Motores de videojuegos

Creación de juegos con Unity 2D

Marcos Núñez Celeiro

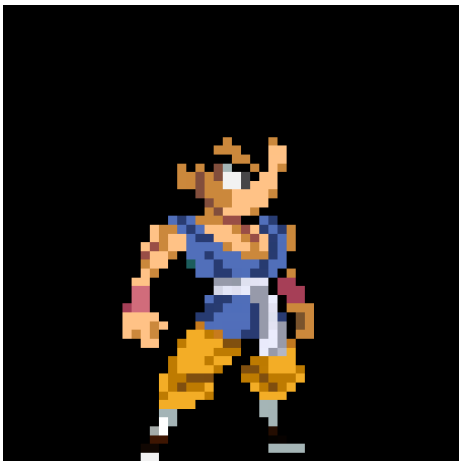
13 de Febrero del 2022

Conceptos (I)

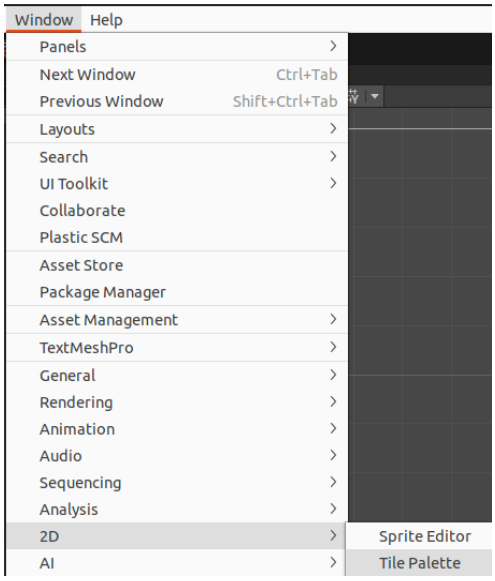
- Sprites: imágenes.
- Spritesheets: listas de imágenes que están juntas en un mismo fichero (divisibles en partes). Los *spritesheets* pueden trocearse para introducirlos en el juego o crear animaciones. Por ejemplo:

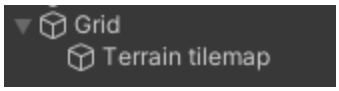
Spritesheets de "Contra" (NES). Fuente: The Spriters Resource

- Unity Asset Store: tienda con recursos para los juegos (fondos, personajes, enemigos, plataformas, etc.). Se accede desde Window -> Asset store o desde Window -> Package manager.
- Frames por segundo (fps): número de imágenes que se muestran en 1 segundo (típicamente en las películas suele ser 24 o 30 y en juegos varía).
- Juego top-down: juego que ves desde arriba (un pokemon antiguo, por ejemplo).
- Pixel art: estilo que utiliza imágenes pixeladas.



- Tile palette: paleta con trozos de mundo que queremos reutilizar a lo largo de nuestro juego (p. ej: trozos de plataforma, de ladrillo, etc.).





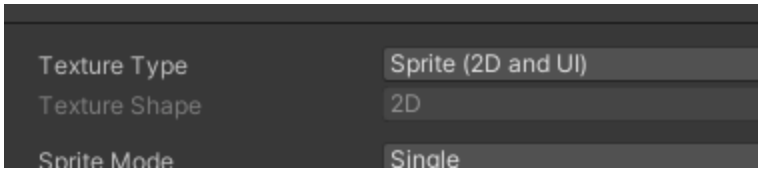
Recursos descargables

- Variados (Kenney, hay ambos gratuitos y de pago): https://www.kenney.nl/assets
- 2D, 3D, sondios... (uso libre): https://opengameart.org/
- Fuentes: https://www.dafont.com/ y https://fonts.google.com/
- Unity Asset Store.

Manipulación de imágenes

Sprites y spritesheets

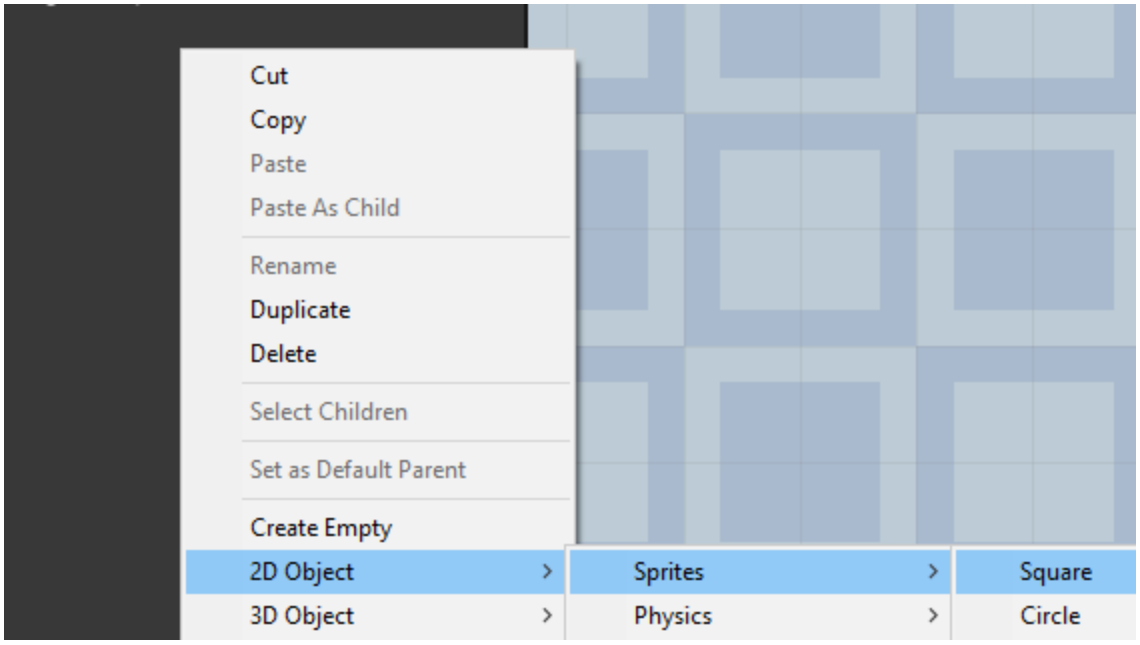
Cuando arrastramos un dibujo cualquiera a un proyecto Unity lo transforma en un “Sprite”, que es básicamente esa misma imagen pero que puede ser dibujada en el juego.



Un dibujo cualquiera que puede ser dibujado en el juego es un sprite. Una lista de dibujos unidos es un spritesheet y puede ser troceado (hay que indicarle los píxeles que debe pillar en cada unidad (recuadro) de unity y partirlo como “multiple” e ir al sprite editor.

Añadir un objeto sprite al juego

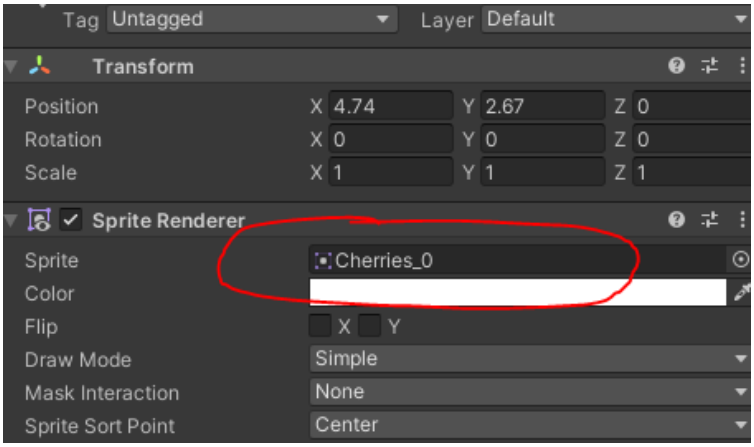
Para añadir, por ejemplo, un jugador o un ítem, basta con crear un nuevo “Sprite” desde la jerarquía:



Botón derecho en la jerarquía → 2D Object → Sprites → Square

Esto crea un **GameObject** con un componente **SpriteRenderer**. Podríamos hacer exactamente lo mismo a mano creando el GameObject y luego añadiendo el componente. El resultado sería el mismo.

Para **cambiar el sprite** de un cuadrado a otra cosa basta con **arrastrar la imagen deseada al “Sprite”** del inspector **(arrastrar la imagen al círculo rojo**):

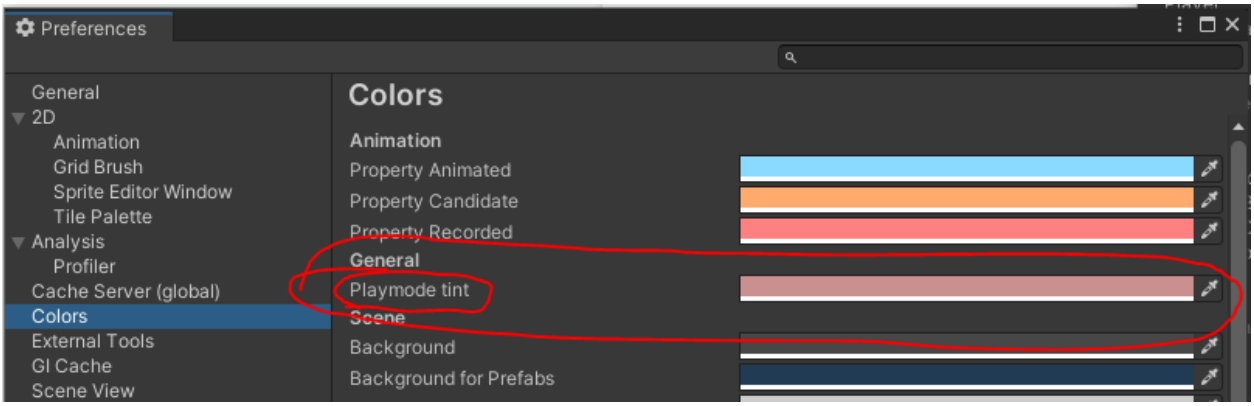


Configuración

Cambio de color en ejecución

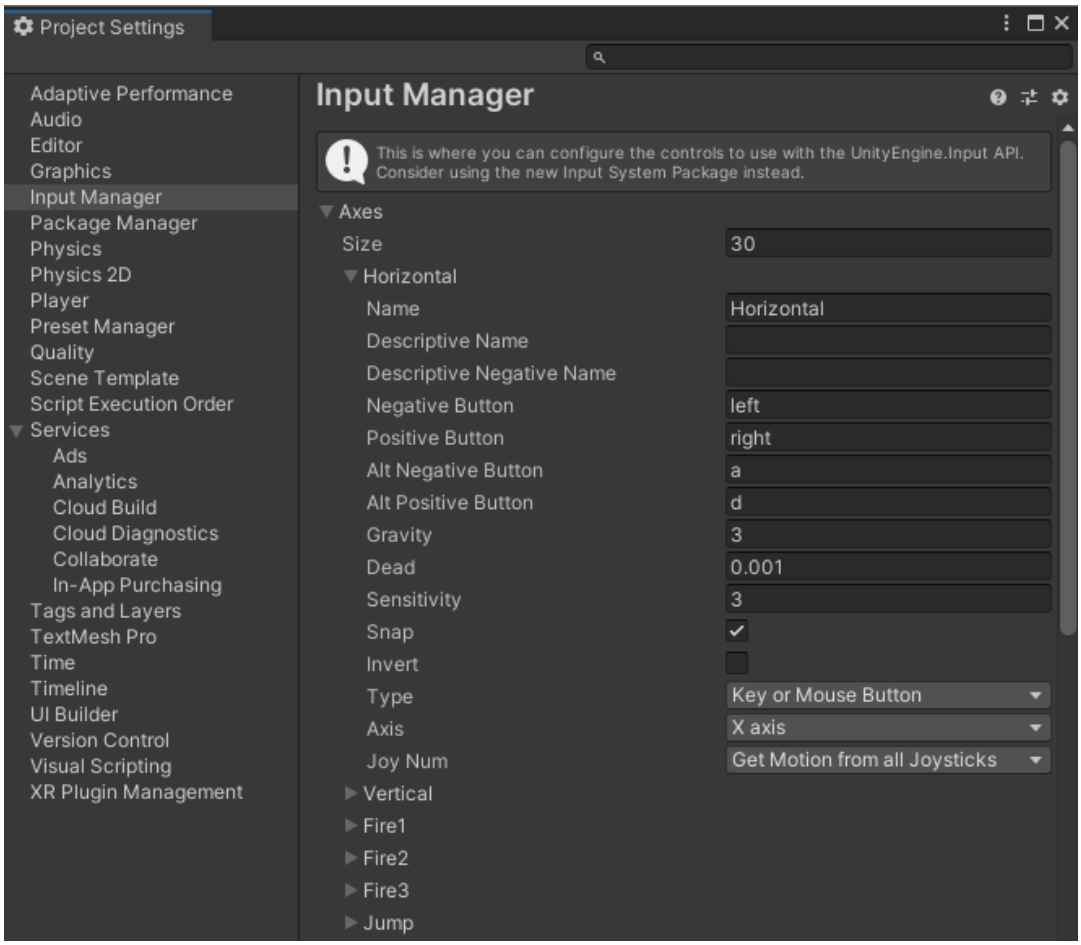
Si ejecutamos un juego y mientras estamos jugándolo cambiamos parámetros del mismo, estos no se cambian. Para identificar cuando estamos en modo “ejecución” es mejor cambiar el color del IDE:

Edit ➔ *Preferences* ➔ *Colors* ➔ *Playmode tint*



Teclas

Para modificar las teclas en tu juego puedes ir a: *Edit* ➔ *Project* ➔ *Input Manager*



Movimiento de un personaje

Conceptos básicos

Tipos de datos:

- **Vector2** (coordenadas x, y)
- **Vector3** (coordenadas x, y, z)

Existen dos métodos básicos:

- **Start**(): se ejecuta **solo una vez, al ejecutar el juego**.
- **Update**(): se ejecuta **una vez por frame** (es decir, tipicamente unas 30 a 60 veces por segundo).

```
public class PersonajeController : MonoBehaviour
```

```
{
      // Start is called before the first frame update
      void Start()
      {

      }

      // Update is called once per frame
      void Update()
      {

      }
}
```

Mensajes

Debug.Log:

```
void Start()

{
    Debug.Log("Hello, the game begins!");
}
```

Movimiento sin físicas

```
void Update()
{
    float horizontal = Input.GetAxis("Horizontal");
    Debug.Log(horizontal);

    Vector2 position = transform.position;
    position.x = position.x + .1f * horizontal;
    transform.position = position;
}
```

Teclas

Salto del personaje

Detección (en cualquier momento) de la tecla espacio: - Hardcoded

```
Input.GetKey("space")
```

Detección (sin mantener presionado) de tecla espacio: - Hardcoded

```
if (Input.GetKeyDown("space")) {
```

Saltar (obteniendo el RigidBody y asignándo una velocidad: - Hardcoded

```
if (Input.GetKeyDown("space")) {
    GetComponent<Rigidbody2D>().velocity = new Vector3(0, 14, 0);
}
```

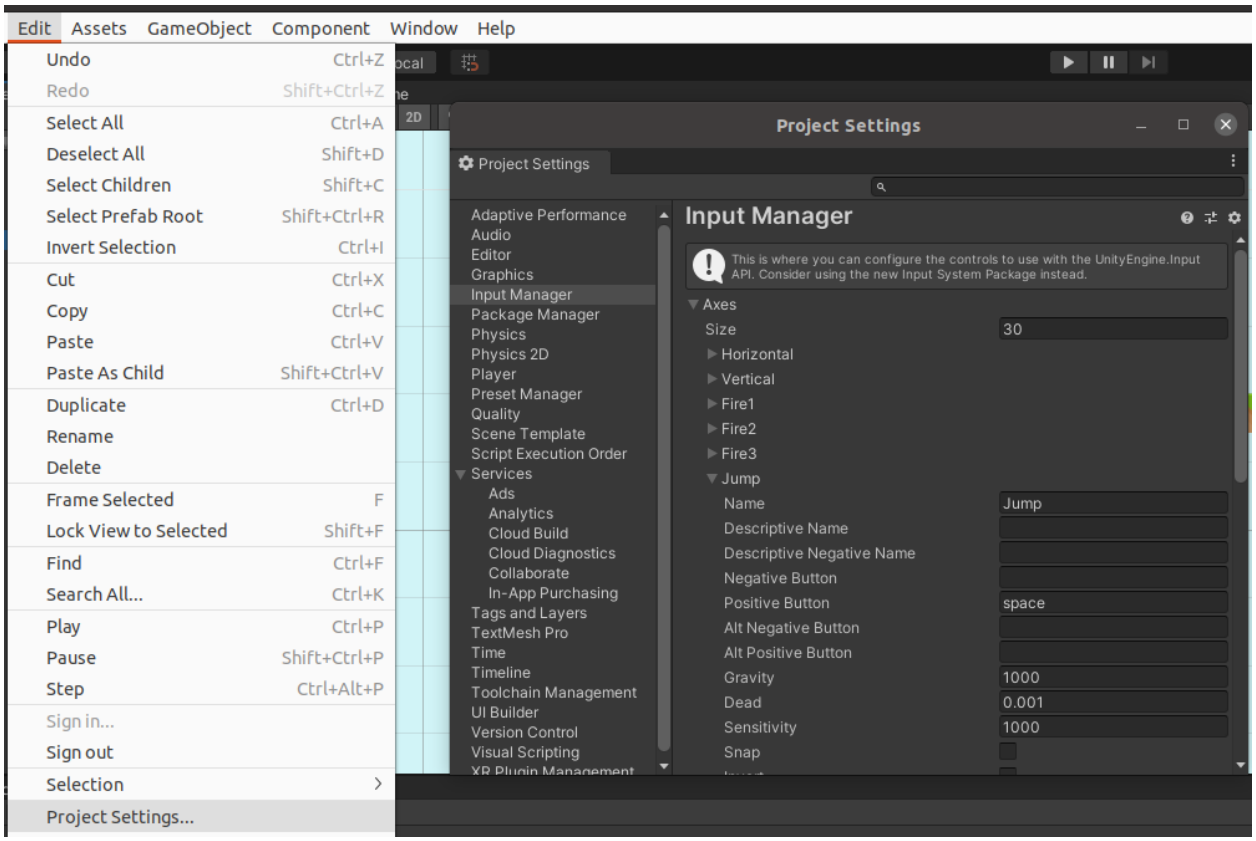
Detección (forma correcta, no hardcoded) de pulsado de tecla de “SALTO”:

```
if (Input.GetButtonDown("Jump")) {
   GetComponent<Rigidbody2D>().velocity = new Vector3(0, 14, 0);
}
```

De esta manera, si cambio la tecla de “salto”, no tengo que cambiar el código.

Movimiento horizontal y vertical

Para cambiar las teclas se hace en el “Input Manager”:



Ahí disponemos de los ejes (Axes) que pueden ser “Horizontal” y “Vertical”. Además, vemos que el salto recibe el nombre de “Jump” (esto es lo que nos permite capturar la acción de saltar y no una tecla en concreto.

Movimiento horizontal (**GetAxis** y **GetAxisRaw**):

```
float dirX = Input.GetAxis("Horizontal");
float dirX = Input.GetAxisRaw("Horizontal");
```

Con el primero (GetAxis) el personaje frena lentamente al soltar. Con GetAxisRaw frena de golpe. El primero es probablemente más realista para juegos 3D, aunque depende de cada uno.

Captura de movimiento horizontal (izquierda-derecha) y salto en cada frame:

```
void Update()
{
   float dirX = Input.GetAxisRaw("Horizontal");

   // dirX toma valores -1 y 1 (lo multiplicamos por la velocidad que queramos)
   rigidBody.velocity = new Vector2(dirX * 7f, rigidBody.velocity.y);
```

```
    if (Input.GetButtonDown("Jump")) {
        rigidBody.velocity = new Vector2(rigidBody.velocity.x, 14);
    }
}
```

Además de esto, cambiamos el salto para que con “cada frame” no vuelva a tener 0 de velocidad en X, sino que tenga la actual velocidad en X (la que ya tenía en la anterior frame, es decir, no reiniciamos la velocidad).

Creación del mundo (Tilemaps y Tile Palettes)

Para crear el mundo la mejor forma es crear paletas de plataformas, nubes, terrenos, montañas y otros elementos que se deseen).

Los spritesheets son imágenes (sprites) pero que tienen muchos elementos en la misma imagen. Se pueden descomponer y crear una paleta con ellos para luego irlos insertando:

Window -> 2D -> Tile palette -> Create New Palette

Se arrastran ahí los sprites que se deseen.

Hay razones para hacer dos Tilemaps distintos (uno para el fondo y otro para el terreno). Esto es porque el fondo debería estar detrás del terreno (capas diferentes).

Físicas

- BoxCollider2D.
- RigidBody 2D.
- Masa (Mass) y escala de gravedad (Gravity Scale).

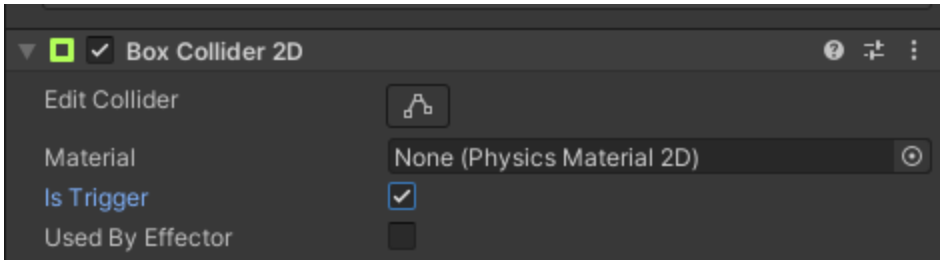
Colisiones

- **BoxCollider2D** (para el personaje, hace forma de rectángulo).

- **TilemapCollider** (automáticamente hace collider de los elementos del tilemap). Por rendimiento,
    - Añadir CompositeCollider2D y en TilemapCollider darle a “Used by composite”. Esto hace que los tilemapcolliders solo colisionen por los bordes externos y mejora el rendimiento.
    - Si queremos que un cuerpo físico (p. ej: un tilemap entero de un terreno) no se caiga por la gravedad, es necesario ponerle su rigidbody con el bodytype a “static”.

Triggering

- Para **detectar colisiones entre dos GameObjects pero que no afecten las físicas y se atraviesen** hay que marcar el atributo “Is Trigger”.



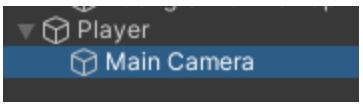
- **Evitar rotación del personaje**: En el RigidBody2D congelar (en constraints) el eje Z para que el personaje no rote.

Detección de colisión (OnTriggerEnter2D):

```
public class ItemCollector : MonoBehaviour
{
      private void OnTriggerEnter2D(Collider2D collision)
      {
          // Detecta la colision del GameObject actual con otro objeto
      }
}
```

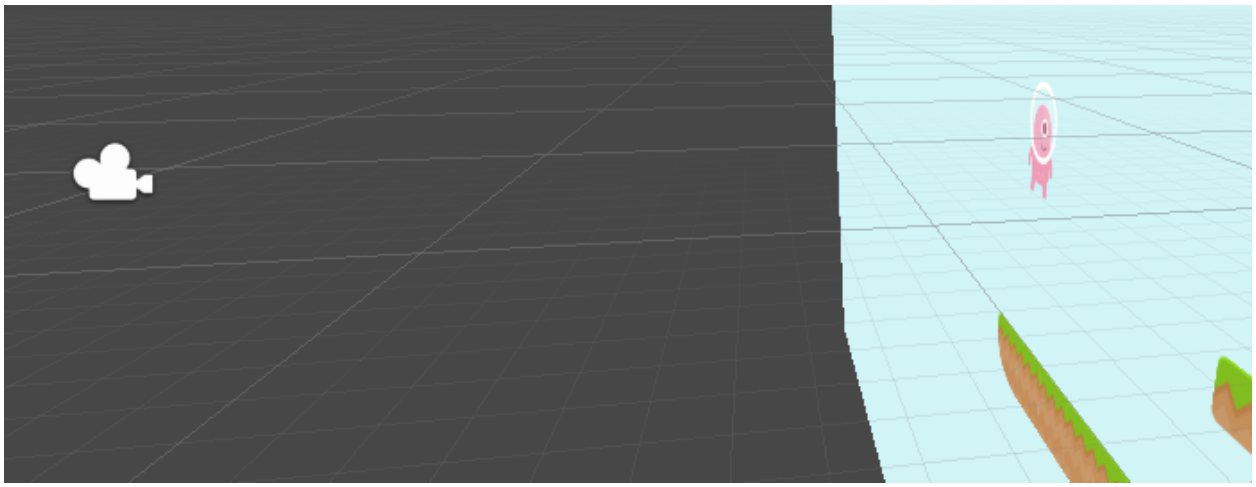
Cámara

Queremos que la cámara siga al jugador. Para que siga al jugador basta con ponerla como hija del mismo.



Además, una vez puesta como hija del jugador nos interesa centrarla (normalmente). Para ello, podemos hacerlo simplemente poniendo la X y la Y a 0 (estaría así en el eje central en relación al GameObject del jugador.

La cámara, por defecto, se encuentra a cierta profundidad en Z (para poder enfocar). Si ponemos modo 3D lo vemos así:

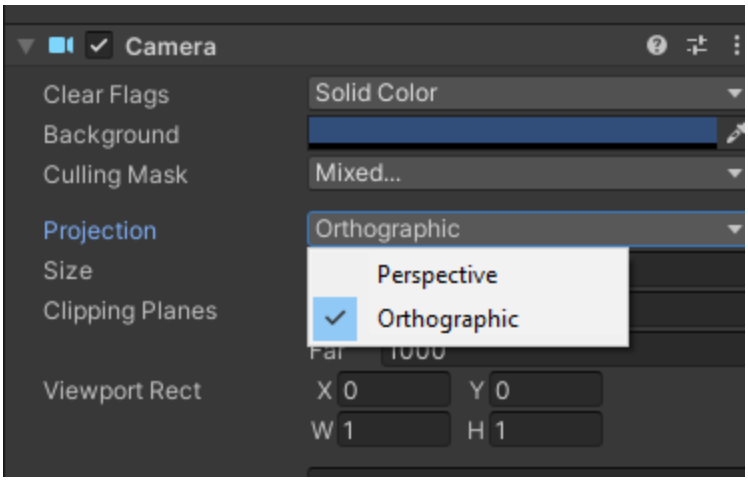


Si la acercamos demasiado (o la ponemos a 0) no podrá enfocar. Podemos darle el valor que queramos según la cercanía que queramos.

Tipos de cámara (en 2d y 3d)

Existen dos modelos de cámara

- **Orthographic**
- **Perspective**

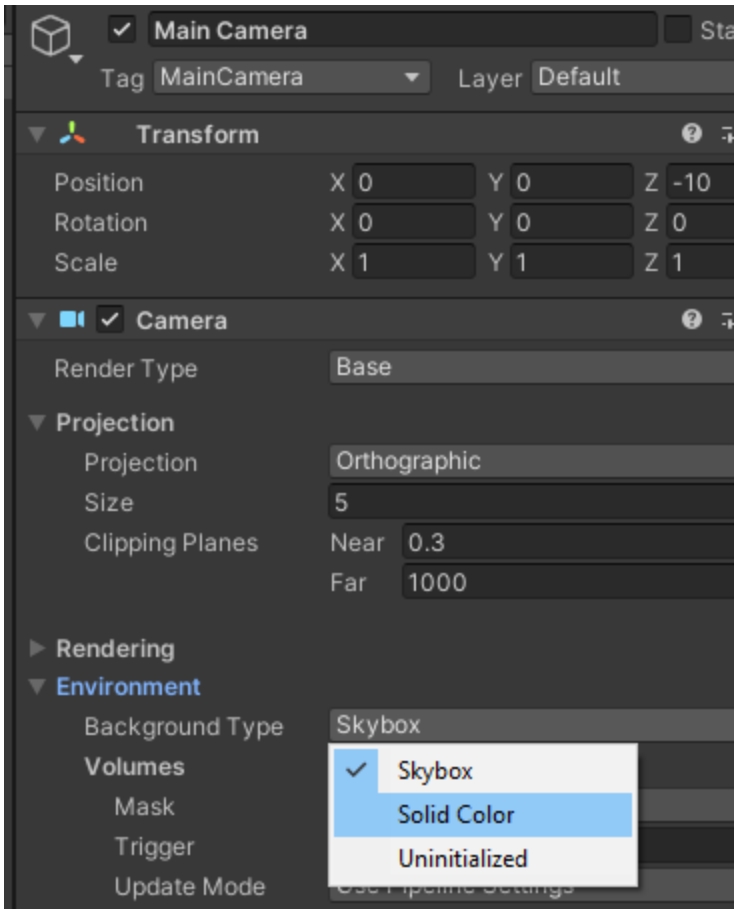


La cámara por defecto es ortográfica. La diferencia es que, usando la cámara en perspectiva, esta se ve como en el mundo real (si miramos una calle muy larga el fondo de la misma se va encogiendo, mientras que en ortográfica sería exactamente igual). En ortográfica (por defecto) no importa la distancia a la que está la Z del plano (siempre se verá con el mismo tamaño), mientras que en perspectiva, si alejas la Z todo se verá más pequeño.

Perspectiva es más bien de entornos 3D, mientras que la ortográfica se usa en 2D.

Cambiando el fondo

Por defecto el fondo es una especie de azul (algo así como el cielo). Esto se puede cambiar a un color fijo así:



Seleccionando en “Environment” la opción “Solid color”. De todas maneras lo óptimo es tener un fondo adecuado que poner.

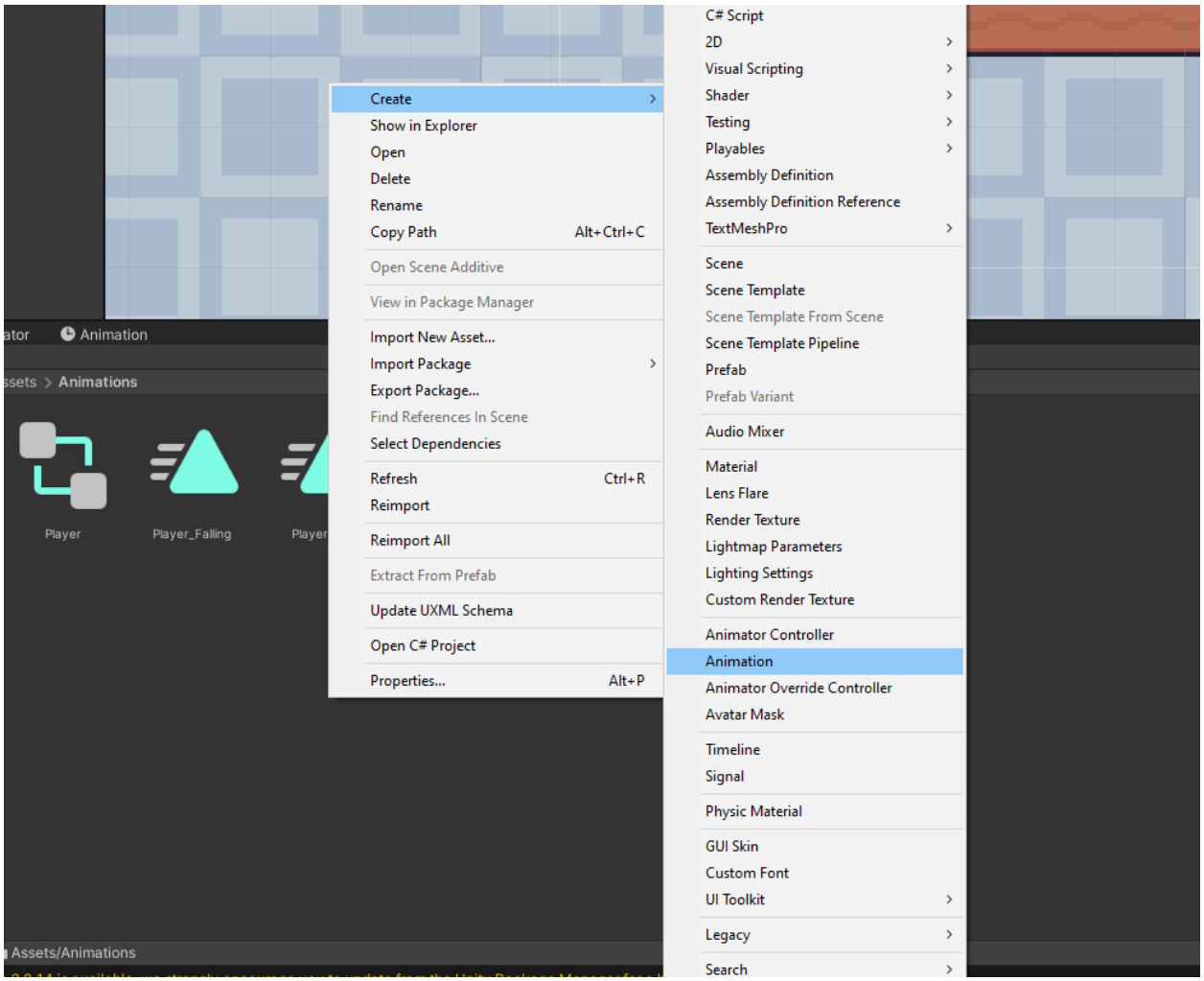
Animaciones

Existen dos ventanas principales:

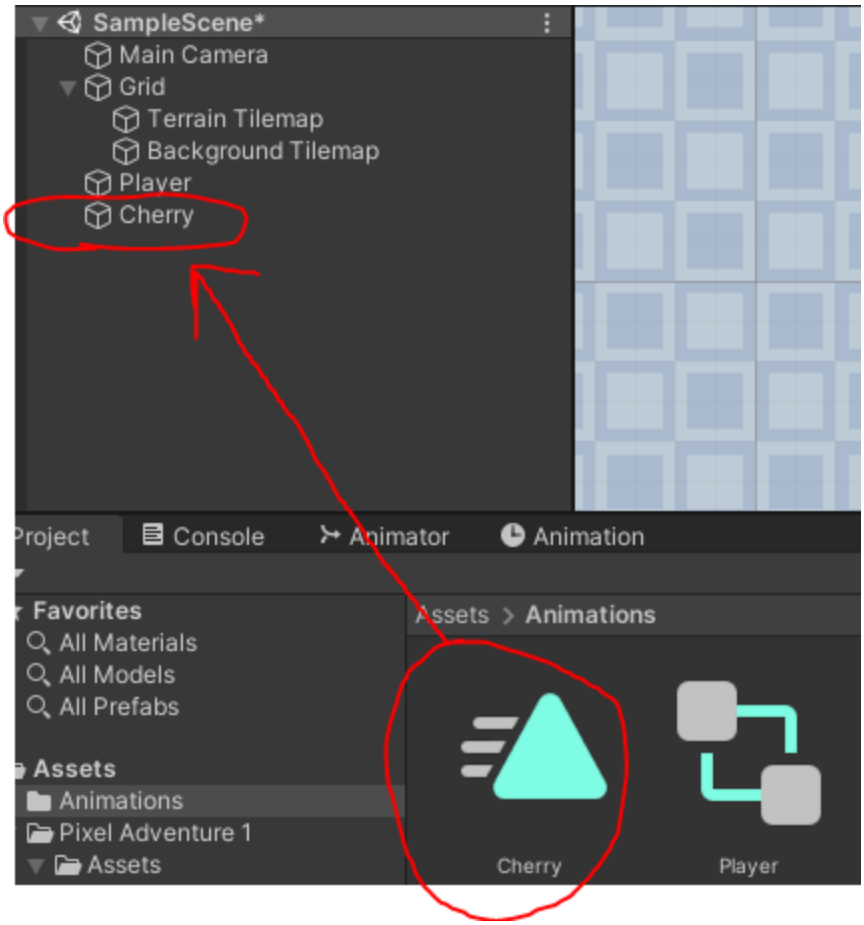
1. **Animation**: ponemos la lista de imágenes que queremos que se vayan ejecutando.
2. **Animator**: como y cuando ejecuto cada animación (p. ej: de estar parado puede pasar a estar corriendo, y de estar corriendo puede volver a pasar a estar parado. El animator es lo que conecta el animator al GameObject.

Para crear una animación:

1. Creamos una nueva animación (p. ej: en una carpeta “animaciones” en nuestro proyecto).

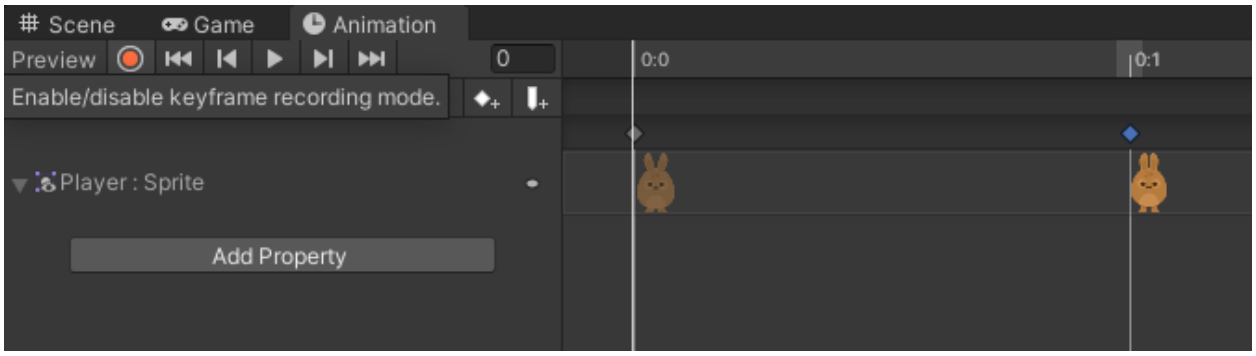


2. Arrastramos una lista de sprites (p. ej, tres imágenes del personaje andando) directamente al GameObject del jugador. Con esto nos guardará la animación.

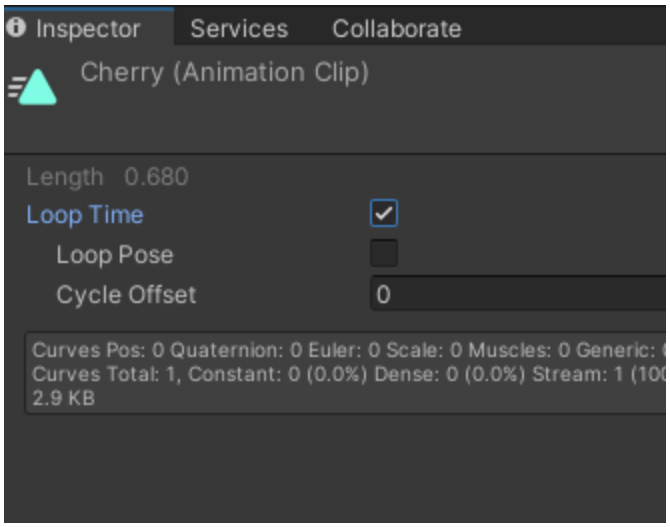


Si arrastramos un "Animation" se crea un "Animator"

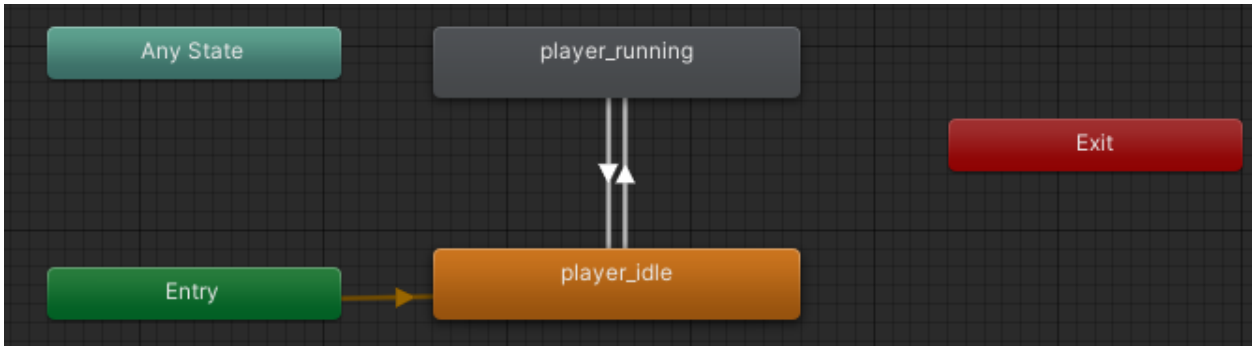
3. Window -> Animation -> Animation y adaptamos lo rápido que queramos que cambie entre imágenes.



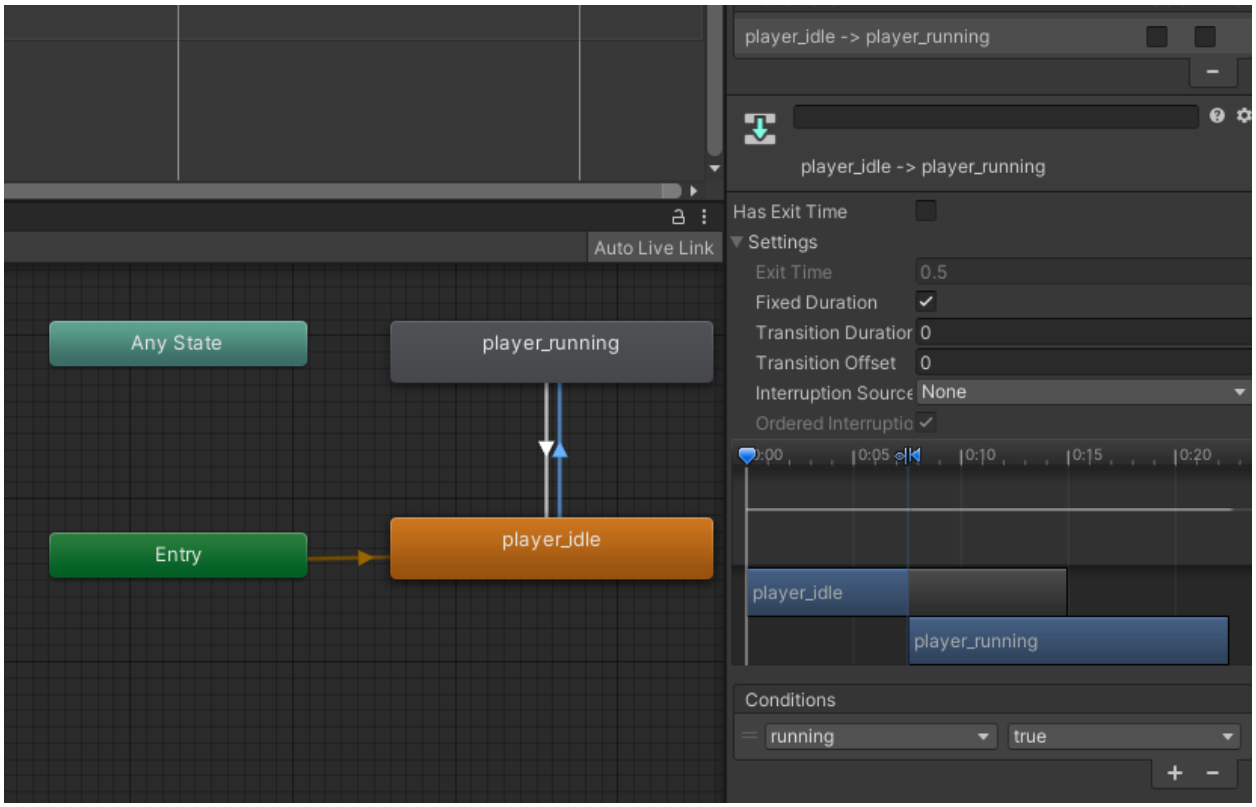
4. Para que al acabar la animación esta vuelva a empezar (normalmente es lo deseable, aunque no siempre) hay que indicar que se ejecute en bucle (loop). Para ello, pulsa en el elemento "Animation" que has creado y en el inspector activa "Loop Time":



5. Window -> Animation -> Animator y podemos hacer flechas entre estados.



6. Pinchamos sobre una de las flechas y configuramos el cambio de estado. Por ejemplo:
    a. Has exit time (unchecked).
    b. Transition duration (0). Esta opción queda mejor en 3D.
7. Pinchando también sobre una de las flechas (p. ej: de idle a running) ponemos en **conditions** una variable “running” = a *true*.



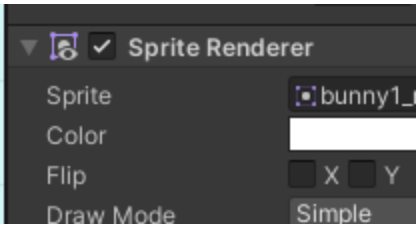
Control dinámico de las animaciones

Para esto utilizamos el objeto **Animator**.

```
private Animator anim;
...
anim = GetComponent<Animator>();
...
if (dirX > 0f) {
        anim.SetBool("running", true);
}
```

En el código anterior vemos cómo obtener el componente y cómo establecer una variable a true o false.

Girar el personaje (flip)



Para **girar el personaje** (por ejemplo, que corra para atrás) se hace un FLIP del Sprite Renderer.

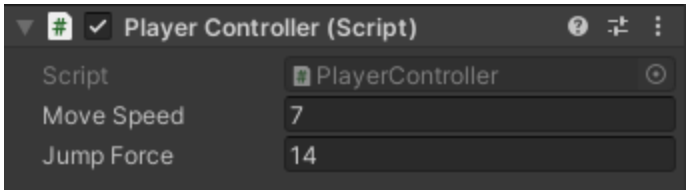
Para hacerlo en código, esto podría hacerse de la siguiente manera:

```
private SpriteRenderer sprite;
...
sprite = GetComponent<SpriteRenderer>();
...
sprite.flipX = true;
```

Exposición de variables

Para exponer variables a la interfaz de usuario de Unity y modificarlas más facilmente solo es necesario, o bien **ponerlas como públicas** (con lo cuál otros scripts podrían acceder a ellas) **o ponerlas como campos serializables**:

```
[SerializeField] private float moveSpeed = 7f;
[SerializeField] private float jumpForce = 14f;
```



Solución de problemas comunes y bugs

Evitar salto en el aire

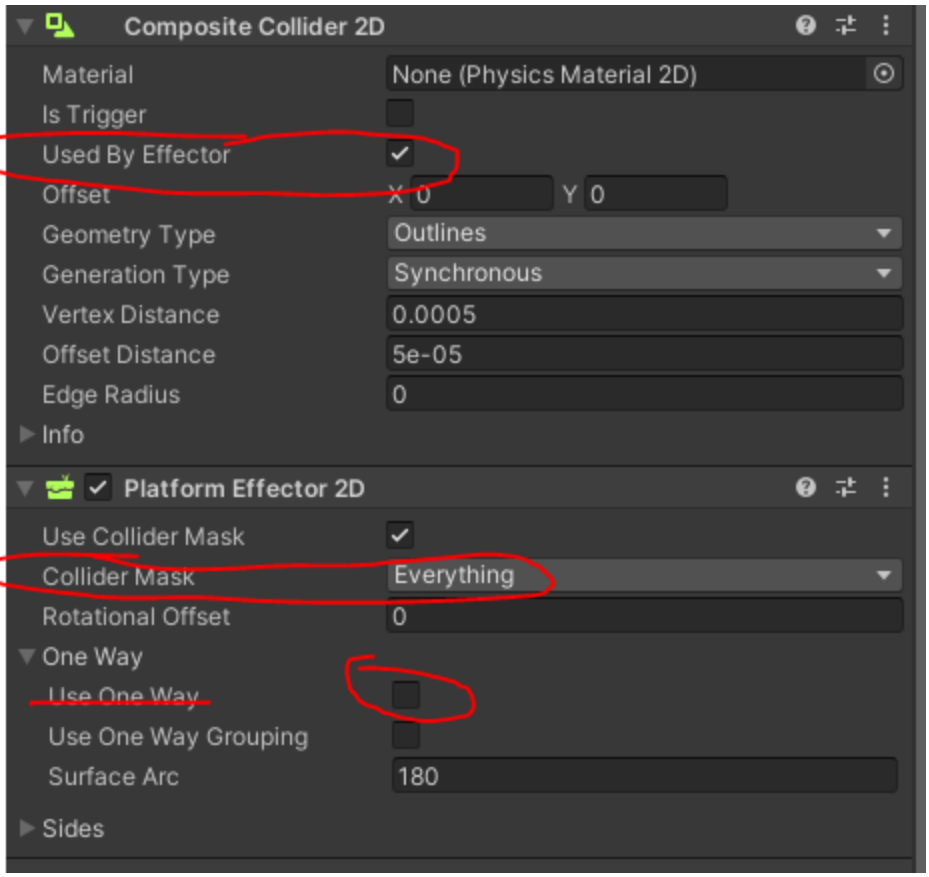
Puedes evitar el salto mientras se está en el aire de varias formas, pero utilizar un método como este es sencillo y rápido (aunque puede ser difícil comprenderlo en un principio):

```
private bool isGrounded() => Physics2D.BoxCast(
      coll.bounds.center,
      coll.bounds.size,
      0f,
```

```
        Vector2.down,
        .1f,
        jumpableGround
);
```

Caer al chocar en el aire contra una pared

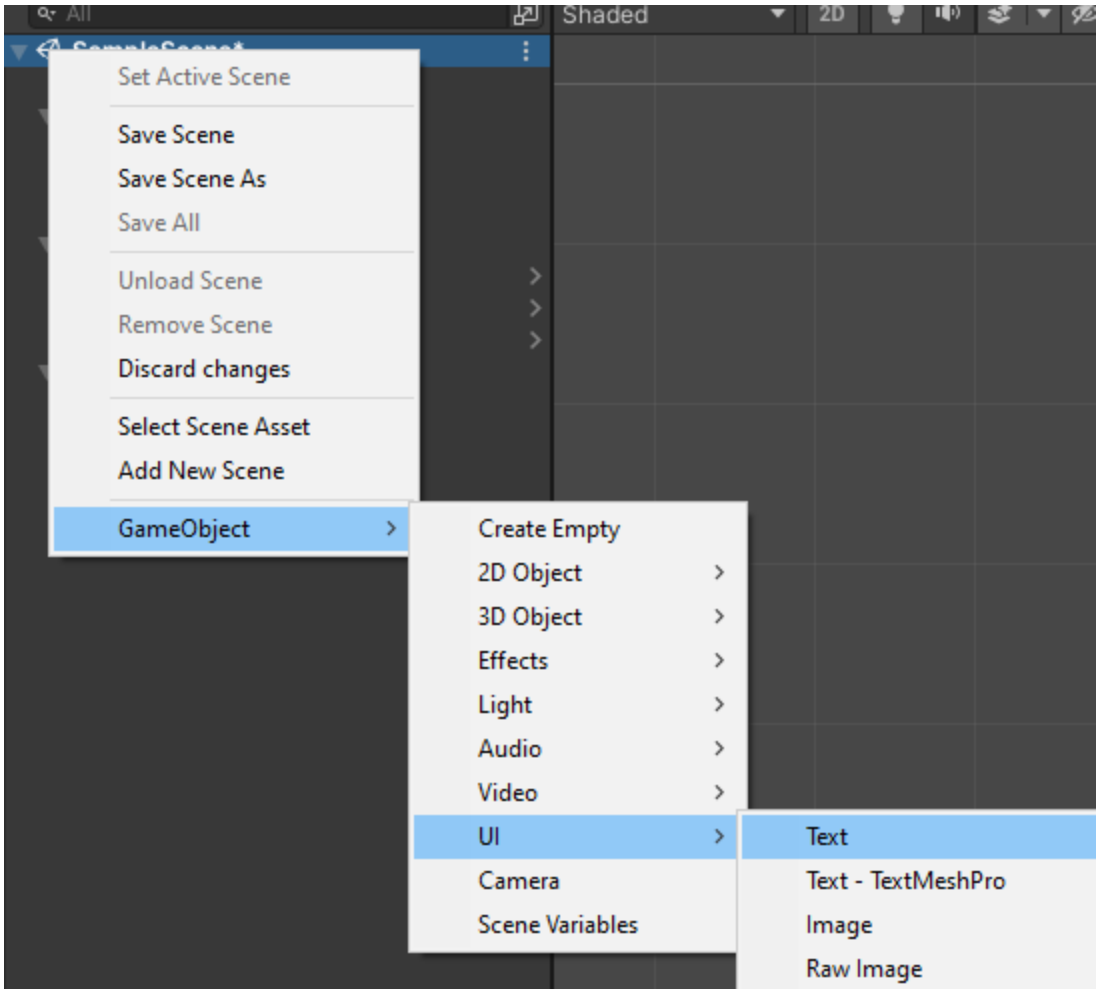
Hay un componente predefinido que sirve para evitar esto. **En el TERRENO**: add component → **plattform effector 2D**.



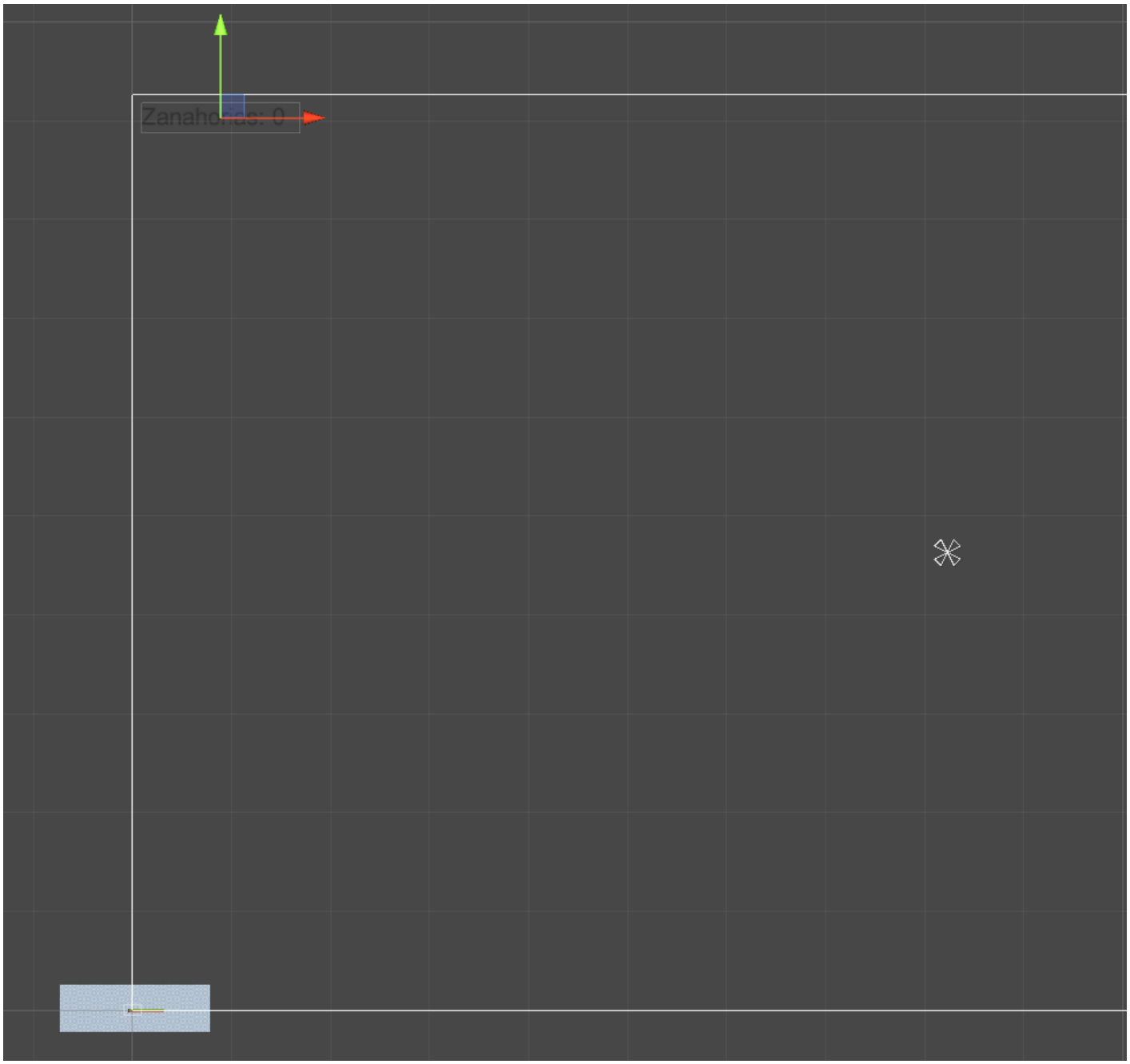
Con esta configuración nuestro jugador no se quedará en el aire al chocar con el terreno de frente (solo cuando caiga sobre él).

Elementos del HUD

Para añadir un texto:

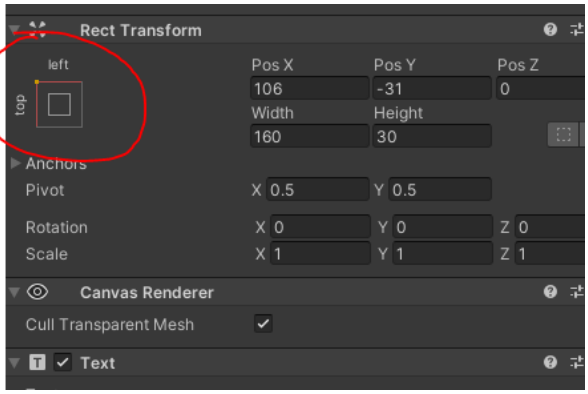


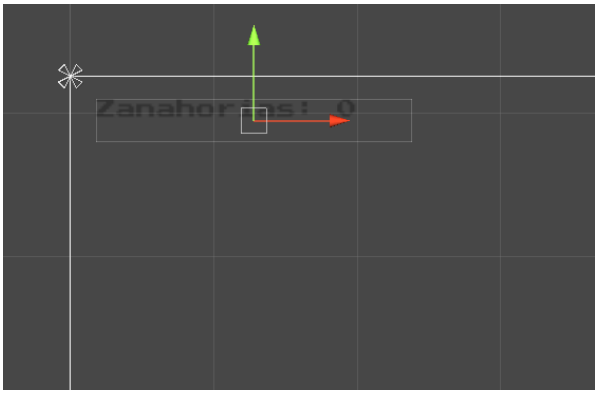
Si queremos ese texto arriba a la izquierda, tenemos que alejar el zoom y arrastrarlo ahí:



La parte de abajo azulada es nuestra escena. Esto puede resultar un poco extraño, pero si ejecutas esto aparecerá el texto “Zanahorias: 0”.

Si queremos asegurarnos de no tener problemas con distintas resoluciones de pantalla **anclamos el rectángulo a la parte superior izquierda:**





Adicionales

- Fuentes libres (Google fonts): https://fonts.google.com/specimen/Press+Start+2P?query=press+start+2p

Referencias

- Coding In Flow. Youtube. Build a 2D Platformer Game in Unity | Unity Beginner. Disponible en: https://www.youtube.com/playlist?list=PLrnPJCHvNZuCVTz6lvhR81nnaf1a-b67U
- Juan Gabriel Gomila. Curso de desarrollo de videojuegos con Unity 2021. Udemy. Disponible en: https://www.udemy.com/course/unity-2021/
-