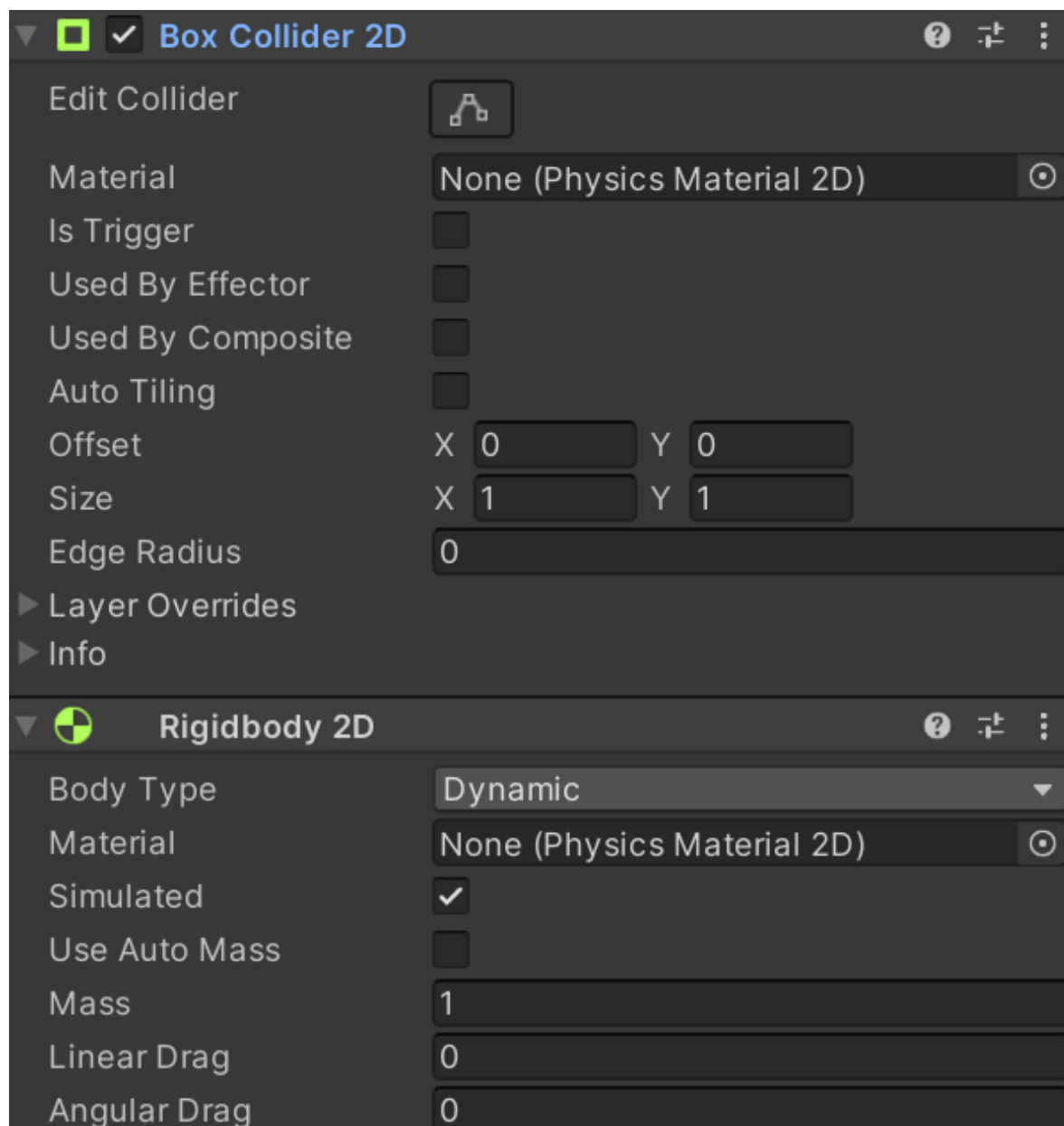
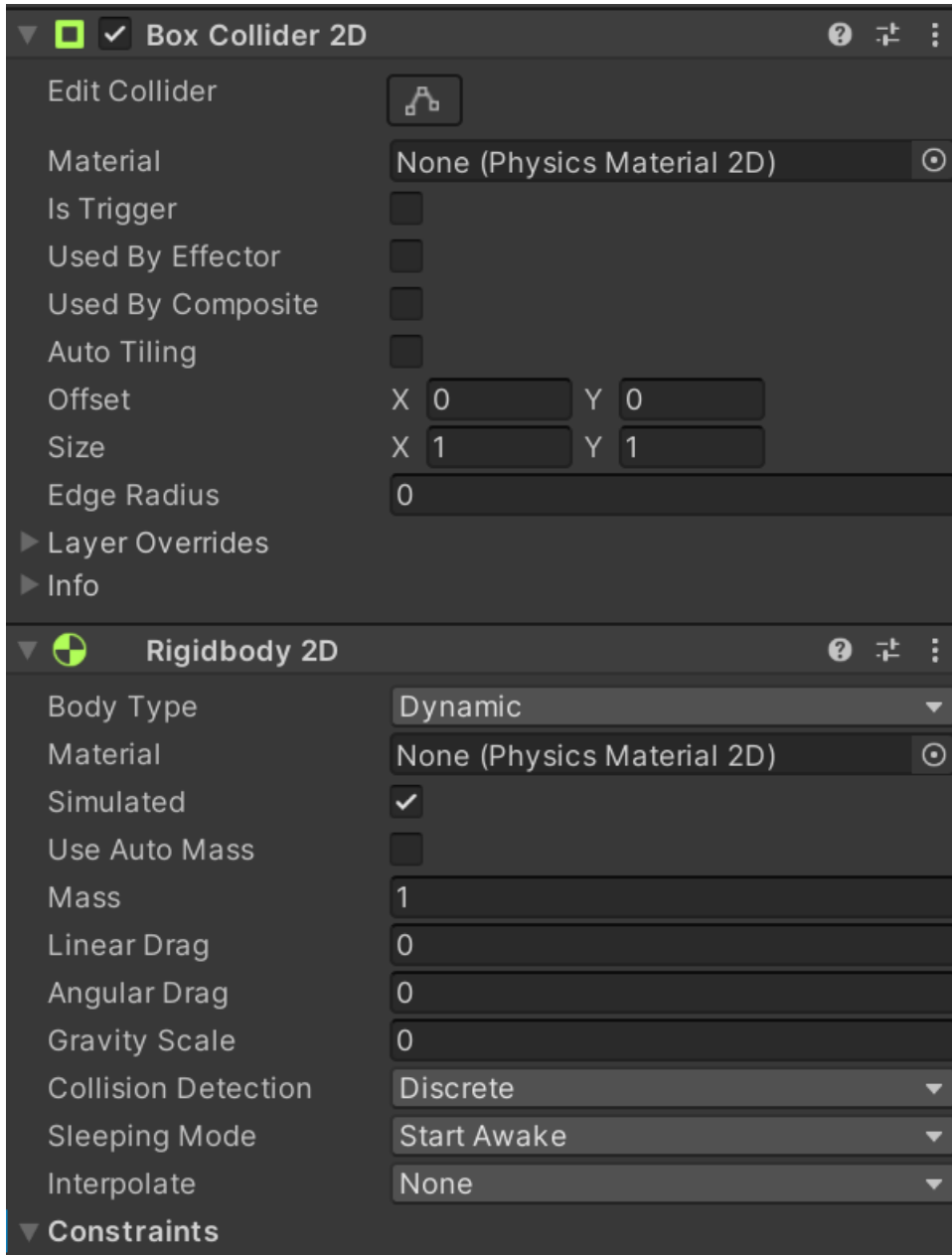


TUTORIAL 2 - IMAGEN 1

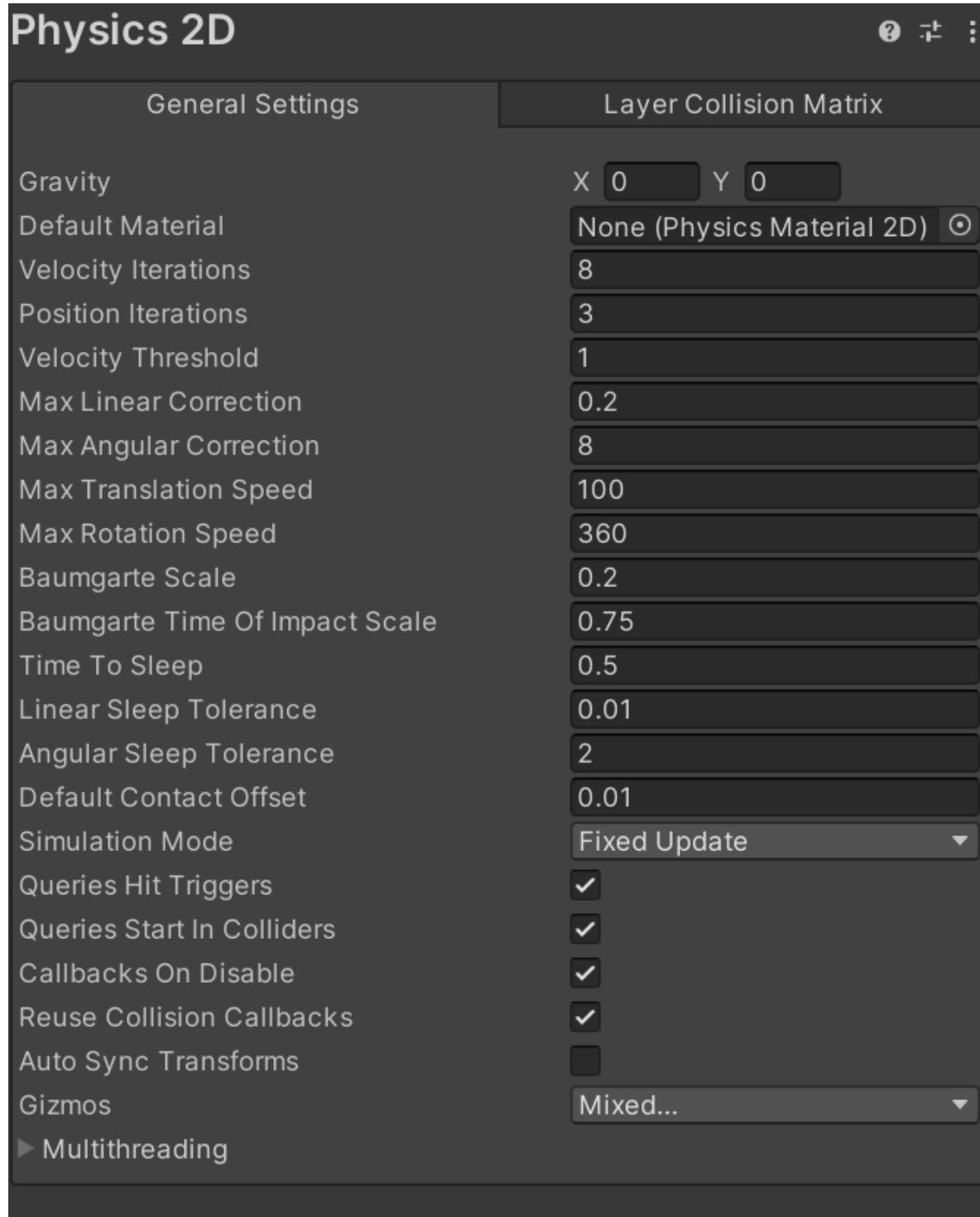
Configuración de Colliders y física básica. Agregamos Box Collider 2D tanto al PlayerObject como al EnemyObject para detectar colisiones. Los colisionadores definen los límites físicos de los objetos y permiten que el motor de física de Unity detecte cuando dos objetos entran en contacto.



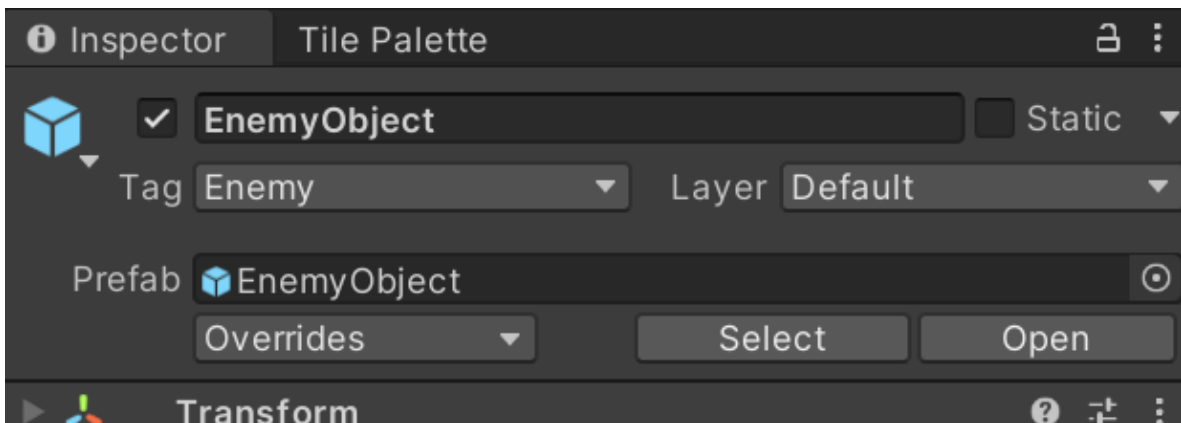
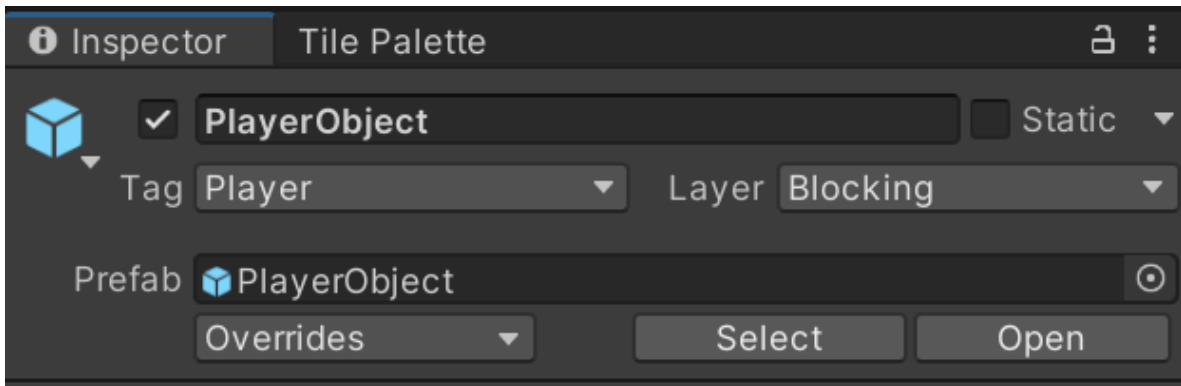
Implementación de Rigidbody 2D. Añadimos el componente Rigidbody 2D al PlayerObject configurándolo como Dynamic con Mass 1, Drag en 0 y Gravity Scale 0. Esto permite que el objeto interactúe con el motor de física sin ser afectado por la gravedad, ideal para juegos top-down.



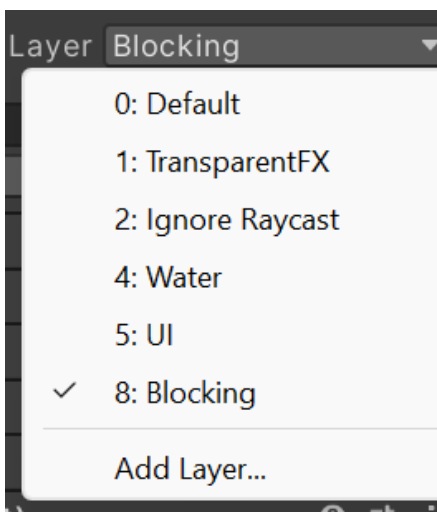
Desactivación de la gravedad global. Vamos a Edit → Project Settings → Physics 2D y cambiamos la gravedad Y de -9.81 a 0. Esto evita que los objetos caigan en nuestro juego 2D con perspectiva top-down.



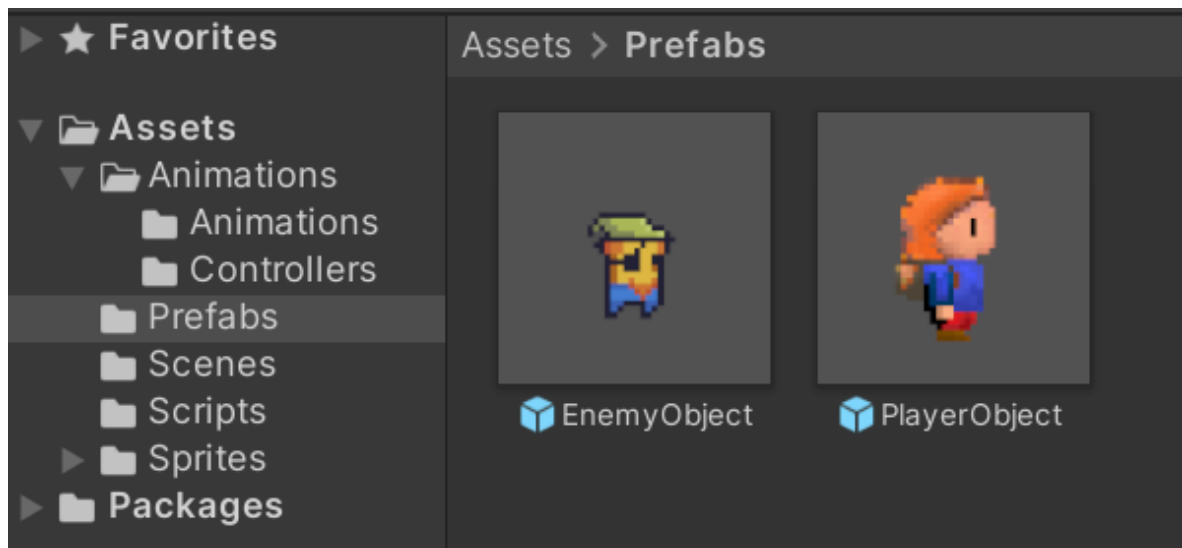
Implementación del sistema de Tags. Creamos y asignamos tags "Player" y "Enemy" a los respectivos objetos. Las tags permiten identificar y referenciar fácilmente los GameObjects durante la ejecución del juego.



Creación de Layers personalizadas. Configuramos la layer "Blocking" para objetos que no deben ser atravesados. Asignamos esta layer al PlayerObject y posteriormente a paredes, árboles y enemigos para controlar las interacciones de colisión.



Creación de Prefabs. Organizamos los objetos reutilizables creando prefabs del PlayerObject y EnemyObject. Los arrastramos desde la Hierarchy a la carpeta Prefabs, creando plantillas que pueden ser instanciadas múltiples veces.



Desarrollo del script MovementController. Creamos un nuevo script C# que controla el movimiento del personaje. Configuramos variables para velocidad, referencia al Rigidbody2D, y comenzamos la implementación del sistema de entrada.

```
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     Animator animator; //Referencia a componente animator
15     string animationState = "AnimationState"; //Variable en Animator
16
17     //Enumeración de los estados - CORREGIDO: idleSouth debe ser 0
18     enum CharStates
19     {
20         walkEast = 1,
21         walkSouth = 2,
22         walkWest = 3,
23         walkNorth = 4,
24         idleSouth = 0 // CAMBIADO de 5 a 0
25     }
26
27     // Start is called before the first frame update
28     void Start()
29     {
30         //Establece el componente Rigidbody2D enlazado
31         rb2D = GetComponent<Rigidbody2D>();
32     }
33 }
```

```

30     //Establece el componente Rigidbody2D enlazado
31     rb2D = GetComponent<Rigidbody2D>();
32     //Establece valor de componente Animator del objeto ligado
33     animator = GetComponent<Animator>();
34 }
35
36 // Update is called once per frame
37 void Update()
38 {
39     this.UpdateState(); //Invoca al método
40 }
41
42 private void FixedUpdate()
43 {
44     MoveCharacter(); //Método definido para ingresar la dirección
45 }
46
47 /*
48 * Método que define la animación a ejecutar en base al movimiento realizado por el usuario.
49 */
50 private void UpdateState()
51 {
52     // DEBUG: Mostrar estado actual del movimiento y animación
53     int currentState = animator.GetInteger(animationState);
54     Debug.Log($"MOVIMIENTO - X: {movement.x}, Y: {movement.y} | Estado Actual: {currentState}");
55
56     if (movement.x > 0)
57     { //ESTE
58         Debug.Log("● CAMBIANDO a WALK EAST (1)");

```

```

58         Debug.Log("● CAMBIANDO a WALK EAST (1)");
59         animator.SetInteger(animationState, (int)CharStates.walkEast);
60     }
61     else if (movement.x < 0)
62     { //OESTE
63         Debug.Log("● CAMBIANDO a WALK WEST (3)");
64         animator.SetInteger(animationState, (int)CharStates.walkWest);
65     }
66     else if (movement.y > 0)
67     { //NORTE
68         Debug.Log("● CAMBIANDO a WALK NORTH (4)");
69         animator.SetInteger(animationState, (int)CharStates.walkNorth);
70     }
71     else if (movement.y < 0)
72     { //SUR
73         Debug.Log("● CAMBIANDO a WALK SOUTH (2)");
74         animator.SetInteger(animationState, (int)CharStates.walkSouth);
75     }
76     else
77     { //IDLE
78         Debug.Log("● CAMBIANDO a IDLE SOUTH (0)");
79         animator.SetInteger(animationState, (int)CharStates.idleSouth);
80     }
81 }
82
83 /*
84 * Método para mover el personaje
85 */
86 private void MoveCharacter()
87 {
88     //Captura los datos de entrada del usuario
89     movement.x = Input.GetAxisRaw("Horizontal");

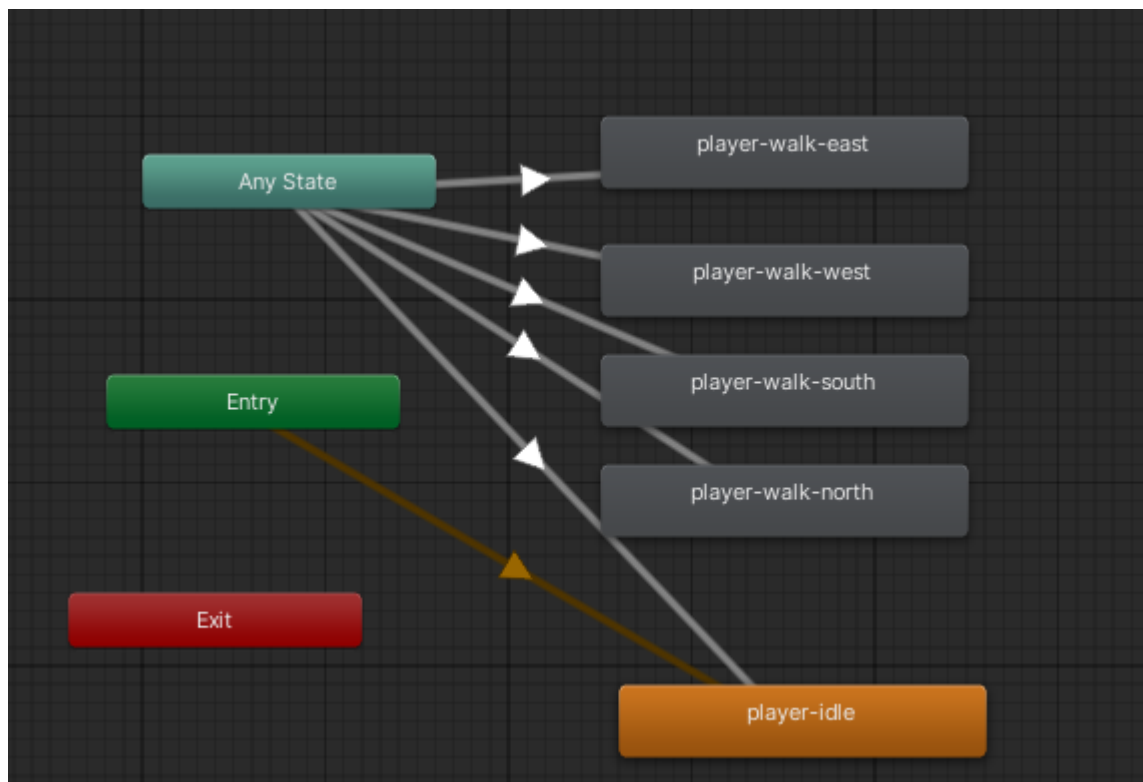
```

```

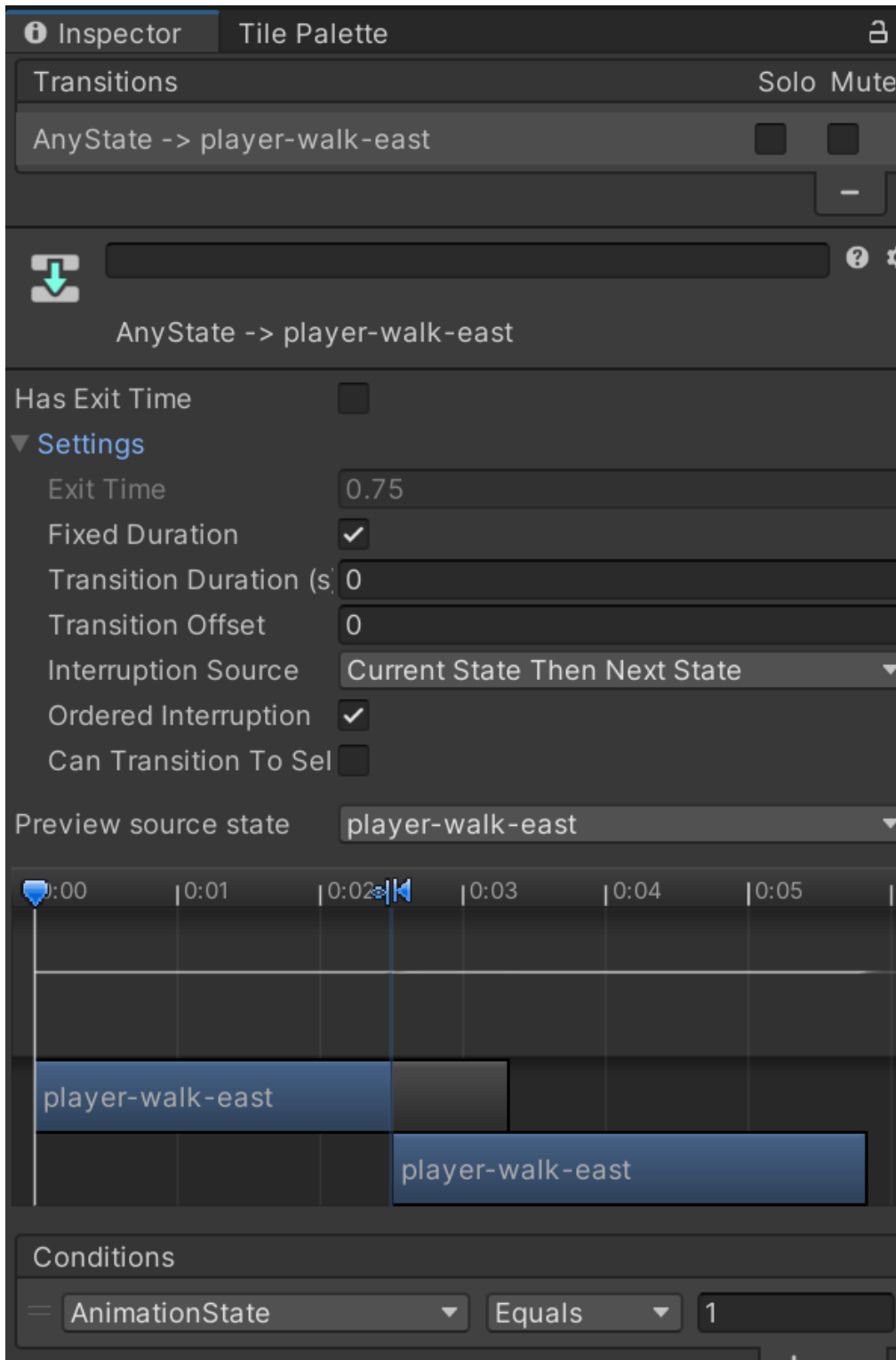
80     }
81   }
82
83   /*
84   * Método para mover el personaje
85   */
86   private void MoveCharacter()
87   {
88       //Captura los datos de entrada del usuario
89       movement.x = Input.GetAxisRaw("Horizontal");
90       movement.y = Input.GetAxisRaw("Vertical");
91
92       // DEBUG: Mostrar inputs raw
93       if (movement.x != 0 || movement.y != 0)
94       {
95           Debug.Log($"INPUT - Horizontal: {Input.GetAxisRaw("Horizontal")}, Vertical: {Input.GetAxisRaw("Vertical")}");
96       }
97
98       //Conserva el rango de velocidad
99       movement.Normalize();
100       rb2D.velocity = movement * movementSpeed;
101
102       // Prevenir rotación no deseada
103       rb2D.angularVelocity = 0f;
104   }
105

```

Creación de transiciones de animación. Conectamos "Any State" con todos los estados de animación mediante transiciones. Esto permite cambiar entre animaciones desde cualquier estado actual del personaje.



Configuración de parámetros del Animator. Creamos el parámetro "AnimationState" de tipo Int que controlará las transiciones entre animaciones. Establecemos las condiciones para cada transición basadas en valores numéricos.




Inspector



Tile Palette

Transitions

Solo Mute

AnyState -> player-walk-west





AnyState -> player-walk-west

Has Exit Time

☐

▼ Settings

Exit Time

0.75

Fixed Duration

☐

Transition Duration (%)

0

Transition Offset

0

Interruption Source

Current State Then Next State

▼

Ordered Interruption

☒

Can Transition To Self

☐

Preview source state

player-walk-east

▼

0:00

0:01

0:02

0:03

0:04

0:05

0:06

player-walk-east

player-walk-west

Conditions

=

AnimationState

▼

Equals


▼



3

Transitions

Solo Mute

AnyState -> player-walk-south





AnyState -> player-walk-south

Has Exit Time

☐

Settings

Exit Time

0.75

Fixed Duration

☐

Transition Duration (%)

0

Transition Offset

0

Interruption Source

Current State Then Next State

Ordered Interruption

☒

Can Transition To Self

☐

Preview source state

player-walk-east


0:00

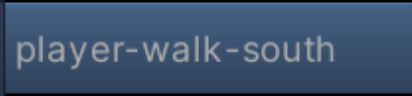
0:01

0:02

0:03

0:04





Conditions

=

AnimationState

Equals

2


+



-

Transitions

Solo Mute

AnyState -> player-walk-north



AnyState -> player-walk-north

Has Exit Time

☐

▼ Settings

Exit Time

0.75

Fixed Duration

☒

Transition Duration (s)

0

Transition Offset

0

Interruption Source

Current State Then Next State ▼

Ordered Interruption



☒

Can Transition To Self

☐

Preview source state

player-walk-east ▼

 0:00 | 0:01 | 0:02  | 0:03 | 0:04

player-walk-east

player-walk-north

Conditions

= AnimationState ▼

Equals ▼

4

Inspector


Tile Palette



Transitions

Solo Mute

AnyState -> player-idle

-



AnyState -> player-idle

Has Exit Time

☐

▼ Settings

Exit Time

0.75

Fixed Duration

☒

Transition Duration (s)

0

Transition Offset

0

Interruption Source

Current State Then Next State

▼

Ordered Interruption

☒

Can Transition To Self

☐

Preview source state

player-walk-east

▼

0:00

0:01

0:02

0:03

0:04

0:05

0:06

player-walk-east

player-idle

Conditions

=

AnimationState

▼

Equals

▼

0

