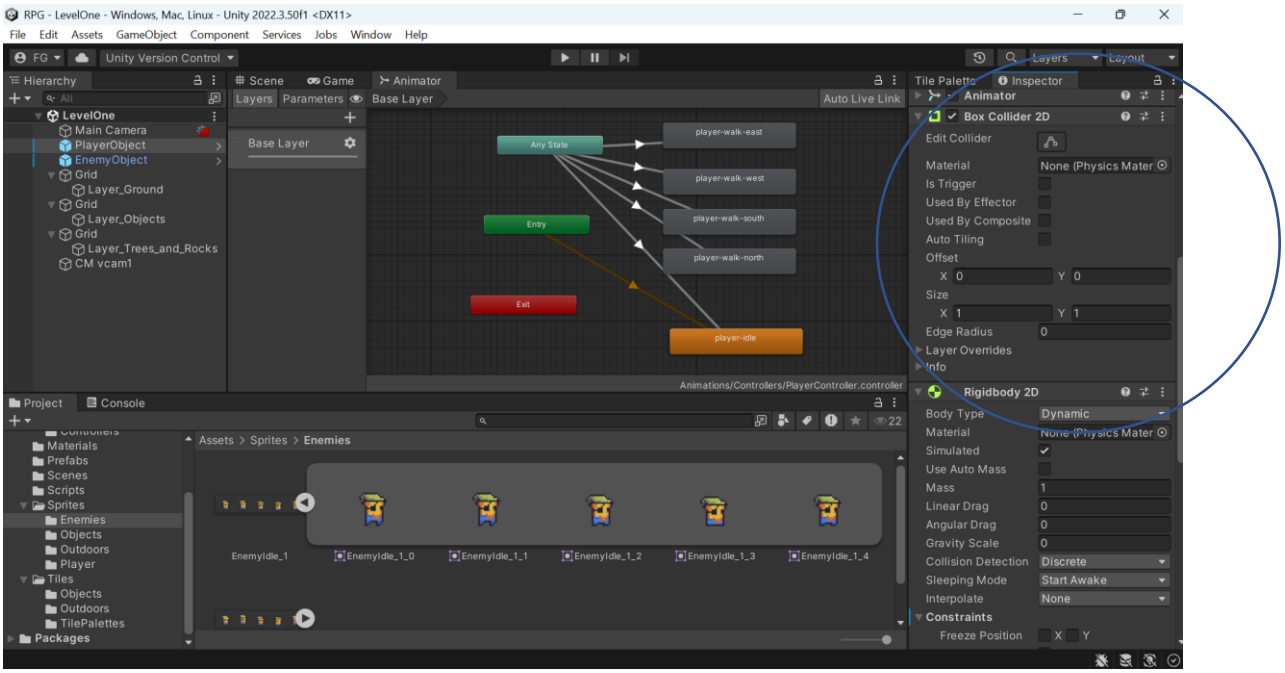
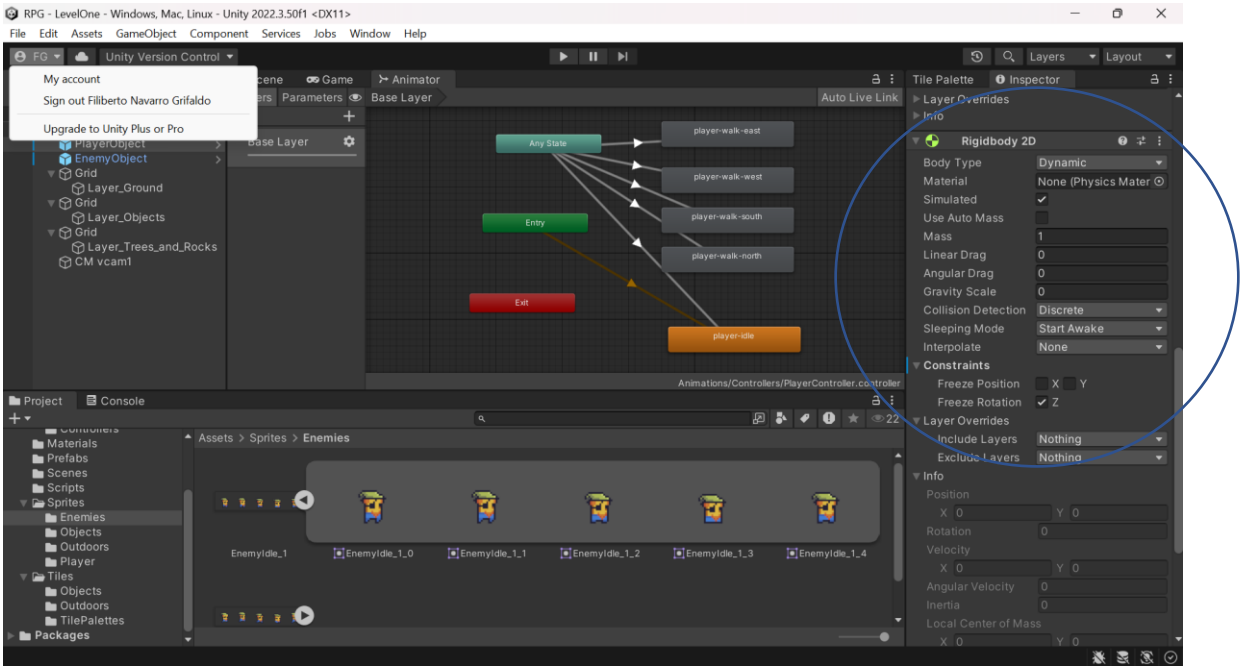
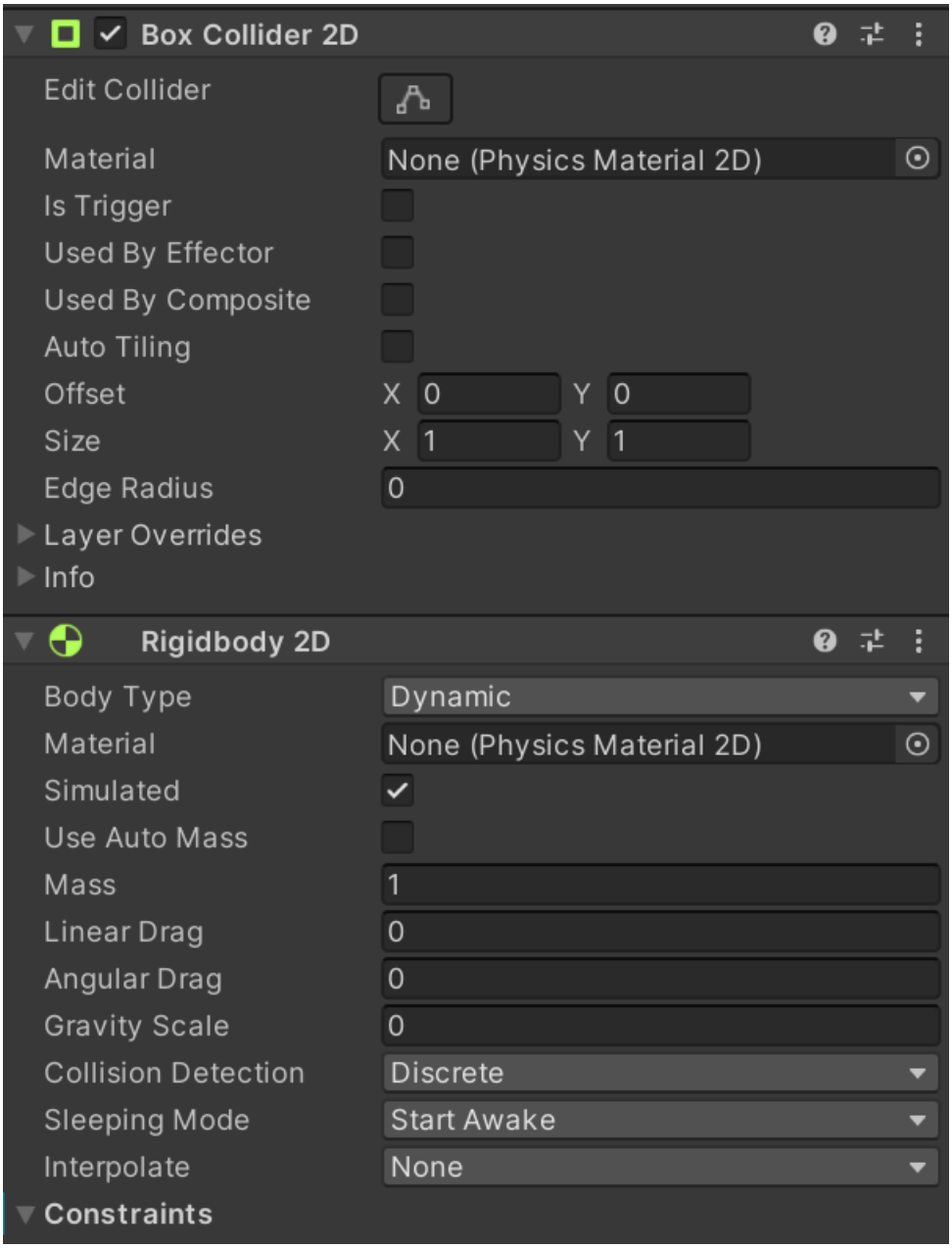
TUTORIAL 2 - IMAGEN 1

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.

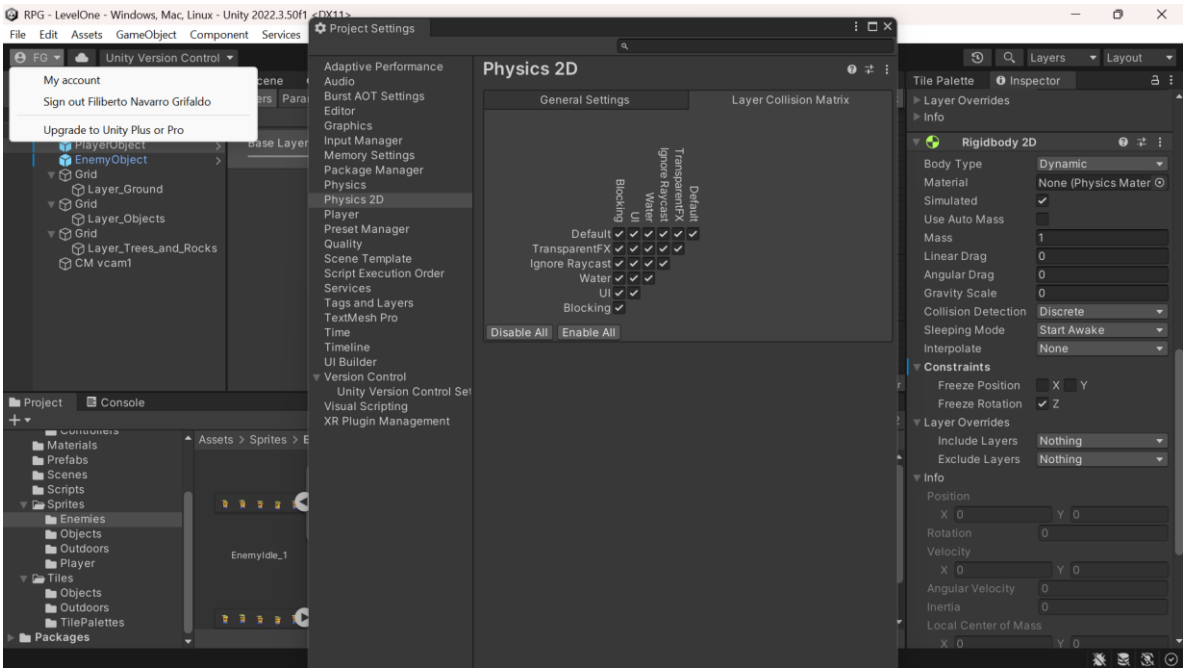




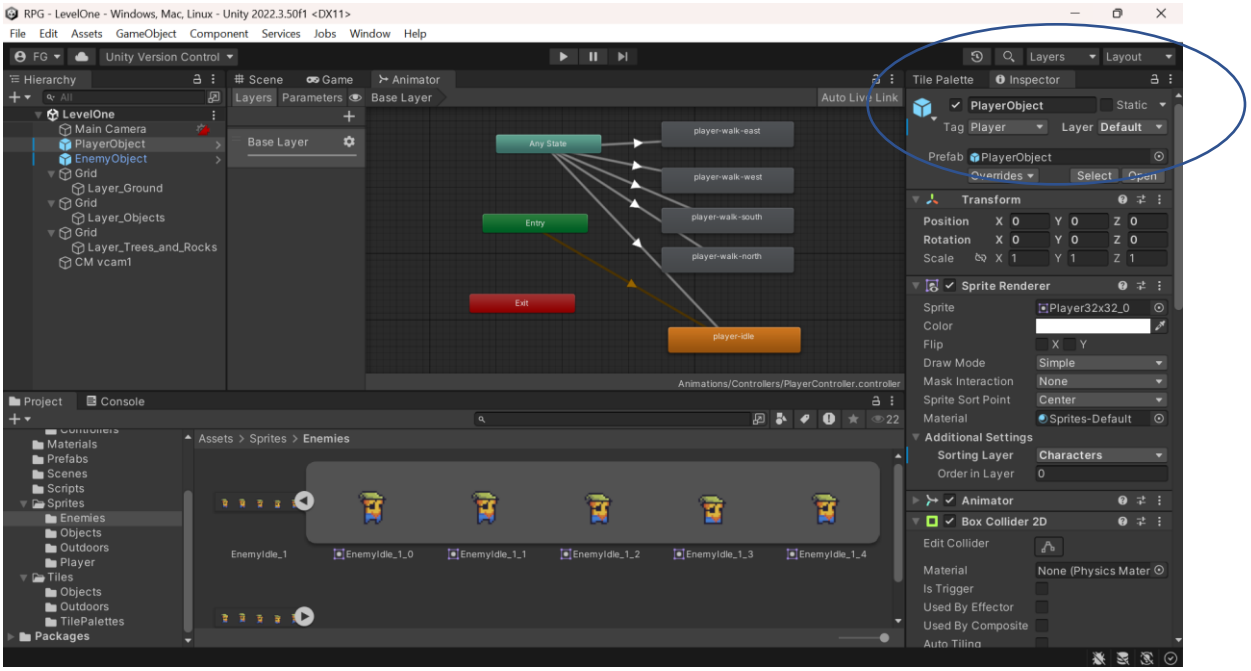
Implementación de Rigidbody 2D. Añadimos el componente Rigidbody 2D al PlayerObject configurándolo como Dynamic con Mass 1, Drag en 0 y Gravity Scale 0. Esto permite que el objeto interactúe con el motor de física sin ser afectado por la gravedad, ideal para juegos top-down.



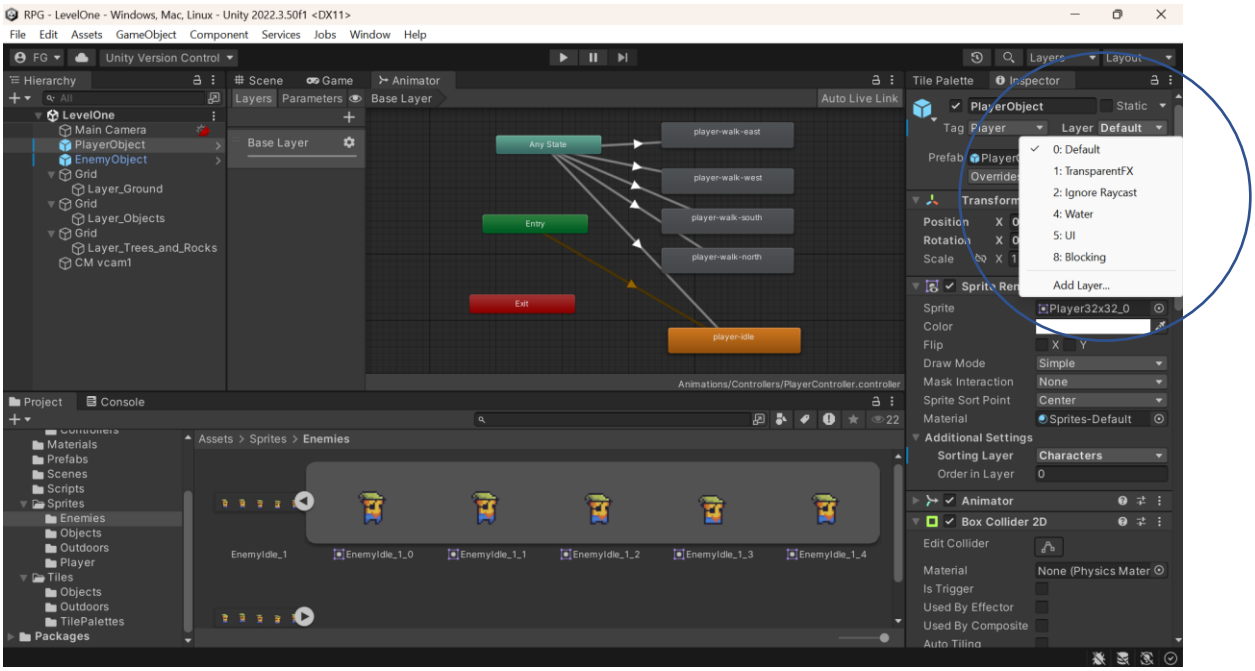
Desactivación de la gravedad global. Vamos a Edit → Project Settings → Physics 2D y cambiamos la gravedad Y de -9.81 a 0. Esto evita que los objetos caigan en nuestro juego 2D con perspectiva top-down.



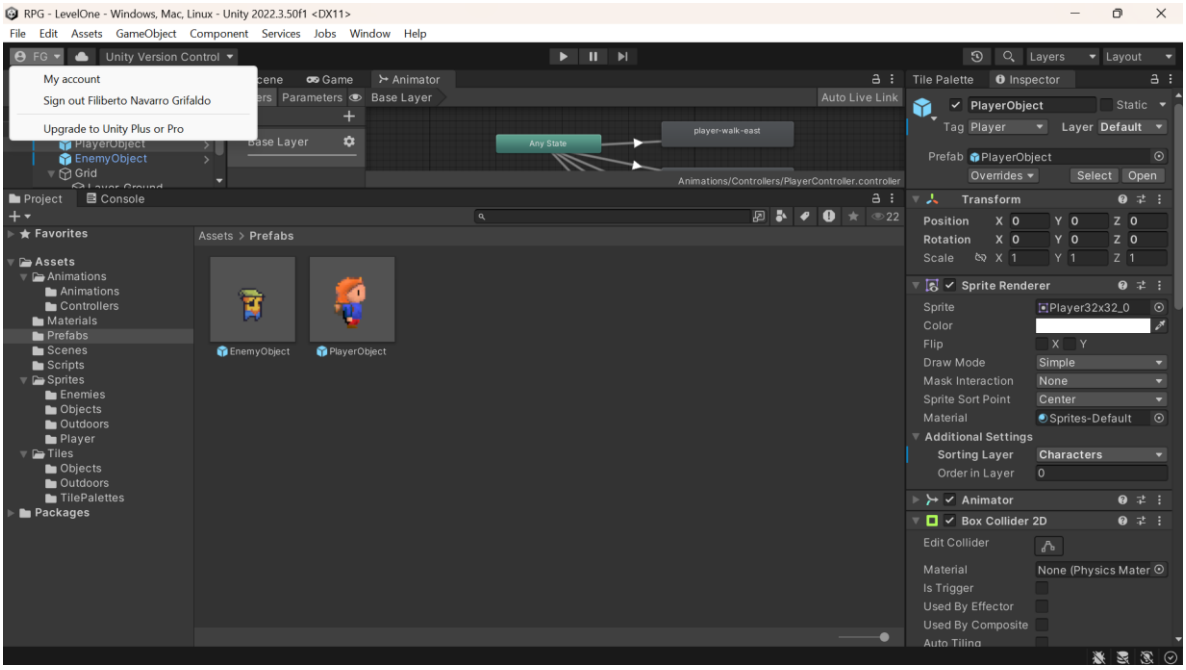
Implementación del sistema de Tags. Creamos y asignamos tags "Player" y "Enemy" a los respectivos objetos. Las tags permiten identificar y referenciar fácilmente los GameObjects durante la ejecución del juego.



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.

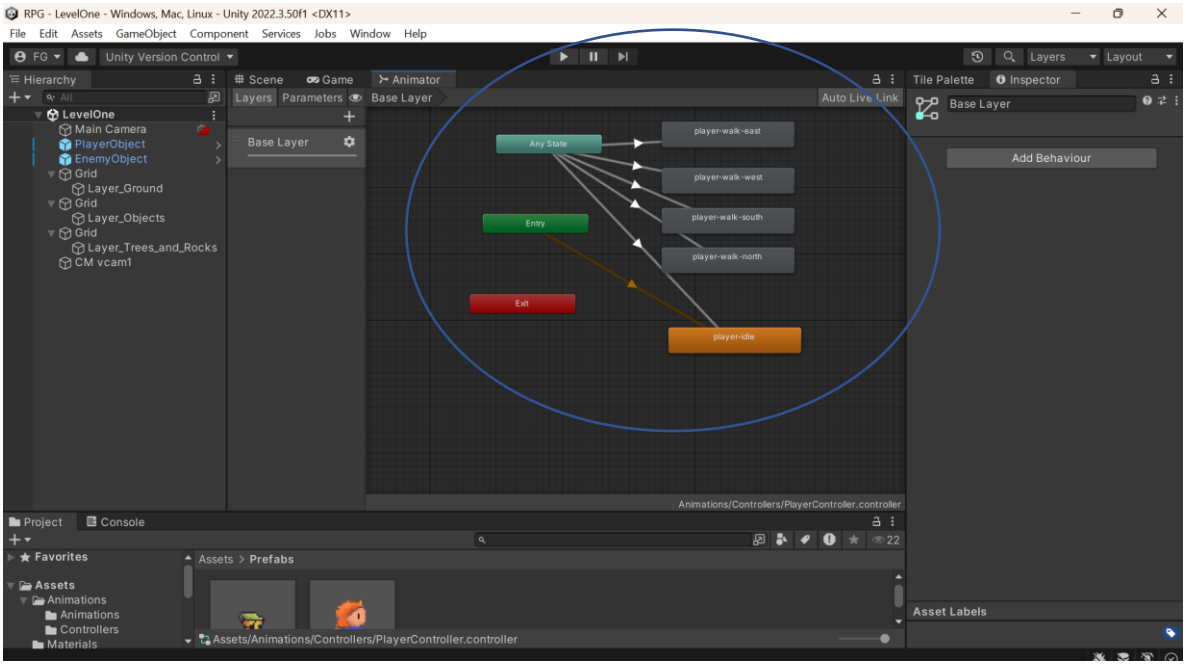
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

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;

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

    //Enumeración de los estados - CORREGIDO: idleSouth debe ser 0
    enum CharStates
    {
        walkEast = 1,
        walkSouth = 2,
        walkWest = 3,
        walkNorth = 4,
        idleSouth = 0  // CAMBIADO de 5 a 0
    }

    // Start is called before the first frame update
    void Start()
    {
        //Establece el componente Rigidbody2D enlazado
        rb2D = GetComponent<Rigidbody2D>();
```

```
        //Establece el componente Rigidbody2D enlazado
        rb2D = GetComponent<Rigidbody2D>();
        //Establece valor de componente Animator del objeto ligado
        animator = GetComponent<Animator>();
    }

    // Update is called once per frame
    void Update()
    {
        this.UpdateState(); //Invoca al método
    }

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

    /*
    * Método que define la animación a ejecutar en base al movimiento realizado por el usuario.
    */
    private void UpdateState()
    {
        // DEBUG: Mostrar estado actual del movimiento y animación
        int currentState = animator.GetInteger(animationState);
        Debug.Log($"MOVIMIENTO - X: {movement.x}, Y: {movement.y} | Estado Actual: {currentState}");

        if (movement.x > 0)
        { //ESTE
            Debug.Log("🟤 CAMBIANDO a WALK EAST (1)");
```

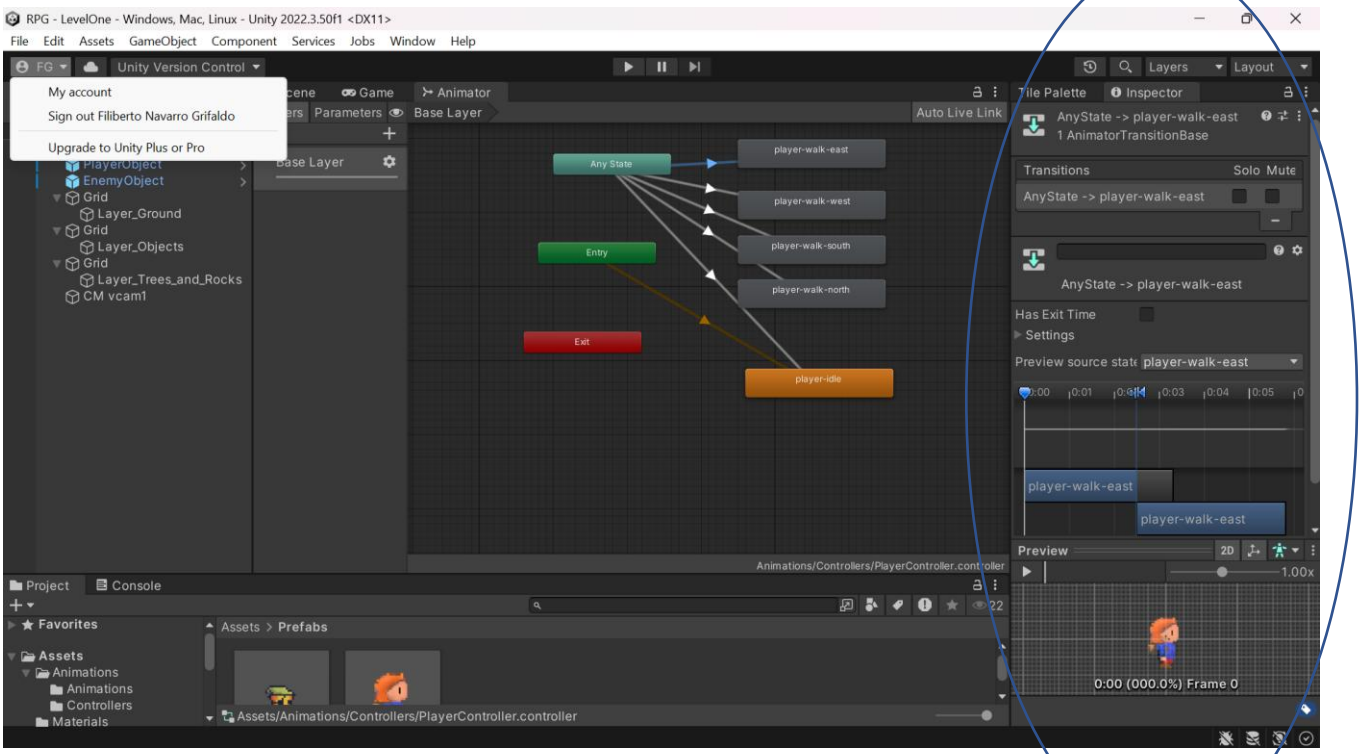
```
58                  Debug.Log("🟤 CAMBIANDO a WALK EAST (1)");
59                  animator.SetInteger(animationState, (int)CharStates.walkEast);
60              }
61              else if (movement.x < 0)
62              { //OESTE
63                  Debug.Log("🟤 CAMBIANDO a WALK WEST (3)");
64                  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");
```

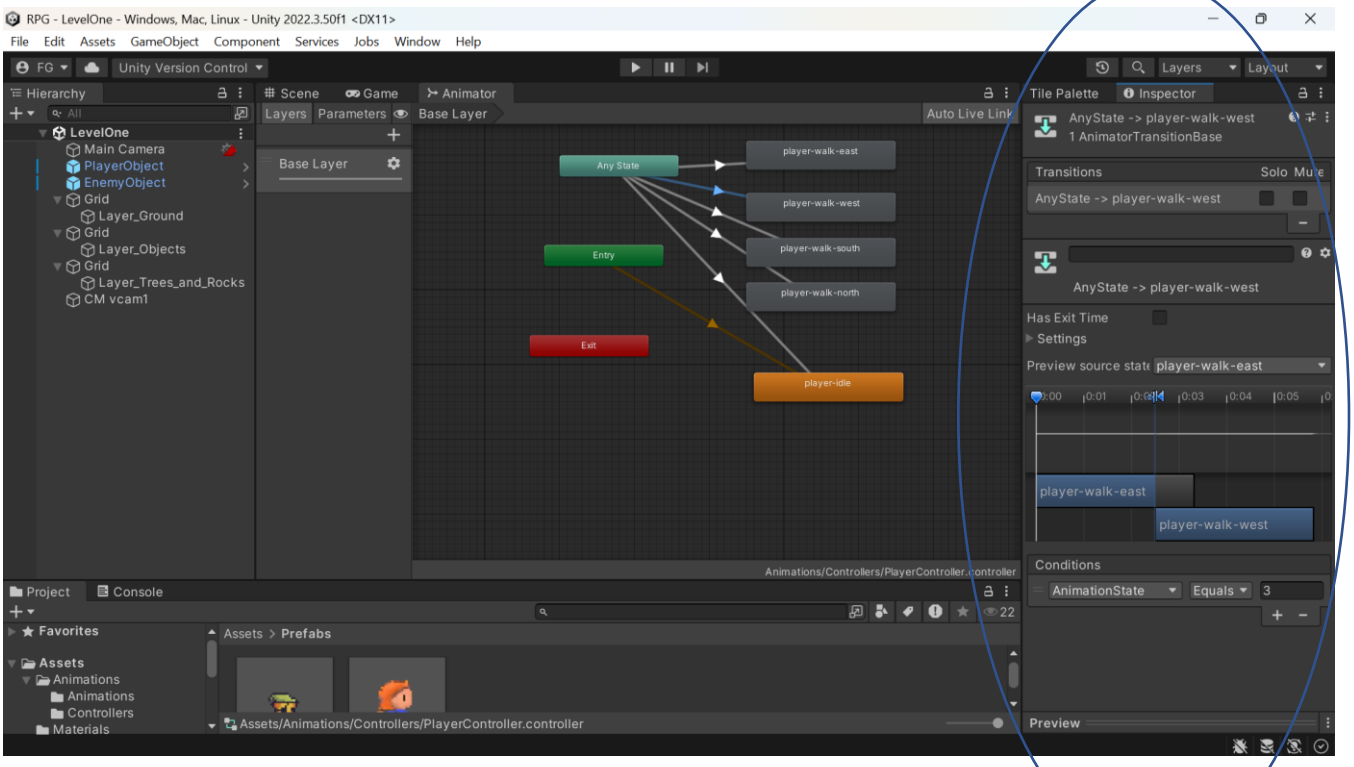
```
80              }
81          }
82
83          /*
84          * Método para mover el personaje
85          */
86          private void MoveCharacter()
87          {
88              //Captura los datos de entrada del usuario
89              movement.x = Input.GetAxisRaw("Horizontal");
90              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 }
```

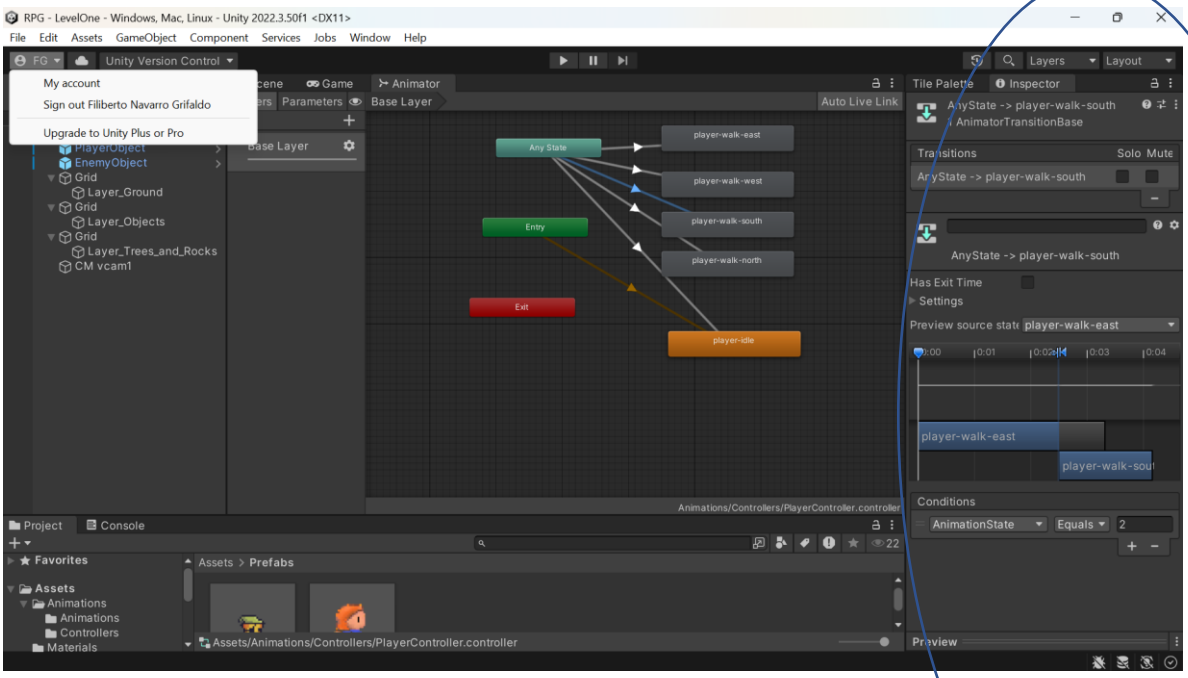
Creación de transiciones de animación. Conectamos "Any State" con todos los estados de animación mediante transiciones. Esto permite cambiar entre animaciones desde cualquier estado actual del personaje.

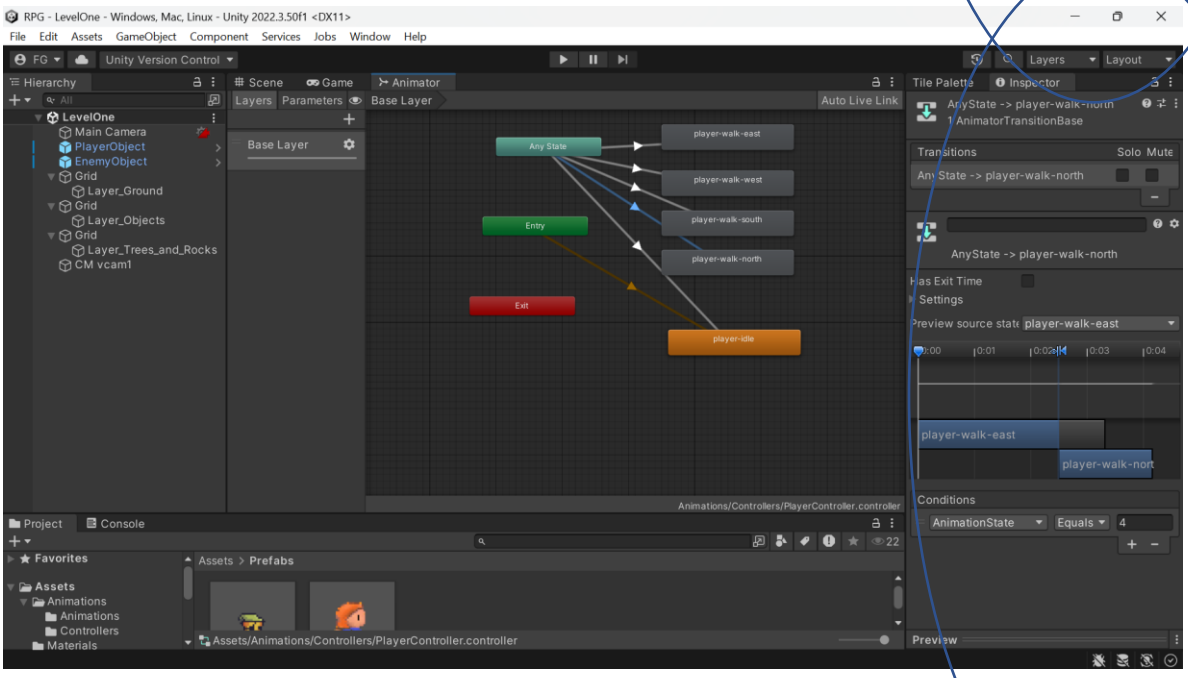


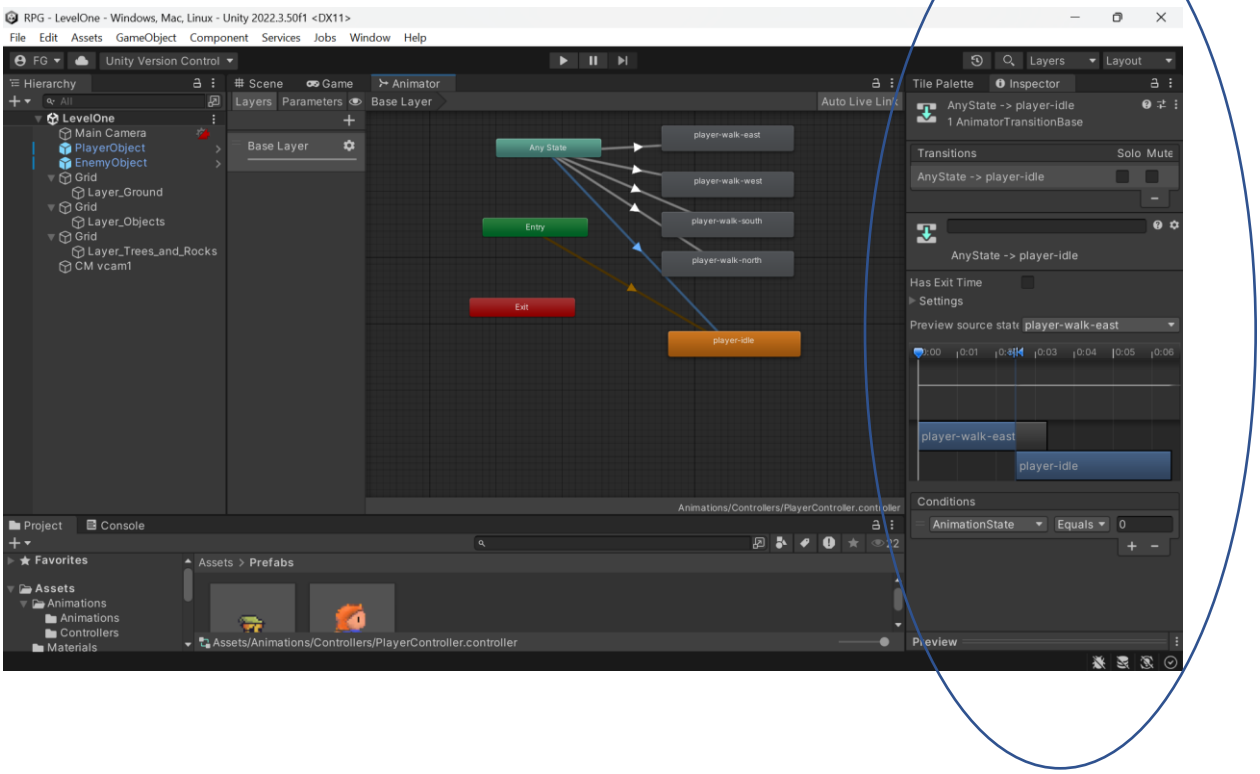
Configuración de parámetros del Animator. Creamos el parámetro "AnimationState" de tipo Int que controlará las transiciones entre animaciones. Establecemos las condiciones para cada transición basadas en valores numéricos.


RPG - LevelOne - Windows, Mac, Linux - Unity 2022.3.50f1 <DX11>
File Edit Assets GameObject Component Services Jobs Window Help
Unity Version Control
My account
Sign out Filiberto Navarro Grifaldo
Upgrade to Unity Plus or Pro
Layers
Layout
Hierarchy
Scene
Game
Animator
Parameters
Base Layer
Auto Live Link
Any State
Entry
Exit
player-walk-east
player-walk-west
player-walk-south
player-walk-north
player-idle
Tile Palette
Inspector
AnyState -> player-walk-east
1 AnimatorTransitionBase
Transitions
Solo Mute
AnyState -> player-walk-east
Has Exit Time
Settings
Preview source state
player-walk-east
Preview
2D
1.00x
0:00 (000.0%) Frame 0
Project
Console
Favorites
Assets
Prefabs
Animations
Controllers
Materials
Assets/Animations/Controllers/PlayerController.controller


RPG - LevelOne - Windows, Mac, Linux - Unity 2022.3.50f1 <DX11>
File Edit Assets GameObject Component Services Jobs Window Help
Unity Version Control
Layers
Layout
Hierarchy
LevelOne
Main Camera
PlayerObject
EnemyObject
Grid
Layer_Ground
Layer_Objects
Layer_Trees_and_Rocks
CM vcam1
Scene
Game
Animator
Base Layer
Auto Live Link
Any State
Entry
Exit
player-walk-east
player-walk-west
player-walk-south
player-walk-north
player-idle
Tile Palette
Inspector
AnyState -> player-walk-west
1 AnimatorTransitionBase
Transitions
Solo Mute
AnyState -> player-walk-west
Has Exit Time
Settings
Preview source state
player-walk-east
player-walk-west
Conditions
AnimationState
Equals
3
Preview
Project
Console
Favorites
Assets
Prefabs
Animations
Controllers
Materials
Assets/Animations/Controllers/PlayerController.controller


RPG - LevelOne - Windows, Mac, Linux - Unity 2022.3.50f1 <DX11>
File Edit Assets GameObject Component Services Jobs Window Help
FG
Unity Version Control
My account
Sign out Filiberto Navarro Grifaldo
Upgrade to Unity Plus or Pro
Layers
Layout
Scene
Game
Animator
Base Layer
Parameters
Auto Live Link
Any State
Entry
Exit
player-walk-east
player-walk-west
player-walk-south
player-walk-north
player-idle
Tile Palette
Inspector
AnyState -> player-walk-south
AnimatorTransitionBase
Transitions
Solo Mute
AnyState -> player-walk-south
Has Exit Time
Settings
Preview source state
player-walk-east
Conditions
AnimationState
Equals
2
Project
Console
Favorites
Assets
Animations
Controllers
Materials
Prefabs
Assets/Animations/Controllers/PlayerController.controller
Preview


RPG - LevelOne - Windows, Mac, Linux - Unity 2022.3.50f1 <DX11>
File Edit Assets GameObject Component Services Jobs Window Help
Hierarchy
LevelOne
Main Camera
PlayerObject
EnemyObject
Grid
Layer_Ground
Layer_Objects
Layer_Trees_and_Rocks
CM vcam1
Base Layer
Any State
Entry
Exit
player-walk-east
player-walk-west
player-walk-south
player-walk-north
player-idle
AnyState -> player-walk-north
AnimatorTransitionBase
AnyState -> player-walk-north
Conditions
AnimationState
Equals
4
Assets/Animations/Controllers/PlayerController.controller


RPG - LevelOne - Windows, Mac, Linux - Unity 2022.3.50f1 <DX11>
File Edit Assets GameObject Component Services Jobs Window Help
Hierarchy
LevelOne
Main Camera
PlayerObject
EnemyObject
Grid
Layer_Ground
Grid
Layer_Objects
Grid
Layer_Trees_and_Rocks
CM vcam1
Scene
Game
Animator
Layers
Parameters
Base Layer
Any State
Entry
Exit
player-walk-east
player-walk-west
player-walk-south
player-walk-north
player-idle
Tile Palette
Inspector
AnyState -> player-idle
1 AnimatorTransitionBase
Transitions
Solo Mute
AnyState -> player-idle
Has Exit Time
Settings
Preview source state
player-walk-east
player-idle
Conditions
AnimationState
Equals
0
Project
Console
Favorites
Assets
Animations
Controllers
Materials
Assets > Prefabs
Assets/Animations/Controllers/PlayerController.controller
Preview