using System;

using UnityEngine;

namespace UnityStandardAssets.\_2D

{

public class PlatformerCharacter2D : MonoBehaviour

{

[SerializeField] private float m\_MaxSpeed = 10f; // The fastest the player can travel in the x axis.

[SerializeField] private float m\_JumpForce = 400f; // Amount of force added when the player jumps.

[Range(0, 1)] [SerializeField] private float m\_CrouchSpeed = .36f; // Amount of maxSpeed applied to crouching movement. 1 = 100%

[SerializeField] private bool m\_AirControl = false; // Whether or not a player can steer while jumping;

[SerializeField] private LayerMask m\_WhatIsGround; // A mask determining what is ground to the character

private Transform m\_GroundCheck; // A position marking where to check if the player is grounded.

const float k\_GroundedRadius = .2f; // Radius of the overlap circle to determine if grounded

private bool m\_Grounded; // Whether or not the player is grounded.

private Transform m\_CeilingCheck; // A position marking where to check for ceilings

const float k\_CeilingRadius = .01f; // Radius of the overlap circle to determine if the player can stand up

private Animator m\_Anim; // Reference to the player's animator component.

private Rigidbody2D m\_Rigidbody2D;

private bool m\_FacingRight = true; // For determining which way the player is currently facing.

private void Awake()

{

// Setting up references.

m\_GroundCheck = transform.Find("GroundCheck");

m\_CeilingCheck = transform.Find("CeilingCheck");

m\_Anim = GetComponent<Animator>();

m\_Rigidbody2D = GetComponent<Rigidbody2D>();

}

private void FixedUpdate()

{

m\_Grounded = false;

// The player is grounded if a circlecast to the groundcheck position hits anything designated as ground

// This can be done using layers instead but Sample Assets will not overwrite your project settings.

Collider2D[] colliders = Physics2D.OverlapCircleAll(m\_GroundCheck.position, k\_GroundedRadius, m\_WhatIsGround);

for (int i = 0; i < colliders.Length; i++)

{

if (colliders[i].gameObject != gameObject)

m\_Grounded = true;

}

m\_Anim.SetBool("Ground", m\_Grounded);

// Set the vertical animation

m\_Anim.SetFloat("vSpeed", m\_Rigidbody2D.velocity.y);

}

public void Move(float move, bool crouch, bool jump)

{

// If crouching, check to see if the character can stand up

if (!crouch && m\_Anim.GetBool("Crouch"))

{

// If the character has a ceiling preventing them from standing up, keep them crouching

if (Physics2D.OverlapCircle(m\_CeilingCheck.position, k\_CeilingRadius, m\_WhatIsGround))

{

crouch = true;

}

}

// Set whether or not the character is crouching in the animator

m\_Anim.SetBool("Crouch", crouch);

//only control the player if grounded or airControl is turned on

if (m\_Grounded || m\_AirControl)

{

// Reduce the speed if crouching by the crouchSpeed multiplier

move = (crouch ? move\*m\_CrouchSpeed : move);

// The Speed animator parameter is set to the absolute value of the horizontal input.

m\_Anim.SetFloat("Speed", Mathf.Abs(move));

// Move the character

m\_Rigidbody2D.velocity = new Vector2(move\*m\_MaxSpeed, m\_Rigidbody2D.velocity.y);

// If the input is moving the player right and the player is facing left...

if (move > 0 && !m\_FacingRight)

{

// ... flip the player.

Flip();

}

// Otherwise if the input is moving the player left and the player is facing right...

else if (move < 0 && m\_FacingRight)

{

// ... flip the player.

Flip();

}

}

// If the player should jump...

if (m\_Grounded && jump && m\_Anim.GetBool("Ground"))

{

// Add a vertical force to the player.

m\_Grounded = false;

m\_Anim.SetBool("Ground", false);

m\_Rigidbody2D.AddForce(new Vector2(0f, m\_JumpForce));

}

}

private void Flip()

{

// Switch the way the player is labelled as facing.

m\_FacingRight = !m\_FacingRight;

// Multiply the player's x local scale by -1.

Vector3 theScale = transform.localScale;

theScale.x \*= -1;

transform.localScale = theScale;

}

}

}