using System;

using System.Collections;

using UnityEngine;

namespace UnityStandardAssets.Cameras

{

public class ProtectCameraFromWallClip : MonoBehaviour

{

public float clipMoveTime = 0.05f; // time taken to move when avoiding cliping (low value = fast, which it should be)

public float returnTime = 0.4f; // time taken to move back towards desired position, when not clipping (typically should be a higher value than clipMoveTime)

public float sphereCastRadius = 0.1f; // the radius of the sphere used to test for object between camera and target

public bool visualiseInEditor; // toggle for visualising the algorithm through lines for the raycast in the editor

public float closestDistance = 0.5f; // the closest distance the camera can be from the target

public bool protecting { get; private set; } // used for determining if there is an object between the target and the camera

public string dontClipTag = "Player"; // don't clip against objects with this tag (useful for not clipping against the targeted object)

private Transform m\_Cam; // the transform of the camera

private Transform m\_Pivot; // the point at which the camera pivots around

private float m\_OriginalDist; // the original distance to the camera before any modification are made

private float m\_MoveVelocity; // the velocity at which the camera moved

private float m\_CurrentDist; // the current distance from the camera to the target

private Ray m\_Ray; // the ray used in the lateupdate for casting between the camera and the target

private RaycastHit[] m\_Hits; // the hits between the camera and the target

private RayHitComparer m\_RayHitComparer; // variable to compare raycast hit distances

private void Start()

{

// find the camera in the object hierarchy

m\_Cam = GetComponentInChildren<Camera>().transform;

m\_Pivot = m\_Cam.parent;

m\_OriginalDist = m\_Cam.localPosition.magnitude;

m\_CurrentDist = m\_OriginalDist;

// create a new RayHitComparer

m\_RayHitComparer = new RayHitComparer();

}

private void LateUpdate()

{

// initially set the target distance

float targetDist = m\_OriginalDist;

m\_Ray.origin = m\_Pivot.position + m\_Pivot.forward\*sphereCastRadius;

m\_Ray.direction = -m\_Pivot.forward;

// initial check to see if start of spherecast intersects anything

var cols = Physics.OverlapSphere(m\_Ray.origin, sphereCastRadius);

bool initialIntersect = false;

bool hitSomething = false;

// loop through all the collisions to check if something we care about

for (int i = 0; i < cols.Length; i++)

{

if ((!cols[i].isTrigger) &&

!(cols[i].attachedRigidbody != null && cols[i].attachedRigidbody.CompareTag(dontClipTag)))

{

initialIntersect = true;

break;

}

}

// if there is a collision

if (initialIntersect)

{

m\_Ray.origin += m\_Pivot.forward\*sphereCastRadius;

// do a raycast and gather all the intersections

m\_Hits = Physics.RaycastAll(m\_Ray, m\_OriginalDist - sphereCastRadius);

}

else

{

// if there was no collision do a sphere cast to see if there were any other collisions

m\_Hits = Physics.SphereCastAll(m\_Ray, sphereCastRadius, m\_OriginalDist + sphereCastRadius);

}

// sort the collisions by distance

Array.Sort(m\_Hits, m\_RayHitComparer);

// set the variable used for storing the closest to be as far as possible

float nearest = Mathf.Infinity;

// loop through all the collisions

for (int i = 0; i < m\_Hits.Length; i++)

{

// only deal with the collision if it was closer than the previous one, not a trigger, and not attached to a rigidbody tagged with the dontClipTag

if (m\_Hits[i].distance < nearest && (!m\_Hits[i].collider.isTrigger) &&

!(m\_Hits[i].collider.attachedRigidbody != null &&

m\_Hits[i].collider.attachedRigidbody.CompareTag(dontClipTag)))

{

// change the nearest collision to latest

nearest = m\_Hits[i].distance;

targetDist = -m\_Pivot.InverseTransformPoint(m\_Hits[i].point).z;

hitSomething = true;

}

}

// visualise the cam clip effect in the editor

if (hitSomething)

{

Debug.DrawRay(m\_Ray.origin, -m\_Pivot.forward\*(targetDist + sphereCastRadius), Color.red);

}

// hit something so move the camera to a better position

protecting = hitSomething;

m\_CurrentDist = Mathf.SmoothDamp(m\_CurrentDist, targetDist, ref m\_MoveVelocity,

m\_CurrentDist > targetDist ? clipMoveTime : returnTime);

m\_CurrentDist = Mathf.Clamp(m\_CurrentDist, closestDistance, m\_OriginalDist);

m\_Cam.localPosition = -Vector3.forward\*m\_CurrentDist;

}

// comparer for check distances in ray cast hits

public class RayHitComparer : IComparer

{

public int Compare(object x, object y)

{

return ((RaycastHit) x).distance.CompareTo(((RaycastHit) y).distance);

}

}

}

}