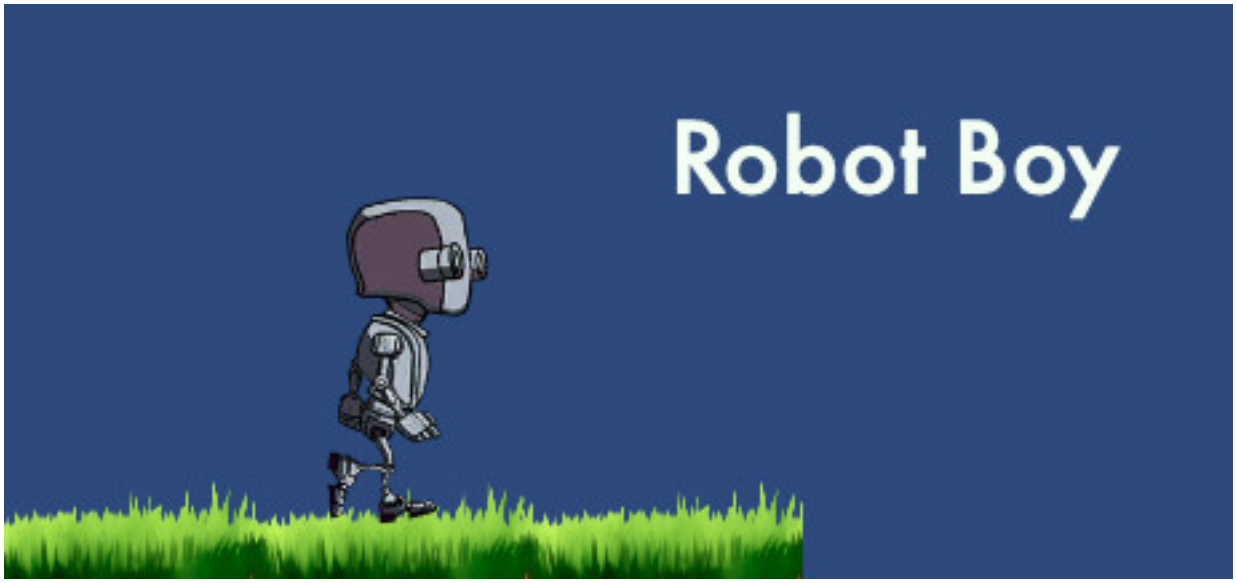
Ciclo: Animaciones 3D, Juegos y Entornos Interactivos **Curso:** 2021/22

Módulo: Desarrollo de Entornos Interactivos Multidispositivo (tarde)

PRÁCTICA 03 JUEGO DE PLATAFORMAS 2D

Vamos a crear un juego basado de plataformas 2D utilizando los sprites incorporados en los Standard Assets de Unity (que puedes encontrar en el Drive del curso):



1. Editar los sprites y crear las animaciones

Instala el paquete de Sprite Editor 2D y recorta las hojas de sprites que encontrarás en la carpeta de materiales. Los sprites del personaje utilizan un tamaño de celda de 275x275px

Animaciones disponibles para el robot:

- Reposo
- Saltar
- Agacharse (quieto y caminando)
- Muerte.
- Rodar (opcional)

Algunas acciones, como saltar o morir, se podrían acceder desde cualquier estado (Any State)

Además, tendrás que crear 2 animaciones para la torreta: en reposo y disparando.

Animation & Animator

Mediante la herramienta de *Animation* crea todas las animaciones necesarias y guárdalas para manejarlas en el Animator Controller. Ajusta los tiempos entre fotogramas para que queden realistas, y desactiva tanto el Loop como el "Has Exit Time" donde sea necesario.

Crea los parámetros necesarios para controlar las animaciones, del tipo más adecuado (trigger, float, bool). En algunas acciones tendrás que añadir más de una condición.

2. Control del personaje

El personaje se desplazará por el escenario, a izquierda y derecha y hacia arriba mediante saltos.

Movimientos laterales

Puedes usar el parámetro velocity perteneciente al RigidBody para desplazar al personaje. Al aplicar un movimiento lateral, asegúrate de que las fuerzas verticales permanecen intactas:

```
desplX = Input.GetAxis("Horizontal");
rb.velocity = new Vector2(desplX * maxSpeed, rb.velocity.y);
```

La velocidad del personaje deberá variar si estamos corriendo o agachados (podemos obtener esos datos del Animator).

Trata de crear una variable booleana que controla si el personaje está vivo o no, y que determina si se puede controlar.

Salto

Se debe evitar el doble salto, para ello deberás controlar si el personaje está tocando el suelo. Puedes usar un colisionador que detecte cuándo está tocando el suelo.

Giro del personaje

Para que el personaje mire a un lado o a otro, deberás cambiar su escala en X mediante "transform.ocalScale. Puedes usar la siguiente fórmula que invierte siempre el valor de X :

```
Vector3 theScale = transform.localScale;
theScale.x *= -1;
transform.localScale = theScale;
```

Recuerda: no debe hacerlo constantemente, solo una vez al girarse, por lo que deberás crear una booleana y usarla a modo de interruptor (solo se gira si no está girado)

Agachado

En la animación de agachado (y roll si se usa) deberá actualizarse el colisionador del personaje para que no le puedan impactar los proyectiles a la altura de la cabeza

3. Armas y creación de una IA

Usando las hojas de sprites, deberemos crear enemigos en forma de torretas que disparan proyectiles. Para ello, deberás crear al menos 2 animaciones: una en reposo y otra disparando.

Crea tantos prefabs como tipos de torretas que necesites. Deberán tener los siguientes comportamientos:

Torretas apuntando

Para que la torreta gire mirando al jugador NO podemos usar el "LookAt" de Transform porque eso hará que gire en 3 ejes. Podemos usar este código (donde "player" es una variable Transform

que contendrá a nuestro personaje, y que podemos modificar levemente para que "apunte a la cabeza"):

```
Vector3 dir = player.position - transform.position;
float angle = Mathf.Atan2(dir.y, dir.x) * Mathf.Rad2Deg;
transform.rotation = Quaternion.AngleAxis(angle, Vector3.forward);
```

Torretas disparando

Deberás crear un sprite con el proyectil y añadirle el movimiento a una velocidad y en una dirección concretas (posiblemente hacia la derecha). Crea un empty Object de referencia en el cañón de la torreta, donde instanciarás el proyectil. El proyectil deberá desaparecer al impactar con cualquier objeto (o cuando pasa un tiempo, para evitar que se quede pululando por el escenario).

Lanza un evento en la animación de la torreta que dispara el proyectil en el momento adecuado. Deberás crear un método que haga aparecer el proyectil mediante una instanciación.

Añade un colisionador al proyectil que si impacta con el Robot, ejecutará las acciones necesarias para que sea destruido.

Activar las torretas mediante "trampas"

Recuerda: para detectar colisionadores debes usar los métodos para 2D:

```
private void OnTriggerEnter2D(Collider2D other) { }
```

Podemos activar todas las torretas a la vez, creando un array de GameObjects que luego llenaremos con el método FindGameObjectsWithTag

```
torretas = GameObject.FindGameObjectsWithTag("Torreta1");
```

Al activar la trampa, podemos hacer un bucle mediante foreach, accediendo a cada torreta que tenga ese tag y lanzando un mensaje que ejecute el método adecuado:

```
foreach (GameObject torreta in torretas)    { … }
```

Activar las torretas mediante IA

Otra opción para que se activen las torretas es mediante un RayCasta que lance un rayo en la dirección adecuada, y si impacta con el jugador, que se inicie la animación de disparar (con los eventos asociados para instanciar el proyectil).

Crea una variable que determine el alcance de ese raycast, para poder controlarlo mejor.

```
Float alcance = 20f;
RaycastHit2D hit = Physics2D.Raycast(cannon.position, -Vector2.right, alcance);
if (hit.collider != null) { … }
```

NOTA: puedes evitar que el raycast "impacte" con los proyectiles ya lanzados, incluyendo estos en la capa 2 ("Ignore Raycast")

4. Escenario

Crea los suelos y los techos que creas convenientes, con sus propios colisionadores.

Puedes usar el ejeZ y una cámara con perspectiva para organizar los elementos. Y si colocas varios elementos en la misma posición del eje Z, deberas organizarlos mediante Sorting Layers.

Añade las plataformas al juego. Pueden ser de varios tipos:

Plataformas estáticas

Usa el efector de plataformas para que el usuario pueda atravesarlas desde abajo.

Si vas a crear varias iguales, es recomendable crear un prefab con las características adecuadas (aunque las añadas directamente al escenario, sin instanciar)

Plataformas animadas

Si las plataformas se animan, tendrás que hacer que el robot se anime con ellas. Para ello, puedes hacer que si la está tocando, se convierta en un elemento hijo de ella:

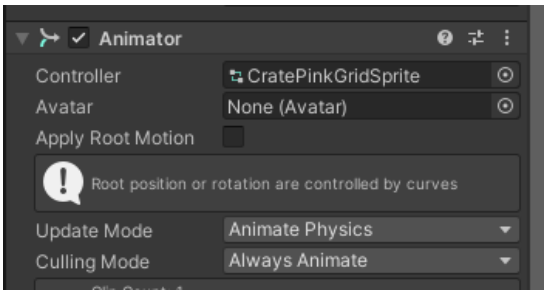
```
if(collision.gameObject.tag == "Plataforma")
```

```
{
    transform.parent = collision.gameObject.transform;
}
```

Y si deja de tocarlas, deberá pasar a no tener padre

```
transform.parent = null;
```

IMPORTANTE: como el Robot se mueve por físicas, es importante que la plataforma esté también animada por físicas. Selecciona el componente animator y activa el Update Mode en Animate Physics:



Seguimiento de la cámara

Puedes usar los script vistos para que siga al objeto. Pero es importante que si el jugador se mueve usando "FixedUpdate", ya que utiliza físicas, es importante que la cámara también lo use a la hora de establecer su posición respecto al jugador.

Entrega y evaluación

El juego se desarrollará en clase. La fecha límite de entrega es el **lunes 20 de diciembre.**

Comprime tu proyecto Unity en un archivo ZIP, **sin incluir la carpeta "Library" del proyecto para que no ocupe demasiado** y súbelo a la plataforma (el límite es 50MBs, asegúrate de que tu juego comprimido no pesa más).

ApellidoNombre_PR03.ZIP

Este ejercicio **será calificado con nota**, siguiendo los siguientes criterios:

1. Animaciones y control del personaje (4 puntos)
2. Enemigos e IA (3 puntos)
3. Escenario: plataformas y colisionadores (2 puntos)
4. Orden en el proyecto y el código (1 punto)