using System;

using UnityEngine;

#if UNITY\_EDITOR

#endif

namespace UnityStandardAssets.Cameras

{

[ExecuteInEditMode]

public class AutoCam : PivotBasedCameraRig

{

[SerializeField] private float m\_MoveSpeed = 3; // How fast the rig will move to keep up with target's position

[SerializeField] private float m\_TurnSpeed = 1; // How fast the rig will turn to keep up with target's rotation

[SerializeField] private float m\_RollSpeed = 0.2f;// How fast the rig will roll (around Z axis) to match target's roll.

[SerializeField] private bool m\_FollowVelocity = false;// Whether the rig will rotate in the direction of the target's velocity.

[SerializeField] private bool m\_FollowTilt = true; // Whether the rig will tilt (around X axis) with the target.

[SerializeField] private float m\_SpinTurnLimit = 90;// The threshold beyond which the camera stops following the target's rotation. (used in situations where a car spins out, for example)

[SerializeField] private float m\_TargetVelocityLowerLimit = 4f;// the minimum velocity above which the camera turns towards the object's velocity. Below this we use the object's forward direction.

[SerializeField] private float m\_SmoothTurnTime = 0.2f; // the smoothing for the camera's rotation

private float m\_LastFlatAngle; // The relative angle of the target and the rig from the previous frame.

private float m\_CurrentTurnAmount; // How much to turn the camera

private float m\_TurnSpeedVelocityChange; // The change in the turn speed velocity

private Vector3 m\_RollUp = Vector3.up;// The roll of the camera around the z axis ( generally this will always just be up )

protected override void FollowTarget(float deltaTime)

{

// if no target, or no time passed then we quit early, as there is nothing to do

if (!(deltaTime > 0) || m\_Target == null)

{

return;

}

// initialise some vars, we'll be modifying these in a moment

var targetForward = m\_Target.forward;

var targetUp = m\_Target.up;

if (m\_FollowVelocity && Application.isPlaying)

{

// in follow velocity mode, the camera's rotation is aligned towards the object's velocity direction

// but only if the object is traveling faster than a given threshold.

if (targetRigidbody.velocity.magnitude > m\_TargetVelocityLowerLimit)

{

// velocity is high enough, so we'll use the target's velocty

targetForward = targetRigidbody.velocity.normalized;

targetUp = Vector3.up;

}

else

{

targetUp = Vector3.up;

}

m\_CurrentTurnAmount = Mathf.SmoothDamp(m\_CurrentTurnAmount, 1, ref m\_TurnSpeedVelocityChange, m\_SmoothTurnTime);

}

else

{

// we're in 'follow rotation' mode, where the camera rig's rotation follows the object's rotation.

// This section allows the camera to stop following the target's rotation when the target is spinning too fast.

// eg when a car has been knocked into a spin. The camera will resume following the rotation

// of the target when the target's angular velocity slows below the threshold.

var currentFlatAngle = Mathf.Atan2(targetForward.x, targetForward.z)\*Mathf.Rad2Deg;

if (m\_SpinTurnLimit > 0)

{

var targetSpinSpeed = Mathf.Abs(Mathf.DeltaAngle(m\_LastFlatAngle, currentFlatAngle))/deltaTime;

var desiredTurnAmount = Mathf.InverseLerp(m\_SpinTurnLimit, m\_SpinTurnLimit\*0.75f, targetSpinSpeed);

var turnReactSpeed = (m\_CurrentTurnAmount > desiredTurnAmount ? .1f : 1f);

if (Application.isPlaying)

{

m\_CurrentTurnAmount = Mathf.SmoothDamp(m\_CurrentTurnAmount, desiredTurnAmount,

ref m\_TurnSpeedVelocityChange, turnReactSpeed);

}

else

{

// for editor mode, smoothdamp won't work because it uses deltaTime internally

m\_CurrentTurnAmount = desiredTurnAmount;

}

}

else

{

m\_CurrentTurnAmount = 1;

}

m\_LastFlatAngle = currentFlatAngle;

}

// camera position moves towards target position:

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

// camera's rotation is split into two parts, which can have independend speed settings:

// rotating towards the target's forward direction (which encompasses its 'yaw' and 'pitch')

if (!m\_FollowTilt)

{

targetForward.y = 0;

if (targetForward.sqrMagnitude < float.Epsilon)

{

targetForward = transform.forward;

}

}

var rollRotation = Quaternion.LookRotation(targetForward, m\_RollUp);

// and aligning with the target object's up direction (i.e. its 'roll')

m\_RollUp = m\_RollSpeed > 0 ? Vector3.Slerp(m\_RollUp, targetUp, m\_RollSpeed\*deltaTime) : Vector3.up;

transform.rotation = Quaternion.Lerp(transform.rotation, rollRotation, m\_TurnSpeed\*m\_CurrentTurnAmount\*deltaTime);

}

}

}