ROLL A BALL

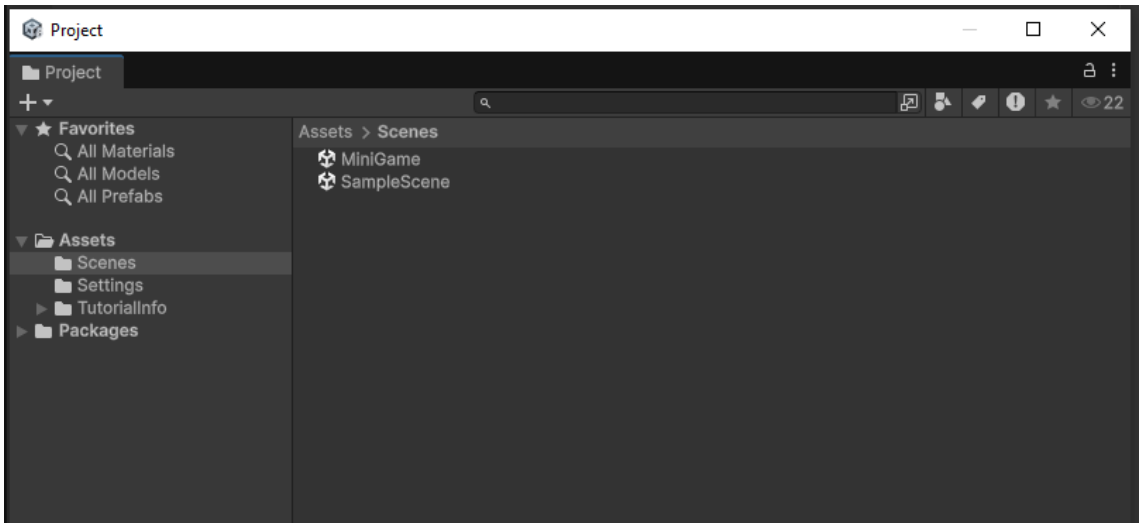
CREACIÓN DE JUEGO ROLL A BALL USANDO UNITY

Jerry Wong Cal

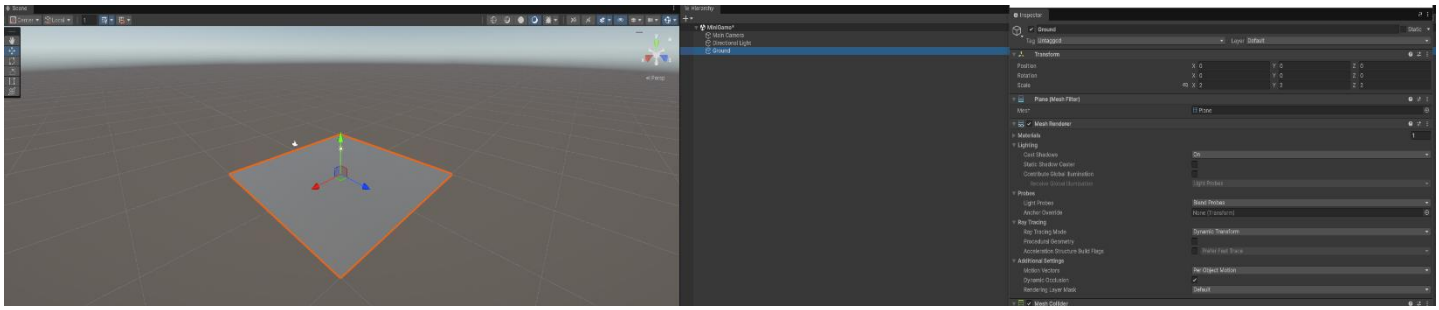
Github: https://github.com/chinese86/Roll-a-Ball

Abrimos un proyecto y elegimos Universal 3D template, le llamamos al proyecto Roll a Ball Bueno

Crea una nueva escena a partir de una plantilla.
Para crear una nueva escena, selecciona File > New Scene.
Selecciona la plantilla Basic URP y luego haz clic en Create.
Guarda la escena.
Selecciona File > Save As.
Ponle a la escena el nombre "MiniGame".
Guarda la escena en la carpeta llamada "Scenes".

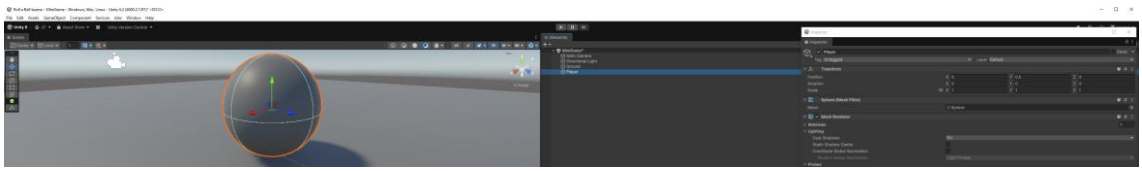


Crea un GameObject de tipo Plane.
En el menú principal, selecciona GameObject > 3D Object > Plane.
y le pondremos de nombre Ground.

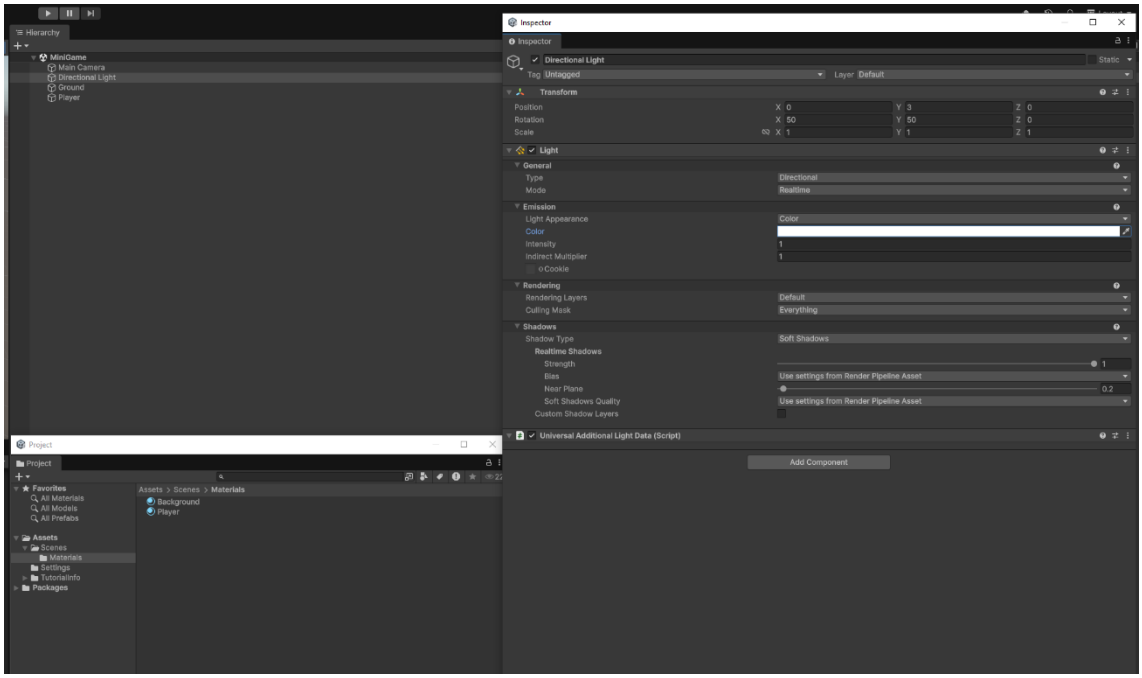


Ponemos los valores Scale X, Y, y Z a 2.

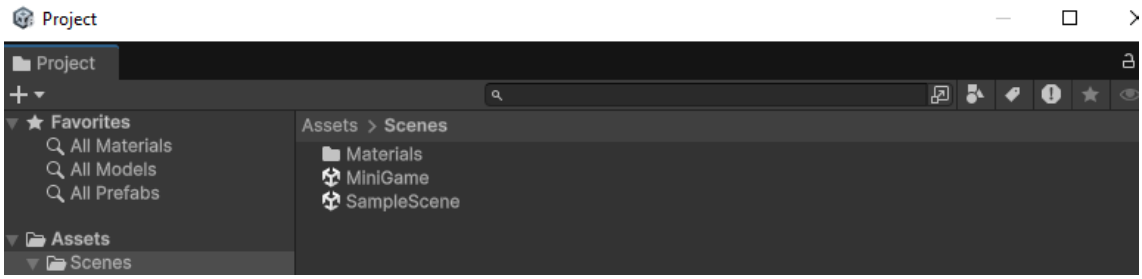
Para crear la esfera vamos a Hierarchy, botón derecho 3D object/Sphere y lo nombramos como Player, luego cambiamos los valores Y a 0,5

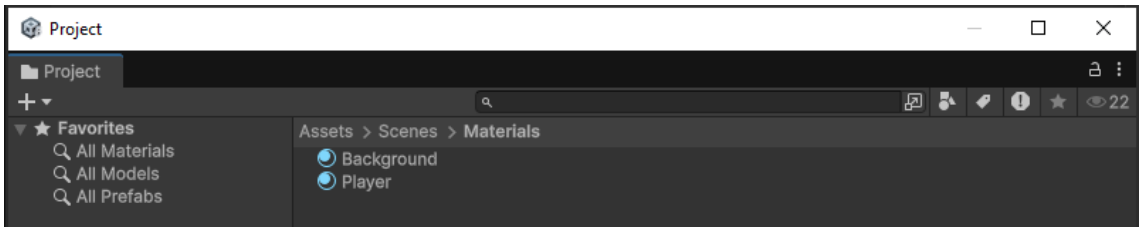


Vamos a colocar la luz direccional
Navega desde la vista Escena (Scene) a la vista Juego (Game) para observar la iluminación y sombras en el GameObject Player.
En la ventana Jerarquía (Hierarchy), selecciona la Directional Light.
2. Cambia el color de la luz a blanco.
En el componente Light, usa la pequeña flecha para expandir el módulo Emission y utiliza el selector de color para modificar el tono de la luz de amarillo a blanco puro, para un mejor equilibrio de la escena.
En el diálogo de color, también puedes establecer manualmente los valores RGB en 255, 255 y 255.
Nota: Para editar los valores RGB desde la ventana de color, utiliza el desplegable para cambiar el modo de color de HSV a RGB 0–255

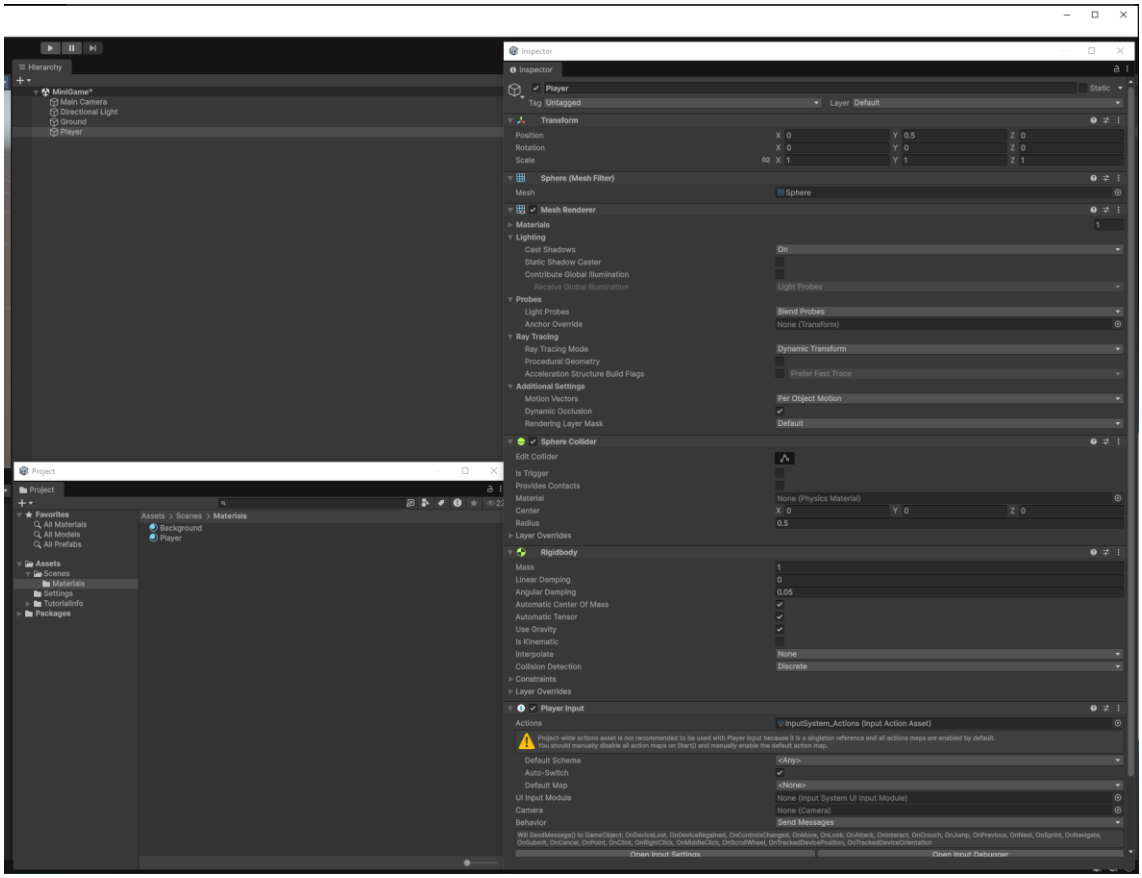


2. Crea un nuevo material llamado Background.
   En la carpeta Materials recién creada, haz clic derecho > Create > Material para crear un nuevo material, y luego nómbralo "Background".
   En la ventana Inspector, utiliza la flecha para expandir el módulo Surface Inputs y selecciona el selector de color Base Map.
   Cambia el color a un gris claro con los valores RGB en 130, 130 y 130.
   Asegúrate de que Metallic Map esté en 0 y que Smoothness esté en aproximadamente 0.25 para un acabado mate.
   Aplica el material Background al GameObject Ground arrastrándolo desde la ventana Project sobre el GameObject Ground en la vista de Escena.]



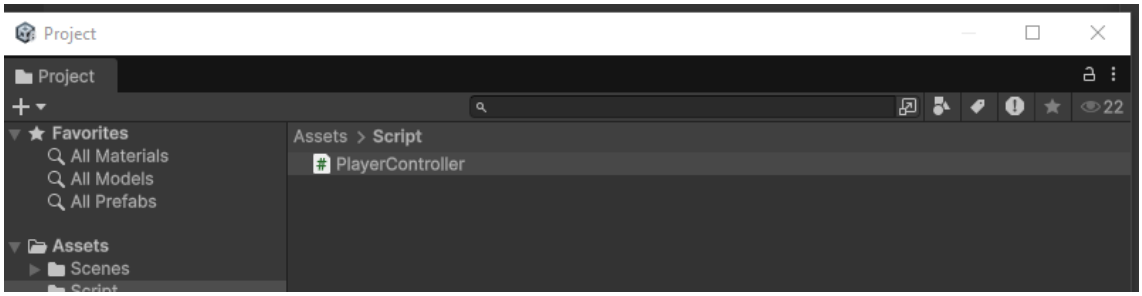


Ahora vamos a seleccionar Player y en el inspector vamos a Add Component y elegimos RigidBody y luego añadimos otro componente que es Player Input



Crea un nuevo script llamado PlayerController.
En la ventana Proyecto, crea una nueva carpeta llamada "Scripts".
Con el GameObject Player seleccionado, selecciona Añadir Componente > Nuevo Script y luego nombra el nuevo script "PlayerController".
El archivo del script creado aparecerá, por defecto, en el nivel raíz de la carpeta "Assets".
Mueve el nuevo archivo PlayerController a la carpeta "Scripts".

Abre el script en un editor de código.
Haz doble clic sobre el asset del script en la ventana Por defecto se abre Visual Studio si lo tienes instalado.



Con el Visual Studio escribimos el siguiente código:

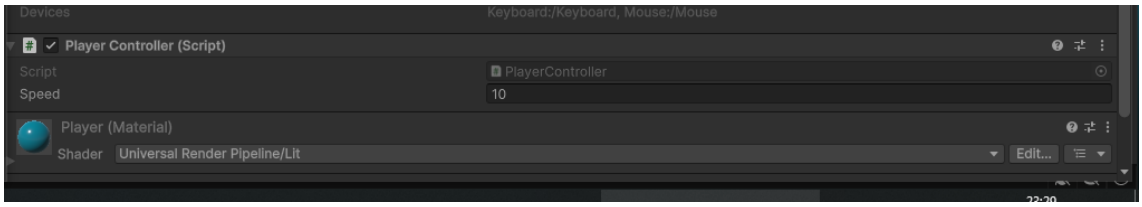
```
using UnityEngine;
using UnityEngine.InputSystem;
using System.Collections;
using System.Collections.Generic;

public class PlayerController : MonoBehaviour
{
    private Rigidbody rb;
    private Vector2 movementValue;
    private Vector3 movementVector;

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

    void OnMove(InputValue movementValue)
    {
        this.movementValue = movementValue.Get<Vector2>();
    }

    void FixedUpdate()
    {
        movementVector = new Vector3(movementValue.x, 0.0f,
movementValue.y);
        rb.AddForce(movementVector);
    }
}
```

Le damos a guardar y volvemos al proyecto en Unity, al darle al play podemos ver cómo la bola se mueve muy lentamente cuando apretamos los botones de dirección del teclado.

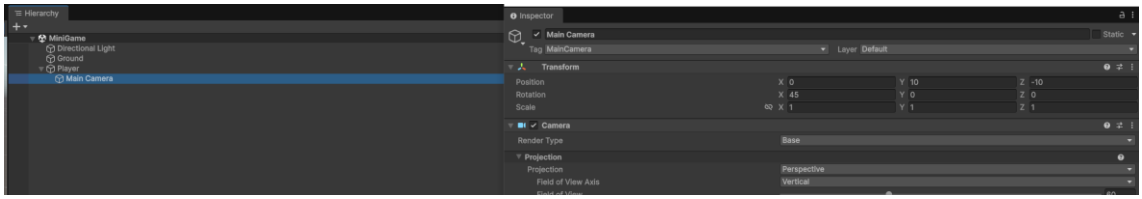
Para hacer que la bola se mueva con mas agilidad vamos a añadir código y quedaría de la siguiente manera:

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

public class PlayerController : MonoBehaviour
{

    public float speed = 0;
    private Rigidbody rb;
    private float movementX;
    private float movementY;

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

    void OnMove(InputValue movementValue)
    {

        Vector2 movementVector = movementValue.Get<Vector2>();
        movementX = movementVector.x;
        movementY = movementVector.y;
    }

    void FixedUpdate()
    {
        Vector3 movement = new Vector3 (movementX, 0.0f, movementY);
        rb.AddForce(movement * speed);
    }
}
```

Y en Hierarchy con el Player seleccionado vamos a Speed y le damos un valor de 10.



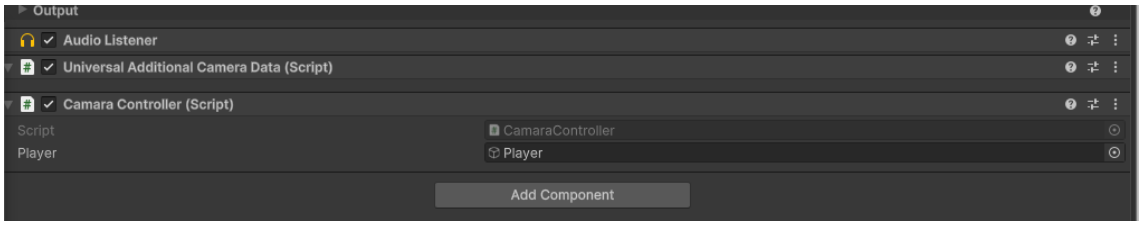
Para que la cámara haga un seguimiento de nuestra bola (Player) vamos a Hierarchy/Main Camera y le damos de valores Position Y=10 y Rotation X=45, luego arrastramos Main Camera y la ponemos dentro de Player para que la cámara siga siempre a Player.



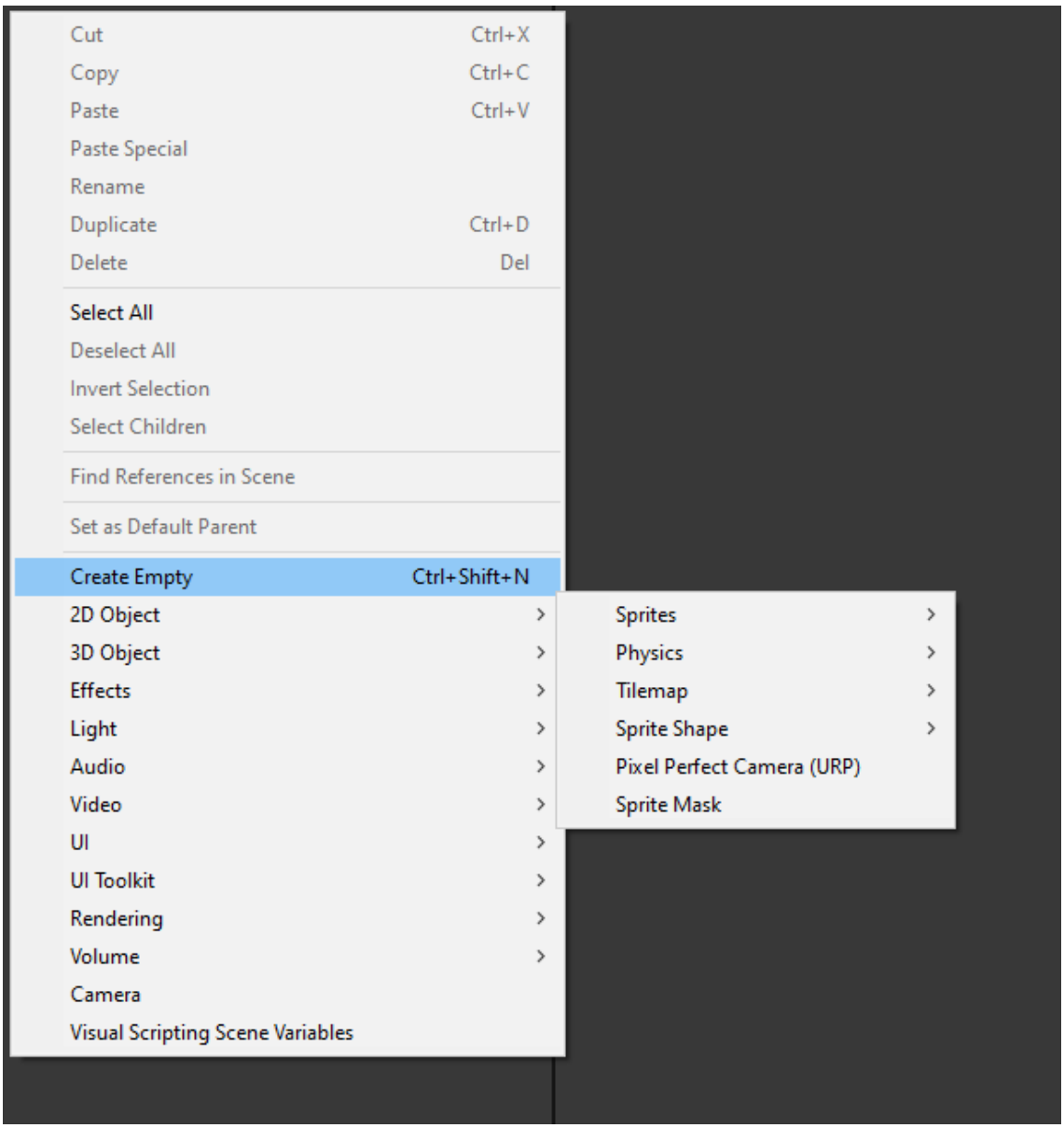
Ahora vamos a crear un Script para Main Camera, para que siga a player pero siempre desde el mismo plano, vamos a Main Camera/Add Component/Script y le ponemos de nombre CameraController, lo guardamos en la carpeta Scripts junto con el Script que teníamos de Player y ahora vamos a editarlo para dejarlo de la siguiente manera:

```
using UnityEngine;

public class CamaraController : MonoBehaviour
{

    public GameObject player;
    private Vector3 offset;
    // Start
    // is called once before the first execution of Update after the
MonoBehaviour is created
    void Start()
    {
        offset = transform.position - player.transform.position;
    }

    // Update is called once per frame
    void LateUpdate()
    {
        transform.position = player.transform.position + offset;
    }
}
```

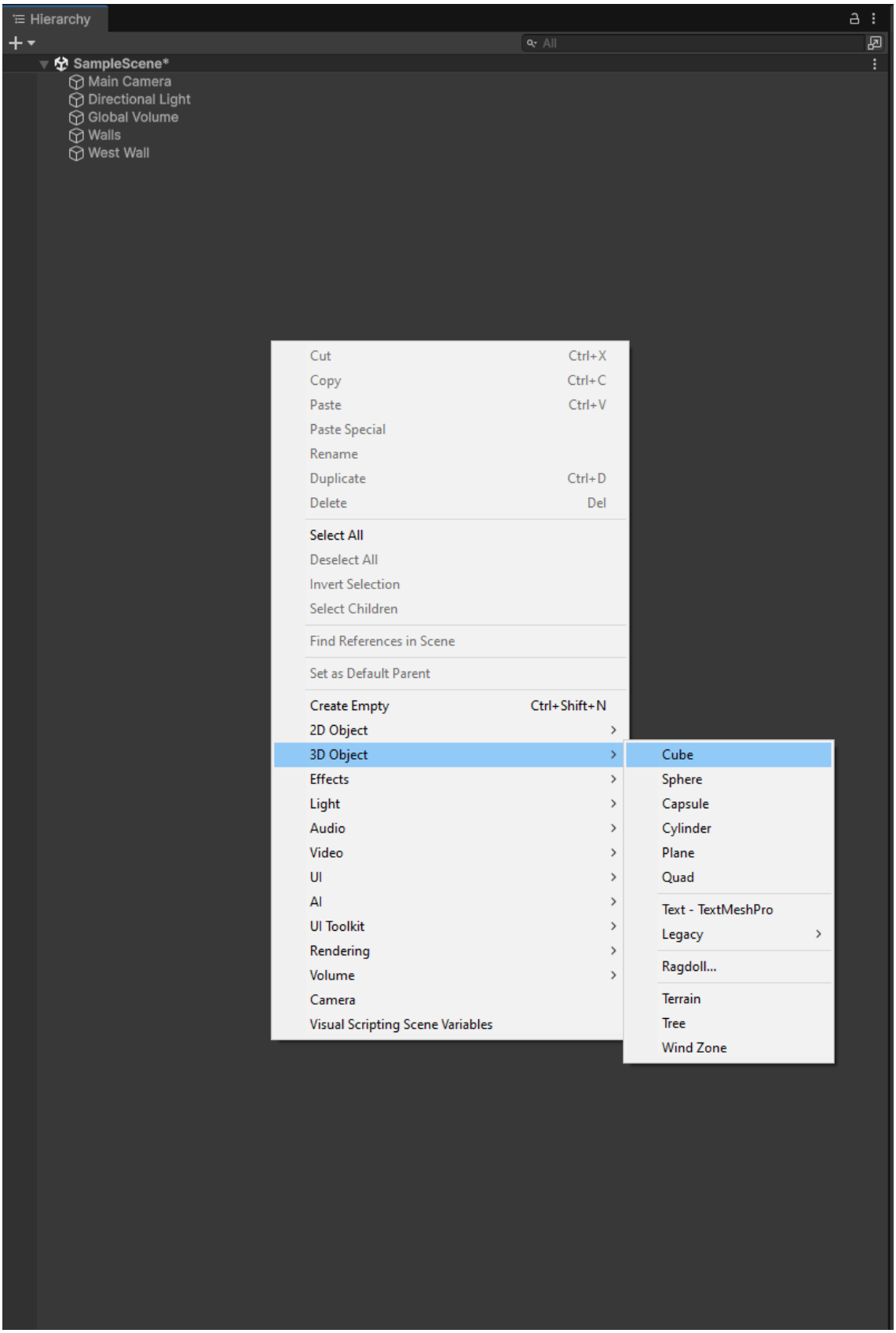
Despues de esto solo nos queda un paso para dejar bien ya la cámara .



A continuación vamos a crear las paredes que delimitan nuestro juego. En la ventana Hierarchy le damos al botón derecho del ratón y pulsamos Create Empty, con esto creamos un objeto vacío, ahora pulsamos sobre él con el botón derecho del ratón, le damos a rename y lo llamamos walls.

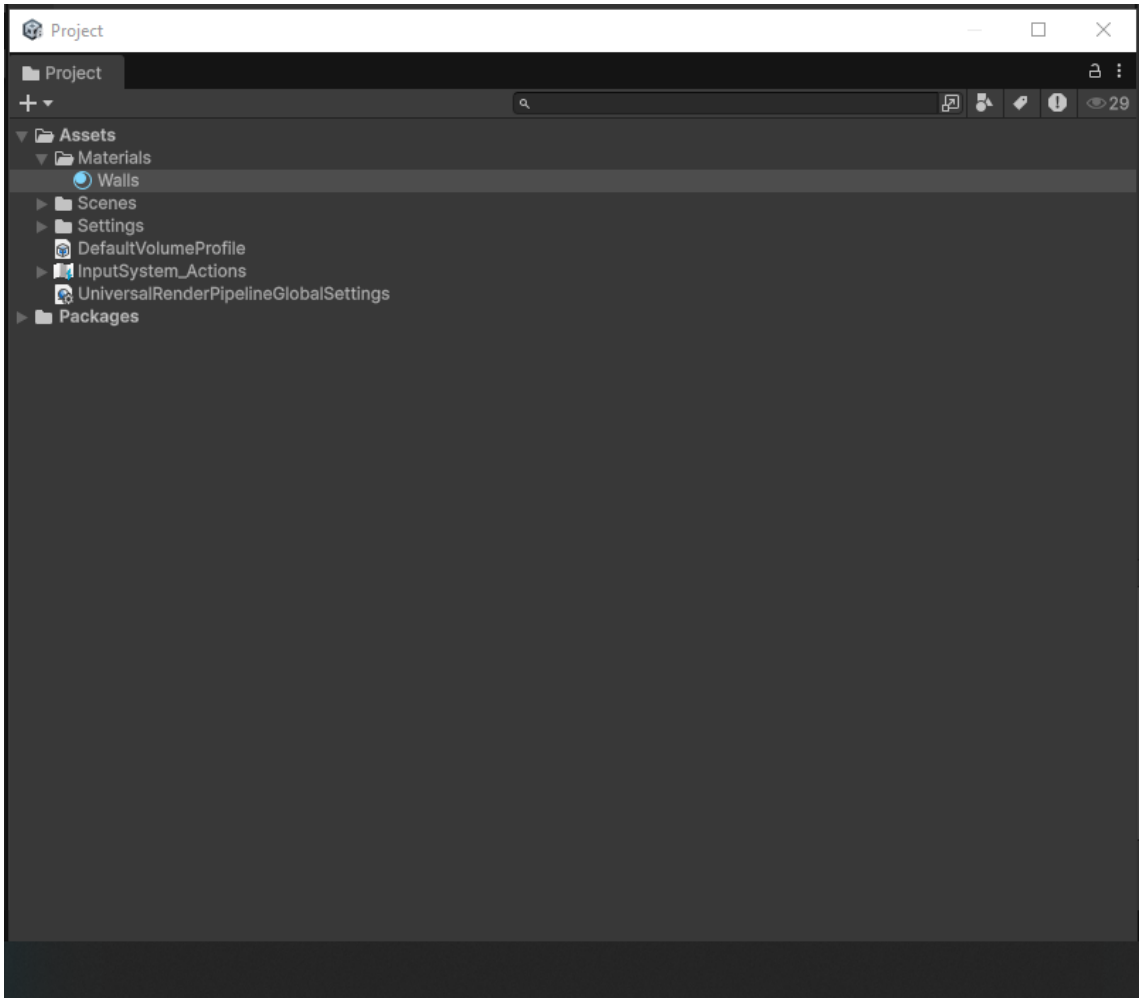


A continuación crearemos un objeto en 3d/ Cube y le llamaremos West Wall

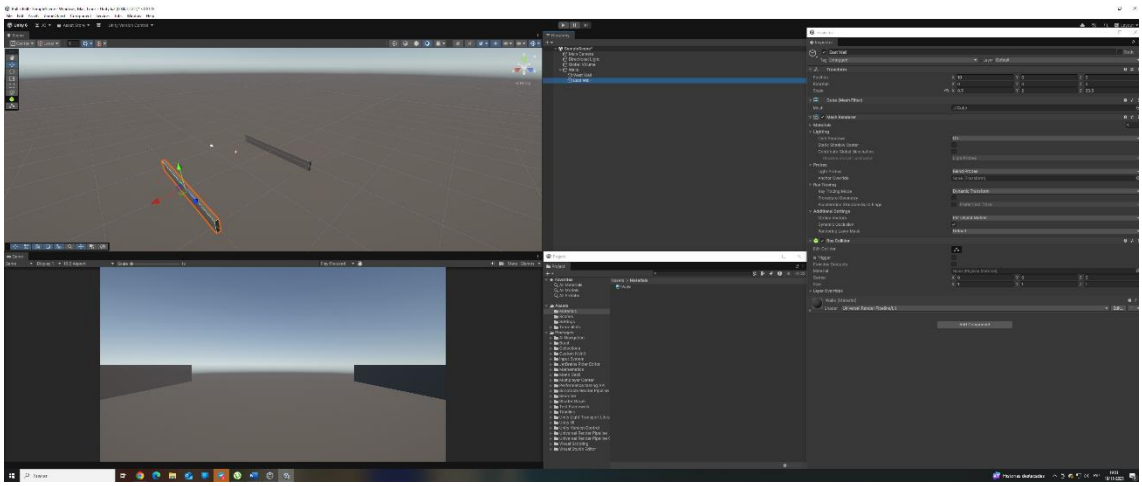


Tenemos que ir a Project (A mi no me aparecía y tuve que ir a la pestaña Windows/General/Project) y crear una carpeta llamada Materials (Botón derecho new folder) y dentro vamos a crear un material llamado Walls (Botón derecho new Material y lo renombramos como Walls)

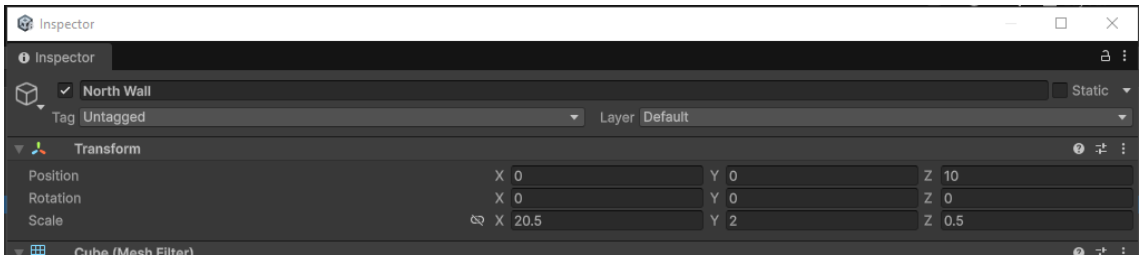
En la ventana Inspector, selecciona el campo de color Base Map para abrir el selector de color, luego pon los valores RGB en 79, 79, 79 para obtener un gris oscuro.
Pon el Metallic Map en 0 y cambia la Suavidad (Smoothness) a 0.25 para un acabado mate.
Arrastra el material Walls desde la ventana Proyecto al GameObject “West Wall” en la vista de Escena.



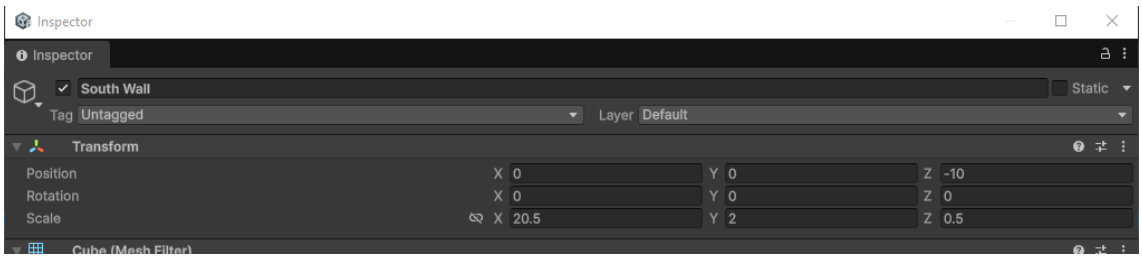
Ahora vamos a crear otra pared, para ello vamos con el botón derecho sobre la pared que ya teníamos (West Wall) y le damos a duplicar, le cambiamos el nombre y le ponemos East Wall y cambiamos en position que teníamos -10 por 10



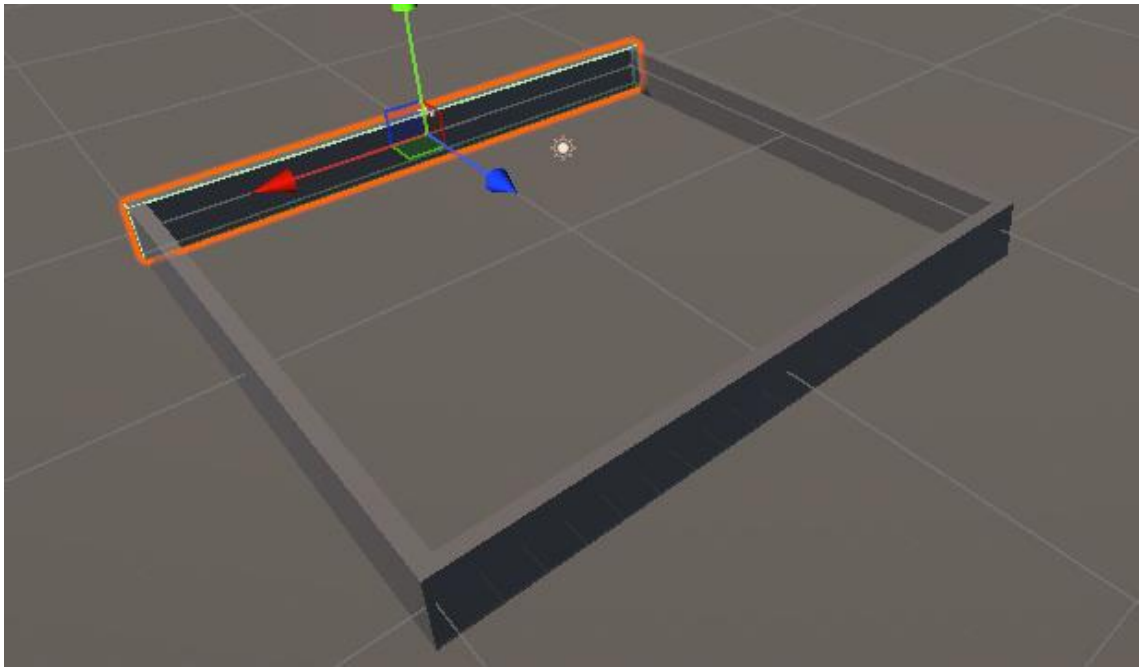
Para la pared Norte hacemos lo mismo, la duplicamos y le cambiamos el nombre a North Wall y ponemos los siguientes valores para colocarla.



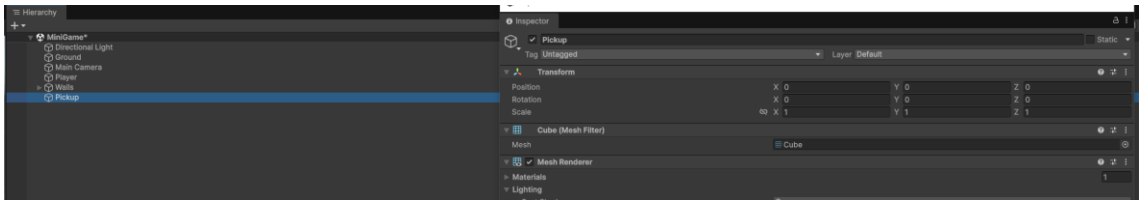
Ahora para la última pared, duplicamos y la nombramos como South Wall y dejamos los valores como la imagen siguiente



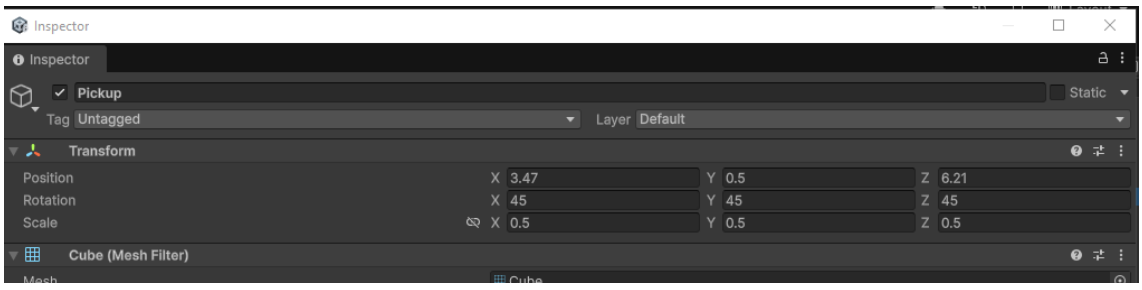
Y ya tendríamos nuestras 4 paredes



Una vez tenemos nuestras 4 paredes vamos a crear los objetos coleccionables, vamos a Hierarchy, creamos un nuevo objeto 3D lo nombramos como Pickup y reseteamos los valores a 0.



Luego le damos estos valores



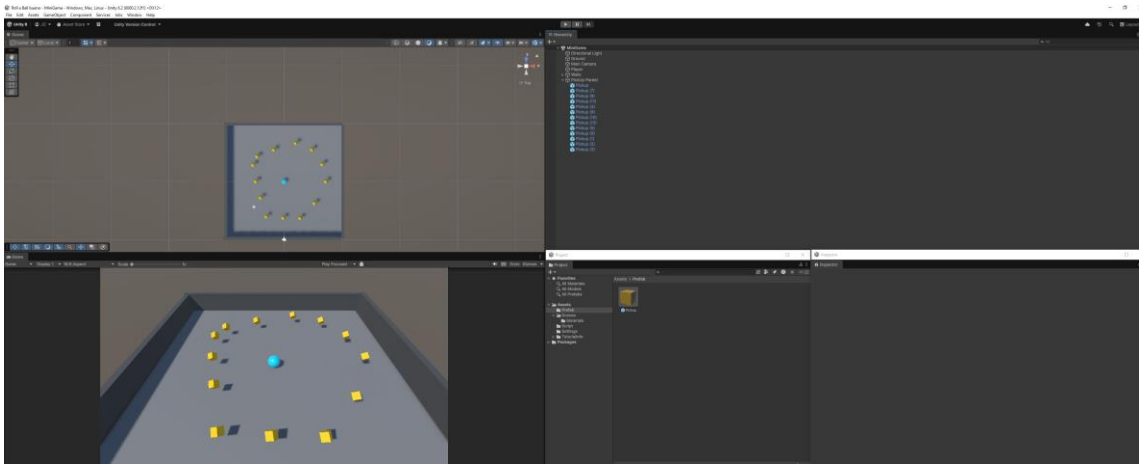
En Project creamos un nuevo material para darle un toque estético más llamativo, al material lo llamaremos Pickup y los valores RGB serán 255, 200, y 0

Ahora vamos a darle movimiento de rotación sobre si mismo al objeto 3D Pickup, para ello seleccionamos Pickup/Add Component/New Script, le llamamos Rotator y vamos al Visual Studio para crear el código.

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

public class Rotator : MonoBehaviour
{
    // Start is called once before the first execution of Update after
the MonoBehaviour is created
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
        transform.Rotate (new Vector3 (15, 30, 45) * Time.deltaTime);
    }
}
```

Guardamos y ya podemos ver el objeto Pickup rotando.

Ahora creamos un PickUp Parent y metemos dentro todos los PickUp que queremos crear, en este caso vamos duplicando el que ya tenemos y los vamos colocando donde queremos que estén.

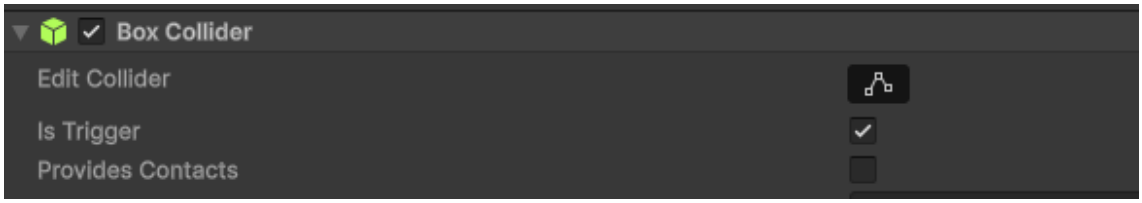


Ahora vamos a ponerles un Tag a los PickUps y cambiamos el código a esto:

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

public class PlayerController : MonoBehaviour
{

    public float speed = 0;
    private Rigidbody rb;
    private float movementX;
    private float movementY;

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

    void OnMove(InputValue movementValue)
    {

        Vector2 movementVector = movementValue.Get<Vector2>();
        movementX = movementVector.x;
        movementY = movementVector.y;
    }

    void FixedUpdate()
    {
        Vector3 movement = new Vector3 (movementX, 0.0f, movementY);
        rb.AddForce(movement * speed);
    }
```

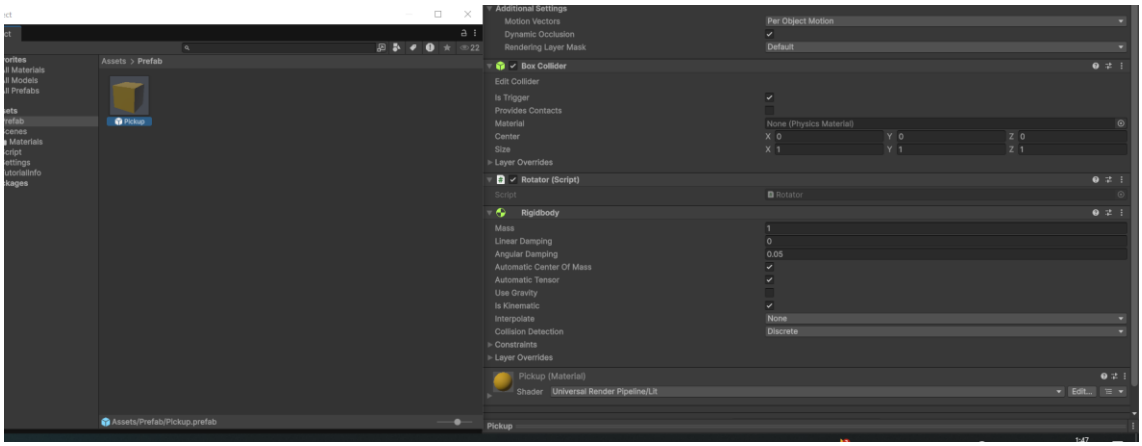
```
    void OnTriggerEnter(Collider other)
{
    if (other.gameObject.CompareTag("PickUp"))
    {
     other.gameObject.SetActive(false);
    }

}
}
```

Al ejecutar el juego aún la bola choca contra los pickUps pero para solucionarlo vamos a la carpeta Prefab en Project, Pickup y en el inspector buscamos is Trigger y lo marcamos



Ahora al ejecutar el programa, cuando la bola pasa por encima de los Pickups, estos desaparecen.

En Project/Assets/Prefab/Pickup vamos a add Component y le ponemos un Rigidbody, deseleccionamos Use gravity para que los Pickups no desaparezcan debajo del suelo atraídos por la gravedad y seleccionamos is Kinematic.



Ahora que al pasar la bola por los Pickups estos desaparecen, vamos a crear un contador de puntuación, para ello vamos al Script de PlayerController y lo dejaremos así:

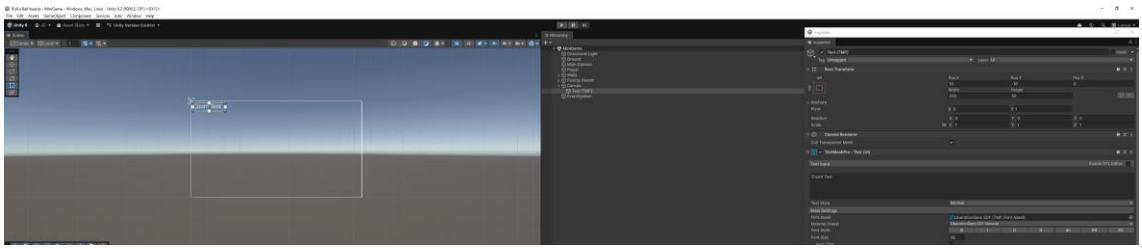
```
using UnityEngine;
```

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

public class PlayerController : MonoBehaviour
{

    public float speed = 0;
    private Rigidbody rb;
    private int count;
    private float movementX;
    private float movementY;

    void Start()
    {
        rb = GetComponent<Rigidbody>();
        count = 0;
    }

    void OnMove(InputValue movementValue)
    {

        Vector2 movementVector = movementValue.Get<Vector2>();
        movementX = movementVector.x;
        movementY = movementVector.y;
    }

    void FixedUpdate()
    {
        Vector3 movement = new Vector3 (movementX, 0.0f, movementY);
        rb.AddForce(movement * speed);
    }

     void OnTriggerEnter(Collider other)
   {
      if (other.gameObject.CompareTag("PickUp"))
       {
        other.gameObject.SetActive(false);
        count = count + 1;
       }

   }
}
```

Guardamos el proyecto.

Ahora vamos a crear en nuestro juego un cuadro de texto para poder ver el contador físicamente en el videojuego. Vamos a Hierarchy/add/UI/Text/TextMeshPRO y una vez

añadido vamos a Inspector y cambiamos el texto y le ponemos Count Text y ponemos en Rect Transform los valores siguientes:



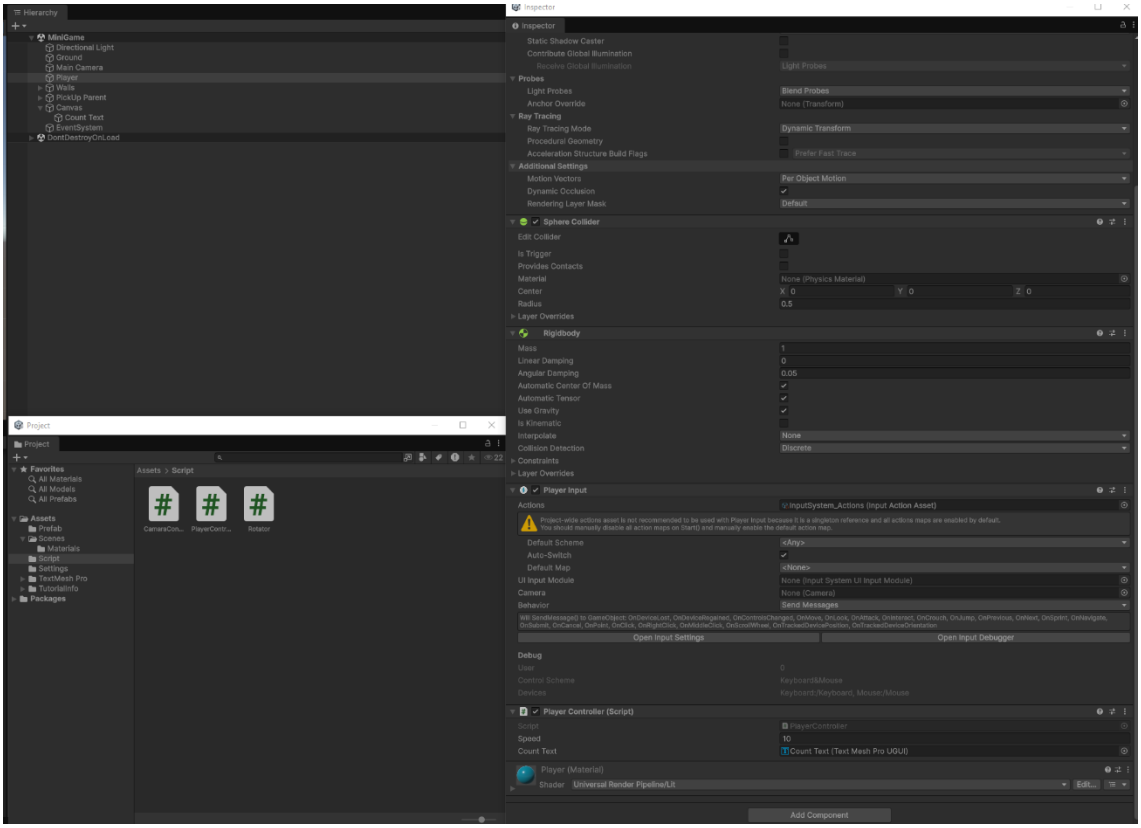
Ahora ya tenemos un contador en la parte superior izquierda.

Vamos al Script/PlayerController y lo modificamos:

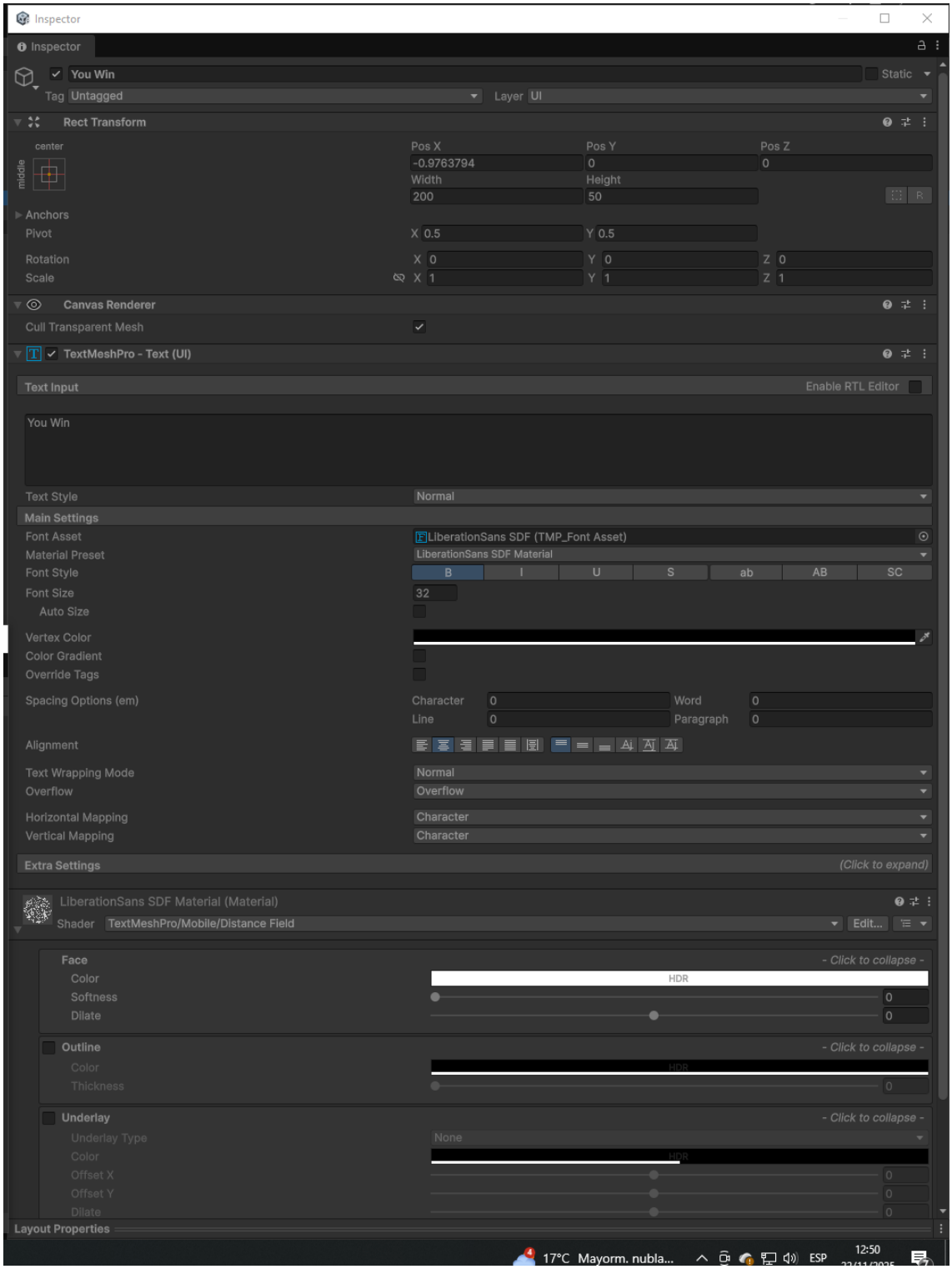
```
using UnityEngine;
using UnityEngine.InputSystem;
using System.Collections;
using System.Collections.Generic;
using TMPro;

public class PlayerController : MonoBehaviour
{

    public float speed = 0;
    public TextMeshProUGUI countText;
    private Rigidbody rb;
    private int count;
    private float movementX;
    private float movementY;

    void Start()
    {
        rb = GetComponent<Rigidbody>();
        count = 0;

        SetCountText();
    }

    void OnMove(InputValue movementValue)
    {

        Vector2 movementVector = movementValue.Get<Vector2>();
        movementX = movementVector.x;
        movementY = movementVector.y;
    }
       void SetCountText()
   {
       countText.text =  "Count: " + count.ToString();
```

```
    }

    void FixedUpdate()
    {
        Vector3 movement = new Vector3 (movementX, 0.0f, movementY);
        rb.AddForce(movement * speed);
    }

    void OnTriggerEnter(Collider other)
    {
        if (other.gameObject.CompareTag("PickUp"))
        {
            other.gameObject.SetActive(false);
            count = count + 1;

            SetCountText();
        }

    }
}
```

Asigna la variable CountText en la ventana Inspector.
Selecciona el GameObject Player en la ventana Hierarchy, luego arrastra el GameObject CountText hasta el campo Count Text para referenciar el elemento de texto de la UI.
Importante: Este paso es muy importante y fácil de pasar por alto. Si no lo realizas, verás un error NullReferenceException en la ventana Console y tu juego no funcionará.



Ahora vamos a crear un texto de You Win cuando hayamos capturado todos los objetos PickUp, para ello vamos a Hierarchy/ Add/UI/Text/TextMeshPro, con esto creamos un cuadro de texto al que vamos a llamarlo You Win, en texto ponemos You Win, le ponemos de color negro, alineamos al centro, lo ponemos Bold (Negrita) y ajustamos el tamaño, ahora toca irnos al código.




```
      using UnityEngine;

using UnityEngine.InputSystem;
using System.Collections;
```

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

public class PlayerController : MonoBehaviour
{

    public float speed = 0;
    public TextMeshProUGUI countText;
    private Rigidbody rb;
    private int count;
    public GameObject winTextObject;
    private float movementX;
    private float movementY;

    void Start()
    {
        rb = GetComponent<Rigidbody>();
        count = 0;

        SetCountText();
        winTextObject.SetActive(false);
    }

    void OnMove(InputValue movementValue)
    {

        Vector2 movementVector = movementValue.Get<Vector2>();
        movementX = movementVector.x;
        movementY = movementVector.y;
    }
       void SetCountText()
   {
       countText.text =  "Count: " + count.ToString();
         if (count >= 12)
       {
           winTextObject.SetActive(true);
       }
   }

    void FixedUpdate()
    {
        Vector3 movement = new Vector3 (movementX, 0.0f, movementY);
        rb.AddForce(movement * speed);
    }

     void OnTriggerEnter(Collider other)
   {
      if (other.gameObject.CompareTag("PickUp"))
       {
```

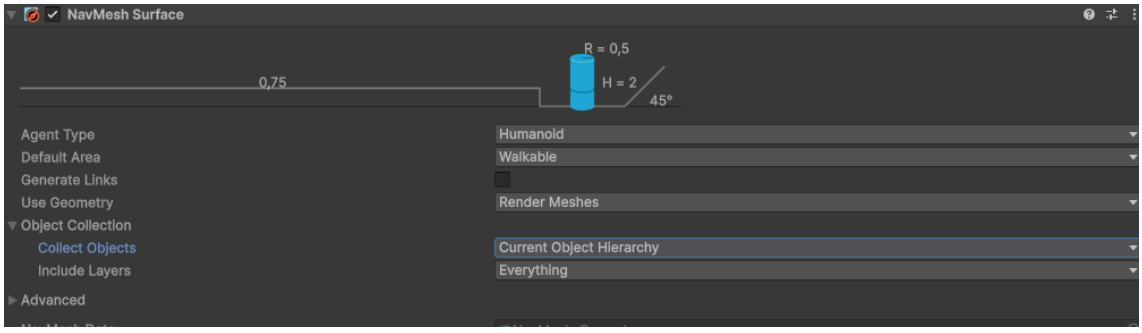
```
            other.gameObject.SetActive(false);
            count = count + 1;

            SetCountText();
           }

    }
}
```

Vamos a poner el juego más interesante creando un enemigo que nos siga y que pueda acabar el juego si nos atrapa, para ello creamos en Herarchy un nuevo objeto 3D de tipo cubo al que vamos a llamar Enemy

Vamos a la carpeta Materials para crear un nuevo material para el enemigo, le llamaremos Enemy, así le daremos un color llamativo

Una vez creado arrastramos y le damos ese color al enemigo que hemos creado.

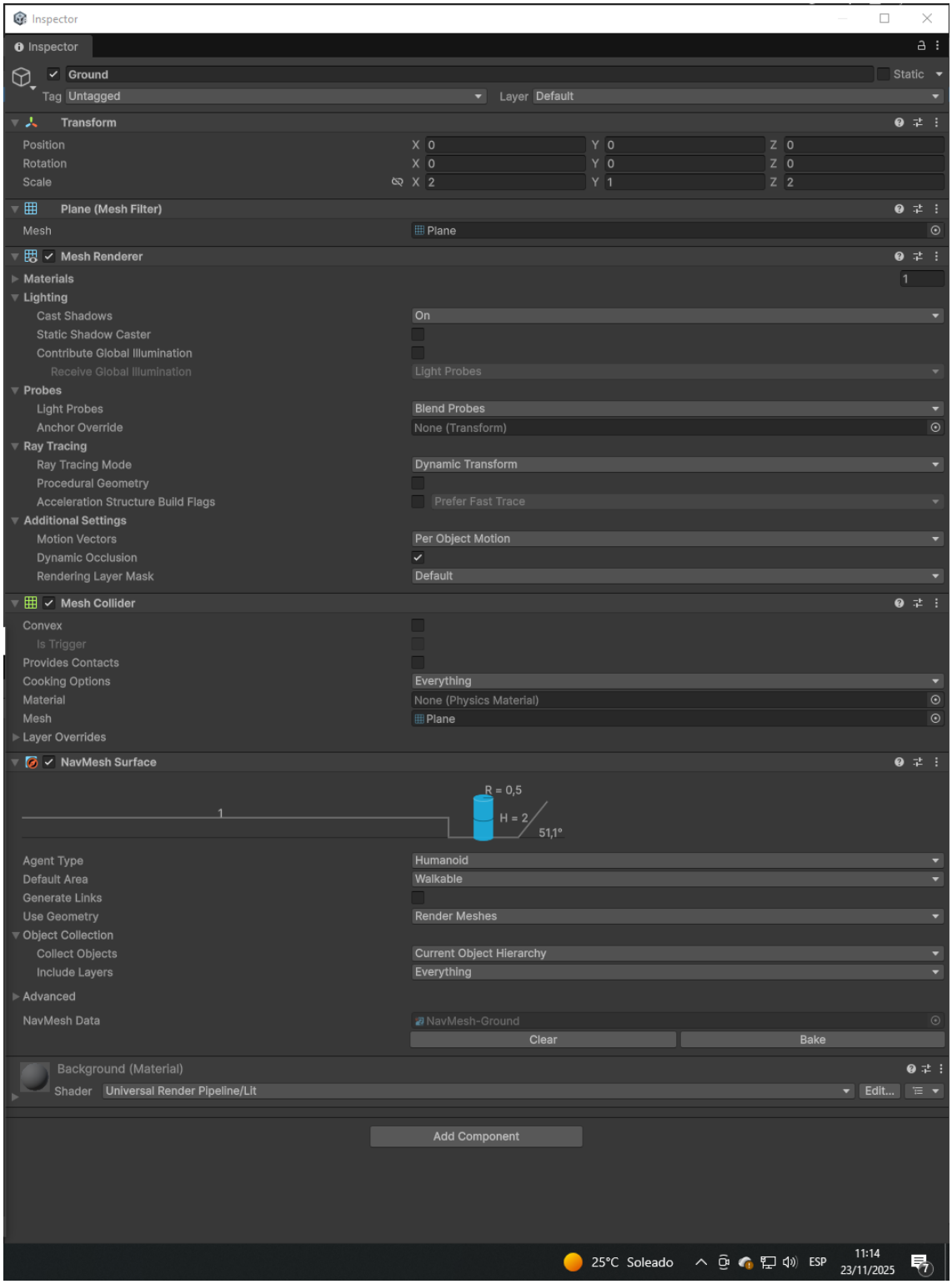
A continuación vamos a decirle al enemigo el área por el que puede moverse, para ello vamos a Ground/Add Component y en Collect Objects elegimos Current Object Hieralchy como vemos en la imagen.



Hemos delimitado el área de movimiento del enemigo pero tenemos que hacer que vaya detrás de nosotros, para ello seleccionamos Enemy en Hierarchy y vamos a Add Component, elegimos Nav Mesh Agent, en speed vamos a poner 2.5 para que no vaya más rápido que nosotros. Ahora vamos a Add Component de nuevo y elegimos New Script, le llamaremos EnemyMovement y guardamos en la carpeta Scripts y pasamos a añadir el código para que el enemigo nos persiga.

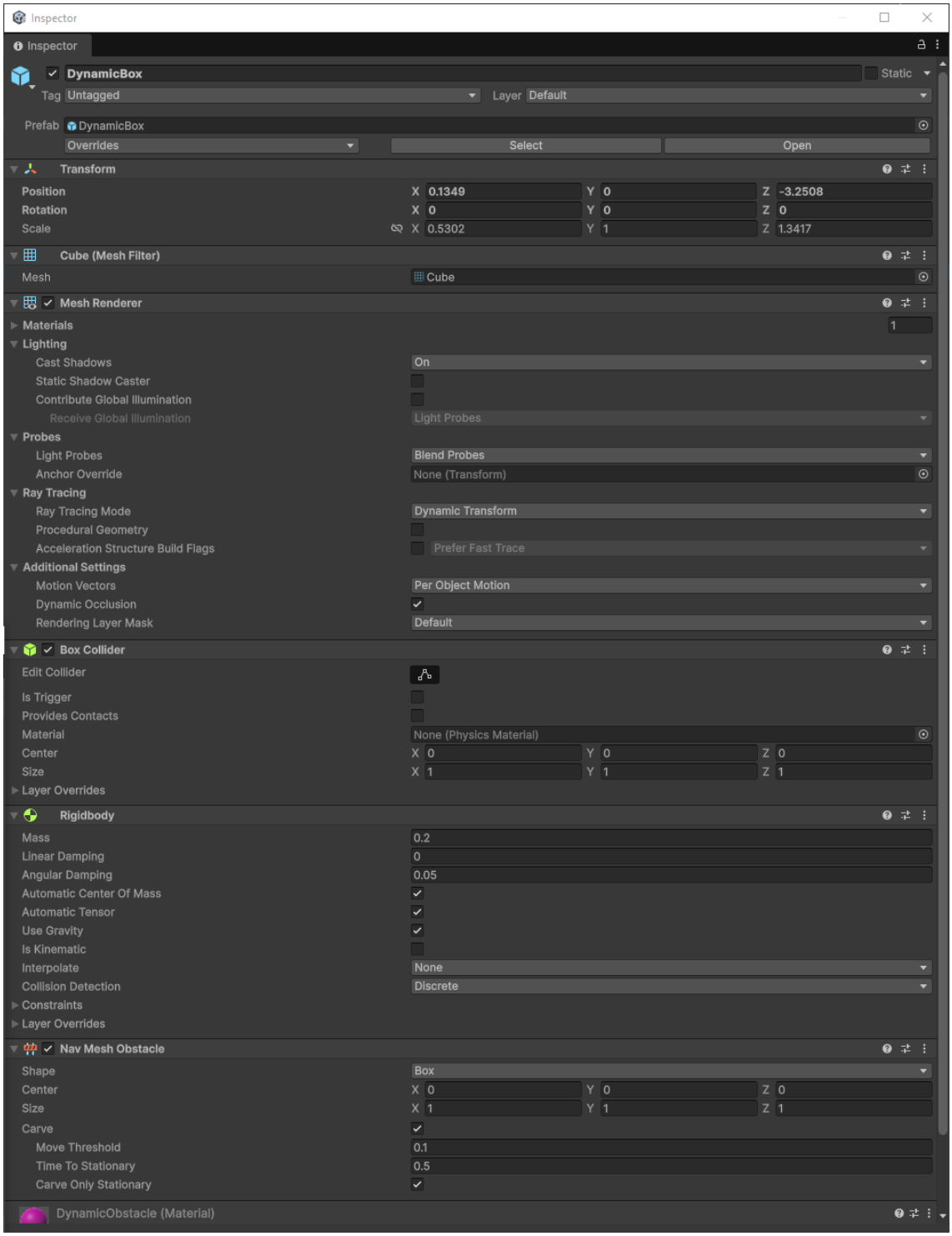
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.AI;

public class EnemyMovement : MonoBehaviour
{

    public Transform player;
    private NavMeshAgent navMeshAgent;

    // Start is called once before the first execution of Update after
the MonoBehaviour is created
    void Start()
    {
        navMeshAgent = GetComponent<NavMeshAgent>();
    }

    // Update is called once per frame
    void Update()
    {
            if (player != null)
       {
           navMeshAgent.SetDestination(player.position);
       }
    }
}
```

Ahora que ya tenemos al enemigo persiguiéndonos vamos a crear obstáculos para hacer aún más complicado el juego si cabe. Vamos a Hierarchy, botón derecho y añadimos objetos 3D, en este caso todo cubos y le vamos cambiando la forma, algunos con forma de rampa y otros solo obstáculos que estén ahí molestando. Despues de crear unos cuantos los vamos a meter dentro de Ground para que éste sea el objeto padre. Con el Ground seleccionado vamos a darle a Bake dentro de NavMesh Surface.


Inspector
Ground
Static
Tag Untagged
Layer Default
Transform
Position X 0 Y 0 Z 0
Rotation X 0 Y 0 Z 0
Scale X 2 Y 1 Z 2
Plane (Mesh Filter)
Mesh Plane
Mesh Renderer
Materials 1
Lighting
Cast Shadows On
Static Shadow Caster
Contribute Global Illumination
Receive Global Illumination Light Probes
Probes
Light Probes Blend Probes
Anchor Override None (Transform)
Ray Tracing
Ray Tracing Mode Dynamic Transform
Procedural Geometry
Acceleration Structure Build Flags Prefer Fast Trace
Additional Settings
Motion Vectors Per Object Motion
Dynamic Occlusion
Rendering Layer Mask Default
Mesh Collider
Convex
Is Trigger
Provides Contacts
Cooking Options Everything
Material None (Physics Material)
Mesh Plane
Layer Overrides
NavMesh Surface
R = 0,5
H = 2
1
51,1°
Agent Type Humanoid
Default Area Walkable
Generate Links
Use Geometry Render Meshes
Object Collection
Collect Objects Current Object Hierarchy
Include Layers Everything
Advanced
NavMesh Data NavMesh-Ground
Clear
Bake
Background (Material)
Shader Universal Render Pipeline/Lit
Edit...
Add Component
25°C Soleado
ESP
11:14
23/11/2025

Ahora vamos a crear obstáculos en movimiento, vamos a Hierarchy, botón derecho/3D Object/Cube y creamos nuestro obstáculo, le damos forma y vamos a crear en la carpeta Materials con el botón derecho/Create/Material un nuevo material para darle un color, en este caso le he llamado DynamicObstacle.

Vamos de nuevo a Hierarchy y seleccionamos nuestro DynamicBox ahora en el Inspector vamos a Add Component y le añadimos un RigidBody y Nav Mesh Obstacle.



Ahora vamos a añadir código en Script/PlayerController para que cuando el enemigo nos toque, nuestro player se destruya y aparezca el mensaje “You lose”

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

public class PlayerController : MonoBehaviour
{

    public float speed = 0;
    public TextMeshProUGUI countText;
    private Rigidbody rb;
    private int count;
    public GameObject winTextObject;
    private float movementX;
    private float movementY;

    void Start()
    {
        rb = GetComponent<Rigidbody>();
        count = 0;

        SetCountText();
        winTextObject.SetActive(false);
    }

    void OnMove(InputValue movementValue)
    {

        Vector2 movementVector = movementValue.Get<Vector2>();
        movementX = movementVector.x;
        movementY = movementVector.y;
    }
       void SetCountText()
   {
       countText.text =  "Count: " + count.ToString();
         if (count >= 12)
       {
           winTextObject.SetActive(true);
       }
   }

    void FixedUpdate()
    {
        Vector3 movement = new Vector3 (movementX, 0.0f, movementY);
        rb.AddForce(movement * speed);
```

```
    }

     void OnTriggerEnter(Collider other)
   {
      if (other.gameObject.CompareTag("PickUp"))
       {
        other.gameObject.SetActive(false);
        count = count + 1;

        SetCountText();
       }

   }
   private void OnCollisionEnter(Collision collision)
{
   if (collision.gameObject.CompareTag("Enemy"))
   {
       // Destroy the current object
       Destroy(gameObject);
       // Update the winText to display "You Lose!"
       winTextObject.gameObject.SetActive(true);
       winTextObject.GetComponent<TextMeshProUGUI>().text = "You Lose!";
   }
}
}
```

1. Guarda tu script y vuelve al Editor de Unity.
2. Selecciona el GameObject EnemyBody en la ventana Hierarchy.
3. En la ventana Inspector, localiza el menú desplegable Tag (está en la parte superior del Inspector).
4. Haz clic en Add Tag
5. Pulsa el botón Add (+) para crear una nueva etiqueta y llama a la etiqueta Enemy.
6. Guarda la nueva etiqueta.
7. Vuelve a seleccionar el GameObject EnemyBody y, en el menú Tag del Inspector, elige la etiqueta Enemy que acabas de crear.
8. Ejecuta el juego para probar: ahora, cuando el GameObject Enemy colisiona con el jugador, el GameObject Player se destruye y el texto se actualiza a “You lose!”

Ya hemos terminado, ahora vamos a crear un ejecutable para Windows en nuestro caso, pero se puede hacer para MacOs y otros sistemas.

Configura los ajustes del jugador.
Selecciona Player Settings para abrir varias opciones de configuración de la vista Game.
Si quieres, cambia Company Name, Product Name y Version.
Selecciona Resolution and Presentation y cambia el cuadro Fullscreen Mode a Windowed.
Por defecto será una resolución de 1024 por 768, pero puedes poner un ancho y alto menor si tu pantalla usa una resolución más baja.
Cierra la ventana Project Settings y vuelve a la ventana Build Settings.

Compila tu juego.
Selecciona Build. Esto mostrará un diálogo para elegir la ubicación de la compilación.
Para mantener todo ordenado, crea una nueva carpeta dentro de tu proyecto llamada “Builds” al nivel raíz del proyecto, junto a Assets y Library.
Si usas macOS, también puedes elegir un nombre para la compilación.
Confirma que quieres Select Folder (Windows) o Save (macOS). Unity ahora compilará la aplicación y la guardará en la carpeta Builds.]