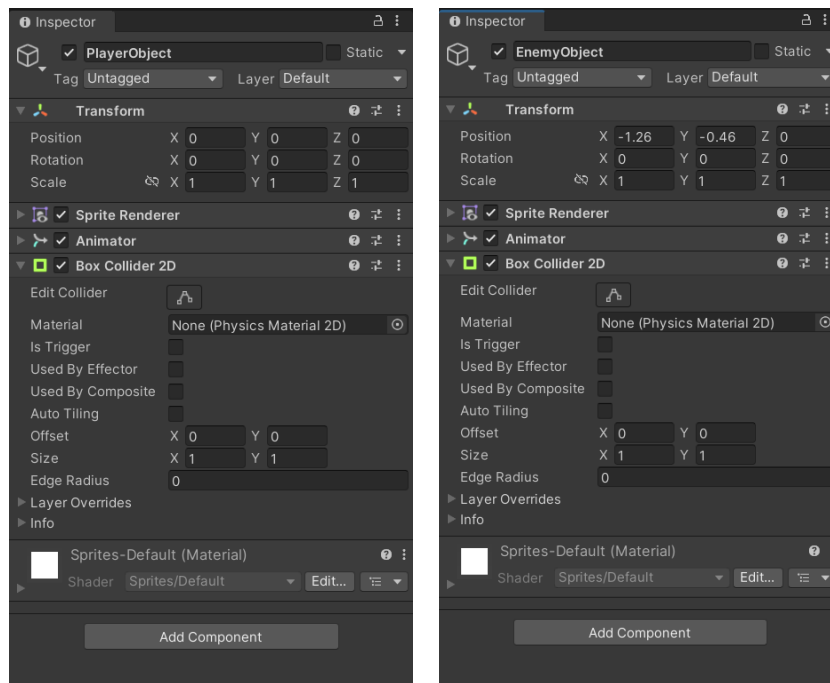
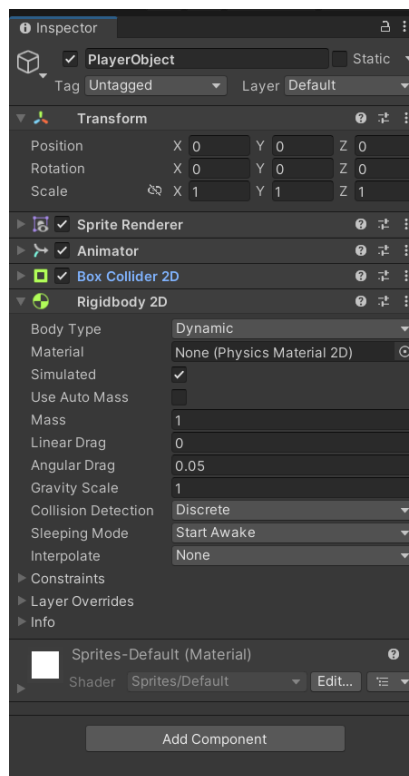


agregar un Box Collider 2D al **PlayerObject** y **EnemyObject**

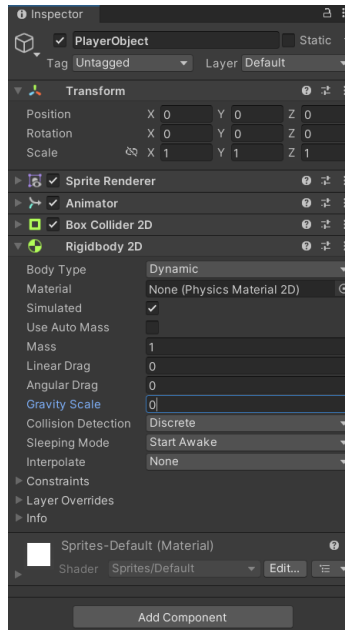


agregar un Box Rigidbody 2D al **PlayerObject**

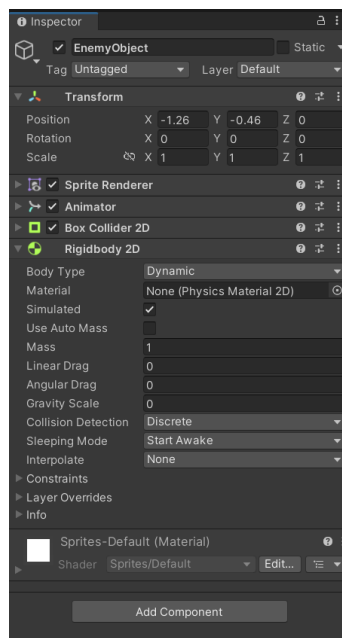


En el menú desplegable establezca los valores a las siguientes propiedades:

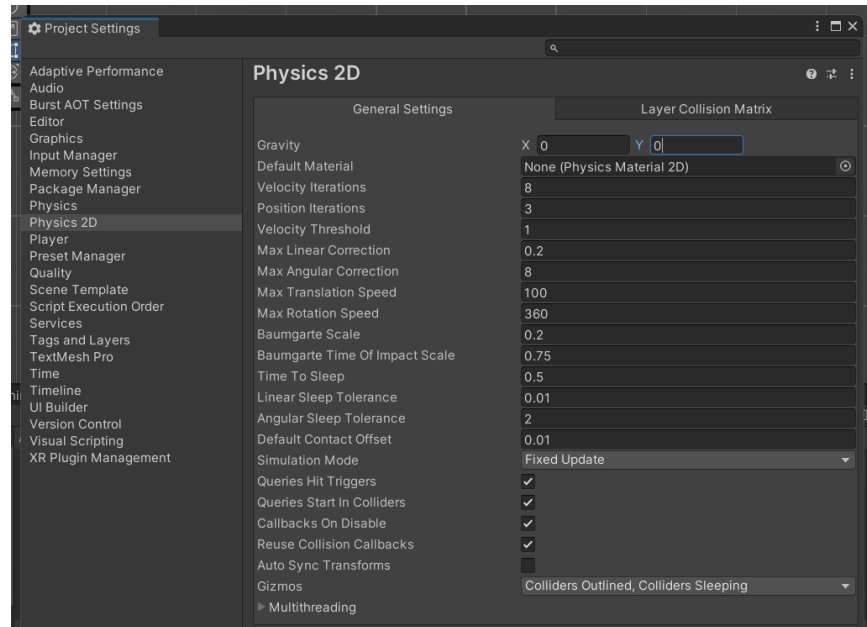
- Body Type Dynamic
- Mass 1
- Linear Drag 0
- Angular Drag 0
- Gravity Scale 0



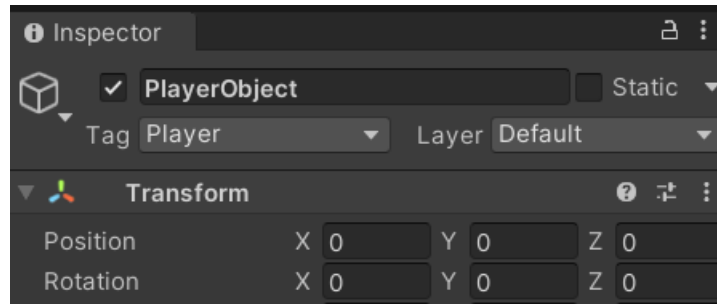
Seleccione el **EnemyObject** y agregue un Componente Rigidbody 2D de tipo Dinámico para él también.



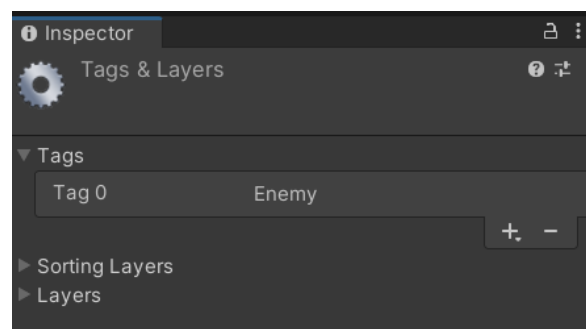
Vaya a la opción Edit ► Project settings ► Physics 2D y cambie el valor para la gravedad Y de $-9,81$ a 0.

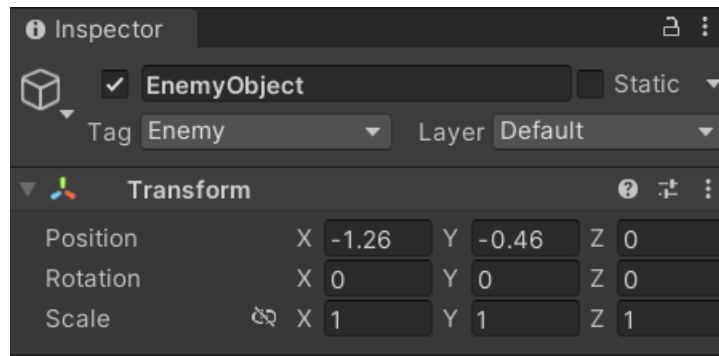


seleccione la etiqueta de Jugador para agregar una etiqueta a nuestro **PlayerObject**.

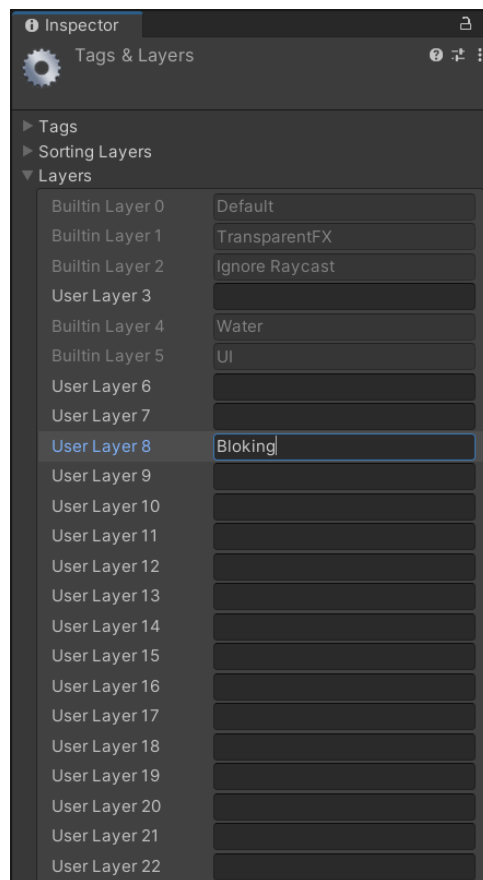


Crear una nueva etiqueta llamada "Enemy" y úsala para configurar la etiqueta del objeto **EnemyObject**.

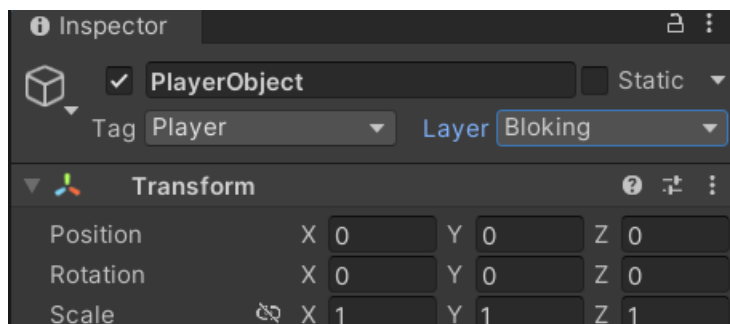




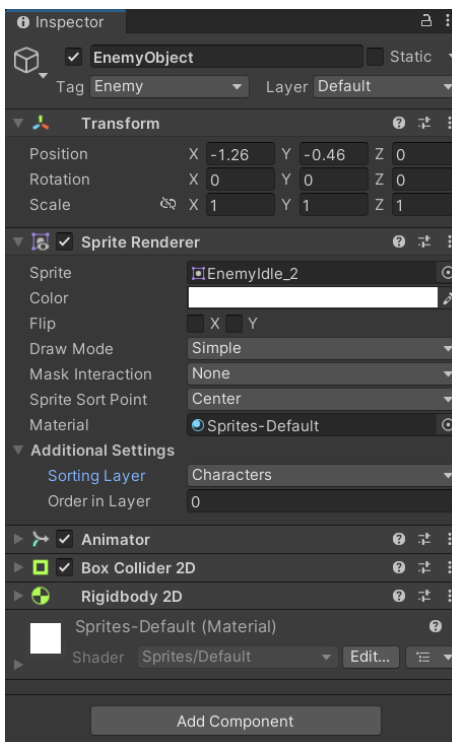
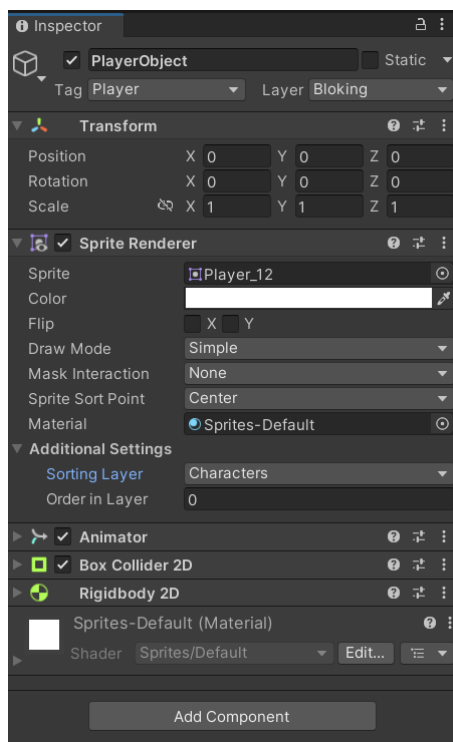
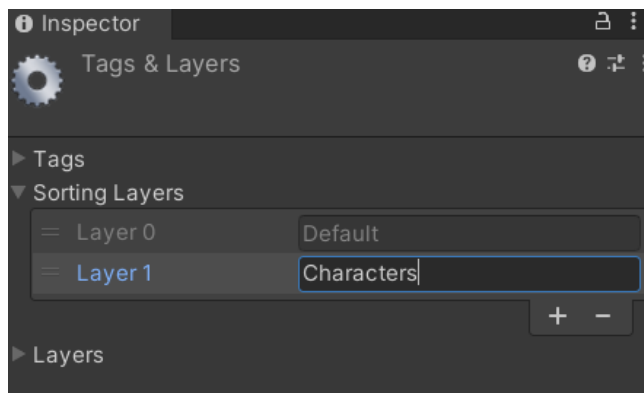
Crea una nueva Layer llamada “Blocking”



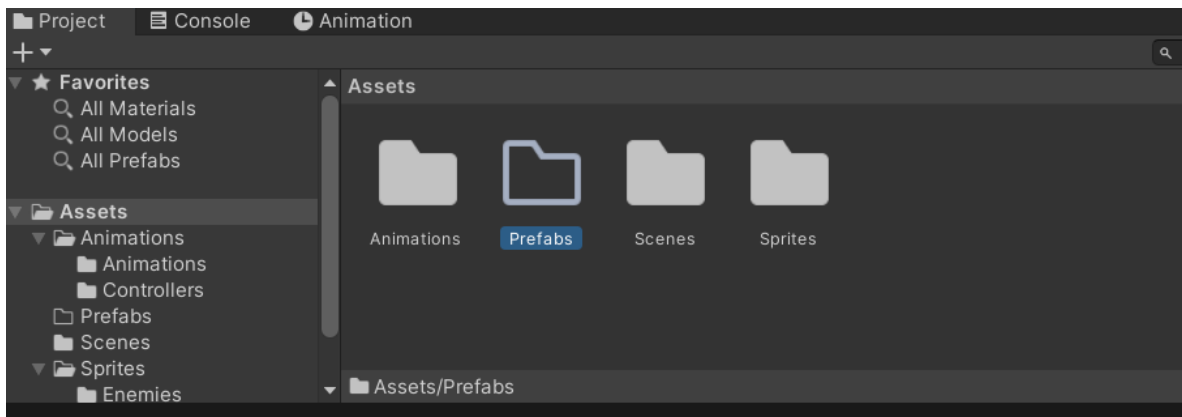
Agregar la capa que se creó al player



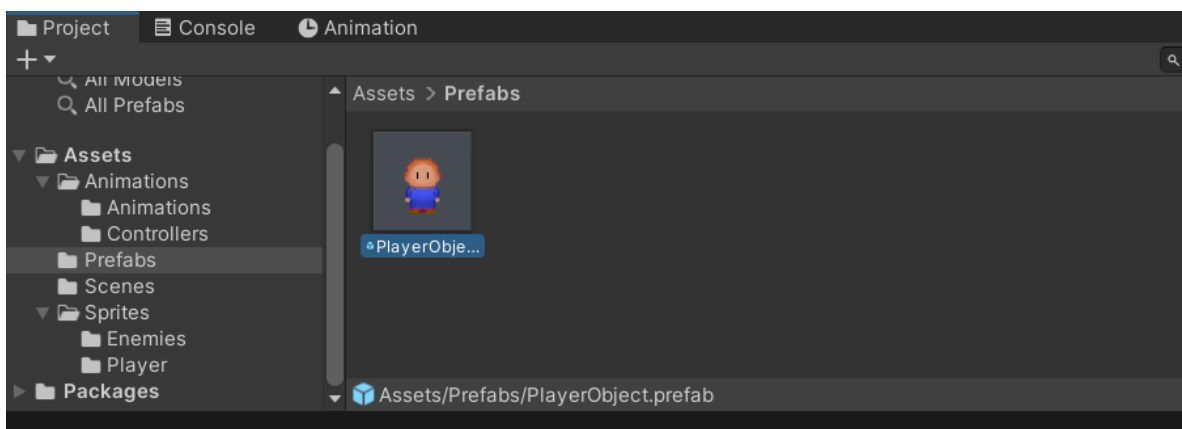
Vamos a agregar una capa de ordenamiento llamada "Caracteres" que usaremos para nuestro jugador y todos los enemigos.



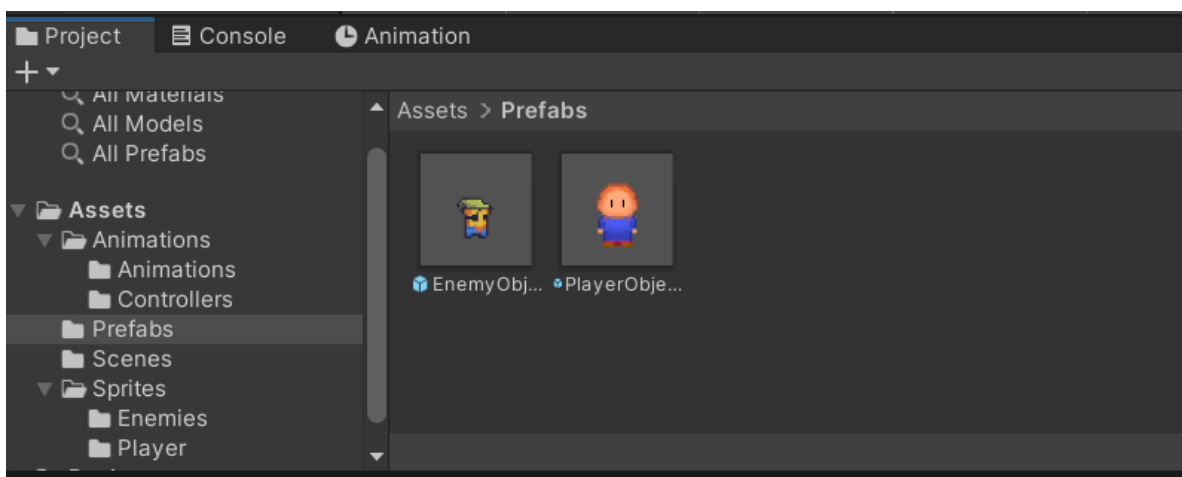
crea una carpeta llamada “**Prefabs**” en nuestra carpeta Assets en la vista Proyecto.



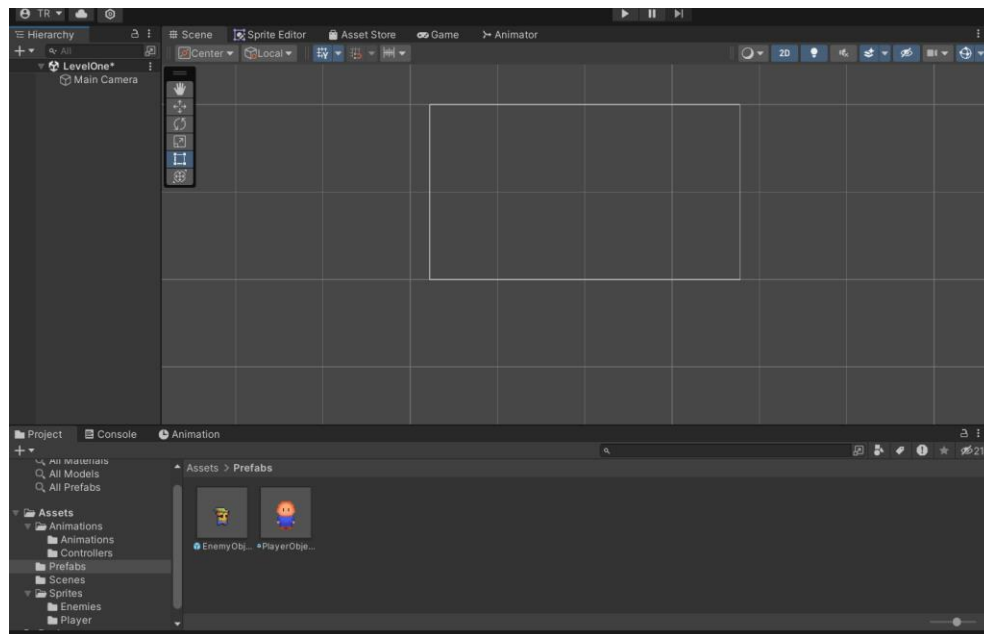
seleccione nuestro **PlayerObject** de la vista de Hierarchy y simplemente arrástralo a la carpeta **Prefabs**



seleccione nuestro **EnemyObject** de la vista de Hierarchy y simplemente arrástralo a la carpeta **Prefabs**



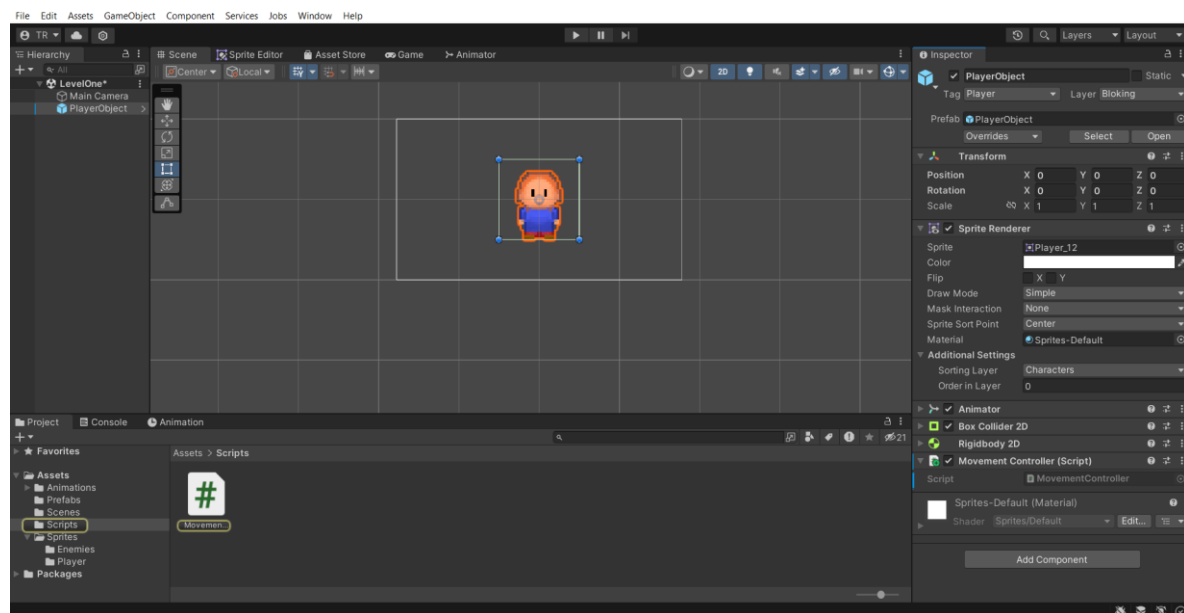
eliminar de forma segura el **PlayerObject** y **EnemyObject** de la vista de Hierarchy.



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

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**".

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



Agreguemos las siguientes propiedades a la clase y Establezcamos la referencia **Rigidbody2D**

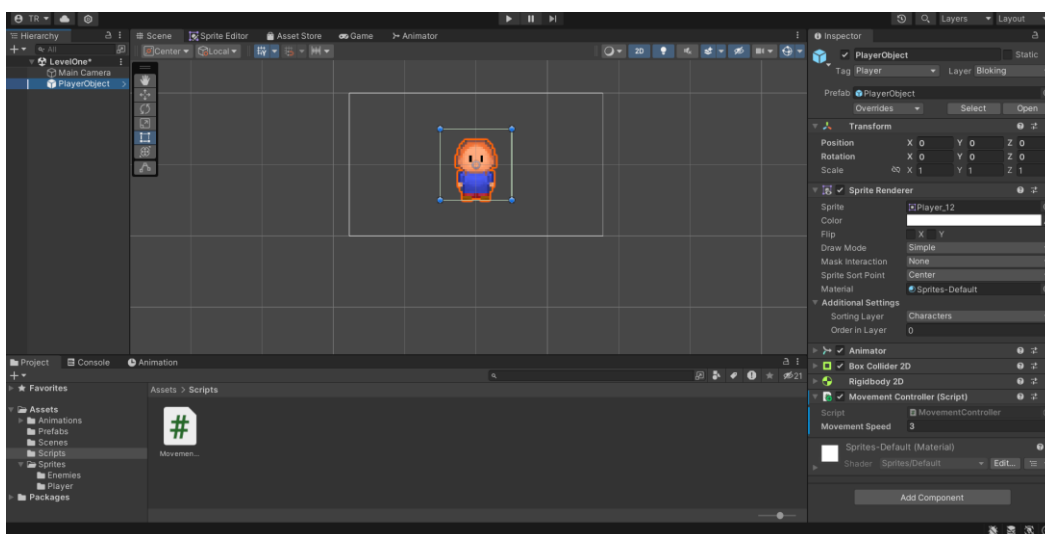
```
MovementController.cs
Assembly-CSharp
MovementController
Start()

1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class MovementController : MonoBehaviour
6 {
7     //Velocidad de los personajes
8     public float movementSpeed = 3.0f;
9     //Representa la ubicación del player o Enemy
10    Vector2 movement = new Vector2();
11    //Referencia a Rigidbody2D
12    Rigidbody2D rb2D;
13
14    // Start is called before the first frame update
15    // Mensaje de Unity | 0 referencias
16    void Start()
17    {
18        //Establece el componente Rigidbody2D enlazado
19        rb2D = GetComponent<Rigidbody2D>();
20    }
21
22    // Update is called once per frame
23    // Mensaje de Unity | 0 referencias
24    void Update()
25    {
26    }
27
28    }
29
30 100 % No se encontraron problemas. Línea: 18 Carácter: 44 SPC CRLF
31 Salida
```

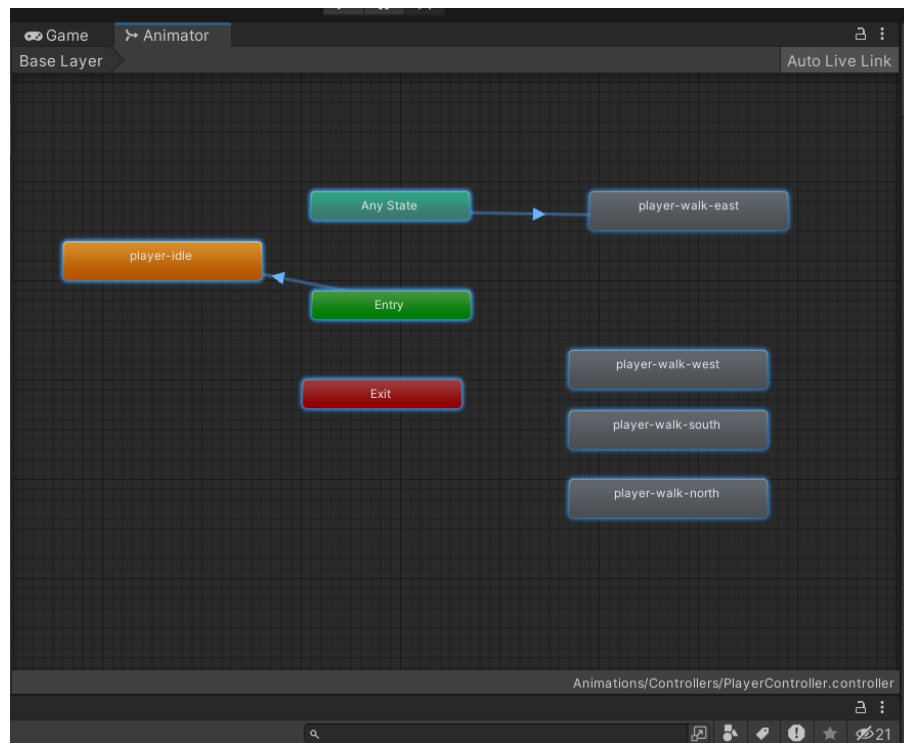
Programemos el método FixedUpdate

```
MovementController.cs
Assembly-CSharp
MovementController
FixedUpdate()

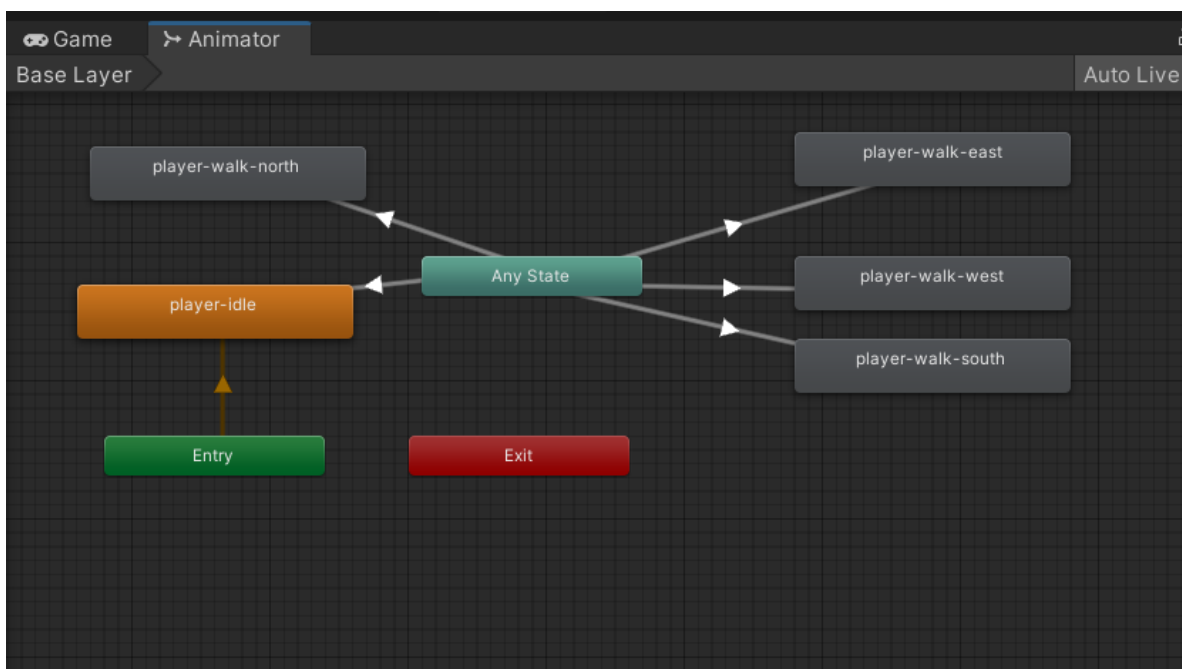
17 //Establece el componente Rigidbody2D enlazado
18 rb2D = GetComponent<Rigidbody2D>();
19
20
21
22 // Update is called once per frame
23 // Mensaje de Unity | 0 referencias
24 void Update()
25 {
26 }
27
28 // Mensaje de Unity | 0 referencias
29 private void FixedUpdate()
30 {
31     //Captura los datos de entrada del usuario
32     movement.x = Input.GetAxisRaw("Horizontal");
33     movement.y = Input.GetAxisRaw("Vertical");
34
35     //Conservar el rango de velocidad
36     movement.Normalize();
37     rb2D.velocity = movement * movementSpeed;
38 }
39
40 }
```



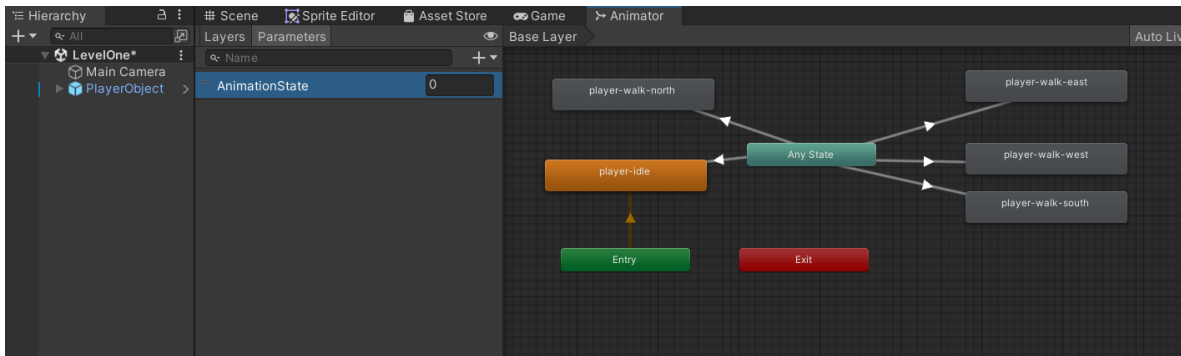
Seleccione y, a continuación, haga clic con el botón derecho en el estado de animación "player-idle" y seleccione "Set as Layer Default State". Después crea una transición entre el objeto Any State y player-walk-east.



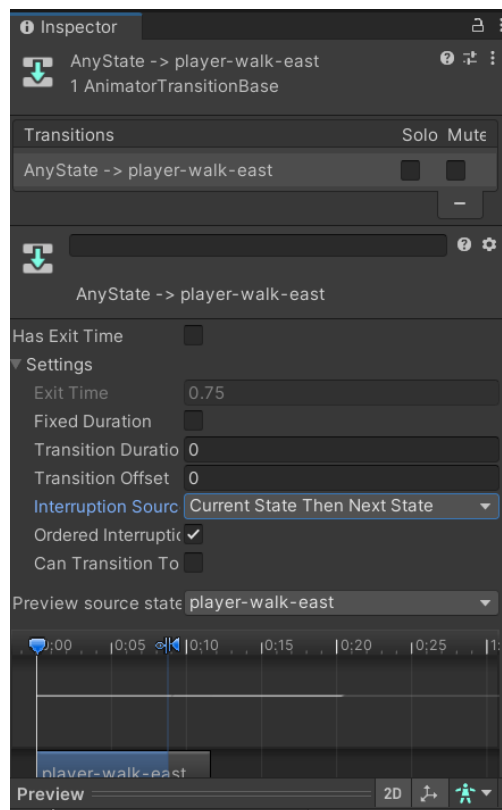
crea una transición entre el objeto Any State y todas las demás animaciones



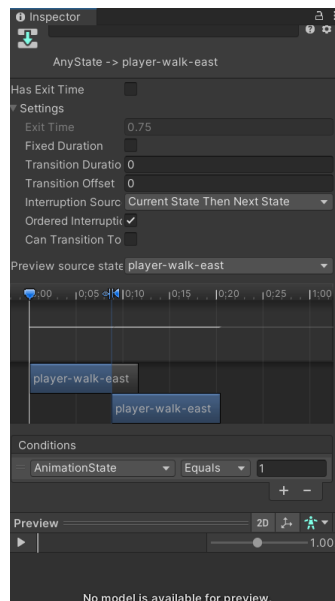
Seleccione la pestaña Parámetros en el lado izquierdo de la Ventana Animator. Presione el símbolo + y seleccione "Int" en la lista desplegable. Cambie el nombre del parámetro de animación creado a "AnimationState".



Seleccione la línea de transición blanca que conecta cualquier estado con el estado en la Inspector, cambie la configuración para que coincida.



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 1



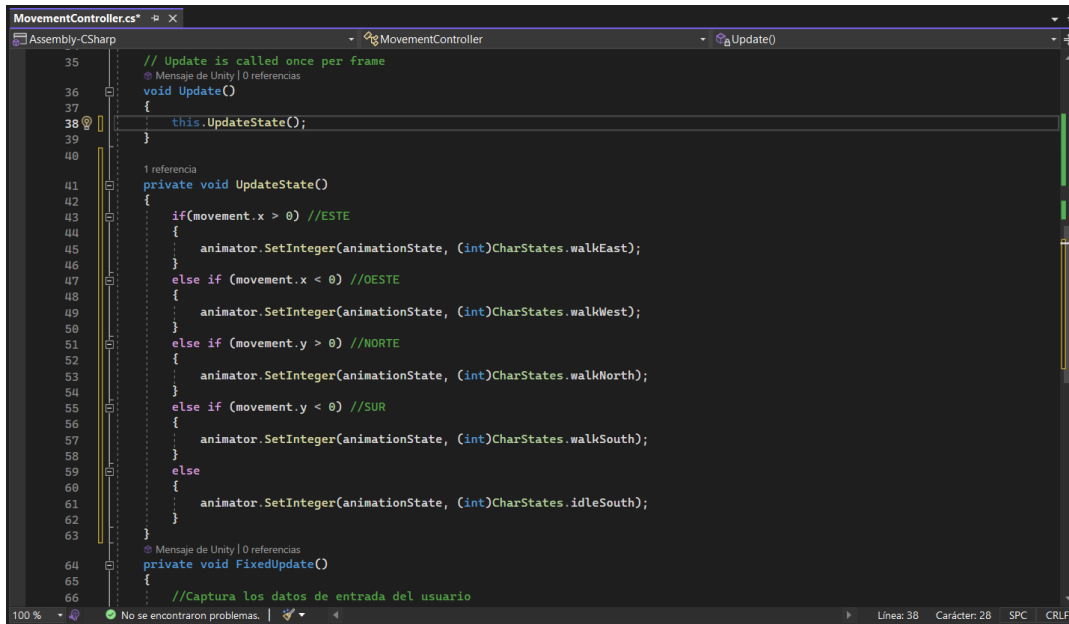
Lo siguiente que vamos a hacer es establecer que el parámetro AnimationState igual a 1 en nuestro script.

- Agreguemos la referencia del objeto Animator; referencia de la variable qué guarda los estados y la definición de los estados
- Inicializamos en el método start el valor del componente Animator del objeto enlazado

```
MovementController.cs
Assembly-CSharp
MovementController
Start()

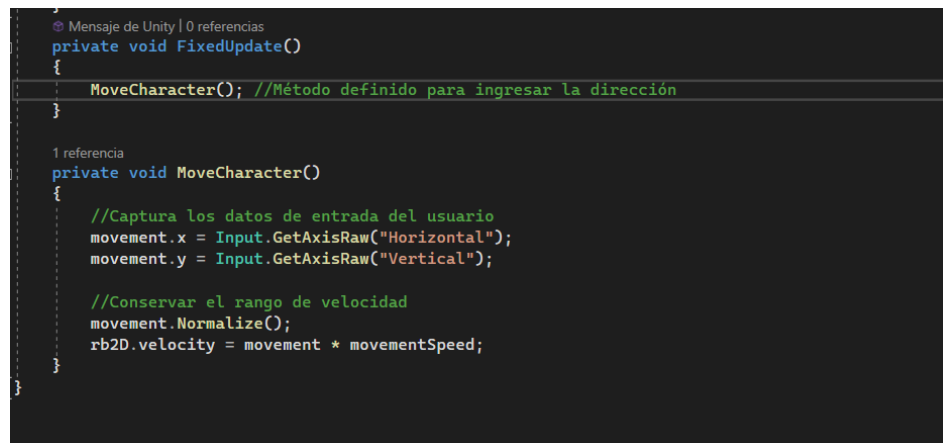
4
5 public class MovementController : MonoBehaviour
6 {
7     //Velocidad de los personajes
8     public float movementSpeed = 3.0f;
9     //Representa la ubicación del player o Enemy
10    Vector2 movement = new Vector2();
11    //Referencia a Rigidbody2D
12    Rigidbody2D rb2D;
13
14    Animator animator; //Referencia a componente animator
15    string animationState = "AnimationState"; //Variable en animator
16
17    //Enumeración de los estados
18    enum CharStates {
19        walkEast = 1,
20        walkSouth = 2,
21        walkWest = 3,
22        walkNorth = 4,
23        idleSouth = 5
24    }
25
26    // Start is called before the first frame update
27    void Start()
28    {
29        //Establece el componente Rigidbody2D enlazado
30        rb2D = GetComponent<Rigidbody2D>();
31        //Establecer valor de componente Animator el objeto ligado
32        animator = GetComponent<Animator>();
33    }
34
35    // Update is called once per frame
```

Modifiquemos el método Update donde se invoca a otro método que define en base a las entradas WASD por el usuario



```
35 // Update is called once per frame
36 // Mensaje de Unity | 0 referencias
37 void Update()
38 {
39     this.UpdateState();
40 }
41
42 1 referencia
43 private void UpdateState()
44 {
45     if (movement.x > 0) //ESTE
46     {
47         animator.SetInteger(animationState, (int)CharStates.walkEast);
48     }
49     else if (movement.x < 0) //OESTE
50     {
51         animator.SetInteger(animationState, (int)CharStates.walkWest);
52     }
53     else if (movement.y > 0) //NORTE
54     {
55         animator.SetInteger(animationState, (int)CharStates.walkNorth);
56     }
57     else if (movement.y < 0) //SUR
58     {
59         animator.SetInteger(animationState, (int)CharStates.walkSouth);
60     }
61     else
62     {
63         animator.SetInteger(animationState, (int)CharStates.idleSouth);
64     }
65 }
66 // Mensaje de Unity | 0 referencias
67 private void FixedUpdate()
68 {
69     //Captura los datos de entrada del usuario
```

Modifiquemos el método FixedUpdate



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

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

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

Seleccione cada estado de animación de caminata del jugador objeto y ajuste la velocidad a 0,6, y ajuste player-idle a 0.25. Esto hará que las animaciones de nuestros jugadores se vean bien.

