using System;

using UnityEngine;

using UnityStandardAssets.CrossPlatformInput;

namespace UnityStandardAssets.Cameras

{

public class FreeLookCam : PivotBasedCameraRig

{

// This script is designed to be placed on the root object of a camera rig,

// comprising 3 gameobjects, each parented to the next:

// Camera Rig

// Pivot

// Camera

[SerializeField] private float m\_MoveSpeed = 1f; // How fast the rig will move to keep up with the target's position.

[Range(0f, 10f)] [SerializeField] private float m\_TurnSpeed = 1.5f; // How fast the rig will rotate from user input.

[SerializeField] private float m\_TurnSmoothing = 0.1f; // How much smoothing to apply to the turn input, to reduce mouse-turn jerkiness

[SerializeField] private float m\_TiltMax = 75f; // The maximum value of the x axis rotation of the pivot.

[SerializeField] private float m\_TiltMin = 45f; // The minimum value of the x axis rotation of the pivot.

[SerializeField] private bool m\_LockCursor = false; // Whether the cursor should be hidden and locked.

[SerializeField] private bool m\_VerticalAutoReturn = false; // set wether or not the vertical axis should auto return

private float m\_LookAngle; // The rig's y axis rotation.

private float m\_TiltAngle; // The pivot's x axis rotation.

private const float k\_LookDistance = 100f; // How far in front of the pivot the character's look target is.

private Vector3 m\_PivotEulers;

private Quaternion m\_PivotTargetRot;

private Quaternion m\_TransformTargetRot;

protected override void Awake()

{

base.Awake();

// Lock or unlock the cursor.

Cursor.lockState = m\_LockCursor ? CursorLockMode.Locked : CursorLockMode.None;

Cursor.visible = !m\_LockCursor;

m\_PivotEulers = m\_Pivot.rotation.eulerAngles;

m\_PivotTargetRot = m\_Pivot.transform.localRotation;

m\_TransformTargetRot = transform.localRotation;

}

protected void Update()

{

HandleRotationMovement();

if (m\_LockCursor && Input.GetMouseButtonUp(0))

{

Cursor.lockState = m\_LockCursor ? CursorLockMode.Locked : CursorLockMode.None;

Cursor.visible = !m\_LockCursor;

}

}

private void OnDisable()

{

Cursor.lockState = CursorLockMode.None;

Cursor.visible = true;

}

protected override void FollowTarget(float deltaTime)

{

if (m\_Target == null) return;

// Move the rig towards target position.

transform.position = Vector3.Lerp(transform.position, m\_Target.position, deltaTime\*m\_MoveSpeed);

}

private void HandleRotationMovement()

{

if(Time.timeScale < float.Epsilon)

return;

// Read the user input

var x = CrossPlatformInputManager.GetAxis("Mouse X");

var y = CrossPlatformInputManager.GetAxis("Mouse Y");

// Adjust the look angle by an amount proportional to the turn speed and horizontal input.

m\_LookAngle += x\*m\_TurnSpeed;

// Rotate the rig (the root object) around Y axis only:

m\_TransformTargetRot = Quaternion.Euler(0f, m\_LookAngle, 0f);

if (m\_VerticalAutoReturn)

{

// For tilt input, we need to behave differently depending on whether we're using mouse or touch input:

// on mobile, vertical input is directly mapped to tilt value, so it springs back automatically when the look input is released

// we have to test whether above or below zero because we want to auto-return to zero even if min and max are not symmetrical.

m\_TiltAngle = y > 0 ? Mathf.Lerp(0, -m\_TiltMin, y) : Mathf.Lerp(0, m\_TiltMax, -y);

}

else

{

// on platforms with a mouse, we adjust the current angle based on Y mouse input and turn speed

m\_TiltAngle -= y\*m\_TurnSpeed;

// and make sure the new value is within the tilt range

m\_TiltAngle = Mathf.Clamp(m\_TiltAngle, -m\_TiltMin, m\_TiltMax);

}

// Tilt input around X is applied to the pivot (the child of this object)

m\_PivotTargetRot = Quaternion.Euler(m\_TiltAngle, m\_PivotEulers.y , m\_PivotEulers.z);

if (m\_TurnSmoothing > 0)

{

m\_Pivot.localRotation = Quaternion.Slerp(m\_Pivot.localRotation, m\_PivotTargetRot, m\_TurnSmoothing \* Time.deltaTime);

transform.localRotation = Quaternion.Slerp(transform.localRotation, m\_TransformTargetRot, m\_TurnSmoothing \* Time.deltaTime);

}

else

{

m\_Pivot.localRotation = m\_PivotTargetRot;

transform.localRotation = m\_TransformTargetRot;

}

}

}

}