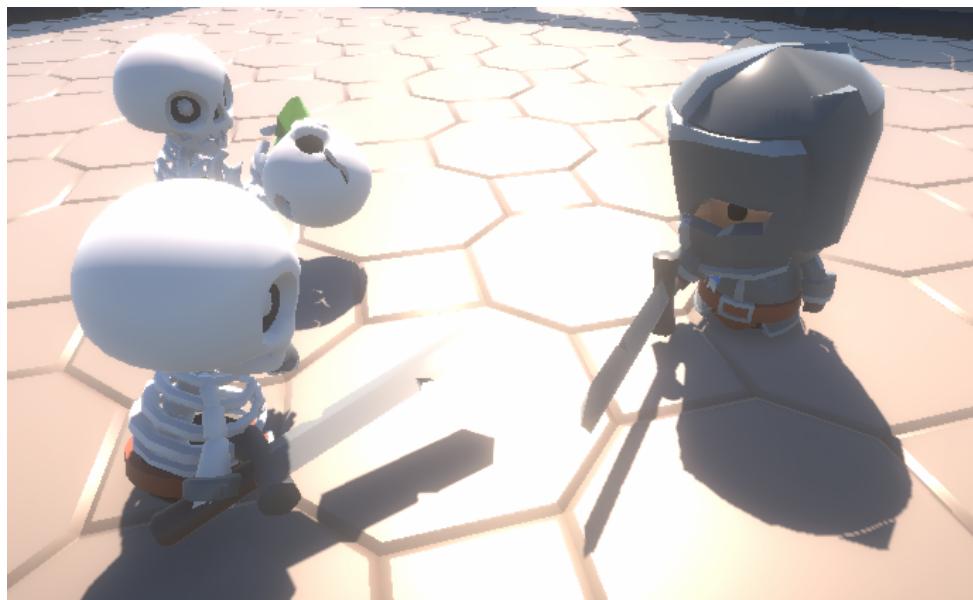


Roguelike Chibi

Carlos Requena Doña



Contents

1	Introducción	3
2	Concepto del juego	4
3	Diagramas del Proyecto	5
4	Diseño de niveles	7
5	Diseño de personajes y enemigos	9
5.1	Diseño y lógica del personaje a controlar	9
5.1.1	Movimiento del personaje	9
5.1.2	Detección de colisiones	9
5.1.3	Interacción del personaje	9
5.2	Diseño y lógica del enemigo	9
5.2.1	IA (Inteligencia Artificial)	9
5.2.2	Comportamiento en partida	10
5.3	Animaciones de personajes	10
6	Elementos interactuables	12
6.1	Estructura de los elementos interactuables	12
6.1.1	Clase objetos	12
6.1.2	Clase Arma	12
6.1.3	Clase Poción	12
6.1.4	Clase Puerta	12
6.1.5	Clase Llave	12
6.1.6	Clase Contenedor	12
7	Diseño de la interfaz de usuario (UI)	13
7.1	Vida del jugador	13
7.2	Menú de pausa	13
7.2.1	Pantalla del menú	13
7.2.2	Opciones	14
7.3	Menú principal del juego	14
7.3.1	Pantalla inicial del juego	14
7.3.2	Selección de personaje	14
8	Implementación en Unity	16
8.1	Pool de Objetos	17
8.2	Gestor de la partida	18
8.3	Gestor de sonido	18
8.4	Gestor de música	18
8.5	Creador de la Dungeon	18
8.6	Gestor de feedback	18
9	Referencias	19

1 Introducción

El proyecto presenta el desarrollo de un videojuego tipo Roguelike de exploración de una dungeon generada mediante procedimientos o como se dice en el mercado, proceduralmente. El juego es desarrollado en **Unity** en base a unos paquetes de assets que presentan distintos modelos 3D para el juego y poder facilitar la labor de diseño. El juego utiliza los Assets visuales de kay Lousberg[3].



Se busca una implementación propia en todos los aspectos sin utilizar assets que proporcionen código ya hecho o mecánicas ya programadas. La única excepción es el Asset que se presenta en el apartado de Diseño de nivel dada la complejidad de lo que logra.

El juego presenta los siguientes requisitos funcionales:

- Ajuste de opciones del juego. Cambio de controles.
- Detección de Gamepad.
- Generación aleatoria de entorno.
- Comportamiento indeterminado de los elementos del juego.
- Implementación de sistema de dificultad
- Implementación de sistema de sonido y música

2 Concepto del juego

Roguelike es un género que se ha popularizado enormemente no hace mucho tiempo gracias a estudios independientes que desarrollaron juegos tales como The Binding of Isaac, Enter the gungeon y muchos más. Aunque su procedencia es mucho más lejana en el tiempo su popularidad explota con dichos lanzamientos que a día de hoy son un culto de los videojuegos actuales. El género explota la aleatoriedad de varios elementos concretos tales como el botín del juego, la aparición de enemigos y demás elementos tales como el propio escenario del juego.

El juego que se desarrolla en este documento se basa en dicho género, cuyas mecánicas principales son la exploración de la mazmorra principal obteniendo botín o también dicho 'looteando'[2] de modo que se progresá por esta hasta llegar al final de la misma. El motivo detrás de la exploración de la proceduralmente generada mazmorra reside en aquellos factores aleatorios que se mencionan en el primer párrafo además de misiones y progresión del personaje.

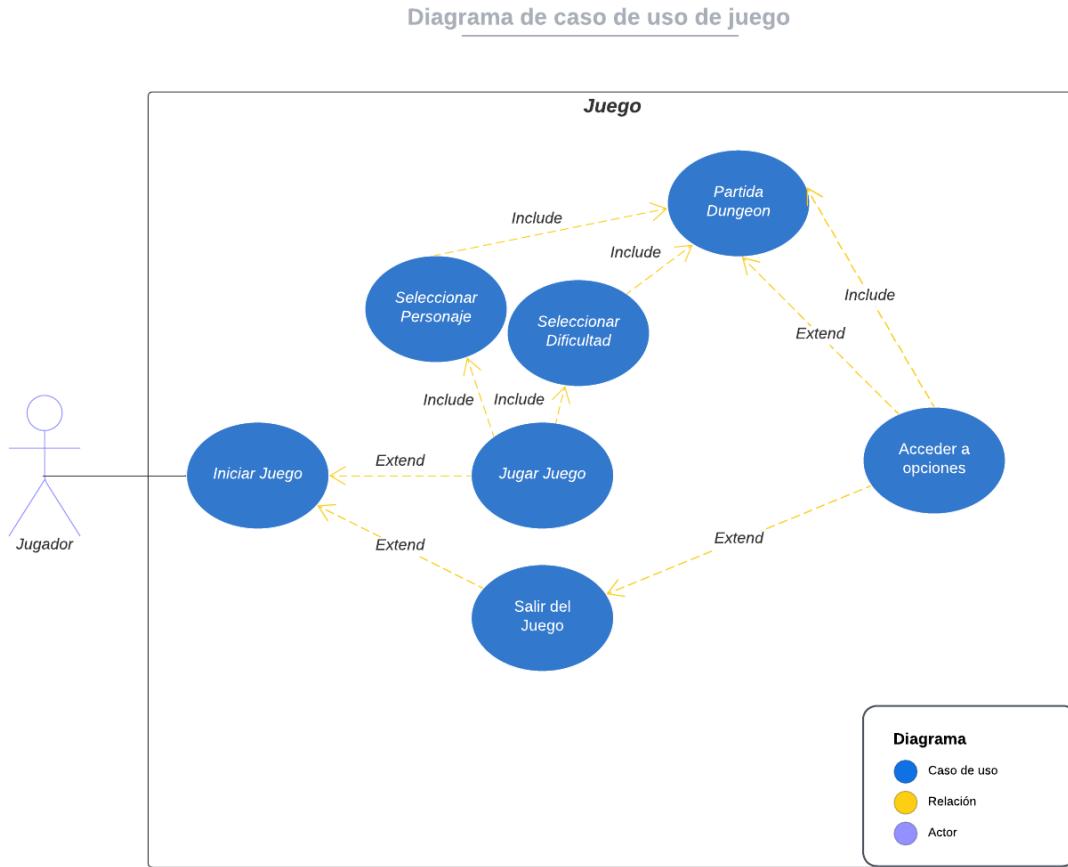
El juego presenta una ambientación medieval con un toque Chibi que proporciona una estética agradable de ver y bonita. La mazmorra está completamente estilizada gracias a ciertos Assets.



Figure 1: Imagen del pack de Assets utilizado en el proyecto

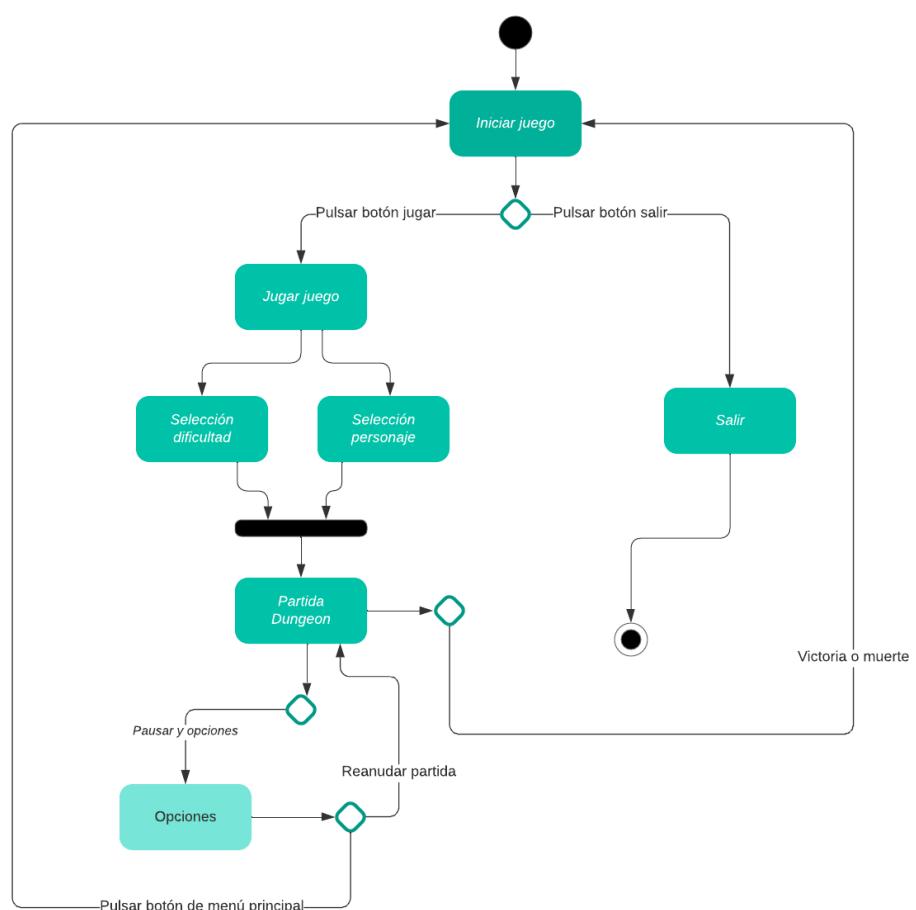
3 Diagramas del Proyecto

El proyecto tiene diagramas asociados al juego. En primer lugar tenemos el diagrama de casos de uso.



Por otro lado, tenemos el diagrama de actividades que representa el flujo normal del software en su uso.

Diagrama de actividades Proyecto



4 Diseño de niveles

El diseño de nivel es el único aspecto del juego a nivel lógica que realmente no está creado explícitamente por nosotros. Utilizamos un Asset llamado "Dungeon Architect"[1]. Este asset es sencillo de usar y en dos días sabes lo suficiente para empezar a incorporar y diseñar tus propias creaciones.

Los niveles se crean de forma procedural siguiendo un Grafo como modelo base para la generación. Básicamente se utiliza el grafo para determinar conexiones entre ramas, habitaciones y poder determinar relaciones tales como puerta-llave. Obviamente la lógica se debe implementar aparte, pero la generación del escenario es completamente tarea del Asset.

Para crear una Dungeon por ejemplo, utilizamos el Grid Flow Editor que nos permite definir relaciones y conexiones en el mapa. El flujo de ejecución traza un recorrido para generar el grafo de forma aleatoria siguiendo ciertas directrices. Este flujo de ejecución pasa por varias etapas en las que por ejemplo se comienza con las ramas principales de la Dungeon, se continúa con distintos objetos... Finalmente se creará un TileMap que es la representación del mapa y con un algoritmo de ruido se pueden generar elementos sobre el mapa como por ejemplo rocas.

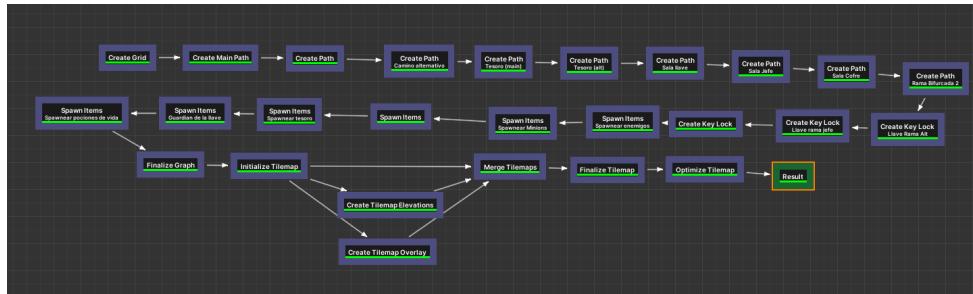


Figure 2: Flujo de ejecución de la Dungeon en dificultad normal

Con dicho flujo de ejecución se creará un grafo junto a su representación.

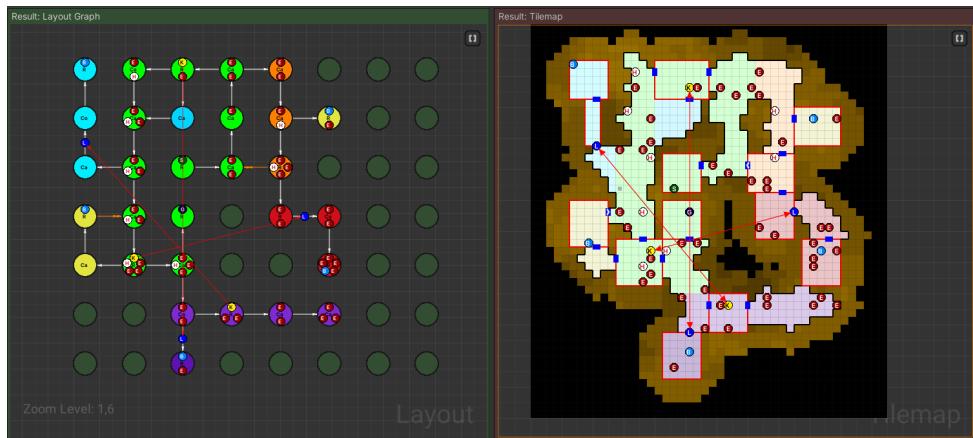


Figure 3: Grafo y representación de la Dungeon en dificultad normal

Para trabajar correctamente se crea un Tema de la Dungeon que nos permite determinar los Prefabs a utilizar y colocar en el proceso de creación. El tema recorre de izquierda a derecha los elementos de un atributo por probabilidad para determinar que Prefab se utiliza. Es decir, si tenemos "Pared" podemos colocar distintos tipos de pared en el mismo con distinto porcentaje

para dar cierta variedad a la mazmorra. También puedes hacer que en una zona aparezcan varios objetos poniendo probabilidad 1 como por ejemplo, el Caballero con su espada dado que queremos a ambos en la zona de aparición de la Dungeon.

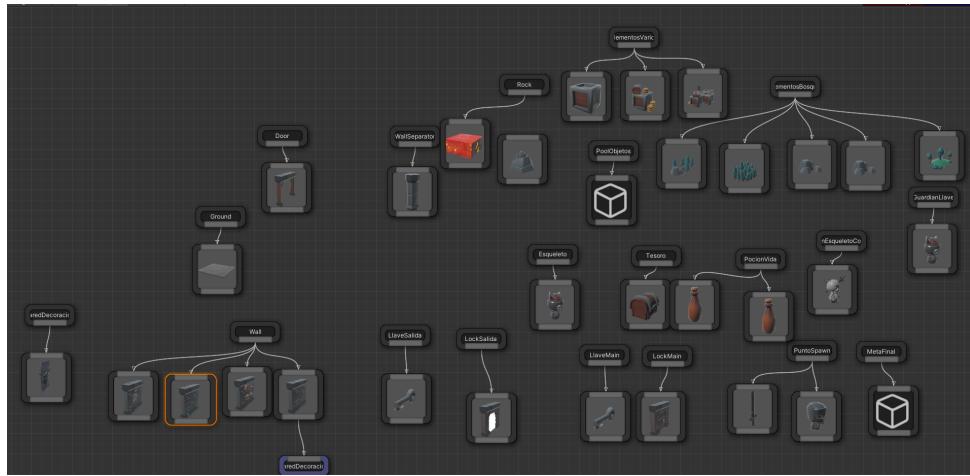


Figure 4: Tema de la Dungeon en modo difícil

En nuestras dungeons seguimos un patrón a la hora de generarlas. Dividimos el recorrido en tres ramas las cuales tienen puertas cerradas que son desbloqueadas con llave de modo que una de las ramas es la de salida y las otras dos contienen un cofre bloqueado. La rama salida presenta una puerta muy brillante y las demás son simples puertas. La llave para acceder a la salida aparecerá en una de las ramas con sala bloqueada con llave.

La rama de salida representa la salida y final de la partida. Las ramas contienen enemigos y algunas pociones de vida.

5 Diseño de personajes y enemigos

5.1 Diseño y lógica del personaje a controlar

5.1.1 Movimiento del personaje

El personaje se controla con las teclas WASD en teclado y el Joystick si se juega con Gamepad. El personaje está pensado para no tener verticalidad a la hora de manejarse, podríamos decir que solo se puede mover horizontalmente, sin embargo, a la hora de subir escaleras por ejemplo este se adaptará a estas aunque para el jugador el movimiento no presenta verticalidad ninguna.

El personaje solo modifica su transformada a fin de cuentas, por esto mismo se incorpora una interpolación que se realiza de su transformada que representa la dirección del objeto. Esta será modificada en función del Vector2D que representa el movimiento. Cabe destacar que el movimiento va en una clase distinta mientras que la colisión y demás es parte del script del jugador.

5.1.2 Detección de colisiones

Para detectar colisiones se utiliza un "CapsuleCast", es decir, trazamos una esfera que detecta colisiones. Esta capsula detiene el movimiento al interaccionar con un Collider dado que el Vector2D que determina la dirección de movimiento es 0 por lo que el movimiento no es efectivo. Se comprueba si se puede mover únicamente en la X o la Z tras una comprobación inicial con la dirección del Input. Tras verificar que ningún movimiento es posible no se mueve el personaje aunque se presione el botón de movimiento evitando atravesar objetos. Esta detección de colisiones se reutilizará para detectar objetos y cofres que interactúen con el jugador.

Entrando en mayor detalle el Vector2D que representa el movimiento se normaliza para ofrecer movimiento diagonal a la misma velocidad.

5.1.3 Interacción del personaje

La interacción del personaje es simple. El usuario emite un Raycast trazando una línea recta hacia delante detectando objetos y guardando su referencia para poder interactuar en caso de presionar el botón de interacción en el juego. Al interactuar, cualquier objeto o elemento que implemente la interfaz procederá con su tarea, ya sea abrirse una puerta como recoger un objeto.

El jugador puede recibir vida tras consumir pociones. Esto se gestiona desde un Gestor de Buffs que tiene el jugador para reproducir las animaciones y partículas pertinentes.

5.2 Diseño y lógica del enemigo

5.2.1 IA (Inteligencia Artificial)

El enemigo traza su dirección en base a su IA de modo que esta reconoce su entorno de manera dinámica.

La inteligencia artificial se basa en un array de pesos y un array de direcciones definidos en torno a un círculo. Es decir, para el enemigo, el mundo se representa mediante un círculo dividido en direcciones. Cada dirección representa un peso en el array de pesos cuyos valores determinan que dirección se debe tomar. A mayor interés, mayor será el peso de una dirección.

Dada esta estructura de la inteligencia artificial, las decisiones del mejor camino dependen del entorno y la posición del jugador. En primer lugar, el peso va escalando y modificándose en base

al producto escalar entre la dirección deseada que es hacia la que está el jugador en la mayoría de los casos y la dirección del array que está siendo considerada. Una vez se tiene, procedemos con el ajuste de los pesos en función de distintos elementos tales como la distancia hacia el jugador y el mantenimiento de una distancia de seguridad con el jugador.

La ventaja principal de la IA que uso es su adaptación al entorno con su mapa de peligros e intereses los cuales usa para enmascarar los valores dados en los pesos de las direcciones.

Esta inteligencia artificial se basa en "Context steering". Presentada en un libro sobre IA en videojuegos por expertos del área.

5.2.2 Comportamiento en partida

El comportamiento es bastante simple. Sin embargo, existen dos tipos de enemigos. Tenemos los Minions que persiguen al enemigo a bocajarro y los Guerreros que mantendrán distancia y te rodearán siendo más molestos puesto que debes acercarte tú a ellos.



Figure 5: Enemigos persiguiendo al jugador

Una vez estén a cierta distancia del Jugador atacarán y tendrán un tiempo de refresco hasta poder volver a atacar. Ciertamente diseñado para aprovecharlo. Los Minions no incorporan prácticamente nada del Context Steering de modo que siguen al jugador y nada más.

Los guerreros sí aprovechan su vector de pesos para mantener distancia con el jugador y poder rodearle en movimientos laterales.

Los enemigos obtendrán armas aleatorias de la "Pool de objetos" que se explica más adelante. En resumidas cuentas es una manera de ahorrar memoria por instancia de objetos.

5.3 Animaciones de personajes

Tanto el jugador como el enemigo realizan animaciones durante su partida. Se gestionan en base a una interfaz "IAnimatorUsaObjeto" que determina no solo los movimientos, sino el uso de objetos de ambos. Basicamente recibe un evento del arma o el jugador para notificar el inicio de ataque o uso de objeto para realizar las animaciones pertinentes. Este objeto tiene referencia al componente Animator de Unity de modo que controla y sabe en cada momento que ocurre y

que estado tiene la animación del personaje. Gracias a los booleanos que indican si se mueven o están quietos podrán reproducir las animaciones pertinentes. Por ejemplo, el enemigo también tiene un booleano para su muerte.

6 Elementos interactuables

6.1 Estructura de los elementos interactuables

Los elementos interactuables se estructuran por interfaz. La interfaz de ElementoInteractuable se encarga de proporcionar los métodos necesarios para poder interactuar con estos. Sin embargo, esto vale para puertas, cofres, objetos...

Para entender su funcionamiento es necesario identificar los distintos tipos de objeto/elementos interactuables disponibles para el jugador.

6.1.1 Clase objetos

Con objetos se refiere a cualquier elemento interactuável cuyo valor reside en ser utilizado como herramienta para el jugador, es decir, que lo use cogiendo dicho elemento tal como cogerías una antorcha o cualquier otro objeto.

Los objetos definen una jerarquía basada en la herencia.

Por otro lado, tienen un script que permite que brillen de color dorado si el jugador apunta al objeto. Esto es un feedback para el usuario.

6.1.2 Clase Arma

La clase Arma hereda directamente de la clase Objetos de modo que representa todas las armas del juego. Entre sus propiedades más importantes presenta un Vector dinámico de ObjetosColisionados de modo que se asegura de no aplicar el efecto de daño dos veces a un mismo elemento en una animación de ataque.

Esta clase tiene definidos un método para detectar al jugador y al enemigo dado que toda la lógica de ataque la gestiona la propia arma. Basicamente busca una Hitbox de un IRecibeDaño que es una interfaz definida para recibir daño.

6.1.3 Clase Poción

La clase Poción guarda la variable "buff" que es un scriptable object que guarda toda la información de la poción. Basicamente define su propia interacción.

6.1.4 Clase Puerta

Esta clase define una puerta que está bloqueada y requiere de una llave que el jugador debe recoger. Existen dos tipos de puerta, que son la puerta de salida y la puerta normal. La puerta de salida sirve como acceso a la sala de salida de la Dungeon. La puerta normal simplemente sirve para abrir puertas básicas que guardan cofres.

6.1.5 Clase Llave

Clase cuya razón de existir es servir como objeto que permite el acceso a puertas. Esto lo gestiona el jugador que almacena una lista de llaves obtenidas.

6.1.6 Clase Contenedor

Esta clase sirve para los cofres y su animación. Representa a aquel elemento con el que se debe interactuar para acceder a un objeto que ofrece. Esta clase se usa en los cofres.

7 Diseño de la interfaz de usuario (UI)

7.1 Vida del jugador

La vida del jugador se representa mediante cinco corazones rojos de los cuales cada uno representa un 20 por ciento de la vida del jugador. Los corazones desaparecerán siendo negros si se pierde la vida proporcional a la cantidad de un contenedor de corazón.



Figure 6: Imagen del juego en ejecución y la interfaz de vida

7.2 Menú de pausa

7.2.1 Pantalla del menú

La pantalla del menú consta de 3 botones los cuales permiten reanudar la partida, acceder al submenú de opciones y salir al menú principal del juego.

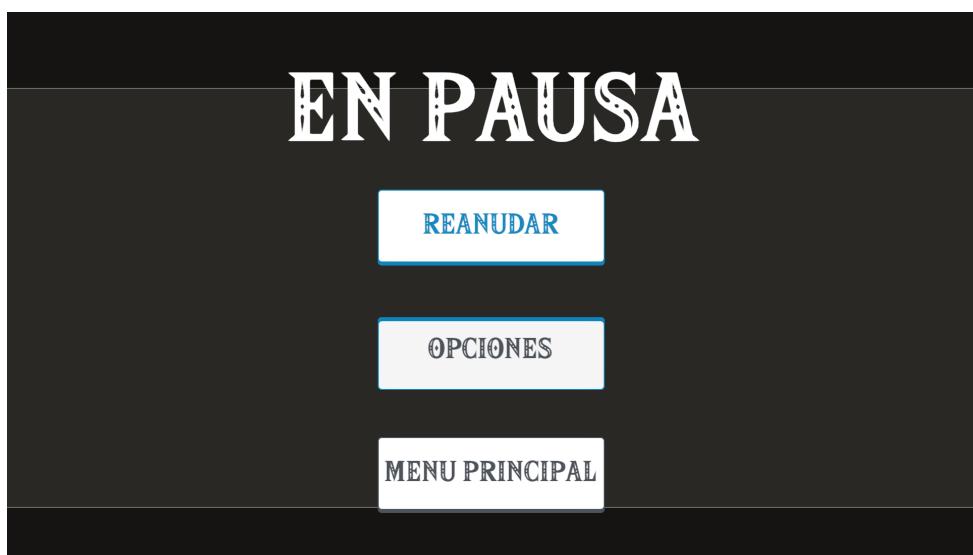


Figure 7: Imagen del juego en pausa

7.2.2 Opciones

Las opciones permiten modificar el volumen del juego y reasignar teclas según preferencia del jugador. Importante recordar que la columna de la derecha del submenú opciones corresponde a botones del mando.



Figure 8: Imagen del juego en el menú de opciones

7.3 Menú principal del juego

7.3.1 Pantalla inicial del juego

El juego nada más cargar muestra el menú principal del juego con dos botones que nos permiten acceder al juego como tal mandándonos a la selección de personajes.

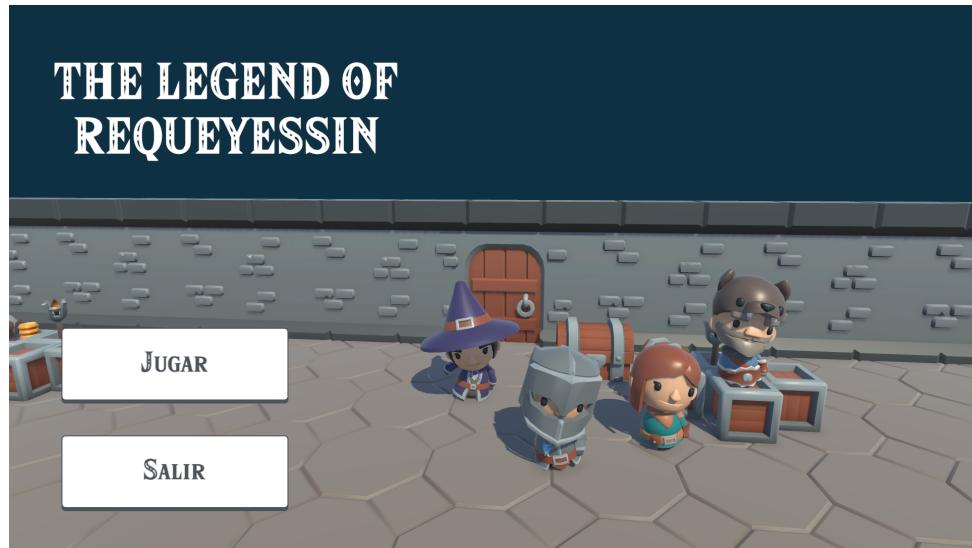


Figure 9: Pantalla principal del juego

7.3.2 Selección de personaje

Una vez pulsado el botón de jugar podemos acceder a la selección de personaje desde el cuál se puede desplazar de derecha a izquierda con los botones laterales en pantalla para poder seleccionar

el personaje. Los dos botones situados debajo de la descripción del personaje permiten cambiar la dificultad y elegir el personaje de modo que al elegir personaje comienza la partida.



Figure 10: Selección de personajes en la pantalla principal

8 Implementación en Unity

La implementación en Unity nos permite comprender mejor como se forman distintas partes del juego y su conexión. No se trata de profundizar sino de entender la estructura general del proyecto.

Podemos definir el proyecto como una combinación entre varias escenas. La escena del menú principal, la escena que representa la pantalla de carga y el propio juego/dungeon como tal.

La escena del menú principal consta de varios elementos que representan los personajes bailando, y demás elementos de gestión de Interfaz, música... Hay un objeto sin embargo, que realmente es importante. Se encarga de limpiar los datos estáticos de aquellos eventos disparados tras determinadas acciones en el juego. Por ejemplo, para actualizar la vida en la interfaz se dispara un evento cuando el jugador recibe daño. Dicho evento es estático y no se borrará entre escenas, sin embargo, el resto de objetos si lo hará.

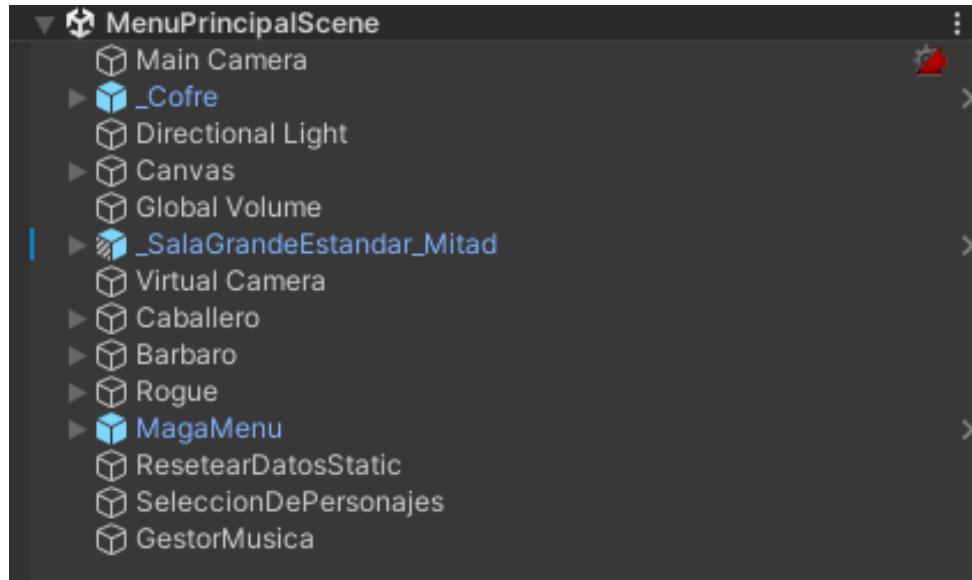


Figure 11: Objetos de la escena del menú principal

La escena de la pantalla de carga es muy simple y no requiere ningún tipo de habilidad. Lo único que puede presentar un desafío es el Callback para poder usar de forma eficiente la pantalla de carga. Este basicamente sirve como puente entre transición de escenas de modo que siempre se muestra la pantalla de carga durante al menos un frame para asegurar que la carga sea completa en la escena nueva a cargar.

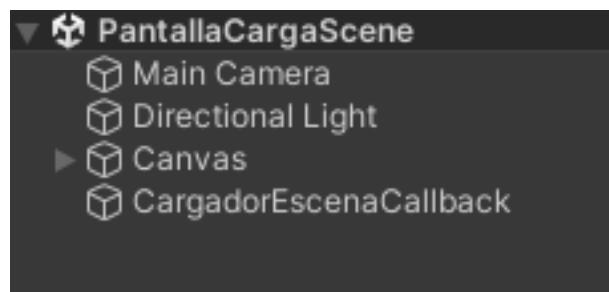


Figure 12: Objetos de la escena de carga

Finalmente la escena del juego es más compleja y consta de más elementos que permiten componer el producto final del juego.

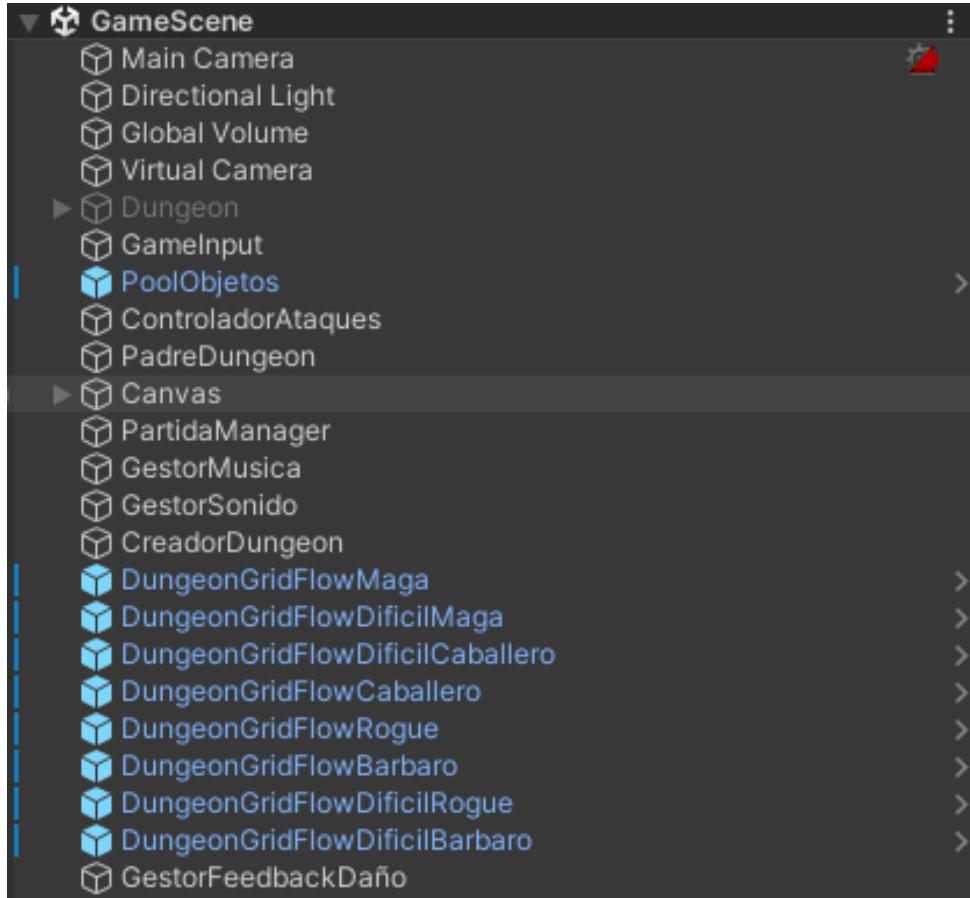


Figure 13: Objetos de la escena de juego

La escena del juego consta de un objeto que es el GameInput script que controla y avisa con eventos a aquellos métodos de movimiento para el jugador. Esto aplica para movimiento, ataque, interacción...

8.1 Pool de Objetos

Tenemos por otro lado la Pool de objetos. La Pool de objetos se puede interpretar como una urna donde se meten bolitas de distintos tipos. Aquel que quiera una bolita debe sacarla de la urna, pero esta urna está creada desde el primer momento por lo que una persona no tiene que meter las bolitas en medio del juego. Entonces, una bolita es un arma y el que coge una bolita es un enemigo o un cofre por ejemplo. La gracia de la Pool de objetos es que instanciamos un set de 20 armas por ejemplo de entre unos pocos prefabs, de modo que aquel que quiera un arma aleatoria obtendrá dicha instancia no teniendo que crearla en medio de la ejecución ahorrando memoria. Si un enemigo muere devuelve el arma a la Pool de Objetos para que pueda ser usada por otro enemigo o elemento que lo necesite. Obviamente si no quedasen bolitas en la urna se crean más de forma extraordinaria. Es un método universal en este tipo de juegos y quienes somos nosotros para alterarlo. La implementación es personal y carece de muchos elementos no necesarios habitualmente usados.

8.2 Gestor de la partida

Partida Manager se encarga de controlar el estado del juego en todo momento, ya sea Victoria, Pausa, Game Over o Jugando. Gracias a ello y a tener una instancia Singleton cualquier objeto puede en cualquier momento actuar acorde a la situación de la partida. Por ejemplo, en la pausa, victoria o derrota el tiempo se congela. También se encarga de escuchar a otras clases para poder determinar el estado de la partida, véase el jugador muere por lo que la partida debe entrar a estado Game Over. La gestión de estados de la partida es un ENUM. También lleva la cuenta de la puntuación del jugador.

8.3 Gestor de sonido

El gestor de Sonido es una clase que se suscribe a los eventos de otras muchas para saber cuando ciertas acciones son ejecutadas y reproduce los sonidos pertinentes. Si bien esto podría hacerlo la propia clase que realiza la acción, es buena práctica centralizarlo todo en una sola clase que gestiona todos los sonidos de la partida.

8.4 Gestor de música

El gestor de Música simplemente sirve para reproducir la música de fondo. También permite el ajuste de volumen desde otra clase al igual que el gestor de sonido. Realmente es la interfaz y su clase la que desde las opciones del juego transmita el cambio del volumen por parte del usuario.

8.5 Creador de la Dungeon

El Creador de Dungeon es simplemente el generador de Dungeon que elige un preset de Dungeon en base a la selección de personaje y dificultad en el menú principal. Esta clase también asigna a el objeto PadreDungeon como padre de todos los elementos de la dungeon para una estructura ordenada de la ejecución.

8.6 Gestor de feedback

En último lugar tenemos el Gestor de Feedback de Daño que se encarga de cambiar los materiales del enemigo o Player al recibir daño para transmitirlo al jugador. Es importante que el jugador sepa que su arma está aplicando daño o que su personaje lo está sufriendo.

9 Referencias

References

- [1] Dungeon Architect Asset. URL: <https://dungeonarchitect.dev/unity>.
- [2] Lootear. “Anglicismo proveniente de la habla inglesa usado para indicar el saqueo de objetos en un videojuego. La palabra proviene de "loot" del inglés”. In: ().
- [3] Kay Lousberg. *Assets Dungeon*. URL: <https://kaylousberg.itch.io/>.