```
1  using System;
2  using UnityEngine;
3
4  #pragma warning disable 649
5  namespace UnityStandardAssets.Vehicles.Car
6  {
7      internal enum CarDriveType
8      {
9          FrontWheelDrive,
10         RearWheelDrive,
11         FourWheelDrive
12     }
13
14     internal enum SpeedType
15     {
16         MPH,
17         KPH
18     }
19
20     public class CarController : MonoBehaviour
21     {
22         [SerializeField] private CarDriveType m_CarDriveType = CarDriveType.FourWheelDrive;
23         [SerializeField] private WheelCollider[] m_WheelColliders = new WheelCollider[4];
24         [SerializeField] private GameObject[] m_WheelMeshes = new GameObject[4];
25         [SerializeField] private WheelEffects[] m_WheelEffects = new WheelEffects[4];
26         [SerializeField] private Vector3 m_CentreOfMassOffset;
27         [SerializeField] private float m_MaximumSteerAngle;
28         [Range(0, 1)] [SerializeField] private float m_SteerHelper; // 0 is raw physics , 1 the car will grip in the
             direction it is facing
29         [Range(0, 1)] [SerializeField] private float m_TractionControl; // 0 is no traction control, 1 is full interference
30         [SerializeField] private float m_FullTorqueOverAllWheels;
31         [SerializeField] private float m_ReverseTorque;
32         [SerializeField] private float m_MaxHandbrakeTorque;
33         [SerializeField] private float m_Downforce = 100f;
34         [SerializeField] private SpeedType m_SpeedType;
```

```
35          [SerializeField] private float m_Topspeed = 200;
36          [SerializeField] private static int NoOfGears = 5;
37          [SerializeField] private float m_RevRangeBoundary = 1f;
38          [SerializeField] private float m_SlipLimit;
39          [SerializeField] private float m_BrakeTorque;
40          public float MySteerHelper = 0.66f;
41          private Quaternion[] m_WheelMeshLocalRotations;
42          private Vector3 m_Prevpos, m_Pos;
43          private float m_SteerAngle;
44          private int m_GearNum;
45          private float m_GearFactor;
46          private float m_OldRotation;
47          private float m_CurrentTorque;
48          private Rigidbody m_Rigidbody;
49          private const float k_ReversingThreshold = 0.01f;
50
51          public bool Player = true;
52          public bool F1Car = true;
53          public GameObject BrakeLightsOff;
54          public GameObject BrakeLightsOn;
55
56          public bool Skidding { get; private set; }
57          public float BrakeInput { get; private set; }
58          public float CurrentSteerAngle{ get { return m_SteerAngle; }}
59          public float CurrentSpeed{ get { return m_Rigidbody.velocity.magnitude*2.23693629f; }}
60          public float MaxSpeed{get { return m_Topspeed; }}
61          public float Revs { get; private set; }
62          public float AccelInput { get; private set; }
63
64          // Use this for initialization
65          private void Start()
66          {
67              m_WheelMeshLocalRotations = new Quaternion[4];
68              for (int i = 0; i < 4; i++)
69              {
```

```
70                  m_WheelMeshLocalRotations[i] = m_WheelMeshes[i].transform.localRotation;
71              }
72              m_WheelColliders[0].attachedRigidbody.centerOfMass = m_CentreOfMassOffset;
73
74              m_MaxHandbrakeTorque = float.MaxValue;
75
76              m_Rigidbody = GetComponent<Rigidbody>();
77              m_CurrentTorque = m_FullTorqueOverAllWheels - (m_TractionControl*m_FullTorqueOverAllWheels);
78
79              SaveScript.TopSpeed = m_Topspeed;
80
81          }
82
83          private void Update()
84          {
85              if (Player == true)
86              {
87                  SaveScript.Speed = CurrentSpeed;
88                  SaveScript.Gear = m_GearNum;
89              }
90              if(SaveScript.BrakeSlide == true)
91              {
92                  m_SteerHelper = 0.99f;
93              }
94              if(SaveScript.BrakeSlide == false)
95              {
96                  m_SteerHelper = MySteerHelper;
97              }
98          }
99
100         private void GearChanging()
101         {
102             float f = Mathf.Abs(CurrentSpeed/MaxSpeed);
103             float upgearlimit = (1/(float) NoOfGears)*(m_GearNum + 1);
104             float downgearlimit = (1/(float) NoOfGears)*m_GearNum;
```

```
105
106             if (m_GearNum > 0 && f < downgearlimit)
107             {
108                 m_GearNum--;
109             }
110
111             if (f > upgearlimit && (m_GearNum < (NoOfGears - 1)))
112             {
113                 m_GearNum++;
114             }
115         }
116
117
118         // simple function to add a curved bias towards 1 for a value in the 0-1 range
119         private static float CurveFactor(float factor)
120         {
121             return 1 - (1 - factor)*(1 - factor);
122         }
123
124
125         // unclamped version of Lerp, to allow value to exceed the from-to range
126         private static float ULerp(float from, float to, float value)
127         {
128             return (1.0f - value)*from + value*to;
129         }
130
131
132         private void CalculateGearFactor()
133         {
134             float f = (1/(float) NoOfGears);
135             // gear factor is a normalised representation of the current speed within the current gear's range of speeds.
136             // We smooth towards the 'target' gear factor, so that revs don't instantly snap up or down when changing gear.
137             var targetGearFactor = Mathf.InverseLerp(f*m_GearNum, f*(m_GearNum + 1), Mathf.Abs(CurrentSpeed/MaxSpeed));
138             m_GearFactor = Mathf.Lerp(m_GearFactor, targetGearFactor, Time.deltaTime*5f);
139         }
```

```
140
141
142         private void CalculateRevs()
143         {
144             // calculate engine revs (for display / sound)
145             // (this is done in retrospect - revs are not used in force/power calculations)
146             CalculateGearFactor();
147             var gearNumFactor = m_GearNum/(float) NoOfGears;
148             var revsRangeMin = ULerp(0f, m_RevRangeBoundary, CurveFactor(gearNumFactor));
149             var revsRangeMax = ULerp(m_RevRangeBoundary, 1f, gearNumFactor);
150             Revs = ULerp(revsRangeMin, revsRangeMax, m_GearFactor);
151         }
152
153
154         public void Move(float steering, float accel, float footbrake, float handbrake)
155         {
156                 for (int i = 0; i < 4; i++)
157                 {
158                     Quaternion quat;
159                     Vector3 position;
160                     m_WheelColliders[i].GetWorldPose(out position, out quat);
161                     m_WheelMeshes[i].transform.position = position;
162                     m_WheelMeshes[i].transform.rotation = quat;
163                 }
164
165                 //clamp input values
166                 steering = Mathf.Clamp(steering, -1, 1);
167                 AccelInput = accel = Mathf.Clamp(accel, 0, 1);
168                 BrakeInput = footbrake = -1 * Mathf.Clamp(footbrake, -1, 0);
169                 handbrake = Mathf.Clamp(handbrake, 0, 1);
170
171                 //Set the steer on the front wheels.
172                 //Assuming that wheels 0 and 1 are the front wheels.
173                 m_SteerAngle = steering * m_MaximumSteerAngle;
174                 m_WheelColliders[0].steerAngle = m_SteerAngle;
```

```
175                 m_WheelColliders[1].steerAngle = m_SteerAngle;
176
177                 SteerHelper();
178                 ApplyDrive(accel, footbrake);
179                 CapSpeed();
180
181                 //Set the handbrake.
182                 //Assuming that wheels 2 and 3 are the rear wheels.
183                 if (handbrake > 0f)
184                 {
185                     var hbTorque = handbrake * m_MaxHandbrakeTorque;
186                     m_WheelColliders[2].brakeTorque = hbTorque;
187                     m_WheelColliders[3].brakeTorque = hbTorque;
188             }
189
190
191                 CalculateRevs();
192                 GearChanging();
193
194                 AddDownForce();
195                 CheckForWheelSpin();
196                 TractionControl();
197
198             if(F1Car == false)
199             {
200                 if(footbrake > 0 && SaveScript.IsReversing == false || handbrake > 0)
201                 {
202                     BrakeLightsOff.SetActive(false);
203                     BrakeLightsOn.SetActive(true);
204                 }
205                 else if (footbrake <= 0 || handbrake <= 0 || SaveScript.IsReversing == true)
206                 {
207                     BrakeLightsOff.SetActive(true);
208                     BrakeLightsOn.SetActive(false);
209                 }
```

```
210             }
211         }
212
213
214         private void CapSpeed()
215         {
216             float speed = m_Rigidbody.velocity.magnitude;
217             switch (m_SpeedType)
218             {
219                 case SpeedType.MPH:
220
221                     speed *= 2.23693629f;
222                     if (speed > m_Topspeed)
223                         m_Rigidbody.velocity = (m_Topspeed/2.23693629f) * m_Rigidbody.velocity.normalized;
224                     break;
225
226                 case SpeedType.KPH:
227                     speed *= 3.6f;
228                     if (speed > m_Topspeed)
229                         m_Rigidbody.velocity = (m_Topspeed/3.6f) * m_Rigidbody.velocity.normalized;
230                     break;
231             }
232         }
233
234
235         private void ApplyDrive(float accel, float footbrake)
236         {
237
238            float thrustTorque;
239             switch (m_CarDriveType)
240             {
241                 case CarDriveType.FourWheelDrive:
242                     thrustTorque = accel * (m_CurrentTorque / 4f);
243                     for (int i = 0; i < 4; i++)
244                     {
```

```
245                         m_WheelColliders[i].motorTorque = thrustTorque;
246                     }
247                     break;
248 
249                 case CarDriveType.FrontWheelDrive:
250                     thrustTorque = accel * (m_CurrentTorque / 2f);
251                     m_WheelColliders[0].motorTorque = m_WheelColliders[1].motorTorque = thrustTorque;
252                     break;
253 
254                 case CarDriveType.RearWheelDrive:
255                     thrustTorque = accel * (m_CurrentTorque / 2f);
256                     m_WheelColliders[2].motorTorque = m_WheelColliders[3].motorTorque = thrustTorque;
257                     break;
258 
259             }
260 
261             for (int i = 0; i < 4; i++)
262             {
263                     if (CurrentSpeed > 5 && Vector3.Angle(transform.forward, m_Rigidbody.velocity) < 50f)
264                     {
265                         m_WheelColliders[i].brakeTorque = m_BrakeTorque * footbrake;
266                     }
267                     else if (footbrake > 0)
268                       {
269                           m_WheelColliders[i].brakeTorque = 0f;
270                           m_WheelColliders[i].motorTorque = -m_ReverseTorque*footbrake;
271                       }
272             }
273         }
274 
275 
276         private void SteerHelper()
277         {
278             for (int i = 0; i < 4; i++)
279             {
```

```
280                 WheelHit wheelhit;
281                 m_WheelColliders[i].GetGroundHit(out wheelhit);
282                 if (wheelhit.normal == Vector3.zero)
283                     return; // wheels arent on the ground so dont realign the rigidbody velocity
284             }
285 
286             // this if is needed to avoid gimbal lock problems that will make the car suddenly shift direction
287             if (Mathf.Abs(m_OldRotation - transform.eulerAngles.y) < 10f)
288             {
289                 var turnadjust = (transform.eulerAngles.y - m_OldRotation) * m_SteerHelper;
290                 Quaternion velRotation = Quaternion.AngleAxis(turnadjust, Vector3.up);
291                 m_Rigidbody.velocity = velRotation * m_Rigidbody.velocity;
292             }
293             m_OldRotation = transform.eulerAngles.y;
294         }
295 
296 
297         // this is used to add more grip in relation to speed
298         private void AddDownForce()
299         {
300             m_WheelColliders[0].attachedRigidbody.AddForce(-transform.up*m_Downforce*
301                                                            m_WheelColliders[0].attachedRigidbody.velocity.magnitude);
302         }
303 
304 
305         // checks if the wheels are spinning and is so does three things
306         // 1) emits particles
307         // 2) plays tiure skidding sounds
308         // 3) leaves skidmarks on the ground
309         // these effects are controlled through the WheelEffects class
310         private void CheckForWheelSpin()
311         {
312             // loop through all wheels
313                 for (int i = 0; i < 4; i++)
314                 {
```

```
315                     WheelHit wheelHit;
316                     m_WheelColliders[i].GetGroundHit(out wheelHit);
317 
318                 if (wheelHit.collider)
319                 {
320                     if (Player == true)
321                     {
322                         if (SaveScript.OnTheTerrain == true)
323                         {
324                             if (wheelHit.collider.CompareTag("Road"))
325                             {
326                                 Debug.Log("On the road");
327                                 SaveScript.OnTheRoad = true;
328                                 SaveScript.OnTheTerrain = false;
329                             }
330                         }
331                         if (SaveScript.OnTheRoad == true)
332                         {
333                             if (wheelHit.collider.CompareTag("Terrain"))
334                             {
335                                 Debug.Log("On the terrain");
336                                 SaveScript.OnTheRoad = false;
337                                 SaveScript.OnTheTerrain = true;
338                             }
339                         }
340                     }
341                 }
342 
343                 // is the tire slipping above the given threshhold
344                 if (Mathf.Abs(wheelHit.forwardSlip) >= m_SlipLimit || Mathf.Abs(wheelHit.sidewaysSlip) >= m_SlipLimit)
345                     {
346                         m_WheelEffects[i].EmitTyreSmoke();
347 
348                     // avoiding all four tires screeching at the same time
349                     // if they do it can lead to some strange audio artefacts
```

```
350                     if (!AnySkidSoundPlaying())
351                         {
352                             m_WheelEffects[i].PlayAudio();
353                         }
354                         continue;
355                     }
356 
357                 // if it wasnt slipping stop all the audio
358                 if (m_WheelEffects[i].PlayingAudio)
359                 {
360                     m_WheelEffects[i].StopAudio();
361                 }
362                 // end the trail generation
363                 m_WheelEffects[i].EndSkidTrail();
364             }
365         }
366 
367         // crude traction control that reduces the power to wheel if the car is wheel spinning too much
368         private void TractionControl()
369         {
370             WheelHit wheelHit;
371             switch (m_CarDriveType)
372             {
373                 case CarDriveType.FourWheelDrive:
374                     // loop through all wheels
375                     for (int i = 0; i < 4; i++)
376                     {
377                         m_WheelColliders[i].GetGroundHit(out wheelHit);
378 
379                         AdjustTorque(wheelHit.forwardSlip);
380                     }
381                     break;
382 
383                 case CarDriveType.RearWheelDrive:
384                     m_WheelColliders[2].GetGroundHit(out wheelHit);
```

```
385                     AdjustTorque(wheelHit.forwardSlip);
386
387                     if (wheelHit.collider)
388                     {
389                         if (Player == true)
390                         {
391                             if (wheelHit.collider.CompareTag("RumbleStrip") && CurrentSpeed > 10)
392                             {
393                                 SaveScript.Rumble1 = true;
394                             }
395                             else
396                             {
397                                 SaveScript.Rumble1 = false;
398                             }
399                         }
400                     }
401
402
403                     m_WheelColliders[3].GetGroundHit(out wheelHit);
404                     AdjustTorque(wheelHit.forwardSlip);
405
406                     if (wheelHit.collider)
407                     {
408                         if (Player == true)
409                         {
410                             if (wheelHit.collider.CompareTag("RumbleStrip") && CurrentSpeed > 10)
411                             {
412                                 SaveScript.Rumble2 = true;
413                             }
414                             else
415                             {
416                                 SaveScript.Rumble2 = false;
417                             }
418                         }
419                     }
```

```
420                     break;
421
422                 case CarDriveType.FrontWheelDrive:
423                     m_WheelColliders[0].GetGroundHit(out wheelHit);
424                     AdjustTorque(wheelHit.forwardSlip);
425
426                     m_WheelColliders[1].GetGroundHit(out wheelHit);
427                     AdjustTorque(wheelHit.forwardSlip);
428                     break;
429             }
430         }
431
432
433         private void AdjustTorque(float forwardSlip)
434         {
435             if (forwardSlip >= m_SlipLimit && m_CurrentTorque >= 0)
436             {
437                 m_CurrentTorque -= 10 * m_TractionControl;
438             }
439             else
440             {
441                 m_CurrentTorque += 10 * m_TractionControl;
442                 if (m_CurrentTorque > m_FullTorqueOverAllWheels)
443                 {
444                     m_CurrentTorque = m_FullTorqueOverAllWheels;
445                 }
446             }
447         }
448
449
450         private bool AnySkidSoundPlaying()
451         {
452             for (int i = 0; i < 4; i++)
453             {
454                 if (m_WheelEffects[i].PlayingAudio)
```

```
455                 {
456                     return true;
457                 }
458             }
459             return false;
460         }
461     }
462 }
463
```