Script: Mov_Flechas

Descripción

Este script controla el movimiento del jugador, incluyendo desplazamiento horizontal, salto, ataques y animaciones asociadas. Utiliza el nuevo sistema de entrada de Unity y maneja diversas interacciones como colisiones con enemigos y rampas.

Ubicación

`Assets/Scripts/Mov_Flechas.cs`

Clase: Mov_Flechas

Propiedades

- **Configuración de Movimiento**:
  - `public float velocidadInicial`: Velocidad inicial del jugador.
  - `public float incrementoVelocidad`: Incremento de la velocidad por segundo.
  - `public float velocidadMax`: Velocidad máxima que puede alcanzar el jugador.
  - `public float suavizadoDesplazamiento`: Factor de suavizado para el desplazamiento del jugador.
- **Movimiento Horizontal**:
  - `private float desplazamientoDer = 0f`: Desplazamiento hacia la derecha.
  - `private float desplazamientoIzq = 0f`: Desplazamiento hacia la izquierda.
  - `private float frenado`: Factor de frenado.
- **Rigidbody**:
  - `public Rigidbody2D rb2D`: Componente Rigidbody2D del jugador.
- **Velocidad**:
  - `private Vector3 velocidad = Vector3.zero`: Vector de velocidad para el suavizado del movimiento.
- **Estados de Movimiento**:
  - `private bool Salto = false`: Indica si el jugador está saltando.
  - `private bool DesplazarDerecha = false`: Indica si el jugador se está desplazando hacia la derecha.
  - `private bool DesplazarIzquierda = false`: Indica si el jugador se está desplazando hacia la izquierda.
  - `public bool Frenar = false`: Indica si el jugador está frenando.
  - `private bool MirarDer = true`: Indica si el jugador está mirando hacia la derecha.

- **Animaciones**:
  - `public float delayAnim = 1f`: Retraso para desactivar la animación de salto.
  - `public float delayAtackAnim = 1f`: Retraso para desactivar la animación de ataque.
- **Instrucciones**:
  - `public float delayInstructions = 3f`: Retraso para mostrar las instrucciones.
  - `public GameObject instructions1`: Objeto que muestra las primeras instrucciones.
- **Ataque**:
  - `public float duracionAtaque = 1f`: Duración del ataque en segundos.
  - `public float radioAtaque = 0.5f`: Radio de detección del ataque.
  - `private bool atacando = false`: Indica si el jugador está atacando.
- **Salto**:
  - `public float fuerzaSalto`: Fuerza del salto del jugador.
  - `public LayerMask esSuelo`: Capas consideradas como suelo.
  - `public Transform DetectorSuelo`: Transform para detectar si el jugador está en el suelo.
  - `public Vector3 tamañoDetector`: Tamaño del detector de suelo.
  - `private bool enSuelo`: Indica si el jugador está en el suelo.
- **Animación y Sonido**:
  - `[SerializeField] Animator animator`: Componente Animator para las animaciones del jugador.
  - `[SerializeField] AudioSource audioSource`: Fuente de audio para los efectos de sonido.
  - `[SerializeField] PlayerSoundEffects sonidoCode`: Script para manejar los efectos de sonido del jugador.
- **Movimientos**:
  - `private Movimientos movimientos`: Referencia al script de movimientos para el nuevo sistema de entrada.

Métodos

Awake()

Inicializa el sistema de movimientos.

```
csharp
Copiar código
private void Awake()
{
    movimientos = new Movimientos();
}
```

OnEnable()

Habilita el sistema de movimientos.

```
csharp
Copiar código
private void OnEnable()
{
    movimientos.Enable();
}
```

OnDisable()

Deshabilita el sistema de movimientos.

```
csharp
Copiar código
private void OnDisable()
{
    movimientos.Disable();
}
```

Update()

Actualiza las variables de desplazamiento y manejo de entradas para el movimiento, salto y ataque.

```
csharp
Copiar código
void Update()
{
    desplazamientoDer =
movimientos.Desplazamiento.MovDerecha.ReadValue<float>() *
velocidadInicial;
    desplazamientoIzq =
movimientos.Desplazamiento.MovIzquierda.ReadValue<float>() * -
velocidadInicial;
    frenado = movimientos.Desplazamiento.Frenar.ReadValue<float>();

    if (movimientos.Desplazamiento.MovDerecha.triggered)
    {
        DesplazarIzquierda = false;
        DesplazarDerecha = true;
        Frenar = false;
    }

    if (movimientos.Desplazamiento.MovIzquierda.triggered)
    {
        DesplazarDerecha = false;
        DesplazarIzquierda = true;
        Frenar = false;
    }

    if (movimientos.Desplazamiento.Frenar.triggered)
```

```
    {
        Frenar = true;
    }

    if (DesplazarDerecha && !Frenar)
    {
        desplazamientoDer = velocidadInicial;
    }
    else if (DesplazarIzquierda && !Frenar)
    {
        desplazamientoIzq = -velocidadInicial;
    }

    if (movimientos.Desplazamiento.Saltar.triggered)
    {
        Salto = true;
    }

    if (movimientos.Desplazamiento.Atacar.triggered && !atacando)
    {
        ActivarAtaque();
    }
}
```

OnCollisionEnter2D(Collision2D collision)

Gestiona las colisiones del jugador con diferentes objetos, como enemigos y rampas.

```
csharp
Copiar código
private void OnCollisionEnter2D(Collision2D collision)
{
    if (collision.gameObject.tag == "Enemy")
    {
        ActivarImpacto();
    }

    if (collision.gameObject.tag == "Rampa")
    {
        ActivarSubida();
    }
    if (collision.gameObject.tag == "DeathZone")
    {
        Frenar = true;
    }
}
```

FixedUpdate()

Aplica el movimiento y otras acciones físicas del jugador.

```
csharp
Copiar código
private void FixedUpdate()
{
```

```
    if (Frenar)
    {
        desplazamientoDer = Mathf.Lerp(desplazamientoDer, 0f,
suavizadoDesplazamiento * Time.fixedDeltaTime);
        DesplazarDerecha = false;
        animator.SetBool("isStoping", true);

        desplazamientoIzq = Mathf.Lerp(desplazamientoIzq, 0f,
suavizadoDesplazamiento * Time.fixedDeltaTime);
        DesplazarIzquierda = false;
        animator.SetBool("isStoping", true);

        Desplazar(desplazamientoDer * Time.fixedDeltaTime, Salto);
    }

    if (DesplazarDerecha)
    {
        animator.SetBool("isStoping", false);
        Desplazar(desplazamientoDer * Time.fixedDeltaTime, Salto);
        instructions1.SetActive(false);
    }

    if (DesplazarIzquierda)
    {
        animator.SetBool("isStoping", false);
        Desplazar(desplazamientoIzq * Time.fixedDeltaTime, Salto);
        instructions1.SetActive(false);
    }

    enSuelo = Physics2D.OverlapBox(DetectorSuelo.position,
tamañoDetector, 0f, esSuelo);
    Salto = false;
}
```

Desplazar(float despl, bool saltar)

Maneja el desplazamiento y el salto del jugador.

```
csharp
Copiar código
private void Desplazar(float despl, bool saltar)
{
    Vector3 velocidadObjetivo = new Vector2(despl, rb2D.velocity.y);
    rb2D.velocity = Vector3.SmoothDamp(rb2D.velocity, velocidadObjetivo,
ref velocidad, suavizadoDesplazamiento);

    if (despl > 0 && !MirarDer)
    {
        Girar();
        animator.enabled = true;
    }
    else if (despl < 0 && MirarDer)
    {
        Girar();
        animator.enabled = true;
    }
```

```
    if (enSuelo && saltar)
    {
        rb2D.AddForce(new Vector2(0f, fuerzaSalto));
        animator.SetBool("isJumpingR", true);
        sonidoCode.Jump();
        enSuelo = false;
    }
    else if (enSuelo && animator.GetBool("isJumpingR"))
    {
        Invoke("JumpAnimation", delayAnim);
    }

    if (Mathf.Abs(despl) < velocidadMax)
    {
        desplazamientoDer += incrementoVelocidad * Time.fixedDeltaTime;
        desplazamientoIzq -= incrementoVelocidad * Time.fixedDeltaTime;
    }
}
```

ActivarAtaque()

Inicia el ataque del jugador.

```
csharp
Copiar código
public void ActivarAtaque()
{
    StartCoroutine(AtaqueCorutina());
}
```

AtaqueCorutina()

Corutina que maneja la lógica de ataque.

```
csharp
Copiar código
IEnumerator AtaqueCorutina()
{
    atacando = true;
    Collider2D[] enemigos =
Physics2D.OverlapCircleAll(transform.position, radioAtaque);
    foreach (Collider2D enemy in enemigos)
    {
        if (enemy.CompareTag("Enemy"))
        {
            enemy.GetComponent<Atacar>().SerGolpeado();
        }
    }

    animator.SetBool("isAttacking", true);
    if (audioSource != null)
        audioSource.Play();

    yield return new WaitForSeconds(0.9f);
```

```
    animator.SetBool("isAttacking", false);
    atacando = false;
}
```

ActivarImpacto()

Inicia la animación de impacto.

```
csharp
Copiar código
public void ActivarImpacto()
{
    StartCoroutine(ImpactoCorutina());
}
```

ImpactoCorutina()

Corutina que maneja la lógica de impacto.

```
csharp
Copiar código
IEnumerator ImpactoCorutina()
{
    animator.SetBool("isCrashing", true);
    yield return new WaitForSeconds(0.5f);
    animator.SetBool("isCrashing", false);
}
```

ActivarSubida()

Inicia la animación de subida en rampas.

```
csharp
Copiar código
public void ActivarSubida()
{
    StartCoroutine(SubidaCorutina());
}
```

SubidaCorutina()

Corutina que maneja la lógica de subida en rampas.

```
csharp
Copiar código
IEnumerator SubidaCorutina()
{
    animator.SetBool("isGoUp", true);
    yield return new WaitForSeconds(0.6f);
    animator.SetBool("isGoUp", false);
}
```

JumpAnimation()

Desactiva la animación de salto.

```
csharp
Copiar código
private void JumpAnimation()
{
    animator.SetBool("isJumpingR", false);
}
```

Girar()

Invierte la dirección del jugador.

```
csharp
Copiar código
private void Girar()
{
    MirarDer = !MirarDer;
    Vector3 escala = transform.localScale;
    escala.x *= -1;
    transform.localScale = escala;
}
```

OnDrawGizmos()

Dibuja gizmos en el editor para visualización.

```
csharp
Copiar código
private void OnDrawGizmos()
{
    Gizmos.color = Color.yellow;
    Gizmos.DrawWireCube(DetectorSuelo.position, tamañoDetector);
    Gizmos.DrawWireSphere(transform.position, radioAtaque);
}
```

Notas Adicionales

- Asegúrate de asignar los componentes `Rigidbody2D`, `Animator`, `AudioSource` y `PlayerSoundEffects` en el Inspector de Unity.
- Configura los valores de las variables públicas en el Inspector según las necesidades de tu juego.
- Las corutinas se utilizan para manejar animaciones y efectos temporizados como ataques e impactos.
- La detección de suelo y colisiones se realiza mediante `Physics2D`