Principado de Asturias Consejería de Educación	IES N 1 Gijón	**PROYECTO DAM**	Cofinanciado por la Unión Europea

INSTITUTO DE EDUCACIÓN SECUNDARIA NÚMERO 1 DE GIJÓN
FAMILIA PROFESIONAL DE INFORMÁTICA Y COMUNICACIONES

Ciclo Formativo Desarrollo de Aplicaciones Multiplataforma
2º Curso

Proyecto de Desarrollo de Aplicaciones Multiplataforma
Modalidad Presencial
Tipo: Desarrollo de aplicación

Eternal Eclipse

Videojuego al estilo Roguelike

Documento-Memoria

Autor/a: Ainhoa Lodeiro López
Fecha: 01/06/2024

ÍNDICE

1. Introducción

1.1. Presentación del alumno/a

Este Proyecto de Desarrollo ha sido realizado por Ainhoa Lodeiro López quien realiza el 2º curso del Ciclo de Grado Superior de Desarrollo de Aplicaciones Multiplataforma en el I.E.S Número 1 de Gijón, siendo mi tutor de proyecto Pablo Lera Menéndez y con mi tutora de ciclo Pilar García López.
A lo largo de esta memoria se mostrará todo lo relacionado al proyecto escogido por mí, desde una descripción general, hasta partes de código que serán interesantes a resaltar.

1.2. Título del proyecto y tipo de proyecto elegido

El proyecto se centra en el desarrollo de un videojuego titulado "Eternal Eclipse", que pertenece al género roguelike. Los juegos roguelike se caracterizan por su complejidad y los desafíos que presentan, basándose en combates por turnos. Una de las características distintivas de este género es la pérdida de progreso al morir, lo que obliga al jugador a reiniciar desde el comienzo.

"Eternal Eclipse" será desarrollado utilizando Unity, un motor de juegos ampliamente reconocido por su versatilidad y capacidad para crear experiencias inmersivas. Para garantizar una interfaz de usuario atractiva y funcional, se emplearán assets de alta calidad. Los scripts del juego se programarán en C#, aprovechando las capacidades del lenguaje para gestionar la lógica del juego y las interacciones del jugador.

2. Definición de proyecto

2.1. Descripción general

La definición de requisitos es esencial para el desarrollo del juego. En este apartado, asumiendo el rol de jefe de proyecto y analista, mi tarea es comprender a fondo las necesidades del juego y justificar la importancia de su desarrollo.

El juego es sencillo, surge como respuesta a la demanda de experiencias de juego accesibles y divertidas. Con la saturación del mercado de videojuegos, existe un nicho para juegos simples pero entretenidos que brindan diversión instantánea sin requerir una curva de aprendizaje complicada. La justificación principal del proyecto radica en ofrecer una alternativa refrescante y accesible para los jugadores casuales que buscan una experiencia de juego divertida y desafiante.

Además, puede atraer a aficionados de los juegos retro y a aquellos que disfrutan de desafíos simples pero estimulantes.

2.2. Descripción de requisitos

Requisitos funcionales:

1. Menú Principal

   Nada más abrir el juego se encuentra el menú principal con el título del juego, y tres botones, el primero para comenzar a jugar, el segundo para mostrar las mejoras de la última partida ganada, y, por último, el botón para poder cerrar el juego.
   También dentro de esta en la esquina superior derecha, tendremos un poco de información y una pequeña historia que tiene el juego para darle un poco de sentido a toda la situación dentro de este.

2. Puntuaciones

   Como se dijo anteriormente se podrá acceder a la pantalla de puntuaciones donde se podrán ver las mejoras realizadas en la última partida ganada.

3. Nivel Básico

   El juego se ve con una perspectiva de juego donde la cámara se encuentra situada perpendicular al suelo y, por tanto, ofrece una visión orientada hacia abajo, ha este tipo de vista se le nombra vista cenital, muy conocida en juegos de este estilo y otros.
   Dentro del juego tenemos un nivel básico de lo que sería el juego con la vista cenital, dentro encontraremos un mapa.
   Se puede apreciar una llanura en medio de un bosque, donde los árboles delimitan el mapa, mientras que las rocas actúan como obstáculos con los que el jugador puede chocar. De este modo, tanto el jugador como los enemigos solo podrán moverse dentro de la llanura.

4. Detalles de la pantalla

Se coloco una barra de vida que irá bajando en caso de que los enemigos ataquen al jugador.
Tambien tendremos dos etiquetas una para saber el número de oleada por el que va el jugador, y otro el temporizador para saber cuanto tiempo tiene que aguantar para poder conseguir la proxima mejora y poder pasar a la siguiente oleada.
En la última oleada aparecera otra barra de vida que pertenece al boss, para saber cuanta vida tiene ha medida que lo vas matando.
Otro detalle es que cuando empieza el juego comienza a sonar música.

5. Personaje principal

El personaje principal tiene una animación cuando esta quieto y otra de movimiento.
A su vez, en el caso de que se quede sin vida y pierda, cuenta con una animación de muerte.
Esta tira una bola de fuego que le servirá para matar a los enemigos, la cual tiene también su correspondiente animación cuando esta se lanza.

6. Enemigos

Tendremos enemigos que están controlados para que una vez aparezcan dentro del mapa comiencen a perseguir al personaje principal.
Cada enemigo cuenta con tres animaciones, una de movimiento para cuando sigue al jugador, otra de ataque, y otra de muerte.
Primero tendremos un esqueleto que se encargará de pegar al jugador cuando este a una cierta distancia. Después tendremos un mago que lanzará una habilidad hacia el jugador cuando este también a una cierta distancia, y, por último, el jefe final que hará lo mismo que el esqueleto con la única diferencia que este tendrá una barra de vida.

7. Pantallas durante el juego

Nada más empezar la partida nos aparecerá un menú con los controles básicos para que el jugador sepa como jugar, y una vez le dé el botón del play comenzará la partida (solo aparecerá cuando entres desde el menú de inicio y le des al play o reinicies el juego al completo).

He creado una pantalla de Pause para que el jugador pueda pausar el juego en cualquier momento (excepto cuando hay otras pantallas como las del Game Over, Victory, Mejoras o Controles). También desde esta podre ver las mejoras que esta haciendo, y tendrá botones para reiniciar, volver al menú o cerrar el juego.

También hay una pantalla de Game Over para cuando el personaje principal muere al quedarse sin vida. En esta pantalla, el jugador podrá elegir entre comenzar de nuevo, regresar al menú principal o cerrar el juego por completo.

Otra pantalla sería la de las mejoras, donde aparecerán tres botones y en ellos aparecerán las habilidades de mejora para el personaje principal, y a la izquierda

aparecerán las habilidades mejoradas. Para saber que hace cada mejora simplemente será necesario pasar el cursor por encima.

Una vez se mata al jefe la partida se da por ganada apareciendo así la pantalla confirmando que el jugador ha ganado.

Por último, desde el menú principal se puede ver las mejoras de esta última partida ganada.

3. Planificación del proyecto

3.1. Planificación de actividades y tareas. Temporización. Diagrama de Gantt

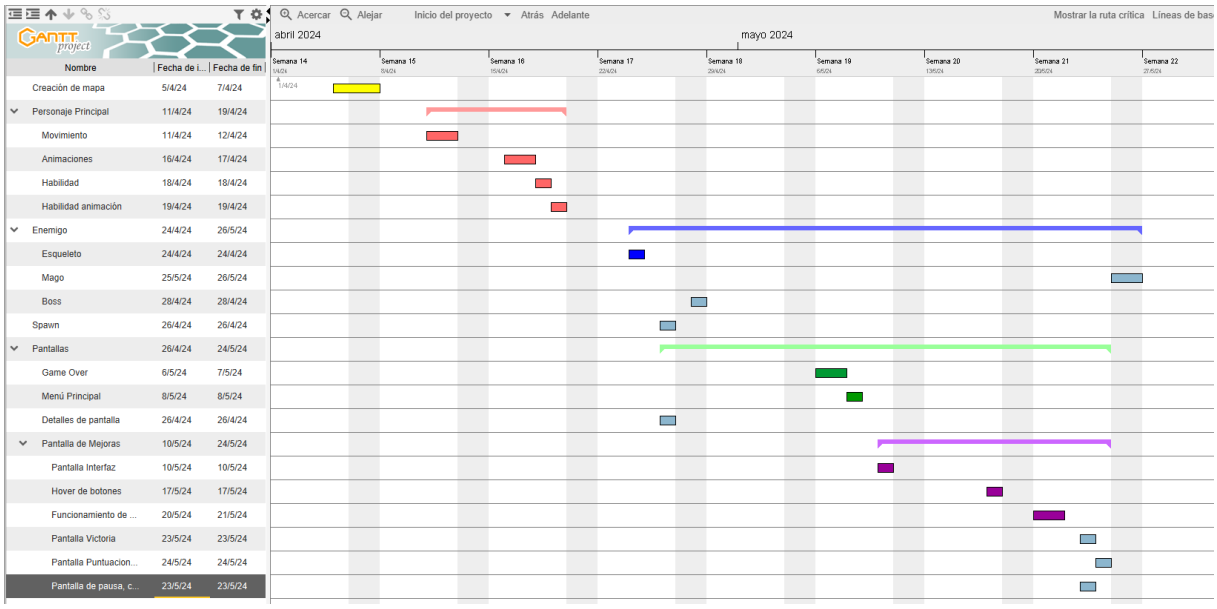


Diagrama de Gantt

1. **Creación de mapa (5/4/24 - 9/4/24):** Se desarrolla el mapa del juego.

2. **Personaje Principal:**

    - **Movimiento (11/4/24 - 12/4/24)**: Implementación de las mecánicas de movimiento del personaje principal. Esto incluye definir cómo el personaje se desplazará por el mapa. Se asegura que el movimiento sea fluido y responda adecuadamente a los controles del jugador.
    - **Animaciones (16/4/24 - 17/4/24):** Animaciones que deben sincronizarse con las mecánicas de movimiento para asegurar una experiencia de juego coherente.
    - **Habilidad (18/4/24):** Diseño e implementación de mecanica de la habilidad para el personaje.
    - **Habilidad animación (19/4/24):** Animaciones que deben sincronizarse con las mecánicas de movimiento.

3. **Enemigo:**

    - **Esqueleto (24/4/24**): Implementación del esqueleto enemigo.
    - **Mago (25/4/24 - 26/5/24):** Implementación del enemigo mago.
    - **Boss (28/4/24 - 28/4/24):** Implementación del jefe final.

4. **Spawn (26/4/24):** Implementación del sistema de aparición (spawn) de enemigos. Esto incluye definir cuándo y dónde aparecen los enemigos en el juego.

5. **Pantallas:**

    - **Game Over (6/5/24 - 7/5/24):** Diseño y desarrollo de la pantalla de Game Over. Esto incluye la creación del diseño visual y la programación de los elementos que se mostrarán al jugador cuando pierda el juego, como opciones para reiniciar, salir o cerrar el juego.
    - **Menú Principal (8/5/24):** Creación del menú principal del juego. Esto incluye el diseño de la interfaz y la programación de las opciones disponibles, como iniciar el juego, mirar las puntuaciones o salir.
    - **Detalles de Pantalla (26/4/24):** Se añade a la pantalla una barra de vida para el personaje principal, un temporizador y el número de oleada.
    - **Pantalla de Mejoras (22/4/24 - 17/5/24):**

- **Pantalla Interfaz (10/5/24):** Creación de la interfaz a nivel de diseño con los assets y el font.
- **Hover de botones (17/5/24):** Implementación de efectos de hover en los botones para saber que hace las mejoras que aparecen.
- **Funcionamiento de la pantalla de mejoras (22/4/24 - 23/4/24):** Programación de la funcionalidad a nivel de si las mejoras se colocan bien en el personaje principal y cumplen con su función.

- **Pantalla Victoria (23/5/24 - 23/5/24)**: Diseño y desarrollo de la pantalla de victoria, mostrando el éxito del jugador al completar el juego, mostrando a su vez todas las mejoras realizadas en esa partida.
- **Pantalla Puntuación (24/5/24 - 24/5/24):** Implementación de una pantalla de puntuaciones para que el jugador vea las mejoras realizadas en la última partida ganada.
- **Pantalla de pausa, controles e info (23/5/24 - 23/5/24)**: Diseño y control de las pantallas pausa, controles e info.

3.2. Estimación de costes

En este apartado se presenta la estimación de costes del coste total del proyecto tanto a nivel de personal como a nivel de equipo. El proyecto requiere la colaboración de varios perfiles profesionales (sobre todo a nivel de videojuegos), dentro del mundo de la programación y también a nivel de diseño, además, del uso de diferentes equipos y herramientas potentes.

Recursos humanos:

Perfil	**Tarifa por hora (€)**	**Esfuerzo (Horas)**	**Coste Final (€)**
Programador 1	35	200	7.000
Programador 2	35	200	7.000
Diseñador Gráfico 1	30	200	6.000
Diseñador Gráfico 2	30	200	6.000
Tester	20	100	2.000
Jefe de Proyecto	50	80	4.000

Analista	40	150	6.000
Total			38.000

Recursos materiales:

Recurso	**Descripción**	**Coste Unitario (€)**	**Cantidad**	**Coste Total (€)**
Software	Licencias de Unity Pro y Adobe Creative Cloud	1.800 + 810,84 (67,57 x 12)	1	2.610,84
Hardware	Ordenadores de alta gama	1.500	4	6.000
Otros	Servidores para pruebas y despliegue	1.000	1	1.000
Total				9.610,84

Total:

Recursos	**Coste total (€)**
Recursos humanos	38.000
Recursos materiales	9.610,84
Total	**47.610,84**

Esta estimación proporciona una visión clara de los costos asociados al desarrollo del proyecto, siendo el total general estimado en unos 47.610,84 €.

3.3. Previsión de riesgos del proyecto

Para la previsión de riesgos del proyecto, se tiene en cuenta la realización de copias de seguridad de manera regular. Esto es crucial en caso de fallos del software, problemas con los equipos, o pérdida de algún script o asset importante. Para ello, se utiliza Git para mantener un historial completo de todos los cambios del código fuente, acompañados de sus respectivos comentarios (commits) para documentar claramente los cambios realizados. Esta práctica permite rastrear fácilmente cualquier modificación y facilita la recuperación en caso de errores.

Además, es fundamental documentar el código a medida que se desarrolla. La documentación detallada asegura que todos los miembros del equipo comprendan el propósito y funcionamiento de cada script, minimizando los problemas de comprensión y facilitando el mantenimiento y futuras modificaciones del código.

También se sigue un cronograma de tiempo bien definido, que nos ayuda a mantener la organización y asegurar que todas las tareas se completen a tiempo. Esto es esencial para cumplir con las fechas de entrega y asegurar la finalización exitosa del proyecto.
Para ello se complementarán medidas como revisiones de código, pruebas automáticas, comunicación y feedback continuo, y seguridad de la información.

4. Análisis y Diseño de proyecto

4.1. Diseño y arquitectura

Durante el desarrollo del juego se han utilizado diferentes herramientas y tecnologías para así garantizar un proceso de desarrollo eficiente.
Los entornos de desarrollo principalmente utilizados han sido Unity como motor de juego por su rápido entendimiento para el desarrollo de videojuegos en 2D y que este ofrece grandes librerías y herramientas muy útiles y no muy complejas lo que hace que mejore la velocidad de producción.

Por otro lado, para el código se utiliza Visual Studio el cual sirve para describir y depurar el código en C#. He decidido utilizar este entorno ya que es el creo que mejor se integra con Unity ya que ofrece grandes herramientas de productividad y depuración avanzada (además de todas las extensiones que pueden ser de ayuda a la hora del desarrollo).

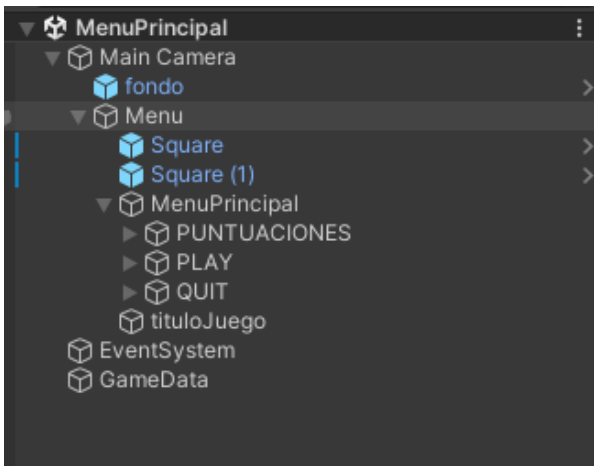
Como se dijo anteriormente el lenguaje que se está utilizando para el desarrollo del videojuego es C# el cual además de ser altamente compatible con Unity también permite una programación orientada a objetos eficiente y estructurada.

Al ser un proyecto que lleva tiempo y durante ese tiempo pueden ocurrir errores en el sistema o pérdida de datos se ha decidido utilizar Git a nivel de control de versiones para poder tener una copia de seguridad de todo el proyecto en el caso de que ocurra algo inesperado y/o se haya realizado algún cambio que no era necesario y se tenga que volver a lo que anteriormente teníamos.

Para el tema de los assets se han ido recogiendo de páginas como Unity Assets Store o, la que en su mayoría hemos utilizado, Itch.io, que es una página muy conocida donde la gente puede vender sus proyectos digitales, de videojuegos completos, a muchos sprites animados. Esta última cuenta con numerosos assets, creados por usuarios con muy buenas animaciones y diseños.

También he utilizado la página Dafont, que principalmente sirve para la descarga de múltiples fuentes para texto. De esta manera he logrado crear el título del juego y el de las puntuaciones, y de otras pantallas como la del Game Over y Victory
Todas estas herramientas han sido escogidas para asegurar un desarrollo eficiente y un producto final de alta calidad, siendo, cada decisión tomada considerando las necesidades especificas del proyecto y la experiencia del equipo de desarrollo.

Dentro del menú principal tenemos una estructura de proyecto que sería la siguiente:

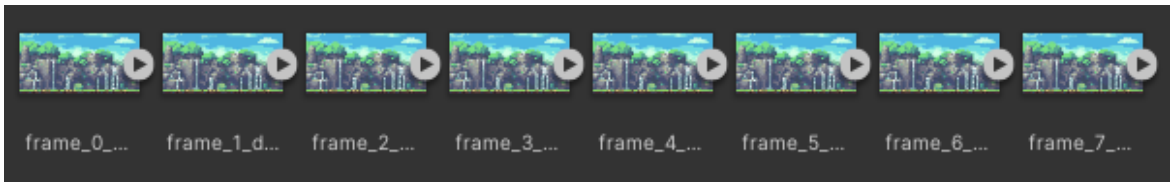


Jerarquía del Menú Principal

Siendo así como se vería:

Pantalla inicio

Dentro de la estructura tenemos el título "*Eternal Eclipse"* junto con el menú donde tenemos el botón de jugar, para comenzar el juego, el de puntuaciones para ver las puntuaciones guardadas de la última partida ganada, y, por último, para poder salir del juego.



Frames del fondo

Para los botones se usaron assets de Itchi.io lo cuales sirven para el diseño de interfaces UI.

Por otro lado, el gif fue descargado y animado, para que, cuando entres al juego el fondo se vea con movimiento.

Arriba a la derecha se puede ver un símbolo como de información que lleva a la siguiente pantalla donde se explica un poco la historia del juego.

Pantalla de información

Después tenemos el juego con la siguiente jerarquía:

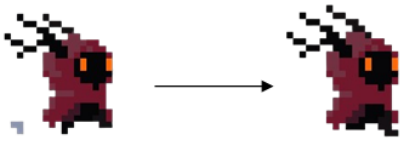
Jerarquía del juego

Viéndose de la siguiente forma:

Mapa y jugador

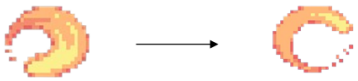
Primero, tenemos el *GameData,* el cual se encargará de guardar el número de oleada por el que vamos, para que así, al pasar a la siguiente oleada (se reinicie todo el juego), guarde por la oleada que vas, además de todas las mejoras que tiene que conservar el personaje mientras las oleadas van avanzando.
También añadí un audio para que suene mientras el jugador está jugando.

Segundo, tenemos lo que serian los detalles de la pantalla donde tenemos la barra de vida que es la barra roja con un corazón, el texto con el número de oleada y un temporizador que maneja cuanto tiempo tenemos que sobrevivir para poder conseguir la siguiente mejora.



Jugador en movimiento

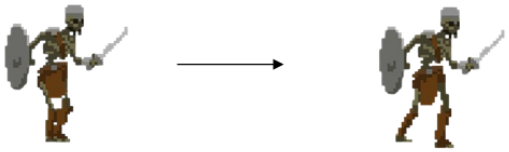
Luego tendríamos el personaje principal el cual esta diseñado para que se pueda mover por todo el mapa excepto por los árboles, ya que estos delimitan el mapa, y las rocas, que hacen de obstáculo para el jugador. Además, este cuenta con una animación de movimiento, una habilidad que lanza para poder matar al enemigo, y la animación en caso de que el jugador pierda toda la vida



Habilidad Jugador

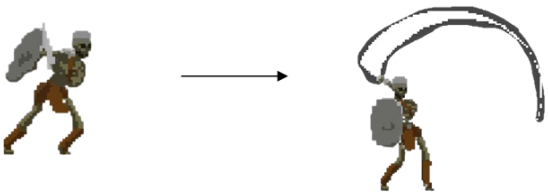
Muerte del jugador

Para la aparición de los enemigos se utilizará el spawn, del cual cada cierto 3 segundos aparecerá un enemigo esqueleto o un mago al cual tendremos que matar para que este no nos mate a nosotros, ya que una vez aparezca nos empezará a perseguir por todo el mapa.



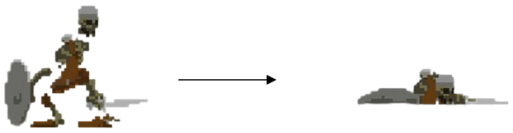
Esqueleto en movimiento

El esqueleto por una parte para poder hacernos daño se tendrá que acercar bastante, y cuando este a la distancia adecuada comenzará la animación de ataque.



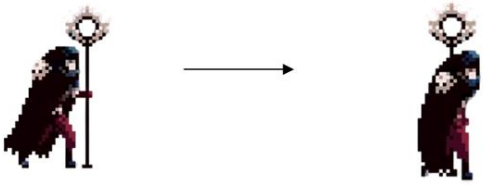
Esqueleto atacando

En caso de que el jugador le lance la bola de fuego en su dirección y le logré dar, entonces el enemigo morirá y hará una animación de muerte.

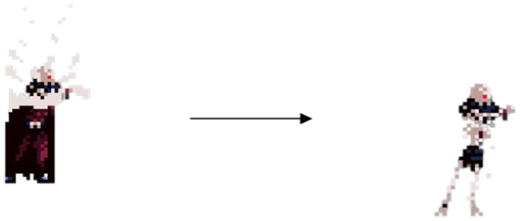


Muerte del esqueleto

Dentro del spawn también puede aparecer un mago, el cual, como el esqueleto, tiene animación de movimiento y muerte.

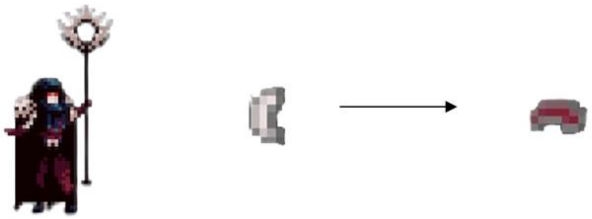


Mago en movimiento



Mago muerto

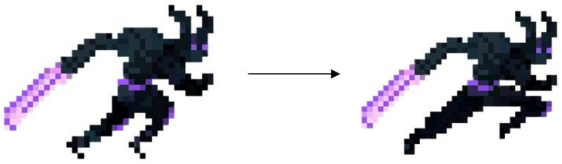
El mago, evidentemente, también podrá atacar, pero lo hará de una forma diferente al esqueleto, y es que este lanzará una habilidad en dirección del jugador.



Mago atacando y habilidad de mago

Una vez lleguemos a la oleada 5 y el temporizador llegue a cero, entonces aparecerá el jefe final.

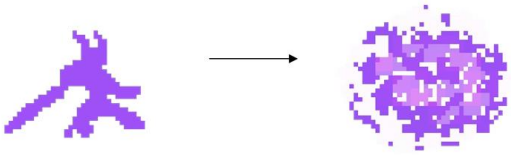
Este tiene animación de movimiento ya que también seguirá al jugador y animación de ataque.



Jefe en movimiento

Jefe atacando

Tendremos por otro lado la animación de muerte. Este solo morirá cuando su barra de vida llegue a cero.

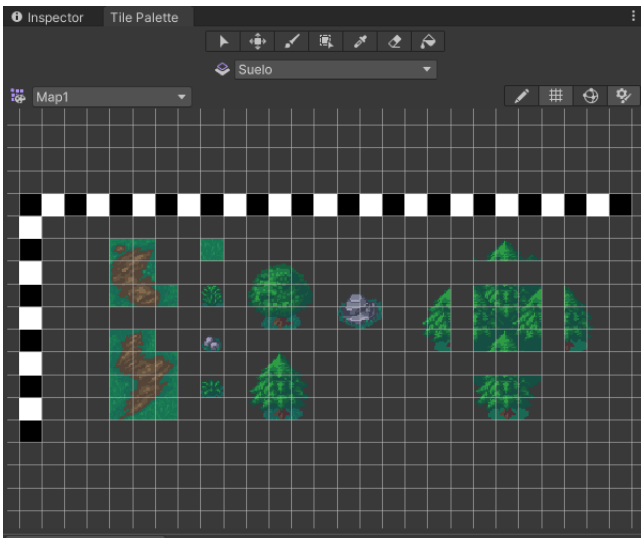


Jefe muerte

Vida del jefe

Como se nombró anteriormente el mapa está dividido en tres, el suelo donde estará y se podrá mover el jugador y los enemigos, los árboles que delimitan el mapa junto con las rocas que sirven de obstáculo para el jugador, y luego un apartado de detalles ya sea para decoración y arreglo de algunos bugs o incompatibilidades al poner los sprites.

Todo ello fue trabajado con el TileMaps, este sirve como herramienta para cargar un conjunto de “baldosas” (tileset), crear una paleta a partir de ese conjunto y poder pintar el nivel como si fuera una cuadrícula.



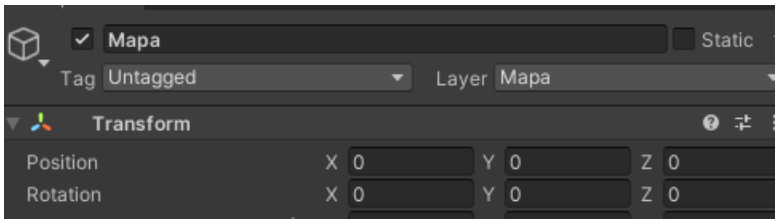
TileMaps del mapa

Una parte para nombrar importante es, además, del cambio de capas a los diferentes objetos dentro de la escena, como poner todo lo referido al mapa en las capas en

números negativos para que esto fuera el fondo y todo lo que este del cero hacia adelante este colocado todo por delante del mapa. También está el tema de las colisiones específicamente en las capas, ya que en este juego he buscado de que el jugador tenga varios obstáculos, como las piedras, y los árboles, con los cuales choca, y a su vez estos delimiten el mapa, pero que con los enemigos no tenga ningún problema de colisiones, para ello se ha añadido al apartado de Layer tres etiquetas, el mapa, el enemigo y el ataque del jugador.

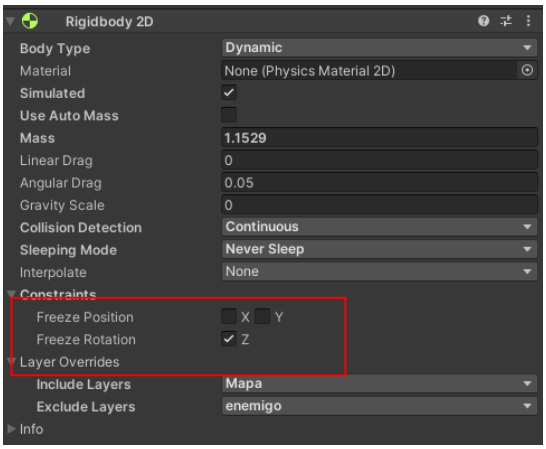
Layers

Estas capas serán referidas al objeto concreto como por ejemplo el del mapa



GameObject Mapa

Ahora dentro del jugador en el apartado de "*RigidBody2D*" colocamos las capas que quiero que choque y las que quiero que excluya.
De esta manera las que están incluidas son con las que chocará, y las excluidas, las que no tendrán físicas de choque.



RigidBody2D del jugador

Nada más comenzar a jugar se puede observar

la pantalla de controles, en ella, el título es escrito con un Font descargado previamente llamado pixelart, con el botón de abajo, podremos comenzar el juego.

Pantalla de controles

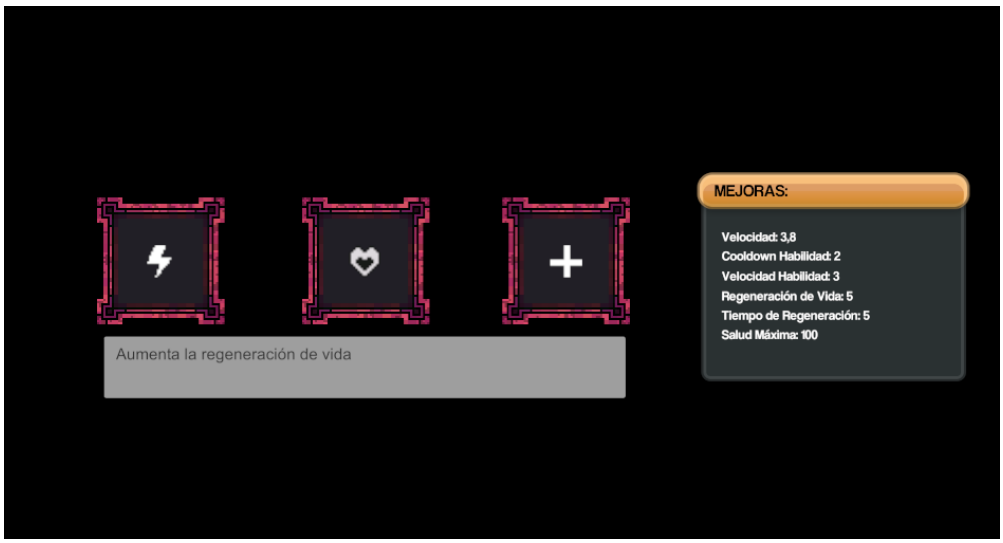
Otra pantalla estará siempre presente para el jugador cuando le de a la tecla de escapa para poder pausar el juego cuando quiera.

Pantalla de pausar

Se cuenta también con una pantalla para cuando el personaje muera.

Pantalla de Game Over

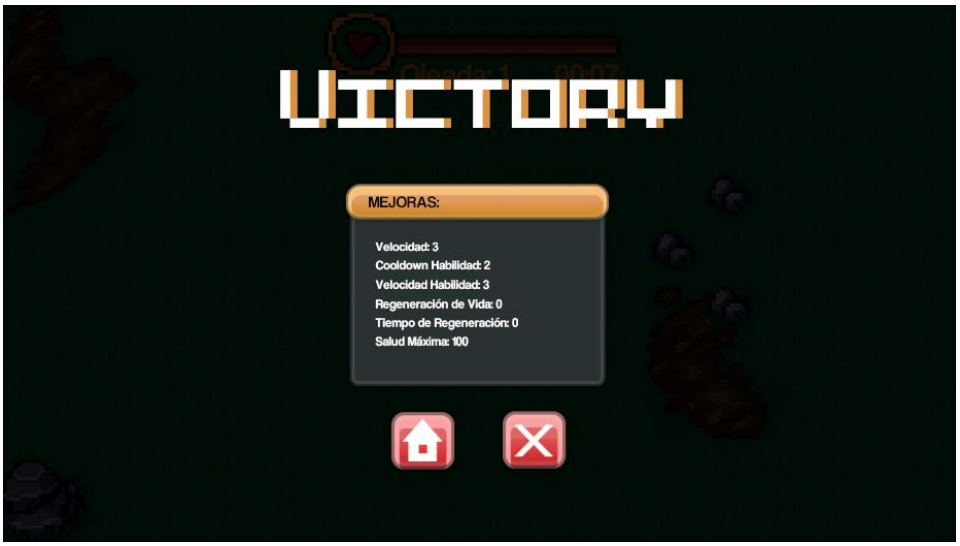
Los botones son del mismo paquete que los del menú, en el primero reiniciamos partida, en el segundo volvemos al menú y en el último cerramos el juego.



Pantalla de mejoras

Después tendríamos la pantalla de mejoras. Para la interfaz se han usado assets como los de las habilidades para tener una imagen de cual es cada habilidad, pero a su vez si pasas el ratón por encima te encontraras con un texto que explica el tipo de mejora, y a su derecha estaría todo el resumen de las mejoras las cuales irían cambiando a medida que el jugador escoja estas.

Una vez mates al jefe final conseguirás la victoria, donde te aparecerán todas las mejoras realizadas, y los botones para ir al menú principal o cerrar el juego.



Pantalla de Victory

Una vez vuelvas al menú principal, puedes ir al apartado de “Puntuaciones” y te aparecerá el resumen de las mejoras realizadas en la última partida ganada.

Pantalla de mejoras de la última partida ganada

4.2. Modelo de datos

He utilizado dos formas de guardar datos, siendo cada una de ellas para diferentes utilidades.

La primera ha sido a través de PlayerPrefs, una forma de guardar datos que se usa mucho dentro de Unity, la cual permite guardar y cargar datos del tipo int, float o string, y que estos datos a su vez se puedan repartir entre las diferentes escenas, a través de un objeto llamado GameData.

Esto nos servirá para guardar los datos de las mejoras DURANTE la partida, cada vez que el jugador escoja una mejora, esta será guardada y mejorada dentro del juego. Una vez el juego termine, ya sea porque ha ganado o perdido, estas mejoras serán reiniciadas a como están predefinidas desde un inicio.

El script requiere de un "*instace*", esto se refiere a que si hay otra instancia de esta clase (PlayerPrefsSettings) hace que se convierta en una instancia única y se asegura que no se pueda destruir entre escenas, en el caso de que ya exista una instancia con ese nombre entonces se destruye ese objeto.

```
2 references
public static PlayerPrefsSettings instance;
```

Instance del GameData

```
0 references
void Awake()
{
    // Se mira si la intstancia existe sino se crea una
    if (instance == null)
    {
        instance = this; // Intancia única
        DontDestroyOnLoad(gameObject); // Para que no se elimine si se cambia de escena
        this.Clear(); // Limpia las variables de mejoras cuando reinicia el juego

        // Para guardar los datos en archivo json
        Filepath = Path.Combine(Application.persistentDataPath, "gamedata.json");
    }
    else
    {
        Destroy(gameObject); // Destruye la instancia si esta ya existe
    }
}
```

Awake del GameData

Todo ello está dentro del método "*Awake()*" en el cual se inicia antes de todos los start de los scripts.

El método "*Clear()*" es que al que se llama para poder reiniciar todas las mejoras cuando el jugador reinicie la partida.

Dentro de estos datos guardamos, por un lado, todas las mejoras (velocidad, cooldown de habilidad, velocidad de habilidad, regeneración de vida, tiempo de regeneración de vida y salud máxima). Como todas ellas se guardan más o menos de la misma manera (aunque puede cambiar el tipo de dato), pondré de ejemplo la velocidad.

```
1 reference
public float getVelocidadHabilidad()
{
    return velocidadHabilidad = PlayerPrefs.GetFloat("velocidadHabilidad", 3f);
}

1 reference
public void setVelocidadHabilidad(float velocidadHabilidad)
{
    PlayerPrefs.SetFloat("velocidadHabilidad", velocidadHabilidad);
}
```

Get y Set de la velocidad del jugador

En estos dos métodos primero recogemos la velocidad para, por ejemplo, colocársela al jugador

```
Vector3 movimiento = new Vector3(movimientoHorizontal, movimientoVertical, 0f) * PlayerPrefsSettings.instance.getVelocidadPersonaje() * Time.deltaTime;
```

Velocidad del jugador

y en el segundo se coloca la nueva velocidad, que es cuando se hace la mejora de esta.

```
void MejorarVelocidad()
{
    PlayerPrefsSettings.instance.setVelocidadPersonaje(PlayerPrefsSettings.instance.getVelocidadPersonaje() + 0.8f);
    proximaOleada();
}
```

Mejora velocidad del jugador

Así es con todas las mejoras colocándolas en el lugar que corresponde cada una.
Otro dato también importante pero no dirigido a las mejoras es el número de oleada que el jugador está jugando.

```
0 references
public int getNumOleada()
{
    return PlayerPrefs.GetInt("numOleada", 1);
}

1 reference
public void setNumOleada(int numOleada)
{
    PlayerPrefs.SetInt("numOleada", numOleada);
}
```

Get y set del número de oleada

Donde los metodos son lo mismo que lo explicado anteriormente, pero este es dirijido a guardar el valor de la oleada para cuando se reinicie el juego se entienda que va por la siguiente oleada (este número cambiará una vez escogida una mejora, entendiendose asi que el jugador la ha superado).
Por otro lado, como se nombró anteriormente hay otra forma de guardar información del juego, siendo a traves de los archivos json.
En este tipo de juegos es muy importante las mejoras que has hecho sobre todo en las partidas ganadas ya que estas nos sirven para darnos cuenta que mejoras nos han venido bien para ganar la partida. Es por ello, que he decidido guardar las mejoras de la última partida ganada para poder verlas antes de comenzar a jugar de nuevo.
La forma de guardar los datos ha sido primero creado una clase que se llama última partida donde guardo todos los datos que quiero guardar en el json.

```
[System.Serializable]
5 references
public class ultimaPartida
{
    1 reference
    public float velocidadPersonaje;
    1 reference
    public float cooldownHabilidad;
    1 reference
    public float velocidadHabilidad;
    1 reference
    public float regeneracionVida;
    1 reference
    public float tiempoRegeneracionVida;
    1 reference
    public float saludMaxima;
}
```

Clase ultimaPartida

El “*System.Serializable*” sirve para poder guardar datos localmente y poder acceder a ellos y reconstruirlos.
Primero para guardar los datos se crea un método, que dentro de el se llamará a la anterior clase creada, y se recogeran todos los valores del PlayerPrefs. Una vez se tengan esos datos, se pasaran a formato json, y despues todo ese formato a un archivo que se guardará dentro del proyecto.

```
0 references
public void guardarJson()
{
    // Clase con los atributos que se van a guardar
    ultimaPartida ganador = new ultimaPartida();

    ganador.velocidadPersonaje = getVelocidadPersonaje();
    ganador.cooldownHabilidad = getCooldownHabilidad();
    ganador.velocidadHabilidad = getVelocidadHabilidad();
    ganador.regeneracionVida = getRegeneracionVida();
    ganador.tiempoRegeneracionVida = getTiempoRegeneracionVida();
    ganador.saludMaxima = getSaludMaxima();

    // Se pasa todo el objeto a json
    string playerToJson = JsonUtility.ToJson(ganador);

    //Se guarda el json en una ruta
    File.WriteAllText(Filepath, playerToJson);
}
```

Método para guardar en Json

C: > Users > Ainhoa > AppData > LocalLow > DefaultCompany > EternalEclipse > {} gamedata.json > ...

```
{
    "velocidadPersonaje": 3.7999999523162284,
    "cooldownHabilidad": 2.0,
    "velocidadHabilidad": 4.0,
    "regeneracionVida": 1.0,
    "tiempoRegeneracionVida": 2.5,
    "saludMaxima": 105.0
}
```

Datos guardados en archivo json

Una vez guardado tendremos que hacer otro para poder cargarlo, para ello usamos este siguiente método

```
// Metodo para cargar los datos guardados en json
0 references
public void cargarJson()
{
    if (File.Exists(Filepath)) // Si el archivo existe
    {
        // Lee el archivo y lo pasa de json a la clase ultimaPartida
        string json = File.ReadAllText(Filepath);
        ganadorJson = (ultimaPartida)JsonUtility.FromJson(json, typeof(ultimaPartida));
    }
```

Método para guardar en Json

Este nos servirá para hacer la inversa de lo anterior, es decir, leer el archivo guardado anteriormente y pasar los datos del json a la clase ultimaPartida.

Para saber cuál es el path en el "*Awake()*" se coloco lo siguiente:

```
// Para guardar los datos en archivo json
Filepath = Path.Combine(Application.persistentDataPath, "gamedata.json");
```

Ubicación del archivo Json

Esta es la forma de guardarlos, ahora queda saber en que momento se guardan, y en qué momento se cargan.

Estos datos son guardados cuando derrotamos al jefe final, es decir, una vez a este se le termine la vida llamaremos a un método que además de mostrarnos la pantalla de victoria, a su vez, nos administrara todos los datos del juego.

```
0 references
void seMuere()
{
    Destroy(gameObject);
    PlayerPrefsSettings.instance.guardarJson();
    pantallaVictory.SetActive(true); // Muestra el canvas de victoria
    Time.timeScale = 0f; // Detén el tiempo del juego
}
```

Método de muerte del jefe

Para cargarlos, se utiliza otro script donde se cargan y se colocan todos los datos en el texto correspondiente de la pantalla de puntuaciones, ya que desde ahí podrá el jugador ver, como ya se dijo al inicio, todas las mejoras realizadas, en la última partida ganada.

```
0 references
public class mejorasUltimaPartidaGanador : MonoBehaviour
{
    1 reference
    public Text velocidad;
    1 reference
    public Text cooldownHabilidad;
    1 reference
    public Text velocidadHabilidad;
    1 reference
    public Text regeneracionVida;
    1 reference
    public Text tiempoRegeneracionVida;
    1 reference
    public Text saludMax;

    // Start is called before the first frame update
    0 references
    void Start()
    {
        var gameData = PlayerPrefsSettings.instance;
        gameData.cargarJson();
        Debug.Log("Se cargaron los datos" + gameData.ganadorJson.velocidadPersonaje.ToString());
        velocidad.text = "Velocidad: " + gameData.ganadorJson.velocidadPersonaje.ToString();
        cooldownHabilidad.text = "Cooldown habilidad: " + gameData.ganadorJson.cooldownHabilidad.ToString();
        velocidadHabilidad.text = "Velocidad habilidad: " + gameData.ganadorJson.velocidadHabilidad.ToString();
        regeneracionVida.text = "Regeneración vida: " + gameData.ganadorJson.regeneracionVida.ToString();
        tiempoRegeneracionVida.text = "Tiempo de regeneración de vida: " + gameData.ganadorJson.tiempoRegeneracionVida.ToString();
        saludMax.text = "Salud Máxima: " + gameData.ganadorJson.saludMaxima.ToString();
    }
```

Clase que muestra las mejoras

Viéndose así la pantalla final con los datos guardados:

Pantalla de mejoras

4.3. Modelo de procesos

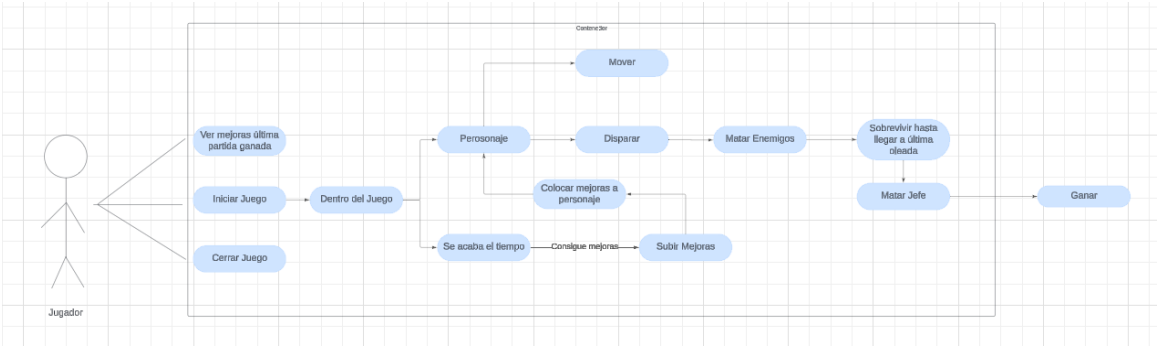


Diagrama general

Este diagrama de flujo describe el ciclo de todo el juego y las propias interacciones del jugador dentro de él.

Nada más empezar el jugador tiene tres opciones iniciales, ver las mejoras de la última partida ganada, cerrar el juego o iniciar una partida.

Una vez inicia la partida tiene un personaje, que se puede mover y disparar a los enemigos para matarlos y así sobrevivir, a su vez, mientras este juega habrá un temporizador que una vez acabe el tiempo le dará a escoger entre tres mejoras aleatorias, y una vez escogida una pasara a la siguiente oleada, y se repetirá el ciclo hasta la última, donde tendrá que matar al boss para ganar la partida.

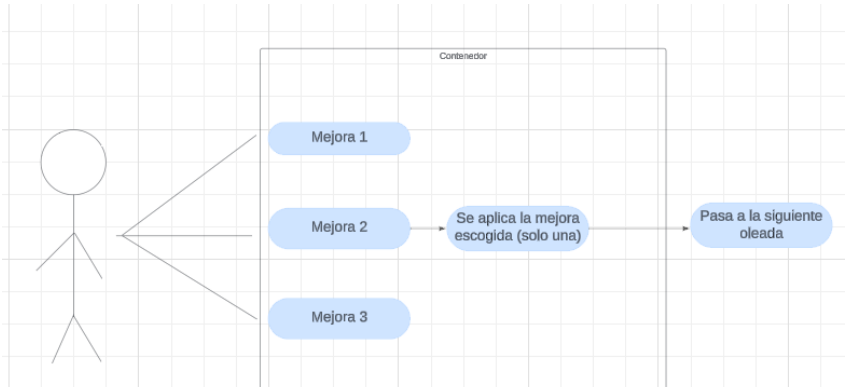


Diagrama especifico de funcionamiento de mejoras

Después tenemos el diagrama de caso de uso en especifico a una funcionalidad, en este caso las mejoras, donde el jugador tendrá para escoger entre tres mejoras completamente aleatorias y diferentes entre así, pudiendo escoger solo una, y pasando así a la siguiente oleada.

5. Construcción del proyecto.

5.1. Codificación.

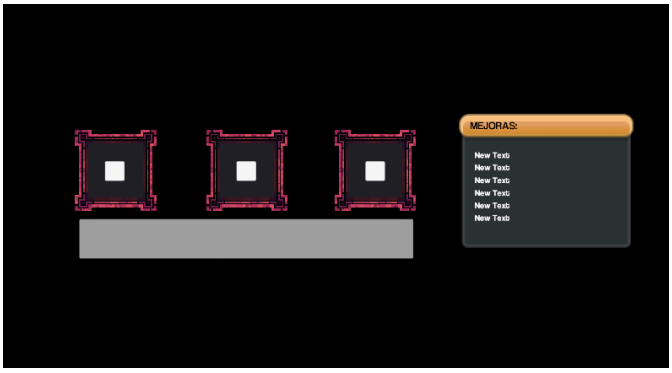
Para este apartado me gustaría dar énfasis en un script en concreto que es el que controla las mejoras, ya que es algo en lo que me he centrado mucho en que salga y quede bien

```
public class MejoraManager : MonoBehaviour
{
    1 reference
    public GameObject pantallaMejoras; // Pantalla donde se van a mostar las mejoras
    10 references
    public Button[] botonesMejora;   // Botones que tendrán las mejoras
    1 reference
    public GameObject tooltipPanel; // El panel que actúa como el tooltip

    1 reference
    public Text tooltipText; // El texto dentro del panel que muestra la descripción de la mejora

    8 references
    private List<Mejora> mejorasDisponibles = new List<Mejora>(); // Lista con todas las mejoras

    6 references
    public Sprite[] abilitySprites;
```

Atributos de la clase de MejorasManager

Dentro de este script tenemos primero las variables como la pantalla de mejoras para saber donde mostrar las mejoras, los botones donde irán las mejoras para poder seleccionarlas, un tooltipPanel para mostrar una breve descripción de lo que hace la habilidad, y el texto que aparecerá, luego la lista con todas las mejoras que se inicializará nada más aparezca la pantalla, y por último la lista de sprites para colocar según la mejora.

Nada más aparezca el canvas con la interfaz de las mejoras:



Pantalla de mejoras

```
0 references
void Start()
{
    // Inicializar las mejoras
    mejorasDisponibles.Add(new Mejora("Velocidad", "Aumenta la velocidad del personaje", MejorarVelocidad));
    mejorasDisponibles.Add(new Mejora("Cooldown Habilidad", "Reduce el cooldown de la habilidad", MejorarCooldownHabilidad));
    mejorasDisponibles.Add(new Mejora("Velocidad Habilidad", "Aumenta la velocidad de la habilidad", MejorarVelocidadHabilidad));
    mejorasDisponibles.Add(new Mejora("Regeneración de Vida", "Aumenta la regeneración de vida", MejorarRegeneracionVida));
    mejorasDisponibles.Add(new Mejora("Tiempo de Regeneración", "Reduce el tiempo de regeneración de vida", MejorarTiempoRegeneracionVida));
    mejorasDisponibles.Add(new Mejora("Salud Máxima", "Aumenta la salud máxima", MejorarSaludMaxima));

    // Asignar mejoras aleatorias a los botones
    AsignarMejorasAleatorias();
}
```

Método Start() de MejorasManager

Se inicializará el siguiente script de la siguiente manera:

Se colocarán todas las mejoras dentro de la lista, con el nombre de la habilidad, su descripción y para poder aplicar dicha mejora.

Para ello se utiliza la clase Mejora que cuenta con el siguiente script, donde se puede ver toda la información recogida de cada habilidad.

```
public class Mejora
{
    1 reference
    public string nombre; // Nombre de la mejora
    1 reference
    public string descripcion; // Descripción de la mejora
    1 reference
    public System.Action aplicarMejora; // Método que aplica la mejora

    0 references
    public Mejora(string nombre, string descripcion, System.Action aplicarMejora)
    {
        this.nombre = nombre;
        this.descripcion = descripcion;
        this.aplicarMejora = aplicarMejora;
    }
}
```

Clase Mejora

```
void AsignarMejorasAleatorias()
{
    List<Mejora> mejorasSeleccionadas = new List<Mejora>();

    while (mejorasSeleccionadas.Count < 3) // Se repite hasta que tenga 3 mejoras
    {
        int index = Random.Range(0, mejorasDisponibles.Count);
        Mejora mejora = mejorasDisponibles[index];
        if (!mejorasSeleccionadas.Contains(mejora)) // Si contiene esa mejora ya no es valido
        {
            mejorasSeleccionadas.Add(mejora); // Añade la mejora a las mejoras seleccionadas
        }
    }
```

Método asignar mejoras aleatorias

```
for (int i = 0; i < 3; i++) // Bucle para colocar en los botones las tres mejoras
{
    string nombreHabilidad = mejorasSeleccionadas[i].nombre;
    Sprite habilidadSprite = null;
    switch (nombreHabilidad) // Según el nombre de la habilidad se aplica dicho metodo y se coloca el sprite correspondiente
    {
        case "Velocidad":
            botonesMejora[i].onClick.AddListener(MejorarVelocidad);
            habilidadSprite = abilitySprites[0];
            break;
        case "Cooldown Habilidad":
            botonesMejora[i].onClick.AddListener(MejorarCooldownHabilidad);
            habilidadSprite = abilitySprites[1];
            break;
        case "Velocidad Habilidad":
            botonesMejora[i].onClick.AddListener(MejorarVelocidadHabilidad);
            habilidadSprite = abilitySprites[2];
            break;
        case "Regeneración de Vida":
            botonesMejora[i].onClick.AddListener(MejorarRegeneracionVida);
            habilidadSprite = abilitySprites[3];
            break;
        case "Tiempo de Regeneración":
            botonesMejora[i].onClick.AddListener(MejorarTiempoRegeneracionVida);
            habilidadSprite = abilitySprites[4];
            break;
        case "Salud Máxima":
            botonesMejora[i].onClick.AddListener(MejorarSaludMaxima);
            habilidadSprite = abilitySprites[5];
            break;
        default:
            Debug.LogWarning("Habilidad no existe: " + nombreHabilidad);
            break;
    }
```

Bucle para recoger las mejoras aleatorias y colocarlas

Primero se realiza la búsqueda de un número random aleatorio y el número escogido es la posición dentro de la lista y de esta forma se escogen mejoras aleatorias, y se meten dentro de una nueva lista que sería las mejoras seleccionadas.

Después dentro de esta lista se mira el nombre de la habilidad, se mejora esa habilidad escogida y se coloca el sprite en el botón para que sea algo más visual.

Un ejemplo de cómo se coloca la habilidad escogida sería el de la velocidad del jugador

```
2 references
void MejorarVelocidad()
{
    PlayerPrefsSettings.instance.setVelocidadPersonaje(PlayerPrefsSettings.instance.getVelocidadPersonaje() + 0.8f);
    proximaOleada();
}
```

Método para mejorar velocidad del jugador

Por otro lado, además de colocar el sprite sobre el botón, para que al pasar sobre el botón se vea la descripción se añade esta al panel y con el texto correspondiente.

```
botonesMejora[i].image.sprite = habilidadSprite; // Se coloca el sprite
botonesMejora[i].gameObject.AddComponent<BotonesHover>().tooltipPanel = tooltipPanel; // Panel para mostrar descripción de mejora
botonesMejora[i].gameObject.GetComponent<BotonesHover>().tooltipText = tooltipText; // Tooltip con la descripción de la mejora
botonesMejora[i].gameObject.GetComponent<BotonesHover>().upgradeDescription = mejorasSeleccionadas[i].descripcion; // Se coloca dic
```

Se coloca la descripción de la habilidad en el TooltipPanel

Aquí se puede ver la clase:

```
public class BotonesHover : MonoBehaviour, IPointerEnterHandler, IPointerExitHandler
{
    3 references
    public GameObject tooltipPanel; // El panel que actúa como el tooltip
    2 references
    public Text tooltipText; // El texto dentro del panel que muestra la descripción de la mejora
    1 reference
    public string upgradeDescription = ""; // La descripción de la mejora

    0 references
    public void OnPointerEnter(PointerEventData eventData)
    {
        if (tooltipPanel != null && tooltipText != null) // Si da error en caso de quitar el cursor para no de error de null
        {
            tooltipPanel.SetActive(true); // Se activa el hover
            tooltipText.text = upgradeDescription; // Se coloca la descripcion correspondiente
        }
    }

    0 references
    public void OnPointerExit(PointerEventData eventData)
    {
        tooltipPanel.SetActive(false); // Se desactiva si sale
    }
}
```

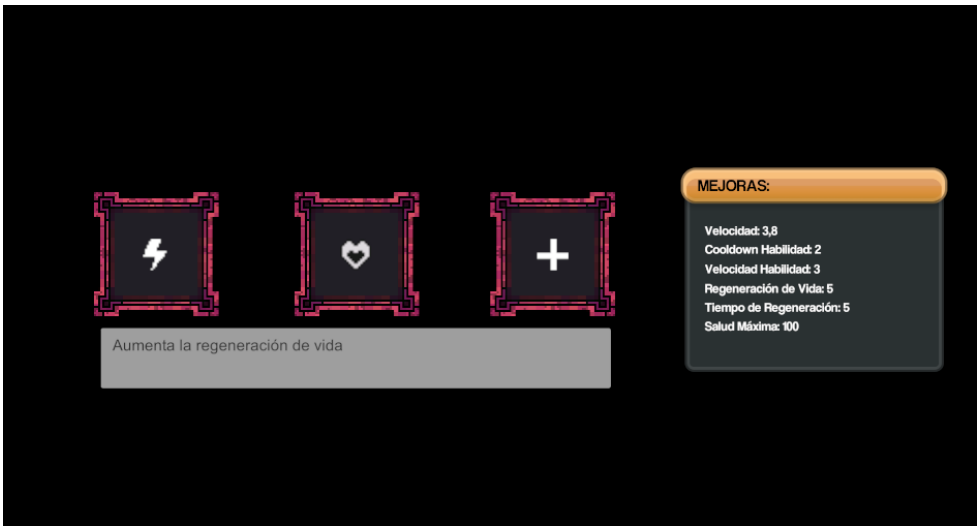
Clase BotonesHover

Ya, por último, una vez escogida la habilidad y colocada la mejora al personaje principal, lo siguiente sería pasar a la siguiente oleada de la siguiente forma

```
void proximaOleada()
{
    Time.timeScale = 1f; //Vuelva a empezar la oleada
    pantallaMejoras.SetActive(false); //Pantalla mejoras quitar
    // Incrementar el número de oleada
    int numOleada = PlayerPrefsSettings.instance.getNumOleada() + 1;
    PlayerPrefsSettings.instance.setNumOleada(numOleada);
    // Actualizar el texto del número de la oleada
    SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex, LoadSceneMode.Single);
}
```

Método para cambiar el número de oleada

Finalmente, la pantalla quedaría así:



Pantalla de mejoras

5.2. Pruebas

- Las pruebas realizadas fueron primero dentro del menú principal en el hay un fondo animado, el cual tuve que animar frame por frame y observar que este estuviera bien y no hubiera ningún corte extraño que hiciera que la animación se viese mal.
- Los botones de este menú son el de jugar donde tiene que pasar al siguiente escenario que sería el juego, el de puntuaciones que cargase las puntuaciones correctamente de la última partida jugada, y, por último, que pudieras cerrar el juego correctamente.
- También poder acceder a la pantalla donde explica un poco la información del juego a nivel de historia y tener un poco de contexto de la situación.
- Dentro del juego, se comprobó que el mapa estuviera correctamente colocado y con su correcto funcionamiento, comprobando que el jugador no podía salir de los limites de mapa establecidos y que también chocase con los obstáculos en el mapa. Además de que todo el mapa estuviera bien diseñado y no se viera nada fuera de lo común.
- El personaje principal contiene animaciones que según hacia donde te dirigieras (izquierda o derecha) el personaje se giraría en esa dirección para darle algo más de realismo, con ello también iría la animación correspondiente al movimiento, es decir que cuando este se moviera se activara la animación.
- La habilidad que dispara el jugador había que comprobar que se dirigía hacia donde estaba el cursor, además de que hiciera la animación de movimiento correspondiente.
- El enemigo debía hacer daño al personaje principal y además este también cuenta con una animación de movimiento.
- El spawn debía estar colocado de manera correcta para que no hubiera bugs con el mapa y el personaje principal. Además de que había que controlar cada cuanto aparecen enemigos, y comprobar que estos siguieran al jugador.
- Comprobar que el temporizador funciona de manera correcta, junto con el aumento del número de oleadas, y que se restase la vida cuando un enemigo pega al personaje principal.
- Cuando el enemigo pierda toda la vida, se haga una animación de muerte y después de que termine la animación aparezca la pantalla con el Game Over.

- Que apareciera nada más empezar el juego la pantalla de controles, y solamente volvería a salir en el caso de salir del juego y volver a entrar, o al reiniciar la partida.
- El funcionamiento de los botones en el canvas del Game Over, que se pueda reiniciar la partida, que vuelvas al menú principal y que puedas cerrar el juego.
- El correcto funcionamiento de la pantalla de mejoras, que esta se tiene que activar cuando el temporizador llega a cero, junto con las tres mejoras en pantalla escogidas aleatoriamente, y su especifica descripción que al pasar por encima del botón te aparece, y el resumen de las habilidades que a medida que las cambias también te aparece en pantalla para que sepas que mejoras has hecho.
- Las pruebas de que la pantalla de victoria se activase cuando el boss estuviera muerto.
- Que todos los enemigos siguieran al jugador una vez aparecen en pantalla, ya sea el esqueleto, mago o jefe, y que los tres hiciesen la animación de movimiento, ataque, y de muerte.
- Por otro lado, que el jefe además de las animaciones que tiene el esqueleto y el mago también que aparezca la barra de vida cuando aparece él, y que cuando el jugador atacase este perdiera vida.
- Que se guardara y se mostrara las mejoras de la última partida ganada en la pantalla de puntuaciones.
- Tuve que probar que al darle al botón de pausa no se pueda acceder a él si ya está activo el menú de mejoras, Game Over, Victory, o Controles.
- Que la música comience cuando se reinicia el juego, pero cuando estes en el menú no suene.

5.3. Manual de instalación

1. **Requisitos del Sistema:**
    - Sistema Operativo: Windows 10/11 o macOS
    - Procesador: Intel Core i3 o equivalente
    - Memoria RAM: 4 GB
    - Almacenamiento: 500 MB de espacio libre
    - Tarjeta Gráfica: Compatible con DirectX 11 (para Windows) o Metal (para macOS)
    - Conexión a Internet: Para descargar el juego

2. Pasos de Instalación:

Descargar el Juego

- Acceder al Enlace de Descarga:
- Abre tu navegador web preferido y accede al siguiente enlace [Enlace]

Iniciar la Descarga:

- Haz clic en el botón de descarga en la página (espera a que se complete la descarga del archivo. Esto puede tomar unos minutos dependiendo de la velocidad de tu conexión a Internet)

Extraer el Archivo Descargado

- Ubicar el Archivo Descargado:

- Una vez completada la descarga, localiza el archivo en la carpeta de descargas de tu computadora. El archivo descargado será un archivo comprimido (por ejemplo, EternalEclipse.zip).

Extraer el Contenido:

- Haz clic derecho sobre el archivo comprimido y selecciona "Extraer todo" (en Windows) o "Abrir con > Utilidad de Compresión" (en macOS).
- Elige una ubicación para extraer los archivos, como el escritorio o una carpeta específica.

Instalar el Juego

- Abrir la Carpeta Extraída:
- Navega hasta la carpeta donde extrajiste los archivos del juego.
- Abre la carpeta y ejecuta el EternalEclipse.exe

5.4. Manual de usuario

Nada más comenzar el juego tenemos la pantalla principal son los respectivos tres botones, el primero de ellos nos servirá para acceder al juego, el segundo para las puntuaciones, y el tercero para cerrar el juego.

Pantalla de inicio

Una vez entremos al juego comenzará la partida, y nos aparecerá una pantalla con los controles básicos como el "*WASD*" (recuerda que dentro del mapa hay unos límites los cuales no puedes sobre pasar, y unos obstáculos por el mapa que harán que te choques con ellos) para mover al personaje, y podremos disparar hacia la dirección que se encuentre nuestro cursor dándole click izquierdo, de esta forma lanzará una habilidad de fuego que al darle a un enemigo este morirá.

Jugador

Habilidad de jugador

Durante todo el juego el jugador tendrá la oportunidad de pausar el juego dándole al escape, y desde ahí poder continuar por donde estaba dándole de nuevo al escape, reiniciar el juego, volver al menú o quitar el juego. Además, desde esa pantalla se puede observar las diferentes mejoras que van cambiando según las que escoja el jugador.

Esqueleto

Los enemigos irán apareciendo según pase el tiempo y comenzarán a perseguir al jugador, a estos hay que matarlos disparándoles. Pero ten cuidado, ya que si se te acercan demasiado estos pueden comenzar a quitarte vida (esta se encuentra arriba del todo sobre el personaje principal), por un lado, el esqueleto se acercará lo suficiente para darte con su espada, en cambio el mago una vez este a un cierto rango de distancia comenzará a lanzar su habilidad en dirección al jugador, lo que podría resultar en perder y entonces aparecería la pantalla de Game Over.

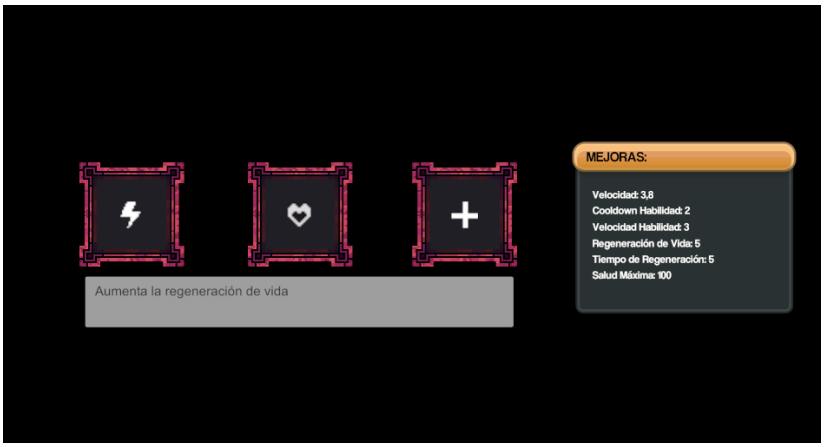
Mago

Vida, número de oleada y temporizador

Pantalla de GameOver

En ella podrás comenzar de nuevo dándole al primer botón, el según sería para volver al menú principal, y el último, para cerrar completamente el juego.

En el caso de que lograr librarte de los enemigos, si te fijas arriba hay un contador (justo debajo de la barra de vida). Este temporizador te ayudará a saber cuanto te queda para sobrevivir esa oleada, y poder conseguir alguna mejora.



Pantalla de mejoras

Cada una sirve para una cosa:

Salud máxima: Aumenta la vida del jugador (por ejemplo, de 100 120)

Velocidad: Aumenta la velocidad del jugador

Velocidad de habilidad: La habilidad es más rápida y por lo tanto llega más lejos

Tiempo de regeneración: El tiempo que el jugador puede regenerar vida (por ejemplo, puede que cada dos segundos regeneres vida, pues puedes regenerar después de ese intervalo de tiempo, si te falta vida, uno de vida)

Cooldown de la habilidad: La habilidad tarda unos segundos en salir, por lo tanto, esta mejora hace que ese tiempo de espera sea menor, y por lo tanto atacar más seguido

Regeneración de vida: Si te han quitado vida regeneras puntos (por ejemplo, cada dos segundo puedes regenerar cinco de vida)

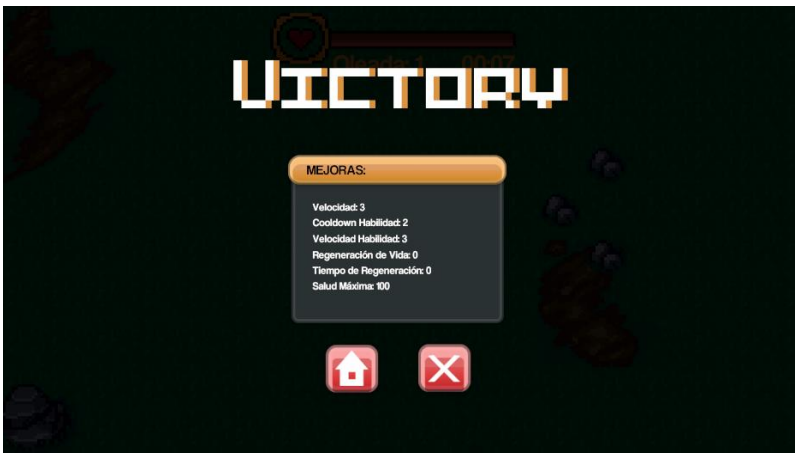
Una vez escogida tu mejora, pasarás automáticamente a la siguiente oleada, y se repetirá el mismo ciclo, hasta la oleada 5.

Jefe final

A partir de la oleada 5 tendrás que pelear contra los enemigos y una vez termine el tiempo aparecerá el jefe al cual tendrás que matar. Para saber cuando de vida tiene lo podrás ver debajo de la pantalla.

Barra de vida del jefe

Una vez lo mates conseguirás la victoria, con todas las mejoras realizadas, y podrás o volver al menú principal o cerrar el juego.



Pantalla de Victory

6. Evaluación final del proyecto.

6.1. Evaluación de diseño, del proceso y del resultado

Durante el desarrollo del proyecto, se realizaron varios cambios significativos en relación con el diseño inicial. Se hizo hincapié en mejorar la experiencia del jugador, priorizando la fluidez del juego. Uno de los cambios más destacados fue la modificación de los controles del jugador. En lugar de utilizar las flechas del teclado, se optó por una configuración más común en los juegos de este estilo, utilizando las teclas "WASD" para mover al jugador y el clic del ratón para disparar. Esta decisión se tomó para que los jugadores se sintieran más familiarizados y cómodos con los controles, facilitando así su adaptación al juego.

Además, se implementaron otras mejoras importantes, como la pantalla de "Game Over", “Victory”, “Pause” y “Controls”, que proporciona una experiencia más completa al jugador al finalizar una partida. En la pantalla de mejoras, se añadió la funcionalidad de mostrar información detallada sobre cada mejora al pasar el ratón por encima de ella, junto con un resumen claro de todas las mejoras disponibles. Este cambio permite a los jugadores tomar decisiones más informadas sobre cómo mejorar su personaje y optimizar su estrategia de juego.

Estos ajustes y mejoras no solo contribuyeron a una experiencia de juego más fluida, sino que también aumentaron la accesibilidad y la comprensión del juego para los jugadores, mejorando en general la calidad y la satisfacción del usuario.

6.2. Conclusiones y lecciones aprendidas.

El desarrollo de este proyecto ha sido una experiencia invaluable que ha proporcionado una amplia gama de aprendizajes, tanto técnicos como de habilidades blandas. Entre las conclusiones y lecciones aprendidas destacan:

- Resolución de problemas: Durante el desarrollo, se enfrentaron diversos desafíos técnicos y de diseño que requerían soluciones creativas e innovadoras. La capacidad para identificar y abordar problemas, incluso aquellos que inicialmente no fueron percibidos, fue fundamental para mejorar la calidad del juego y reducir errores.
- Mejora constante: A medida que el proyecto avanzaba, se descubrieron nuevas técnicas y funcionalidades que contribuyeron a enriquecer la experiencia del juego. Esta constante búsqueda de mejoras y aprendizajes permitió un progreso significativo en el desarrollo y la calidad del producto final.
- Constancia en el trabajo: La organización y la constancia fueron pilares fundamentales para el éxito del proyecto. Mantener un plan de trabajo estructurado y seguirlo de manera disciplinada no solo facilitó el avance del proyecto, sino que también permitió la incorporación de nuevas ideas y funcionalidades, incluso aquellas que no estaban inicialmente contempladas.

En resumen, el proyecto no solo proporcionó conocimientos técnicos en el desarrollo de juegos, sino que también fortaleció habilidades como la resolución de problemas, la mejora continua y la gestión del tiempo. Estas lecciones aprendidas no solo son aplicables al ámbito del desarrollo de juegos, sino que también son transferibles a otros proyectos y áreas de la vida profesional y personal.

6.3. Posibles ampliaciones futuras

A pesar de haber alcanzado los objetivos principales del proyecto, hay varias áreas que podrían ampliarse o mejorarse en el futuro:

- Contenido adicional: Se podría agregar contenido adicional al juego, como nuevos niveles, enemigos, poderes y objetos, para aumentar la profundidad y la rejugabilidad.
- Personalización del jugador: Se podría implementar la capacidad de personalizar el aspecto y las habilidades del jugador para proporcionar una experiencia de juego más personalizada.

7. Bibliografía/webgrafía

Herramientas/paginas utilizadas

- Plataforma de desarrollo en tiempo real de Unity | Motor de 3D, 2D, VR y AR. (s. f.). Unity. https://unity.com/es
- *Visual Studio Code - Code editing. Redefined*. (2021, 3 noviembre). https://code.visualstudio.com/
- Top game assets. (s. f.). itch.io. https://itch.io/game-assets
- DaFont - Descargar fuentes. (s. f.). https://www.dafont.com/es/

Páginas utilizadas para la memoria

- ¿Qué es un roguelite? Su significado y más - Plarium. (2024, 10 marzo). plarium.com. https://plarium.com/es/glossary/roguelite/#:~:text=Vamos%20a%20aclararlo.,directamente%20al%20punto%20de%20partida.
- Unity – Guardar y cargar muchas variables de manera rápida y eficiente – Arjierda Games blog. (2023, 21 mayo). https://arjierdagames.com/blog/unity/unity-guardar-y-cargar-muchas-varaibles-de-manera-rapida-y-eficiente/#:~:text=Las%20funciones%20de%20PlayerPrefs%20de,convertir%20algo%20molesto%20de%20hacer.
- Definición de «Vista cenital». (s. f.). GamerDic. https://www.devuego.es/gamerdic/termino/vista-cenital/#:~:text=Perspectiva%20de%20juego%20en%20la,visi%C3%B3n%20orientada%20de%20arriba%20abajo.
- Technologies, U. (s. f.). UnityEngine.PlayerPrefs - Unity Scripting API. https://docs.unity3d.com/es/2019.4/ScriptReference/PlayerPrefs.html