using System;

using UnityEngine;

namespace UnityStandardAssets.Cameras

{

public class LookatTarget : AbstractTargetFollower

{

// A simple script to make one object look at another,

// but with optional 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 greater than 360.

[SerializeField] private Vector2 m\_RotationRange;

[SerializeField] private float m\_FollowSpeed = 1;

private Vector3 m\_FollowAngles;

private Quaternion m\_OriginalRotation;

protected Vector3 m\_FollowVelocity;

// Use this for initialization

protected override void Start()

{

base.Start();

m\_OriginalRotation = transform.localRotation;

}

protected override void FollowTarget(float deltaTime)

{

// we make initial calculations from the original local rotation

transform.localRotation = m\_OriginalRotation;

// tackle rotation around Y first

Vector3 localTarget = transform.InverseTransformPoint(m\_Target.position);

float yAngle = Mathf.Atan2(localTarget.x, localTarget.z)\*Mathf.Rad2Deg;

yAngle = Mathf.Clamp(yAngle, -m\_RotationRange.y\*0.5f, m\_RotationRange.y\*0.5f);

transform.localRotation = m\_OriginalRotation\*Quaternion.Euler(0, yAngle, 0);

// then recalculate new local target position for rotation around X

localTarget = transform.InverseTransformPoint(m\_Target.position);

float xAngle = Mathf.Atan2(localTarget.y, localTarget.z)\*Mathf.Rad2Deg;

xAngle = Mathf.Clamp(xAngle, -m\_RotationRange.x\*0.5f, m\_RotationRange.x\*0.5f);

var targetAngles = new Vector3(m\_FollowAngles.x + Mathf.DeltaAngle(m\_FollowAngles.x, xAngle),

m\_FollowAngles.y + Mathf.DeltaAngle(m\_FollowAngles.y, yAngle));

// smoothly interpolate the current angles to the target angles

m\_FollowAngles = Vector3.SmoothDamp(m\_FollowAngles, targetAngles, ref m\_FollowVelocity, m\_FollowSpeed);

// and update the gameobject itself

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

}

}

}