# Proyecto Unity
# Videojuego

Doriana Angélica Da Costa Magello

Programación multimedia y dispositivos móviles

Índice

# Introducción

Unity es una herramienta muy popular para los videojuegos, en este proyecto le presento mi proyecto, un juego en 2D que aprovecha las capacidades de Unity para ofrecer una experiencia envolvente y emocionante.

El jugador tiene la capacidad de caminar, correr y saltar, permitiéndole explorar el entorno y superar obstáculos con destreza.

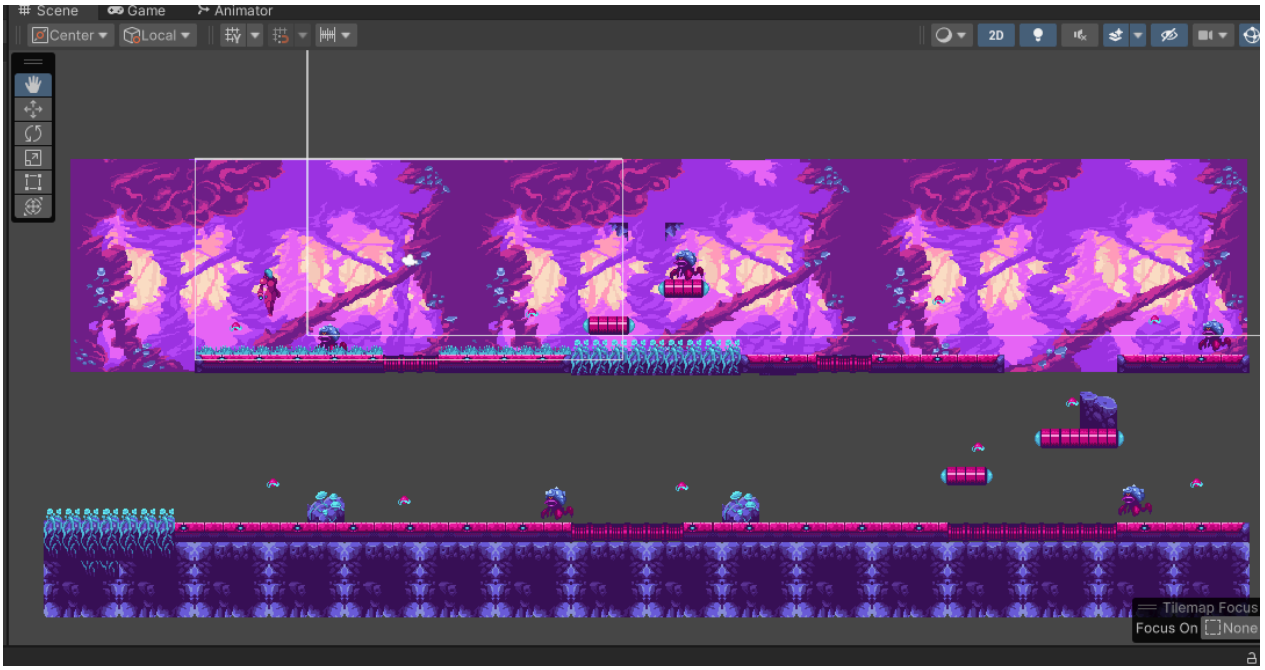
El camino del jugador está plagado de peligros, ya que enemigos acechan en cada esquina. Estos enemigos son una amenaza constante, ya que infligen daño al jugador y pueden poner en peligro su progreso. Con solo tres vidas disponibles, el jugador debe ser cauteloso y hábil para evitar ser derrotado por los enemigos.

La misión del jugador, es recolectar diez power up dispersos por el mundo del juego, sin que los enemigos lo maten, también el videojuego cuenta el tiempo transcurrido.

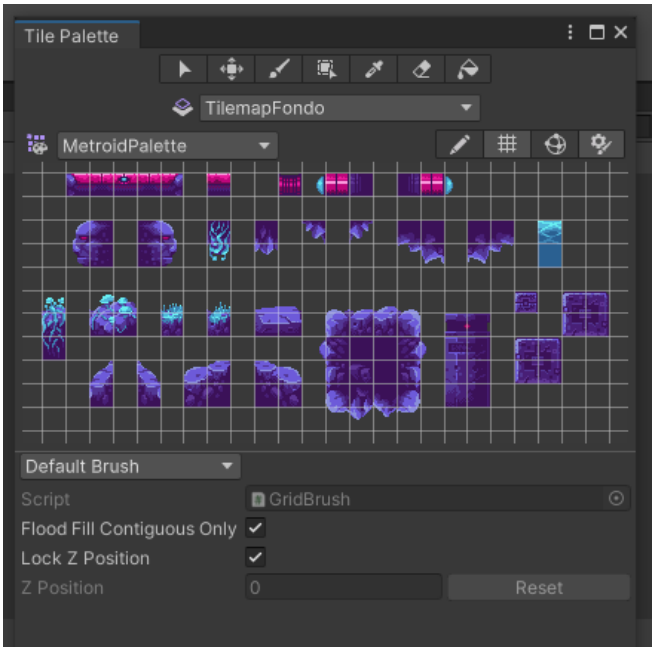
El mundo del juego está diseñado, con atención al detalle en cada sección. Desde la decoración hasta las colisiones y los fondos, cada elemento contribuye a la inmersión del jugador en este universo en 2D.

## Interacción con la aplicación

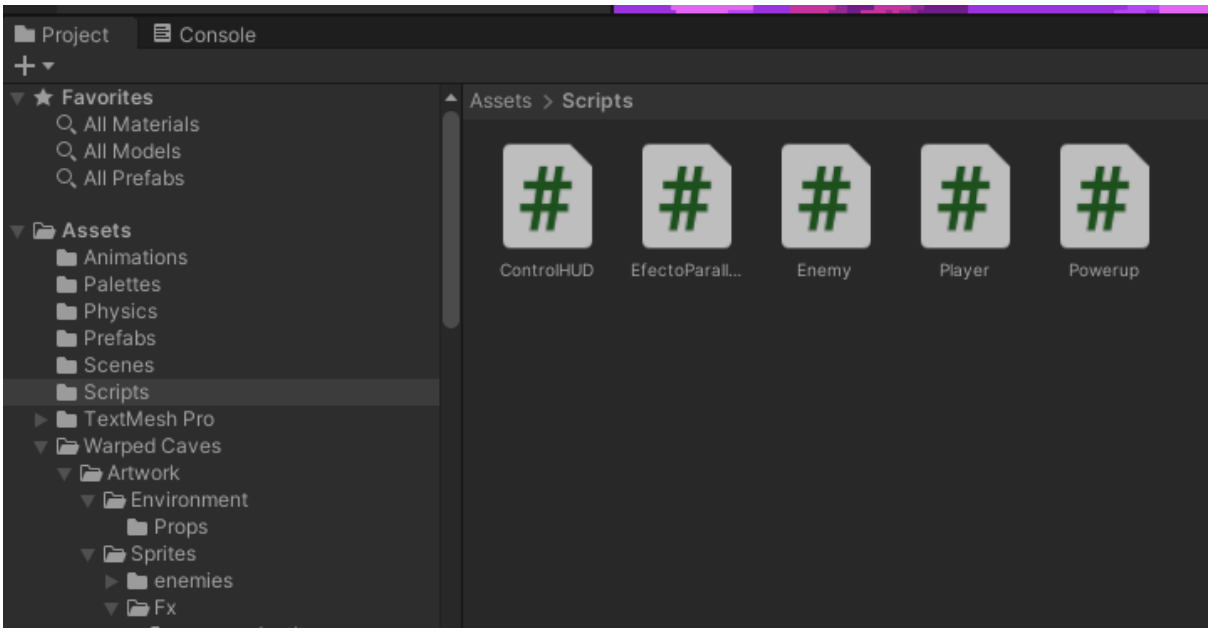
**Juego terminado:**
Se puede observar los dos niveles del juego, con sus enemigos y los power up que hay que recolectar.




**Paleta utilizada:**
Para la decoración y colisión del videojuego.



**Scripts utilizados:**
Como el del jugador, el del enemigo, el del power up, el efecto parallax y el control HUB.




**Textos en el juego:**

- Muestra las vidas del jugador.
- El número de power ups en total, cada vez que el jugador va recolectando se van restando al número de los que le faltan.
- El tiempo transcurrido.

**Juego ejecutado:**

Se observa el juego ejecutado, con sus textos, el enemigo, el jugador.


**Segundo nivel del videojuego:**

Se puede subir al otro nivel mediante de los siguientes objetos

## Explicación del código:

**Clase Control HUB:**

Permite controlar y actualizar un HUD (Head-Up Display) El HUD es la interfaz gráfica que muestra información importante al jugador mientras está jugando, como el número de vidas, el tiempo restante y la cantidad de power-ups disponibles.

Define los elementos del HUD que se van a actualizar dinámicamente durante el juego. En este caso, se definen tres elementos de texto utilizando la clase TextMeshProUGUI de Unity, que proporciona funcionalidades avanzadas para mostrar texto en la pantalla de manera más flexible y eficiente que el texto por defecto.

Los métodos públicos SetVidasTxt, SetTiempoTxt y SetPowerUpsTxt se encargan de actualizar cada uno de los elementos de texto del HUD con la información correspondiente. Estos métodos toman parámetros como el número de vidas, el tiempo restante y la cantidad de power-ups, respectivamente, y actualizan el texto mostrado en pantalla con esta información.

Por ejemplo, SetVidasTxt concatena la palabra "Vidas: " con el número de vidas y actualiza el texto del elemento VidasTxt, mientras que SetTiempoTxt calcula los minutos y segundos a partir del tiempo total y los formatea en el formato "mm:ss" para actualizar el texto del elemento TiempoTxt.

**Clase del jugador:**

Este script representa el comportamiento del jugador en el juego.

La clase llamada Player que hereda de MonoBehaviour, lo que significa que este script puede ser adjuntado a objetos en Unity y utilizado como un componente.

Variables públicas:
numVidas: Representa el número de vidas del jugador.
velocidad: La velocidad de movimiento horizontal del jugador.
fuerzaSalto: La fuerza aplicada al realizar un salto.
puntuacion: La puntuación actual del jugador.
TiempoNivel: El tiempo límite para completar el nivel.

Variables privadas:
fisica: Almacena el componente Rigidbody2D del jugador para controlar la física.
sprite: Almacena el componente SpriteRenderer del jugador para controlar su apariencia.
animacion: Almacena el componente Animator del jugador para controlar sus animaciones.

vulnerable: Indica si el jugador es vulnerable a los enemigos en este momento.
tiempoInicial: Almacena el tiempo en el que se inició el nivel.
tiempoEmpleado: Almacena el tiempo transcurrido desde el inicio del nivel.
Start(): Se llama una vez al inicio del juego. Aquí se inicializan las variables y se obtienen las referencias a los componentes necesarios.

FixedUpdate(): Se llama en intervalos fijos y se utiliza para manejar la física del jugador, como el movimiento horizontal.

Update(): Se llama en cada frame del juego. Aquí se manejan las acciones del jugador, como saltar y cambiar la dirección del sprite.

animarJugador(): Un método privado que controla las animaciones del jugador dependiendo de su estado (saltando, corriendo o en reposo).

TocandoSuelo(): Un método privado que verifica si el jugador está tocando el suelo utilizando un raycast hacia abajo.

OnTriggerEnter2D(Collider2D collision): Se llama cuando el jugador entra en contacto con un objeto con un collider activo. Aquí se maneja la lógica de colisión con los enemigos.

HacerVulnerable(): Un método privado que hace al jugador vulnerable después de un breve período de invulnerabilidad.

finJuego(): Un método privado que verifica si se ha alcanzado el final del juego, ya sea por quedarse sin tiempo o por quedarse sin vidas.

IncrementarPuntos(int cantidad): Un método público que permite aumentar la puntuación del jugador.

**Clase del enemigo:**

Define una clase llamada Enemy que hereda de MonoBehaviour, lo que significa que puede ser adjuntada a objetos en Unity y utilizada como un componente.

public float velocidad;: Una variable pública que determina la velocidad de movimiento del enemigo.

public Vector3 posicionFin;: Una variable pública que indica la posición a la que el enemigo se dirige.
private Vector3 posicionInicio;: Una variable privada que almacena la posición inicial del enemigo.
private bool movimientoAFin;: Una variable privada que indica si el enemigo está moviéndose hacia la posición final (true) o hacia la posición inicial (false).

void Start(): Este método se llama una vez al inicio del juego o cuando se activa el objeto. En este caso, se inicializa la variable posicionInicio con la posición actual del enemigo y se establece movimientoAFin en true para que comience moviéndose hacia la posición final.

void Update(): Este método se llama en cada frame del juego. Aquí se controla el movimiento del enemigo.

Si movimientoAFin es true, el enemigo se mueve hacia posicionFin utilizando Vector3.MoveTowards, que mueve gradualmente el objeto desde su posición actual hacia un destino, con una velocidad determinada por velocidad y Time.deltaTime, que asegura que el movimiento sea suave independientemente de la velocidad del juego. Si el enemigo alcanza posicionFin, se cambia movimientoAFin a false.

Si movimientoAFin es false, el enemigo se mueve hacia posicionInicio. Si alcanza posicionInicio, se cambia movimientoAFin a true, haciendo que el enemigo vuelva a moverse hacia posicionFin.

**Clase EfectoParallax:**

Variables públicas:
efectoParallax: Determina la cantidad de efecto de parallax que se aplicará al fondo. Un valor más alto producirá un parallax más pronunciado.
Variables privadas:

Cámara: Almacena la referencia al transform de la cámara principal.
ultimaPosicionCamara: Almacena la última posición de la cámara registrada.
Start(): Se llama una vez al inicio del juego. Aquí se inicializan las variables, obteniendo la referencia a la cámara principal y registrando su posición inicial.

LateUpdate(): Se llama en cada frame después de que todas las actualizaciones hayan sido realizadas. Esto se usa para asegurar que el parallax se aplique después de que la cámara haya terminado de moverse en el frame actual.

Se calcula el vector movimientoFondo restando la posición actual de la cámara (camara.position) de su última posición registrada (ultimaPosicionCamara). Esto nos da el movimiento relativo de la cámara desde el último frame.

Luego, se mueve el objeto de fondo en la dirección opuesta al movimiento de la cámara, multiplicando el movimiento horizontal (movimientoFondo.x) por el factor efectoParallax. Esto produce el efecto de parallax, donde los objetos más lejanos se mueven más lentamente que los objetos más cercanos.

Finalmente, se actualiza ultimaPosicionCamara con la posición actual de la cámara, para ser utilizada en el siguiente frame.

**Clase Powerup:**

Variables públicas:

cantidad: Representa la cantidad de puntos o beneficios que otorga este power-up al jugador al ser recogido.
OnTriggerEnter2D(Collider2D collision): Se llama cuando el objeto con el script colisiona con otro objeto que tenga un collider activo. Aquí se maneja la lógica cuando el power-up es recogido por el jugador.

Se verifica si el objeto con el que colisiona tiene la etiqueta "PlayerPrincipal" (presumiblemente el jugador principal del juego). Si es así, se obtiene una referencia al componente Player del objeto jugador y se llama al método IncrementarPuntos con la cantidad especificada en la variable cantidad.

Luego, se destruye el objeto power-up (gameObject) para que desaparezca de la escena después de ser recogido.