using System;

using UnityEngine;

using UnityStandardAssets.CrossPlatformInput;

namespace UnityStandardAssets.Characters.FirstPerson

{

[RequireComponent(typeof (Rigidbody))]

[RequireComponent(typeof (CapsuleCollider))]

public class RigidbodyFirstPersonController : MonoBehaviour

{

[Serializable]

public class MovementSettings

{

public float ForwardSpeed = 8.0f; // Speed when walking forward

public float BackwardSpeed = 4.0f; // Speed when walking backwards

public float StrafeSpeed = 4.0f; // Speed when walking sideways

public float RunMultiplier = 2.0f; // Speed when sprinting

public KeyCode RunKey = KeyCode.LeftShift;

public float JumpForce = 30f;

public AnimationCurve SlopeCurveModifier = new AnimationCurve(new Keyframe(-90.0f, 1.0f), new Keyframe(0.0f, 1.0f), new Keyframe(90.0f, 0.0f));

[HideInInspector] public float CurrentTargetSpeed = 8f;

#if !MOBILE\_INPUT

private bool m\_Running;

#endif

public void UpdateDesiredTargetSpeed(Vector2 input)

{

if (input == Vector2.zero) return;

if (input.x > 0 || input.x < 0)

{

//strafe

CurrentTargetSpeed = StrafeSpeed;

}

if (input.y < 0)

{

//backwards

CurrentTargetSpeed = BackwardSpeed;

}

if (input.y > 0)

{

//forwards

//handled last as if strafing and moving forward at the same time forwards speed should take precedence

CurrentTargetSpeed = ForwardSpeed;

}

#if !MOBILE\_INPUT

if (Input.GetKey(RunKey))

{

CurrentTargetSpeed \*= RunMultiplier;

m\_Running = true;

}

else

{

m\_Running = false;

}

#endif

}

#if !MOBILE\_INPUT

public bool Running

{

get { return m\_Running; }

}

#endif

}

[Serializable]

public class AdvancedSettings

{

public float groundCheckDistance = 0.01f; // distance for checking if the controller is grounded ( 0.01f seems to work best for this )

public float stickToGroundHelperDistance = 0.5f; // stops the character

public float slowDownRate = 20f; // rate at which the controller comes to a stop when there is no input

public bool airControl; // can the user control the direction that is being moved in the air

}

public Camera cam;

public MovementSettings movementSettings = new MovementSettings();

public MouseLook mouseLook = new MouseLook();

public AdvancedSettings advancedSettings = new AdvancedSettings();

private Rigidbody m\_RigidBody;

private CapsuleCollider m\_Capsule;

private float m\_YRotation;

private Vector3 m\_GroundContactNormal;

private bool m\_Jump, m\_PreviouslyGrounded, m\_Jumping, m\_IsGrounded;

public Vector3 Velocity

{

get { return m\_RigidBody.velocity; }

}

public bool Grounded

{

get { return m\_IsGrounded; }

}

public bool Jumping

{

get { return m\_Jumping; }

}

public bool Running

{

get

{

#if !MOBILE\_INPUT

return movementSettings.Running;

#else

return false;

#endif

}

}

private void Start()

{

m\_RigidBody = GetComponent<Rigidbody>();

m\_Capsule = GetComponent<CapsuleCollider>();

mouseLook.Init (transform, cam.transform);

}

private void Update()

{

RotateView();

if (CrossPlatformInputManager.GetButtonDown("Jump") && !m\_Jump)

{

m\_Jump = true;

}

}

private void FixedUpdate()

{

GroundCheck();

Vector2 input = GetInput();

if ((Mathf.Abs(input.x) > float.Epsilon || Mathf.Abs(input.y) > float.Epsilon) && (advancedSettings.airControl || m\_IsGrounded))

{

// always move along the camera forward as it is the direction that it being aimed at

Vector3 desiredMove = cam.transform.forward\*input.y + cam.transform.right\*input.x;

desiredMove = Vector3.ProjectOnPlane(desiredMove, m\_GroundContactNormal).normalized;

desiredMove.x = desiredMove.x\*movementSettings.CurrentTargetSpeed;

desiredMove.z = desiredMove.z\*movementSettings.CurrentTargetSpeed;

desiredMove.y = desiredMove.y\*movementSettings.CurrentTargetSpeed;

if (m\_RigidBody.velocity.sqrMagnitude <

(movementSettings.CurrentTargetSpeed\*movementSettings.CurrentTargetSpeed))

{

m\_RigidBody.AddForce(desiredMove\*SlopeMultiplier(), ForceMode.Impulse);

}

}

if (m\_IsGrounded)

{

m\_RigidBody.drag = 5f;

if (m\_Jump)

{

m\_RigidBody.drag = 0f;

m\_RigidBody.velocity = new Vector3(m\_RigidBody.velocity.x, 0f, m\_RigidBody.velocity.z);

m\_RigidBody.AddForce(new Vector3(0f, movementSettings.JumpForce, 0f), ForceMode.Impulse);

m\_Jumping = true;

}

if (!m\_Jumping && Mathf.Abs(input.x) < float.Epsilon && Mathf.Abs(input.y) < float.Epsilon && m\_RigidBody.velocity.magnitude < 1f)

{

m\_RigidBody.Sleep();

}

}

else

{

m\_RigidBody.drag = 0f;

if (m\_PreviouslyGrounded && !m\_Jumping)

{

StickToGroundHelper();

}

}

m\_Jump = false;

}

private float SlopeMultiplier()

{

float angle = Vector3.Angle(m\_GroundContactNormal, Vector3.up);

return movementSettings.SlopeCurveModifier.Evaluate(angle);

}

private void StickToGroundHelper()

{

RaycastHit hitInfo;

if (Physics.SphereCast(transform.position, m\_Capsule.radius, Vector3.down, out hitInfo,

((m\_Capsule.height/2f) - m\_Capsule.radius) +

advancedSettings.stickToGroundHelperDistance))

{

if (Mathf.Abs(Vector3.Angle(hitInfo.normal, Vector3.up)) < 85f)

{

m\_RigidBody.velocity = Vector3.ProjectOnPlane(m\_RigidBody.velocity, hitInfo.normal);

}

}

}

private Vector2 GetInput()

{

Vector2 input = new Vector2

{

x = CrossPlatformInputManager.GetAxis("Horizontal"),

y = CrossPlatformInputManager.GetAxis("Vertical")

};

movementSettings.UpdateDesiredTargetSpeed(input);

return input;

}

private void RotateView()

{

//avoids the mouse looking if the game is effectively paused

if (Mathf.Abs(Time.timeScale) < float.Epsilon) return;

// get the rotation before it's changed

float oldYRotation = transform.eulerAngles.y;

mouseLook.LookRotation (transform, cam.transform);

if (m\_IsGrounded || advancedSettings.airControl)

{

// Rotate the rigidbody velocity to match the new direction that the character is looking

Quaternion velRotation = Quaternion.AngleAxis(transform.eulerAngles.y - oldYRotation, Vector3.up);

m\_RigidBody.velocity = velRotation\*m\_RigidBody.velocity;

}

}

/// sphere cast down just beyond the bottom of the capsule to see if the capsule is colliding round the bottom

private void GroundCheck()

{

m\_PreviouslyGrounded = m\_IsGrounded;

RaycastHit hitInfo;

if (Physics.SphereCast(transform.position, m\_Capsule.radius, Vector3.down, out hitInfo,

((m\_Capsule.height/2f) - m\_Capsule.radius) + advancedSettings.groundCheckDistance))

{

m\_IsGrounded = true;

m\_GroundContactNormal = hitInfo.normal;

}

else

{

m\_IsGrounded = false;

m\_GroundContactNormal = Vector3.up;

}

if (!m\_PreviouslyGrounded && m\_IsGrounded && m\_Jumping)

{

m\_Jumping = false;

}

}

}

}