using System;

using UnityEngine;

using UnityStandardAssets.CrossPlatformInput;

namespace UnityStandardAssets.Utility

{

public class SimpleMouseRotator : MonoBehaviour

{

// A mouselook behaviour with constraints which operate relative to

// this gameobject's initial rotation.

// Only rotates around local X and Y.

// Works in local coordinates, so if this object is parented

// to another moving gameobject, its local constraints will

// operate correctly

// (Think: looking out the side window of a car, or a gun turret

// on a moving spaceship with a limited angular range)

// to have no constraints on an axis, set the rotationRange to 360 or greater.

public Vector2 rotationRange = new Vector3(70, 70);

public float rotationSpeed = 10;

public float dampingTime = 0.2f;

public bool autoZeroVerticalOnMobile = true;

public bool autoZeroHorizontalOnMobile = false;

public bool relative = true;

private Vector3 m\_TargetAngles;

private Vector3 m\_FollowAngles;

private Vector3 m\_FollowVelocity;

private Quaternion m\_OriginalRotation;

private void Start()

{

m\_OriginalRotation = transform.localRotation;

}

private void Update()

{

// we make initial calculations from the original local rotation

transform.localRotation = m\_OriginalRotation;

// read input from mouse or mobile controls

float inputH;

float inputV;

if (relative)

{

inputH = CrossPlatformInputManager.GetAxis("Mouse X");

inputV = CrossPlatformInputManager.GetAxis("Mouse Y");

// wrap values to avoid springing quickly the wrong way from positive to negative

if (m\_TargetAngles.y > 180)

{

m\_TargetAngles.y -= 360;

m\_FollowAngles.y -= 360;

}

if (m\_TargetAngles.x > 180)

{

m\_TargetAngles.x -= 360;

m\_FollowAngles.x -= 360;

}

if (m\_TargetAngles.y < -180)

{

m\_TargetAngles.y += 360;

m\_FollowAngles.y += 360;

}

if (m\_TargetAngles.x < -180)

{

m\_TargetAngles.x += 360;

m\_FollowAngles.x += 360;

}

#if MOBILE\_INPUT

// on mobile, sometimes we want input mapped directly to tilt value,

// so it springs back automatically when the look input is released.

if (autoZeroHorizontalOnMobile) {

m\_TargetAngles.y = Mathf.Lerp (-rotationRange.y \* 0.5f, rotationRange.y \* 0.5f, inputH \* .5f + .5f);

} else {

m\_TargetAngles.y += inputH \* rotationSpeed;

}

if (autoZeroVerticalOnMobile) {

m\_TargetAngles.x = Mathf.Lerp (-rotationRange.x \* 0.5f, rotationRange.x \* 0.5f, inputV \* .5f + .5f);

} else {

m\_TargetAngles.x += inputV \* rotationSpeed;

}

#else

// with mouse input, we have direct control with no springback required.

m\_TargetAngles.y += inputH\*rotationSpeed;

m\_TargetAngles.x += inputV\*rotationSpeed;

#endif

// clamp values to allowed range

m\_TargetAngles.y = Mathf.Clamp(m\_TargetAngles.y, -rotationRange.y\*0.5f, rotationRange.y\*0.5f);

m\_TargetAngles.x = Mathf.Clamp(m\_TargetAngles.x, -rotationRange.x\*0.5f, rotationRange.x\*0.5f);

}

else

{

inputH = Input.mousePosition.x;

inputV = Input.mousePosition.y;

// set values to allowed range

m\_TargetAngles.y = Mathf.Lerp(-rotationRange.y\*0.5f, rotationRange.y\*0.5f, inputH/Screen.width);

m\_TargetAngles.x = Mathf.Lerp(-rotationRange.x\*0.5f, rotationRange.x\*0.5f, inputV/Screen.height);

}

// smoothly interpolate current values to target angles

m\_FollowAngles = Vector3.SmoothDamp(m\_FollowAngles, m\_TargetAngles, ref m\_FollowVelocity, dampingTime);

// update the actual gameobject's rotation

transform.localRotation = m\_OriginalRotation\*Quaternion.Euler(-m\_FollowAngles.x, m\_FollowAngles.y, 0);

}

}

}