Creación de videojuegos

01. Unity

Estos días estamos dedicando las sesiones a trabajar el desarrollo de videojuegos. Para ello utilizamos el motor `Unity`, que nos permite crear proyectos 2D y 3D.

Posibilidades de Unity

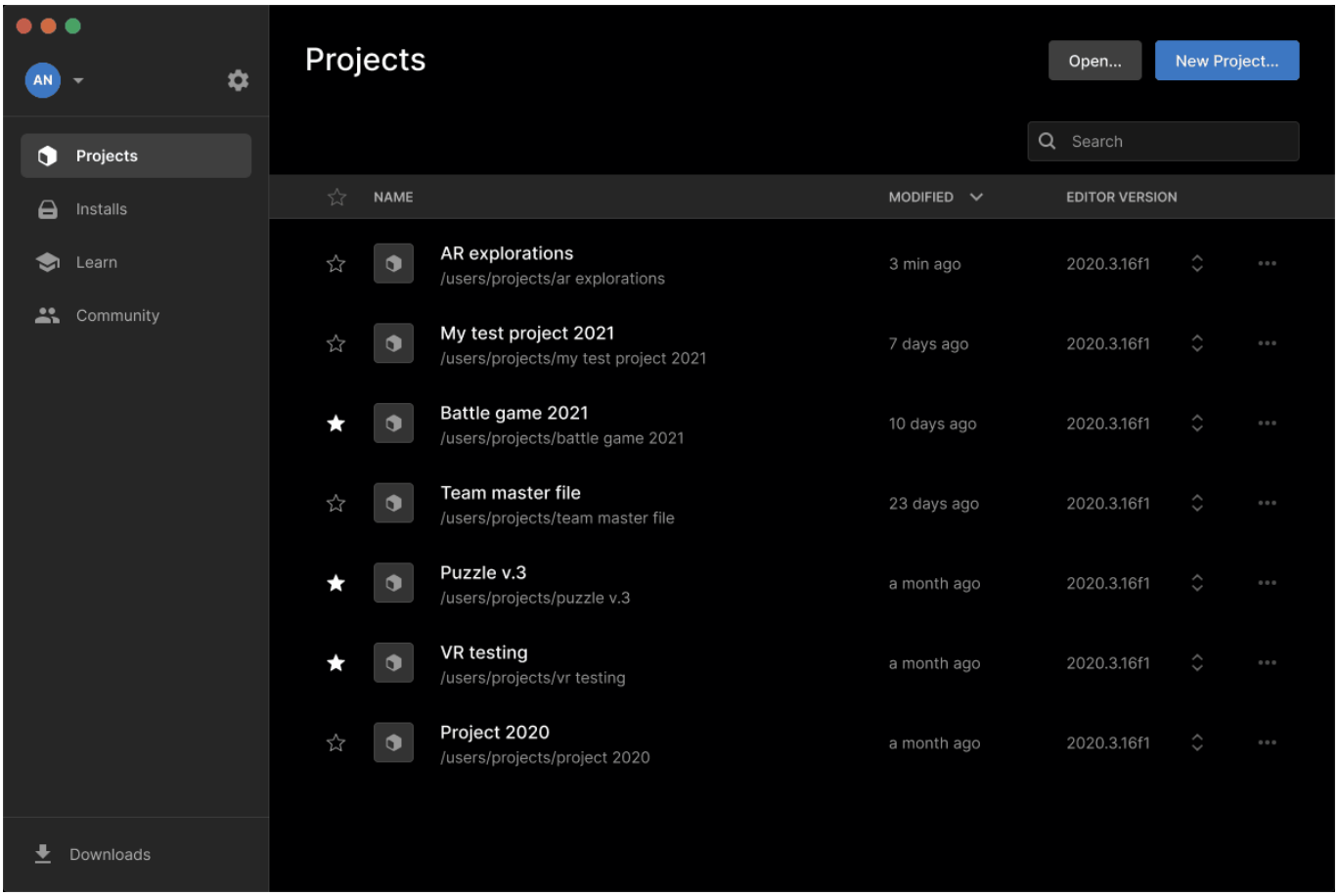
En este proyecto aprenderemos a realizar lo siguiente:

- Crear `objetos` de sprite 2D y colocarlos en escena
- Programar `scripts` que, asignados a los objetos, controlen su comportamiento
- Utilizar `controles` de teclado para mover los objetos
- Asignar `componentes` rigidbody para agregar respuesta a físicas en nuestros `sprites`
- Agregar `colliders` a los objetos para controlar las colisiones entre ellos y modificar el comportamiento del juego.

02. Instalación

Unity hub

En primer lugar nos descargaremos el `Unity hub`, una aplicación desde la que podremos gestionar las instalaciones y actualizaciones de todos los programas relacionados con Unity. También se nos mostrarán los proyectos que tenemos creados.



Cuenta de usuario

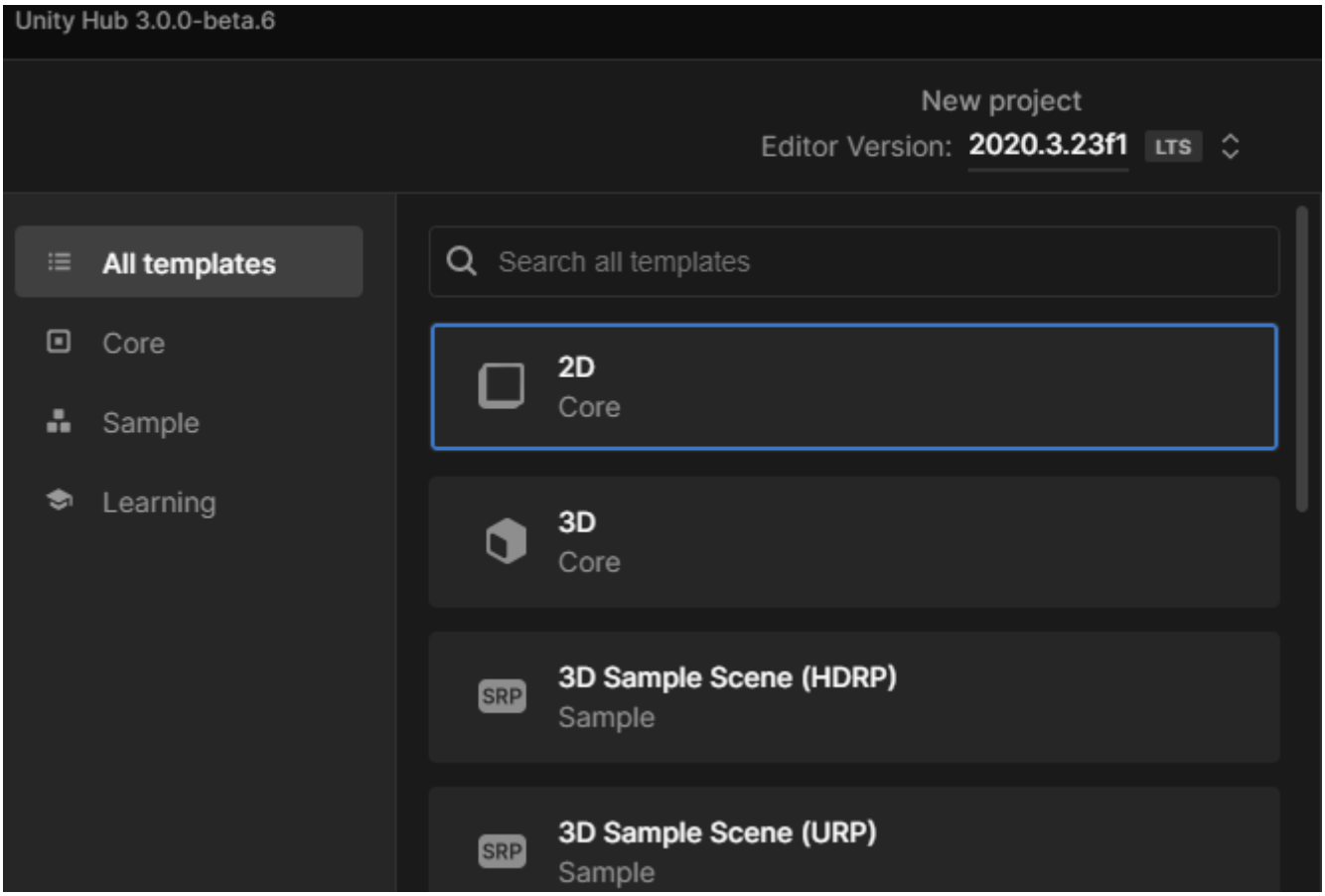
Para gestionar nuestras instalaciones, necesitaremos crear una cuenta de usuario de `Unity`, que deberemos crear para poder iniciar sesión y trabajar en nuestros proyectos.

Editor de unity

Posteriormente, instalaremos el `editor de unity`, que nos permitirá crear nuestros juegos.

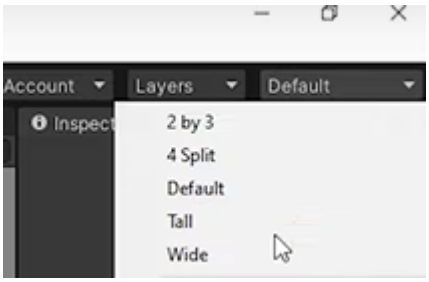
03. Crear un proyecto

Vamos a seleccionar lo importante es el template `2d core` que básicamente vamos a hacer el juego en 2d.



El proyecto el nombre pues le ponéis el que queráis, seleccionar pues una localización y cuando lo tenga listo pues le das a create.

Cambiar el layout



04. Crear pelota

Dentro de nuestra ventana lo que vamos a hacer es clic derecho y darle a judíos el spritesquare y como veis pues directamente seme pone pues un cuadrado que es el quevamos a utilizar para la bola.

Para los jugadores para las paredes para todoporque vamos a poder pues escalarlo enlos diferentes ejes y pues nos va aayudar a crear por como comentó el juegoen sí vamos a necesitar ni bajarnosninguna se ni nada va a ser todochocaron aquí en juniti así que buenogenera seguido un primer momento vamos acrear las paredes de arriba y de abajoentonces para ello

1. Vamos a la escala y vamos a colocarla en el eje x alo que sería un valor de 18
2. Ahora vamos a colocarlo en la posición 0 y 0 en el eje x y y
3. Ahora lo que vamos ahacer es subirlo hacia arriba para ello

4. También podéis pulsar +w+ cuando tengáis este objeto seleccionado y entonces pues podréis moverlo en ese eje en específico.

Cambiar resolución

Vamos a cambiar la `resolución` por la típica que se utiliza en pantallas que es la **16:9** que está se adapta muy bien pues ahora que es 1080 720 dos cada4 acá en ese caso que es la resolución este vertical y horizontal porque mantiene un aspecto de relación que cuadra en ese sentido entonces cuando como veis si yo creo a 18 y pongo esto en la posición de y a 4.5 queda perfectoahí en su posición de hecho es lo quevoy a hacer es ponerlo a 5 ya poner un poco más arriba a mí me gusta más que se vea pero que tampoco me ocupe pues media pantalla esa pared perfecto

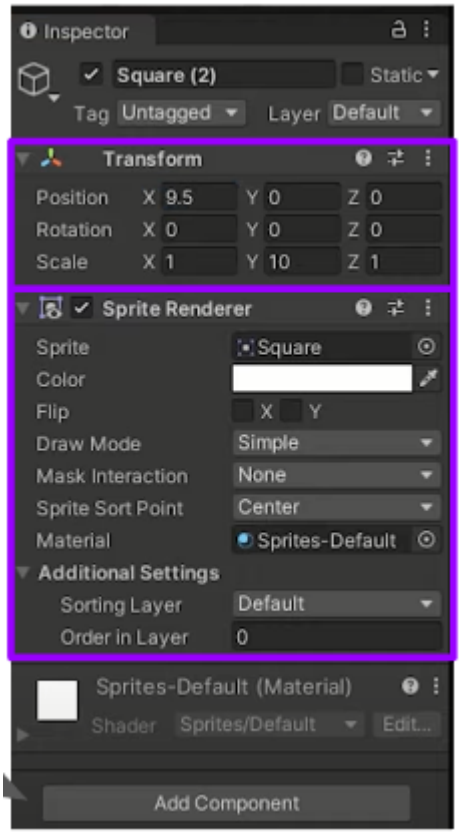
Podemos `duplicar` un objeto en lugar de crear uno nuevo. este que tenemos y ponerlo pues abajo del todo para ello pues lo que podéis hacer es control de o `clic derecho > duplicate` se duplicará y ahora lo único que tenemos que hacer es en vez de ir arrastrando lo va a estar en la posición contraria a 5 en este caso pues sería menos 5 a esta

05. Sprites

Un `sprite` es una imagen bidimensional que se utiliza como elemento gráfico en un videojuego. Se dibuja con herramientas de gráficos vectoriales o bitmap y se usa para representar personajes, objetos, entornos o cualquier otra imagen.

En `Unity`, los sprites se usan para crear gráficos 2D, como personajes, fondos, objetos y cualquier otra imagen que se use en un juego 2D. Estos sprites se pueden importar directamente desde un archivo de imagen o se pueden crear desde cero usando `Unity`.

Los sprites se almacenan en el formato .PNG y se pueden manipular fácilmente en `Unity`. Los sprites se pueden mover, girar, escalar y rotar fácilmente con `Unity`. También se pueden usar para crear animaciones y efectos especiales.



Todos los objetos por defecto tienen una posición y un tamaño. Esto se cambia en el componente Transform

06. Crear paredes de los lados

Para ello pues igual voy a duplicar el objeto y ahora lo que pasa esque mi escala en el eje x no va a ser sino que va a ser en el eje y entonces en este caso puedo dejar la escala en el eje x a1 y la escala en el eje y a

Si pulsáis la rueda del ratón podéis hacer una especie de paneo.

Ahora lo que quedaría es ponerlo a la derecha y ala izquierda entonces en este caso lo que vamos a hacer es poner en la posición de y a 0 y aquí vamos a modificar la posición en el eje x en este caso la podemos poner por ejemplo en ocho y más o menos vemos que se queda cerca de lo que sería pero estas paredes no se tienen que ver tienen que estar por la parte de fuera

Vamos a colocar pues por ejemplo ahí lo quesería pues yo creo que el nueve y medio va bien estas paredes van a ser diferentes a esta nueva entre colisión en el hecho de que no van a hacer que rebote la pelota sino que cuando colisionan con esta pared va a ser como la portería de un jugador o del otro y entonces pues se añadirá un punto cuando colisiones perfecto

Podemos duplicar el objeto. y ponerlo en el lado contrario que simplemente poniéndole un negativo en el 9,5 pues se pondrá justo en el otro lado.

07. Nombrando los objetos

Ahora importante nombrar las cosas porque vamos a empezara tener muchos objetos y nos podemos confundir. Además, deberemos poder identificarlos en los scripts que crearemos para manipularlos.

Para renombrar un objeto hacéis doble clic o con +F2+. La portería derecha vamos a llamarle en goal1 porque va a ser la portería en la que tiene que colar el jugador 1 que va a estar a nuestra izquierda.

Este otro lado de la izquierda pues sería **goal2** adiós con lage mayúscula perfecto porque íbamos a pasar a hacerla línea del centro un poco para que sepamos cuál es el centro todo lo que puedes hacer es duplicar un gol que tenéis vosotros aquí colocado en la posición 0 0 y ahora es reducir un poco

La escala en el eje x podéis reducir la escala también pulsando la +r+ teniendo seleccionado huevas y entonces sancionando vuestro jugador y pulsando deseos pondrá aquí lo que sería el ritmo de escala que es lo que tenéis aquí arriba mismo de movimiento rotación escala y luego otro raro que tienen por aquí que es de escala rotación movimiento todo y vamos para ir a colom seleccionar es el de rotación y

Vamos a reducir esto como es si yo cojo lo quees el eje rojo y lo reduzco o el aumento pues ese aumento se reduce y se ve reflejado en la escala aquí en x vamos a colocarlo a 0 con 2.

08. Creando los jugadores

Vamos a crear los `players`. entonces vamos a hacer control de sobre ese amigo aquí de la derecha lo voy a arrastrar a esta posición que de hecho lo vamos a poner en la posición 8 y ahora hay que reducir la escala. Así imaginaros que vuestro `player` es así de grande en 3 cuando venga la bola va a rebotar sí o sí, al ser demasiado grande.

Crear `jugador2`

Vamos a reducirlo unpoco a lo que sería en el eje y a 2.5 luego pues podemos modificar elmovimiento de la velocidad de la bolapodemos modificar muchas cosas pues paraque se adapte a lo que cada uno puesesté buscando en este caso como está ala derecha pues lo llamaremos player2.

Crear jugador 1

Ahora ++ctrl+D++ y duplicamos y lo llevamos a la posición contraria que sería -8 y le llamamos player1 para tenerlo ahí bien diferenciado.

Ahora lo único que ayudaría sería pues nuestra pelota del medio así que podemos hacer directamente clic derecho dentro de nuestro `Unity` su día dietsprite square y si la colocamos en el 0,0.

Para poder diferenciar lo mejor vamos a cambiarle el color y esto lo podéis hacer con todos los objetos en verdad que tengáis en la escena como veis son de tipo sprite renderer por tanto si lo seleccionas podéis modificar aquí el color con lo que tenéis a la derecha.

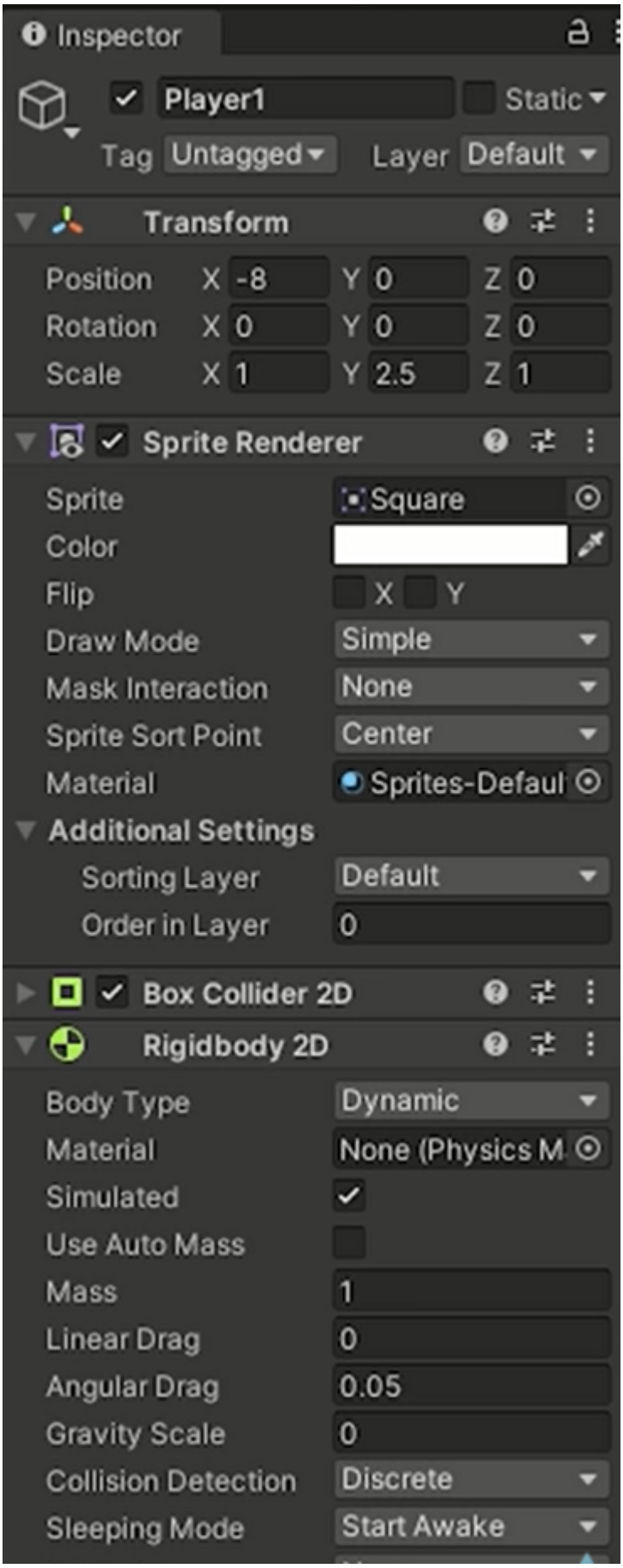
Yo lo puedo poner pues este color amarillento a la pelota y sin soluciones cualquier otro por ejemplo podéis solucionar asaco y mirar puedo modificar pues todo el mapa en sí de hecho lo vamos a hacer voy a seleccionar el goal escuela y demás y lo vamos a poner en un tono rojizo

Ahora podéis seleccionar vuestro player2 le ponéis otro color. Voy a ponerle un tono verdoso lo mejor paraje verde parece que es como otro no puedo colocar creo que voy a dejar blanco me está gustando más blanco lo voy a dejarblanco los dos pero bueno:

09. Componentes rigidbody

`Rigid Body 2D` es un componente de `Unity` que se utiliza para añadir físicas a un objeto 2D. Un `Rigid Body 2D` le permite a un objeto 2D afectado por la gravedad, el empuje, la fricción y otras fuerzas físicas. Esto permite a los desarrolladores añadir realismo y jugabilidad a los juegos 2D.

Asignaremos los componentes `Rigidbody 2D` a nuestros jugadores y a la pelota. Podemos seleccionar todos los objetos y seleccionar el componente `Rigidbody 2D` para añadirlo. Al dar a play los objetos con Rigidbody caerán, pues sobre ellos actúa la gravedad. Para quitarles el efecto de la gravedad, donde pone gravity podemos ponerlo a 0 y así ya no se caerán.



Por ejemplo, la **gravedad** afectará la posición del objeto para simular su efecto, haciendo que caiga hacia abajo en el eje Y.

Los `rigidbodies` son componentes que añadiremos a nuestros `players` y para nuestra `pelota` y se encargarán del tema de física para el movimiento.

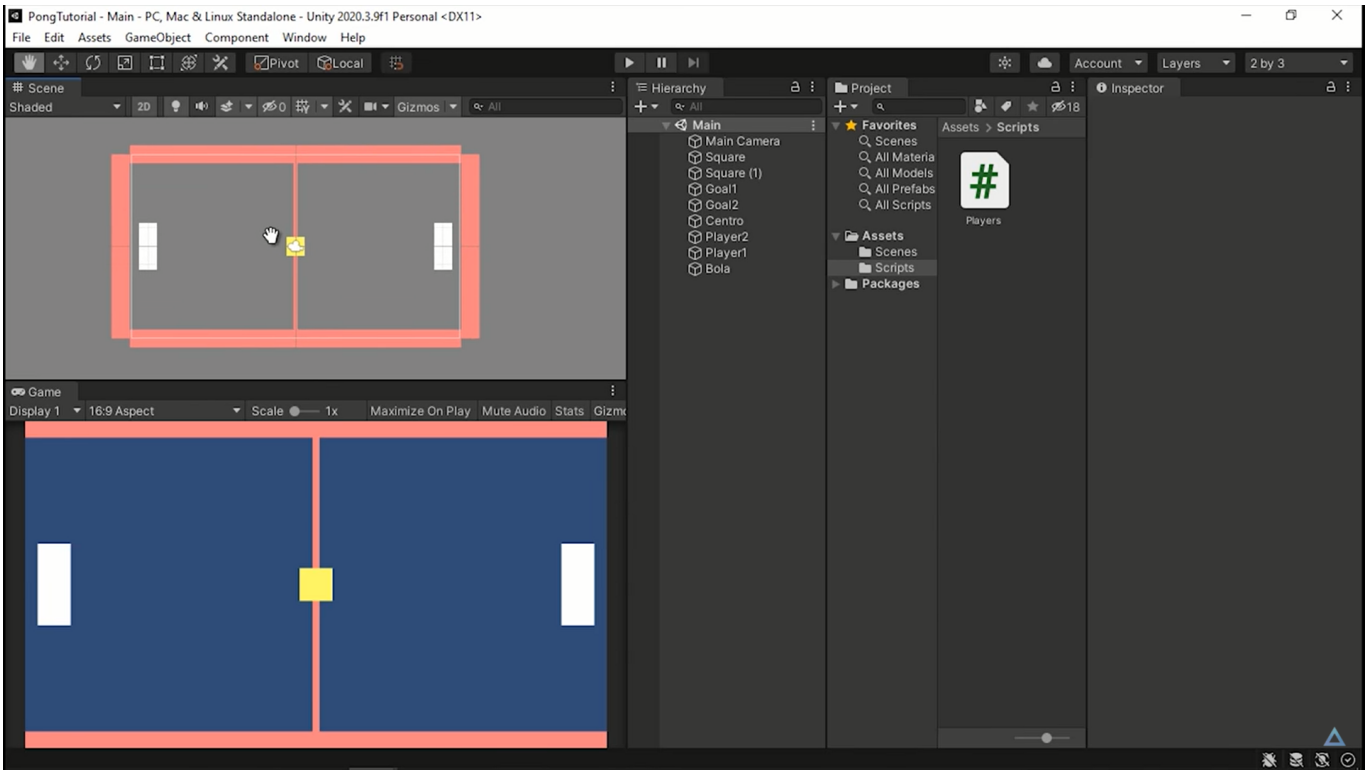
- Vamos a seleccionar nuestro `player` y nuestra `bola`.
- A continuación seleccionamos todos en el componente `Rigid Body 2D` y ahí le añadimos.

Si dejamos esto así tal cual cuando yo le diera el `play` vais a ver qué los elementos caen. Esto es porque tienen física y por tanto, les afecta la gravedad. En nuestro caso esto no lo queremos.

Lo que tenemos que hacer es dentro de nuestro componente `Rigid Body 2D` donde pone `gravity` es que a uno vamos a ponerlo a 0 y así ya no se caerán.

10. Ordenar objetos

tra cosa que modificando que a lo mejor les ocurre es que la línea del centro se está dibujando por encima de nuestra pelota yeso pues la verdad que no queda muy buena lo mejor sí a lo mejor lo queréis vale pues lo dejáis lo dejáis así si os gusta pues lo dejáis pero si no lo que podéis hacer es se detiene al puesto al bola



En el elemento `sprite renderer` podemos cambiar el orden. Esto sirve para diferenciar la altura a la que se dibujan las diferentes elementos dentro de nuestra pantalla porque ahora mismo son todo imágenes entonces para saber diferenciar cuál está por delante de una de otra utilizamos el orden y léger. __

Podemos poner un 2 ya se dibuja por encima así que así no habrá problema.

La `línea del centro` se está dibujando por encima de nuestra `pelota`.

Para modificarlo, seleccionamos el objeto `bola` y en el elemento `Sprite Renderer` en el `orden` ponemos un número más alto. Esto sirve para diferenciar la altura a la que se dibujan las diferentes elementos dentro de nuestra pantalla. Ahora la `bola` se dibuja por encima.

11. Movimiento de personajes

Ahora vamos a pasar directamente a lo que sería el tema de los movimientos de nuestros personajes. Vamos a hacerlo con:

- Teclas ++w++ y ++s++ para el jugador 1 (izquierda)
- Teclas flecha de arriba y hacia abajo para el `jugador2` (derecha)

De este modo, podremos jugar 2 jugadores en el mismo teclado.

12. Redefinir controles

Tenemos que definir pues esos controles y para ello:

- Nos vamos a ir a `edit > project settings`.
- Donde tenemos `input manager` tenemos declarado varias cosillas que si no lo vais a expandir el `axis` y estos son como `shortcuts` o controles que tiene definido djinnit y para diferentes teclas de nuestro teclado.
- En en este caso tenemos `horizontal` que se encarga de saber cuándo estamos pulsando a la izquierda o derecha de las flechas o el add de nuestro teclado que son las típicas teclas que se utilizan para jugar wsb y las flechas de abajo a la derecha entonces pues con esto sabrá cuando llamemos a la horizontal pues si estamos yendo hacia la izquierda o hacia la derecha pasó lo mismo con arriba o abajo pero en este caso tendríamos el ws y la flecha de arriba y hacia abajo entonces tenemos el vertical y

Tendremos que crear el `vertical2` que es para nuestro `jugador2` y tenemos que diferenciar pues que uno utilice el ws y el otro utilice la flecha hacia arriba y hacia abajo espero que me haya explicado lo mismo escribano con el culo pero bueno podéis seguir tranquilamente con el vídeo y pues si no queda claro pues lo vamos a ir viendo y seguramente que se entienda entonces seguramente a vosotros os haga aquí `down` y `up` y w no y yo salgo aquí s&w y esto es porque el vertical ya pilla directamente tanto la parte izquierda de teclado como la de la derecha pero aquí lo tenemos que diferenciar

En `vertical` vais a borrar el sw que tenemos aquí y lo que vamos a hacer es `duplicarlo` para tener un `vertical2` para nuestro `jugador2` en cáceres clic derecho duplicate a ride element y se duplicará que vendrá por aquí otra como vertical.

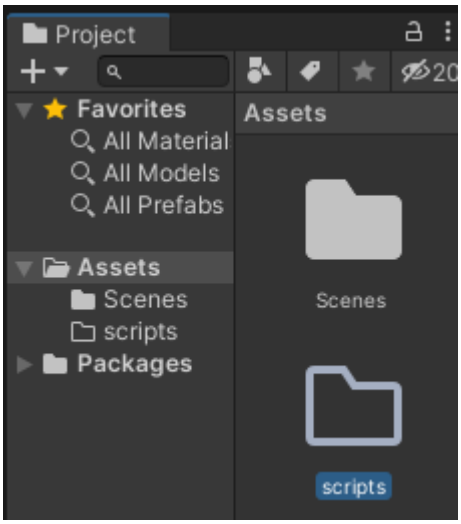
Entonces le vamos a hacer este más blog `vertical2` y entonces ahora aquí lo que faltaría es cambiarle los controles si ni si el `player1` juega con la flecha de arriba y hacia abajo nosotros con el `jugador2` jugaremos con la s w entonces en negativo button tenéis que poner ese y en positivo ton tenía que poner ++w++ ycon eso cuando pasemos al código será muchísimo más sencillo porque además sólo tendremos que hacer un script para los dos players.

13. Script de programación

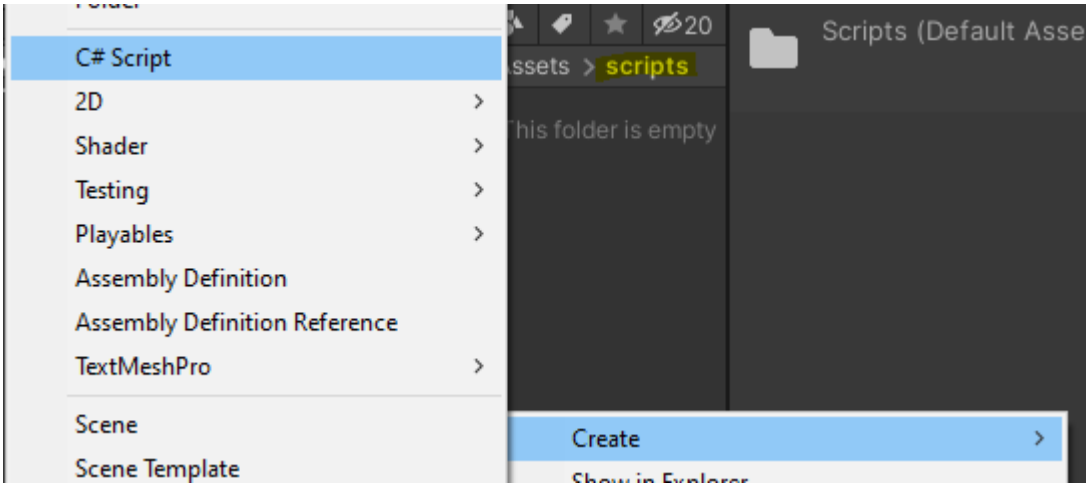
Los `scripts` son pequeños programas que controlan el comportamiento de los objetos y la dinámica del juego en `Unity`.

Estos se crean utilizando el `lenguaje C#` y se guardan en archivos con extensión `.cs`. Para editarlos se utiliza Visual Studio Code, y se guardan dentro de la carpeta `assets`.

Dentro dela carpeta `assets` vamos a hacer clic `derecho > new folder` que vamos a llamar `scripts` y así tendremos guardados todos nuestros programas en esta carpeta. Para crear un `script`, hacemos clic derecho en la carpeta y elegimos la opción `create C# script`.



Ahora vamos a pasar a crear nuestro script vamos aquí a entrar a la carpeta que acabamos de crear y vamos a hacer `clic derecho > create C# script`. A este script le vamos a llamar `Player.cs`. Podríamos abrirlo y vamos a utilizar el programa visual studio.



Estructura de un script

Los scripts en `Unity` tienen una estructura básica compuesta por dos partes principales: la parte de declaración de variables, y las funciones.

La parte de **declaración de variables** es donde se definen los campos, variables y propiedades que se usarán en el script.

La segunda parte es la **sección de funciones**, donde se escribe el código que controla el comportamiento de objetos en el juego.

- La función `Start()` se llama al comienzo del juego (una vez) y generalmente se usa para inicializar variables y configurar el estado inicial del objeto.
- La función `Update()` se llama una vez por frame y se usa para actualizar el estado del objeto.

Además de estas dos funciones, podemos crear todas las funciones que queramos para controlar el comportamiento de un objeto, desde eventos de entrada (como cuando un usuario presiona una tecla) hasta eventos de salida (como cuando un objeto sale del juego). Estas funciones se pueden llamar en el script para ejecutar el código deseado.

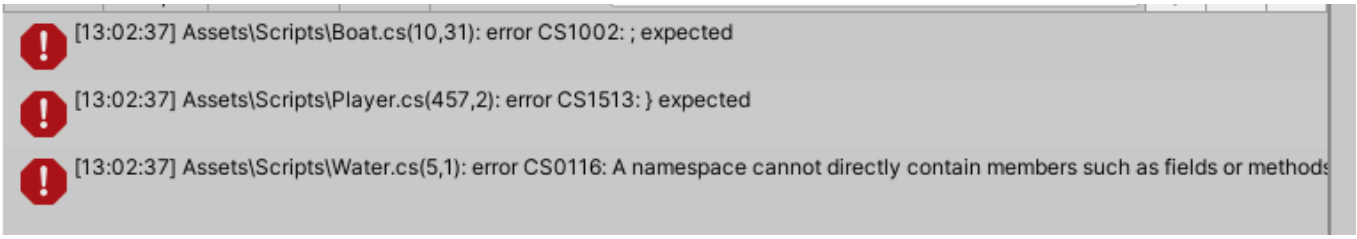
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class NewBehaviourScript : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {

    }

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

    }
}
```

Errores de programación

Los errores de programación más comunes en `Unity` son errores de sintaxis. Estos se producen cuando el programador escribe algo de forma incorrecta, por ejemplo, olvidarse de poner ; al final de las líneas o cerrar un }, o escribiendo mal mayúsculas o minúsculas.

Si hay errores de sintaxis, `Unity` no podrá ejecutar el juego correctamente, por lo que el programador debe solucionar los errores antes de poder continuar.

Algunas de las formas más comunes de solucionar estos errores son comprobar el código con cuidado, revisar la documentación para asegurarse de que está escribiendo cada línea correctamente



Crear script players

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

public class Players : MonoBehaviour
{

    public bool player1;
    public float speed = 3;
    public Rigidbody2D rb;
```

```
    private float move;
    private Vector2 startPos;

    void Start()
    {
        startPos = transform.position;
    }
    void Update()
    {
        if(player1)
        {
            move = Input.GetAxisRaw("Vertical");
        }

        else
        {
            move = Input.GetAxisRaw("Vertical2");
        }

        rb.velocity = new Vector2(rb.velocity.x, move*speed );
    }
    public void Reset()
    {
        rb.velocity = Vector2.zero;
        transform.position = startPos;
    }
}
```

14. Asignar script al jugador

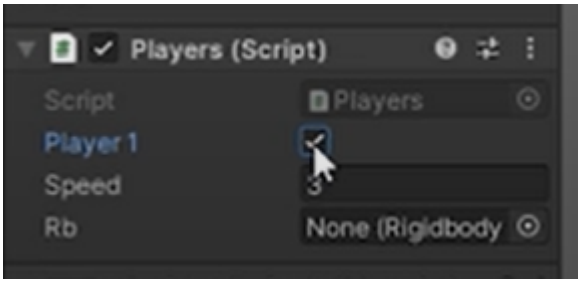
Para que un script tenga efecto, hay que `asignar` el script a uno o más objetos. Para ello los scripts se arrastran y sueltan sobre los objetos que queremos que los utilicen.

Cada vez que volvamos a `Unity` después de modificar nuestros scripts, se recargará el proyecto para incluir los cambios.

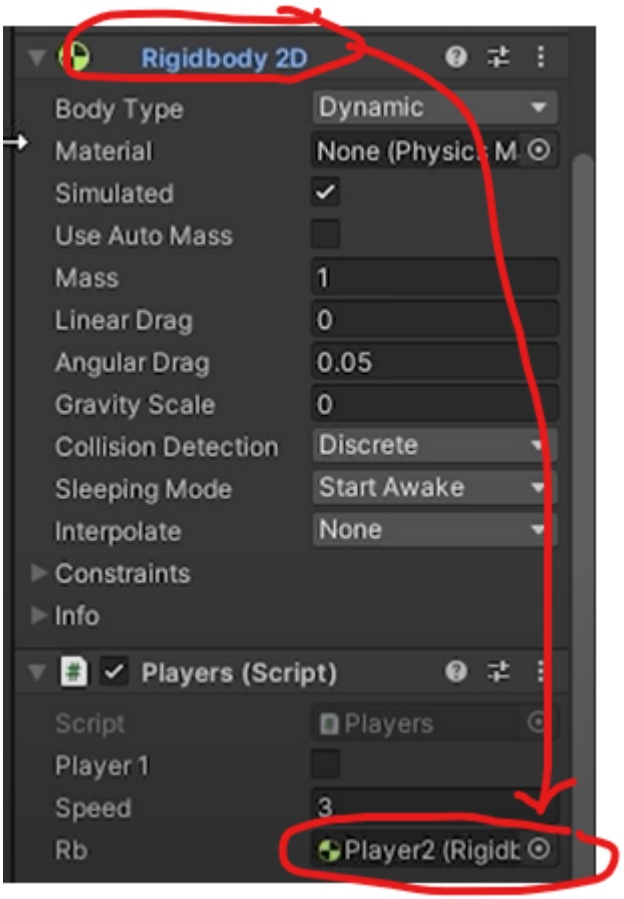
En el caso de haber errores de programación, deberemos primero subsanarlos.

Una vez terminado, podemos asignar el script `player.cs` a los objetos `player1` y `player2`, arrastrándolos y soltándolos sobre estos objetos.

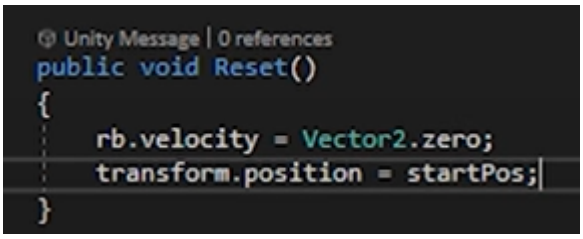
Cambiar el booleano para player2



Asignar rigidbody al script



Crear método reset en el script



Crear script para la pelota

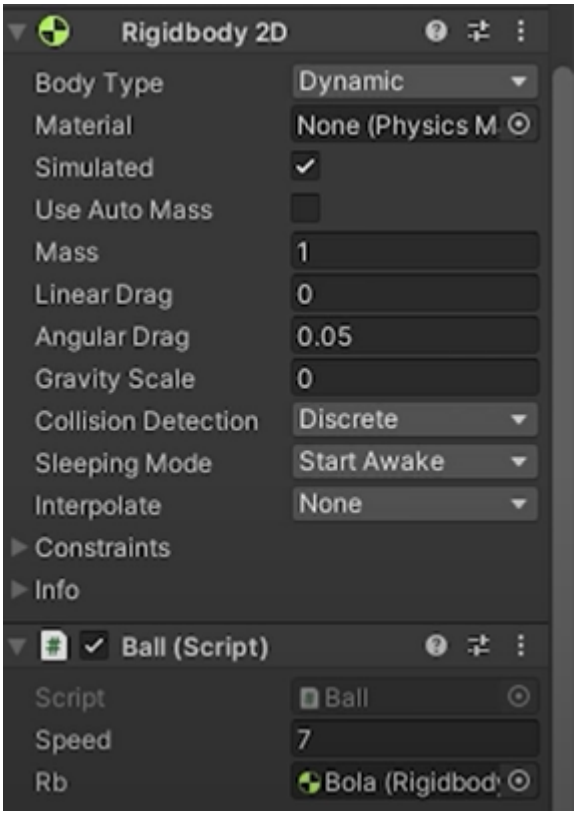
Crearemos un script para la pelota al que llamaremos `ball.cs`.

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

public class Goal : MonoBehaviour
{
    public bool player1Goal;
    public GameObject gameManager;

    private void OnTriggerEnter2D(Collider2D collision)
    {
        if(collision.CompareTag("Ball"))
```

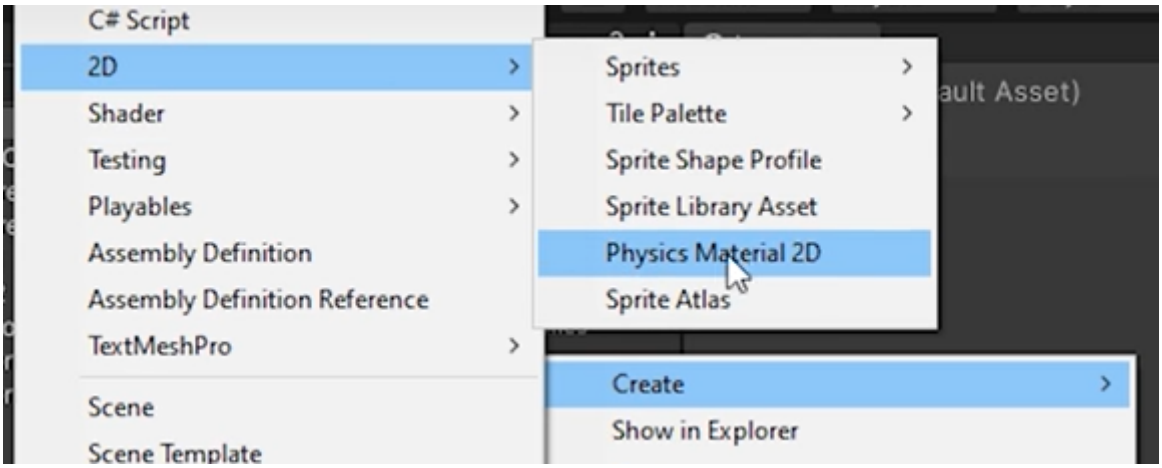
```
        {

            if (player1Goal)
            {
                gameManager.GetComponent<GameManager>().Player1Scored();
            }

            else
            {
                gameManager.GetComponent<GameManager>().Player2Scored();
            }
        }
    }
}
```

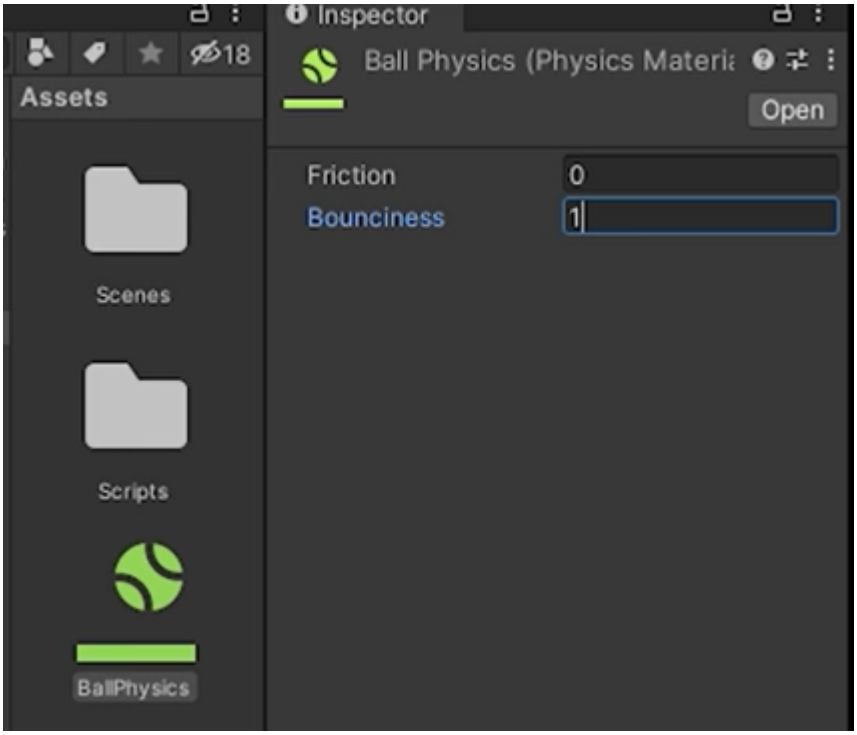
Asignar script a la bola

Asignar el rigibody



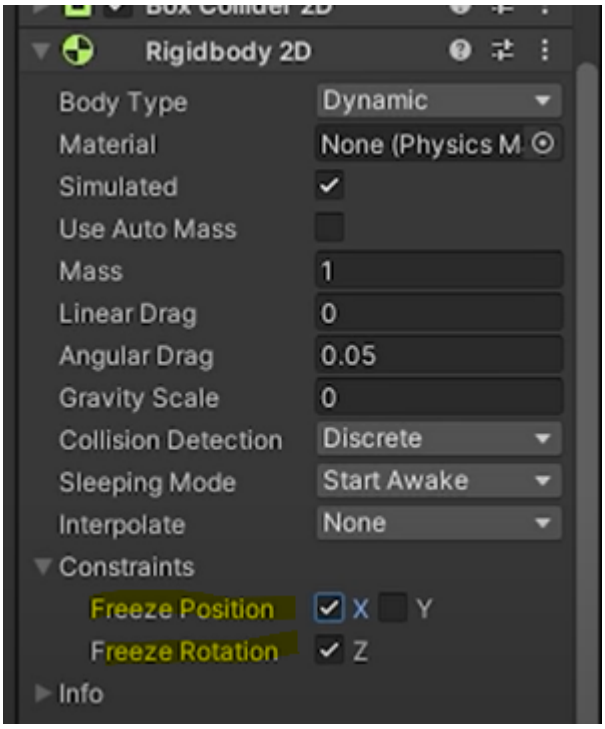
Crear material para la pelota





Restringir movimientos en los players

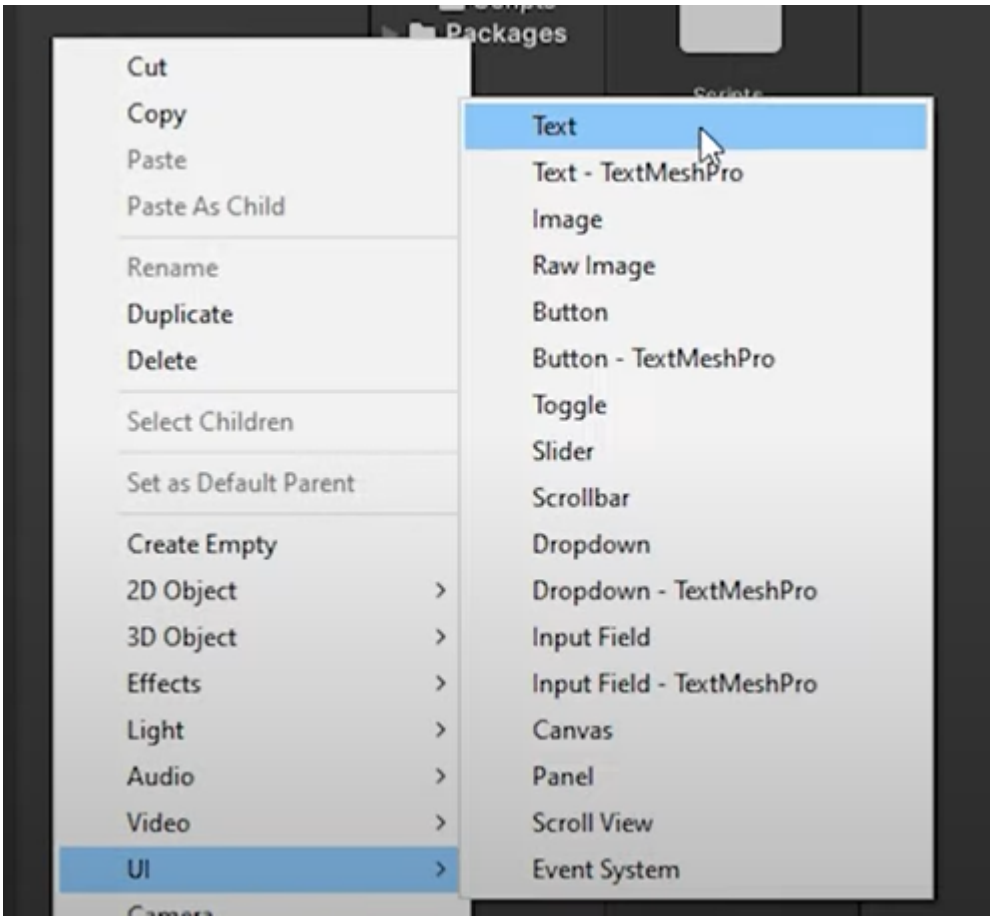
Solo queremos que los jugadores se muevan en un eje, por lo que restringimos los otros dos.



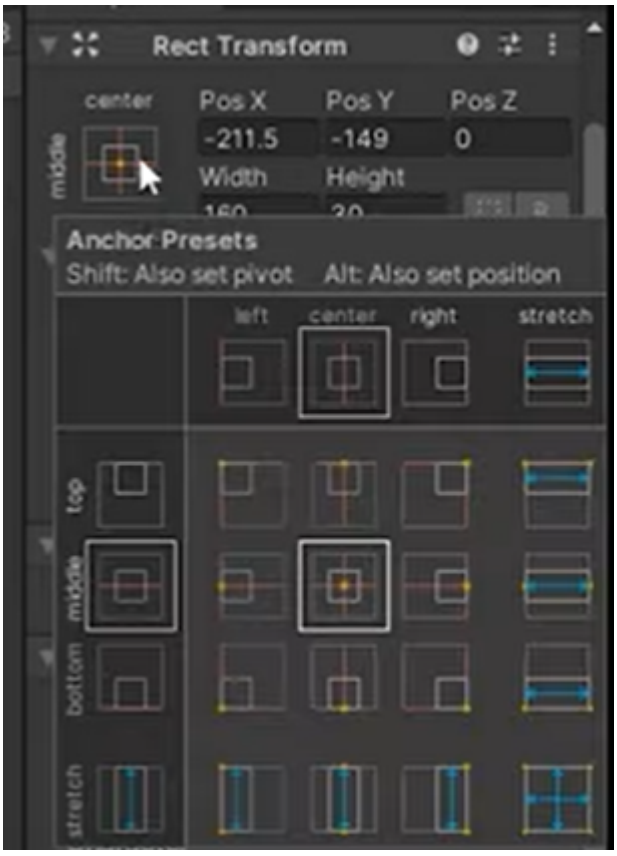
Reorganizar archivos

Crear interfaz puntuación

Crearemos las puntuaciones. Al crear una UI, este objeto se nos pondrá dentro de una carpeta canvas.



Cambiar posición



Una vez creado el texto y colocado, lo duplicaremos para tener dos objetos texto, a los que modificaremos el nombre y se llamarán `Player1Text` y `Player2Text`.

Cambiar pelota y hacerla redonda

En el `Sprite Renderer` tenéis que cambiar la propiedad `Sprite` por un círculo.

Crear el script del juego

Crearemos un `script` que llamaremos `GameManager` (veréis que cambia el icono por un engranaje).

Borraremos los métodos `start()` y `update()`.

Crearemos los métodos `Player1Scored()` y `Player1Scored()`.

Al marcar un gol:

1. La pelota vuelve al centro
2. Las palas vuelven a su posición iniciar
3. Cambiamos los valores del marcador

`ResetPosition()` será un método que resteará los objetos.

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

public class GameManager : MonoBehaviour
{
    public GameObject bola;
    public GameObject player1;
    public GameObject player1goal1;
    public GameObject player2;
    public GameObject player2goal2;

    public Text player1Text;
    public Text player2Text;

    private int player1Score;
    private int player2Score;

    public bool IAGame;

    public int maxScore = 7;

    public void CheckVictory()
    {
        if(player1Score >= maxScore){
            SceneManager.LoadScene("VictoryPlayer1");
        }

        if(player2Score >= maxScore){
            SceneManager.LoadScene("VictoryPlayer2");
```

```
            }
        }

        public void Player1Scored()
        {
            player1Score++;
            player1Text.text = player1Score.ToString();
            CheckVictory();
            ResetPosition();
        }

        public void Player2Scored()
        {
            player2Score++;
            player2Text.text = player2Score.ToString();
            CheckVictory();
            ResetPosition();
        }

        private void ResetPosition()
        {
            if (IAGame)
            {
                bola.GetComponent<Ball>().Reset();;
                player2.GetComponent<Players>().Reset();
            }
            else
            {
                bola.GetComponent<Ball>().Reset();;
                player2.GetComponent<Players>().Reset();
                player1.GetComponent<Players>().Reset();
            }
        }
    }
```

Añadir las referencias a objetos

Arrastraremos todos los objetos a las propiedades del script `GameManager`.

Marcar trigger en las porterías

Necesitamos marcar la opción `is Trigger` del componente `Box Collider 2D`.

Crear script para las porterías

Creamos un script llamado `Goal.cs`. Utilizaremos el método `OnTriggerEnter2D()` para detectar colisión entre la pelota y alguna de las porterías.

`CompareTag` comprobará si el objeto que colisiona es la bola y, en caso de ser así, según si colisiona con `Goal1` o con `Goal2` cambiaremos la puntuación correspondiente.

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

public class Goal : MonoBehaviour
{
    public bool player1Goal;
    public GameObject gameManager;

    private void OnTriggerEnter2D(Collider2D collision)
    {
        if(collision.CompareTag("Ball"))
        {

            if (player1Goal)
            {
                gameManager.GetComponent<GameManager>().Player1Scored();
            }

            else
            {
                gameManager.GetComponent<GameManager>().Player2Scored();
            }
        }
    }
}
```

Asignar scripts

Vamos a asignar el script creado a `Goal1` y a `Goal2` y marcamos en `Goal1` el check player1goal.

Asignar etiqueta a la bola

Necesitamos asignar el `tag` que llamaremos `ball` al objeto pelota, seleccionando en `Tag` y `Add Tag`.

Crear referencias

Nos hemos dejado crear referencias en el script `Goal.cs`. Una vez lo hayamos hecho, arrastramos `GameManager` a las referencias.

Cambiar colores

Utilizar la página `coolors` para elegir paletas.

Inteligencia artificial

Vamos a hacer que un jugador sea controlado por la máquina. Crear el script `IA.cs` y la completamos.

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

```
using UnityEngine;

public class IA : MonoBehaviour
{
    public float speed=3;
    public GameObject ball;
    private Vector2 ballPos;

    void Update()
    {
        Move();
    }

    void Move() {
        ballPos = ball.transform.position;

        if (transform.position.y > ballPos.y)
        {
            transform.position += new Vector3(0, -speed*Time.deltaTime);
        }

        if (transform.position.y < ballPos.y)
        {
            transform.position += new Vector3(0, speed*Time.deltaTime);
        }
    }
}
```

Una vez completado el script.

Asignamos el script `IA.cs` a `Player1` y desamarcamos el checkbox del scripts `Players` para que no interfiera.

Arrastramos la referencia de `Bola` al script.

Crear variable en `GameManager` para decidir si el juego es PvP o PvsPC. Será un booleano. En `ResetPosition()` miraremos este valor para decidir resetear o no.

Seleccionar el objeto `GameManager` y marcar la opción `IA Game`.

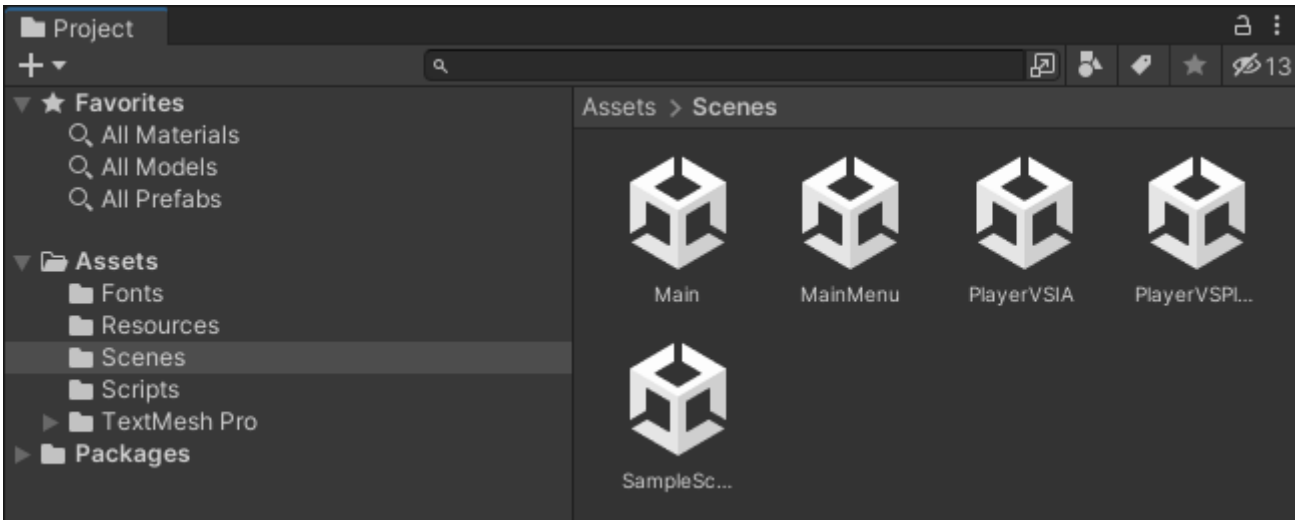
Crear menú

Necesitaremos 3 escenas. Ahora solo tenemos la escena `Main`. La vamos a llamar `PlayerVSIA` para diferenciarla.

La duplicamos y le llamamos `PlayerVSPlayer`. En esta escena, desmarcamos el check de `IA Game`.

Creamos una escena 2D nueva yendo a `File > New scene > 2D`.

Las escenas deberán estar todas dentro de la carpeta `Scenes`. Haciendo clic en cada una de ellas, podremos abrirlas y modificarlas



Crear botones

Crear botón. Clic derecho en `Hierarchy` y `Create > UI > Button`.

El botón se hará grande o pequeño según la resolución y el aspect ratio. Si queremos fijar su tamaño, haremos lo siguiente.

Iremos al objeto `Canvas` en el que se ha creado el botón y en el componente `Canvas Scaler` vamos a la propiedad `UI Scale Mode` y elegimos `Scale With Screen Size`.

Duplicamos el botón y le colocamos el texto `Player VS Player`. Lo movemos y lo situamos.

Colocar texto con nombre del juego

Pondremos un texto y le cambiaremos el texto por PONG, y lo haremos más grande. Para evitar problemas al hacerlo grande o pequeño, vamos al `inspector` y buscamos en paragraph las opciones `horizontal overflow` y `vertical overflow` y les asignamos el valor `overflow`.

Lo hacemos grande y lo situamos.

Guardamos la escena (que ahora se llama `Untitled*`) con +ctrl+ y +s+ y le llamaremos `MainMenu`.

Crear script MainMenu.cs

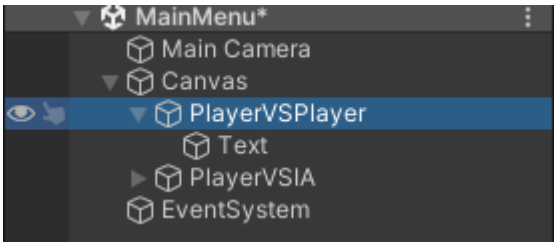
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class MainMenu : MonoBehaviour
{

    void Update()
    {
        if (Input.GetKeyDown(KeyCode.Escape))
        {
            Application.Quit();
```

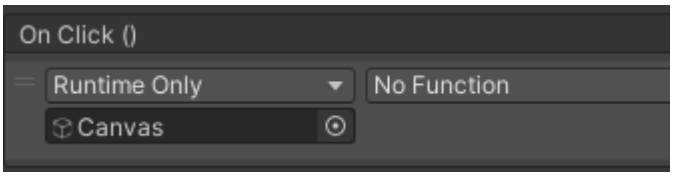
```
        }
    }

    public void PlayerVSIA(){
        SceneManager.LoadScene("PlayerVSIA");
    }

    public void PlayerVSPlayer(){
        SceneManager.LoadScene("PlayerVSPlayer");
    }
}
```

Asignar script al canvas

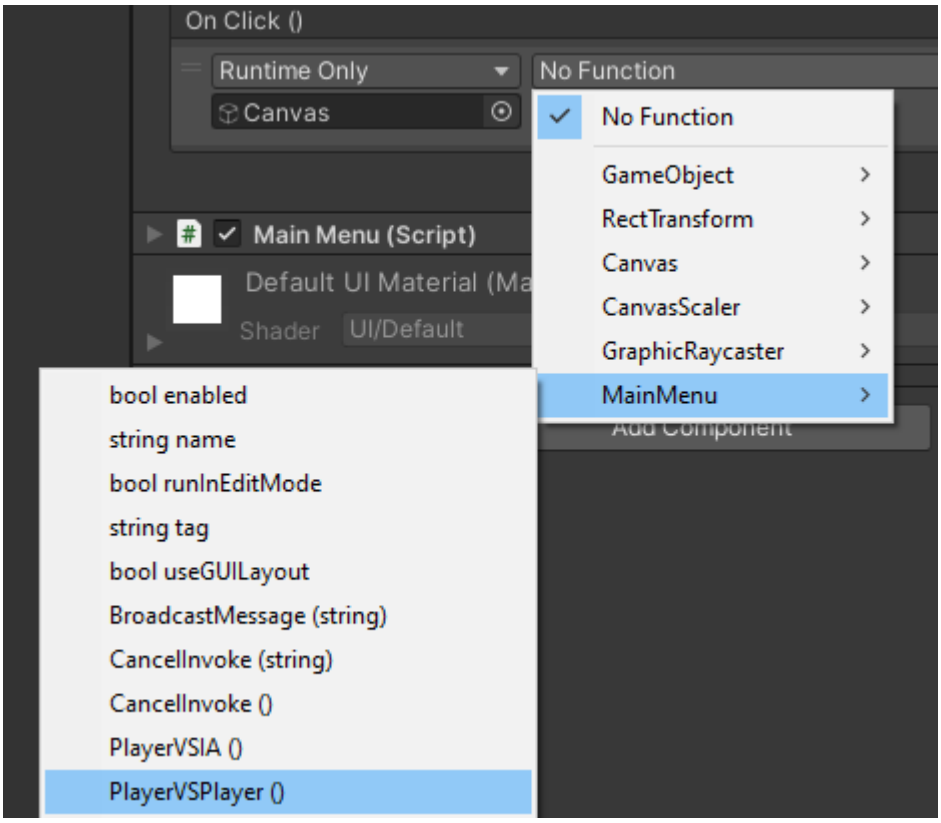
Tendremos que asignar el script `MainMenu.cs` al `canvas` de la escena del menú.

Por último tendremos que arrastrar `canvas` a las referencias de los dos botones, y cambiar el onclick para que llame a las funciones correspondientes.





Elegimos la función correspondiente:

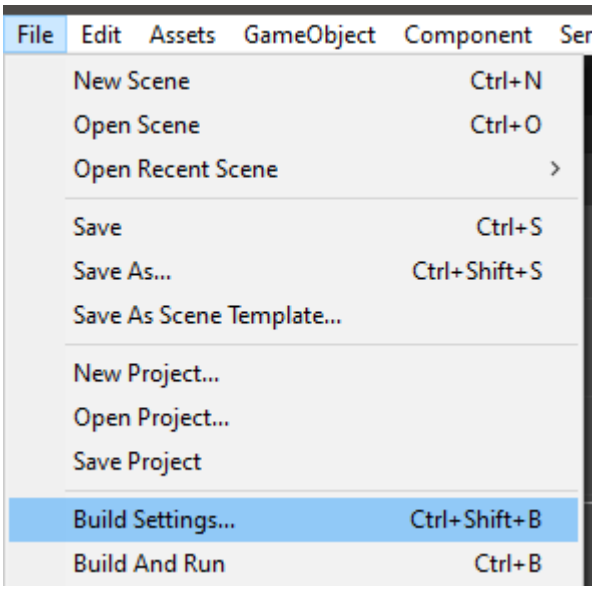


Modificación de build settings

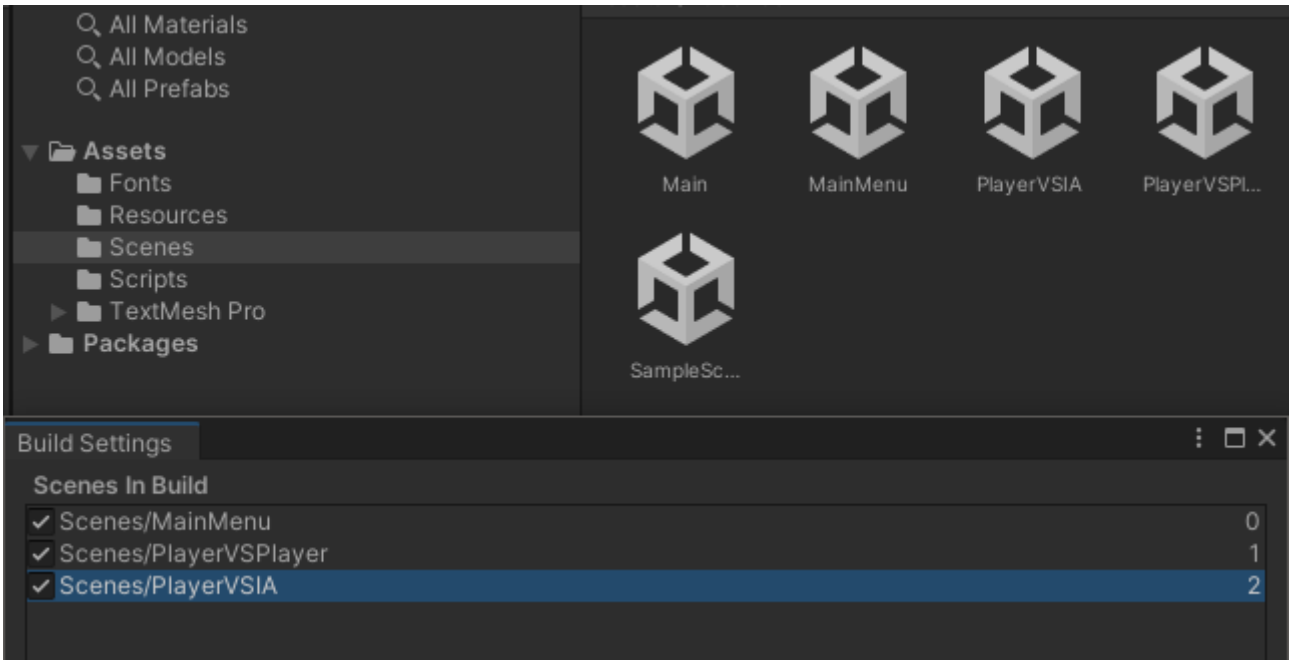
Tenemos que agregar las escenas que formarán parte en el juego en el orden correcto.

Para agregar escenas, sigue estos pasos:

1. Ve al menú `File` (Archivo) y selecciona `Build Settings` (Configuración de compilación).



2. En la ventana Build Settings, verás una lista de escenas. En la parte inferior deberemos colocar las escenas que van a formar parte de nuestra `build`.



Comprobación

Comprobaremos que ejecutando el juego, podemos pasar del menú a cada una de las dos escenas siguientes.