



PROYECTO FINAL

Videojuego 2D

Andrés Pérez, Jaime Sansón y Francisco
Pagadizabal



ÍNDICE

Introducción	2
Justificación	3
Objetivos del Proyecto	3
Metodología	3
Herramientas de Trabajo	4
Diseño del Videojuego	6
Desarrollo Técnico.....	10
Control de Calidad	15
Estado.....	16
Referencias.....	17
Anexos	17
Conclusión	19

Introducción

Antes de llegar a la idea final, exploramos diferentes opciones en el mundo de los motores de juego. Inicialmente, consideramos trabajar con Unreal Engine 5. Sin embargo, debido a la complejidad de la idea del juego, nos vimos obligados a optar por el motor de Unity. Allí, nos enfocamos en el género que queremos seguir como idea principal, siendo este aún popular, el roguelike. Un estilo de videojuegos que se caracteriza por ser un subgénero de los juegos de rol, el cual nos enfrentaremos a una serie de niveles con una temática de mazmorras, laberintos y una muerte permanente del personaje. Con la incorporación de una vista en 2D muy similar a la de un Metroidvania. Otro subgénero de los videojuegos que se basa en la acción-aventura mediante escenarios en plataformas.

Con todas las ideas recogidas, finalmente decidimos de crear un juego de plataformas, usando los elementos principales del estilo roguelike, escenarios con diversas plataformas para el desarrollo de los diferentes niveles, una aleatoriedad entre los mapas para que no tenga siempre el mismo desarrollo de niveles y terminamos unificando en una arquitectura de 16 bits en todos los aspectos visuales del juego. Con la arquitectura mencionada, veremos una fusión de los juegos clásicos y juegos indie contemporáneos de su época.

Nuestro objetivo con este proyecto es la creación de un videojuego en el motor Unity. Al nunca haber usado este motor, nos ha supuesto un reto, que lo podemos usar para desarrollarnos en el entorno y la creación de los videojuegos.

Nos desafiamos en un entorno nuevo de programa y en el lenguaje de programación, nos ha ayudado saber la estructura C#, pero al estar pensada para el desarrollo de unity 2D varía en varias funcionalidades para el personaje, objetos, enemigos etc... Son cosas que no habíamos visto en estos años académicos. Esto significa que el aprendizaje y la puesta en marcha de la misma, ha sido continua en todos los aspectos técnicos que hemos visto en el juego, como el diseño y la implementación de los diferentes sprites, la unión y vista de los niveles, la arquitectura para el sistema del juego etc... En el lenguaje de programación, nos ha ayudado saber la estructura C#, pero al estar pensada para el desarrollo de unity 2D varía en varias funcionalidades para el personaje, objetos enemigos etc...

Justificación

Nuestro interés radica en aprender un nuevo lenguaje de programación en profundidad y adentrarnos en el mundo del desarrollo de videojuegos. Todos los integrantes del grupo, tenemos la misma afición en común, que son los videojuegos, y ahora queremos probar el proceso desde el punto de vista del desarrollo y la creación de uno mismo, y ver de primera mano cómo funciona todo el proceso.

Partiendo desde cero, nos embarcamos en este proyecto con el objetivo de explorar hasta dónde podemos llegar. Nos motiva especialmente la combinación géneros que hemos mezclados y que hemos mencionados anteriormente en la introducción. En la idea que estamos desarrollando nos motiva por el hecho que poder jugar con la imaginación y la creatividad de cada uno.

Objetivos del Proyecto

Nuestro objetivo principal es desarrollar un videojuego que mezcla las diferentes mecánicas de los juegos de plataformas con los elementos característicos de los juegos roguelike.

Ofrecemos una aventura con el formato de modo historia, cuenta con la singularidad de un comienzo que no será igual, cada partida será variada, debido a que al momento de morir y de empezar una partida, los mapas irán variando. Esto proporciona un estilo que capta la intriga de un nuevo comienzo de partida, para descubrir a que se enfrentara, ya que las posibilidades se limitan con los mapas que estén creados. Este sería el principal protagonismo del juego, creando así una variante de los juegos clásicos que una vez tenga un final se termina. Rompemos la monotonía y ofrecemos una nueva manera de entretenimiento para los usuarios.

Usamos la esencia de la aleatoriedad del estilo de los juegos roguelike, donde cada partida se generará de manera nueva y no caemos en la monotonía de manera instantánea, y nos asegurando que ningún recorrido será igual. Además, pondremos énfasis en la diversidad y la importancia de los objetos dentro del juego para aumentar varios aspectos del jugador que le ayudaran en su experiencia dentro del juego. Estos objetos serán clave para otorgar ventajas estratégicas, agregando ese sentimiento de emoción dentro de la partida.

Metodología

Scrum

Hemos optado por la metodología ágil Scrum, dividiendo nuestro trabajo en cuatro sprints mensuales. Durante cada sprint, recopilamos información sobre nuestras actividades individuales, buscamos oportunidades para unificar o corregir el trabajo realizado, y asignamos nuevas tareas para el próximo sprint.

Diagrama de Gantt



diagrama_gantt.xlsx

Herramientas de Trabajo

Unity

Elegimos utilizar Unity como plataforma de desarrollo por varias razones clave. Primero, su integración con GitHub facilita un control de versiones robusto, permitiéndonos colaborar de manera eficiente, con cada miembro trabajando en su propia rama para luego fusionar los cambios en la rama principal, garantizando un entorno de trabajo seguro y organizado. Segundo, Unity ofrece una amplia gama de funcionalidades gratuitas. Tras una investigación detallada, descubrimos que pocos otros motores de desarrollo de videojuegos proporcionan tantas herramientas y servicios sin costo alguno, lo que nos permitió desarrollar un juego completo sin restricciones.

Comparativa Unity / UE5

Características	Unreal Engine	Unity
Lenguaje de Programación	C++ y Blueprint	C#
Gráficos	Altamente detallados, utilizado para gráficos 3D realistas y proyectos AAA.	Buen rendimiento en 2D y 3D, pero menos útil con gráficos avanzados.
Facilidad para el desarrollo	Más difícil de aprender, sobre todo por el lenguaje utilizado.	Más amigable para principiantes ya que consta de una interfaz intuitiva.
Plataformas Soportadas	PC, consolas, móviles y VR principalmente. Enfocado a consolas de 9 Gen.	Más de 25 plataformas incluyendo móviles, PC, consolas, web y VR.
Precios	Gratis, pero con un 5% de impuestos para ingresos	Gratis para ingresos menores a \$100,000 suscripciones

	superiores a \$1 millón.	pagas para más funcionalidades.
Rendimiento	Optimizado para gráficos intensivos y proyectos de gran escala.	Optimizado para juegos móviles y proyectos indie.
Capacidad y disponibilidad de Activos	Menor variedad en comparación con Unity.	Gran cantidad de activos disponibles en el Unity Asset Store.
Colaboración y Pipeline	Excelente integración con herramientas de producción como FBX, USD, Python scripting.	Buena integración, pero con menos opciones avanzadas que Unreal.

Unity Control Version

Al principio del desarrollo, optamos por utilizar el servicio de control de versiones de Unity. Sin embargo, encontramos que esta herramienta no se ajustaba completamente a nuestras necesidades y resultaba difícil de comprender.

GitHub

Este repositorio se ha convertido en nuestra opción definitiva, ya que la capacidad de crear ramas (que no pudimos averiguar cómo hacer en el Control de Versiones de Unity) nos permite agilizar el proyecto. Ahora, el trabajo se divide en cuatro ramas, además de la principal, con una rama de pruebas asignada a cada miembro del equipo. Además, hemos integrado la aplicación de escritorio de Unity, que ofrece una gran compatibilidad con la plataforma de desarrollo. Esta herramienta nos permite cambiar de rama al instante en Unity, facilitando la coordinación y la gestión del proyecto.

Visual Studio Comunity 2022

Hemos utilizado la versión gratuita de Visual Studio como nuestra herramienta principal para el desarrollo de scripts. Esta versión incluye el paquete de Unity, lo que garantiza la instalación automática de todas las dependencias necesarias para un funcionamiento correcto. Sin embargo, es necesario seleccionar este editor dentro de los ajustes del proyecto en Unity para asegurar una integración adecuada.

Visual Code

Esta versión también ha sido utilizada por un miembro del equipo, para poder utilizar este tubo

que descargar extensiones relacionadas con Unity y habilitar la edición dentro de las preferencias de Unity. La extensión que se han utilizado para el uso de visual code son:

Unity: esta extensión desarrollada por Microsoft nos permite detectar los paquetes de unity y visual code para que no den errores de formato y conectar visual code para que a la hora de abrir los diferentes scripts en cs nos redirija a visual sin problema. La id de la extensión es “visualstudiotoolsforunity.vstuc”. La versión es la 1.0.1.

C# Dev Kit: Como indica su nombre, el kit general para detectar el lenguaje de C#. Con esta extensión podemos administrar el código en el explorador de soluciones, ejecución y detección de errores, así como el desarrollo en diferentes SO, como Windows, Linux o macOS. La versión es la 1.4.29.

Diseño del Videojuego

Trama

El principio de la historia trata de un buscador de tesoros llamado Hasbulla. Este chico, encontró en unas ruinas una reliquia muy extraña, un anillo, el cual albergaba un poder inimaginable hasta para él. Con este extraño anillo, su portador, tenía el poder de atravesar y cambiar de dimensiones. Sin saber el terrible error que cometía, Hasbulla decidió llevarse el anillo con la excusa de que, si caía en malas manos, podría ser utilizado para destinos malignos. Lo que no sabía, es que este anillo solo lo podía portar su creador, Komanche, y que en el mismo instante en que la reliquia tocara su dedo, un futuro fatal le esperaba a su querido mundo. Cuando este se encajó en el dedo del explorador, la tierra empezó a temblar, y en frente suyo, se abrió un portal. Maravillado, Hasbulla se acercó al portal para cruzarlo, pero cuando intentó pasar la mano, una barrera invisible le impedía cruzar esta puerta a lo desconocido. Desconcertado y decepcionado, Hasbulla se quitó el anillo, pero fue una sorpresa ver que el portal seguía allí, y de repente, sus ojos presenciaron como una criatura parecida a una rata, deforme y sucia, atravesaba este puente entre dimensiones. Al principio estaba maravillado, había conseguido contactar con un ser vivo de otra dimensión, sin embargo, esa sensación de alegría se transformó en miedo cuando no dejaron de salir criaturas del portal. La última en salir fue lo que parecía su líder, y justo después, se cerró el portal. Este tenía una apariencia absurda, era un simple cubo amarillo. El explorador se dirigió al líder para hablar con él, y comprobar sus intenciones. El nombre de esta especie de caja era Christian El Globo De La Cruz González. Al principio se mostró respetuoso, pero con un tono dominante, decía que era su deber salvar a esta dimensión de la tiranía que presenciaban día a día. Al ver que no tenían buenos fines, Hasbulla se fue corriendo del lugar, dejando el futuro de este mundo a su suerte.

Estos seres empezaron a conquistar el mundo, aldea por aldea, hasta que llegan al pueblo de nuestro personaje, llamado Cuco. Cuco era un cazador que solía salir y estar fuera durante días, consiguiendo alimento para todo el pueblo. En una de estas salidas, cuando volvió al pueblo, vio que estaba destrozado por los invasores. Ya había escuchado sobre este ejército que poco a poco se iba haciendo con más territorios, pero rezaba para que el día en el que atacaran su pueblo nunca llegase. Todo el lugar donde había crecido estaba hecho cenizas. Esto enfureció a Cuco, y decidió que acabaría con todo aquel que destrozó su vida. En ese momento inició su viaje para acabar con su líder.

Aquí empieza el videojuego, con el jugador en una habitación, con un personaje explicándole la situación (es Hasbulla solo que él no lo sabe). En este momento, tienes que elegir un arma y adentrarte a la aventura para poder saciar tu sed de venganza.

Gráficos

En el proceso de creación de los elementos visuales para nuestro juego, hemos optado por utilizar una herramienta conocida como ASEPRITE. Esta herramienta nos ha proporcionado la facilidad y flexibilidad necesarias para realizar píxel art. Con ASEPRITE, hemos diseñado personajes, escenarios, objetos y el decorado, centrándonos en la estética característica de los juegos de 16 bits.

El término "16 bits" se refiere a un estilo de videojuegos de los años 80 y 90. Durante esta época, las consolas de videojuegos tenían procesadores de 16 bits. Estas consolas tenían limitaciones técnicas en comparación con las plataformas modernas, lo que resultaba en gráficos pixelados y una paleta de colores limitada.

Para lograr un estilo visual coherente y atractivo para nuestro juego, hemos realizado una investigación y hemos obtenido referencias de varios juegos que se basan en la estética de los 16 bits. Estas referencias nos han servido como inspiración y guía para diseñar algo propio y original, adaptado a la trama y el estilo del juego que estamos desarrollando.

Sonido (En progreso)

En cuanto al aspecto del sonido, hasta el momento no hemos iniciado su desarrollo.

Personajes (Ampliar / Aclarar)

Jugador

El jugador es el único personaje que el usuario puede controlar. Equipado con un arma para el combate, posee habilidades especiales que pueden mejorar o adaptarse para enfrentarse a los enemigos durante el juego.

NPC

Los NPC's son personajes no controlados por el usuario que cumplen diversas funciones dentro del juego. Estos personajes suelen estar relacionados con la trama y pueden proporcionar diálogos significativos que contribuyen al desarrollo de la historia. Además, algunos NPC's pueden ofrecer sistemas de intercambio de objetos.

Enemigos (añadir listado de enemigos)

Los enemigos son los antagonistas del jugador. Equipados para el combate cuerpo a cuerpo, estos enemigos siguen rutas predefinidas de patrulla y exhiben una variedad de comportamientos durante el juego, ofreciendo desafíos y enfrentamientos únicos para el jugador.

Jefes

Los jefes son enemigos más poderosos que se encuentran al final del juego. Estos enfrentamientos son considerados más difíciles que el resto debido a su alta defensa y potencia ofensiva. Los jefes suelen requerir estrategias específicas y habilidades avanzadas para ser derrotados con mayor facilidad, proporcionando un punto culminante emocionante para la experiencia del jugador.

Entorno (Ampliar / Aclarar)

Plataformas

Las plataformas son elementos del escenario que el jugador puede utilizar para desplazarse o alcanzar áreas específicas. Esto incluye tanto plataformas estáticas como móviles, como las bidireccionales, que ofrecen una variedad de desafíos y oportunidades de juego.

Escaleras

Las escaleras son elementos del escenario que se enfocan en facilitar el movimiento vertical controlado por el jugador. Estas estructuras permiten al jugador ascender o descender entre diferentes niveles de altura dentro de los niveles del juego, ofreciendo una forma intuitiva y fluida de explorar entornos y acceder a nuevas áreas.

Pinchos

Los pinchos son un elemento del escenario el cual está diseñado para dar un reto al jugador y molestar, estos se pueden encontrar tanto en fosos como en los techos de algunos niveles, acotando así los movimientos posibles para el jugador en ciertas circunstancias.

Una vez tocas un pincho la vida del jugador baja automáticamente a 0 y se pierde la partida.

Mecánicas (Ampliar / Aclarar)

Movimiento

El movimiento se divide en dos secciones: el utilizado por el jugador y el utilizado por los enemigos. El jugador puede moverse en ocho direcciones y cuenta con diversas habilidades que mejoran su movilidad, como el doble salto o el dash. Además, tiene la capacidad de escalar escaleras y

subirse a plataformas bidireccionales. Por otro lado, los enemigos tienen un movimiento limitado a dos direcciones, centrado en una zona designada como su patrulla.

Ataque

El jugador tiene un patrón de ataque determinado por el arma elegida al inicio del juego, que puede ser potenciado por objetos obtenibles durante la partida. En contraste, los enemigos realizan ataques por contacto.

Vida

El sistema de vida determina la salud del jugador y de los enemigos. El jugador comienza con 100 puntos de salud, que pueden aumentar o disminuir según los objetos encontrados a lo largo de los niveles. Los enemigos tienen una salud variable adaptada a su diseño y nivel de peligro.

Objetos (Aclarar, como se obtienen y como interactúan con el jugador) (Incluir listado)

Los objetos son elementos que se encuentran dispersos de manera aleatoria por los niveles durante la partida.

Estos se pueden obtener una vez nos encontramos dentro de un nivel, la manera de la que aparecen estos son aleatorios, estos los podemos adquirir acercándonos al objeto en cuestión y presionando la tecla "E".

Estos tienen 2 formas de interactuar con el jugador, por vía de estadísticas o por vía de habilidades. En el caso de las estadísticas proporciona un aumento considerable para amenizar la partida y volverla más equilibrada a medida que progrese dentro del juego.

Arquitectura

Diseño de niveles

En el diseño del juego, se distinguen elementos tanto obligatorios como opcionales. Entre los elementos obligatorios se encuentran el jugador, los enemigos y las plataformas, los cuales conforman la estructura básica del juego.

Por otro lado, los elementos opcionales incluyen objetos especiales, así como también elementos ambientales como escaleras, pinchos u otros obstáculos que añaden profundidad y desafío al diseño del nivel. Estos elementos opcionales permiten una mayor diversidad y complejidad en la jugabilidad, ofreciendo diferentes caminos y estrategias para los jugadores mientras exploran el mundo del juego.

UI

La interfaz del juego se divide en dos partes distintas: los menús y la interfaz dentro del juego.

En los menús, los jugadores pueden acceder a opciones como ajustes, salir del juego y configuraciones de resolución, entre otras. Estos menús proporcionan un entorno donde los jugadores pueden gestionar aspectos técnicos y de configuración antes de iniciar o durante la partida.

Dentro del juego, la interfaz muestra información crucial para la experiencia de juego, como la vida del jugador, un mapa que indica la ubicación actual, y los objetos obtenidos durante la partida. Esta interfaz dentro del juego es fundamental para que los jugadores se mantengan informados sobre su progreso, salud y recursos disponibles mientras exploran el mundo del juego y enfrentan desafíos.

Desarrollo Técnico

Programación

Menú principal

Pantalla con 4 botones, cada botón está enlazado con una escena, este cuenta con Jugar, que lanza la primera escena dentro del juego, tutorial, que lanza la escena relacionada con el mismo, jefe, que lanza la escena relacionada con el jefe y la de salir que cierra la aplicación.

```
0 referencias
public void Play()
{
    SceneManager.LoadScene(2); // Cambiar a la escena de juego
}
```

Para navegar entre las escenas usamos la clase SceneManager y cambiamos entre las mismas con LoadScene.

Final de partida *

Pantalla encargada de permitir al usuario volver a empezar una nueva partida o volver al menú principal, se basa en la misma lógica que el menú principal, pero añadiendo el factor que dependiendo si el jugador ha perdido o ganado se muestre una información u otra.

Cámara

Uno de los pilares, sin esto no se podría jugar ya que no sabríamos que está sucediendo, la cámara se compone de 4 posiciones las cuales indican los límites de la cámara en las en los planos x e y. También sigue al jugador mediante su posición.

```
transform.position = new Vector3(Mathf.Clamp(player.position.x, minWidth, maxWidth),
    Mathf.Clamp(player.position.y, minHeight, maxHeight),
    transform.position.z); // FIJAR CAMARA AL JUGADOR EN EL EJE X E Y
```

Aquí se utiliza la clase Mathf, más concretamente Clamp para establecer límites y asignando en

vector 3 las posiciones x, y e z a las del jugador.

Movimiento del jugador *

Se encarga del movimiento del jugador, en este obtenemos parámetros del jugador y estadísticas, constantemente se comprueban los controles de movimiento horizontal y salto. Calculando así la velocidad de movimiento del jugador para saber hacia dónde mira el Sprite de este. Aquí también está la función que controla la capacidad de realizar doble salto y dash (próximamente).

(Insertar imagen de código sobre movimiento básico)

La función de doble salto es un “evento” extra dentro de la comprobación del salto normal, hay un interruptor el cual es un objeto y en caso de este estar activado te permite ese salto extra. En caso de agotar el número de saltos este se reiniciará una vez el personaje toque el suelo. Esto es bastante parecido para el dash, aunque este se basa más en una mecánica explicada en un apartado posterior, usando la misma lógica que el retroceso del jugador al recibir daño, pero activándolo nosotros a gusto, obviamente estando limitado por un tiempo entre dashes para no arruinar la experiencia.

(Insertar imagen de código sobre doble salto / dash)

Vida del jugador

El sistema de vida este compuesto de la vida misma y la barra de vida, ambas se actualizan de manera sincronizada dependiendo de los eventos que transcurren dentro del juego, uno de los eventos que se esperan es el de recibir daño, al entrar en contacto con un enemigo este activa el método para efectuar el daño al jugador, este se encarga de comprobar de manera constante la vida para mostrarla por pantalla.

También cuenta con la invencibilidad, la cual se activa una vez el jugador recibe daño para así evitar que se le haga de manera instantánea daño otra vez, esta esta señalizada con un cambio visual en el jugador.

En caso de que la vida del jugador llegue a 0 este se deshabilita y hace saltar la escena encargada del final de partida.

Ataque del jugador *

Se compone de la estadística de daño del jugador y el rango de ataque, con estas 2 estadísticas podemos saber qué tipo de arma tenemos, hay 2 armas que son objetos, dependiendo de cual este activo se vera reflejado en las mismas.

También cuenta con métodos para hacer daño al enemigo, esto se consigue cuando la colisión del ataque detecta la etiqueta “Enemy” esta llama a un método instanciado en la clase de las estadísticas del enemigo.

(Insertar imagen de código sobre como detecta al enemigo)

De forma adicional hay objetos dentro del juego que modifican las estadísticas.

Estadísticas del enemigo

Contiene las estadísticas básicas del enemigo, velocidad de movimiento, daño y vida, esta última también cuenta con varios métodos para poder actualizar la misma desde la clase relacionada con la lógica del enemigo.

Lógica del enemigo

Se encarga de la zona de patrulla del enemigo y de la detección del jugador.

El enemigo patrulla entre el punto A y B, esto se hace obteniendo la posición en el eje x o y (dependiendo del enemigo) y luego con un método hacemos que camine solo hacia esa dirección, en este caso tenemos solo 2 puntos por lo cual usamos un booleano como interruptor para alternar entre ambas.

Para poder atacar al jugador lo hacemos mediante las colisiones, una vez colisiona contra un objeto con la etiqueta de "Player" esta llama a la clase encargada de la vida del jugador y la actualiza. Esto también funciona con un interruptor para alternar entre el modo patrulla y el modo de ataque.

```
Mensaje de Unity | 0 referencias
void OnCollisionEnter2D(Collision2D collision)
{
    if (collision.gameObject.CompareTag("Player"))
    {
        // INSTANCIAMOS EL ENEMIGO CON EL QUE COLISIONO
        EnemyStats enemyStats = GetComponent<EnemyStats>();

        // COMPROBAMOS QUE TENGA ENLAZADO EL SCRIPT
        if (enemyStats != null)
        {
            PlayerHealth.instance.DealMonsterDamage(enemyStats.dmg);
        }
        else
        {
            Debug.LogWarning("El enemigo no tiene el script EnemyStats asociado");
        }
    }
}
```

```
1 referencia
public void DealMonsterDamage(int dmg)
{
    if (iFrames <= 0)
    {
        health -= dmg; // Bajamos la vida del jugador
        hpBar.HpSet(health); // Actualizamos la barra de vida
        if (health > 0) // Comprobamos que el jugador siga teniendo vida
        {
            iFrames = iFramesCountdown; // Le volvemos invulnerable
            PlayerMovement.instance.Knockback(); // Empujamos al jugador hacia atras
            sr.color = new Color(sr.color.r, sr.color.g, sr.color.b, 0.5f); // Le bajamos la transparencia para indicar la invulnerabilidad
        }
    }
}
```

Para la detección el enemigo cuenta con un rango, por medio la posición del jugador y del enemigo de manera constante. Una vez entra en ese rango el enemigo cambia el foco a la posición del jugador en el plano horizontal, en caso de perder el foco este vuelve a la zona de patrullaje.

```
1 referencia
private void AttackPlayer()
{
    EnemyStats enemyStats = GetComponent<EnemyStats>();

    if (enemyStats != null) // En caso de que el objeto exista
    {
        if (player != null) // En caso de que el jugador exista
        {
            // Obtiene la distancia entre el enemigo y el jugador
            float playerDistance = Vector2.Distance(transform.position, player.position);

            // Si la distancia es menor que la distancia de persecución, comienza a atacar al jugador
            if (playerDistance <= detectionRange)
            {
                // Dirección hacia la que se moverá el enemigo a la máxima velocidad
                Vector2 direccion = (player.position - transform.position).normalized;

                // Mueve al enemigo hacia el jugador
                transform.position = Vector2.MoveTowards(transform.position, player.position, enemyStats.movementSpeed * Time.deltaTime);

                attacking = true; // Modo ataque activado
            }
            else
            {
                attacking = false; // Modo ataque desactivado
            }
        }
        else
        {
            Debug.Log("El jugador esta muerto o no se ha generado correctamente");
        }
    }
    else
    {
        Debug.Log("El enemigo esta muerto o no se ha generado correctamente");
    }
}
}
```

Una cosa que remarcar es que para hacer una acción que conlleve un cambio de estadísticas para el enemigo esta se tiene que instanciar previamente, en caso de no hacerlo siempre afectaría al primer enemigo generado, una vez instanciado aislamos de manera individual a los enemigos que puedan hacer distinto daño, tener distinta velocidad o vida.

Daño de los pinchos

Los pinchos cuentan con una caja de colisión la cual al entrar en contacto con la etiqueta "Player" hace una llamada a un método instanciado de la clase encargada de la vida del jugador.

```
📧 Mensaje de Unity | 0 referencias
private void OnTriggerEnter2D(Collider2D collision)
{
    if (collision.tag == "Player") // Comprobar que ha colisionado con el jugador
    {
        PlayerHealth.instance.DealDamageSpikes(); // Llamar a la funcion para bajar la vida
    }
}
}
```

Una vez ahí se le baja la vida a 0, se actualiza en la interfaz y se deshabilita el jugador indicando el final de la partida.

```
1 referencia
public void DealDamageSpikes()
{
    hpBar.HpSet(health);
    health = 0;
}
}
```

Aleatorizar Niveles

Para aleatorizar los niveles se ha generado una lista de niveles en el momento que el jugador aparece dentro del juego, lo que se hace es generar unos números dentro de los límites de los niveles que queremos, estos luego se comprueban que no se encuentren dentro de un array y en caso de que no se encuentren se añaden a la lista de los niveles de esa partida. En este caso está limitado a nivel inicial, luego 5 niveles aleatorios dentro de la lista de estos y el nivel del jefe.

```
private int[] GenerateScenes()
{
    for (int i = 0; i < 5; i++)
    {
        int index = UnityEngine.Random.Range(2, 16); // Límites de escenas
        while (scenesCheck.Contains(index))
        {
            index = UnityEngine.Random.Range(2, 16); // Si ya está, genera otro número
        }
        scenesCheck[i] = index; // Guardar el valor
    }
    return scenesCheck;
}
```

Avanzar los niveles

Para avanzar los niveles primero tenemos que detectar el final del nivel, el cual tiene la etiqueta de "LevelEnd" una vez el jugador colisiona con esa etiqueta rescata el array de los niveles previamente generados y avanza dependiendo del número de niveles que ya se han completado.

```
//PASAR DE NIVEL
// Mensaje de Unity | 0 referencias
private void OnCollisionEnter2D(Collision2D collision)
{
    if (collision.gameObject.CompareTag("LevelEnd") && !nextScene)
    {
        nextScene = true; // Activar cuenta atras

        if (stages < 5)
        {
            SceneManager.LoadScene(scenesCheck[stages]); // Cambiar a la siguiente escena
            stages++;
        }
        else if (stages == 5)
        {
            SceneManager.LoadScene(16); // Cambiar a la escena del jefe
        }
        this.transform.position = levelStart; // Resetear posición del jugador a x:0 y:0
    }
}
```

Para evitar problemas al avanzar en el nivel generamos una "barrera" el cual es un temporizador para activar y desactivar el método de avanzar de nivel y así evitar la duplicidad. Esta parte se sitúa dentro de un Update() para que deltaTime pueda funcionar. Este método es prácticamente idéntico al de la invencibilidad dentro del sistema de vida del jugador.

```
// Cooldown escenas
if (nextScene)
{
    sceneTimer += Time.deltaTime; // Contar el tiempo en segundos
    if (sceneTimer >= sceneCooldown)
    {
        nextScene = false; // Habilitar cambio de escena
        sceneTimer = 0f; // Restablecer contador
    }
}
```

Aleatorizar objetos *

Para generar objetos de manera aleatoria primero hemos obtenido una lista de todos los posibles objetos, una vez con esa lista hemos generado 5 de ellos con un método parecido al de las escenas, una vez con estos obtenidos al cambiar de escena se selecciona uno de los 2 lugares posibles para que aparezca un objeto y se posiciona. Esto se repite en las 5 escenas intermedias.

(Insertar imagen de código sobre cómo un array de Game Objects)

Control de Calidad

Funcionalidad

En esta fase, nos hemos enfocados en verificar los controles, mecánicas e interacciones dentro del juego, con especial atención en la experiencia del jugador. Nos aseguramos de que todos los controles respondan correctamente, que las mecánicas de juego funcionen como se espera y que la interacción entre elementos del juego sea coherente y satisfactoria.

Esto se comprueba mediante la creación del ejecutable del juego y esto se comprueba cada vez que se llega a completar un objetivo de prioridad alta. Por ejemplo, el último control de calidad relacionado con el ejecutable se hizo al implementar el sistema de generación de niveles aleatoria. Con esto hemos sacado errores a solucionar para poder mejorar el sistema de generación y adaptarlo en otros aspectos.

Usabilidad

Nuestro objetivo con estas pruebas es garantizar que el juego sea intuitivo y disfrutable para el jugador. Evaluamos el equilibrio entre la dificultad y la recompensa, así como la sensación general al jugar. Nos esforzamos por crear una experiencia de juego que sea fluida, emocionante y satisfactoria para todos los jugadores.

Este tipo de control de calidad se lleva de manera síncrona con el control de calidad de funcionalidad, ya que al estar probando de manera constante el juego se empieza a encontrar fallas o injusticias dentro del juego.

También lo acompañamos con un tutorial para una previa explicación de las mecánicas básicas del juego, los peligros del mundo y los enemigos y algunas mecánicas de mundo como podría ser las plataformas en movimiento.

Localización

Esta fase es crucial, con esto queremos comprobar que el jugador y los game objects están posicionados de manera correcta, tanto al cambiar de pantalla como al estar en la misma. Esto llega desde posición de aparición del jugador cuando sale hasta el patrullaje del enemigo, todo esto debe mantener un orden independientemente de las acciones del jugador.

Con esto buscamos que el juego sea consistente y evitar una mala experiencia dentro del juego.

Estado y Objetivos

Actualmente el juego se encuentra en una fase de pruebas en la cual le falta la batalla contra el jefe para poder completar el juego. Una vez conseguido eso el enfoque principal pasara a pulir físicas, diseños, sonido y aspectos varios o adición de nuevo contenido complementario.

Prioridad alta

- Sistema de Movimiento (Jugador y Enemigos)
- Sistema de combate (Jugador)
- Sistema de salud (Jugador y Enemigos)
- Gráficos propios
- Objetos
- Interfaz gráfica
- Aleatorización de los niveles y objetos

Prioridad media

- Pantalla de inicio
- Tutorial
- Pantalla de fin del juego
- Detección del jugador
- Plataformas en movimiento
- Escaleras de mano

- Sonido
- Retroceso al colisionar con enemigo
- Batalla contra el jefe
- Conversación con NPC
- Comercio con NPC

Prioridad baja (Opcional)

- Pantalla de ajustes.
- Limitar cantidad de objetos obtenibles
- Selección de arma
- Plataformas bidireccionales
- Interfaz de objetos
- Orden coherente al cambiar de nivel (Interiores y exteriores)
- Historia
- Dash

Referencias

Durante el desarrollo de este proyecto, hemos aprovechado una variedad de recursos para familiarizarnos con el nuevo lenguaje de programación. Entre ellos, hemos realizado cursos en plataformas como Udemy, los cuales nos han brindado un primer acercamiento a la aplicación y sus scripts. Estos cursos han sido fundamentales para adquirir los conocimientos necesarios y comenzar a desarrollar el proyecto con solidez y confianza.

Comunidad de Unity: <https://discussions.unity.com>


StackOverflow: <https://stackoverflow.com>

Cursos gratuitos YouTube.

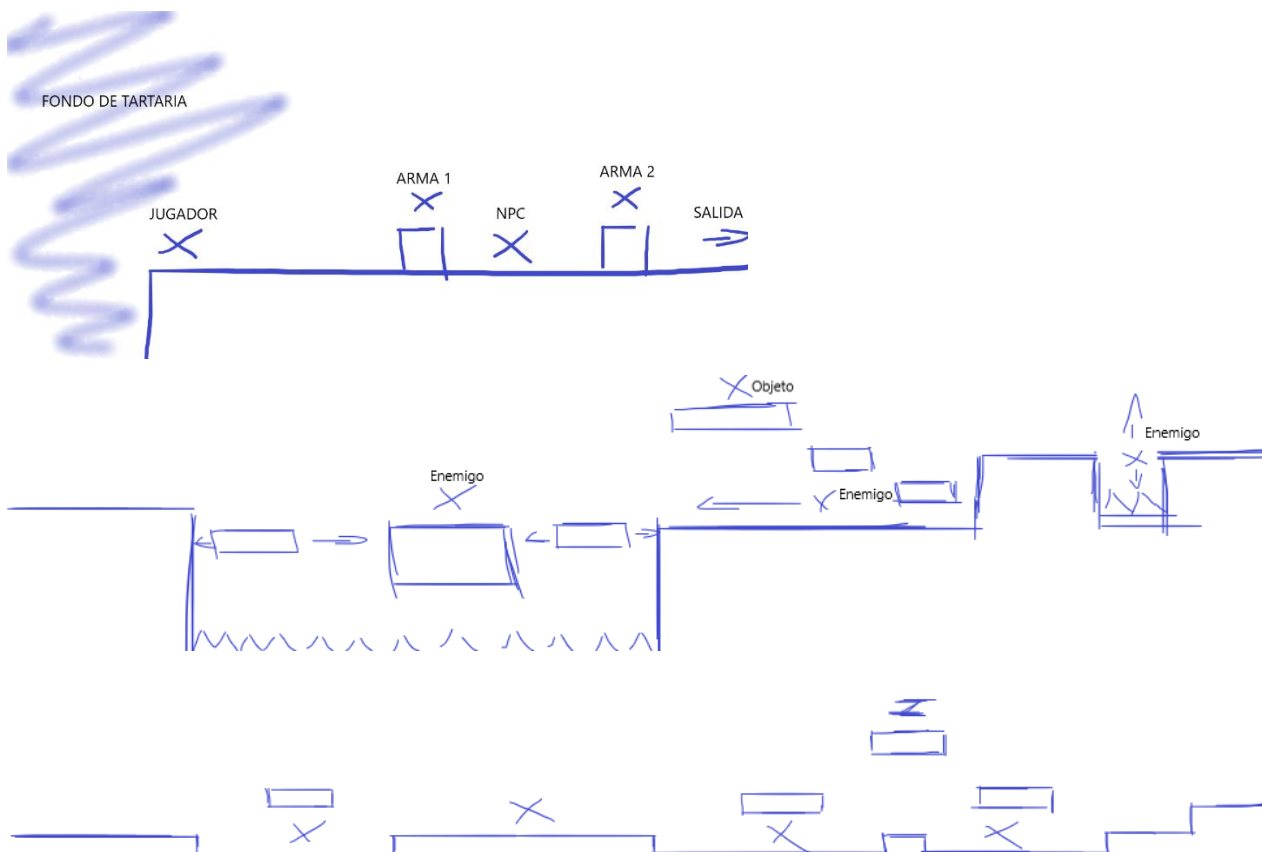
Anexos

Diseños

Los diseños han sido creados desde 0 con el programa ASEPRITE, el pack de assets se compone de 2 bloques principales con sus variantes correspondientes, objetos interactuables como objetos, escaleras y plataformas, enemigos y elementos del mundo como pichos.

Entorno	Mundo	Objetos	Enemigos, NPC y Jugador	Interfaz
				

Bocetos



Videos (En progreso)

Conclusión

Hablar en la conclusión sobre los problemas y visión del futuro