using System;

using UnityEngine;

namespace UnityStandardAssets.Vehicles.Aeroplane

{

public class AeroplaneControlSurfaceAnimator : MonoBehaviour

{

[SerializeField] private float m\_Smoothing = 5f; // The smoothing applied to the movement of control surfaces.

[SerializeField] private ControlSurface[] m\_ControlSurfaces; // Collection of control surfaces.

private AeroplaneController m\_Plane; // Reference to the aeroplane controller.

private void Start()

{

// Get the reference to the aeroplane controller.

m\_Plane = GetComponent<AeroplaneController>();

// Store the original local rotation of each surface, so we can rotate relative to this

foreach (var surface in m\_ControlSurfaces)

{

surface.originalLocalRotation = surface.transform.localRotation;

}

}

private void Update()

{

foreach (var surface in m\_ControlSurfaces)

{

switch (surface.type)

{

case ControlSurface.Type.Aileron:

{

// Ailerons rotate around the x axis, according to the plane's roll input

Quaternion rotation = Quaternion.Euler(surface.amount\*m\_Plane.RollInput, 0f, 0f);

RotateSurface(surface, rotation);

break;

}

case ControlSurface.Type.Elevator:

{

// Elevators rotate negatively around the x axis, according to the plane's pitch input

Quaternion rotation = Quaternion.Euler(surface.amount\*-m\_Plane.PitchInput, 0f, 0f);

RotateSurface(surface, rotation);

break;

}

case ControlSurface.Type.Rudder:

{

// Rudders rotate around their y axis, according to the plane's yaw input

Quaternion rotation = Quaternion.Euler(0f, surface.amount\*m\_Plane.YawInput, 0f);

RotateSurface(surface, rotation);

break;

}

case ControlSurface.Type.RuddervatorPositive:

{

// Ruddervators are a combination of rudder and elevator, and rotate

// around their z axis by a combination of the yaw and pitch input

float r = m\_Plane.YawInput + m\_Plane.PitchInput;

Quaternion rotation = Quaternion.Euler(0f, 0f, surface.amount\*r);

RotateSurface(surface, rotation);

break;

}

case ControlSurface.Type.RuddervatorNegative:

{

// ... and because ruddervators are "special", we need a negative version too. >\_<

float r = m\_Plane.YawInput - m\_Plane.PitchInput;

Quaternion rotation = Quaternion.Euler(0f, 0f, surface.amount\*r);

RotateSurface(surface, rotation);

break;

}

}

}

}

private void RotateSurface(ControlSurface surface, Quaternion rotation)

{

// Create a target which is the surface's original rotation, rotated by the input.

Quaternion target = surface.originalLocalRotation\*rotation;

// Slerp the surface's rotation towards the target rotation.

surface.transform.localRotation = Quaternion.Slerp(surface.transform.localRotation, target,

m\_Smoothing\*Time.deltaTime);

}

// This class presents a nice custom structure in which to define each of the plane's contol surfaces to animate.

// They show up in the inspector as an array.

[Serializable]

public class ControlSurface // Control surfaces represent the different flaps of the aeroplane.

{

public enum Type // Flaps differ in position and rotation and are represented by different types.

{

Aileron, // Horizontal flaps on the wings, rotate on the x axis.

Elevator, // Horizontal flaps used to adjusting the pitch of a plane, rotate on the x axis.

Rudder, // Vertical flaps on the tail, rotate on the y axis.

RuddervatorNegative, // Combination of rudder and elevator.

RuddervatorPositive, // Combination of rudder and elevator.

}

public Transform transform; // The transform of the control surface.

public float amount; // The amount by which they can rotate.

public Type type; // The type of control surface.

[HideInInspector] public Quaternion originalLocalRotation; // The rotation of the surface at the start.

}

}

}