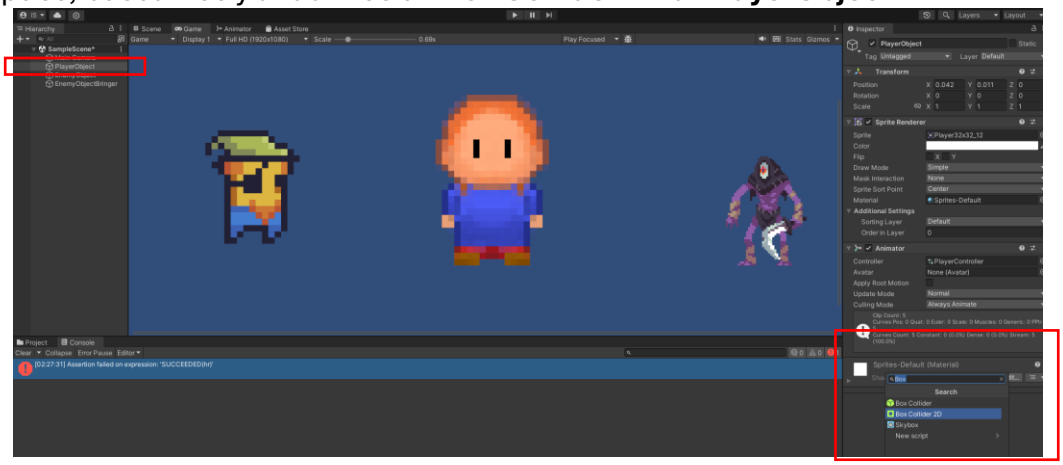
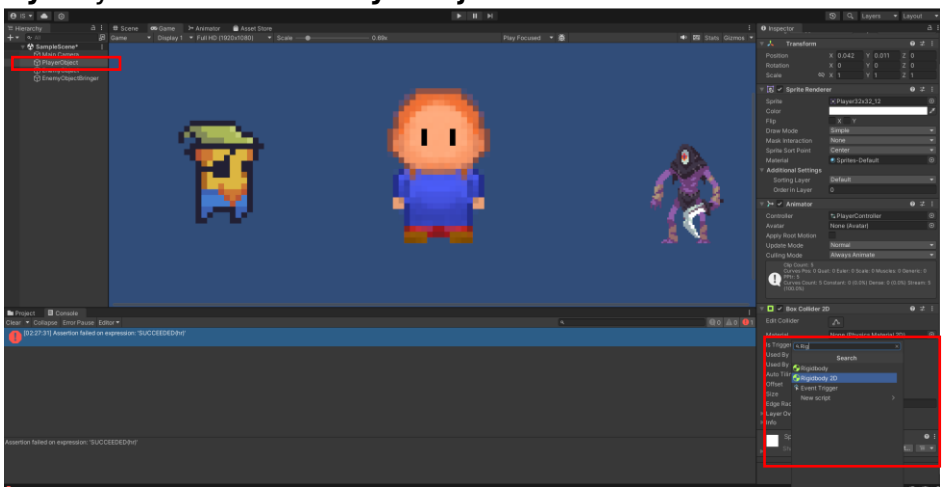
Seleccionamos el **PlayerObject** y luego presionamos el botón **Add Component** en la ventana del inspector. Después, buscamos y añadimos un **Box Collider 2D** al **PlayerObject**.



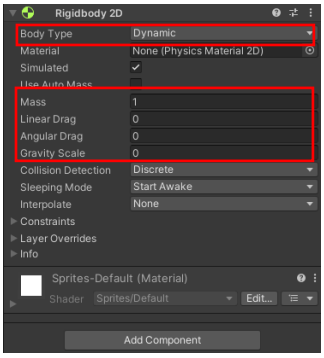
Hacemos lo mismo para EnemyObject.

Con el **PlayerObject** seleccionado, hicimos clic en el botón **Add Component** en la ventana del inspector, buscamos **Rigidbody 2D** y lo añadimos al **PlayerObject**.



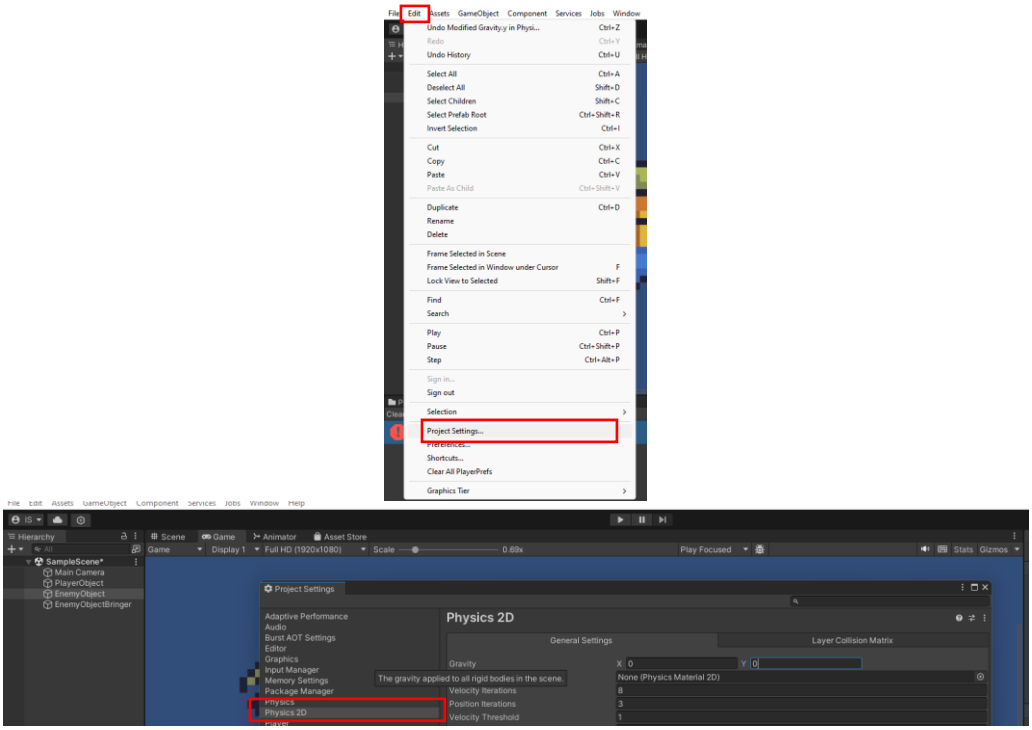
En el menú desplegable, establecimos los valores de las siguientes propiedades.

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

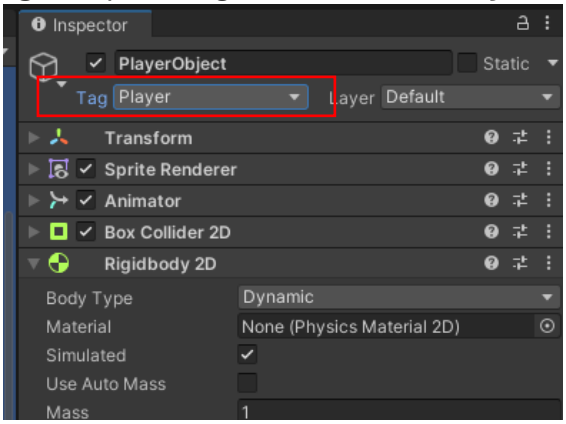


Seleccionamos el **EnemyObject** y añadimos un componente **Rigidbody 2D** configurado como tipo Dinámico igual que las configuraciones anteriores.

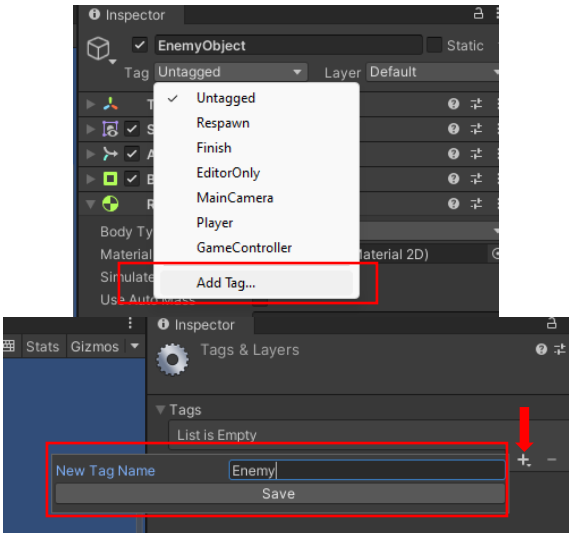
Fuimos a la opción **Edit| Project Settings | Physics 2D** y cambiamos el valor de la gravedad en **Y** de **-9.81** a **0**.



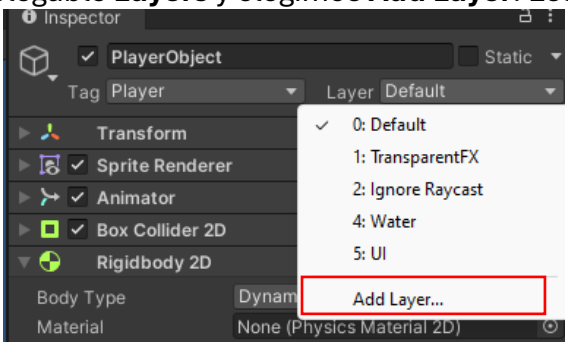
Seleccionamos el **PlayerObject**. En el menú desplegable **Tag**, ubicado en la parte superior izquierda del inspector, elegimos la etiqueta **Jugador** para asignarla a nuestro **PlayerObject**.

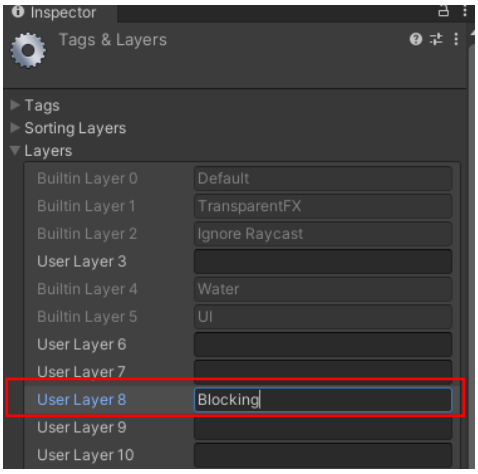


Creamos una nueva etiqueta llamada **Enemy** y la utilizamos para asignarla como etiqueta al **EnemyObject**.

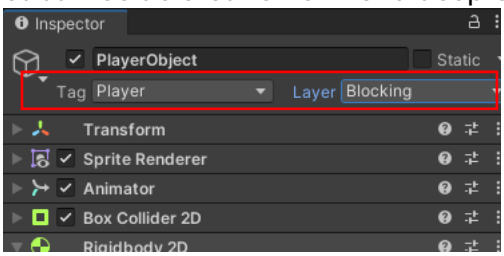


Seleccionamos en el menú desplegable **Layers** y elegimos **Add Layer**. Esto abrió la ventana de **Layers**.

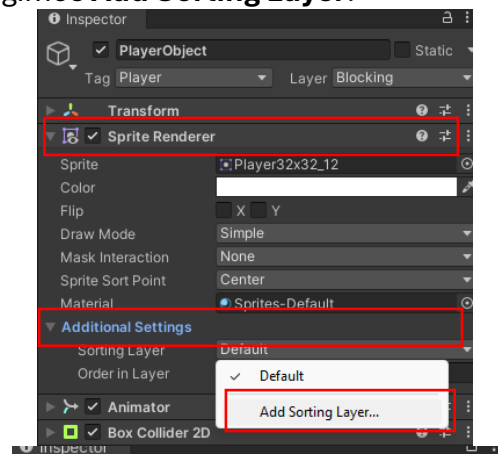


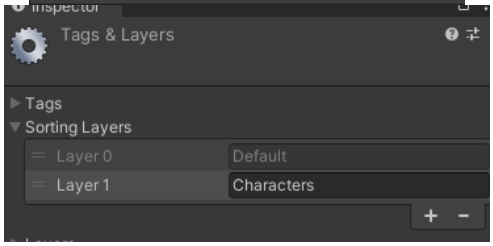


Ahora seleccionamos nuevamente el **PlayerObject** para ver sus propiedades en el Inspector. Luego, elegimos la capa de bloqueo que acabamos de crear en el menú desplegable.

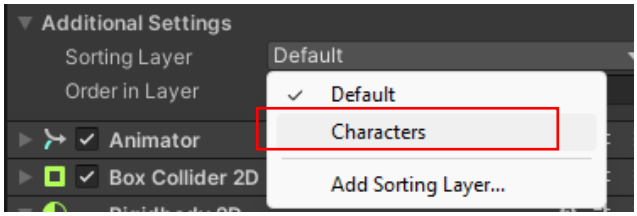


En el componente **Sprite Renderer** de la ventana del Inspector, seleccionamos la opción **Sorting Layer** en el menú desplegable de capa y elegimos **Add Sorting Layer**.



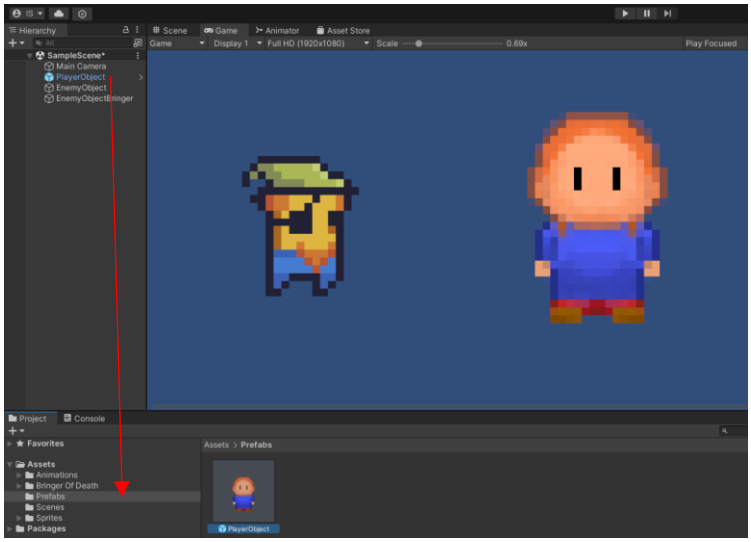


Agregamos una capa de ordenamiento llamada **"Characters"**. Luego, hicimos clic nuevamente en el **PlayerObject** para ver su Inspector y seleccionamos la nueva capa de ordenamiento **"Characters"** en el menú desplegable.



Seleccionamos nuestro **EnemyObject** y establecimos su **Capa de ordenamiento** en **Characters**, ya que queremos que los enemigos también aparezcan por encima de los demás objetos.

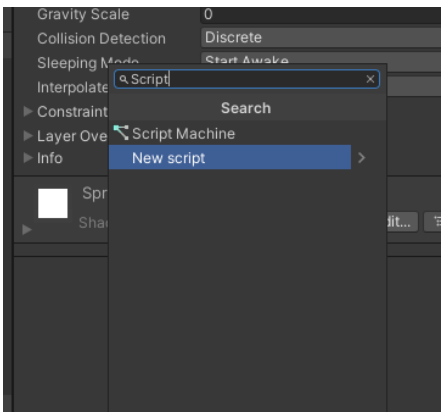
Creamos una carpeta llamada **"Prefabs"** en nuestra carpeta **Assets** en la vista **Proyecto**. Luego, seleccionamos nuestro **PlayerObject** en la vista **Hierarchy** y simplemente lo arrastramos a la carpeta **Prefabs**.



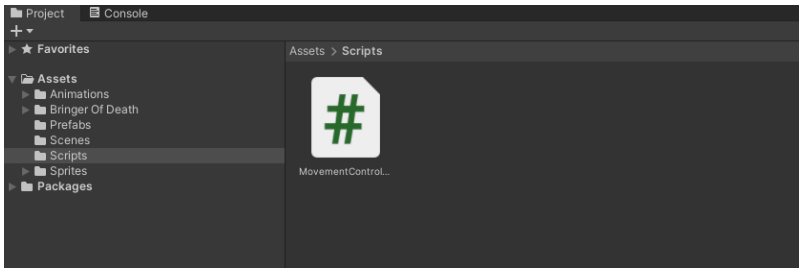
Ahora, podemos eliminar de forma segura el **PlayerObject** de la vista **Hierarchy**.
Hicimos lo mismo con el **EnemyObject**: lo arrastramos a la carpeta **Prefabs** y eliminamos el **EnemyObject** original de la vista **Hierarchy**.

Entonces tenemos nuestro PlayerObject y tenemos nuestro EnemyObject. ¡Hagamos qué se muevan!

1. Seleccionamos nuestro **PlayerObject Prefab** y lo arrastramos a la vista **Hierarchy**.
2. Luego, nos desplazamos hasta la parte inferior del **Inspector** y presionamos el botón **Add Component**.
3. Escribimos la palabra **Script**, seleccionamos **New Script** y lo nombramos como **MovementController**.



4. Creamos 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**. Luego, arrastramos el script **MovementController** a la carpeta **Scripts** y hacemos doble clic en él para abrirlo en **Visual Studio**.



Es hora de programar nuestro primer Script. Los scripts en Unity están escritos en un lenguaje llamado C#. El espacio de nombres UnityEngine contiene muchas clases específicas de Unity como MonoBehaviour, GameObject, Rigidbody2D y BoxCollider2D

Agreguemoslas siguientes propiedades a la clase

```
MovementController.cs
Assembly-CSharp
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
   Script de Unity | 0 referencias
5  public class MovementController : MonoBehaviour
6  {
7      // Velocidad de los personajes
8      public float movementSpeed = 3.0f;
9
10     // Representa la ubicación del Player o Enemy
11     Vector2 movement = new Vector2();
12
13     // Referencia a Rigidbody2D
14     Rigidbody2D rb2D;
```

Establezcamos la referencia RigidBody2D

```
   Mensaje de Unity | 0 referencias
28 void Start()
29 {
30     // Establece el componente Rigidbody2D enlazado
31     rb2D = GetComponent<Rigidbody2D>();
32 }
33
```

Programemosel método FixedUpdate

```
74 private void MoveCharacter()
75 {
76     // Captura los datos de entrada del usuario
77     movement.x = Input.GetAxisRaw("Horizontal");
78     movement.y = Input.GetAxisRaw("Vertical");
79
80     // Conserva el rango de velocidad
81     movement.Normalize();
82
83     rb2D.velocity = movement * movementSpeed;
84 }
85 }
```

Regresamos al **Editor de Unity** y nos aseguramos de ver nuestro **PlayerObject** en la vista **Hierarchy**. Si no lo veíamos, arrastramos el **PlayerObject** desde la carpeta **Prefabs** en la vista **Hierarchy**.

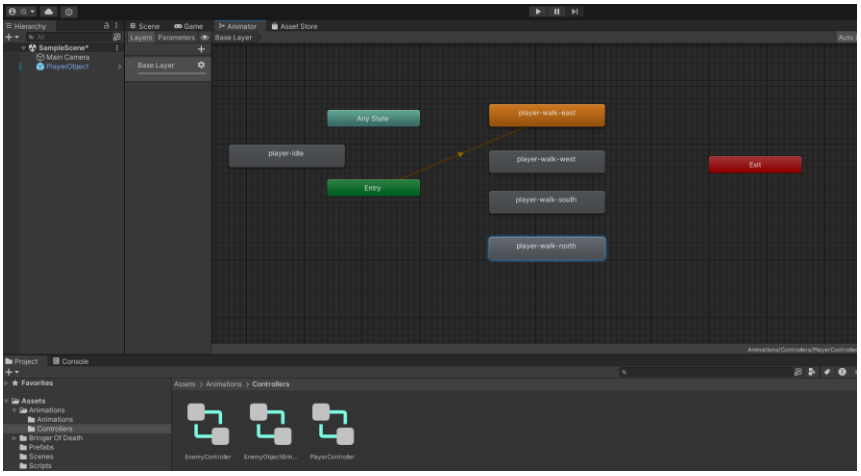
Para agregar el script a nuestro **PlayerObject**, arrastramos el script **MovementController** desde la carpeta **Scripts** directamente sobre el **PlayerObject** en la jerarquía, o lo arrastramos al **Inspector** mientras el **PlayerObject** está seleccionado.

Finalmente, presionamos el botón **Play**. Deberíamos ver a nuestro personaje de jugador caminando en su lugar. Presionamos las teclas de flecha o **W, A, S, D** en el teclado y observamos cómo se mueve.

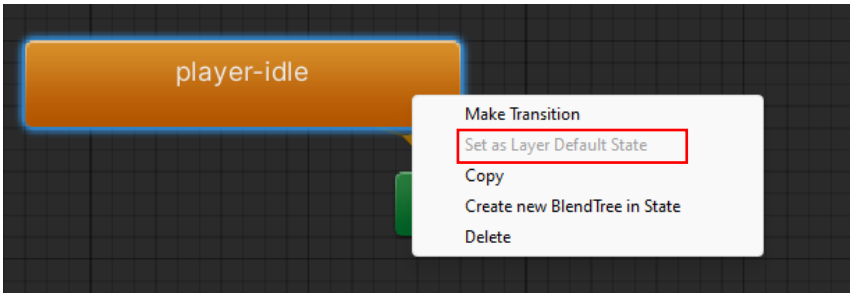
Ahora que sabemos cómo mover a nuestro personaje por la pantalla, vamos a hablar sobre cómo cambiar entre animaciones basadas en el estado actual del jugador.

Vamos a la carpeta **Animations | Controllers** y hacemos doble clic en el objeto **PlayerController**.

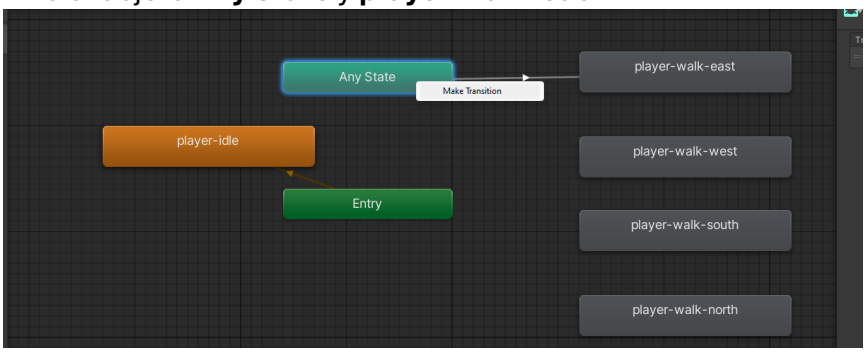
Hicimos clic y arrastramos los objetos de estado de animación hasta que aparecieron en la pantalla, con el reproductor inactivo a un lado. Las animaciones de caminata del jugador fueron agrupadas correctamente.



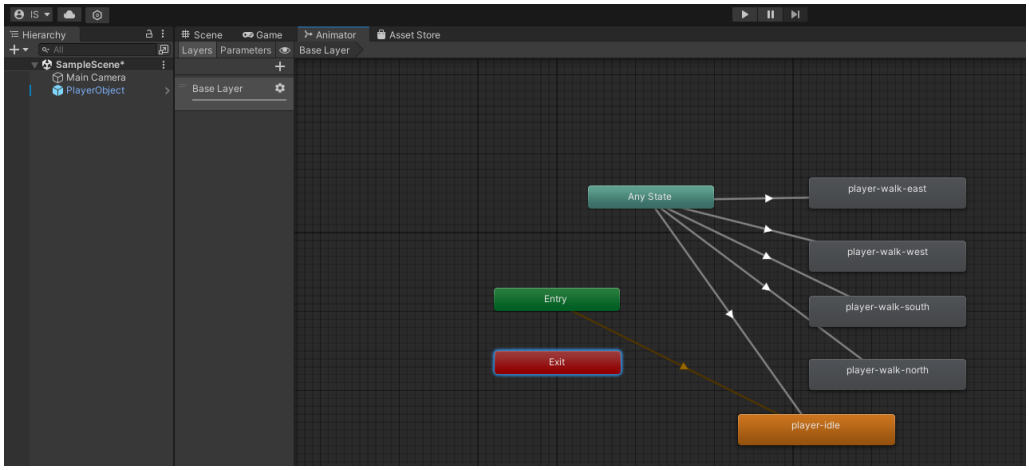
El color naranja indica que es el estado predeterminado para este **Animator**. Seleccionamos el estado de animación **"player-idle"**, luego hicimos clic con el botón derecho y seleccionamos **"Set as Layer Default State"**.



Seleccionamos **"Any State"** e hicimos clic con el botón derecho, luego seleccionamos **"Make Transition"**. Aparecerá una línea con una flecha adjunta, y siguiendo el ratón, hacemos clic en **"player-walk-east"** para crear una transición entre el objeto **Any State** y **player-walk-east**.

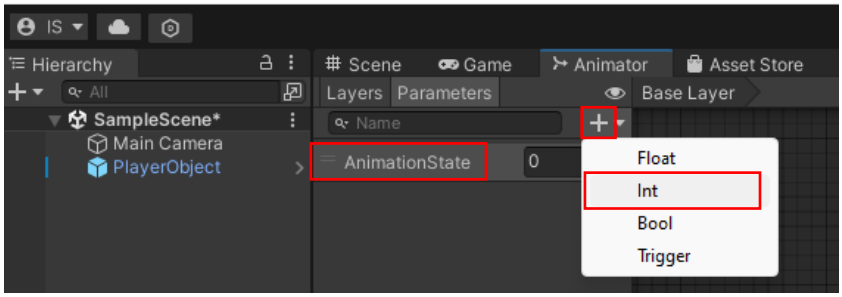


Hicimos lo mismo para el resto de los estados de animación: clic derecho en **Any State | Make Transition** y seleccionamos cada uno de los estados de animación restantes para crear una transición hacia ellos.



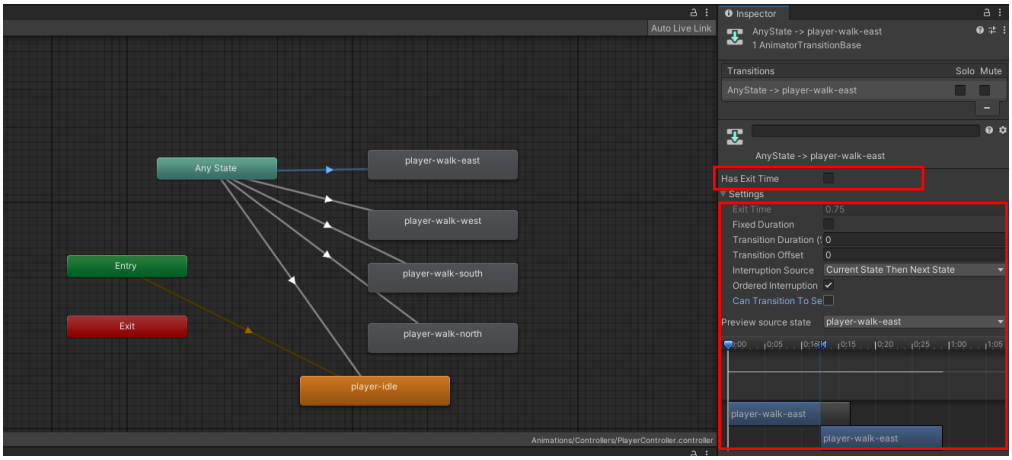
Creamos un total de cinco flechas de transición blancas que apunten desde **Any State** a los cuatro estados de animación de caminata del jugador y al estado de animación inactiva **player-idle**. Además, debería haber una flecha de color naranja, que indica el estado predeterminado, apuntando desde el estado de animación de entrada hacia la animación **player-idle**.

Seleccionamos la pestaña **Parámetros** en el lado izquierdo de la ventana **Animator**. Presionamos el símbolo **+** y seleccionamos **"Int"** en la lista desplegable. Luego, cambiamos el nombre del parámetro de animación creado a **"AnimationState"**.



Vamos a establecer el parámetro de animación en cada transición a una condición específica. Si durante el juego esta condición es cierta, entonces el animador pasará a ese estado de animación y al correspondiente se reproducirá el clip de animación. Debido a que este componente Animator se adjunta a PlayerObject, los clips de animación se mostrarán en el componente Transform la ubicación del componente en la escena. Usamos un script para configurar esta animación. La condición del parámetro debe ser verdadera y desencadenar la transición de estado

Seleccionamos la línea de transición blanca que conecta **Any State** con el estado en el **Inspector**. Luego, cambiamos la configuración de la transición para que coincida con los parámetros adecuados, como el **AnimationState** o cualquier otra condición que sea necesaria para que la transición funcione correctamente.



Desmarcamos la opción **Has Exit Time** porque queremos que la animación se interrumpa si el usuario presiona una tecla diferente. Si dejamos marcada la opción **Has Exit Time**, la animación tendría que terminarse completamente hasta alcanzar el porcentaje configurado en **Exit Time** antes de que pueda comenzar la siguiente animación. Esto podría resultar en una mala experiencia para el jugador, ya que las transiciones entre animaciones no serían instantáneas.

Lo siguiente que vamos a hacer es establecer que el parámetro AnimationState igual a 1 en nuestro script. Regresamos a Visual Studio y abrimos nuestro Script MovementController.cs

Agreguemos la referencia del objeto Animator; referencia de la variable qué guarda los estados y la definición de los estados

```
16      Animator animator;
17      string animationState = "AnimationState";
18
        5 referencias
19      enum CharStates
20      {
21          walkEast = 1,
22          walkSouth = 2,
23          walkWest = 3,
24          walkNorth = 4,
25          idleSouth = 5,
26      }
27
        Mensaje de Unity | 0 referencias
```

Inicializamos en el método start el valor del componente Animator del objeto enlazado.

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

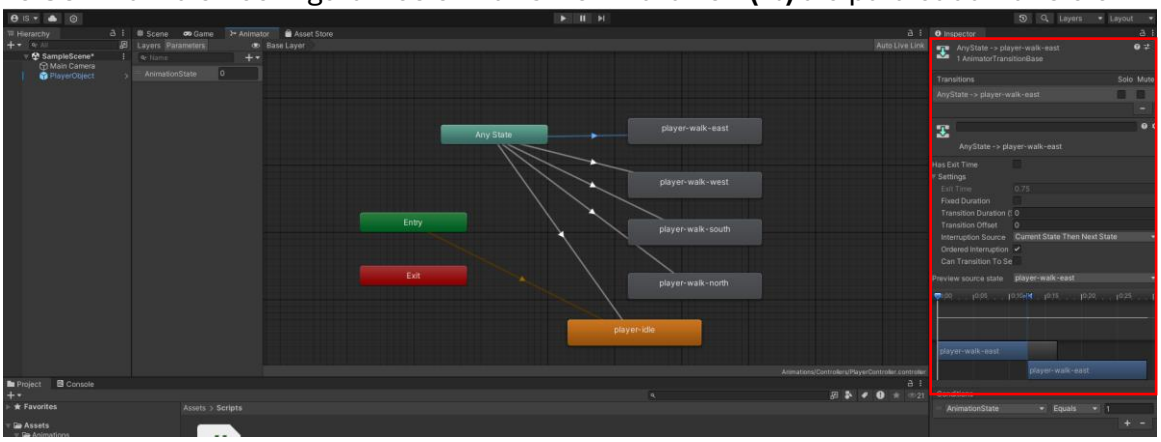
Modificamos el método Update donde se invoca a otro método qué define en base a las entradas WASD por el usuario

```
45      private void UpdateState()
46      {
47          if (movement.x > 0) // ESTE
48          {
49              animator.SetInteger(animationState, (int)CharStates.walkEast);
50          }
51          else if (movement.x < 0) // OESTE
52          {
53              animator.SetInteger(animationState, (int)CharStates.walkWest);
54          }
55          else if (movement.y > 0) // NORTE
56          {
57              animator.SetInteger(animationState, (int)CharStates.walkNorth);
58          }
59          else if (movement.y < 0) // SUR
60          {
61              animator.SetInteger(animationState, (int)CharStates.walkSouth);
62          }
63          else // IDLE
64          {
65              animator.SetInteger(animationState, (int)CharStates.idleSouth);
66          }
67      }
68
```

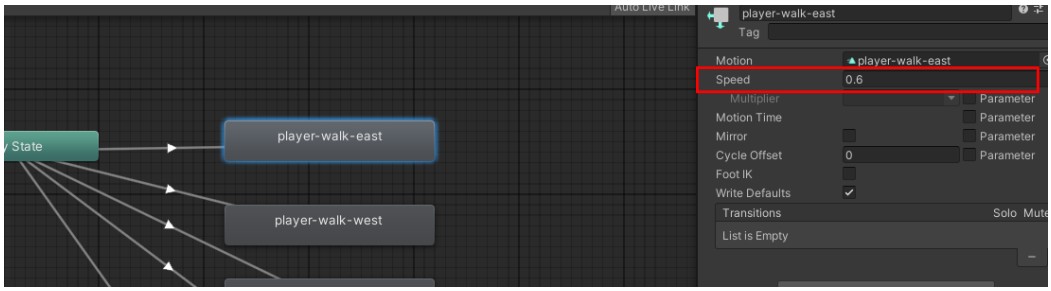
Modificamos el método FixedUpdate

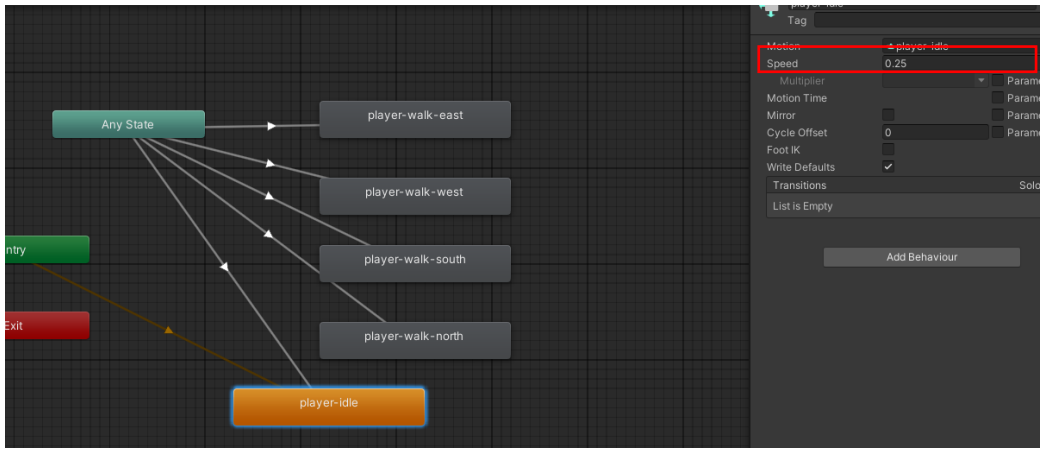
```
Mensaje de Unity | 0 referencias
69  private void FixedUpdate()
70  {
71      MoveCharacter(); // Método definido para ingresar la dirección
72  }
73
    1 referencia
74  private void MoveCharacter()
75  {
76      // Captura los datos de entrada del usuario
77      movement.x = Input.GetAxisRaw("Horizontal");
78      movement.y = Input.GetAxisRaw("Vertical");
79
80      // Conserva el rango de velocidad
81      movement.Normalize();
82
83      rb2D.velocity = movement * movementSpeed;
84  }
85  }
```

Regresamos al **Animator** de Unity y verificamos cada estado de transición. A medida que avanzamos por cada flecha de transición, desmarcamos los cuadros de opciones como **Exit Time**, **Fixed Duration**, y **Can Transition to Self**. También configuramos el **Transition Duration (%)** a **0** para cada transición.



Seleccionamos cada estado de animación de caminata del jugador y ajustamos la **velocidad** a **0.6**. Luego, ajustamos **player-idle** a **0.25**. Esto hará que las animaciones del jugador se vean de manera fluida y adecuada.

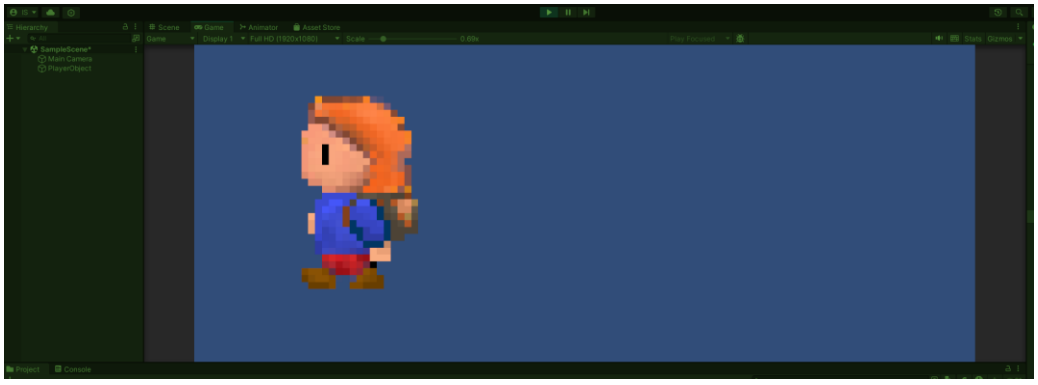




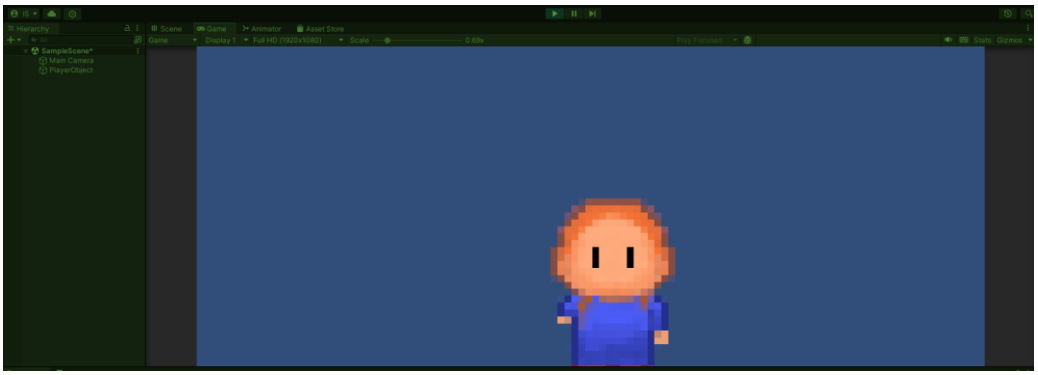
Ahora que hemos configurado una gran parte de las animaciones del jugador necesarias para nuestro juego, presionamos el botón **Play**. Luego, movemos nuestro personaje por la pantalla utilizando las teclas de flecha o **W, A, S, D**.

D

A



S



W