using UnityEngine;

namespace UnityStandardAssets.Characters.ThirdPerson

{

[RequireComponent(typeof(Rigidbody))]

[RequireComponent(typeof(CapsuleCollider))]

[RequireComponent(typeof(Animator))]

public class ThirdPersonCharacter : MonoBehaviour

{

[SerializeField] float m\_MovingTurnSpeed = 360;

[SerializeField] float m\_StationaryTurnSpeed = 180;

[SerializeField] float m\_JumpPower = 12f;

[Range(1f, 4f)][SerializeField] float m\_GravityMultiplier = 2f;

[SerializeField] float m\_RunCycleLegOffset = 0.2f; //specific to the character in sample assets, will need to be modified to work with others

[SerializeField] float m\_MoveSpeedMultiplier = 1f;

[SerializeField] float m\_AnimSpeedMultiplier = 1f;

[SerializeField] float m\_GroundCheckDistance = 0.1f;

Rigidbody m\_Rigidbody;

Animator m\_Animator;

bool m\_IsGrounded;

float m\_OrigGroundCheckDistance;

const float k\_Half = 0.5f;

float m\_TurnAmount;

float m\_ForwardAmount;

Vector3 m\_GroundNormal;

float m\_CapsuleHeight;

Vector3 m\_CapsuleCenter;

CapsuleCollider m\_Capsule;

bool m\_Crouching;

void Start()

{

m\_Animator = GetComponent<Animator>();

m\_Rigidbody = GetComponent<Rigidbody>();

m\_Capsule = GetComponent<CapsuleCollider>();

m\_CapsuleHeight = m\_Capsule.height;

m\_CapsuleCenter = m\_Capsule.center;

m\_Rigidbody.constraints = RigidbodyConstraints.FreezeRotationX | RigidbodyConstraints.FreezeRotationY | RigidbodyConstraints.FreezeRotationZ;

m\_OrigGroundCheckDistance = m\_GroundCheckDistance;

}

public void Move(Vector3 move, bool crouch, bool jump)

{

// convert the world relative moveInput vector into a local-relative

// turn amount and forward amount required to head in the desired

// direction.

if (move.magnitude > 1f) move.Normalize();

move = transform.InverseTransformDirection(move);

CheckGroundStatus();

move = Vector3.ProjectOnPlane(move, m\_GroundNormal);

m\_TurnAmount = Mathf.Atan2(move.x, move.z);

m\_ForwardAmount = move.z;

ApplyExtraTurnRotation();

// control and velocity handling is different when grounded and airborne:

if (m\_IsGrounded)

{

HandleGroundedMovement(crouch, jump);

}

else

{

HandleAirborneMovement();

}

ScaleCapsuleForCrouching(crouch);

PreventStandingInLowHeadroom();

// send input and other state parameters to the animator

UpdateAnimator(move);

}

void ScaleCapsuleForCrouching(bool crouch)

{

if (m\_IsGrounded && crouch)

{

if (m\_Crouching) return;

m\_Capsule.height = m\_Capsule.height / 2f;

m\_Capsule.center = m\_Capsule.center / 2f;

m\_Crouching = true;

}

else

{

Ray crouchRay = new Ray(m\_Rigidbody.position + Vector3.up \* m\_Capsule.radius \* k\_Half, Vector3.up);

float crouchRayLength = m\_CapsuleHeight - m\_Capsule.radius \* k\_Half;

if (Physics.SphereCast(crouchRay, m\_Capsule.radius \* k\_Half, crouchRayLength))

{

m\_Crouching = true;

return;

}

m\_Capsule.height = m\_CapsuleHeight;

m\_Capsule.center = m\_CapsuleCenter;

m\_Crouching = false;

}

}

void PreventStandingInLowHeadroom()

{

// prevent standing up in crouch-only zones

if (!m\_Crouching)

{

Ray crouchRay = new Ray(m\_Rigidbody.position + Vector3.up \* m\_Capsule.radius \* k\_Half, Vector3.up);

float crouchRayLength = m\_CapsuleHeight - m\_Capsule.radius \* k\_Half;

if (Physics.SphereCast(crouchRay, m\_Capsule.radius \* k\_Half, crouchRayLength))

{

m\_Crouching = true;

}

}

}

void UpdateAnimator(Vector3 move)

{

// update the animator parameters

m\_Animator.SetFloat("Forward", m\_ForwardAmount, 0.1f, Time.deltaTime);

m\_Animator.SetFloat("Turn", m\_TurnAmount, 0.1f, Time.deltaTime);

m\_Animator.SetBool("Crouch", m\_Crouching);

m\_Animator.SetBool("OnGround", m\_IsGrounded);

if (!m\_IsGrounded)

{

m\_Animator.SetFloat("Jump", m\_Rigidbody.velocity.y);

}

// calculate which leg is behind, so as to leave that leg trailing in the jump animation

// (This code is reliant on the specific run cycle offset in our animations,

// and assumes one leg passes the other at the normalized clip times of 0.0 and 0.5)

float runCycle =

Mathf.Repeat(

m\_Animator.GetCurrentAnimatorStateInfo(0).normalizedTime + m\_RunCycleLegOffset, 1);

float jumpLeg = (runCycle < k\_Half ? 1 : -1) \* m\_ForwardAmount;

if (m\_IsGrounded)

{

m\_Animator.SetFloat("JumpLeg", jumpLeg);

}

// the anim speed multiplier allows the overall speed of walking/running to be tweaked in the inspector,

// which affects the movement speed because of the root motion.

if (m\_IsGrounded && move.magnitude > 0)

{

m\_Animator.speed = m\_AnimSpeedMultiplier;

}

else

{

// don't use that while airborne

m\_Animator.speed = 1;

}

}

void HandleAirborneMovement()

{

// apply extra gravity from multiplier:

Vector3 extraGravityForce = (Physics.gravity \* m\_GravityMultiplier) - Physics.gravity;

m\_Rigidbody.AddForce(extraGravityForce);

m\_GroundCheckDistance = m\_Rigidbody.velocity.y < 0 ? m\_OrigGroundCheckDistance : 0.01f;

}

void HandleGroundedMovement(bool crouch, bool jump)

{

// check whether conditions are right to allow a jump:

if (jump && !crouch && m\_Animator.GetCurrentAnimatorStateInfo(0).IsName("Grounded"))

{

// jump!

m\_Rigidbody.velocity = new Vector3(m\_Rigidbody.velocity.x, m\_JumpPower, m\_Rigidbody.velocity.z);

m\_IsGrounded = false;

m\_Animator.applyRootMotion = false;

m\_GroundCheckDistance = 0.1f;

}

}

void ApplyExtraTurnRotation()

{

// help the character turn faster (this is in addition to root rotation in the animation)

float turnSpeed = Mathf.Lerp(m\_StationaryTurnSpeed, m\_MovingTurnSpeed, m\_ForwardAmount);

transform.Rotate(0, m\_TurnAmount \* turnSpeed \* Time.deltaTime, 0);

}

public void OnAnimatorMove()

{

// we implement this function to override the default root motion.

// this allows us to modify the positional speed before it's applied.

if (m\_IsGrounded && Time.deltaTime > 0)

{

Vector3 v = (m\_Animator.deltaPosition \* m\_MoveSpeedMultiplier) / Time.deltaTime;

// we preserve the existing y part of the current velocity.

v.y = m\_Rigidbody.velocity.y;

m\_Rigidbody.velocity = v;

}

}

void CheckGroundStatus()

{

RaycastHit hitInfo;

#if UNITY\_EDITOR

// helper to visualise the ground check ray in the scene view

Debug.DrawLine(transform.position + (Vector3.up \* 0.1f), transform.position + (Vector3.up \* 0.1f) + (Vector3.down \* m\_GroundCheckDistance));

#endif

// 0.1f is a small offset to start the ray from inside the character

// it is also good to note that the transform position in the sample assets is at the base of the character

if (Physics.Raycast(transform.position + (Vector3.up \* 0.1f), Vector3.down, out hitInfo, m\_GroundCheckDistance))

{

m\_GroundNormal = hitInfo.normal;

m\_IsGrounded = true;

m\_Animator.applyRootMotion = true;

}

else

{

m\_IsGrounded = false;

m\_GroundNormal = Vector3.up;

m\_Animator.applyRootMotion = false;

}

}

}

}