Rubio Lucas Iván Omar

Matricula: 2183061692

Licenciatura en Computación

Guía para hacer un videojuego.

Introducción

En el ámbito universitario, los estudiantes se enfrentan a un abanico de materias que abarcan desde la ingeniería de software y programación concurrente hasta la inteligencia artificial y las bases de datos. Sin embargo, a menudo se encuentran con la ausencia de cursos dedicados a una de las disciplinas más apasionantes y demandadas en la actualidad: el desarrollo de videojuegos. Es por esta carencia que surge esta guía, como una solución a esta problemática pendiente.

El mundo de los videojuegos es vasto y diverso, y esta guía tiene como objetivo brindar a los estudiantes y entusiastas la oportunidad de adentrarse en él. En sus páginas, exploraremos el proceso de creación de juegos en Unity, abarcando aspectos esenciales como animaciones, movimientos de personajes, interacciones entre personajes y obstáculos, seguimiento de la cámara al jugador, técnicas de optimización de animaciones y programación de elementos clave, como contadores en pantalla, temporizadores y scripts.

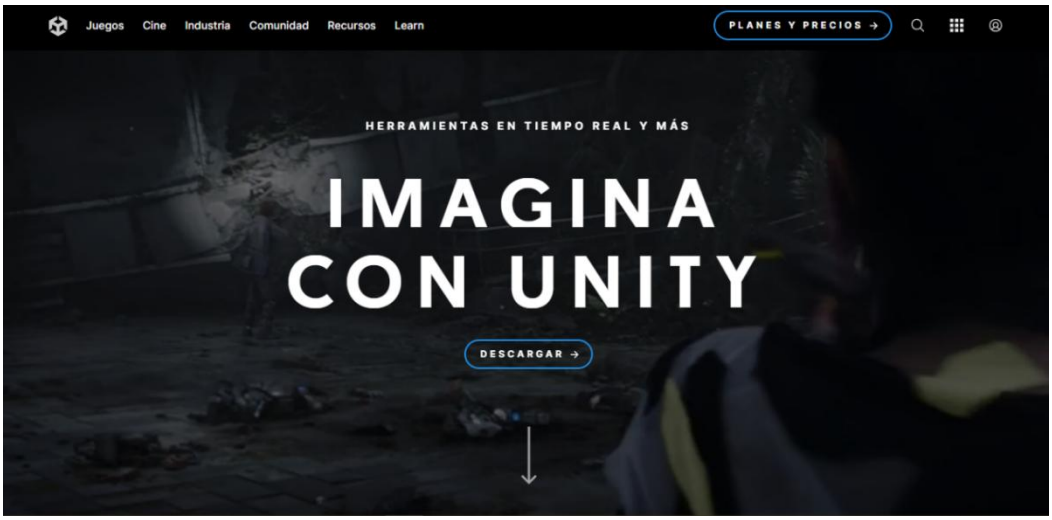
A través de esta guía, te introduciremos en el emocionante mundo de la creación de videojuegos, utilizando un proyecto concreto llamado 'UAMI Carrera', un juego apto para todo tipo de público y en formato 2D. Aquí, encontrarás información valiosa que te será especialmente útil si eres principiante en el uso de Unity o si careces de experiencia previa en el desarrollo de videojuegos.

Es importante destacar que las animaciones e imágenes utilizadas en este proyecto provienen de la guía para crear pixel art, las cuales están disponibles en el repositorio de Git de este juego. Así que, sin más demora, ¡comencemos nuestro viaje en el emocionante mundo de la creación de videojuegos con Unity!

Índice

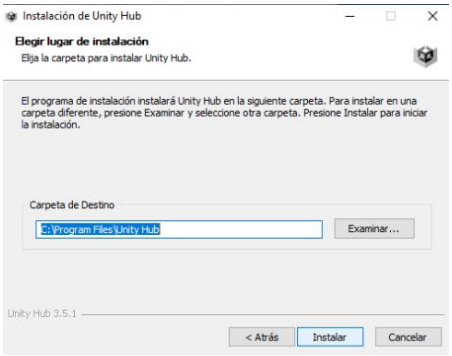
Descargar Unity Hub y Visual Studio

Para descargar Unity entramos a https://unity.com/es o con tan solo escribir Unity en el navegador saldrá la página oficial con el botón descargar justo enfrente:

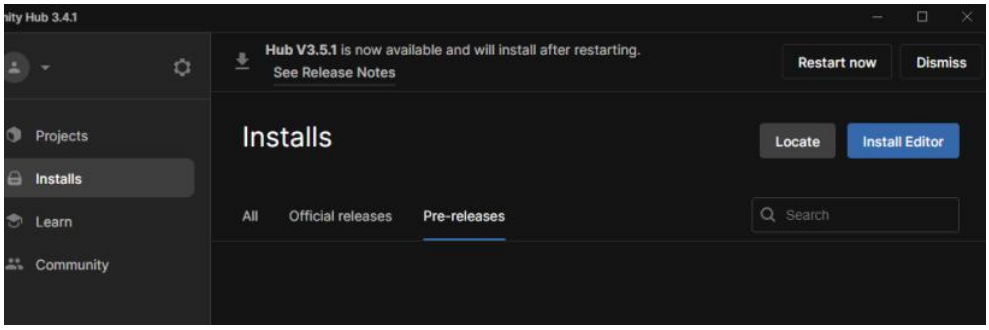


Enseguida nos saldrán las opciones y en azul es la recomendada para la computadora:

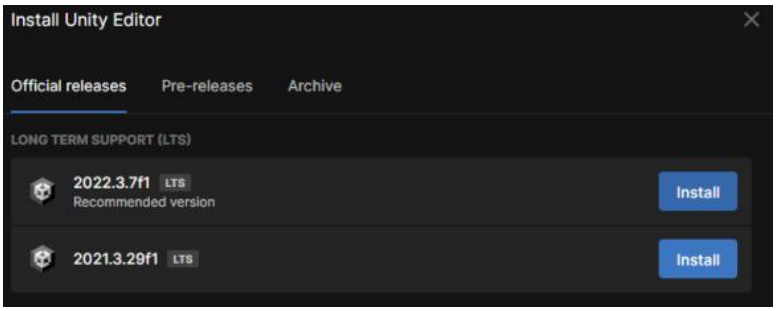
Al abrir el instalador, aceptamos permisos y presionamos instalar:



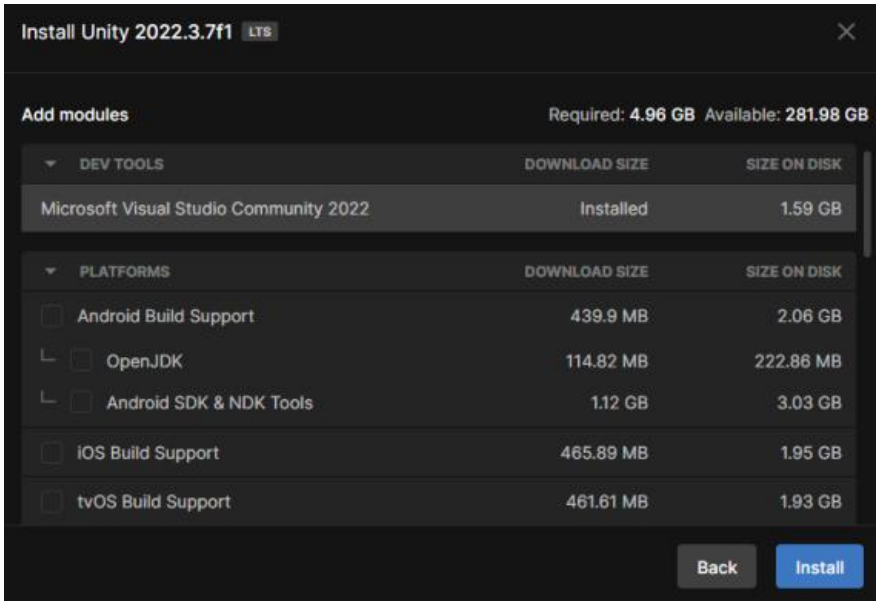
Al abrir Unity Hub vamos al lado izquierdo donde dice Installs y presionamos el botón azul que dice Install Editor:



Seleccionamos una versión LTS, que son versiones que funcionan bien a largo plazo, y la instalamos:

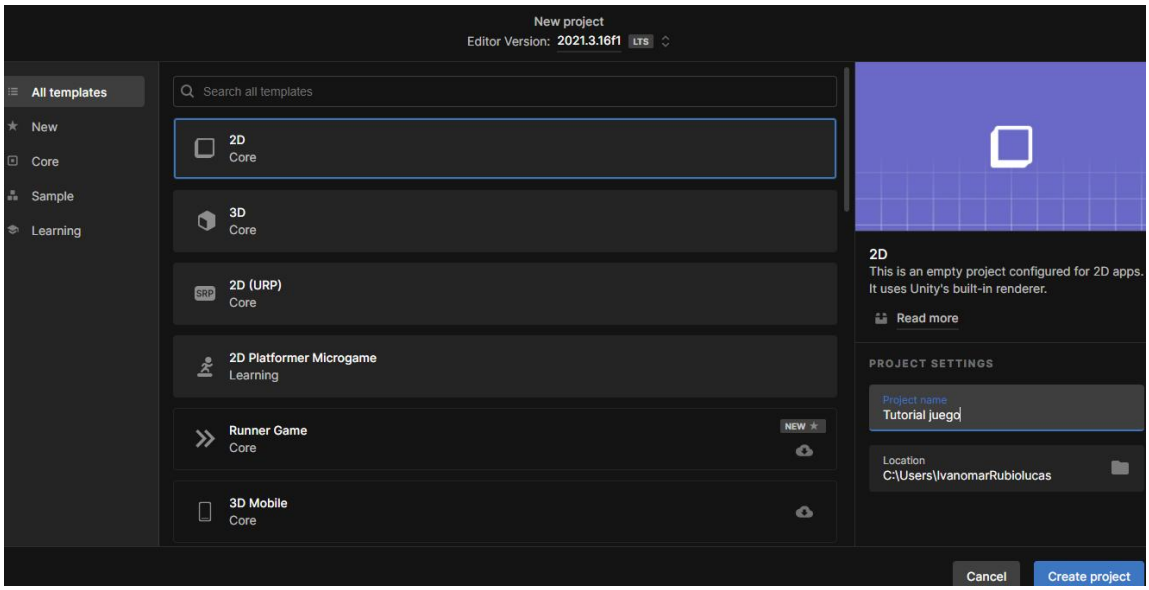


Para terminar con la instalación nos aparecerá marcada la opción de instalar Visual Studio y presionan instalar. Tambien se puede utilizar visual studio code que es más ligero.

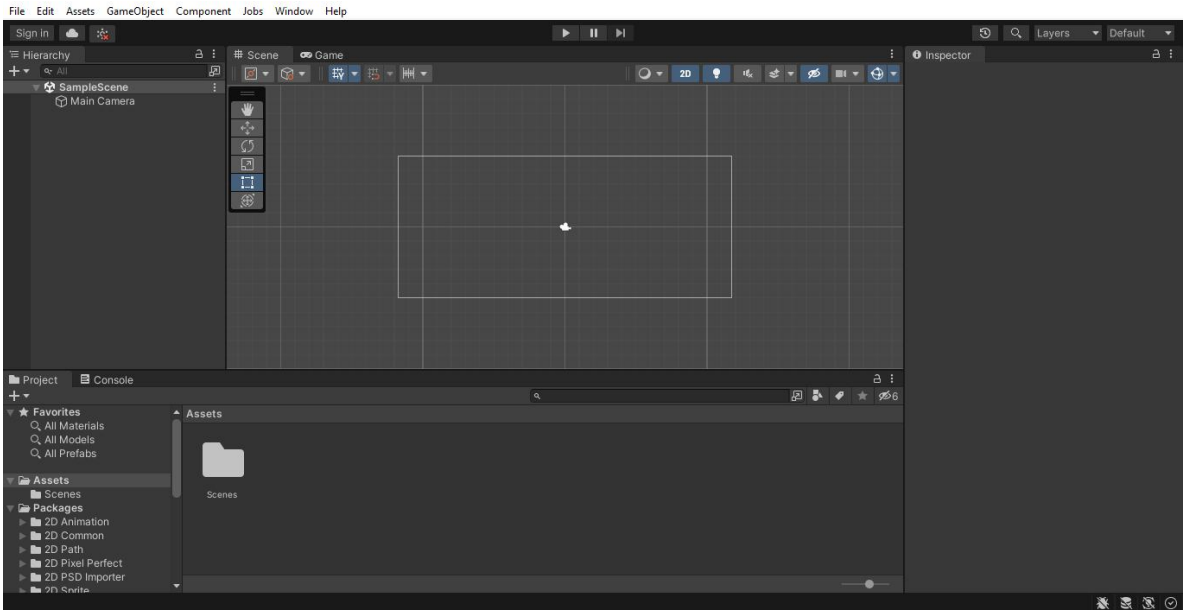


Crear el proyecto

En Unity, lo primero es hacer un nuevo proyecto 2D y ponerle un nombre y también podemos elegir la ruta donde guardarlo:



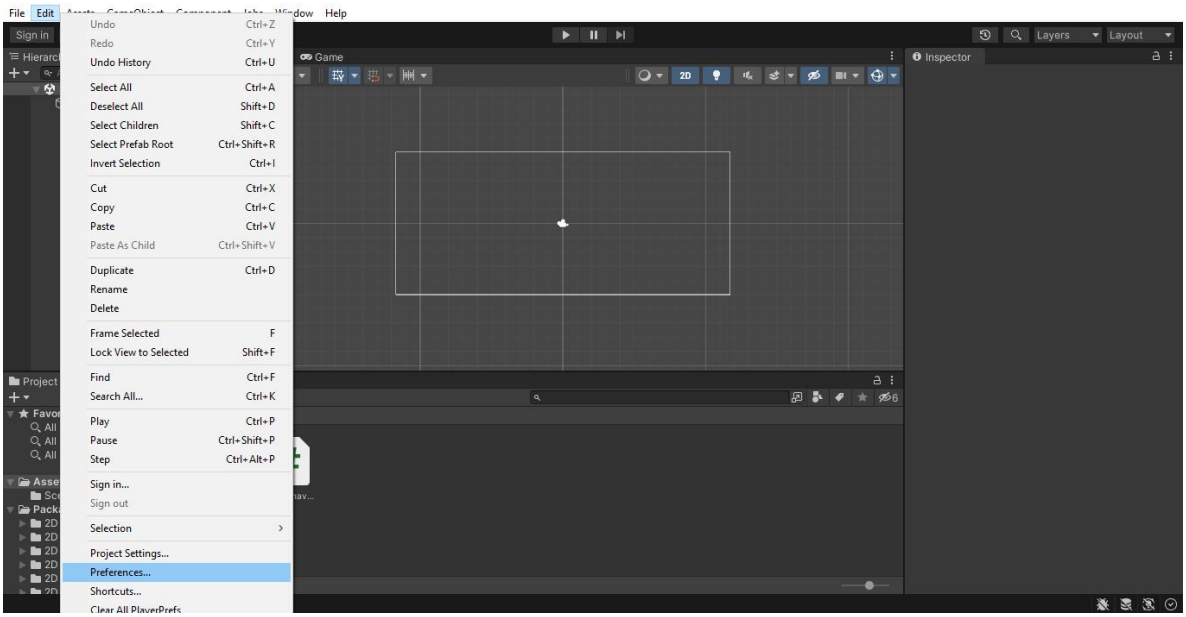
Podemos ver a la izquierda los objetos y su jerarquía (Hierarchy), en la parte de abajo vemos las cosas que podemos usar divididas en carpetas (Assets), en el centro tenemos la escena (Scene) y el juego (Game), a la derecha podemos ver las características de lo que seleccionamos (Inspector), y arriba tiene botones para correr el juego o dejar de correrlo, las ventanas pueden moverse al lugar que se te haga más cómodo usarlas.



Cambiar color cuando corra el juego

Es muy útil y recomendable cambiar el color cuando corramos el juego y para quitarlo cuando se hagan cambios, pues todos los cambios que realicemos no se guardarán si se está corriendo el juego. Este paso es importante para no perder el progreso que llevemos, que es muy común.

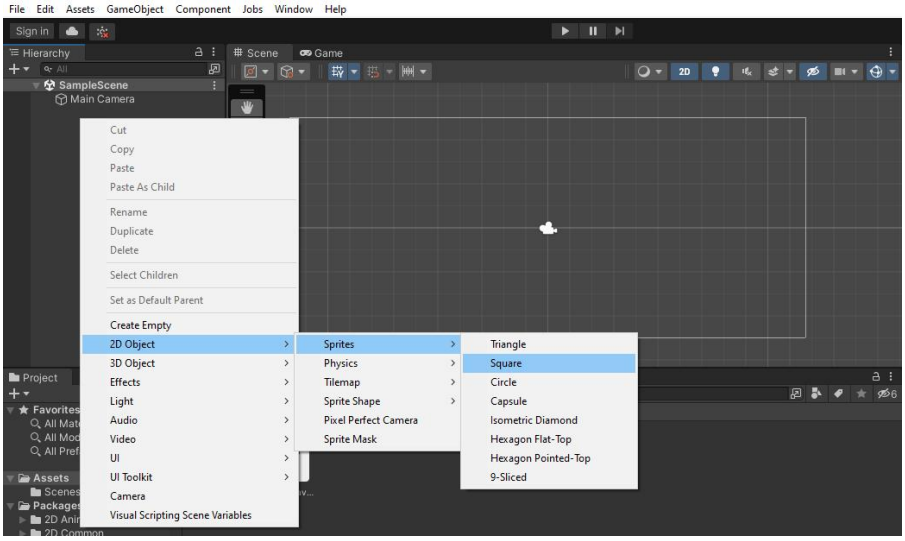
Vamos a Edit y a Preferences:



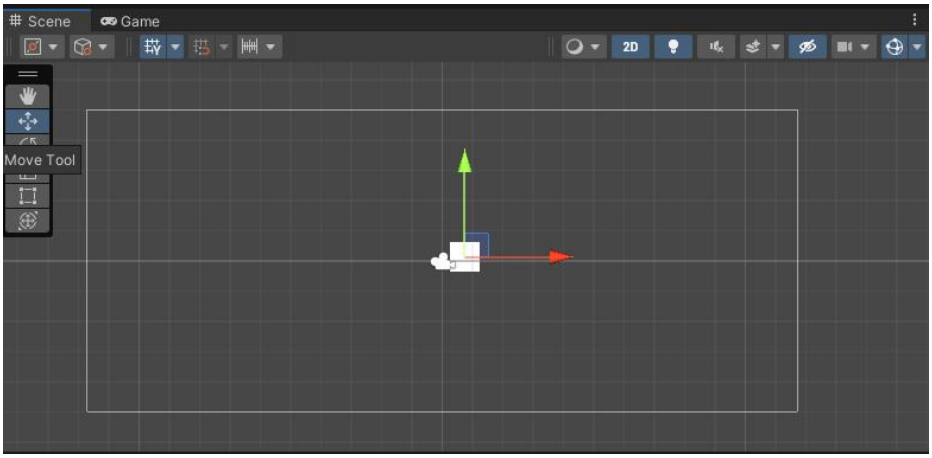
Presionamos Colors del lado izquierdo y en Playmode tint elegimos un color reconocible, en este caso verde es una buena opción.

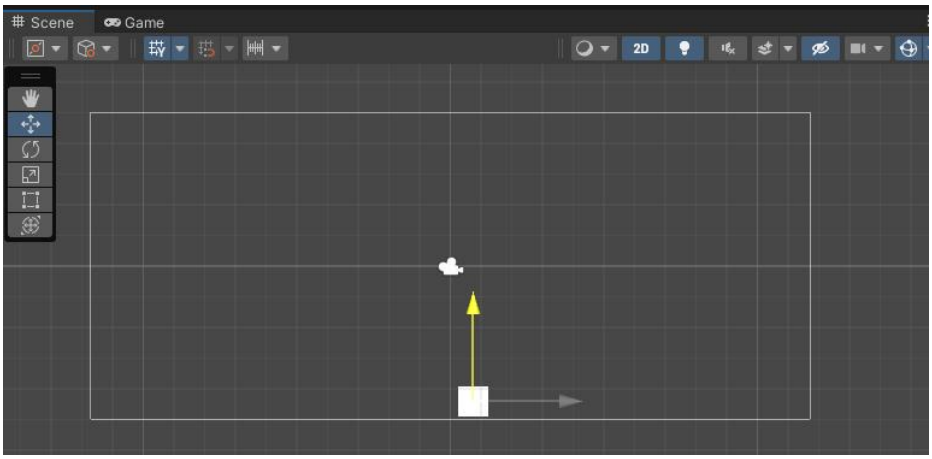
Crear personaje y suelo

Para agregar el suelo: En Hierarchy presionamos clic derecho, 2D Object, Sprites, Square y finalmente le ponemos nombre al objeto que creamos.

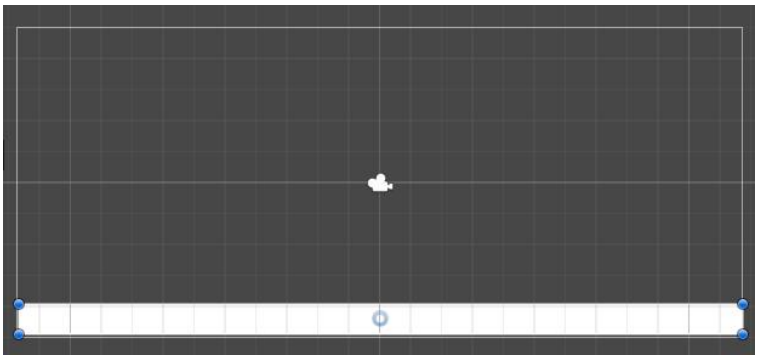


Elegimos la opción Move Tool y movemos el cuadrado a la parte de abajo.

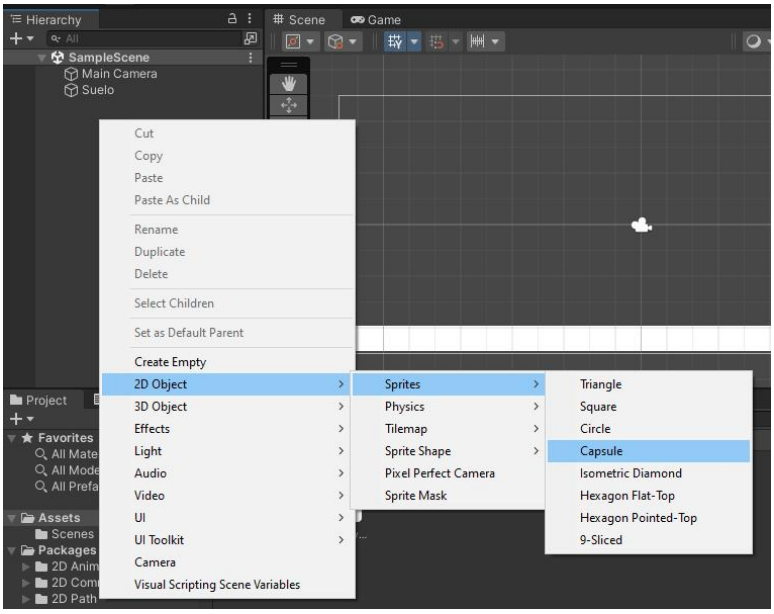




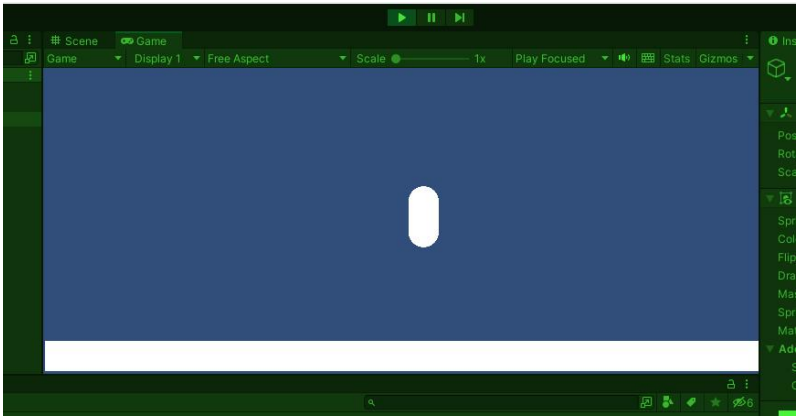
Ahora con la opción Rect Tool estiramos el cuadrado:



Para agregar al personaje vamos a usar una capsula: presionando clic derecho en Hierarchy, 2D Object, Sprites y Capsule, también agregándole el nombre de personaje:



Si corremos el juego vemos que no se mueven, son estáticos, entonces tenemos que agregarles físicas para que tenga gravedad y colisiones.



1. Seleccionar al suelo y en el inspector presionamos Add Component, Physics 2D, Box Collider.
2. Seleccionar al personaje y en el inspector presionamos Add Component, Physics 2D, Capsule Collider.
3. Seleccionar al personaje y en el inspector presionamos Add Component, Physics 2D, Rigidbody 2D.

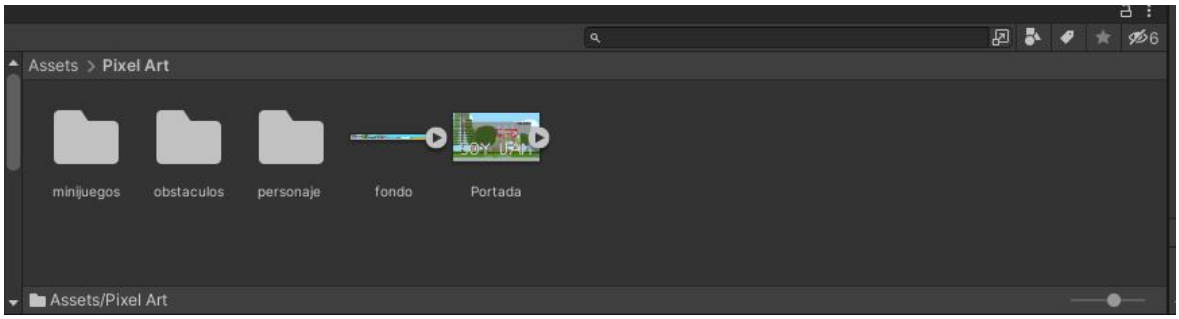
El Collider hace que los objetos que tiene Collider no se traspasen, mientras que el Rigidbody hace que el personaje tenga físicas, como la gravedad.

Con esto podremos trabajar con colores, animaciones y un fondo llamativo para que sea más motivador trabajar.

Agregando Pixel Art

Podemos presionar clic derecho en Assets y crear un folder para organizar los Sprites que tengamos o arrastrar una carpeta que ya tengamos con las imágenes que utilizaremos.

En este caso usamos los archivos de la guía para hacer pixel art o se pueden descargar unos de https://itch.io/ donde algunos son gratuitos (recuerda que hay derechos de autor al utilizar assets de otras personas y que la guía para pixel Art te ayuda a evitar temas de autoría).

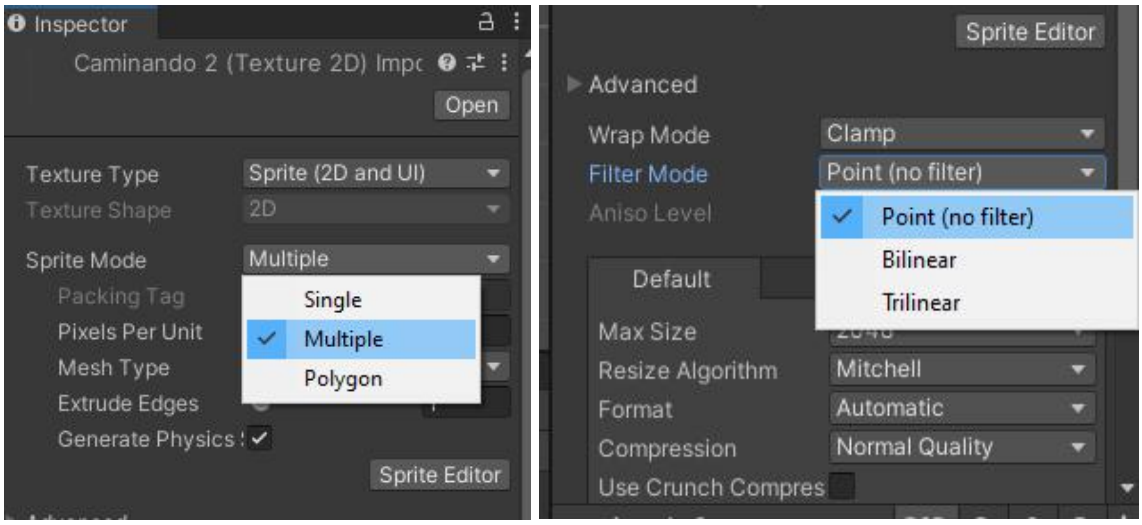


Ahora notemos que los SpriteSheet, al presionar en la fecha a la derecha, tienen una imagen de la animación y debemos separarlas, en el caso del personaje caminando:

Para separar los archivos tipo Spritesheet, lo seleccionamos y en el inspector cambiamos:

1) Sprite Mode de Single a Multiple.
2) En el Filter Mode cambiamos a Point (no filter).

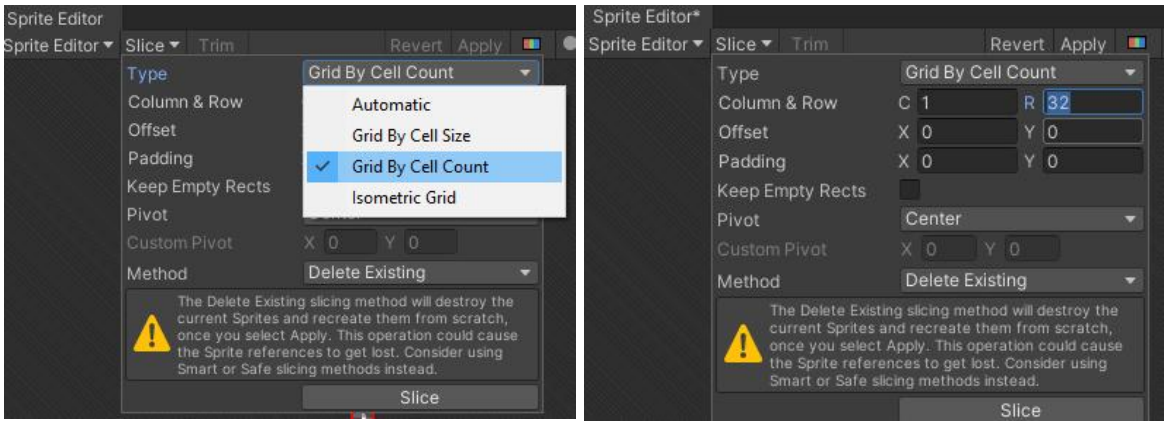
Aplicamos los cambios, en la parte de abajo, y luego presionamos en Sprite editor:



(Nota: Si la imagen se ve borrosa, se soluciona cambiando la Compression de Normal Quality a None.)

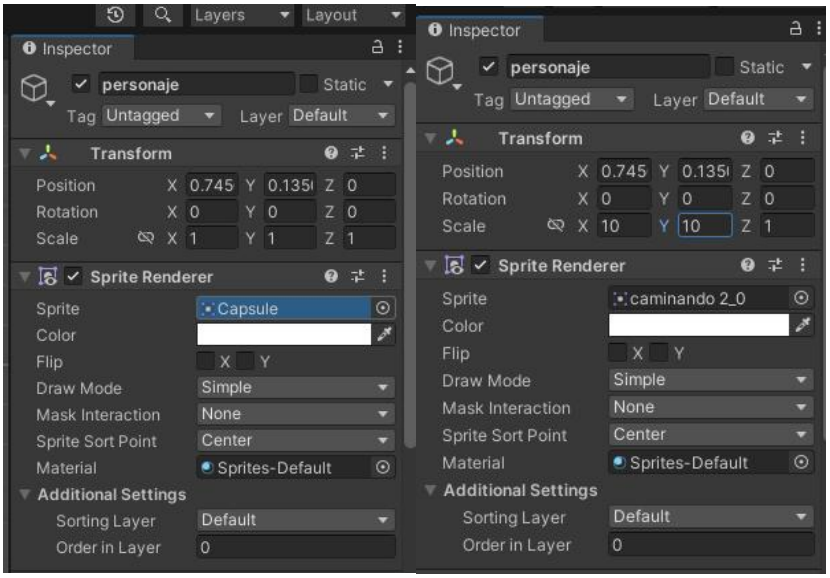
Nos saldrá una ventana, presionamos Slice y en Type cambiamos a Grid By Cell Count, así podemos separarlas por celdas y columnas, en este caso 1 columna y 32 renglones.

Al presionar el botón Slice se pueden ver las imágenes separadas y presionamos Apply para guardar los cambios.

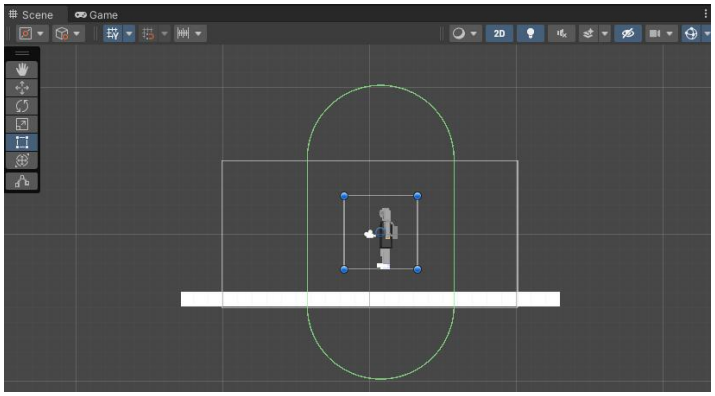


Ahora vemos las imágenes separadas frame por frame:

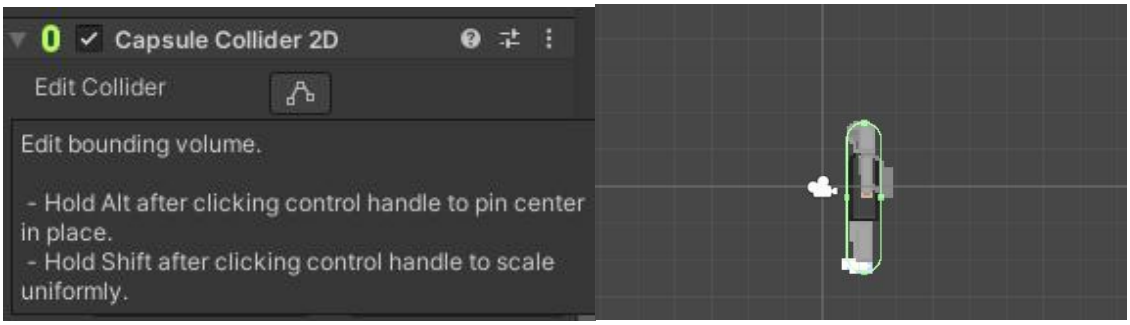
En el inspector, al seleccionar al personaje al que queremos agregarle la imagen, en donde dice Sprite le cambiamos la imagen de capsule a el primer sprite de caminando (Podemos arrastrarla desde los Assets).



Ahora en lugar de ver la capsula vemos al personaje y su collider al rededor:

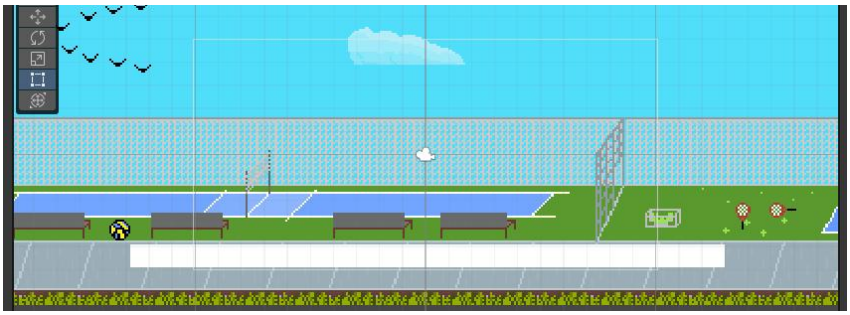


Notamos que ahora el Collider es demasiado grande, para solucionarlo, vamos a la parte del inspector, en capsule Collider 2D presionamos el botón de Edit Collider y ajustamos su tamaño:



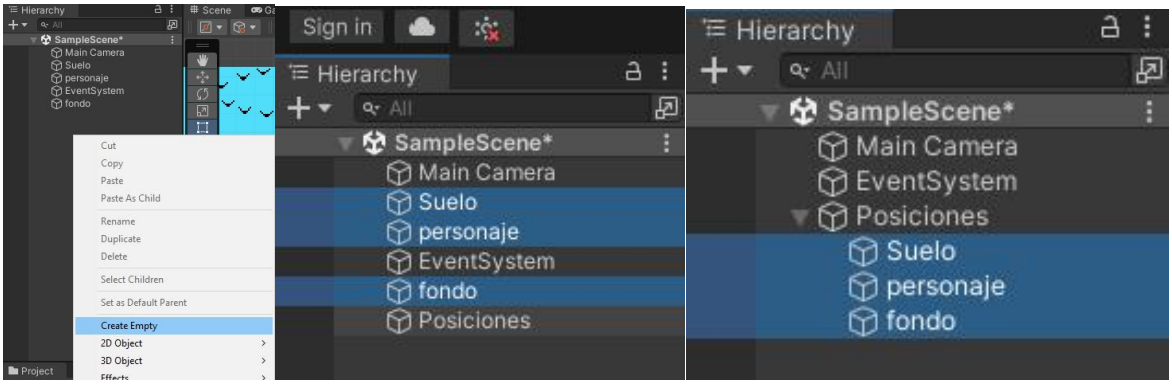
Ahora para agregar el fondo arrastramos la imagen que queramos sea el fondo, si es muy grande probablemente se vea borrosa, ajustamos el max size a 4096, pues la imagen es grande (Muy parecido a lo que se hace con los SpriteSheet, solo sin el paso del sprite editor).

Ahora notemos que el personaje no se ve, está detrás del fondo y enfrente del fondo está el suelo blanco. También empezamos a tener muchos objetos en Hierarchy, por lo que debemos organizarlos:



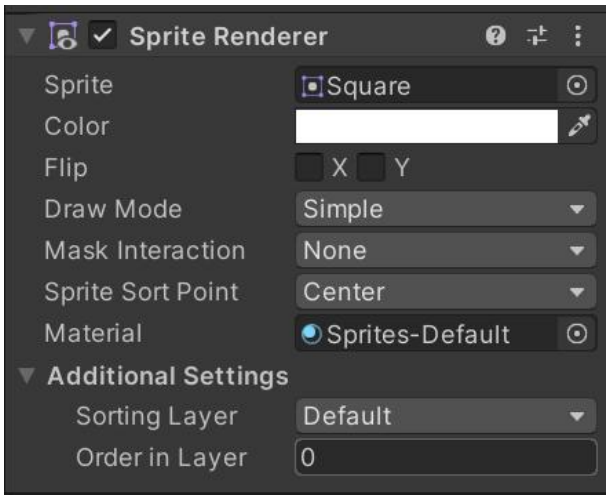
Podemos crear un canvas o un nuevo objeto vacío para organizarlos y saber que parte son los objetos en 2D (yo cree un nuevo objeto llamado posiciones).

Seleccionamos los objetos que queremos cambiar su fondo y los arrastramos al nuevo objeto:



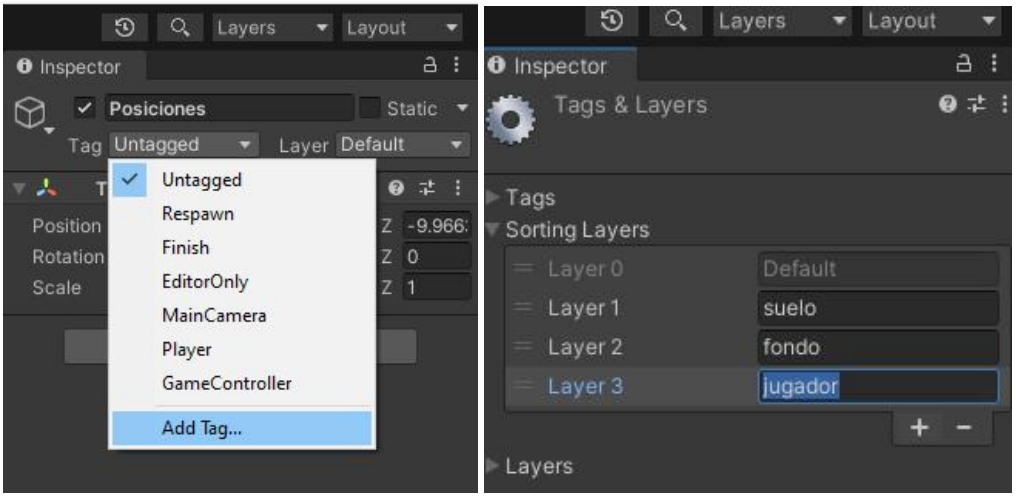
Existen 2 formas para ordenar los objetos por capas: La primera es para un juego corto, o mini juego donde hay pocos objetos y no se piensan agregar más objetos en un futuro.

En este caso basta con cambiar el numero de Order in Layer en el sprite Renderer, en la pestaña Additional Settings de todos los objetos. Con un numero más grande se verá más adelante el objeto.

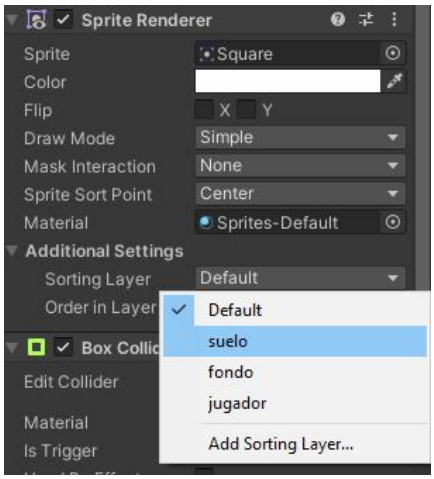


En el segundo caso es para un juego grande, con muchos objetos o que en un futuro se agregaran más objetos. Para esto usamos Sorting Layer:

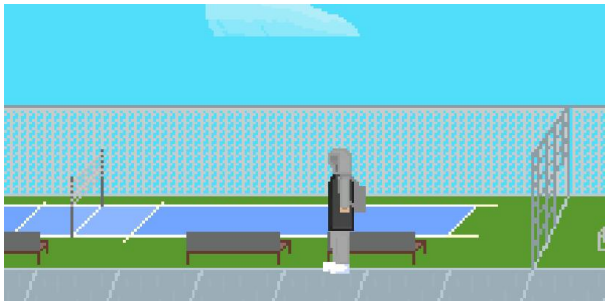
1) En el inspector, al seleccionar cualquier objeto, en Tag elegimos Add Tag.
2) En Sorting Layers agregamos el nombre de las capas, la capa de arriba es la que se colocará hasta la parte de atrás, así podemos organizar las capas que van más al fondo o más enfrente que otras.



3) A cada objeto, en Additional Settings ponemos la Sorting Layer correspondiente.

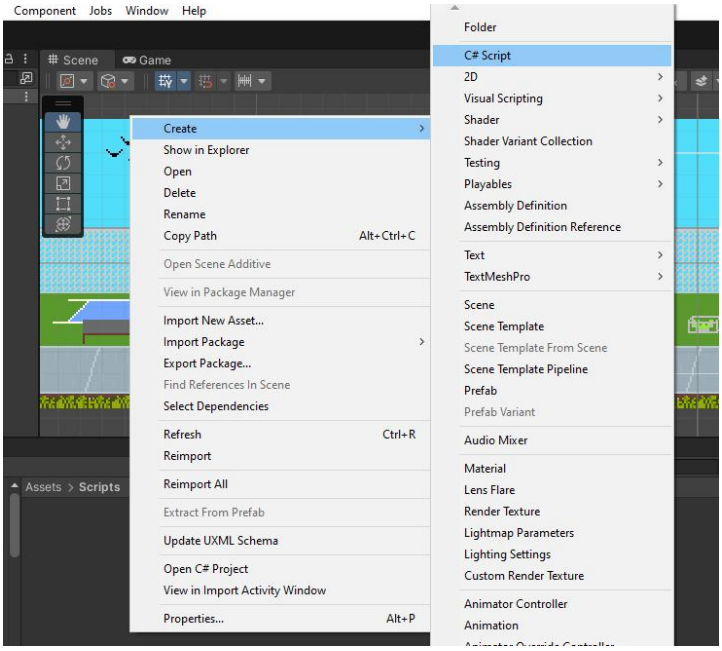


De esta forma, el suelo estará atrás del fondo, por lo que no se verá el rectángulo blanco que creamos hace rato y el personaje se verá enfrente del fondo:

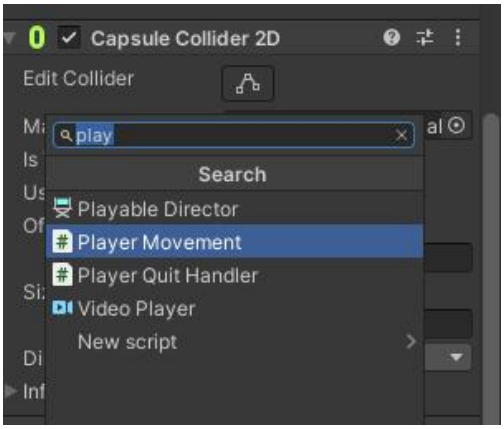


Movimientos y animaciones

Para que el personaje se mueva en una dirección: En Assets creamos una carpeta llama Scripts y dentro creamos un C# Script y su nombre:



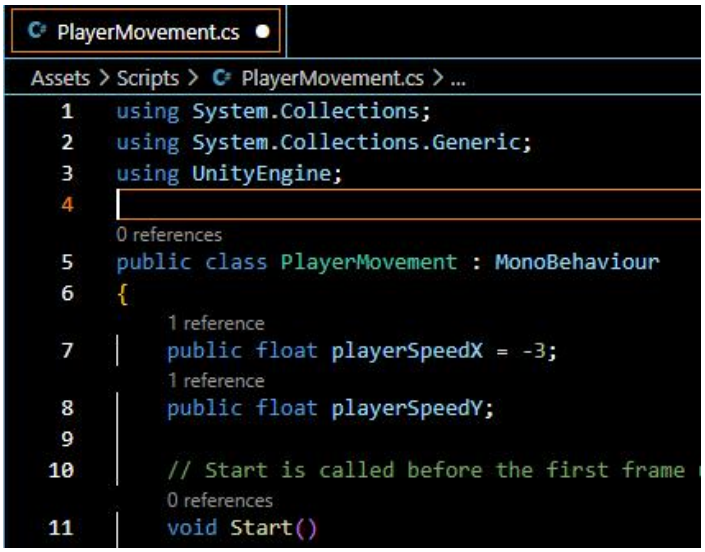
Al personaje en Add Component (en el inspector) hasta la parte de abajo, se puede agregar arrastrando el script o buscándolo por su nombre:



En los juegos 2D, la velocidad son 2 ejes, el eje X que es dirección, el eje Y que es altura y en juegos 3D se agrega el eje Z que es profundidad.

En Assets, abrimos el Script dándole doble clic y creamos las variables velocidad del jugador, como se va a mover a la izquierda ponemos velocidad negativa.

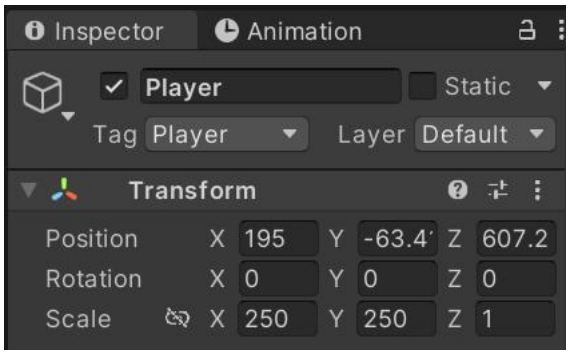
La posición y velocidad de los objetos son vectores de tipo Float. Al poner las variables globales de tipo public, se pueden modificar desde el inspector de unity, al probar el juego.



Por defecto, unity crea 2 métodos: el ‘start’ que se hace 1 sola vez al iniciar el juego, utilizado comúnmente para inicializar variables y métodos, y el método Update que se repite varias veces (las veces depende de la potencia de la computadora), con este método se miden los FPS (frames por segundo) de los juegos.

El update se repite más veces en una computadora potente, por lo que en una computadora de 60 FPS se moverá el doble que en una de 30 FPS. Para eso tenemos que multiplicar la velocidad por la función Time.deltaTime que hará que se muevan lo mismo en cualquier computadora.

Para que nuestro personaje se mueva, debemos actualizar su posición, en el inspector vemos que Position está dentro de Transform y los 3 ejes, aunque al ser un juego 2D solo se usan 2.

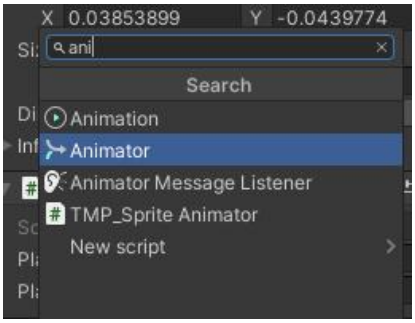


Para actualizar la posición usamos un vector de 2 dimensiones, y a las posiciones X y Y le sumamos su velocidad multiplicada por el delta time:

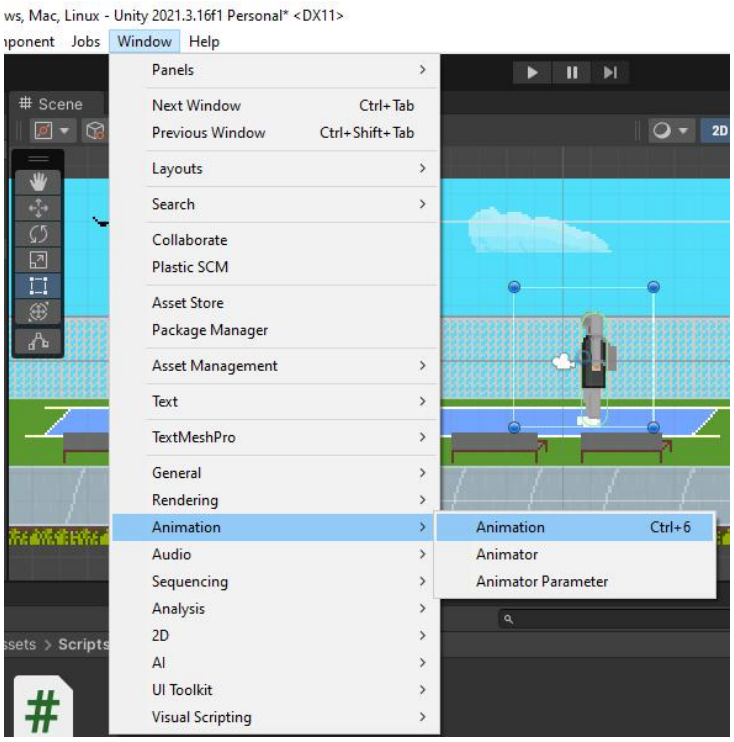
```
// Update is called once per frame
0 references
void Update()
{
    transform.position = new Vector2(transform.position.x + playerSpeedX * Time.deltaTime,
        transform.position.y + playerSpeedY * Time.deltaTime);
}
```

Si queremos que se mueva con solo presionar una tecla, por ejemplo la tecla A entonces se pone un if (Input.GetKeyDown(KeyCode.A)) y dentro del if poner el código anterior.

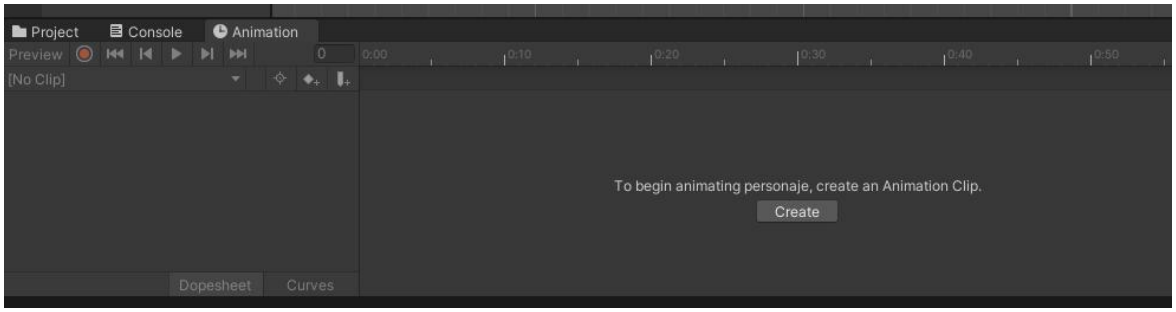
Ahora para agregarle la animación de caminar agregamos el componente Animator al personaje en el inspector:



En window, animation, animation podemos tener la nueva ventana para trabajar la animación:

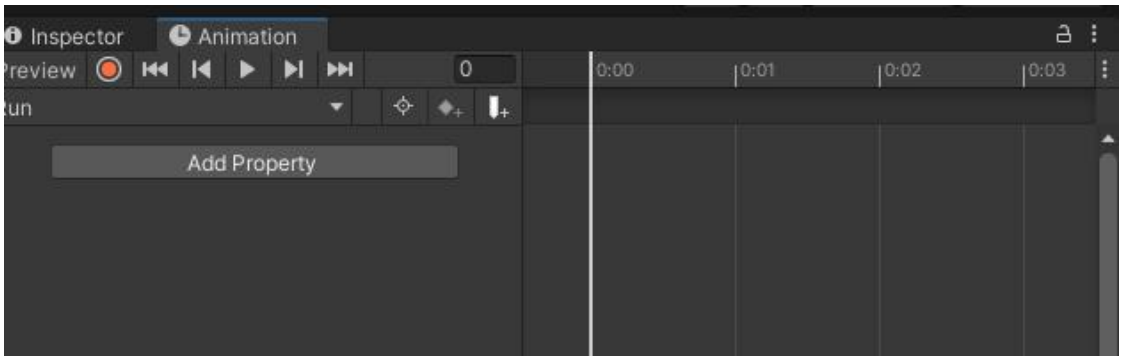


Presionamos Create.

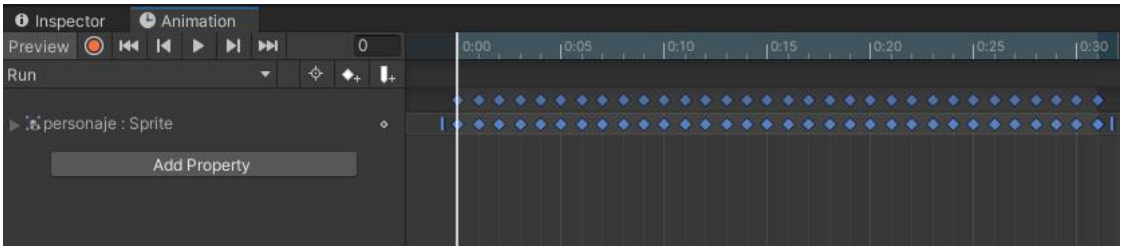


Nos saldrá una ventana que guarda el archivo justo donde tengamos la carpeta de los Assets en el proyecto (recuerda guardarlo en la carpeta correcta) y colocamos un nombre.

Nos muestra una nueva ventana donde debemos pasar las imágenes de la animación:



En la ventana tenemos el botón para correr la animación sin necesidad correr el juego y poder ir modificando el tiempo en que cambia una cada imagen:



Podremos crear las otras animaciones de la misma forma: agacharse, interactuar, sufrir daño, entre otras acciones.

Para poder saltar, donde usamos el Rigidbody para las físicas, el jumpforce será la fuerza del salto y para saber si está en el suelo o saltando la variable isGrounded.

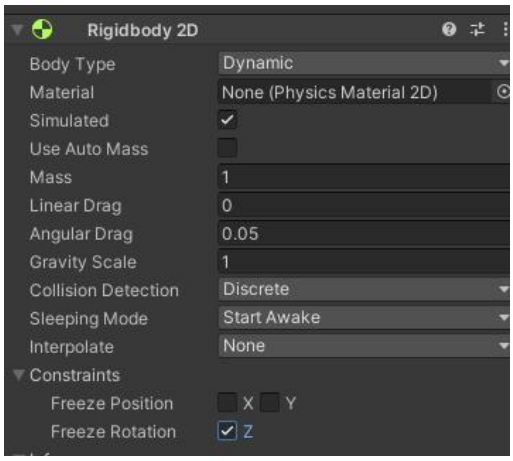
```
//Variables para saltar
3 references
private Rigidbody2D rb;
1 reference
public float jumpForce;
1 reference
private bool isGrounded = true;
1 reference
private bool isJumping = false;
```

```
// Start is called before the first fra
0 references
void Start()
{
    rb = GetComponent<Rigidbody2D>();
}
```

Y el código para saltar al presionar la tecla espacio:

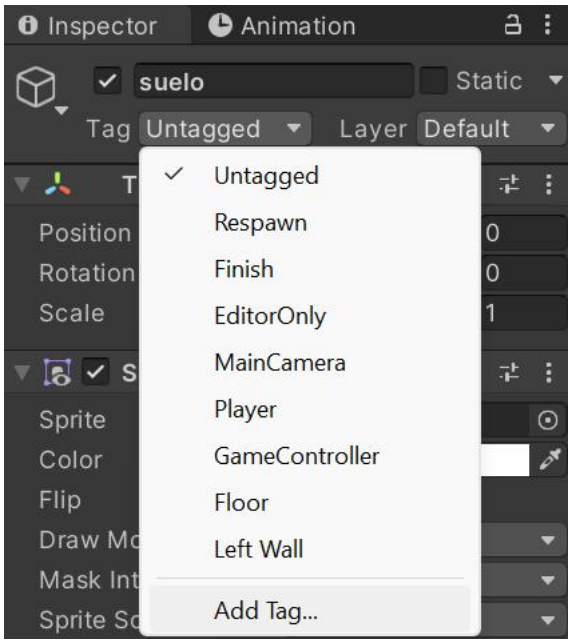
```
31      //salto
32      if (isGrounded && Input.GetKeyDown(KeyCode.Space)){
33          rb.velocity = new Vector2(rb.velocity.x,jumpForce);
34          isJumping = true;
35      }
```

Para que nuestro personaje no termine rotando, en el Inspector de nuestro personaje, en el Rigidbody está una pestaña Contraints y activamos Freeze rotation Z

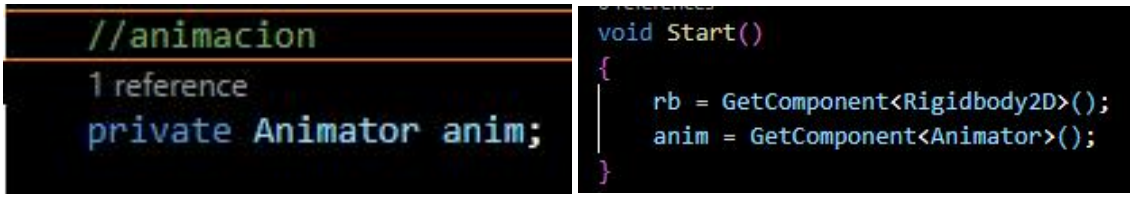


Para hacer que solo pueda saltar cuando toca el suelo, hay que detectar la colision del jugador un objeto y si el tag del objeto es el suelo ponerla en true, si no toca el suelo, ponerla en false.

Para agregar un tag solo presionamos add tag y agregamos el tag con su nombre.



Esto también nos sirve para la animación del salto donde tenemos que agregar el animator e inicializarlo:



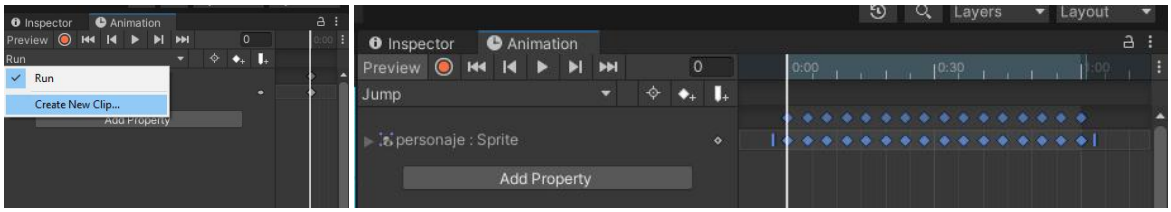
En el update agregamos el código para la animación que se hace cuando la variable bool isJumping es igual a true:

```
//Animador
anim.SetBool("IsJumping",isJumping);
```

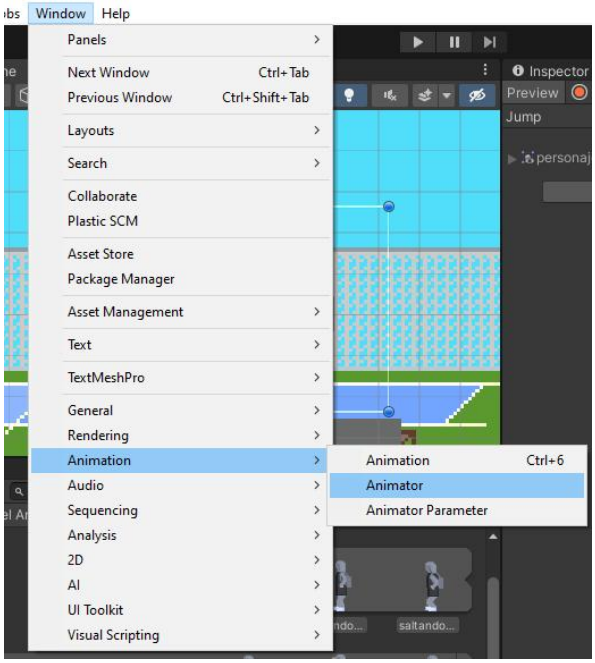
Para detener la animación de saltar cuando toque el suelo:

```
private void OnCollisionEnter2D(Collision2D collision)
{
    if (collision.gameObject.CompareTag("Floor"))
    {
        isJumping = false; // Si el personaje colisiona
        isGrounded = true;
    }else{
        isGrounded=false;
    }
}
```

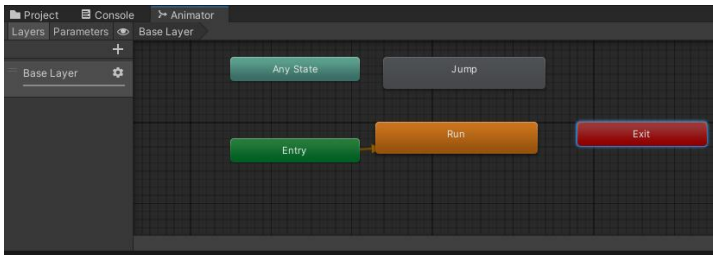
Para agregar más animaciones tenemos que ir al Animation del personaje al que queramos agregarle la nueva animación y creamos un nuevo clip de la misma manera que hicimos la animación de correr:



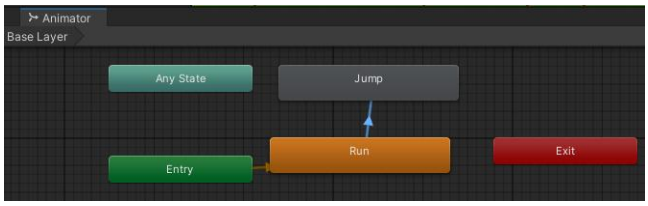
Abrimos Window, Animation, Animator:



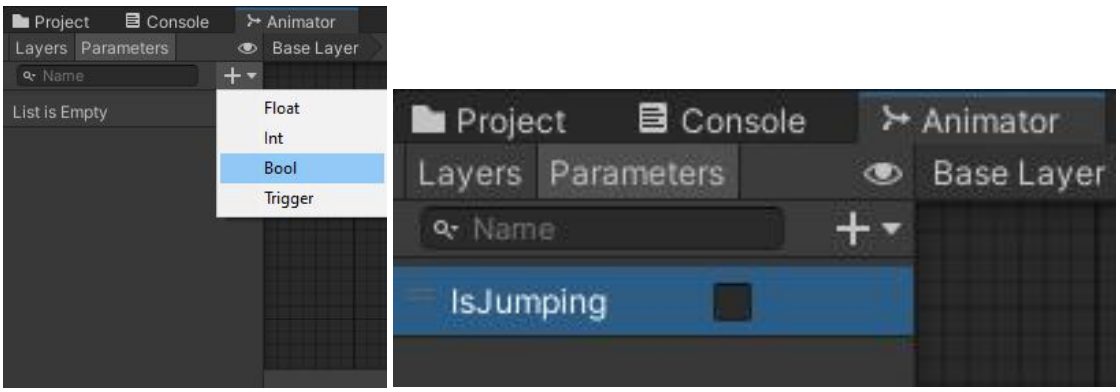
Tendremos una nueva ventana que nos deja ver el orden de las animaciones:



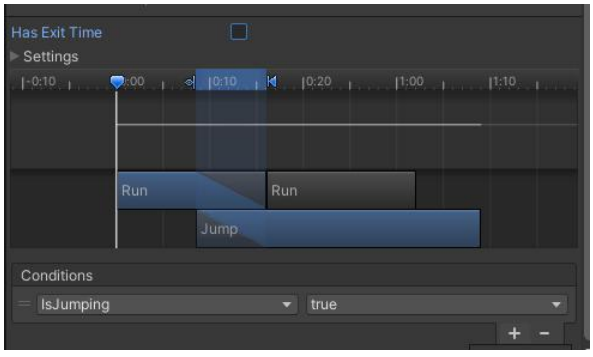
Podemos mover el diagrama y agregar condiciones para que se hagan las transiciones, con clic derecho make transition y seleccionamos la otra animación:



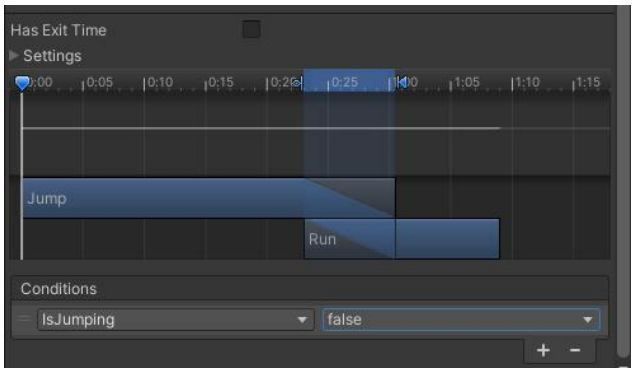
En la parte izquierda del Animator, en parámetros agregamos el bool que se llamará IsJumping (usamos el bool por que nuestro código solo era agregar el isJumping en el mismo lugar que el isGrounded, pero se puede hacer con trigger que lo veremos más delante)



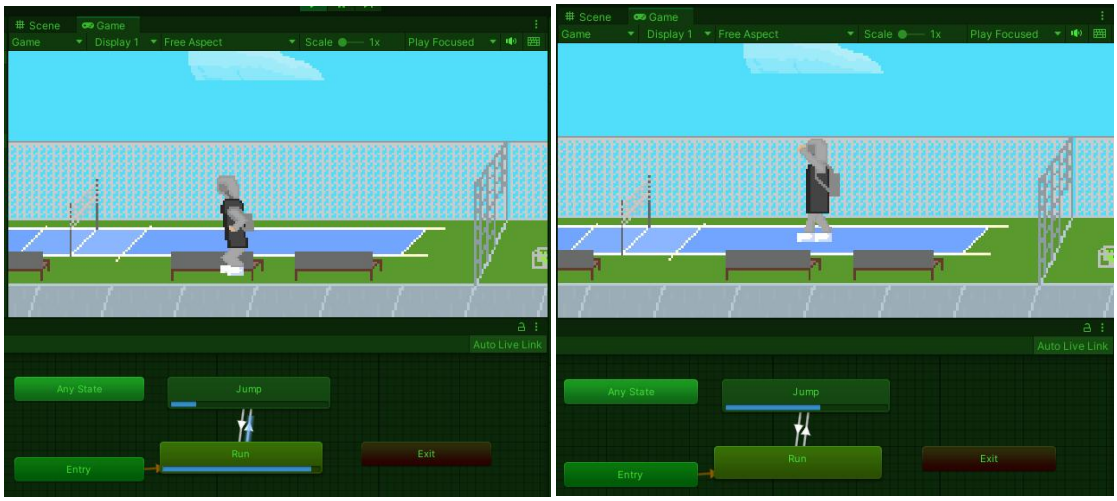
Ahora seleccionamos la transición que hicimos de run a jump y en el inspector desmarcamos la casilla Has Exit Time, (es una función para hacer mejores transiciones en 3D) y agregamos la condición de que IsJumping sea True.



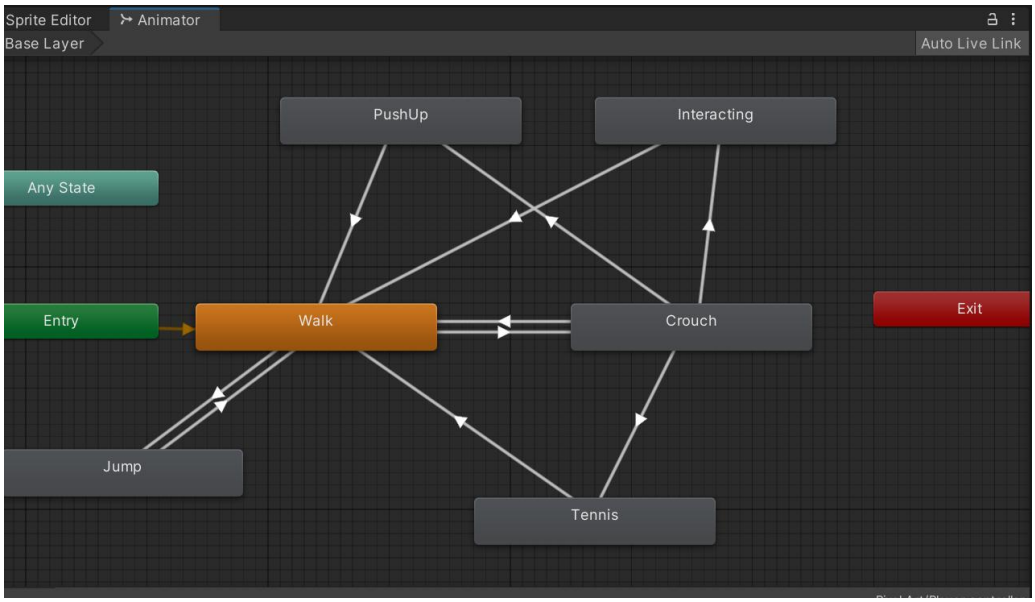
También hacemos la transición de regreso, cuando IsJumping sea False:



Al correr el juego debemos hacer que la animacion de salto se haga solo 1 vez, ya sea que el salto no sea tan alto o que la animacion dure más tiempo.



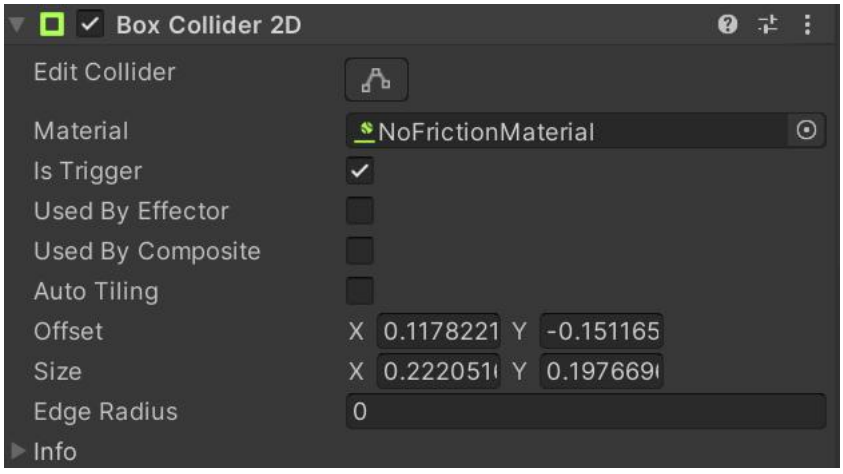
Hacemos lo mismo con las distintas animaciones y nos quedaría algo así:



Animaciones: obstaculos y jugador interactuando

Algo a notar son las animaciones Interacting, Push up y Tennis se activan cuando se está haciendo la animación Crounch y estas interactúan con cosas como la siguiente de basketbol.

Para explicar como hacer estas interacciones primero debemos tener los animator del jugador, del obstáculo y agregar el collider del obstáculo activando la opción de is trigger:



La opción trigger hace que al colisionar se puedan traspasar los objetos y existen 2 métodos importantes que usaremos llamados onTriggerEnter2D y onTriggerExit2D que nos indican cuando estén tocándose el jugador y el obstáculo.

Agregaremos un script que puede funcionar con todos los obstáculos. Inicialmente lleva las variables globales, algunas son del obstáculo y algunas son del personaje, como intAnimation que nos indica que numero de animación del jugador es la que hará al interactuar con ese obstáculo:

```
private Animator anim;
public PlayerMovement player;
private bool crounching;
private bool inTrigger;
public bool onlyTouch;
public int intAnimation;
```

El código son la función start, los visible on camera (solo se muestra un método pero también se debe agregar el otro método llamado OnBecameInvisible), el update:

```
void Start()
{
    anim = GetComponent<Animator>();
    // Desactiva la animación al inicio
    anim.enabled = false;
}

// Update is called once per frame
void Update()
{
    if(!onlyTouch){
        crounching = player.isCrouching;
        if(inTrigger && crounching){
            GetComponent<Animator>().SetTrigger("Interact");
            player.AnimCrunch(intAnimation);

            inTrigger=false;
        }
    }
}

 private void OnBecameVisible()
{
    // Cuando el objeto es visible en la cámara, activa la animación
    anim.enabled = true;
}
```

Como este código se utilizará en varios obstáculos algunos obstáculos interactuarán al tocarse en la función OnTriggerEnter2D, pero en caso de que deba estar agachado el personaje entra al if cuando se agacha estando colisionando y por lo tanto debe hacer la animación.

```
private void OnTriggerEnter2D(Collider2D collision)
{
    if (collision.CompareTag("Player"))
    {
        inTrigger = true;
        if(onlyTouch){
            // está saltando
            GetComponent<Animator>().SetTrigger("Interact");

        }else{
            // interactua si está agachado:
            crounching = player.isCrouching;
            if(inTrigger && crounching){
                GetComponent<Animator>().SetTrigger("Interact");
                player.AnimCrunch(intAnimation);

                inTrigger=false;
            }
        }
    }
}

private void OnTriggerExit2D(Collider2D collision) {
    if(collision.CompareTag("Player")){
        inTrigger=false;
    }
}
```

Vemos que al colisionar con el jugador, la variable InTrigger = true y al salir la pone el false, en caso de entrar al trigger pueden pasar 2 escenarios:

1- Si la animación se hace con solo tocarse, entonces se hace la animación del obstáculo.

2- En caso de que deba estar agachado se hace el mismo código del update.

En unity los configuramos el script a cada obstáculo:

Vemos que en este obstáculo el personaje hará la animación 1 que debemos agregar en el script del personaje:

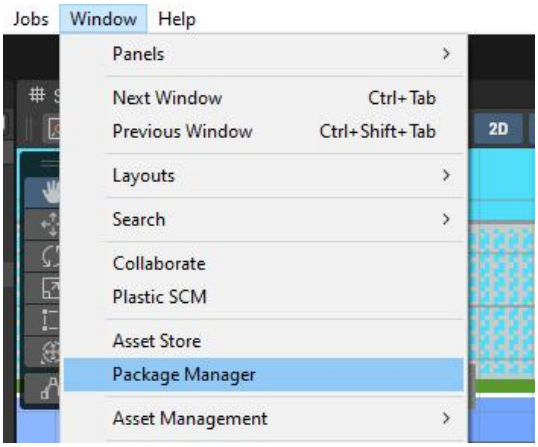
```
public void AnimCrunch(int animation){
    if(animation==2){
        GetComponent<Animator>().SetTrigger("PushUp");
    }else if (animation==1){
        GetComponent<Animator>().SetTrigger("Interact");
    }else{
        GetComponent<Animator>().SetTrigger("Tennis");
    }
    playerSpeedX = 0;
}
```

Este método se agrega al script playerAnimator, como resultado tendríamos las 2 animaciones ejecutándose si el jugador se agacha mientras colisiona con el obstáculo, si no se agacha y pasa caminando no lo hace:

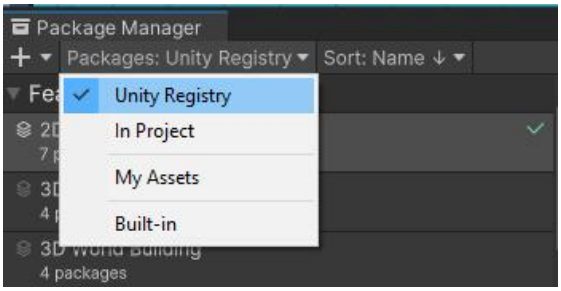
Cámara sigue al jugador

Por el momento el personaje se mueve y se sale del escenario, pero es importante poder posicionar el personaje a la derecha y recorra el escenario mientras la cámara lo sigue. Esto lo hacemos fácilmente instalando cinemachine.

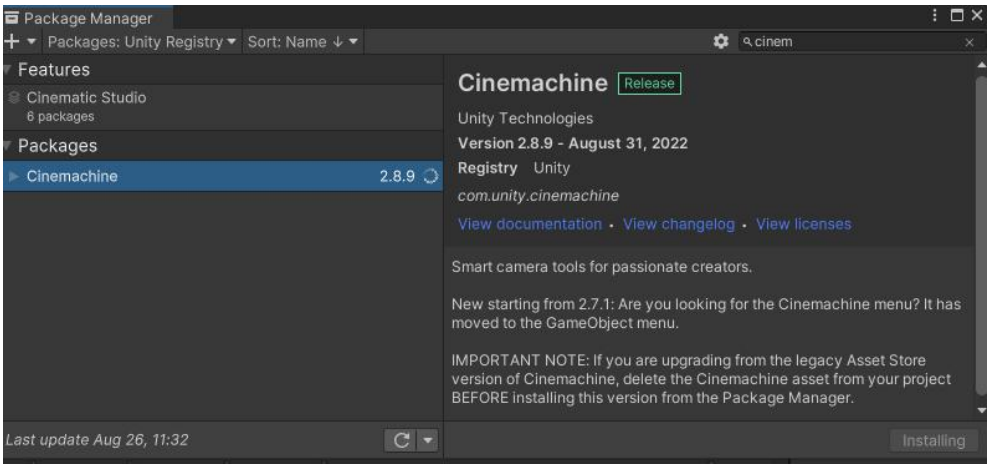
Vamos a Windows, Package Manager:



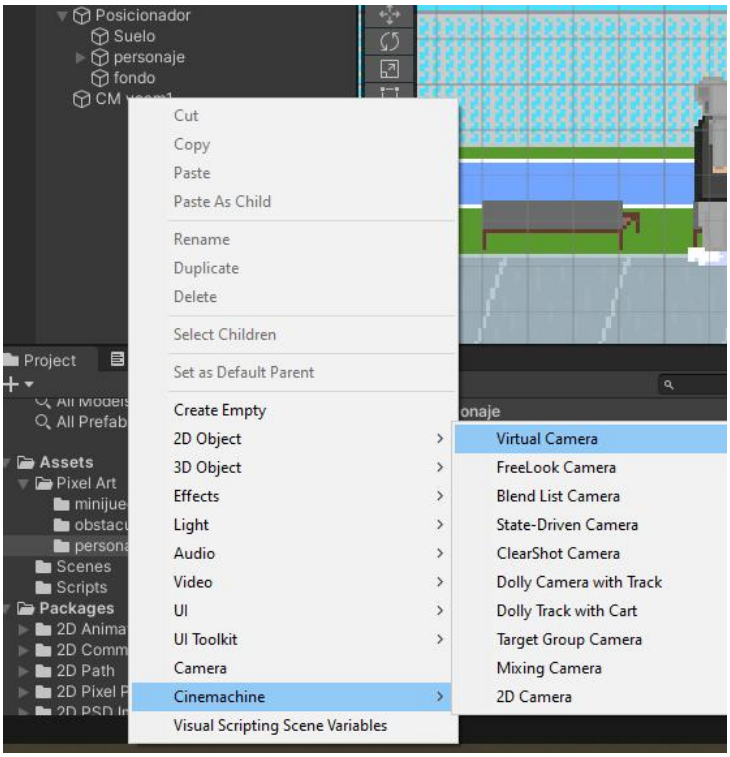
En la ventana, en packages, seleccionamos la opción Unity Registry:



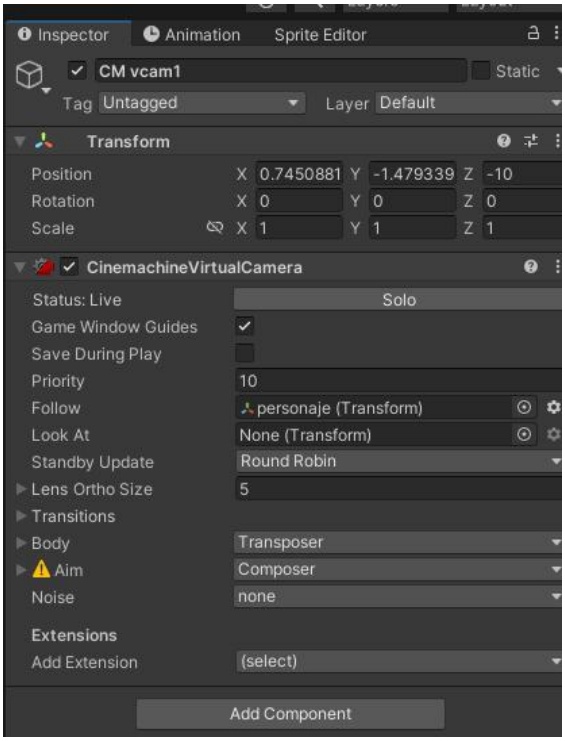
Buscamos Cinemachine y lo instalamos en el botón de abajo a la derecha:



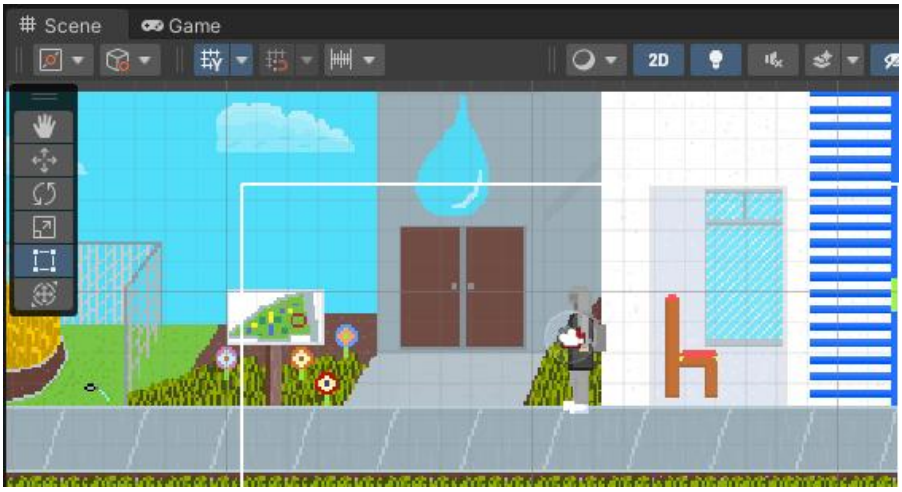
Una vez instalado en Hierarchy presionamos clic izquierdo, cinemachine, Virtual Camera:



Seleccionamos la cámara virtual y en el inspector le colocamos el personaje donde dice Follow. Si necesitas acercar o alejar la cámara cambiar el valor de Lens Ontho Size



Listo, ahora solo posicionamos el personaje en la parte donde queremos que inicie y quedaría listo:

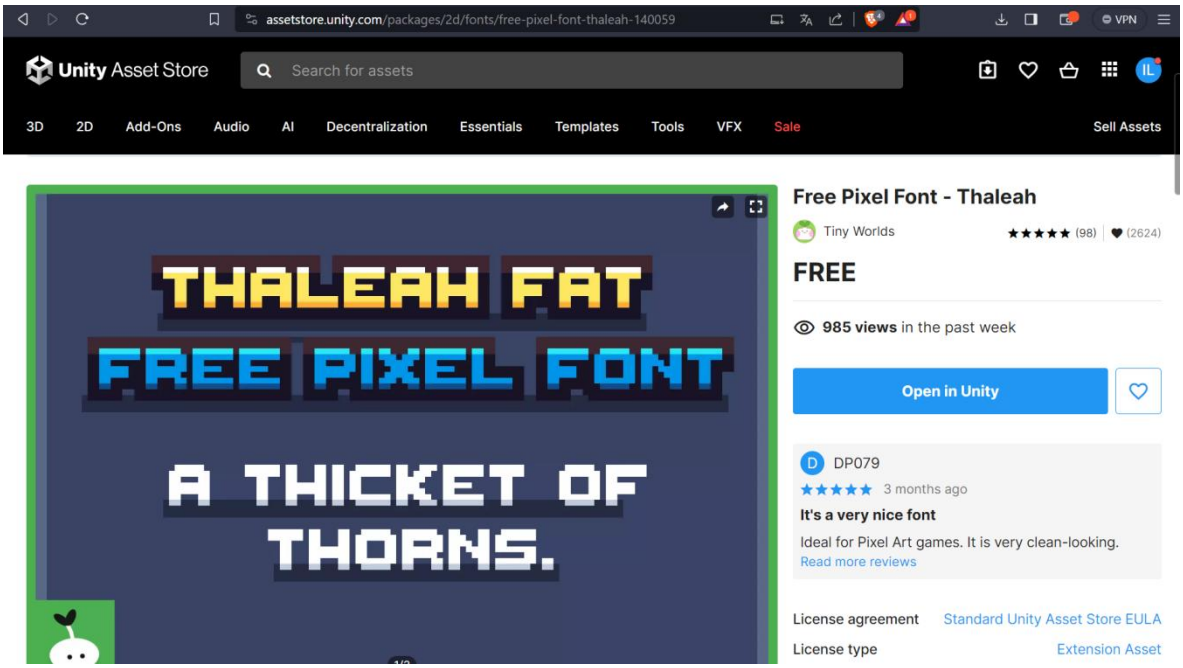


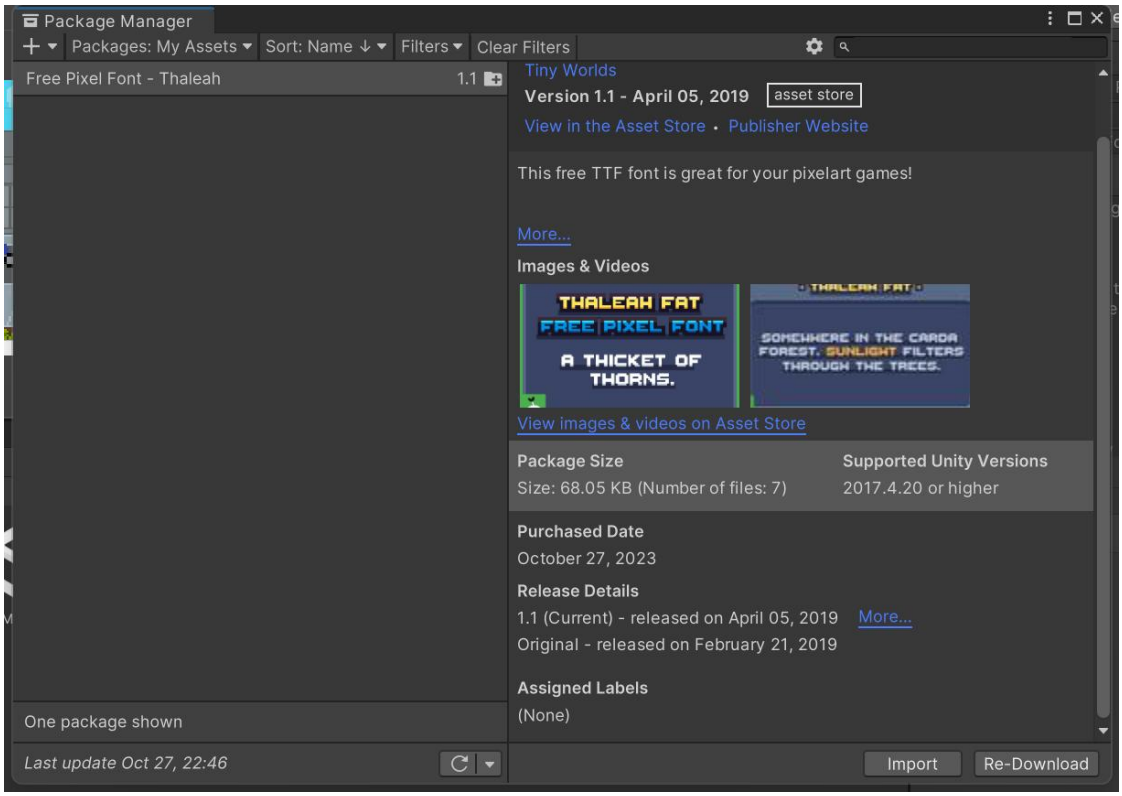
Agregar puntaje y tipo de letra Font

Aprenderemos a importar tipos de letra, es importante recordar que al importar cosas que no son tuyas debes revisar bien si puedes utilizar estos archivos y que condiciones deben cumplir, por ejemplo: algunos autores piden que los pongas en los créditos, otros piden que se utilice solo cuando no se utilice para ganar dinero, etc.

La manera de importarlos es la siguiente:

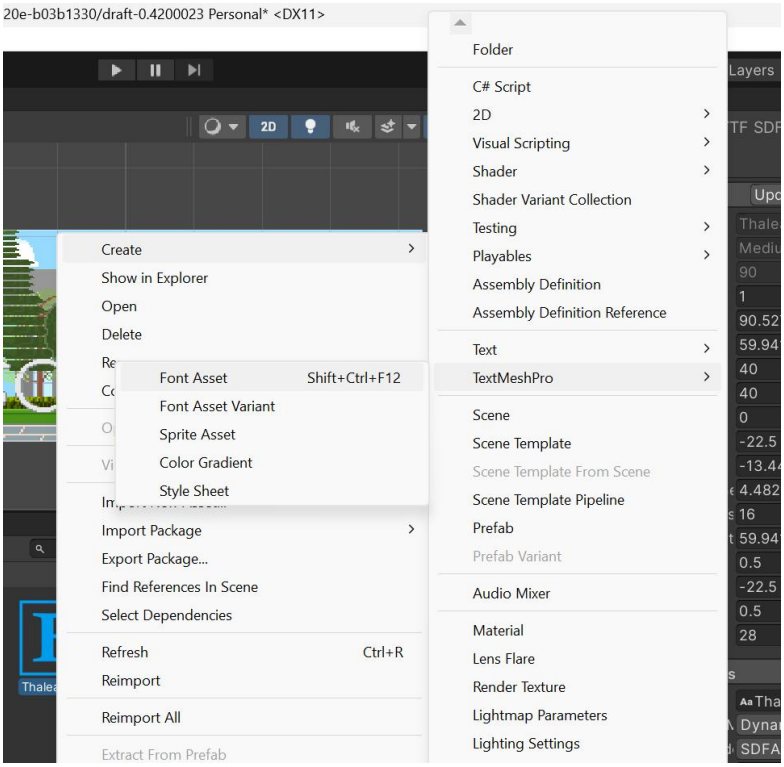
Existen varios lugares donde puedes descargarlos o hacerlos tu mismo, en este caso el usar unity Asset Store, que te manda directamente a unity:



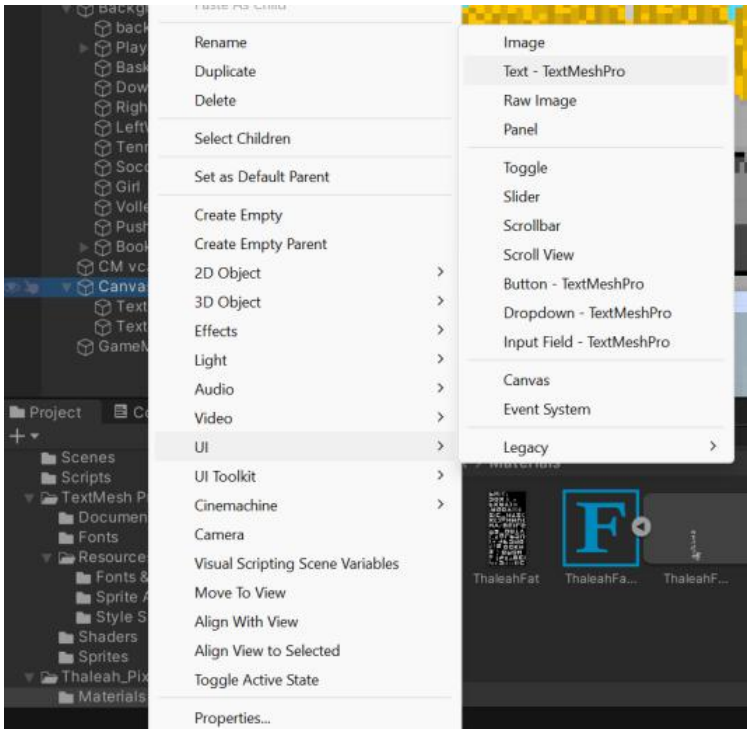


Abrirá la ventana anterior y lo que hacemos es presionar en los botones de abajo a la derecha, que dirán descargar e importar.

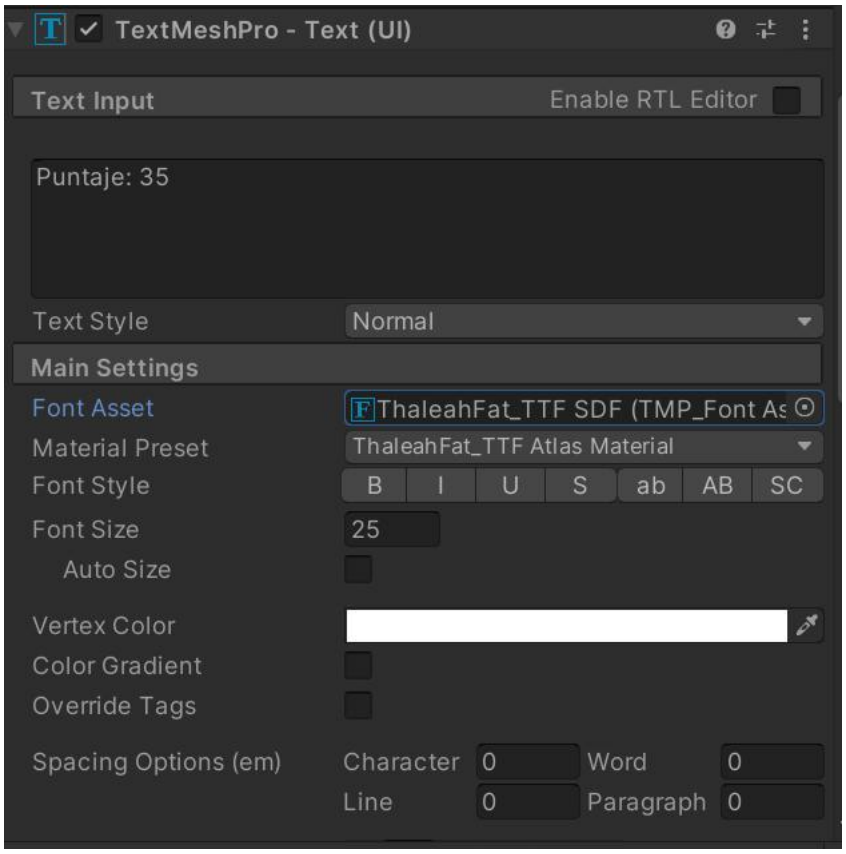
Presionamos import y creará una carpeta nueva en Assets con el nuevo font, pero debemos convertirlo a tipo Font con clic derecho, create, textMeshPro, Font Asset:



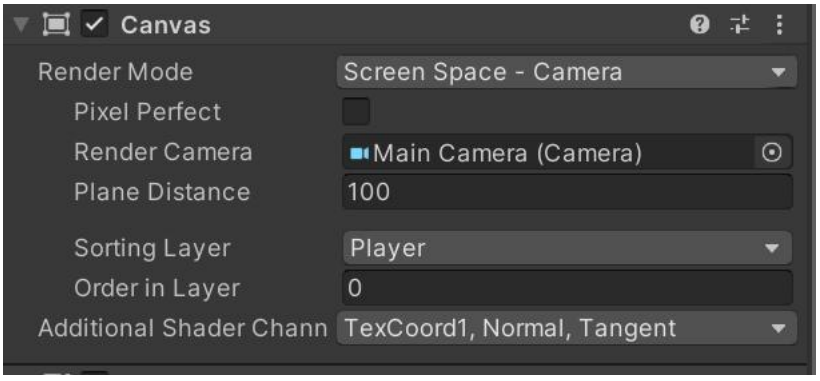
Una vez tengamos el font creamos un texto, yo crearé 2 text dentro de un canvas, uno que dirá el tiempo y otro que dirá el puntaje (el canvas se crea automáticamente al crear un texto):



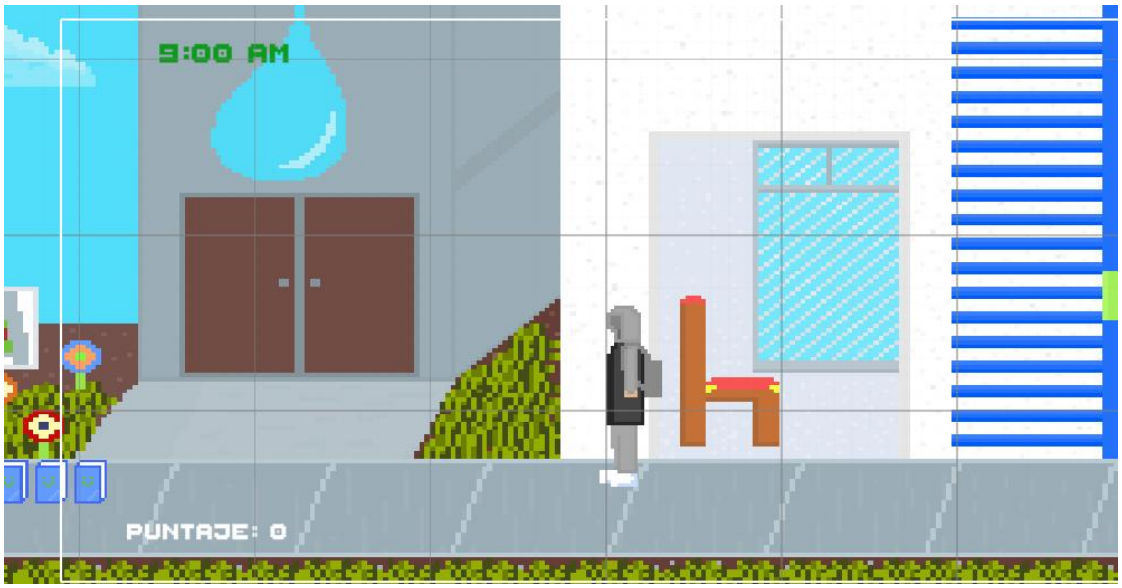
A ese texto le importamos el tipo de letra en Font Asset:



Para que el texto se vea siempre en la cámara, en el canvas donde están los textos cambiamos el Render Mode a Camera y en Render Camera ponemos la camara principal:



Así obtendríamos como resultado la cámara del personaje, con el tiempo en la mismo posición,aunque el jugador se mueva y a veces es necesario para que el texto se pueda ver:



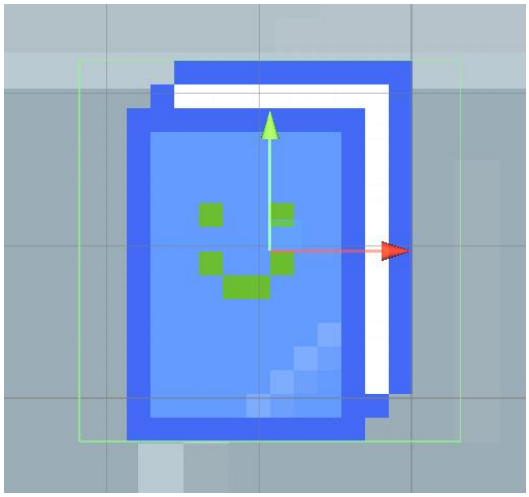
Para el caso del temporizador se agrega un Script llamado Timer en el text del tiempo y su código es bastante simple e intuitivo:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using TMPro;
using UnityEngine.UI;

public class Timer : MonoBehaviour
{
    public TextMeshProUGUI textoCronometro;
    private float tiempoInicial;
    private bool cronometroActivo = false;
    private int extraSeconds = 0;

    void Start()
    {
        tiempoInicial = Time.time;
        IniciarCronometro();
    }

    void Update()
    {
        if (cronometroActivo)
        {
            float tiempoTranscurrido = Time.time - tiempoInicial;
            tiempoTranscurrido = tiempoTranscurrido - extraSeconds;
            int minutos = 9 + Mathf.FloorToInt(tiempoTranscurrido / 60);
            int segundos = Mathf.FloorToInt(tiempoTranscurrido % 60);

            string textoTiempo = string.Format("{0:00}:{1:00}", minutos, segundos);
            textoCronometro.text = textoTiempo + " am";
        }
    }
```

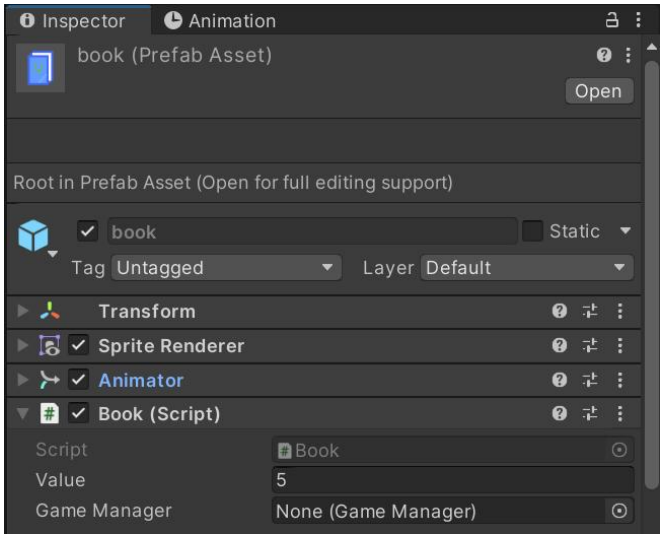
```
public void IniciarCronometro()
{
    cronometroActivo = true;
    tiempoInicial = Time.time;
}
```

El código es un simple cronómetro que se puede hacer de otra forma si se quiere. También vemos la variable extraSeconds para agregar o restar tiempo al hacer alguna acción, por ejemplo al agarrar un libro especial.

En el caso del puntaje, hay que hacer los libros que al tocarlos nos darán un punto y desaparecerán, así que hacemos el nuevo objeto con su imagen y animación:



Si vamos a usar muchas veces estos libros podemos hacer un prefab con solo arrastrar el objeto a la parte de los assets, sin embargo, los prefab tiene algunas limitaciones, por ejemplo en los scripts algunos valores quedan vacíos sobre todo los objetos que se encuentran en Hierarchy como el GameManager:



En este caso no lo usaremos como prefab, aunque si es útil sobre todo con cosas como las monedas, las vidas y objetos que se repitan en un juego, incluso plantas o flores, pero tendríamos que planear como hacer su código sin usar el GameManager.

El código usado para que al tocar un libro nos de puntos y desaparezca es en un script llamado book.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Book : MonoBehaviour
{
    private Animator anim;
    public int value = 5;
    public GameManager gameManager;

    // Start is called before the first frame update
    void Start()
    {
        anim = GetComponent<Animator>();
        // Desactiva la animación al inicio
        anim.enabled = false;
    }

    private void OnTriggerEnter2D(Collider2D collision){
        if (collision.CompareTag("Player")){
            gameManager.addPoints(value);
            Destroy(this.gameObject);
        }
    }
```

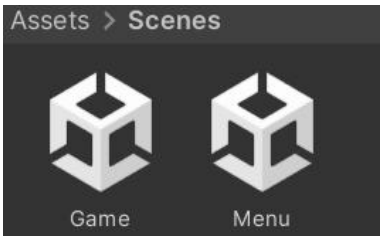
Y como se vio en la imagen anterior tambien usaremos el script gameManager.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using TMPro;

public class GameManager : MonoBehaviour
{
    public int TotalPoints{ get{return totalPoints;} }
    private int totalPoints = 0;
    public TextMeshProUGUI points;


    public void addPoints(int pointsToAdd){
        totalPoints = totalPoints + pointsToAdd;
        points.text = "Puntaje: " + TotalPoints.ToString();
    }
```

Cambiar escenas

En caso de cambiar de escenas, por ejemplo del menú al juego o de un nivel a otro, lo que hay que hacer es lo siguiente. En el caso de tener un menú:

Las escenas se guardan en Assets y en la carpeta de scenes,donde ya tenemos la escena del juego.



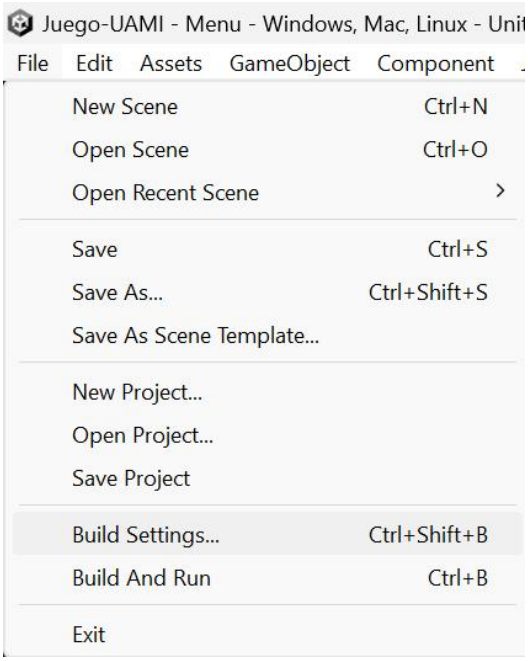
Lo que hay que hacer es el siguiente script llamado SceneManagerControl al tocar el botón:

SceneManagerControl.cs

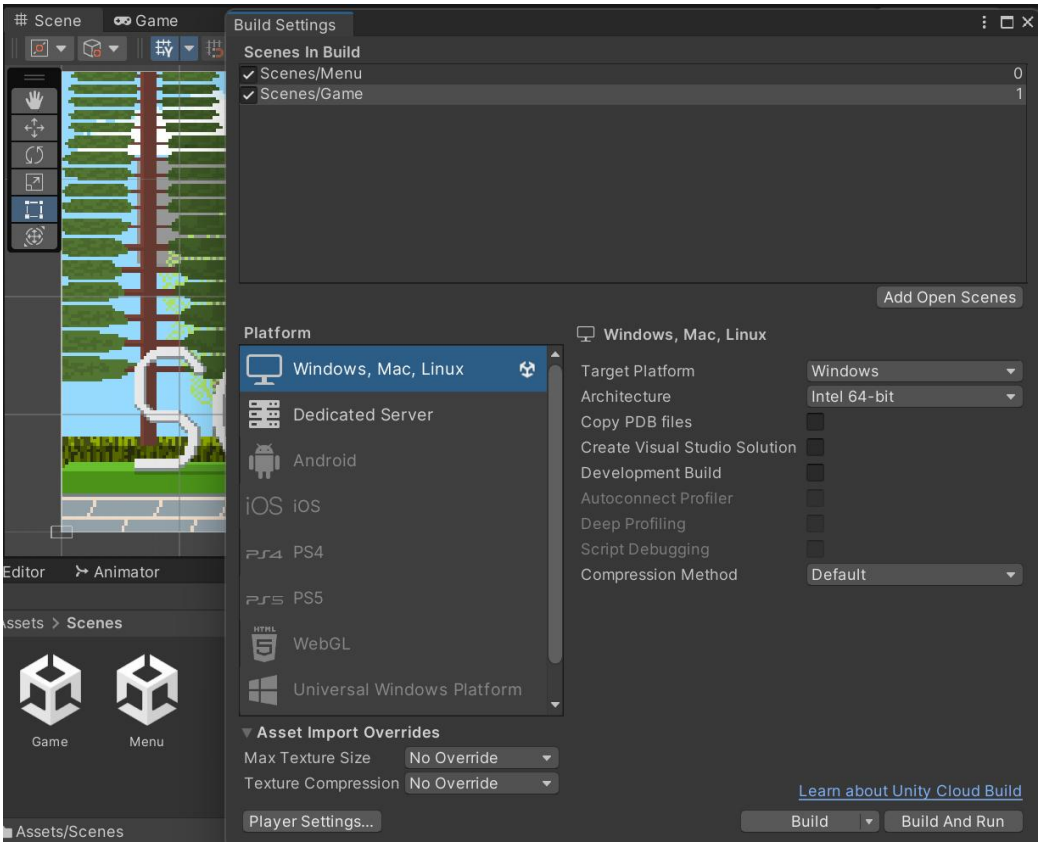
C: > Users > soudi > Documents > GitHub > Juego-UAMI > Assets > Scripts > SceneManagerControl.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class SceneManagerControl : MonoBehaviour
{
    // Función para cambiar de escena utilizando el número de la escena
    public void ChangeSceneByIndex(int sceneIndex)
    {
        // Asegúrate de que el índice de la escena sea válido
        if (sceneIndex >= 0 && sceneIndex < SceneManager.sceneCountInBuildSettings)
        {
            // Cargar la escena por su índice
            SceneManager.LoadScene(sceneIndex);
        }
        else
        {
            Debug.LogError("Índice de escena no válido: " + sceneIndex);
        }
    }
}
```

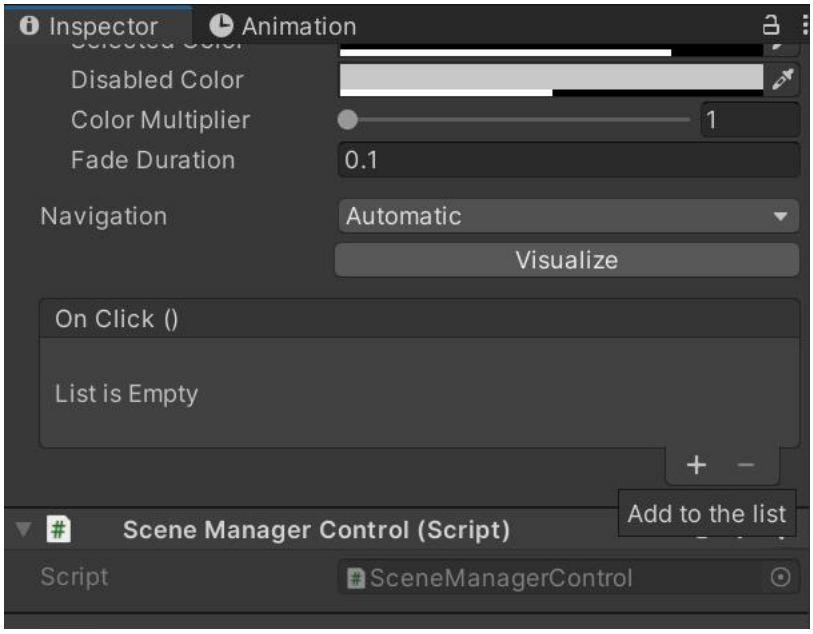
El código cambia de escenas anotando su numero. La manera de ver el numero de la escena y agregarlo es la siguiente:



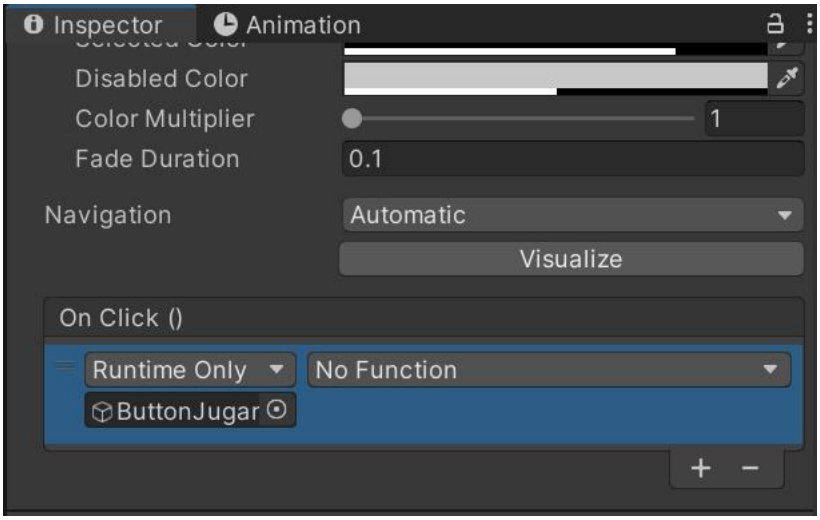
En file, Build Settings nos sale una ventana nueva donde arrastramos las escenas que hemos creado y del lado derecho podemos encontrar cual es su numero:



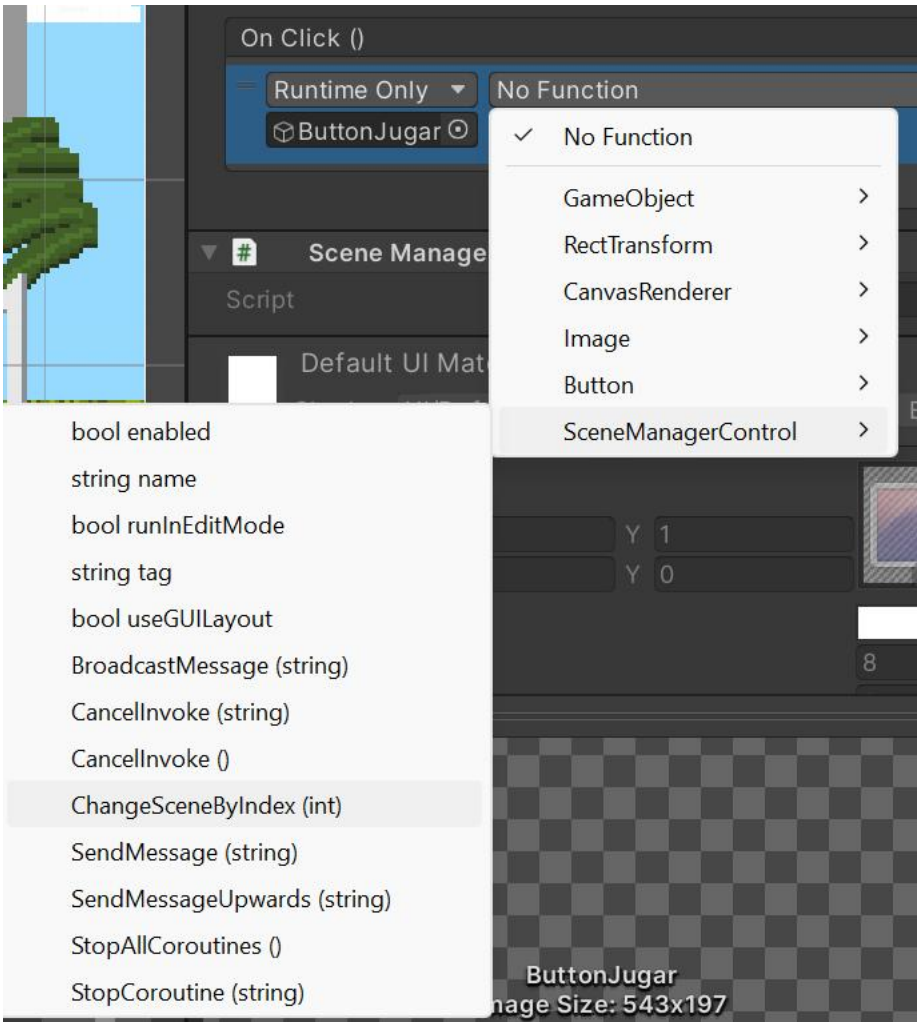
Ahora debemos agregar el script al botón y en la opción On click() presionar en el botón '+':



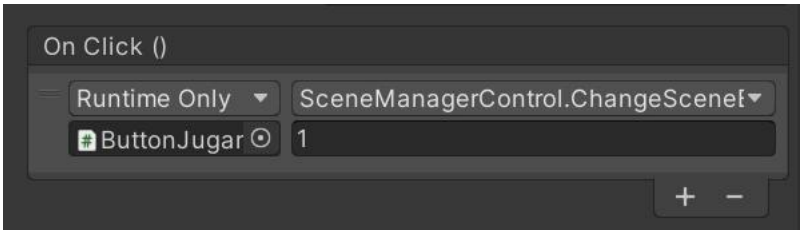
En runtime only debemos agregar el mismo botón, en este caso el botón jugar:



En la parte de No función elegimos la opción SceneManagerControl y la función del script ChangeSceneByIndex(int):



Para finalizar solo agregamos el numero que tiene la escena, en este caso pasamos a la escena 1:



De esta manera al presionar el botón jugar, cambiará de la escena del menú a la escena del juego.

Este mismo código se puede hacer con un trigger, por ejemplo al llegar a un salón y que empiece un mini juego como el siguiente:

Este minijuego empieza cuando se llega al final del mapa del juego anterior, puede tener una jugabilidad simple como presionar espacio para que el porcentaje aumente, si dejas de presionarlo el porcentaje baja, y al llegar al 100 pasamos al segundo minijuego:

Este minijuego también puede tener una jugabilidad simple, si se presiona la tecla que dice en el pizarrón aumenta el porcentaje y al llegar al 100% termina el juego con la escena final, donde todos se gradúan.