SPRITE FLIGHT

CREACIÓN DE JUEGO SPRITE FLIGHT USANDO UNITY

Jerry Wong Cal

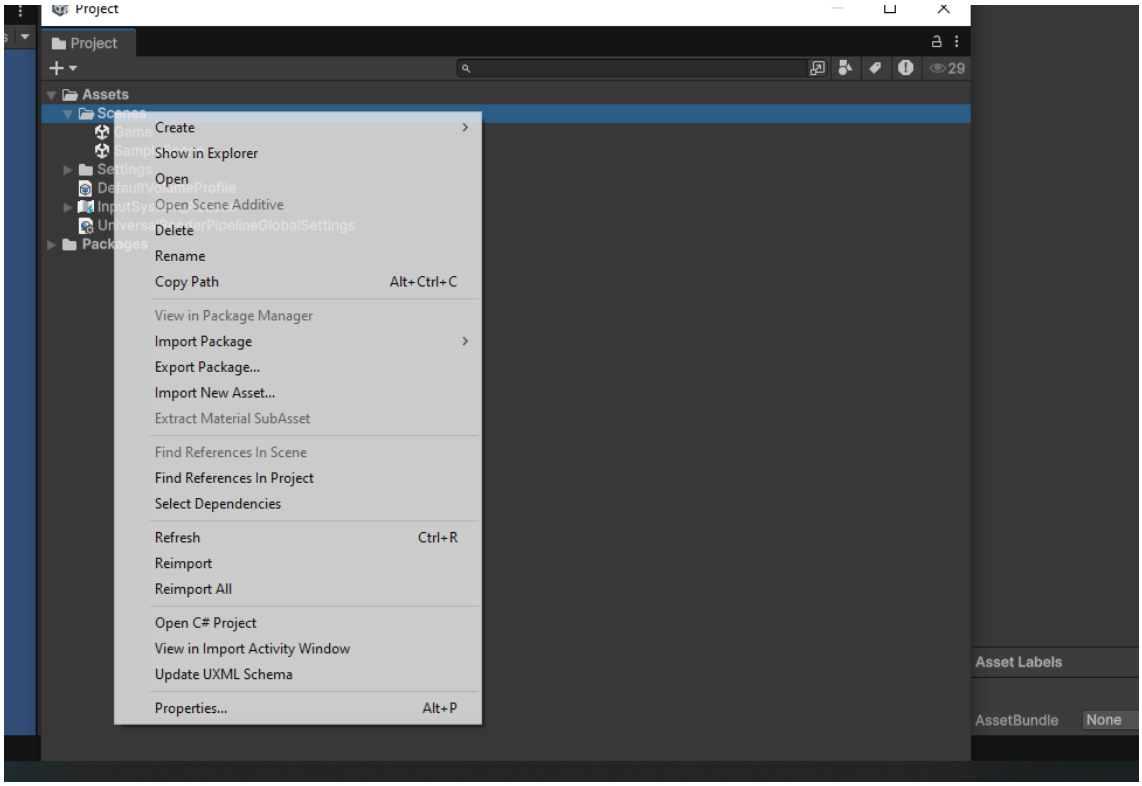
Introducción

En este documento se detalla el proceso de creación del juego Sprite Flight utilizando Unity. El objetivo es desarrollar un juego 2D básico donde aprenderemos a configurar la escena, crear objetos visuales, definir bordes de pantalla y organizar los elementos del proyecto de forma eficiente.

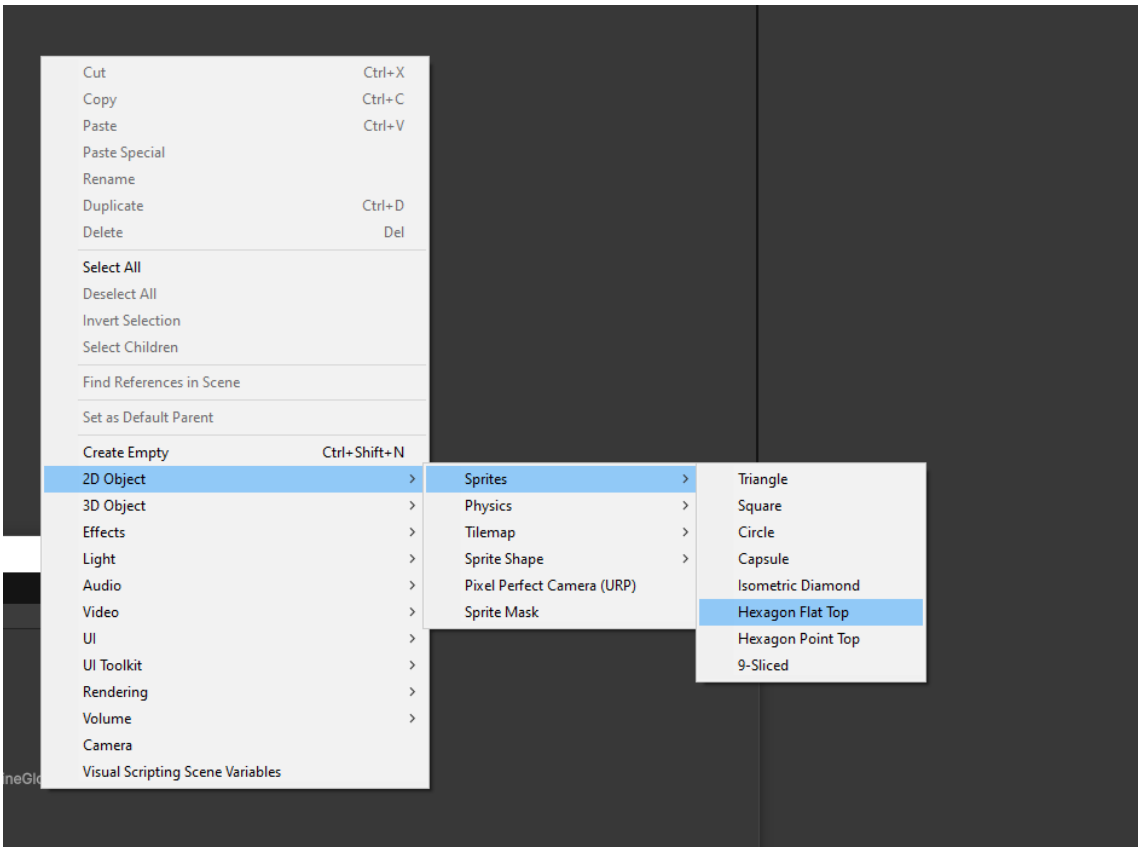
Este tutorial cubre la configuración inicial del entorno de juego, incluyendo la creación de obstáculos, ajuste de cámara, delimitación del área de juego y organización de assets.

GitHub: https://github.com/chinese86/SpriteFlight

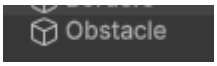
Con el botón derecho sobre Scenes le damos a Create/scene/scene y le ponemos de nombre Game



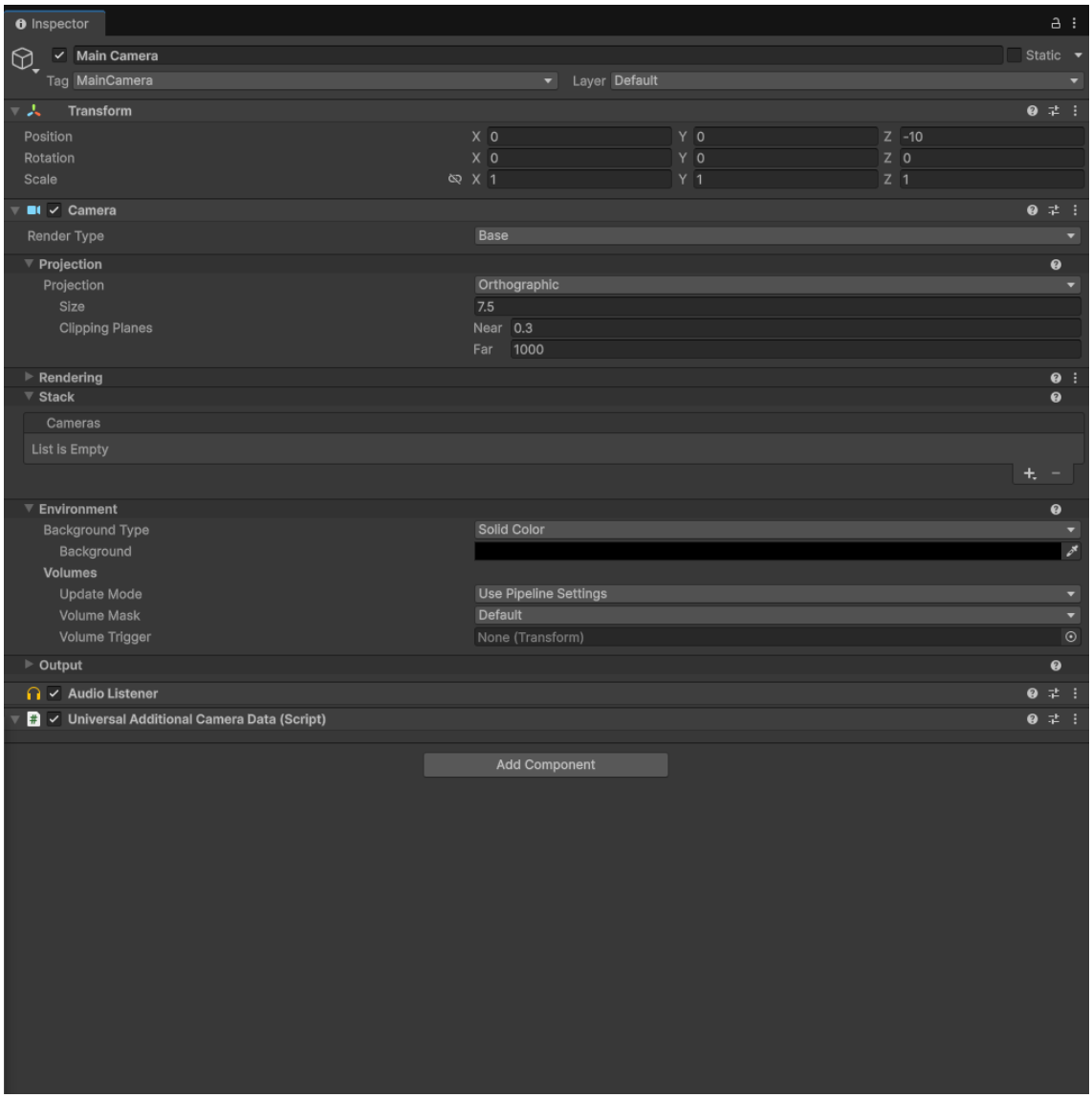
Le damos al botón derecho en Hierarchy y vamos a 2D object/Sprites/Hexagon Flat Top como se muestra en la imagen



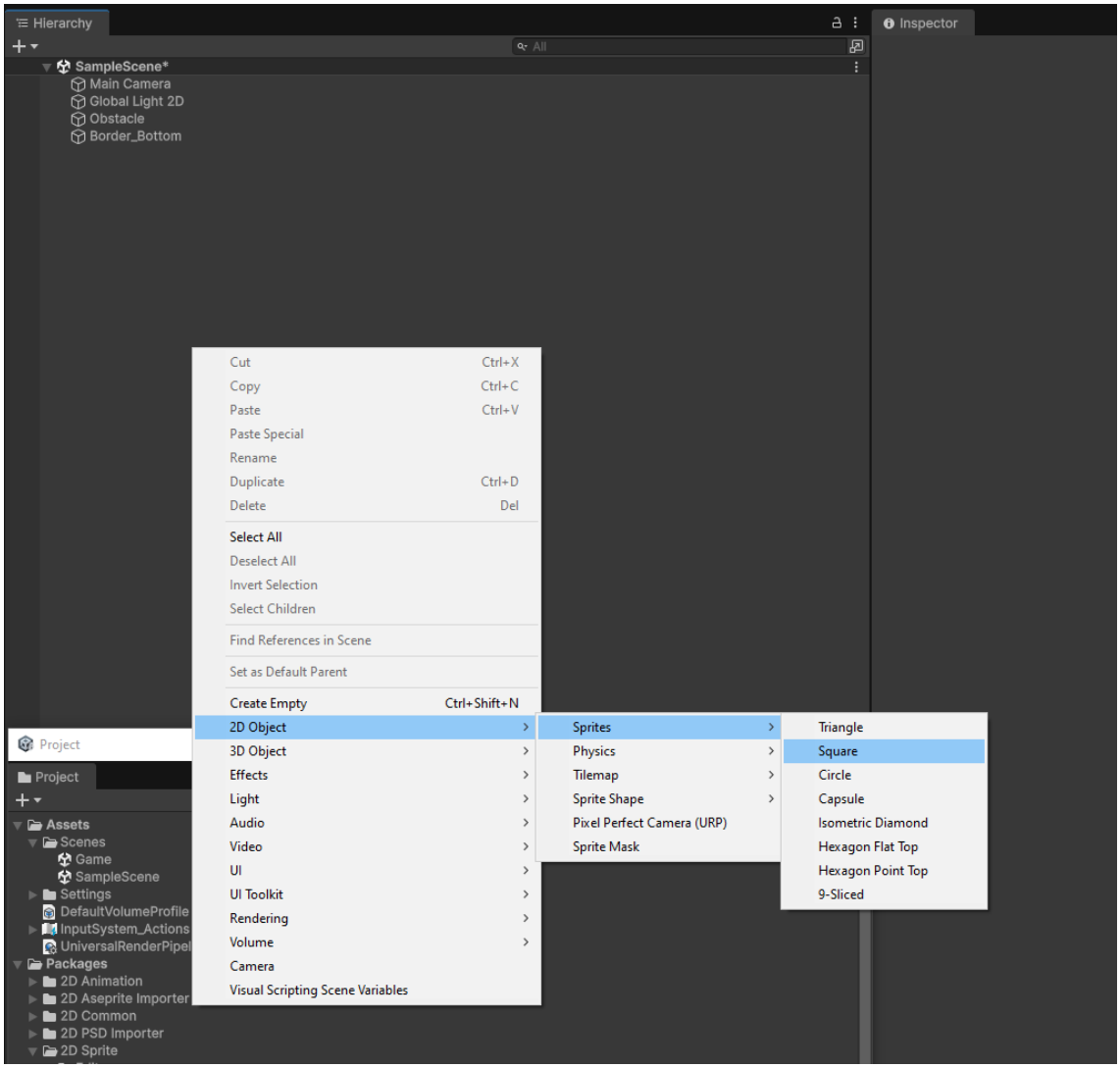
Lo nombramos Obstacle y ponemos los valores a 0 y elegimos un color.



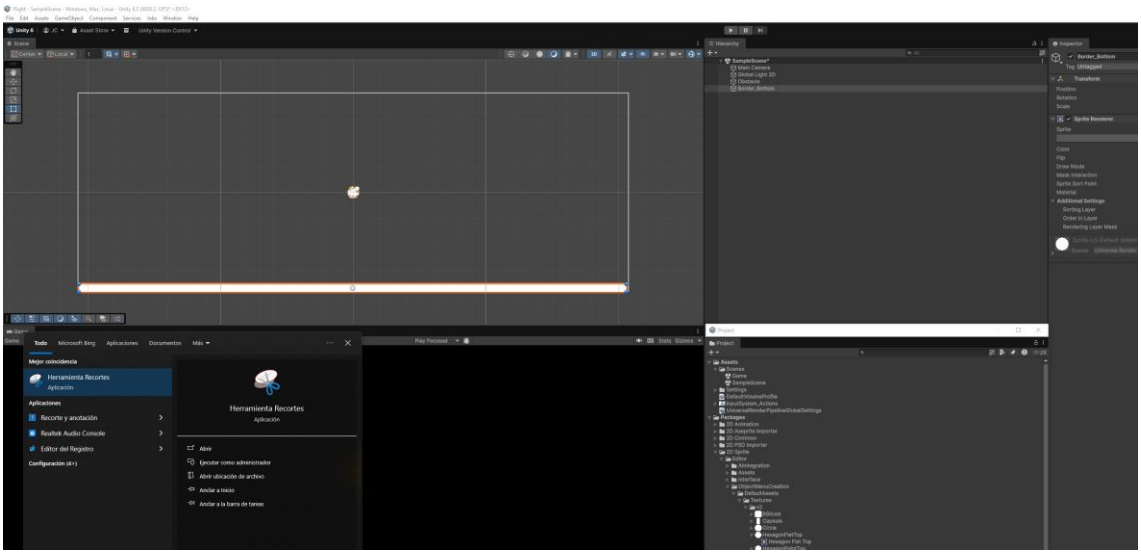
Ahora vamos a cambiar algunos valores de la cámara principal, vamos a main camera y le ponemos en Size 7.5 de tamaño y en Background Type elegimos la opción Solid Color y lo dejamos negro, ya tenemos el fondo de color negro y un hexágono de color anaranjado.



Nuestro siguiente paso es definir los bordes de la pantalla, en este caso, apretamos con el botón derecho y vamos a 2D object/Sprites/Square, al crearlo lo nombramos como Border_Bottom, yo ya lo he hecho así que en la imagen podemos ver al fondo ya el objeto creado con ese nombre.

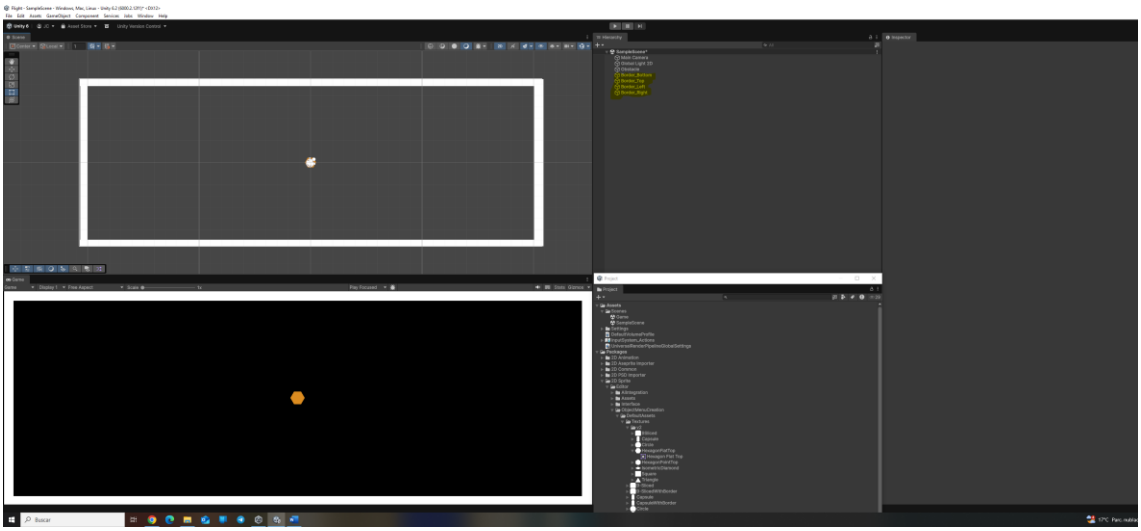


En la vista general a la izquierda vemos que redimensiono el rectángulo para que sea el borde inferior de la pantalla.

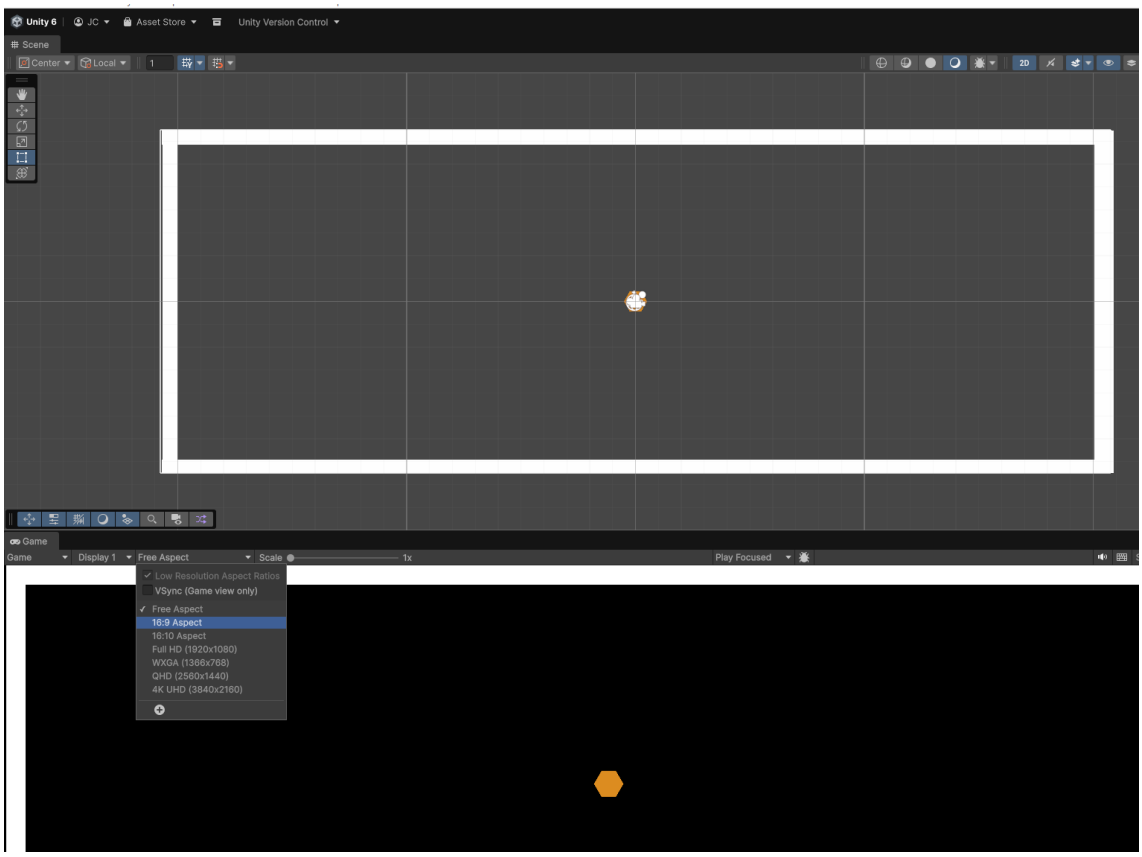


Ahora duplicamos el Borde que tenemos (Botón derecho sobre Border_Bottom y Duplicar) y los llevamos a todo el borde del background formando el área que queremos delimitar, estos

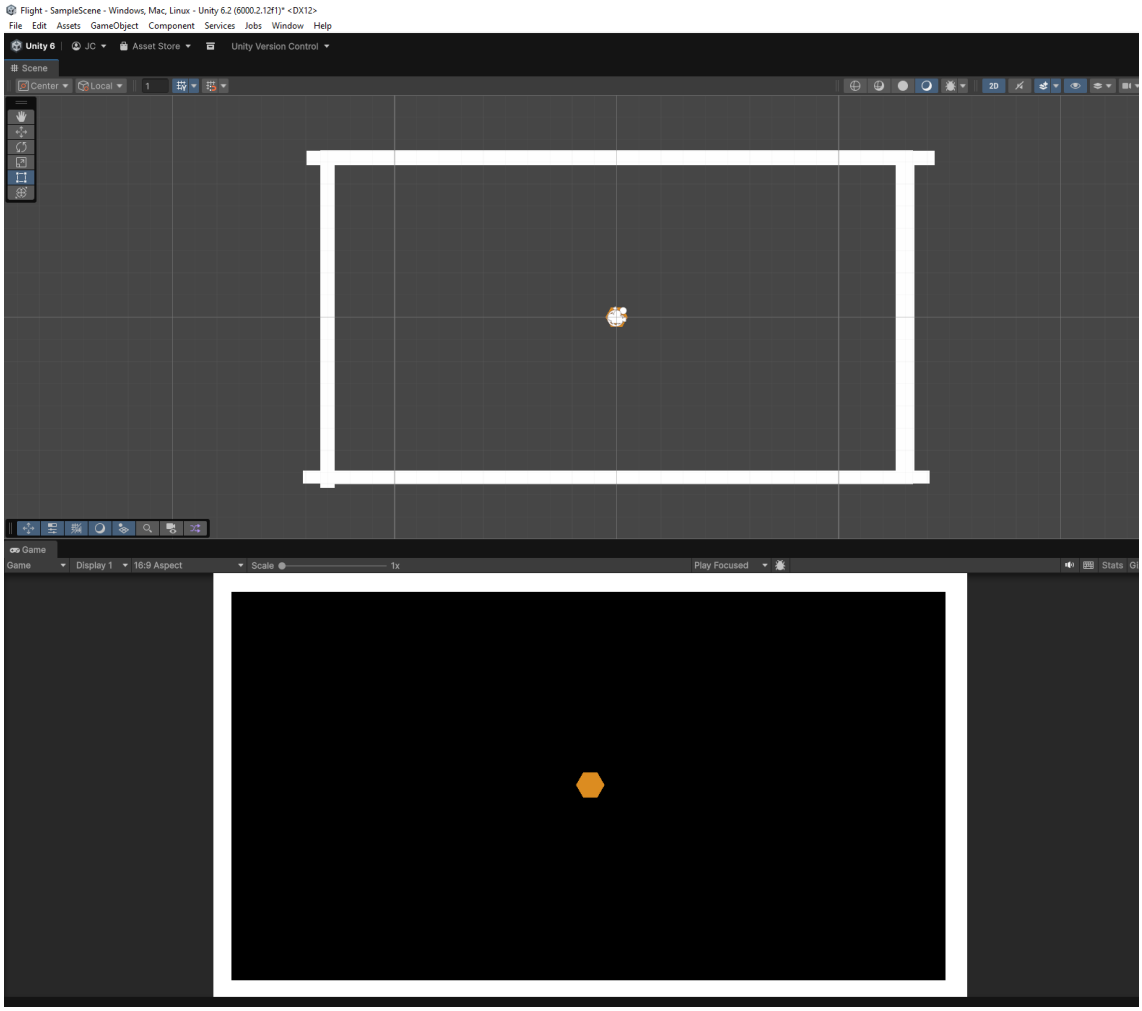
serán los límites del juego, luego los renombramos como Border_Left, Border_Top y Border_Right los colocamos y listo.



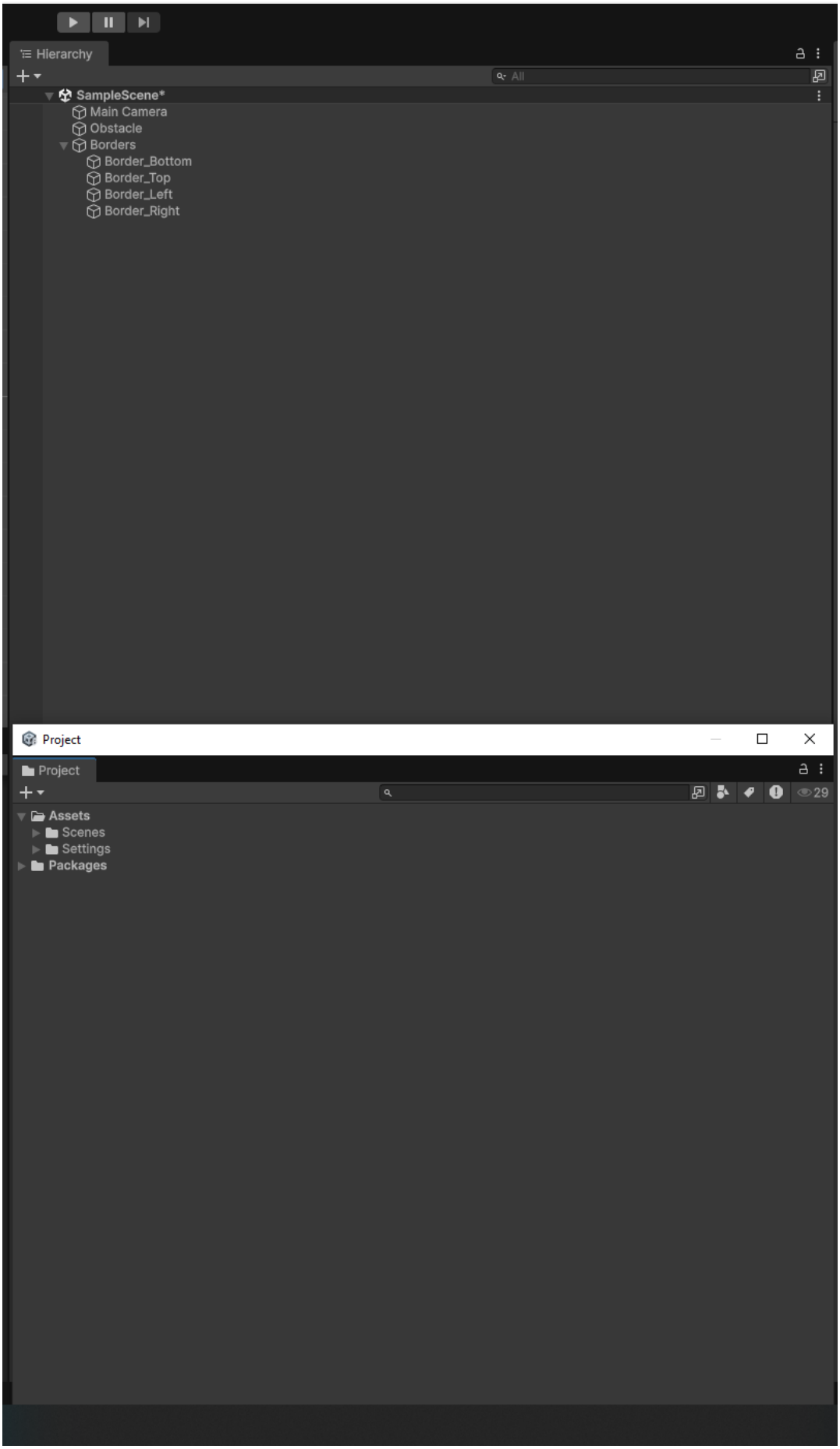
Para asegurarnos de que el juego siempre se ajuste a la pantalla como se desea, vamos a configurar la vista de juego en un "aspect ratio" fijo. En nuestro caso configuramos un "aspect ratio" fijo de 16:9, que es un formato común en teléfonos y monitores.



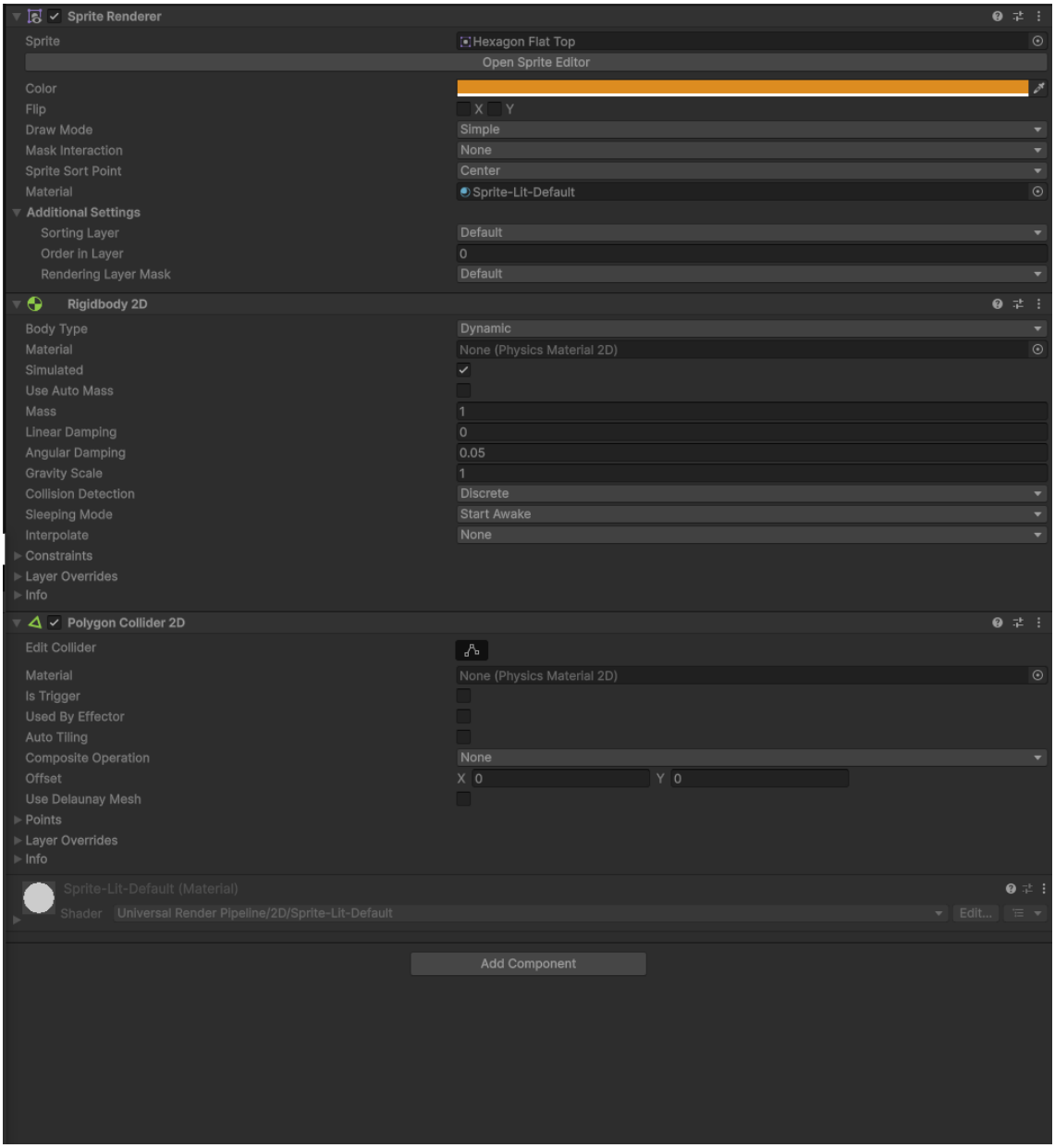
Aquí ya vemos los bordes para un ratio fijo de 16:9. Arriba es donde configuramos y abajo tenemos una vista previa de lo que vamos a ver.



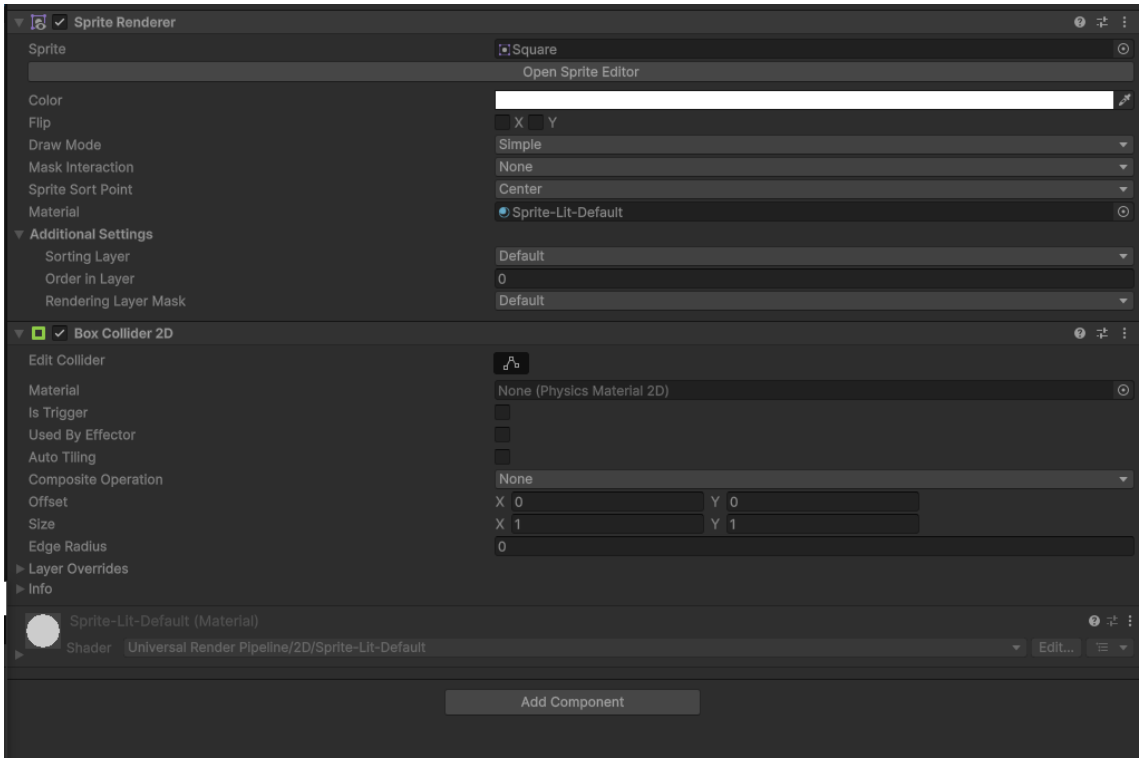
El siguiente paso es solo un paso que haremos para ordenar un poco y tener todo mas claro. Con el botón derecho en la parte superior en el Hierarchy hacemos un Create Empty y le llamamos Borders y ahí metermos todos los bordes para tenerlos juntos, abajo metemos los archivos que tenemos sueltos en Settings.


Hierarchy
All
SampleScene*
Main Camera
Obstacle
Borders
Border_Bottom
Border_Top
Border_Left
Border_Right
Project
Project
29
Assets
Scenes
Settings
Packages

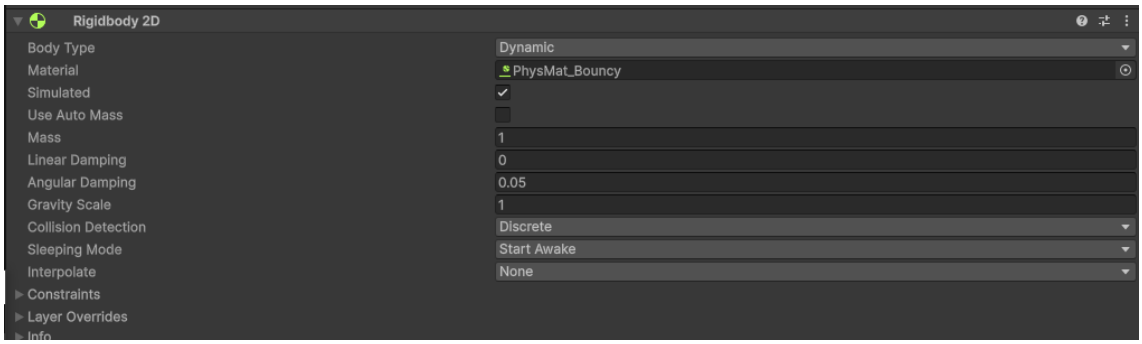
Continuaremos creando un poco de física a nuestro obstáculo para que caiga y no se salga de los ímites. Vamos a seleccionar Obstacle y en el Inspector vamos a Add Component/Rigid Body 2D. ahora vamos a añadir otro componente Add Component/ Polygon Colider 2D. Ahora cuando le damos al play, nuestro obstáculo cae por la gravedad pero choca con el borde inferior.



Y a los Borders vamos a añadir Componente Add Component/Box Colider 2D



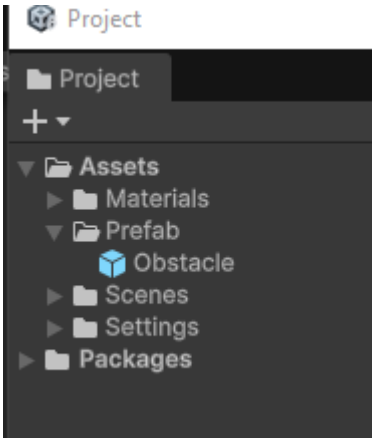
Ya el obstáculo cae y no se sale del límite pero vamos a hacer que tenga rebote, para ello vamos a Project y con el botón derecho del ratón creamos una carpeta, para ello botón derecho del ratón/Create/Folder y le llamamos Materials ahora dentro de esta carpeta botón derecho del ratón/Create/2D/Physics Material 2D, en Bounciness ponemos valor 1. Seleccionamos Obstacle y en el Inspector en Material le añadimos el material que acabamos de crear.



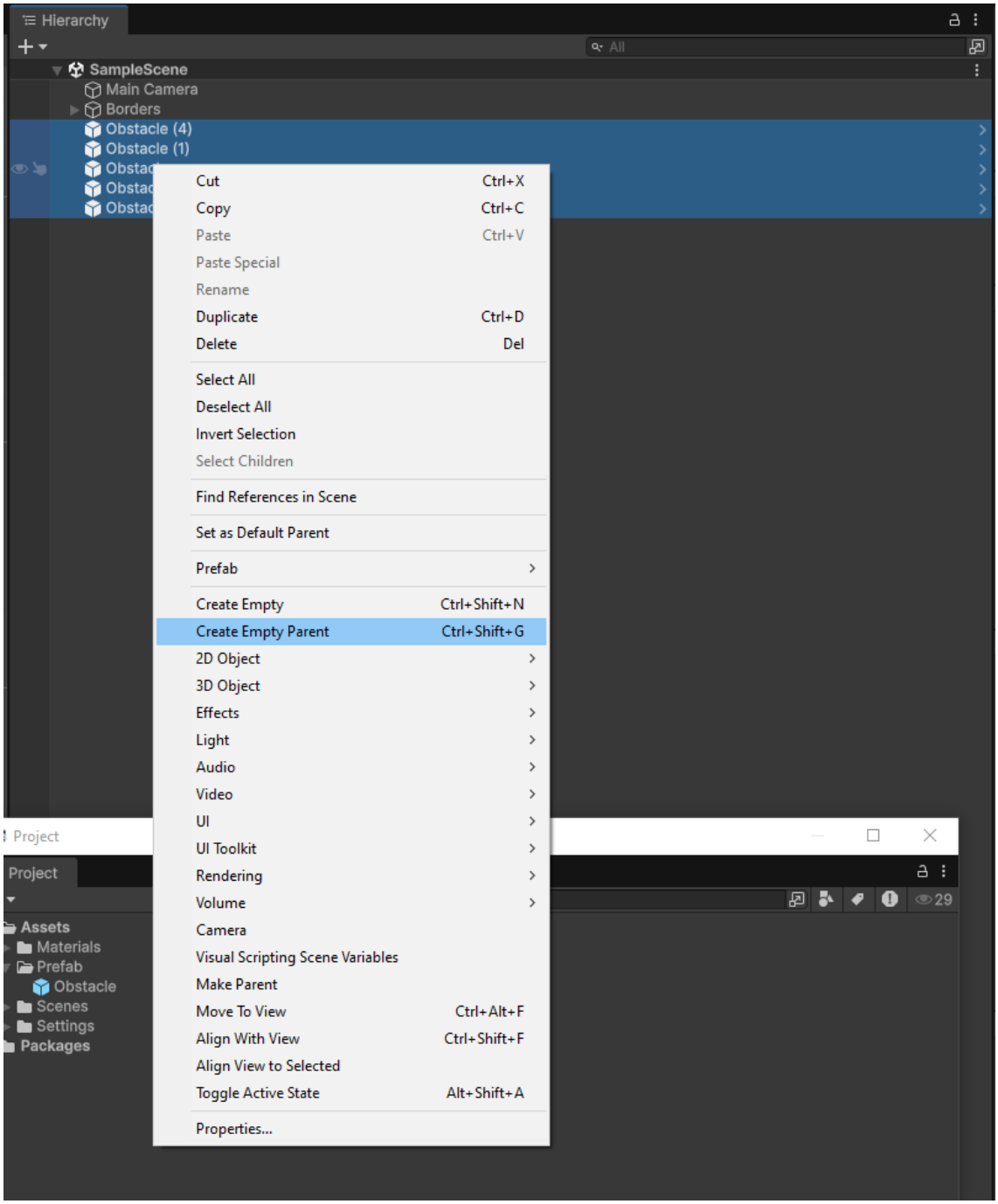
Convierte el obstáculo en un prefab:
Arrastra el GameObject Obstacle desde la ventana Hierarchy hasta la nueva carpeta Prefabs. Observa que el icono del GameObject Obstacle en la ventana Hierarchy se vuelve azul — esto significa que ahora es una instancia de un prefab.

Prueba con algunos prefabs más:
Desde la ventana Proyecto, arrastra el prefab Obstacle a la vista de escena varias veces. Distribuye los prefabs Obstacle y rota algunos usando la herramienta de rotar (pulsa E). Selecciona el botón Play y observa cómo rebotan independientemente.

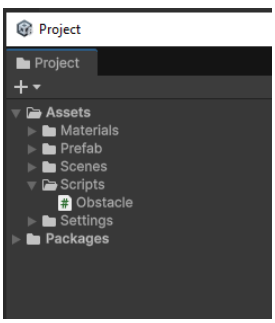
Ahora ya has creado un prefab reutilizable. Siempre que quieras hacer cambios a todos los obstáculos en tu escena, solo tienes que editar el prefab fuente en la ventana Proyecto.]



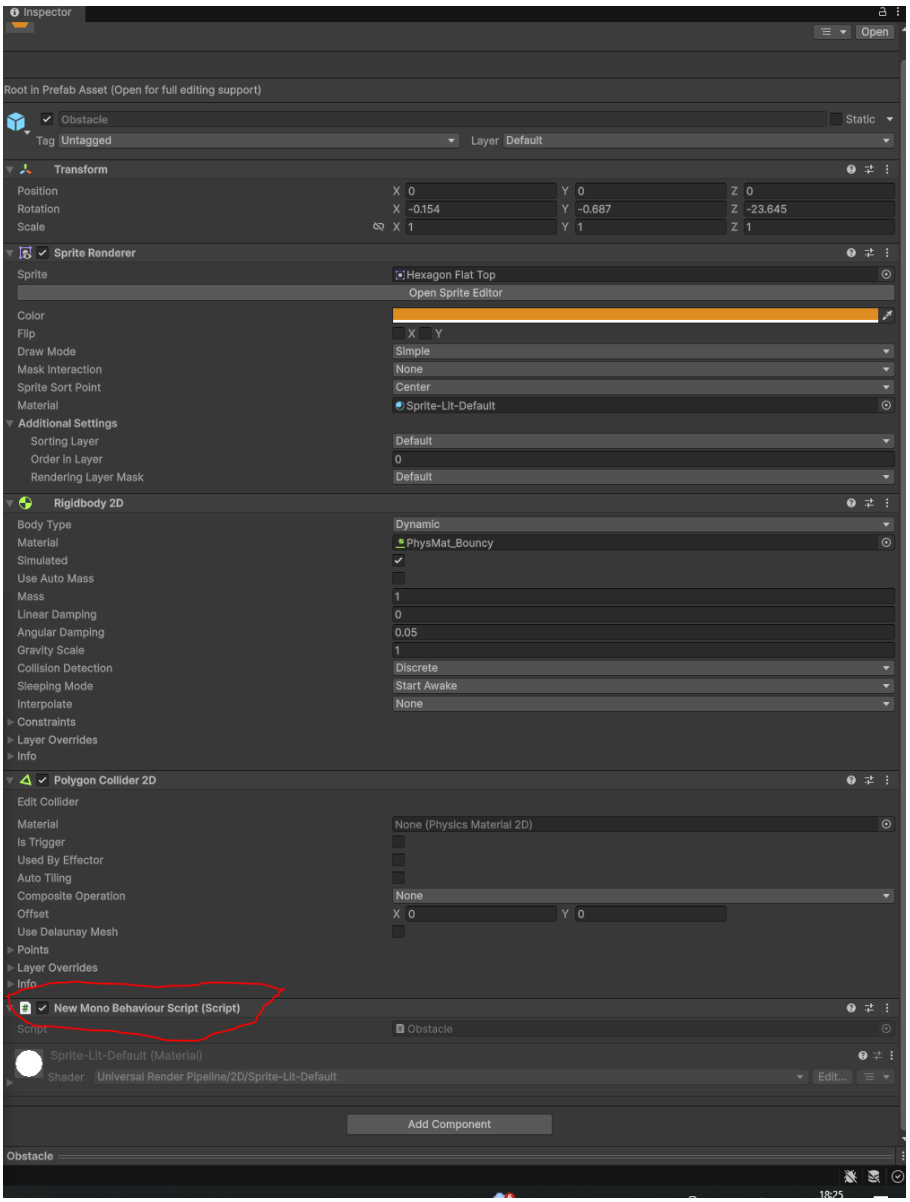
En Hierarchy vemos que tenemos creados varios obstacles, pues vamos a seleccionarlos todos y con el botón derecho vamos a Create Empty parent y lo llamamos Obstacles, con esto dejamos todos nuestros obstacles dentro de un elemento padre.



Vamos a Assets y con el botón derecho/Create/Folder crearemos una carpeta que vamos a llamar Scripts ahora en la carpeta Scripts vamos a botón derecho/Create/MonoBehaviour Script.



Ahora dentro de la carpeta Prefab seleccionamos Obstacle y vamos al Inspector y vamos a Add Component y seleccionamos el Script que acabamos de crear llamado Obstacle, ya con este paso al haberlo añadido en la carpeta Prefab el cambio se va a realizar en todos los Obstacles que tenemos.



Ahora vamos al Script que hemos creado llamado Obstacle y añadimos el siguiente código:

```
using UnityEngine;

public class NewMonoBehaviourScript : MonoBehaviour
{
    // Start is called once before the first execution of Update after
the MonoBehaviour is created
    void Start()
    {
        transform.localScale = new Vector3(2, 2, 1);
    }

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

    }
}
```

Los valores X y Y controlan el ancho y la altura.
El valor Z es obligatorio, aunque no afecte a los gráficos en 2D.
El signo = aquí no significa "es igual que", como en matemáticas; significa que le estás asignando a la variable de la izquierda el valor de la derecha.

Procedemos a guardar y salimos.

Ahora vamos a volver al código porque despúes de los cambios anteriores todos los obstáculos se ponen del mismo tamaño y queremos que los tamaños sean random (X e Y).

```
using UnityEngine;

public class NewMonoBehaviourScript : MonoBehaviour
{
    // Start is called once before the first execution of Update after
the MonoBehaviour is created
    void Start()
    {
        float randomSize = Random.Range(0.5f, 2.0f);
        transform.localScale = new Vector3(randomSize, randomSize, 1);
    }

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

```
    }
}
```

Ahora vamos a crear unas variables, es muy útil porque así podemos usar esas variables en distintos métodos siempre que los necesitemos

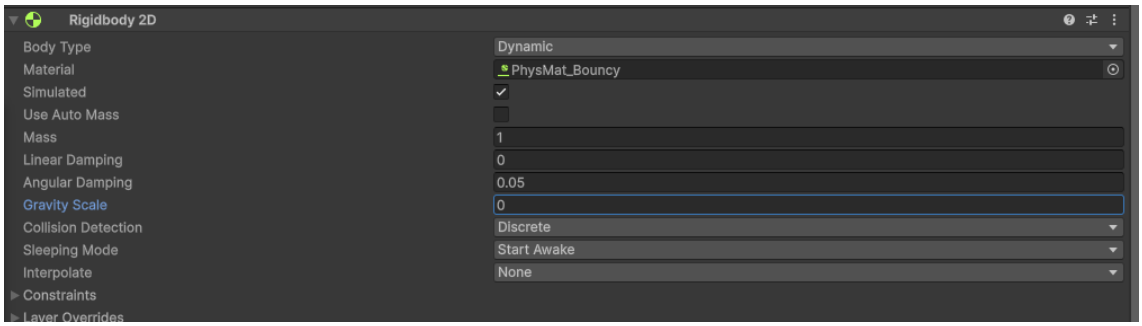
```
using UnityEngine;

public class NewMonoBehaviourScript : MonoBehaviour
{
    float minSize = 0.5f;
    float maxSize = 2.0f;
    // Start is called once before the first execution of Update after
the MonoBehaviour is created
    void Start()
    {
        float randomSize = Random.Range(minSize, maxSize);
        transform.localScale = new Vector3(randomSize, randomSize, 1);
    }

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

    }
}
```

Volvemos al código porque hasta ahora los obstáculos caen hacia abajo por la fuerza de la gravedad pero no tienen movimientos aleatorios complejos, vamos a cambiar esto haciendo que puedan moverse en diferentes direcciones.

```
using UnityEngine;

public class NewMonoBehaviourScript : MonoBehaviour
{
    float minSize = 0.5f;
    float maxSize = 2.0f;
    Rigidbody2D rb;
    // Start is called once before the first execution of Update after
the MonoBehaviour is created
    void Start()
    {
        float randomSize = Random.Range(minSize, maxSize);
        transform.localScale = new Vector3(randomSize, randomSize, 1);

        rb = GetComponent<Rigidbody2D>();
        rb.AddForce(Vector2.right * 100);
```

```
    }

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

    }
}
```

Vamos a quitar la gravedad, para ello vamos a la carpeta Prefab, seleccionamos Obstacle y dentro del Rigidbody 2D vamos a gravity Scale y ponemos 0



Ahora vamos a hacer que la velocidad de los obstáculos sea random, así cuando juguemos varias partidas no podremos saber cómo se van a comportar los obstáculos, queremos generar libre albedrío de los obstáculos.

```
using UnityEngine;

public class NewMonoBehaviourScript : MonoBehaviour
{
    float minSize = 0.5f;
    float maxSize = 2.0f;
    float minSpeed = 50f;
    float maxSpeed = 150f;
    Rigidbody2D rb;
    // Start is called once before the first execution of Update after
the MonoBehaviour is created
    void Start()
    {
        float randomSize = Random.Range(minSize, maxSize);
        transform.localScale = new Vector3(randomSize, randomSize, 1);

        rb = GetComponent<Rigidbody2D>();
        float randomSpeed = Random.Range(minSpeed, maxSpeed);
        rb.AddForce(Vector2.right * randomSpeed);
    }
```

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

    }
}
```

Ahora lo que vamos a randomizar es la dirección a la que van los obstáculos, hasta el momento todos iban hacia la derecha (Vector2.right), ahora pondremos randomDirection.

```
using UnityEngine;

public class NewMonoBehaviourScript : MonoBehaviour
{
    float minSize = 0.5f;
    float maxSize = 2.0f;
    float minSpeed = 50f;
    float maxSpeed = 150f;
    Rigidbody2D rb;
    // Start is called once before the first execution of Update after
the MonoBehaviour is created
    void Start()
    {
        float randomSize = Random.Range(minSize, maxSize);
        transform.localScale = new Vector3(randomSize, randomSize, 1);

        rb = GetComponent<Rigidbody2D>();
        float randomSpeed = Random.Range(minSpeed, maxSpeed);
        Vector2 randomDirection = Random.insideUnitCircle;
        rb.AddForce(randomDirection * randomSpeed);
    }

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

    }
}
```

Vamos a crear a nuestro Player en la ventana Hierarchy, haz clic derecho sobre el GameObject Player y selecciona Objeto 2D > Sprites > Triángulo.

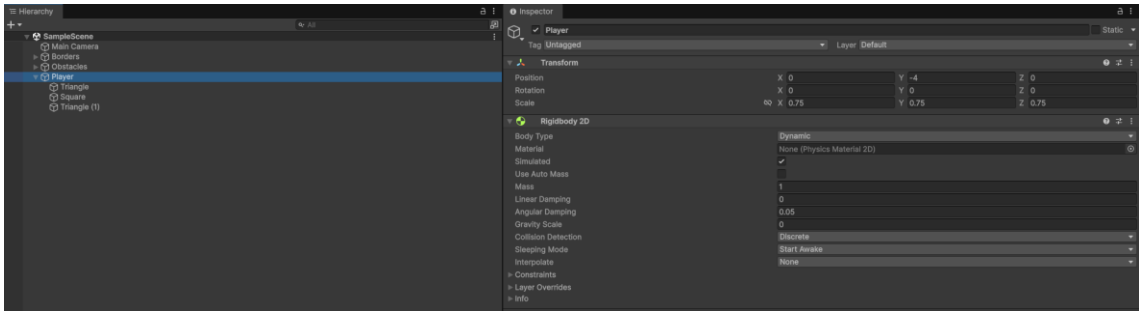
haz clic derecho sobre el GameObject Player y selecciona Objeto 2D > Sprites > Cuadrado.

haz clic derecho sobre el GameObject Player y selecciona Objeto 2D > Sprites > Triángulo.

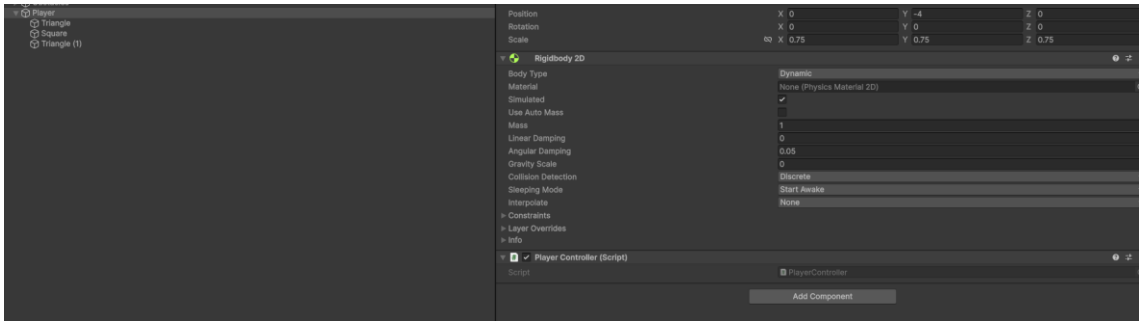
Los juntamos todos para darle forma de cohete y en scala del Player ponemos 0.75 en todo

Seleccionamos Player y vamos a add Component y le ponemos un RigidBody 2D. Gravity Scale 0

Luego seleccionamos las 3 figuras geométricas pulsando control que hemos creado, vamos a Add Component y le ponemos Polygon Collider 2D



Vamos a la carpeta Scripts y creamos un script llamado PlayerController, ahora a player le añadimos el componente script



Ahora vamos a generar el código para dale movimiento al player obedeciendo los clicks que hagamos con el ratón, al chocar contra un obstáculo desaparece y termina el juego.

```
using UnityEngine;
using UnityEngine.InputSystem;

public class PlayerController : MonoBehaviour
{
    public float thrustForce = 1f;
    Rigidbody2D rb;

    void Start()
    {
        rb = GetComponent<Rigidbody2D>();
    }
```

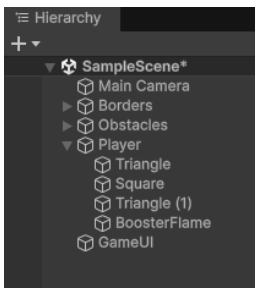
```
    void Update()
    {
        if (Mouse.current.leftButton.isPressed)
        {
            // Calculate mouse direction
            Vector3 mousePos =
Camera.main.ScreenToWorldPoint(Mouse.current.position.value);
            Vector2 direction = (mousePos -
transform.position).normalized;

            // Move player in direction of mouse
            transform.up = direction;
            rb.AddForce(direction * thrustForce);
        }
    }

    void OnCollisionEnter2D(Collision2D collision)
    {
        Destroy(gameObject);
    }
}
```

Vamos al código de nuevo para crear un marcador.

```
using UnityEngine;
using UnityEngine.InputSystem;

public class PlayerController : MonoBehaviour
{
    public float thrustForce = 1f;
    public float maxSpeed = 5f;
    private float elapsedTime = 0f;
    public GameObject boosterFlame;
    private float score = 0f;
    public float scoreMultiplier = 10f;
    Rigidbody2D rb;

    void Start()
    {
        rb = GetComponent<Rigidbody2D>();
    }

    void Update()
    {
        elapsedTime += Time.deltaTime;
        score = Mathf.FloorToInt(elapsedTime * scoreMultiplier);
```

```
        Debug.Log("Score: " + score);
        // Booster visual ON/OFF según el ratón
        if (Mouse.current.leftButton.wasPressedThisFrame)
        {
            boosterFlame.SetActive(true);
        }
        else if (Mouse.current.leftButton.wasReleasedThisFrame)
        {
            boosterFlame.SetActive(false);
        }

        // Movimiento hacia el mouse
        if (Mouse.current.leftButton.isPressed)
        {
            Vector3 mousePos =
Camera.main.ScreenToWorldPoint(Mouse.current.position.value);
            Vector2 direction = (mousePos -
transform.position).normalized;

            transform.up = direction;
            rb.AddForce(direction * thrustForce);

            // Limitar velocidad máxima
            if (rb.linearVelocity.magnitude > maxSpeed)
            {
                rb.linearVelocity = rb.linearVelocity.normalized *
maxSpeed;
            }
        }
    }

    void OnCollisionEnter2D(Collision2D collision)
    {
        Destroy(gameObject);
    }
}
```

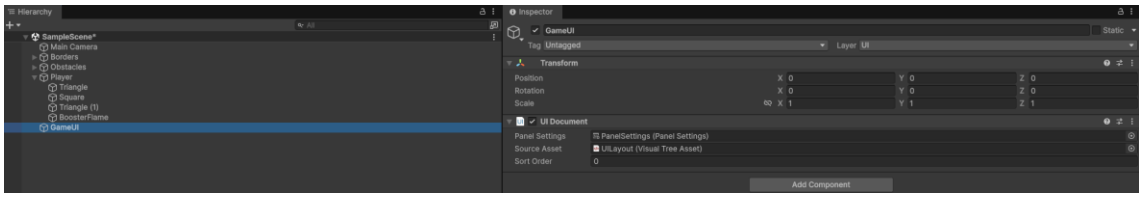
Ahora vamos a Hierarchy/Botón derecho/UI Toolkit/UIDocument le llamaremos GameUI



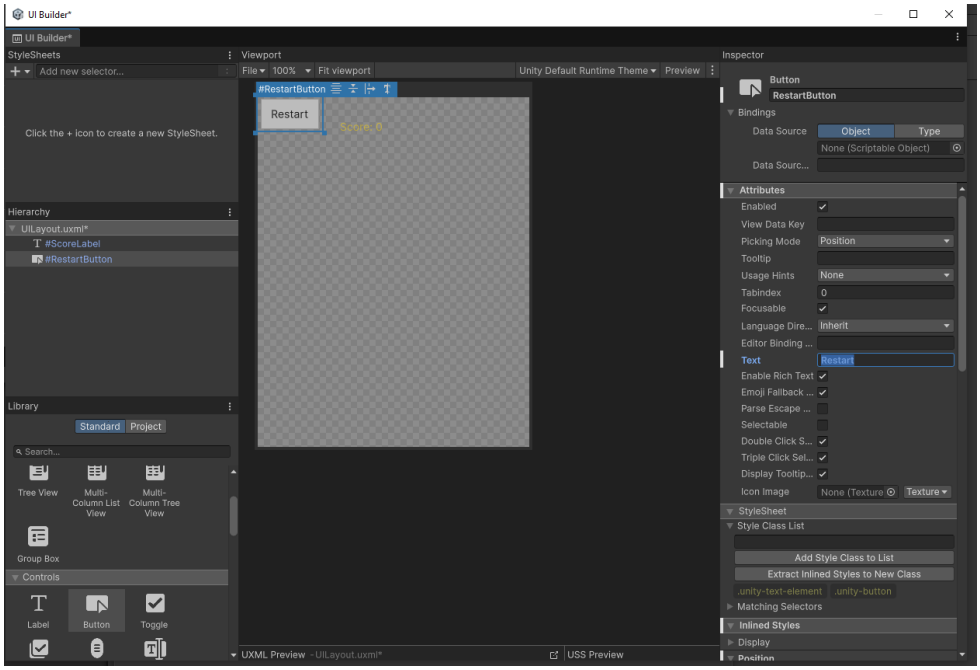
Y ahora en la Ventana Project/Botón derecho/create/Folder y le llamamos “UI”.

Botón derecho directamente sobre la carpeta UI/Create/UI Toolkit/UI Document, le llamaremos UILayout.

Seleccionamos en Hierarchy GameUI y arrastramos de Project el elemento UIlayout que hemos creado en donde pone Source asset.



Abrimos el UILayout y creamos un botón de score para medir la puntuación y que sea visual para el jugador y un botón de restart para que pueda reinciar el juego cuando éste termine.



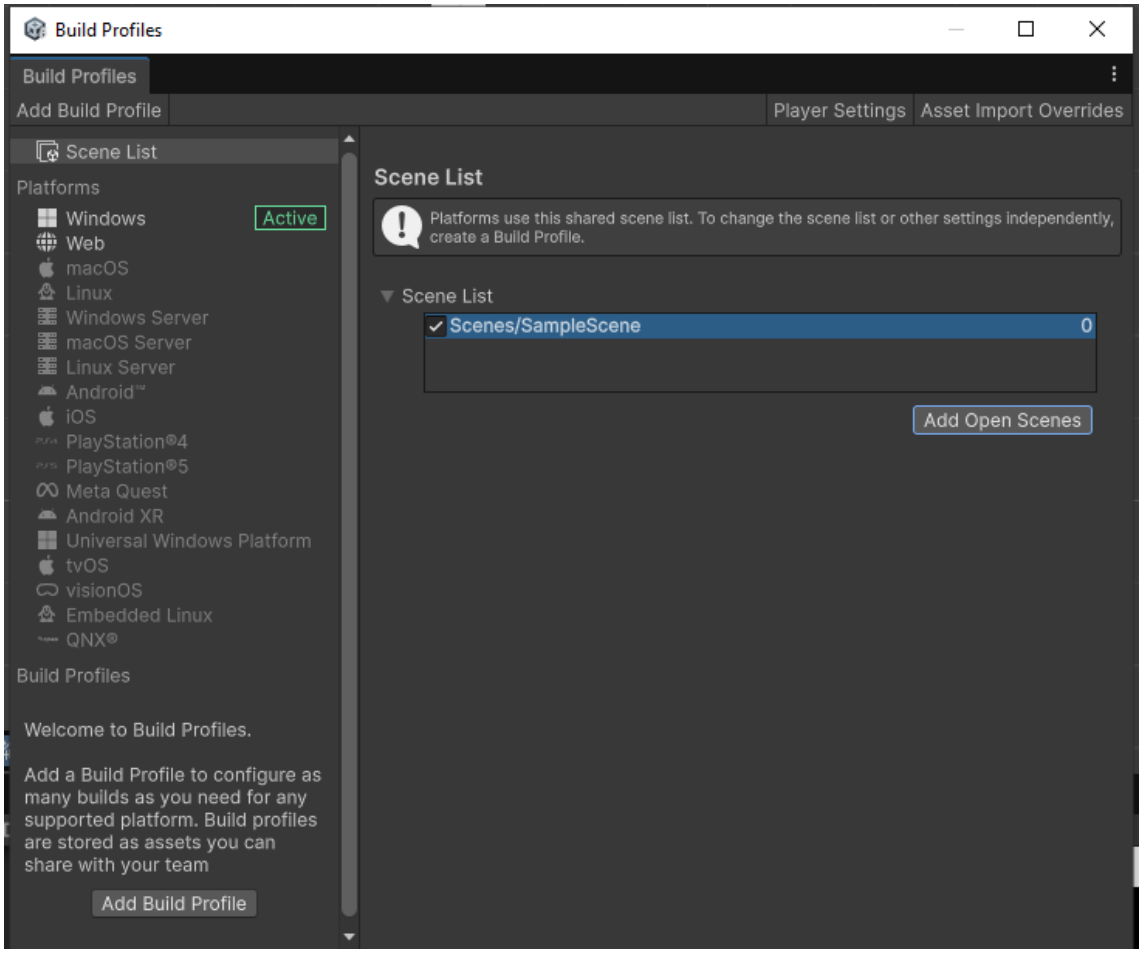
Vamos al código para hacer que estos botones funcionen.

```
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UIElements;
using UnityEngine.InputSystem;

public class PlayerController : MonoBehaviour
{
    public float thrustForce = 1f;
    public float maxSpeed = 5f;
    private float elapsedTime = 0f;
    public GameObject boosterFlame;
    private float score = 0f;
    public float scoreMultiplier = 10f;
    public UIDocument uiDocument;
    private Label scoreText;
    private Button restartButton;
    public GameObject explosionEffect;
    Rigidbody2D rb;

    void Start()
    {
        rb = GetComponent<Rigidbody2D>();
        scoreText = uiDocument.rootVisualElement.Q<Label>("ScoreLabel");
        restartButton =
uiDocument.rootVisualElement.Q<Button>("RestartButton");
        restartButton.style.display = DisplayStyle.None;
        restartButton.clicked += ReloadScene;
    }

    void Update()
    {
        elapsedTime += Time.deltaTime;
        score = Mathf.FloorToInt(elapsedTime * scoreMultiplier);
        Debug.Log("Score: " + score);
        scoreText.text = "Score: " + score;
        // Booster visual ON/OFF según el ratón
        if (Mouse.current.leftButton.wasPressedThisFrame)
        {
            boosterFlame.SetActive(true);
        }
        else if (Mouse.current.leftButton.wasReleasedThisFrame)
        {
            boosterFlame.SetActive(false);
        }

        // Movimiento hacia el mouse
        if (Mouse.current.leftButton.isPressed)
        {
```

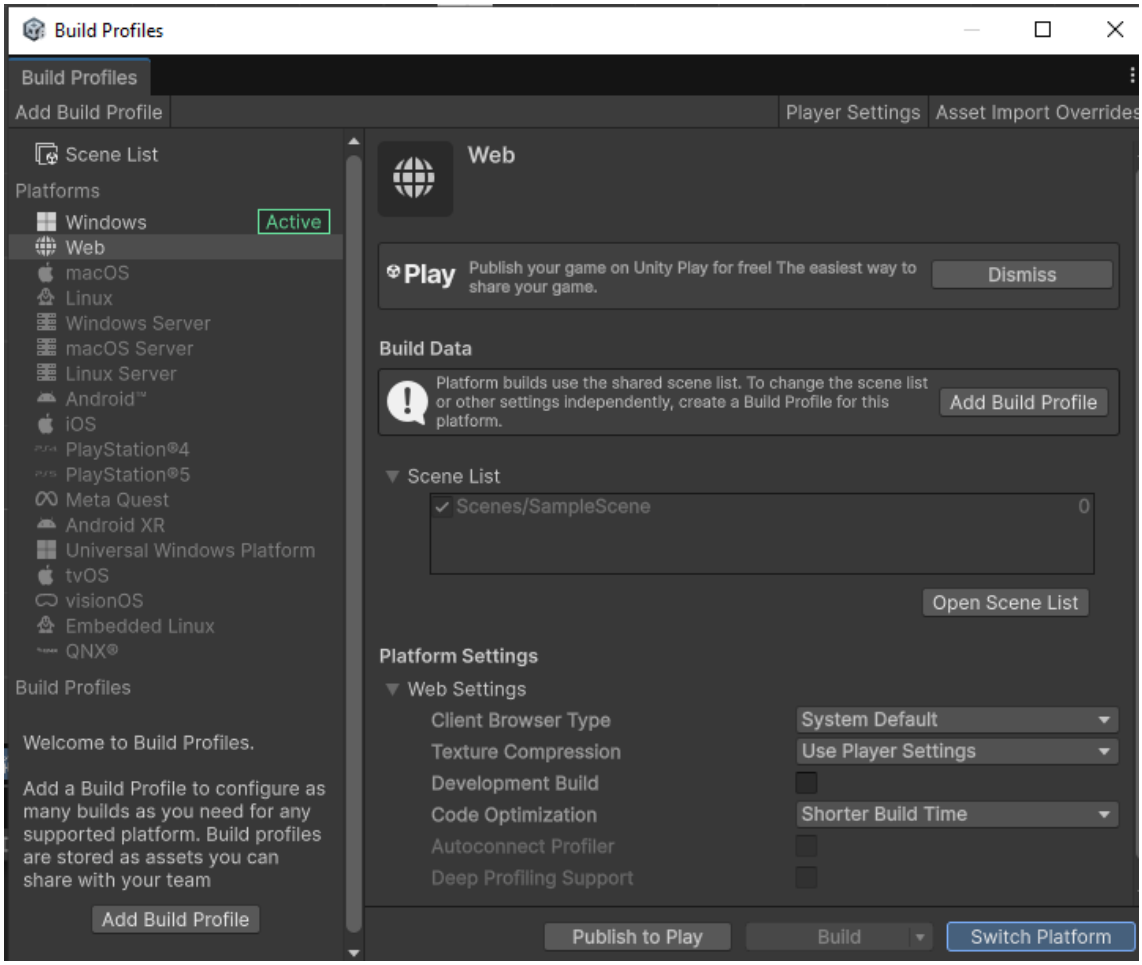
```
            Vector3 mousePos =
Camera.main.ScreenToWorldPoint(Mouse.current.position.value);
            Vector2 direction = (mousePos -
transform.position).normalized;

            transform.up = direction;
            rb.AddForce(direction * thrustForce);

            // Limitar velocidad máxima
            if (rb.linearVelocity.magnitude > maxSpeed)
            {
                rb.linearVelocity = rb.linearVelocity.normalized *
maxSpeed;
            }
        }
    }

    void OnCollisionEnter2D(Collision2D collision)
    {
        Destroy(gameObject);
        Instantiate(explosionEffect, transform.position,
transform.rotation);
        restartButton.style.display = DisplayStyle.Flex;
    }

    void ReloadScene()
    {
    SceneManager.LoadScene(SceneManager.GetActiveScene().name);
    }

}
```

Por fin vamos a crear la build de nuestro juego. Vamos a File/Build Profiles

Vamos a Scene List
Si ya hay escenas presentes en la sección Lista de escenas, selecciónalas y presiona la tecla Suprimir para eliminarlas.
Con la escena Game abierta, selecciona Añadir escenas abiertas y tu escena aparecerá ahora en la Lista de escenas.
Tu escena Game ya está correctamente añadida a la compilación, asegurando que se incluya cuando generes una versión jugable.



Vamos a la pestaña Web y apretamos donde pone Switch Platforms



Una vez termina ya podemos darle a Build para que nos haga una copia de nuestro juego que podamos guardar y usar desde cualquier dispositivo, seleccionamos donde la queremos guardar y listo.

Conclusión

Hasta aquí hemos completado la configuración básica del entorno para el juego Sprite Flight. Hemos aprendido a:

- Crear una escena de juego
- Añadir objetos 2D (hexágono y cuadrados)
- Configurar la cámara principal con fondo y tamaño adecuados
- Definir bordes para delimitar el área de juego
- Establecer un aspect ratio fijo para asegurar consistencia visual
- Organizar el proyecto agrupando objetos y ordenando assets

Aunque el tutorial base termina en este punto, los fundamentos establecidos son esenciales para continuar desarrollando el juego. Los siguientes pasos lógicos serían:

- Añadir movimiento al hexágono u otros objetos
- Mejorar el aspecto visual con formatos y colores
- Crear mecánicas de juego (puntuación, vidas, etc.)
- Añadir efectos visuales y sonoros

Este proyecto sirve como base sólida para explorar las capacidades de Unity en el desarrollo de juegos 2D.