DOCUMENTATION

PROMETEO

CAR CONTROLLER

mena

Content

1 Getting Started

1.1 How does it work?.

PROMETEO is a car controller that uses the wheel colliders inside the Unity 3D game engine in order to set up drivable vehicles. The driving style of this tool is semi-arcade.

You can set up your car in the following aspects:

- Maximum speed of the car.
- Maximum reverse speed of the car.
- Car's acceleration.
- Maximum steering angle.
- Steering speed.
- Brake force.
- Drifting multiplier.
- Car's mass center.
- Engine and drifting sounds.
- Particle systems and trail renderers for the smoke and skid of tires.
- UI text that shows the car's speed.

2 Tutorial (How to implement this controller on your cars?)

2.1 Step 1. Organizing the parts of your car.

First of all, all the parts of your car (body, wheels, spoiler, etc.) must be within an empty game object.

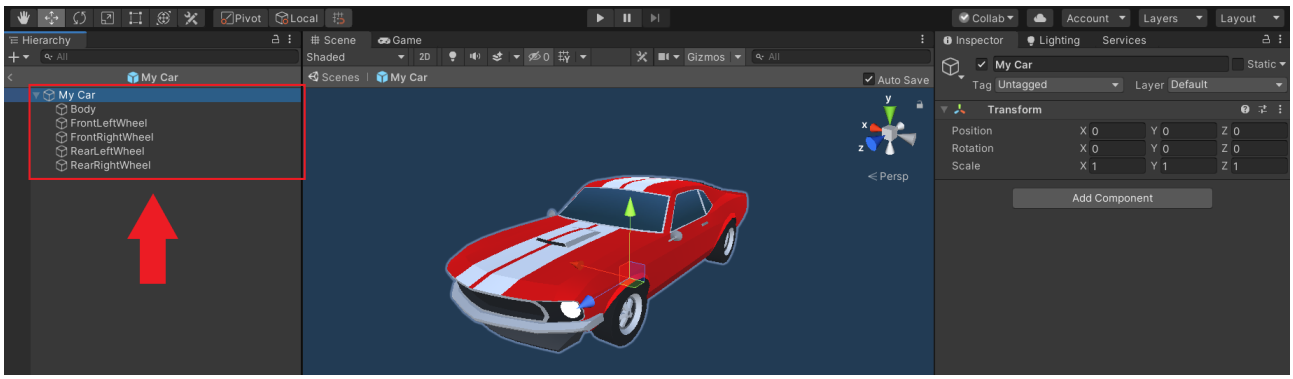

Figure 1: All the parts of the car are within an empty game object.

Now we are going to create an empty game object inside the 'My Car' game object in order to group all the wheels. You can do this with Right Click (with the root of the car's game object selected) > Create Empty. Make sure the position of this new empty game object is set to 0 in all directions. Then, rename it to 'Wheels'.

After that, create another empty game object inside 'Wheels' and call it 'Meshes'. You will drag and drop all the wheels of your car inside this game object called 'Meshes'. After doing this, you will get a hierarchy like the next one:

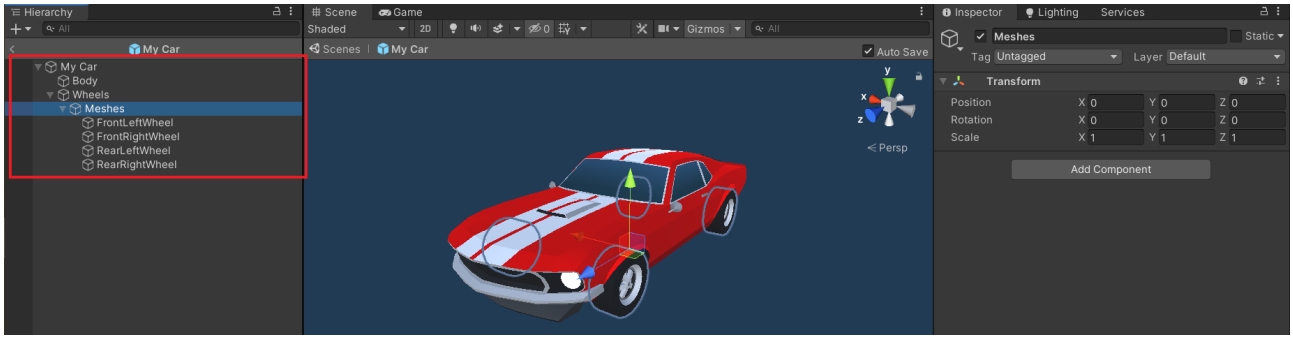

Figure 2: All the wheels of the car are within a game object called 'Meshes' and 'Meshes' is a child game object of 'Wheels'. Notice that the position of 'Meshes' and 'Wheels' is 0.

2.2 Step 2. Setting up the colliders.

Now we are going to set up the rigidbody of the car and also its colliders. First of all, you must select the root of the car's game object (the object that contains all the parts of your car). Then, you must go to Inspector > Add Component > Rigidbody. I recomment you to set the mass of your car's rigidbody in range from 700 kg to 1200 kg.

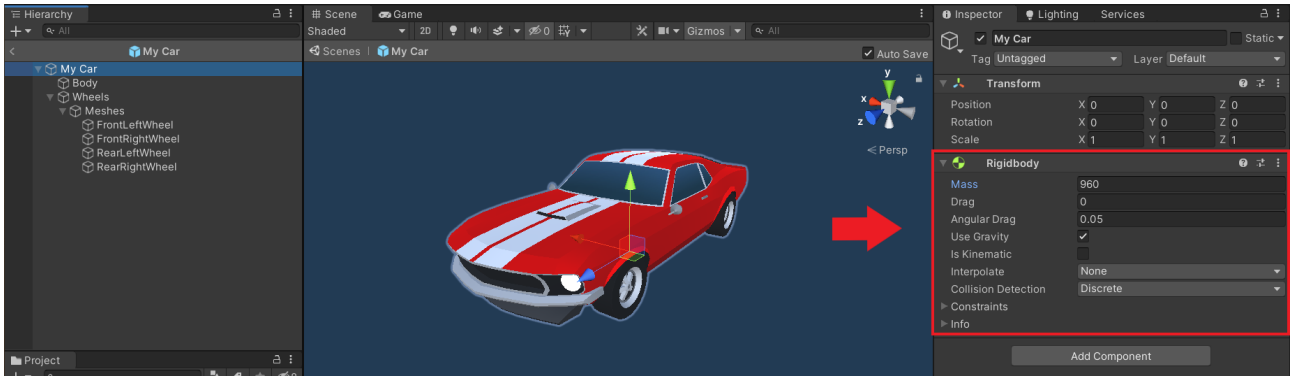

Figure 3: We added a rigidbody to our car and set its mass to 960 kg.

Now we are going to start adding colliders. First, select the 'Body' component of your car and then go to Inspector > Add Component > Box Collider. Now you should see that a green box is surrounding your car's body.

Lastly, we are going to add wheel colliders. For this, I recomment you to select the 'Meshes' game object (the object that contains all the car wheels), press *Ctrl + C* and then press *Ctrl + V*. Now rename this copied element to 'Colliders'.

After that, select all the wheels inside the object called 'Colliders' and go to Inspector > Add Component > Wheel Collider. This will add wheel colliders to all the 4 wheels. Now, under the wheel collider settings, go to Wheel Collider > Center > y = 0.15. Then, adjust the radius of the wheels collider until it matches the radius of the wheel meshes.

IMPORTANT: It is necessary that both the wheel colliders and wheel meshes are in different gameObjects, otherwise the controller will not work correctly.

Finally, delete the *MeshFilter* and *MeshRenderer* components from the wheels inside the 'Colliders' game object.

You will get something like this:

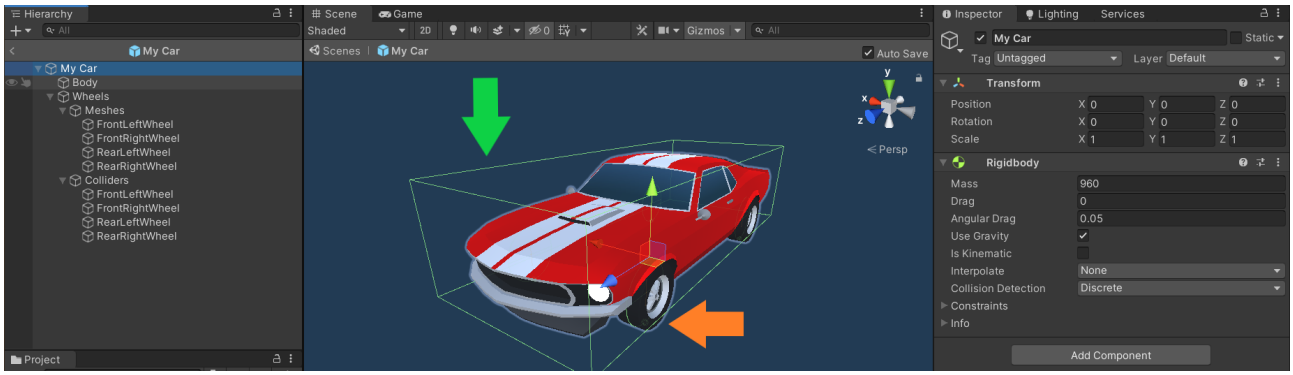

Figure 4: Green arrow: Collider of the body (it is a box sorrounding the car). Orange arrow: If you look closely, you will notice that a circle is sorrounding the wheels of the car (these circles are the wheel colliders).

2.3 Step 3. Adding the car controller.

This is the last step. Select the car container (the game object that has the rigidbody component) and go to Inspector > Add Component > Prometeo Car Controller. Now you just have to set up the features of your car such as the maximum speed, the maximum steering angle, the drifting multiplier and so on. If you need more information about these parameters you can check the Public Variables section within this document.

Below these settings you will need to drag and drop the wheel (meshes) gameobjects and the wheel collider components of your car.

If you need help to set up the 'PrometeoCarController' component you can check the demo scene inside this asset and compare the setup of the car in the demo with yours.

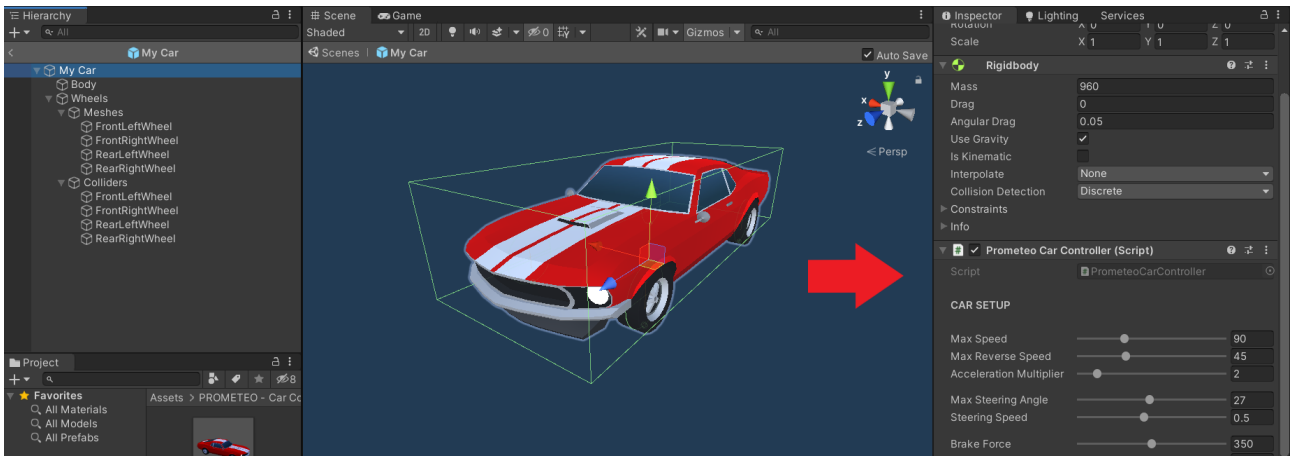

Figure 5: The root gameObject of the car has the *Prometeo Car Controller* component on it.

3 Tutorial (How to add touch controls?)

You can find a complete tutorial video (4 minutes) on how to use touch controls with this car controller. You can find it here: https://youtu.be/EiHQ88jYnlA.

4 Scripting

4.1 Public Variables.

Each one of the following variables can be edited from another script using a similar structure to the following code:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class MyScript : MonoBehaviour{

    public PrometeoCarController PCC;

    // Start is called before the first frame update
    void Start()
    {
```

```
12        PCC.variableName = value;
13    }
14
15 }
```

Name	Type	Description
maxSpeed	int	The maximum speed that the car can reach in km/h. Its value goes from 20 to 250.
maxReverseSpeed	int	The maximum speed that the car can reach while going on reverse in km/h. Its value goes from 10 to 120.
accelerationMultiplier	int	How fast the car can accelerate. 1 is a slow acceleration and 10 is the fastest. Its value goes from 1 to 10.
maxSteeringAngle	int	The maximum angle that the tires can reach while rotating the steering wheel. Its value goes from 10 to 45 degrees.
steeringSpeed	float	How fast the steering wheel turns. Its value goes from 0.1f to 1f.
brakeForce	int	The force of the wheel brakes. Its value goes from 100 to 600.
decelerationMultiplier	int	How fast the car decelerates when the user is not using the throttle. Its value goes from 1 to 10.
handbrakeDriftMultiplier	int	How much grip the car loses when the user hit the handbrake. Its value goes from 1 to 10.
bodyMassCenter	Vector3	This is a vector that contains the center of mass of the car. I recommend to set this value in the points $x = 0$ and $z = 0$ of your car. You can select the value that you want in the y axis, however, you must notice that the higher this value is, the more unstable the car becomes. Usually the y value goes from 0 to 1.5.
frontLeftMesh	GameObject	This variable will store the game object that contains the front left wheel of your car. It is the same case for the other variables of the wheel meshes.

frontLeftCollider	WheelCollider	This variable will store the wheels collider of the front left wheel of your car. It is the same case for the other variables of the wheel colliders.
RLWParticleSystem	ParticleSystem	This variable is used to store the particle systems of your car that emulate the tire smoke when the car drifts. It is the same case for the another variable *RRWParticleSystem*.
RLWTireSkid	TrailRenderer	This variable is used to store the trail renderers of your car that emulate the tire skid when the car loses traction.
carSpeed	float	This variable gives you the actual speed of the car. This value should not be changed manually and must be used only to get the information of the car's speed.
isDrifting	bool	This variable tells you if the car is drifting. This value should not be changed manually.
isTractionLocked	bool	This variable tells you if the car has locked its traction. This value should not be changed manually.
carEngineSound	AudioSource	This variable is used to store the sound of the car's engine.
tireScreechSound	AudioSource	This variable is used to store the sound of the car losing its traction. It will be played the the car starts to drift.

4.2 Public Methods.

4.2.1 GoForward ()

This function applies positive torque to the wheels in order to go forward. It will apply positive torque in all wheels if *maxSpeed* has not been reached (previously set by user), otherwise it will stop applying torque to the wheels.

If the car is going in reverse, then it will automatically apply brakes before applying positive torque to the wheels.

The next script calls this method whenever the user presses the *W* key.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class MyScript : MonoBehaviour{

    public PrometeoCarController PCC;

    // Update is called once per frame
    void Update()
    {
      if(Input.GetKey(KeyCode.W)){
        PCC.GoForward();
      }
    }

}
```

4.2.2 GoReverse ()

This function applies torque to the wheels in order to go in reverse. It will apply torque in all wheels if *maxReverseSpeed* has not been reached (previously set by user), otherwise it will stop applying torque to the wheels.

If the car is going forward, then it will automatically apply brakes before applying negative torque to the wheels.

The next script calls this method whenever the user presses the *S* key.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class MyScript : MonoBehaviour{

    public PrometeoCarController PCC;

    // Update is called once per frame
    void Update()
    {
      if(Input.GetKey(KeyCode.S)){
        PCC.GoReverse();
      }
    }

}
```

4.2.3 DecelerateCar ()

This function decelerates the speed of the car according to the *decelerationMultiplier* variable, where 1 is the slowest and 10 is the fastest deceleration. This method is called by the function InvokeRepeating, usually every 0.1f when the user is not pressing W (throttle), S (reverse) or Space bar (handbrake).

The next script calls this method repeatedly every 0.1 seconds after the user is not pressing W (throttle), S (brakes/reverse) or Space (handbrake) until the car stops.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class MyScript : MonoBehaviour{

    public PrometeoCarController PCC;

    // Update is called once per frame
    void Update()
    {
      if((!Input.GetKey(KeyCode.S) && !Input.GetKey(KeyCode.W)) &&
       ↪  !Input.GetKey(KeyCode.Space) && !PCC.deceleratingCar){

        PCC.InvokeRepeating("DecelerateCar", 0f, 0.1f);
        PCC.deceleratingCar = true;

      }
    }

}
```

4.2.4 Brakes ()

This function applies brake torque to the wheels according to the brake force given by the user (*brakeForce* variable).

The next script calls this method when the user presses the *B* key.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class MyScript : MonoBehaviour{

    public PrometeoCarController PCC;

    // Update is called once per frame
    void Update()
    {
      if(Input.GetKey(KeyCode.B)){
        PCC.Brakes();
      }
    }

}
```

4.2.5 Handbrake ()

This function is used to make the car lose traction. By using this, the car will start drifting. The amount of traction lost will depend on the *handbrakeDriftMultiplier* variable. If this value is small, then the car will not drift too much, but if it is high, then you could make the car to feel like going on ice.

The next script calls this method when the user presses the *Space* key.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class MyScript : MonoBehaviour{

    public PrometeoCarController PCC;

    // Update is called once per frame
    void Update()
    {
      if(Input.GetKey(KeyCode.Space)){
        PCC.CancelInvoke("DecelerateCar");
        PCC.deceleratingCar = false;
        PCC.Handbrake();
      }
    }

}
```

4.2.6 RecoverTraction ()

This function is used to recover the traction of the car when the user has stopped using the car's handbrake. Basically what it does is to reset the *extremumSlip* values of the wheels to their original state.

The next script calls this method when the user stops pressing the *Space* (handbrake) key.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class MyScript : MonoBehaviour{

    public PrometeoCarController PCC;

    // Update is called once per frame
    void Update()
    {
      if(Input.GetKeyUp(KeyCode.Space)){
        PCC.RecoverTraction();
      }
    }

}
```

4.2.7 TurnLeft () and TurnRight()

These functions turn the front car wheels either to left or right. The speed of this movement will depend on the steeringSpeed variable.

The next script calls these methods when the user presses either *A* (turn left) key or *D* (turn right) key.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class MyScript : MonoBehaviour{

    public PrometeoCarController PCC;

    // Update is called once per frame
    void Update()
    {
      if(Input.GetKey(KeyCode.A)){
        PCC.TurnLeft();
      }
      if(Input.GetKey(KeyCode.D)){
        PCC.TurnRight();
      }
    }

}
```

4.2.8 ResetSteeringAngle()

This function takes the front car wheels to their default position (rotation = 0). The speed of this movement will depend on the steeringSpeed variable.

The next script calls these methods when the user has stopped pressing both *A* (turn left) key and *D* (turn right) key and the *steeringAxis* variable is not 0 (which means that the rotation of the wheels is not 0).

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class MyScript : MonoBehaviour{

    public PrometeoCarController PCC;

    // Update is called once per frame
    void Update()
    {
      if(!Input.GetKey(KeyCode.A) && !Input.GetKey(KeyCode.D) &&
       ↪  PCC.steeringAxis != 0){
        PCC.ResetSteeringAngle();
      }
    }

}
```

5 Full Script (With Comments)

```
/*
MESSAGE FROM CREATOR: This script was coded by Mena. You can use it in your games either these are commercial or
personal projects. You can even add or remove functions as you wish. However, you cannot sell copies of this
script by itself, since it is originally distributed as a free product.
I wish you the best for your project. Good luck!

P.S: If you need more cars, you can check my other vehicle assets on the Unity Asset Store, perhaps you could find
something useful for your game. Best regards, Mena.
*/

using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class PrometeoCarController : MonoBehaviour
{

    //CAR SETUP

      [Space(20)]
      [Header("CAR SETUP")]
      [Space(10)]
      [Range(20, 250)]
      public int maxSpeed = 90; //The maximum speed that the car can reach in km/h.
      [Range(10, 120)]
      public int maxReverseSpeed = 45; //The maximum speed that the car can reach while going on reverse in km/h.
      [Range(1, 10)]
      public int accelerationMultiplier = 2; // How fast the car can accelerate. 1 is a slow acceleration and 10 is the fastest.
      [Space(10)]
      [Range(10, 45)]
      public int maxSteeringAngle = 27; // The maximum angle that the tires can reach while rotating the steering wheel.
      [Range(0.1f, 1f)]
      public float steeringSpeed = 0.5f; // How fast the steering wheel turns.
      [Space(10)]
      [Range(100, 600)]
      public int brakeForce = 350; // The strength of the wheel brakes.
      [Range(1, 10)]
```

```
40      public int decelerationMultiplier = 2; // How fast the car decelerates
        ↪ when the user is not using the throttle.
41      [Range(1, 10)]
42      public int handbrakeDriftMultiplier = 5; // How much grip the car loses
        ↪ when the user hit the handbrake.
43      [Space(10)]
44      public Vector3 bodyMassCenter; // This is a vector that contains the
        ↪ center of mass of the car. I recommend to set this value
45                                     // in the points x = 0 and z = 0 of your
                                       ↪ car. You can select the value that
                                       ↪ you want in the y axis,
46                                     // however, you must notice that the
                                       ↪ higher this value is, the more
                                       ↪ unstable the car becomes.
47                                     // Usually the y value goes from 0 to
                                       ↪ 1.5.
48
49    //WHEELS
50
51      [Space(20)]
52      [Header("WHEELS")]
53      [Space(10)]
54      /*
55      The following variables are used to store the wheels' data of the car.
    ↪ We need both the mesh-only game objects and wheel
56      collider components of the wheels. The wheel collider components and 3D
    ↪ meshes of the wheels cannot come from the same
57      game object; they must be separate game objects.
58      */
59      public GameObject frontLeftMesh;
60      public WheelCollider frontLeftCollider;
61      [Space(10)]
62      public GameObject frontRightMesh;
63      public WheelCollider frontRightCollider;
64      [Space(10)]
65      public GameObject rearLeftMesh;
66      public WheelCollider rearLeftCollider;
67      [Space(10)]
68      public GameObject rearRightMesh;
69      public WheelCollider rearRightCollider;
70
71    //PARTICLE SYSTEMS
72
73      [Space(20)]
74      [Header("EFFECTS")]
75      [Space(10)]
76      // The following particle systems are used as tire smoke when the car
        ↪ drifts.
77      public ParticleSystem RLWParticleSystem;
```

```
78      public ParticleSystem RRWParticleSystem;
79
80      [Space(10)]
81      // The following trail renderers are used as tire skids when the car
        ↪ loses traction.
82      public TrailRenderer RLWTireSkid;
83      public TrailRenderer RRWTireSkid;
84
85    //CAR DATA
86
87      [HideInInspector]
88      public float carSpeed; // Used to store the speed of the car.
89      [HideInInspector]
90      public bool isDrifting; // Used to know whether the car is drifting or
        ↪ not.
91      [HideInInspector]
92      public bool isTractionLocked; // Used to know whether the traction of
        ↪ the car is locked or not.
93
94    //SPEED TEXT (UI)
95
96      [Space(20)]
97      [Header("UI")]
98      [Space(10)]
99      public Text carSpeedText; // Used to store the UI object that is going
        ↪ to show the speed of the car.
100
101   //SOUNDS
102
103     [Space(20)]
104     [Header("Sounds")]
105     [Space(10)]
106     public AudioSource carEngineSound; // This variable stores the sound of
        ↪ the car engine.
107     public AudioSource tireScreechSound; // This variable stores the sound
        ↪ of the tire screech (when the car is drifting).
108     float initialCarEngineSoundPitch; // Used to store the initial pitch of
        ↪ the car engine sound.
109
110   //PRIVATE VARIABLES
111
112     /*
113     IMPORTANT: The following variables should not be modified manually
    ↪ since their values are automatically given via script.
114     */
115     Rigidbody carRigidbody; // Stores the car's rigidbody.
116     float steeringAxis; // Used to know whether the steering wheel has
        ↪ reached the maximum value. It goes from -1 to 1.
117     float throttleAxis; // Used to know whether the throttle has reached
        ↪ the maximum value. It goes from -1 to 1.
```

```
118       float driftingAxis;
119       float localVelocityZ;
120       float localVelocityX;
121       bool deceleratingCar;
122       /*
123       The following variables are used to store information about sideways
      ↪ friction of the wheels (such as
124       extremumSlip,extremumValue, asymptoteSlip, asymptoteValue and
      ↪ stiffness). We change this values to
125       make the car to start drifting.
126       */
127       WheelFrictionCurve FLwheelFriction;
128       float FLWextremumSlip;
129       WheelFrictionCurve FRwheelFriction;
130       float FRWextremumSlip;
131       WheelFrictionCurve RLwheelFriction;
132       float RLWextremumSlip;
133       WheelFrictionCurve RRwheelFriction;
134       float RRWextremumSlip;
135
136     // Start is called before the first frame update
137     void Start()
138     {
139       //In this part, we set the 'carRigidbody' value with the Rigidbody
          ↪ attached to this
140       //gameObject. Also, we define the center of mass of the car with the
          ↪ Vector3 given
141       //in the inspector.
142       carRigidbody = gameObject.GetComponent<Rigidbody>();
143       carRigidbody.centerOfMass = bodyMassCenter;
144
145       //Initial setup to calculate the drift value of the car. This part
          ↪ could look a bit
146       //complicated, but do not be afraid, the only thing we're doing here is
          ↪ to save the default
147       //friction values of the car wheels so we can set an appropiate
          ↪ drifting value later.
148       FLwheelFriction = new WheelFrictionCurve ();
149         FLwheelFriction.extremumSlip =
            ↪ frontLeftCollider.sidewaysFriction.extremumSlip;
150         FLWextremumSlip = frontLeftCollider.sidewaysFriction.extremumSlip;
151         FLwheelFriction.extremumValue =
            ↪ frontLeftCollider.sidewaysFriction.extremumValue;
152         FLwheelFriction.asymptoteSlip =
            ↪ frontLeftCollider.sidewaysFriction.asymptoteSlip;
153         FLwheelFriction.asymptoteValue =
            ↪ frontLeftCollider.sidewaysFriction.asymptoteValue;
154         FLwheelFriction.stiffness =
            ↪ frontLeftCollider.sidewaysFriction.stiffness;
```

```
155         FRwheelFriction = new WheelFrictionCurve ();
156           FRwheelFriction.extremumSlip =
              ↪ frontRightCollider.sidewaysFriction.extremumSlip;
157           FRWextremumSlip = frontRightCollider.sidewaysFriction.extremumSlip;
158           FRwheelFriction.extremumValue =
              ↪ frontRightCollider.sidewaysFriction.extremumValue;
159           FRwheelFriction.asymptoteSlip =
              ↪ frontRightCollider.sidewaysFriction.asymptoteSlip;
160           FRwheelFriction.asymptoteValue =
              ↪ frontRightCollider.sidewaysFriction.asymptoteValue;
161           FRwheelFriction.stiffness =
              ↪ frontRightCollider.sidewaysFriction.stiffness;
162         RLwheelFriction = new WheelFrictionCurve ();
163           RLwheelFriction.extremumSlip =
              ↪ rearLeftCollider.sidewaysFriction.extremumSlip;
164           RLWextremumSlip = rearLeftCollider.sidewaysFriction.extremumSlip;
165           RLwheelFriction.extremumValue =
              ↪ rearLeftCollider.sidewaysFriction.extremumValue;
166           RLwheelFriction.asymptoteSlip =
              ↪ rearLeftCollider.sidewaysFriction.asymptoteSlip;
167           RLwheelFriction.asymptoteValue =
              ↪ rearLeftCollider.sidewaysFriction.asymptoteValue;
168           RLwheelFriction.stiffness =
              ↪ rearLeftCollider.sidewaysFriction.stiffness;
169         RRwheelFriction = new WheelFrictionCurve ();
170           RRwheelFriction.extremumSlip =
              ↪ rearRightCollider.sidewaysFriction.extremumSlip;
171           RRWextremumSlip = rearRightCollider.sidewaysFriction.extremumSlip;
172           RRwheelFriction.extremumValue =
              ↪ rearRightCollider.sidewaysFriction.extremumValue;
173           RRwheelFriction.asymptoteSlip =
              ↪ rearRightCollider.sidewaysFriction.asymptoteSlip;
174           RRwheelFriction.asymptoteValue =
              ↪ rearRightCollider.sidewaysFriction.asymptoteValue;
175           RRwheelFriction.stiffness =
              ↪ rearRightCollider.sidewaysFriction.stiffness;
176
177           // We save the initial pitch of the car engine sound.
178           initialCarEngineSoundPitch = carEngineSound.pitch;
179
180           // We invoke 2 methods inside this script. CarSpeedUI() changes the
              ↪ text of the UI object that stores
181           // the speed of the car and CarSounds() controls the engine and
              ↪ drifting sounds. Both methods are invoked
182           // in 0 seconds, and repeatedly called every 0.1 seconds.
183           InvokeRepeating("CarSpeedUI", 0f, 0.1f);
184           InvokeRepeating("CarSounds", 0f, 0.1f);
185       }
186
```

```
187     // Update is called once per frame
188     void Update()
189     {
190 
191       //CAR DATA
192 
193       // We determine the speed of the car.
194       carSpeed = (2 * Mathf.PI * frontLeftCollider.radius *
          ↪ frontLeftCollider.rpm * 60) / 1000;
195       // Save the local velocity of the car in the x axis. Used to know if
          ↪ the car is drifting.
196       localVelocityX =
          ↪ transform.InverseTransformDirection(carRigidbody.velocity).x;
197       // Save the local velocity of the car in the z axis. Used to know if
          ↪ the car is going forward or backwards.
198       localVelocityZ =
          ↪ transform.InverseTransformDirection(carRigidbody.velocity).z;
199 
200       //CAR PHYSICS
201 
202       /*
203       The following methods are called whenever a certain key is pressed. For
    ↪ example, in the first 'if' we call the
204       method GoForward() if the user has pressed W.
205 
206       In this part of the code we specify what the car needs to do if the
    ↪ user presses W (throttle), S (reverse),
207       A (turn left), D (turn right) or Space bar (handbrake).
208       */
209 
210       if(Input.GetKey(KeyCode.W)){
211         CancelInvoke("DecelerateCar");
212         deceleratingCar = false;
213         GoForward();
214       }
215       if(Input.GetKey(KeyCode.S)){
216         CancelInvoke("DecelerateCar");
217         deceleratingCar = false;
218         GoReverse();
219       }
220 
221       if(Input.GetKey(KeyCode.A)){
222         TurnLeft();
223       }
224       if(Input.GetKey(KeyCode.D)){
225         TurnRight();
226       }
227       if(Input.GetKey(KeyCode.Space)){
228         CancelInvoke("DecelerateCar");
```

```
229         decelaratingCar = false;
230         Handbrake();
231       }
232       if(Input.GetKeyUp(KeyCode.Space)){
233         RecoverTraction();
234       }
235       if((!Input.GetKey(KeyCode.S) && !Input.GetKey(KeyCode.W)) &&
          ↪  !Input.GetKey(KeyCode.Space) && !decelaratingCar){
236         InvokeRepeating("DecelerateCar", 0f, 0.1f);
237         decelaratingCar = true;
238       }
239       if(!Input.GetKey(KeyCode.A) && !Input.GetKey(KeyCode.D) && steeringAxis
          ↪  != 0){
240         ResetSteeringAngle();
241       }
242
243       // We call the method AnimateWheelMeshes() in order to match the wheel
          ↪  collider movements with the 3D meshes of the wheels.
244       AnimateWheelMeshes();
245
246     }
247
248     // This method converts the car speed data from float to string, and then
          ↪  set the text of the UI carSpeedText with this value.
249     public void CarSpeedUI(){
250       float absoluteCarSpeed = Mathf.Abs(carSpeed);
251       carSpeedText.text = Mathf.RoundToInt(absoluteCarSpeed).ToString();
252     }
253
254     // This method controls the car sounds. For example, the car engine will
          ↪  sound slow when the car speed is low because the
255     // pitch of the sound will be at its lowest point. On the other hand, it
          ↪  will sound fast when the car speed is high because
256     // the pitch of the sound will be the sum of the initial pitch + the car
          ↪  speed divided by 100f.
257     // Apart from that, the tireScreechSound will play whenever the car
          ↪  starts drifting or losing traction.
258     public void CarSounds(){
259       float engineSoundPitch = initialCarEngineSoundPitch +
          ↪  (Mathf.Abs(carSpeed) / 100f);
260       carEngineSound.pitch = engineSoundPitch;
261       if((isDrifting || isTractionLocked) && !tireScreechSound.isPlaying){
262         tireScreechSound.Play();
263       }else if(!isDrifting && !isTractionLocked){
264         tireScreechSound.Stop();
265       }
266     }
267
268     //
```

```
    //STEERING METHODS
    //

    //The following method turns the front car wheels to the left. The speed of this movement will depend on the steeringSpeed variable.
    public void TurnLeft(){
      steeringAxis = steeringAxis - (Time.deltaTime * 10f * steeringSpeed);
      if(steeringAxis < -1f){
        steeringAxis = -1f;
      }
      var steeringAngle = steeringAxis * maxSteeringAngle;
      frontLeftCollider.steerAngle = Mathf.Lerp(frontLeftCollider.steerAngle, steeringAngle, 0.5f);
      frontRightCollider.steerAngle = Mathf.Lerp(frontRightCollider.steerAngle, steeringAngle, 0.5f);
    }

    //The following method turns the front car wheels to the right. The speed of this movement will depend on the steeringSpeed variable.
    public void TurnRight(){
      steeringAxis = steeringAxis + (Time.deltaTime * 10f * steeringSpeed);
      if(steeringAxis > 1f){
        steeringAxis = 1f;
      }
      var steeringAngle = steeringAxis * maxSteeringAngle;
      frontLeftCollider.steerAngle = Mathf.Lerp(frontLeftCollider.steerAngle, steeringAngle, 0.5f);
      frontRightCollider.steerAngle = Mathf.Lerp(frontRightCollider.steerAngle, steeringAngle, 0.5f);
    }

    //The following method takes the front car wheels to their default position (rotation = 0). The speed of this movement will depend
    // on the steeringSpeed variable.
    public void ResetSteeringAngle(){
      steeringAxis = Input.GetAxis("Horizontal");
      var steeringAngle = steeringAxis;
      frontLeftCollider.steerAngle = Mathf.Lerp(frontLeftCollider.steerAngle, steeringAngle, steeringSpeed);
      frontRightCollider.steerAngle = Mathf.Lerp(frontRightCollider.steerAngle, steeringAngle, steeringSpeed);
    }

    // This method matches both the position and rotation of the WheelColliders with the WheelMeshes.
    void AnimateWheelMeshes(){
      try{
        Quaternion FLWRotation;
```

```
307         Vector3 FLWPosition;
308         frontLeftCollider.GetWorldPose(out FLWPosition, out FLWRotation);
309         frontLeftMesh.transform.position = FLWPosition;
310         frontLeftMesh.transform.rotation = FLWRotation;
311 
312         Quaternion FRWRotation;
313         Vector3 FRWPosition;
314         frontRightCollider.GetWorldPose(out FRWPosition, out FRWRotation);
315         frontRightMesh.transform.position = FRWPosition;
316         frontRightMesh.transform.rotation = FRWRotation;
317 
318         Quaternion RLWRotation;
319         Vector3 RLWPosition;
320         rearLeftCollider.GetWorldPose(out RLWPosition, out RLWRotation);
321         rearLeftMesh.transform.position = RLWPosition;
322         rearLeftMesh.transform.rotation = RLWRotation;
323 
324         Quaternion RRWRotation;
325         Vector3 RRWPosition;
326         rearRightCollider.GetWorldPose(out RRWPosition, out RRWRotation);
327         rearRightMesh.transform.position = RRWPosition;
328         rearRightMesh.transform.rotation = RRWRotation;
329       }catch(Exception ex){
330         Debug.LogWarning(ex);
331       }
332     }
333 
334     //
335     //ENGINE AND BRAKING METHODS
336     //
337 
338     // This method apply positive torque to the wheels in order to go
        ↪ forward.
339     void GoForward(){
340       //If the forces aplied to the rigidbody in the 'x' asis are greater
          ↪ than
341       //3f, it means that the car is losing traction, then the car will start
          ↪ emitting particle systems.
342       if(Mathf.Abs(localVelocityX) > 2.5f){
343         isDrifting = true;
344         DriftCarPS();
345       }else{
346         isDrifting = false;
347         DriftCarPS();
348       }
349       // The following part sets the throttle power to 1 smoothly.
350       throttleAxis = throttleAxis + (Time.deltaTime * 3f);
351       if(throttleAxis > 1f){
352         throttleAxis = 1f;
```

```
353       }
354       //If the car is going backwards, then apply brakes in order to avoid
          ↪ strange
355       //behaviours. If the local velocity in the 'z' axis is less than -1f,
          ↪ then it
356       //is safe to apply positive torque to go forward.
357       if(localVelocityZ < -1f){
358         Brakes();
359       }else{
360         if(Mathf.RoundToInt(carSpeed) < maxSpeed){
361           //Apply positive torque in all wheels to go forward if maxSpeed has
              ↪ not been reached.
362           frontLeftCollider.brakeTorque = 0;
363           frontLeftCollider.motorTorque = (accelerationMultiplier * 50f) *
              ↪ throttleAxis;
364           frontRightCollider.brakeTorque = 0;
365           frontRightCollider.motorTorque = (accelerationMultiplier * 50f) *
              ↪ throttleAxis;
366           rearLeftCollider.brakeTorque = 0;
367           rearLeftCollider.motorTorque = (accelerationMultiplier * 50f) *
              ↪ throttleAxis;
368           rearRightCollider.brakeTorque = 0;
369           rearRightCollider.motorTorque = (accelerationMultiplier * 50f) *
              ↪ throttleAxis;
370         }else {
371           // If the maxSpeed has been reached, then stop applying torque to
              ↪ the wheels.
372           // IMPORTANT: The maxSpeed variable should be considered as an
              ↪ approximation; the speed of the car
373           // could be a bit higher than expected.
374                               frontLeftCollider.motorTorque = 0;
375                               frontRightCollider.motorTorque = 0;
376           rearLeftCollider.motorTorque = 0;
377                               rearRightCollider.motorTorque = 0;
378                     }
379       }
380     }
381
382     // This method apply negative torque to the wheels in order to go
        ↪ backwards.
383     public void GoReverse(){
384       //If the forces aplied to the rigidbody in the 'x' asis are greater
          ↪ than
385       //3f, it means that the car is losing traction, then the car will start
          ↪ emitting particle systems.
386       if(Mathf.Abs(localVelocityX) > 2.5f){
387         isDrifting = true;
388         DriftCarPS();
389       }else{
```

```
390         isDrifting = false;
391         DriftCarPS();
392       }
393       // The following part sets the throttle power to -1 smoothly.
394       throttleAxis = throttleAxis - (Time.deltaTime * 3f);
395       if(throttleAxis < -1f){
396         throttleAxis = -1f;
397       }
398       //If the car is still going forward, then apply brakes in order to
          ↪ avoid strange
399       //behaviours. If the local velocity in the 'z' axis is greater than 1f,
          ↪ then it
400       //is safe to apply negative torque to go reverse.
401       if(localVelocityZ > 1f){
402         Brakes();
403       }else{
404         if(Mathf.Abs(Mathf.RoundToInt(carSpeed)) < maxReverseSpeed){
405           //Apply negative torque in all wheels to go in reverse if
              ↪ maxReverseSpeed has not been reached.
406           frontLeftCollider.brakeTorque = 0;
407           frontLeftCollider.motorTorque = (accelerationMultiplier * 50f) *
              ↪ throttleAxis;
408           frontRightCollider.brakeTorque = 0;
409           frontRightCollider.motorTorque = (accelerationMultiplier * 50f) *
              ↪ throttleAxis;
410           rearLeftCollider.brakeTorque = 0;
411           rearLeftCollider.motorTorque = (accelerationMultiplier * 50f) *
              ↪ throttleAxis;
412           rearRightCollider.brakeTorque = 0;
413           rearRightCollider.motorTorque = (accelerationMultiplier * 50f) *
              ↪ throttleAxis;
414         }else {
415           //If the maxReverseSpeed has been reached, then stop applying
              ↪ torque to the wheels.
416           // IMPORTANT: The maxReverseSpeed variable should be considered as
              ↪ an approximation; the speed of the car
417           // could be a bit higher than expected.
418                               frontLeftCollider.motorTorque = 0;
419                               frontRightCollider.motorTorque = 0;
420           rearLeftCollider.motorTorque = 0;
421                               rearRightCollider.motorTorque = 0;
422                     }
423       }
424     }
425
426     // The following method decelerates the speed of the car according to the
        ↪ decelerationMultiplier variable, where
427     // 1 is the slowest and 10 is the fastest deceleration. This method is
        ↪ called by the function InvokeRepeating,
```

```
    // usually every 0.1f when the user is not pressing W (throttle), S
    ↪ (reverse) or Space bar (handbrake).
    public void DecelerateCar(){
      if(Mathf.Abs(localVelocityX) > 2.5f){
        isDrifting = true;
        DriftCarPS();
      }else{
        isDrifting = false;
        DriftCarPS();
      }
      // The following part resets the throttle power to 0 smoothly.
      if(throttleAxis != 0f){
        if(throttleAxis > 0f){
          throttleAxis = throttleAxis - (Time.deltaTime * 10f);
        }else if(throttleAxis < 0f){
            throttleAxis = throttleAxis + (Time.deltaTime * 10f);
        }
        if(Mathf.Abs(throttleAxis) < 0.15f){
          throttleAxis = 0f;
        }
      }
      carRigidbody.velocity = carRigidbody.velocity * (1f / (1f + (0.025f *
      ↪ decelerationMultiplier)));
      // Since we want to decelerate the car, we are going to remove the
      ↪ torque from the wheels of the car.
      frontLeftCollider.motorTorque = 0;
      frontRightCollider.motorTorque = 0;
      rearLeftCollider.motorTorque = 0;
      rearRightCollider.motorTorque = 0;
      // If the magnitude of the car's velocity is less than 0.25f (very slow
      ↪ velocity), then stop the car completely and
      // also cancel the invoke of this method.
      if(carRigidbody.velocity.magnitude < 0.25f){
        carRigidbody.velocity = Vector3.zero;
        CancelInvoke("DecelerateCar");
      }
    }

    // This function applies brake torque to the wheels according to the
    ↪ brake force given by the user.
    public void Brakes(){
      frontLeftCollider.brakeTorque = brakeForce;
      frontRightCollider.brakeTorque = brakeForce;
      rearLeftCollider.brakeTorque = brakeForce;
      rearRightCollider.brakeTorque = brakeForce;
    }

    // This function is used to make the car lose traction. By using this,
    ↪ the car will start drifting. The amount of traction lost
```

```
471     // will depend on the handbrakeDriftMultiplier variable. If this value is
        ↪ small, then the car will not drift too much, but if
472     // it is high, then you could make the car to feel like going on ice.
473     public void Handbrake(){
474       CancelInvoke("RecoverTraction");
475       // We are going to start losing traction smoothly, there is were our
          ↪ 'driftingAxis' variable takes
476       // place. This variable will start from 0 and will reach a top value of
          ↪ 1, which means that the maximum
477       // drifting value has been reached. It will increase smoothly by using
          ↪ the variable Time.deltaTime.
478       driftingAxis = driftingAxis + (Time.deltaTime);
479       float secureStartingPoint = driftingAxis * FLWextremumSlip *
          ↪ handbrakeDriftMultiplier;
480
481       if(secureStartingPoint < FLWextremumSlip){
482         driftingAxis = FLWextremumSlip / (FLWextremumSlip *
            ↪ handbrakeDriftMultiplier);
483       }
484       if(driftingAxis > 1f){
485         driftingAxis = 1f;
486       }
487       //If the forces aplied to the rigidbody in the 'x' asis are greater
          ↪ than
488       //3f, it means that the car lost its traction, then the car will start
          ↪ emitting particle systems.
489       if(Mathf.Abs(localVelocityX) > 2.5f){
490         isDrifting = true;
491       }else{
492         isDrifting = false;
493       }
494       //If the 'driftingAxis' value is not 1f, it means that the wheels have
          ↪ not reach their maximum drifting
495       //value, so, we are going to continue increasing the sideways friction
          ↪ of the wheels until driftingAxis
496       // = 1f.
497       if(driftingAxis < 1f){
498         FLwheelFriction.extremumSlip = FLWextremumSlip *
            ↪ handbrakeDriftMultiplier * driftingAxis;
499         frontLeftCollider.sidewaysFriction = FLwheelFriction;
500
501         FRwheelFriction.extremumSlip = FRWextremumSlip *
            ↪ handbrakeDriftMultiplier * driftingAxis;
502         frontRightCollider.sidewaysFriction = FRwheelFriction;
503
504         RLwheelFriction.extremumSlip = RLWextremumSlip *
            ↪ handbrakeDriftMultiplier * driftingAxis;
505         rearLeftCollider.sidewaysFriction = RLwheelFriction;
506
```

```
507         RRwheelFriction.extremumSlip = RRWextremumSlip *
              ↪ handbrakeDriftMultiplier * driftingAxis;
508         rearRightCollider.sidewaysFriction = RRwheelFriction;
509       }
510
511       // Whenever the player uses the handbrake, it means that the wheels are
            ↪ locked, so we set 'isTractionLocked = true'
512       // and, as a consequense, the car starts to emit trails to simulate the
            ↪ wheel skids.
513       isTractionLocked = true;
514       DriftCarPS();
515
516     }
517
518     // This function is used to emit both the particle systems of the tires'
          ↪ smoke and the trail renderers of the tire skids
519     // depending on the value of the bool variables 'isDrifting' and
          ↪ 'isTractionLocked'.
520     public void DriftCarPS(){
521       try{
522         if(isDrifting){
523           RLWParticleSystem.Play();
524           RRWParticleSystem.Play();
525         }else if(!isDrifting){
526           RLWParticleSystem.Stop();
527           RRWParticleSystem.Stop();
528         }
529       }catch(Exception ex){
530         Debug.LogWarning(ex);
531       }
532
533       try{
534         if(isTractionLocked || Mathf.Abs(localVelocityX) > 8f){
535           RLWTireSkid.emitting = true;
536           RRWTireSkid.emitting = true;
537         }else {
538           RLWTireSkid.emitting = false;
539           RRWTireSkid.emitting = false;
540         }
541       }catch(Exception ex){
542         Debug.LogWarning(ex);
543       }
544
545     }
546
547     // This function is used to recover the traction of the car when the user
          ↪ has stopped using the car's handbrake.
548     public void RecoverTraction(){
549       isTractionLocked = false;
```

```
      driftingAxis = driftingAxis - (Time.deltaTime / 1.5f);
      if(driftingAxis < 0f){
        driftingAxis = 0f;
      }

      //If the 'driftingAxis' value is not 0f, it means that the wheels have not recovered their traction.
      //We are going to continue decreasing the sideways friction of the wheels until we reach the initial
      // car's grip.
      if(FLwheelFriction.extremumSlip > FLWextremumSlip){
        FLwheelFriction.extremumSlip = FLWextremumSlip * handbrakeDriftMultiplier * driftingAxis;
        frontLeftCollider.sidewaysFriction = FLwheelFriction;

        FRwheelFriction.extremumSlip = FRWextremumSlip * handbrakeDriftMultiplier * driftingAxis;
        frontRightCollider.sidewaysFriction = FRwheelFriction;

        RLwheelFriction.extremumSlip = RLWextremumSlip * handbrakeDriftMultiplier * driftingAxis;
        rearLeftCollider.sidewaysFriction = RLwheelFriction;

        RRwheelFriction.extremumSlip = RRWextremumSlip * handbrakeDriftMultiplier * driftingAxis;
        rearRightCollider.sidewaysFriction = RRwheelFriction;

        Invoke("RecoverTraction", Time.deltaTime);

      }else if (FLwheelFriction.extremumSlip < FLWextremumSlip){
        FLwheelFriction.extremumSlip = FLWextremumSlip;
        frontLeftCollider.sidewaysFriction = FLwheelFriction;

        FRwheelFriction.extremumSlip = FRWextremumSlip;
        frontRightCollider.sidewaysFriction = FRwheelFriction;

        RLwheelFriction.extremumSlip = RLWextremumSlip;
        rearLeftCollider.sidewaysFriction = RLwheelFriction;

        RRwheelFriction.extremumSlip = RRWextremumSlip;
        rearRightCollider.sidewaysFriction = RRwheelFriction;

        driftingAxis = 0f;
      }
    }

}
```