using System;

using UnityEngine;

using Random = UnityEngine.Random;

namespace UnityStandardAssets.Vehicles.Car

{

[RequireComponent(typeof (CarController))]

public class CarAudio : MonoBehaviour

{

// This script reads some of the car's current properties and plays sounds accordingly.

// The engine sound can be a simple single clip which is looped and pitched, or it

// can be a crossfaded blend of four clips which represent the timbre of the engine

// at different RPM and Throttle state.

// the engine clips should all be a steady pitch, not rising or falling.

// when using four channel engine crossfading, the four clips should be:

// lowAccelClip : The engine at low revs, with throttle open (i.e. begining acceleration at very low speed)

// highAccelClip : Thenengine at high revs, with throttle open (i.e. accelerating, but almost at max speed)

// lowDecelClip : The engine at low revs, with throttle at minimum (i.e. idling or engine-braking at very low speed)

// highDecelClip : Thenengine at high revs, with throttle at minimum (i.e. engine-braking at very high speed)

// For proper crossfading, the clips pitches should all match, with an octave offset between low and high.

public enum EngineAudioOptions // Options for the engine audio

{

Simple, // Simple style audio

FourChannel // four Channel audio

}

public EngineAudioOptions engineSoundStyle = EngineAudioOptions.FourChannel;// Set the default audio options to be four channel

public AudioClip lowAccelClip; // Audio clip for low acceleration

public AudioClip lowDecelClip; // Audio clip for low deceleration

public AudioClip highAccelClip; // Audio clip for high acceleration

public AudioClip highDecelClip; // Audio clip for high deceleration

public float pitchMultiplier = 1f; // Used for altering the pitch of audio clips

public float lowPitchMin = 1f; // The lowest possible pitch for the low sounds

public float lowPitchMax = 6f; // The highest possible pitch for the low sounds

public float highPitchMultiplier = 0.25f; // Used for altering the pitch of high sounds

public float maxRolloffDistance = 500; // The maximum distance where rollof starts to take place

public float dopplerLevel = 1; // The mount of doppler effect used in the audio

public bool useDoppler = true; // Toggle for using doppler

private AudioSource m\_LowAccel; // Source for the low acceleration sounds

private AudioSource m\_LowDecel; // Source for the low deceleration sounds

private AudioSource m\_HighAccel; // Source for the high acceleration sounds

private AudioSource m\_HighDecel; // Source for the high deceleration sounds

private bool m\_StartedSound; // flag for knowing if we have started sounds

private CarController m\_CarController; // Reference to car we are controlling

private void StartSound()

{

// get the carcontroller ( this will not be null as we have require component)

m\_CarController = GetComponent<CarController>();

// setup the simple audio source

m\_HighAccel = SetUpEngineAudioSource(highAccelClip);

// if we have four channel audio setup the four audio sources

if (engineSoundStyle == EngineAudioOptions.FourChannel)

{

m\_LowAccel = SetUpEngineAudioSource(lowAccelClip);

m\_LowDecel = SetUpEngineAudioSource(lowDecelClip);

m\_HighDecel = SetUpEngineAudioSource(highDecelClip);

}

// flag that we have started the sounds playing

m\_StartedSound = true;

}

private void StopSound()

{

//Destroy all audio sources on this object:

foreach (var source in GetComponents<AudioSource>())

{

Destroy(source);

}

m\_StartedSound = false;

}

// Update is called once per frame

private void Update()

{

// get the distance to main camera

float camDist = (Camera.main.transform.position - transform.position).sqrMagnitude;

// stop sound if the object is beyond the maximum roll off distance

if (m\_StartedSound && camDist > maxRolloffDistance\*maxRolloffDistance)

{

StopSound();

}

// start the sound if not playing and it is nearer than the maximum distance

if (!m\_StartedSound && camDist < maxRolloffDistance\*maxRolloffDistance)

{

StartSound();

}

if (m\_StartedSound)

{

// The pitch is interpolated between the min and max values, according to the car's revs.

float pitch = ULerp(lowPitchMin, lowPitchMax, m\_CarController.Revs);

// clamp to minimum pitch (note, not clamped to max for high revs while burning out)

pitch = Mathf.Min(lowPitchMax, pitch);

if (engineSoundStyle == EngineAudioOptions.Simple)

{

// for 1 channel engine sound, it's oh so simple:

m\_HighAccel.pitch = pitch\*pitchMultiplier\*highPitchMultiplier;

m\_HighAccel.dopplerLevel = useDoppler ? dopplerLevel : 0;

m\_HighAccel.volume = 1;

}

else

{

// for 4 channel engine sound, it's a little more complex:

// adjust the pitches based on the multipliers

m\_LowAccel.pitch = pitch\*pitchMultiplier;

m\_LowDecel.pitch = pitch\*pitchMultiplier;

m\_HighAccel.pitch = pitch\*highPitchMultiplier\*pitchMultiplier;

m\_HighDecel.pitch = pitch\*highPitchMultiplier\*pitchMultiplier;

// get values for fading the sounds based on the acceleration

float accFade = Mathf.Abs(m\_CarController.AccelInput);

float decFade = 1 - accFade;

// get the high fade value based on the cars revs

float highFade = Mathf.InverseLerp(0.2f, 0.8f, m\_CarController.Revs);

float lowFade = 1 - highFade;

// adjust the values to be more realistic

highFade = 1 - ((1 - highFade)\*(1 - highFade));

lowFade = 1 - ((1 - lowFade)\*(1 - lowFade));

accFade = 1 - ((1 - accFade)\*(1 - accFade));

decFade = 1 - ((1 - decFade)\*(1 - decFade));

// adjust the source volumes based on the fade values

m\_LowAccel.volume = lowFade\*accFade;

m\_LowDecel.volume = lowFade\*decFade;

m\_HighAccel.volume = highFade\*accFade;

m\_HighDecel.volume = highFade\*decFade;

// adjust the doppler levels

m\_HighAccel.dopplerLevel = useDoppler ? dopplerLevel : 0;

m\_LowAccel.dopplerLevel = useDoppler ? dopplerLevel : 0;

m\_HighDecel.dopplerLevel = useDoppler ? dopplerLevel : 0;

m\_LowDecel.dopplerLevel = useDoppler ? dopplerLevel : 0;

}

}

}

// sets up and adds new audio source to the gane object

private AudioSource SetUpEngineAudioSource(AudioClip clip)

{

// create the new audio source component on the game object and set up its properties

AudioSource source = gameObject.AddComponent<AudioSource>();

source.clip = clip;

source.volume = 0;

source.loop = true;

// start the clip from a random point

source.time = Random.Range(0f, clip.length);

source.Play();

source.minDistance = 5;

source.maxDistance = maxRolloffDistance;

source.dopplerLevel = 0;

return source;

}

// unclamped versions of Lerp and Inverse Lerp, to allow value to exceed the from-to range

private static float ULerp(float from, float to, float value)

{

return (1.0f - value)\*from + value\*to;

}

}

}