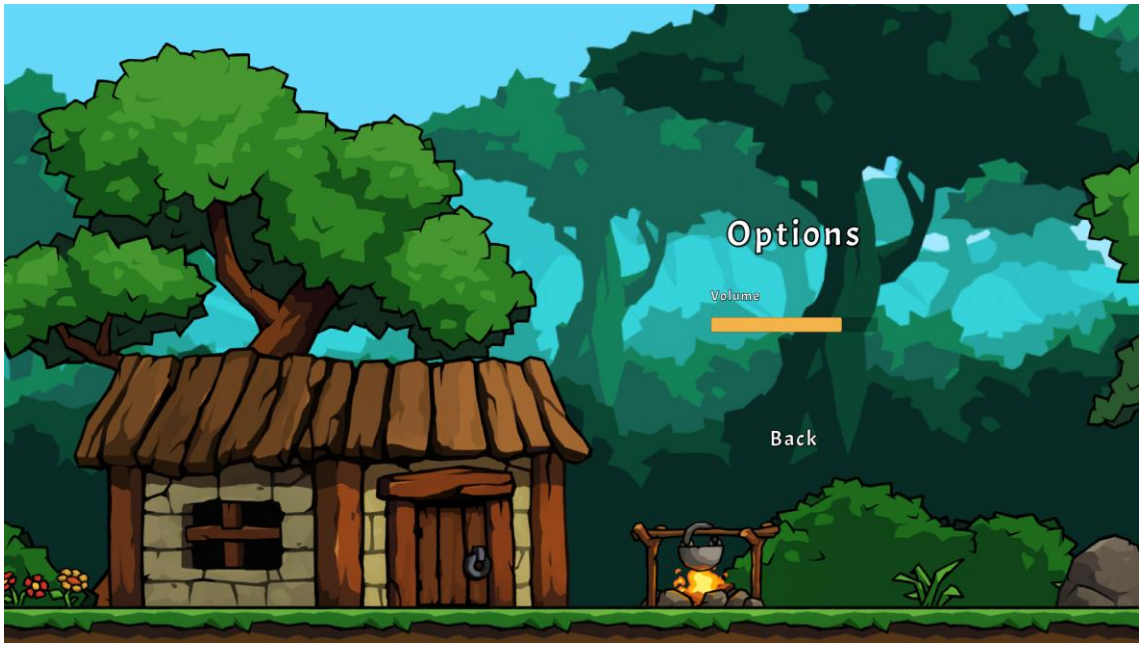

Figura 63. Menú de opciones

En este menú, los jugadores pueden personalizar diferentes aspectos del juego según sus preferencias.

- Volumen: En este menú, mediante el deslizador, los jugadores pueden ajustar el volumen del juego a un nivel cómodo.

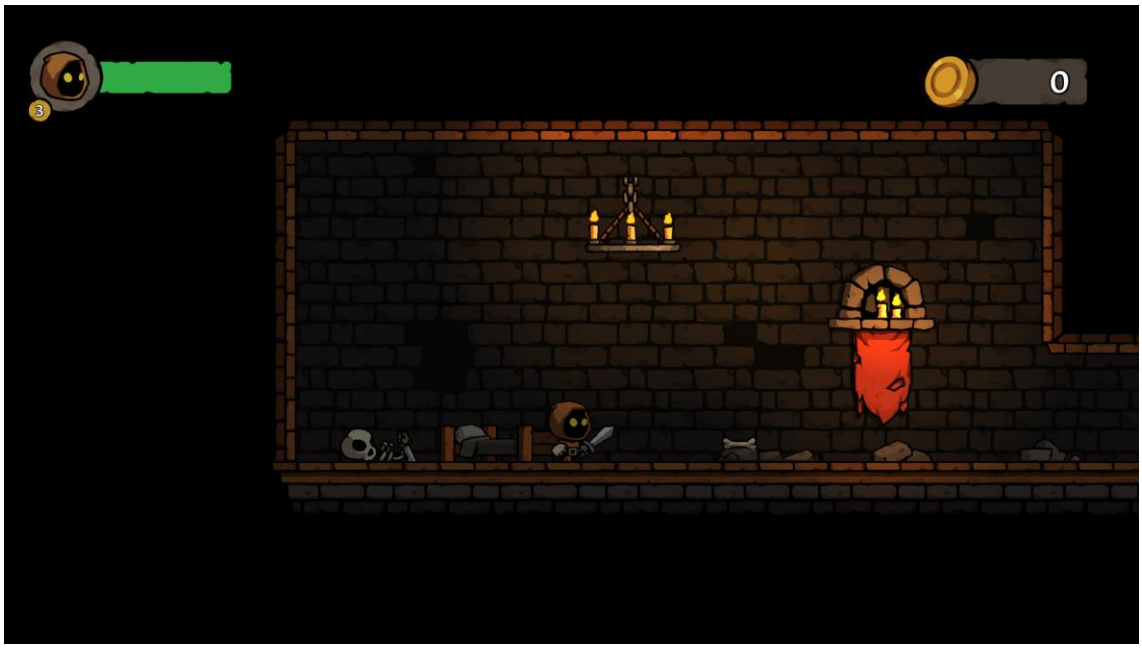

Figura 64. Interfaz de juego

En esta imagen, en la parte superior izquierda se puede ver la disposición del avatar del personaje principal, la barra de vida y las vidas que tiene actualmente.

En la parte superior derecha aparecerá el contador de monedas que haya recogido el jugador en cada nivel del juego.

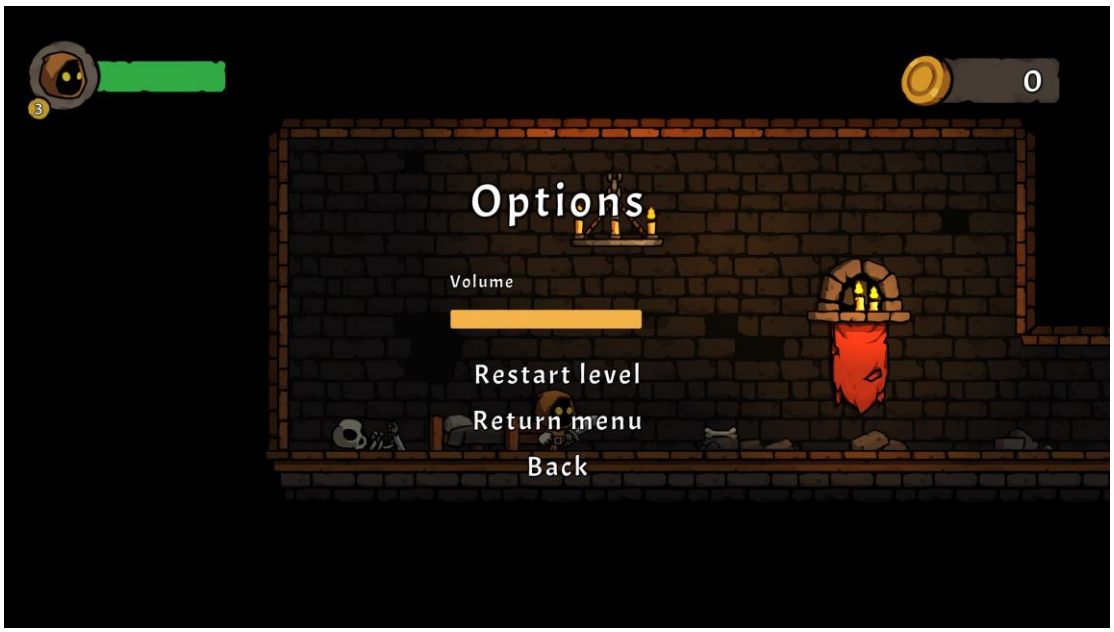

Figura 65. Interfaz de opciones

Durante el juego se puede acceder al menú de la anterior imagen. Ofrece las siguientes opciones:

- **Continuar**: Esta opción cierra el menú de pausa y regresa al jugador al juego.
- **Reiniciar nivel**: Al seleccionar esta opción, el nivel actual se reiniciará desde el principio.
- **Salir al menú**: Esta opción llevará al jugador de vuelta al 'Menú de Inicio', terminando la sesión de juego actual.

5. Implantación

5.1 Puesta en marcha

Para la puesta en marcha del videojuego desarrollado, se puede realizar directamente desde el editor.

Unity permite la creación de la *build* de un proyecto de manera muy fácil, por lo que cualquier persona puede probar el juego en su ordenador sin necesitar tener todas las aplicaciones que se han utilizado en el proyecto.

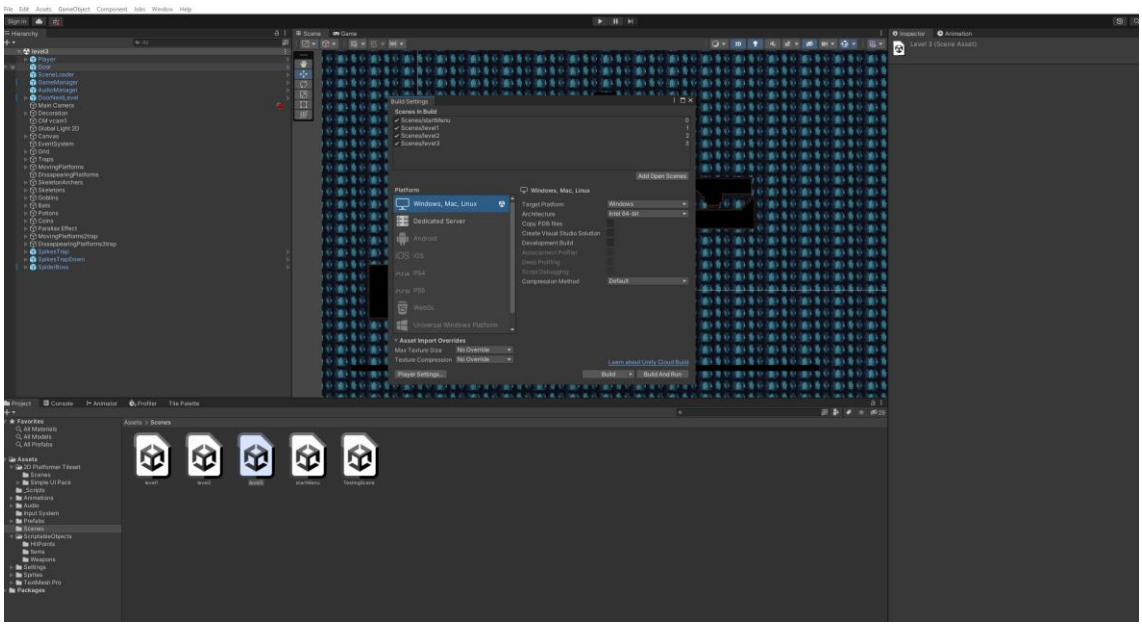

Figura 66. Exportación de build

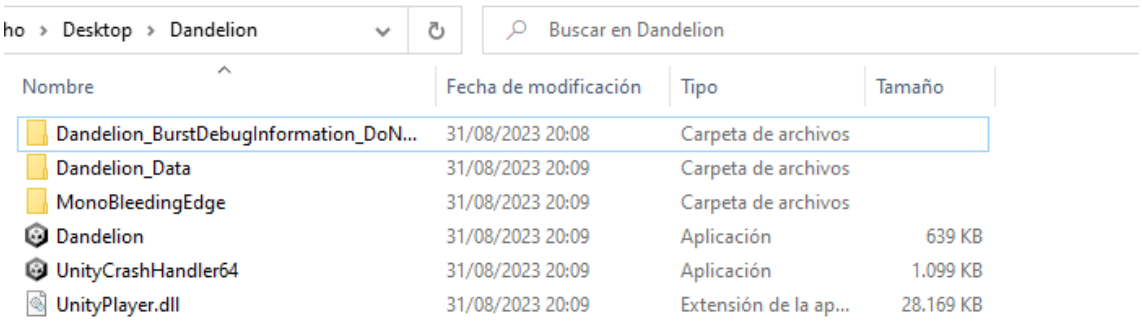

Figura 67. Build exportada del juego

Al realizar una nueva *build*, en la carpeta de la imagen anterior aparece el directorio con los ficheros necesarios para la ejecución del juego, y su ejecutable.

Por lo que, de esta manera, se puede compartir de manera sencilla a cualquier persona o en las redes como itch.io [11]

5.2 Pruebas

Se han realizado pruebas manuales, ejecutadas directamente por el desarrollador, y por personas externas que han tenido acceso al juego.

Mediante la creación de escenas específicas se ha facilitado el proceso de pruebas. En estas pruebas se han tratado aspectos como el correcto movimiento del jugador, la mecánica del salto y del doble salto, el combate y el proceso de muerte, pasando de una escena a otra y comprobando que el sistema funcionaba correctamente.

En cuanto al rendimiento, se ha utilizado el *Profiler* de *Unity* y el programa *MSI Afterburner* para comprobar las temperaturas, uso de CPU y GPU y los fps conseguidos.

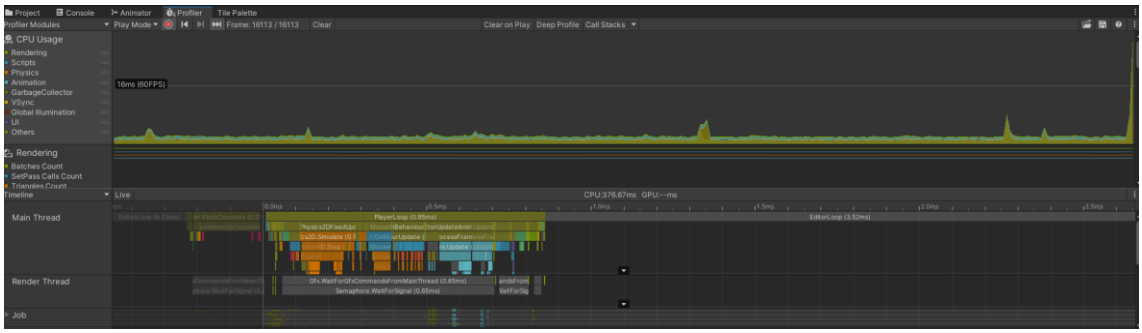

Figura 68. Datos del profiler al iniciar el juego

Se llega a una tasa de frames/s de 144 de manera continua, y llegando a 400+ desactivando la limitación a la tasa de refresco. Se comprueba también que la GPU y la CPU están a niveles bajos de % de uso. La transición entre escenas es instantánea, por lo que, en cuanto al rendimiento, los requisitos se han conseguido.

Estos datos se corresponden al siguiente ordenador:

- **CPU** – i9 9900k 4,8 GHz
- **Tarjeta gráfica** – RTX 3080 12 GB
- **RAM** – 16 G

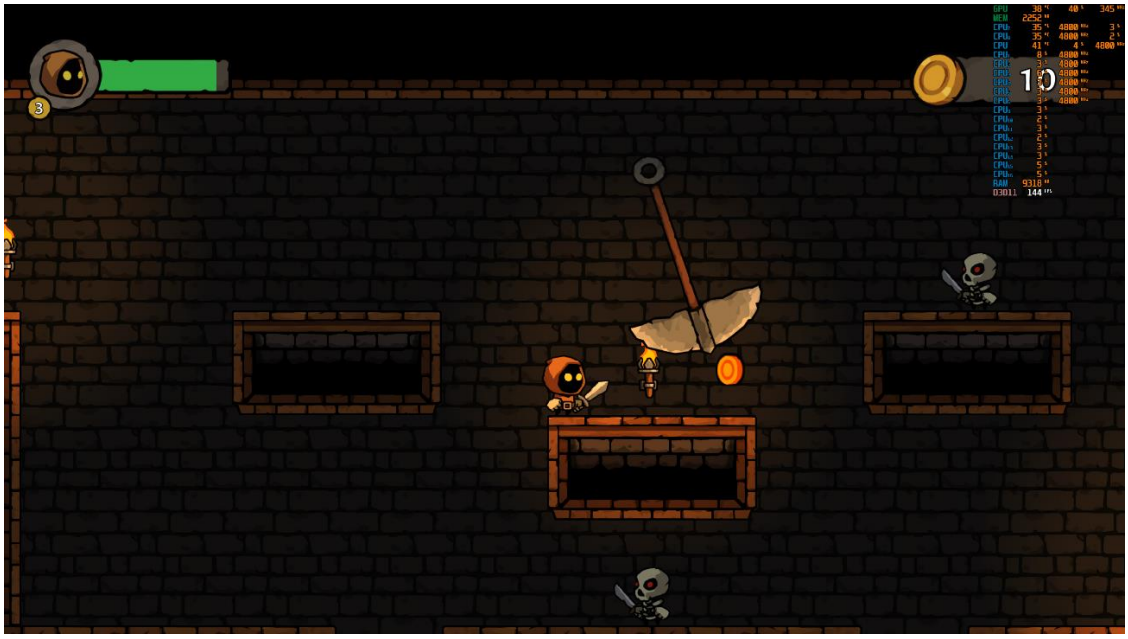

Figura 69. Control de CPU, GPU, temperaturas y FPS del nivel 1

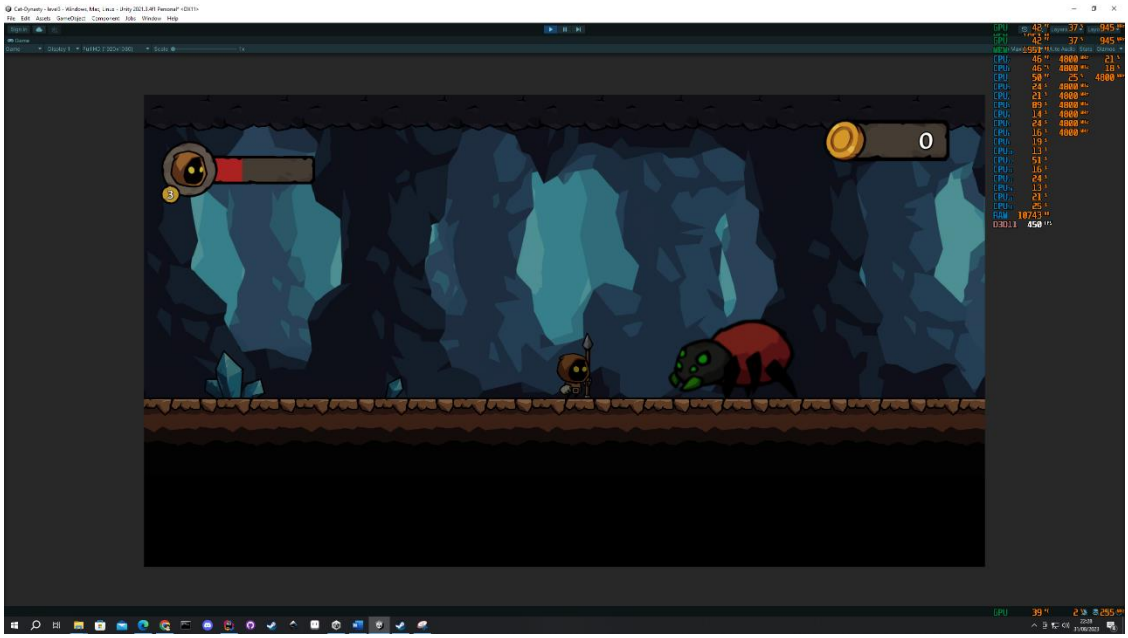

Figura 70. Control de CPU, GPU, temperaturas y FPS del nivel 3

6. Resultados

Con el fin de obtener información y *feedback* sobre el videojuego, se ha facilitado a varias personas el ejecutable de éste y se les ha enviado una encuesta sobre los puntos clave a considerar.

La encuesta ha sido completada por diez personas y sus respuestas al formulario han sido las siguientes:

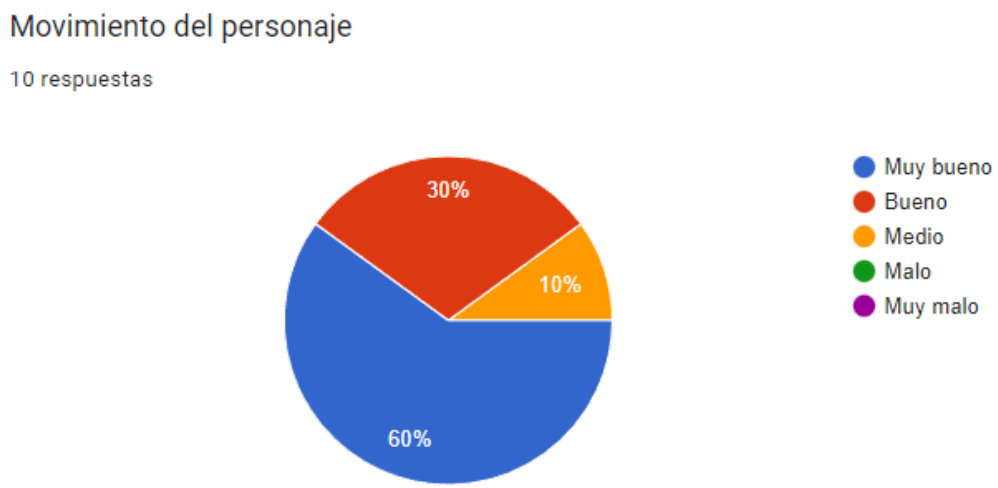

Figura 71. Encuesta del movimiento del personaje

Una de las preocupaciones principales era que el movimiento del personaje principal no fuera satisfactorio. Es una de las mecánicas clave y que más va a notar el jugador. La fluidez del movimiento, del salto entre plataformas, y el combate era necesaria para que al jugador le resultara entretenido explorar el mapa. Se le dedicó bastante tiempo en equilibrar esta parte, por lo que en general la respuesta ha sido buena.

La mayor parte de las personas que lo probaron les pareció muy bueno el comportamiento del jugador a los mandos.

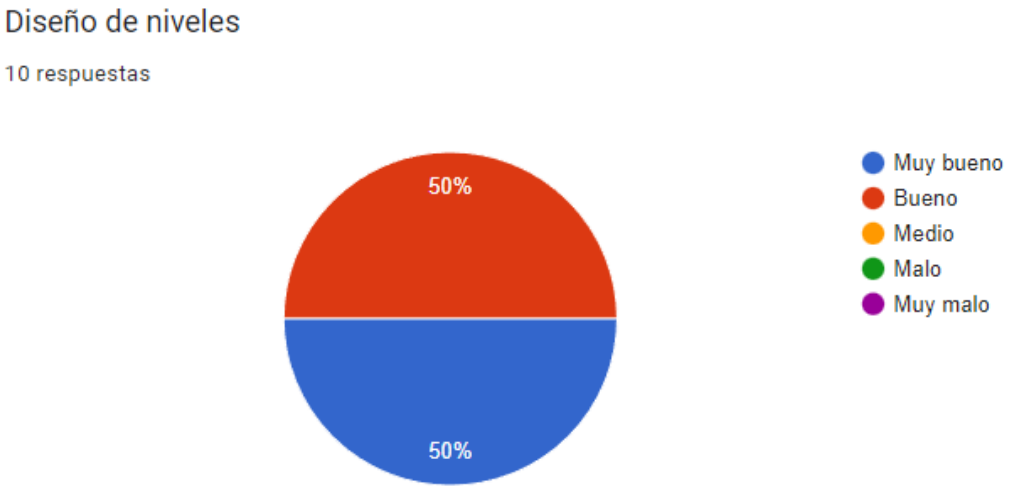

Figura 72. Encuesta del diseño de niveles

En cuanto al diseño de niveles, se barajaron dos opciones, realizarlo de manera procedural, o manual. En este caso se hizo manual para poder colocar de manera minuciosa cada enemigo, cada trampa y cada plataforma.

Esto parece que ha sido una buena decisión a la hora de ponerlo a prueba, ya que a todo el mundo le ha parecido bueno.

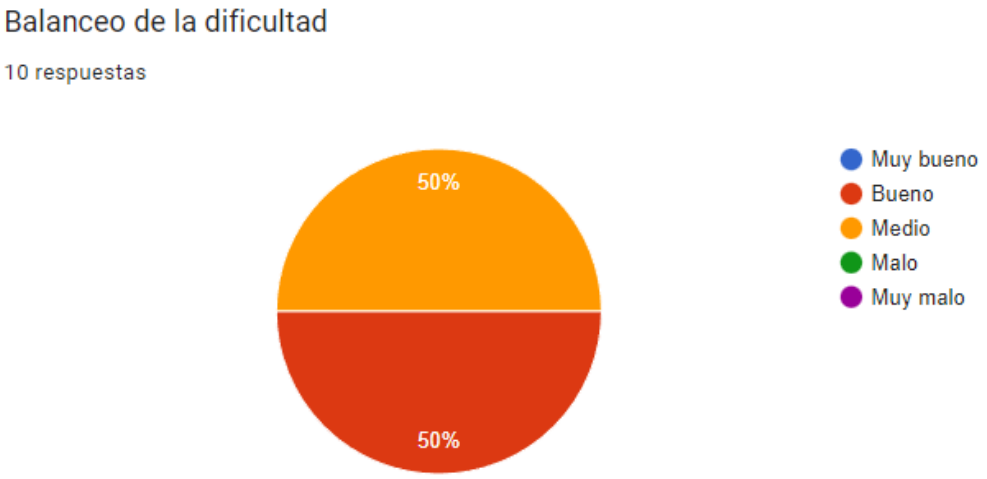

Figura 73. Encuesta del balanceo de la dificultad

El balanceo de la dificultad es donde parece que hace falta hacer algunos ajustes.

A muchas de las personas entrevistadas les pareció injusta la dificultad del nivel 2 y de algunos tramos del nivel 3. En general el nivel 1 es el más pulido en este aspecto.

A futuro no se descarta equilibrar la curva de aprendizaje de los dos últimos niveles.

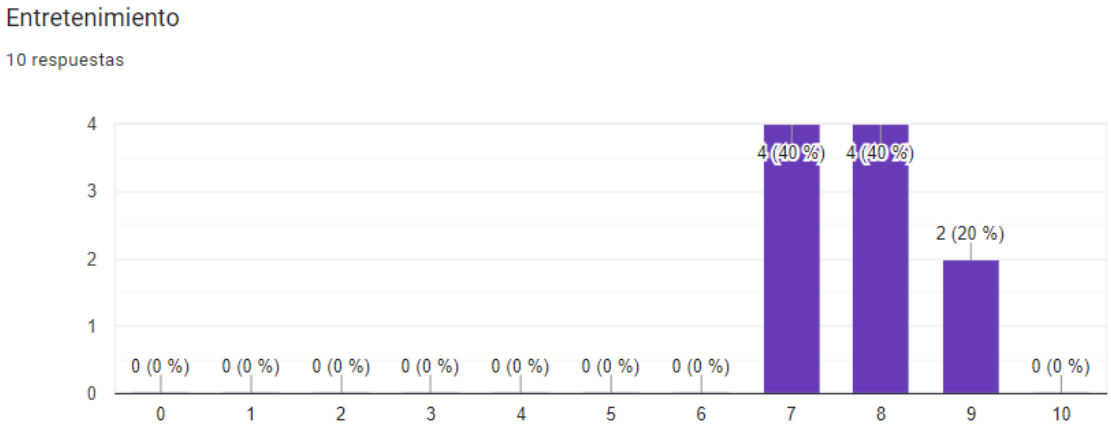

Figura 74. Encuesta del entretenimiento

Al fin y al cabo, lo que más se valora en un videojuego es el entretenimiento. En este caso, pese a las malas valoraciones sobre la dificultad, en general les ha parecido divertido y desafiante.

7. Conclusiones

En este trabajo de fin de Máster se ha realizado la creación de un videojuego funcional, entretenido y desafiante.

Se ha conseguido completar todos los objetivos que se estimaron inicialmente por lo que estoy satisfecho con el trabajo realizado.

Los niveles, al estar hechos a medida han sido bastante complejos de realizar por lo que de seguir sería adecuado buscar una forma óptima de realizarlos.

La creación de las IA de los jefes finales ha sido todo un reto y se ha aprendido mucho con el desarrollo.

Respecto a Unity como herramienta elegida ha sido todo un acierto, gracias a la gran cantidad de material y de tutoriales se han resuelto todas las dudas que surgían.

Cabe destacar la metodología de desarrollo utilizada y el modelo de *sprints*, pues me ha permitido desarrollar el producto mínimo viable poco a poco, lo que ha permitido una buena organización y una motivación constante.

Esta metodología, así como el uso de herramientas hasta ahora desconocidas para mi me han ayudado a aprender conceptos nuevos que voy a poder aplicar en mi trabajo en un futuro cercano.

También cabe destacar que las reuniones semanales con mi tutor han sido muy fructíferas y ha facilitado mucho tanto el desarrollo como el aprendizaje de este trabajo.

8. Bibliografía

[1] Buttfield-Addison, Paris, Jonathon Manning, y Tim Nugent. Unity Game Development Cookbook: Essentials for Every Game. Sebastopol, CA, 2019.

[2] «Unity Asset Store - The Best Assets for Game Making». Accedido 7 de enero de 2023. https://assetstore.unity.com/.

[3] Technologies, Unity. «5 Unity Tutorials for New Game Developers». Accedido 8 de Enero de 2023. https://unity.com/how-to/beginner/5-unity-tutorials-new-game-developers.

[4] OpenWebinars.net. «Guía rápida para aprender Scrum», 20 de febrero de 2018. https://openwebinars.net/blog/la-guia-para-aprender-scrum/.

[5] JetBrains. «Rider: El IDE .NET multiplataforma de JetBrains». Accedido 5 de febrero de 2023. https://www.jetbrains.com/es-es/rider/.

[6] JetBrains Rider Help. «Keyboard Shortcuts | JetBrains Rider». Accedido 10 de febrero de 2023. https://www.jetbrains.com/help/rider/mastering_keyboard_shortcuts.html.

[7] How to make a BOSS in Unity!, 2020. https://www.youtube.com/watch?v=AD4JIXQDw0s.

[8] Creando un BOSS de HOLLOW KNIGHT en UNITY, 2022. https://www.youtube.com/watch?v=I6LxjKvI3C8.

[9] GitHub Docs. «Configuring Git Large File Storage». Accedido 2 de marzo de 2023. https://ghdocs-prod.azurewebsites.net/en/repositories/working-with-files/managing-large-files/configuring-git-large-file-storage.

[10] HacknPlan. «Project Management for Game Development». Accedido 3 de marzo de 2023. https://hacknplan.com/.

[11] itch.io. «Download the Latest Indie Games». Accedido 4 de marzo de 2023. https://itch.io/. [12] Semantic UI: User Interface is the language of the Web. <https://semantic-ui.com/>

[13] «2D Platformer Tileset | 2D Environments | Unity Asset Store». Accedido 5 de marzo de 2023. https://assetstore.unity.com/packages/2d/environments/2d-platformer-tileset-173155.

[14] «Unity Animator Tutorial - Comprehensive Animation Guide - GameDev Academy», 19 de diciembre de 2022. https://gamedevacademy.org/unity-animator-tutorial/. [15] The World’s Largest Web Developer Site: Ajax Introduction.

[15] «Education Ecosystem». Accedido 15 de marzo de 2023. https://educationecosystem.com/ramondev/RYrEr-how-to-create-boss-fight-mechanics-in-unity.

[16] Sauco, Alex, y Eduardo Lozano. El Viaje del Jugador: Guía de Diseño de Videojuegos, 2021.

[17] «El Sprint En Scrum», 7 de febrero de 2021. https://www.scrumio.com/scrum/el-sprint/.

9. Anexos

Anexo 1: Documento de diseño del videojuego

El Documento de Diseño del Videojuego (GDD) es un documento vivo que detalla todos los aspectos de un videojuego. Es fundamentalmente un documento de referencia para el equipo de desarrollo que describe todos los elementos y las decisiones relacionadas con el diseño del videojuego.

Anexo II: Objetivos de desarrollo sostenible

Los objetivos de desarrollo sostenible, también conocidos como Objetivos Globales, fueron adoptados por las Naciones Unidas en 2015 como un llamamiento universal para poner fin a la pobreza, proteger el planeta y garantizar que para el 2030 todas las personas disfruten de paz y prosperidad.

ANEXO I: Documento de diseño de videojuegos

Cat Dynasty
Game Design Document

Ignacio José Serra Almenar

1 Visión General

1.1 Resumen ejecutivo

Concepto general

Juego de plataformas en el cual debes de escapar de una mazmorra llena de enemigos y trampas.

Metáfora

Sobrevive a diferentes enemigos mientras te abres paso hacia la superficie, subiendo pisos de la mazmorra en la que te han encarcelado.

Sinopsis

El reino de Bevurin y Drozatis van a firmar un tratado de paz, uniendo en matrimonio a sus dos herederos al trono. En el banquete de bodas, el reino de Drozatis traiciona el tratado de paz y encarcela al primogénito de Bevurin.

Dandelion deberá escapar del calabozo al que ha sido enviado y vengarse de aquellos que han sido los precursores de esta traición.

Alcance

El proyecto tendrá 3 niveles, enemigos y trampas. El primer nivel es de fácil acceso y se muestran las mecánicas del juego. El segundo y tercer nivel muestran un desafío de plataformas y enemigos que retará al jugador más experimentado.

Estilo visual

Diseño 2D dibujado a mano de manera digital. Estilo *cartoon* y medieval.

Motor

Uso del monitor Unity.

Jugabilidad

El personaje principal tendrá los siguientes movimientos:

- Movimiento lateral izquierda y derecha [3].
- Avance rápido o *dash* que le permitirá esquivar.
- Salto y doble salto
- Escalada con medidor de resistencia
- Ataque básico y especial dependiendo del arma
- Bloqueo y contraataque

El estilo de juego será rápido tanto por el diseño de nivel como por el combate contra los enemigos. Estará basado en la habilidad, los enemigos tendrán poca vida, pero podrán realizar mucho daño si no se esquivan sus ataques.

Cuando se derrota a un jefe final de nivel, el protagonista recibe la habilidad especial y arma del enemigo.

Público objetivo

Jóvenes y adolescentes mayores de 14 años, sin conflictos con las descripciones pegi 12

Plataformas

Principalmente para **PC** (Windows). Posteriormente se evaluará la opción para móviles ***Android***.

Se evaluará también el rendimiento en la portátil ***Steamdeck***

2 Historia, entorno y personajes

2.1 Historia

La historia tiene lugar en Catheim, un continente dividido en dos grandes naciones: Bevurin y Drozatis. Las dos han vivido una tensa paz desde hace 20 años, tras una guerra por el control del recurso más preciado: los cristales de poder.

Con el fin de zanjar esta tensión, se concierta un matrimonio entre el protagonista, heredero al trono de Bevurin, Dandelion y la hija del actual archiduque de Drozatis.

En la celebración de dicho acontecimiento, el ducado de Drozatis traiciona la nación de Bevurin asesinando a los miembros del banquete y arrestando al protagonista, el cual es llevado a una prisión de máxima seguridad.

Con toda la determinación que le queda, el protagonista intentará por todos los medios escapar de esta prisión con el fin de avisar a sus aliados y obtener justicia, por su familia, amigos y prometida.

2.2 Entornos

Estética medieval, poblados, castillos y los diferentes niveles de la prisión de Drozatis, donde transcurrirá la mayor parte del juego.

2.3 Personajes

Dandelion

El protagonista del juego. Heredero al trono de Bevurin.

Solo quiere lo mejor para su pueblo, diestro con la espada, lanza y arco.

Ninev

Hija del archiduque de Drozatis. Está prometida con el protagonista. No está de acuerdo con la política de su padre ni con la del Rey de Bevurin.

Mowing

General del ejército del Archiduque.

La falta de una mano no le ha impedido dirigir y entrenar el ejército del ducado con dureza.

Es implacable, firme y leal.

Lanvilot

El mejor amigo y guardaespaldas del príncipe (protagonista), se conocen desde que eran pequeños. Han entrenado y han ido a la escuela juntos.

3 Combate

El personaje cuenta con la posibilidad de atacar con una espada, lanza y arco.

Espada

La espada es la primera arma que consigue el protagonista, es un arma estándar rápida y de cuerpo a cuerpo

Alcance	Daño	Velocidad
Corto	Bajo	Alta

Lanza

La lanza, gracias a su alcance es capaz de atacar a un grupo mayor de enemigos, sin embargo, hay que controlar la velocidad, ya que dejará expuesto al jugador.

Alcance	Daño	Velocidad
Medio	Medio	Bajo

Arco

Arma de largo alcance y de velocidad lenta, sirve para derrotar enemigos desde largas distancias

Alcance	Daño	Velocidad
Largo	Bajo	Bajo

4 Controles

Los controles en ordenador se han asignado de manera que se puedan utilizar tanto mando como teclado y ratón

4.1 Teclado y ratón

Salto/Doble Salto	Movimiento horizontal	Mirar arriba/abajo	Esquivar	Ataque básico	Ataque fuerte	Cambio de arma
Espacio	A/D	W/S	Shift	Mouse L	Mouse R	X

4.2 Mando

Basado en el mando de Xbox.

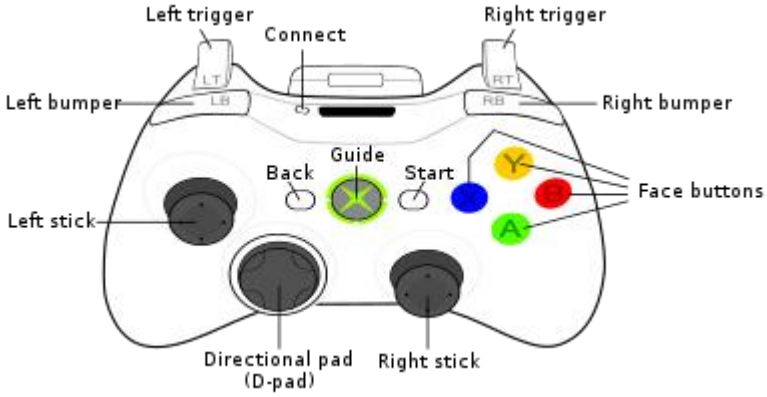

Figura 75. Estructura básica del mando de Xbox

Salto/Doble Salto	Movimiento horizontal	Mirar arriba/abajo	Esquivar	Ataque básico	Ataque fuerte	Cambio de arma
A	D-pad	D-pad	O	LT	RT	Y

5 Interfaz

5.1 Diagrama de flujo

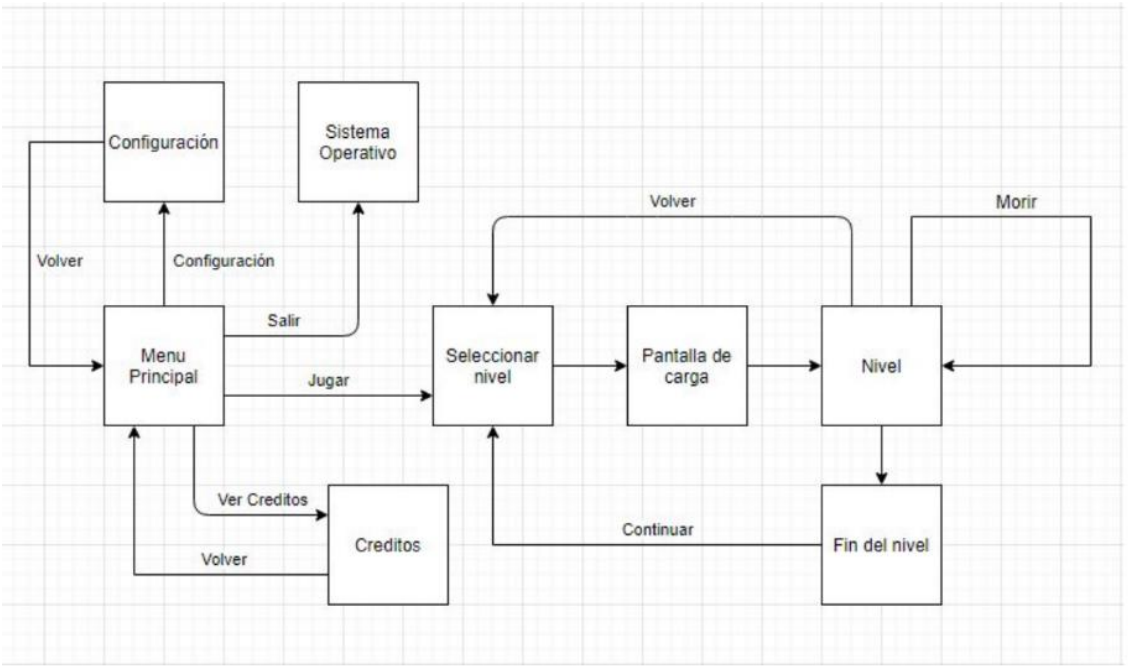

Figura 76. Diagrama de flujo del juego

En el diagrama podemos comprobar todo el flujo a realizar desde que ejecutamos la aplicación en el ordenador, pasando por el menú principal, la elección de nivel, el final, y la pantalla de créditos.

5.2 Menú principal

A continuación, se muestran los bocetos para el menú principal

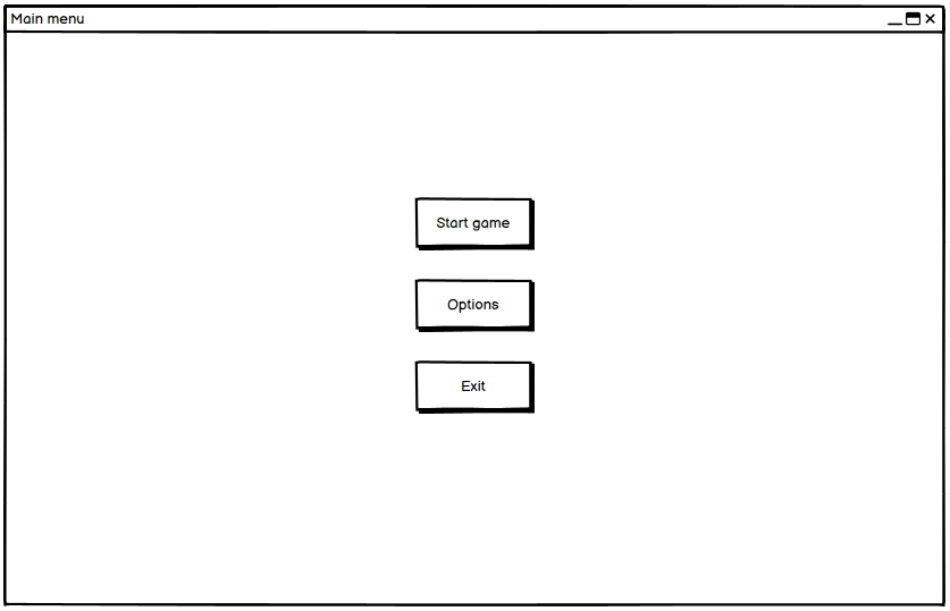

Figura 77. Mockup del menú principal

Lista y descripción de componentes:

- **Start game***:* al pulsarlo lleva a la pantalla del primer nivel
- **Options**: al pulsarlo lleva a la pantalla de opciones
- **Exit**: al pulsarlo cierra la aplicación

5.3 Configuración

Boceto para el menú de opciones:

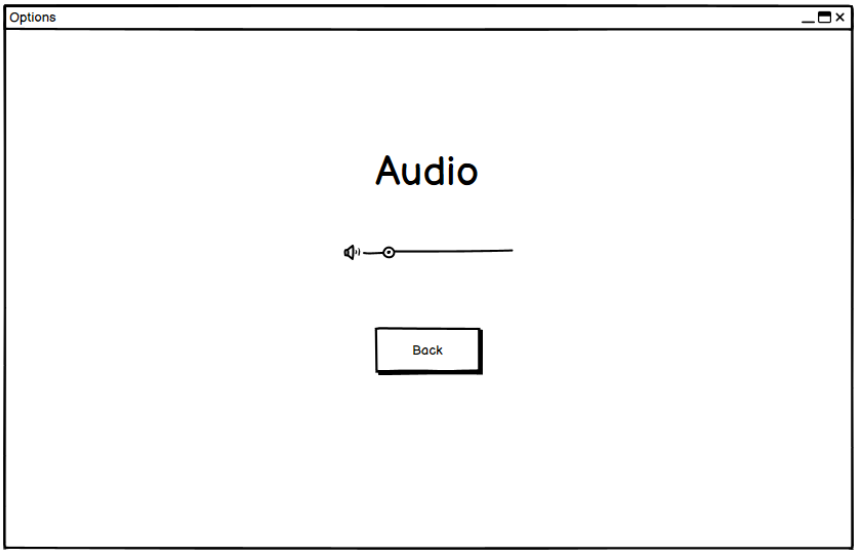

Figura 78. Mockup de opciones

- **Audio**: al desplazar el *slider* permite subir o bajar el volumen

5.4 Nivel

Boceto para el nivel jugado y el mapa del escenario:

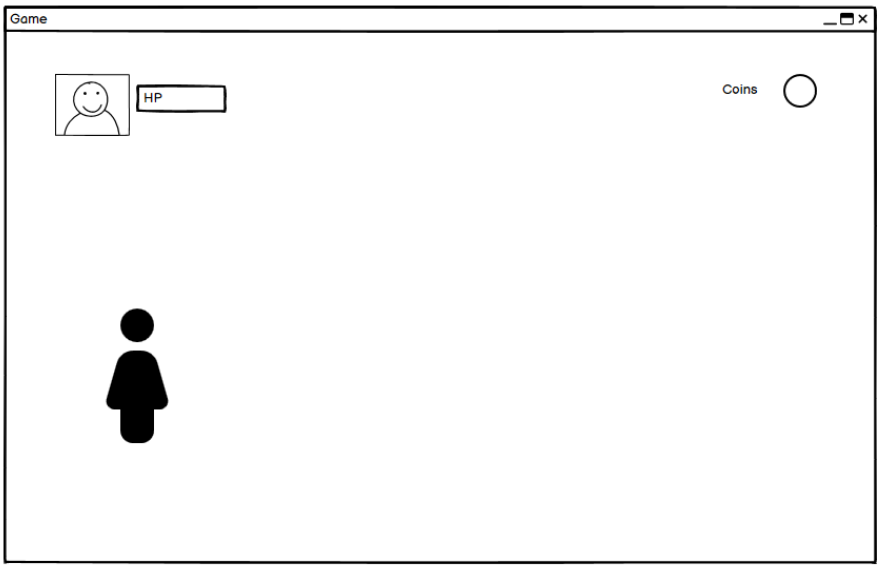

Figura 79. Mockup del juego

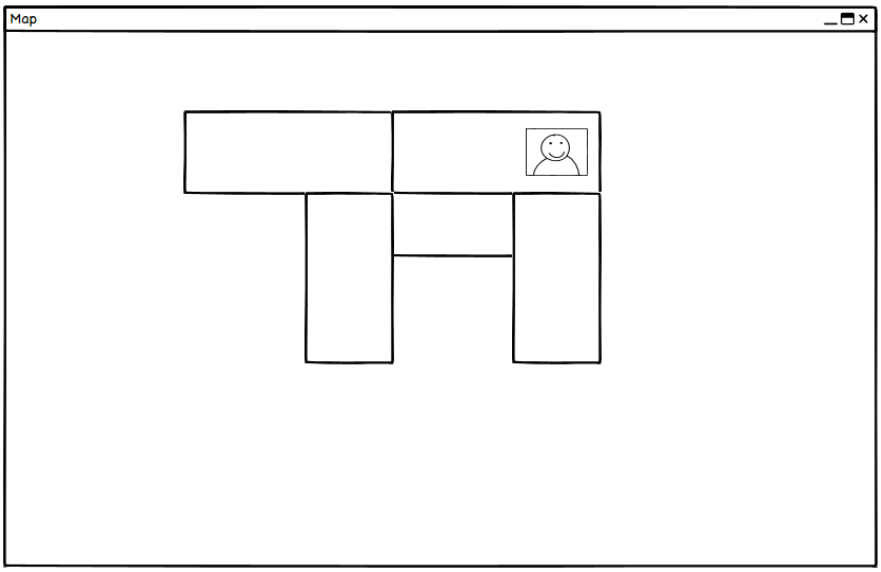

Figura 80. Mockup del mapa

En el menú de juego, se colocará en la parte superior izquierda la vida del jugador, y en la parte superior derecha el número de monedas obtenidas.

El menú del mapa permitirá al jugador saber dónde se encuentra en la mazmorra.

6 Inteligencia artificial

En esta sección se presenta el comportamiento de todos los enemigos, tanto los enemigos básicos como los jefes finales.

6.1 Enemigos normales

6.1.1 Esqueleto

El enemigo de tipo esqueleto permanece inmóvil hasta que su campo de visión detecta al jugador.

Cuando esto ocurre el enemigo procede a seguir al jugador y cuando lo tiene a cierta distancia, ataca con su espada.

Daño	Vida	Alcance	Movimiento
Bajo	Media	Bajo	Bajo

Figura 81. Esqueleto

6.1.2 Murciélago

Enemigo fijo en el mapa, ataca cuando avista al jugador. Tiene daño y vida baja.

Daño	Vida	Alcance	Movimiento
Bajo	Bajo	Bajo	Bajo

Figura 82. Murciélago

6.1.3 Esqueleto arquero

Enemigo fijo en el mapa, ataca a distancia. Tiene daño medio y vida baja

Daño	Vida	Alcance	Movimiento
Bajo	Bajo	Alto	-

.

Figura 83. Esqueleto arquero

6.1.4 Goblin

Enemigo que patrulla, ataca a corta distancia, velocidad de movimiento alta. Tiene daño medio y vida alta.

Daño	Vida	Alcance	Movimiento
Medio	Media	Bajo	Medio

Figura 84. Goblin

6.2 Jefes

6.2.1 Jefe esqueleto

El enemigo permanece estático hasta que el jugador entra en la sala.

A continuación, perseguirá al jugador y cuando esté lo suficientemente cerca, ejecutará de manera aleatoria un golpe con el hacha en arco, o una estocada.

Al llegar al 50% de su vida, el jefe enfurece y aumenta su velocidad, daño y tiempo de espera entre ataques.

Tabla en estado normal

Daño	Vida	Alcance	Movimiento
Alto	Alta	Bajo	Medio

Tabla en estado de furia

Daño	Vida	Alcance	Movimiento
Muy alto	Media	Medio	Alto

Figura 85. Esqueleto jefe

6.2.2 Esqueleto mago

Jefe del segundo nivel. Lanza una bola de fuego que persigue al jugador y una lluvia de meteoritos que caen por el escenario.

Daño	Vida	Alcance	Movimiento
Medio	Media	Alto	Medio

Figura 86. Esqueleto mago

6.2.3 Araña gigante

Es muy veloz, efectúa un mordisco a corta distancia y lanza una tela de araña a larga distancia.

Daño	Vida	Alcance	Movimiento
Alto	Alto	Alto	Muy alto

Figura 87. Araña gigante

7 Niveles

Nivel 1 – Prisión

Figura 88. Nivel 1 - Prisión

El primer nivel está pensado como nivel introductorio. Poco a poco se introducen las mecánicas necesarias para avanzar. Haciendo necesario el uso del salto y del doble salto para poder subir por los primeros compases del juego. Aparecen los primeros enemigos más fáciles, el esqueleto y el murciélago para que el jugador se acostumbre a ellos y aprenda las mecánicas de combate.

Hay dos zonas secretas que esconden monedas y pociones, pero también trampas que el jugador debe de sortear. Aparece la mecánica de escalar, donde el jugador deberá esquivar una serie de trampas para poder llegar hasta el jefe final, que le espera en la última sala.

Nivel 2 – Mazmorra

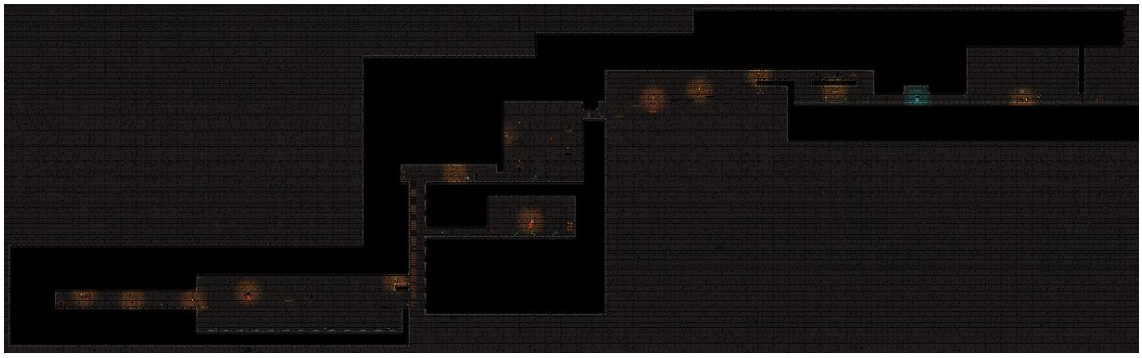

Figura 89. Nivel 2 - Mazmorra

En el segundo nivel la dificultad se incrementará. La primera sección consistirá en unas plataformas móviles que el jugador debe de sortear sin caerse, pues hay pinchos que provocarán

la muerte instantánea. Además, se añadirá el primer enemigo que ataca a largo alcance, el arquero esqueleto.

Aunque sea un enemigo débil, si el flechazo impacta sobre el jugador, lo empujará y deberá de estar atento de no caerse. En el nivel también se implementará un nuevo objeto, la palanca. Al ser golpeada, activa un temporizador y abre una puerta. Si no se pasa en el tiempo indicado, la puerta se cerrará y el jugador deberá de volver a empezar la sección.

Por último, antes del jefe final, aparecerá las plataformas invisibles, que el jugador deberá cronometrar para saber cuándo debe de saltar.

En la sala final, estará esperando el mago esqueleto, el cual, con su muerte, el jugador recibirá el arco.

Nivel 3 – Cueva

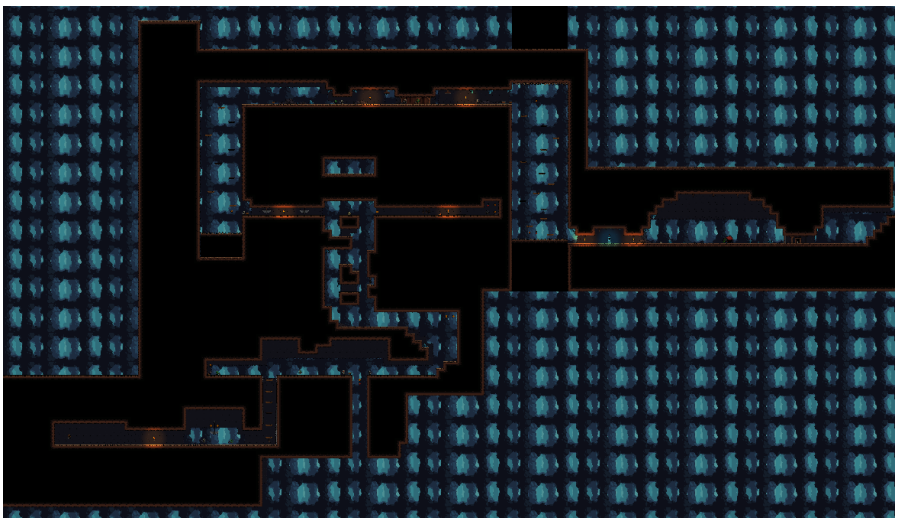

Figura 90. Nivel 3 - Cueva

Como último nivel, tendremos la cueva, el jugador estará a punto de salir a la superficie, pero antes deberá de superar este nuevo reto. Se pondrán en juego todo lo aprendido en los anteriores niveles. Se añadirá un nuevo enemigo, el goblin, más fuerte que los enemigos básicos anteriores. Se implementarán trampas más difíciles como terreno con pinchos que se mueven verticalmente y que el jugador deberá de sortear.

En la sección final se encontrará con la araña gigante y cuando la derrote, el jugador saldrá a la superficie y se habrá pasado el juego.

8 Animaciones

Reposo

Figura 91. Animación de reposo del jugador

Correr

Figura 92. Animación de correr del jugador

Saltar

Figura 93. Animación de saltar del jugador

Escalar

Figura 94. Animación de escalar del jugador

Atacar

Figura 95. Animación de atacar del jugador

Morir

Figura 96. Animación de muerte del jugador

9 Puntuación, trampas y bonificaciones

Cada enemigo derrotado generará un número de puntos que posteriormente se verá reflejada en la puntuación final del escenario. Las monedas encontradas por el mapa también servirán añadirán puntos.

Tipo	Puntos
Monedas	10
Esqueleto	20
Murciélago	10
Arquero esqueleto	15
Goblin	30
Esqueleto jefe	100
Mago esqueleto	110
Araña	120
No recibir daño	x2 de la puntuación recibida

A parte de estos puntos, el jugador será premiado dependiendo de a la velocidad que se pase el nivel.

Las trampas que el jugador se podrá encontrar tendrán esta tabla de daños

Trampa	Daño
Pinchos laterales	20
Pinchos en foso	100
Guadaña giratoria	10
Pincho vertical con movimiento	15
Fuego	20

10 Lista de recursos

10.1 Arte

Dandelion quiere transmitir un carácter alegre con su modo *cartoon* pero a la vez tenso y con una esencia medieval y mágica. El dibujo a mano entra por los ojos y resulta agradable a la vista.

Se ha utilizado un asset 2D de la tienda de Unity que satisfacía los requisitos.

Figura 97. Asset del arte del juego

Para el diseño de escenarios se han usado diferentes *tiles* proporcionadas por el *asset 2D*.

Figura 98. Spritesheet de elementos decorativos

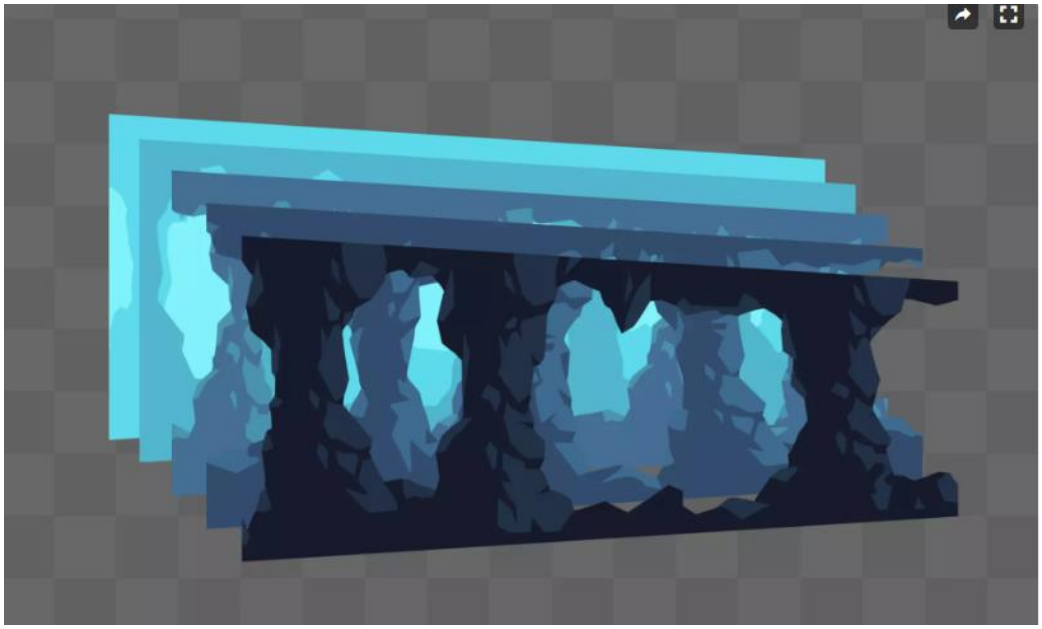

Figura 99. Capas parallax del fondo de la cueva

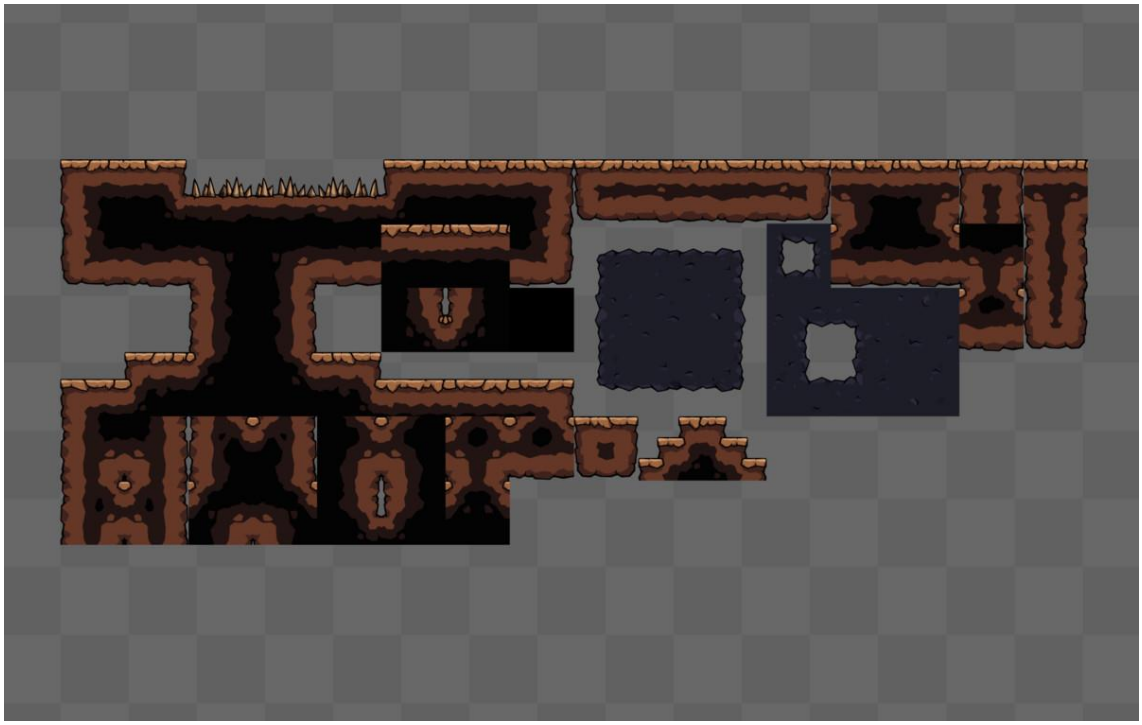

Figura 100. Tilesets de la cueva

Este *asset* ha permitido desarrollar escenarios muy parecidos a los que mostraba en la tienda.

10.2 Música

- **Menú principal**: Música de aventura y tensión, pero más relajada que la correspondiente a los niveles.
- **Juego**: Música ambientada en un paso subterráneo, de mazmorra y prisión. Lúgubre pero con toques de aventura. Cada jefe final tendrá su propia banda sonora épica.
- **Victoria**: Cuando el jugado vence a un jefe final, se reproducirá una música de victoria para que se sienta recompensado.
- **Game Over**: Al morir, se reproducirá una música triste.

Todas descargadas de la web https://freesound.org/

10.3 Efectos

- **Pisadas**: Las pisadas del personaje principal y de los enemigos serán reproducidas con un efecto.
- **Golpe**: Cada golpe tendrá variaciones de sonido.
- **Daño**: Al recibir daño, el jugador emitirá un sonido de queja, al igual que los enemigos.
- **Muerte**: El personaje emitirá un sonido de último suspiro al morir.
- **Salto**: Tanto el sato como la caída tendrán un sonido característico.

11 Resumen técnico

11.1 Requisitos mínimos

- Sistema Operativo: Windows 10
- Procesador: Dual Core CPU
- Memoria: 4 GB RAM
- Tarjeta gráfica: OpenGL 4.0 con 2GB de video RAM.
- Disco duro: 2 GB espacio disponible

11.2 Requisitos recomendados

- Sistema Operativo: Windows 10
- Procesador: Procesador Intel i5-6900 o superior
- Memoria: 8 GB RAM
- Tarjeta gráfica: GeForce 960 o superior
- Disco duro: 4 GB espacio disponible

12 Referencias

Hollow Knight

Metroidvania que destacó particularmente por su estética atractiva y mundo misterioso. La manera en la que combina los combates, historia y estética han servido de referencia para el desarrollo.

Blasphemous

El brutal combate que presenta este juego, como las mecánicas de los enemigos han servido de ayuda para tomar ideas en el desarrollo.

ANEXO II: 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 TFM con los ODS y con los ODS más relacionados.

De los anteriores objetivos de desarrollo sostenibles mencionados, el proyecto está relacionado con:

- **Salud y bienestar**: El uso de videojuegos de forma moderada está ligado a beneficios como la disminución de ansiedad, depresión e incrementos de autoestima, es por ello que considero que el proyecto aporta en este ámbito.
- **Producción y consumo responsables**: Siendo un producto de tipo digital no se consumirán recursos propios de la producción de ediciones físicas o coleccionistas y por tanto no contribuye a la tala de árboles o procesamiento medioambiental de plástico
- **Trabajo decente y crecimiento económico**: Al tener que buscar financiación para fundar un estudio de videojuegos, considero que crearía industria en el país que aportaría trabajo a personas, por lo que habría un crecimiento