



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

CAMPUS D'ALCOI

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Politécnica Superior de Alcoy

Desarrollo de un juego generado proceduralmente con
Unity

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Fernández Verdú, Samuel

Tutor/a: Izquierdo Doménech, Juan Jesús

CURSO ACADÉMICO: 2022/2023

Resumen

El proyecto abarca el desarrollo de un videojuego RPG de estilo roguelike en éste se estudiarán diversas técnicas de generación procedural. Para la realización del mismo haremos uso del motor Unity así como diversas herramientas de creación de assets y sprites para el apartado artístico. La idea básica del juego será la de avanzar por niveles generados proceduralmente y la obtención de mejoras para llegar cada vez más lejos en una mazmorra que cambia con cada intento.

El desarrollo se centrará en el estudio de diversas técnicas de generación procedural para poder crear enemigos, niveles y objetos con el fin de estudiar la posibilidad de desarrollar un juego infinito a través de estas técnicas de generación automática y el uso de la programación para optimizar el tiempo invertido de manera específica a cada uno de los integrantes de estos apartados.

El juego será programado utilizando Visual Studio Code que permitirá trabajar con la API de Unity, además de llevar un control del tiempo invertido en todo el proceso utilizando diversas herramientas y un sistema de tarjetas estilo Trello para ordenar, distribuir y elaborar flujos de trabajo con tal de organizar mejor el proyecto.

El transcurso de este TFG se centra principalmente en el desarrollo de la infraestructura y algoritmia que se usará para generar los niveles, enemigos y distintos objetos del juego, con un estudio del transcurso que ha llevado a las soluciones proporcionadas, distintas iteraciones de los propios sistemas y cómo se han llevado a cabo dichos desarrollos. No se descarta añadir más funcionalidades con el fin de proporcionar una experiencia más completa.

El objetivo final es plantear distintos sistemas de generación procedural y otros algoritmos para la generación de assets, enemigos u objetos con el fin de crear código limpio, autocontenido y reutilizable. Además de servir como un estudio de las complejidades que presentan los sistemas de generación procedural y los sistemas intrínsecos al género roguelike.

Palabras clave: Unity, juego, procedural, roguelike, RPG

Abstract

The project covers the development of a roguelike-style RPG video game in which various procedural generation techniques will be studied. To carry it out, we will use the Unity engine as well as various asset and sprite creation tools for the artistic section. The basic idea of the game will be to progress through procedurally generated levels and gain upgrades to get further and further in a dungeon that changes with each attempt.

The development will focus on the study of various procedural generation techniques to be able to create enemies, levels and objects in order to study the possibility of developing an infinite game through these automatic generation techniques and the use of programming to optimize the game. time invested specifically in each of the members of these sections.

The game will be programmed using Visual Studio Code that will allow working with the Unity API, in addition to keeping track of the time spent in the entire process using various tools and a Trello-style card system to order, distribute and create workflows with such to better organize the project.

The course of this TFG focuses mainly on the development of the infrastructure and algorithm that will be used to generate the levels, enemies and different objects of the game, with a study of the course that has led to the solutions provided, different iterations of the systems themselves and how these developments have been carried out. It is not ruled out to add more features in order to provide a more complete experience.

The ultimate goal is to propose different procedural generation systems and other algorithms for the generation of assets, enemies or objects in order to create clean, self-contained and reusable code. In addition to serving as a study of the complexities presented by the procedural generation systems and the systems intrinsic to the roguelike genre.

Keywords: Unity, game, procedural, roguelike, RPG

Resum

El projecte abasta el desenvolupament d'un videojoc RPG d'estil roguelike en aquest s'estudiaran diverses tècniques de generació procedural. Per a la realització del mateix farem ús del motor Unity així com diverses eines de creació de assets i sprites per a l'apartat artístic. La idea bàsica del joc serà la d'avançar per nivells generats proceduralment i l'obtenció de millores per a arribar cada vegada més lluny en una masmorra que canvia amb cada intent.

El desenvolupament se centrarà en l'estudi de diverses tècniques de generació procedural per a poder crear enemics, nivells i objectes amb la finalitat d'estudiar la possibilitat de desenvolupar un joc infinit a través d'aquestes tècniques de generació automàtica i l'ús de la programació per a optimitzar el temps invertit de manera específica a cadascun dels integrants d'aquests apartats.

El joc serà programat utilitzant Visual Studio Code que permetrà treballar amb la API de Unity, a més de portar un control del temps invertit en tot el procés utilitzant diverses eines i un sistema de targetes estil Trello per a ordenar, distribuir i elaborar fluxos de treball amb la condició d'organitzar millor el projecte.

El transcurs d'aquest TFG se centra principalment en el desenvolupament de la infraestructura i algorísmia que s'usarà per a generar els nivells, enemics i diferents objectes del joc, amb un estudi del transcurs que ha portat a les solucions proporcionades, diferents iteracions dels propis sistemes i com s'han dut a terme aquests desenvolupaments. No es descarta afegir més funcionalitats amb la finalitat de proporcionar una experiència més completa.

L'objectiu final és plantejar diferents sistemes de generació procedural i altres algorismes per a la generació de assets, enemics o objectes amb la finalitat de crear codi net, autocontingut i reutilitzable. A més de servir com un estudi de les complexitats que presenten els sistemes de generació procedural i els sistemes intrínsecs al gènere roguelike

Paraules Clau: Unity, joc, procedural, roguelike, RPG

Índice general

Resumen	I
Índice general	IV
Índice de figuras	VI
Agradecimientos	VIII
1. Introducción	1
1.1. Motivación	1
1.2. Descripción del proyecto	1
1.3. Situación actual del mercado de los videojuegos	2
1.4. Género Roguelike	5
2. Planificación del proyecto	9
2.1. Planificación temporal del proyecto	9
2.2. ¿Por qué Unity?	10
2.3. Herramientas para la gestión del proyecto	11
2.4. Otras herramientas utilizadas para el proyecto	12
3. Interfaz	15
3.1. Menú	15
3.2. Interfaz In-game	17
4. Lógica personajes	21
4.1. Jugador	21
4.2. Enemigo	25
5. Generación de nivel	29
5.1. Salas	29
5.2. Algoritmo de generación de nivel	30
5.3. Errores encontrados durante el desarrollo	33
5.4. Lógica de generación	35
6. Música y efectos de sonido	38
6.1. Música	38

6.2. Efectos de sonido	39
7. Trabajo Futuro	40
8. Conclusiones	42

Índice de figuras

1.1. Ganancias de la industria del videojuego desde 2017 hasta 2027	2
1.2. Ganancias de la empresa King 2010-2020	3
1.3. Ganancias de Twitch 2016-2021	3
1.4. Gameplay de Rogue, precursor de los juegos roguelike	6
1.5. Gameplay de The Binding of Issac, roguelike más exitoso de la década y que repopularizó el género	7
2.1. Tabla de gestión de progreso del proyecto	11
2.2. Tabla de gestión de tareas	11
2.3. Vista de edición de Photopea.com	12
2.4. Vista de edición de SoundTrap.com	13
3.1. Menú principal del juego	15
3.2. Menú de opciones	16
3.3. Vista de selección de clase	17
3.4. Vida actual y monedas recogidas durante la partida	18
3.5. Código de la lógica de las monedas soltadas por los enemigos	18
3.6. Pantalla y menú de Game Over	19
3.7. Sistema de talentos	20
4.1. Personaje hecho por Blink utilizado para ser el jugador	21
4.2. Máquina de estados de las animaciones del jugador	22
4.3. Código para el funcionamiento completo de las animaciones y lógica del jugador	23
4.4. Código para el funcionamiento completo de las animaciones y lógica del jugador	24
4.5. Modelo del esqueleto utilizado para el enemigo	25
4.6. Código para el funcionamiento de la lógica del enemigo	26
4.7. Código para el funcionamiento de la lógica del enemigo	27
5.1. Prefab usado como plantilla para crear las salas	29
5.2. Prefab usado para la generación de nivel - Sala NS	30
5.3. Estructura de los arrays que se usan para el algoritmo de generación	31
5.4. Ejemplo del nivel generado con el algoritmo	32
5.5. Error de generación duplicidad de sala	33
5.6. Error de generación ausencia de sala	34
5.7. Código encargado de generar el layout del nivel	35

5.8. Código encargado de generar el layout del nivel	35
5.9. Código encargado de generar el layout del nivel	36
6.1. Configuración audio source background music	38

Agradecimientos

Muchas gracias a mis amigos y familia por apoyarme durante estos meses a pesar de que no han sido los más fáciles y confiar en que iba a poder acabar la carrera a pesar de todo. Gracias también a toda la comunidad universitaria por la ayuda, a mi tutor y a los diversos profesor que me han enseñado montones de cosas durante estos años haciendo todo esto posible. Gracias también a toda la comunidad de desarrolladores, creadores y diseñadores que han trabajado durante años con Unity y dedican parte de su tiempo a facilitarnos las experiencias a otros.

Y en especial gracias a ti David, junto a ti descubrí mi placer por los videojuegos y espero poder seguir disfrutándolo muchos más años aunque ya no estés aquí. Descansa en paz.

1 Introducción

1.1 Motivación

Mi iniciación en el mundo de la tecnología vino de la mano de una pequeña gameboy color y un juego de pokemon. Desde ese instante siempre me encontré fascinado por como se debían crear aquellas pequeñas cajitas que introducía en la consola y me permitirían vivir montones de historias y aventuras.

Fue unos años después cuando por fin puse mis manos en mi primer ordenador que me di cuenta de que estaba enamorado de los videojuegos, de lo que significaban y permitían. Desde ese momento en adelante siempre quise desarrollar mi propio videojuego o dedicarme al sector y poder transmitir esas mismas sensaciones a nuevas personas.

1.2 Descripción del proyecto

El proyecto consiste en el desarrollo de un videojuego 3D con mecánicas Roguelike y generación procedural utilizando Unity como motor para la realización del mismo.

Los juegos de este género consisten de una mazmorra a explorar que se genera de forma procedural o aleatoria y que nos permite experimentar una aventura distinta cada vez que jugamos de nuevo. Además de añadir toques RPG que nos permitirán sentir que tenemos el control de personalizar nuestro personaje y hacerlo sentir diferente en cada partida.

Si bien el género se ha visto muy ampliado a lo largo de los años y muchísimos juegos de diversa índole pueden caer en esta categoría, la base de lo que asentó el género cae en el rango de la definición anterior y es entorno a la premisa que se centra el desarrollo del proyecto aquí presentado.

1.3 Situación actual del mercado de los videojuegos

1.3.1 Evolución económica del mercado

En cuanto al mercado del videojuego durante la última década no ha hecho más que crecer hasta situarse como uno de los pilares de la industria del entretenimiento.

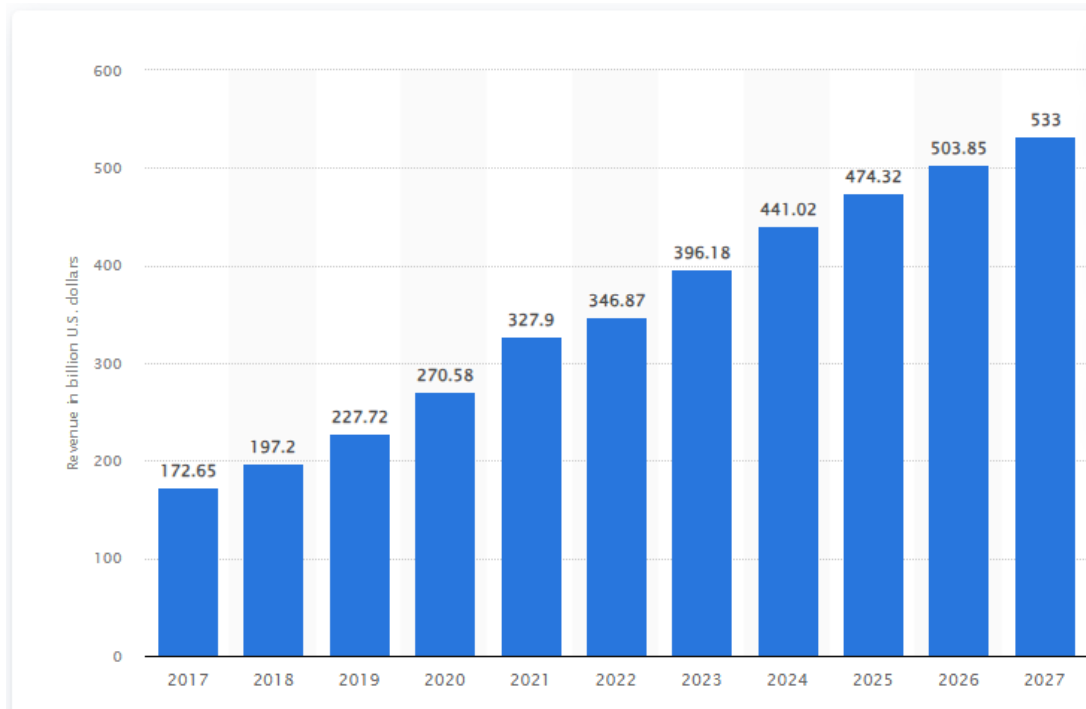


Figura 1.1: Ganancias de la industria del videojuego desde 2017 hasta 2027

En el gráfico superior extraído del artículo de Clement, [24/01/2023a](#) podemos observar como la industria del videojuego se ha disparado en los últimos 6 años y como va a progresar hasta situarse todavía más arriba.

Hoy en día, la industria está separada en 2 grandes sectores los videojuegos móvil y los videojuegos clásicos que salen para consolas y ordenadores. Es gracias a los primeros que la industria del videojuego ha evolucionado hasta ser una de las más grandes de todo el planeta. Al fin y al cabo quién no tiene un teléfono hoy en día. Por situar un pequeño ejemplo la empresa KING, responsable del famoso videojuego Candy Crush ingreso hasta 2164 millones en 2020 como podemos ver a continuación.

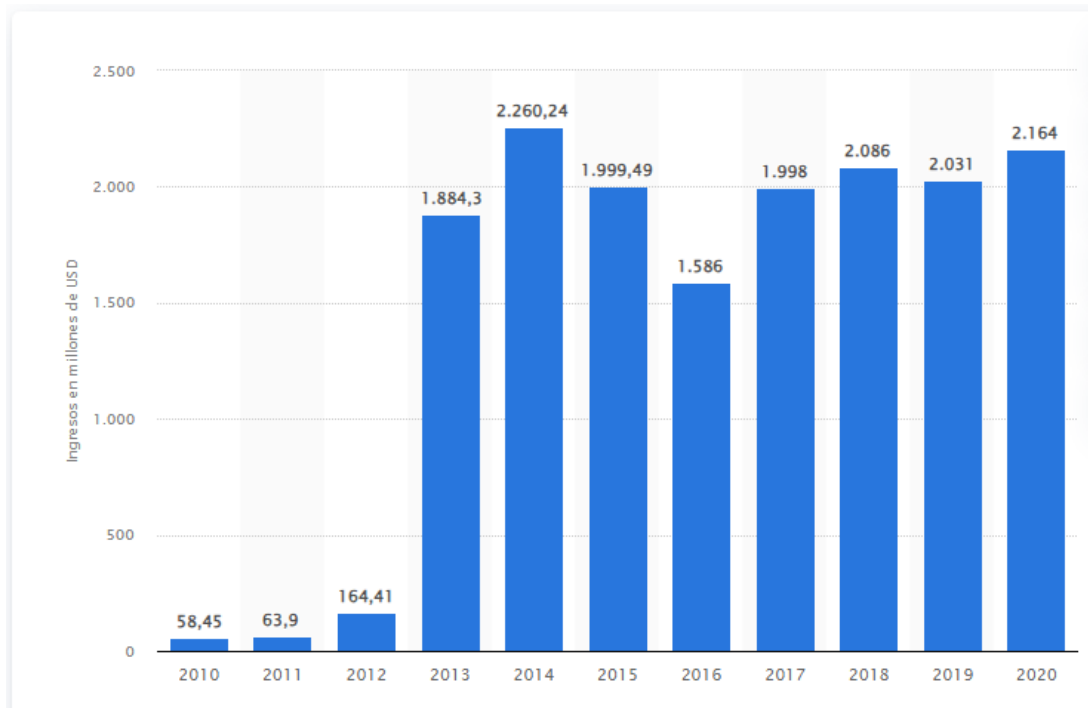


Figura 1.2: Ganancias de la empresa King 2010-2020

En el gráfico se puede apreciar el salto de la empresa gracias al lanzamiento de los smartphones y el famoso Candy Crush que catapultó a la empresa hasta el estrellato. Es difícil poner en perspectiva cuán grande es una empresa como esta que solo es una de tantas de las que forman la industria del videojuego. Por tanto, con tal de dar un poco de perspectiva podemos compararlo con otra industria del entretenimiento que ha ido en crecimiento estos últimos años.

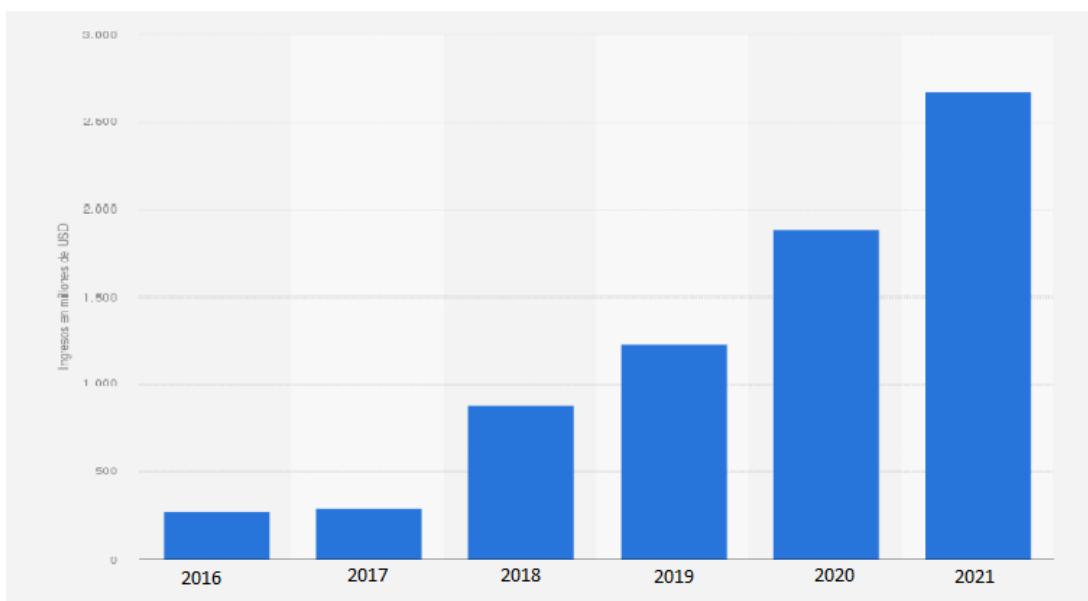


Figura 1.3: Ganancias de Twitch 2016-2021

La plataforma más grande de streaming la famosa Twitch, es sin lugar a dudas uno de los lugares de entretenimiento más visitados hoy en día. Lugar de nacimiento de una de las profesiones más famosas de esta década y sin embargo si comprobamos sus ganancias de los últimos años podemos comprobar como King ya era un titán cuando Twitch apenas andaba despegando.

Si bien es cierto que ha ganado mucha repercusión desde 2020 en adelante hasta donde tenemos datos de King podemos comprobar que hasta en el mejor año de Twitch la diferencia es solo de un 10 por ciento aproximadamente de ganancias.

Con esto puesto en perspectiva podemos hablar de cómo la industria del videojuego ha llegado hasta superar al cine y la música en conjunto en 2022:

"los videojuegos representaron el 42,1 de los ingresos totales del entretenimiento en 2022, creciendo en 2,3 respecto a 2021. Estas cifras han provocado que el sector logre alcanzar un total de 5.421 millones de euros, un récord hasta la fecha" Arias, [04/06/2023](#)

Los videojuegos se han situado en las últimas décadas como uno de los pilares del entretenimiento y han venido para quedarse, ya sea en nuestro smartphone, consola o ordenador. Todos consumimos videojuegos en mayor o menor medida y esto crea un mercado competitivo lleno de productos y posibilidades.

1.3.2 La competitividad en el sector

Debido al notable crecimiento del sector y a la simplicidad que se ha logrado en el desarrollo de videojuegos gracias a motores gratuitos, así como a la disponibilidad de numerosos cursos, vídeos y herramientas gratuitas, la industria de los videojuegos ha experimentado un crecimiento desmedido.

Las principales plataformas de videojuegos, como Steam y en los últimos años Epic Games, presencian miles de lanzamientos a lo largo del año. Solo en 2021, aproximadamente 2300 videojuegos de pago fueron lanzados con una buena cantidad de marketing. Es importante mencionar que estos números no incluyen DLCs ni contenido Free to Play, es decir, aquel que los jugadores pueden disfrutar sin coste adicional.

Esto ha creado una situación en la que los jugadores se encuentran con un exceso de contenido disponible, superando la capacidad de consumo. En otras palabras, se lanzan más videojuegos al año de los que el público puede jugar. Además, los videojuegos compiten con otras formas de entretenimiento, como la música, el cine o el streaming, lo que aumenta la competencia. Aunque la industria genera enormes ingresos, estos no se distribuyen de manera equitativa entre los trabajadores, lo que representa un problema especialmente para los pequeños desarrolladores.

Durante los últimos años, a pesar del aumento de desarrollos independientes, es decir, aquellos realizados por pequeñas empresas o incluso un solo desarrollador, estos se enfrentan a la difícil tarea de obtener visibilidad en medio de una gran cantidad de juegos disponibles. Por tanto, la importancia del marketing se ha incrementado aún más, siendo crucial que los clientes conozcan los lanzamientos antes de que estos lleguen al mercado.

La mayoría de las ventas de videojuegos ocurren durante el período de lanzamiento y novedad, ya que las ventas tienden a estancarse y la visibilidad a disminuir drásticamente una vez pasado ese momento inicial.

1.4 Género Roguelike

En cuanto al género del videojuego a continuación vamos a hablar un poco de su origen, costumbres, evolución y algunos de los mayores éxitos del género en los que me he basado para el desarrollo de este proyecto. Traduciendo literalmente la definición obtenemos lo siguiente:

Roguelike (o rogue-like) es un subgénero de videojuegos de rol tradicionalmente caracterizado por investigar mazmorras a través de niveles generados proceduralmente, juego por turnos, movimiento basado en cuadrícula y muerte permanente del personaje del jugador. La mayoría de los roguelikes se basan en una narrativa de alta fantasía, lo que refleja su influencia de los juegos de rol de mesa como Dungeons and Dragons."Wikipedia, [s.f.](#)

1.4.1 Origen del género

El género originado en los años 80 inspirado como ya hemos dicho por Dungeons and Dragons también bebió fuertemente de las aventuras de texto que rondaban los ordenadores de la época y adaptó un sistema parecido para generar mazmorras mediante texto y explorarlas al más puro estilo Dungeons and Dragons. El primer videojuego que se considera que marcó el género fue Rogue, ver figura 1.4, desarrollado por Glenn Wichman and Michael Toy durante sus años de universidad. La popularidad del juego les llevaría a crear su propia empresa y empezar una ola de desarrolladores del género que crearía los precursores de toda la industria Roguelike.



Figura 1.4: Gameplay de Rogue, precursor de los juegos roguelike

1.4.2 Bases del género

Además de la definición anterior en la convención del desarrollo del videojuego Roguelike, en Berlín en 2008, se definió que permitía a un videojuego determinarse como roguelike según la "Berlin Interpretation".^{esta interpretación estaba basada en 8 factores específicos:}

- *Generación procedural*: El juego utiliza mazmorras generadas aleatoriamente con tal de mejorar la rejugabilidad, en realidad esta generación no es completamente aleatoria sino que genera mazmorras de forma procedural usando un set de normal que permiten generar seeds que permiten una mazmorra que sea capaz de completarse y divertida.
- *Permadeath*: El juego incluye mecánicas de muerte permanente. Una vez un jugador muere debe empezar una nueva partida conocida como "run", que reiniciará el estado de la mazmorra y permitirá obtener algunas mejoras que se traspasarán entre runs.^o bien reiniciará por completo la experiencia de juego. Según Michael Toy, la gente del género ve esta aproximación hacia la muerte permanente no como una herramienta para hacer el videojuego más difícil o frustrante sino para poner peso en cada decisión que el jugador toma y crear una experiencia más inmersiva.
- *Sistemas por turnos*: El juego esta basado en un sistema por turnos dando un tiempo para el jugar a tomar una decisión antes de que la acción continúe.
- *Juego no modal*: El juego es no modal, es decir, toda acción esta disponible para los players en caso de que estén dentro del videojuego.
- *Nivel de complejidad*: El juego incluye un nivel de complejidad debido a los sistemas de juego que hay permitiendo al jugador completar diversos de retos de formas distin-

tas. Es decir, distantes clases, estrategias o armas te permiten superar las dificultades de distintos modos.

- *Mecánicas de supervivencia*: El juego incluye mecánicas de supervivencia necesarias para que el personaje tenga dificultades añadidas a la hora de atravesar la mazmorra, desde hambre, sed o incluso temporizadores que dificulten el mantenerse mucho tiempo en una planta de la mazmorra.
- *Experiencia centrada entorno al combate*: El juego está centrado hacia el combate y la meta es eliminar grandes cantidades de enemigos sin otras opciones pacíficas.
- *Exploración y objetos*: El juego requiere que los jugadores exploren un mapa, descubriendo objetos no identificados y que estos se reseten con cada run. Descubriendo nuevas variedades con características distintas, desde maldiciones a increíbles objetos mágicos como espadas llameantes.

1.4.3 El videojuego roguelike en la última década

Si bien esta lista de objetos se usó para definir lo que era un juego Roguelike en 2008, la mayor parte de jugadores y personas que disfrutaban del género centran la experiencia roguelike entorno a 3 aspectos clave. Generación procedural, un sistema de runs y una experiencia con un nivel de complejidad considerable. Estos tres aspectos son los que han definido y moldeado el género a lo largo de las décadas y nos han traído grandes juegos como The Binding of Isaac, ver figura 1.5 Pokemon Mystery Dungeon, Enter the Gungeon, Crypt of the Necromancer, Hades, Spelunky, The Rogue Legacy, etc.



Figura 1.5: Gameplay de The Binding of Isaac, roguelike más exitoso de la década y que repopularizó el género

A pesar de compartir género todos estos juegos, las experiencias son completamente distintas y por esto decimos que el género se centra en 3 aspectos clave. Es cierto, todos tienen combate, todos generan sus niveles proceduralmente y todos utilizan un sistema de runs o permadeath para general un ciclo de rejugabilidad muy adictivo.

No obstante, todos estos juegos tienen un gameplay muy distintivo el uno del otro, mientras que Hades un hack and slash muy frenético en que tienes que eliminar a todos los enemigos, The Rogue Legacy es un metroidvania que mezcla lo mejor del género con un sistema de clases muy humorístico en el que tu siguiente personaje es el descendiente del actual y tratan de completar el castillo. Por otro lado tenemos Crypt of the Necromancer, que si bien se adhiere más al clásico dungeon crawler inspirado por Rogue, todo en la mazmorra se mueve al ritmo de la música. Mientras que en Spelunky somos una especie de Indiana Jones desenterrando tesoros y peleando con bestias en ruinas 2d.

Cómo podemos observar las experiencias son ilimitadas y si bien todos los juegos entran en la categoría roguelike a la hora de la verdad no tienen tanto en común como nos pudiese parecer más allá de la regla de oro. Explorar, morir y repetir, este es el ciclo de los roguelike y probablemente eso es lo que los vuelve tan adictivos. Esa incertidumbre, esa exploración de cada run, esa tensión de saber que un error puede acabar con la vida de tu personaje y necesitas volver a pelear por llegar hasta donde estabas, no obstante no de una forma amarga sino de una forma adictiva y que no te permite despegarte de la pantalla.

2 Planificación del proyecto

A la hora de planificar el proyecto, era importante tener en cuenta unas cuantas cosas, primero que todo la idea del proyecto y lo que abarcaba. Crear desde cero sistemas que pudiesen general proceduralmente una mazmorra suponía un reto que, si bien no era imposible, sin mucha experiencia previa podía tornarse un problema si me encontraba con según que errores. Por esto era importante realizar un buen estudio previo sobre el tipo de algoritmos que podían usarse y como funcionaban algunos de los sistemas utilizados por otros videojuegos de un estilo parecido.

Por el otro lado estaba elegir motor, gestionar el tiempo y preparar todo lo referente a la gestión del proyecto, que en su propia medida ya era un reto, pues tratar de abarcar todos los aspectos que conllevan crear un videojuego siendo una sola persona sin muchos conocimientos previos supone un coste temporal bastante alto, más de lo que pudiese haberme planteado en un primer instante.

2.1 Planificación temporal del proyecto

Las fechas de inicio del proyecto rondan Octubre de 2022, al menos los primeros registros de trabajo que tengo son de esas fechas por tanto si tenemos en cuenta que la idea es presentar este proyecto para Julio de 2023 el entorno total de tiempo sería de unos 9-10 meses. Debido a problemas el desarrollo se aplazó hasta Marzo de 2023 por lo que el tiempo de desarrollo se redujo drásticamente.

En este punto, llevaba ya unas cuantas horas trabajadas, pero lo cierto es que no tengo un registro previo específico de cuantas, especialmente fueron de estudio previo y preparación para poder desarrollar el proyecto correctamente. La idea era distribuir unas 300h de trabajo aproximadamente en los meses que me quedaban ya que consideraba que ese era tiempo que llevaba hacer un proyecto en condiciones para poder defender ante el tribunal. Así que distribuí la lista de tareas que creía necesarias para tener una base decente y considerable a lo largo de los 5 meses.

Así que el plan era dedicar unas 15h semanales al proyecto desde Marzo hasta finales de Julio, si todo iba bien debería poder llegar así que distribuí el trabajo mensualmente.

60h mensualmente al proyecto con tal de obtener el mejor resultado posible. No obstante los problemas encontrados durante el desarrollo reducirían la cantidad de cosas que se podrían añadir al desarrollo final y me quedé a mitad camino del objetivo de lanzar un juego finalizado. Creo que con la cantidad de tiempo inicial hubiese sido más realista obtener un proyecto completo, pero considero que se ha hecho un buen uso del tiempo de desarrollo durante estos 5 meses.

Durante los aproximadamente 200 días de desarrollo y a lo largo de unas 300 horas según vemos en la tabla superior se implementaron las bases de todos los sistemas citados y otros sistemas que no llegaron a ser implementados por falta de tiempo o problemas encontrados durante el desarrollo.

Marzo	Abril	Mayo	Junio	Julio
Generación procedural de niveles	Lógica de los enemigos y protagonista	Clases y Talentos	Menus, Interfaz, mejoras	Mejoras y memoria
Creación de los prefabs	Animaciones del protagonista	Musica	Error de errores con los sistemas	
Estudiar lógica y algoritmos	Sistemas de salud y puntuación	Efectos de sonido		
Arreglo de bugs y errores				

2.2 ¿Por qué Unity?

La principal razón para elegir este motor es que es el motor en el que más experiencia previa tenía, otra buena opción hubiese sido Unreal Engine, pero si además de aprender como funciona sumamos todo el trabajo que era necesario para llevar a cabo el proyecto sin lugar a dudas no hubiese dado tiempo a realizar una base jugable decente.

Partiendo de esto las opciones eran claras tanto Unreal Engine, Gamemaker o Godot que eran las alternativas, implicaban dedicar un tiempo de curva de entrada a familiarizarme con el entorno que no tenía que dedicar en la misma medida a Unreal Engine gracias a los conocimientos previos obtenidos en la asignatura de Introducción al desarrollo de videojuegos además de conocimientos propios adquiridos por mi interés en la materia.

Otras de las razones para utilizar Unity son que:

- Motor muy potente y con muchas posibilidades.
- Capacidad de realizar tanto desarrollos 2d como 3d con facilidad, lo que permitía flexibilidad a la hora de plantear que tipo de proyecto quería crear mientras exploraba las posibilidades de la generación procedural.
- Infinidad de contenido de aprendizaje gratuito en todas sus formas, desde tutoriales en vídeos, cursos completos en udemy o otras plataformas.
- Se habían realizado proyectos propios con anterioridad lo que consideraba importante a la hora de trabajar con muchos de los sistemas.
- Sistema integrado de audio, música y animaciones muy simple y fácil de utilizar.

- Infinidad de assets a un click de distancia en una store incluida en el propio motor, pudiendo reducir el tiempo de desarrollo considerablemente al no tener que diseñar todo el proyecto desde cero.

2.3 Herramientas para la gestión del proyecto

Durante el proyecto se ha pensado en utilizar diversas herramientas de gestión, pero definitivamente me decanté por preparar un excel. Pensé que resultaría lo más cómodo una vez estuviese toda la lógica relacionada con el proyecto integrada en el mismo y me permitiría realizar operaciones, tablas o otro tipo de gestión de datos. Así si quería o planteaba usarlos en un futuro para este mismo proyecto, o futuros proyectos, podrías hacerlo. Por tanto realicé un pequeño desarrollo para adaptar las hojas a la gestión del proyecto en cuestión.








ROGUELIKE	PROGRESS	
Sistema de Generación de salas	100,00%	
Enemigos	70,00%	
Personaje	30,00%	
Menus	100,00%	
Interfaz	60,00%	
Clases	20,00%	
Talentos	10,00%	

Figura 2.1: Tabla de gestión de progreso del proyecto

Cree 2 funcionalidades principales para gestionar el proyecto. Primero una tabla de progreso del proyecto en el que podía ver aproximadamente según todas las tareas asignadas a cada bloque de desarrollo que progreso total tenía con respecto a las tareas asignadas, véase la figura 2.1

TAREAS			
Tarea	Tipo	Estado	Horas dedicadas
Añadir suelo a todas los prefabs de las salas	Development	Done	0:15
Arreglar spawnpoints y error de duplicación de salas	Bug	Done	0:25
Salas tratando de conectarse a salas sin entrada	Bug	Work In Progress	1:25
Salas superponiendose una sobre otra	Bug	Work In Progress	2:25
Gestionar la generación del nivel mediante un árbol	Idea	Not Started	3:25
Estudio y planteamiento de que tipo de cámara se quiere	Idea	Done	1

Figura 2.2: Tabla de gestión de tareas

En segundo lugar tenemos una tabla de gestión de tareas en la que podemos crear una lista de tareas asignar el tipo de tarea que es entre idea, desarrollo o bug y el estado en el es que se encuentra actualmente además de mostrar la cantidad de tiempo dedicado a la tarea en cuestión. De este modo se puede llevar un registro genérico del estado de las tareas y relacionarlo con la tabla de gestión del proceso del proyecto.

2.4 Otras herramientas utilizadas para el proyecto

Además de Unity para el desarrollo principal y Excel se ha hecho uso de una lista de programas para apoyos en el desarrollo en apartados como las animaciones, modelos, texturas y música. Los programas que han sido utilizados son:

2.4.1 *Blender*

Es un programa de edición 3d que nos permite realizar multitud de tareas desde modelado, hasta preparar personajes que no tienen esqueletos o simplemente diseñar unos modelos rápidos. Si bien no se ha diseñado nada desde cero en el proyecto con tal de evitar problemas relacionados con el diseño y sumar horas de trabajo ajenas a la programación, si se hizo algún arreglo al modelo utilizado para el enemigo con tal de que cuadrara correctamente en Unity y no diese problemas.

2.4.2 *Photopea.com*

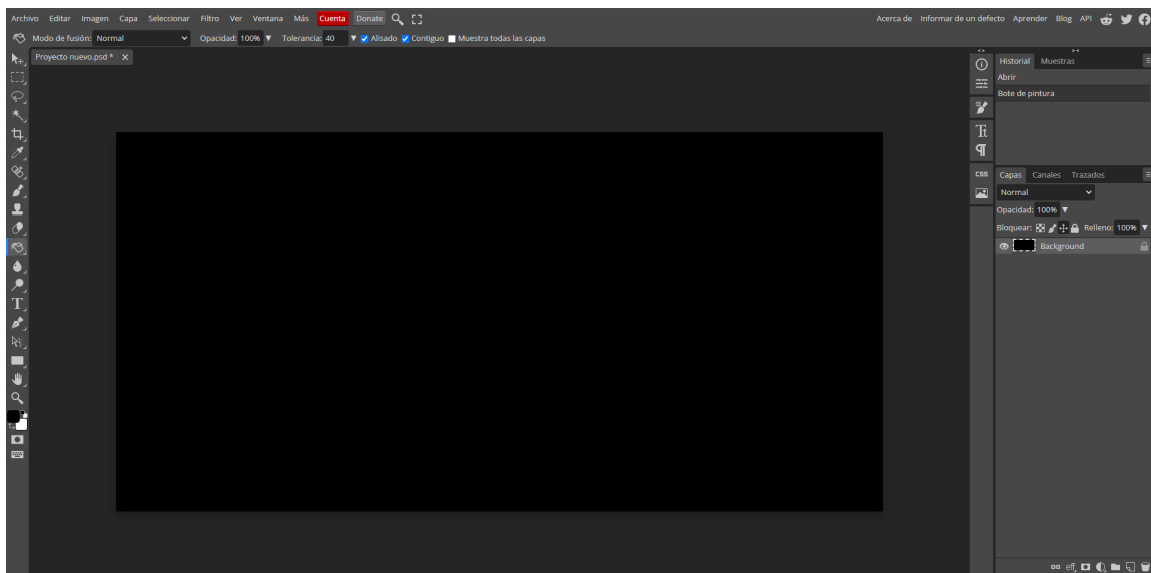


Figura 2.3: Vista de edición de Photopea.com

Programa de edición de imagen bastante potente, gratuito y online que imita las prestaciones de Photoshop de una forma bastante eficiente, resultando en una herramienta muy potente y portátil que permite multitud de ediciones, se ha utiliza para crear la mayoría de Interfaces del videojuego junto a Paint.

2.4.3 *Paint*

Programa nativo de Windows perfecto para realizar algún arreglo rápido en alguna textura o situación con alguna interfaz, cuadrar píxeles o arreglar algún tipo de problema del estilo. No es especialmente potente, pero nos permite realizar arreglos de una forma bastante eficiente y rápido y nos permite exportar imágenes en multitud de formatos, cosa para la que resulta bastante útil

2.4.4 *Microsoft Visual Studio*

Básicamente el IDE por excelencia para trabajar en C Sharp, viene de la mano de Unity usualmente aunque hay otras alternativas como Visual Studio Code en el que también se estuvo trabajando al inicio del proyecto, sin ningún problema a la hora de editar con Unity los scripts. Viene configurado de base con Unity, pero se pueden configurar otros IDE's si tienes algún tipo de preferencia a la hora de trabajar con C Sharp.

2.4.5 *SoundTrap.com*

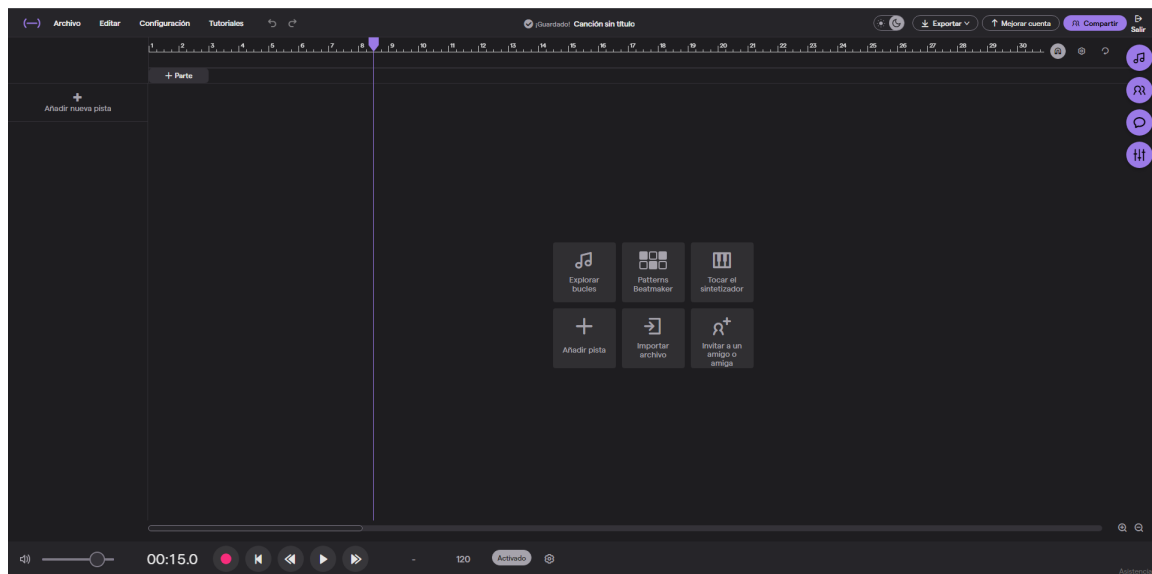


Figura 2.4: Vista de edición de SoundTrap.com

Potente herramienta online de edición de música facilitada por Spotify. Se ha utilizado para generar las pistas de música que suenan de fondo en el menú y nivel, es una herramienta muy potente, pero con una curvad de entrada muy alta si no tienes conocimientos previos musicales ya que es bastante compleja.

2.4.6 *sfxr.me*

Sencilla y potente herramienta online que nos permite generar simples efectos de sonido estilo 8bit para complementar nuestro juego, se ha usado para generar sistemas como el sonido de las monedas y otros aunque no se han implementado al código finalmente por falta de tiempo.

3 Interfaz

A la hora de crear interfaces Unity nos facilita un sistema en el que superpone un canvas en la escena de juego, es decir, renderiza un plano que contiene todos los objetos necesarios para componer la interfaz necesaria para diversas tareas. Desde la creación de menús hasta el muestreo de datos relativos a la partida, como la vida del personaje o la cantidad de monedas que tenemos en nuestra posesión en cada instante.

3.1 Menú

En cuanto a menús podemos diferenciar 2 menús anteriores a la vista de juego que ocupan un mismo canvas, pero que se activan y desactivan las respectivas vistas para permitir ver el otro apartado del menú.

3.1.1 *Menú principal*

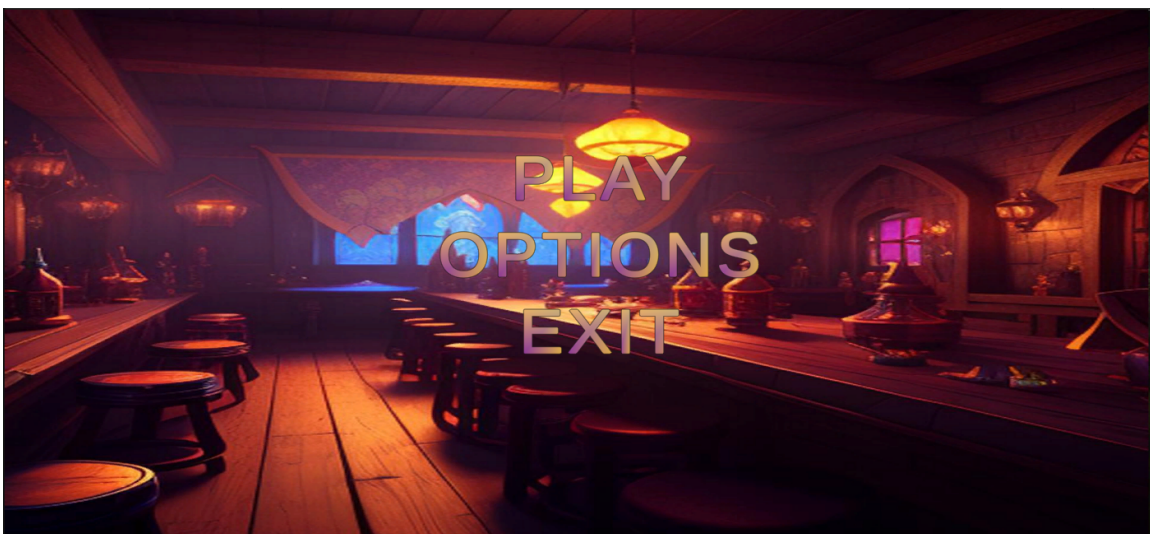


Figura 3.1: Menú principal del juego

Finalmente opte por un menú simple en que pudiésemos elegir entre 3 opciones:

- *Play*: Ejecuta la escena de juego, lanzando las instrucciones necesarias para llamar al menú de selección de clase y empezar la generación del nivel.
- *Options*: Desactiva el canvas que contiene el menú principal y activa el menú de opciones permitiéndonos cambiar algunas especificaciones del juego.
- *Exit*: Cierra el juego.

3.1.2 Menú opciones



Figura 3.2: Menú de opciones

El menú de opciones contempla unas opciones simples puesto que no hay muchas cosas que se pueden optimizar actualmente en cuanto a rendimiento, pero es una de las ideas a futuro. Las opciones que encontramos en este menú son:

- *Volume*: Permite regular la barra de volumen al nivel deseado.
- *Resolution*: Despliega una lista de resoluciones para poder elegir la que más se adapte al dispositivo en el que se está ejecutando.
- *Fullscreen*: Es un campo booleano que nos permite seleccionar si queremos o no pantalla completa.
- *Back*: Te devuelve a al menú principal.

3.2 Interfaz In-game

3.2.1 Selección de clase

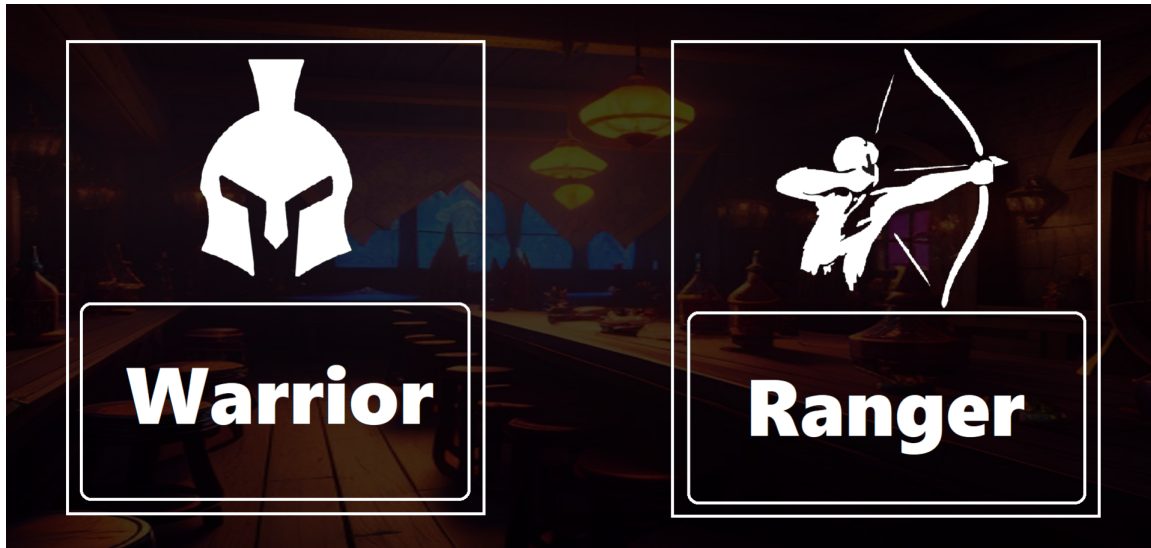


Figura 3.3: Vista de selección de clase

Una vez se selecciona comenzar la partida la vista superior es la primera vista que se obtiene del juego, en ella se plantea una selección simple entre dos botones que determinarán la lógica de la clase del personaje definiendo una serie de parámetros:

- *Atk*: Cantidad de daño realizado a los enemigos por ataque.
- *Defense*: Cantidad de daño que se mitiga de forma pasiva.
- *Speed*: Velocidad de movimiento.
- *Hp*: Cantidad máxima de vida.
- *Atk Speed*: Velocidad de ataque por segundo de la clase.
- *Atk Range*: Rango de ataque, valor booleano para determinar si el ataque es un proyectil o no.

3.2.2 Información de partida



Figura 3.4: Vida actual y monedas recogidas durante la partida

En la partida he optado por una interfaz minimalista que distraiga lo menos posible y no se interponga mucho en el gameplay o resulte aparatosa.

Una barra de vida que se reduce proporcionalmente hasta llegar a 0 y vaciarse y un pequeño display que muestra las monedas recogidas de los enemigos.

```
Script de Unity (1 referencia de recurso) | 0 referencias
6 public class Money : MonoBehaviour
7 {
8     private int coins;
9     private AudioSource audioSource;
10    [SerializeField] TextMeshProUGUI money;
11    // Start is called before the first frame update
12    void Start()
13    {
14        audioSource = GetComponent<AudioSource>();
15    }
16
17    // Update is called once per frame
18    void Update()
19    {
20        transform.Rotate(0, 50 * Time.deltaTime, 0);
21    }
22
23    private void OnTriggerEnter(Collider other)
24    {
25        audioSource.Play();
26        coins++;
27        money.SetText(""+coins);
28        if (other.name == "Player") {
29            Destroy(gameObject);
30        }
31    }
32 }
```

Figura 3.5: Código de la lógica de las monedas soltadas por los enemigos

Podemos ver la simple lógica implementada en la que al entrar en contacto con el jugador las monedas son destruidas y sumadas a la interfaz citada anteriormente. Además de añadir una pequeña animación que hace que la moneda ruede hasta ser recogida.

3.2.3 *Game Over*



Figura 3.6: Pantalla y menú de Game Over

Cómo se puede ver se trata de un menú sencillo con 3 botones y un pequeño texto de mensaje que nos dice cuantas monedas hemos recogido durante la partida, estas monedas se añaden a una variable global que se guarda entre partidas y que nos permite utilizar las monedas para desbloquear los talentos. Los botones del menú ejecutan las siguientes acciones.

- *Continue*: Ejecuta la escena de juego, lanzando las instrucciones necesarias para llamar al menú de selección de clase y empezar la generación del nivel.
- *Talents*: Abre la vista de talentos descrita a continuación.
- *Exit*: Nos devuelve al menú principal.

3.2.4 Talentos y mejoras

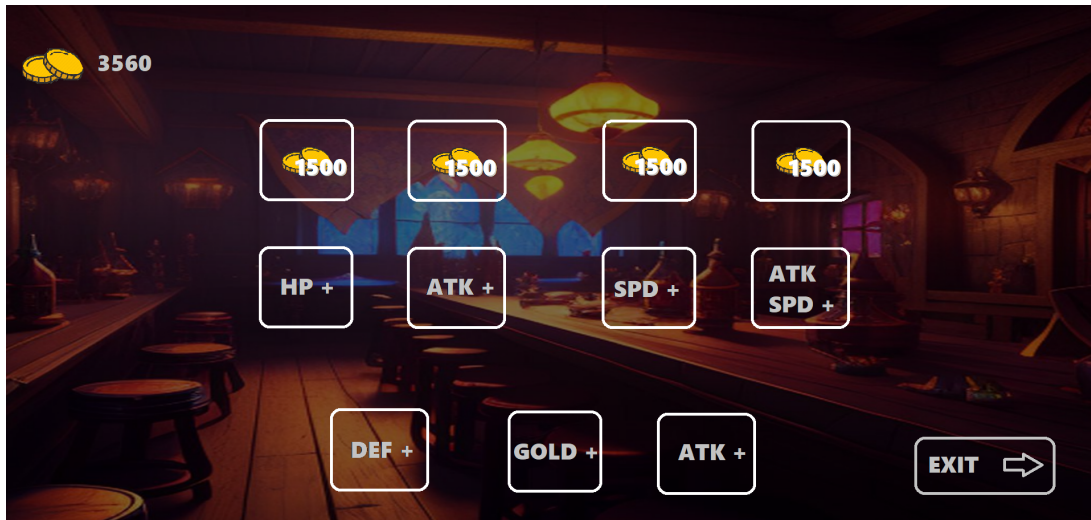


Figura 3.7: Sistema de talentos

Como podemos apreciar en la figura superior encontramos una pequeña interfaz con botones que utilizan una lógica muy simple de booleanos y comprobación de cuantas monedas se han conseguido hasta el momento. Arriba a la izquierda tenemos un texto que nos muestra las monedas que tenemos para poder gastar y abajo a la izquierda un botón que nos devuelve a la pantalla de Game Over.

4 Lógica personajes

4.1 Jugador



Figura 4.1: Personaje hecho por Blink utilizado para ser el jugador

Para el personaje del jugador se ha utilizado un personaje lowpoly desarrollado y animado por Blink, [s.f.](#) y proporcionado a la comunidad de Unity mediante la asset Store, se puede ver el personaje en la figura anterior. Además de estar completamente riggeado, es decir, con un esqueleto completo con articulaciones que permiten animarlo. Incluido con todo esto viene un pack de animaciones que aceleró bastante la implementación viene con un par de complementos simples para el personaje que nos permiten hacerlo variar en función de la clase que se ha seleccionado.

4.1.1 Animaciones

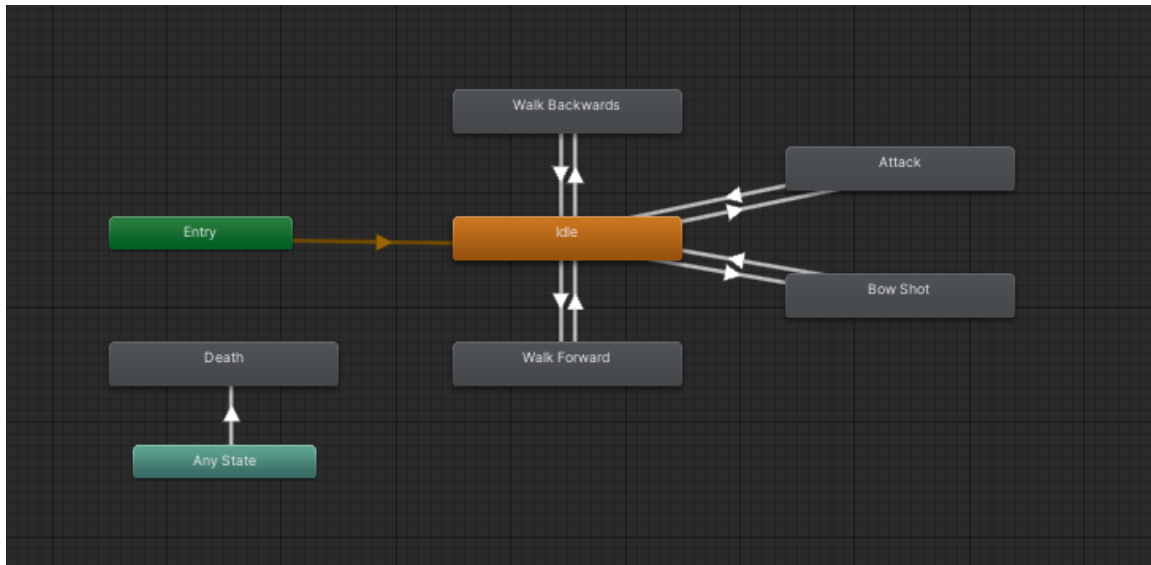


Figura 4.2: Máquina de estados de las animaciones del jugador

Lo que se puede ver en la figura anterior es una máquina de estados que representa los estados de animación referentes a las acciones del personaje. En esta máquina podemos distinguir los siguientes estados con sus razones de activación:

- *Idle*: Posición de espera antes de hacer cualquier tipo de acción.
- *Moving Forward*: Árbol de animaciones que engloba los movimientos hacia la parte superior de la pantalla.
- *Moving Backward*: Árbol de animaciones que engloba los movimientos hacia la parte inferior de la pantalla.
- *Death*: Animación de muerte al ver reducida tu vida a 0.
- *Attack*: Animación de ataque de la clase Warrior.
- *Bow Shot*: Animación de ataque de la clase Ranger.

4.1.2 Código

```
6 public class PlayerController : MonoBehaviour
7 {
8     [SerializeField] private Rigidbody _rb;
9     [SerializeField] private Transform _model;
10    [SerializeField] private float _speed = 5;
11    [SerializeField] private float _turnSpeed = 360;
12    private Vector3 _input;
13    private const float MAX_HEALTH = 100f;
14    public float health = MAX_HEALTH;
15    public Image healthBar;
16    private bool attacking = false;
17    private GameObject attackArea = default;
18    private float atkSpeed = 0.25f;
19    private float timer = 0;
20    private bool atkRange = false;
21    public Animator animator;
22    public Transform player;
23    private GameObject gameOver;
24
25    0 Mensaje de Unity | 0 referencias
26    private void Start()
27    {
28        attackArea = transform.GetChild(0).gameObject;
29    }
30
31    0 Mensaje de Unity | 0 referencias
32    private void Update()
33    {
34        ObtainInput();
35        Look();
36        Health();
37        if (Input.GetKeyDown(KeyCode.Space)) Attack();
38        if (health == 0) Death();
39    }
40
41    1 referencia
42    private void Death() {
43
44        if (player.GetComponent<PlayerController>().health <= 0) animator.SetFloat("Health", 0);
45        gameOver.SetActive(true);
46        gameObject.SetActive(false);
47    }
48
49    1 referencia
50    private void Attack() {
51
52        attacking = true;
53        attackArea.SetActive(attacking);
54        animator.SetBool("Attacking", attacking);
55        if (attacking)
56        {
57            timer += Time.deltaTime;
58            if (timer >= atkSpeed)
59            {
60                timer = 0;
61                attacking = false;
62                attackArea.SetActive(attacking);
63                animator.SetBool("Attacking", attacking);
64            }
65        }
66    }
67 }
```

Figura 4.3: Código para el funcionamiento completo de las animaciones y lógica del jugador


```

61 private void FixedUpdate()
62 {
63     Move();
64 }
65
66 1 referencia
67 private void ObtainInput()
68 {
69     _input = new Vector3(Input.GetAxisRaw("Horizontal"), 0, Input.GetAxisRaw("Vertical"));
70
71     if (Input.GetAxis("Vertical") < 0)
72     {
73         player.transform.rotation = new Quaternion(0, -45, 0, 0);
74     }
75     animator.SetFloat("Vertical", Input.GetAxis("Vertical"));
76     animator.SetFloat("Horizontal", Input.GetAxis("Horizontal"));
77 }
78 1 referencia
79 private void Health()
80 {
81     healthBar.fillAmount = health / MAX_HEALTH;
82 }
83 1 referencia
84 private void Move()
85 {
86     _rb.MovePosition(transform.position + _input.ToIso() * _input.normalized.magnitude * _speed * Time.deltaTime);
87 }
88 1 referencia
89 private void Look()
90 {
91     if (_input == Vector3.zero) return;
92
93     Quaternion rot = Quaternion.LookRotation(_input.ToIso(), Vector3.up);
94     _model.rotation = Quaternion.RotateTowards(_model.rotation, rot, _turnSpeed * Time.deltaTime);
95 }
96 Mensaje de Unity | 0 referencias
97 private void OnCollisionEnter(Collision collision)
98 {
99     if (collision.gameObject.tag == "enemy")
100     {
101         health = health-10;
102         print(health);
103     }
104 }
105 0 referencias
106 public static class Helpers
107 {
108     private static Matrix4x4 _isoMatrix = Matrix4x4.Rotate(Quaternion.Euler(0, 45, 0));
109     2 referencias
110     public static Vector3 ToIso(this Vector3 input) => _isoMatrix.MultiplyPoint3x4(input);
111 }

```

Figura 4.4: Código para el funcionamiento completo de las animaciones y lógica del jugador

El código que encontramos en las figuras 4.3 y 4.7 engloba todo el funcionamiento del personaje del Jugador. Podemos diferenciar tres grandes bloques de funcionamiento en el código:

- *Animaciones*: Todas instrucciones relacionadas con el animator básicamente implementan el correcto funcionamiento de todas las animaciones relacionadas con el movimiento, el ataque y la muerte. Se ven implicados métodos como Attack, Death o ObtainInput.
- *Movimiento*: Debido a la orientación de la cámara es importante transformar el movimiento vectorial al de la vista isométrica, gracias a la clase Helpers se realizan todas las operaciones necesarias para la conversión. Los métodos implicados en el movimiento son Look, Move y ObtainInput.

- *Combate*: Por último tenemos el combate, relacionado con él están todas las variables de vida y ataque, como `atkSpeed`, `timer`, `atkRange`, `attacking` a la vez que los métodos que hacen usos de estas. `Attack`, `Health` y `Death`.

4.2 Enemigo

Para esta primera versión se ha optado por crear un enemigo con una lógica no muy compleja y con la capacidad de atacar al jugador, persiguiéndole y lanzándole unos proyectiles de tal forma que este tenga que esquivar estos y acabar con el enemigo.



Figura 4.5: Modelo del esqueleto utilizado para el enemigo

4.2.1 Animaciones

En la versión actual del proyecto los enemigos no tienen un paquete de animaciones más allá de instanciar el prefab del ataque y moverse hacia el jugador en animación de idle.

4.2.2 Código

```
6 public class EnemyAI : MonoBehaviour
7 {
8     public NavMeshAgent agent;
9     public Transform player;
10    public LayerMask whatIsGround, whatIsPlayer;
11    public float health;
12
13    public Vector3 walkPoint;
14    bool walkPointSet;
15    public float walkRange;
16    public float AtkSpeed;
17    bool attacking;
18    public GameObject attack;
19    public float sightRange, attackRange;
20    public bool InSightRange, InAttackRange;
21
22    [Mensaje de Unity | 0 referencias]
23    private void Awake()
24    {
25        player = GameObject.Find("Player").transform;
26        agent = GetComponent<NavMeshAgent>();
27    }
28
29    [Mensaje de Unity | 0 referencias]
30    private void Update()
31    {
32        InSightRange = Physics.CheckSphere(transform.position, sightRange, whatIsPlayer);
33        InAttackRange = Physics.CheckSphere(transform.position, attackRange, whatIsPlayer);
34        if (!InSightRange && !InAttackRange) Patrolling();
35        if (InSightRange && !InAttackRange) Chasing();
36        if (InAttackRange && InSightRange) Attacking();
37    }
38
39    [1 referencia]
40    private void Patrolling()
41    {
42        if (!walkPointSet) newWalkPoint();
43        if (walkPointSet)
44            agent.SetDestination(walkPoint);
45        Vector3 distanceToWalkPoint = transform.position - walkPoint;
46        if (distanceToWalkPoint.magnitude < 1f)
47            walkPointSet = false;
48    }
49
50    [1 referencia]
51    private void newWalkPoint()
52    {
53        float randomZ = Random.Range(-walkRange, walkRange);
54        float randomX = Random.Range(-walkRange, walkRange);
55        walkPoint = new Vector3(transform.position.x + randomX, transform.position.y, transform.position.z + randomZ);
56        if (Physics.Raycast(walkPoint, -transform.up, 2f, whatIsGround))
57            walkPointSet = true;
58    }
59 }
```

Figura 4.6: Código para el funcionamiento de la lógica del enemigo

```

57 private void Attacking()
58 {
59     agent.SetDestination(transform.position);
60     transform.LookAt(player);
61
62     if (!attacking)
63     {
64         Rigidbody rb = Instantiate(attack, transform.position, Quaternion.identity).GetComponent<Rigidbody>();
65         attacking = true;
66         rb.AddForce(transform.forward * 32f, ForceMode.Impulse);
67         rb.AddForce(transform.up * 8f, ForceMode.Impulse);
68         Invoke(nameof(ResetAttack), AtkSpeed);
69     }
70 }
71 1 referencia
72 private void ResetAttack()
73 {
74     attacking = false;
75 }
76 1 referencia
77 private void DestroyEnemy()
78 {
79     Destroy(gameObject);
80     Instantiate(money, transform.position, Quaternion.identity);
81 }
82 0 referencias
83 public void TakeDamage(int damage)
84 {
85     health -= damage;
86
87     if (health <= 0) Invoke(nameof(DestroyEnemy), 0.5f);
88 }
89 1 referencia
90 private void Chasing()
91 {
92     agent.SetDestination(player.position);
93 }
94
95 Mensaje de Unity | 0 referencias
96 private void OnDrawGizmosSelected()
97 {
98     Gizmos.color = Color.red;
99     Gizmos.DrawWireSphere(transform.position, attackRange);
100    Gizmos.color = Color.yellow;
101    Gizmos.DrawWireSphere(transform.position, sightRange);
102 }

```

Figura 4.7: Código para el funcionamiento de la lógica del enemigo

El enemigo funciona como una máquina de estados de tres posiciones. Es decir:

- *Patrulla*: Si el enemigo no tiene al jugador a su alcance de visión, `sightRange`, buscará un punto aleatorio cercano al que moverse y repetirá esta acción hasta que el jugador entre en el `sightRange`.
- *Persigue*: Si el enemigo tiene al jugador a su alcance de visión, `sightRange`, caminará en la dirección del jugador hasta alcanzar el rango de ataque, `attackRange`, si el jugador consigue abandonar el rango de visión, el enemigo volverá al estado patrulla, pero si el enemigo se acerca hasta el `attackRange` pasará a atacar.
- *Ataca*: Mientras el jugador se encuentre en el rango de ataque del enemigo, `attackRange`, el enemigo atacará instanciando unos proyectiles que simularán un hueso de esqueleto que volará hasta la posición del enemigo, estos proyectiles son destruidos al tocar cualquier superficie y si golpean al jugador le harán daño.

Para gestionar los rangos de esta máquina de estados el enemigo dibuja unas esferas que determinan el rango en el que detecta al jugador, si el jugador entra en estas esferas el enemigo cambia de estado y procede a realizar la acción correspondiente al rango en el que se encuentre el jugador. Finalmente podemos ver como en al morir instancia las monedas anteriormente comentadas en la sección [3.2.2](#)

Esta clase se podrá usar para futuras implementaciones separando parte del código referente a las acciones específicas de este enemigo y permitiendo crear otras máquinas de estados para otros enemigos que por ejemplo solo patrullen o que no patrullen y simplemente persigan al enemigo entre otros.

5 Generación de nivel

Tras mucho estudio y revisión de como podía general el nivel opté por una mazmorra modular estilo BindingOfIsaacRebirth, [s.f.](#)

Este estilo de generación se caracteriza por tener un sistema de salas previamente generadas que posteriormente se utilizan para generar un nivel aleatorio mediante la conexión de las mismas y son rellenas de forma aleatoria con enemigos y objetos por partes iguales.

5.1 Salas

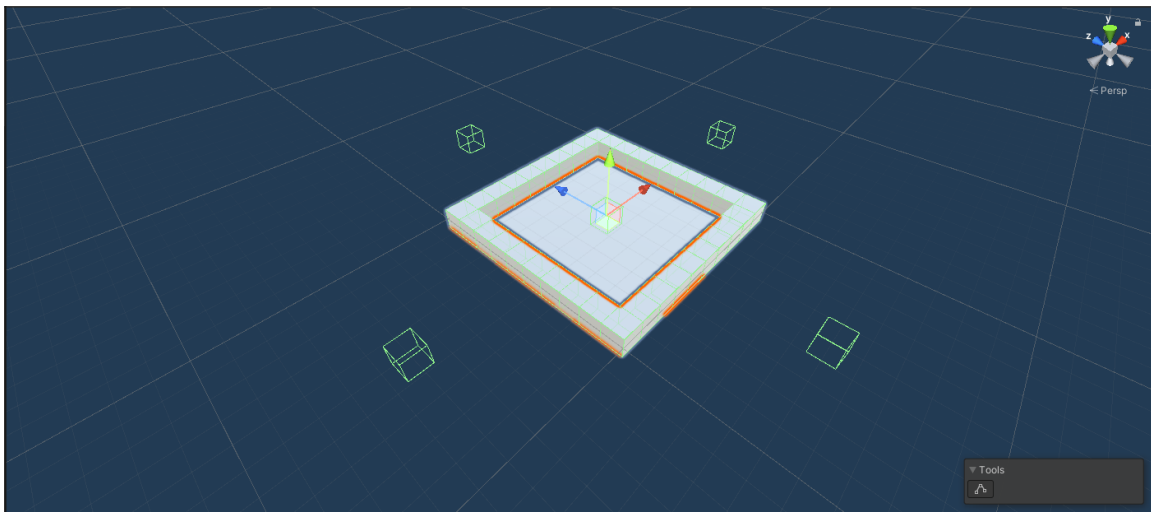


Figura 5.1: Prefab usado como plantilla para crear las salas

Al inicio del desarrollo el primer paso fue generar un prefab que permitiese trabajar con la lógica elegida, para ello diseñe un grupo de salas que pudiesen interconectarse entre ellas.

Aquí fue donde empecé a encontrarme con los primeros problemas del desarrollo, olvidando poner suelos a las salas, lo cual me hizo tener que revisar todos los prefabs y añadir 1 a

1 los suelos de nuevo. Además de nombrar las salas correctamente, opte por un sistema de puntos cardinales en los que podía saber que entradas tenían abiertas cada sala en función de dicha nomenclatura como podremos ver más abajo.

Una vez creado el prefab este estaba compuesto por un plano que serviría de suelo y un montón de cubos que funcionarían como paredes modulares que podría editar para generar las distintas aberturas. Además en cada uno de los puntos cardinales se sitúa un spawnpoint, se trata de un objeto vacío que se utilizará posteriormente para la generación del nivel y estará presente en todas las caras de la sala que tengan una puerta, a continuación se puede ver un ejemplo de un prefab editado y finalizado.

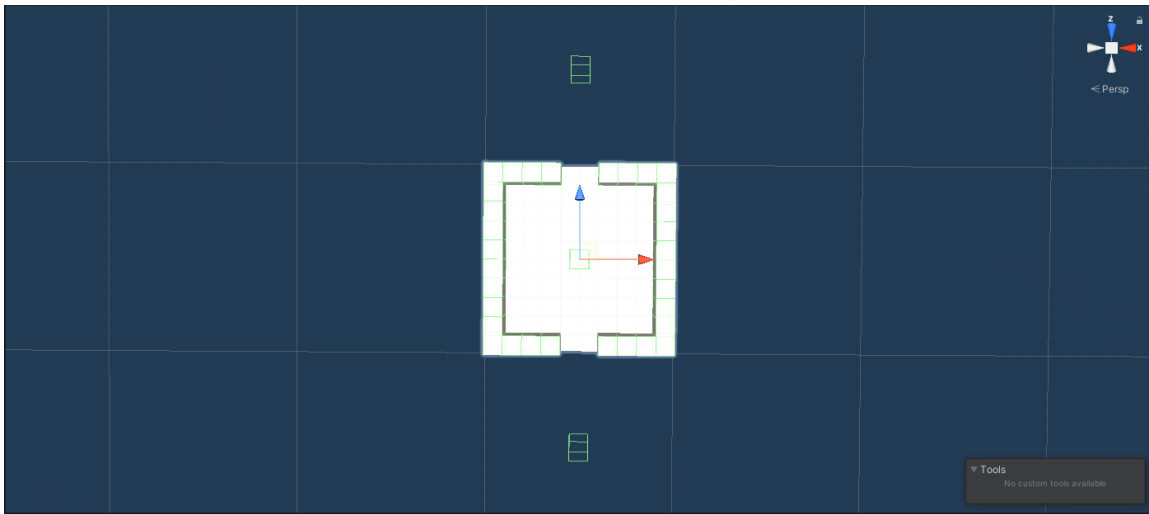


Figura 5.2: Prefab usado para la generación de nivel - Sala NS

Cómo podemos apreciar en el nombre de la figura, esta sala es una Sala NS, es decir, una sala con aperturas en el Norte y Sur. Por tanto encontraremos spawnpoints en dichos puntos que nos permitirán realizar la generación del nivel cómo veremos más adelante al hablar del algoritmo.

5.2 Algoritmo de generación de nivel

El algoritmo de generación de nivel sigue una estructura bastante sencilla las salas se clasifican por sus aperturas en distintos arrays que determinan que tipo de salas necesitan para poder conectarse correctamente con toda la mazmorra. Este script contiene 4 arrays con la lista de habitaciones clasificadas, la closed room y una lista con todas las habitaciones que componen el nivel:



Figura 5.3: Estructura de los arrays que se usan para el algoritmo de generación

- *North Rooms*: Salas con aberturas al norte.
- *East Rooms*: Salas con aberturas al este.
- *South Rooms*: Salas con aberturas al sur.
- *West Rooms*: Salas con aberturas al oeste.
- *Closed Room*: Es una sala completamente opaca, no tiene aberturas y es una de las soluciones que planteé para el principal problema que encontré con la generación del que hablaremos más adelante.
- *Level*: Es una lista que incluye todas las salas generadas desde la SpawnRoom hasta el final de los distintos pasillos, almacena todas las salas por nombre y nos permite acceder a ellas para futuras acciones que tengamos que realizar sobre las mismas.

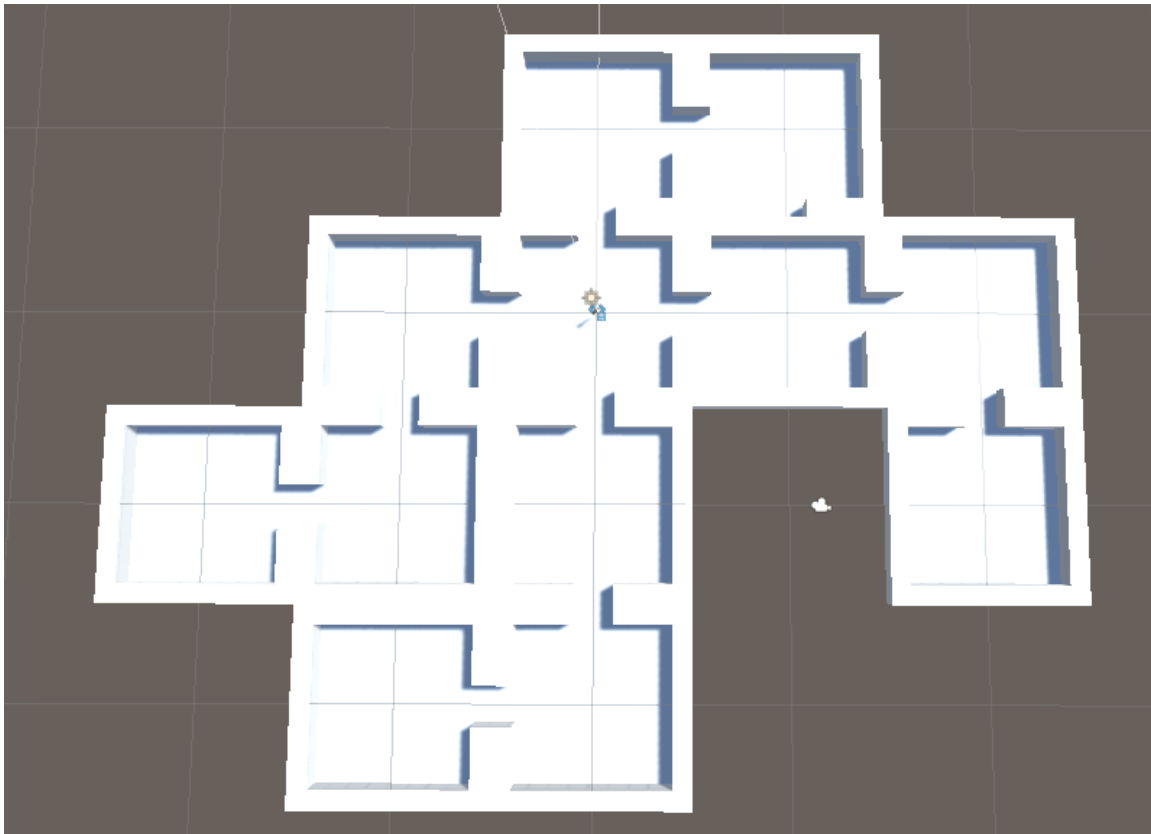


Figura 5.4: Ejemplo del nivel generado con el algoritmo

En la figura anterior podemos ver el nivel correspondiente a la lista *level* de la figura 5.3. En este caso se ha generado un nivel con 11 salas además de la sala inicial que siempre está presente. Podemos ver pasillos claros que llevan hasta distintas salas. En la lógica de la generación, veremos como hay distintos parámetros con los que podemos trabajar para cambiar la longitud del nivel entre otros.

5.3 Errores encontrados durante el desarrollo

5.3.1 Error de duplicación de sala

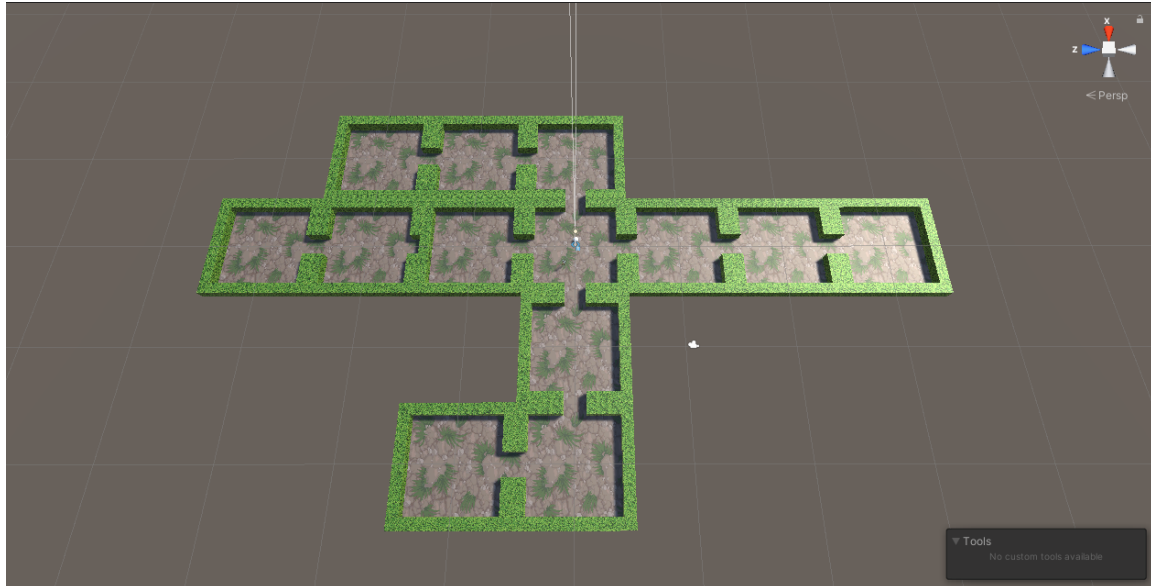


Figura 5.5: Error de generación duplicidad de sala

En el inicio del desarrollo del algoritmo, en ocasiones dos pasillos se superponían generando salas una encima de la otra ocasionando intersecciones que cortaban aberturas y creaban niveles con problemas de acceso a todos los pasillos.

Se solucionó creando un sistema de protección al añadir un spawnpoint al centro de todas las salas, que permitía comprobar si ya había una sala en esa posición y así evitar crear otra encima, terminando así la generación de esa ruta de la mazmorra.

5.3.2 *Error de ausencia de sala*

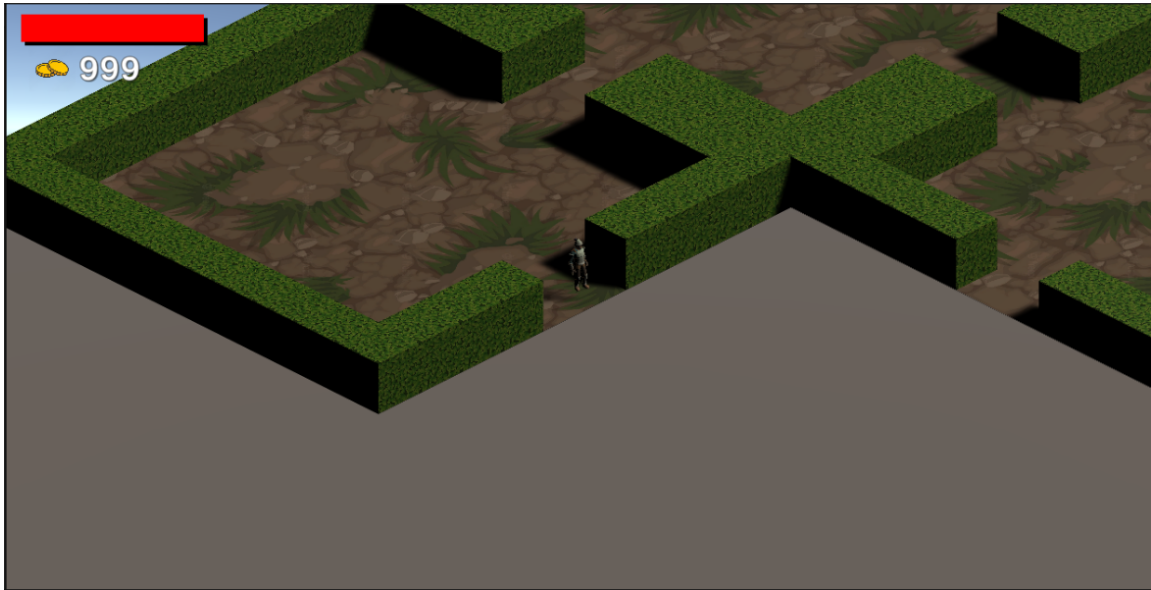


Figura 5.6: Error de generación ausencia de sala

Durante un punto del desarrollo del algoritmo, tras implementar la solución de la duplicación de salas esto empezó a causar otro error en el que en ocasiones la generación eliminaba salas que quedaban en las esquinas al estar conectadas a dos pasillos distinto.

Se solucionó mediante la revisión del código de eliminación de duplicidad y añadiendo un código salvaguarda que generaría una `closedRoom` en caso de que se ocasionase de nuevo una situación de este tipo.

5.4 Lógica de generación

```

5 public class RoomSpawner : MonoBehaviour
6 {
7     public int openSide;
8
9     private RoomTemplates templates;
10    private GameObject level;
11    public GameObject enemy;
12    private int rand, randEnemies;
13    private GameObject room;
14    private bool spawned = false;
15    private int size = 5;
16    private float spawnClearTimer = 0f;
17    private bool exit = true;
18    public GameObject stairs;
19    Message de Unity | 0 referencias
20    void Start()
21    {
22        //Destroy(gameObject, spawnClearTimer);
23        templates = GameObject.FindWithTag("Rooms").GetComponent<RoomTemplates>();
24        level = GameObject.FindWithTag("Level");
25        Invoke("Spawn", 0.1f);
26    }
27
28    0 referencias
29    void Spawn()
30    {
31        if (spawned == false)
32        {
33            if (openSide == 1)
34            {
35                if (size - templates.level.Count <= 0)
36                {
37                    room = Instantiate(templates.southRooms[0], transform.position, templates.southRooms[0].transform.rotation);
38                    room.transform.parent = level.transform;
39                    room.name = "DeadEnd";
40                    if (exit) Instantiate(stairs, transform.position, templates.southRooms[0].transform.rotation);
41                    exit = false;
42                    for (int i = 0; i < randEnemies; i++)
43                    {
44                        Instantiate(enemy, transform.position + new Vector3(Random.Range(1, 3), 0, Random.Range(1, 3)), templates.northRooms[0].transform.rotation);
45                    }
46                }
47                else
48                {
49                    rand = Random.Range(1, templates.southRooms.Length);
50                    room = Instantiate(templates.southRooms[rand], transform.position, templates.southRooms[rand].transform.rotation);
51                    room.transform.parent = level.transform;
52                    if (rand == 0) {
53                        room.name = "DeadEnd";
54                        if (exit) Instantiate(stairs, transform.position, templates.southRooms[rand].transform.rotation);
55                        exit = false;
56                    }
57                    for (int i = 0; i < randEnemies; i++)
58                    {
59                        Instantiate(enemy, transform.position + new Vector3(Random.Range(1, 3), 0, Random.Range(1, 3)), templates.northRooms[rand].transform.rotation);
60                    }
61                }
62            }
63        }
64    }

```

Figura 5.7: Código encargado de generar el layout del nivel

```

62    if (openSide == 2)
63    {
64        if (size - templates.level.Count <= 0)
65        {
66            room = Instantiate(templates.westRooms[0], transform.position, templates.westRooms[0].transform.rotation);
67            room.transform.parent = level.transform;
68            room.name = "DeadEnd";
69            if (exit) Instantiate(stairs, transform.position, templates.southRooms[0].transform.rotation);
70            exit = false;
71            for (int i = 0; i < randEnemies; i++)
72            {
73                Instantiate(enemy, transform.position + new Vector3(Random.Range(1, 3), 0, Random.Range(1, 3)), templates.northRooms[0].transform.rotation);
74            }
75        }
76        else
77        {
78            rand = Random.Range(1, templates.westRooms.Length);
79            room = Instantiate(templates.westRooms[rand], transform.position, templates.westRooms[rand].transform.rotation);
80            room.transform.parent = level.transform;
81            if (rand == 0) {
82                room.name = "DeadEnd";
83                if (exit) Instantiate(stairs, transform.position, templates.southRooms[rand].transform.rotation);
84                exit = false;
85            }
86            for (int i = 0; i < randEnemies; i++)
87            {
88                Instantiate(enemy, transform.position + new Vector3(Random.Range(1, 3), 0, Random.Range(1, 3)), templates.northRooms[rand].transform.rotation);
89            }
90        }
91    }
92
93    if (openSide == 3)
94    {
95        if (size - templates.level.Count <= 0)
96        {
97            room = Instantiate(templates.northRooms[0], transform.position, templates.northRooms[0].transform.rotation);
98            room.transform.parent = level.transform;
99            room.name = "DeadEnd";
100           if (exit) Instantiate(stairs, transform.position, templates.southRooms[0].transform.rotation);
101           exit = false;
102           for (int i = 0; i < randEnemies; i++)
103           {

```

Figura 5.8: Código encargado de generar el layout del nivel

```

184     Instantiate(enemy, transform.position + new Vector3(Random.Range(1, 3), 0, Random.Range(1, 3)), templates.northRooms[rand].transform.rotation);
185 }
186 }
187 else
188 {
189     randEnemies = Random.Range(1, templates.northRooms.Length);
190     rand = Random.Range(1, templates.northRooms.Length);
191     room = Instantiate(templates.northRooms[rand], transform.position, templates.northRooms[rand].transform.rotation);
192     room.transform.parent = level.transform;
193     if (rand == 0)
194     {
195         room.name = "DeadEnd";
196         if (exit) Instantiate(stairs, transform.position, templates.southRooms[rand].transform.rotation);
197         exit = false;
198     }
199     for (int i = 0; i < randEnemies; i++)
200     {
201         Instantiate(enemy, transform.position + new Vector3(Random.Range(1, 3), 0, Random.Range(1, 3)), templates.northRooms[rand].transform.rotation);
202     }
203 }
204 }
205 }
206 }
207 }
208 }
209 }
210 }
211 }
212 }
213 }
214 }
215 }
216 }
217 }
218 }
219 }
220 }
221 }
222 }
223 }
224 }
225 }
226 }
227 }
228 }
229 }
230 }
231 }
232 }
233 }
234 }
235 }
236 }
237 }
238 }
239 }
240 }
241 }
242 }
243 }
244 }
245 }
246 }
247 }
248 }
249 }
250 }
251 }
252 }
253 }
254 }
255 }
256 }
257 }
258 }
259 }
260 }
261 }
262 }
263 }
264 }
265 }
266 }
267 }
268 }
269 }
270 }
271 }
272 }
273 }
274 }
275 }
276 }
277 }
278 }
279 }
280 }
281 }
282 }
283 }
284 }
285 }
286 }
287 }
288 }
289 }
290 }
291 }
292 }
293 }
294 }
295 }
296 }
297 }
298 }
299 }
300 }
301 }
302 }
303 }
304 }
305 }
306 }
307 }
308 }
309 }
310 }
311 }
312 }
313 }
314 }
315 }
316 }
317 }
318 }
319 }
320 }
321 }
322 }
323 }
324 }
325 }
326 }
327 }
328 }
329 }
330 }
331 }
332 }
333 }
334 }
335 }
336 }
337 }
338 }
339 }
340 }
341 }
342 }
343 }
344 }
345 }
346 }
347 }
348 }
349 }
350 }
351 }
352 }
353 }
354 }
355 }
356 }
357 }
358 }
359 }
360 }
361 }
362 }
363 }
364 }
365 }
366 }
367 }
368 }
369 }
370 }
371 }
372 }
373 }
374 }
375 }
376 }
377 }
378 }
379 }
380 }
381 }
382 }
383 }
384 }
385 }
386 }
387 }
388 }
389 }
390 }
391 }
392 }
393 }
394 }
395 }
396 }
397 }
398 }
399 }
400 }
401 }
402 }
403 }
404 }
405 }
406 }
407 }
408 }
409 }
410 }
411 }
412 }
413 }
414 }
415 }
416 }
417 }
418 }
419 }
420 }
421 }
422 }
423 }
424 }
425 }
426 }
427 }
428 }
429 }
430 }
431 }
432 }
433 }
434 }
435 }
436 }
437 }
438 }
439 }
440 }
441 }
442 }
443 }
444 }
445 }
446 }
447 }
448 }
449 }
450 }
451 }
452 }
453 }
454 }
455 }
456 }
457 }
458 }
459 }
460 }
461 }
462 }
463 }
464 }
465 }
466 }
467 }
468 }
469 }
470 }
471 }
472 }
473 }
474 }
475 }
476 }
477 }
478 }
479 }
480 }
481 }
482 }
483 }
484 }
485 }
486 }
487 }
488 }
489 }
490 }
491 }
492 }
493 }
494 }
495 }
496 }
497 }
498 }
499 }
500 }
501 }
502 }
503 }
504 }
505 }
506 }
507 }
508 }
509 }
510 }
511 }
512 }
513 }
514 }
515 }
516 }
517 }
518 }
519 }
520 }
521 }
522 }
523 }
524 }
525 }
526 }
527 }
528 }
529 }
530 }
531 }
532 }
533 }
534 }
535 }
536 }
537 }
538 }
539 }
540 }
541 }
542 }
543 }
544 }
545 }
546 }
547 }
548 }
549 }
550 }
551 }
552 }
553 }
554 }
555 }
556 }
557 }
558 }
559 }
560 }
561 }
562 }
563 }
564 }
565 }
566 }
567 }
568 }
569 }
570 }
571 }
572 }
573 }
574 }
575 }
576 }
577 }
578 }
579 }
580 }
581 }
582 }
583 }
584 }
585 }
586 }
587 }
588 }
589 }
590 }
591 }
592 }
593 }
594 }
595 }
596 }
597 }
598 }
599 }
600 }
601 }
602 }
603 }
604 }
605 }
606 }
607 }
608 }
609 }
610 }
611 }
612 }
613 }
614 }
615 }
616 }
617 }
618 }
619 }
620 }
621 }
622 }
623 }
624 }
625 }
626 }
627 }
628 }
629 }
630 }
631 }
632 }
633 }
634 }
635 }
636 }
637 }
638 }
639 }
640 }
641 }
642 }
643 }
644 }
645 }
646 }
647 }
648 }
649 }
650 }
651 }
652 }
653 }
654 }
655 }
656 }
657 }
658 }
659 }
660 }
661 }
662 }
663 }
664 }
665 }
666 }
667 }
668 }
669 }
670 }
671 }
672 }
673 }
674 }
675 }
676 }
677 }
678 }
679 }
680 }
681 }
682 }
683 }
684 }
685 }
686 }
687 }
688 }
689 }
690 }
691 }
692 }
693 }
694 }
695 }
696 }
697 }
698 }
699 }
700 }
701 }
702 }
703 }
704 }
705 }
706 }
707 }
708 }
709 }
710 }
711 }
712 }
713 }
714 }
715 }
716 }
717 }
718 }
719 }
720 }
721 }
722 }
723 }
724 }
725 }
726 }
727 }
728 }
729 }
730 }
731 }
732 }
733 }
734 }
735 }
736 }
737 }
738 }
739 }
740 }
741 }
742 }
743 }
744 }
745 }
746 }
747 }
748 }
749 }
750 }
751 }
752 }
753 }
754 }
755 }
756 }
757 }
758 }
759 }
760 }
761 }
762 }
763 }
764 }
765 }
766 }
767 }
768 }
769 }
770 }
771 }
772 }
773 }
774 }
775 }
776 }
777 }
778 }
779 }
780 }
781 }
782 }
783 }
784 }
785 }
786 }
787 }
788 }
789 }
790 }
791 }
792 }
793 }
794 }
795 }
796 }
797 }
798 }
799 }
800 }
801 }
802 }
803 }
804 }
805 }
806 }
807 }
808 }
809 }
810 }
811 }
812 }
813 }
814 }
815 }
816 }
817 }
818 }
819 }
820 }
821 }
822 }
823 }
824 }
825 }
826 }
827 }
828 }
829 }
830 }
831 }
832 }
833 }
834 }
835 }
836 }
837 }
838 }
839 }
840 }
841 }
842 }
843 }
844 }
845 }
846 }
847 }
848 }
849 }
850 }
851 }
852 }
853 }
854 }
855 }
856 }
857 }
858 }
859 }
860 }
861 }
862 }
863 }
864 }
865 }
866 }
867 }
868 }
869 }
870 }
871 }
872 }
873 }
874 }
875 }
876 }
877 }
878 }
879 }
880 }
881 }
882 }
883 }
884 }
885 }
886 }
887 }
888 }
889 }
890 }
891 }
892 }
893 }
894 }
895 }
896 }
897 }
898 }
899 }
900 }
901 }
902 }
903 }
904 }
905 }
906 }
907 }
908 }
909 }
910 }
911 }
912 }
913 }
914 }
915 }
916 }
917 }
918 }
919 }
920 }
921 }
922 }
923 }
924 }
925 }
926 }
927 }
928 }
929 }
930 }
931 }
932 }
933 }
934 }
935 }
936 }
937 }
938 }
939 }
940 }
941 }
942 }
943 }
944 }
945 }
946 }
947 }
948 }
949 }
950 }
951 }
952 }
953 }
954 }
955 }
956 }
957 }
958 }
959 }
960 }
961 }
962 }
963 }
964 }
965 }
966 }
967 }
968 }
969 }
970 }
971 }
972 }
973 }
974 }
975 }
976 }
977 }
978 }
979 }
980 }
981 }
982 }
983 }
984 }
985 }
986 }
987 }
988 }
989 }
990 }
991 }
992 }
993 }
994 }
995 }
996 }
997 }
998 }
999 }
1000 }

```

Figura 5.9: Código encargado de generar el layout del nivel

En cuanto a la lógica de generación se realiza un llamado de la instrucción `Spawn()` por cada sala que se genera y se utiliza un bool `spawned` para determinar si las salas contiguas a la sala en cuestión se han generado. Si no se han generado comprueba que tipo de aperturas tiene la sala y selecciona aleatoriamente una de las salas que permita seguir construyendo el nivel.

Las salas elegidas dependen específicamente de las aperturas y de la propiedad `size` determinada al inicio del código, esta propiedad determina el tamaño del nivel y se utiliza para limitar la cantidad de salas que se spawnen.

Además este mismo código comprueba si entra en contacto con otro trigger, `OnTriggerEnter`, para solucionar el problema de duplicidad de salas como cometamos más arriba

en la figura 5.5, en caso de generarse una sala sobre otra se elimina y para solucionar el caso de que sea una esquina o un punto ajeno elimine la sala y se quede una apertura en el nivel se utiliza el objeto `closedRoom` para instanciarlo en estos casos, eliminando el problema en la figura 5.6

Además se realiza un renombrado de salas sin otras salidas y que acaban los pasillos como `DeadEnd`, con la intención de utilizar estas salas para futuras implementaciones en las que se generan la escalera, futura tienda y habitación del tesoro, pero ya se habla de esto en futuras implementaciones.

El primer pasillo marcado como `DeadEnd`, utilizará el booleano `exit` para instanciar unas escaleras que nos permitirán llegar hasta el siguiente nivel y continuar en la mazmorra. Acto seguido se desactivará el booleano y continuará con la construcción del nivel.

Por último se elige aleatoriamente una cifra entre 1 y 3 y se instancian enemigos en cada sala al acabar en una posición randomizada en el interior de los muros de la misma dando lugar a un nivel generado.

6 Música y efectos de sonido

6.1 Música

Para la música como hablamos en el 2.4.5 se ha utilizado SoundTrap una herramienta muy potente que nos permite infinidad de posibilidades. Debido a mi desconocimiento musical el resultado ha sido un poco catastrófico en parte así que al final he optado por dejar la pieza que cree utilizando este método para el menú mientras que la del nivel es una pieza de uso libre obtenida de Curtis, [s.f.](#)

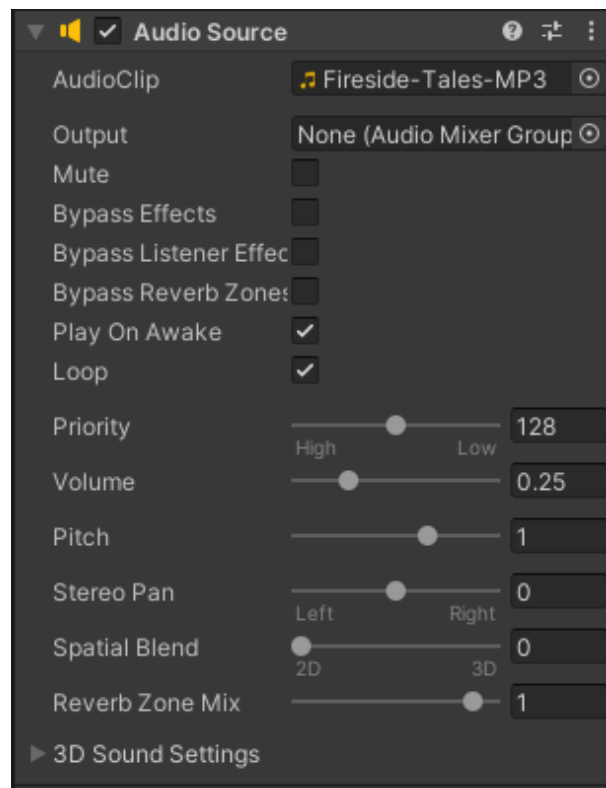


Figura 6.1: Configuración audio source background music

Para utilizar básicamente creamos un `emptyObject` que contenga un `audioSource` y asignamos los parámetros como podemos ver en la figura anterior. Para que se ejecute al empezar el nivel es importante marcar la opción de `Play On Awake` y marcar `Loop` para que siga reproduciéndose mientras nos encontremos en la escena.

6.2 Efectos de sonido

Para los efectos de sonido se utilizó la web [SFDX.me](https://sfdx.me), que facilita una interfaz muy intuitiva para poder editar tus sonidos en un estilo 8bit muy atractivo y que sienta bastante bien para según que efectos como la recolección de monedas.

En el script de la figura 3.5 se puede apreciar como se utiliza el objeto `audio source` nativo de unity para incluir dichos sonidos y cómo se ejecutan mediante código utilizando el comando `audioSource.Play()` de tal forma que se ejecute el clip proporcionado al `audioSource`.

7 Trabajo Futuro

Si bien se han asentado las bases de lo que define una experiencia roguelike y un uso de los sistemas para el desarrollo de un juego completo la experiencia actual dista bastante de un producto completamente pulido y puede implementar un montón de mejoras. Muchas de estas no se han podido implementar para la realización del trabajo de final de grado debido a que se ha tratado de sentar unas bases estables para que la ampliación del juego sea lo más cómoda posible y fue en lo que se centro esta primera parte del desarrollo y estudio que se incluye en el proyecto.

No obstante en la planificación del proyecto tenemos una lista de las mejoras que me gustaría implementar en un futuro mientras sigo trabajando en el proyecto para asegurar un producto más pulido y completo:

- *Generación de niveles.* Si bien la generación del nivel funciona increíblemente, con las salas actuales y la lógica actual todas las salas son de 1x1, en un futuro me gustaría añadir las salas diseñadas de 2x1, 2x2, etc. Permitiendo una mayor diversidad y haciendo que los niveles se sientan más diversos.
- *Talentos.* El sistema de talentos fue una de las últimas implementaciones y por tanto se nota que no he tenido todo el tiempo del mundo que me gustaría, este sistema me gustaría que fuese integrado a al nivel de la ciudad del que se habla más abajo, pero con la cantidad de tiempo que he tenido entre unas cosas y otras no me ha dado tiempo a hacerlo todo lo intrincado que me gustaría. En su lugar se ha visto sustituido por un sistema simple que te da unos aumentos planos de vida/daño, pero que sientan la base necesaria para un sistema más complejo.
- *Clases.* En principio la idea es añadir un sistema de clases y talentos más profundo que haga que cada nueva partida sea incluso más distinta, pero siendo realistas me iba a ser imposible añadir todas las que me hubiese gustado así que habiéndome decantado por un guerrero y un arquero. Se quedaron fuera del planning inicial el mago y el pícaro, que son quizá otras de las clases más icónicas del género. Además de estas me gustaría incluir otras que se irían desbloqueando conforme avanzases en el juego.

- *Ciudad*. Un punto de control en el que gastar tus monedas y decidir quién va a ser el próximo aventurero que te lleve al éxito. La idea era crear una especie de lobby entre partidas que te permitiese seleccionar todas las opciones de configuración para tu aventurero, comprar los talentos y gestionar el inventario con tal de darle a todo un tono todavía más profesional y acabado.
- *Inventario*. Al principio del desarrollo y en la lista esta una especie de inventario que nos permita obtener objetos que mandar de vuelta una vez acabe nuestra aventura, si bien se estudio no me veía en condiciones de implementar todo el sistema de forma que quedase algo completo y correcto así que lo aparté y dejé para implementarlo en el futuro junto a la ciudad.
- *Cooperativo*. Al inicio del desarrollo y como se comentaba en el planteamiento itere entre varias opciones y versiones para el juego, entre ellas un cooperativo local que me trajo más quebraderos de cabeza que otra cosa debido a como gestionar la cámara y según que sistemas. Si tuviese que aproximarle diría, que una vez abandoné la idea del cooperativo estaba hecho entorno a un tercio del sistema completo, pero al ir añadiendo nueva lógica entorno a los personajes y enemigos supe que no iba a dar tiempo a realizar algo funcional y lo aparté para dedicarle tiempo en específico en un futuro.
- *Enemigos*. Si bien se ha añadido la lógica más básica de los enemigos y las interacciones necesarias para que estos puedan funcionar como tal, dañar al protagonista, perseguirle, etc. La variedad de los mismos y la dificultad no es especialmente ajustable en su estado actual por tanto me gustaría experimentar más con otros tipos de enemigos y situaciones.
- *Efectos de sonido*. Todos los paquetes de sonido referentes a como funcionan según que interacciones todavía no se han añadido y se han quedado por implementar, pero está estudiado como hacerlo y no se ha hecho por falta de tiempo.
- *Música*. A pesar de que se ha realizado alguna pieza los conocimientos musicales además de la poca práctica que se tiene con el programa en cuestión no han permitido quizá profundizar de la forma que me gustaría en este apartado y es una de las cosas que me gustaría mejorar mucho para la versión final, bien sea tratando de aprender más sobre este tema en particular o en su lugar buscando alguien que conozca más del tema.

8 Conclusiones

Como supuse al ver todo lo que quería hacer durante el desarrollo el tiempo dedicado al proyecto no ha sido suficiente para cumplir con todo lo que buscaba del desarrollo de un videojuego al completo. Sin embargo estoy muy satisfecho con el trabajo realizado, ya que, me ha permitido aprender infinidad de cosas relacionadas con el sector, el desarrollo de los videojuegos y más específicamente del funcionamiento de Unity.

Es cierto que muchos puntos que quería en un principio al plantear el proyecto se han quedado para futuras implementaciones y muchos otros necesitan un nivel de profundización mayor antes de poder determinarlos como completos. No obstante, el proyecto aquí expuesto representa una primera versión que sin duda engloba, si bien no todo lo que esperaba poder conseguir con estos meses de trabajo, un proyecto lleno de pasión y que sin duda me ha permitido practicar muchas de las cosas que he aprendido en la carrera. Desde la gestión del tiempo, planificación y planteamiento de un proyecto de este estilo por mi cuenta, y todo lo que eso ha conllevado. Hasta el estudio, razonamiento y ejecución de soluciones en código que me permitiesen implementar las ideas que he ido teniendo a lo largo de estos meses.

A pesar de todo esto he disfrutado y a la vez sufrido durante el desarrollo del proyecto, la cantidad de inspiración requerida a ratos para estar satisfecho con el trabajo realizado ha dificultado en diversas ocasiones el planteamiento de muchas de las soluciones. Hasta el punto de llegar a eliminar partes del proyecto por completo y plantearlas de nuevo, bien puesto que la base no era la correcta por la falta de experiencia en el entorno o porque simplemente el resultado obtenido no se correspondía a lo que mi visión del mismo contemplaba.

Además de todo lo relacionado con Unity, he disfrutado mucho de poder realizar un estudio más en profundidad de como funciona un mercado como es el de los videojuegos y cómo se debería plantear todo el lanzamiento del juego una vez esté completado, como pudimos ver durante la introducción de este mismo trabajo.

Si pudiese decirme algo a mí mismo hace unos meses sería que intentase plantear más fuertemente fechas y objetivos de forma semanal con tal de que el desarrollo fuese más fluido y fácil. Pensar más las cosas con vistas a largo plazo para no tener que volver tantas

veces sobre mis pasos a arreglar pequeños detalles que se quedaban aquí y allá y que han dificultado en ocasiones algunas implementaciones que dependían de otros sistemas y sobre todo informarme mucho sobre el tema en cuestión antes de intentar hacer las cosas por mi propio pie. Pues al principio trataba bastante de realizar algunos desarrollos que ya han hecho decenas de personas y que podía tomar como ejemplo para realizar mi implementación de un sistema parecido.

Teniendo en cuenta todo esto considero toda la experiencia del TFG una mezcla entre un éxito rotundo y mucha frustración que se experimenta al trabajar en un entorno en el que eres una única persona y además algo inexperimentada, pero que trata de obtener unos resultados que le permitan estar satisfecho consigo mismo. No obstante he disfrutado mucho de la experiencia y me llevo un montón de ideas nuevas tanto a la hora de gestionar un proyecto de este calibre, como a la hora de defenderlo y exponer cual ha sido el trabajo llevado a cabo. Un hecho que sin duda creo que me va a ayudar a poder gestionar mejor proyectos de ahora en adelante en el mundo laboral y a la hora de ponerme en contacto con clientes o compañeros de trabajo para exponerles mis ideas.

Bibliografía

- Arias, J. (04/06/2023). La industria de los videojuegos ya genera más ingresos que la música y el cine juntos [<https://theobjective.com/economia/2023-06-04/videojuegos-ingresos-musica-cine/>]. (Vid. pág. 4).
- BindingOfIsaacRebirth. (s.f.). Binding Of Isaac Level Generation [https://bindingofisaacrebirth.fandom.com/wiki/Level_Generation]. (Vid. pág. 29).
- Blink. (s.f.). free-low-poly-human-rpg-character [<https://assetstore.unity.com/packages/3d/characters/humanoids/fantasy/free-low-poly-human-rpg-character-219979>]. (Vid. pág. 21).
- Clement, J. (24/01/2023a). Revenue video game worldwide [<https://www.statista.com/statistics/1344668/revenue-video-game-worldwide/>]. (Vid. pág. 2).
- Clement, J. (24/01/2023b). Video game industry - Statistics and Facts. *www.statista.com*.
- Curtis, D. (s.f.). Fireside-Tales-MP3 [<https://www.chosic.com/download-audio/28506/>]. (Vid. pág. 38).
- Dávila, D. P. (s.f.). How to play Sounds in unity [<https://levelup.gitconnected.com/how-to-play-sound-effects-in-unity-6a122bb32970>].
- Divers, G. (24/01/2023). Gaming industry dominates as the highest grossing entertainment industry. *www.gamerhub.co.uk*.
- Epitome. (s.f.). Learn Unity Engine by creating a real top down RPG [FULL COURSE][Unity Tutorial] [https://www.youtube.com/watch?v=b8YUfee_pzc].
- iHeartGameDev. (s.f.). How to animate character in Unity 3d [https://www.youtube.com/watch?v=-FhvQDqmgmU&list=PLwyUzJb_FNeTQwyGujWRLqnfKpV-cjeO].

- Iskillitoon. (s.f.). Stylized Skeleton Warrior Mobile Game Ready Model Free low-poly 3D model [<https://www.cgtrader.com/free-3d-models/character/fantasy-character/iskillitoon-stylized-skeleton-warrior-mobile-game-ready-model>].
- Kaupenjoe. (s.f.). Unity BEGINNER Tutorialsl [https://www.youtube.com/playlist?list=PLKGarocXCE1H7pJk6k_CSWS359mtt3MVI].
- Orús, A. (22/06/2021). Evolución anual de los ingresos de King.com a nivel mundial desde 2010 hasta 2020 [<https://es.statista.com/estadisticas/636271/ingresos-anuales-de-king-a-nivel-mundial/>].
- Orús, A. (22/09/2022). Ingresos totales generados por Twitch a nivel mundial entre 2016 y 2021 [<https://es.statista.com/estadisticas/1317850/ingresos-de-twitch-a-nivel-mundial/>].
- Unity. (s.f.). 2D Roguelike Unity Tutorial] [<https://www.youtube.com/watch?v=Fdcnt2-Jf4w>].
- Wikipedia. (s.f.). Roguelike [<https://en.wikipedia.org/wiki/Roguelike>]. (Vid. pág. 5).