UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

The last hanyou, videojuego metroidvania desarrollado con Unity

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Font Gil, Erik

Tutor/a: Carsí Cubel, José Ángel

CURSO ACADÉMICO: 2021/2022

Agradecimientos

A mi tutor José Ángel, por ayudarme a realizar mi Trabajo Fin de Grado soñado.

A mi familia, por todo el amor, apoyo y paciencia a lo largo de la carrera.

A mis amigos, por desestresarme en las etapas duras.

A mi hermano Christian, porque sin él esto no habría sido posible.

Muchas gracias.

Resumen

The last hanyou es un videojuego 2D de género metroidvania con elementos RPG desarrollado con Unity y escrito con el lenguaje de programación C# para la plataforma de *Windows*, cuya finalidad es la creación de un producto mínimo viable que en un futuro pueda ser vendido en tiendas digitales de videojuegos.

En este Trabajo Final de Grado se abarcará la creación de las diferentes características que son parte de un metroidvania como el sistema de inventario, guardado y carga de datos, sistema de mapas, objetos que puedan tanto equiparse como consumirse, fabricación de objetos por medio de materiales, creación de un árbol de diálogos para crear la narrativa del videojuego, enemigos, combate y movimiento del jugador fluido.

Palabras clave: videojuego, metroidvania, unity, 2d, rpg.

Abstract

The last hanyou is a 2D metroidvania RPG video game developed with Unity and written with the C# programming language for the Windows platform, whose purpose is to create a minimum viable product that in the future it can be sold in digital game stores.

This Final Degree Project will cover the creation of the different characteristics that are part of a metroidvania such as the inventory system, saving and loading data, map system, objects that can be equipped as well as being consumed, manufacture of objects by means of materials, creation of a dialogue tree to create the narrative of the video game, enemies, combat and fluid player movement.

Keywords: videogame, metroidvania, unity, 2d, rpg.

Índice general

Índice de imágenes

Índice de tablas

1. Introducción

1.1 Motivación

En la última década, la industria de los videojuegos se ha convertido en uno de los sectores más prolíferos del mercado que no ha hecho sino dispararse una vez comenzó la pandemia.

Este sector no solo se centra en el entretenimiento, sino que también se propone la idea de los *serious games* que son juegos educativos donde aprendemos a la vez que nos divertimos, por esto podemos entender el gran crecimiento, ya que las empresas cada vez tienen un interés mayor de introducirse en el mercado, puesto que estos son utilizados desde los más pequeños hasta los más mayores.

Por mis antecedentes como jugador de videojuegos desde mi más temprana edad, de donde vino mi inquietud por aprender a programar, se me hace muy atractivo e interesante entrar en el sector del desarrollo de videojuegos con este Trabajo Fin de Grado, por el hecho de que es una oportunidad perfecta para aprender nuevas habilidades y herramientas que pueden servirme en mi futuro laboral.

1.2 Objetivos

Los objetivos por conseguir son:

La creación de un MVP de un videojuego *metroidvania* con elementos de rol que en un futuro pueda ser vendido en plataformas digitales de venta de videojuegos como lo son Steam, GOG o itch.io.

La puesta en práctica de los conocimientos obtenidos durante el grado, como, por ejemplo, la programación orientada a objetos con la asignatura de “ingeniería del software” o la implementación de patrones de diseño con la asignatura de “diseño de software”.

Por último, se pretende aprender conocimientos de la creación de videojuegos, desarrollando sistemas que pueden ser reutilizados en otros proyectos, como, el inventario donde guardaremos nuestros objetos, el guardado y cargado de partida, mapas con la posición del jugador y las zonas descubiertas, objetos que puedan ser tanto equipados como consumidos, fabricación de objetos por medio de materiales, creación de un árbol de diálogos para crear la narrativa del videojuego, inteligencia artificial para los enemigos, combate y movimiento del jugador fluido.

2. Estado del arte

El género *metroidvania* es un subgénero de acción-aventura de plataformas no lineal centrado en la exploración, este se originó en Japón donde se muestran los primeros elementos que darían forma a este género con el lanzamiento en 1985 del videojuego *Brain Breaker* teniendo este un mundo abierto a explorar con plataformas y acción.

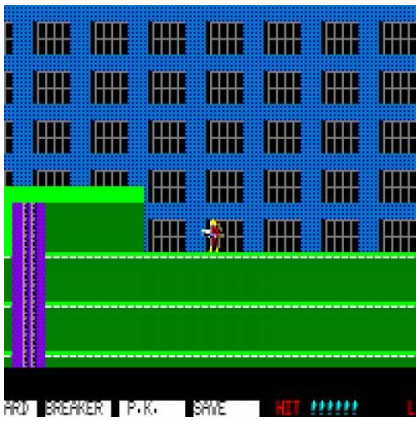

Figura 1: escenario de «Brain Breaker»

Fue en el año 1986 con el videojuego Metroid que se empezó a popularizar los videojuegos de plataformas no lineales. Estos videojuegos se basan en la exploración de las áreas que solo podían ser accedidas una vez se obtiene un objeto de otra área que proporcionaba la habilidad necesaria para llegar. Por ello es normal que se haga mucho *back-tracking*[1] para explorar esas zonas anteriores que todavía no se han visitado.

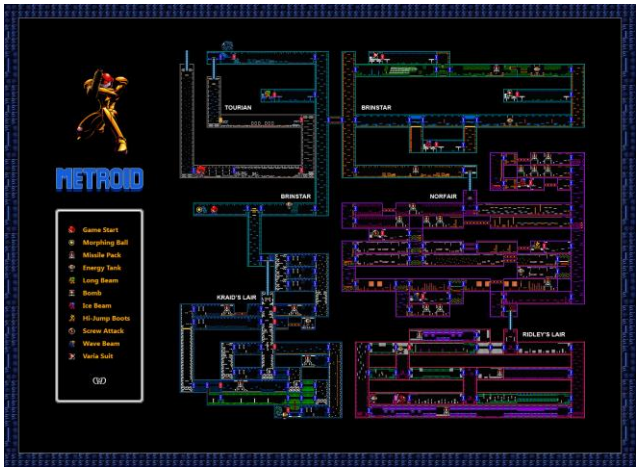

Figura 2: mapa y objetos de «Metroid»

Subsiguientemente la serie de videojuegos Castlevania, empezando por Vampire Killer en 1986, empezó a experimentar con este género y no fue hasta Castlevania: Symphony of the Night en 1997 que se empezó a innovar en el género añadiendo multitud de elementos RPG, ya que los jugadores nuevos al género tenían problemas para finalizar los videojuegos y estos elementos creaban una progresión para hacer al personaje más poderoso y así disminuir la dificultad.

[1] Poder volver a zonas que ya han sido visitadas.

Figura 3: elementos RPG en «Castlevania: Symphony of the Night»

Los videojuegos *metroidvania* cada vez generan más interés en la comunidad, llegando así a que en este momento sea un género al alza principalmente desarrollada por desarrolladoras indie.

Actualmente, se considera al género *souls-like*[2] iniciado por la desarrolladora de videojuegos japonesa From Software con el videojuego Demon's Souls en 2009 como la evolución del género *metroidvania*. Este género es muy querido por la comunidad, llegando así a que Dark Souls: Remastered (continuación de Demon's Souls) ganó el Golden Joystick Awards en 2021 como mejor juego de todos los tiempos.


Figura 4: jugador en «Dark Souls Remastered»

2.1 Análisis del mercado

En este apartado se analizarán videojuegos similares al planteado que han servido de inspiración para la realización del diseño, la ambientación y las mecánicas[3] del proyecto.

La idea principal es obtener información sobre puntos en común y características únicas entre los diferentes videojuegos del género *metroidvania* para poder implementarlas en el proyecto.

2.1.1 Hollow Knight

Hollow Knight (Team Cherry, 2017) es un videojuego 2D de género *metroidvania* creado por la desarrolladora de videojuegos *indie*[4] Team Cherry, publicado en 2017 y desarrollado utilizando el motor de videojuegos Unity.

[2] El género *souls-like* se refiere a videojuegos similares a los de la saga *souls*, videojuegos los cuales tienen una dificultad muy alta pero los retos que proponen al jugador pueden superarse en base a la observación y el aprendizaje del entorno, los enemigos y las propias mecánicas del jugador.
[3] Regla o conjunto de reglas del videojuego.

Este videojuego tiene una ambientación lúgubre, un apartado musical de índole clásica que nos transporta a un mundo destruido por la infección. Presenta un mundo extenso, zonas ocultas, un mapa oculto mediante el denominado sistema *fog of war*[5]que tendremos que explorar y fijarnos en los detalles de cada zona para descubrir la historia del lugar, ya que cada zona tiene un entorno único, además, nos ofrece un combate fluido con ligeras inspiraciones en el género *souls-like*, un sistema de guardado en puntos determinados, en este caso los "bancos", donde además suele haber cerca un sistema de viaje rápido entre zonas, también ofrece comercio con los habitantes.

Figura 5: protagonista de «Hollow Knight» peleando contra el primer jefe del videojuego

2.1.2 Axiom Verge

Axiom Verge (Zimmerman, 2015) es un videojuego 2D de género *metroidvania* creado enteramente por el desarrollador Thomas Happ, publicado en 2015 y desarrollado utilizando el motor de videojuegos MonoGame.

Este videojuego de corte clásico se mantiene fiel a las bases que propusieron los precursores del género sin mucha innovación, se caracteriza por un arte *pixel art*[6], ofrece un amplio arsenal, un mini mapa dividido en celdas, y un mapa amplio donde se recompensa la exploración, ya que será como se obtenga la mayoría de los objetos y así poder progresar.

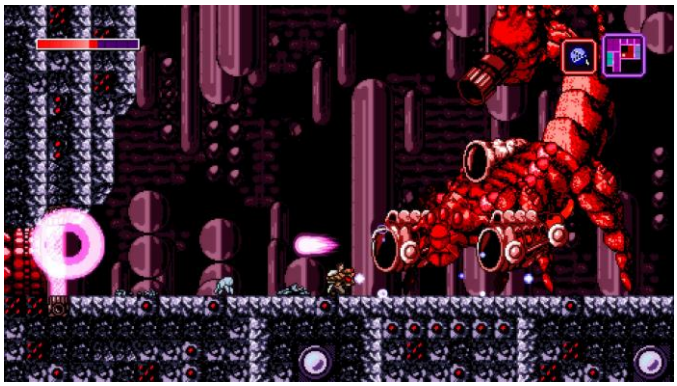

Figura 6: pelea contra un jefe en «Axiom Verge»

[4] Independiente. Desarrollado por un grupo reducido de personas comúnmente sin apoyo financiero de distribuidoras y publicistas.
[5] El termino *fog of war* se refiere a un sistema el cual un mapa está cubierto por un manto gris que nos oculta las zonas adyacentes hasta que no vayamos a ellas.
[6] Es una forma de arte digital originaria de los videojuegos arcade clásicos donde se trabaja con modelos 2D con pocos píxeles, normalmente no se supera el tamaño de 64x64.

2.1.3 Bloodstained: Ritual of the Night

Bloodstained: Ritual of the Night (Igarashi, 2015) es un videojuego 3D de género *metroidvania* creado por la desarrolladora de videojuegos ArtPlay utilizando el motor de videojuegos Unreal Engine 4. Este videojuego fue financiado gracias a una campaña de financiación en Kickstarter en 2015 y finalmente fue publicado en 2019.

El creador de este videojuego, Koji Igarashi, es el mismo que el del videojuego Castlevania que dio vida al género, por lo tanto, este es considerado un sucesor espiritual donde se puede ver la innovación en el género con la inclusión de aún más elementos RPG[7].

Este videojuego es la principal fuente de inspiración para el desarrollo de las mecánicas del juego. En este videojuego nos encontramos con mecánicas como la fabricación de objetos, la obtención de habilidades mágicas al derrotar enemigos comunes, el inventario y la obtención de recursos, un mapa por celdas, la posibilidad de poder equiparse objetos y aumentar nuestras estadísticas, donde también subiendo de nivel podemos aumentarlas, un mini mapa, el viaje rápido y el comercio.

Figura 7: exploración en «Bloodstained: Ritual of the Night»

2.1.4 Blasphemous

Blasphemous (The Game Kitchen, 2017) es un videojuego 2D de género *metroidvania* creado por la desarrolladora de videojuegos *indie* española The Game Kitchen utilizando el motor de videojuegos Unity. Este videojuego fue financiado gracias a una campaña de financiación en Kickstarter en 2017 y finalmente fue publicado en 2019 en distintas plataformas de ordenador y consola.

Este videojuego es la principal fuente de inspiración para la realización del diseño del mundo y el combate de este proyecto. En este videojuego se puede destacar la gran inspiración en el género *souls-like*. Ofrece una gran amplitud de movimientos que puede realizar el personaje, ya sean enfocados en la movilidad como en el combate, podemos hacer viajes rápidos entre los altares, las peleas contra los enemigos nos proporcionan “lágrimas de expiación” que son la moneda del juego para comprar objetos y nuevas habilidades al subir de nivel.

[7] Role-playing game.

También presenta un mapa por habitaciones donde la exploración es crucial para desbloquear nuevas habilidades que nos permitan llegar a zonas antes inalcanzables.

Figura 8: el penitente luchando contra unos de los jefes de «Blasphemous»

2.1.5 Comparación de videojuegos

En esta sección se comparará las características de cada videojuego metroidvania presentado en la sección anterior y se decidirá qué características se proponen implementar en el proyecto.

	Hollow Knight	Axiom Verge	Bloodstained	Blasphemous
Parar ataque	✓	✗	✓	✓
2D	✓	✓	✗	✓
Mundo sin pantallas de carga	✗	✓	✗	✗
Sistema de diálogos	✓	✓	✓	✓
Habilidades mágicas	✓	✗	✓	✓
Fabricación de objetos	✗	✗	✓	✗
Árbol de habilidades	✗	✗	✗	✓
Inventario	✓	✓	✓	✓
Mapa	✓	✓	✓	✓
Objetos consumibles	✗	✗	✓	✗
Objetos equipables	✓	✗	✓	✓
Cancelación de animaciones	✗	✗	✓	✓
Personajes jugables personalizables	✗	✗	✓	✓
Sistema de niveles de jugador	✗	✗	✓	✓
Esquivar	✓	✗	✓	✓
Se obtienen nuevas habilidades al derrotar enemigos	✓	✗	✓	✗
Comercio	✓	✗	✓	✓
Viaje rápido	✓	✗	✓	✓

Tabla 1: comparación entre videojuegos

Para este proyecto se quiere juntar todas estas mecánicas que hemos podido ver en la tabla anterior en un solo videojuego para crear un videojuego *metroidvania* moderno con elementos RPG y la dificultad de un *souls-like* para hacer un avance en el género y crear una experiencia refrescante y desafiante para el jugador. Por ello, se propone implementar en el proyecto la parada de ataque, que sea 2D, puesto que es más fácil realizar el arte, que no haya pantallas de carga, un sistema de árbol de diálogos donde cada opción provocara una acción, habilidades mágicas, fabricación de objetos, un inventario para guardar los objetos que suelten los enemigos, un mapa para orientarnos por el escenario, objetos consumibles y equipables que alteraran nuestras estadísticas, un sistema de niveles del jugador para que haya una sensación de progreso, esquivar ataques para hacer que el combate dependa más de la habilidad del jugador que de los objetos que lleve equipado, que se obtengan habilidades de un enemigo al derrotarlo, comercio con *NPC* para comprar y vender objetos únicos y por último, el viaje rápido para transportar al jugador entre las distintas áreas del videojuego.

2.2 Motores de videojuegos

En este apartado se analizarán los motores de videojuegos[8] más populares, donde podremos comparar sus distintas características para ver la idoneidad de estos en relación con el desarrollo de este proyecto.

2.2.1 Unity

Unity (Unity Technologies, 2021) es uno de los mejores motores de videojuegos hasta la fecha debido a todas las herramientas que proporcionan para el desarrollo, esta trabaja con el lenguaje de programación C# que es muy utilizado por el sector, también ofrece una opción de programación por nodos con la herramienta *Bolt* (Unity Technologies, 2021).

Esta herramienta destaca por la comunidad tan activa que tiene, por ser multiplataforma permitiéndonos vender nuestro producto a todas estas y por la curva de aprendizaje mínima que se necesita para poder empezar en el desarrollo de videojuegos, ya sean 2D como 3D, destacando como motor favorito para las desarrolladoras *indie* por el tipo de licencias económicas y gratuitas que ofrece. Además, cuenta con una tienda digital para la compraventa de *assets*[9].

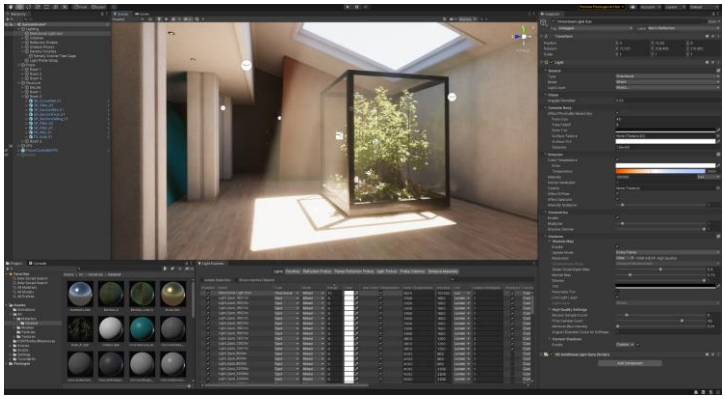

Figura 9: editor de *Unity*

[8] Los motores de videojuegos son herramientas que nos permiten agilizar el proceso software del producto a desarrollar.
[9] Elementos que pueden usarse en un proyecto. Como, por ejemplo, modelos 3D, sistemas personalizados, sonidos…

Comúnmente se ve a *Unity* como un motor de videojuegos inferior a otros motores como *Unreal Engine* debido a la cantidad de juegos anodinos que se publican al año, pero nada más lejos de la realidad, esta cantidad ascendiente de videojuegos desarrollados con Unity nos indica la facilidad de uso de la herramienta, facilidad de publicación en tiendas digitales y la confianza en esta herramienta, lo cual no hace más que hablar en su favor.

2.2.2 Unreal Engine

Unreal Engine (Epic Games, 2021) es el motor de videojuegos *cutting-edge*[10] en lo que se refiere a gráficos 3D. Utiliza el lenguaje de programación C++ por lo que le permite crear productos altamente optimizados, además ofrece una herramienta para la programación por nodos llama *Blueprint*.

Esta herramienta destaca por la comunidad activa que tiene, por ser multiplataforma, por tener una licencia gratuita y además ofrece una tienda digital donde se pueden obtener *assets* tanto de manera gratuita como de pago.

Figura 10: editor de *Unreal Engine 4*

2.2.3 Godot

Actualmente, Godot (Linietsky & Manzur, 2022) es la herramienta *open-source*[11] más popular en el mercado para el desarrollo de videojuegos. Permite la creación de videojuegos tanto 2D como 3D sin coste alguno, con una curva de aprendizaje mínima con herramientas como la programación por nodos y es una aplicación ligera. Usa su propio lenguaje de programación llamado GDScript que es similar a Python.

Por desgracia, actualmente no soporta la exportación de proyectos a consolas, pero no hay ningún problema en la exportación del resto de plataformas móviles, web y de escritorio.

[10] Que es pionera en una determinada disciplina.
[11] También conocido como código abierto, es un modelo de desarrollo basado en la colaboración.

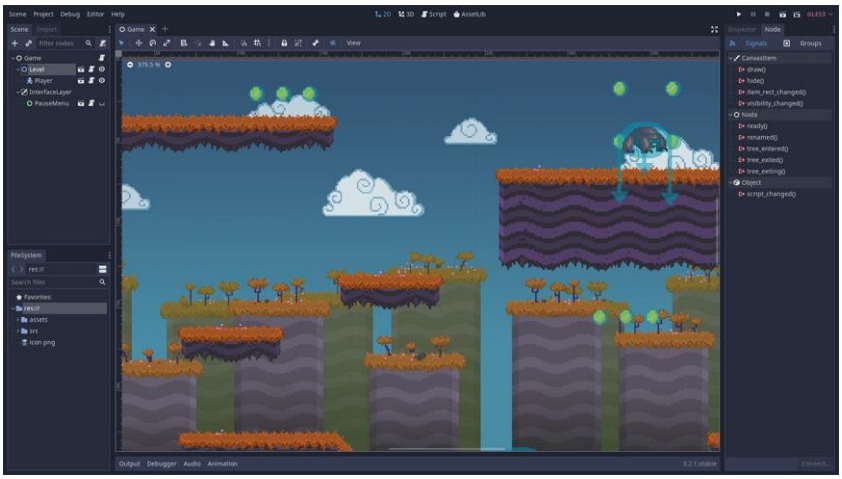

Figura 11: editor de *Godot*

2.2.4 GDevelop

GDevelop (Rival, 2020) es un motor de videojuegos sencillo de éxito moderado, principalmente popular por gente que está empezando a entrar en el sector del desarrollo de aplicaciones o en entornos educativos. Este se basa enteramente en un sistema de programación por bloques con una curva de aprendizaje mínima para la creación de videojuegos 2D tanto como aplicación de escritorio como móvil.

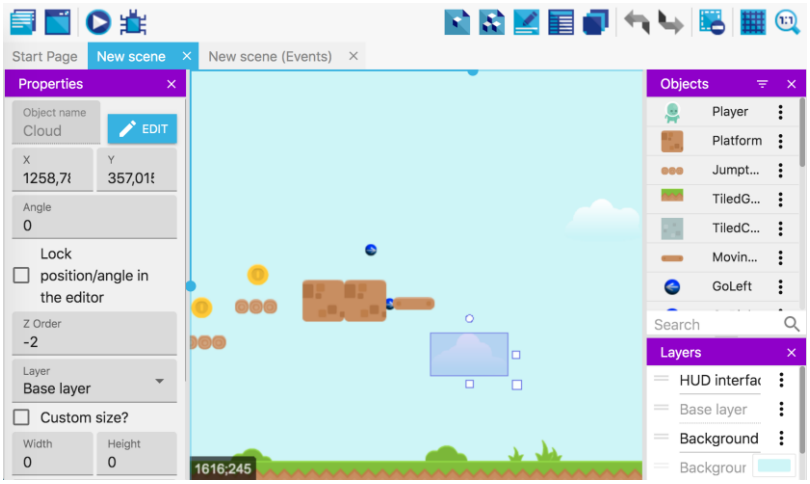

Figura 12: editor de *GDevelop 5*

2.2.5 GameMaker

GameMaker (YoYo Games, 2022) es un motor de videojuegos para el desarrollo de videojuegos 2D donde ofrece una licencia gratuita, pero esta no permite la exportación de videojuegos a escritorio o consolas, solo a plataformas web. Usa tanto su propio lenguaje llamado GML que es similar a C++ y tiene una curva de aprendizaje elevada como un sistema de programación por nodos similar a Bolt de Unity llamado GML Visual que es más apto para principiante, ya que no tiene una curva de aprendizaje muy elevada.

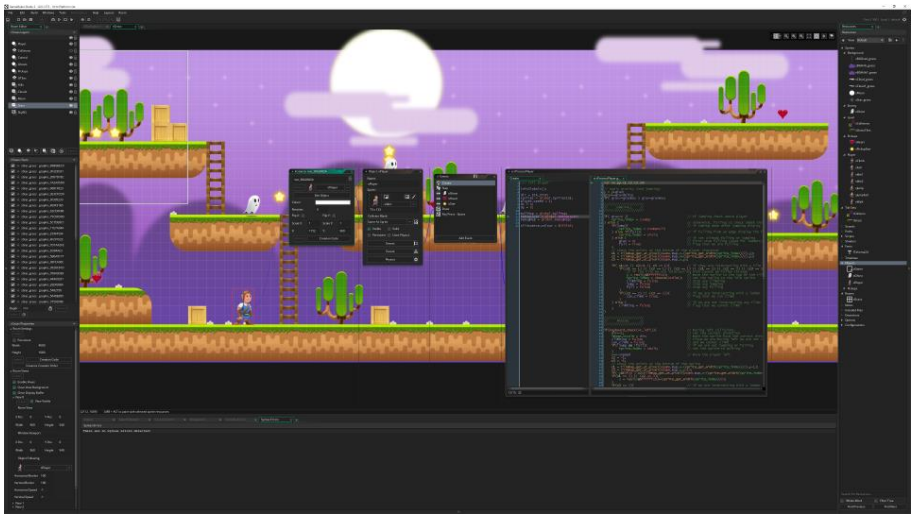

Figura 13: editor de *GameMaker Studio 2*

2.2.6 Comparación de motores de videojuegos

En esta sección se comparará las características de cada motor de videojuegos presentado en la sección anterior y se decidirá cuál es la mejor opción para ser usado en la creación del proyecto.

	Unity	Unreal Engine	Godot	GDevelop	GameMaker Studio 2
Permite desarrollo de videojuegos 2D	✓	✓	✓	✓	✓
Permite desarrollo de videojuegos 3D	✓	✓	✓	✗	✗
Código abierto	✗*	✗	✓	✓	✗
Exportación multiplataforma	✓	✓	✓**	✓**	✓
Amplio soporte de la comunidad	✓	✓	✓	✗	✓
Experiencia previa	✓	✓	✗	✗	✗
Licencia gratuita	✓	✓	✓	✓	✓
Curva de aprendizaje mínima	✓	✗	✓	✓	✓
Exportación gratuita	✓	✓	✓	✓	✗

Tabla 2: comparación de motores de videojuegos

(*) Si se dispone de una licencia Pro, si se puede ver el código fuente.
(**) No puede exportarse a consolas.

Como conclusión, podemos ver que Unity es un claro ganador en la comparativa, teniendo así una licencia gratuita[12], un amplio soporte de la comunidad ya sea en tutoriales como en *assets* en la tienda, un fácil uso de la herramienta, exportación multiplataforma y lo más importante, experiencia previa.

2.3 Análisis DAFO

En esta sección se realizará un análisis DAFO con el objetivo de determinar las ventajas competitivas tanto internas como externas del proyecto.

Fortalezas	**Debilidades**
• Experiencia previa con la herramienta • Aprendizaje rápido • Creatividad • Gran motivación • Conocimientos avanzados en programación	• Sin experiencia en el diseño de elementos metroidvania • Una única persona en el desarrollo del producto • Se necesita de recursos de otras disciplinas donde se tiene nula o poca experiencia como la creación de *sprites* o música • El proyecto tiene una complejidad alta debido a la gran magnitud de sistemas que se planean implementar • Bajo poder adquisitivo

Tabla 3: análisis interno del proyecto

Oportunidades	**Amenazas**
• Género de videojuego en auge y amado por la comunidad • Aprendizaje de sistemas complejos • Asistencia del tutor en el proyecto • Posibilidad de venta del producto en tiendas digitales una vez finalizado	• Mucha competencia entre desarrolladores *indie* • Plazo de entrega ajustado para lo que se quiere conseguir

Tabla 4: análisis externo del proyecto

[12] Se debe comprar una licencia de pago de 400€/año si se supera una ganancia de 100.000 dólares en un lapso de 12 meses con nuestro producto.

3. Análisis del problema

3.1 Identificación de actores

En esta sección se identificarán a los actores que interactuarán con el producto.

Identificador	**ACT1**
Nombre	Personaje jugador
Descripción	Usuario principal de la aplicación.

Tabla 5: ACT1

3.2 Diagrama de contexto

El diagrama de contexto define los límites entre un sistema y su entorno mostrando los actores que interactúan con el sistema, en este caso, es muy sencillo, puesto que se trata de un videojuego de un jugador y es el usuario el único que puede interactuar con el sistema.

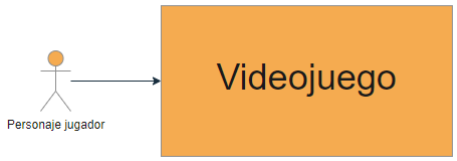

Figura 14: diagrama de contexto

3.3 Diagrama de casos de uso

En esta sección se definirá el comportamiento de los actores sobre como interactuarán con el producto. Se ha creado el diagrama gracias a la aplicación online diagrams.net (JGraph Ltd, 2021).

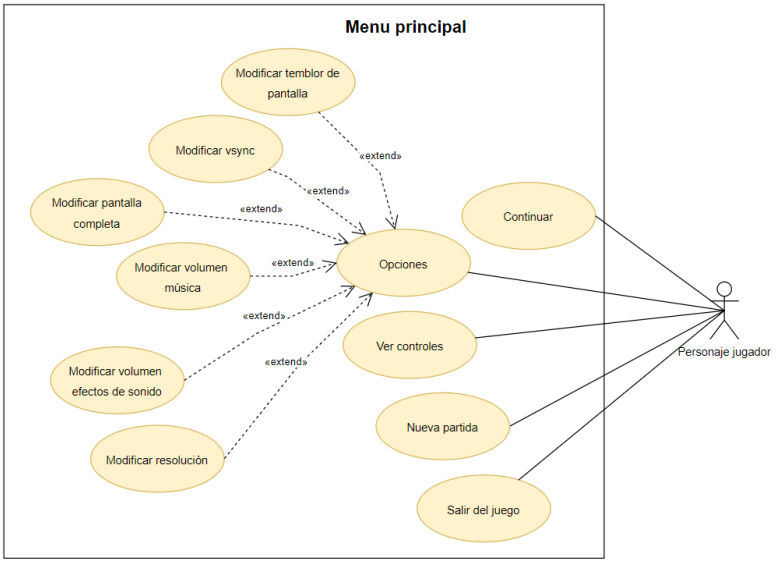

Figura 15: diagrama de casos de uso del menú principal

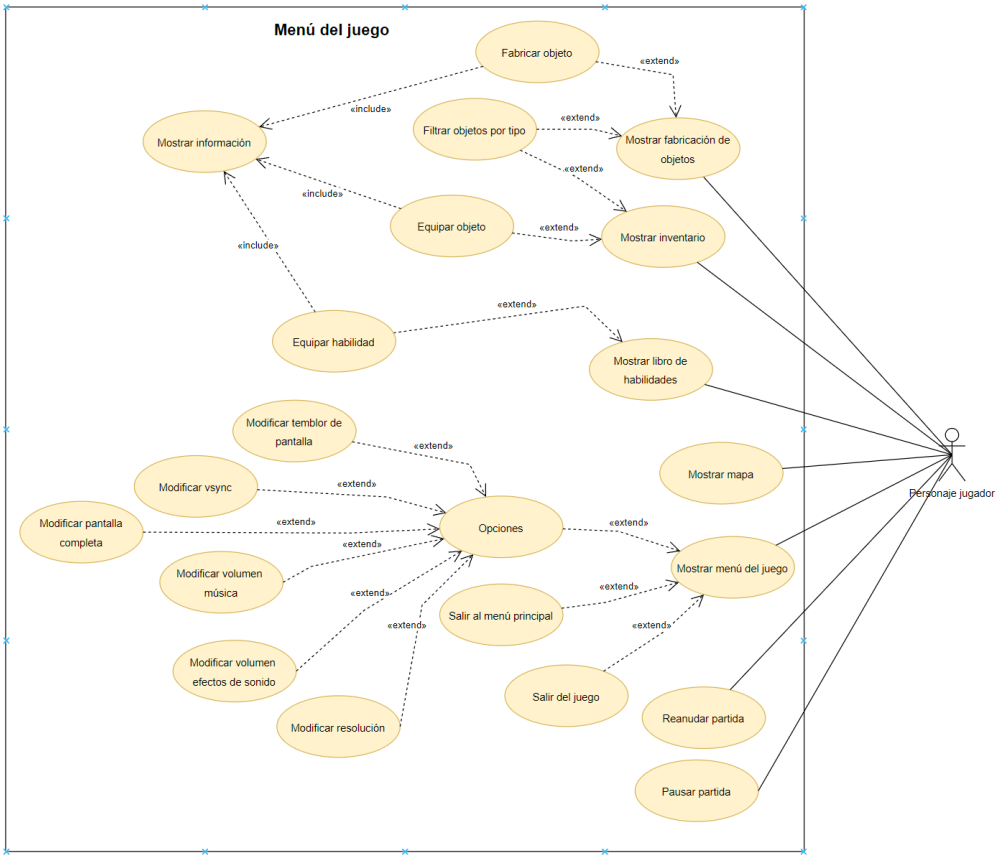

Figura 16: diagrama de casos de uso del Menú del juego

Figura 17: diagrama de casos de uso de la jugabilidad

3.4 Requisitos funcionales

En esta sección se especificarán los requisitos funcionales y no funcionales del producto siguiendo el estándar IEEE 830 a partir de los diagramas de casos de uso anteriormente mostrados.

Identificador	**RF1**
Nombre	Movimiento
Descripción	El jugador deberá poder desplazarse horizontalmente.
Prioridad	Alta

Tabla 6: RF1

Identificador	**RF2**
Nombre	Aumentar atributo
Descripción	Al alcanzar o superar el límite de puntos de experiencia del actual nivel, el jugador será recompensado con un punto de maestría, el cual deberá permitirle aumentar un atributo principal a su elección (fuerza, inteligencia, destreza) cuando interactúe con un santuario.
Prioridad	Media

Tabla 7: RF2

Identificador	**RF3**
Nombre	Saltar
Descripción	El jugador deberá poder realizar un salto cuando esté en el suelo.
Prioridad	Alta

Tabla 8: RF3

Identificador	**RF4**
Nombre	Saltar hacia abajo
Descripción	El jugador deberá poder saltar hacia abajo cuando esté sobre una plataforma especial.
Prioridad	Alta

Tabla 9: RF4

Identificador	**RF5**

Nombre	Atacar
Descripción	El jugador deberá poder efectuar ataques a corta distancia, que disminuirán la vida del enemigo al colisionar con él, basándonos en el daño del jugador que irá escalando con la fuerza hasta que la salud del objetivo llegue a cero, lo que nos recompensará con puntos de experiencia, almas y posibles objetos/habilidades.
Prioridad	Alta

Tabla 10: RF5

Identificador	**RF6**
Nombre	Contraatacar
Descripción	El jugador deberá poder contraatacar un ataque enemigo a corta distancia si hace una parada en el momento adecuado, haciendo así un ataque poderoso contra el objetivo.
Prioridad	Media-alta

Tabla 11: RF6

Identificador	**RF7**
Nombre	Guardar partida
Descripción	El jugador deberá poder guardar la configuración actual del juego (zonas desbloqueadas en el mapa, ultimo santuario visitado, progreso del juego) junto al estado actual del jugador (fuerza, destreza, inteligencia, defensa, suerte, puntos de salud, velocidad, mana, experiencia actual, nivel, monedas, objetos del inventario, habilidades desbloqueadas, habilidad equipada) que sobrescribirá la partida guardada actual.
Prioridad	Alta

Tabla 12: RF7

Identificador	**RF8**
Nombre	Continuar
Descripción	El jugador deberá poder cargar la última partida guardada desde el menú principal donde cargará la configuración del juego (zonas desbloqueadas en el mapa, ultimo santuario visitado, progreso del juego) junto al estado del jugador (fuerza, destreza, inteligencia, defensa, suerte, puntos de salud, velocidad, mana, experiencia actual, nivel, monedas, objetos del inventario, habilidades desbloqueadas, habilidad equipada).
Prioridad	Alta

Tabla 13: RF8

Identificador	**RF9**
Nombre	Beber poción
Descripción	La poción tiene un límite de usos y el jugador deberá poder beber una poción que restaura parte de su salud y restará en uno a estos usos hasta que no quede ninguna poción y tenga que volver a un santuario para rellenar todas las pociones.
Prioridad	Media

Tabla 14: RF9

Identificador	**RF10**
Nombre	Recoger objeto
Descripción	Al matar a un enemigo, este puede soltar objetos de diferentes tipos (materiales, consumibles, armadura y receta) donde el jugador deberá poder recogerlos al interactuar con ellos y añadirlos a su inventario.
Prioridad	Media

Tabla 15: RF10

Identificador	**RF11**
Nombre	Interactuar con NPC
Descripción	El jugador debe poder hablar con los personajes no jugables amistosos que le proporcionarán información y/o comercio de objetos de todo tipo.
Prioridad	Alta

Tabla 16: RF11

Identificador	**RF12**
Nombre	Usar habilidad
Descripción	El jugador debe poder usar la habilidad que esté equipada actualmente, puede ser tanto defensiva como ofensiva, lo cual gastará cierta cantidad de maná y pondrá la habilidad en enfriamiento.
Prioridad	Alta

Tabla 17: RF12

Identificador	**RF13**
Nombre	Mostrar mapa
Descripción	El jugador debe poder abrir el mapa donde se indicarán las zonas descubiertas,

	la posición actual del jugador y la localización de los santuarios.
Prioridad	Alta

Tabla 18: RF13

Identificador	**RF14**
Nombre	Mostrar menú del juego
Descripción	El jugador debe poder mostrar el menú del juego que será similar al menú principal, donde se podrá ir a opciones, salir del juego o volver al menú principal.
Prioridad	Alta

Tabla 19: RF14

Identificador	**RF15**
Nombre	Ataque cargado
Descripción	El jugador debe poder realizar un ataque cargado manteniendo pulsado el botón de ataque por cierto tiempo, lo cual lo dejará inmóvil hasta que suelte el botón y aseste un poderoso golpe a corta distancia y provocando una onda de energía que destruirá los proyectiles enemigos.
Prioridad	Alta

Tabla 20: RF15

Identificador	**RF16**
Nombre	Esquivar
Descripción	El jugador debe poder efectuar un esquive que proporciona invulnerabilidad temporal al daño de los enemigos mientras dure.
Prioridad	Alta

Tabla 21: RF16

Identificador	**RF17**
Nombre	Interactuar con santuario
Descripción	El jugador debe poder interactuar con los santuarios, los cuales restaurarán la salud, el maná, las pociones del jugador y tendrás la opción de gastar tus puntos de atributo, viajar a otros santuarios y guardar la partida.
Prioridad	Media

Tabla 22: RF17

Identificador	**RF18**
Nombre	Otear
Descripción	El jugador debe poder mover la cámara ligeramente en todas las direcciones alrededor del jugador mientras este parado.
Prioridad	Baja

Tabla 23: RF18

Identificador	**RF19**
Nombre	Aprender habilidad
Descripción	Al matar a un enemigo, el jugador debe poder aprender habilidades tanto activas como pasivas, las cuales se añadirán a tu lista de habilidades donde se podrán ver más detalladamente y equipar.
Prioridad	Alta

Tabla 24: RF19

Identificador	**RF20**
Nombre	Pausar partida
Descripción	Al abrir el menú se pausará la partida.
Prioridad	Baja

Tabla 25: RF20

Identificador	**RF21**
Nombre	Reanudar partida
Descripción	Al cerrar el menú se reanudará la partida.
Prioridad	Baja

Tabla 26: RF21

Identificador	**RF22**
Nombre	Salir al menú principal
Descripción	El jugador debe poder salir al menú principal.
Prioridad	Media

Tabla 27: RF22

Identificador	**RF23**

Nombre	Nueva partida
Descripción	El jugador debe poder empezar una nueva partida desde el menú principal donde el jugador comenzará una nueva historia desde cero borrando la partida guarda si la hubiera.
Prioridad	Media

Tabla 28: RF23

Identificador	**RF24**
Nombre	Salir del juego
Descripción	El jugador debe poder salir del juego desde el menú principal y el menú del juego saliendo al escritorio.
Prioridad	Alta

Tabla 29: RF24

Identificador	**RF25**
Nombre	Ver controles
Descripción	El jugador debe poder ver los controles necesarios para jugar desde el menú principal.
Prioridad	Baja

Tabla 30: RF25

Identificador	**RF26**
Nombre	Aprender receta
Descripción	El jugador debe poder aprender una receta automáticamente si esta está entre los objetos que ha recogido.
Prioridad	Media

Tabla 31: RF26

Identificador	**RF27**
Nombre	Viajar
Descripción	El jugador debe poder viajar entre santuarios de manera instantánea.
Prioridad	Baja

Tabla 32: RF27

Identificador	**RF28**

Nombre	Comerciar
Descripción	El jugador debe poder comprar y vender objetos al interactuar con personajes no jugables amistoso que sean comerciantes.
Prioridad	Media

Tabla 33: RF28

Identificador	**RF29**
Nombre	Dialogo
Descripción	El jugador debe poder comenzar un diálogo con cualquier personaje no jugable amistoso.
Prioridad	Media

Tabla 34: RF29

Identificador	**RF30**
Nombre	Agacharse
Descripción	El jugador debe poder agacharse para evadir los proyectiles enemigos.
Prioridad	Baja

Tabla 35: RF30

Identificador	**RF31**
Nombre	Agarrarse a un saliente
Descripción	El jugador debe poder agarrarse a un saliente si se salta a este.
Prioridad	Media

Tabla 36: RF31

Identificador	**RF32**
Nombre	Opciones
Descripción	El jugador debe poder abrir el menú de opciones desde el menú principal y el menú del juego, mostrando las siguientes opciones: modificar temblor pantalla, modificar el volumen de los SFX, modificar el volumen de la música, modificar resolución, modificar pantalla completa, modificar la sincronización vertical.
Prioridad	Media

Tabla 37: RF32

Identificador	**RF33**
Nombre	Modificar temblor de pantalla
Descripción	El jugador debe poder activar/desactivar el temblor de pantalla desde el menú de opciones.
Prioridad	Baja

Tabla 38: RF33

Identificador	**RF34**
Nombre	Modificar vsync
Descripción	El jugador debe poder activar/desactivar la sincronización vertical del monitor para limitar los cuadros por segundo del videojuego desde el menú de opciones.
Prioridad	Media

Tabla 39: RF34

Identificador	**RF35**
Nombre	Modificar resolución
Descripción	El jugador debe poder modificar la resolución del videojuego para ajustarla a la de su monitor desde el menú de opciones.
Prioridad	Media

Tabla 40: RF35

Identificador	**RF36**
Nombre	Modificar pantalla completa
Descripción	El jugador debe poder activar/desactivar la pantalla completa del videojuego desde el menú de opciones.
Prioridad	Media

Tabla 41: RF36

Identificador	**RF37**
Nombre	Modificar volumen efectos de sonido
Descripción	El jugador debe poder modificar el volumen de los efectos de sonido del videojuego desde el menú de opciones.
Prioridad	Media

Tabla 42: RF37

Identificador	**RF38**
Nombre	Modificar volumen música
Descripción	El jugador debe poder modificar el volumen de la música del videojuego desde el menú de opciones.
Prioridad	Media

Tabla 43: RF38

Identificador	**RF39**
Nombre	Mostrar libro de habilidades
Descripción	El jugador debe poder mostrar el libro de habilidades desde el menú del videojuego donde se almacenarán todas las que haya conseguido, se podrán equipar y mostrar toda su información.
Prioridad	Media

Tabla 44: RF39

Identificador	**RF40**
Nombre	Equipar habilidad
Descripción	El jugador debe poder equiparse una de las habilidades disponibles en su libro de habilidades.
Prioridad	Media

Tabla 45: RF40

Identificador	**RF41**
Nombre	Mostrar inventario
Descripción	El jugador debe poder mostrar el inventario donde se guardarán todos los objetos que haya recogido, se mostrarán las estadísticas del jugador y los objetos equipados.
Prioridad	Media

Tabla 46: RF41

Identificador	**RF42**
Nombre	Mostrar fabricación de objetos
Descripción	El jugador debe poder mostrar la ventana de fabricación de objetos donde se

	guardarán todas las recetas aprendidas y se podrán fabricar objetos
Prioridad	Media

Tabla 47: RF42

Identificador	**RF43**
Nombre	Fabricar objeto
Descripción	El jugador debe poder fabricar un objeto desde la ventana de fabricación de objetos si es que este dispone de los objetos necesarios para fabricarlo.
Prioridad	Media

Tabla 48: RF43

Identificador	**RF44**
Nombre	Equipar objeto
Descripción	El jugador debe poder equiparse con las armaduras que cambiarán las estadísticas del jugador y con consumibles que podrán ser consumidos durante el videojuego.
Prioridad	Media

Tabla 49: RF44

Identificador	**RF45**
Nombre	Filtrar objeto por tipo
Descripción	El jugador debe poder filtrar los objetos del inventario y de la ventana de fabricación de objetos mostrando solo los objetos del botón de filtro al que se haya pulsado.
Prioridad	Media

Tabla 50: RF45

Identificador	**RF46**
Nombre	Mostrar información
Descripción	El jugador al pulsar él un objeto este mostrará su información en una ventana lateral.
Prioridad	Media

Tabla 51: RF46

Identificador	**RF47**

Nombre	Consumir consumible
Descripción	El jugador debe poder consumir el objeto consumible que tenga equipado en ese momento, lo cual le provocara algún efecto.
Prioridad	Media

Tabla 52: RF47

Identificador	**RF48**
Nombre	Cambiar consumible
Descripción	El jugador debe poder cambiar de consumible entre los que tiene equipados.
Prioridad	Media

Tabla 53: RF48

3.5 Requisitos no funcionales

Identificador	**RNF1**
Nombre	Archivo de guardado con cifrado binario
Descripción	Las partidas guardadas se cifrarán en formato binario para que el jugador no pueda modificarlas fácilmente.
Prioridad	Media

Tabla 54: RNF1

Identificador	**RNF2**
Nombre	Disponibilidad del juego
Descripción	El juego debe poder garantizar al usuario que el 99% de las veces que se inicie el juego funcionará correctamente.
Prioridad	Alta

Tabla 55: RNF2

Identificador	**RNF3**
Nombre	Motor de videojuegos
Descripción	El proyecto será desarrollado utilizando Unity.
Prioridad	Alta

Tabla 56: RNF3

Identificador	**RNF4**

Nombre	Duración de la transición entre escenarios
Descripción	La transición no debe durar más de 5000 milisegundos.
Prioridad	Alta

Tabla 57: RNF4

Identificador	**RNF5**
Nombre	Movimiento del jugador
Descripción	El movimiento del jugador será agradable.
Prioridad	Alta

Tabla 58: RNF5

Identificador	**RNF6**
Nombre	Interfaz del usuario
Descripción	La interfaz del usuario debe de tener un diseño amigable e intuitivo.
Prioridad	Alta

Tabla 59: RNF6

Identificador	**RNF7**
Nombre	Capacidad para ser instalado del juego
Descripción	El sistema debe funcionar en sistemas operativos Windows 10.
Prioridad	Alta

Tabla 60: RNF7

Identificador	**RNF8**
Nombre	Cuadros por segundo
Descripción	Los cuadros por segundo del juego deben ser mayores a 30.
Prioridad	Alta

Tabla 61: RNF8

Identificador	**RNF9**
Nombre	Utilización de recursos
Descripción	El juego no debe de ocupar más de 1 GB en el disco duro.

Prioridad	Media-Baja

Tabla 62: RNF9

Identificador	**RNF10**
Nombre	Mantenibilidad
Descripción	El código debe ser fácilmente mantenible.
Prioridad	Alta

Tabla 63: RNF10

Identificador	**RNF11**
Nombre	Fácil de aprender
Descripción	Los controles del personaje deben de ser sencillos.
Prioridad	Media

Tabla 64: RNF11

Identificador	**RNF12**
Nombre	Lenguaje de programación
Descripción	El sistema debe estar escrito en C#.
Prioridad	Alta

Tabla 65: RNF12

Identificador	**RNF13**
Nombre	Dimensión
Descripción	El juego debe ser creado desde una perspectiva 2D.
Prioridad	Alta

Tabla 66: RNF13

Identificador	**RNF14**
Nombre	Escenarios
Descripción	Debe de haber al menos dos áreas fácilmente distinguibles.
Prioridad	Media

Tabla 67: RNF14

Identificador	**RNF15**
Nombre	Accesibilidad
Descripción	El juego debe dar la opción de desactivar el temblor de la cámara para evitar daños en la salud física del usuario.
Prioridad	Media

Tabla 68: RNF15

Identificador	**RNF16**
Nombre	Tolerancia a fallos
Descripción	El sistema no se congelará el 99% de las veces que ocurra un error.
Prioridad	Media

Tabla 69: RNF16

4. Diseño de la solución

4.1 Arquitectura software

Se ha utilizado una arquitectura abierta de tres capas y un nivel, donde se distinguen las siguientes capas:

- **Capa de presentación**: aquí es donde residen las clases pertenecientes a las interfaces de usuario y su única función será la de interactuar con el usuario mostrándole información. Esta capa solo interactúa con la capa de lógica de negocio.
- **Capa de lógica de negocio**: aquí es donde residen las clases que procesan los datos de la aplicación.
- **Capa de persistencia**: aquí es donde residen las clases pertenecientes a los datos de la aplicación y las bases de datos.

4.2 Capa de presentación

Se mostrarán los *mockups* pertenecientes al menú principal y a la jugabilidad realizados con la aplicación online Balsamiq (Balsamiq Studios, 2022). Esta herramienta permite la creación de *wireframes*[13] de una forma rápida y de baja fidelidad que fuerza al diseñador a enfocarse en la estructura y el contenido de cada interfaz de usuario.

4.2.1 Menú principal

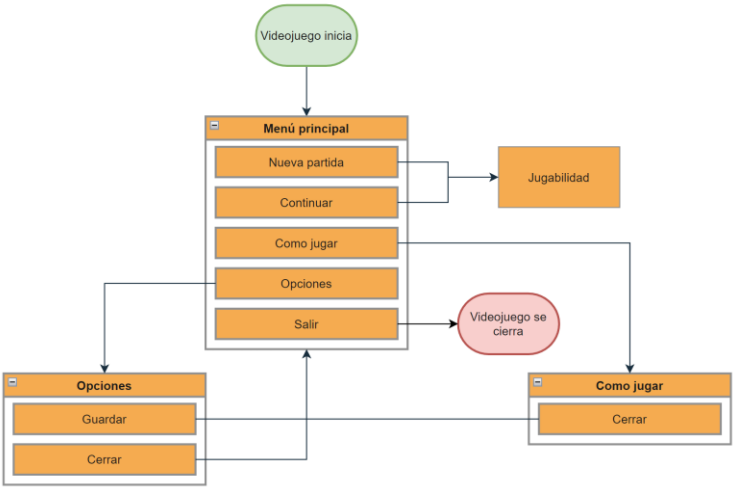

Figura 18: mapa de navegación de las interfaces del menú principal

[13] Un esbozo de la interfaz de usuario.

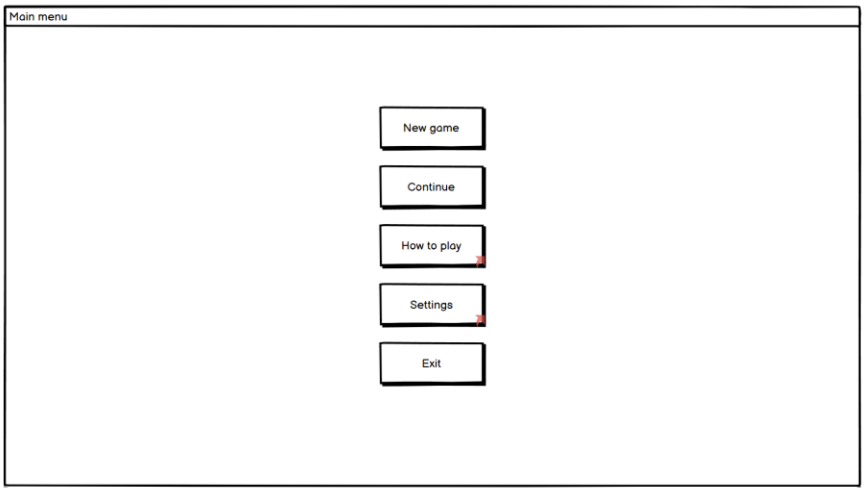

Figura 19:mockup de Menú principal

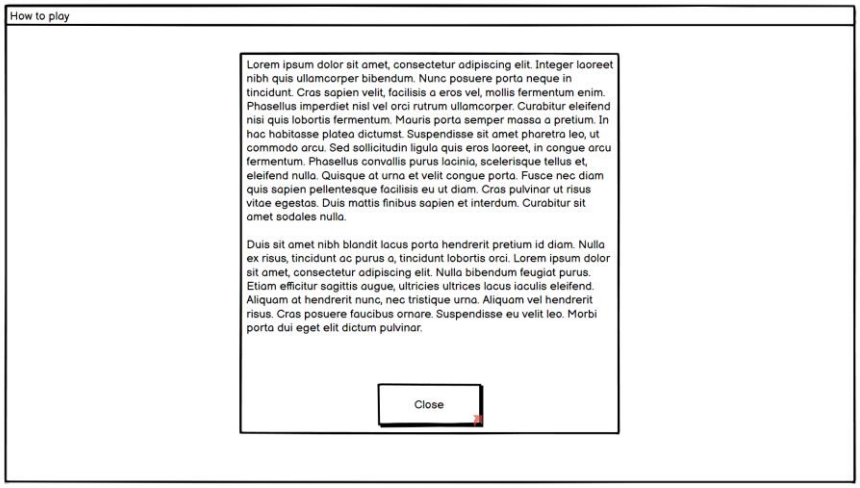

Figura 20:mockup de Como jugar

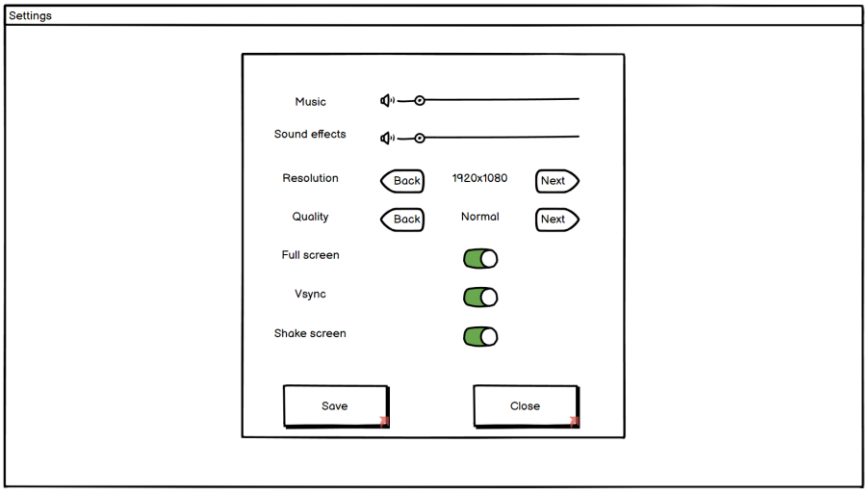

Figura 21:mockup de Opciones del menú principal

4.2.2 Jugabilidad

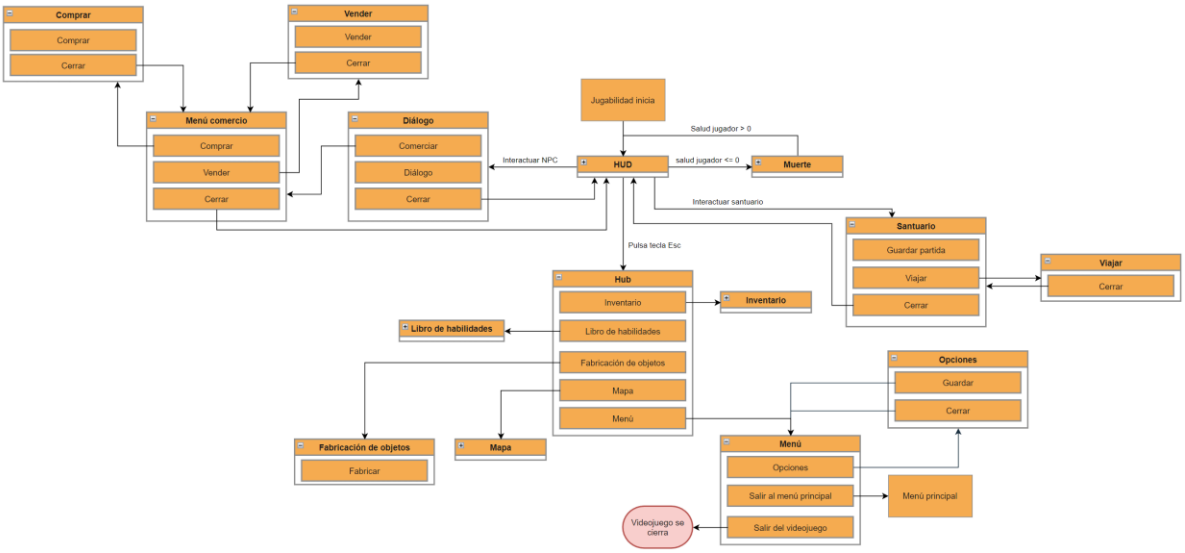

Figura 22: mapa de navegación de las interfaces de la jugabilidad

Figura 23: mockup del HUD

Figura 24: mockup del hub del videojuego

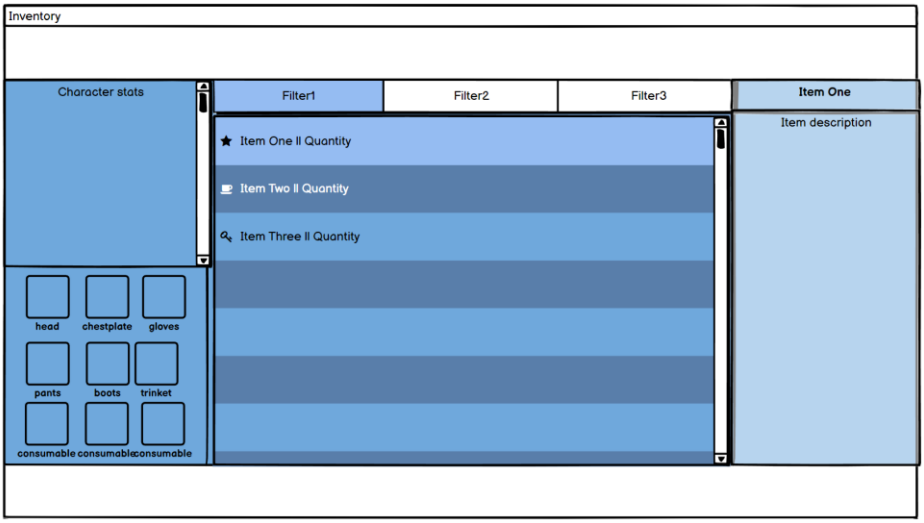

Figura 25: mockup del menú inventario

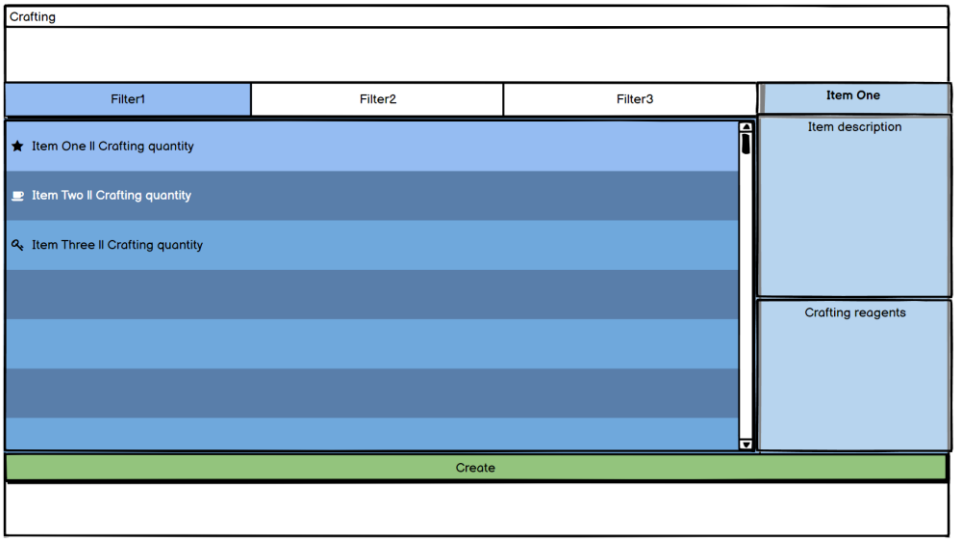

Figura 26: mockup del menú de fabricación de objetos

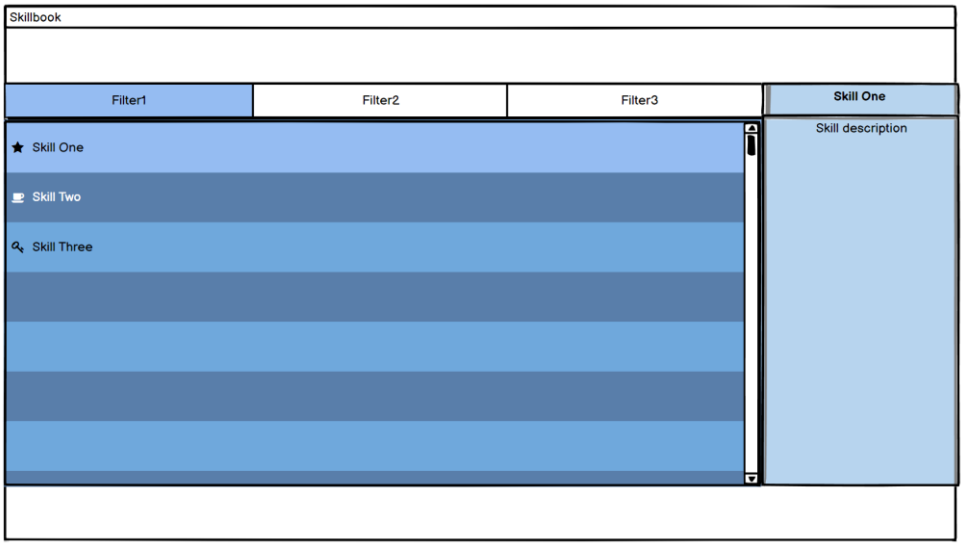

Figura 27: mockup del menú de selección de habilidades

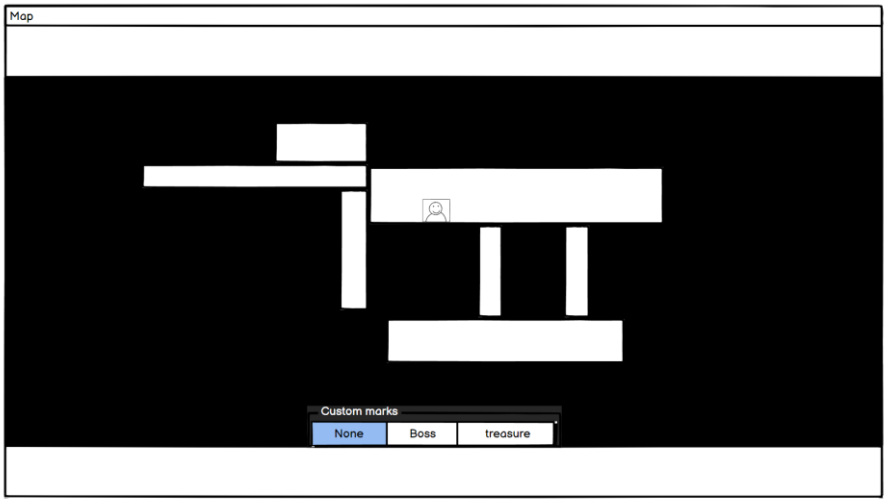

Figura 28: mockup del menú del mapa

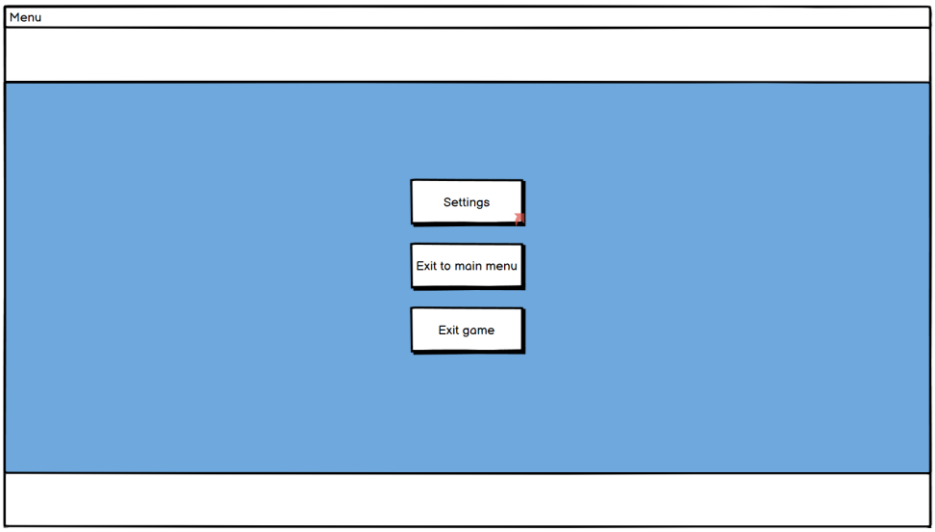

Figura 29: mockup del menú de opciones

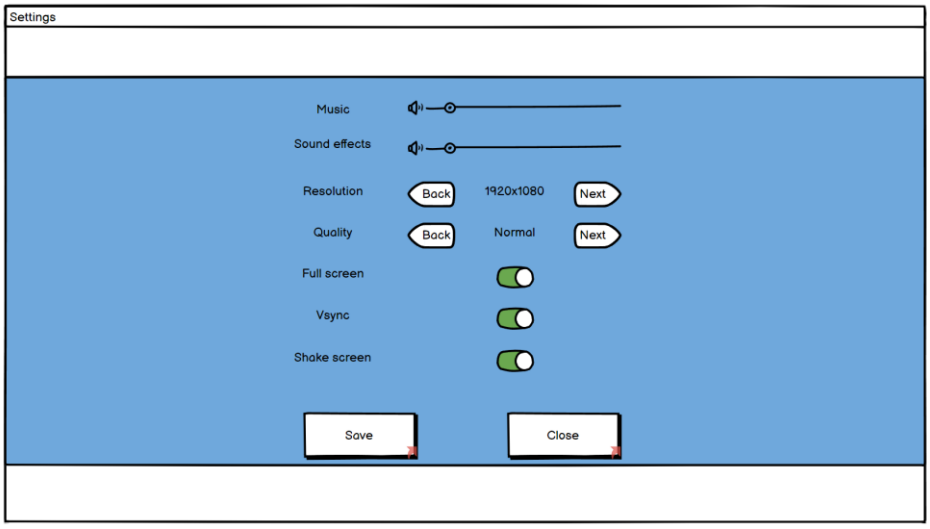

Figura 30: mockup del menu de ajustes

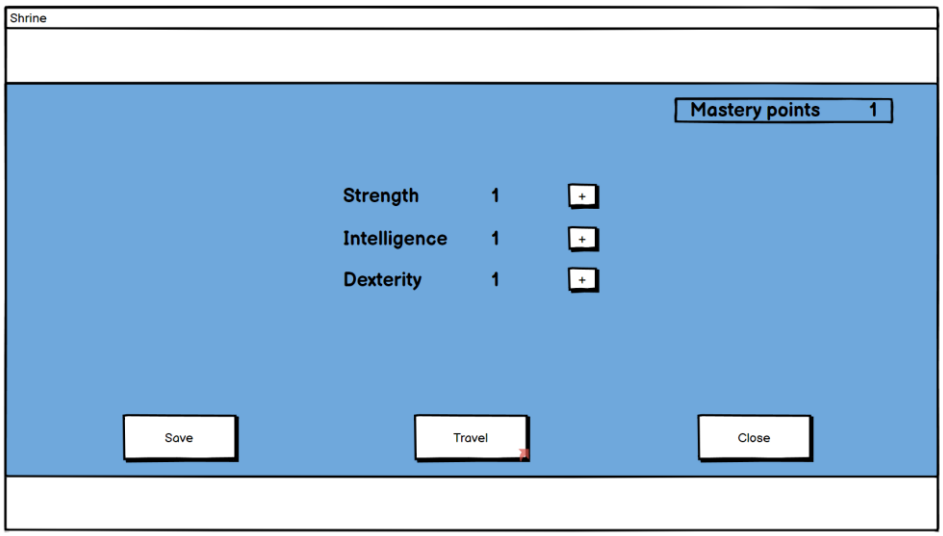

Figura 31: mockup del menú de los santuarios

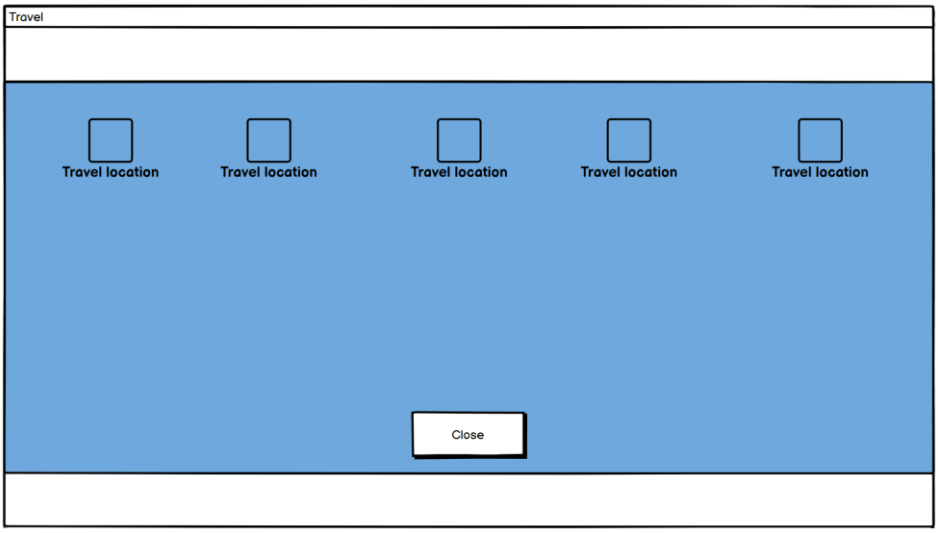

Figura 32: mockup del menú de viaje de los santuarios

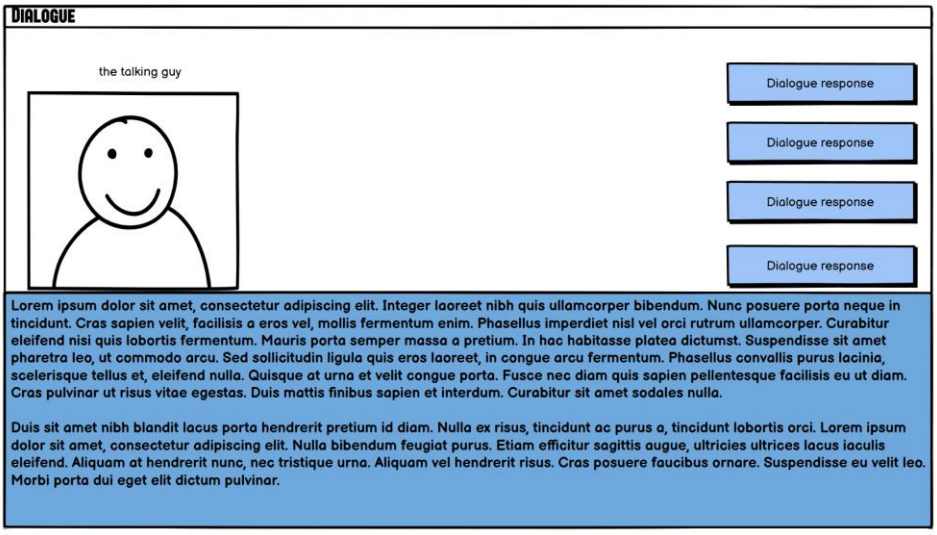

Figura 33: mockup del diálogo

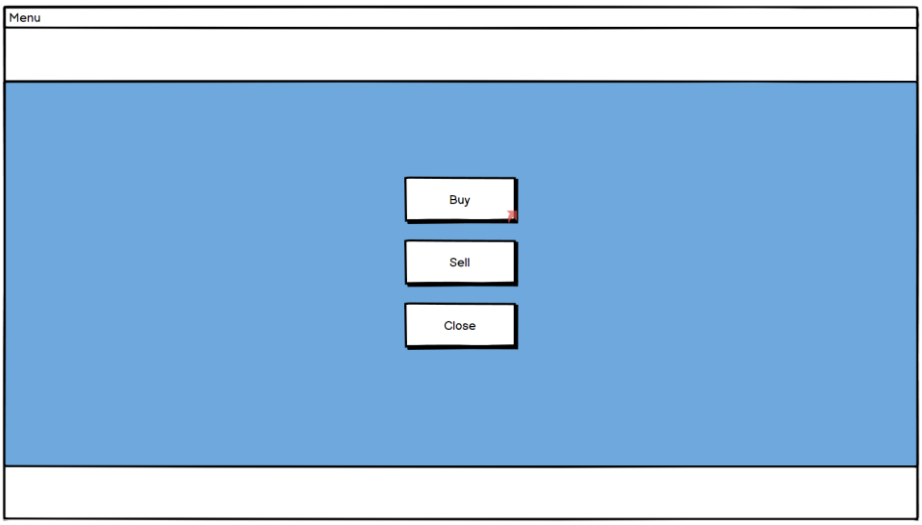

Figura 34: mockup del menú de opciones del mercader

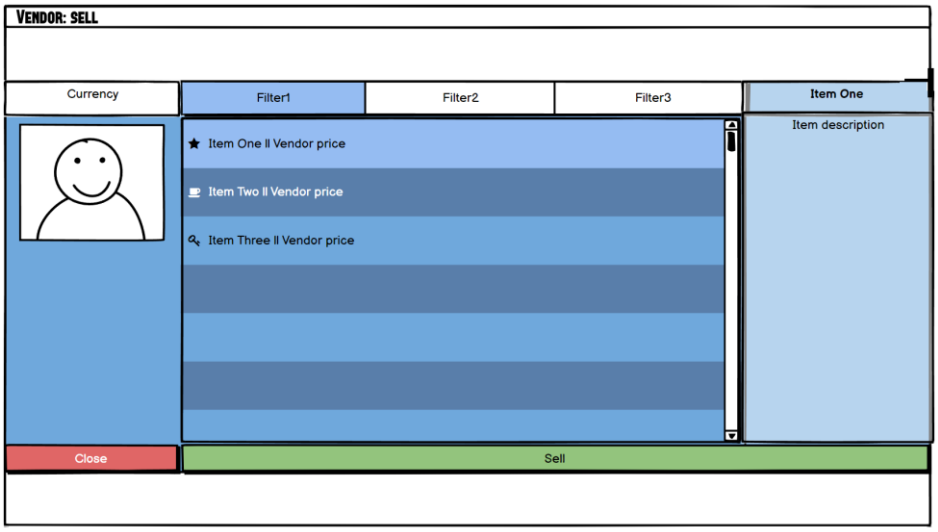

Figura 35: mockup del menú de venta del mercader

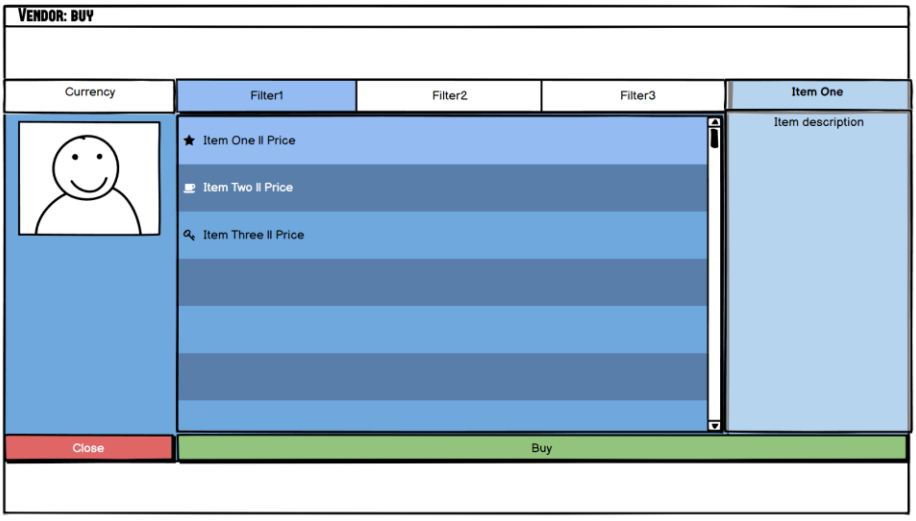

Figura 36: mockup del menú de compra del mercader

Figura 37: mockup del menú de muerte

4.3 Capa de lógica de negocio

En las siguientes subsecciones se muestra el comportamiento de cada enemigo mediante un diagrama de flujo. El diagrama de flujo representa gráficamente las decisiones que toma un proceso para realizar una tarea.

4.3.1 Comportamiento del farolillo maldito

El enemigo farolillo maldito ataca a larga distancia con ataques mágicos y cada cierto tiempo se protegerá con una barrera de fuego que hará daño al contactar con ella, además, este también explotará pasado un tiempo tras morir.

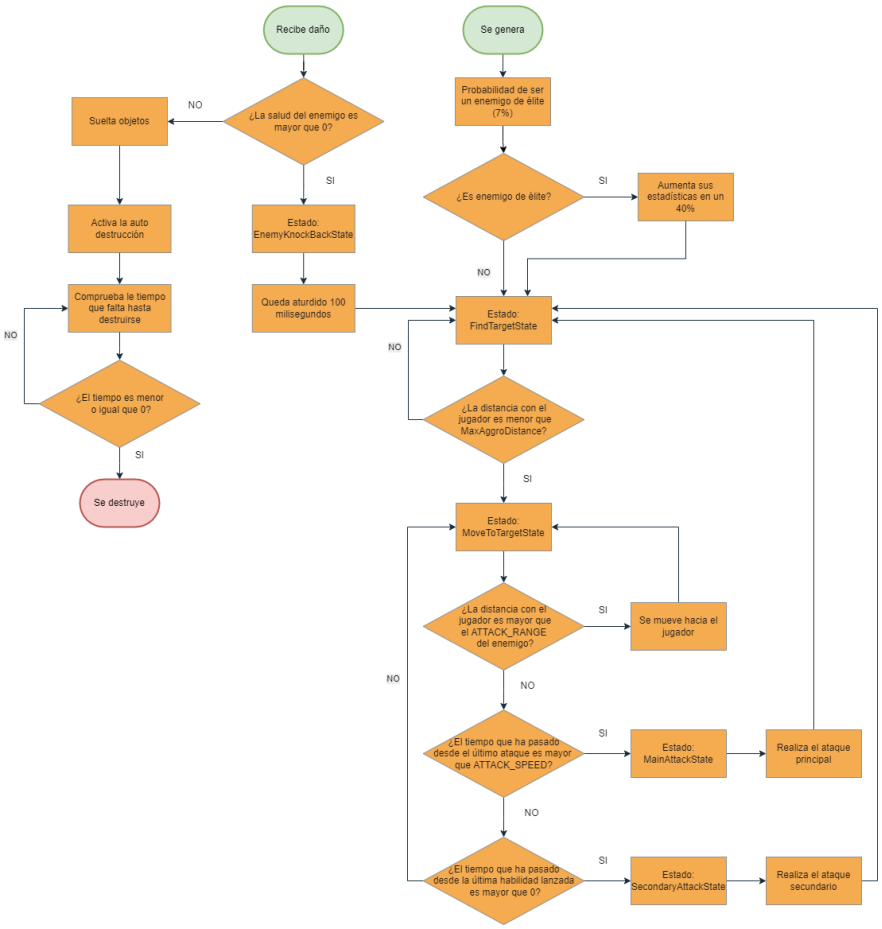

Figura 38: diagrama de flujo del comportamiento del farolillo maldito

4.3.2 Comportamiento del no-muerto

El enemigo no-muerto ataca a corta distancia con ataques físicos que infligirán un gran daño y cuando el jugador esté a media distancia o a corta distancia si este no puede efectuar un ataque físico atacará con ataques mágicos y cuando este tenga poca salud, entra en estado de frenesí aumentando su fuerza y velocidad de ataque.

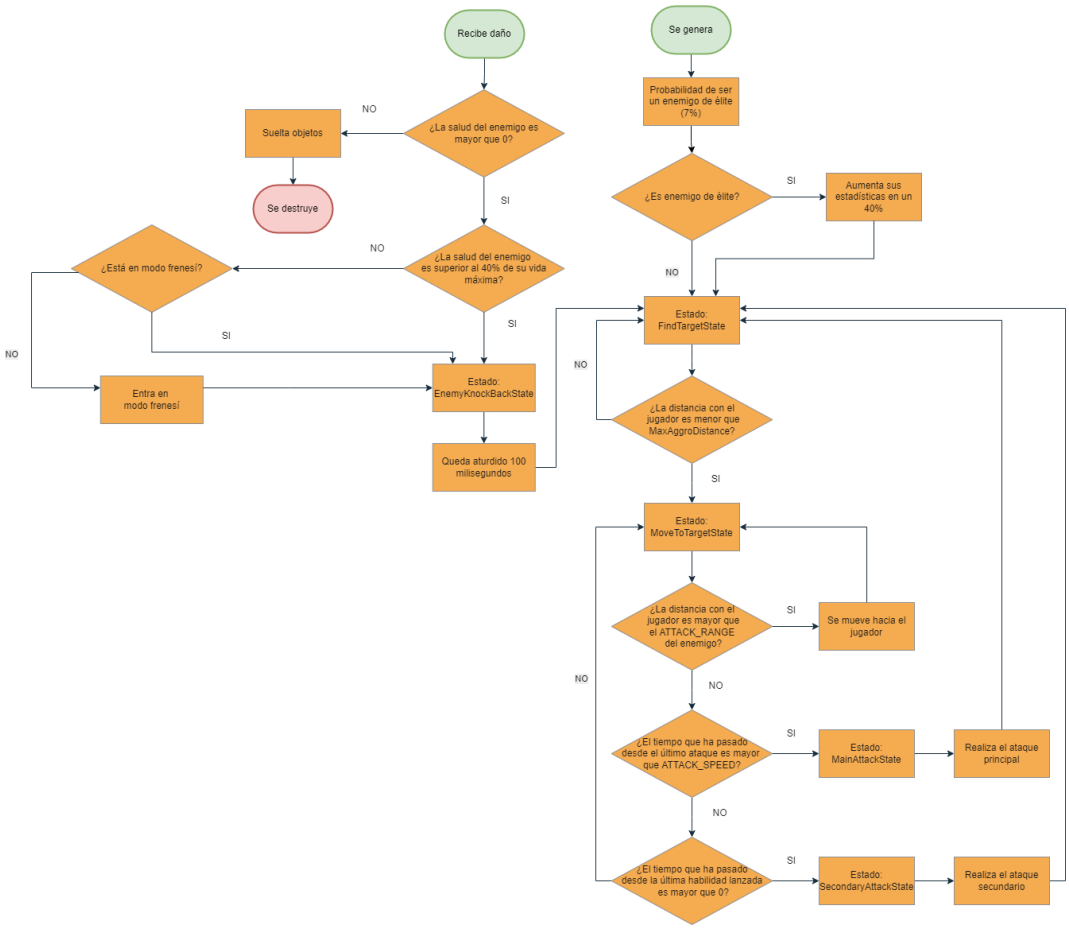

Figura 39: diagrama de flujo del comportamiento del no-muerto

4.4 Capa de persistencia

Para mantener la información de las clases que queramos, como los objetos o las estadísticas del jugador, debemos usar *ScriptableObjects* que además facilita su creación debido a que se crean directamente desde el editor de *Unity*. Lamentablemente, los *ScriptableObjects* no guardan el cambio que se haya hecho en *runtime* solo los cambios desde el editor, por ello, debemos tener un sistema de guardado y cargado de partida.

El sistema de guardado y cargado de partida se hará utilizando la serialización y deserialización binaria que aporta una gran seguridad, ya que el usuario no podrá editar los archivos directamente para hacer trampas, puesto que estos archivos se corromperían y se volverían a generar una vez se inicie el videojuego. El sistema de guardado debe guardar toda la información del personaje del videojuego y, además, también debe guardar las opciones que haya seleccionado el usuario.

Además, se deben crear bases de datos que guarden los distintos tipos de objetos, habilidades, efectos gráficos, sonidos y enemigos que se irán instanciando el *runtime*.

5. Desarrollo de la solución

A partir del diseño detallado anteriormente se realizó el desarrollo de la aplicación, donde se implementó el código junto con el arte y más adelante se añadieron todos los efectos, sonidos y el *look-and-feel*[14]. Se explicará en detalle todo lo que se ha creado y cómo en el proyecto.

5.1 Planificación del proyecto

En lo referido a la gestión del proyecto se ha decidido seguir un desarrollo iterativo e incremental, siguiendo así la corriente del uso de metodologías ágiles, cada vez más demandada en los ambientes laborales. Esta metodología nos permite tener un producto mínimo viable o MVP desde la primera iteración. Con cada iteración añadimos tanto nuevas funcionalidades como mejoramos aquellas que ya han sido implementadas.

En cuanto al establecimiento de roles, como el proyecto solo es desarrollado por una persona, esta también hará de *Product Owner* y decidirá que tareas se implementarán y cuáles se incluirán en cada iteración.

Para este proyecto se hizo una estimación inicial de aproximadamente 4 meses. Se realizaron tres iteraciones o entregas donde, además, se diseñó un diagrama de Gantt utilizando *TeamGantt* que es una extensión o *power-up* de Trello para planificar las tareas a realizar en las distintas entregas, mostrando así el esfuerzo necesario empleado en cada entrega.

5.1.1 Primera iteración

La primera entrega comienza el 24/06/2021 y finaliza el 28/07/2021. Esta entrega tiene como objetivo llevar a cabo todo aquello imprescindible para tener una primera versión jugable, esto incluye la mayor parte de la movilidad del jugador, el combate, los enemigos, la mayor parte de las habilidades, un inventario básico que almacene objetos y el sistema de diálogo.

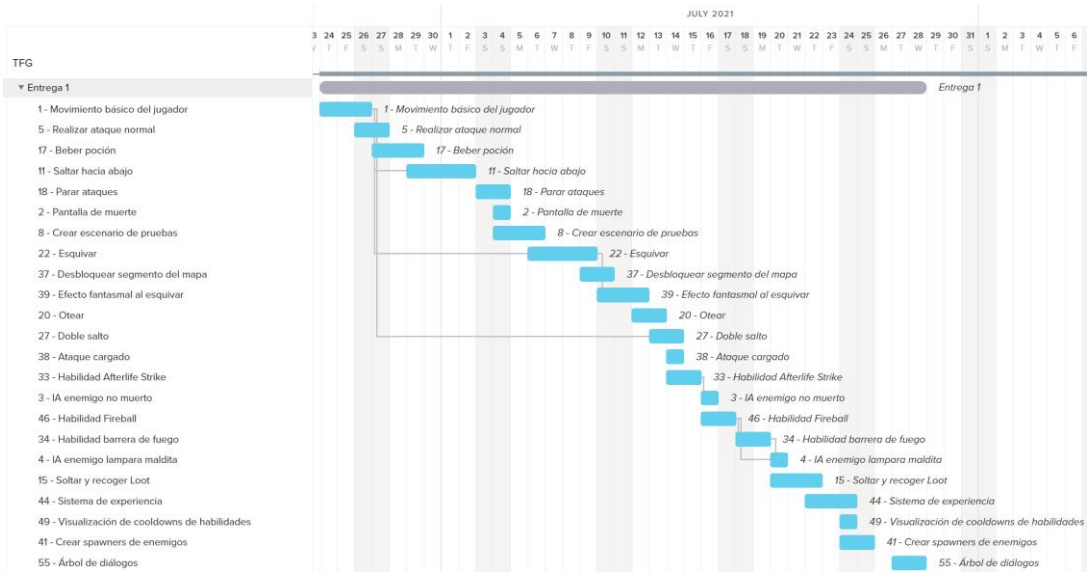

Figura 40: diagrama de Gantt de la Entrega 1

[14] Estas son las características que dan una identidad visual única y pueden ser percibidos de manera diferente de acuerdo con cada usuario.

5.1.2 Segunda iteración

La segunda entrega comienza el 29/07/2021 y finaliza el 01/09/2021. Esta entrega tiene como objetivo realizar el sistema base de cargado y guardado de partida, visualizar el inventario, modificar las opciones del videojuego, los escenarios, se expandirá el movimiento del jugador y se refactorizaran los flecos de la anterior entrega.

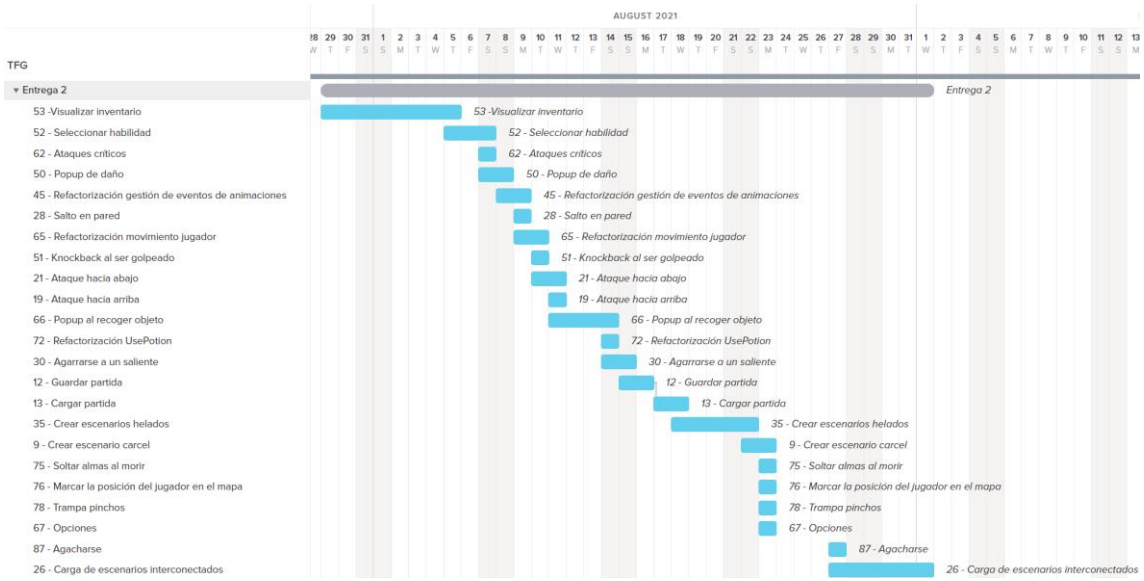

Figura 41: diagrama de Gantt de la Entrega 2

5.1.3 Tercera iteración

La tercera entrega empieza el 03/10/2021 y finaliza el 05/11/2021, pausando el desarrollo todo septiembre por vacaciones. Esta entrega tiene como objetivo realizar el sistema de fabricación de objetos, comercio, se expandirá el sistema de objetos usables y que pueden ser equipados, se expandirá el combate del jugador, el sistema de santuarios y el viaje entre estos, se refactorizaran el jugador, los enemigos y las cámaras para que utilicen el patrón de diseño Estado y se mejora el *look-and-feel* del videojuego.

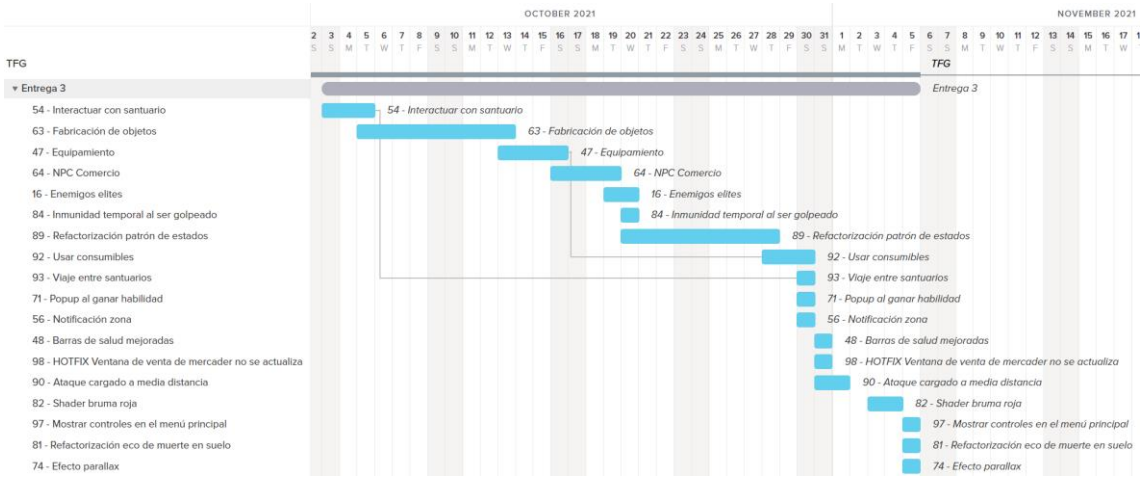

Figura 42: diagrama de Gantt de la Entrega 3

5.2 Creación del arte

De entre los muchos estilos artísticos 2D se ha escogido el estilo artístico *pixel art*, puesto que debido a su pequeño tamaño es mucho más fácil para dibujantes principiantes, se requiere menos tiempo en realizar un dibujo que con otros estilos artísticos, es perfecto para equipos de desarrollo con pocas personas y hace que el videojuego requiera muy pocos recursos, puesto que cualquier tarjeta gráfica o inclusive los gráficos integrados pueden trabajar holgadamente con los gráficos *pixel art*.

La creación del arte se hizo utilizando Aseprite, una aplicación de pago que está especializada en el estilo de arte *pixel art* y es muy fácil de usar. Se hizo un estudio de las dimensiones que usaban otros videojuegos, se decidió que el *canvas* del videojuego tuviera unas dimensiones de 320x180 píxeles debido a que su relación de aspecto es 16:9 y esta relación de aspecto es la más popular entre los monitores de ordenador. Cada pieza de arte de la interfaz de usuario tiene unas dimensiones máximas de 32x32 píxeles, mientras que el personaje y enemigos tienen unas dimensiones máximas de 32x64 píxeles.

5.3 Diseño de escenarios

El diseño de escenarios se hizo utilizando la herramienta *Tilemap* integrada en Unity. Donde se puede pintar cada *tile* individualmente por capas dándoles colisiones y efectos diferentes.

El escenario se divide en las siguientes capas:

- ***TerrainBackground***: Aquí se ponen los elementos con los que el jugador no colisiona, pero pueden verse de fondo y sirven para dar ambiente al escenario.
- ***TerrainBackgroundFront***: Al igual que la anterior capa, el jugador no puede colisionar con estos elementos, pero estos puestos delante del jugador en el eje Z por lo que da una sensación de profundidad en el escenario.
- ***TerrainCollision***: Esta capa se reserva para crear el suelo de los escenarios donde el jugador podrá colisionar y tendrán el tag *Ground*, importante para que el jugador sepa que está pisando suelo y no en el aire.
- ***TerrainCollisionWall***: Esta capa es para los elementos que hacen de pared donde podremos saltar sobre ellos deslizándonos por la pared y en caso de que sea un saliente, nos colgamos de este, la detección de la colisión con una pared se hace mediante la detección de la capa *Wall*.
- ***JumpOffPlatform***: Son las colisiones que tienen las plataformas por las que se pueden saltar hacia abajo. Estas tienen un componente añadido llamado PlatformEffector2D que nos permite desde abajo atravesar el *tile* sin colisionar. Esta capa no está incluida en todas las escenas.
- ***TerrainSpikes***: Es la capa donde se añade la colisión con las trampas de pinchos. Esta capa no está incluida en todas las escenas.

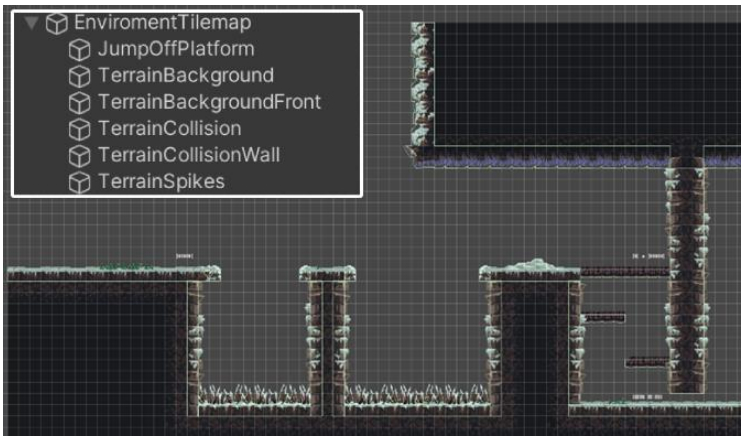

Figura 43: colisiones de las capas de *Tilemap* del nivel SnowLevel2

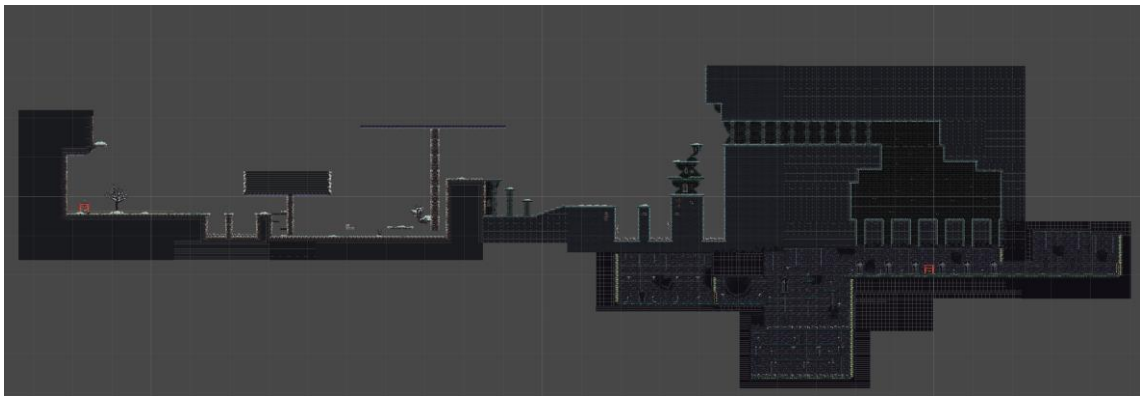

Figura 44: escenario completo

Cada área se divide en cinco segmentos de nivel debido a que así la carga de escenarios al ser estos muy pequeños se hace casi instantáneamente y no se requiere de pantallas de carga. Es similar al sistema de *chunks* que tienen los videojuegos de mundo abierto para evitar las pantallas de carga, donde el mapa se divide en trozos más pequeños del mismo tamaño y se van cargando, dependiendo de la posición del jugador y la cámara.

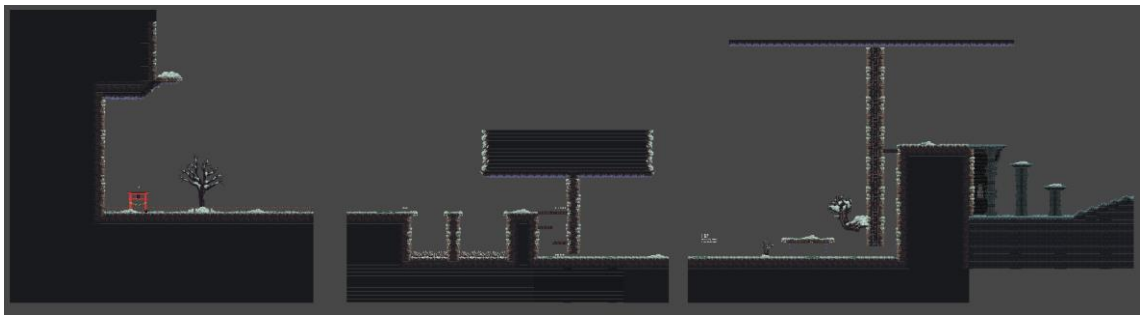

Figura 45: segmento uno, dos y tres del área Monte Fuji

Todos los segmentos tienen un *CameraLimit* que limitará el movimiento de la capa por la escena, *EnvironmentTilemap* que es donde se dibuja el escenario mediante una paleta de *tiles*, *MapSegments* que almacena todos los *triggers* que dibujaran la posición que tengan almacenada en el mapa. Los segmentos alternativamente pueden tener un santuario o un grupo generadores de enemigos.

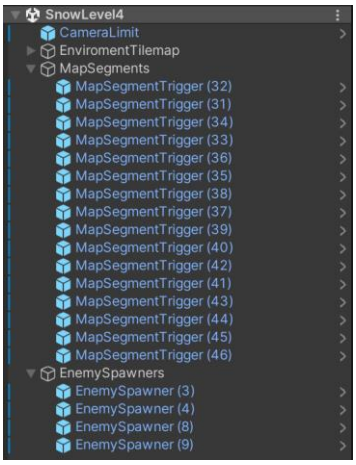

Figura 46: estructura común de un segmento de escenario

La inicialización de los controladores se hacen desde el menú principal donde no se destruirán al cambiar de escena mediante la función *DontDestroyOnLoad()* y en la escena *GameCore* que no se descarga una vez iniciada, puesto que todas las escenas que pertenecen a los segmentos de escenarios se cargan de forma aditiva.

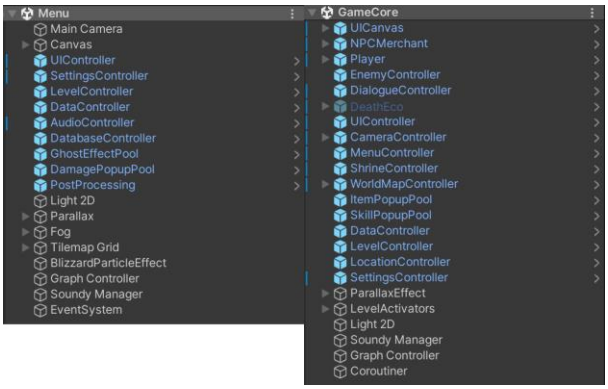

Figura 47: escenas principales

Mediante los *LevelActivators* se van cargando las escenas al colisionar con el jugador y se descargan cuando el jugador sale del trigger por lo que la mayor parte del tiempo solamente tendremos un segmento activado al mismo tiempo.

Figura 48: todos los cargadores de segmentos de escenario

5.4 Cámaras

Se ha usado el *asset Cinemachine que está* integrado en *Unity*, ya que proporciona un mayor control sobre la cámara básica de *Unity*.

En el proyecto se han desarrollado tres tipos de cámaras donde solo una puede estar activa al mismo tiempo.

- ***GameplayCamera***: es la cámara por defecto, esta sigue al jugador y está limitada por un *CameraLimit* del segmento de escenario en el que este actualmente. Cuando el jugador recibe o realiza algún tipo de daño, este activa el temblor por 100 milisegundos. Esta cámara muestra todas las capas del *Culling Mask* excepto la capa *Map*.
- ***GlimpseCamera***: es la cámara que se activa cuando el jugador otea, dejando de seguir al jugador y empezando a seguir al ratón. Esta cámara muestra todas las capas del *Culling Mask* excepto la capa Map.
- ***WorldMapCamera***: es la cámara que se activa cuando el jugador abre el inventario, está limitada por el *CameraLimit* del mapa y únicamente puede moverse a través de las teclas asignadas al movimiento del jugador. Esta cámara muestra solamente la capa *Map* del *Culling Mask.*

Todos los segmentos de escenario tienen un *CameraLimit* que actúa como límite de la cámara para que no siempre esté en el centro siguiendo al personaje y muestre más el entorno. Estos segmentos se asignan a *CinemachineConfiner2D* de la cámara principal al interactuar con ellos.

El *Culling Mask* se utiliza para decidir que se está renderizando en el momento que queramos. Para modificar su valor en *runtime* tiene que ser a través de operaciones de desplazamiento y operaciones de complemento bit a bit. Esto nos sirve para ahorrar recursos en caso de que estemos seguros de que algún elemento no se va a ver, mejorado considerablemente el rendimiento del videojuego.

Se usa la técnica de *Parallax* en el fondo para dar la sensación de que el fondo es inmenso debido a que son montañas. Esta técnica consiste en mover las diferentes capas a distintas velocidades dependiendo de su profundidad y de la posición del jugador.

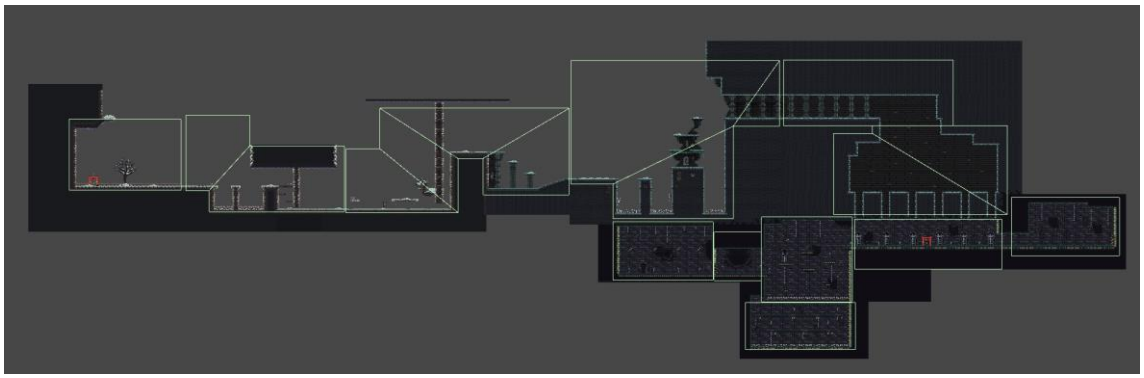

Figura 49: todos los límites de cámara de cada segmento de escenario

5.5 Jugador

Los *gizmos* son representaciones gráficas de la depuración en el editor de *Unity*, para que estos se muestren en las clases que correspondan debe de estar implementado la función *OnDrawGizmos()*.

En la siguiente imagen se muestran los *gizmos* del jugador. El área amarilla representa su zona de *parada* donde si el jugador recibe un ataque enemigo mientras está realizando una parada y colisiona con esta área se ejecutará un contraataque automáticamente, las áreas rojas representan

su zona de ataque donde dañara a los enemigos en un área dependiendo de si realiza un ataque hacia arriba como lateral, las líneas magenta representan los *raycast* para detectar las paredes, y si solo está detectando la pared el rayo inferior este detectará que el jugador está en un saliente y se agarrara a él, las líneas verde oscuro detectan si el jugador no puede levantarse mientras está esquivando, lo que provocará que esquive de nuevo hasta que pueda levantarse y por último, las líneas blancas de sus pies detectan si el jugador está tocando el suelo.

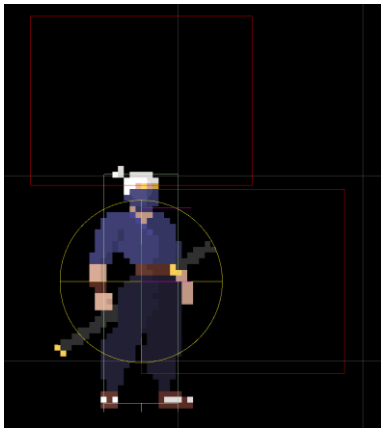

Figura 50: *gizmos* del jugador

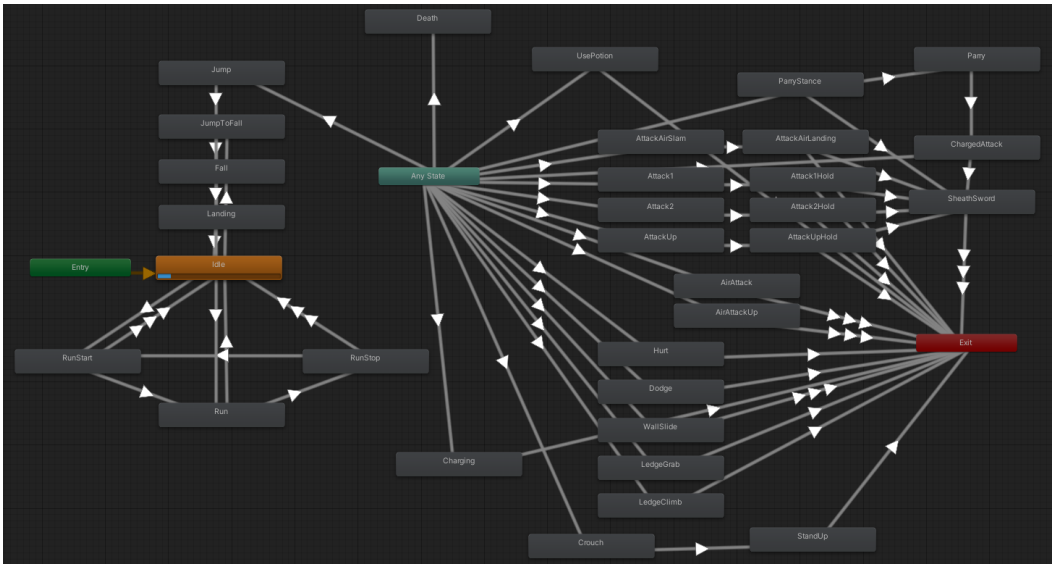

Figura 51: *animator* del jugador

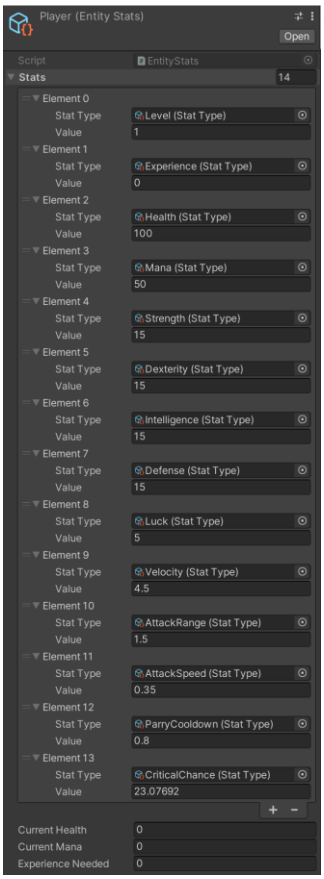

Figura 52: creación de las estadísticas del jugador

Para la creación del aspecto del personaje jugable se utilizó un *asset*, que se muestra al final del capítulo, como base debido a que el equipo no dispone de ningún artista profesional. Aunque se han redibujado todas las animaciones con nuestro personaje principal que sí es original, no quiero infringir el *copyright* por error, por lo que se mostrará únicamente algunas de las animaciones originales que se han creado.

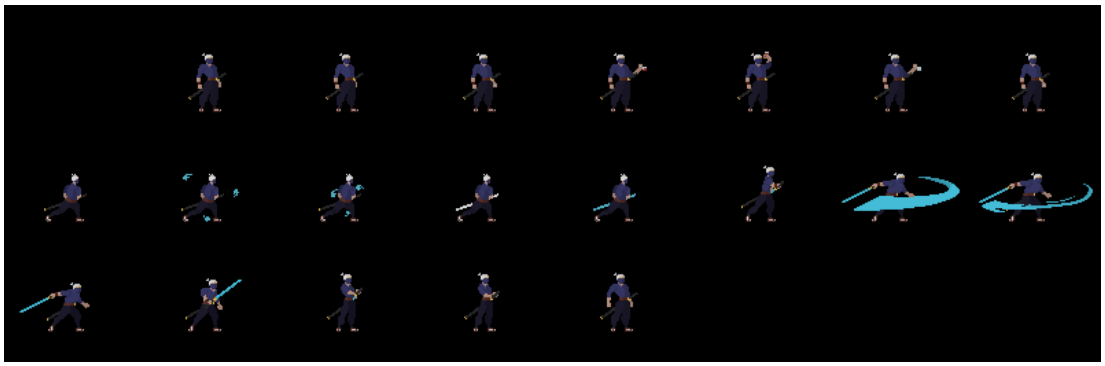

Figura 53: *spritesheet* parcial del jugador

Tanto en las animaciones de enemigos como del jugador tienen insertados eventos para realizar acciones en un *frame* especifico de la animación.

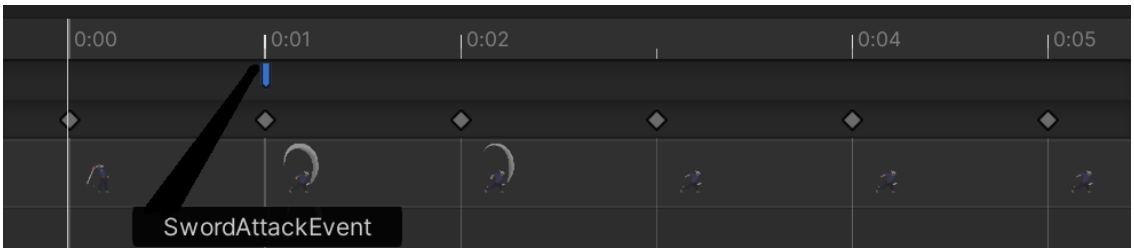

Figura 54: Evento de animación de la animación ataque hacia arriba

5.5.1 Movimiento

El jugador puede realizar una gran cantidad de movimientos diferentes:

- **Salto**: el jugador puede saltar si está en el suelo.
- **Doble salto**: el jugador puede volver a saltar una vez más en el aire.
- **Salto en pared**: cuando se tope con una pared, el jugador puede saltar en dirección opuesta a esta.
- **Salto hacia abajo**: si el jugador se encuentra sobre unas plataformas especiales, este puede saltar hacia abajo.
- **Esquivar**: el jugador puede esquivar lateralmente proporcionándole *i-frames*[15].
- **Agarrarse a un saliente**: cuando el jugador esté cerca de un saliente, se agarra a este automáticamente.

5.5.2 Combate

Todos los ataques del jugador y enemigos se realizan mediante las funciones *Overlap* de *Unity* que permiten obtener una lista de colisiones en un área definida en una capa específica del escenario durante un *frame*. Estas se ejecutan mediante eventos de animación y son mucho más eficientes que tener el personaje lleno de *triggers* activándolos y desactivándolos, ya que estas operaciones son costosas computacionalmente. Se pueden observar las áreas donde se capturan estos objetos en el área roja de la figura anterior de *gizmos* del jugador.

Para calcular el daño de un ataque normal del jugador y los enemigos se ha utilizado la siguiente fórmula:

(daño * (100 / (100 + defensa))).

Y en caso de que el daño del ataque sea crítico, seguirá la siguiente fórmula:

((daño * (2 + exceso crítico)) * (100 / (100 + defensa)))

El **exceso crítico** se calcula como **(probabilidad crítica / 100)** si la probabilidad es mayor al 100%, en caso contrario será cero. Puede superarse el 100% al equiparse objetos.

Todos los ataques aquí comentados tienen una probabilidad basándose en su destreza de hacer daño crítico y se calcula con la siguiente fórmula:

100 * (1 - (100 / (100 + 2 * destreza)))

[15] Cuadros donde el jugador es invencible.

Al derrotar un enemigo obtendremos experiencia y con suficiente experiencia subiremos de nivel y ganaremos un punto de maestría para gastar en los santuarios para aumentar una estadística principal.

La experiencia necesaria para subir de nivel en cada nivel se calcula como:

(50 * nivel actual^3 * 0.7)

Además, el jugador puede realizar una gran cantidad de ataques diferentes:

- **Ataque lateral**: ataque normal del jugador que daña a los enemigos, este puede realizarse tanto en el suelo como en el aire.
- **Ataque hacia arriba**: es un ataque normal hacia arriba, este puede realizarse tanto en el suelo como en el aire.
- **Ataque hacia abajo**: este ataque debe realizarse mientras se está en el aire, al realizarlo se lanzará hacia abajo con su espada dañando todo lo que se tope con él, mediante un trigger que se activará al entrar en este estado, hasta llegar al suelo donde saldrá una explosión que hará daño de áreas.
- **Ataque cargado**: este ataque debe realizarse mientras se está en el suelo, el jugador carga su ataque donde al liberarlo proporcionará un ataque fuerte y una onda de energía que desaparecerá a media distancia y puede destruir proyectiles enemigos.
- **Contraataque**: este ataque debe realizarse mientras se está en el suelo, el jugador puede ponerse en posición de contraataque donde si el enemigo ataca este ignorará el daño y soltará un ataque fuerte.
- **Beber poción**: esta acción debe realizarse mientras se está en el suelo, el jugador puede beber una poción que recupera salud y estas pociones se regeneran al interactuar con un santuario.

5.6 Enemigos

En las siguientes subsecciones se muestra las áreas de acción de los enemigos y sus animaciones.

Además, cabe destacar que las recompensas (almas y experiencia) que sueltan los enemigos al ser derrotados se calcula con la siguiente fórmula:

(Valor base * nivel * 1.1) * multiplicador élite

Cada enemigo tiene un valor base de recompensas y si el enemigo es élite, su multiplicador será 1.7, en caso contrario, será 1.

5.6.1 Farolillo maldito

En la siguiente imagen se muestran los *gizmos* del farolillo maldito. El área verde oscuro representa su zona de amenaza donde si un jugador está dentro de esta le seguirá, el área roja representa su zona de ataque donde ya podrá lanzar ataques tanto principales como secundarios y el área magenta representa la zona de explosión que genera al ser derrotado.

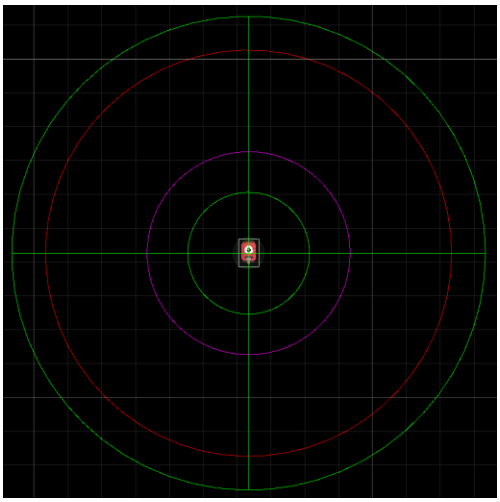

Figura 55: *gizmos* del farolillo maldito

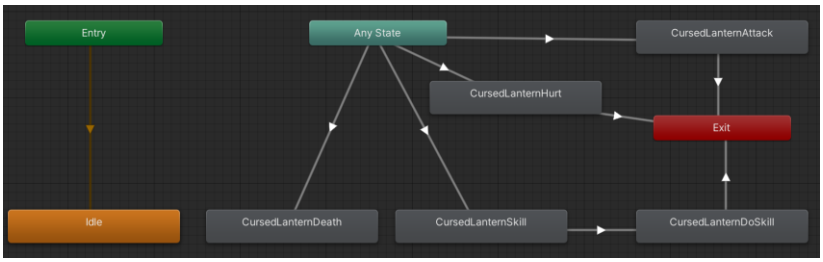

Figura 56: *animator* del enemigo farolillo maldito

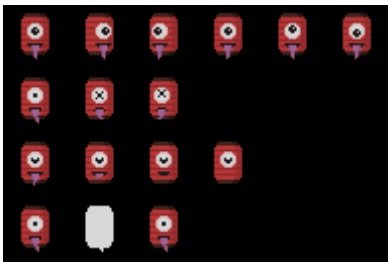

Figura 57: *spritesheet* del Farolillo maldito

5.6.2 No-muerto

En la siguiente imagen se muestran los *gizmos* del no-muerto. El área verde oscuro representa su zona de amenaza donde si un jugador está dentro de esta le seguirá, el área roja representa su zona de ataque donde ya podrá lanzar ataques tanto principales como secundarios.

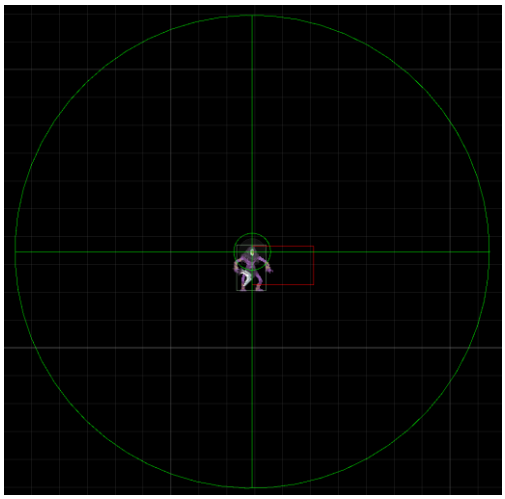

Figura 58: *gizmos* del no-muerto

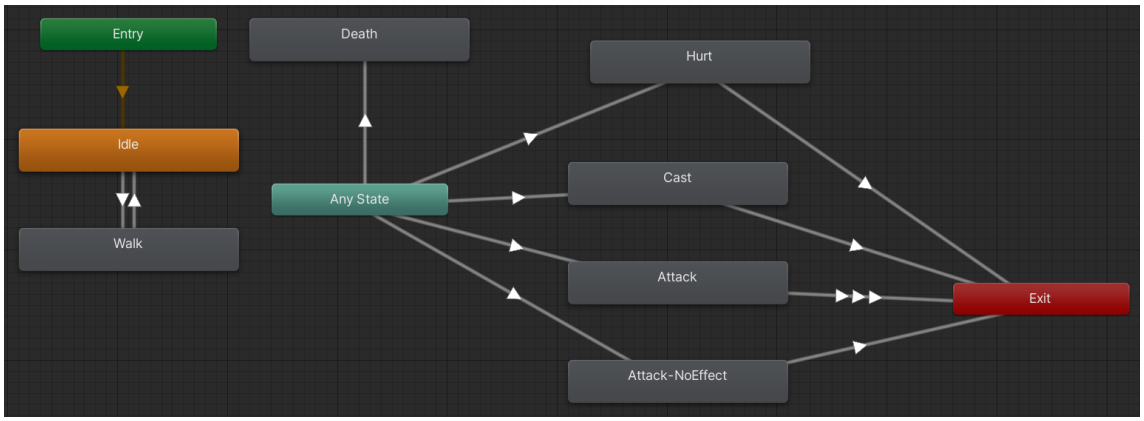

Figura 59: *animator* del enemigo no-muerto

5.6.3 Generador de enemigos

Los generadores de enemigos instanciaran un enemigo de los que tienen en su lista aleatoriamente y con un rango de nivel definido por el generador. Estos enemigos tienen una baja probabilidad de generarse como enemigos de élite que aumentan considerablemente sus estadísticas, pero también se recibe recompensas mucho mejores al derrotarlos.

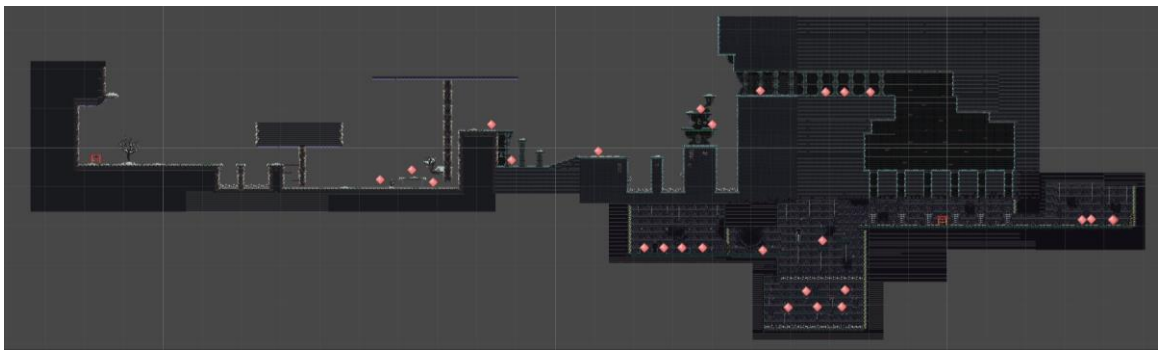

Figura 60: localización de los generadores de enemigos

5.7 Menú del videojuego

5.7.1 Inventario

Al derrotar enemigos habrá una probabilidad de que estos puedan soltar algún objeto, estos pueden ser recogidos y si son armaduras o consumibles los podemos equipar para poder utilizarlos y aumentar las estadísticas del jugador.

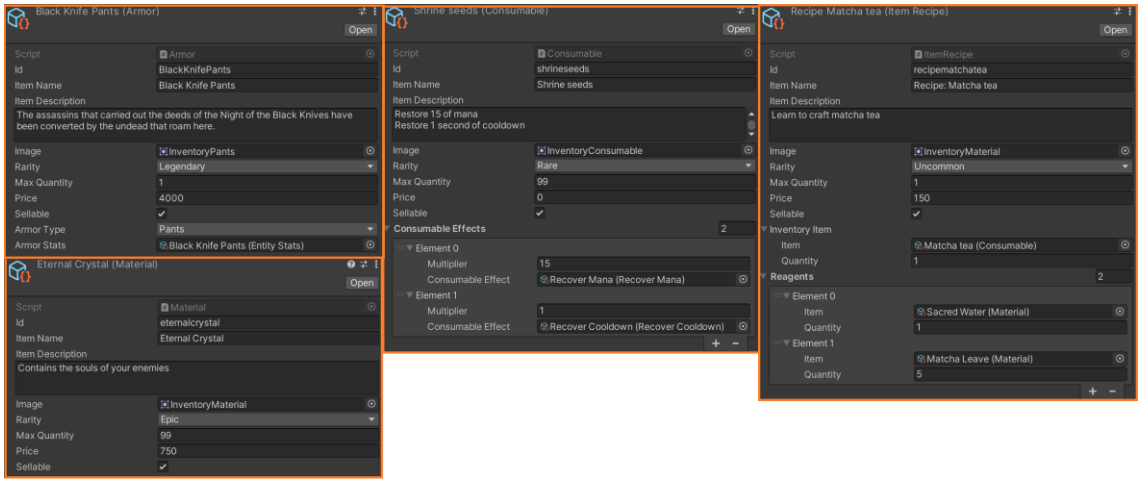

Figura 61: creación de objetos

5.7.2 Libro de habilidades

Al derrotar enemigos habrá una pequeña probabilidad de que podamos obtener alguna de las habilidades que dispone el enemigo. Estas podrán ser equipadas desde el libro de habilidades localizado en el menú del videojuego.

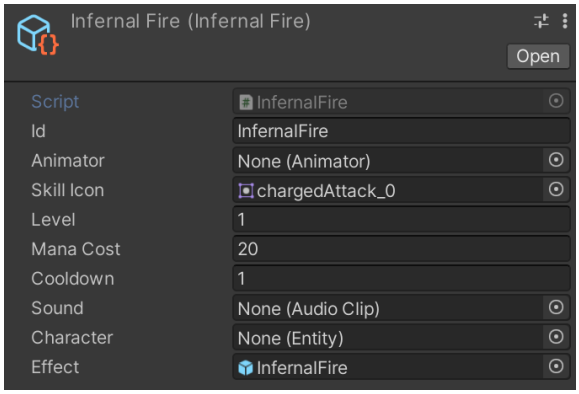

Figura 62: creación de habilidades

5.7.3 Fabricación de objetos

Desde el menú del videojuego se pueden crear objetos con una cantidad predefinida, si disponemos de la receta, mediante otros objetos, principalmente materiales.

5.7.4 Mapa

El escenario está repleto de *triggers* que al colisionar con ellos actualizan el segmento de mapa que tengan asignado y actualizan la posición de la marca del jugador en el mapa.

Figura 63: todos los *WorldMapTrigger* de los segmentos de escenario

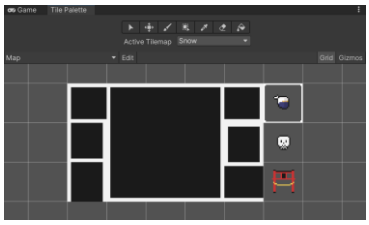

Figura 64: paleta de *tiles* utilizada en el mapa

Como podemos observar, el mapa tiene diferentes capas:

- ***Snow***: se muestra el área del Monte Fuji (área azul).
- ***Prison***: se muestra el área del dominio de los condenados (área verde).
- ***Background***: es una capa totalmente opaca que oculta el resto de las capas y serán estos *tiles* los que se desdibujen al colisionar los *triggers* del escenario.
- ***Marks***: esta capa muestra la localización de los santuarios.
- ***PlayerMark***: esta capa muestra la posición del jugador.

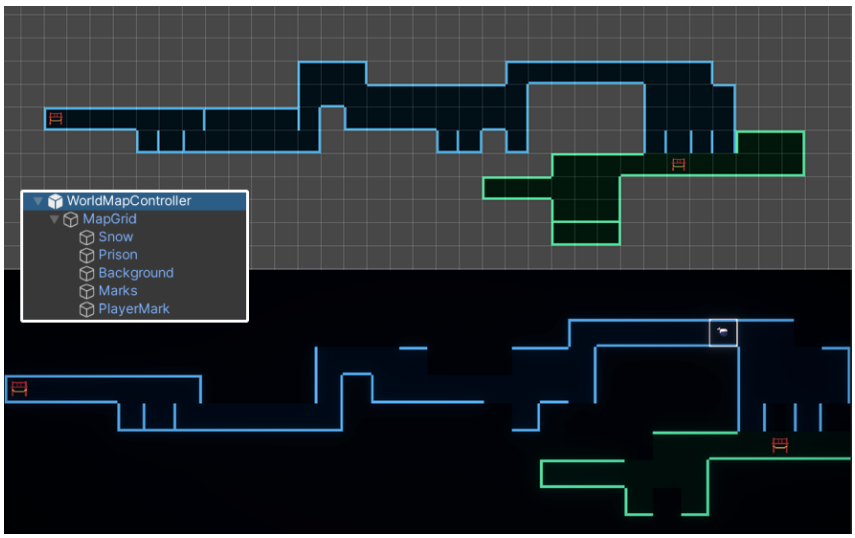

Figura 65: estructura del mapa y muestra desde el editor y el videojuego

5.7.5 Opciones

Desde el menú del videojuego se puede modificar el volumen tanto de la música como de efectos de sonido, se puede escoger de una lista la resolución y calidad del videojuego y se puede activar y desactivar la pantalla completa, la sincronización vertical y el temblor de pantalla.

5.8 NPC

Se han creado los siguientes tipos de NPC:

- ***NPC* normal**: este tipo de *NPC* únicamente puede comenzar diálogos al ser interactuado por el jugador.
- ***NPC* Mercader**: este tipo de *NPC* no solo puede iniciar diálogos, sino que también puede comerciar con el jugador, puede vender objetos de entre su lista de objetos y comprar los objetos del jugador.

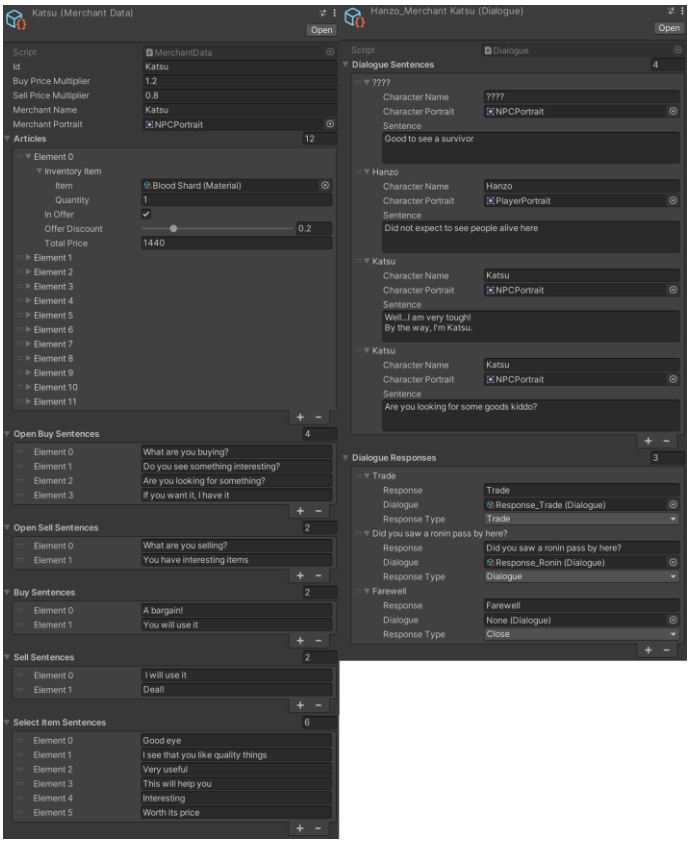

Figura 66: creación de diálogos y datos del mercader

Figura 67: *spritesheet* del *NPC*

5.9 Santuario

Los santuarios son los lugares de paz donde el jugador puede descansar después de pelear contra multitud de enemigos. Al interactuar con los santuarios, estos recuperan la salud del jugador por completo, restaurará las pociones que el jugador haya usado y hace reaparecer a los enemigos derrotados. Además, permite la subida de las estadísticas principales (fuerza, inteligencia y destreza) por medio de puntos de maestría que son ganados con cada nivel de experiencia que el jugador sube, también, permite guardar la partida y viajar rápidamente a otros santuarios que haya interactuado el jugador.

Figura 68: *sprite* de los santuarios

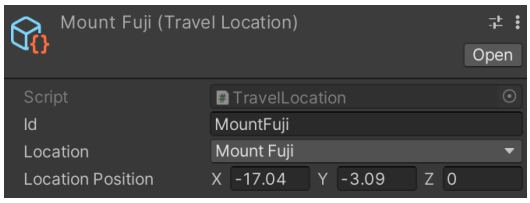

Figura 69: creación de puntos de viaje

5.10 Persistencia

Para el guardado y cargado de los datos del videojuego se ha utilizado la librería *BinaryFormatter* para serializar y deserializar los datos.

Se ha requerido de dos archivos, uno en el que guarde los datos del jugador y el inventario y otro en el que se guarde la configuración del sistema.

Además, se han creado multitud de bases de datos que almacenan distintos tipos de objetos y que pueden llamarse fácil y rápidamente para recuperar el objeto deseado mediante la búsqueda por identificador.

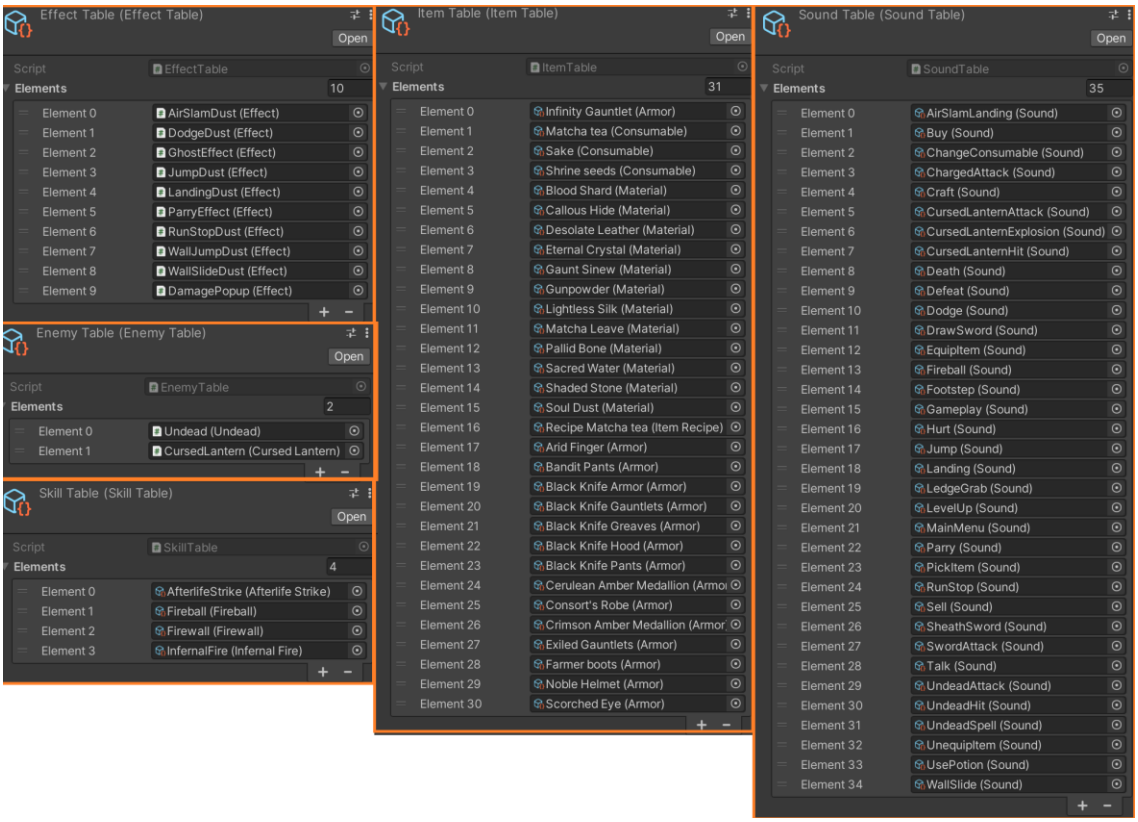

Figura 70: creación de bases de datos

5.11 Diagrama de clases

El diagrama de clases es enorme, con más de 170 clases, por lo que se ha decidido mostrarlo en módulos y se irá explicando la funcionalidad de estos.

Se ha hecho uso de clases *MonoBehaviour*, esta es la clase base que todo componente de Unity debe utilizar para poder instanciarse dentro del juego.

También se ha usado la clase *ScriptableObject* que funcionan como un contenedor de datos donde esta no puede instanciarse dentro del juego al contrario que con la clase *MonoBehaviour*. Estas facilitarán la creación de objetos desde el editor, reduciendo así la cantidad de clases que se deben crear. El principal uso que se le da a esta clase es la reducción del uso de memoria evitando que se copien valores.

Utilities: se han creado atributos propios de nuestro sistema para facilitar la obtención de colores y textos personalizados de los tipos Enumerados mediante la utilización del método *GetAttribute()* genérico. Además, se han creado métodos de extensión para las clases *TravelLocation*, *Vector3* y *DeathEco* que nos permiten transformarlas a clases serializadas para guardarlas en un archivo de guardado y deserializar las clases serializadas para cuando se cargue una partida. Esto es extremadamente útil y necesario en el sistema de guardado que se ha creado en el proyecto.

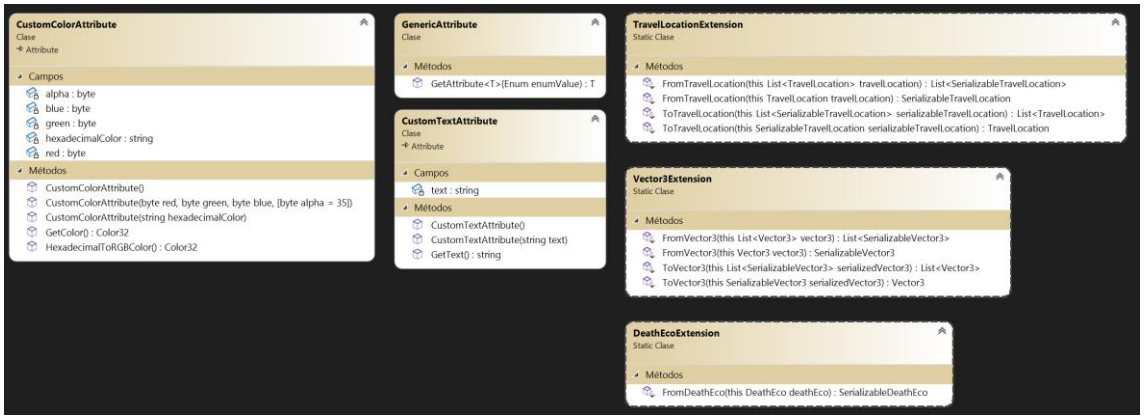

Figura 71: diagrama de clases de utilidad

Sistema de carga de escenarios: la carga de escenarios se hará mediante *LevelActivator* donde nuestro personaje al colisionar con el *trigger* este se disparará cargando el nivel que tenga seleccionado de entre los que hay disponibles en la clase enumerada y al salir del *trigger* este se descargará por lo que solamente habrá como máximo dos niveles cargados a la vez reduciendo la memoria que tiene que utilizar el sistema. También cuando se llegue a un nuevo nivel con una localización distinta a la que se tenía, este mostrará un mensaje en pantalla con el nombre del escenario.

Figura 72: diagrama de clases de la carga de escenarios

Sistema de cámaras: en el videojuego habrá tres tipos de cámaras distintas:

- ***WorldMapCamera***: se usa para visualizar el mapa del videojuego.
- ***GlimpseCamera***: se utiliza cuando el jugador utilice el oteo.
- ***GameplayCamera***: será la que siga al jugador durante el videojuego y esta será *Pixel Perfect* para provocar un mejor efecto visual al usuario.

Con *ParallaxEffect* movemos las capas del fondo a distintas velocidadades dependiendo de la posición del jugador.

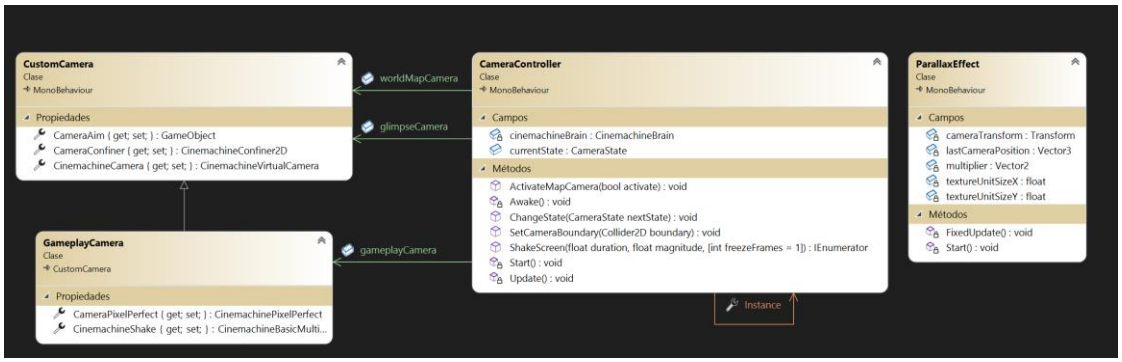

Figura 73: diagrama de clases del sistema de cámaras

Base de datos: en el videojuego se han implementado varias bases de datos simples que permiten realizar una consulta dándoles solamente el identificador del objeto que queremos encontrar donde devuelven el único objeto que tenga ese identificador. Estas bases de datos son heredadas de una base de datos genérica donde se implementa el patrón de diseño Singleton para que cada una de estas bases de datos pueda ser llamada individualmente desde las distintas clases del videojuego. *DatabaseController* utiliza el patrón de diseño Fachada que se encarga de crear e inicializar todas las bases de datos del videojuego, esto es especialmente útil a la hora de probar la aplicación pues que no se requiere de tener una instancia de cada base de datos, sino que pueden ser todas creadas solamente teniendo una instancia de *DatabaseController* ahorrando código innecesario. Se usa para invocar enemigos, objetos, efectos, habilidades y efectos de sonido.

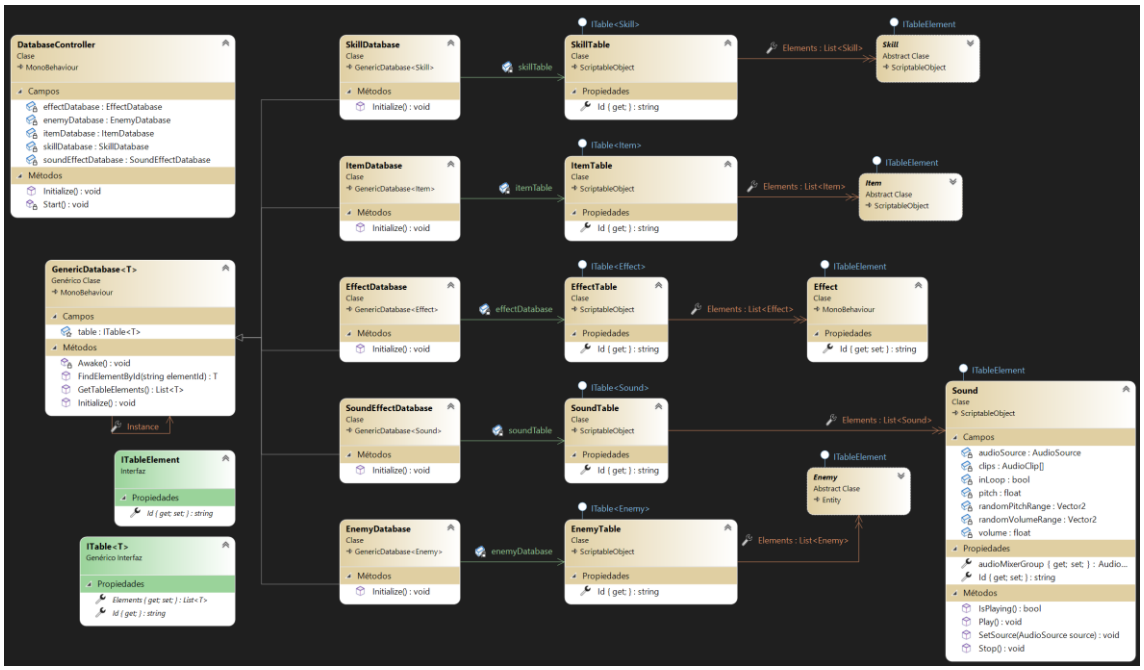

Figura 74: diagrama de clases de las bases de datos

Sistema de diálogos: los diálogos en el videojuego se controlan desde el *DialogueController* donde se inician mediante la función *StartDialogue()*. Estos se componen de varias frases y si es menester, varias respuestas.

Las respuestas pueden ser de varios tipos:

- ***Trade***: al pulsar en este tipo de respuesta se abrirá la ventana de comercio donde podremos tanto comprar los objetos del mercader como vender objetos de nuestro inventario.
- ***Dialogue***: al pulsar en este tipo de respuesta nos lleva a nuevos diálogos, creando así un árbol de diálogos.
- ***Close***: al pulsar en este tipo de respuesta cierra la ventana de diálogo.

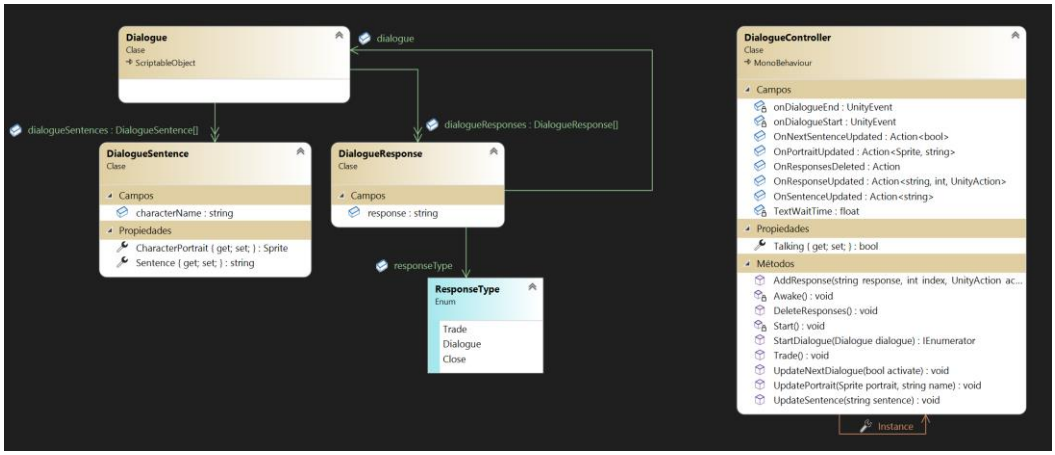

Figura 75: diagrama de clases del sistema de diálogos

Sistema de carga y guardado de datos: el sistema guarda y carga objetos en el disco duro del usuario escribiéndolos en formato binario para que los usuarios no hagan trampas fácilmente. Esto se controla llamando a la clase *DataController* con las funciones *LoadGame()* y

SaveGame(). Se guardan distintos datos en distintos archivos para agilizar la carga de estos, separando los datos de la partida del jugador y las opciones del videojuego.

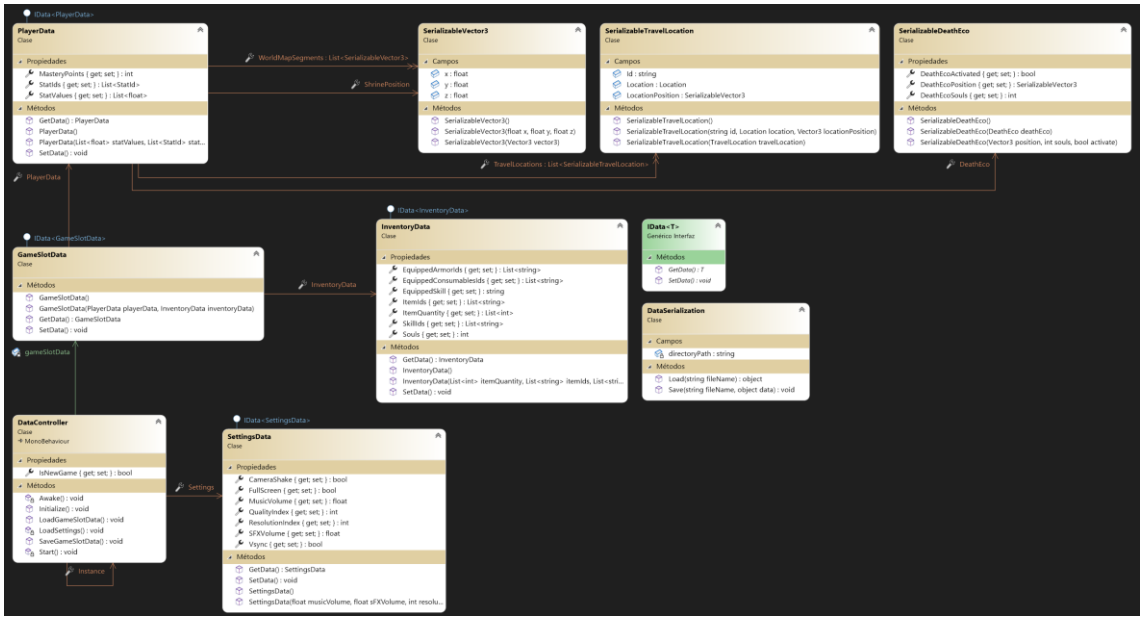

Figura 76: diagrama de clases del sistema de guardado y cargado de datos

Sistema de objetos: los objetos del videojuego que pueden recogerse dentro del videojuego pertenecen a la clase *ItemInstance* donde este se compone de uno o muchos *InventoryItem*, que tienen una cantidad y un objeto de cualquier tipo. Cuanto mayor sea la rareza del objeto, mayores serán sus estadísticas.

Los objetos pueden ser:

- **Armaduras**: el jugador puede equipárselos y desequipárselos para aumentar sus estadísticas. Puede equiparse varios a la vez, ya que tenemos distintos tipos de armadura. Los tipos de armadura que podemos equipar al mismo tiempo son el casco, pechera, guantes, pantalones y botas.
- **Material**: son los principales componentes necesarios para la fabricación de objetos.
- **Consumibles**: estos pueden ser equipados y desequipados al igual que las armaduras, pero estos pueden ser usados y consumidos en el proceso, aportándonos varios efectos, ya sean beneficiosos o perjudiciales. Estos efectos se crean desde la clase *ConsumableEffect*, donde tenemos a *RecoverCooldown* donde restaurará tiempo de enfriamiento de tu habilidad equipada y maná y *RecoverMana* que solo recuperará una cierta cantidad de maná al jugador.
- **Recetas**: las recetas se añadirán automáticamente a la lista de objetos fabricables donde serán necesarios objetos de todo tipo, pero los más comunes serán los materiales. Estos podrán ser utilizados para fabricar objetos consumibles y armaduras.

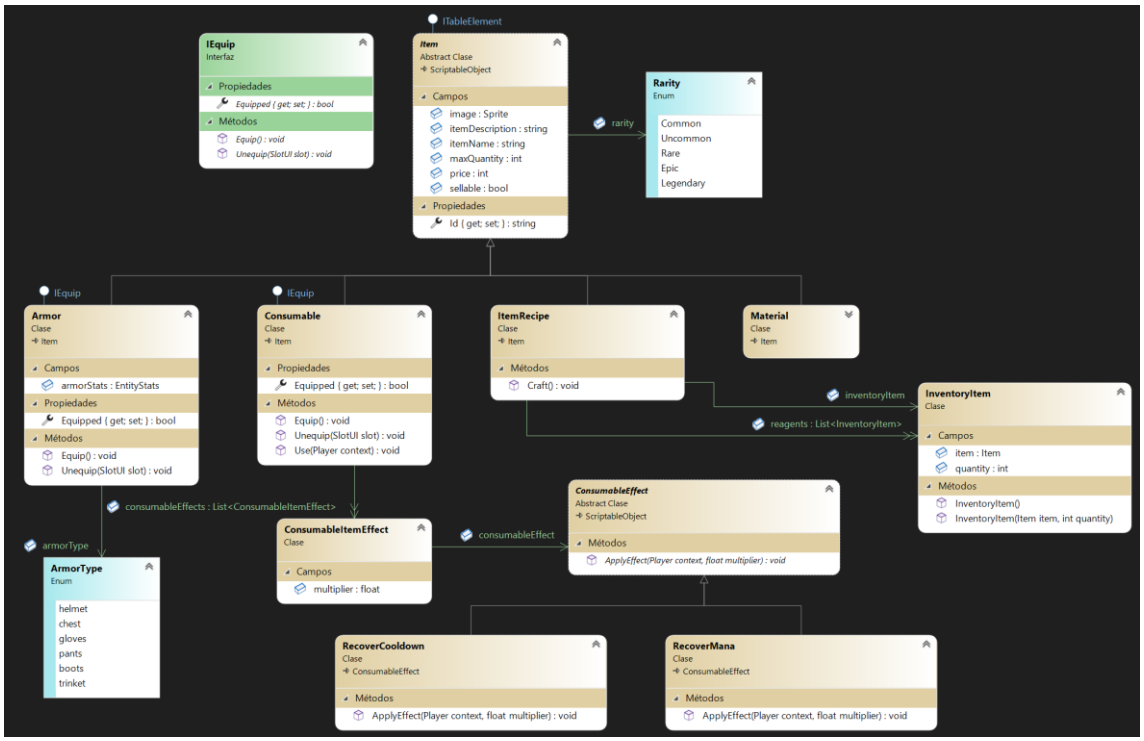

Figura 77: diagrama de clases de los objetos

Habilidades: el jugador tiene una amplia cantidad de habilidades que puede utilizar a discreción, en estas entidades se utiliza un patrón de diseño estrategia para usar la habilidad que tenga equipada en ese momento. Los enemigos también pueden utilizar habilidades, pero estas serán fijas del enemigo.

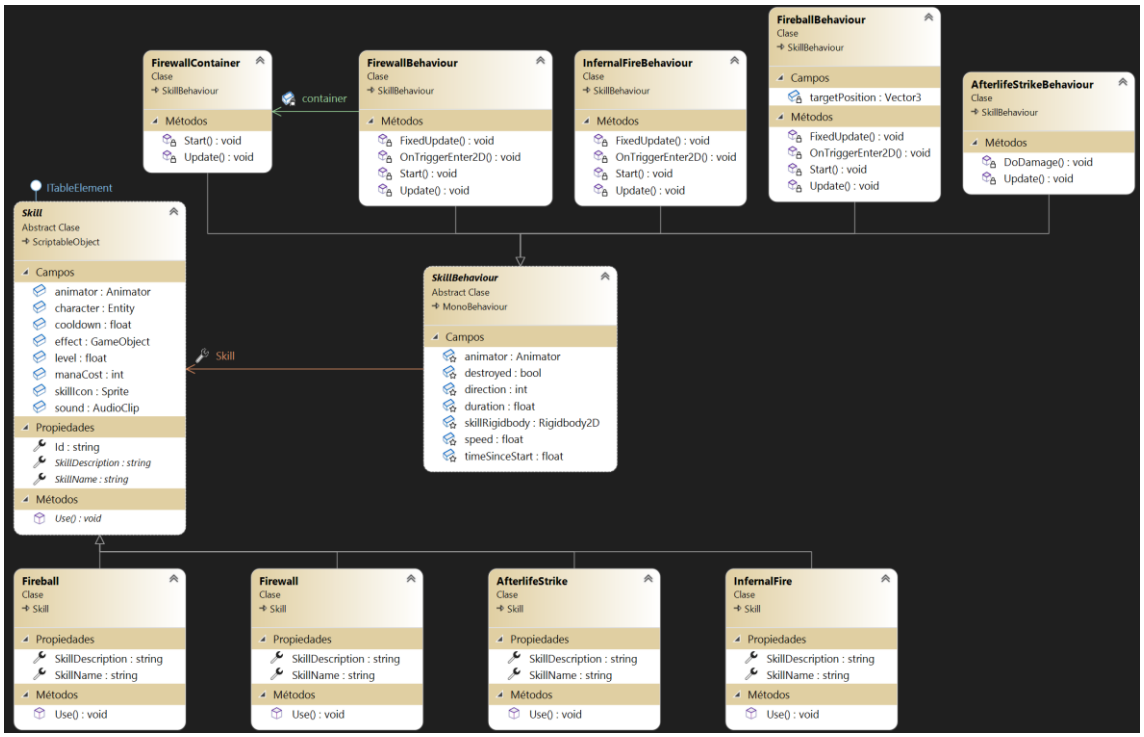

Figura 78: diagrama de clases de las habilidades

Interactuables: los componentes interactuables son aquellos con los que el jugador puede interactuar y cada uno provocará un efecto distinto.

Se tiene varios tipos de interactuables en el videojuego:

- ***NPC***: son los aliados con los que podremos empezar un diálogo para recabar información sobre la situación actual del videojuego.
- ***Merchant***: estos también son *NPC* y pueden realizar la misma función de empezar diálogos, pero además puede comerciar con el jugador, pudiendo comprar sus objetos y vender los que tenga asignado en *MerchantData*.
- ***Shrine***: desde los santuarios podremos viajar a otros santuarios que hayamos descubierto, podremos guardar la partida y subir de nivel al jugador, aumentando así una estadística elegida en un punto por nivel.
- ***ItemInstance***: es un objeto instanciado en el mundo que contiene una lista de objetos que ha dejado caer un enemigo y que puede ser recogido por el jugador, añadiendo todos los objetos que contiene al inventario del jugador.
- ***DeathEco***: al morir se creará un eco de muerte en la última localización donde el jugador haya tocado suelo, evitando así que se genere encima de las trampas o sitios imposibles de llegar. Este eco almacena todas las almas que tenía el jugador al morir, dejando al jugador con cero almas al revivir. Si el jugador logra llegar al eco e interactuar con él, este devolverá todas las almas que haya almacenado de la última muerte, recuperará la salud del jugador por completo y desaparecerá hasta que el jugador muera de nuevo.

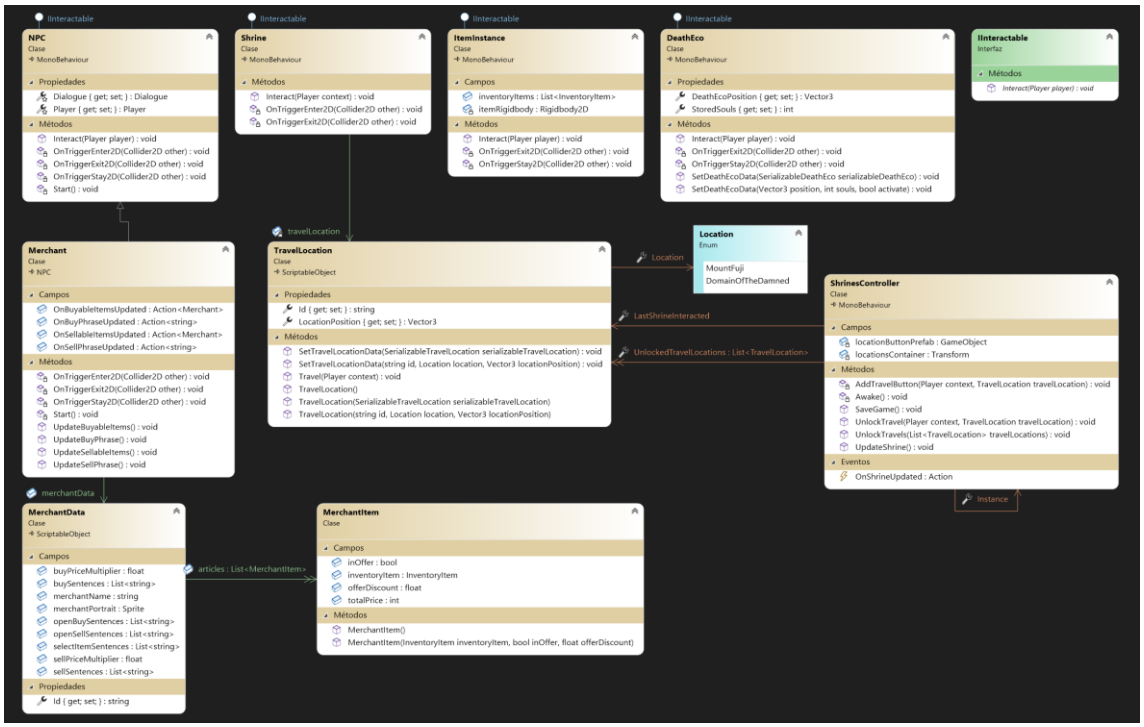

Figura 79: diagrama de clases de los interactuables

Estados: el jugador, los enemigos y las cámaras tiene su propio *FSM (Finite State Machine)* donde se utiliza el patrón de diseño Estado para cambiar el comportamiento de estos componentes.

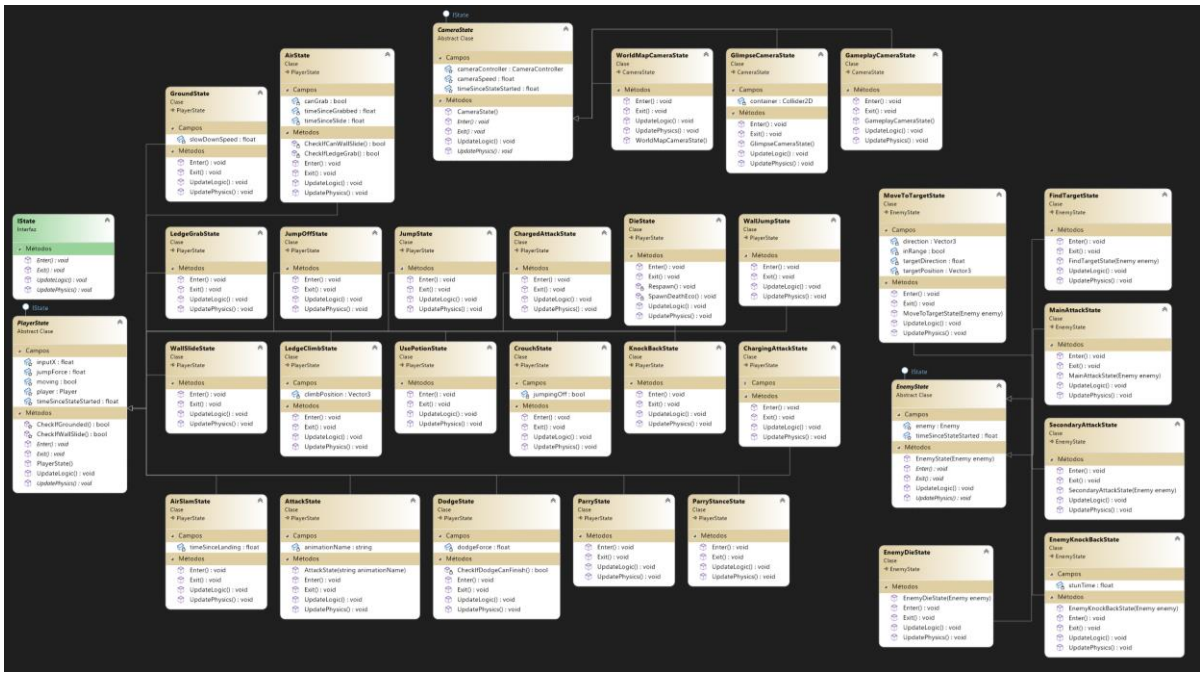

Figura 80: diagrama de clases de los estados del videojuego

Entidades: estas son las clases encargadas de la lógica de negocio del jugador y a los enemigos.

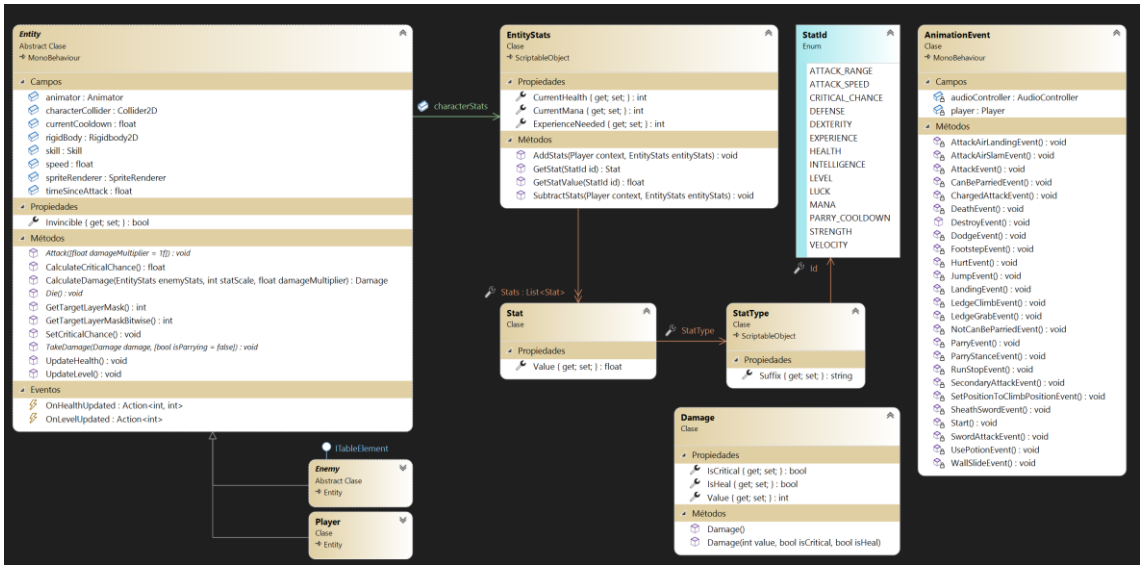

Figura 81: diagrama de clases de las entidades

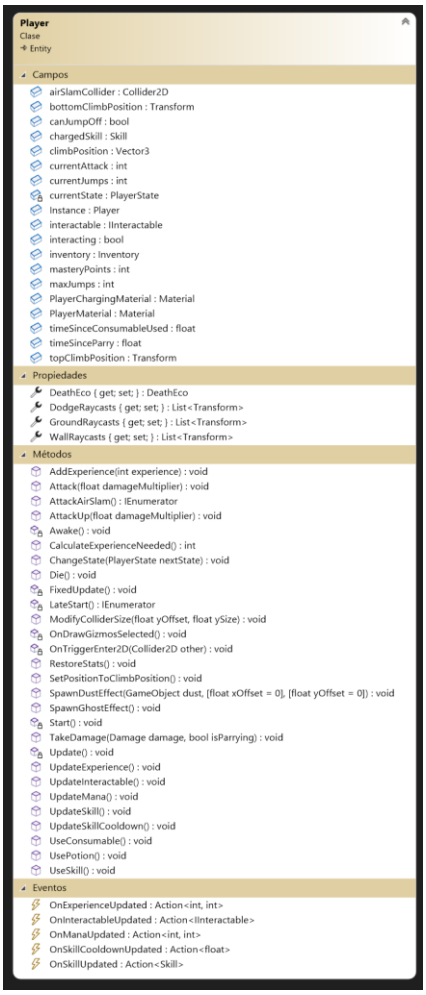

Figura 82: diagrama de clases del jugador

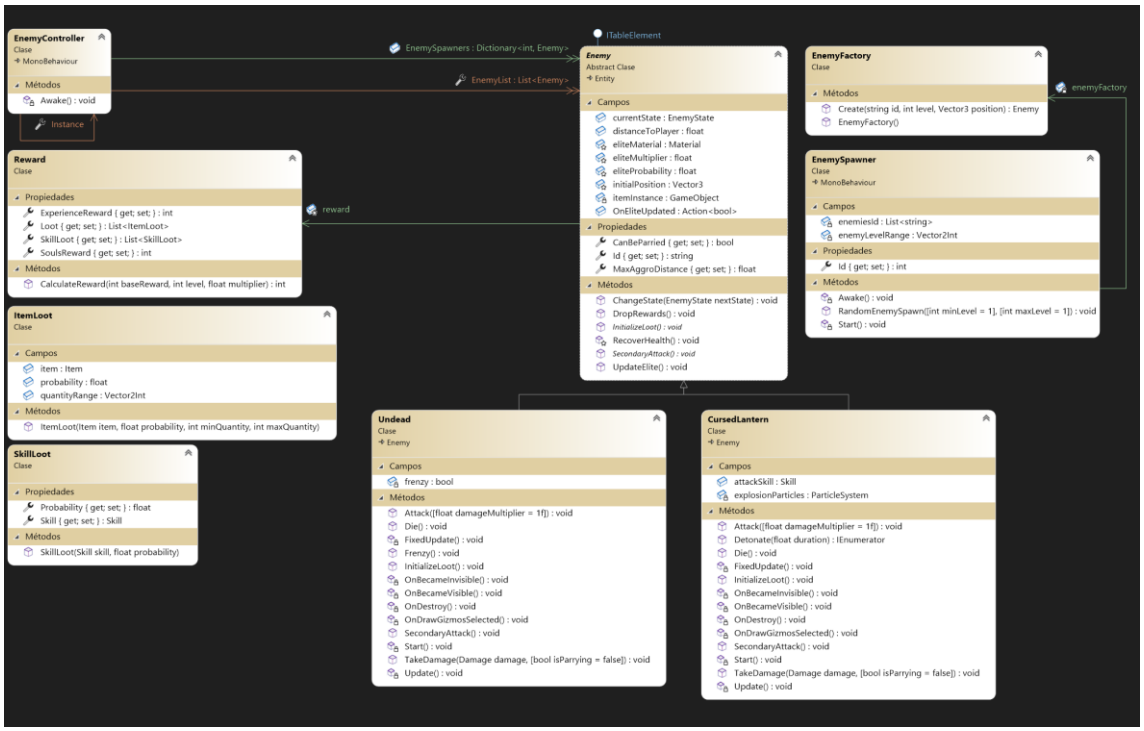

Figura 83: diagrama de clases de los enemigos

Sistema de mapas: con *WorldMapTrigger* desbloquearemos un segmento del mapa y actualizará la posición del jugador en el mapa llamando a la función *DrawMapSegment* del *WorldMapController*, estos están por todo el escenario.

Figura 84: diagrama de clases del sistema de mapas

Inventario: en el inventario guardaremos todos los objetos del jugador, sus pociones actuales, el consumible equipado, sus objetos equipados y sus almas. Se utiliza el patrón de diseño observador para actualizar los valores en la interfaz de usuario, puesto que estas interfaces se suscribirán a esta clase y se actualizarán los valores mediante el uso de eventos. También guardamos una referencia a la lógica de la fabricación, que almacenará las recetas, y el libro de habilidades, que almacenará las habilidades conseguidas.

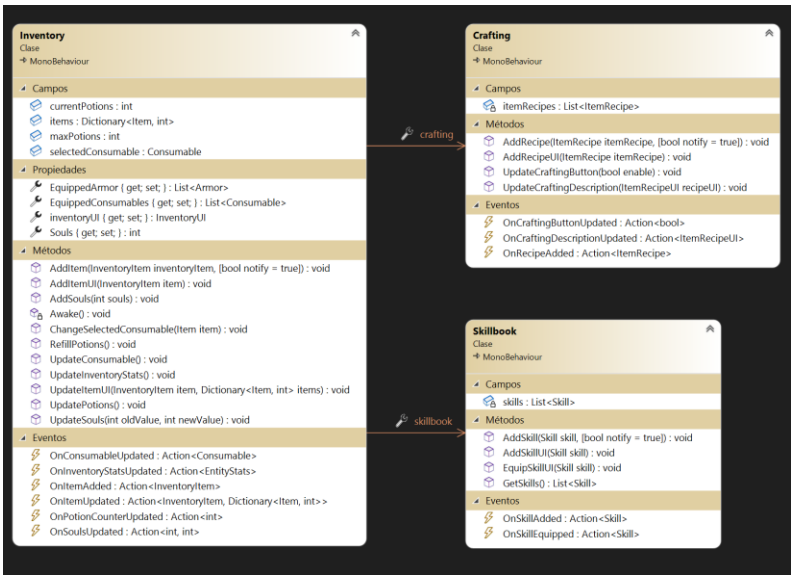

Figura 85: diagrama de clases del inventario

Piscinas de objetos: en las piscinas de objetos se ha usado el patrón *Object Pool* para la reutilización de objetos que se crean y destruyen constantemente, mejorando así significativamente el rendimiento del juego, estos efectos son:

- Los *DamagePopup* se crean al golpear a una entidad o recibir una curación, mostrando el daño que hemos hecho o nos han hecho.
- Los *Popup* se crean al recoger objetos y habilidades, mostrando así su información.

- Los *GhostEffect* se crean al esquivar con el jugador, realizar un ataque hacia abajo, un contraataque o realizar un ataque cargado, estos solamente sirven para dar una mejor sensación visual al jugador.

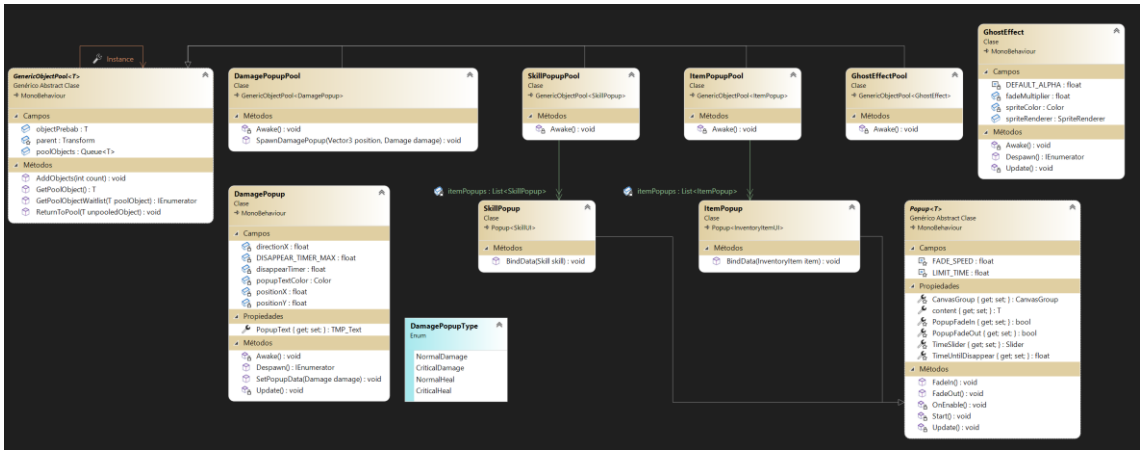

Figura 86: diagrama de clases de las piscinas de objetos

Opciones del videojuego: estas son las clases pertenecientes a los componentes personalizados que encontramos en el menú de opciones.

Con *ListSettingUI* podemos crear un elemento que almacene una lista de valores donde se podrá seleccionar uno de estos. Es similar a un elemento *Dropdown*, pero cambiando los valores mediante botones laterales y no mediante un desplegable.

Con *SliderSettingUI* podemos crear un elemento que controle un valor mediante un control deslizante. Se ha usado para controlar el volumen de la música y los efectos de sonido.

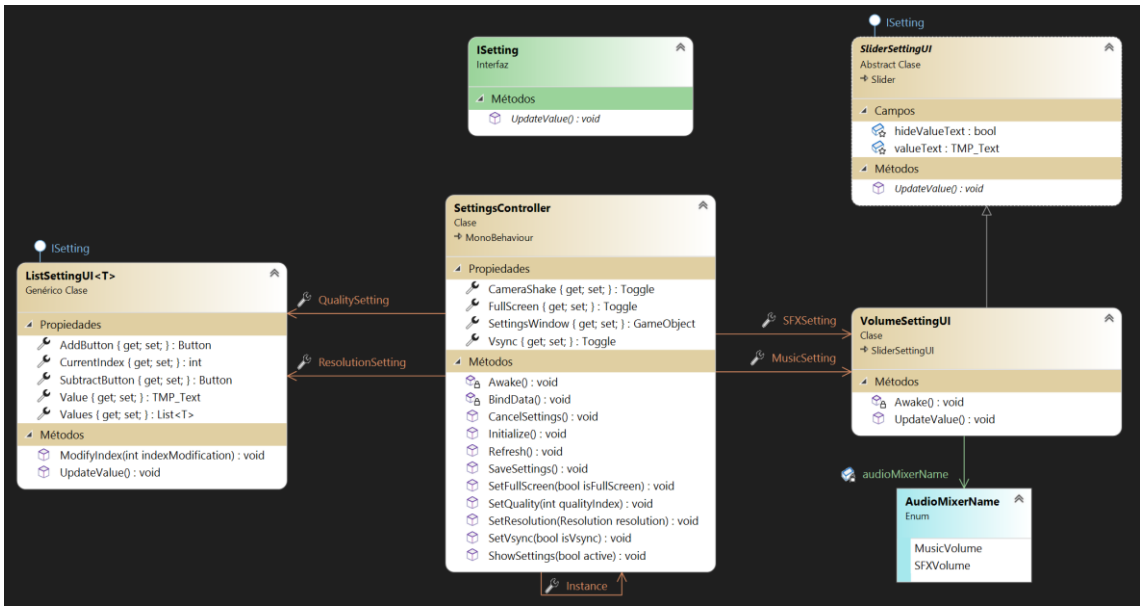

Figura 87: diagrama de clases de las opciones del videojuego

Plataformas: solo se ha implementado un tipo de plataforma en el videojuego, esta se trata de una plataforma que nos permite saltar hacia abajo cuando el jugador est´´a agachado y al saltar sobre ella desde abajo la atravesará sin provocar una colisión.

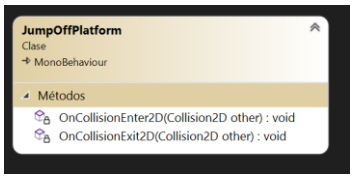

Figura 88: diagrama de clases de las plataformas

Trampas: únicamente se ha implementado un tipo de trampa en el videojuego, esta se trata de una trampa de pinchos que al colisionar con el jugador o un enemigo le provocará daño letal.

Figura 89: diagrama de clases de las trampas

Audio: *AudioController* utiliza el patrón de diseño Singleton para no ser destruido si es cambiado de escena y que el resto de las clases pueda comunicarse fácilmente con esta clase. Su función es de instanciar cada efecto de sonido en un nuevo objeto con su propio *AudioSource* y se encarga de reproducir los sonidos que hay en la base de datos. Esto nos ayuda a poder reproducir varios sonidos al mismo tiempo.

Figura 90: diagrama de clases del audio

UI Diálogo: clases encargadas de mostrar por pantalla la interfaz de usuario de los diálogos.

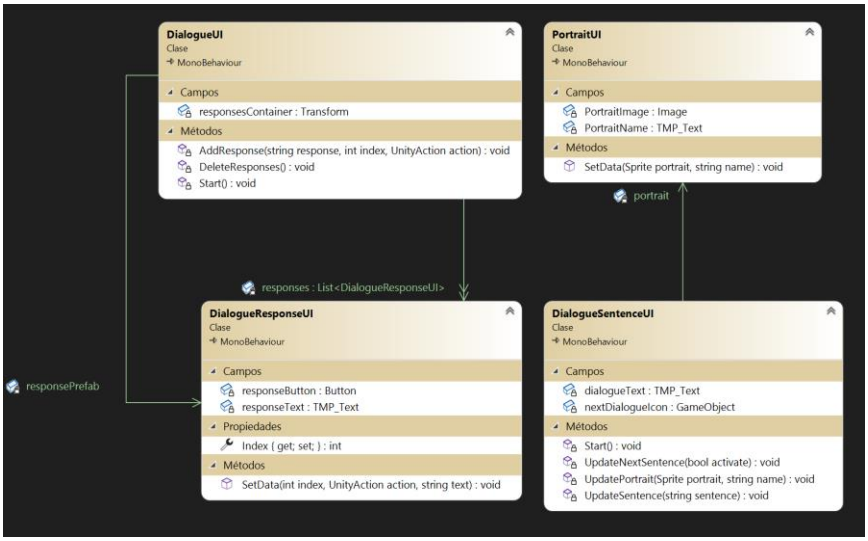

Figura 91: diagrama de clases de la UI de los diálogos

UI Entidad: clases encargadas de mostrar por pantalla la interfaz de usuario del jugador y enemigos.

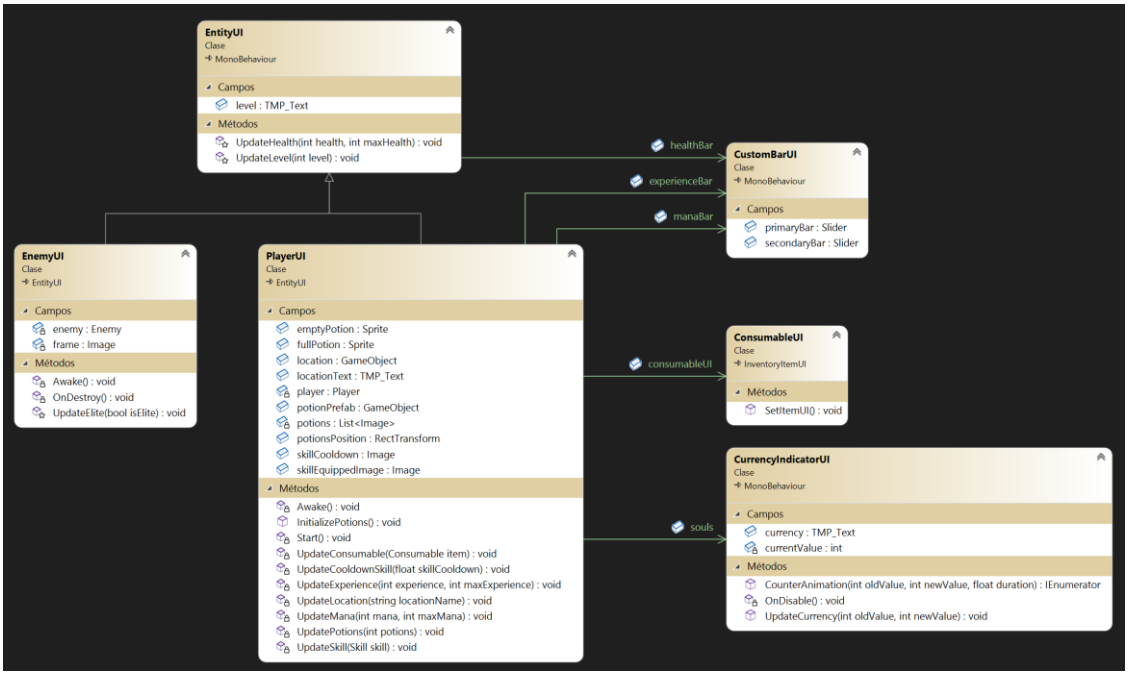

Figura 92: diagrama de clases de la UI de las entidades

UI Inventario: clases encargadas de mostrar por pantalla la interfaz de usuario de las ventanas donde se muestran objetos.

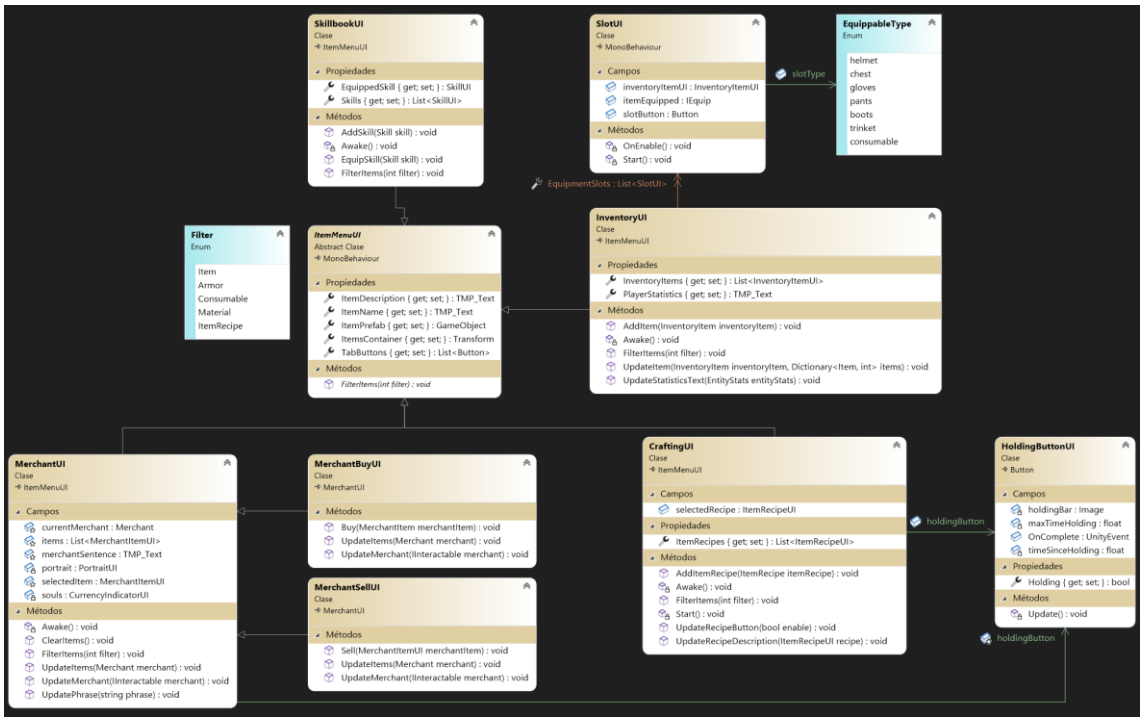

Figura 93: diagrama de clases de la UI del inventario

UI Objeto: clases encargadas de mostrar por pantalla la interfaz de usuario de los objetos.

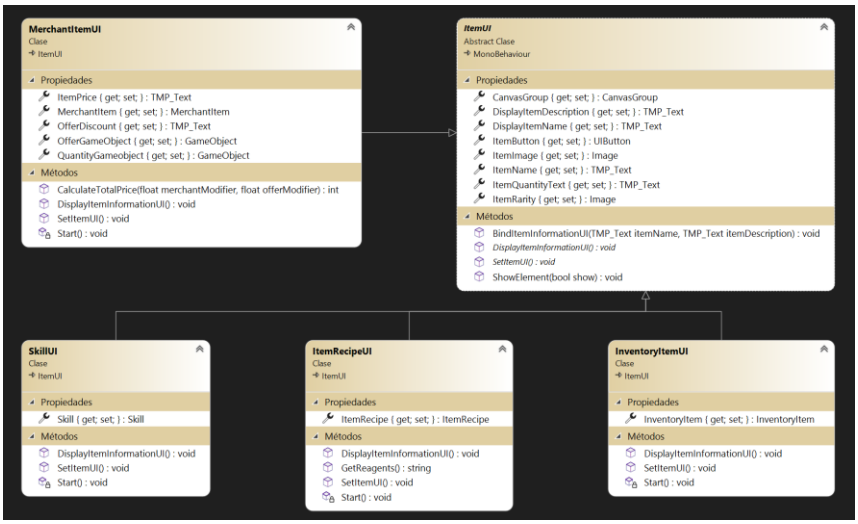

Figura 94: diagrama de clases de la UI de los objetos

UI Santuario: clases encargadas de mostrar por pantalla la interfaz de usuario de las estadísticas principales del jugador y poder aumentarlas y mostrar sus puntos de viaje.

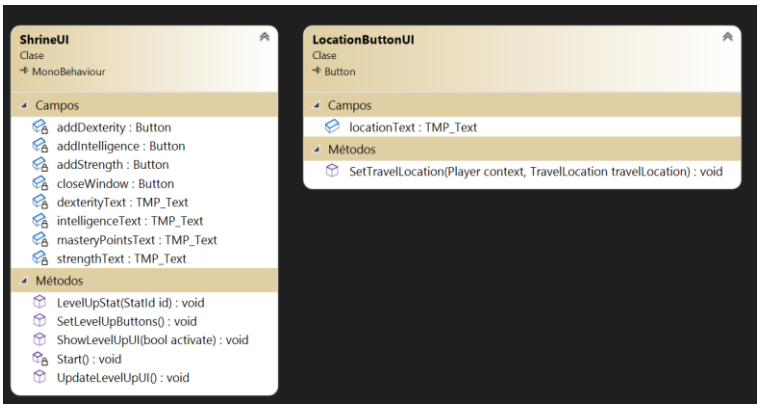

Figura 95: diagrama de clases de la UI de los santuarios

UI Menú: clases encargadas de controlar la interfaz de usuario.

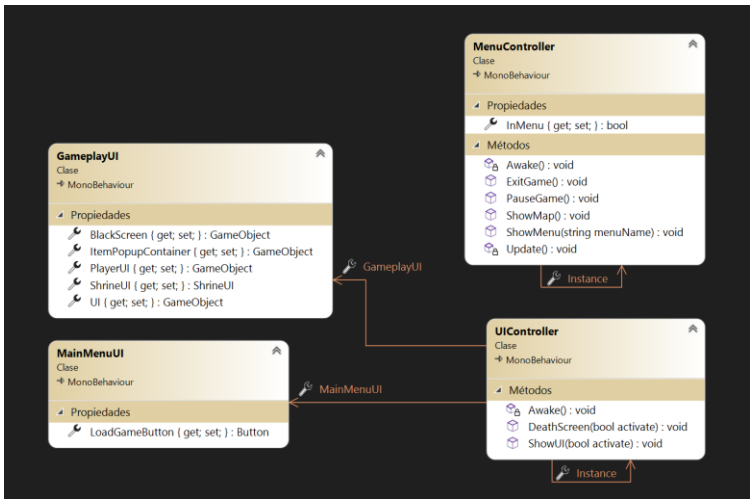

Figura 96: diagrama de clases de la UI del menú

5.12 Audio

Se ha dividido la fuente de sonido *Master* en *Music* que se encarga de gestionar el volumen de la música de fondo y *SFX* que se encarga de gestionar el volumen de los efectos de sonido tales como el de atacar o saltar. Esto es debido a que están vinculadas a los parámetros expuestos *MusicVolume* y *SFXVolume* que se modifican desde la clase *VolumeSliderUI* la cual se puede encontrar en las opciones del videojuego.

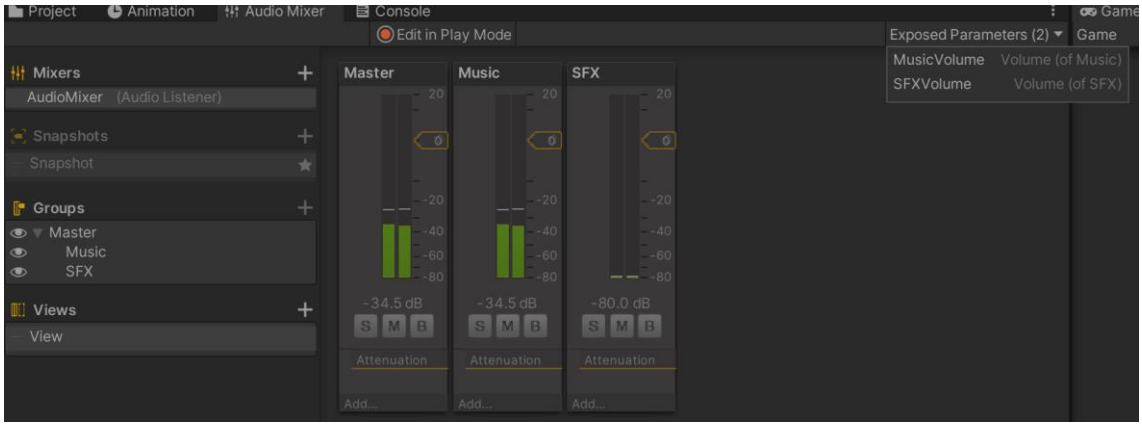

Figura 97: *AudioMixer* de The last hanyou

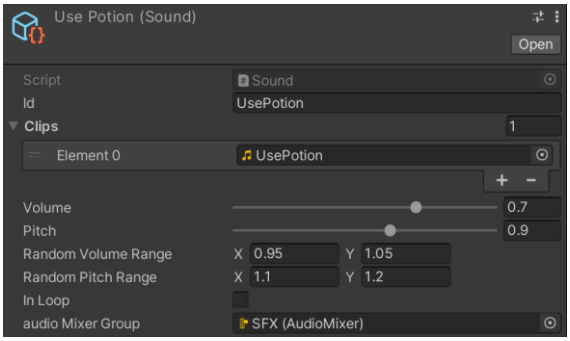

Figura 98: creación de sonidos

5.13 Editor

Las clases que heredan de objetos pertenecientes al código fuente de *Unity* deben tener su propia clase de editor para exponer los nuevos campos añadidos.

Para las clases *VolumeSettingUI*, *HoldingButtonUI* y *LocationButtonUI* se han creado sus clases de editor *VolumeSliderEditor*, *LocationButtonEditor* y *HoldingButtonEditor* lo que facilita el uso de componentes personalizados y aumenta la rapidez con la que se crea el videojuego. En la siguiente imagen se resaltan los campos añadidos en los componentes.

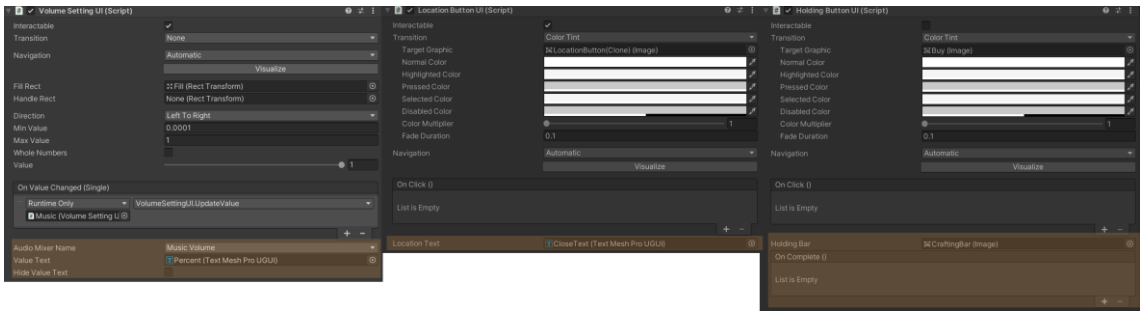

Figura 99: extensión de componentes de *Unity* en el editor

5.14 Orden de ejecución las clases

A medida que la aplicación crece, es necesario especificar un orden de inicialización en algunas de las clases, ya que, por defecto, el orden de inicialización de los *scripts* de *Unity* no es determinista y podría darse el caso de que una clase dependa de algún dato de otra y provoque errores.

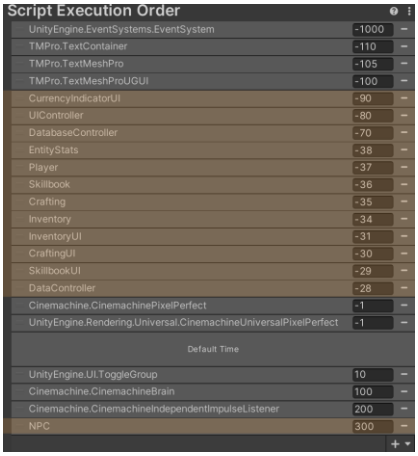

Figura 100: orden de ejecución de las clases

Por ejemplo, en la clase *InventoryUI* para mostrar los objetos en pantalla que tenemos en el inventario necesitamos que el inventario se inicialice antes, pero este necesita que la base de datos de objetos se inicialice primero, ya que debemos poblar el inventario del jugador con los objetos obtenidos.

5.15 Shaders

El fondo rojo que simula la bruma provocada por los *yokai* se ha realizado mediante la herramienta *Shader Graph* ya integrada en *Unity*. Esta herramienta permite la creación de *shaders* de manera visual con la programación por nodos.

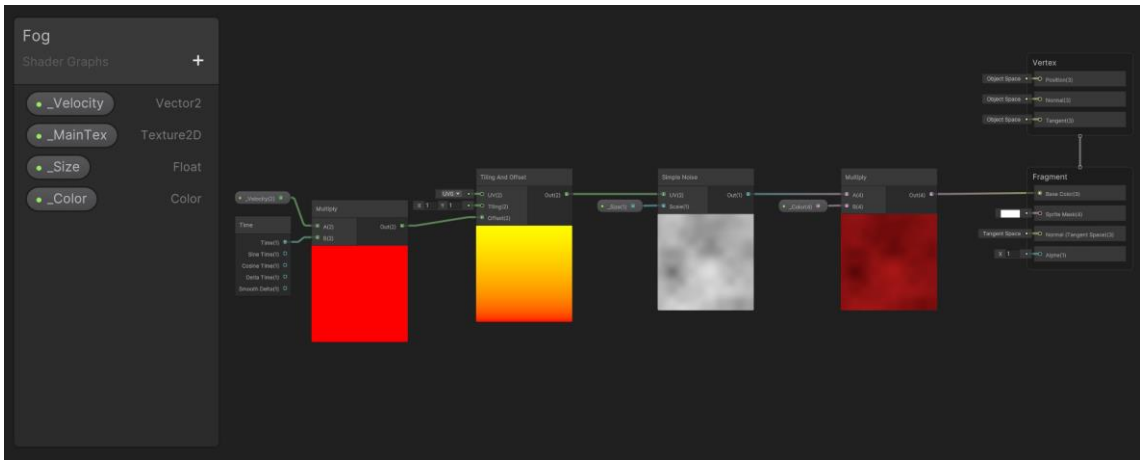

Figura 101: *Shader Graph* de la bruma roja

5.16 Postprocesado

Se utilizó la siguiente configuración del postprocesado[16] para dar un ambiente lúgubre a los escenarios.

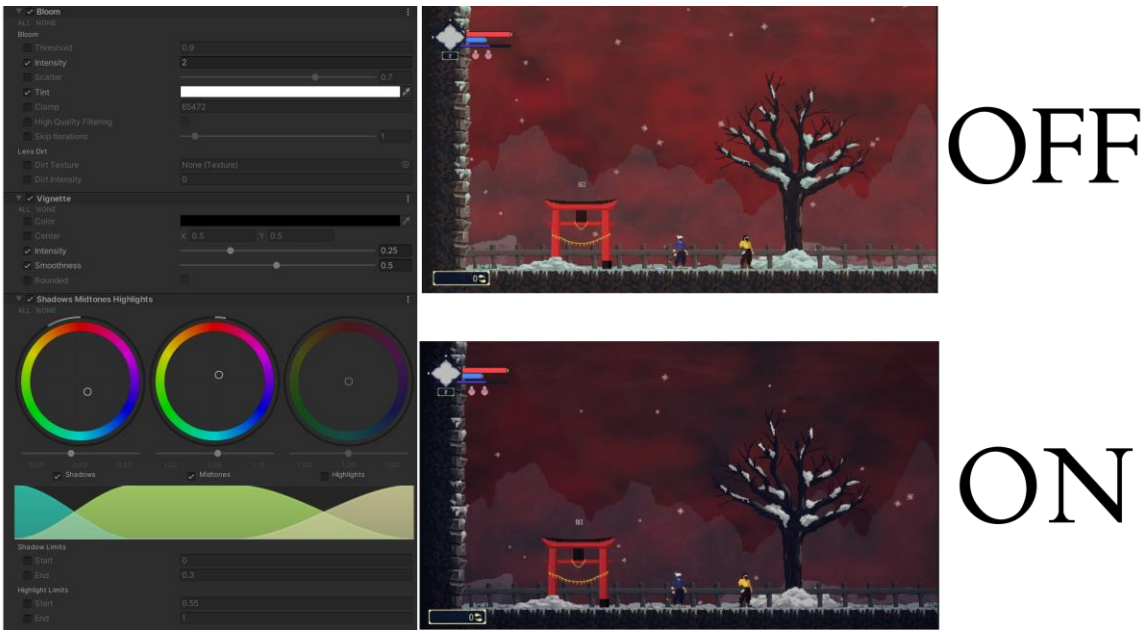

Figura 102: configuración de postprocesado y comparación

5.17 Assets de terceros usados

Debido a la gran magnitud de recursos necesarios de distintas disciplinas, se ha optado por la utilización de algunos *assets* de la *asset store*[17] de *Unity* para agilizar la realización del proyecto.

5.17.1 Programación

DoTween es un asset gratuito que permite crear animaciones complejas en la interfaz de usuario mediante código.

Doozy UI Manager es un asset de pago que actúa como gestor de interfaces de usuario que permite fácil navegación entre estas interfaces de usuario mediante la creación de nodos.

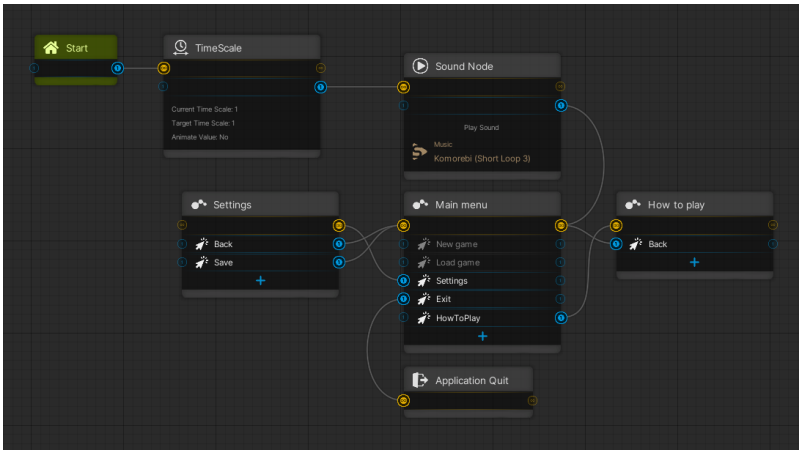

Figura 103: navegación del menú principal en DoozyUI

[16] Mejora la calidad de la imagen mediante efectos y filtros.
[17] Tienda digital de *assets* de *Unity*.

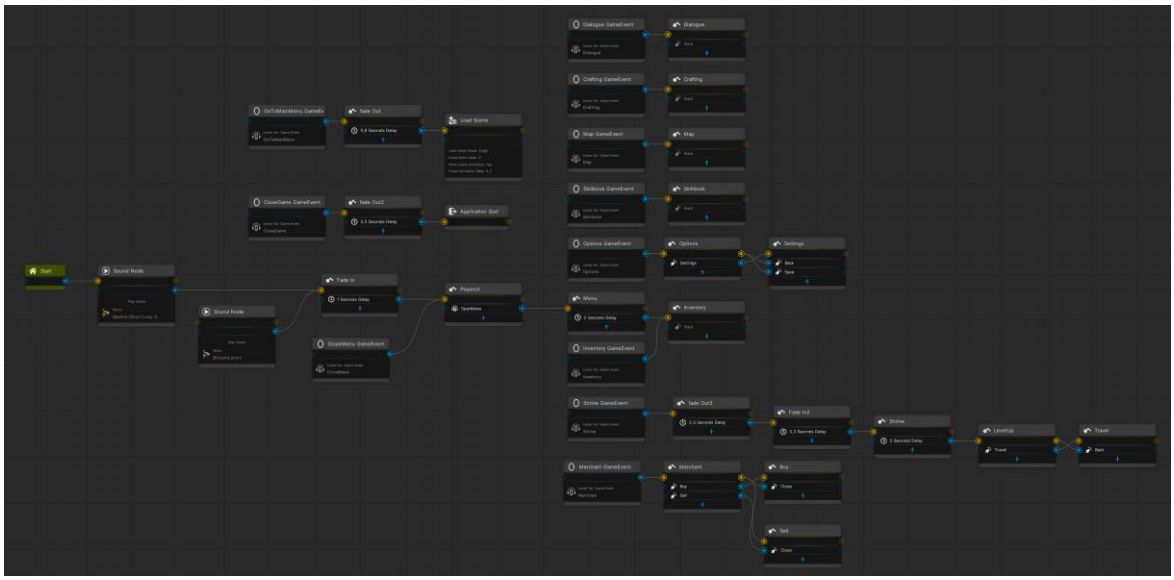

Figura 104: navegación del *gameplay* en DoozyUI

5.17.2 Arte

Los *shaders* para la visualización de enemigos élite y para el aura del ataque cargado del jugador han sido creados con el *asset* de pago ***AllIn1SpriteShader***.

La base del arte del personaje jugador ha sido sacada del *asset* de pago ***Prototype Hero - Pixel Art*** donde se rehicieron las animaciones para ajustarse al aspecto que se quería tener del personaje principal del videojuego y para la realización de las acciones que puede realizar el jugador donde se añadieron nuevas animaciones creadas en *Aseprite*.

El enemigo no-muerto ha sido sacado del *asset* gratuito ***Bringer Of Death***.

Los *tilemaps* utilizados para crear el escenario han sido sacados de los asset de pago *Pixel* ***Fantasy Castle Prison*** y ***Pixel Fantasy Snowy Mountains***.

Las fuentes que se han usado en los textos del videojuego son ***DotGothic16*** y ***PermanentMarker*** y estas han sido sacadas de *Google Fonts*, que ofrece fuentes de forma gratuita (Google, 2010).

5.17.3 Audio

Los sonidos empleados en el proyecto han sido sacados de los asset de pago ***Ultimate Game Music Collection***, ***Universal Sound FX*** y ***Monster Sounds & Atmospheres SFX Pack***.

5.18 Mantenimiento y control de versiones

Para el control de versiones del proyecto se ha utilizado GitHub (Microsoft; GitHub Inc, 2007), este te permite guardar todos tus proyectos privados y públicos de manera gratuita en repositorios para que en el caso de que haya un problema con el proyecto no se pierdan los archivos y también se puede volver a una versión anterior en caso de ser necesario.

Se ha seguido una variación simplificada del flujo de trabajo *GitFlow* (Atlassian, 2010) para la gestión de las ramas Git, en este caso, contaríamos con las ramas *Main*, *Develop*, *Feature* y *Hotfix*, pero no se cuenta con la rama *Release*. Este flujo de trabajo es actualmente uno de los

más populares y nos permite aislar el trabajo realizado en cada tipo de rama. A continuación, se explica que función tiene cada rama:

- ***Main***: en esta rama se contiene la versión de producción.
- ***Develop***: esta rama se basa en *Main*, nos sirve como una rama de integración de las funciones realizadas y se vuelve a unir a *Main* una vez tenga las ramas *Feature* necesarias para crear una nueva versión en producción.
- ***Feature***: se crear una rama por tarea y estas ramas se basan en *Develop* que cuando estén terminadas se volverán a unir a *Develop*.
- ***Hotfix***: estas ramas se basan en la rama *Main* donde al completarse la tarea que se haya asociado a la rama se unirá tanto a *Main* como a *Develop*.

Esto nos permite tener un flujo de trabajo más organizado y seguro donde se hacen acciones específicas dependiendo de la rama en la que estemos.

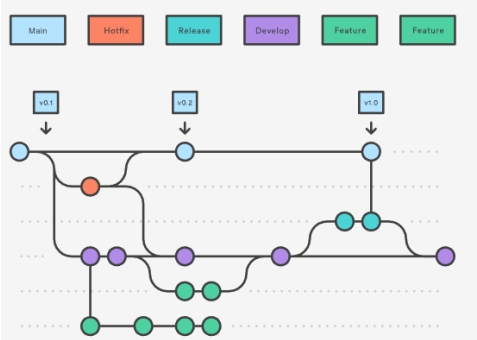

Figura 105: GitFlow

Además, se ha utilizado Trello (Trello Inc, 2011) para la gestión de las tareas del proyecto. Se ha creado un tablero Kanban a través de una de las muchas plantillas que proporciona Trello por defecto. Tiene el siguiente flujo de trabajo:

- **Backlog**: este es el punto de partida donde creamos todas las tareas que vamos a implementar en el proyecto. También se pueden seguir creando tareas a medida que avanza el proyecto.
- **Diseño**: aquí se mueven las tarjetas que se han decidido implementar en la entrega y que tenemos que especificar que se tiene que hacer y cómo.
- **Por hacer**: aquí se mueven las tareas que ya han terminado de diseñarse y están listas para ser tomadas por los miembros del equipo.
- **Haciendo**: aquí se mueven las tareas que han sido asignadas a un miembro del equipo y se están trabajando en el momento.
- **Probar**: aquí se mueven las tareas supuestamente han sido terminadas y están listas para que se prueben y se verifique que están bien hechas.
- **Hecho**: aquí se mueven las tareas que han sido verificadas que han sido terminadas.

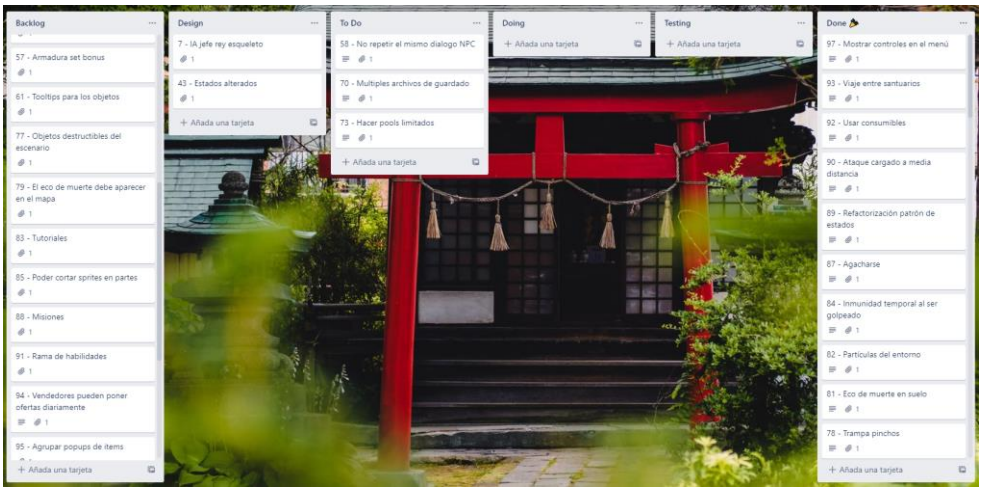

Figura 106: tablero Kanban del proyecto

6. Pruebas

En este apartado se hará una explicación sobre las pruebas realizadas al proyecto.

6.1 Pruebas unitarias

La mejor forma de garantizar que un producto software cumple con lo que se espera de él, es mediante la creación de pruebas unitarias, así nos aseguramos de que, si cambiamos algo en una parte del código, en las otras partes todo seguirá funcionando correctamente.

Se ha utilizado el paquete *Unity Test Framework* donde este usa una integración de *Unity* de la librería NUnit 3.5, que permite la creación de pruebas unitarias en *Unity* tanto en el modo de edición (se ejecutan desde el editor) como en el modo de juego (se ejecutan al entrar al juego).

Estas pruebas unitarias han seguido el patrón AAA (Khorikov, 2020). Este patrón trata de desglosar cada prueba unitaria en 3 secciones:

- ***Arrange***: se inicializan las variables necesarias.
- ***Act***: se ejecutan los métodos que se están probando con las variables que se han inicializado anteriormente.
- ***Assert***: se verifica que el comportamiento del método probado es el correcto.

En el videojuego se han creado pruebas unitarias del modo de edición para probar funciones clave del inventario, como la adición y eliminación de nuevos objetos, de la moneda del videojuego, de las habilidades mágicas o rellenar pociones.

Figura 107: pruebas unitarias del modo editor del inventario

6.2 Pruebas de rendimiento

Un buen rendimiento es crucial para que el usuario tenga una experiencia fluida a lo largo del videojuego, lo que puede ayudar al videojuego a ganar más usuarios y tener una impresión positiva en ellos.

Recientemente, la plataforma de *Steam* realizó una encuesta de hardware donde podemos ver qué componentes son los más usados por la comunidad de jugadores (Valve, 2022), con esta información, se han realizado las pruebas de rendimiento utilizando un ordenador con componentes lo más similares posibles a los que vemos en la encuesta.

Se ha desactivado la sincronización vertical para exprimir la máxima potencia de los componentes y el ordenador tiene las siguientes características:

Resolución	1920x1080
Sistema operativo	Windows 10 Pro de 64 bits
Procesador	AMD Ryzen 7 3700X
Memoria principal	16GB
Gráficos	GeForce GTX 1050Ti 4GB GDDR5

Tabla 70: especificaciones del ordenador usado en la prueba de rendimiento

Estas pruebas tienen como objetivo comprobar que la realización de distintas acciones dentro del juego se realiza dentro de un tiempo razonable. Se ha hecho uso de la herramienta *Profiler*, esta herramienta está integrada dentro de Unity y tiene como uso el análisis detallado del rendimiento de la aplicación en distintas áreas, como, CPU, GPU, memoria, entre otros.

Al ejecutar un análisis con la herramienta *Profiler* se ha de tener en cuenta que *PlayerLoop* es el único proceso que se debe observar, puesto el resto de los procesos pertenecientes al editor y al *Profiler* no estarán presentes, y por ende no afectarán al rendimiento, una vez el videojuego compilado se ejecute.

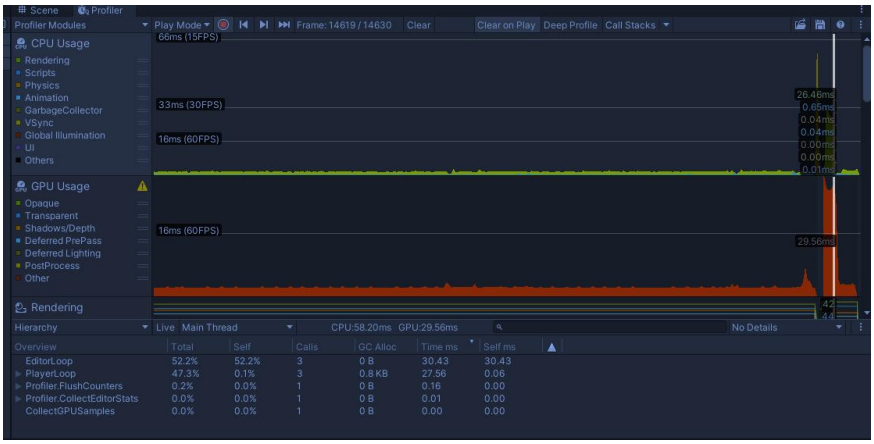

Figura 108: prueba de rendimiento de Nueva partida

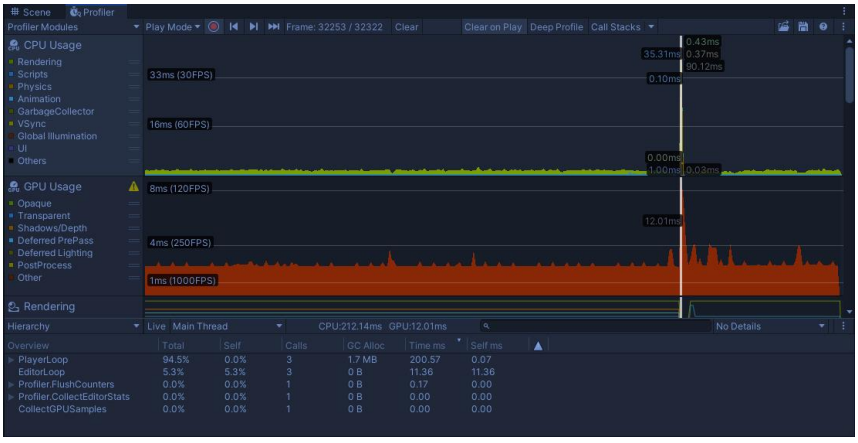

Figura 109: prueba de rendimiento de Cargar partida

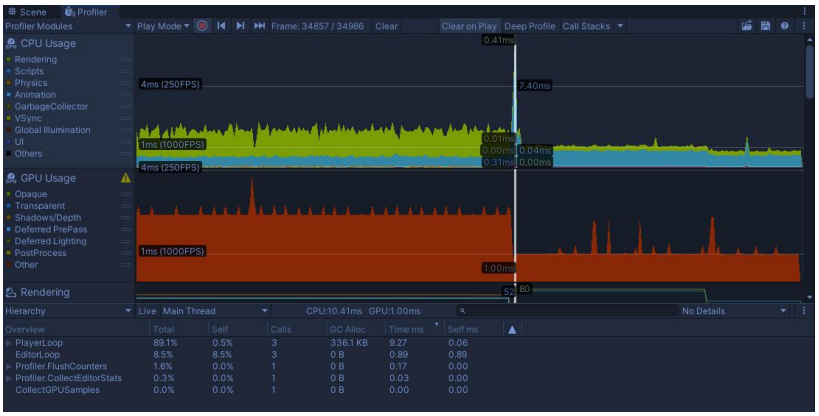

Figura 110: prueba de rendimiento de Abrir inventario

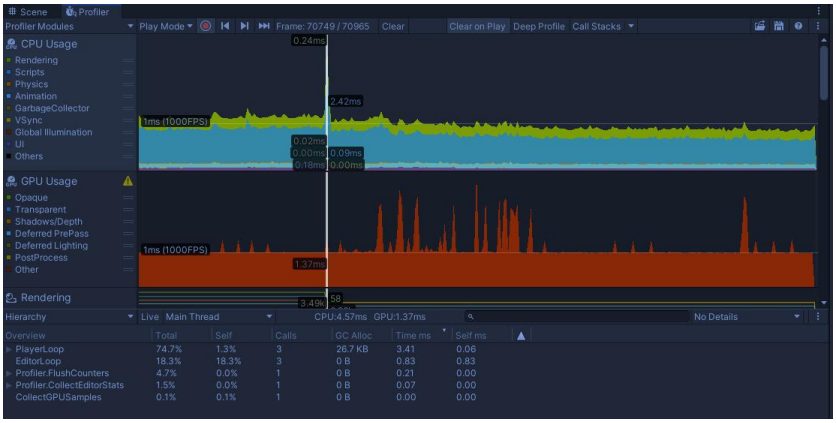

Figura 111: prueba de rendimiento de Equipar objeto

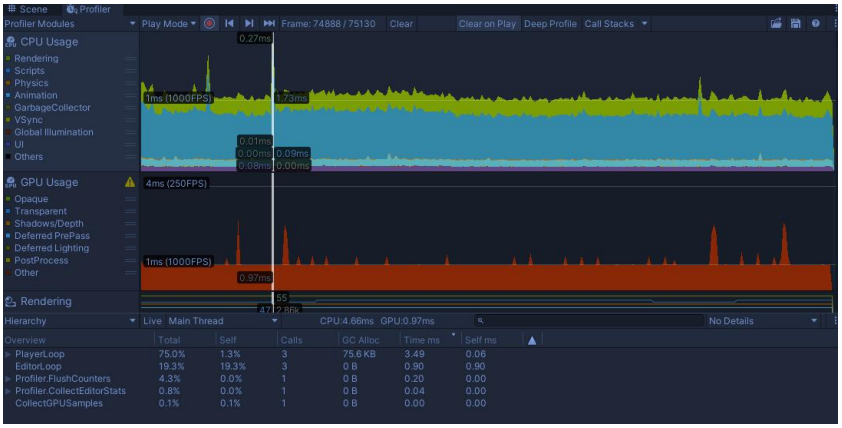

Figura 112: prueba de rendimiento de Crear objeto

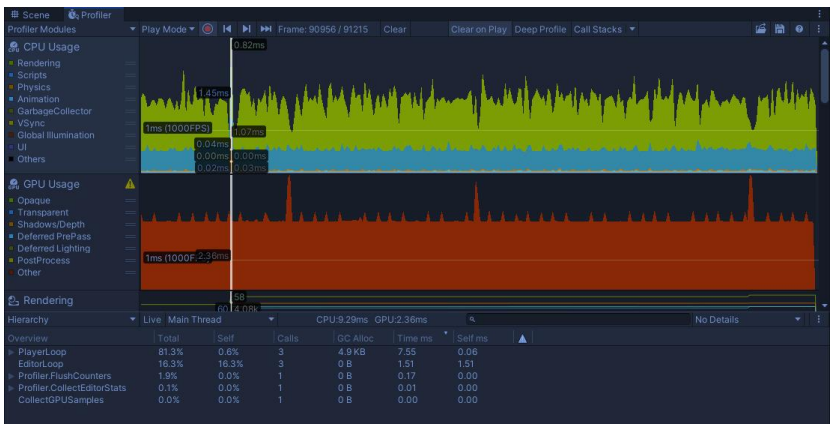

Figura 113: prueba de rendimiento de Cargar escenario

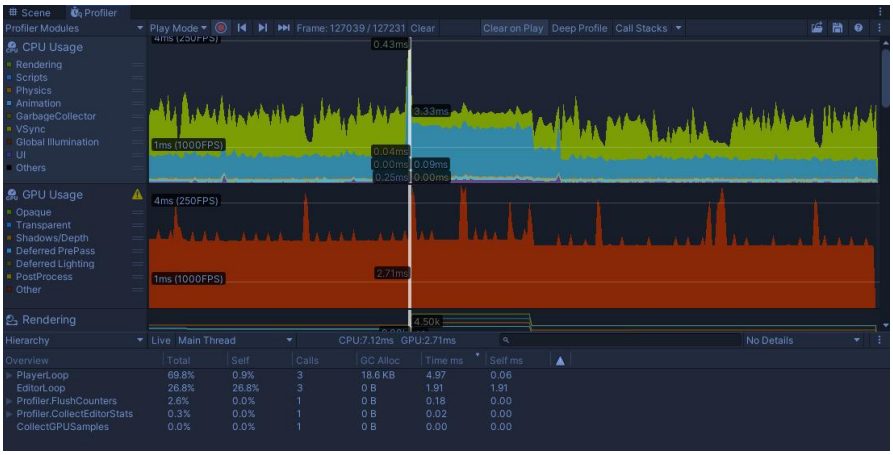

Figura 114: prueba de rendimiento de Guardar partida

Prueba	Resultado aceptable	Resultado obtenido
Nueva partida	< 500 ms	27,56 ms
Cargar partida	< 500 ms	200,57 ms
Abrir inventario	< 100 ms	9,27 ms
Equipar objeto	< 50 ms	3,41 ms
Crear objeto	< 50 ms	3,49 ms
Cargar escenario	< 200 ms	7,55 ms
Guardar partida	< 200 ms	4,99 ms

Tabla 71: resultados de la prueba de rendimiento

Como podemos observar en los resultados de la tabla anterior, el resultado es más que satisfactorio usando un ordenador de gama baja. Cabe destacar la gran diferencia frente a la creación de una nueva partida y la carga de una partida ya creada, esto es debido a la deserialización binaria del archivo de guardado que tiene un gran coste en los tiempos de carga, pero es necesario para evitar que los jugadores hagan trampas modificando su partida, aun así, el tiempo es más que aceptable.

6.3 Validación con usuarios

Una vez se ha probado el videojuego repetidas veces durante todo el desarrollo y se ha arreglado todos los errores que se han encontrado, se ha distribuido la versión beta del producto a ocho personas adeptas al mundo de los videojuegos con una edad comprendida entre los 20 y los 30 años donde solo uno de estos tenía experiencia en el género *metroidvania*.

Principalmente, se ha utilizado esta validación con usuarios para recibir *feedback* sobre el videojuego en general y encontrar errores en este. Entre los errores que se han encontrado, lo más destacable fue un error donde el jugador podía cancelar el diálogo con un *NPC* de forma abrupta al ser atacado y no nos dejaría volver a iniciar una nueva conversación con el *NPC* hasta que se reiniciara el videojuego. Posteriormente, se han corregido los errores encontrados.

Se ha hecho una encuesta a estas personas para saber su opinión sobre el estado actual del videojuego siguiendo la escala Likert de cinco niveles con preguntas cerradas y una última pregunta de respuesta abierta para que los encuestados propongan mejoras e implementaciones que les gustarían ver en el videojuego. La escala Likert es un método de valoración bipolar que mide el grado de conformidad de una pregunta según la puntuación asignada, siendo estas:

1. Totalmente en desacuerdo
2. En desacuerdo
3. Ni de acuerdo ni en desacuerdo
4. De acuerdo
5. Totalmente de acuerdo

A continuación, se hará un análisis del resultado obtenido en cada pregunta de la encuesta.

1. **La interfaz de usuario del videojuego es intuitiva**

Puntuación media: (2*3 + 5*4 + 5) / 8 = 3,875 ≈ 4

Con esta pregunta se ha buscado ver si la interfaz de usuario es lo suficientemente buena sin tener que hacer ningún tipo de explicación de esta. Por lo general, los usuarios han sabido interpretar bien la interfaz de usuario y solo ha habido problemas al intentar avanzar en el sistema de diálogos.

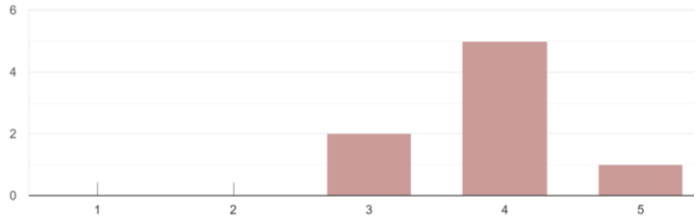

Figura 115: resultado de la primera pregunta de la encuesta

2. **Los controles son difíciles de aprender**

Puntuación media: (4 + 2 + 3*4) / 8 = 2,25 ≈ 2

Un usuario no tendría por qué estudiar un videojuego para poder jugarlo y con esta pregunta se ha buscado ver si es que el videojuego tiene controles demasiado complejos lo cual podría ser un problema si es que se quiere llegar a un público más joven. Por lo general, los usuarios creen que los controle son fáciles de aprender, pero hay una parte de estos que piensan todo lo contrario. Como propuesta de solución, se podrían poner todos los controles en el teclado, ya que hay algunos que están en el ratón y eso puede dificultar el aprendizaje.

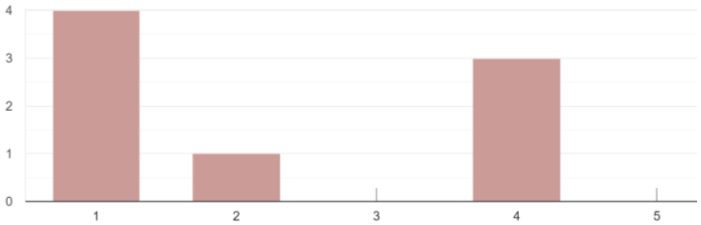

Figura 116: resultado de la segunda pregunta de la encuesta

3. **El videojuego es fácil de usar**

Puntuación media: (2 + 3 + 5*4 + 5) / 8 = 3,75 ≈ 4

Con esta pregunta se ha buscado ver si la cantidad de botones hace difícil al jugador jugar y divertirse dentro del videojuego. Por lo general, los usuarios han podido jugar si problemas y están de acuerdo en que el videojuego es fácil de utilizar pese a la gran cantidad de botones.

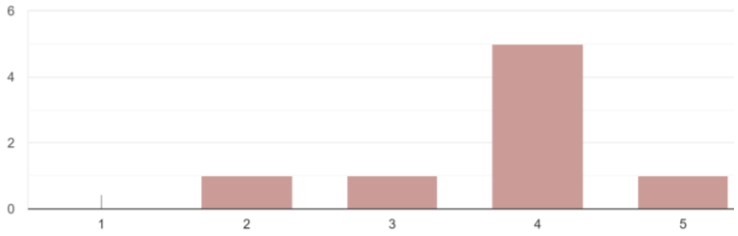

Figura 117: resultado de la tercera pregunta de la encuesta

4. **Las características del videojuego funcionan correctamente**

Puntuación media: (2*3 + 6*4) / 8 = 3,75 ≈ 4

Debido a que se distribuyó una demo técnica con la que se buscaba que se probaran todas las funcionalidades de un metroidvania esta pregunta sirve para saber si ha habido algún problema en alguna. Por lo general, los usuarios están de acuerdo en que las características funcionan correctamente, a excepción de algunos usuarios que tuvieron problemas al cerrar antes de tiempo los diálogos con el *NPC*, pero estos problemas se podían sortear.

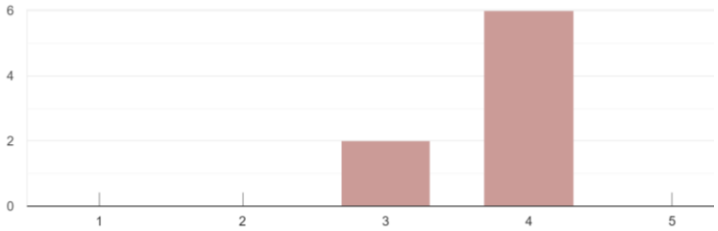

Figura 118: resultado de la cuarta pregunta de la encuesta

5. **Pienso que el videojuego es aburrido**

Puntuación media: (3 + 4*2 + 4) / 8 = 1,875 ≈ 2

Lo más importante para que un videojuego triunfe es que sea divertido, y con esta pregunta se ha buscado ver la opinión de los usuarios a este. Por lo general, a los usuarios les ha parecido un videojuego bien hecho y divertido, pero esperaban una demo más larga.

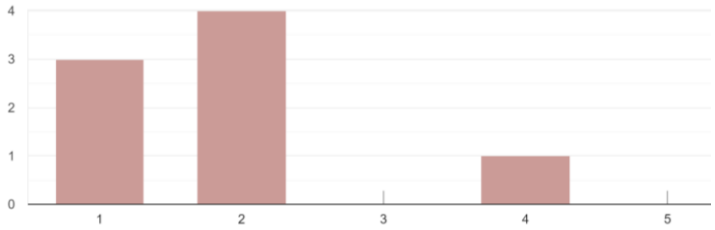

Figura 119: resultado de la quinta pregunta de la encuesta

6. **Pienso que el videojuego tiene una dificultad alta**

Puntuación media: (2*3 + 4 + 5*5) / 8 = 4,375 ≈ 4

El objetivo de esta pregunta ver que el videojuego este equilibrado correctamente, puesto que se busca que el videojuego sea difícil pero justo. Por lo general, los usuarios están de acuerdo en que el videojuego es difícil, pero algunos han manifestado sus quejas en el patrón de ataques de los enemigos, que pueden ser erráticos y excesivamente poderosos.

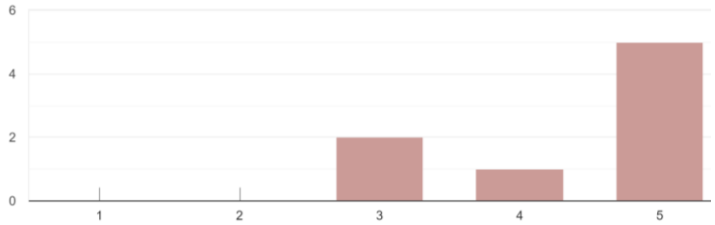

Figura 120: resultado de la sexta pregunta de la encuesta

7. **Pienso que la penalización al morir me ayuda a mejorar en el videojuego**

Puntuación media: (3*3 + 4*4 + 5) / 8 = 3,75 ≈ 4

Las penalizaciones en los videojuegos sirven para que la próxima vez que iniciemos un combate pensemos mejor en la estrategia que vamos a tomar para derrotar al enemigo. Por lo general, los usuarios están de acuerdo que mejoran al perder la moneda del videojuego al ser derrotados, pero teniendo una oportunidad de recuperarlas si conseguimos volver al lugar donde hemos sido derrotados la última vez.

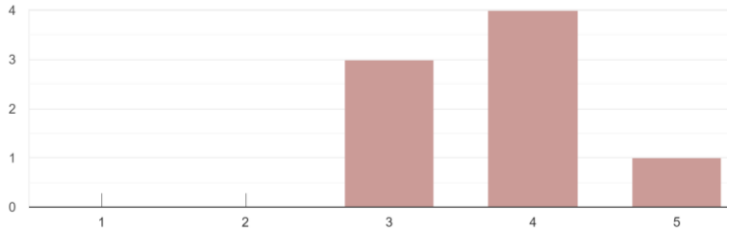

Figura 121: resultado de la séptima pregunta de la encuesta

8. **El videojuego no proporciona un *feedback* correcto**

Puntuación media: (6 + 2 + 4) / 8 = 1,5 ≈ 2

Esta pregunta tenía como objetivo ver si se necesita más *feedback* dentro del videojuego y por lo general, los usuarios están en desacuerdo de que se necesite más *feedback* excepto por una persona que opina que los objetos que suelta el enemigo tendrían que indicar mejor que pueden recogerse por lo que habría que añadir algún tipo de *feedback* visual para que el objeto destacara.

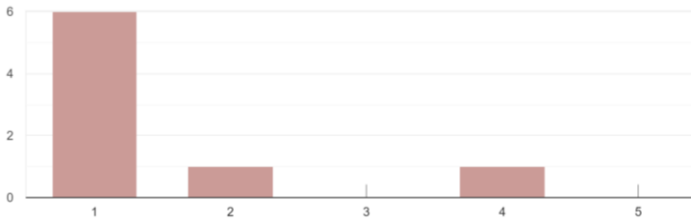

Figura 122: resultado de la octava pregunta de la encuesta

9. **No ha sucedido ningún error durante la sesión de juego**

Puntuación media: (2*3 + 3*4 + 3*5) / 8 = 4,125 ≈ 4

Esta pregunta tenía como objetivo ver lo estable que es la aplicación hasta ahora e incentivar a los encuestados a comentar que errores han encontrado para poner corregirlos. Por lo general, los usuarios están de acuerdo en que han sucedido pocos errores.

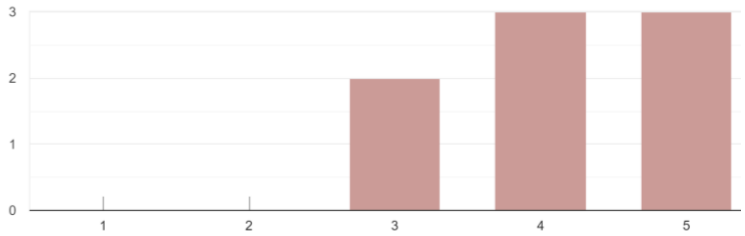

Figura 123: resultado de la novena pregunta de la encuesta

10. Tendría intención de comprar el juego una vez esté terminado

Puntuación media: (3 + 3*4 + 4*5) / 8 = 4,375 ≈ 4

Con esta pregunta se ha buscado ver si merece la pena seguir trabajando en el proyecto a futuro y por lo general, los usuarios que han podido probar la aplicación están de acuerdo en que comprarían el videojuego por lo que es un gran aliciente para que se siga trabajando en la aplicación para poder sacarla al mercado en algún momento futuro.

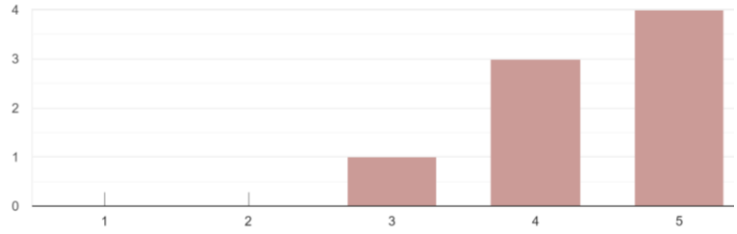

Figura 124: resultado de la décima pregunta de la encuesta

11. ¿Mejorarías y/o añadirías algo en el videojuego?

De entre todas las propuestas de los encuestados, han destacado tres:

En primer lugar, los usuarios han pedido cambiar la música de fondo cuando se juega, ya que esta se torna repetitiva rápidamente.

En segundo lugar, la velocidad de salto y caída del jugador ha resultado ser demasiado lenta, por lo que a los usuarios se les hacía difícil esquivar los ataques enemigos y acababan siendo derrotados rápidamente.

Finalmente, la dificultad general del juego ha resultado ser excesiva, ya que los enemigos pueden derrotar al jugador en dos o tres ataques y atacan muy rápido, por lo que en futuras versiones de la aplicación se buscara equilibrar estas partes del juego para que el usuario tenga una experiencia más justa.

7. Conclusiones

Para concluir la memoria puedo decir que el desarrollo de este proyecto me ha permitido poner en práctica los conocimientos aprendidos tanto en la carrera como de forma independiente sobre el proceso *software*, lo cual también me ha permitido aprender y mejorar en el desarrollo de videojuegos y darme cuenta no solo de la diversión que proporciona desarrollar un videojuego sino de la dificultad de algunas mecánicas que vemos implementadas en muchos de los videojuegos actuales.

Estoy muy satisfecho con lo que se ha conseguido en el proyecto, ya que el género *metroidvania*, que es uno de mis favoritos, tiene mecánicas complejas y no hay mucha literatura sobre ellas y por ello me ha encantado conseguir sortear los problemas e implementarlas.

Además, el proyecto me ha permitido aprender sobre nuevas disciplinas, como la elaboración del arte o la creación de *shaders*, y sus correspondientes herramientas.

Por lo que respecta al futuro del proyecto, le veo potencial y me encantaría poder seguir trabajando en él y llevarlo a la venta en tiendas digitales una vez este completo.

Para finalizar, he de decir que el videojuego tiene una funcionalidad completa y se han conseguido los objetivos propuestos al principio del documento, pero estos pueden extenderse mucho más como veremos en el próximo apartado.

8. Trabajos futuros

A continuación, se introducirán algunas de las ideas de características más importantes que se plantean implantar en el proyecto en un futuro.

8.1 Historia

Actualmente, *The Last Hanyou* es más una demo técnica, en la que se puede usar todas las características importantes del videojuego en un escenario reducido, que un videojuego completo con una historia por desentrañar.

Por ello, el objetivo más prioritario a futuro es construir un mundo a partir de la historia, en la que se incluirá un jefe[18] por cada área, nuevos enemigos, nuevos *NPC*, nuevos objetos, nuevas habilidades y nuevos diálogos que enriquezca la historia que se quiere contar.

8.1.1 Misiones

La inclusión de misiones secundarias ayudará a la creación de subtramas dentro de la historia en la que el jugador podrá recibir recompensas, ya sean en experiencia, almas u objetos. Además, el jugador tendrá tanto la misión principal como las secundarias marcadas con un icono en el mapa, lo cual, al tener escenarios tan grandes interconectados, le servirán como guía para que no pueda perderse.

8.2 Combate

El combate es una de las mecánicas más importantes para que el juego sea disfrutable. Por ello, para que el jugador no sienta que este es repetitivo y tedioso, se han planeado una serie de nuevas características que harán más interesante y dinámico el combate.

8.2.1 Modo yokai

Como el personaje principal es un *hanyou*[19], se ha planeado que puedas desatar su parte *yokai*[20] en la cual se tendrá más velocidad de ataque, fuerza, inteligencia y destreza.

Se ha planeado un nuevo medidor para este modo, el cual aumentará al golpear enemigos o ser golpeado e ira disminuyendo con el paso del tiempo mientras el modo *yokai* este activo.

8.2.2 Configuraciones de personaje

Mi principal objetivo en torno al combate es que el jugador tenga la suficiente flexibilidad para poder jugar de la manera que le parezca más divertida sin tener que estar atado a un estilo de pelea predefinido y ahí es donde entra en juego la creación de configuraciones de personaje, donde a través de conjunto de objetos, habilidades y estadísticas podemos cambiar la forma con la que jugador puede controlar al personaje del videojuego.

[18] Enemigos muy poderosos que se batallan en escenarios acotados con ataques telegrafiados.
[19] Mitad humano, mitad demonio en la mitología japonesa.
[20] Demonios en la mitología japonesa.

8.2.3 Estados alterados

Los estados alterados como quemado, congelado o envenenado proporcionarán dinamismo al entorno, enemigos y a las configuraciones de personaje que pueda utilizar el jugador.

8.2.4 Árbol de habilidades

Añadir un árbol de habilidades donde se desbloqueen pasivas, ataques mágicos y físicos especiales mejorará la progresión del jugador y aumentará considerablemente el número de configuraciones de personaje que se pueden ser creadas.

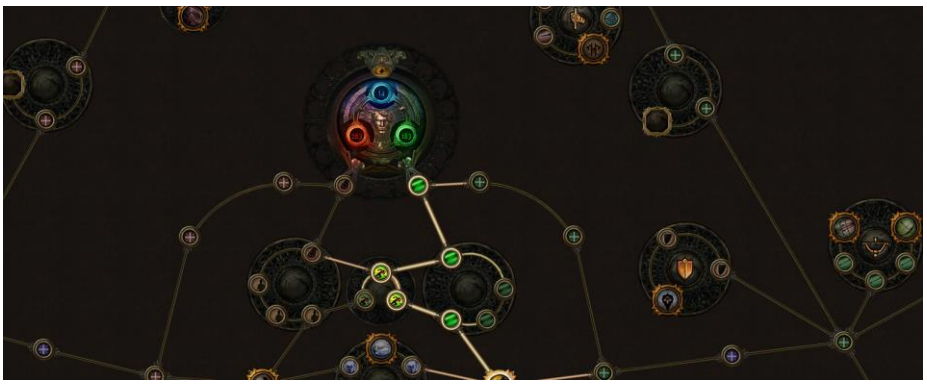

Figura 125: árbol de habilidades de «Path of Exile»

8.2.5 Conjunto de objetos

Al tener equipados todas las piezas de un conjunto de objetos se conseguirán una serie de bonificaciones que favorecerán diferentes estilos de pelea, contribuyendo así a las configuraciones de personaje.

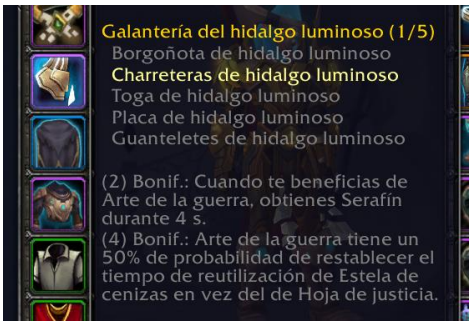

Figura 126: conjunto de objetos en «World of Warcraft»

8.3 Movimiento

Un movimiento fluido es muy importante cuando tenemos un combate dinámico, pero además si tenemos plataformas este tiene que beneficiar ligeramente al jugador para que no sea muy preciso y de una sensación amarga cuando fallamos un salto por estar un píxel más lejos de lo que deberíamos de la plataforma.

Por ello, a lo largo de los años se han creado en los videojuegos distintas características que hacen del movimiento, principalmente en los juegos de plataformas, más disfrutable para el usuario y son estas características las que se quieren implementar en un futuro.

8.3.1 Coyote time

Con *coyote time*, justo antes de caer sobre un precipicio se tendrá unos pocos *frames* en los que se puede realizar un salto antes de caer al vacío, lo que hace que los saltos no tengan que ser tan precisos.

8.3.2 Jump buffer

Con *jump buffer*, cuando falten unos pocos *frames* antes de tocar el suelo ya podrás efectuar otro salto, mejorando así la sensación que tendrá el jugador al correr y saltar por el escenario.

8.3.3 Salto regulable

El salto aumenta o disminuye en altura en función del tiempo que se presione el botón asignado al salto, pudiendo así llegar a plataformas más elevadas o alejadas.

8.4 Localización de textos

Actualmente, el juego solo está disponible en inglés y la localización de los textos ayudará a que el juego sea más accesible para todo el mundo, pudiendo disfrutarse en el idioma preferido del usuario.

9. Bibliografía

- Atlassian. (2010). *Atlassian*. Recuperado el 22 de Junio de 2022, de Atlassian: https://www.atlassian.com/es/git/tutorials/comparing-workflows/gitflow-workflow
- Balsamiq Studios. (3 de Mayo de 2022). *Balsamiq*, 4.5.2. Recuperado el 12 de Mayo de 2022, de Balsamiq: https://balsamiq.com
- CD Projekt. (2008). *GOG*. Recuperado el 13 de Junio de 2022, de GOG: https://www.gog.com/
- Collado, J. C. (Enero de 2018). *Asociación española de videojuegos.* Obtenido de http://www.aevi.org.es/web/wp-content/uploads/2018/01/1801_AEVI_EstudioEconomico.pdf
- Corcoran, L. (18 de Enero de 2013). *Itch.io*. Recuperado el 13 de Junio de 2022, de Itch.io: https://itch.io/
- Epic Games. (29 de Agosto de 2021). *Unreal Engine*, 4.27. Recuperado el 20 de Noviembre de 2021, de Unreal Engine: https://www.unrealengine.com
- Google. (2010). *Google Fonts*. Recuperado el 25 de Junio de 2022, de Google Fonts: https://fonts.google.com/
- Guzsvinecz, T. (2022). The correlation between positive reviews, playtime, design and game mechanics in souls-like role-playing video games. doi:https://doi.org/10.1007/s11042-022-12308-1
- Igarashi, K. (2015). *Kickstarter*. Recuperado el 13 de Junio de 2022, de Kickstarter: https://www.kickstarter.com/projects/iga/bloodstained-ritual-of-the-night/description
- JGraph Ltd. (29 de Noviembre de 2021). *Diagrams.net*. Recuperado el 1 de Diciembre de 2021, de Diagrams.net: https://app.diagrams.net/
- Khorikov, V. (2020). *Unit Testing Principles, Practices, and Patterns* (Primera ed.).
- Linietsky, J., & Manzur, A. (23 de Marzo de 2022). *Godot*, 3.4.4. Recuperado el 12 de Mayo de 2022, de Godot: https://godotengine.org/
- Microsoft; GitHub Inc. (19 de Octubre de 2007). *GitHub*. Recuperado el 20 de Noviembre de 2021, de GitHub: https://github.com/
- Morales, R. (2019). *Patrones de diseño y principios SOLID en C# .NET*. Obtenido de https://www.udemy.com/course/patrones-de-diseno-principios-solid/
- Nystrom, R. (2014). *Game Programming Patterns* (Primera ed.). Genever Benning.
- Pattuzi, S., & Davidson, R. (2020). *Unity Dialogue & Quests: Intermediate C# Game Coding*. Obtenido de https://www.udemy.com/course/unity-dialogue-quests/
- Pinheiro Oliveira, B., de Oliveira da Rocha Franco, A., Wellington Franco da Silva, J., de Carvalho Gomes , F. A., & Rodrigues Maia, J. G. (2020). A Framework for Metroidvania Games. Fortaleza, Brazil: Proceedings of SBGames 2020.
- Rival, F. (28 de Julio de 2020). *Gdevelop*, 5.0.0-beta102. Recuperado el 20 de Mayo de 2022, de Gdevelop: https://gdevelop.io
- Sauco, A., & Lozano, E. (2021). *El Viaje del Jugador: Guía de Diseño de Videojuegos* (Primera ed.).
- Skolnick, E. (2014). *Video Game Storytelling: What Every Developer Needs to Know about Narrative Techniques* (Primera ed.). Watson-Guptill.
- Taylor, C. (5 de Julio de 2015). *Gamedocs.* Recuperado el 10 de Noviembre de 2021, de Gamedocs: https://gamedocs.org/game-design-template-from-chris-taylor/
- Team Cherry. (2017). *Hollow Knight*. Recuperado el 13 de Junio de 2022, de Hollow Knight: https://www.hollowknight.com/

- The Game Kitchen. (2017). *Kickstarter*. Recuperado el 13 de Junio de 2022, de Kickstarter: https://www.kickstarter.com/projects/828401966/blasphemous-dark-and-brutal-2d-non-linear-platform/description
- Trello Inc. (13 de Septiembre de 2011). *Trello*. Recuperado el 20 de Noviembre de 2021, de Trello: https://trello.com
- Unity Technologies. (2 de Abril de 2021). *Bolt*, 1.4.15. Recuperado el 20 de Noviembre de 2021, de Bolt: https://docs.unity3d.com/bolt/1.4/manual/index.html
- Unity Technologies. (7 de Abril de 2021). *Unity*, 2020.3.3f1. Recuperado el 20 de Noviembre de 2021, de Unity: https://unity.com
- Valve. (27 de Mayo de 2022). *Steam*. Recuperado el 13 de Junio de 2022, de Steam: https://store.steampowered.com/
- Valve. (Mayo de 2022). *Steam*. Recuperado el 22 de Junio de 2022, de Steam: https://store.steampowered.com/hwsurvey/Steam-Hardware-Software-Survey-Welcome-to-Steam
- YoYo Games. (25 de Mayo de 2022). *GameMaker*, 2022.5. Recuperado el 12 de Junio de 2022, de GameMaker: https://gamemaker.io/
- Zimmerman, C. (2015). *Destructoid*. Recuperado el 13 de Junio de 2022, de Destructoid: https://www.destructoid.com/reviews/review-axiom-verge/

10. Anexos

1. Anexo I: Documento de diseño del videojuego

El *Game Design Document* (*GDD*) o Documento de diseño del videojuego es un documento altamente detallado que está en constante actualización en el cual se describen todas las decisiones relacionadas con el diseño del videojuego.

Para la elaboración de este documento se ha utilizado una variación de la plantilla de Chris Taylor[21] (Taylor, 2015) combinándolo con otros *GDD* públicos de videojuegos de la misma fuente.

2. Anexo II: Objetivos de desarrollo sostenible

Los objetivos de desarrollo sostenible (ODS) son aquellos que los líderes mundiales adoptaron para poder erradicar la pobreza, proteger el planeta y asegurar la prosperidad para todos. Estos objetivos deben cumplirse en un periodo de 15 años.

[21] Famoso diseñador de videojuegos como *Total Annihilation* y *Dungeon Siege*.

ANEXO I: Documento de diseño de videojuegos

Figura 127: icono del videojuego The last hanyou

The last hanyou

Libera tus poderes demoníacos y sobrevive en un mundo devastado por los *yokai*

Escrito por Erik Font Gil

Versión 1.00

22 de noviembre de 2021

1. Introducción

1.1 Ficha resumen

Nombre: The last hanyou.

Plataforma: Windows 7/8/10.

Idiomas: inglés.

Modos de juego: un jugador.

Género: Metroidvania, 2D, RPG y *souls-like*.

Premisa del juego: La historia del videojuego trata sobre una extraña bruma carmesí que se está extendido por todo japón, cada vez se reportan más casos de avistamientos de criaturas sobrenaturales, crímenes y desapariciones de personas. Han contratado a tu padre, un *rōnin*, para que investigue y acabe con la maldición que asola a la región a cambio de convertirle en un samurái, al no regresar, nuestro protagonista parte en su búsqueda.

2. Sistema operativo objetivo

El sistema operativo objetivo es Windows 7/8/10 de 64 bits.

2.1 Requisitos del sistema

2.1.1 Mínimos

Sistema operativo	Windows 7 64 bits
Procesador	AMD Phenom II x2 550
Memoria	256 MB
Gráficos	GeForce GT 8800 512 MB
Almacenamiento	256 MB de espacio libre

Tabla 72: requisitos mínimos del sistema

2.1.2 Recomendados

Sistema operativo	Windows 10 64 bits
Procesador	AMD Phenom II x4 945
Memoria	256 MB
Gráficos	GeForce GT 710 1GB
Almacenamiento	256 MB de espacio libre

Tabla 73: requisitos recomendados del sistema

3. Jugabilidad

3.1 Historia

3.1.1 Acto 1 - Búsqueda

Eres hijo de Date Masamune, apodado como “el dragón de un ojo”.
Tras la caída de su antiguo señor, se convirtió en un Ronin, un día, un mensajero de un famoso clan, le ofreció la oportunidad de restaurar su honor y convertirse en el samurái de una familia, a cambio, debería de investigar la extraña bruma roja que vuelve loca a las personas por todo japón y acabar con el causante, meses después de su partida, tú, Hanzo Masamune, después de que tu padre no volviera de la misión, sales en su busca junto a tu espada la “devora almas” un regalo de tu padre tras su marcha.

3.1.2 Acto 2 - Enfrentamiento

A lo largo de tu viaje descubrirás tu verdadera identidad, que eres un *hanyou*, un humano mitad demonio, sabrás el origen de tus poderes y la carga que deberás soportar para lograr tu objetivo, salvar japón.
Tu padre ha sido imbuido por la niebla roja, la locura recorre su mente y ataca indiscriminadamente, por desgracia, acabáis enfrentándoos en un amargo desenlace.

3.1.3 Acto 3 - Venganza

Se acerca la batalla final, la niebla roja inunda todo japón, Akuma, el gran demonio nacido de las emociones negativas humanas tras la última guerra que asoló Japón, es el causante de todo el dolor y desesperación causado, te enfrentas a él, tras una larga batalla muestra su verdadera forma y el enfrentamiento es aún más difícil, pero acabas venciendo gracias a las bendiciones de los dioses que te acompañaron a lo largo del viaje, tras el golpe final, Akuma fue sellado en la catana “devora almas” disipando la niebla roja de Japón y consiguiendo el mayor título de los samuráis.

3.2 Personajes

Hanzo Masamune: hijo de Date Masamune, al nacer, era un humano normal, pero años más tarde, al perder a su madre tras el ataque que sufrieron a manos de los *yokai*, su poder latente despertó, se convirtió en un *hanyou*, un humano que era mitad *yokai*, tras su despertar, logro repeler algunos *yokai* menores, pero se vio rápidamente superado por la horda que asaltaba el castillo, con la ayuda de su padre, lograron escapar.

Se refugiaron en un lugar seguro y fue entrenado duramente, se le enseñó a usar la espada y el camino del guerrero, con el objetivo de acabar con los *yokai*.

Date Masamune: era uno de los más grandes samuráis conocido como “El dragón de un solo ojo” debido a su gran destreza y su ceguera de un ojo.

Pero incluso el más grande de los samuráis sucumbió ante una horda de *yokai*, intentó proteger, el castillo tanto como pudo, pero al final, tuvo que decidir, si salvar a su hijo o su honor, como padre, escogió a su hijo y ambos escaparon de las ardientes llamas que asolaban el castillo.

Tiempo más tarde, le ofrecieron un trabajo para restaurar su honor, partiendo solo y dejando atrás a su hijo

Akuma: dios nacido de las emociones negativas de los humanos: rabia, frustración, tristeza, celos, dolor, miedo…

Considerado como el dios de la maldad, le cubre una masa oscura y le envuelve una niebla roja, la única parte visible es su cabeza, una calavera fangosa de ojos rojos brillantes, repudiado por los demás dioses, fue exiliado del palacio celestial, como venganza, se alzó contra ellos y trajo al mundo la oscuridad y la niebla roja que asola todo japón, quitándole la cordura a las personas que respiran su niebla.

Katsu, el mercader: Katsu, en su vida pasada, era un mercader avaricioso que hacia lo posible por embolsarse unas monedas de oro y tras aparecer la niebla roja, sufrió un ataque de unos *yokai*. Al borde de la muerte, suplicó a los 7 dioses de la fortuna por una última oportunidad, a cambio de salvarle la vida, les entregaría todo el dinero que recolectó a lo largo de su vida y en lugar de dinero, comerciaría con almas, pues los 7 dioses preveían que necesitarían de sus servicios en un futuro.

3.3 Mecánicas

3.3.1 Exploración

Otear: El jugador podrá otear el escenario si este está en el estado reposo.

3.3.2 Movimiento

Doble salto: el jugador podrá volver a saltar en el aire.

Salto en pared: cuando se tope con una pared, el jugador podrá saltar en dirección opuesta a esta.

Salto: el jugador podrá saltar si está en el suelo.

Salto hacia abajo: si el jugador se encuentra sobre unas plataformas especiales, este podrá saltar hacia abajo.

Esquivar: el jugador podrá esquivar lateralmente proporcionándole *i-frames*.

Agarrarse a un saliente: cuando el jugador esté cerca de un saliente, se agarrará a este automáticamente.

3.3.3 Combate

Todos los ataques aquí comentados tienen una probabilidad en base a su destreza de hacer daño crítico.

Ataque lateral: ataque normal del jugador que dañara a los enemigos.

Ataque hacia arriba: es un ataque normal hacia arriba.

Ataque hacia abajo: este ataque se lanzará hacia abajo con su espada dañando todo lo que se tope con el hasta llegar a un punto sólido donde saldrá una explosión que hará daño de áreas.

Ataque cargado: el jugador cargará su ataque donde al liberarlo proporcionará un ataque fuerte y una onda de energía que desaparecerá a media distancia.

Contraataque: el jugador podrá ponerse en posición de contraataque donde si el enemigo ataca este ignorará el daño y soltará un ataque fuerte.

Beber poción: el jugador podrá beber una poción que recuperara salud y se regeneraran al interactuar con santuarios.

3.3.4 Inventario

El jugador podrá conseguir objetos de los enemigos donde serán almacenados y podrán ser equipados.

3.3.5 Habilidades

Al derrotar enemigos tenemos una probabilidad de absorber sus poderes.

3.3.6 Fabricación de objetos

Al derrotar enemigos, estos podrán soltar recetas donde podremos fabricar el objeto de la receta si tenemos los objetos necesarios, normalmente materiales.

3.3.7 Mapa

El jugador desbloqueará celdas del mapa por donde pase y se podrán ver desde el menú del videojuego.

3.3.8 Santuarios

Aumentar estadísticas: Si el jugador ha subido de nivel, este podrá aumentar una estadística a su elección.

Viaje rápido: El jugador podrá hacer un viaje rápido entre santuarios.

3.4 Tipo de objetos

- **Armaduras**: proporcionarán un aumento de estadísticas al ser equipadas.
- **Consumibles**: se podrán consumir para recuperar algún recurso como la salud o el maná.
- **Materiales**: se utilizan como materiales de fabricación.
- **Recetas**: se usan para fabricar objetos mediante el consumo de materiales.

3.5 Enemigos

El videojuego contiene actualmente dos enemigos.

Estos enemigos tienen una probabilidad de ser enemigos de élite al generarse, donde cambiarán su aspecto, serán más poderosos y soltarán recompensas mejores y adicionales.

3.5.1 No-muerto

El enemigo no-muerto ataca a corta distancia con ataques físicos que infligirán un gran daño y cuando el jugador esté a media distancia o a corta distancia, pero no pueda realizar ataque físico atacará con ataques mágicos y cuando este tenga poca salud, entra en estado de frenesí aumentando su fuerza y velocidad de ataque.

Figura 128: enemigo no muerto

3.5.2 Farolillo maldito

El enemigo farolillo maldito ataca a larga distancia con ataques mágicos y cada cierto tiempo se protegerá con una barrera de fuego que hará daño al contactar con ella, además, este también explotará pasado un tiempo tras morir.

Figura 129: enemigo farolillo maldito

3.5.3 Controles

3.5.4 Teclado

El mapeo de las teclas del teclado serán las siguientes:

Tecla	Acción
Esc	El jugador abrirá o cerrará el menú del juego.
Q	El jugador se beberá una poción y se restaurará vida.
E	El jugador interactuar con el objeto de que tenga delante.
A	El jugador se moverá hacia la izquierda.
S	El jugador se agachará.
S + Space	El jugador saltará hacia debajo de la plataforma.
S + ratón botón izquierdo	El jugador hará un ataque hacia abajo.
W + ratón botón izquierdo	El jugador hará un ataque hacia arriba.
D	El jugador se moverá hacia la derecha.
F	El jugador usará la habilidad que tenga equipada.
Left Shift	El jugador hará un esquive.
Z	El jugador utilizará el consumible que tenga equipado y seleccionado.
X	El jugador cambiará de consumible entre los

	que tiene equipados.
Left Ctrl	El jugador oteará el escenario.
Space	El jugador saltará.
Space en el aire	El jugador hará un doble salto.

Tabla 74: controles teclado

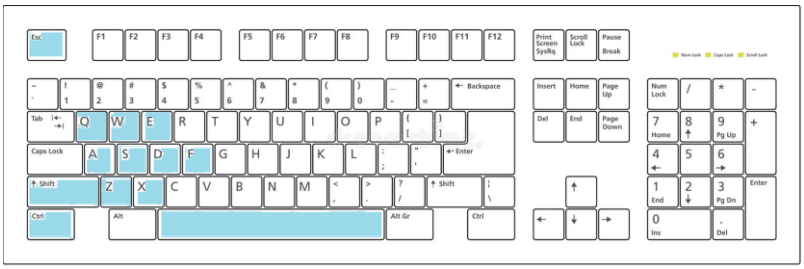

Figura 130: papeo de los controles del teclado

3.5.5 Ratón

El mapeo de los botones del ratón serán los siguientes:

Botón	**Acción**
Izquierdo	El jugador hará un ataque.
Mantener izquierdo	El jugador soltará un ataque potente a corta distancia que lanzará una onda a media distancia que destruirá proyectiles enemigos.
Derecho	El jugador hará se pondrá en posición de contraataque.

Tabla 75: controles ratón

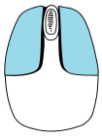

Figura 131: mapeo de los controles del ratón

4. Interfaz de usuario

En esta sección se mostrarán todas las posibles interfaces de usuario que haya en el juego.

4.1 Menú

Menú principal: Es la primera ventana que vemos al iniciar el videojuego, en esta podemos crear una nueva partida, continuar si es que tenemos un archivo de guardado, tenemos la opción de como jugar, las opciones del videojuego y salir del videojuego.

Figura 132: interfaz de usuario Menú principal

Menú principal – Opciones: Ventana de opciones para cambiar la configuración del videojuego desde el menú principal. Podremos cambiar el volumen de la música, los efectos de sonido, cambiar la resolución del videojuego, la calidad, activar o desactivar la pantalla completa, la sincronización vertical y el temblor de la cámara al golpear o ser dañado.

Figura 133: interfaz de usuario opciones menú principal

Menú principal – Controles: Ventana que nos muestra el mapeo de teclas y botones a pulsar en el teclado y ratón para poder movernos y luchar.

Figura 134: interfaz de usuario controles

4.2 Jugabilidad

HUD[22] principal: Se nos muestra el HUD principal con la información más importante sobre el estado del jugador: salud, maná, nivel, experiencia, habilidad equipada, consumible seleccionado, pociones restantes y almas.

El consumible solo se mostrará si el jugador tiene alguno equipado. Al equipar un consumible, este se mostrará arriba de su barra de salud mostrando su icono y su cantidad. Al consumir todos los consumibles, este se ocultará o cambiará al siguiente consumible equipado si lo hubiera.

La ventana que muestra la zona en la que estamos no estará siempre activa, esta únicamente se activará por un segundo al llegar a una nueva zona, mostrando su nombre en la parte superior del HUD principal.

Al recoger objetos, estos se mostrarán por un segundo en la esquina derecha del jugador mostrando su icono, nombre y cantidad obtenida.

Figura 135: HUD principal

HUD enemigo: El enemigo mostrará su salud y nivel con una barra encima de él.

[22] Interfaz de usuario relacionada con la jugabilidad.

Figura 136: HUD enemigo

Inventario: En el inventario se mostrará la información relacionada con las estadísticas del jugador, sus objetos tanto equipados como los no equipados y al pulsar en un objeto estos mostraran toda su información y, si es posible equiparlo, este se equipará. La barra superior filtra los objetos por su tipo. Estos filtros son: todos los objetos, armaduras, consumibles y materiales, en ese orden.

Figura 137: inventario

Libro de habilidades: En esta ventana se almacenan todas las habilidades que se han conseguido al derrotar enemigos, al pulsar en estas se mostrará la información relacionada con estas, como la descripción y su nombre y se equipará.

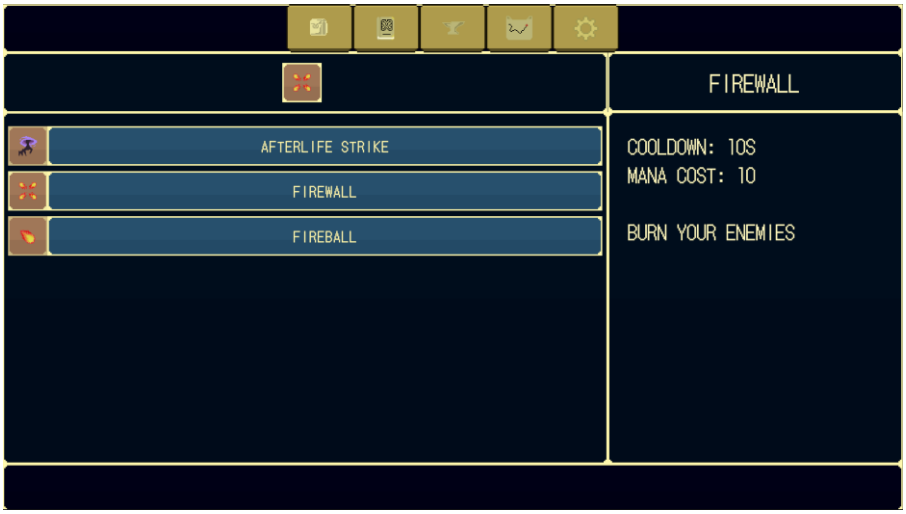

Figura 138: libro de habilidades

Fabricación de objetos: En esta ventana se muestran todas las recetas que hayamos conseguido y al pulsar en estas se mostrará su título, descripción, cantidad que se fabricará, y si tenemos en posesión los objetos adecuados para su fabricación. Se pueden filtrar las recetas por el tipo de

objeto que fabricará. Estos filtros son: todos los objetos, armaduras, consumibles y materiales, en ese orden.

Figura 139: recetas

Mapa: En esta ventana se muestran las secciones del mapa que han sido descubiertas por el jugador.

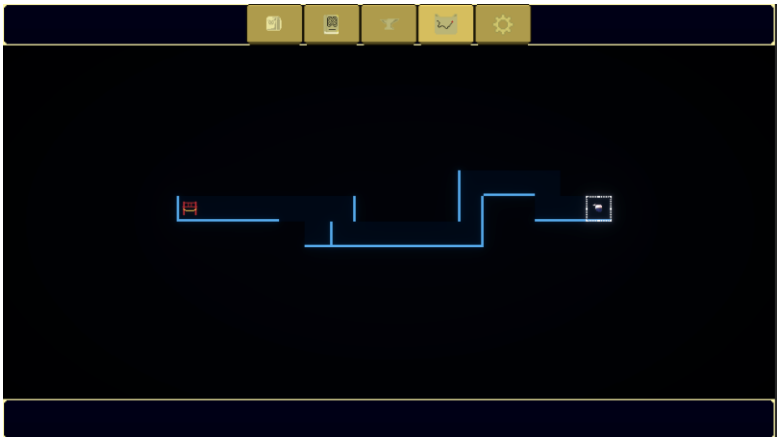

Figura 140: mapa

Menú del juego: En esta ventana se mostrarán las opciones del videojuego, si quiere volver al menú principal o si quiere salir del videojuego.

Figura 141: menú del juego

Menú del juego – Opciones: Ventana de opciones para cambiar la configuración del videojuego desde el menú del videojuego. Podremos cambiar el volumen de la música, los efectos de sonido, cambiar la resolución del videojuego, la calidad, activar o desactivar la pantalla completa, la sincronización vertical y el temblor de la cámara al golpear o ser dañado.

Figura 142: opciones del videojuego

Diálogo: Al interactuar con un NPC se abrirá un diálogo donde se mostrarán o no opciones si es que se desea continuar la conversación.

Figura 143: diálogo

Mercader - Selección: Si el NPC es un mercader, tendremos la opción de poder comprarle objetos y vender objetos de nuestro inventario.

Figura 144: opciones mercader

Mercader – Compra: El mercader nos ofrece una selección de objetos a cambio de nuestras almas, estos pueden estar en oferta a un precio reducido basándose en un porcentaje que se mostrará encima del icono del objeto. Se pueden filtrar los objetos por el tipo de objeto que son. Estos filtros son: todos los objetos, armaduras, consumibles y materiales, en ese orden.

Figura 145: compra mercader

Mercader – Venta: En esta ventana podemos vender nuestros objetos al mercader para ganar almas. Se pueden filtrar los objetos por el tipo de objeto que son. Estos filtros son: todos los objetos, armaduras, consumibles y materiales, en ese orden.

Figura 146: venta mercader

Santuario: Al interactuar con un santuario, se podrá elegir entre subir un punto de estadística, guardar partida o viajar a otro santuario.

Figura 147: menú santuario

Santuario – Viaje: Aquí se muestran los santuarios que hemos desbloqueado y podemos realizar un viaje rápido entre ellos.

Figura 148: menú viaje entre santuarios

HUD muerte: Esta es la ventana que vemos al ser derrotados.

Figura 149: pantalla de muerte

5. Herramientas de desarrollo

En esta sección hablaremos de las herramientas que hemos utilizado para desarrollar *The last hanyou*.

5.1 Unity

Unity es un motor de videojuegos multiplataforma con licencia gratuita creado por *Unity* Technologies y por su facilidad de uso se ha convertido en el motor de videojuegos más popular entre los estudios *indie*.

Se ha utilizado la versión de *Unity 2021.1.12f* para la creación de The last hanyou. Esta es la versión más reciente que había en el momento de la creación del proyecto y proporciona numerosas mejorar en el rendimiento del editor, del compilado del código y el trabajo con elementos 2D frente a sus versiones anteriores.

Figura 150: logo de Unity

5.2 Aseprite

Aseprite es un editor de imágenes rasterizadas con licencia de pago creado por David Capello que se especializa en la creación de *pixel art*. Con esta herramienta somos capaces de crear *sprites* con *pixel perfect*, animarlos, crear paletas de colores y podemos crear *tilemaps* para la creación de escenarios.

Se ha utilizado la versión de Aseprite v1.2.25 para la creación de los *sprite* enemigos, del jugador, los elementos de la interfaz, los objetos, entre otros.

Figura 151: logo de Aseprite

5.3 Visual Studio

Visual Studio es un IDE gratuito compatible con múltiples lenguajes de programación desarrollado por Microsoft.

Se ha utilizado la versión de *Visual Studio Community 2019*, ya que tiene una licencia gratuita y permite la creación del código fuente en C#, depuración del código, creación de pruebas y el modelado UML.

Figura 152: logo de Visual Studio

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 TFG/TFM con los ODS y con el/los ODS más relacionados.

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

- **Salud y bienestar**, creo que algo que hemos aprendido de la reciente crisis sanitaria mundial es que no solo hay que cuidar de nuestra salud física sino también de la salud mental. El uso de videojuegos como divertimento ayuda al usuario a reducir su nivel de estrés y ansiedad, por lo que reduce considerablemente el riesgo de que el usuario pueda desarrollar enfermedades como la depresión.

- **Producción y consumo responsables**, también creo que al tratarse de un producto que será distribuido de manera digital, no se consumirán recursos naturales propios de la producción de ediciones físicas. Por tanto, no contribuye a la tala de árboles y el procesamiento contaminante medioambiental del plástico.

- **Trabajo decente y crecimiento económico**, considero que para terminar el proyecto y que salga a la venta tendría que levantar capital a través de páginas de microfinanciación para fundar un estudio de videojuegos y contratar a profesionales de otras disciplinas, por lo que se crearía industria en el país que aportará trabajo a muchas personas y habrá un crecimiento económico.