

UNIVERSIDAD TECNOLÓGICA DEL NORTE DE GUANAJUATO



Ingeniería en Desarrollo y Gestión de Software

Unidad II.

Programación orientada a objetos para videojuegos

Asignatura:

Creación de Videojuegos

PRESENTA:

Herrera Ramiro

Grupo:

GIDS5101-e

A 10 de octubre del 2025

Dolores Hidalgo Cuna de la Independencia Nacional, Guanajuato.



Secretaría
de Educación
de Guanajuato



SEP
SECRETARÍA DE
EDUCACIÓN PÚBLICA

UTP
DIRECCIÓN GENERAL DE UNIVERSIDADES
TECNOLÓGICAS Y POLITÉCNICAS



Universidad Tecnológica
del Norte de Guanajuato
Órgano Público Descentralizado del Gobierno del Estado
"Educación y progreso para la vida"



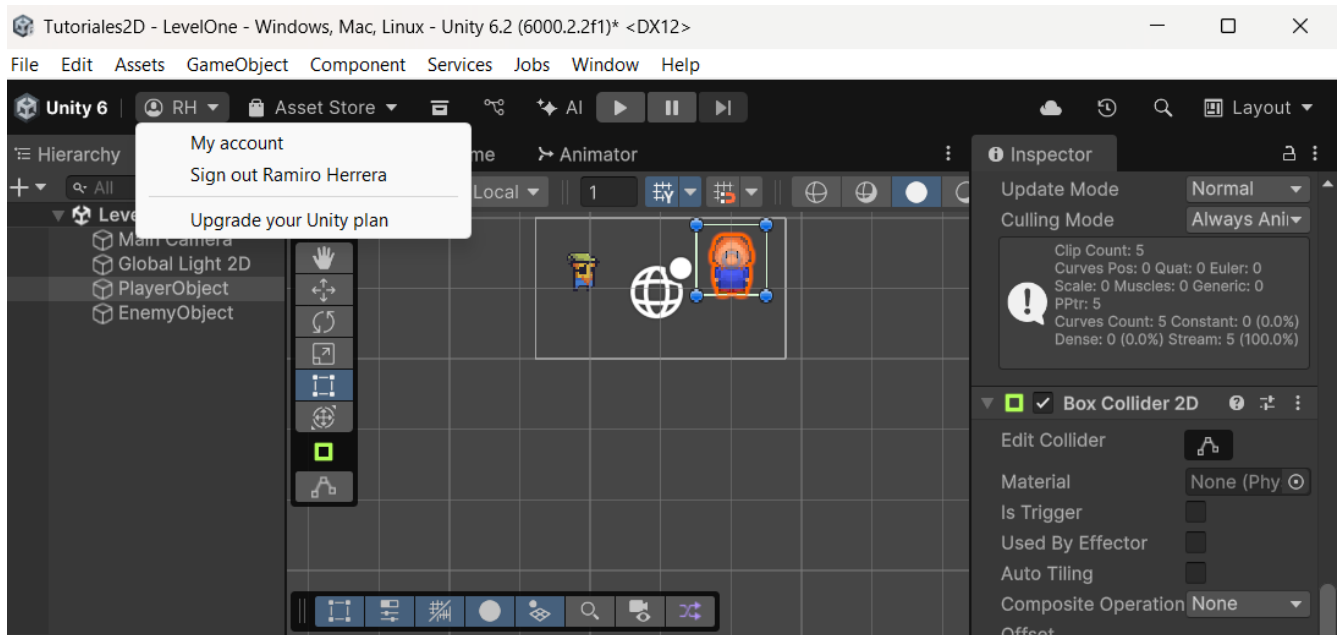
TABLA DE CONTENIDO

Tutorial 02	1
-------------------	---

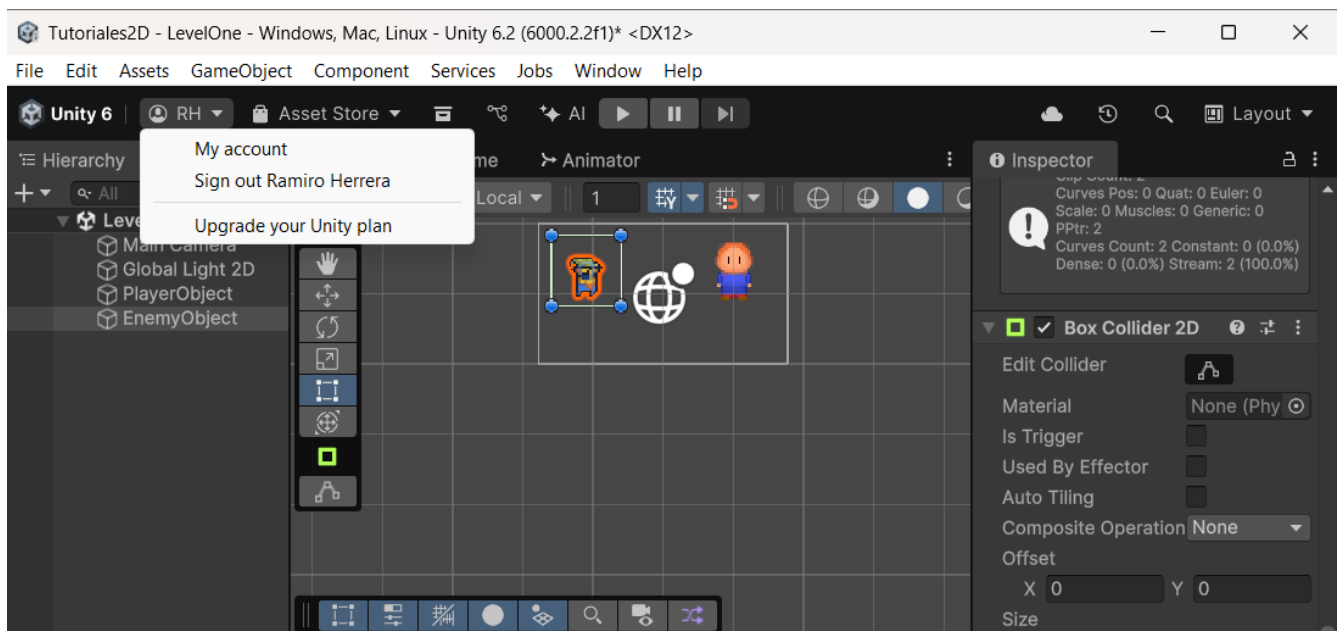
Tutorial 02 - ✂ PROCEDIMIENTO - PASO A PASO ✂

COLISIONADORES (COLLIDERS)

Paso 1: Seleccione **PlayerObject** y luego seleccione el botón **Add Component** en la ventana del inspector. Busque y seleccione "**Box Collider 2D**" para agregar un Box Collider 2D al PlayerObject.

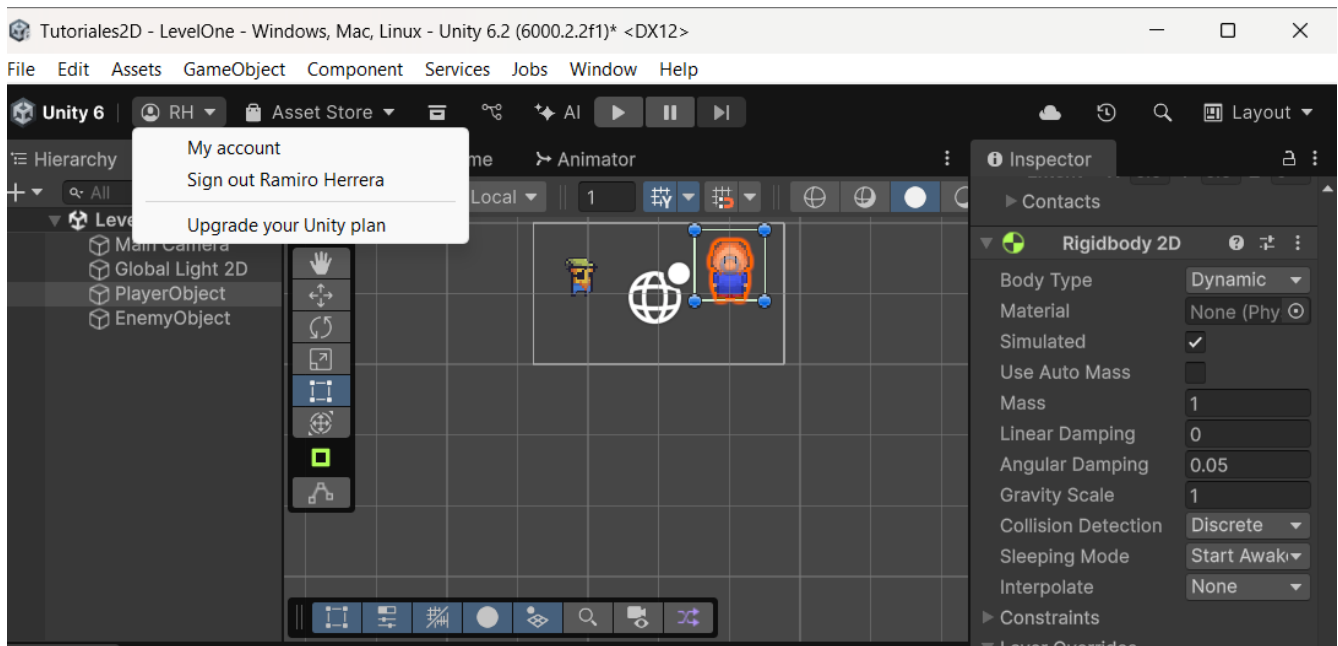


Paso 2: Necesitaremos saber cuándo el jugador choca con un enemigo, así que agregue un **Box Collider 2D** al **EnemyObject** también.



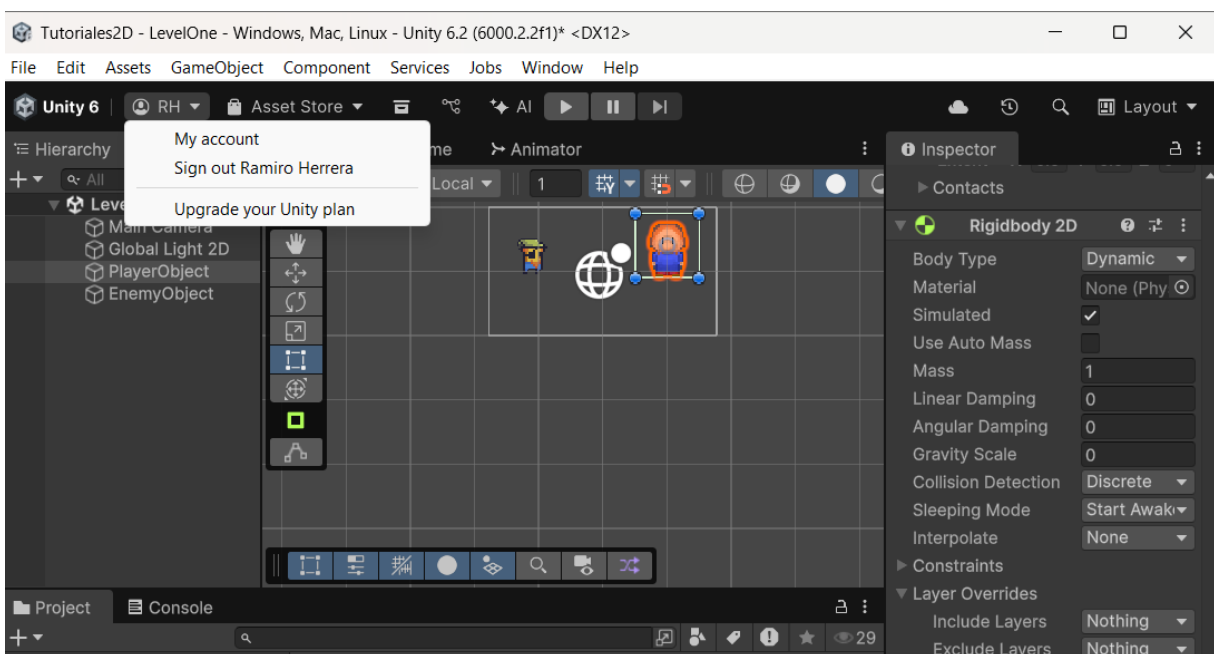
COMPONENTE RIGIDBODY 2D

Paso 3: Con **PlayerObject** seleccionado, haga clic en el botón **"Add component"** en la ventana Inspector, busque **"Rigidbody 2D"** y agréguelo al PlayerObject.

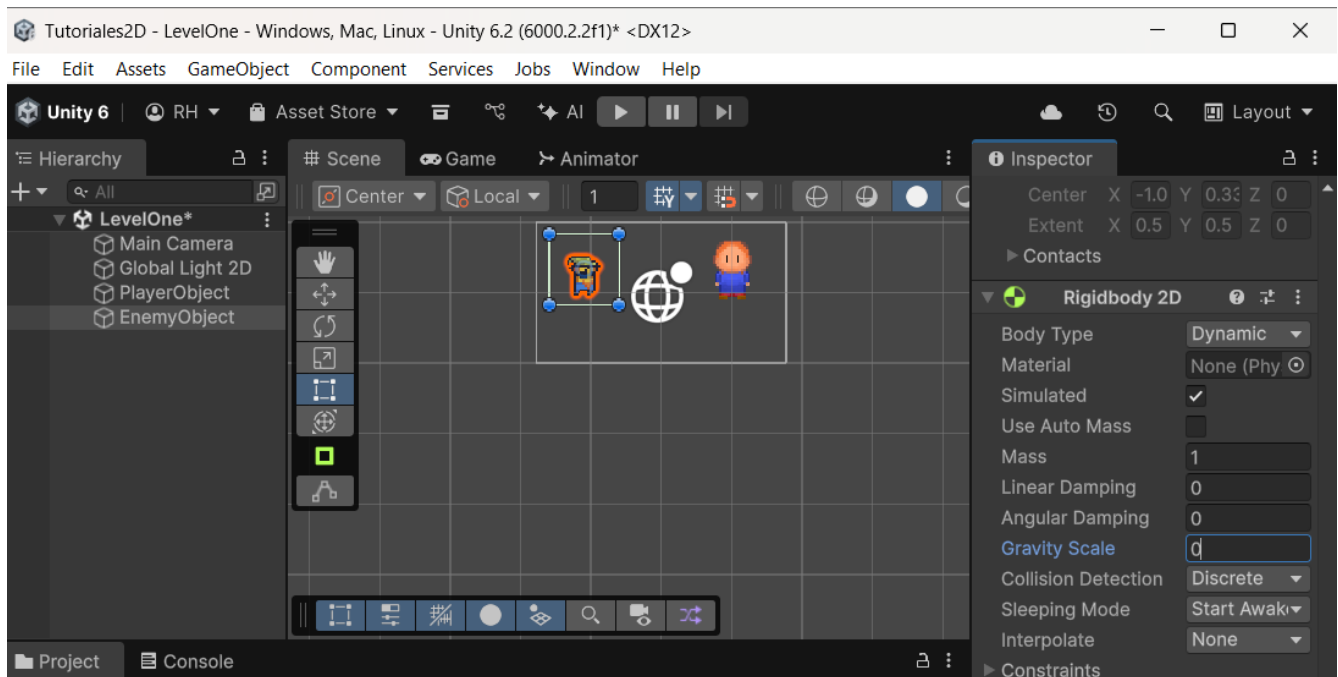


Paso 4: En el menú desplegable establezca los valores a las siguientes propiedades:

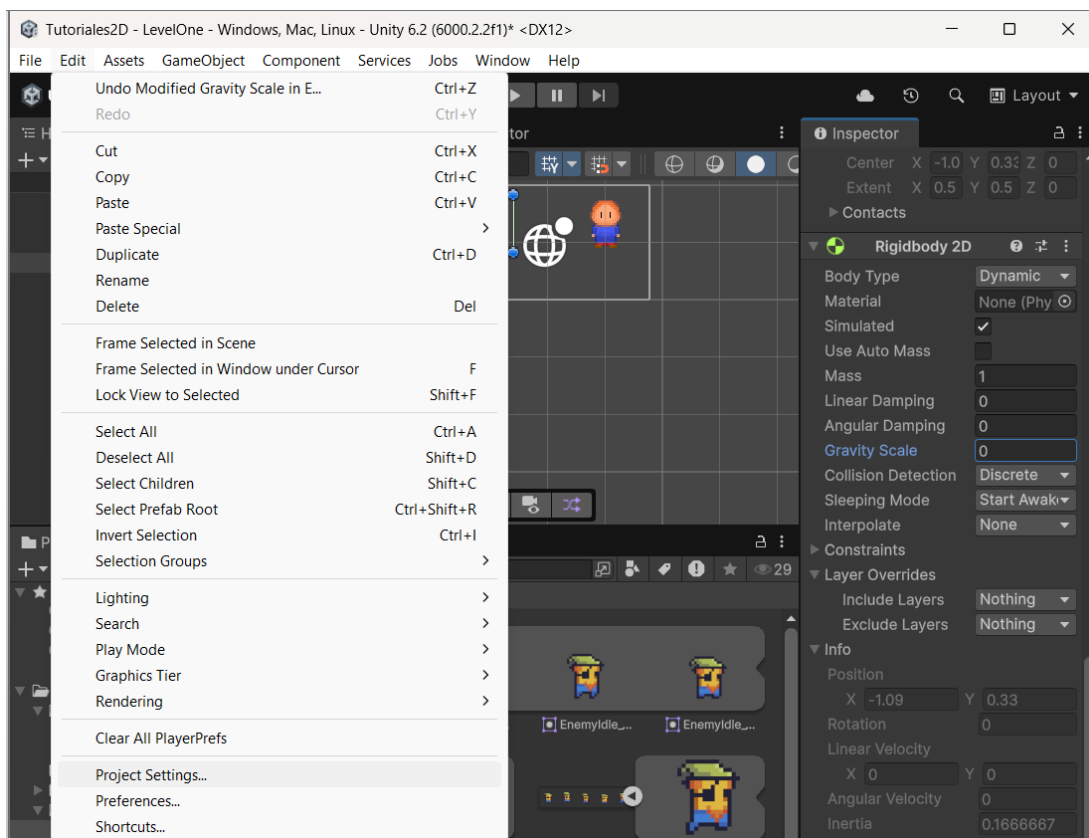
- **Body Type:** Dynamic
- **Mass:** 1
- **Linear Drag:** 0
- **Angular Drag:** 0
- **Gravity Scale:** 0

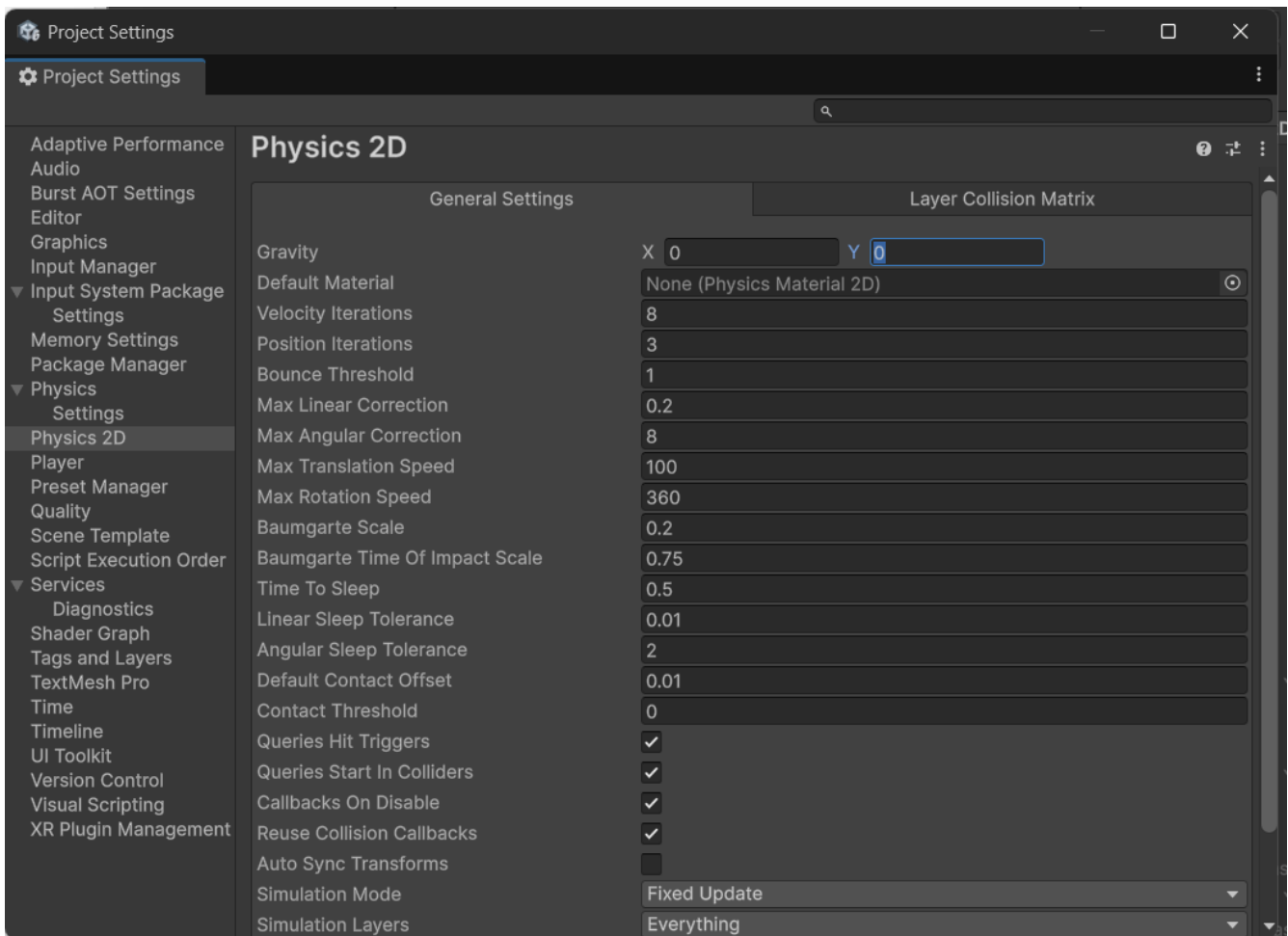


Paso 5: Seleccione el **EnemyObject** y agregue un Componente **Rigidbody 2D** de tipo **Dinámico** para él también con las mismas propiedades.



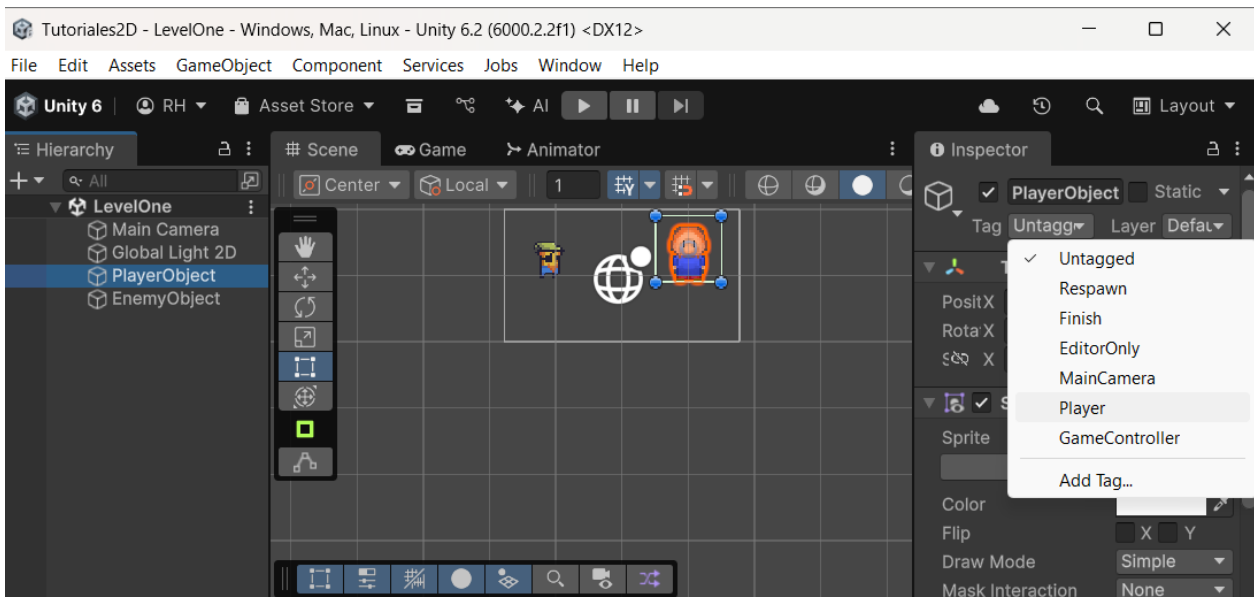
Paso 6: Vaya a la opción **Edit** → **Project settings** → **Physics 2D** y cambie el valor para la gravedad **Y** de **-9,81** a **0**.



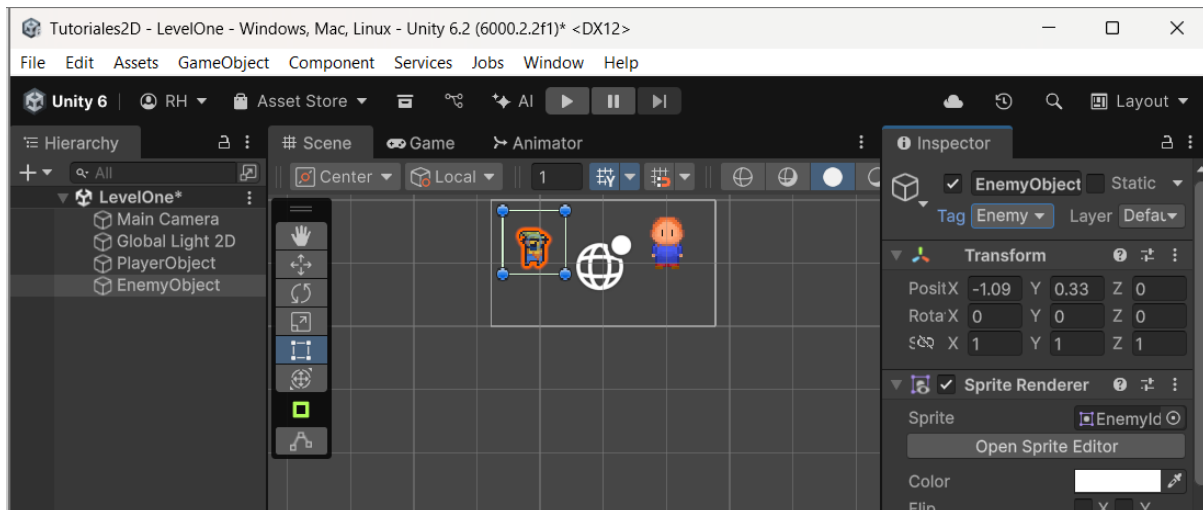
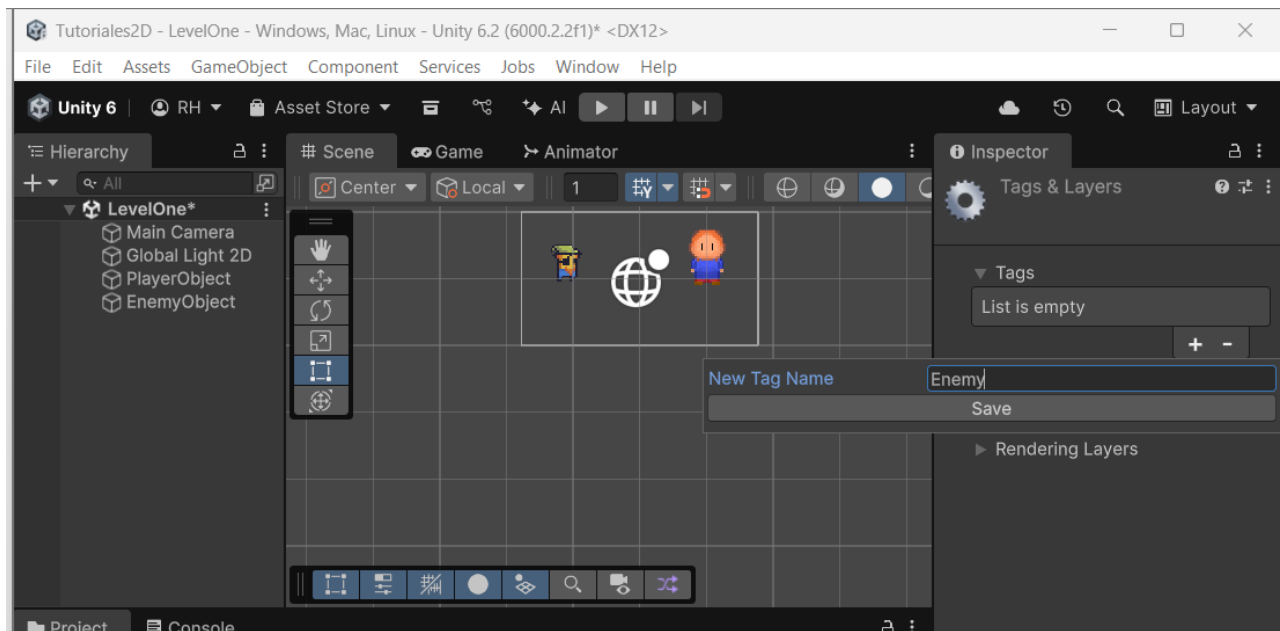
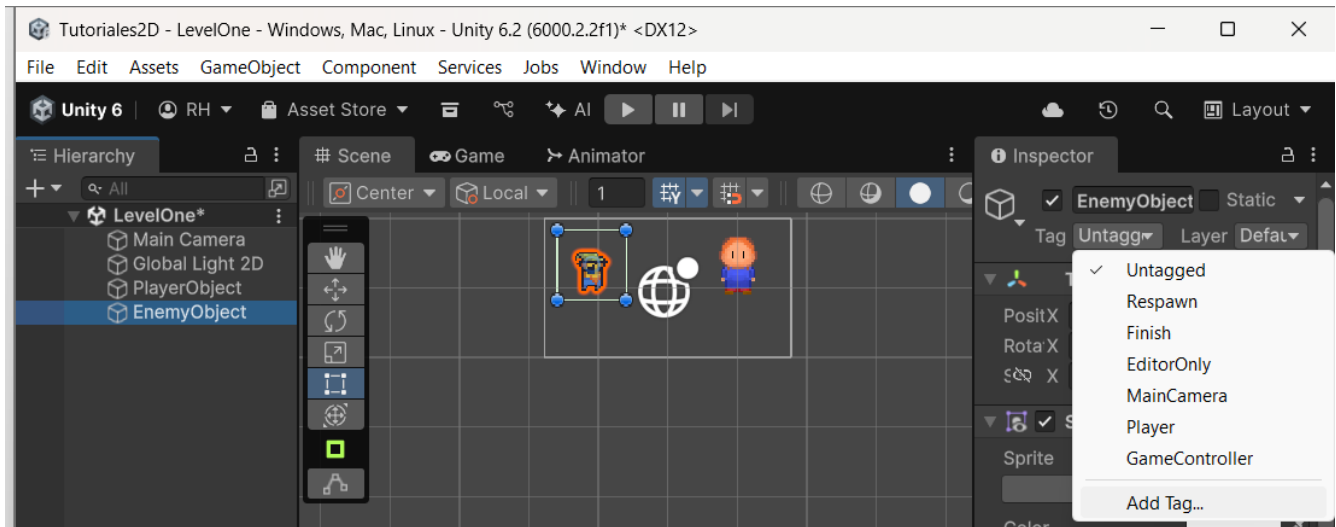


ETIQUETAS Y CAPAS

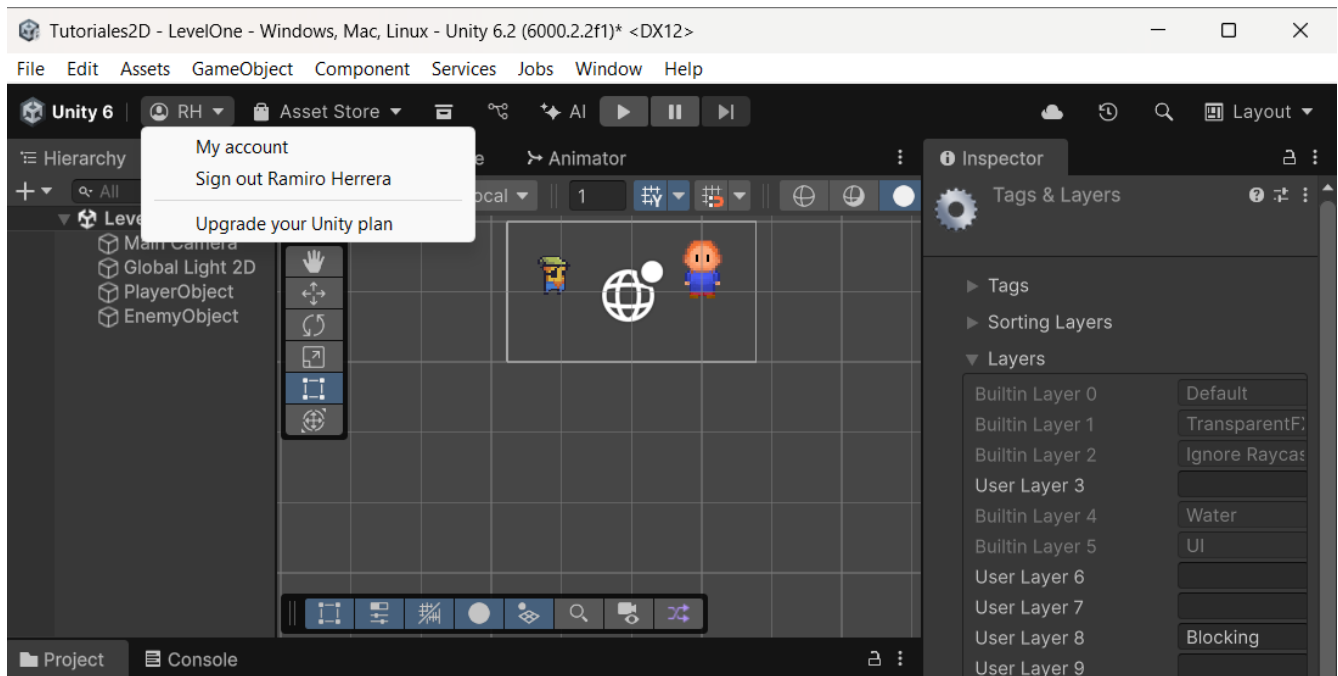
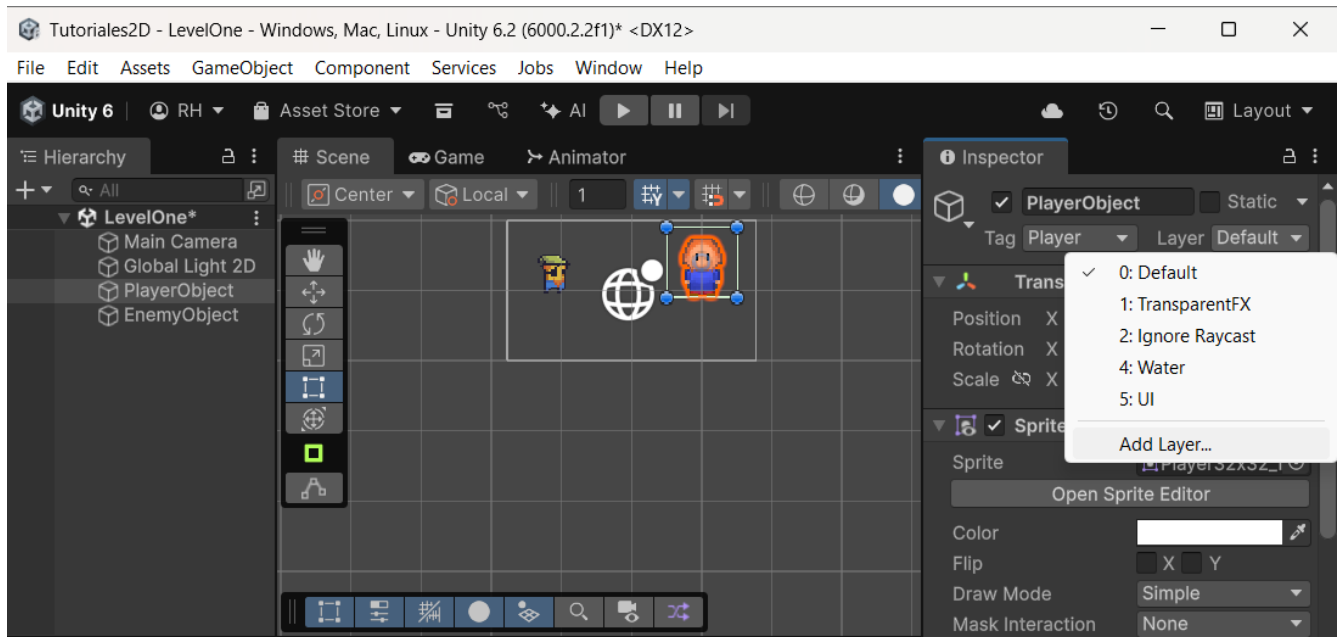
Paso 7: Selecciona el **PlayerObject**. En el menú desplegable **Tag** en la parte superior izquierda del Inspector, seleccione la etiqueta de **Jugador** para agregar una etiqueta a nuestro PlayerObject.



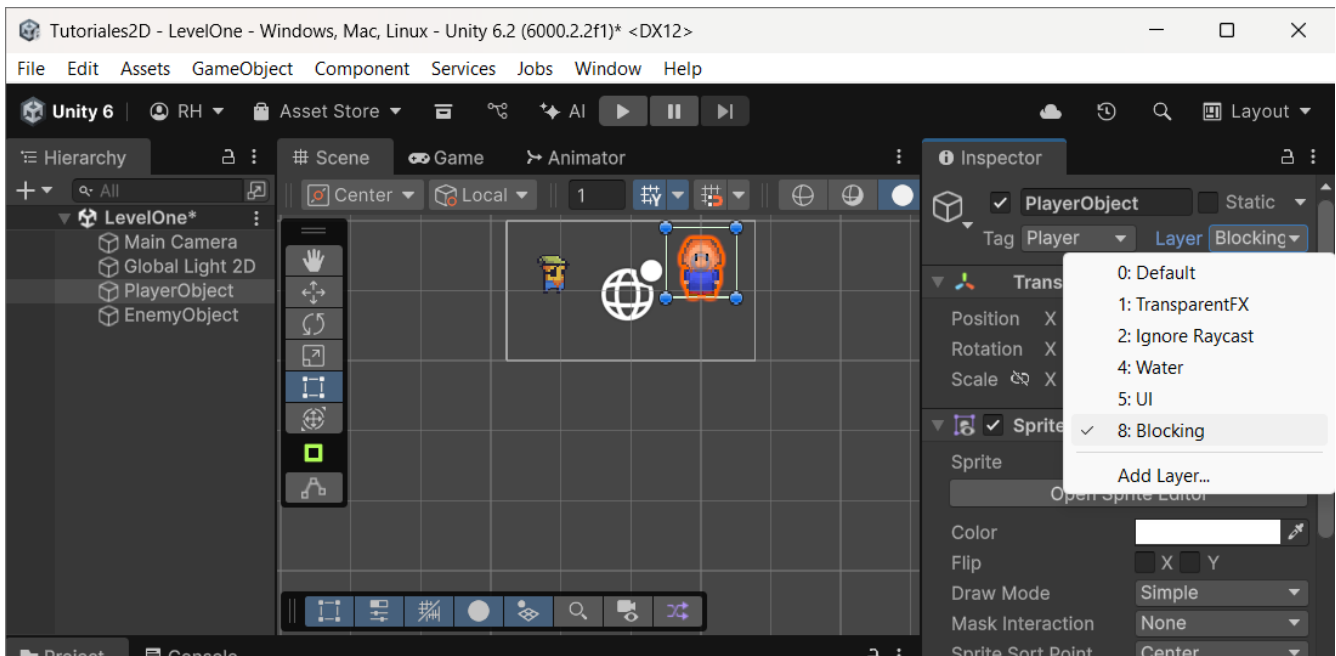
Paso 8: Crear una nueva etiqueta llamada "Enemy" y úsala para configurar la etiqueta del objeto **EnemyObject**.



Paso 9: Seleccionar en el menú desplegable **Layers** y seleccionar **"Add layer"**. Debería ver la ventana Layers.

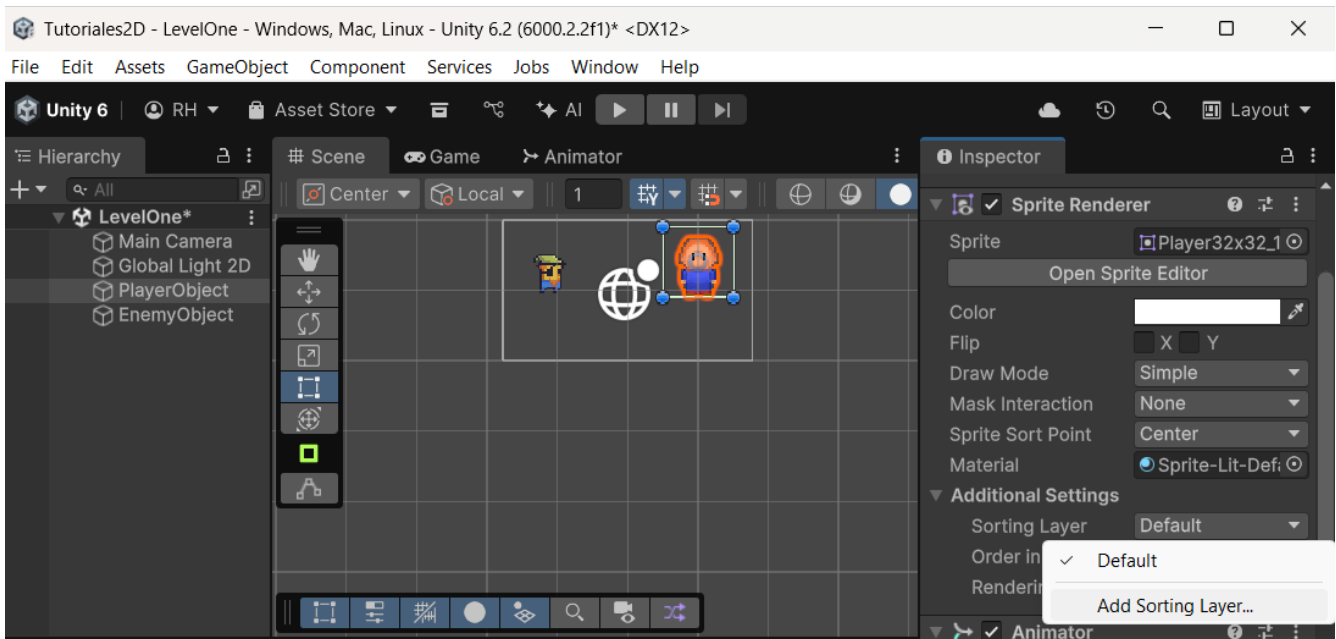


Paso 10: Ahora seleccione el **PlayerObject** nuevamente para ver sus propiedades en el Inspector. Seleccione la capa de **bloqueo** que acabamos de crear en el menú desplegable.



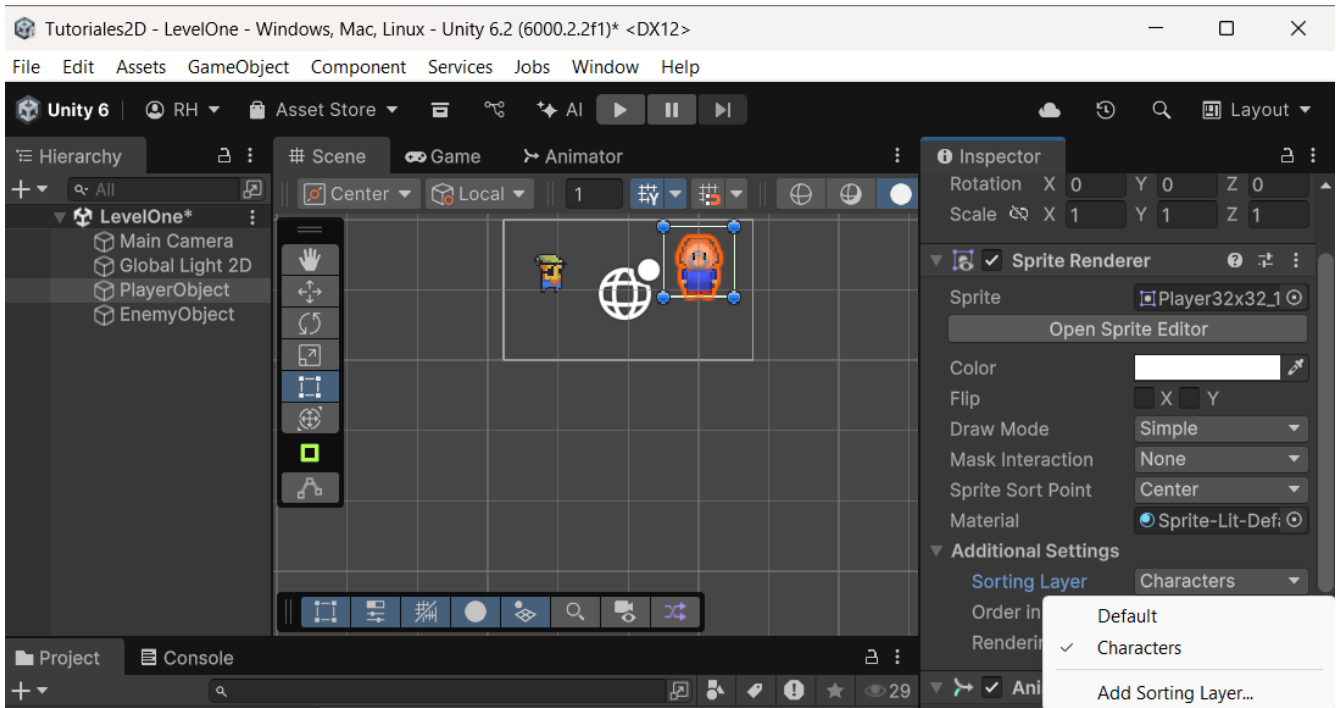
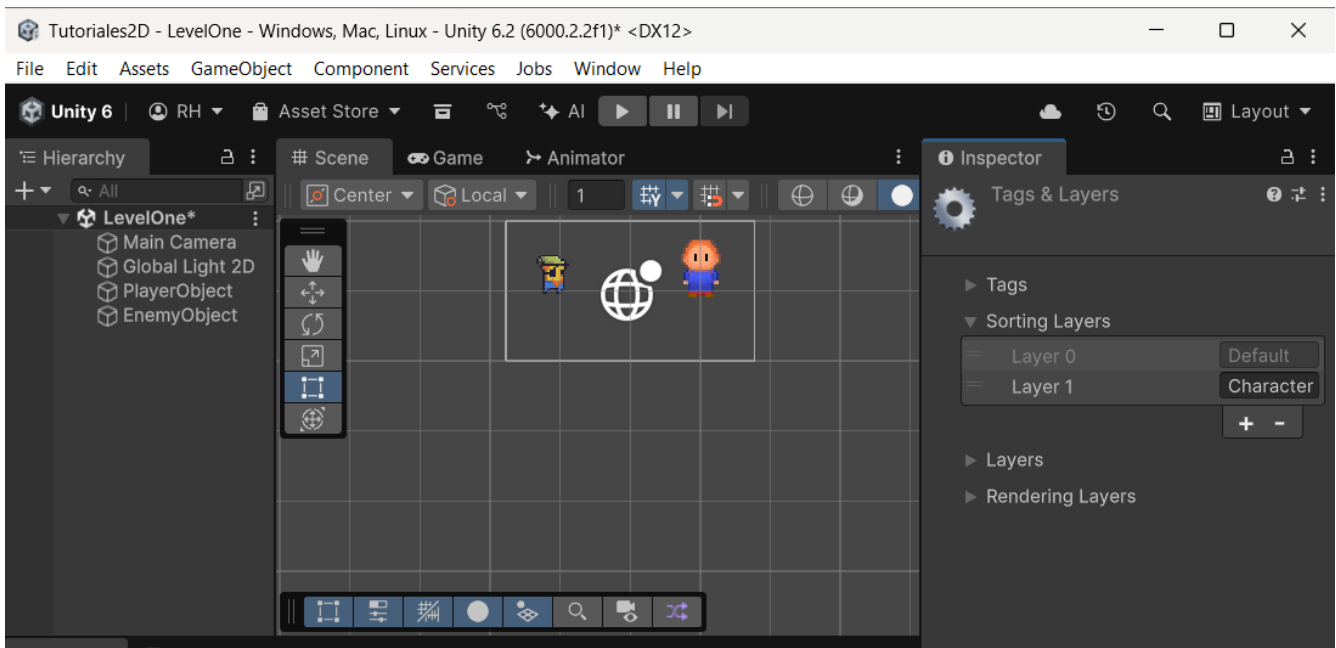
SORTING LAYERS

Paso 11: En el componente **Sprite Renderer** en la ventana Inspector, seleccione la opción **Sorting layer** Menú desplegable de capa y seleccione **"Add Sorting Layer"**.



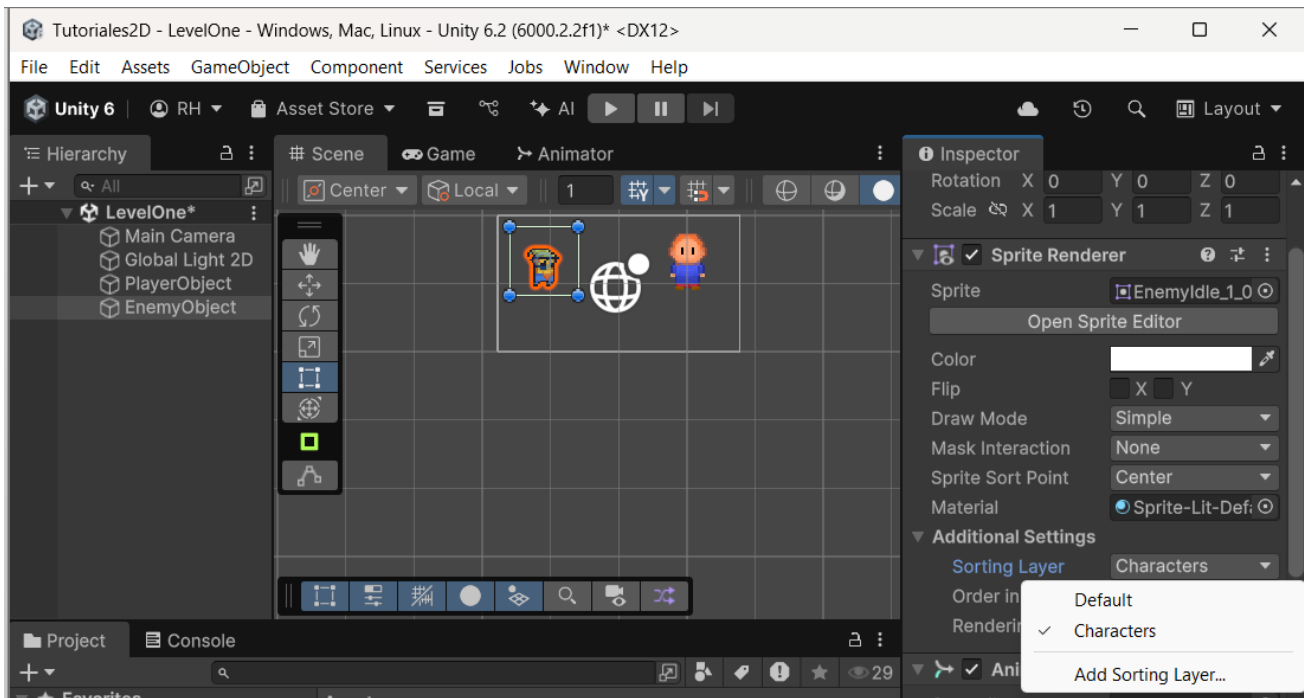
Paso 12: Agregue una capa de ordenamiento llamada **"Characters"**, y luego haga clic en **PlayerObject** nuevamente para ver su Inspector y seleccione la nueva **Capa de ordenamiento** de caracteres del menú desplegable.

Unidad I. Programación orientada a objetos para videojuegos



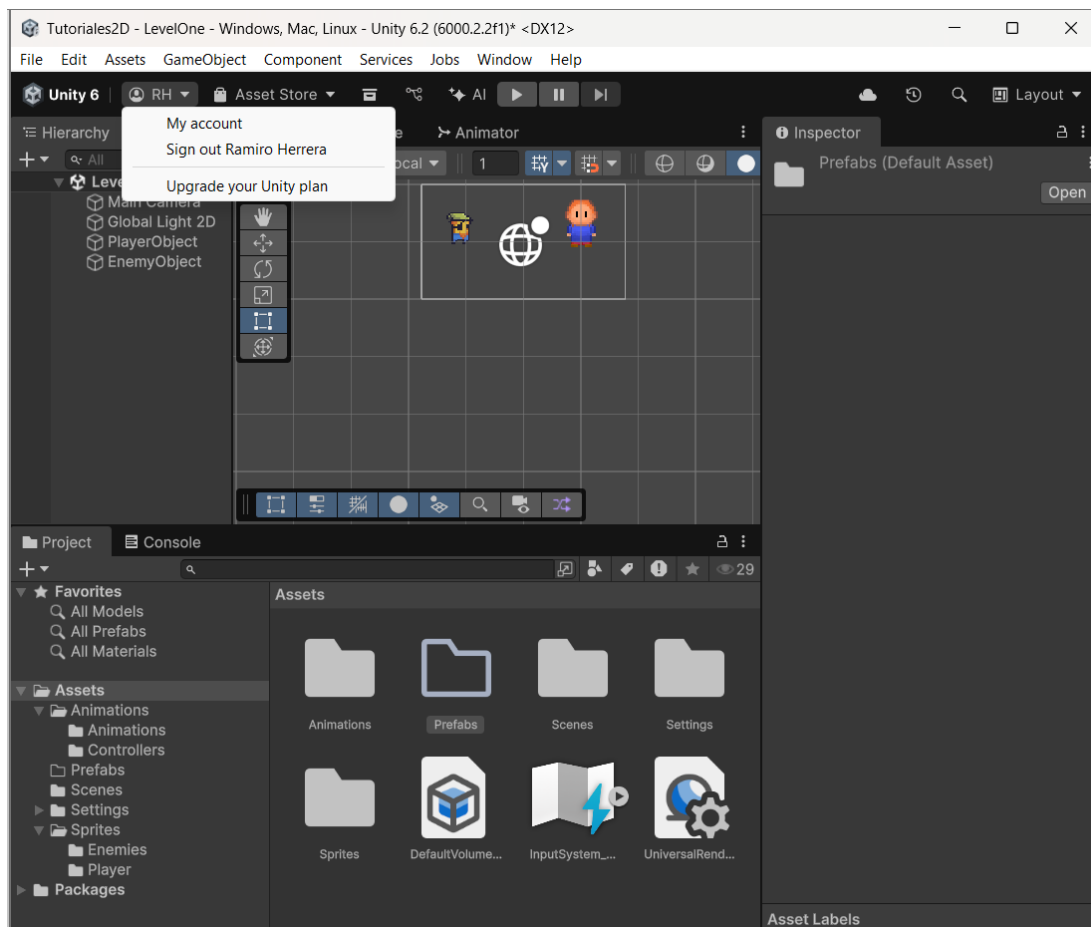
Paso 13: Seleccione nuestro **EnemyObject** y establezca su **Capa de ordenamiento** en **Characters**, porque queremos que los enemigos también aparezcan en la parte superior de las cosas.

Unidad I. Programación orientada a objetos para videojuegos

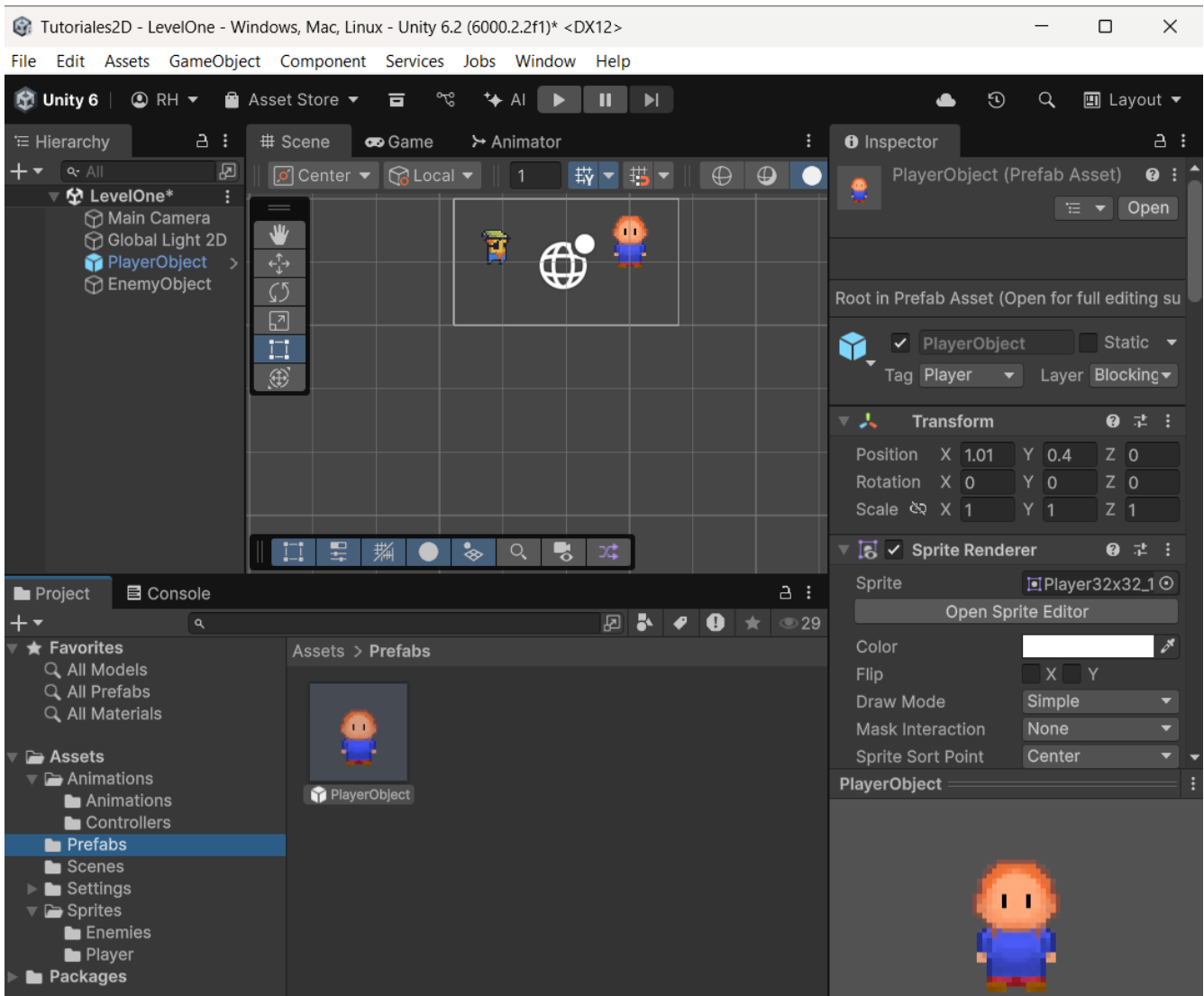


PREFABS

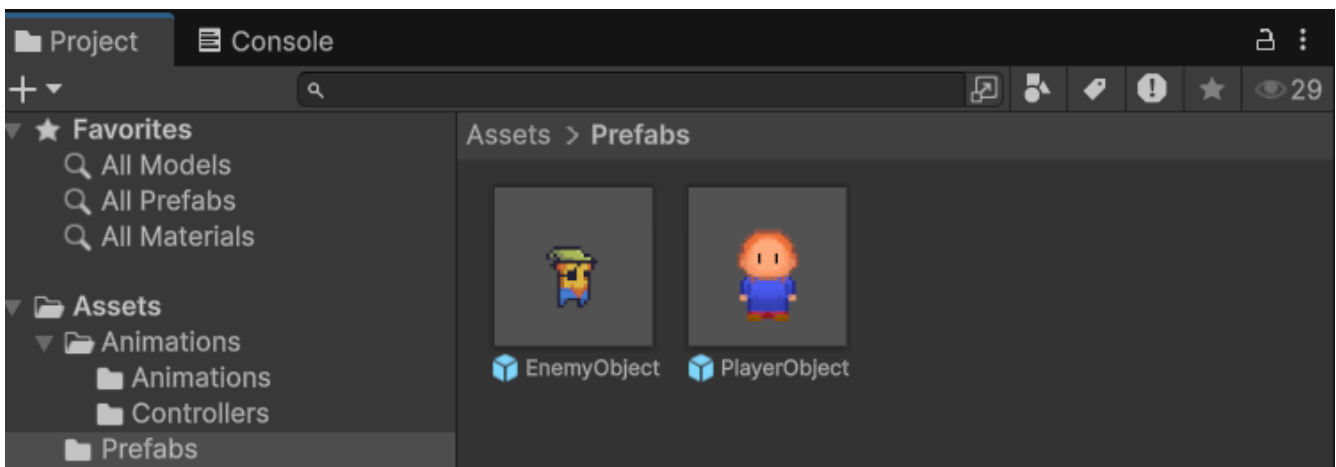
Paso 14: Primero, crea una carpeta llamada **"Prefabs"** en nuestra carpeta **Assets** en la vista Proyecto.



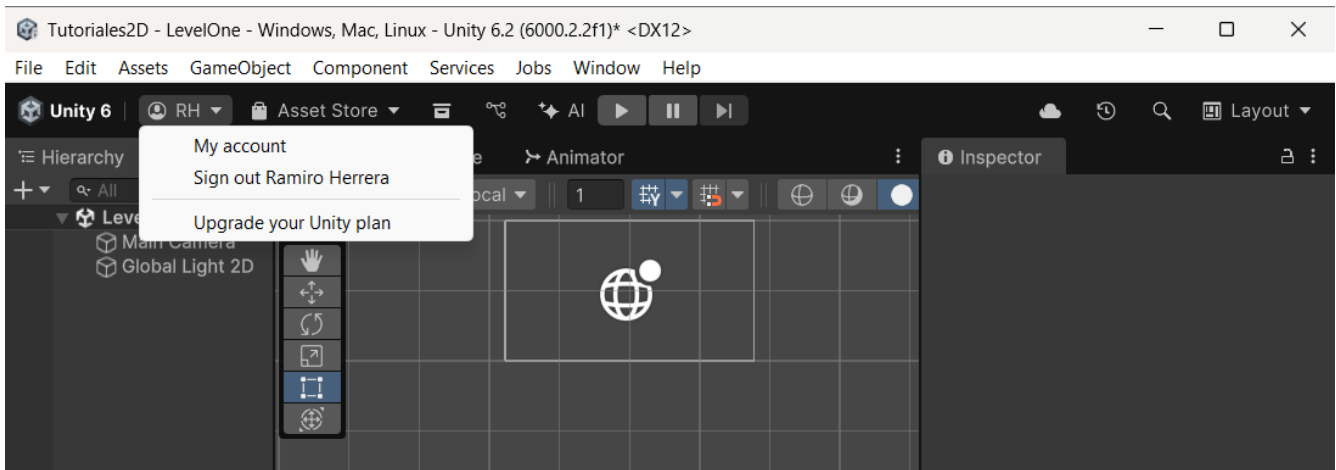
Paso 15: Luego seleccione nuestro **PlayerObject** de la vista de **Hierarchy** y simplemente arrástralo a la carpeta **Prefabs**.



Paso 16: Ahora puedes eliminar de forma segura el **PlayerObject** de la vista de Hierarchy.

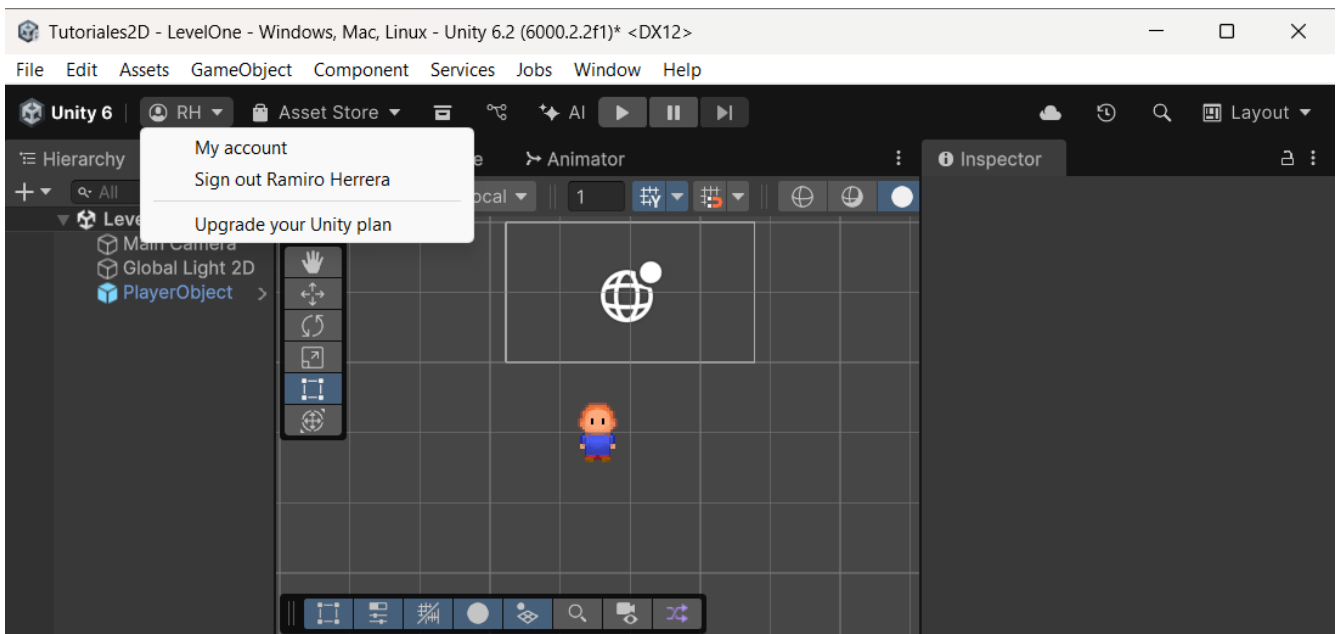


Paso 17: Haga lo mismo con **EnemyObject**: arrástralo a la carpeta **Prefabs** y eliminar el **EnemyObject** original de la vista Hierarchy.



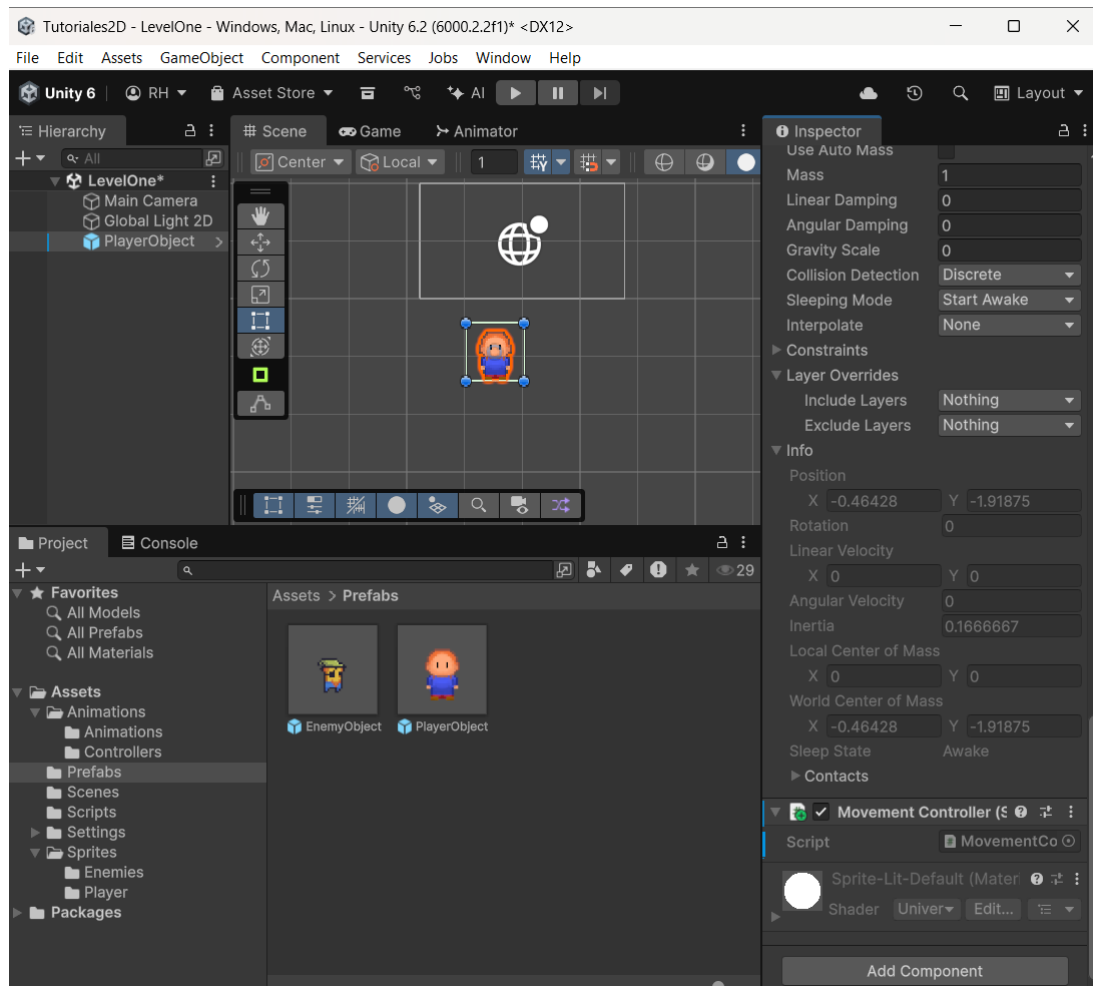
SCRIPTS: MOVIMIENTO DEL PERSONAJE

Paso 18: Seleccione nuestro **PlayerObject Prefab** y arrástralo a la vista Hierarchy.

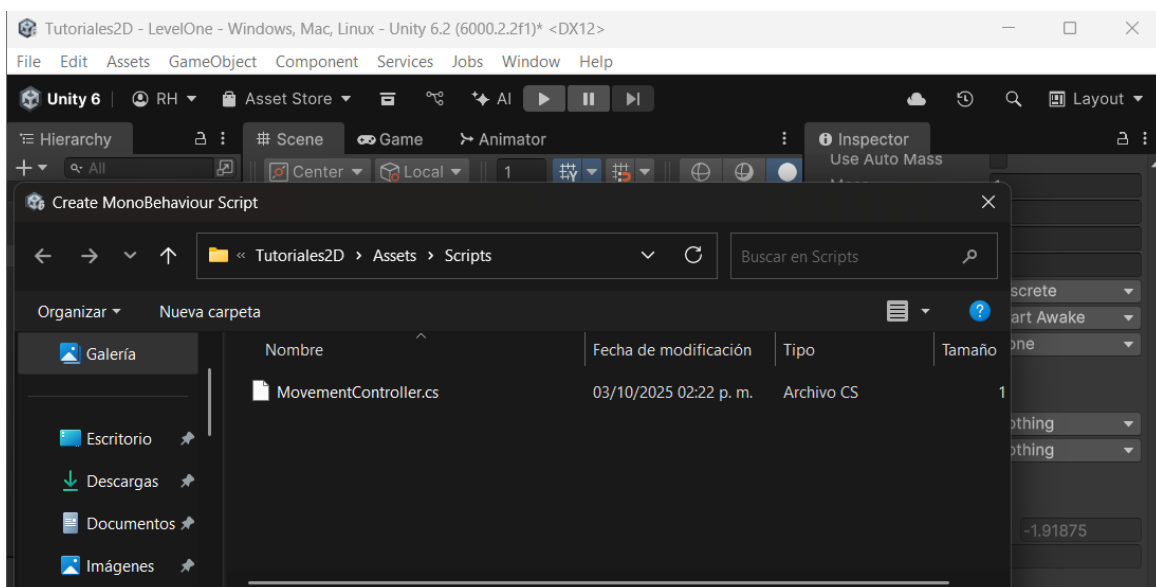


Paso 19: Desplácese hasta la parte inferior del Inspector y presione el botón **Add Component**. Escriba la palabra **Script** y seleccione **"New Script"** y nombrarlo como **"MovementController"**.

Unidad I. Programación orientada a objetos para videojuegos



Paso 20: Cree una nueva carpeta llamada **"Scripts"** en la vista Project. El nuevo script se habrá creado en la carpeta Assets de nivel superior en la vista Project. Arrastre el script **MovementController** a la carpeta **Scripts** y luego haga doble clic en él para abrirlo en Visual Studio.



Paso 21: Agreguemos las siguientes propiedades a la clase:

```
public class MovementController : MonoBehaviour
{
    //Velocidad de los personajes
    public float movementSpeed = 3.0f;
    //Representa la ubicación del Player o Enemy
    Vector2 movement = new Vector2();
    //Referencia a Rigidbody2D
    Rigidbody2D rb2D;
}
```

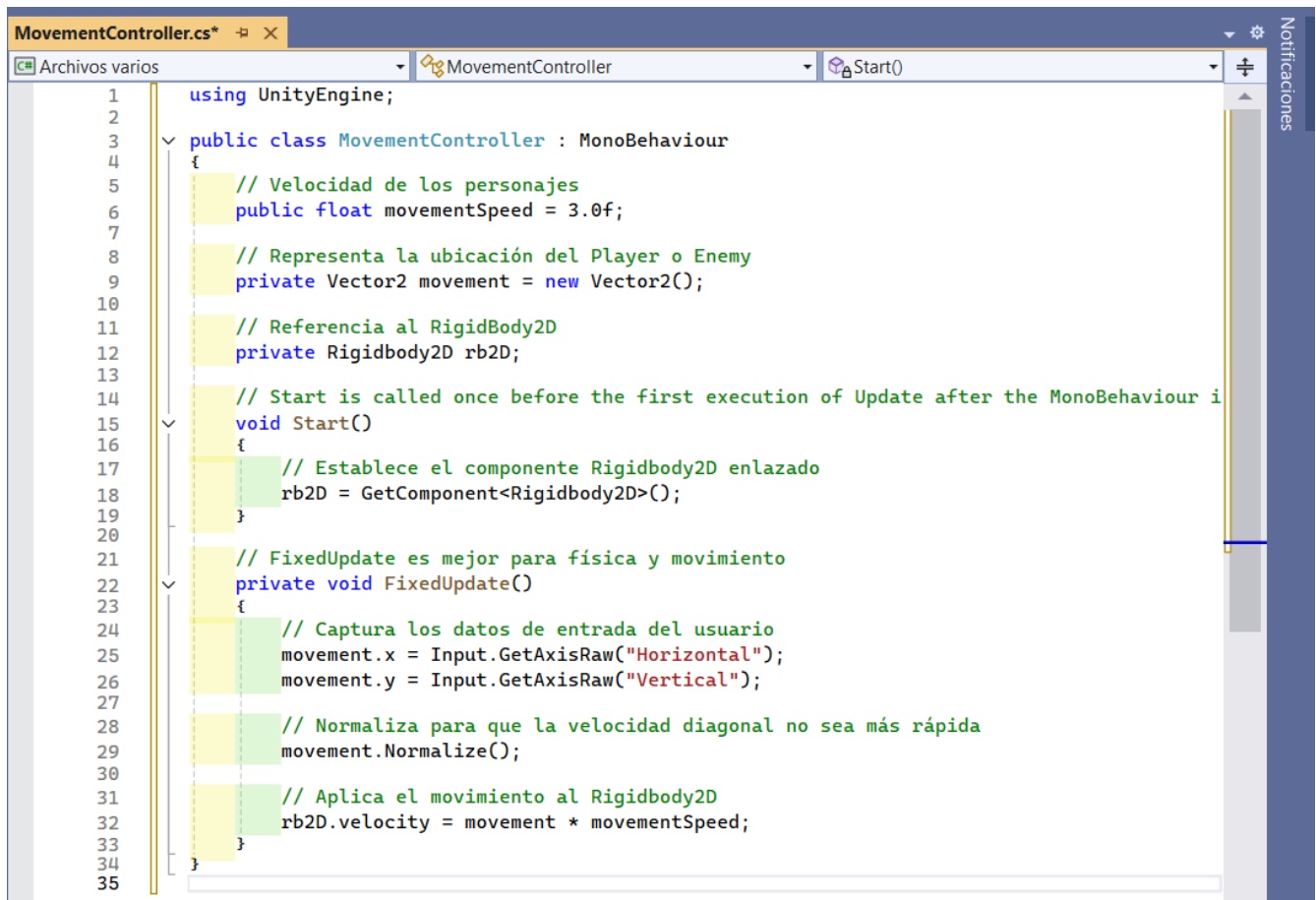
Paso 22: Establezcamos la referencia Rigidbody2D en el método Start:

```
void Start()
{
    //Establece el componente Rigidbody2D enlazado
    rb2D = GetComponent<Rigidbody2D>();
}
```

Paso 23: Programemos el método FixedUpdate:

```
private void FixedUpdate() {
    //Captura los datos de entrada del usuario
    movement.x = Input.GetAxisRaw("Horizontal");
    movement.y = Input.GetAxisRaw("Vertical");

    //Conserva el rango de velocidad
    movement.Normalize();
    rb2D.velocity = movement * movementSpeed;
}
```



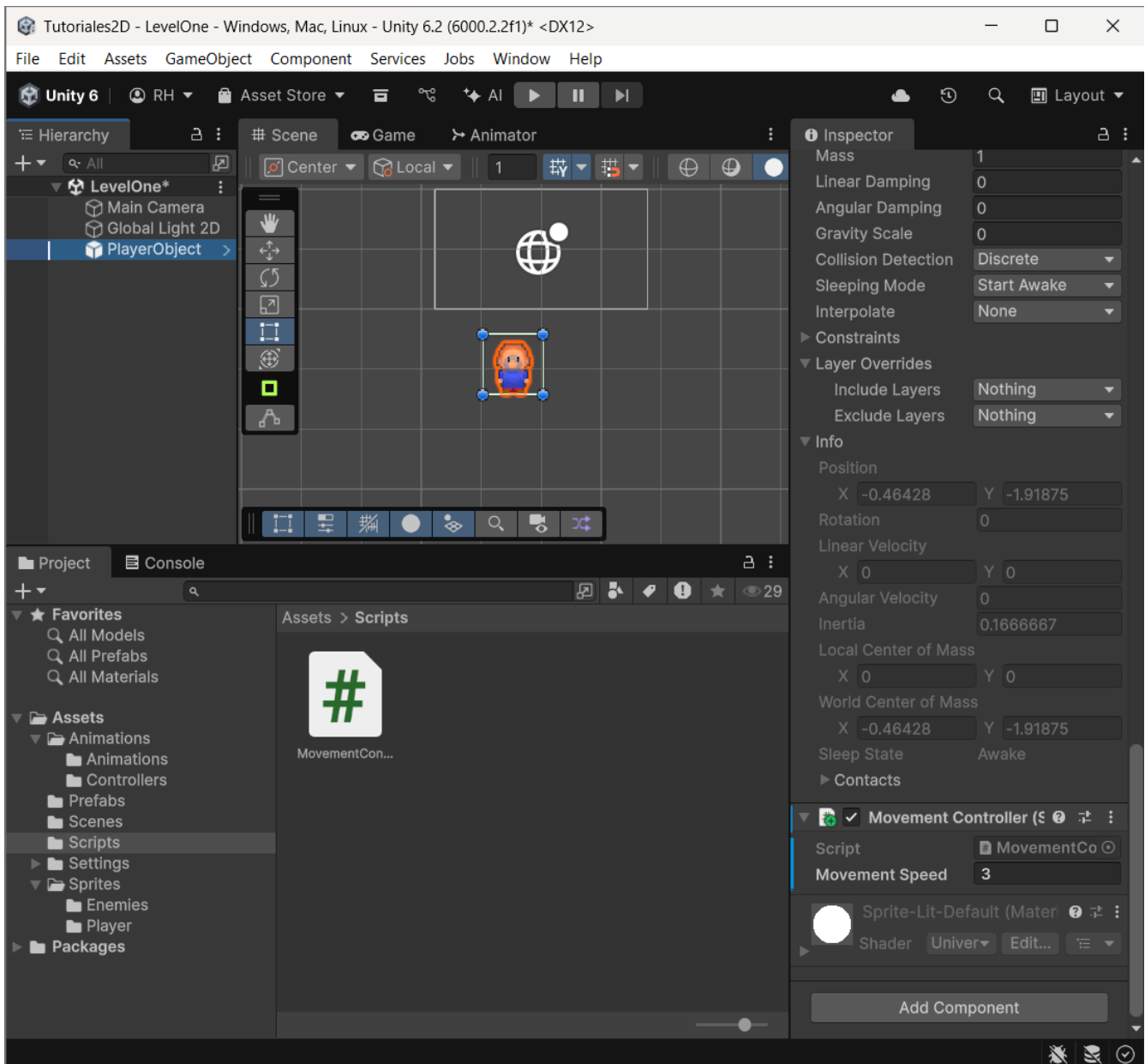
```

1  using UnityEngine;
2
3  public class MovementController : MonoBehaviour
4  {
5      // Velocidad de los personajes
6      public float movementSpeed = 3.0f;
7
8      // Representa la ubicación del Player o Enemy
9      private Vector2 movement = new Vector2();
10
11     // Referencia al Rigidbody2D
12     private Rigidbody2D rb2D;
13
14     // Start is called once before the first execution of Update after the MonoBehaviour i
15     void Start()
16     {
17         // Establece el componente Rigidbody2D enlazado
18         rb2D = GetComponent<Rigidbody2D>();
19     }
20
21     // FixedUpdate es mejor para física y movimiento
22     private void FixedUpdate()
23     {
24         // Captura los datos de entrada del usuario
25         movement.x = Input.GetAxisRaw("Horizontal");
26         movement.y = Input.GetAxisRaw("Vertical");
27
28         // Normaliza para que la velocidad diagonal no sea más rápida
29         movement.Normalize();
30
31         // Aplica el movimiento al Rigidbody2D
32         rb2D.velocity = movement * movementSpeed;
33     }
34 }
35

```

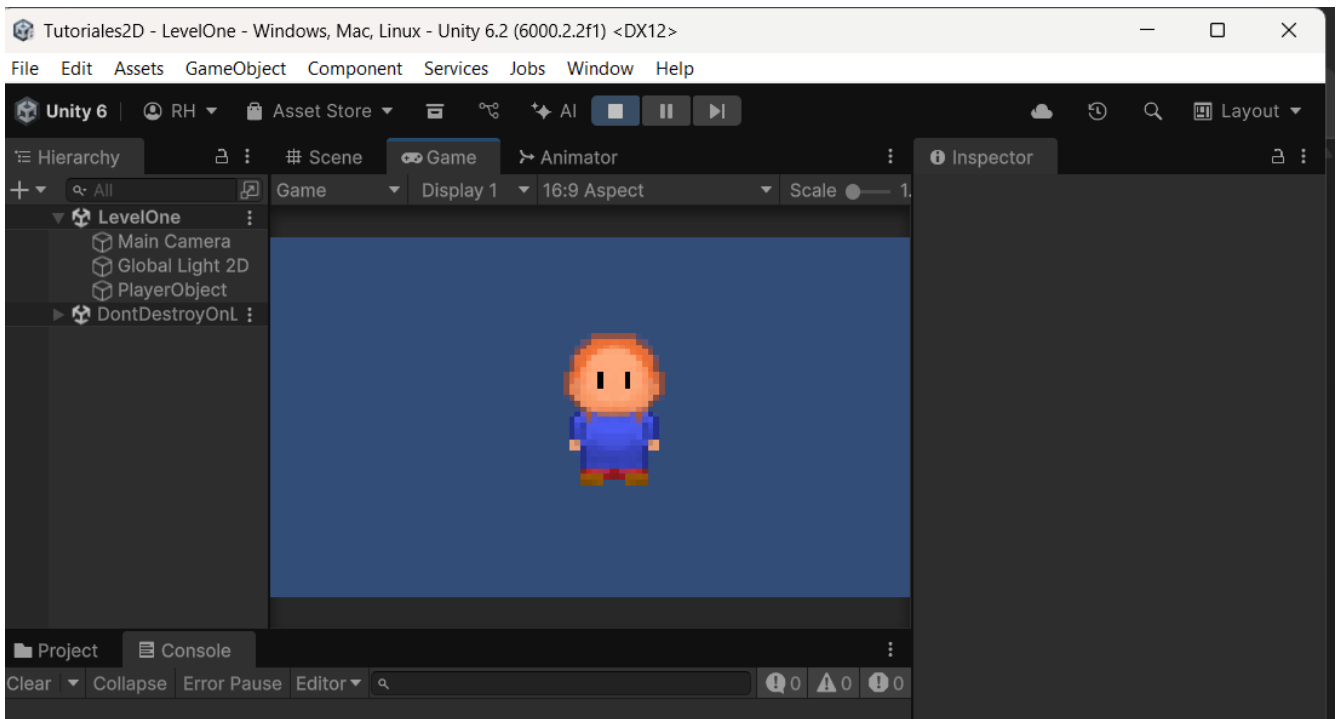
Paso 24: Regrese al Editor de Unity y asegúrese de ver nuestro PlayerObject en la vista de Hierarchy. Para agregar el script a nuestro PlayerObject, arrastre el **MovementController script** de la carpeta Scripts, en PlayerObject en la jerarquía, o arrástralo al Inspector cuando el PlayerObject esté seleccionado.

Unidad I. Programación orientada a objetos para videojuegos

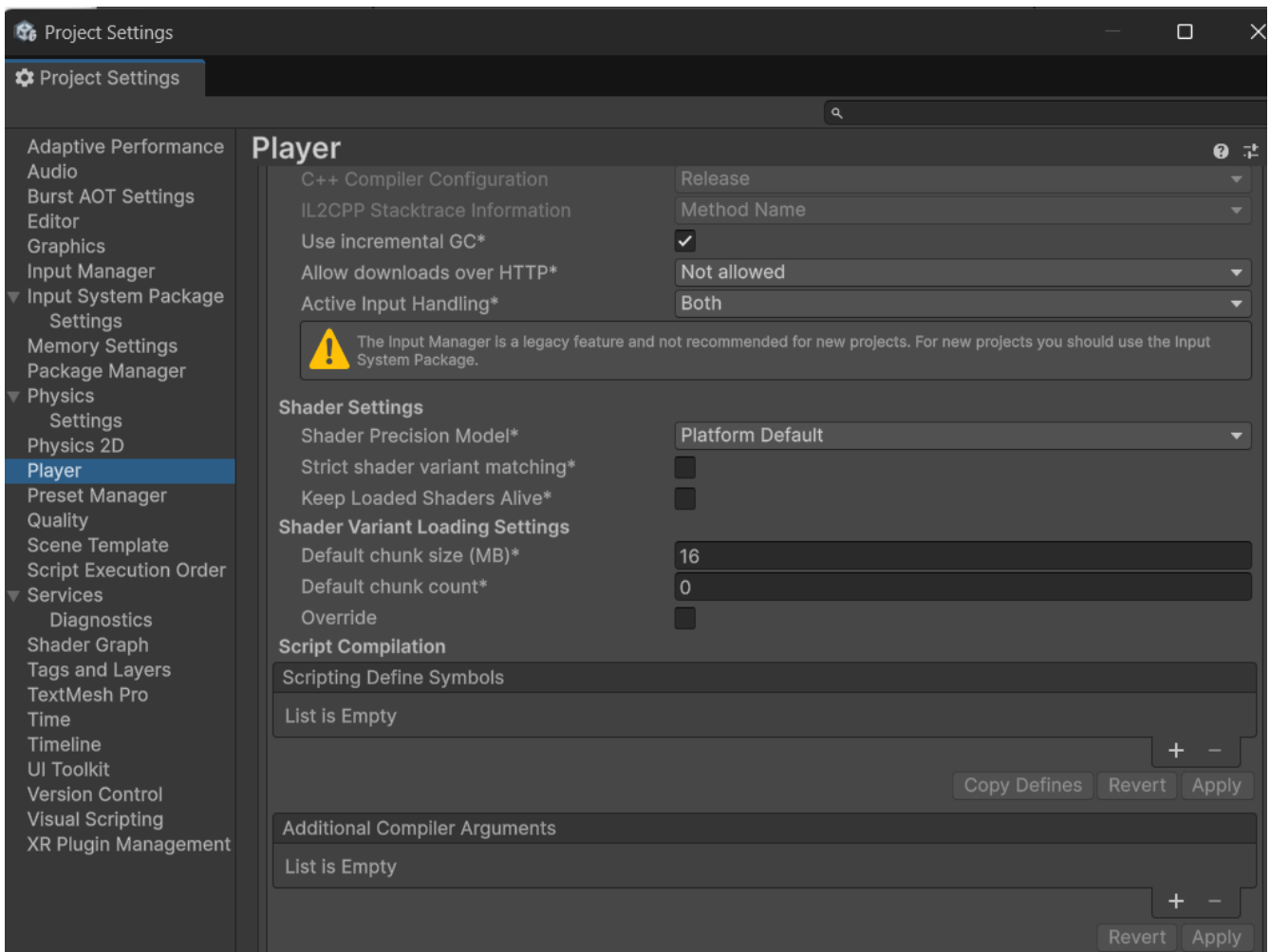


Paso 25: Ahora presione el botón **Play**. Deberías ver a nuestro personaje de jugador caminando en su lugar. Presione las teclas de flecha o **W**, **A**, **S**, **D** en su teclado y mírala moverse.

Unidad I. Programación orientada a objetos para videojuegos



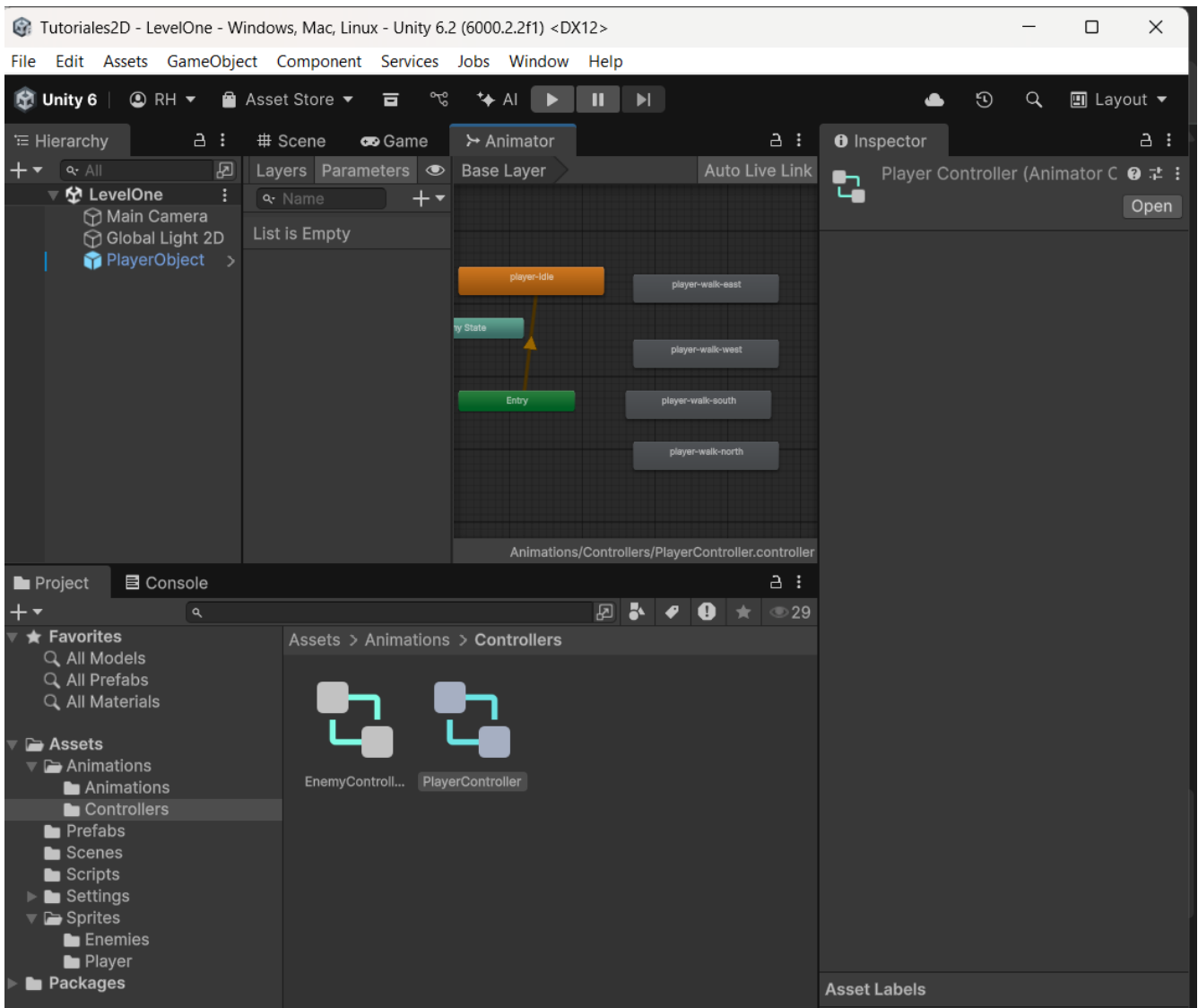
Si no funciona correctamente, es probable que sea un error en el código al momento de llamar los componentes/acciones y debas de configurar lo siguiente en Edit – Player – Active Input Handling



MÁQUINA DE ESTADOS DEL ANIMATOR

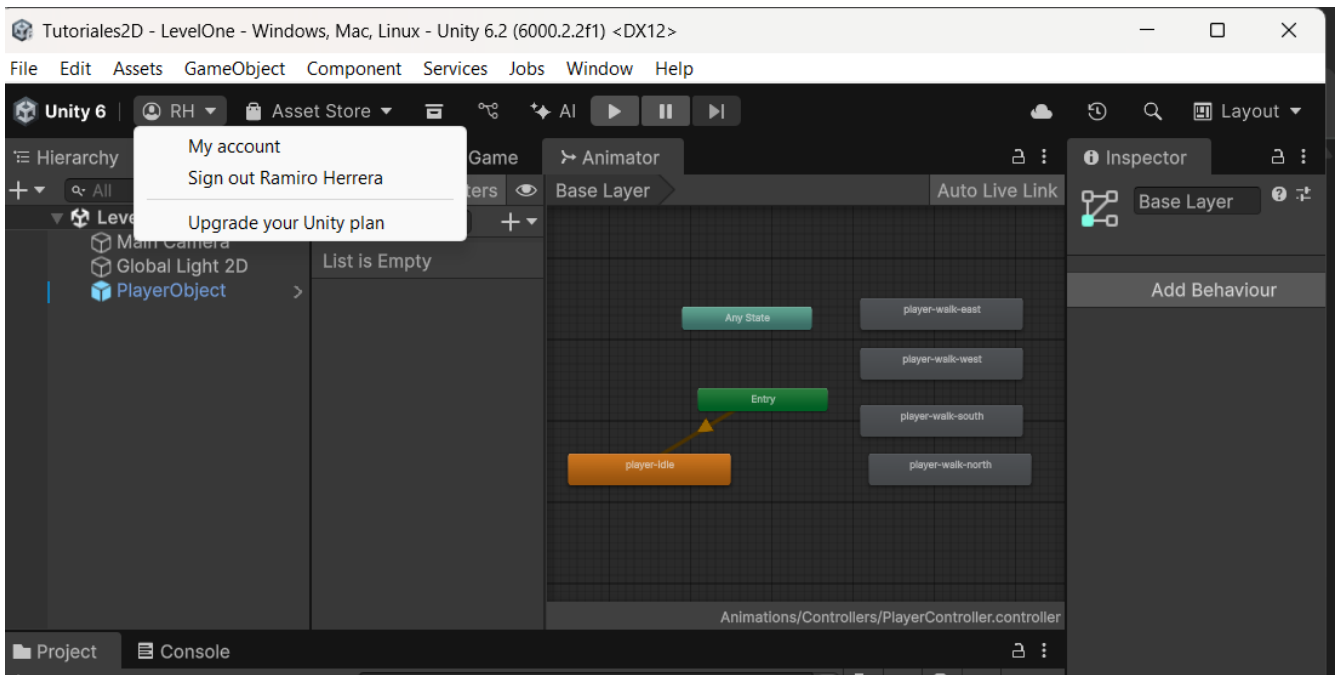
Paso 26: Vaya a la carpeta **Animations** → **Controllers** y haga doble clic en el Objeto **PlayerController**.

Unidad I. Programación orientada a objetos para videojuegos

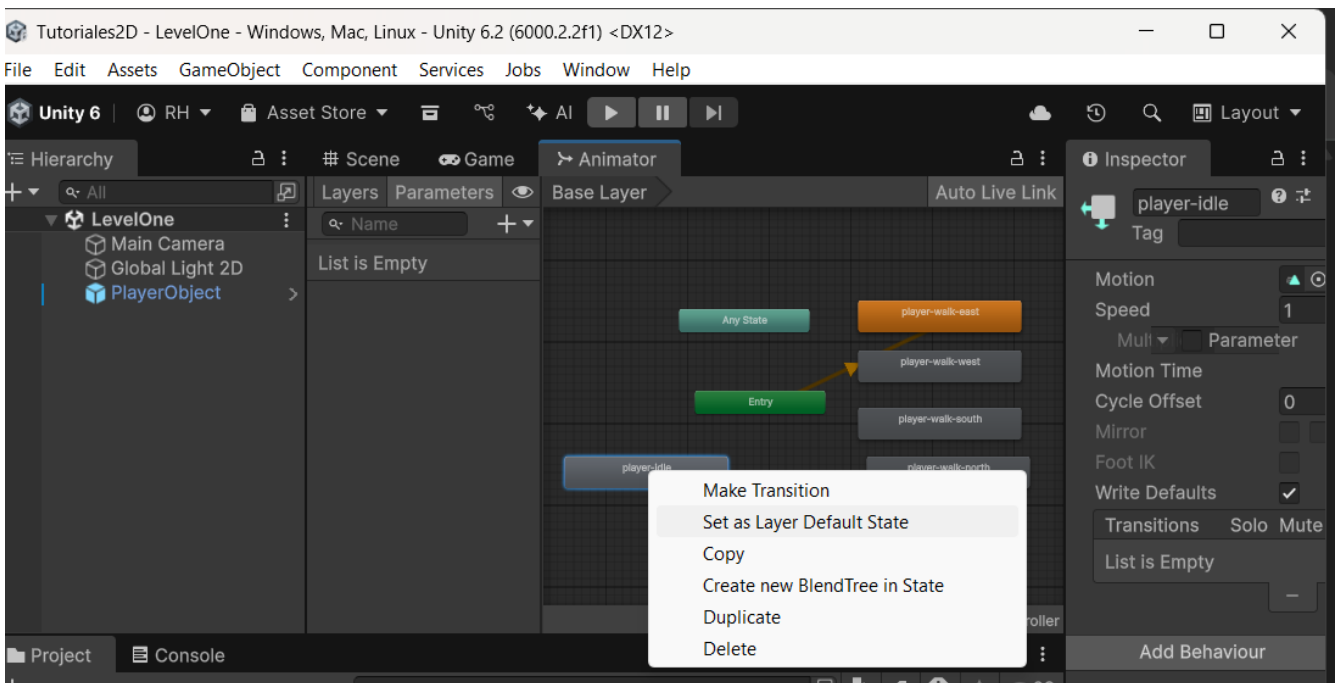


Paso 27: Haga clic y arrastre sus objetos de estado de animación hasta que aparezcan en la pantalla, con el reproductor inactivo a un lado, y las animaciones de caminatas de jugador agrupadas.

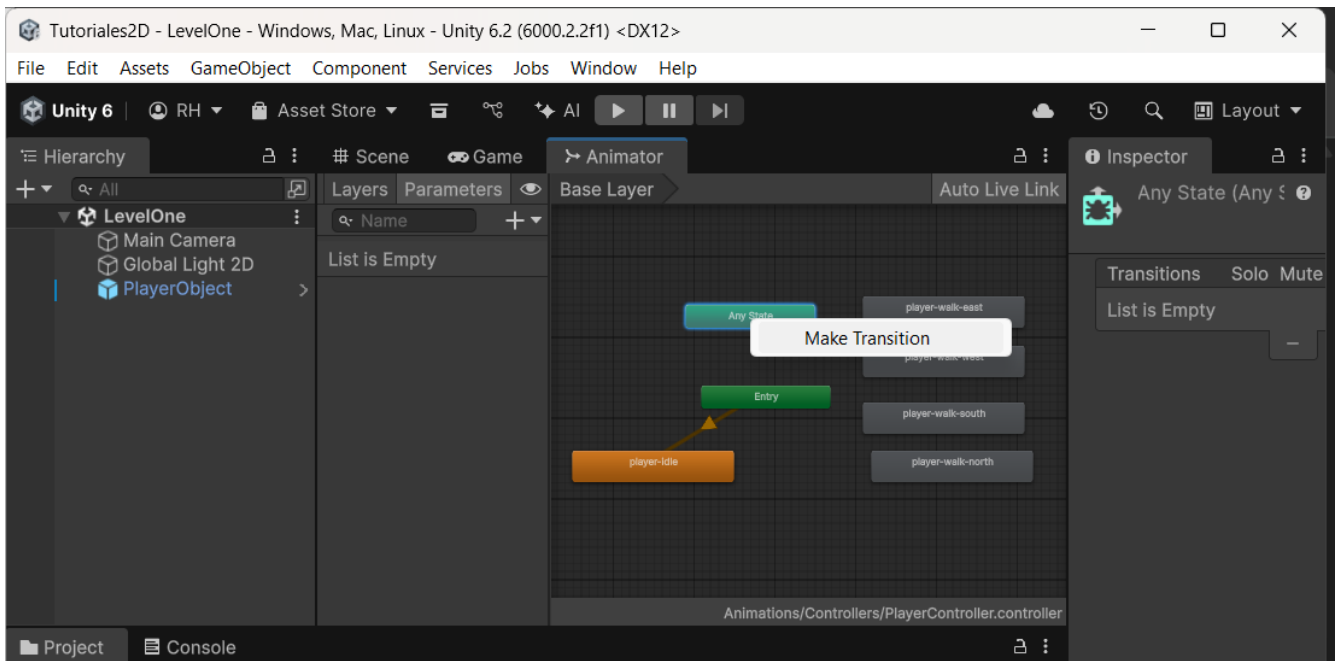
Unidad I. Programación orientada a objetos para videojuegos



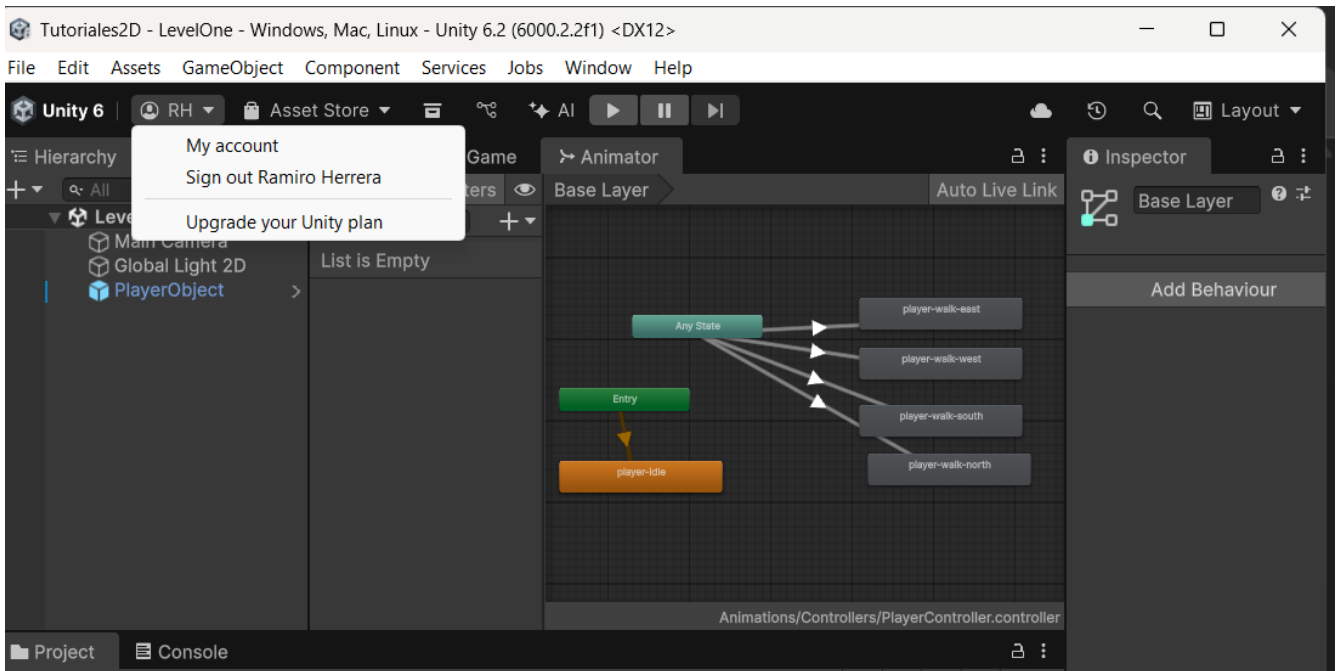
Paso 28: Seleccione y haga clic con el botón derecho en el estado de animación "**player-idle**" y seleccione "**Set as Layer Default State**".



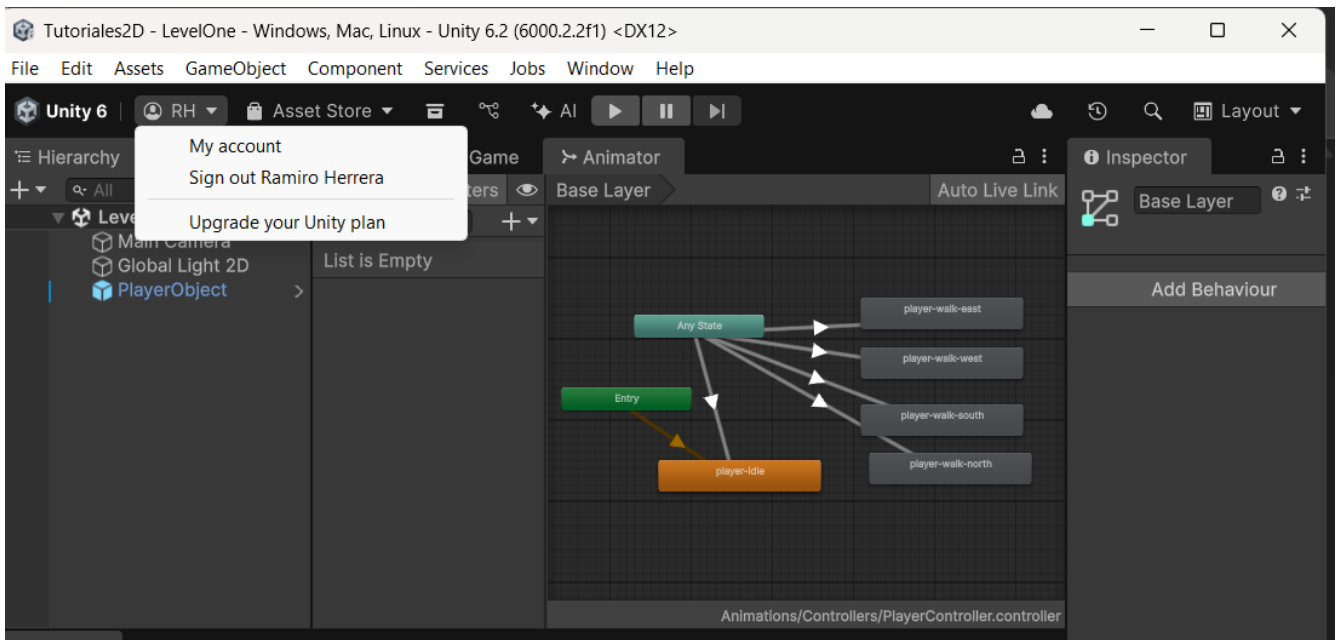
Paso 29: Ahora seleccione y haga clic con el botón derecho en "**Any State**" y seleccione "**Make Transition**". Aparecerá una línea con una flecha adjunta y siguiendo alrededor de su ratón. Haga click en "**player-walk-east**" para crear una transición.



Paso 30: Ahora haga lo mismo para el resto de los estados de animación: haga clic con el botón derecho en **Any State** → **Make Transition** y seleccione cada uno de los Estados de animación para crear una transición.

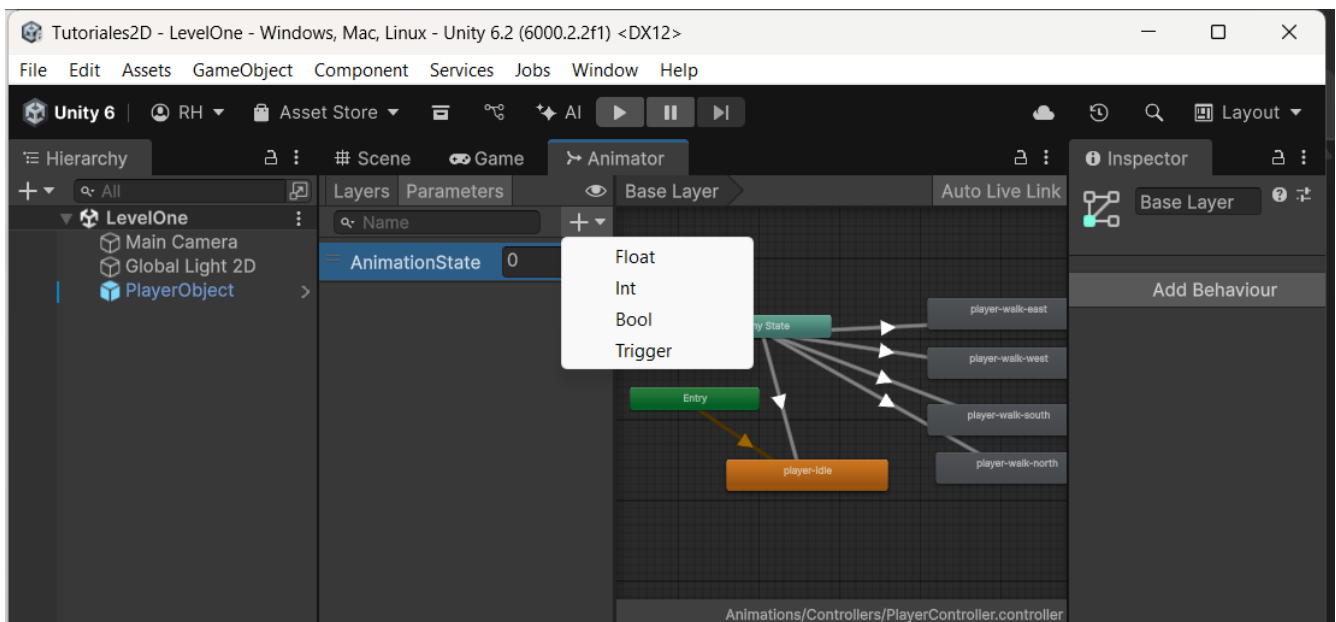


Paso 31: Debe crear un total de **cinco flechas de transición blancas** que apunten desde Any State a los cuatro estados de animación de caminata del jugador y al jugador inactivo player-idle.



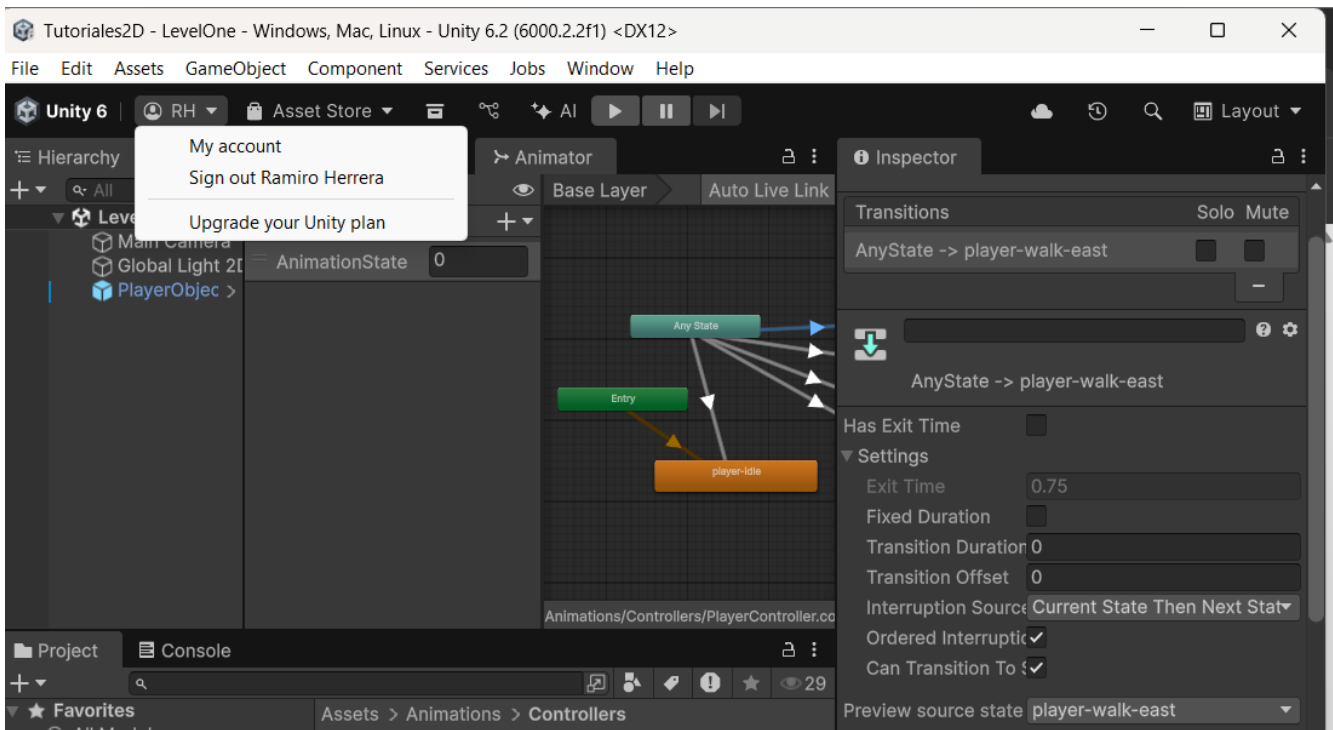
PARÁMETROS DE ANIMACIÓN

Paso 32: Seleccione la pestaña **Parámetros** en el lado izquierdo de la Ventana Animator. Presione el símbolo **+** y seleccione **"Int"** en la lista desplegable. Cambie el nombre del parámetro de animación creado a **"AnimationState"**.



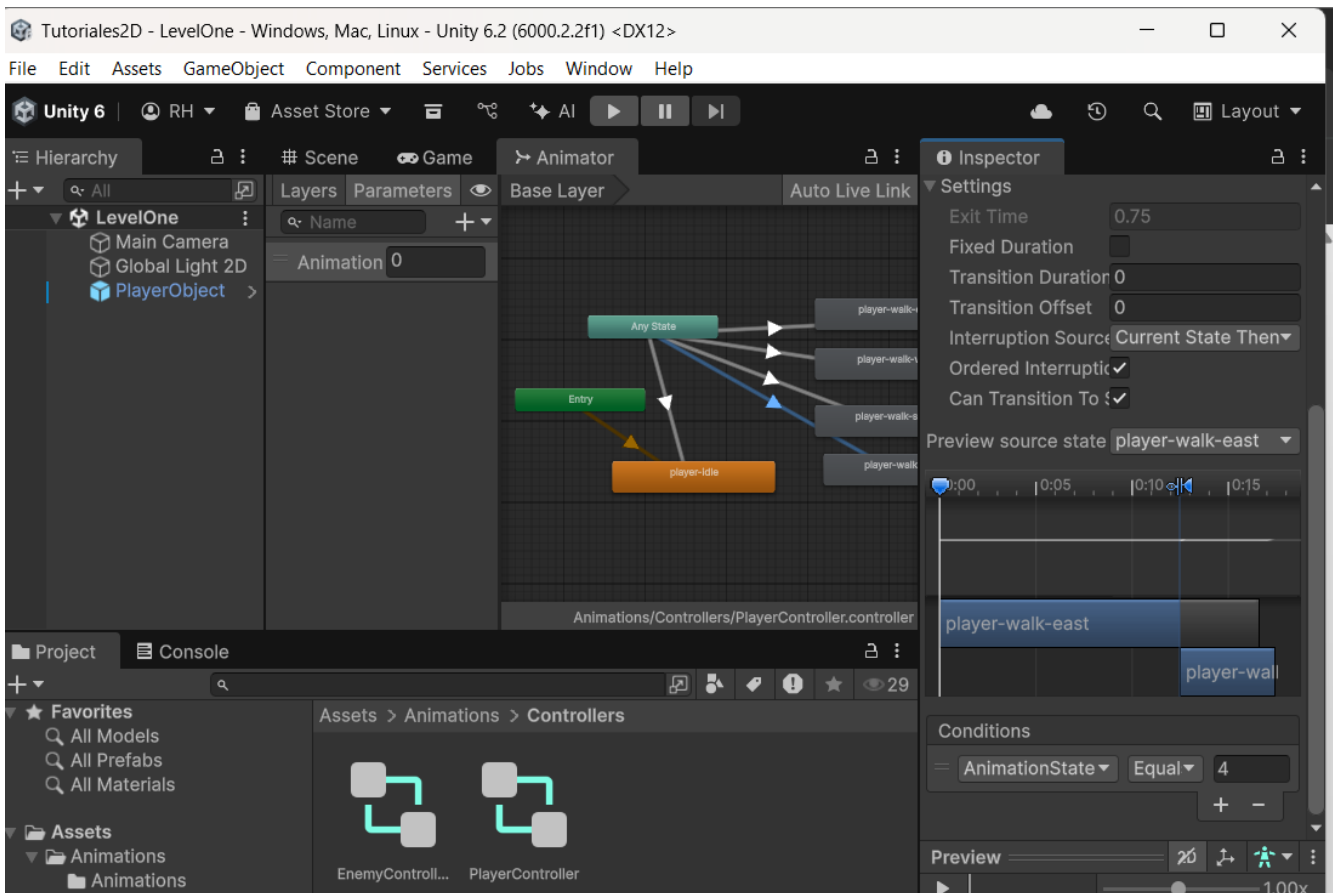
Paso 33: Seleccione la línea de transición blanca que conecta **Any State** con el estado en el Inspector, cambie la configuración para que coincida.

Paso 34: Desmarque **Has Exit Time** porque queremos interrumpir una animación si nuestro usuario presiona una tecla diferente.



Paso 35: En la parte inferior del inspector, verá un área titulada **"Conditions"**. Haga clic en el símbolo **+** en la parte inferior derecha y seleccione **AnimationState** → **Equals**, e ingrese el valor correspondiente para cada transición:

Transición	Condición
Any State to player-walk-east	AnimationState Equals = 1
Any State to player-walk-west	AnimationState Equals = 3
Any State to player-walk-north	AnimationState Equals = 4
Any State to player-walk-south	AnimationState Equals = 2
Any State to player-idle	AnimationState Equals = 0



ACTUALIZACIÓN DEL SCRIPT MOVEMENTCONTROLLER

Paso 36: Agreguemos la referencia del objeto Animator y la definición de estados al script:

```
public class MovementController : MonoBehaviour
{
    //... propiedades anteriores ...

    Animator animator; //Referencia a componente animator
    string animationState = "AnimationState"; //Variable en Animator

    //Enumeración de los estados
    enum CharStates
    {
        walkEast = 1,
        walkSouth = 2,
        walkWest = 3,
```

```

        walkNorth = 4,
        idleSouth = 0
    }
}

```

Paso 37: Inicializamos en el método start el valor del componente Animator:

```

void Start()
{
    //... código anterior ...
    //Establece valor de componente Animator el objeto ligado
    animator = GetComponent<Animator>();
}

```

Paso 38: Modifiquemos el método Update:

```

void Update()
{
    this.UpdateState(); //Invoca al método
}

private void UpdateState() {
    if (movement.x > 0) { //ESTE
        animator.SetInteger(animationState, (int)CharStates.walkEast);
    } else if (movement.x < 0) { //OESTE
        animator.SetInteger(animationState, (int)CharStates.walkWest);
    } else if (movement.y > 0) { //NORTE
        animator.SetInteger(animationState, (int)CharStates.walkNorth);
    } else if (movement.y < 0) { //SUR
        animator.SetInteger(animationState, (int)CharStates.walkSouth);
    } else { //IDLE

```

```

        animator.SetInteger(animationState, (int)CharStates.idleSouth);
    }
}

```

Paso 39: Modifiquemos el método FixedUpdate:

```

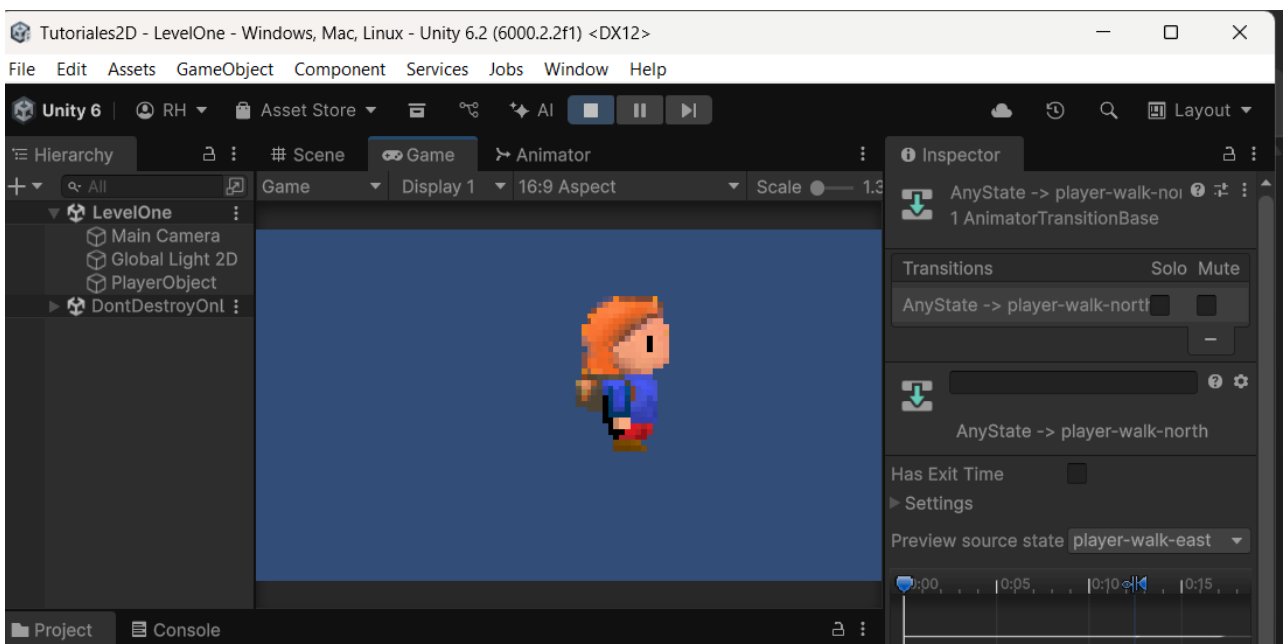
private void FixedUpdate() {
    MoveCharacter(); //Método definido para ingresar la dirección
}

private void MoveCharacter() {
    //Captura los datos de entrada del usuario
    movement.x = Input.GetAxisRaw("Horizontal");
    movement.y = Input.GetAxisRaw("Vertical");

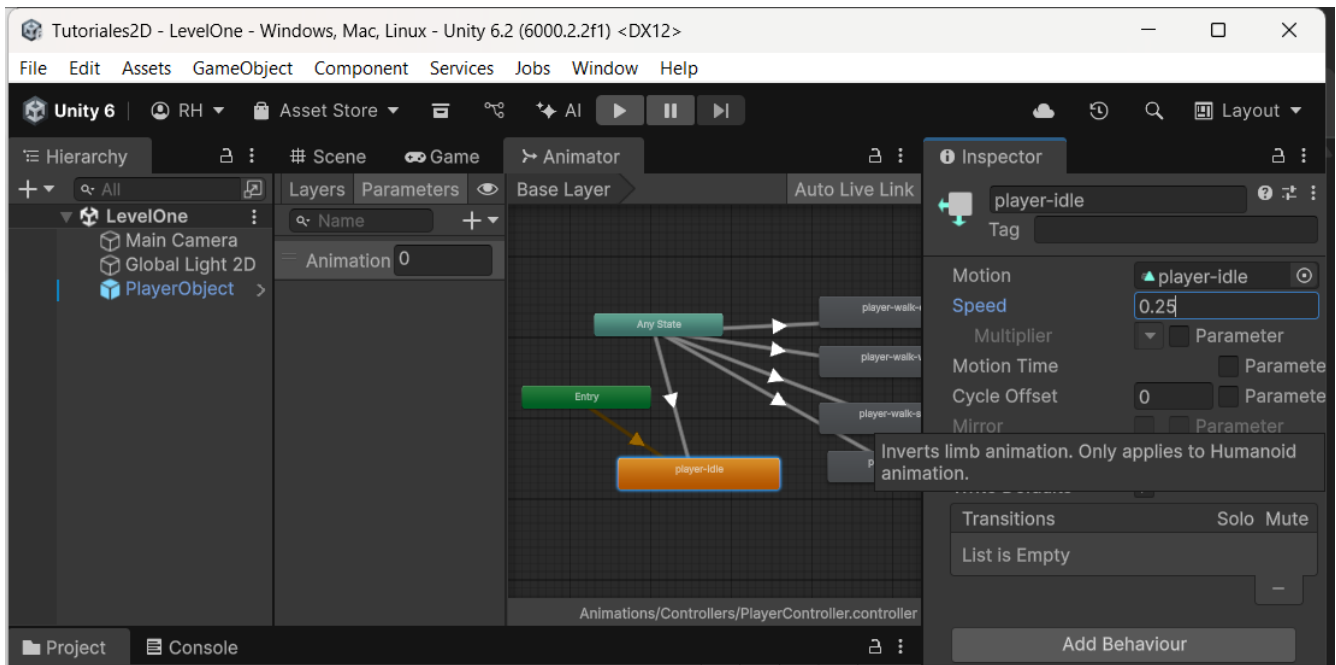
    //Conserva el rango de velocidad
    movement.Normalize();
    rb2D.velocity = movement * movementSpeed;
}

```

Paso 40: Regresa al Animator de Unity y verifica cada estado de transición. A medida que avanza por cada flecha de transición, recuerde desmarcar cuadros como **Exit Time**, **Fixed Duration**, **Can Transition to Self** y establecer **Transition Duration (%)** a 0.

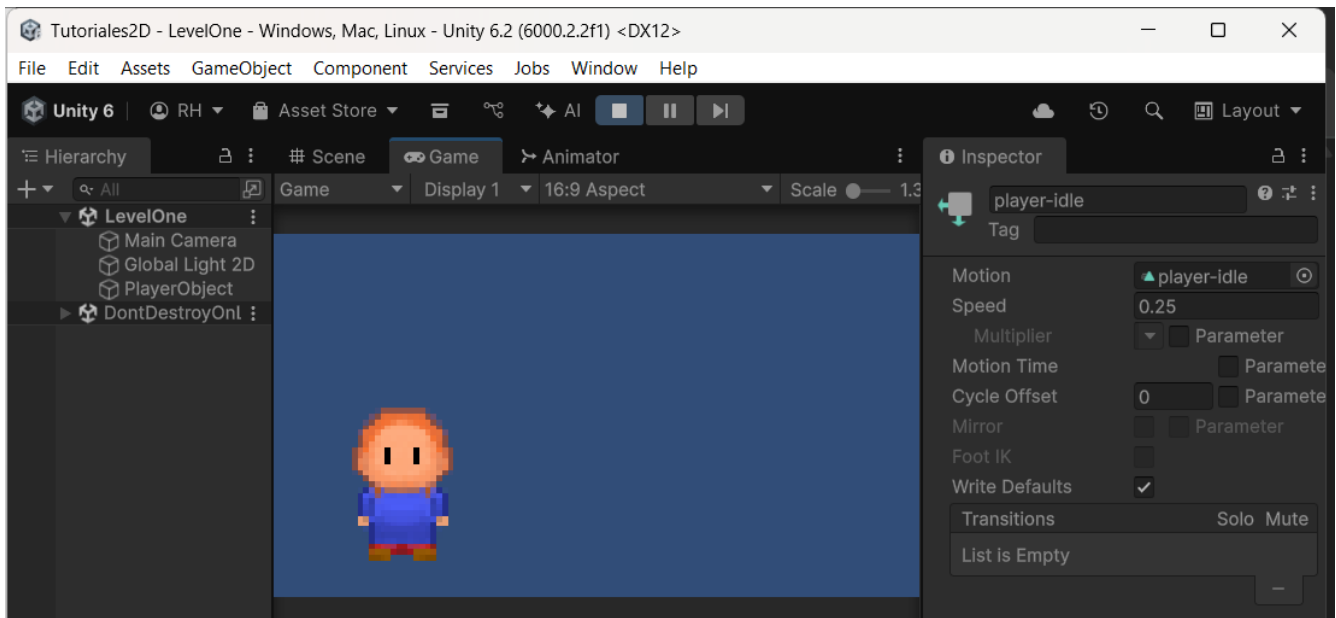


Paso 41: Seleccione cada estado de animación de caminata del jugador objeto y ajuste la **velocidad a 0,6**, y ajuste **player-idle a 0.25**.



RESULTADO FINAL

Paso 42: Ahora ha configurado una gran parte de las animaciones del reproductor necesarias para nuestro juego. Pulsamos el botón **Play** y movemos nuestro personaje por la pantalla con las teclas de flecha o **W, A, S, D**.



Unidad I. Programación orientada a objetos para videojuegos

