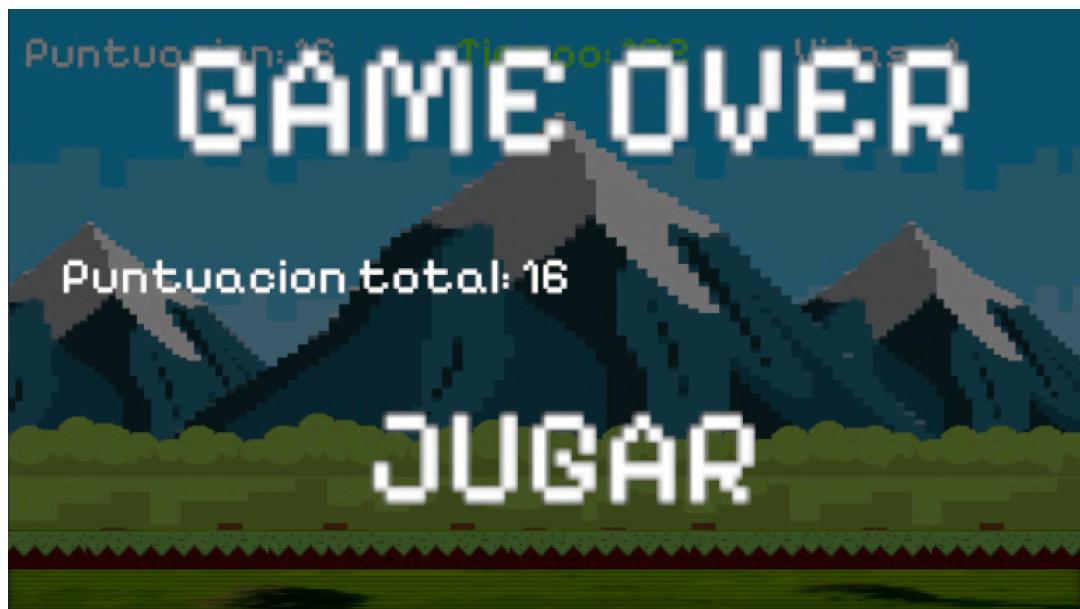


# Tarea 12 para Programación Multimedia y Dispositivos Móviles



Diego Manuel Carrasco Castañares

**Índice:**

Enunciado de la tarea.....	03
Importación de los espríte.....	06
Creación del personaje principal.....	11
Animaciones.....	15
Fondos.....	39
Game Over.....	44
Pantalla inicio.....	49
Sonido.....	50

**Enlace de descarga:**

<https://we.tl/t-9F6WXkayB8>

Realiza un juego en Unity, basándote en el realizado en el pdf de la unidad 12 (juego mapache), que tenga las siguientes características:

- Importación de Sprites de los elementos del juego: los Sprites podrán ser creación propia o plantillas de internet. (0,5 puntos)
- Creación del personaje principal: Hay que crear el personaje con algún software como **Piskel** por partes y montar el personaje como se hace con el mapache. (1 punto)
- Hacer dos animaciones al personaje principal. (1 punto)
- Programación del juego. (**IMPORTANTE: es imprescindible documentar bien internamente el código para saber si se ha entendido**) (2 puntos)
- Generar elementos aleatorios. (1 punto)
- Puntuación del juego. (0,5 puntos)
- Controlar fin del juego y poner pantalla de fin. (0,5 puntos)
- Fondos del juego. (0,5 puntos)
- Pantalla de inicio. (0,5 puntos)
- Poner sonido al juego. (0,5 puntos)
- Documentación del juego. (2 puntos)

#### **CRITERIOS DE CALIFICACIÓN:**

Para que el ejercicio se considere correcto, se debe ajustar a las especificaciones del enunciado. En caso contrario se considerará incorrecto.

**El proyecto se debe entregar sin errores de compilación, si tiene errores su nota es de 0 puntos.**

**Además, si se detecta que se ha copiado íntegramente el juego y no se ha desarrollado paso a paso su calificación será de 0 puntos.**

Recursos necesarios para realizar la Tarea.

- Ordenador con Unity instalado y suficientes recursos para ejecutar el juego.

- Contenidos de la Unidad 12: Desarrollo de videojuegos con Unity.

Consejos y recomendaciones.

- Lee atentamente el enunciado y asegúrate de haber entendido lo que has de hacer.
- Muy importante hacer el juego del mapache en el que se basa los contenidos de esta unidad para poder afrontar hacer un juego propio.
- No dudes en comentarle a tu tutor o tutora cualquier duda que te pueda surgir.
- Envíasela a tu tutor o tutora a través del sistema establecido en la plataforma.
- Las capturas y el contenido de los ficheros deben aparecer en orden.
- Debe llevarse a cabo sobre una versión actual de Unity.

Indicaciones de entrega.

Una vez realizado el juego elaborarás un documento PDF documentando el funcionamiento del juego y todos los pasos que se han seguido de cada uno de los apartados que se piden. Las capturas deben mostrar el fondo de pantalla y la fecha y hora de la barra de estado de tu sistema operativo para garantizar la originalidad del material. Muy importante que personalices tu nombre en el juego.

También se deberá entregar un enlace al drive donde estará alojado todo el proyecto del juego, ya que debido a su tamaño no se puede subir a la plataforma.

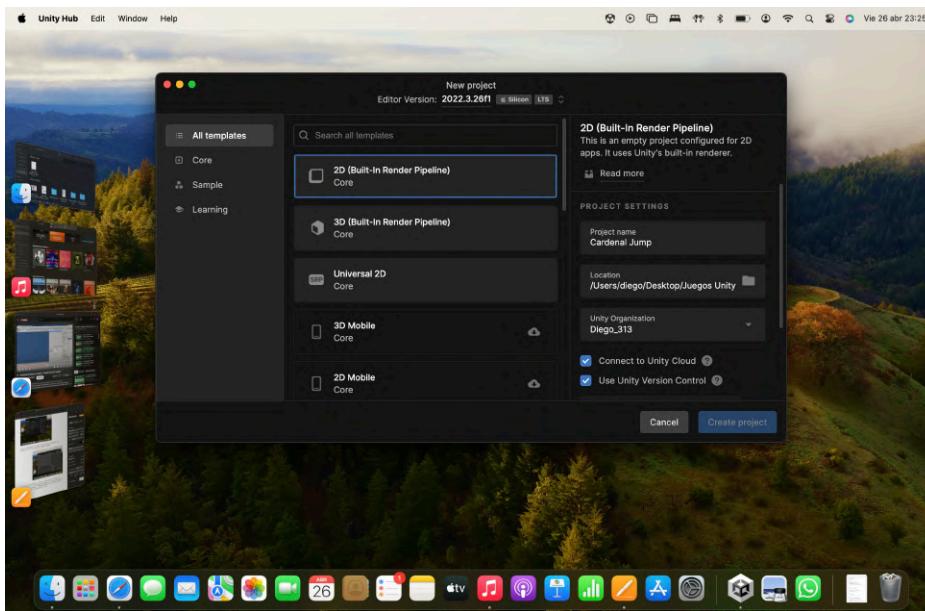
Entrega los archivos a través de la plataforma. Se nombrarán siguiendo las siguientes pautas:

apellido1\_apellido2\_nombre\_PMDM12\_Tarea

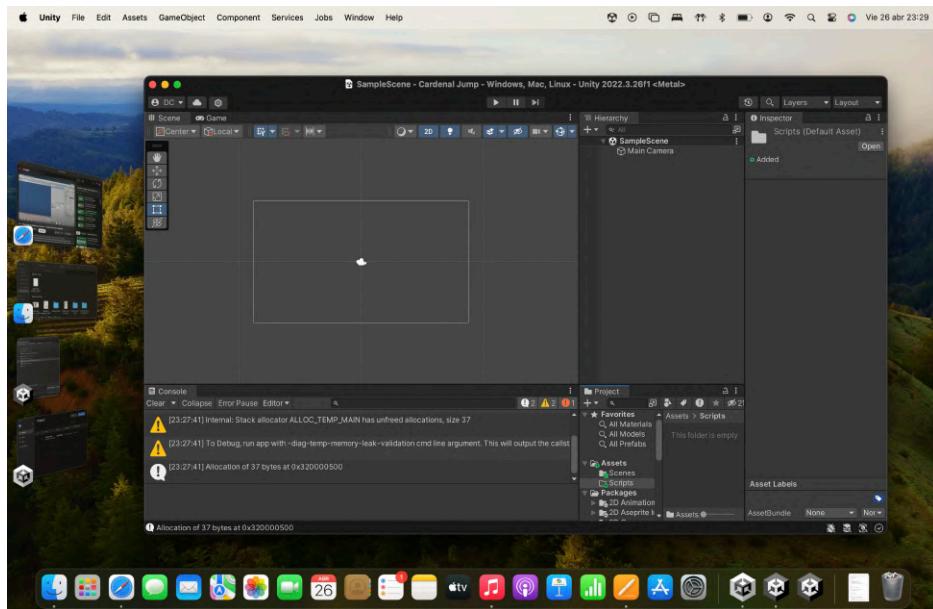
Asegúrate que el nombre no contenga la letra ñ, tildes ni caracteres especiales extraños. Así por ejemplo la alumna Begoña Sánchez Mañas para la tarea de la unidad PMDM, debería nombrar esta tarea como...

sanchez\_manas\_begona\_PMDM12\_Tarea

En primer lugar vamos a empezar creando el proyecto en Unity, al cual llamaremos "Cardenal Jump".

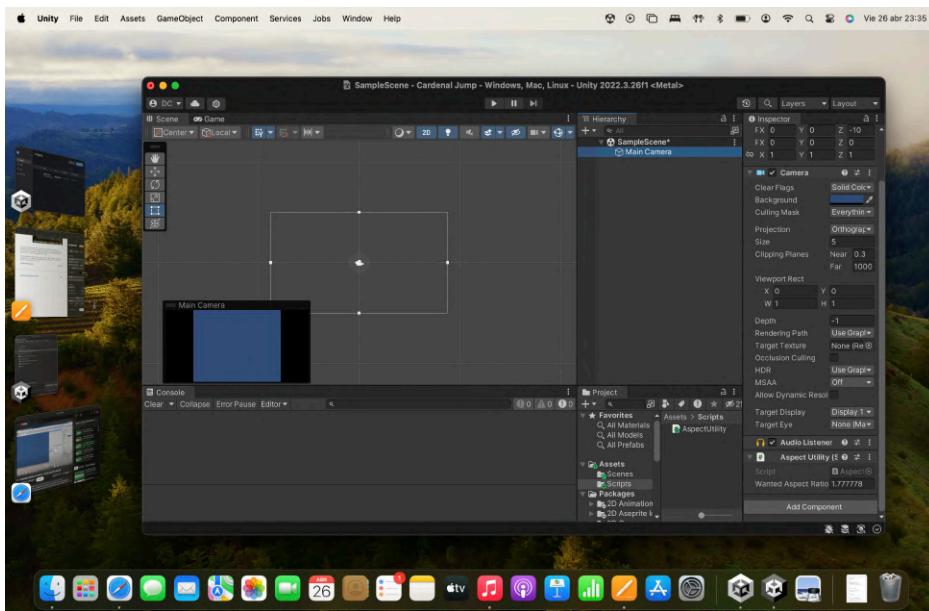


Tras ello crearemos la carpeta Scripts dentro de assets, para guardar en ella los script que vayamos creando.



A continuación incorporaremos el primer script dentro, que será el `AspectUtility.cs` y se encargará de mantener la relación de aspecto de 16:9 para el proyecto, independientemente de la dimensión de pantalla en la que se reproduzca el juego.

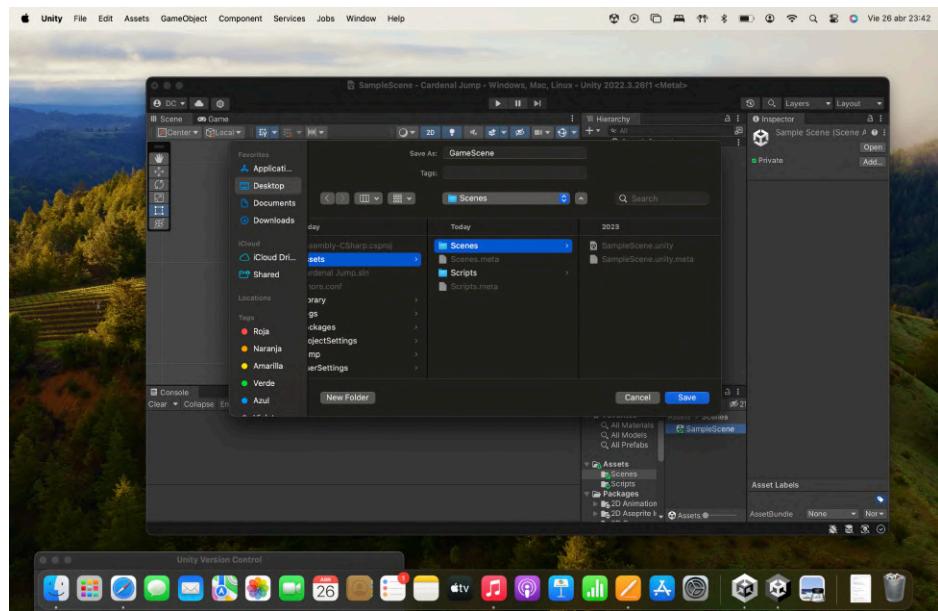
Posteriormente arrastraremos este script a Main Camera, para así aplicárselo a la cámara y pondremos que la relación se 1,777778, que es el equivalente a 16:9.



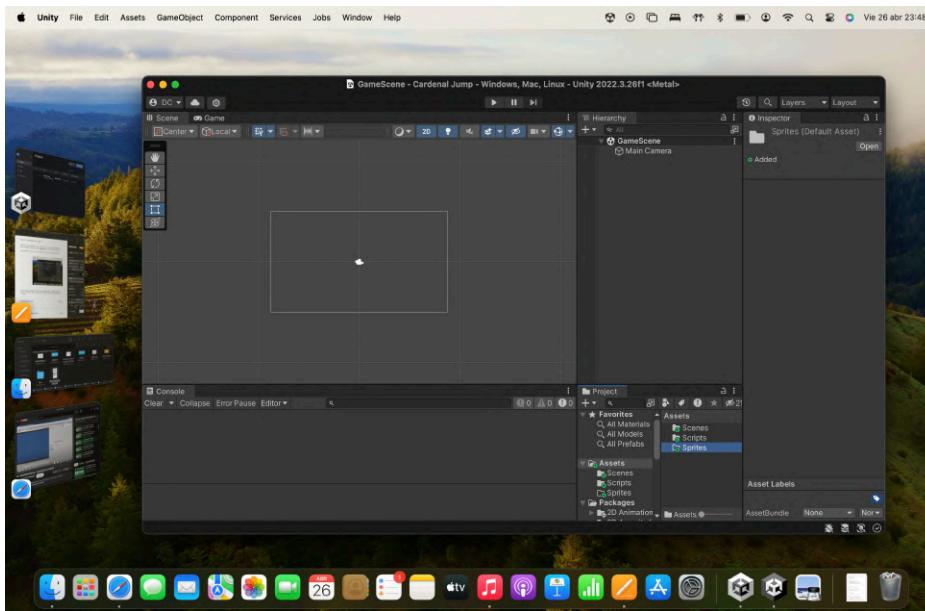
Tras ello guardaremos los cambios.

Para ello clicaremos en file ==> save as, en la ventana que nos sale pondremos, para mi caso, GameScene, y clicaremos en save.

Para guardar el proyecto clicaremos en file ==> save proyecto.



A continuación vamos a crear una carpeta nueva llamada Sprites dentro de assets donde guardaremos los Sprite que necesitemos en el juego.



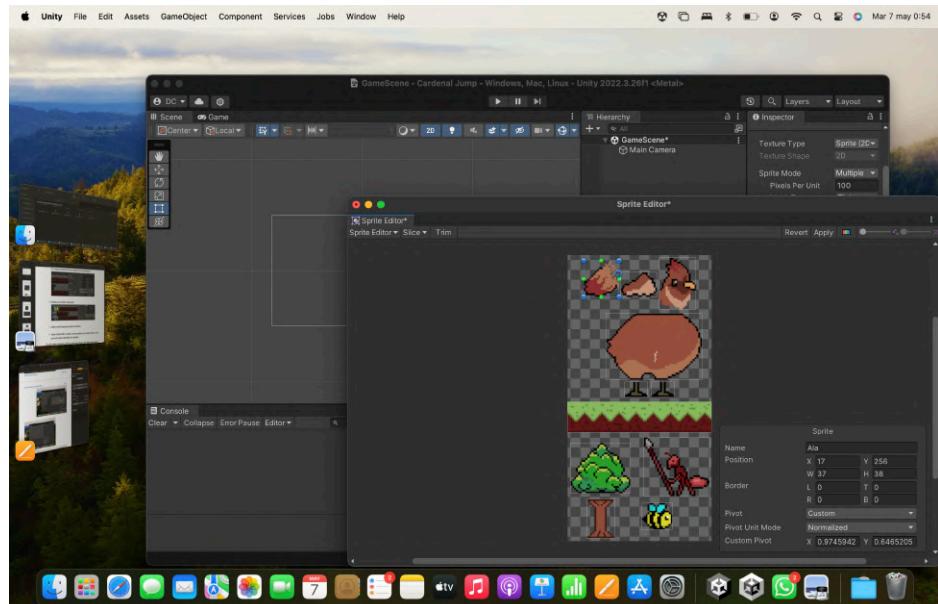
Tras ello importaremos los Sprite.

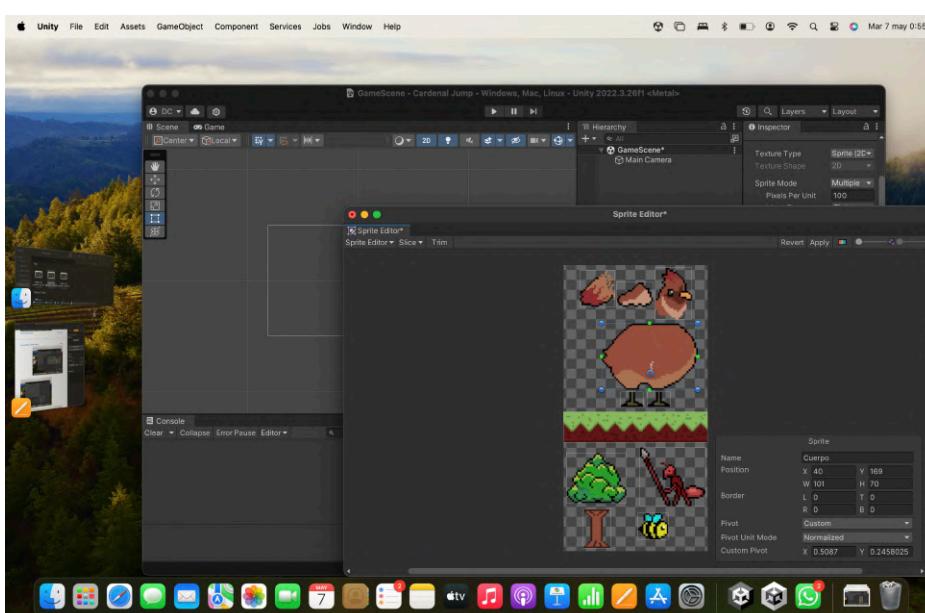
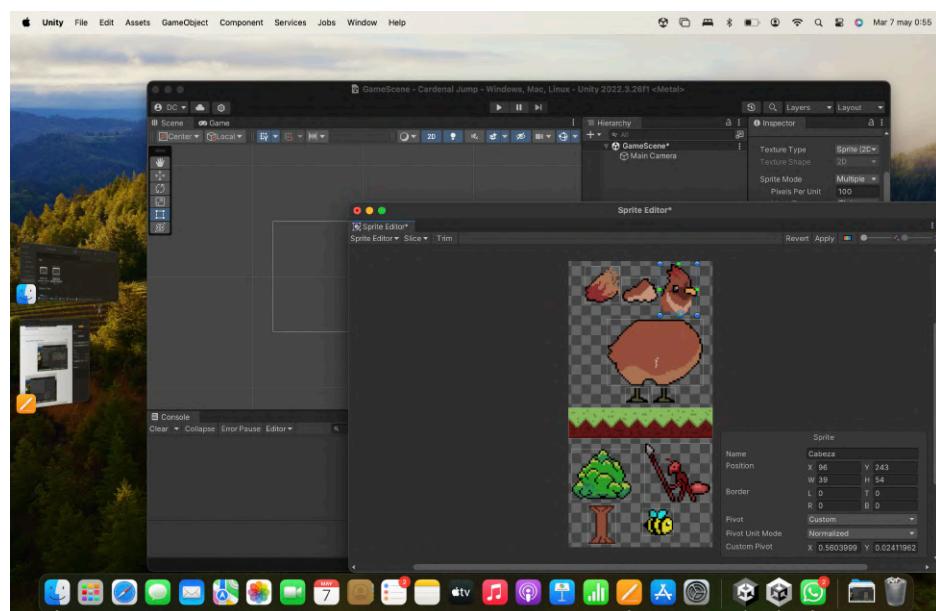
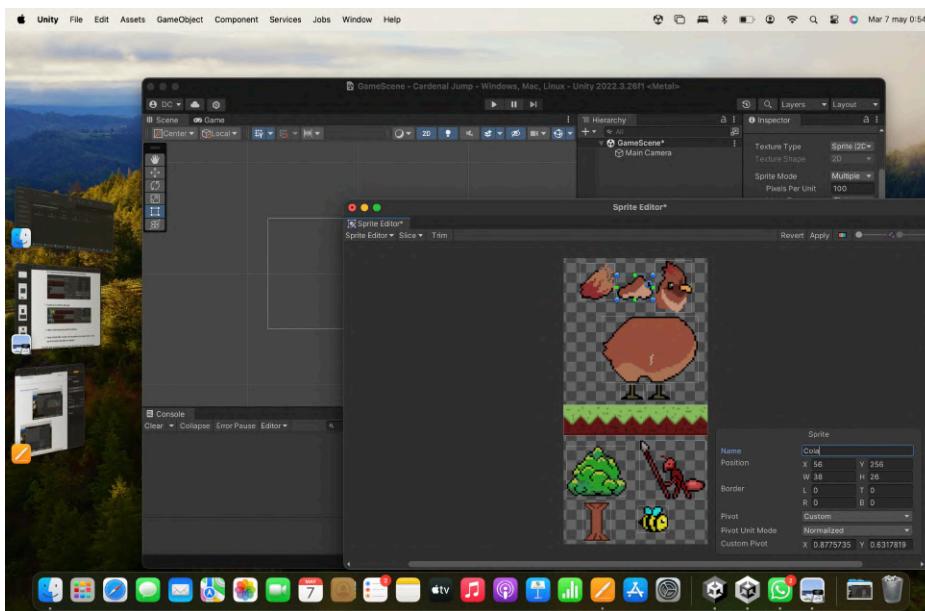
Para ello los arrastraremos al interior de la carpeta Sprites creada anteriormente.

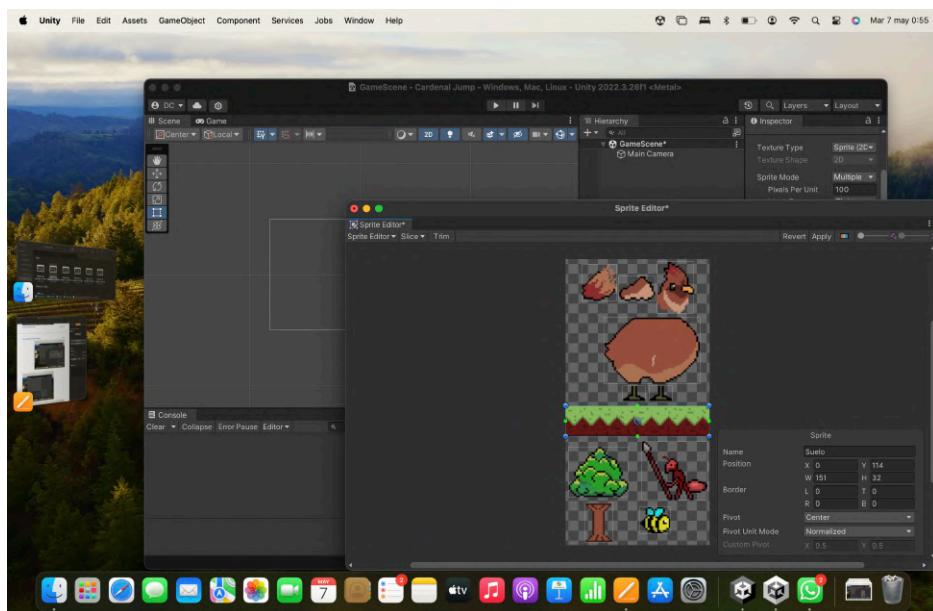
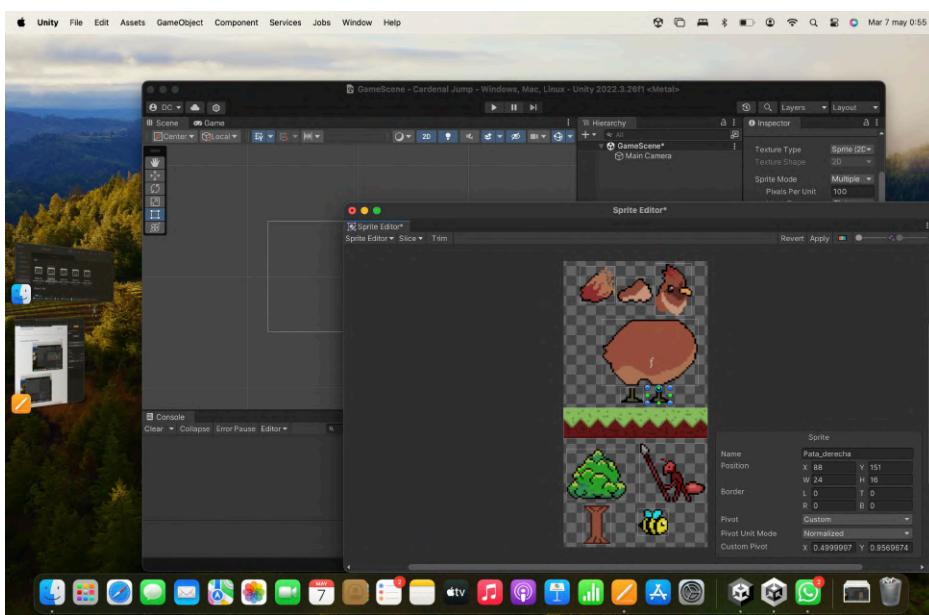
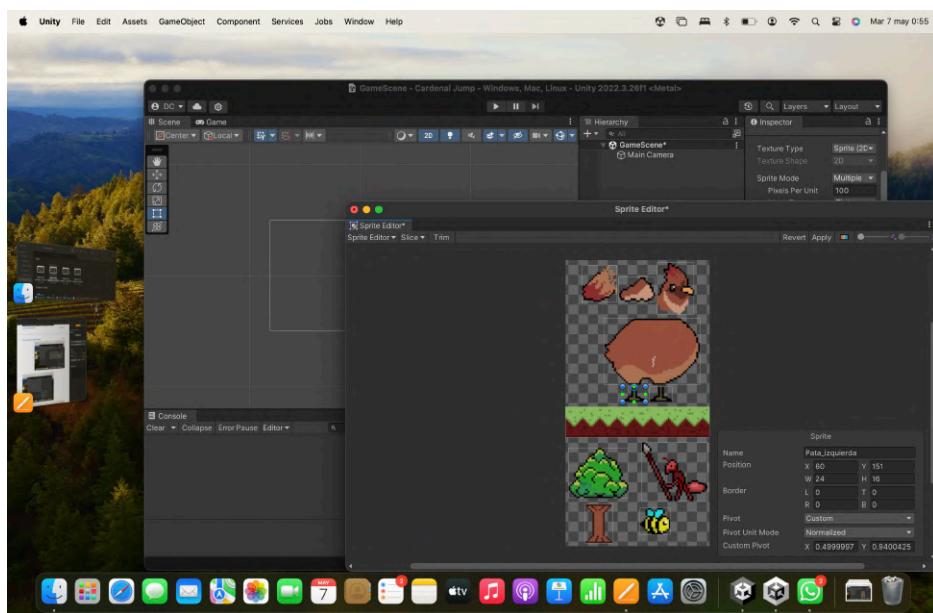
Posteriormente lo seleccionaremos y cambiaremos, en Sprite mode a múltiple y en max size a 512 para nuestro caso y aplicaremos los cambios.

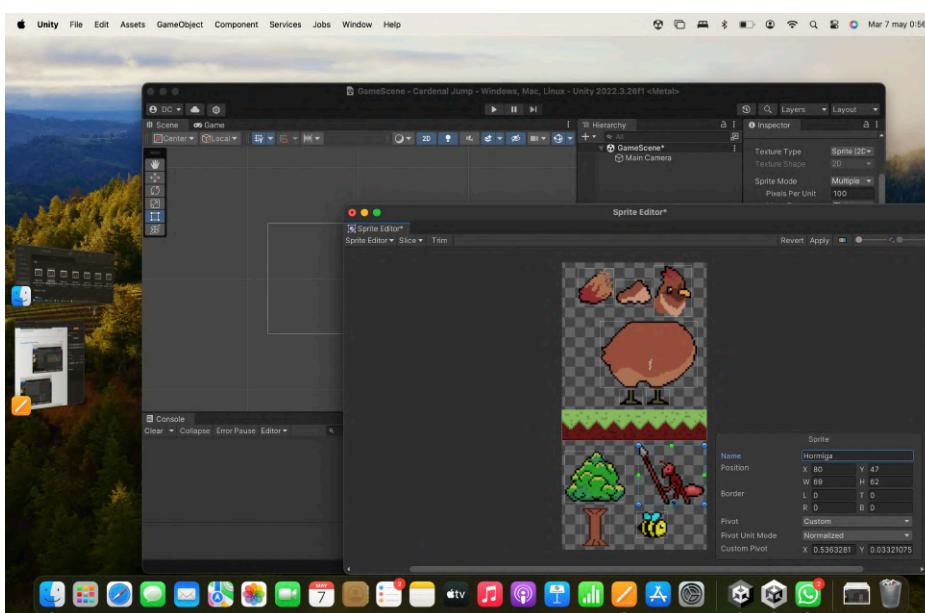
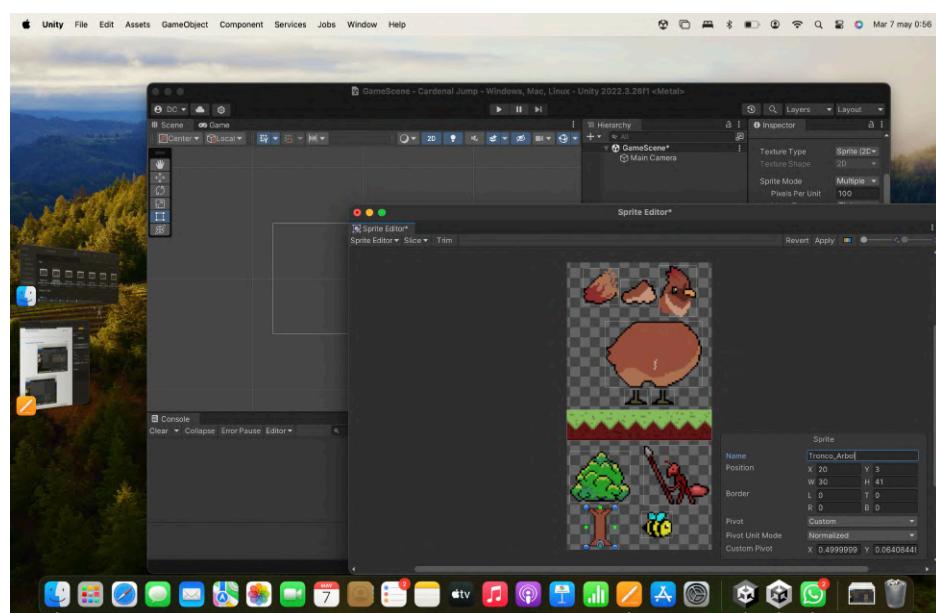
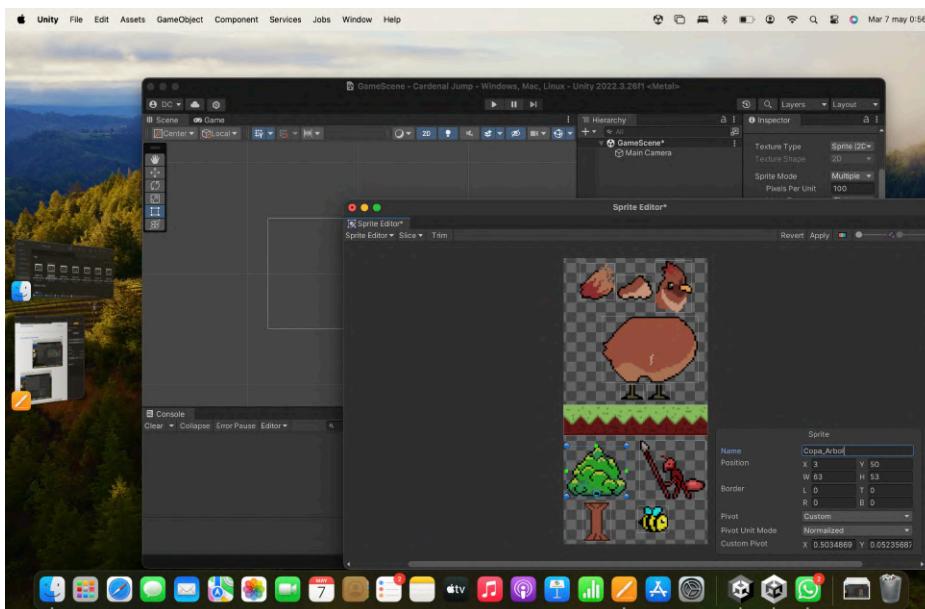
Tras ello accederemos a sprite editor y vamos a seleccionar el modo de recorte automático, vamos clicando sobre cada uno de los recortes y modificamos, en caso necesario, los bordes de los mismos y el punto de rotación y le damos un nombre a cada parte.

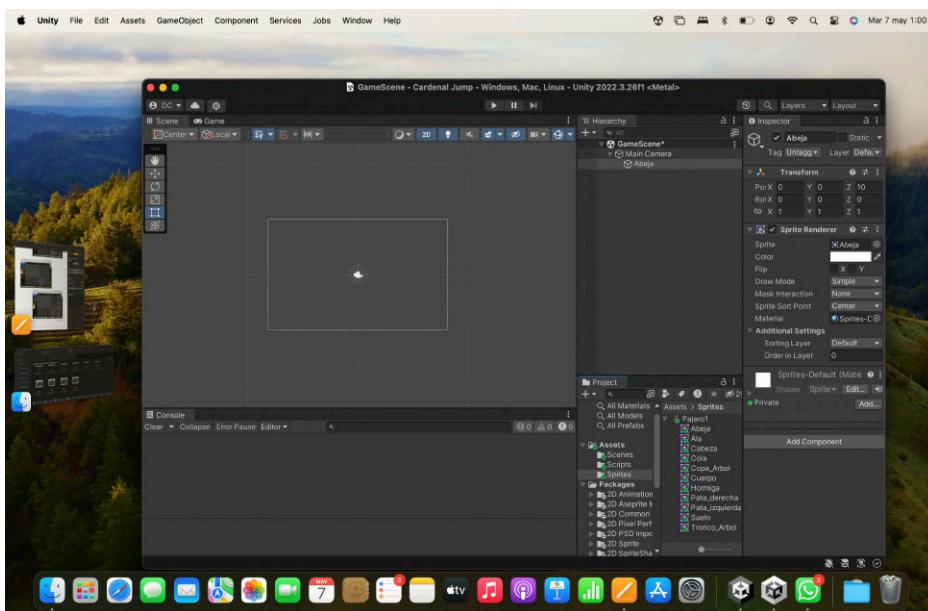
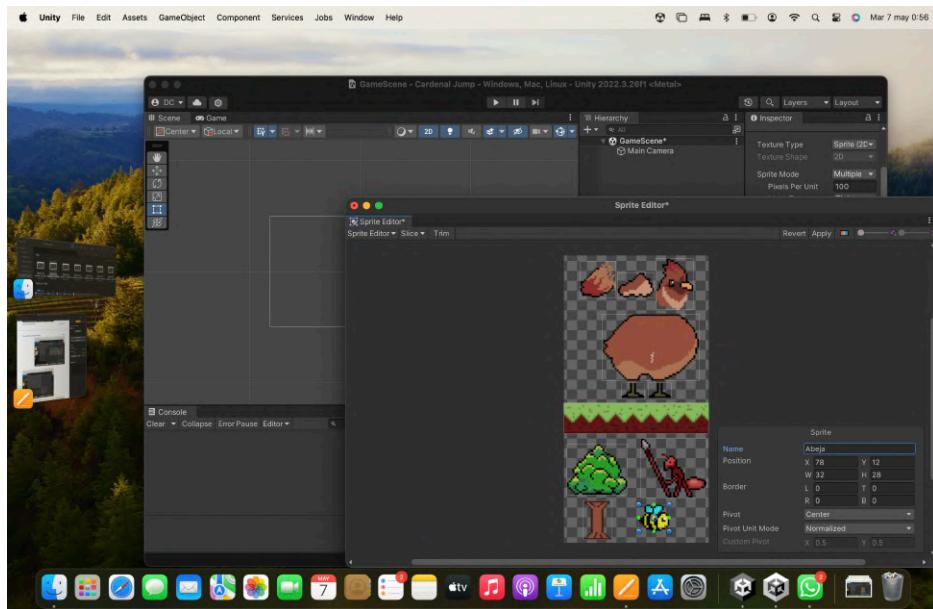
Podemos ver lo mencionado an lineas anteriores en las siguientes imágenes.











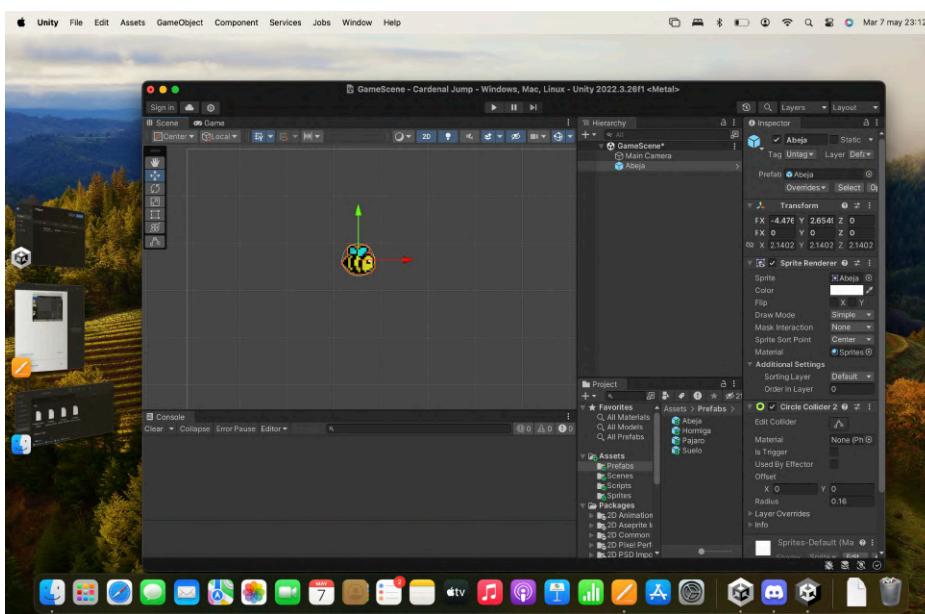
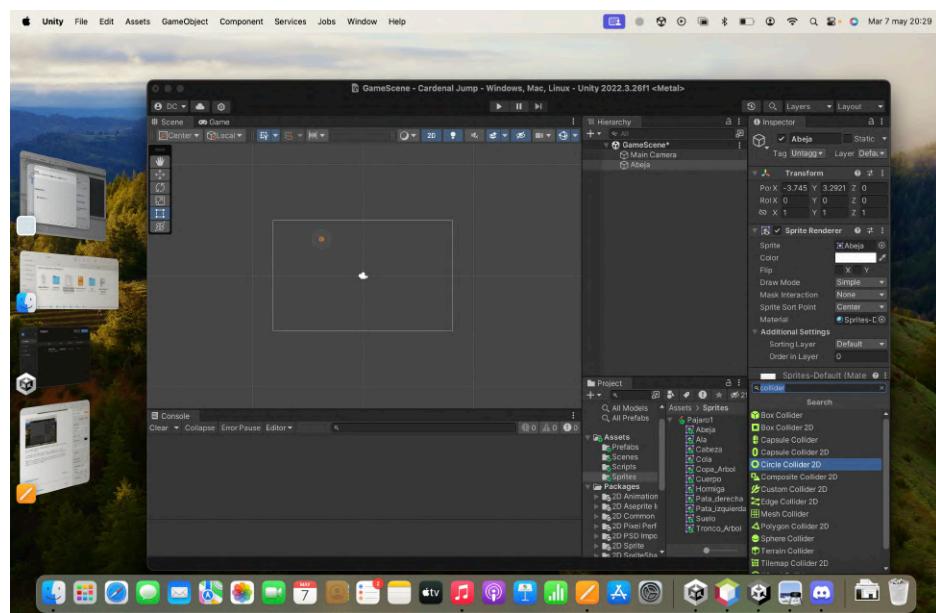
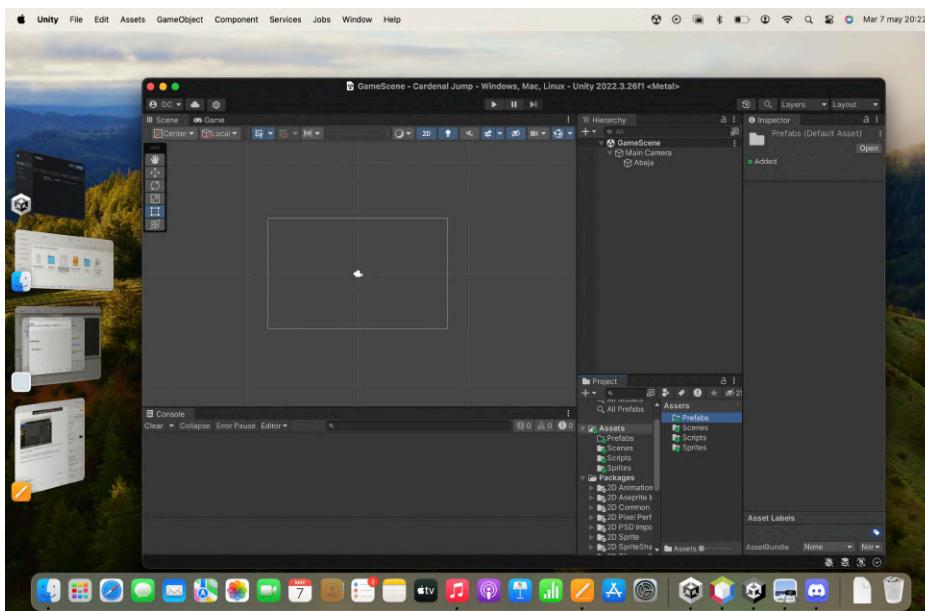
Lo siguiente que haremos será crear los prefabs por lo que necesitamos crear una carpeta, llamada **Prefabs**, para guardarlos.

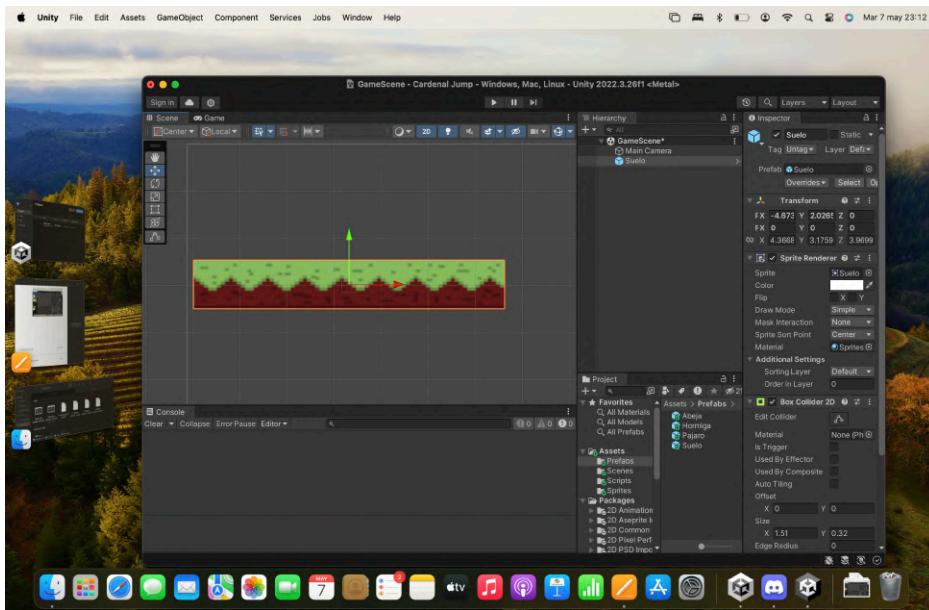
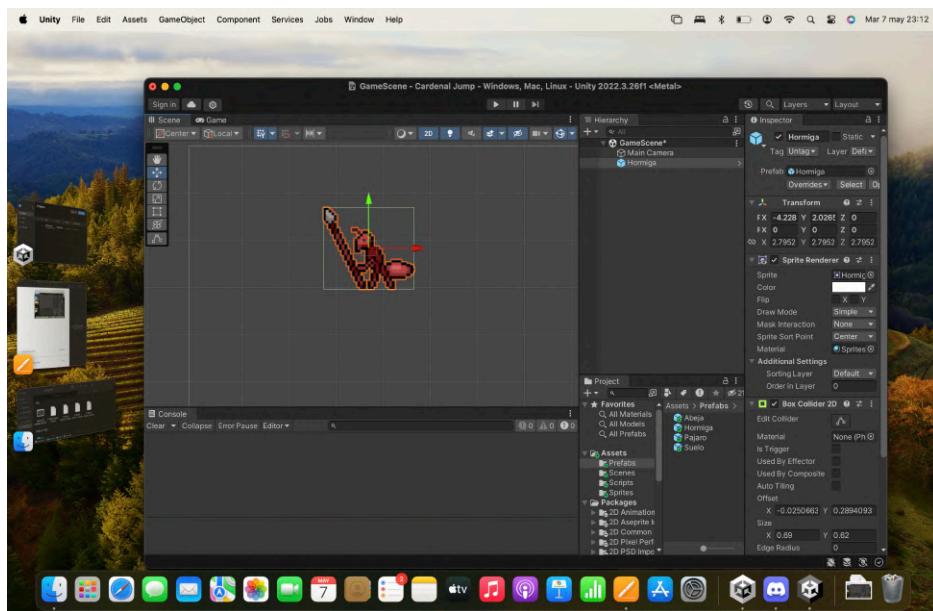
Empezaremos creando el prefab de la abeja, para ello clicaremos sobre la carpeta que hemos creado para guardarlos con el botón derecho ==> **créate ==> prefab**.

Una vez creado el prefab vamos a ir a los sprite, seleccionaremos el de la abeja y la arrastraremos hasta la escena.

Tras ello ajustaremos el tamaño de la abeja, le añadiremos un collider 2D, ajustaremos el collider al objeto y pondremos sus coordenadas a 0 (las capturas las hice antes de poner las coordenadas a 0).

Esto lo repetiremos con la hormiga, el suelo y el árbol (tanto tronco como copa).





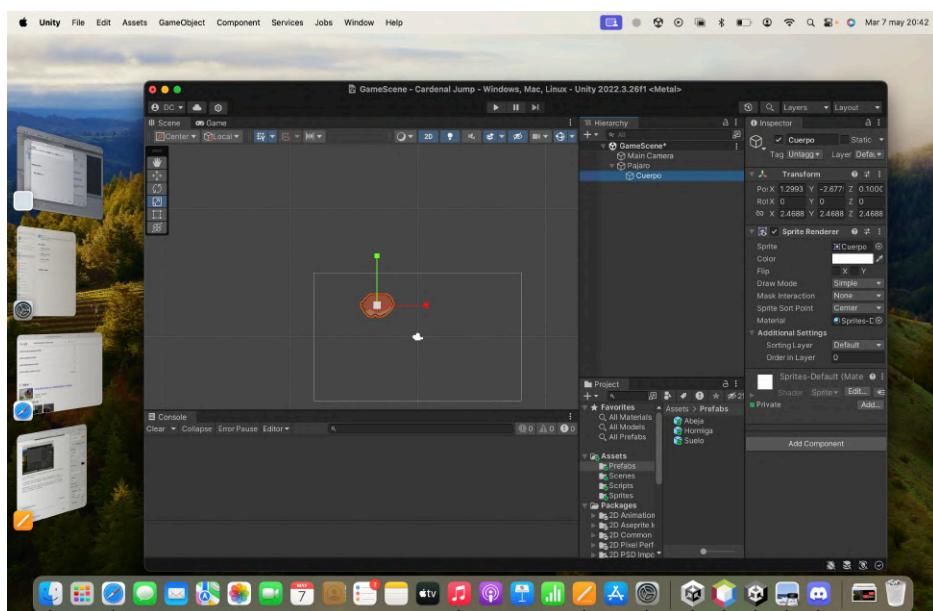
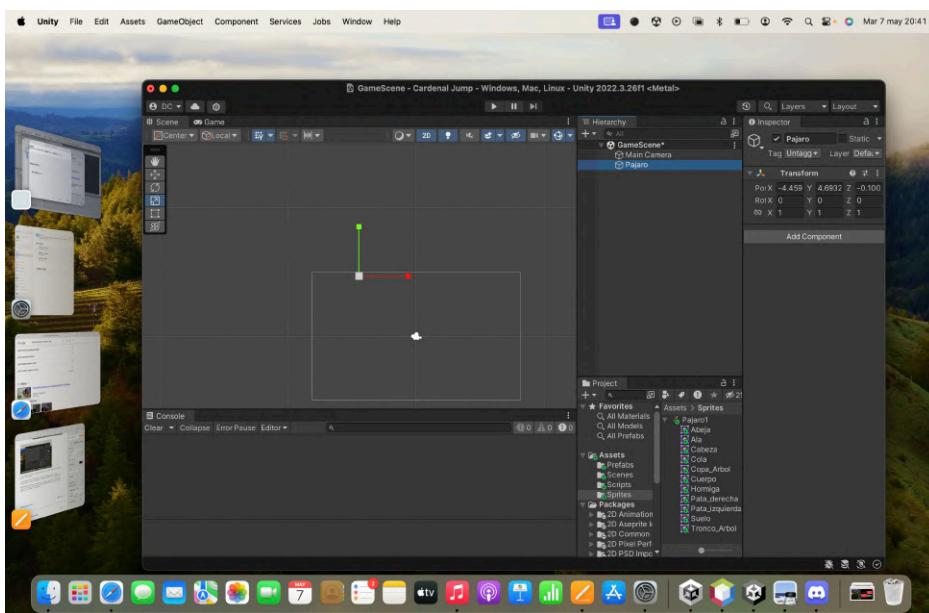
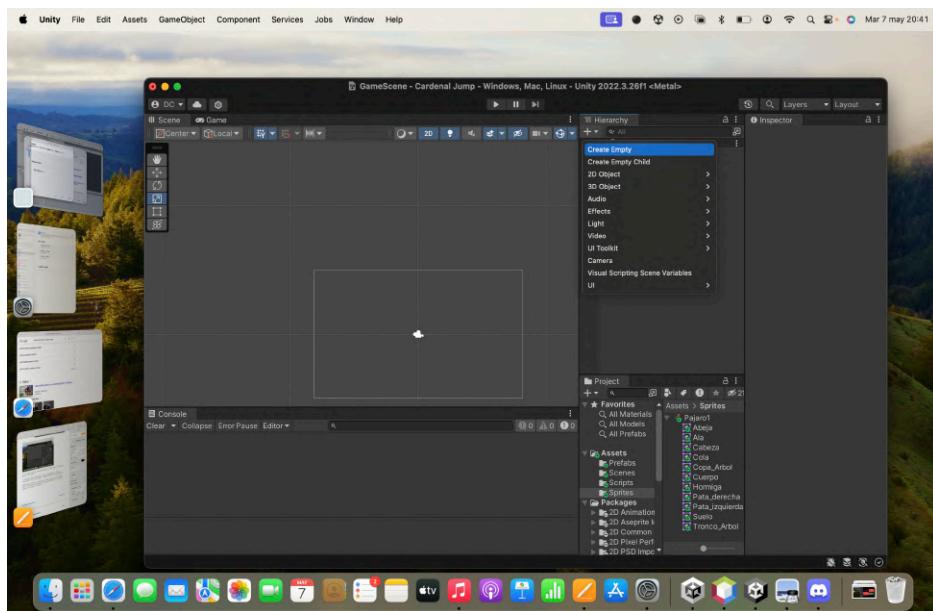
Lo siguiente que haremos será crear el personaje.

Para ello clicaremos en el símbolo + ==> game object ==> create empty y le vamos a llamar Pajaro.

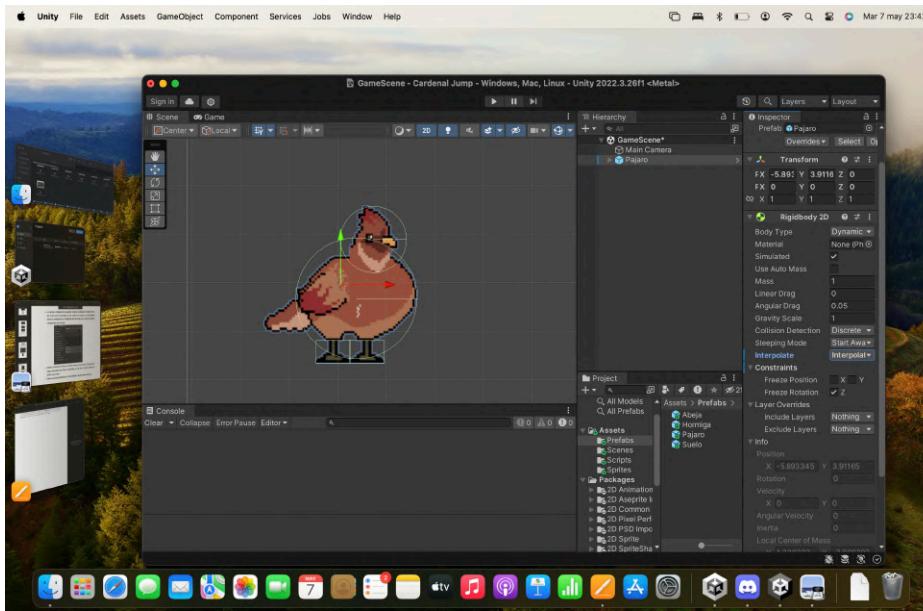
Tras ello vamos a ir añadiendo cada parte del persona a la escena e iremos modificando su tamaño para ajustarlo a en proporciones.

Ajustaremos cada parte en el eje z para que no se superpongan unas con otras.

Una vez que hemos añadido todas las partes añadiremos los collider necesarios, que en nuestro caso serán uno en cada pata, uno en el tronco y uno en la cabeza.

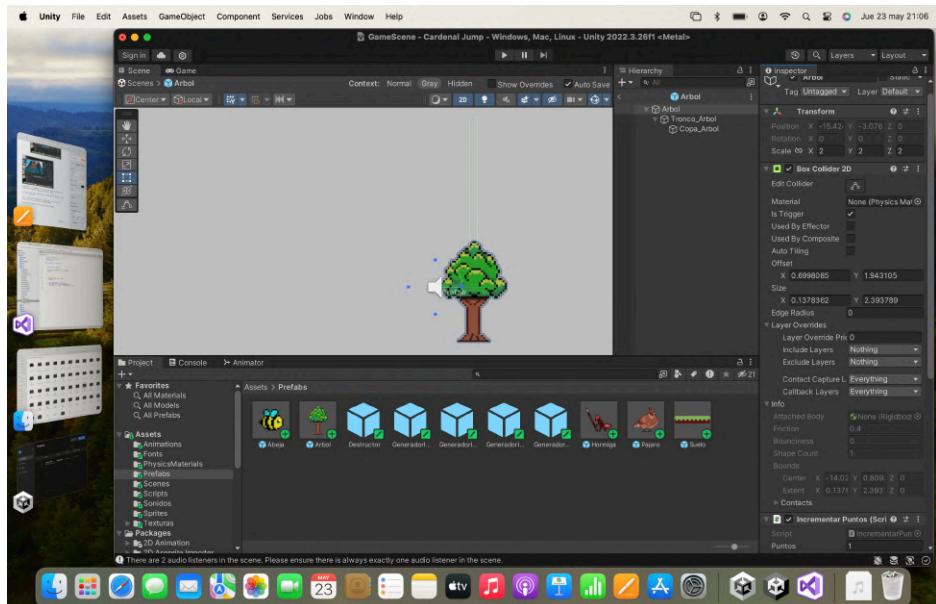


A continuación le vamos a añadir un RigidBody para que le afecte la física (gravedad, velocidad, etc), desactivar la rotación del personaje y activar la interpolación del mismo.



Tras ellos crearemos un nuevo objeto de la misma forma que el personaje y en él añadiremos la copa del árbol y el tronco, le añadiremos los collider a ambos y lo incluiremos en su prefab correspondiente.

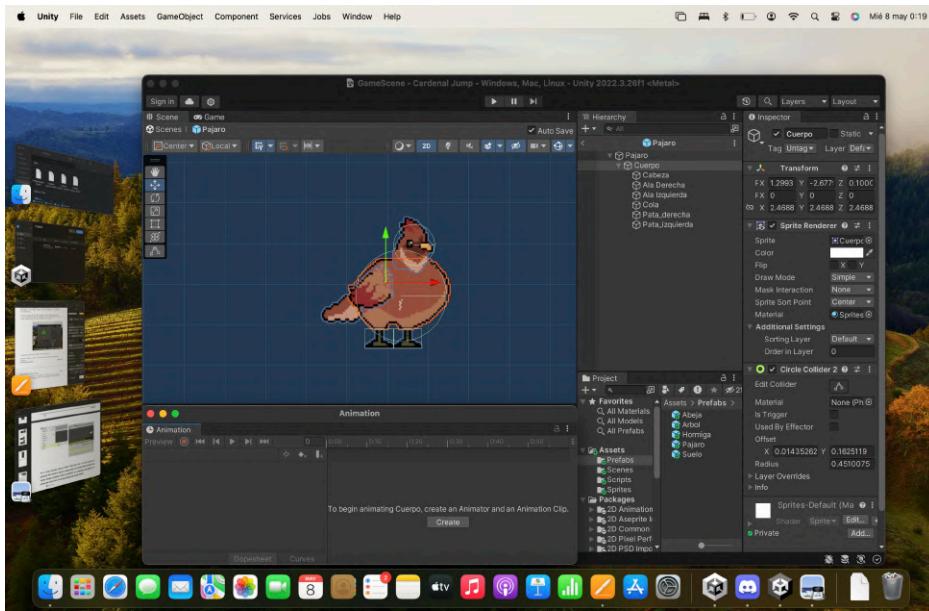
Es árbol lo hice de esta forma para que al chocar con el tronco muriera el personaje y al pasar por encima sumara puntos, cosa que funcionaba hasta que metí los generadores, a partir de ahí no conseguí solucionar el problema y modifique el prefab.



El siguiente paso será crear las animaciones.

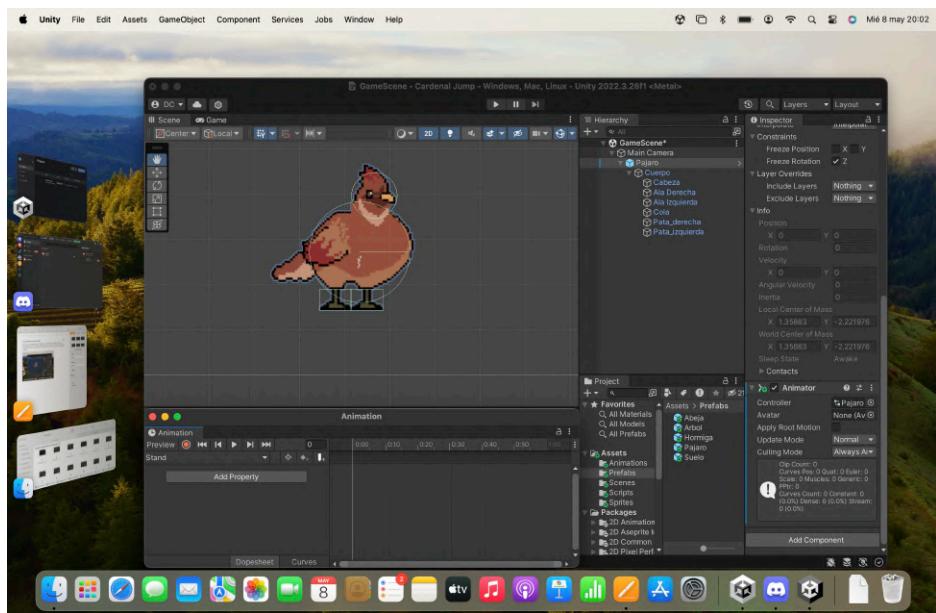
Para ello vamos a añadir la ventana de animaciones a nuestro proyecto.

Tras ello clicaremos en el objeto, el pájaro en nuestro caso, y posteriormente en create, dentro de la ventana de animaciones y en la ventana que se nos abre le diremos que cree una nueva carpeta, llamada



Tras darle a Create y decirle que nos lo guarde en la carpeta Animations veremos como se le ha añadido a nuestro personaje el componente Animador.

Esta animación será Stand, para cuando el personaje esta parado.

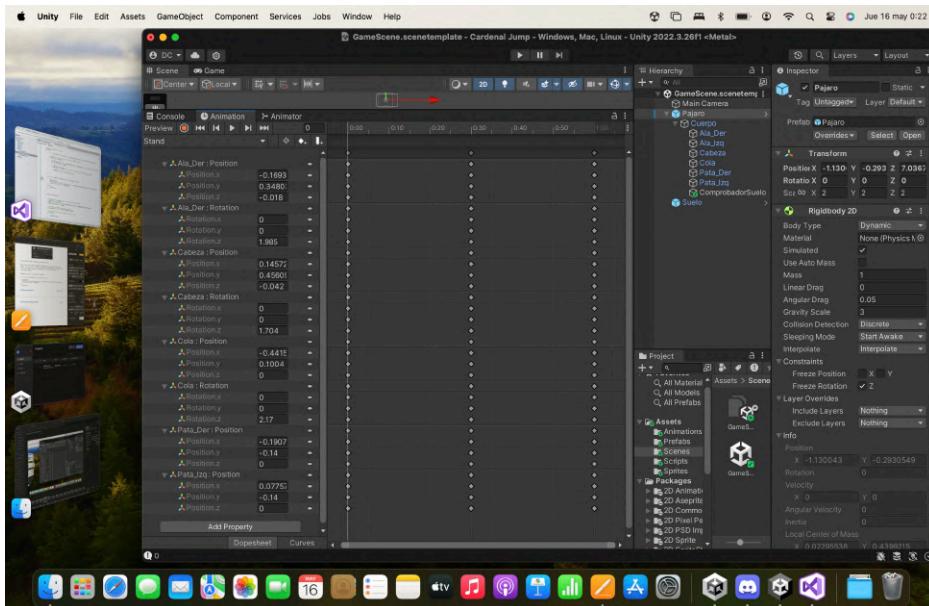


Ahora iremos añadiendo las partes que vamos a modificar en la animación.

En este caso añadiremos el cuerpo, el ala derecha, la cabeza, la cola y las patas para que simule la respiración del personaje.

En el tiempo 30 subiremos un poquito el cuerpo, mantendremos en su lugar las patas, y rotaremos un poquito la cabeza, la cola y el ala.

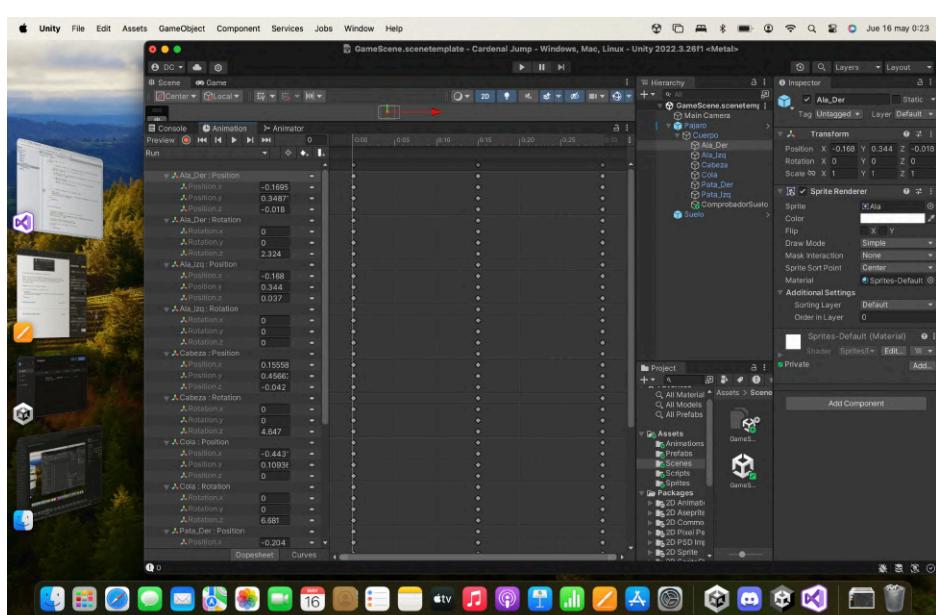
En cambio, en el tiempo 0 y 60 lo mantendremos com estaba por defecto.

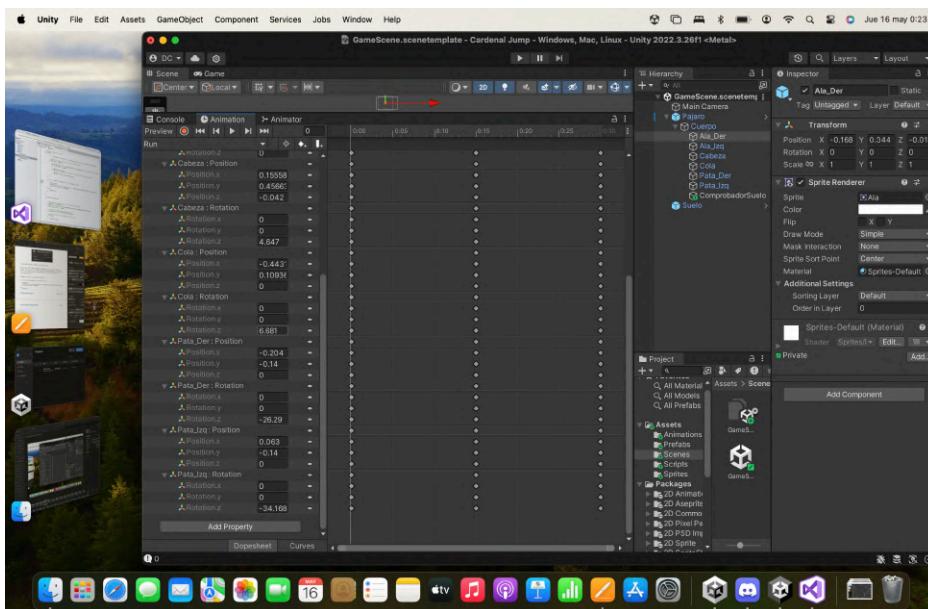


La siguiente animación que haremos la la que simule que nuestro personaje está corriendo, a la cual vamos a llamar Run.

Para ello vamos a recortar el tiempo a 30.

En el tiempo 0 y 30 el personaje estará con la cabeza hacia delante, el ala derecha y la cola un poquito bajadas y las patas hacia atrás, en cambio, en el tiempo 15 estará en posición normal.

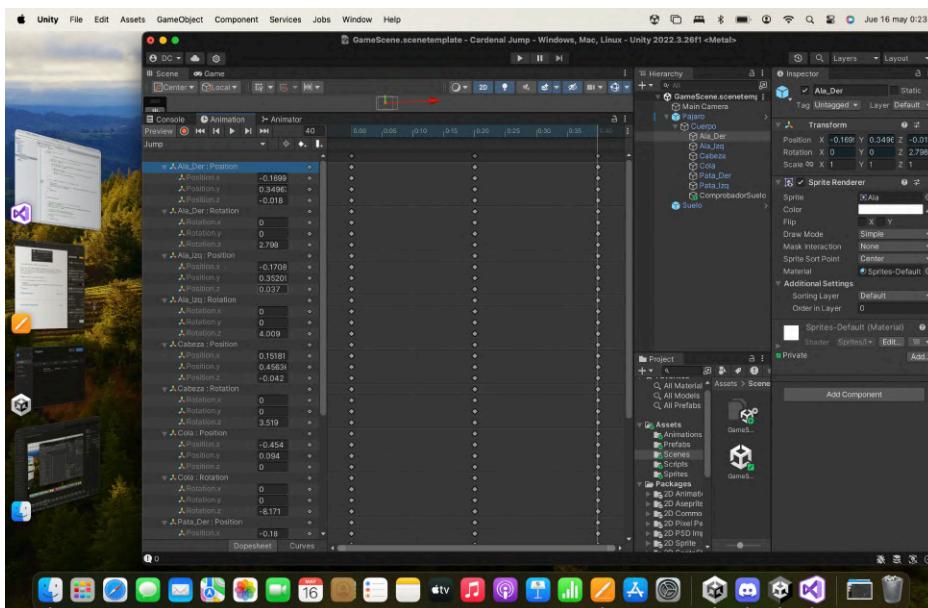


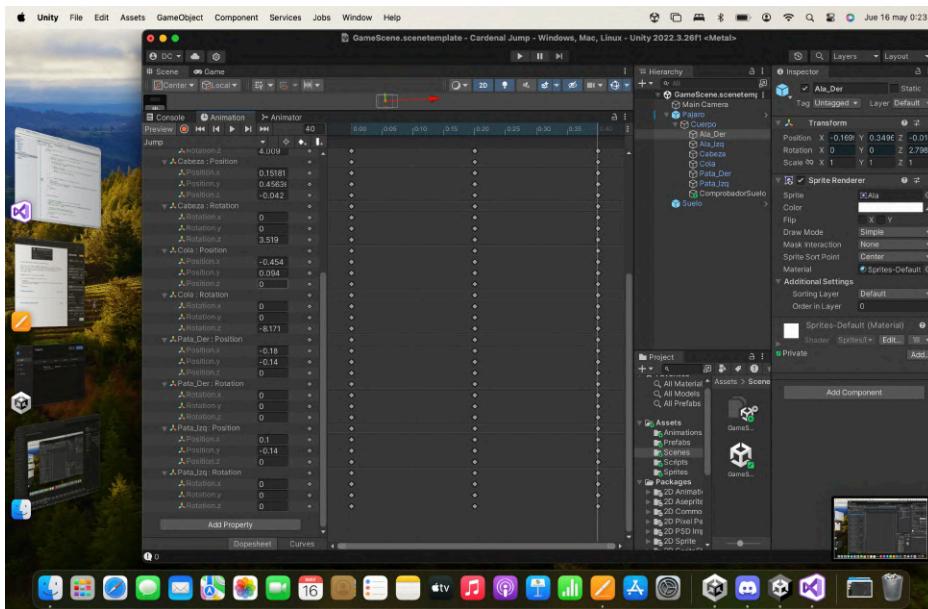


La siguiente animación que crearemos será la que va a simular que nuestro personaje está saltando.

Para ello en los tiempo 0 y 60 el personaje tendrá la cabeza hacia adelante, ambas alas y la cola levantadas y las patas abiertas.

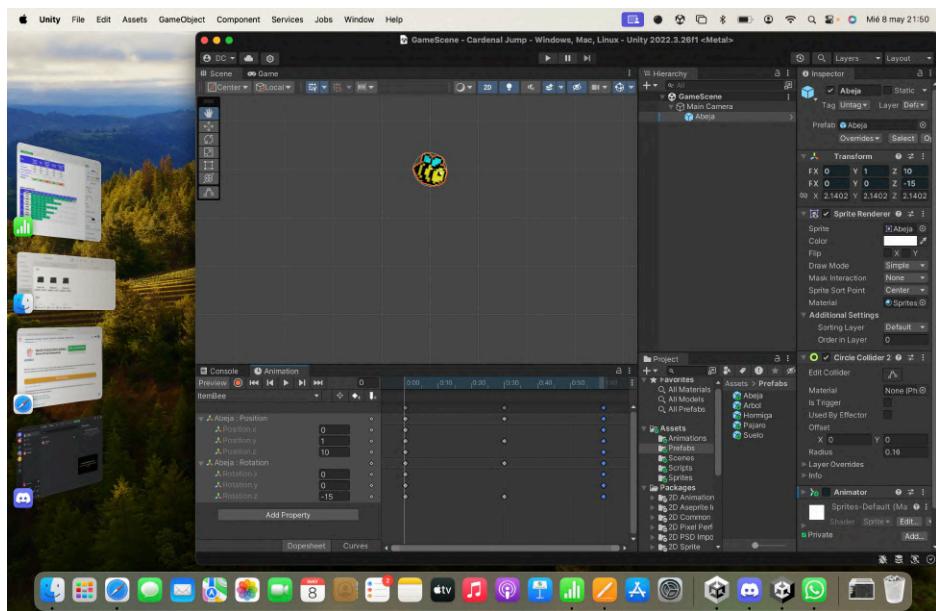
En el tiempo 30 el personaje mantendrá la posición de reposo general.





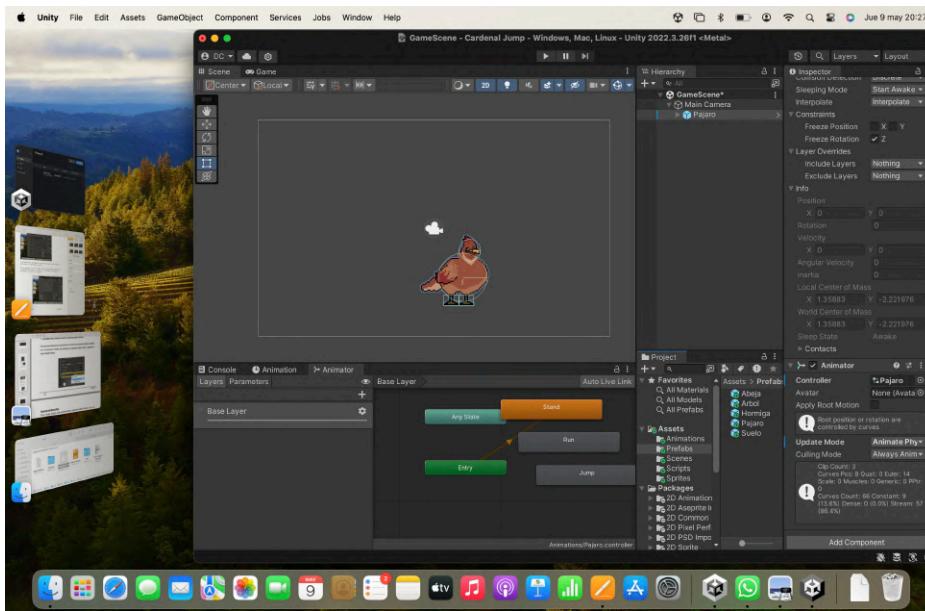
La animación que crearemos a continuación será la del item, la abeja para el juego que nos ocupa.

En los tiempos 0 y 60 la abeja estará en una posición mas alta e inclinada hacia abajo, en cambio, en el tiempo 30 la abeja mantendrá su posición de reposo.



Tras ello nos dirigimos a window ==> Animation ==> Animador.

Seleccionamos el personaje, y en el componente le indicamos que el controller sea el personaje (Pajaro en mi caso) y en update mode le indicamos Animate Physics para que las escenas se sincronicen con la física del juego.



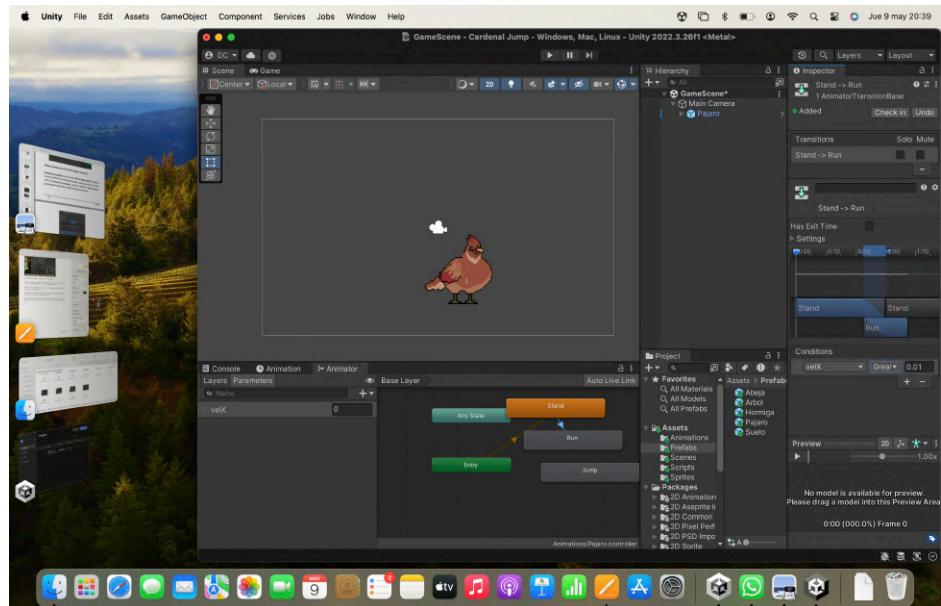
A continuación vamos a crear las transiciones.

Para ello seleccionamos la transición Stand ==> botín derecho del ratón ==> Make transition ==> y clicamos sobre run.

Después, en Parameters clicamos en + y creamos una variable de tipo float llamada velX.

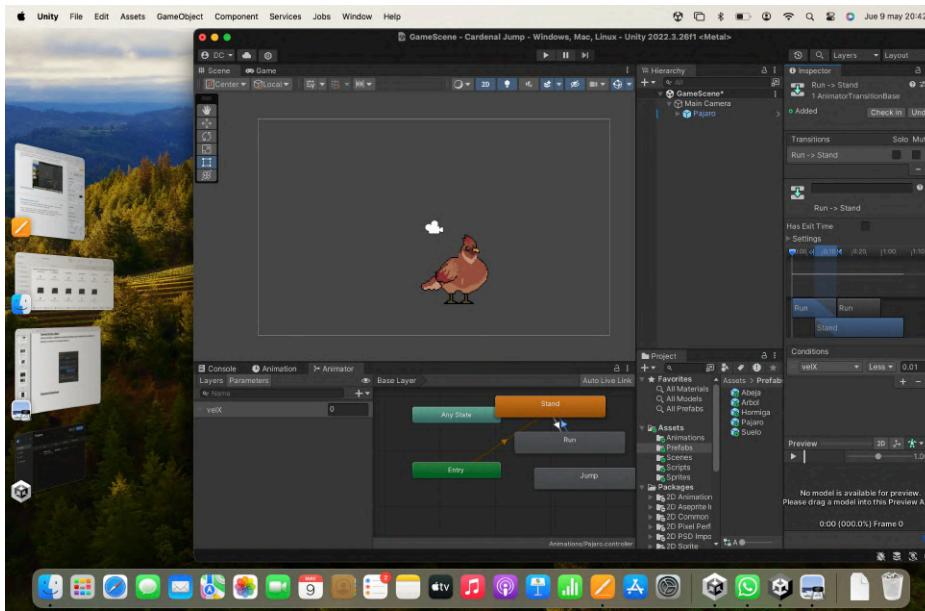
Seguidamente clicaremos sobre la transición que hemos creado y quitamos la selección de Has Exit Time.

Posteriormente crearemos una nueva condición para que cuando la velocidad sea mayor a 0.01 se realice la transición de stand a run.



Procederemos de la misma forma para crear las demás transiciones, con algunas diferencias que iremos resaltando.

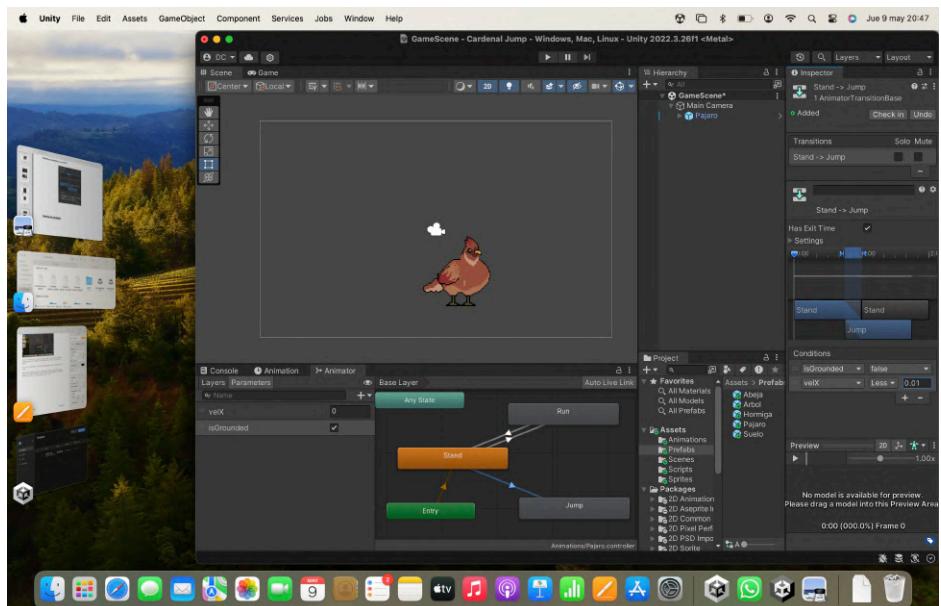
Para la transición de Run a Stand usaremos la misma variable creada anteriormente pero cambiaremos la condición para que cuando la velocidad sea menor a 0.01 realice dicha transición.



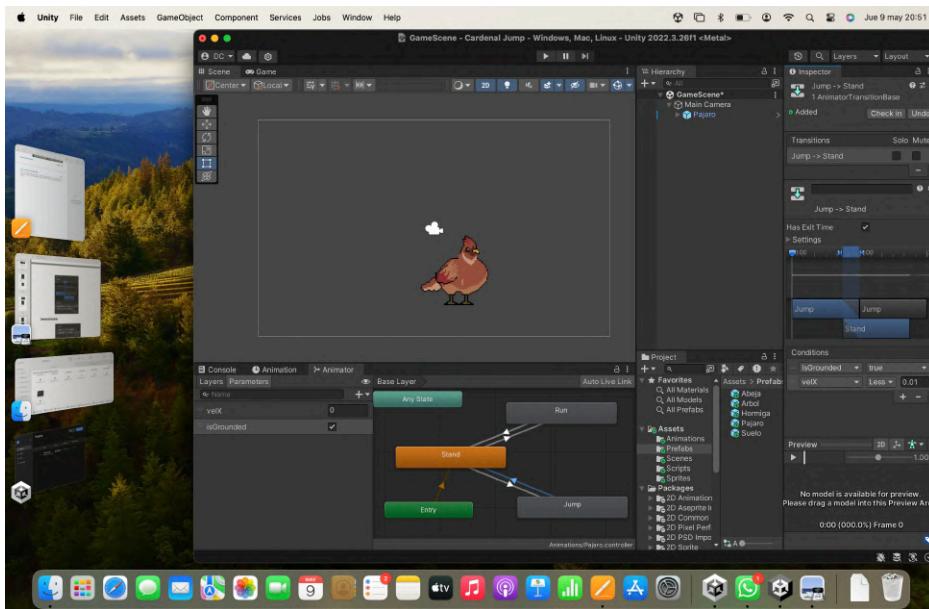
Para la transición de Stand a Jump vamos a crear una variable booleana que nos indique si el personaje esta o no tocando el suelo.

Para ello crearemos al variable `isGrounded` y por defecto será `true`.

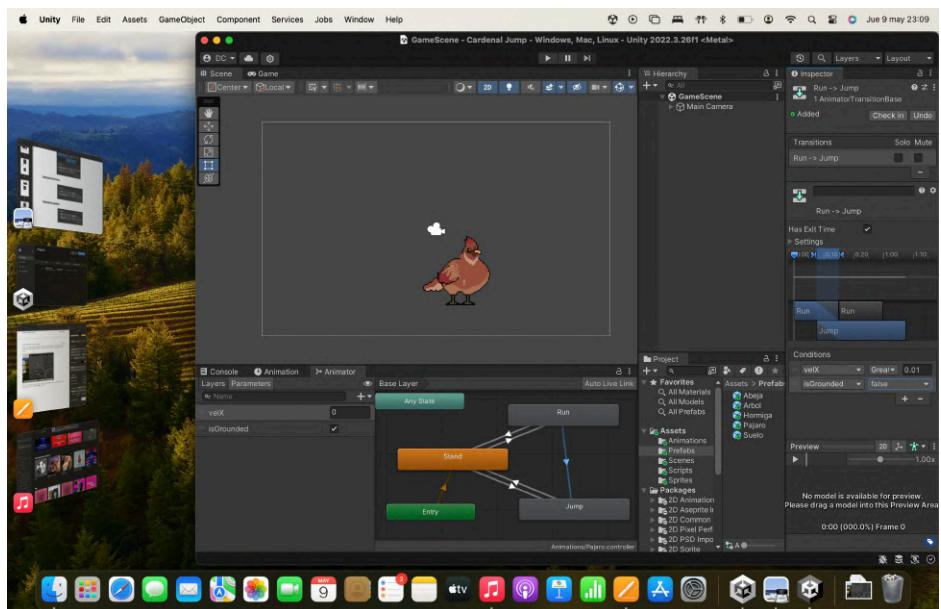
Como esta transición se va a ejecutar cuando el personaje esta parado y no está tocando el suelo vamos a configurar la variable booleana a `false` y la velocidad menor a 0.01.



Para la transición de Jump a Stand configuraremos las variables al contrario que anteriormente (`isGrounded` a `true` y `velX` menor que 0.01).

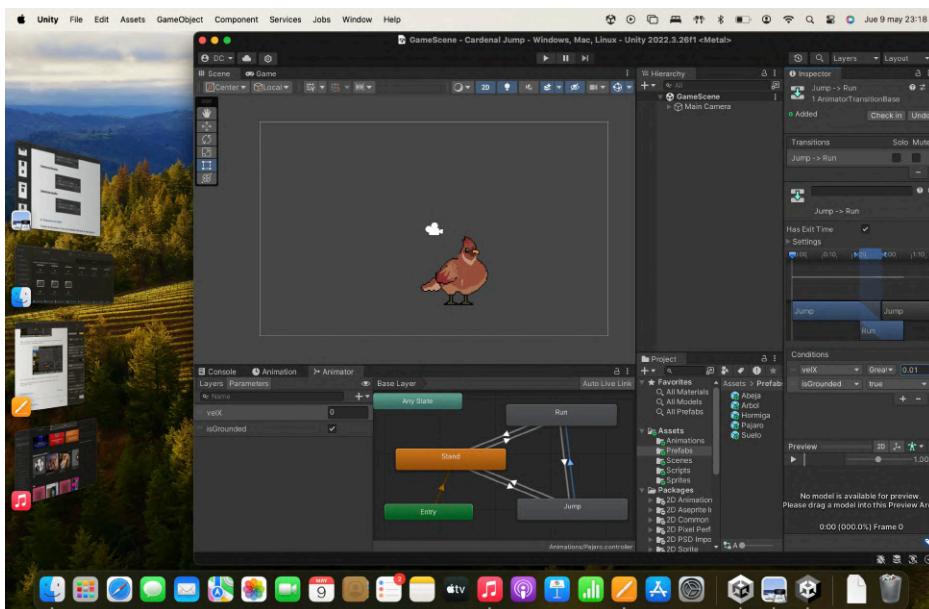


Para la transición de Run a Jump configuraremos las variables `velX` mayor que 0.01 y `isGrounded` a false.



Y por último nos queda por hacer la transición de Jump a Run.

Para ello configuraremos las variables `velX` mayor que 0.01 y `isGrounded` a true.



Lo siguiente que haremos será programar el salto del personaje.

Para ello crearemos un archivo dentro de la carpeta script llamado ControladorPersonaje.

En dicho archivo (en void Update) vamos a crear el código para que el personaje salte cuando tocamos el botón izquierdo del ratón, la pantalla si es táctil, la barra espaciadora o la flecha hacia arriba.

Para ello crearemos una variable de tipo float llamada fuerzaSalto y la inicializaremos a 100f y un if que comprobara si se ha tocado el botón izquierdo del ratón, la pantalla, la barra espaciadora o la flecha hacia arriba, en cuyo caso le aplicara una fuerza en el eje Y equivalente al contenido de la variable fuerzaSalto.

```
//Detección de toque en pantalla, botón izquierdo del ratón, flecha arriba o barra espaciadora
if (Input.GetMouseButtonDown(0) || Input.GetKey(KeyCode.UpArrow) || Input.GetKey(KeyCode.Space))
{
    Debug.Log("Hola");
    GetComponent<Rigidbody2D>().AddForce(new Vector2(0, fuerzaSalto));
}
```

También aumentaremos la fuerza de la gravedad para que el personaje caiga mas fuerte.

Para ello modificaremos el valor de la propiedad Gravity Scale que se encuentra dentro del componente Rigidbody 2D.

Para ello crearemos un objeto vacío que ubicaremos dentro del personaje al que llamaremos ComprobadorSuelo, el cual ubicaremos justo debajo de las patas del personaje.

Tras ello volvemos a abrir el script y declaramos 4 variables nuevas que serán las siguientes:

```

private bool enSuelo = true;
public Transform comprobadorSuelo;
float comprobadorRadio = 0.07f;
public LayerMask mascaraSuelo;

```

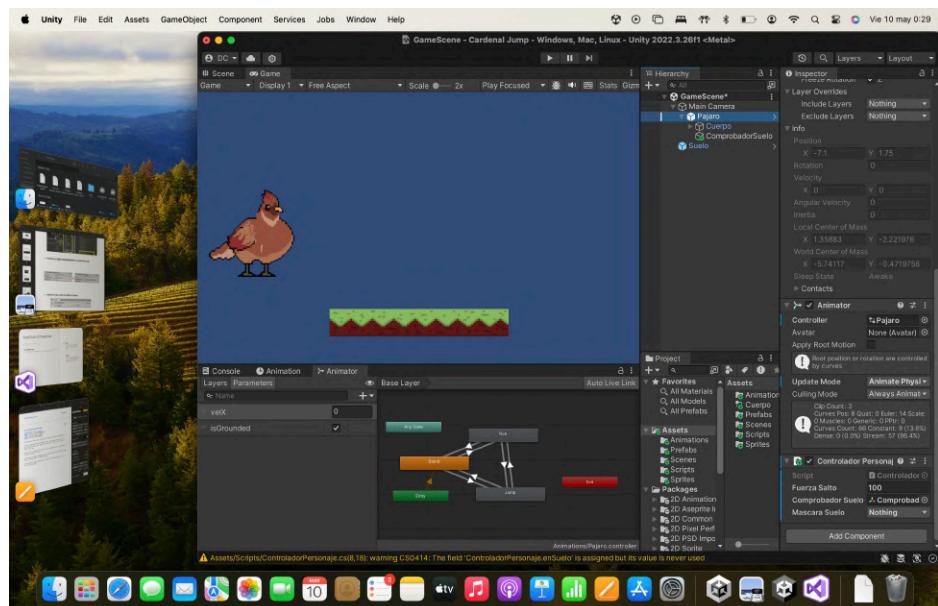
El siguiente paso será limitar el número de saltos consecutivos.

Vamos a limitar que solo pueda saltar dos veces seguidas, por lo que controlaremos, que cuando no esté en el suelo, solo pueda saltar una vez.

Para ello lo primero que haremos será comprobar que el personaje no esta tocando el suelo.

Crearemos una capa llamada suelo clicando en Edit Layers y en User Layer 8 incluiremos el nombre suelo.

A continuación arrastraremos el objeto ComprobadorSuelo a la variable de script.



Tras ellos añadiremos la capa suelo al prefab del suelo.

Seguidamente modificaremos la condición del if que creamos en Update para comprobar que este en el suelo ademas de las condiciones que ya tenia.

A continuación vamos a crear el método FixedUpdate que comprobara si ha tocado suelo el personaje.

Dentro de él incluiremos el siguiente código para que lo compruebe:

```
enSuelo = Physics2D.OverlapCircle(comprobadorSuelo.position,
comprobadorRadio, mascaraSuelo);
```

A continuación vamos a crear una variable booleana llamada dobleSalto que inicializaremos a false y crearemos un if dentro del método FixedUpdate que comprobará si esta en el suelo el personaje, y si es así la variable dobleSalto será true.

```
if (enSuelo) {
    dobleSalto = true;
}
```

Tras ello vamos a modificar la parte del código del método Update.

En él vamos a modificar para que compruebe si, por un lado, esta en el suelo o el doble salto este en false y por el otro lado si el usuario a pulsado alguno de las teclas y/o botones configurados haga que el personaje salte.

Además contendrá un if para que compruebe si no esta en el suelo y el doble salto sea false cambiar el valor de doble salto a true.

```
// Update is called once per frame
void Update()
{
    //Detección de toque en pantalla, botón izquierdo del ratón, flecha arriba o barra espaciadora
    if ((enSuelo || !dobleSalto) &&
        [(Input.GetMouseButtonDown(0) || Input.GetKeyDown(KeyCode.UpArrow) || Input.GetKeyDown(KeyCode.Space))]
    {
        Debug.Log("Hola");
        GetComponent<Rigidbody2D>().AddForce(new Vector2(0, fuerzaSalto));

        // Comprobar lo del doble salto
        if (!enSuelo && !dobleSalto) {
            dobleSalto = true;
        }
    }
}
```

A continuación haremos que se ejecute la animación Jump cuando el personaje salta.

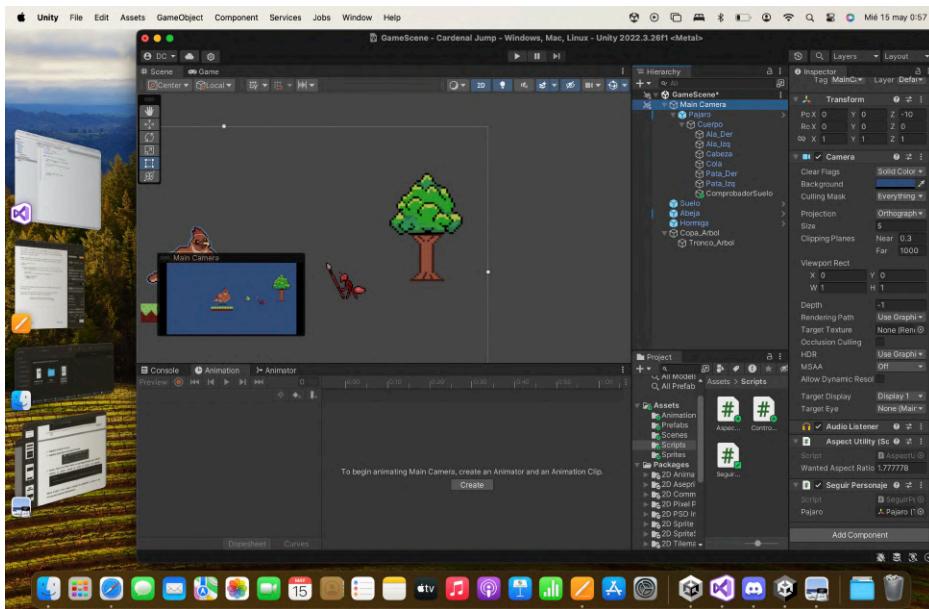
Para ello modificaremos el valor de la variable isGrounded accediendo al componente Animator del personaje.

```
// Se modifica la variable isGrounded del animator
GetComponent<Animator>().SetBool("isGrounded", enSuelo);
```

Tras ello vamos a crear el script que hará que la cámara siga a nuestro personaje.

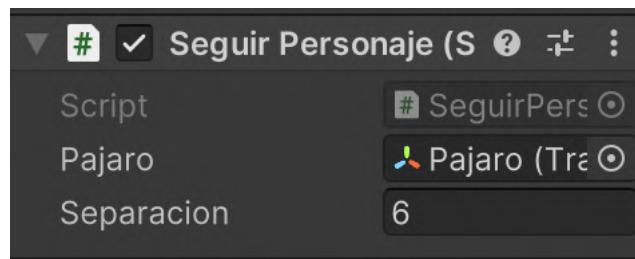
En dicho script haremos referencia al objeto transform del personaje, le asignaremos dicho script a la cámara y posteriormente le asignaremos el personaje al script.

```
// Referencia al objeto transform del personaje
public Transform pajaro;
```



En el método Update vamos a hacer que la cámara modifique su posición en función de la posición del personaje, pero un poco adelantada al mismo, para lo cual creamos una variable de tipo float llamada separación quedando el código como puede verse en la siguiente imagen.

```
// Update is called once per frame
void Update()
{
    transform.position = new Vector3(pajaro.position.x+separacion, transform.position.y, transform.position.z);
}
```



A continuación activaremos la animación run.

Para ello, en el método Update crearemos un if, que en caso de pulsar la tecla F o la flecha derecha estableceremos la velocidad a 10 y se la aplicaremos al personaje, en caso contrario la velocidad será 0.

Podemos verlo en la siguiente imagen.

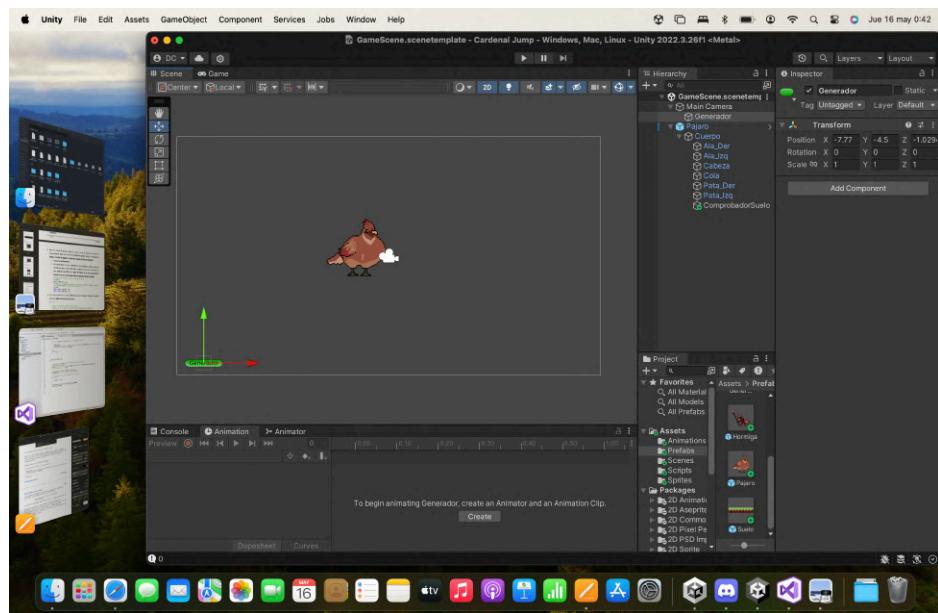
```
if (Input.GetKey(KeyCode.RightArrow) || Input.GetKey(KeyCode.F))
{
    velocidad = 10;
    GetComponent<Rigidbody2D>().velocity = new Vector2(velocidad, GetComponent<Rigidbody2D>().velocity.y);
    Debug.Log("Tecla derecha" + velocidad);
}
else {
    velocidad = 0;
}
```

En el método FixedUpdate incluiremos el siguiente código para actualizar la velocidad horizontal del personaje.

```
GetComponent<Animator>().SetFloat("velX", velocidad);
Debug.Log(velocidad);
```

El siguiente paso será hacer que el suelo se vaya generando solo.

Para ello crearemos un nuevo prefab llamado GeneradorSuelo, un GameObject con el mismo nombre al cual le vamos a cambiar el icono para que sea visible, lo arrastramos a la cámara para que pertenezca a ella y lo ubicamos a la altura a la que queremos que nos vaya creando el suelo.



Tras ello vamos a crear el script al cual llamaremos Generador.

Dicho script contendrá una variable publica de tipo GameObject llamada sueloPrefab que contendrá el prefab del suelo, una variable publica de tipo Transform llamada pájaro que hará referencia al personaje, una variable publica de tipo float con el tamaño del suelo, que será 1, una variable publica de tipo int llamada sueloInicial que será 15 y contraerá el número de piezas del suelo que se generarán al iniciar el juego, una variable privada de tipo float llamada ultimoSueloX que contendrá la última posición donde se generó el suelo, una variable publica de tipo float llamada sueloAltura que será -4 y contendrá la altura a la que se generará el suelo y una variable privada de tipo float que será de -2 y contendrá el número de piezas del suelo que se generarán antes del personaje.

```
public class Generador : MonoBehaviour
{
    public GameObject sueloPrefab; // Prefab del segmento de suelo
    public Transform pajaro; // Referencia al jugador
    public float sueloTamaño = 1f; // Longitud de cada segmento de suelo
    public int sueloInicial = 15; // Número inicial de segmentos de suelo
    private float ultimoSueloX; // Última posición X donde se generó el suelo
    public float sueloAltura = -4f; // Altura a la que se generar el suelo
    private float inicio = -2; // Piezas de suelo que se generaran antes del personaje
```

En el método start generaremos las primera piezas del suelo mediante un bucle for y comprobara la posición de la ultima pieza de suelo puesta.

```
// Start is called before the first frame update
void Start()
{
    // Generar los primeros segmentos de suelo
    for (int i = 0; i < sueloInicial; i++)
    {
        Vector3 spawnPosition = new Vector3(inicio + i * sueloTamanio, sueloAltura, 0);
        Instantiate(sueloPrefab, spawnPosition, Quaternion.identity);
    }
    ultimoSueloX = (sueloInicial - 1) * sueloTamanio;
}
```

En el método Update iremos generando las piezas del suelo según se vaya acercando el jugador a la pieza final del suelo actual.

Esto lo haremos con un if que comprobara la posición del pájaro y la posición de la ultima pieza de suelo actual, en cuyo caso llamará al método GeneradorSuelo.

```
// Update is called once per frame
void Update()
{
    // Generar nuevos segmentos de suelo si el jugador se acerca al final del suelo actual
    if (pajaro.position.x > ultimoSueloX - (sueloInicial - 1) * sueloTamanio)
    {
        GeneradorSuelo();
    }
}
```

Por último crearemos el método GeneradorSuelo, el cual será el encargado de ir generando una pieza nueva de solo según vayamos avanzando.

```
void GeneradorSuelo()
{
    ultimoSueloX += sueloTamanio;
    Vector3 nuevaPosicion = new Vector3(ultimoSueloX, sueloAltura, 0);
    Instantiate(sueloPrefab, nuevaPosicion, Quaternion.identity);
}
```

Tras ello vamos a generar los arboles, las hormigas y las abejas de forma aleatoria.

Para ello vamos a crear un prefab llamado GeneradorItems, un GameObject y un script con el mismo nombre.

El script tendrá una variable que hará referencia al prefab de la pieza de suelo, otra que hará referencia al personaje, otra que contendrá la longitud de cada pieza de suelo, otra que contendrá la posición de la última pieza de suelo, otra que contendrá la altura a la que se va a generar cada pieza de suelo y otra que contendrá las piezas de suelo que se generarán antes del personaje, es decir las que estarán a su izquierda.

En el método Start crearemos un for que se repetirá el número de veces que contenga la variable sueloInicial (la encargada de

guardar el número de piezas de suelo que se generaran al principio).

Por cada pieza de suelo que coloque instanciará una variable de tipo Vector3 con la posición donde debe colocarse más el número de pieza por la que va el for, multiplicado por el tamaño de cada pieza de suelo, e indicará la altura a la que tiene que aparecer (indicado por la variable sueloAltura).

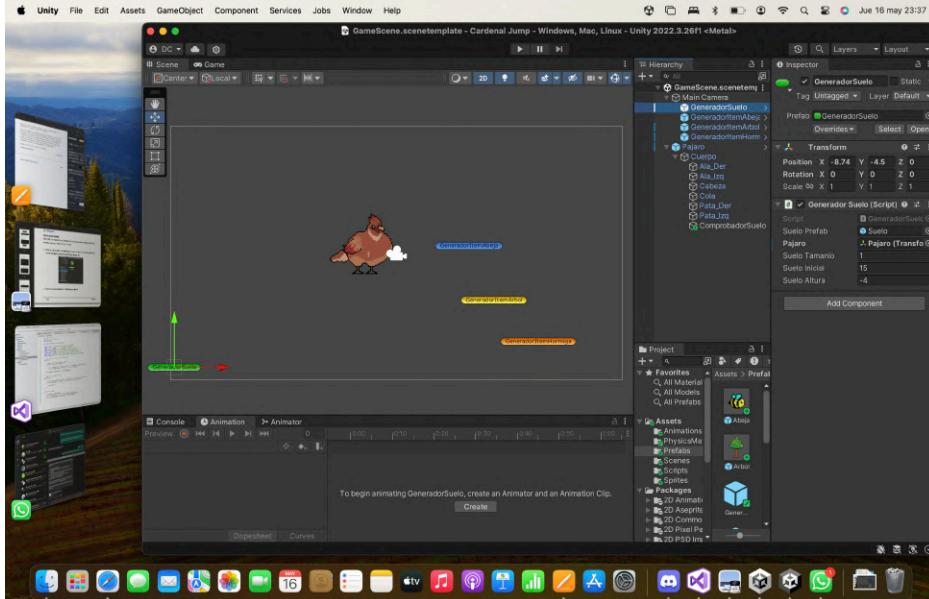
Mediante Instantiate, pasándole el prefab del suelo, la variable de tipo Vector3 creada anteriormente, a la que llamamos spawnPosition, y el quaternion crearán cada pieza del suelo.

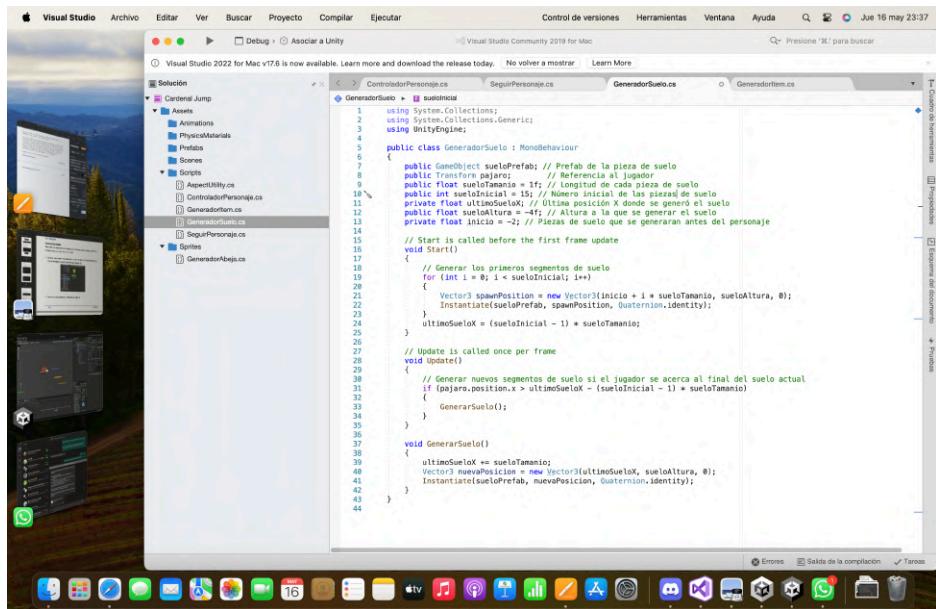
Ya fuera del bucle que nos genera el número de suelos iniciales instanciaremos la variable ultimoSueloX con la posición que ha ocupado el último suelo que ha sido puesto en la escena.

Crearemos un método llamado GenerarSuelo.

Este método irá actualizando el valor de la variable ultimoSueloX y, al igual que en el método Start, creando e instanciando una variable de tipo Vector3 y el método Instantiate crearemos cada bloque según vaya avanzando el personaje.

Podemos verlo en las siguientes imágenes.





Seguiremos los mismos pasos anteriores para crear un prefab para cada item que vamos a generar (abeja, árbol y hormiga), es decir, crearemos un prefab y un GameObject para cada uno y un único script que compartirán los tres.

Necesitamos crear uno para cada objeto ya que no conseguía que me apareciera cada objeto a una altura determinada, por lo cual he creado los tres y puesto cada uno a la altura deseada en la escena.

En una de las imágenes anteriores podemos ver donde está situado cada GameObject.

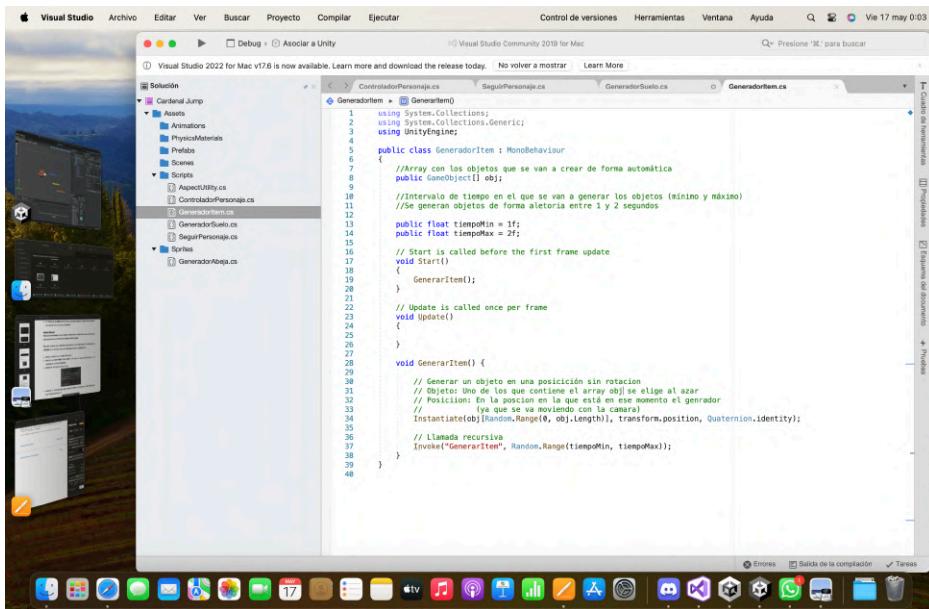
El script que van a compartir contendrá un array de tipo GameObject llamado obj y dos variable de tipo flor que contendrán el tiempo mínimo y el tiempo máximo que tardarán en aparecer cada item, llamadas tiempoMin y tiempoMax respectivamente.

Crearemos un método GenerarItem que llamará al método Instantiate, al cual le pasaremos el objeto que contiene el array, el tiempo mediante un Random.Range, la posición del personaje y el quaternion.

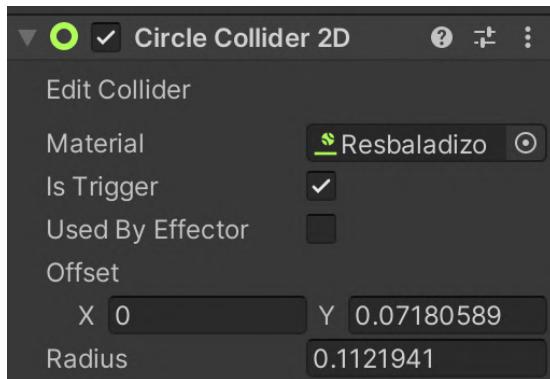
Mediante el método Invoke, haremos una llamada recursiva al método GenerarItem con un tiempo aleatorio entre llamadas dado por las variables tiempoMin y tiempoMax.

A cada uno de los generadores le daremos un tiempo de aparición diferente para que no se solapen los item cuando se van generando de forma aleatoria.

En la siguiente imagen podemos ver el contenido de dicho script.



Tras ello vamos a clicar la opción `is Trigger` en la abeja y la hormiga, para que cuando el personaje se choque con ellos no se detenga (luego gestionaremos que pasa cuando se choca contra estos dos ítems), mientras que en el árbol la dejaremos desmarcada, para que se detenga cuando se choque contra él.

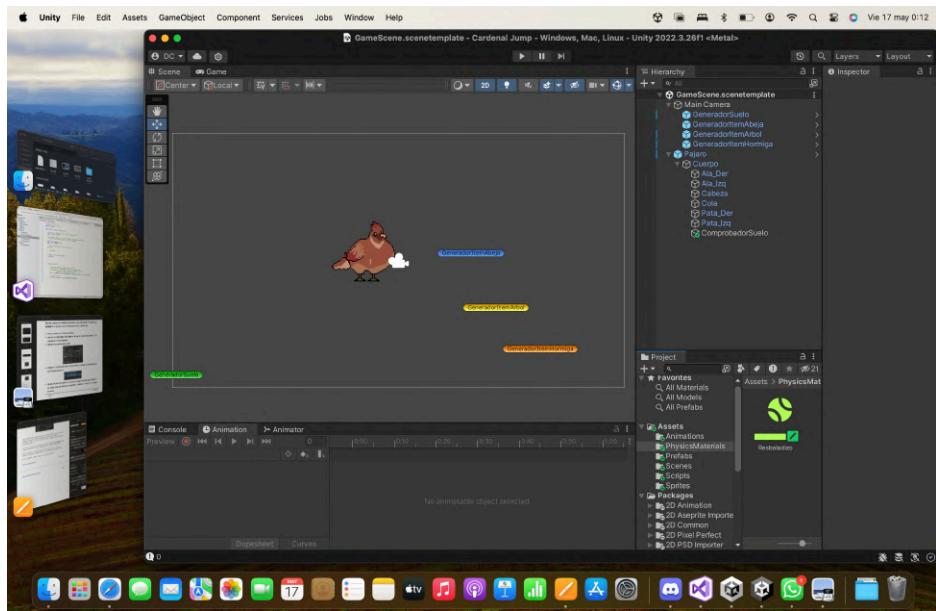


Como también puede apreciarse en la imagen anterior, he creado una tiempos de material llamado Resbaladizo para que cuando el personaje choque contra los collider que condenan dicho material su fricción sea 0.

Para ello he creado una carpeta en asset llamada PhysicsMaterials y dentro de ella un Physics Materials 2D llamado resbaladizo.

Posteriormente, en cada uno de los collider (de los item), he añadido el material Resbaladizo en el apartado correspondiente.

En la siguiente imagen podemos ver dicho material.



Para crear las notificaciones he tenido que añadir el script `NotificationCenter.cs`.

Tras ello he modificado el script `ConroladorPersonaje` y he añadido dos notificaciones, una para cuando el personaje empieza a correr y otra para cuando el personaje para se detiene.

El if que contiene el código para que el personaje empiece a correr ha quedado de la siguiente manera:

```
if (Input.GetKey(KeyCode.RightArrow) || Input.GetKey(KeyCode.F))
{
    velocidad = 10;
    GetComponent<Rigidbody2D>().velocity = new Vector2(velocidad, GetComponent<Rigidbody2D>().velocity.y);
    Debug.Log("Tecla derecha" + velocidad);
    NotificationCenter.DefaultCenter().PostNotification(this, "InicioCorrer");
}
else {
    velocidad = 0;
    NotificationCenter.DefaultCenter().PostNotification(this, "FinCorrer");
}
```

Este código nos permite enviar una notificación llamada `InicioCorrer` cuando el personaje empieza a correr y otra llamada `FinCorrer` cuando el personaje se detiene.

A continuación me toca modificar el script que genera los bloques.

Al hacerlo igual que indica la tarea, cuando el personaje se paraba, se seguían generando ítems por lo que se saturaba la pantalla de ítems.

Para evitar esto he tenido que modificar todo el script `GeneradorItem`.

Le he añadido una variable nueva, privada de tipo booleana llamada `corriendo`.

En el método Start he tenido que registrar los métodos InicioCorrer y FinCorrer como observadores de las notificaciones correspondientes y he llamado al método generarItem.

El método InicioCorrer se llama cuando se recibe la notificación "InicioCorrer". Si no se está generando ítems, inicia GenerarItem.

El método `FinCorrer` es llamado cuando se recibe la notificación “`FinCorrer`”, instancia a `false` al variable corriendo y contiene un `if` en el que comprueba si esta generando ítems, en cuyo caso instanciará la variable corriendo a `false`.

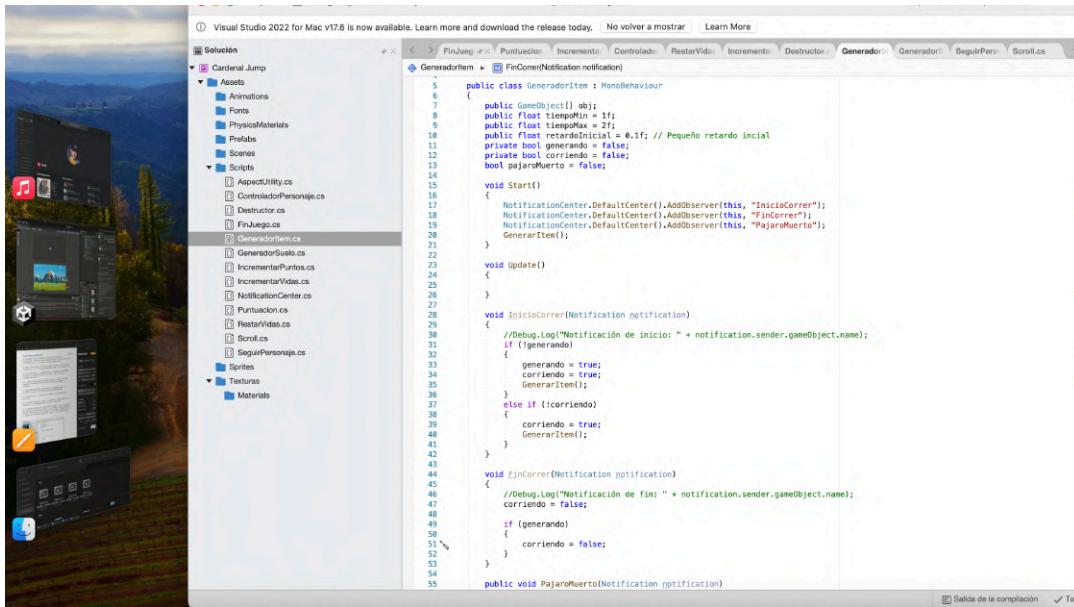
En el método GenerarItem, será de tipo public void y generará ítems en intervalos aleatorios definidos por tiempoMin y tiempoMax mediante el método Invoke.

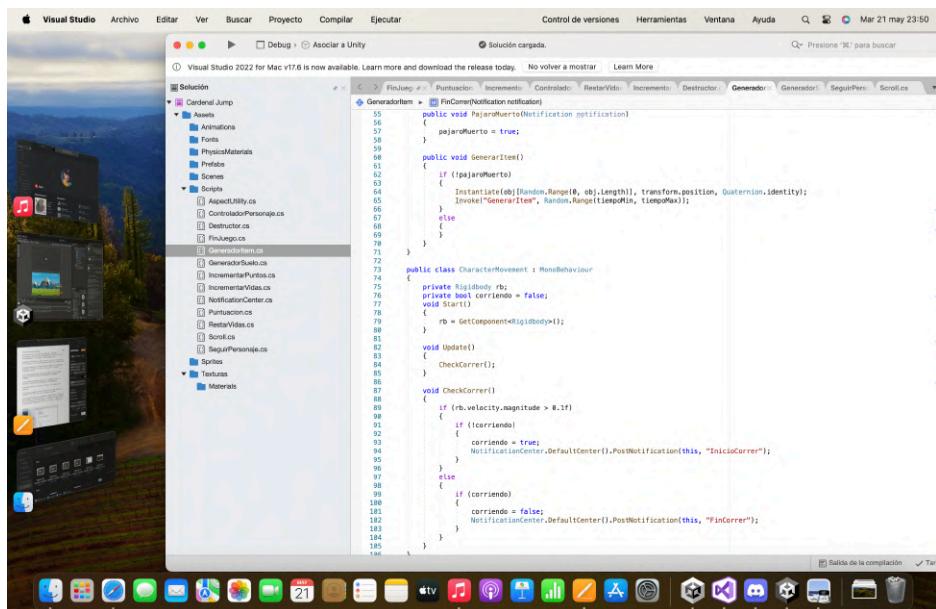
La clase CharacterMovement detecta si el personaje está en movimiento y envía las notificaciones correspondientes.

En el método Start se obtiene la referencia al Rigidbody del personaje.

En el Update llamamos al método CheckCorrer en cada frame para comprobar si el personaje está corriendo o detenido.

En el método CheckCorrer se comprueba la magnitud de la velocidad del Rigidbody. Si es mayor que 0.1, el personaje está corriendo y envía la notificación "InicioCorrer". Si es menor o igual a 0.1, el personaje está detenido y envía la notificación "FinCorrer".





En las imágenes anteriores podemos ver el código del script descrito anteriormente.

El siguiente paso que haremos será crear el destructor de objetos.

Para ello crearemos un objeto, al que llamaremos Destructor, que situaremos detrás de la cámara.

Tras ello le añadiremos un box collider 2D al cual le daremos el alto de la escena de juego para que colisione con todos los objetos que salgan de dicha escena, le activaremos la opción is Trigger y le añadiremos un Rigidbody2D al cual le cambiaremos el valor del GravityScale a 0 para que no se vea afectado por la gravedad.

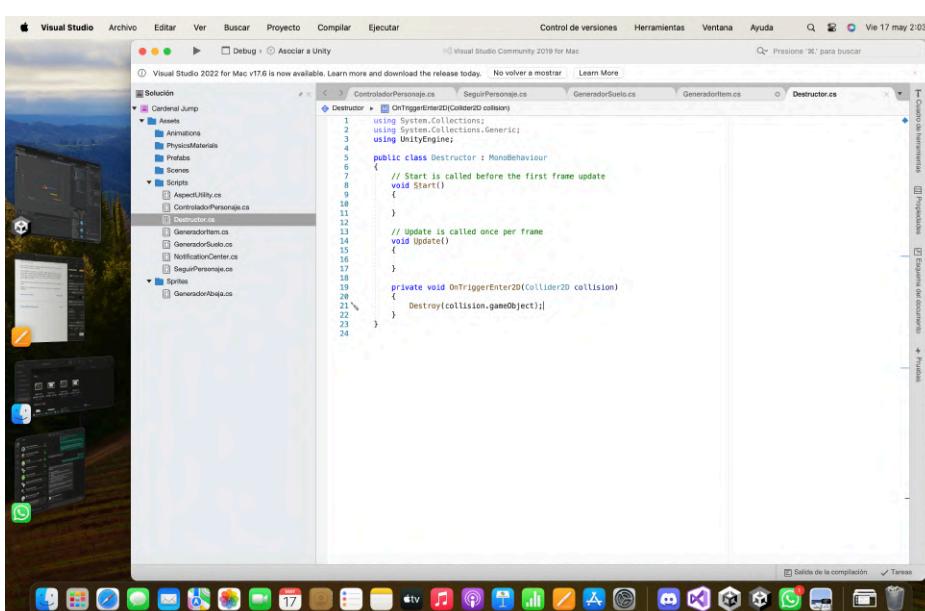
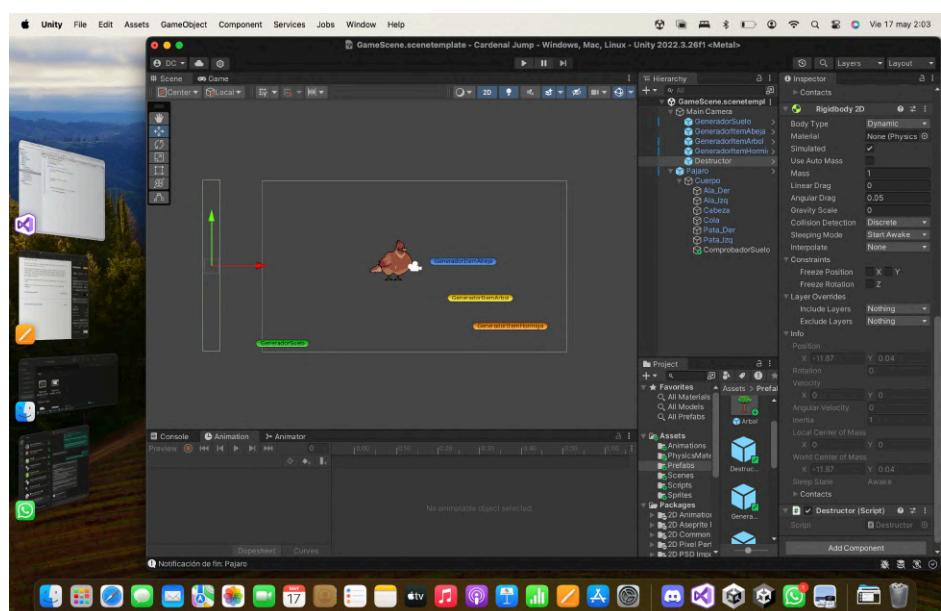
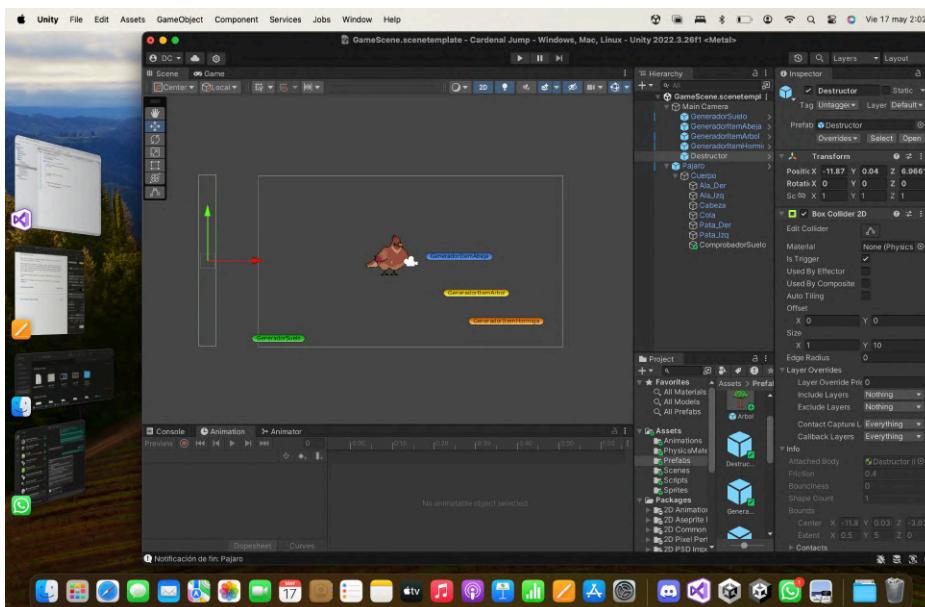
Lo siguiente que haremos será ubicarlo detrás de la escena de juego y haremos que dependa, en la jerarquía, de la cámara para que avance con ella.

Tras ello he creado un prefab llamado Destructor y en incluido en él el objeto Destructor creado.

Posteriormente crearemos un script llamado Destructor en el cual incluiremos el evento OnTriggerEnter2D que será lanzado cuando el destructor choque contra otro collider, y dentro de dicho evento incluiremos el método Destroy que será quien destruya el objeto que choque contra él.

A continuación aplicaremos el script al destructor de la escena de juego.

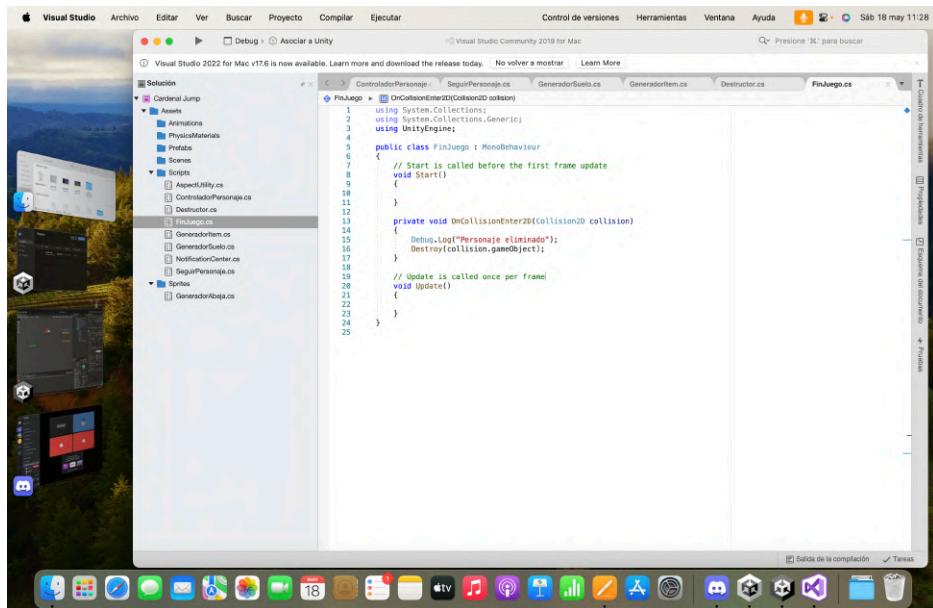
Podemos ver el código del script y el objeto en las siguientes imágenes.



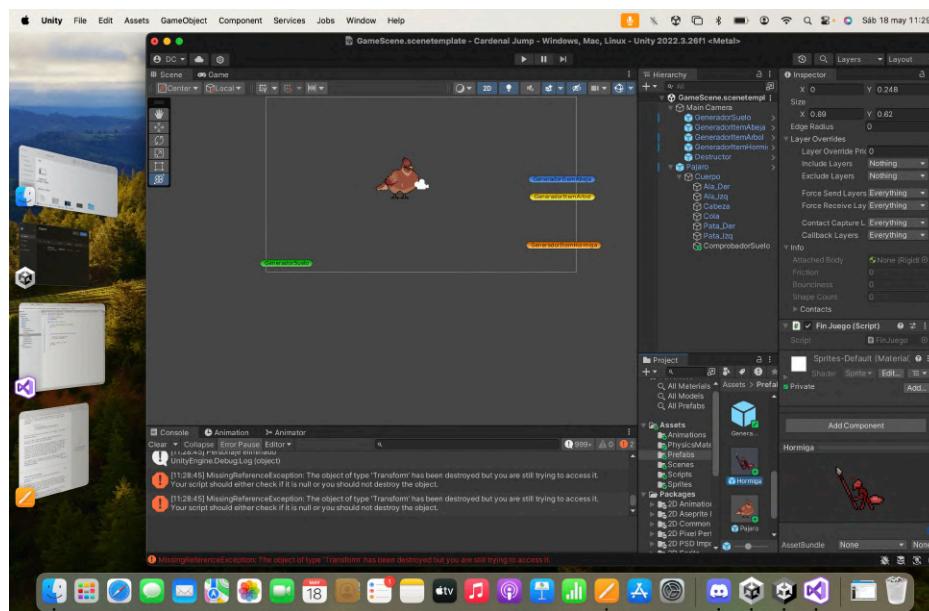
El siguiente paso será crear el script para que el personaje muera cuando choca con la hormiga o cuando choca con el tronco del árbol.

Para ello vamos a crear un script al cual llamaremos FinJuego.

Dicho script contendrá un método de tipo OnCollisionEnter2D con un Destroy(collision.gameObject); dentro de él, para que cuando la hormiga choque contra un objeto, en este caso el pájaro, muera.



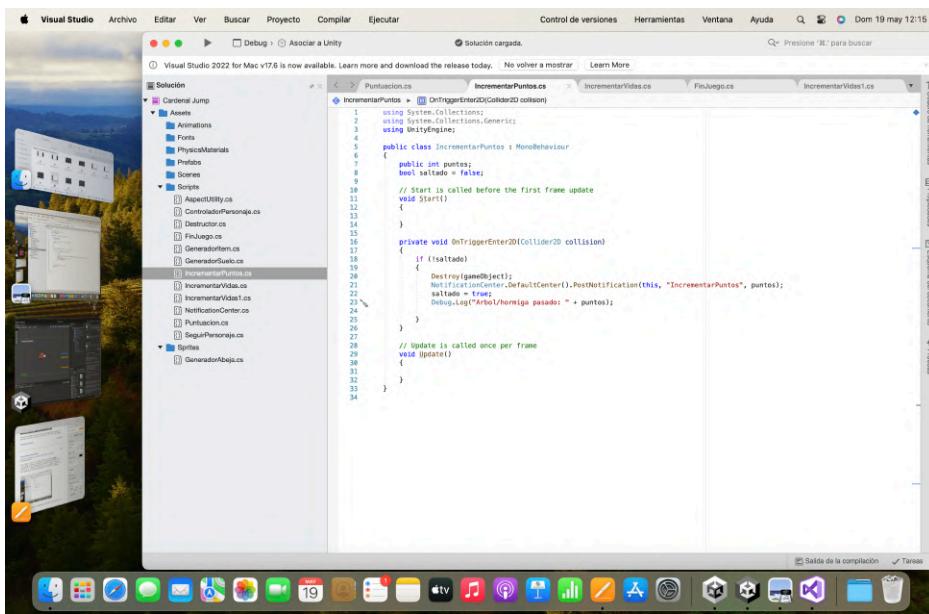
Tras ello le vamos a asignar el script a la hormiga y al tronco del árbol.



Para que nos muestre la puntuación necesitamos saber cuando el personaje salta por encima de la hormiga y por encima del árbol.

Para ello crearemos un script al que vamos a llamar IncrementarPuntos.

En el vamos a crear una variable booleana llamada saltado inicializada a false, un método de tipo OnCollisionEnter2D que contendrá un if que comprobará si el objeto salta por encima del item, en cuyo caso si la variable booleana es true, mandará un NotificationCenter y sumará los puntos.



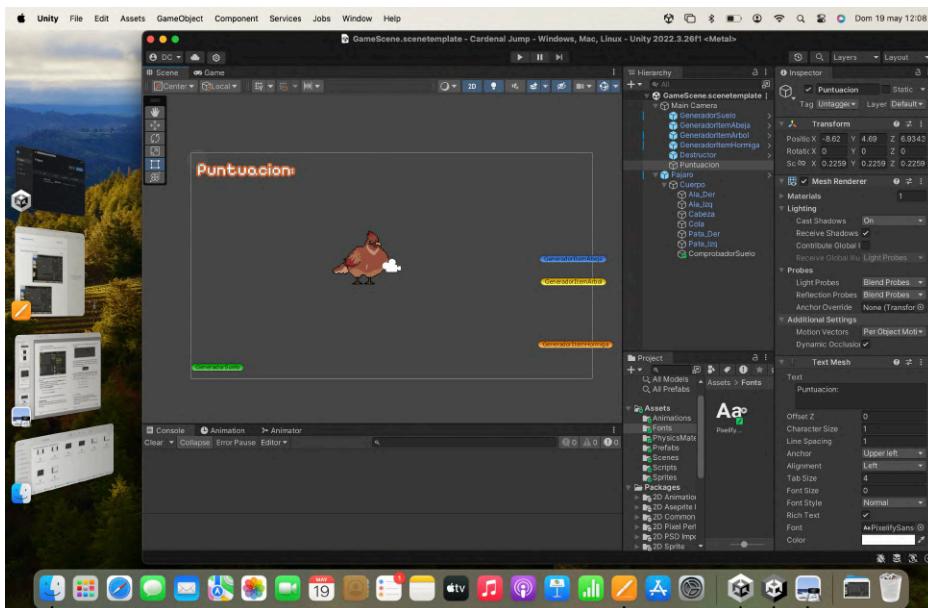
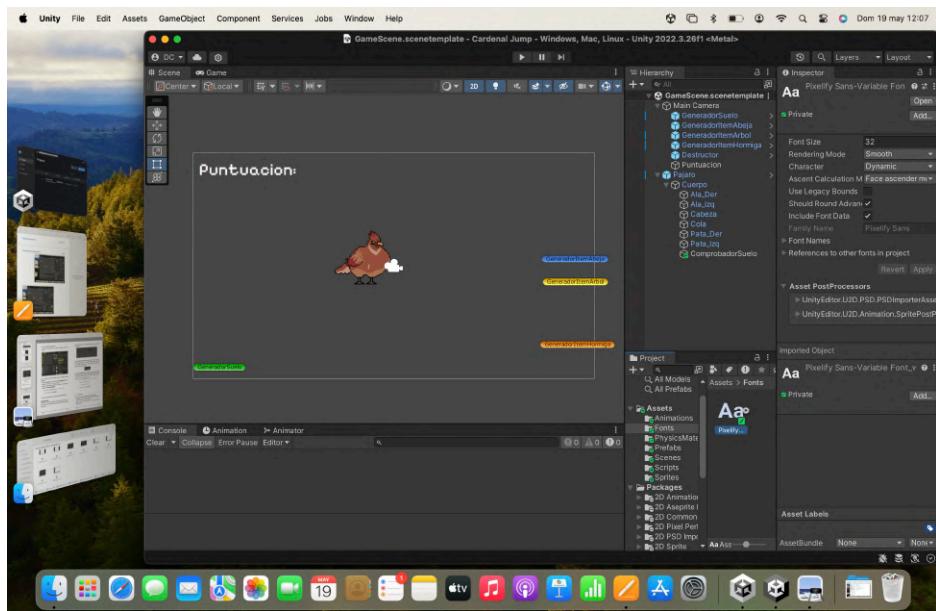
Tras ello vamos a crear el TextMesh que mostrará la puntuación del jugador.

Para darle una fuente diferente y acorde al juego que estamos creando he buscado en google Font una fuente de tipo pixel llamada pixelify.

Dicha fuente la descargamos, creamos una carpeta en asset llamada Fonts e incluimos en ella la fuente descargada.

Tras ello, en el TextMesh creado, seleccionamos como fuente la incluida en la carpeta fonts.

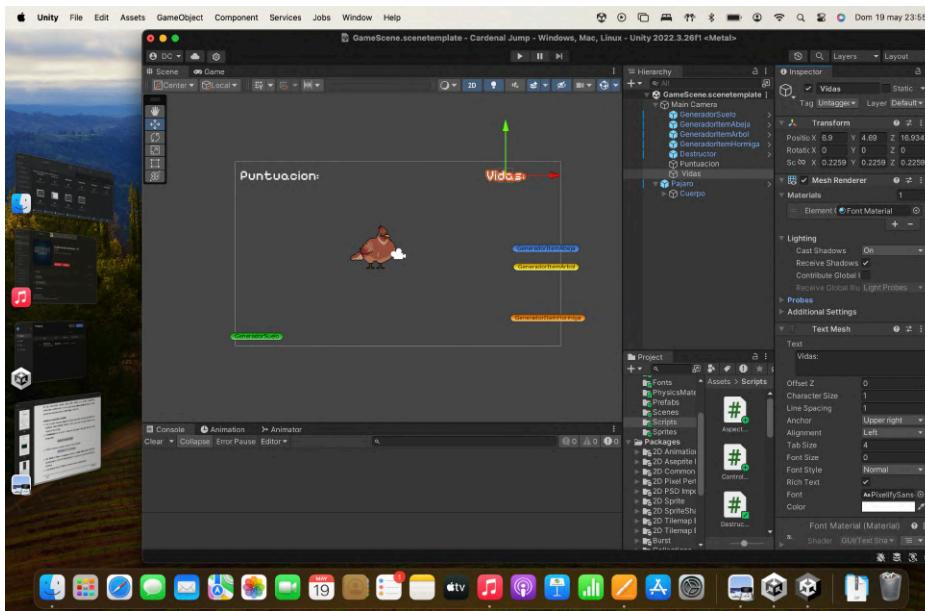
En las siguientes imágenes podemos ver la fuente incluida en la carpeta fonts y dicha fuente seleccionada en el TextMesh creado.



Los mismos pasos anteriores seguiremos para crear el TextMesh para que nos muestre la cantidad de vidas de las que dispone el jugador.

Usaremos el mismo tipo de fuente y tamaño de letra y lo ubicaremos en el la opuesto a la puntuación.

Podemos verlo en la siguiente imagen.



Lo siguiente que haremos será añadir los fondos.

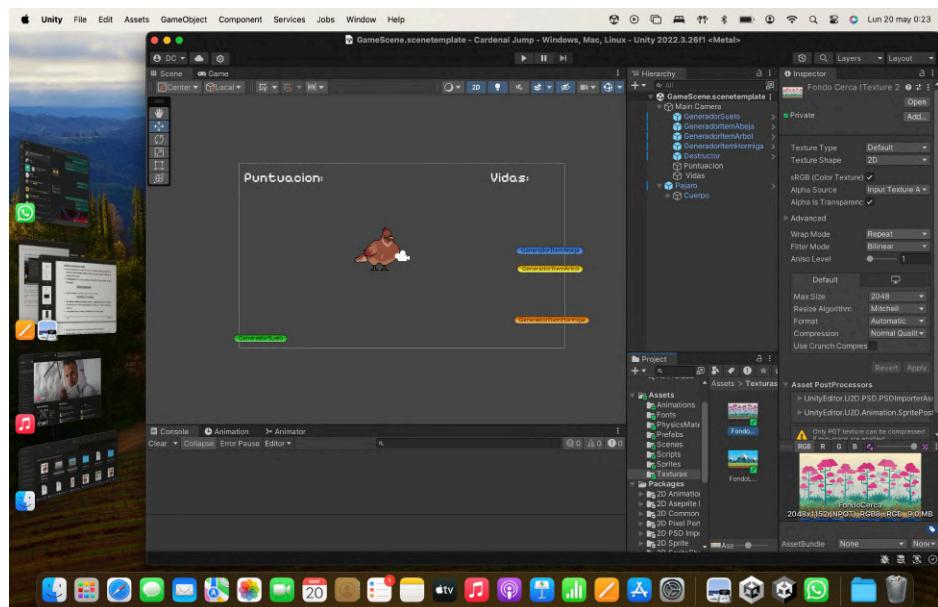
Para ello vamos a crear una carpeta en assets llamada "Texturas" y en ella añadiremos los fondos, el de cerca y el de lejos.

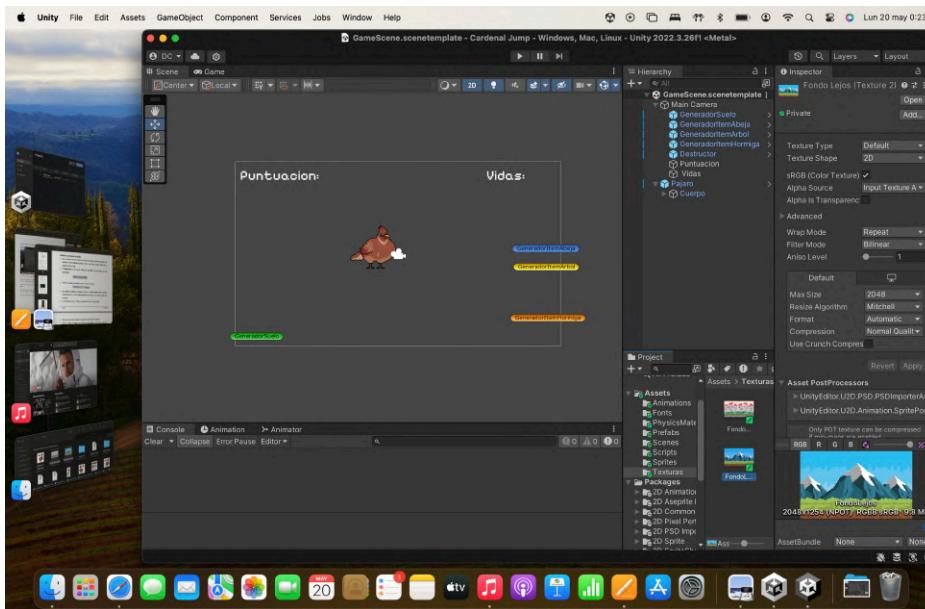
En ambos fondos seleccionaremos Default en la opción "Textures type".

En FondoCerca, como estará delante de FondoLejos activaremos la opción "Alpha is Transparency".

En Wrap Mode seleccionaremos repeat.

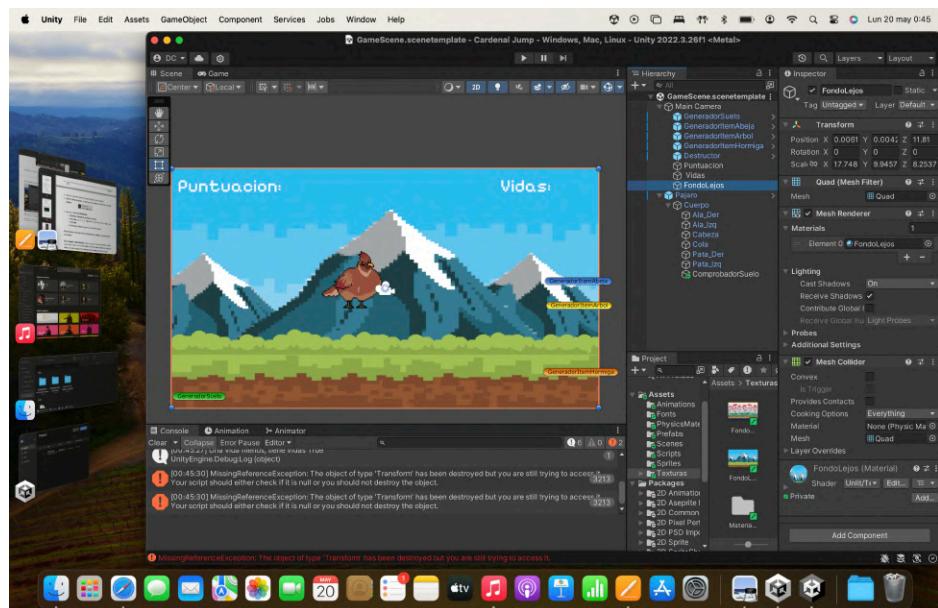
En las siguientes imágenes podemos verlo.





Ahora vamos a crear un objeto Quad, de nombre "FondoLejos" que será hijo de la cámara y tendrá su mismo tamaño.

Arrastramos la extra "FondoLejos" (el fondo que habíamos guardado en Texturas) y le cambiaremos el shader a Unlit/Texture.



Para añadir "FondoCerca" haremos seguiremos los mismos pasos que para el fondo anterior, pero asignándole la textura "FondoCerca".

Lo situaremos ligeramente mas cerca de la cámara que el fondo anterior y le cambiaremos la textura y cambiaremos el shader a Unlit/Transparent.

FondoLejos es del mismo tamaño que la cámara, mientras que FondoCerca solo ocupa la parte inferior de la misma, es decir, justo debajo del suelo.

Para conseguir el efecto Parallax vamos a crear un script al que llamaremos Scroll.

Este script contendrá tres variables, una de tipo publica y float a la que llamaremos velocidad y almacenará la velocidad que tiene el personaje, una de tipo booleana a la que llamaremos enMovimiento que utilizaremos para saber si el personaje está en movimiento o no e inicializaremos a false y un de tipo float llamada tiempoInicio que almacenará el tiempo que pasa el personaje quieto sin darle a correr nada más iniciar la partida.

En el método start escucharemos la notificación de PajaroMuerto, para comprobar si el personaje ha muerto o no.

Crearemos un método llamado InicioCorrer, que tendrá como función pasar a true la variable enMovimiento e instanciar en tiempoInicio el tiempo que ha tardado el jugador en empezar a jugar.

Tras ello crearemos un método llamado PajaroMuerto que recibirá la notificación anteriormente mencionada.

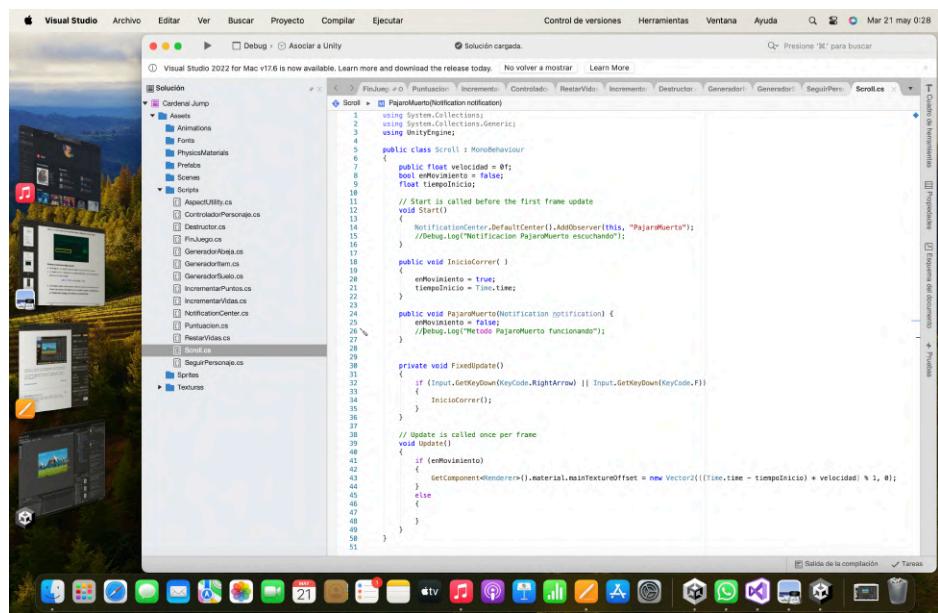
Dicho método instanciará a false la variable enMovimiento para parar el movimiento del fondo.

En el FixedUpdate comprobaremos, mediante un if, si el usuario ha pulsado alguna de las teclas que usamos para empezar a correr (la flecha hacia la derecha y la tecla F) y en el caso de pulsar alguna de las dos teclas mencionadas, llamará al método InicioCorrer.

Por último, en el método Update comprobaremos mediante un if si la variable en movimiento es true, en cuyo caso accederemos al material del renderer, y mediante su offset estableceremos el movimiento a través de un Vector2, el cual calculará el tiempo pasado desde que se inició el juego hasta que el jugador ha pulsado alguna de las teclas de movimiento, lo multiplicará por la velocidad y mantendrá el desplazamiento entre 0 y 1 mediante el modulo de 1 mientras que el desplazamiento en el eje y será 0.

Esto conseguirá que el fondo vaya desplazándose en el eje x a una velocidad predefinida dando así la sensación de avance del personaje.

Podemos ver lo explicado exteriormente en las siguientes imágenes.



Tras ello agregaremos el script a los fondos creados (FondoCerca y FondoLejos) .

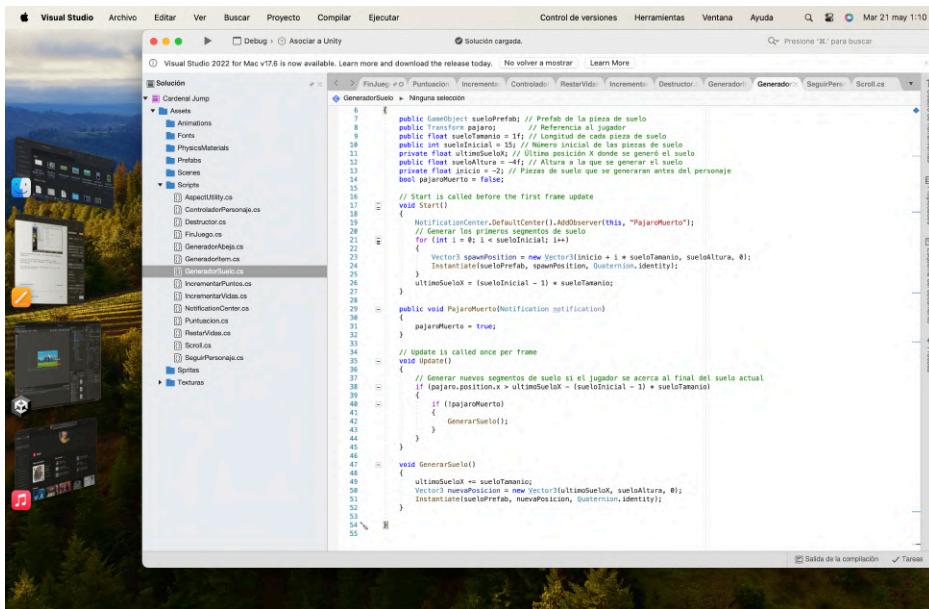
Al clicar sobre los fondos ahora nos aparece la variable velocidad, la cual vamos a establecer en 0.1 para FondoLejos y 0.3 para FondoCerca.

Esto hará que el FondoCerca, situado debajo del suelo vaya mas rápido que el FondoLejos.

Para detener los fondos una vez que el personaje ha muerto, en el script scroll creamos el método PajaroMuerto el cual instanciamos a false la variable enMovimiento.

Para hacerlo en el generador de suelo vamos a crear una variable de tipo bool llamada pajaroMuerto y la instanciaremos a false.

A continuación, dentro del método update, estableceremos un if, que comprobara el valor de la variable creada y, en caso de ser false, generará el suelo.



```

public GameObject sueloPrefab; // Prefab de la pieza de suelo
public Transform pajaro; // Referencia al jugador
public int suelоЦinal = 15; // Número inicial de las piezas de suelo
private float ultimosuelo; // Última posición del suelo se genera el suelo
private float ultimopunto; // Al llegar a la meta se genera el suelo
private float inicio = -2; // Pieza de suelo que se generaron antes del personaje
bool pajaromuerto = false;

// Start is called before the first frame update
void Start()
{
    NotificationCenter.DefaultCenter.AddListener("PajaroMuerto", Pajaromuerto);
}

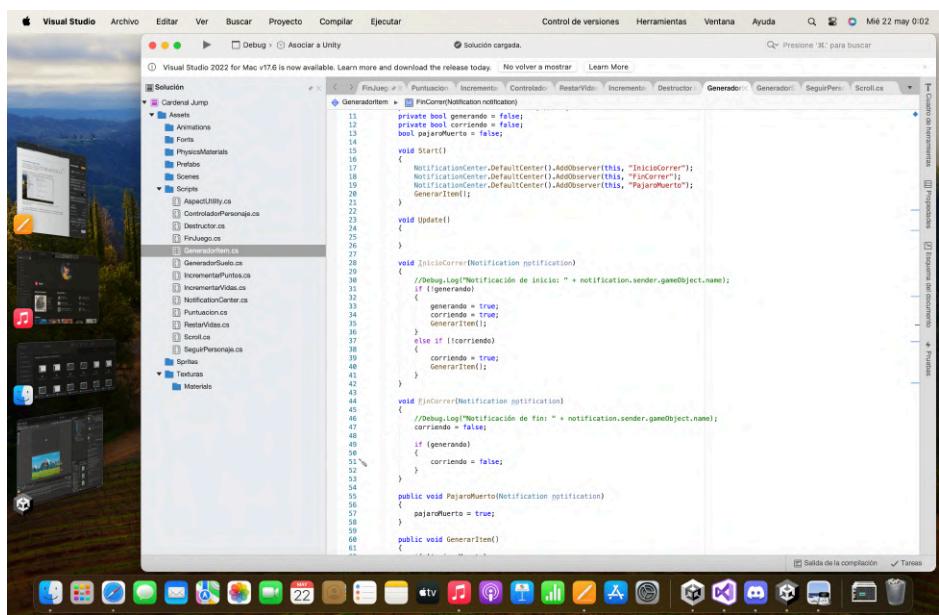
// Generates the first few segments of floor if the player reaches the final point
void Update()
{
    // Generates new segments of floor if the player reaches the final point of the current frame
    if (pajaro.position.x > ultimopunto - (suelоЦinal - 1) * sueloTamaño)
    {
        if (!pajaromuerto)
        {
            GenerarSuelo();
        }
    }
}

void GenerarSuelo()
{
    ultimosuelo += sueloTamaño;
    Vector3 nuevaPosicion = new Vector3(ultimosuelo, suelaAltura, 0);
    Instantiate(sueloPrefab, nuevaPosicion, Quaternion.identity);
}

```

Estos mismos pasos los seguiremos para modificar el script GeneradorItem.

El if que comprobará si el pájaro ha muerto lo incluiremos dentro del método GenerarItem, cuyo contenido será el incluido en dicho if.



```

private bool generando = false;
private bool corriendo = false;
bool pajaroMuerto = false;

void Start()
{
    NotificationCenter.DefaultCenter.AddListener("IniciarCorrer", IniciarCorrer);
    NotificationCenter.DefaultCenter.AddListener("FinJuego", FinJuego);
    GenerarItem();
}

void Update()
{
    IniciarCorrer(Notification notification);
}

void IniciarCorrer(Notification notification)
{
    //Debug.Log("Notificación de inicio: " + notification.sender.gameObject.name);
    if (generando)
    {
        generando = true;
        corriendo = true;
        GenerarItem();
    }
    else if (!corriendo)
    {
        corriendo = true;
        GenerarItem();
    }
}

void FinJuego(Notification notification)
{
    //Debug.Log("Notificación de fin: " + notification.sender.gameObject.name);
    corriendo = false;
    if (generando)
    {
        generando = false;
    }
}

public void PajaroMuerto(Notification notification)
{
    pajaroMuerto = true;
}

public void GenerarItem()
{
}

```

Ya solo nos falta modificar el script FinJuego para desactivar el personaje una vez que este muera.

Para ello, en el método OnCollisionEnter2D crearemos un GameObject llamado pájaro que hará referencia al personaje.

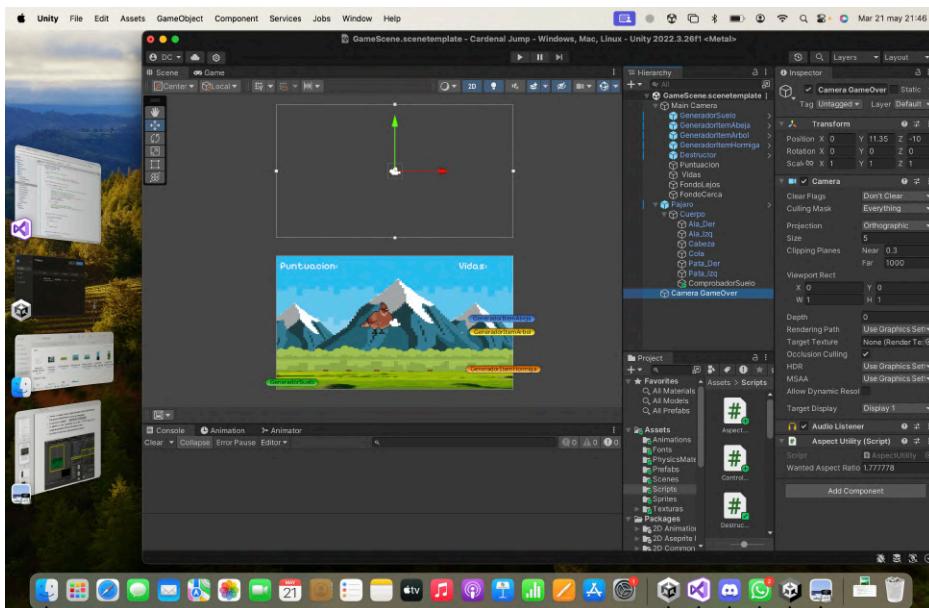
Tras ello desactivaremos el personaje mediante el método SetActive(false).

```
private void OnCollisionEnter2D(Collision2D collision)
{
    if (collision.gameObject.CompareTag("Player"))
    {
        if (!tieneVidas && !vidaRestada) // Verifica si tiene vidas y no se ha restado una vida
        {
            vidaRestada = true; // Marca como vida restada
            //Debug.Log("Una vida menos, tiene vidas " + tieneVidas);
            //Debug.Log("Notificación PajaroMuerto enviada");
            NotificationCenter.DefaultCenter().PostNotification(this, "PajaroMuerto");
            GameObject Pajaro = GameObject.Find("Pajaro");
            Pajaro.SetActive(false);
            Destroy(collision.gameObject);
        }
        else if (!tieneVidas)
        {
            //Debug.Log("Notificación PajaroMuerto enviada");
            //NotificationCenter.DefaultCenter().PostNotification(this, "PajaroMuerto");
            Destroy(gameObject);
        }
    }
}
```

A continuación vamos a crear la pantalla de fin de juego.

Para ello crearemos un nueva cámara llamada Camera GameOver, cambiaremos el atributo Projection a Orthographic, la propiedad clear flags a Don't Clear, le aplicaremos el script Aspect Utility y ajustaremos el valor de relación de aspecto a 1,777778.

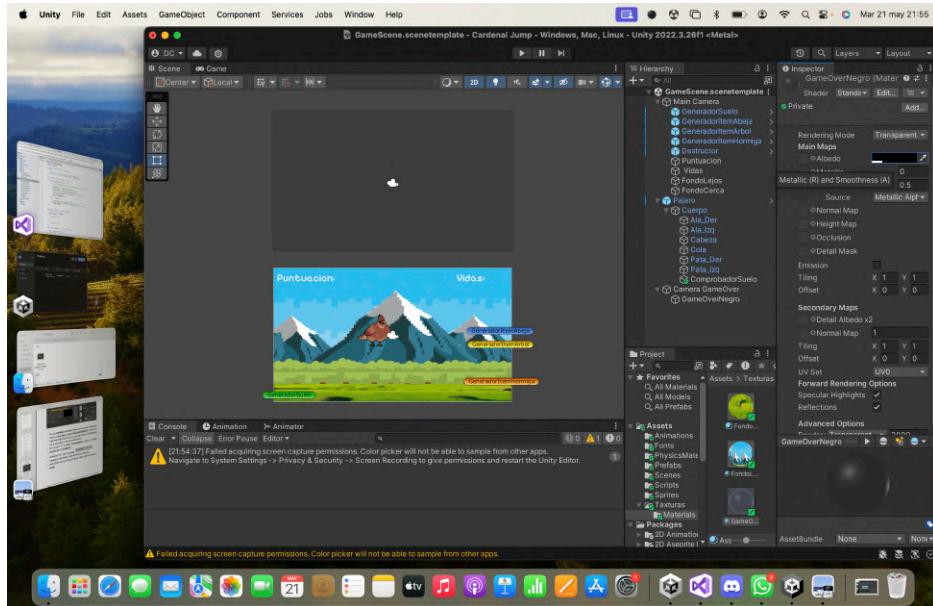
Tras ello vamos a ubicarla encima de la cámara principal en la escena.



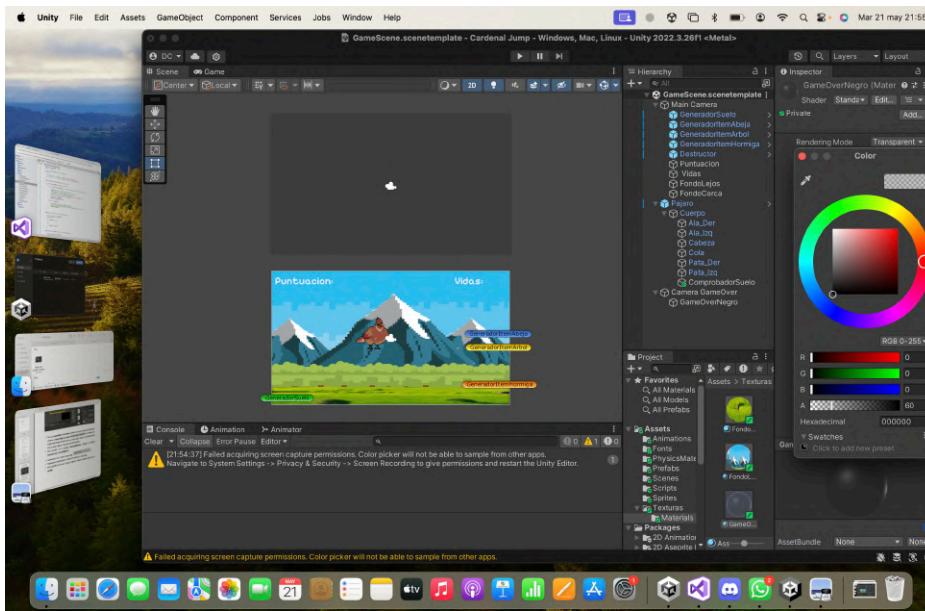
Posteriormente, para que la pantalla se oscurezca, crearemos un objeto Quad que dependerá de la cámara gameover y tendrá el mismo tamaño que la cámara gameover.

Crearemos una nueva textura a la que llamaremos GameOverNegro, le pondremos el color negro y la propiedad rendering mode será transparente.

A continuación le asignaremos el material creado al quad de la cámara gameover.



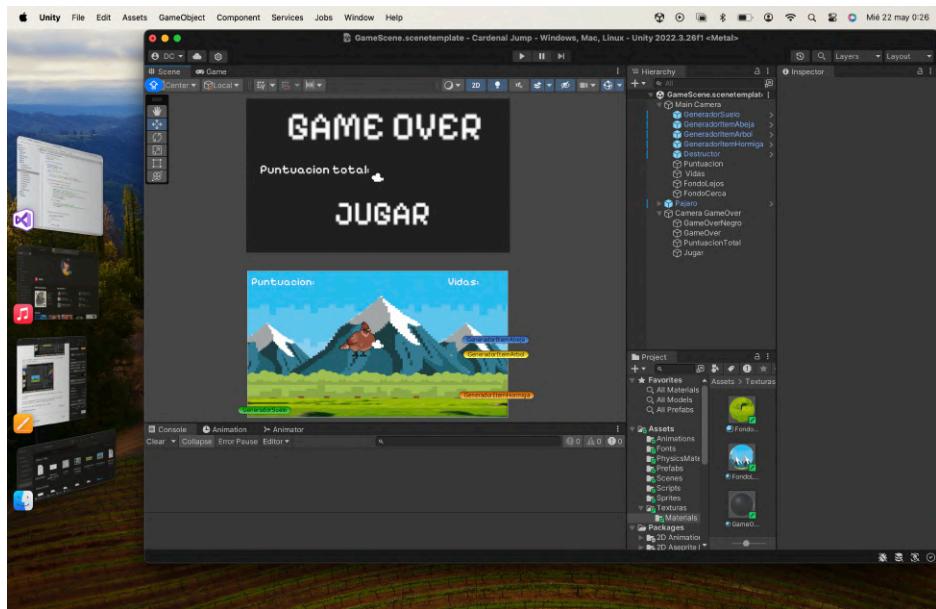
Una vez añadido jugaremos con la propiedad Alta para oscurecerlo hasta el punto deseado.



Lo siguiente que nos ocupa será incluir los textos a la cámara gameover.

Para ello seguiremos los mismos pasos que hicimos al incluir el marcador de puntos y vidas en apartados anteriores.

Añadiremos el texto GAME OVER, total y jugar.



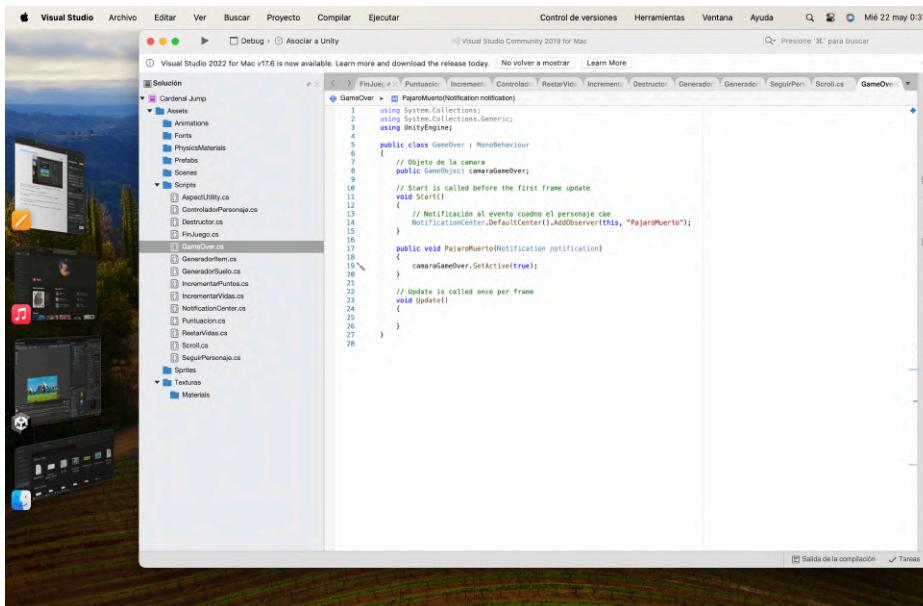
Tras ello vamos a desactivar al cámara para hacer que se active una vez que el personaje muera.

A continuación creamos el script GameOver.

Dicho script contendrá un objeto de tipo GameObject llamado cameraGameOver.

En el método start nos suscribiremos al evento que notifica el fin del juego (PajaroMuerto).

En el método que escucha el evento activaremos la cámara gameOver.



Posteriormente asignaremos el script a la cámara principal y a la cámara gameover.

En el atributo Camara Game Over asignaremos la cámara GameOver, tanto en el script añadido a la cámara principal como en el añadido a la cámara GameOver.

El siguiente paso será añadir la puntuación al TextMesh correspondiente de la cámara GameOver.

Para ello vamos a crear un script llamado ValoresGameOver y asignaremos dicho script a la cámara GameOver.

Para poder mostrar la puntuación desde el script GameOver debemos acceder a dicho atributo del script Puntuacion.cs.

Para ello vamos a crear un método get en el script Puntuacion que nos retornará la puntuación.

The screenshot shows the Visual Studio Community 2019 interface with a C# script named `Puntuacion.cs` open in the code editor. The script contains methods for handling notifications and updating scores and lives. The code includes comments explaining the logic for incrementing points and lives based on notification data.

```
private int vidas = 0;
private int puntos = 0;

void Start()
{
    NotificationCenter.DefaultCenter().AddObserver(this, "incrementarPuntos");
    NotificationCenter.DefaultCenter().AddObserver(this, "incrementarVidas");
    NotificationCenter.DefaultCenter().AddObserver(this, "restarVidas");
}

public void IncrementarPuntos(Notification notification)
{
    // Recoges los puntos a incrementar de la notificación
    int puntosAIncrementar = (int)notification.data;
    // Actualizas la puntuación
    puntuacion += puntosAIncrementar;
    totalPuntos.text = "Puntacion: " + puntuacion;
}

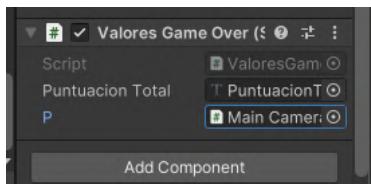
public void IncrementarVidas(Notification notification)
{
    int vidasAIncrementar = (int)notification.data;
    vidas += vidasAIncrementar;
    //Debug.Log("Vidas: " + vidas);
    notificationCenter.PostNotification(this, "totalVidas", vidas);
    totalVidas.text = "Vidas: " + vidas;
}

public void RestarVidas(Notification notification)
{
    int vidasANegar = (int)notification.data;
    vidas -= vidasANegar;
    notificationCenter.PostNotification(this, "totalVidas", vidas);
    notificationCenter.PostNotification(this, "totalVidasRestadas", vidas);
    totalVidas.text = "Vidas: " + vidas;
}

public int GetPuntuacion()
{
    return puntuacion;
}

// Update is called once per frame
void update()
{
}
}
```

Y además, en el script ValoresGameOver debemos definir una variable pública del objeto Puntuación y asignar a dicha variable la cámara principal.



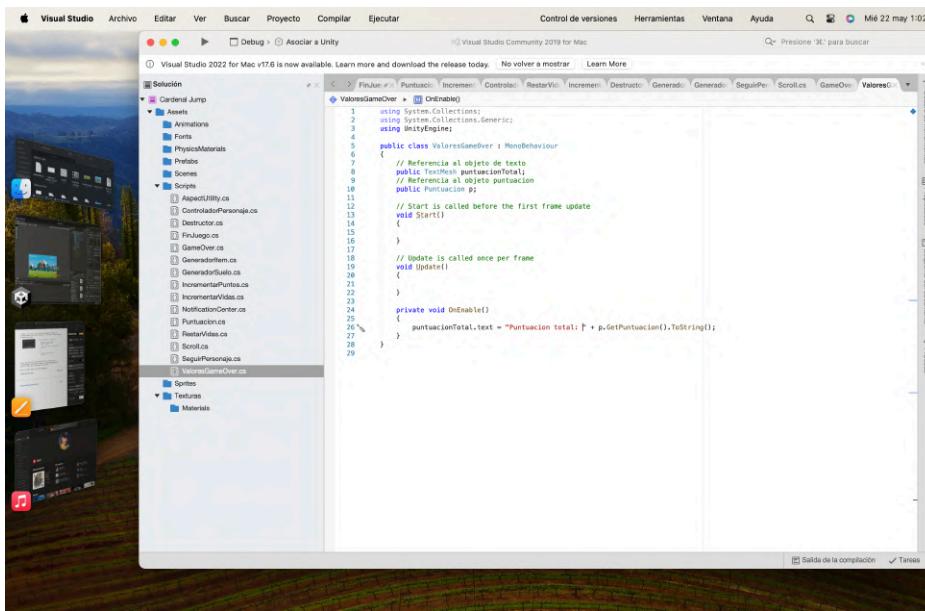
```
public class ValoresGameOver : MonoBehaviour
{
    // Referencia al objeto de texto
    public TextMesh puntuacionTotal;
    public Puntuacion p;

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

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

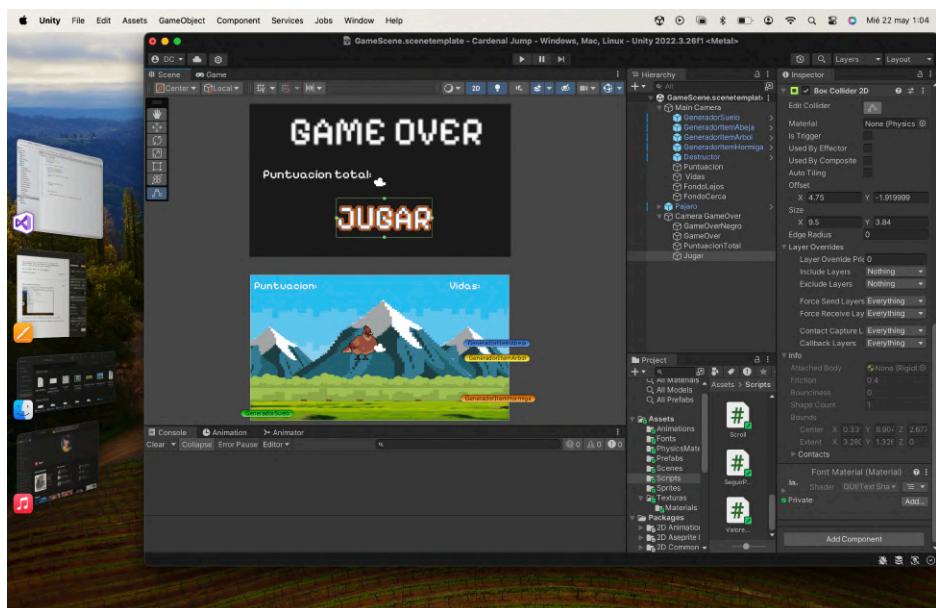
Ahora, para mostrar la puntuación total en el text mesh correspondiente necesitamos llamar al método GetPuntuacion, pasarlo a ToString y asignarlo al text mesh.

Esto lo haremos dentro de evento onEnable.



El siguiente paso será convertir el texto jugar en un botón.

Para ello lo primero que haremos será añadirle un BoxCollider2D al texto.

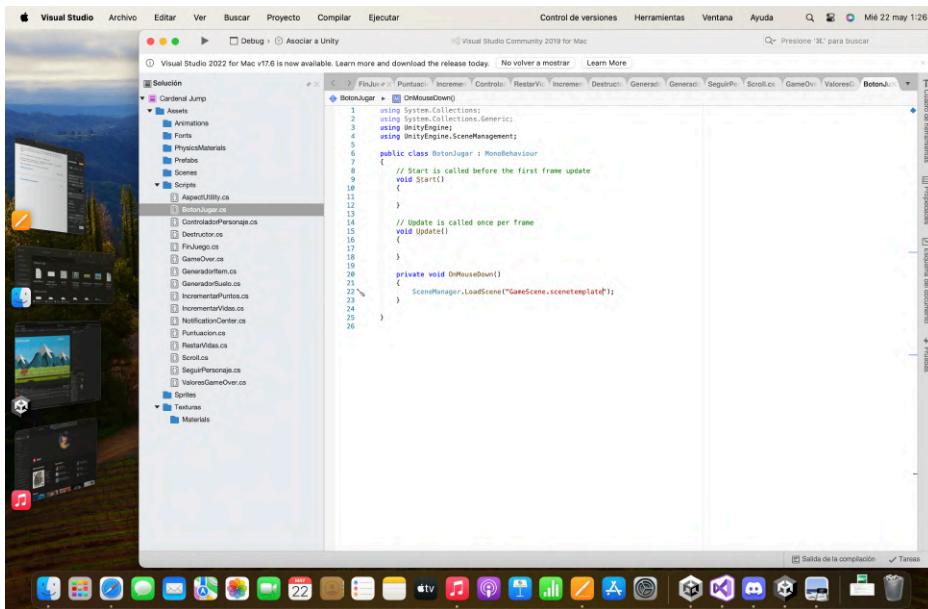


Tras ello crearemos un script llamado BotonJugar que contendrá la lógica para que cuando pulsemos el botón de jugar nos reincide el juego.

Esto lo haremos dentro del evento OnMouseDown.

En dicho método llamaremos a SceneManager.LoadScene y le pasaremos la escena que hemos creado de juego.

Ahora asignaremos dicho script al text mesh.



Lo siguiente que haremos será crear la pantalla de portada.

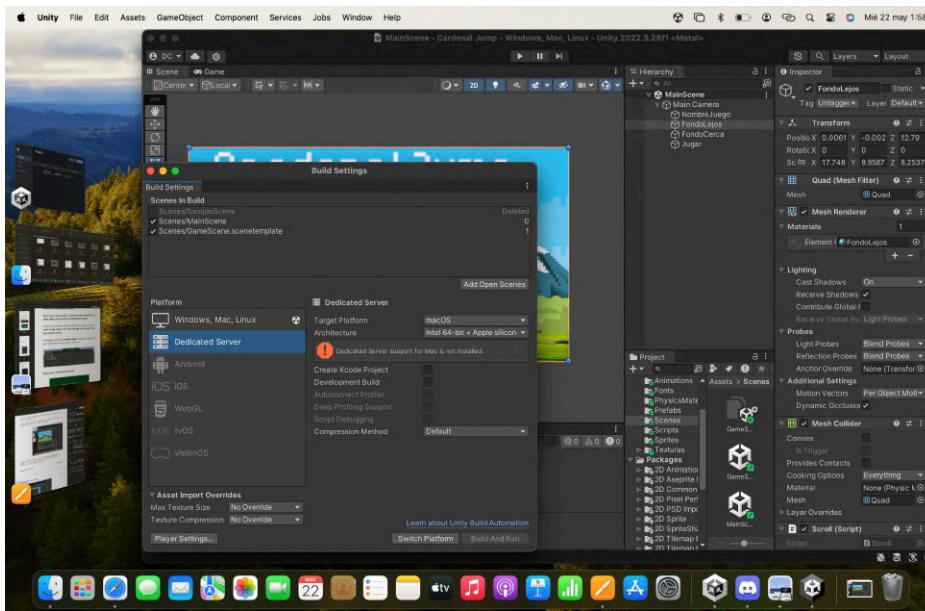
Para ello duplicamos la escena, a la que llamaremos MainScene y eliminamos de ella todos los objetos menos los fondos y el marcador.

Tras ello iremos a GameScene y copiaremos el botón jugar para pegarlo en esta escena.

El text mesh de marcador le cambiaremos el nombre y el texto y lo usaremos para contener el nombre del juego.



Tras ello iremos a File ==> build ==> setting y añadiremos en Scenes in build las dos escenas, poniendo primero MainScene (numero 0) y luego GameScene (numero 1).



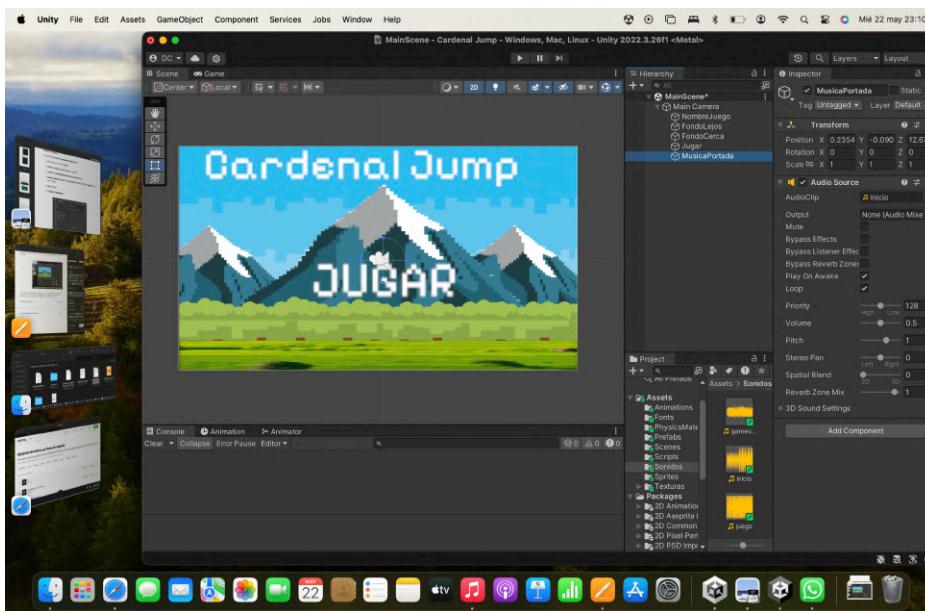
Lo siguiente que haremos será introducir la musica y sonidos del juego.

Lo primero que haremos será crear la carpeta Sonidos en assets.

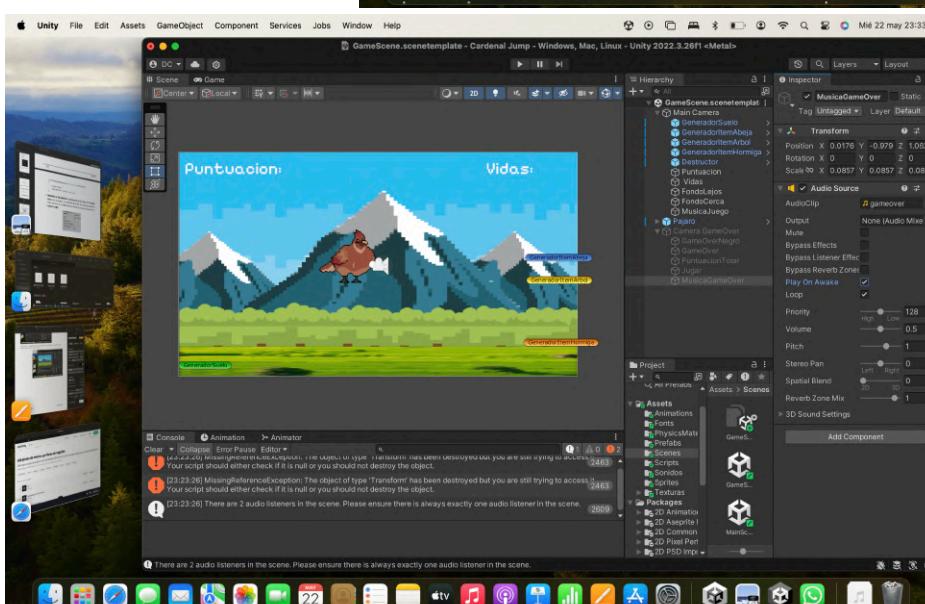
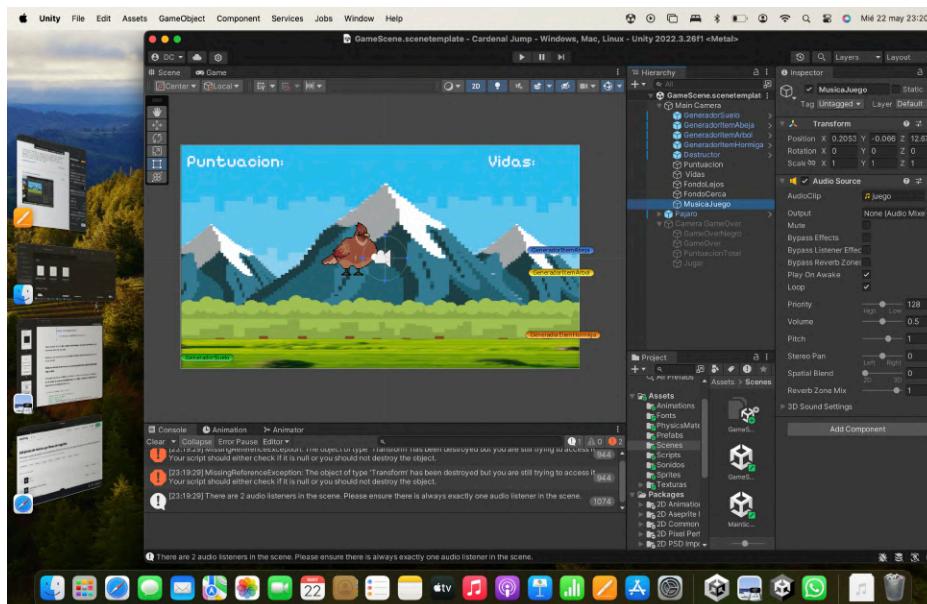
Tras ello vamos a incorporar en ella los tres primeros sonidos, uno para la escena de inicio llamado inicio, otro para la escena de juego llamado juego y otro para la escena de gameover llamado gameover.



A continuación abrimos la escena de juego y añadimos un componente audio source a la cámara, ajustamos los parámetros para que se reproduzca al inicio (Play On Awake) y en bucle (loop) y le añadimos el clip de sonido que va a sonar durante el tiempo que este esta escena, el cual será indio.

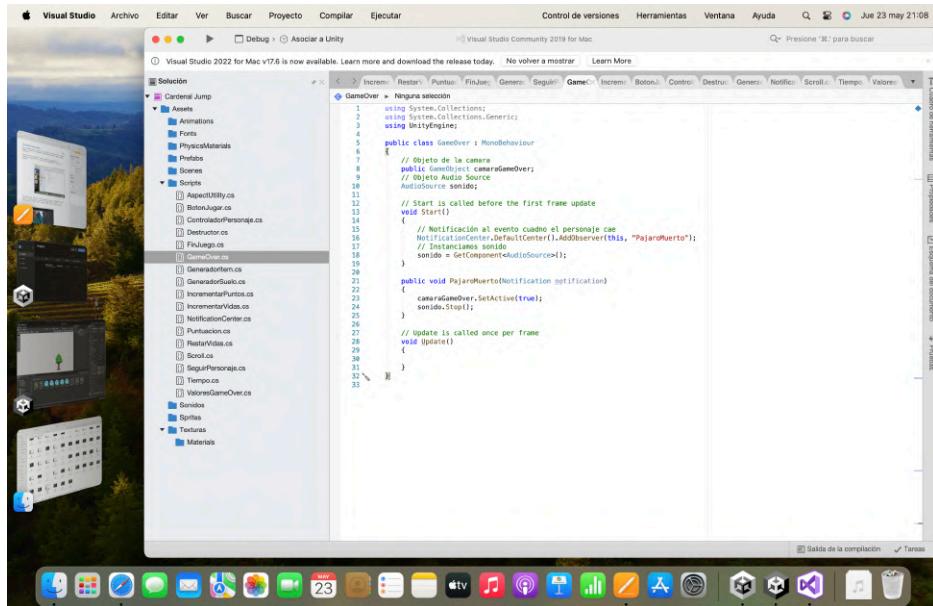


Seguiremos los mismos pasos para añadir la musica a la escena de juego, y a la cámara gameover pero con sus clips pertinentes.

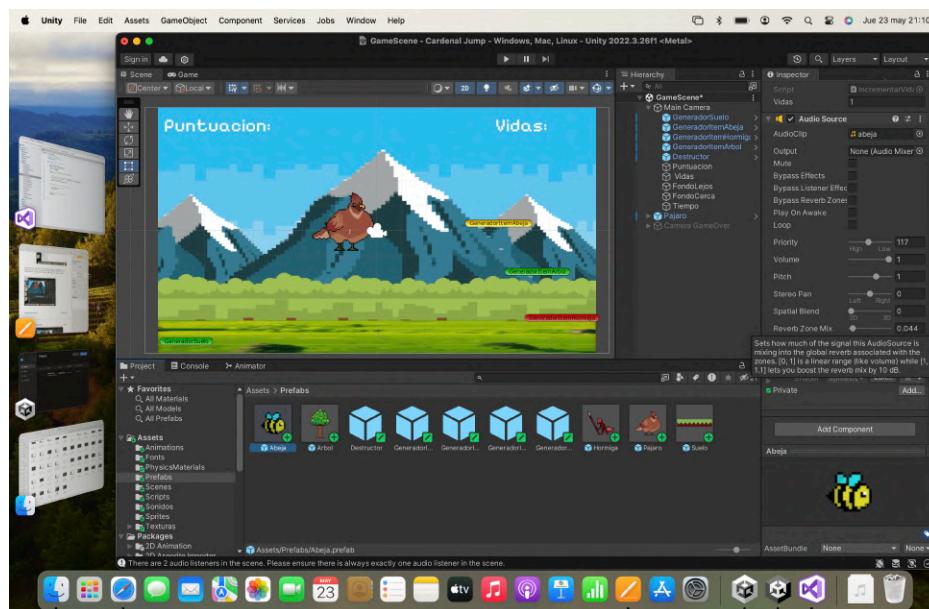


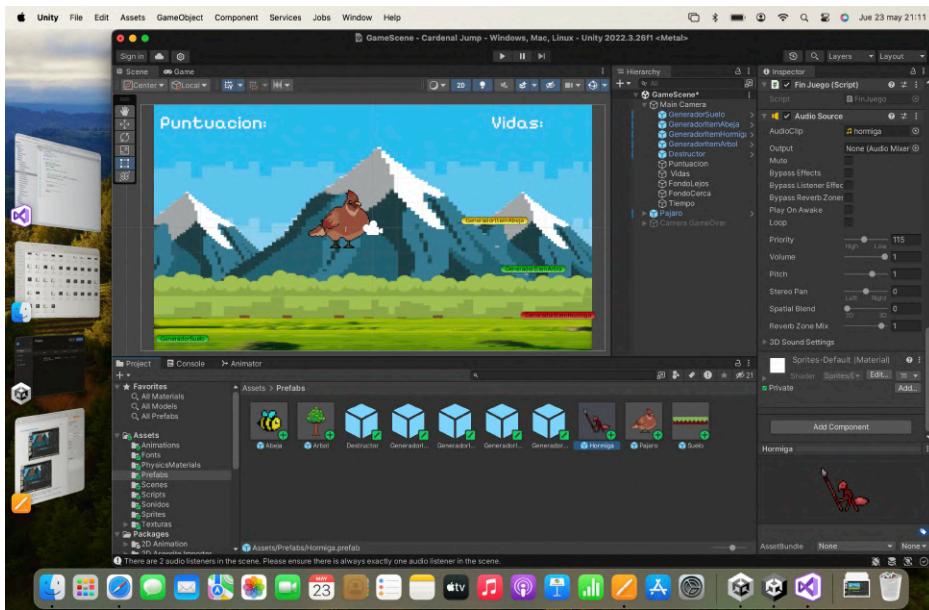
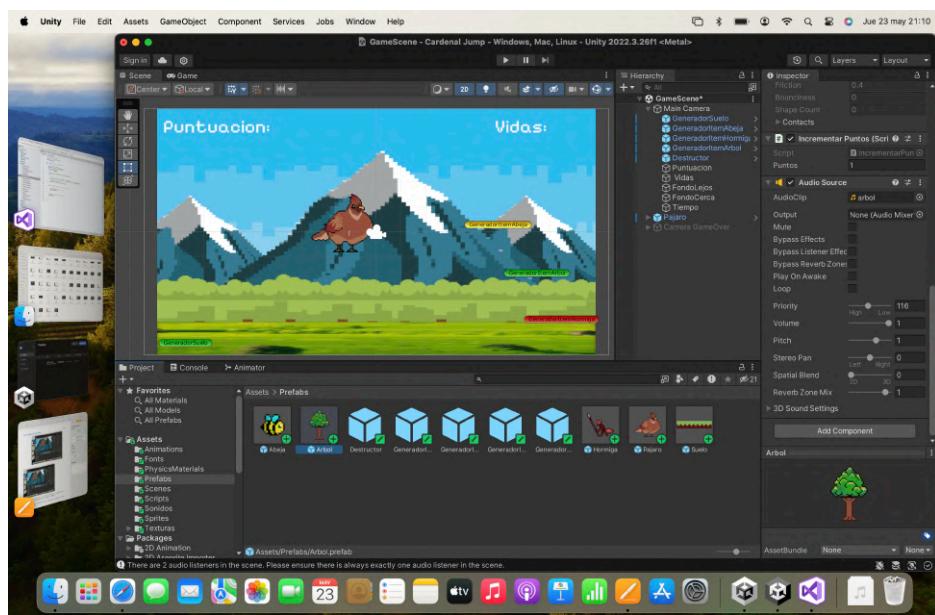
Para conseguir que la musica de la escena de juego se pare y pueda empezar a sonar la de gameover necesitamos parar el sonido desde el código.

Para ello nos dirigimos al script gameover, accedemos al componente audio source y en el método PajaroMuerto paramos el sonido.



Para añadir los sonidos a los item añadiremos un componente audio source a cada uno de los prefab (hormiga, árbol y abeja) y se asignaremos el sonido con el mismo nombre a cada uno de ellos.



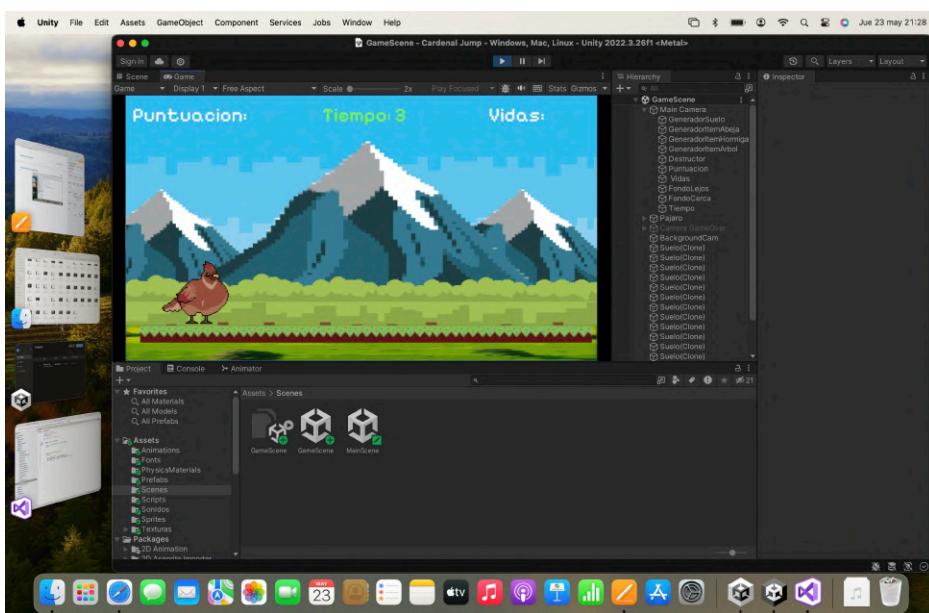
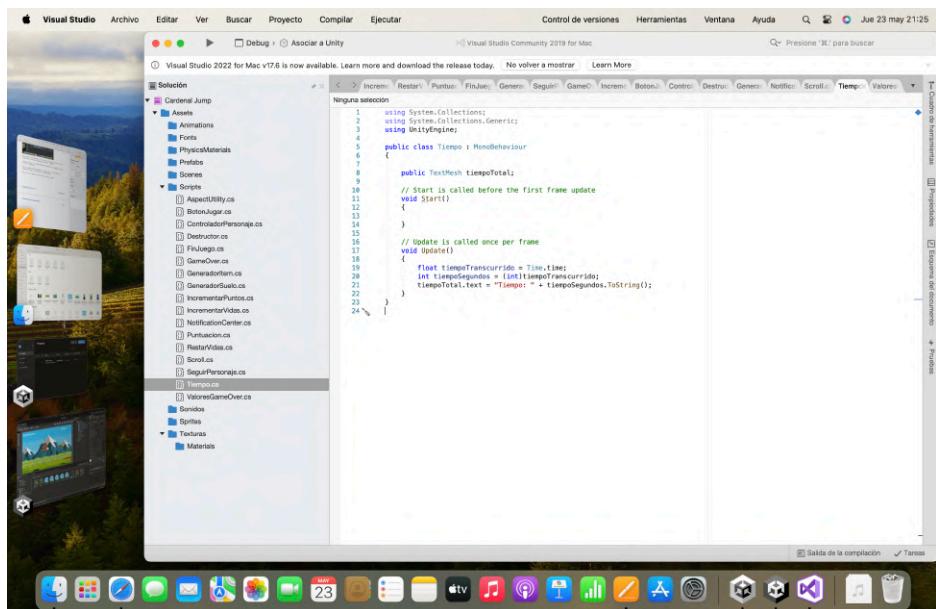


Como mejora a lo expuesto en el juego del mapache he añadido el tiempo que transcurre desde que se abre el juego.

La idea inicial era contabilizar el tiempo de cada partida, pero no lo he conseguido.

Esto lo hemos hecho añadiendo un text mesh a la cámara de la escena de juego, al que hemos llamado Tiempo.

Tras ello he creado un script con el mismo nombre en la cual he creado un variable text mesh y en el método update he obtenido el tiempo transcurrida para luego enviarlo a la variable text mesh mediante el método set text.



Y con esto damos por finalizada la tarea.