Escuela de Doctorado
y Estudios de Posgrado
Universidad de La Laguna

MÁSTER UNIVERSITARIO EN DESARROLLO DE VIDEOJUEGOS

MainRPG

Óscar González García

D. **Jesús Miguel Torres Jorge**, con N.I.F. 43826207-Y profesor contratado doctor de Universidad adscrito al Departamento de Ingeniería informática y de sistemas de la Universidad de La Laguna, como tutor.

C E R T I F I C A (N)

Que la presente memoria titulada:

“MainRPG”

ha sido realizada bajo su dirección por D. **Óscar González García**, con N.I.F. 54109814-Z.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 31 de agosto de 2022.

Agradecimientos

En primer lugar quiero agradecer a mi tutor Jesús Miguel Torres Jorge ya que con sus conocimientos y experiencia en los distintos campos, ha facilitado y atenuado cada una de las etapas vividas/trabajadas durante el proyecto.

También quiero agradecer a mis amigos, familiares y novia que me han brindado apoyo, comprensión y cariño durante estos meses.

Y por último a la Universidad de La Laguna y los profesores asignados en el máster de desarrollo de videojuegos por el aprendizaje y experiencia que me han dado durante este año.

Resumen

El objetivo de este proyecto ha consistido en la creación de un prototipo de videojuego RPG (Role-Playing Game) 2D y combate por turnos. Se controla a un aldeano que vive solo con su abuelo llamado Gabe, el cual habita en un mundo donde los animales y monstruos poco a poco se van volviendo más agresivos por causas desconocidas. El prototipo se desarrolla en el pueblo principal donde se podrá explorar libremente y existirá una misión con una introducción al combate.

El motor para la creación de videojuegos utilizado para la realización del proyecto fue Unity.

Palabras clave: Prototipo, RPG, 2D, Unity.

Abstract

The objective of this project has been to create a prototype of a 2D RPG video game (Role-Playing Game), turn-based combat. A villager is controlled who lives alone with his grandfather named Gabe, who lives in a world where animals and monsters gradually become more aggressive for unknown reasons. The prototype take place in the main town where it can be freely explored and there will be a mission with an introduction to combat.

The video game creation engine used to carry out the project was Unity.

Keywords: Prototype, RPG, 2D, Unity.

Índice general

Introducción **9**

Antecedentes 9

Objetivo General 9

Objetivos Específicos 9

Documento de Diseño de Videojuegos **11**

Desarrollo del proyecto **28**

Estructura del proyecto 28

Tipos de Scriptable Objects 30

Estructura de las escenas 37

Escena “Managers” 38

Escena “Gameplay” 40

Escenas de juego 46

Escenas de combate 49

Escena “MainMenu” 50

Conclusiones y líneas futuras **51**

Summary and Conclusions **53**

Bibliografía **55**

Índice de figuras

Figura 1: Estado del juego 14
Figura 2: Menú principal 15
Figura 3: HUD de pantalla de juego 16
Figura 4: Menú de pausa 17
Figura 5: HUD de pantalla de combate 18
Figura 6: Casa de Gabe 19
Figura 7: Pueblo principal 20
Figura 8: Finca de Israel 21
Figura 9: Casa de Jacob y Fran 22
Figura 10: Gabe 22
Figura 11: Grani 23
Figura 12: Israel 23
Figura 13: Perla 24
Figura 14: Alan 24
Figura 15: Fran 25
Figura 16: Jacob 25
Figura 17: Limo 26
Figura 18: Abeja mutante 26
Figura 19: Espada de madera 27
Figura 20: Espada de plata 27
Figura 21: Cortes de plata 27
Figura 22: Estructura de carpetas 28

Figura 23: Scriptable Objects 30
Figura 24: Game Events 35
Figura 25: Escenas 37
Figura 26: Escena “Managers” 38
Figura 27: Esquema del cambio de escenas 39
Figura 28: Escena “Gameplay” 40
Figura 29: Esquema del sistema de combate 42
Figura 30: Imagen de combate 42
Figura 31: Esquema del sistema de diálogos 44
Figura 32: Imagen de diálogo 44
Figura 33: Esquema del manager de la UI 45
Figura 34: Escenas de juego 46
Figura 35: Escenas de combate 49
Figura 36: Escenas “MainMenu” 50

1. Introducción

1.1. Antecedentes

Desde que me regalaron el primer cartucho de la *Game Boy Advance*, el ámbito de los videojuegos fue un mundo que quise explorar. Cursar Ingeniería Informática y el Máster de Desarrollo de Videojuegos fue, en parte, por ello. Al ser fan de los videojuegos RPG, combate por turnos como *Pokémon* y estar suscrito a un curso de *Unity* tocando los temas anteriores, se decidió realizar el proyecto "MainRPG" en el trabajo de fin de máster

1.2. Objetivo General

El proyecto presenta una serie de objetivos generales:

- Hacer uso de algunas de las herramientas enseñadas y poner en práctica el conocimiento adquirido a lo largo del curso académico.
- Crear un videojuego activo con los objetivos específicos del siguiente apartado presentes.

1.3. Objetivos Específicos

El proyecto presenta una serie de objetivos:

- Creación de escenarios haciendo uso de los *Tilemaps*.

- Creación de algunos personajes.
- Crear un Script para el cambio de escenas.
- Crear un *Time Manager*.
- Crear un sistema de diálogo.
- Crear un sistema de combate.
- Crear un sistema de escalado de niveles.
- Añadir, al menos, una misión que hacer.

2. Documento de Diseño de Videojuegos

Título

MainRPG

Diseñador

Óscar González García

Género

RPG

Plataforma

PC

Sinopsis de jugabilidad y contenido

El jugador controlará a un aldeano en un mundo donde los animales y monstruos se han vuelto agresivos por causas desconocidas y se podrán combatir en un combate por turnos para subir de nivel y conseguir

recompensas. Esta primera versión contará con el pueblo principal del protagonista para explorar y una misión, además de una zona de combate.

Categoría

Al compararlo con algún juego *RPG* como puede ser **Dragon Quest**. En ambos juegos, el mundo es afectado por un poder maligno y el protagonista busca una manera de eliminarlo. Para ello, tendrán que completar misiones y subir de nivel para poder equiparse para el desafío final.

Mecánica

Los controles del personaje son:

- Moverse en todas las direcciones.
- Interactuar con aldeanos, carteles, etc.

Los controles del personaje dentro del combate son:

- Atacar con el arma principal o secundaria.
- Atacar con los poderes que contienen algunas armas.

En esta primera versión, el objetivo será completar una pequeña misión solicitada por uno de los aldeanos. El jugador tendrá que seguir los pasos que se les vayan dando, ya sea interactuando con el entorno o combatiendo.

Público

El público objetivo del videojuego sería para todas las edades.

Visión general del juego

Este proyecto es una versión muy recortada y puesta en marcha de una serie de ideas premeditadas para lo que podría ser un videojuego a gran escala. El objetivo es dar una primera toma de contacto a la historia, a algunos personajes del pueblo y al combate para mostrar las mecánicas.

Mecánica del juego

La cámara durante la exploración seguirá al protagonista dentro de un confinamiento. Durante las escenas de combate, se quedará estática en medio del jugador y del enemigo.

Las mecánicas del juego son:

- Desplazamiento: Con los botones del teclado **A** para el movimiento hacia la izquierda, **D** para el movimiento hacia la derecha, **W** para el movimiento hacia arriba y **S** para el movimiento hacia abajo.

- Interacción: Con la tecla del teclado **E**.

- Pasar al siguiente mensaje: Clic izquierdo con el ratón en cualquier lugar de la pantalla.

- Acciones en escenas de combate: Clic con el ratón en los diferentes botones de la escena tanto para atacar con el arma principal o secundaria como para las habilidades.

El sistema de puntuación existente en el juego es el oro y la experiencia al derrotar a enemigos o completar misiones.

En cuanto al sistema de guardado, no hay en la versión entregada, dado que se puede completar en 5-10 minutos.

Estado del juego

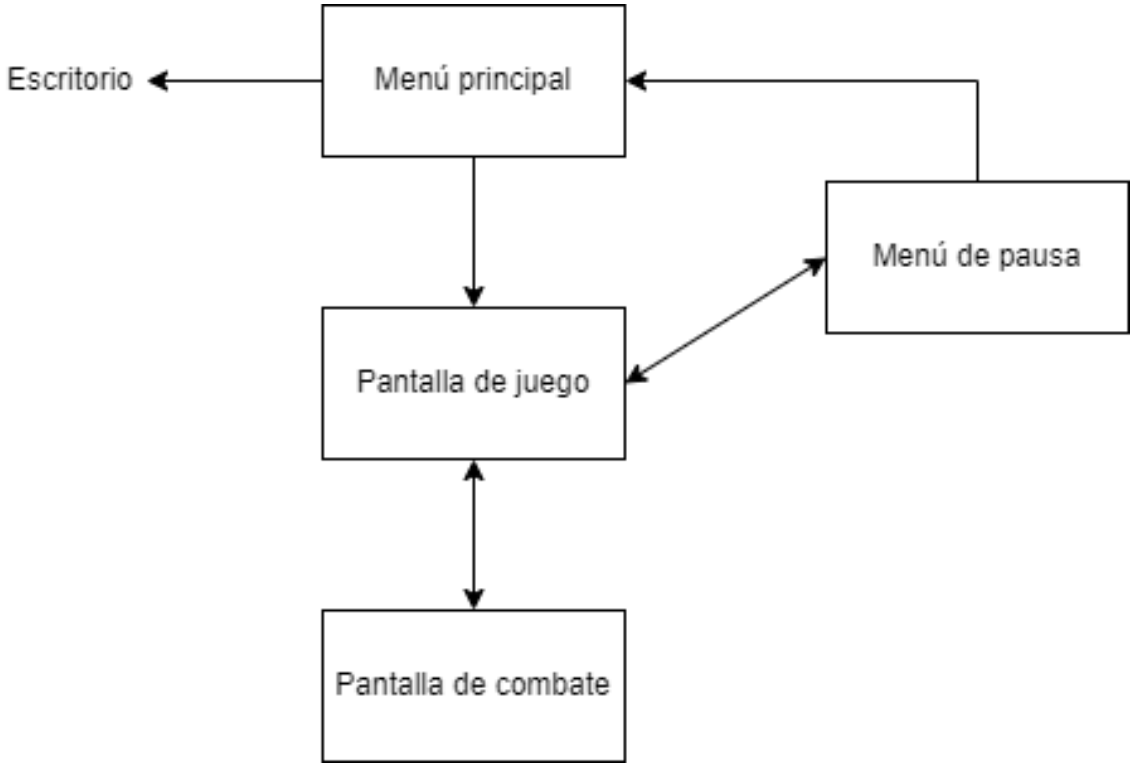

Figura 1: Estado del juego

Interfaces

- **Menú principal**: Esta es la pantalla principal del juego, es a la que se accede nada más entrar al juego. Contará con 2 opciones:

- Empezar la partida.
- Salir del juego.

Esta pantalla es accesible desde los estados de juego:

- Menú de pausa.

Figura 2: Menú principal

- **Pantalla de juego**: Esta es la pantalla en la que el usuario podrá jugar y superar el juego. Esta, a su vez, tiene un HUD arriba a la derecha, donde existe un botón de pausa.

  Desde esta pantalla se puede acceder a:

  - Menú de pausa.

  Esta pantalla es accesible desde los estados de juego:

  - Menú principal.

- Menú de pausa.

Figura 3: HUD de pantalla de juego

- **Menú de pausa**: Esta es la pantalla a la que se accede cuando se le da al botón de pausa. Contará con 2 opciones:

  - Continuar.
  - Menú principal.

  Esta pantalla es accesible desde los estados de juego:

  - Pantalla de juego.

Figura 4: Menú de pausa

- **Pantalla de combate**: Esta será la pantalla predeterminada cuando el usuario entre en combate contra un enemigo. Contendrá un HUD dividido en 3 partes:

    - Nombre, nivel, barra de vida y maná del usuario ubicado arriba a la izquierda de la pantalla.
    - Nombre, nivel, barra de vida y maná del enemigo ubicado arriba a la derecha de la pantalla.
    - Botones para el ataque del arma principal, secundaria y las habilidades colocadas en la parte inferior central de la pantalla.

  Desde esta pantalla se puede acceder a:

    - Pantalla de juego (Solo finalizando el combate).

  Esta pantalla es accesible desde los estados de juego:

    - Pantalla de juego (Al entrar en combate).

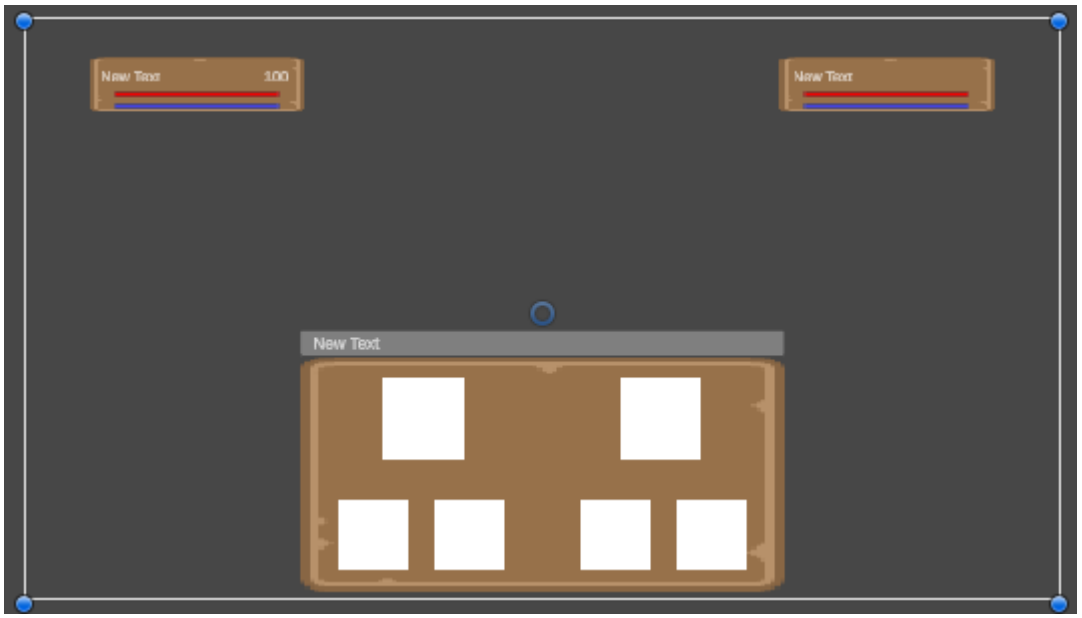

Figura 5: HUD de pantalla de combate

Niveles

- **Casa de Gabe**: Casa del protagonista y de su abuelo, es la primera zona del juego, la que se accede desde el menú principal. Es una casa con dos habitaciones en cada lado y el salón en el medio junto a la salida. Al intentar acceder al salón, tu abuelo te llamará y te encargará un recado (Ayudar a tu vecino Israel), al acabar todo ese evento, se le permitirá al usuario salir.

  En la casa se puede encontrar a los personajes:
  - Grani, abuelo de Gabe.

Figura 6: Casa de Gabe

- **Pueblo principal**: Pueblo donde vive el protagonista, no hay ningún evento en especial para avanzar en la historia, pero existen personajes con los que hablar y una zona de combate. Contiene 3 casas y una salida por la izquierda para acceder hacia la granja de Israel.
  En esta zona se puede encontrar a los personajes:
  - Mani, amiga y vecina de Gabe.
  - Alan, caballero viajero.
  - Fran, Hijo del vecino Jacob.

Figura 7: Pueblo principal

- **Granja de Israel**: Lugar donde te reúnes con Israel para llevar a cabo su misión, la cual consiste en coger un arma de su baúl y derrotar a los 3 *limos* que le comieron la cosecha. Contiene una casa, una zona vallada y saliendo por la parte de la derecha, se accede al pueblo.

  En esta zona se puede encontrar a los personajes:
  - Israel, granjero y vecino.

Figura 8: Finca de Israel

- **Casa de Jacob y Fran**: Lugar secundario al que se accede desde el pueblo, en ella se encontrará Jacob con el que se puede tener una conversación sobre su pasado.

  En esta zona se puede encontrar a los personajes:
  - Jacob, vecino y padre de Fran.

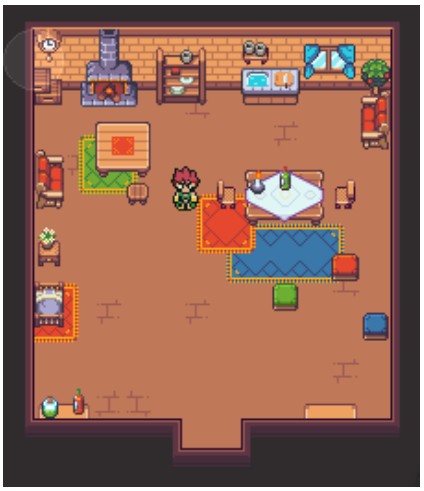

Figura 9: Casa de Jacob y Fran

Cabe recalcar que solamente existe una canción para todo el juego y ningún efecto de sonido.

Progreso del juego

El progreso del juego es el siguiente:

1. Hablar con Grani (abuelo de Gabe)
2. Ir a la granja de Israel.
3. Hablar con Israel.
4. Coger la espada del baúl.
5. Derrotar a los 3 *limos*.

Personajes

- **Gabe**: Aldeano que vive con su abuelo, protagonista y, por lo tanto, personaje jugable del videojuego. Tiene el pelo marrón y su camiseta y pantalón de color verde. Se trata de una persona que le gusta y está siempre dispuesta a ayudar a los demás, sobre todo a su abuelo Grani. Gabe no posee ningún objeto ni arma propia característica, pero sí podrá equiparse 2 armas e irlas cambiando.

Figura 10: Gabe

- **Grani**: Personaje no jugable y abuelo de Gabe. Tiene el pelo canoso peinado haciendo un tupé, además de tener largos bigote y barba. Utiliza

algo que parece un albornoz de color fucsia. Este personaje estará estático en la casa del protagonista y será el que le asigne la misión principal a Gabe.

Figura 11: Grani

- **Israel**: Personaje no jugable y vecino de Gabe. Presenta una calvicie parcial. Las zonas con pelo y la barba larga son de color gris. Va apoyado en su bastón marrón y lleva una camisa verde claro y unos pantalones con tirantes de color verde fuerte. Este personaje es estático, estará siempre en su finca y será la persona a la que debas ayudar, te dará algunos consejos y tendrás que derrotar a los 3 *limos* que se comieron su cultivo.

Figura 12: Israel

Los personajes nombrados anteriormente son los más relevantes en la versión actual del juego, los nombrados a continuación solamente te ofrecen una conversación cuando se interactúa con ellos.

- **Perla**: Amiga y vecina de Gabe. Tiene característicos ojos azules, pelo largo verde recogido con un lazo rojo y camiseta y pantalones azules. Este

personaje es estático, ubicado al lado del puente que atraviesa un pequeño rio y conduce a la finca de Israel.

Figura 13: Perla

- **Alan**: Viajero que pasó por el pueblo y descansa bajo la sombra de un árbol. Pertenece a una raza mitad humano, mitad gato y es un caballero. Tiene el pelaje negro y los ojos amarillos, lleva una espada envainada, una capa roja y la ropa es de color morada. Si se interactúa con él, advierte que últimamente los animales y monstruos actúan extraño y de manera agresiva.

Figura 14: Alan

- **Fran**: Vecino e hijo de Jacob. Un chico joven con pelo marrón y piezas de vestir difíciles de describir de colores violeta y amarillo. Quiere llegar a ser caballero algún día como lo fue su padre. Es un personaje estático ubicado en la parte norte del pueblo.

Figura 15: Fran

- **Jacob**: Vecino y padre de Fran. Caballero retirado por la herida de una flecha en la rodilla. Tiene el pelo alocado de color rojo y lleva una gabardina verde, camiseta gris y pantalones azules junto con guantes y botas amarillas. Es un personaje estático que se puede encontrar en el salón de su casa.

Figura 16: Jacob

Enemigos

- ***Limos***: Enemigos de color azul que suelen presentarse de forma redonda u ovalada, pero realmente no tienen una forma concreta. Son pegajosos y débiles. No tienen ninguna habilidad especial, simplemente atacan saltando y golpeando con todo su cuerpo. Se encuentran 3 estáticos en la finca de Israel. También se pueden combatir con ellos en la zona de combate del pueblo.

Figura 17: Limo

- **Abeja mutante**: Abeja de tamaño descomunal que ataca con su aguijón. Conservan el mismo color que las abejas comunes y son criaturas más peligrosas que los *limos*. No tienen ninguna habilidad especial. Se pueden encontrar en la zona de combate del pueblo.

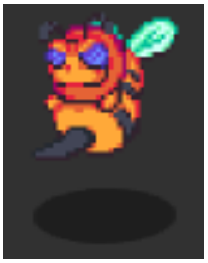

Figura 18: Abeja mutante

Armas

- **Espada de madera**: Es la espada más básica del juego, se encuentra en el baúl de Israel una vez se avance en la misión. Tiene 10 de daño y no contiene ninguna habilidad especial.

Figura 19: Espada de madera

- **Espada de plata**: Espada obtenida en un baúl del pueblo, tiene 20 de daño y una habilidad especial llamada “Cortes de plata”.

Figura 20: Espada de plata

- **Cortes de plata**: Varios golpes consecutivos realizados por una espada de plata, realiza 40 puntos de daño y consume 30 puntos de maná.

Figura 21: Cortes de plata

3. Desarrollo del proyecto

3.1. Estructura del proyecto

La estructura de carpetas utilizada en el proyecto es la siguiente:

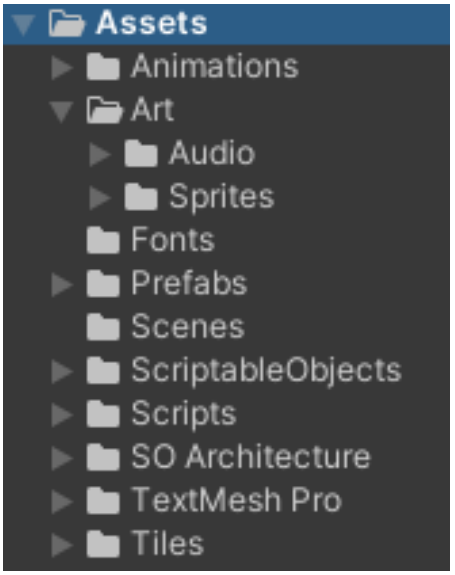

Figura 22: Estructura de carpetas

- **Animations**: Contiene todas las animaciones del proyecto.
- **Art**: Como se puede ver en sus subcarpetas, contiene los sprites y audios utilizados en el videojuego.

- **Fonts**: Contiene tipos de letras descargados para los menús, HUDs y conversaciones en el juego. Aunque en este caso se utilizó solo uno para todo.
- **Prefabs**: Contiene todos los elementos prefabricados utilizados.
- **Scenes**: Como su nombre indica, almacena todas las escenas en ella.
- **ScriptableObjects**: Contiene todos los objetos scriptables del juego, contenedores de datos que se pueden usar para guardar grandes cantidades de información, independientemente de las instancias de clase. Además, los datos almacenados pueden ser modificados por scripts.
- **Scripts**: Todos los scripts están almacenados en esa carpeta y entre todas las subcarpetas, existe una cuyos scripts se basan únicamente en crear tipos de *Scriptable Objects* para usarlos en el proyecto.
- **SO Architecture**: Al utilizar el paquete de los objetos scriptables, toda su estructura y scripts necesarios fueron almacenados en esta carpeta.
- **TextMesh Pro**: Es idéntico al caso anterior, pero en esta carpeta se almacenan los datos de los *TextMeshPro* para la mejora de los textos en la UI.
- **Tiles**: Contiene todos los atlas utilizados para los *tilemaps*.

3.2. Tipos de *Scriptable Objects*

Los *Scriptable Objects*, como se comentó antes, son contenedores de datos e incluso lógica (pueden tener funciones asociadas). En este proyecto se crearon varios tipos:

Figura 23: Scriptable Objects

- **AbilitySO**: Es utilizado para las habilidades de las armas, los campos son:
  - Una cadena de texto para el nombre.
  - Un número flotante para el daño que provoca.
  - Un número flotante para el maná que consume.

    - Una cadena de texto para el tipo de magia a la que pertenece (idea no implementada en esta versión).
    - Un *Sprite* para la imagen de la habilidad.

- **WeaponSO**: Es utilizado para las armas del juego, los campos son:
    - Una cadena de texto para el nombre.
    - Un número flotante para el daño.
    - Una cadena de texto para el tipo de magia que tiene (no implementada en esta versión).
    - Una cadena de texto para señalar la rareza del arma (por ahora irrelevante en esta versión).
    - Un *Sprite* para la imagen del arma.
    - Dos AbilitySO porque cada arma puede llegar a tener hasta 2 habilidades asociadas, este campo puede estar vacío.

- **InventorySO**: Es utilizado para el inventario del jugador (por ahora muy simple), los campos son:
    - Un entero para el oro del jugador.
    - Dos WeaponSO para las armas equipadas.

- **CharacterSO**: En las conversaciones, aparece un cuadro de texto y otro cuadro donde se muestra el personaje que está hablando. Los campos son:
    - Una cadena de texto para el nombre.
    - Un *Sprite* para la imagen de la cara del personaje

- **CombatUnitSO**: Toda la información de los personajes a la hora del combate es sacada de estos *Scriptable Objects*, usados ya sea tanto para el cálculo del daño en combate o actualizar la UI. Los campos son:
    - Una cadena de texto para el nombre de la unidad.
    - Un entero para el nivel, este campo se igualará antes del combate al del jugador.
    - Un flotante para la experiencia base. Se trata de un valor utilizado para que la experiencia necesaria para subir de nivel no sea en todos los niveles la misma, sino que cada vez sea mayor. Todas las estadísticas base servirán para eso en este *Scriptable Object*.
    - Un flotante para la experiencia ganada tras los combates. Ese campo será el que se vaya acumulando y aunque todas las unidades de combate las tenga, solo la usará el jugador.
    - Un flotante para el máximo de experiencia. El valor al que debe llegar la experiencia para subir de nivel.
    - Un flotante para la salud base. Funciona igual que con la experiencia, sirve para aumentar la salud máxima en cada nivel.
    - Un flotante para la salud actual.
    - Un flotante para la salud máxima.
    - Un flotante para el maná base. Una vez más, para que no siempre el maná máximo sea el mismo nivel tras nivel.
    - Un flotante para el maná actual.
    - Un flotante para el maná máximo.
    - Un flotante para el poder de ataque base.
    - Un flotante para el poder de ataque actual.
    - Un flotante para el poder de curación base.
    - Un flotante para el poder de curación actual (Este punto y el anterior no han sido utilizados en esta versión).

    - Un entero para la cantidad de oro que suelta el enemigo al ser derrotado.
    - Un flotante para la cantidad de experiencia que gana el jugador al derrotar a esta unidad. Los dos últimos puntos solo son rellenados para las unidades de combate enemigas.

- **ConversationSO**: Utilizado para los textos y personajes que aparecen en una conversación. Los campos son:
    - Dos CharacterSO, uno para personaje que participe en la conversación.
    - Como las conversaciones pueden tener una cantidad de texto variable y el personaje que habla va alternando, se creó una estructura de datos llamada “Sentence”, la cual requiere de un CharacterSO, de los del punto anterior, y una cadena de texto, que será lo que ese personaje diga. Este campo será de un Array de “Sentences”.

- **GameStateSO**: Almacenará todos los estados del juego (en combate, diálogo, jugando, etc.). Los campos son:
    - Una cadena de texto para el nombre del estado del juego.

- **GameManagerSO**: Es el *Scriptable Object* que almacenará el estado del juego en el que se encuentra el jugador. Dependiendo del estado en el que se encuentre, se desencadenarán acciones que se explicarán más adelante. Los campos son:
    - Un GameStateSO para indicar el estado de juego actual.

    - Un GameStateSO que hará referencia al estado de juego que hubo antes del cambio, por si se da el caso de que se quiere volver al anterior.

- **InGameEventSO**: Utilizados para marcar como completados eventos que ocurran en el juego. Al realizar acciones en el juego, es posible que tengan consecuencias. Existe un script que está observando este tipo de *Scriptable Object* para al marcarse como completado, realizar cambios en el escenario, ya sea cambiando conversaciones, ocultando *GameObjects*, etc. (Se hablará de esto más adelante). Los campos son:
    - Un booleano para marcar el evento como completado.

- **SceneSO**: Solamente almacena el nombre de una escena, esta debe coincidir. Los campos son:
    - Una cadena de texto para el nombre de la escena.

- LevelEntranceSO: Utilizado para la entrada de los niveles. No tiene campos, se debe asignar este *Scriptable Object* a un *GameObject* y ubicarlo donde sea la entrada en la escena.

- EntranceSceneToLoadSO: A la hora de realizar un cambio de escenas, porque, por ejemplo, el jugador entre en una casa, se requiere cargar esa escena y colocar al personaje en una ubicación pensada de antemano (en la entrada de la casa). Este *Scriptable Object* es modificado justo antes del cambio de escenas porque es el que se va a usar para mover al personaje entre escenas. Los campos son:

- Un SceneSO para la escena.
- Un LevelEntranceSo para la entrada en la escena.

- **PlayerPathSO**: Almacena la entrada por la que el personaje entrará en la escena (LevelEntranceSO). Existe un script en cada escena de juego, que se fija en este *Scriptable Object* para saber donde ubicar al jugador cuando se cargue una escena nueva. Los campos son:
  - Un levelEntranceSO para saber donde instanciar al jugador.

Existen también otro tipo de *Scriptable Objects* (*Game Event*) que son capaces realizar acciones cuando les llega un evento. Estos están a la escucha de que se lancen desde el inspector o scripts. También tienen una serie de campos que guardarán los valores para las operaciones.

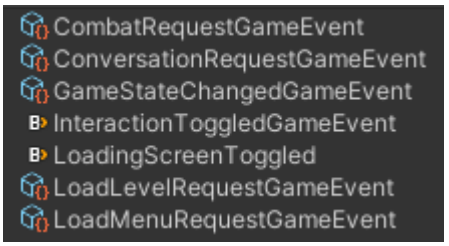

Figura 24: Game Events

- CombatRequestGameEvent: Se encarga de poder iniciar el combate, los parámetros son:
  - La CombatUnitSO del jugador
  - Un Transform con la posición en la que se colocará en pantalla.
  - Un Array de CombatUnitSO para los enemigos.

    - Un Transform donde se posicionarán.

- ConversationRequestGameEvent: Se encarga de empezar una conversación, recibe:
    - La ConversationSO que se quiere mostrar.

- GameStateChangedGameEvent: Es el encargado de cambiar el estado del juego para desencadenar unas acciones u otras. Los parámetros son:
    - Un GameStateSO con el estado de juego actual.

- InteractionToggledGameEvent: En un *Game Event* booleano, lanzará el evento cuando la variable esté a *true*. En los scripts del personaje, existe un *OnTriggerEnter2D* que estará observando si lo que entra tiene el script "interactable" asociado. Este script solamente tiene una serie de UnityEvent que serán los eventos que se ejecuten al interactuar.

- LoadingScreenToggled: Game Event booleano que al estar a true, pondrá la pantalla de carga (pantalla en negro).

- LoadLevelRequestGameEvent: Es el encargado de esperar a que le llamen y lanzar la acción de cargar un nivel, recibe:
    - La SceneSO que se quiere cargar.
    - Un booleano para marcar si habrá pantalla de carga.

- LoadMenuRequestGameEvent: Es igual al del punto anterior pero utilizado para cargar menús.

3.3. Estructura de las escenas

Para poder hablar de la estructura de las escenas, primero es necesario conocer qué tipos hay.

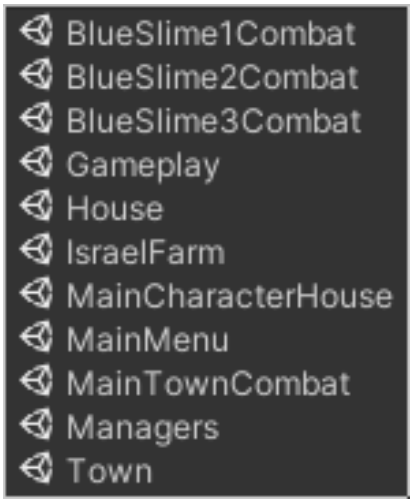

Figura 25: Escenas

- **MainMenu**: Menú principal del juego.
- **Town**, **IsraelFarm**, **House**, **MainCharacterHouse**: Escenas de las diferentes zonas del juego donde el jugador se va a mover.

- **BlueSlime1Combat**, **BlueSlime2Combat**, **BlueSlime3Combat**, **MainTownCombat**: Escenas diseñadas exclusivamente para los combates del juego.
- **Managers**: Escena donde están los managers de audio y de cambio de escena, además del *canvas* para la pantalla de carga (pantalla negra).
- **Gameplay**: Escena donde se encuentran los managers de conversaciones, tiempo, combate y de la UI. Además de un *canvas* que irá cambiando según en qué estado del juego se esté.

Las escenas de "Managers" y "Gameplay" son escenas auxiliares y siempre estarán cargadas junto a las demás.

3.3.1. Escena "Managers"

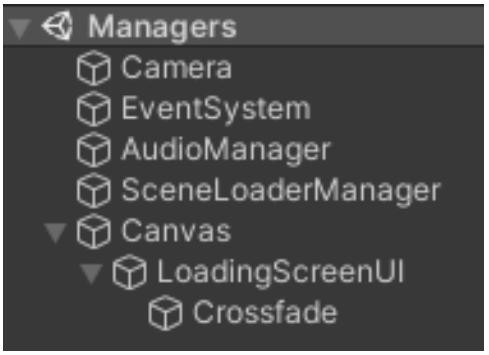

Figura 26: Escena "Managers"

- **AudioManager**: Contiene simplemente un “Audio Source” con la canción principal del juego.

- **SceneLoaderManager**: Es el manager encargado del cambio de escenas. Estará a la espera de que algún script en la escena lo llame. Contiene la carga tanto de niveles como de menú. Una vez sea lanzado el evento, llamará a *SceneManager* y buscará la escena con el mismo nombre que el que contiene el SceneSO y la cargará. Después se pondrá el *Game Event* encargado de la pantalla de carga a *true* o *false*, dependiendo de si se ha seleccionado o no. Acto seguido cambiará el estado del juego al de GameStateSO “Loading”.

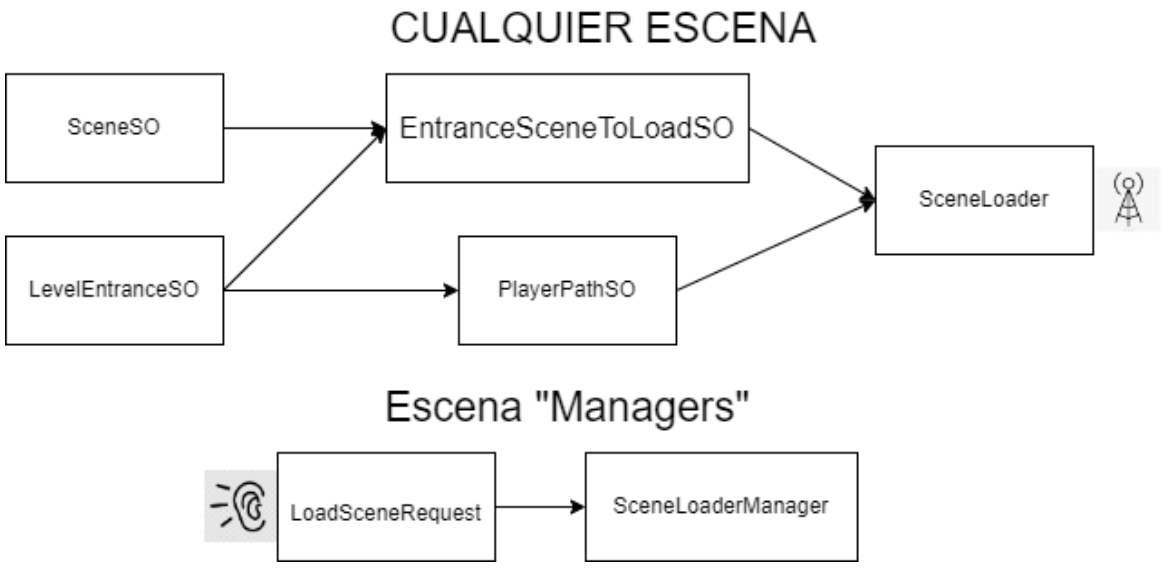

Figura 27: Esquema del cambio de escenas

- **Canvas**: Contiene una pantalla transparente con una animación para oscurecer hasta volverla completamente negra. Esto será llamado desde el SceneLoaderManager.

3.3.2. Escena “Gameplay”

Figura 28: Escena “Gameplay”

- **CombatManager**: como su nombre indica, es el manager encargado de los combates. Estará a la espera de que algún script lo llame y le pase los parámetros. El combate necesita saber de la UI, del InventorySO y de las CombatUnitSO del jugador y del enemigo.

  Primero se instancian los sprites de las unidades de combate en el campo de batalla. Después se configura la batalla. Al pasarle un array de unidades de combate enemigas, se escoge una aleatoria entre ellas, se le sube de nivel hasta tener el del jugador y se le restaura la vida y el maná. Hay que

tener en cuenta de que si previamente ha sido derrotado, los datos son persistentes y la salud la tendría a 0.

Lo siguiente es configurar la UI. Estas funciones las tiene un script de la propia UI para modularlo mejor. Se vacía la caja de texto de los mensajes e inicializan todos los valores residuales de la UI y se configura con los nuevos pasados.

Ahora sí, el combate empieza por el turno del jugador. Se crearon estados de combate con un *enum* para marcar mejor las etapas del combate. Existen **NONE**, **START**, **PLAYERTURN**, **ENEMYTURN**, **WON** y **LOST**. En el turno del jugador, el estado será **PLAYERTURN** y se esperará a que el jugador pulse un botón de la UI para el ataque. Al atacar, si la vida del enemigo no ha bajado a 0, pasará el estado a **ENEMYTURN** y el enemigo atacará (en esta versión, no utilizan habilidades). Esta secuencia se prolongará lo que sea necesario hasta que, o el jugador o el enemigo se debiliten, lo cual provocará que se cambie los estados **LOST** o **WON**. Dependiendo de cuál sea, la UI mostrará o la interfaz de victoria o derrota. En caso de derrota, también el jugador perderá la mitad de su oro y en caso de victoria ganará oro y experiencia. Si ha ganado lo suficiente como para subir de nivel, en vez de mostrar la interfaz de victoria, mostrará la de subida de nivel.

El combate será narrado a través de mensajes mostrados por la UI y entre acciones, existe una espera de un segundo y medio para que de tiempo de leerlos.

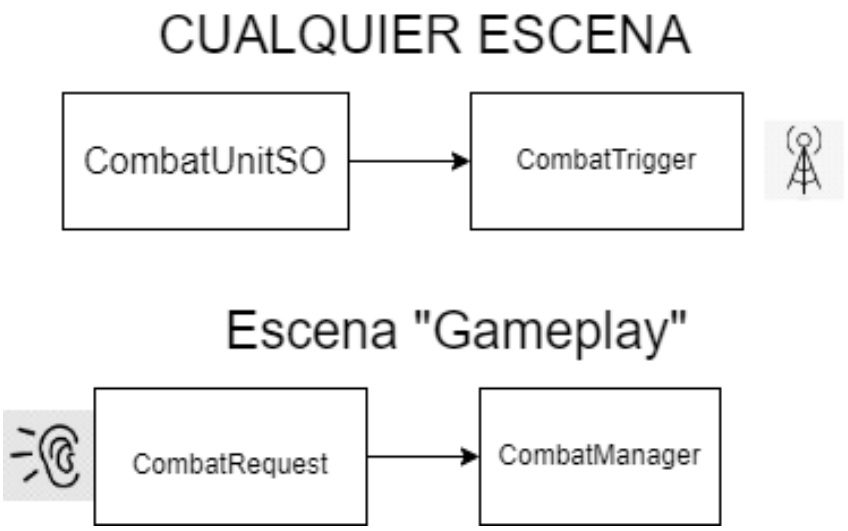

Figura 29: Esquema del sistema de combate

Figura 30: Imagen de combate

- **DialogueManager**: Es el manager encargado de los diálogos y estará a la espera de que lo llamen pasándole una

ConversationSO. Cuando es llamado, cambia el estado de juego al GameStateSO “Dialogue” y después empieza todo lo relacionado con el diálogo. Pondrá en cola todos los textos y posteriormente llamará a la UI para que haga operaciones. Como en el caso anterior, la lógica de la UI y del manager de diálogos está separada. La UI se inicializará e irá mostrando el texto letra por letra y se sabrá qué cara de personaje poner, porque recibirá el nombre del personaje. Si el usuario hace clic en la pantalla antes de que el texto acabe de imprimirse, se llamará a una función que lo mostrará por completo de golpe. Si se clica cuando ya acabó, se empezará a mostrar la siguiente y se eliminará de la cola el anterior. Cuando acabe la conversación, el estado del juego volverá a ser el anterior.

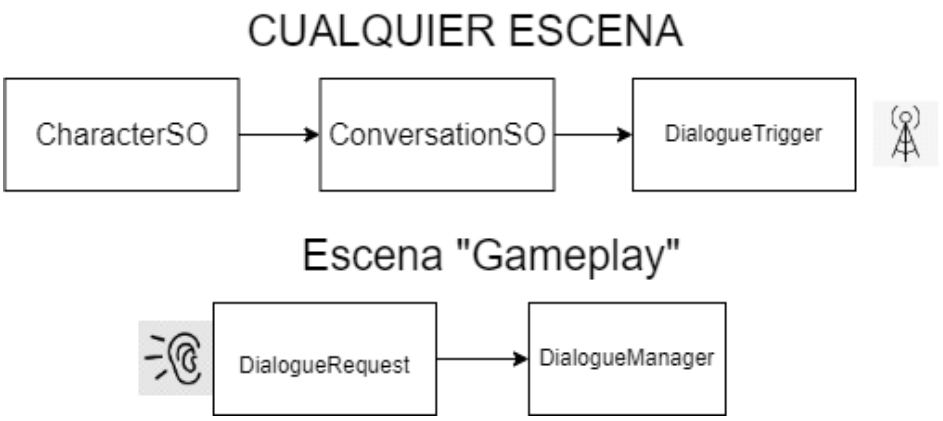

Figura 31: Esquema del sistema de diálogos

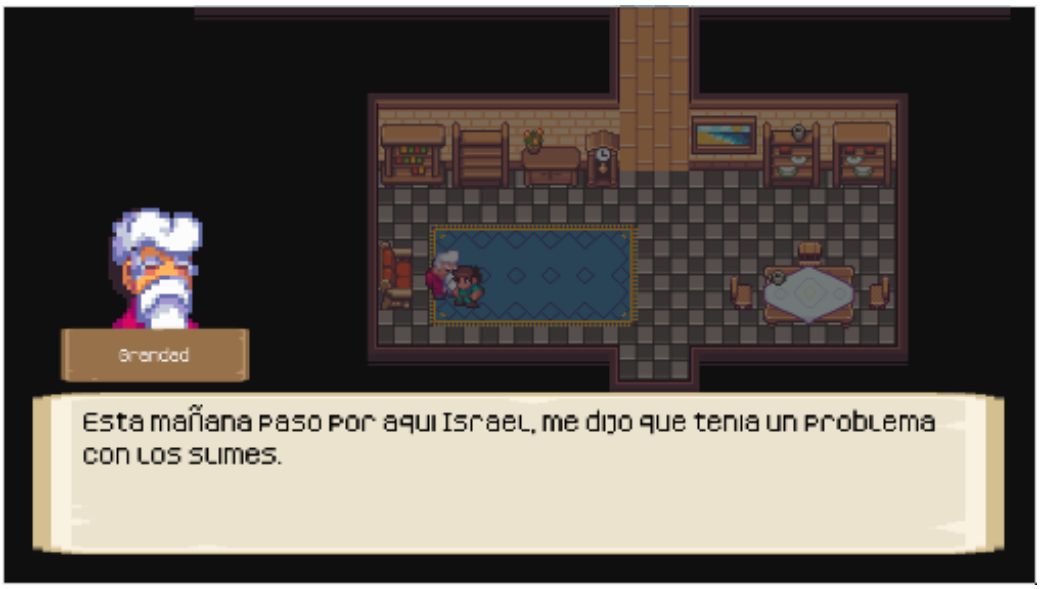

Figura 32: Imagen de diálogo

- **TimeManager**: Es el manager encargado de los escalados de tiempo en el juego. Es principalmente utilizado para modificar el valor del *Time.timeScale* a 0 cuando se está en el menú de pausa. Este manager no puede ser utilizado desde cualquier escena y tampoco hace falta para esta versión.

- **UIManager**: Es el manager encargado de mostrar las interfaces y está muy asociado a los estados del juego. Cada vez que sea lanzado el evento de cambio del estado del juego, se invocarán acciones. Dependiendo del GameStateSO que esté seleccionado, se desactivarán unas interfaces y se activará la correcta.

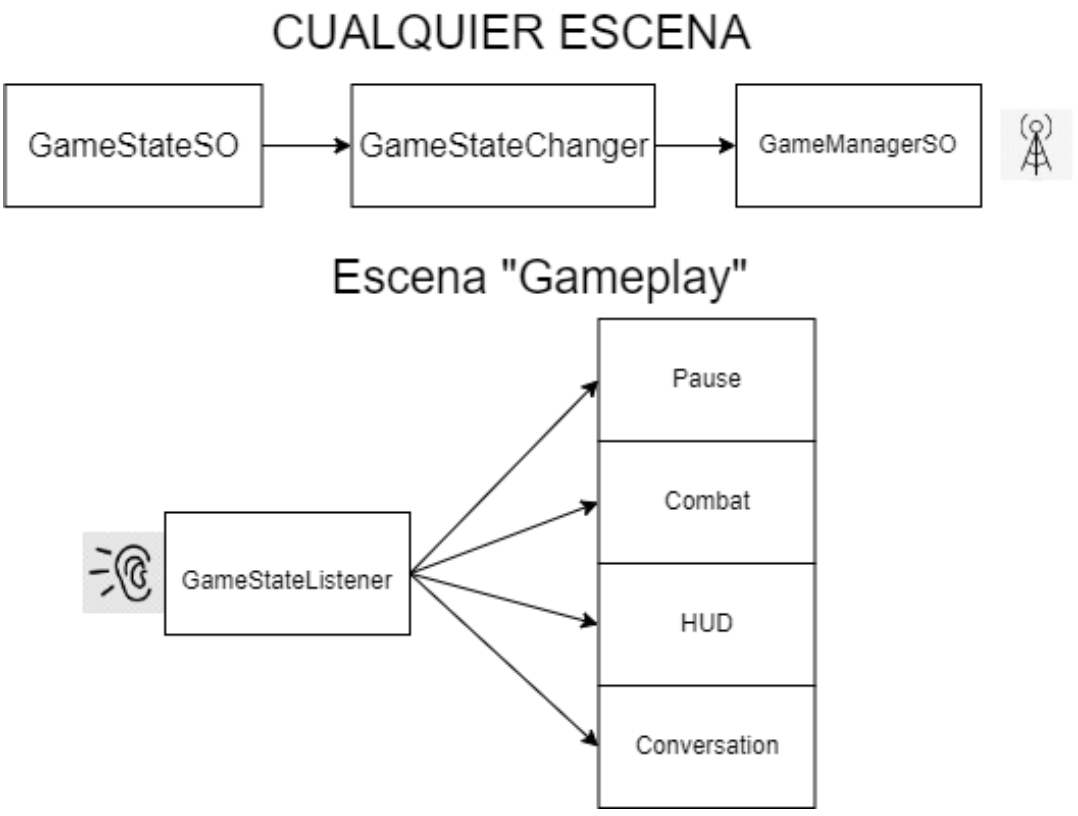

Figura 33: Esquema del manager de la UI

- **Canvas**: Este componente contiene todas las interfaces dentro del juego y, como se explicó en el punto anterior, dependiendo del estado del juego se activarán algunas y se desactivarán otras.

3.3.3. Escenas de juego

Figura 34: Escenas de juego

La estructura de GameObjects de imagen es la utilizada en todas las escenas de juego. Las **CONFIG**, **ENVIRONMENT** y **GAME** están solamente para separar unas de otras.

- **GameCameras**: Contiene todo lo relacionado con la cámara, además de un confiner para que no salga del mapa. En el proyecto se usa *Cinemachine*.

- **LevelInitializer**: Contiene la lógica que debe de ser ejecutada al inicializar la escena. Existe un script que carga las escenas **Gameplay** y **Managers** y otro que se encarga de la instancia del jugador. Para la instancia del jugador, este es pasado PlayerPathSO, para saber en qué entrada ponerlo. Además, cambiará el GameStateSO a "Playing". Este *gameObject* tiene un hijo que es la entrada por defecto, si existe un error, o se quiere testear el nivel directamente, el jugador se instanciará en ese *Trasform*.

- **Grid**: Contiene el *Tilemap* de la escena.

- **Entrances&Exits**: Contiene, como su nombre indica, las ubicaciones de las entradas y salidas de la escena.
    - **Entrances**: Se coloca en la escena, hay que tener en cuenta que su *Transform* es lo que importa, dado que el personaje será instanciado ahí. También se le asignará un LevelEntranceSO.
    - **Exits**: Contienen *Box Collider 2D* y esperarán alguna colisión. En caso de que se produzca se activará el script de cambio de escenas, el cual llamará al manager de cambio de escenas de **Managers** y le pasará la escena que debe ser cargada y la entrada donde se instanciará el personaje (EntranceSceneToLoadSO).

- **InteractionWithObjects**: Serán colocados objetos interactivos que no sean los NPC, como por ejemplo, letreros.

- **Trees**: Los árboles del nivel no pertenecen al *Tilemap*, sino que son objetos prefabricados y están como hijos de este *GameObject*.

- **NPC**: Todos los NPC de la escena están como hijos de este *GameObject*.

- **Houses**.: Igual que en los 2 casos anteriores, pero esta vez con las casas.

- **PlayerParent**: Es un *GameObject* vacío. A la hora de instanciar al jugador, se hará como hijo de este *GameObject*.

- **Events**: Es el *GameObject* cuyos hijos serán los eventos de la escena (como, por ejemplo, zona de descanso, enemigos estáticos, etc). Antes se comentó que los InGameEventSO cuando se marcaban a *true*, activaban una serie de repercusiones (por ejemplo, eliminar un elemento de la escena, cambiar conversaciones, etc). En este *GameObject* existe un script que al cargar el nivel los activa. Pero lo explicado anteriormente, solo es al entrar en la escena (Start()), los eventos tienen su propio script que cuando se

cumplan, llaman a esas repercusiones y así se hacen al momento.

3.3.4. Escenas de combate

Figura 35: Escenas de combate

Existen algunos puntos que son iguales a las escenas de juego y no se repetirá la explicación.

- **LevelInitializer**: En este caso, cargará igual que antes las escenas **Gameplay** y **Managers**, pero esta vez tiene un script que llamará al manager de los combates, pasándole así las unidades de combate enemigas que se quieran usar. Además,

el estado del juego será cambiado a “Combat” para que la interfaz sea cambiada a la adecuada.

- **EnemyPosition** y **PlayerPosition**: Es un *GameObject* vacío ubicado en los lugares donde se quiera colocar al jugador y al enemigo.

3.3.5. Escena “MainMenu”

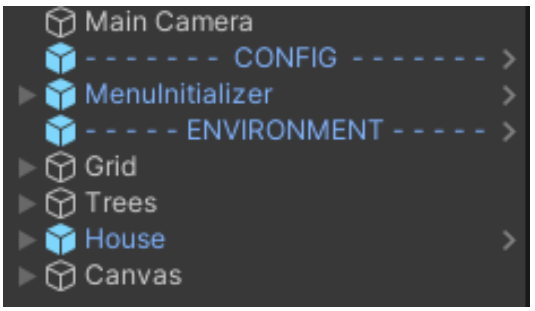

Figura 36: Escenas “MainMenu”

Lo nuevo de esta escena es el canvas con las distintas opciones, donde la opción de “Jugar”, llamará al manager de carga de escenas y cargará la escena correspondiente. Mientras que si se pulsa “Salir”, se cerrará el juego. El resto de cosas es equivalente solo al escenario del pueblo, porque se implementó un script para que, de fondo, la cámara se vaya moviendo por él.

4. Conclusiones y líneas futuras

En este proyecto, se ha logrado realizar un videojuego corto pero funcional, explorando campos interesantes como, por ejemplo, el combate por turnos. Todo lo construido se siente que está en una edad temprana y tiene potencial, pudiendo ser mejorado dedicándole tiempo. Gracias a los *Scriptable Objects*, se ha conseguido tener una estructura modular para los *managers*. En cuanto al uso de los eventos de unity, se ha logrado que casi todo pueda ser configurado y construido desde el propio inspector. Estas dos herramientas han sido un gran descubrimiento para mí y me han ayudado a crear una versión de juego muy innovadora a mi estilo de programación.

Las conclusiones personales sacadas tras acabar esta versión del videojuego han sido:

- Se ha ganado experiencia y soltura en el uso del motor de desarrollo de videojuegos Unity.
- Fue emocionante trabajar en un proyecto a mayor escala de lo que se está acostumbrado en el máster.
- Fue satisfactorio aprender el uso de nuevas herramientas como los *Scriptable Objects* y los eventos de Unity.
- Por último, fue conmovedor ver cómo evolucionó y se convirtió el proyecto en algo funcional.

Existen varias mejoras que se pueden añadir al proyecto e ideas que no dio tiempo de implementar.

Las mejoras a corto plazo (las cuales, cabe la posibilidad de que se presenten en la defensa) son:

- Añadir un botón de huida del combate.
- Añadir en el HUD, la salud, el maná, el nivel y un retrato de la cara.

Las mejoras a largo plazo serían:

- Reemplazar los sprites de los personajes por otros que sean más compatibles con el entorno y añadirles animaciones.
- Añadir animaciones al combate.
- Añadir que al derrotar a un enemigo, tenga un porcentaje de probabilidades de que suelte algún arma.
- Añadir habilidades de cura para usar el *HealingPower* de CombatUnitSO.
- Permitir que los enemigos usen habilidades y, así, darle uso a su barra de maná.
- Hacer uso de las imbuciones (como, por ejemplo, madera, plata, etc.) implementadas de las armas, creando una mecánica parecida a los tipos del juego de *Pokemon*.
- Continuar la historia.

5. Summary and Conclusions

In this project, a short but functional videogame has been made, exploring interesting fields such as, for example, turn-based combat. Everything built feels like it's at an early age and has potential, and can be improved by spending time on it. Thanks to the Scriptable Objects, it has been possible to have a modular structure for the managers. As for the use of unity events, almost everything can be configured and built from the inspector itself. These two tools have been a great discovery for me and have helped me create a very innovative game version of my programming style.

The personal conclusions drawn after finishing this version of the video game have been:

- experience and fluency have been gained in the use of the Unity game development engine.
- It was exciting to work on a project on a larger scale than you are used to in the master's degree.
- It was satisfying to learn the use of new tools such as Scriptable Objects and Unity events.
- Finally, it was moving to see how the project evolved and became functional.

There are several improvements that can be added to the project and ideas that did not take time to implement.

The short-term improvements (which, it is possible, that they will be presented in the defense) are:

- Add a combat escape button.

- Add in the HUD, health, mana, level and a portrait of the face.

Long-term improvements would be:

- Replace the character sprites with others that are more compatible with the environment and add animations to them.
- Add animations to combat.
- Add that when you defeat an enemy, you have a percentage chance that they will drop a weapon.
- Add healing abilities to use CombatUnitSO's HealingPower.
- Allow enemies to use abilities and thus put their mana bar to use.
- Make use of the imbues (wood, silver, etc.) implemented in weapons, creating a mechanic similar to the Pokemon game types.
- Continue the story.
    -

6. Bibliografía

[1] Juan Diego V. M.. Diseño avanzado de videojuegos RPG con Unity [MOOC]

https://www.domestika.org/es/courses/1761-diseno-avanzado-de-videojuegos-rpg-con-unity/course

[2] Unity Technologies. Unity Manual.

https://docs.unity3d.com/Manual/index.html

[3] Game assets. Itch.io.

https://itch.io/game-assets

[4] Unity Asset Store.

https://assetstore.unity.com/2d