TPS Shooter (Military style)
Documentation

Thanks for downloading TPS Shooter (Military style).
We hope that you can easily create your games using this asset.

YouTube video of how to setup Player and Weapons
https://www.youtube.com/watch?v=JNtcQBJYJmo

YouTube video of how to setup Enemy.
https://www.youtube.com/watch?v=7GXQbrdOM-o

YouTube video of how to setup Vehicle.
https://www.youtube.com/watch?v=cSV9LXr3o98

If you have any **questions**, **problems** or **proposals** fell free to **contact** support@lightdev.io

Since update 6.22 asset uses LigthDev Framework. See Framework documentation in Framework folder.

P.S. **My friend**, please, write a **good review**, as it **motivates** us to improve our package ☺

Content:

Essential scripts

1. **Layers.** This class contains all layers that the game contains.

SaveLoad scripts

1. **PlayerPreferences.** Serializable class.
   Contains all variables that can be used in the game.

2. **SaveLoad**. Static class.
   1) Before using this script, call LoadPlayerPreferences() method.
   2) Then make change.
   3) Call SavePlayerPreferences() method.

Player Setup

To create Player you have to create 2 GameObjects:

1) **GameObject** with **PlayerBehaviour** script and **FootstepSounds** scripts.

2) **GameObject** with **TPSCamera** script.

Implementation

Player:

1) PlayerBehaviour has to be layered as Player.

2) In MovemetSettings in PlayerBehaviour and in FootstepSounds do not forget to define GroundLayers, otherwise the player will falling all time and foot step sounds will not play.

3) To create weapon add PlayerWeapon script to it. It is better if each weapon will use different bullets. To create bullet prefab, tag it as PlayerBullet and add PlayerBullet script.

TPS Camera:

1) The root GameObject has TPSCamera script.

2) This GameObject has child empty GameObject (pivot), that will be rotated.

3) MainCamera Container has to be a child GameObject of pivot. MainCamera is a child of MainCamera Container. Rotation and position of MainCamera has to be zero.

Weapon PickUp

1) We highly recommend to create Weapon using creator tool.

2) Then add Weapon Pick Up script.

3) Also add Weapon Pick Up Additional script for visual effects. If you want, you can create custom visual effects (like weapon rotation). Use events in Weapon Pick Up script for these purposes.

Bullet Decals

1) Use BulletsDecals scriptable object.
2) Define which Physics Material has to be used by decals.
3) Assign this Physics Material to Collider where these decals has to be used.
4) Assign this scriptable object in Bullet script.

Enemy

1) To create an enemy, you have to attach EnemyBehaviour script.

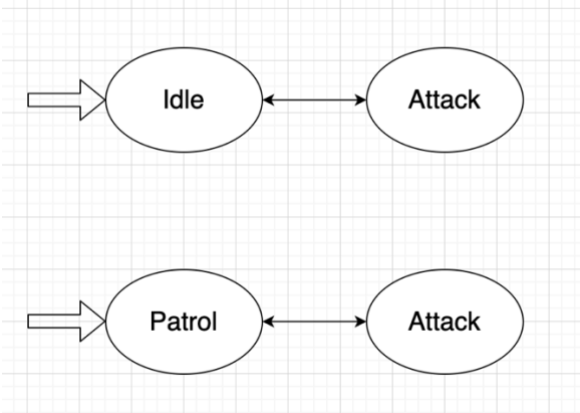
2) VisionLayers in VisionSettings. It defines what layers enemy will raycast (we recommend to choose Player and Default).

3) Add EnemyDamagable script to every collider that enemy contains in order to enemy has different damage in different body parts. Remember, after adding Collider also add Rigidbody. Because Collider without Rigidbody is a static rigidbody.

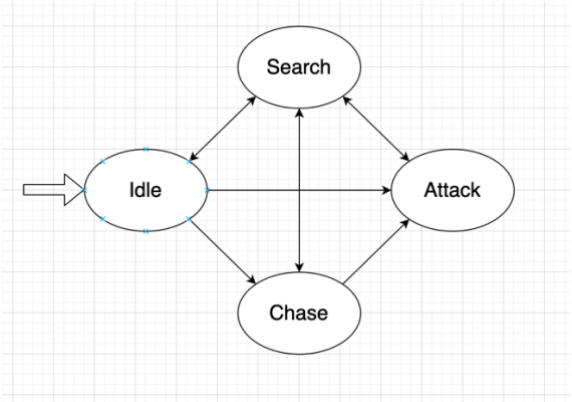
Enemy has different states that depends on AI_Behaviour -> SearchSettings and PatrollingSettings -> Waypoints.

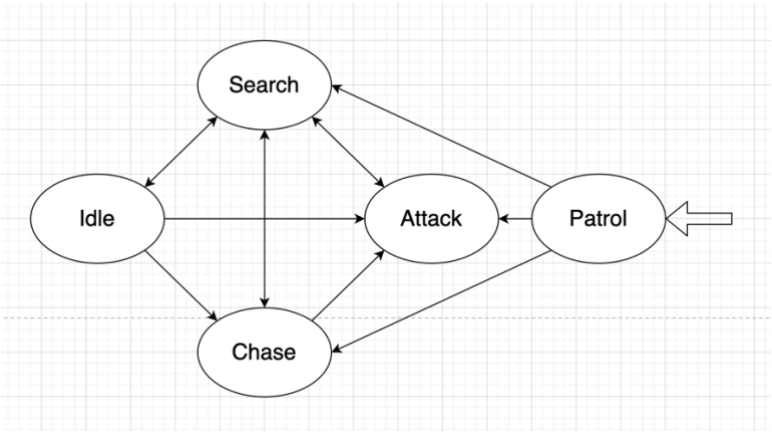
If you have added waypoints, the start state will Patrol.

1) If AI_Behaviour -> SearchSettings has chosen to None.



2) If AI_Behaviour -> SearchSettings has chosen to Search.





Enemy Generator

1) Generation Wave contains data about Enemy Prefab that will be instantiated at Spawn Points.

2) After all enemies from first generation wave are killed, next generation wave will spawn enemies.

Radar

1) Add Radar script to radar GameObject and tag it as Radar.

2) GameObject that has to be shown at this radar must contain RadarableObject script.

3) Enemy uses EnemyRadarableObject.

Menu

1) The asset has Menu scene with menu scripts.

2) It has Settings Preview, where you can change Mobile input preferences.

3) If you want to add weapon, add GameObject in BuyWeaponPreview GameObject in weapons. Player has to have this weapon in all weapons. Queue of weapons in BuyWeaponPreview is the same as in the Player.

Vehicle

How to setup a vehicle (YouTube video):
https://www.youtube.com/watch?v=cSV9LXr3o98

Vehicle Intro

A default WheelCollider is set up for a standard use case: a 1.5 tonne car with 4 wheels, where all wheels are perfectly symmetrical relative to the car's centre of mass. If you have a different use case, such as a vehicle with more wheels, you risk violating implicit assumptions in the physics simulation code and may see unpredictable results.

In addition to this, the meaning of the explicit spring stiffness and damper values in the WheelCollider component are somewhat unclear. It is not obvious how to tweak these values to simulate different kinds of suspension, such as a stiff sports suspension or one of a family car.

Vehicle Tools solves these problems. This package provides you with a wizard to create a drivable skeleton of a car with any amount of wheels, and with suspension settings that are very easy to tweak. It also provides a tool for debugging problems with friction calculations, to help with debugging acceleration and cornering.

EasySuspension

The EasySuspension script is a utility script that updates suspension settings across all child WheelCollider components. Simply enter the mass, natural frequency and damping ratio for the wheels, and the EasySuspension script will take care of updating each WheelCollider component with these settings.

Unless otherwise specified, units of measurement are SI units.
https://en.wikipedia.org/wiki/SI_base_unit

Natural Frequency	The natural frequency of the suspension springs. Natural frequency might be thought of as a mass-independent analogue for stiffness. Typical range [4..20]. A family car might have springs with the natural frequency of about 10. https://en.wikipedia.org/wiki/Natural_frequency
Damping Ratio	The damping ratio of the suspension springs. This parameter sets how fast the oscillations in the suspension comes to rest.

	Typical range [0..1]. A family car might have springs with the damping ratio of about 0.8. https://en.wikipedia.org/wiki/Damping_ratio
Force Shift	The distance to the point where the tyre forces are applied, starting from the center of mass of the vehicle, along the local Y axis.
Set Suspension Distance	Whether or not to adjust the travel distance of the suspension springs. This setting helps to avoid having physically incorrect configurations that still apply force at maximum suspension elongation.

WheelDrive

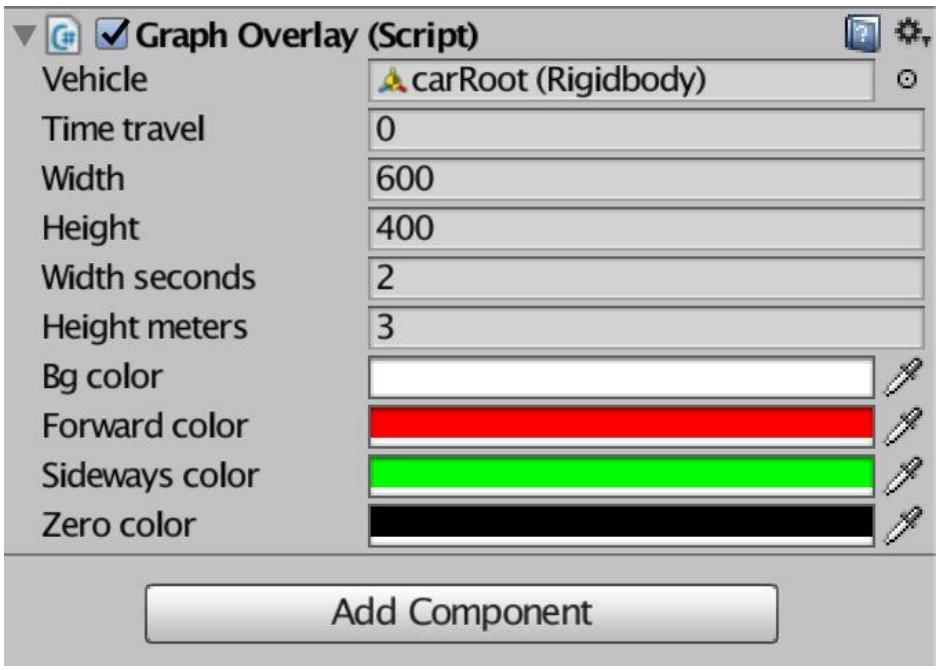
The WheelDrive component is a minimalistic vehicle controller. This script allows for quick testing out of the box, and provides a good start for your own customized vehicle controller.

Wheel Shape	If a prefab of a wheel is assigned here, wheels will be instantiated automatically when we enter Play Mode. The provided *toywheel* prefab can be used as an example.
Critical Speed, Steps Below, Steps Above	The parameters of the vehicle substepping algorithm. If the speed of the vehicle is below the speed specified in Critical Speed, the vehicle integrator will perform the number of substeps specified in Steps Below. If the speed of the vehicle is at or above the speed specified in Critical Speed, the vehicle integrator will perform the number of substeps specified in Steps Above.
Drive Type	Choose the drive type from front wheel drive, rear wheel drive or all wheel drive. Front wheels are those having their local z coordinate positive.

Debugging friction with the GraphOverlay component

Issues with friction are the most frequent cause of simulated vehicles failing to accelerate and corner correctly. Most often, this happens because the physics code is not performing enough substeps - that is to say, it is not making frequent enough calculations.

The GraphOverlay component helps us to debug the friction calculations of our generated drivable cars.



With the information from this component we can decide whether we need to adjust the number of substeps.

An introduction to the slip-based friction in Unity

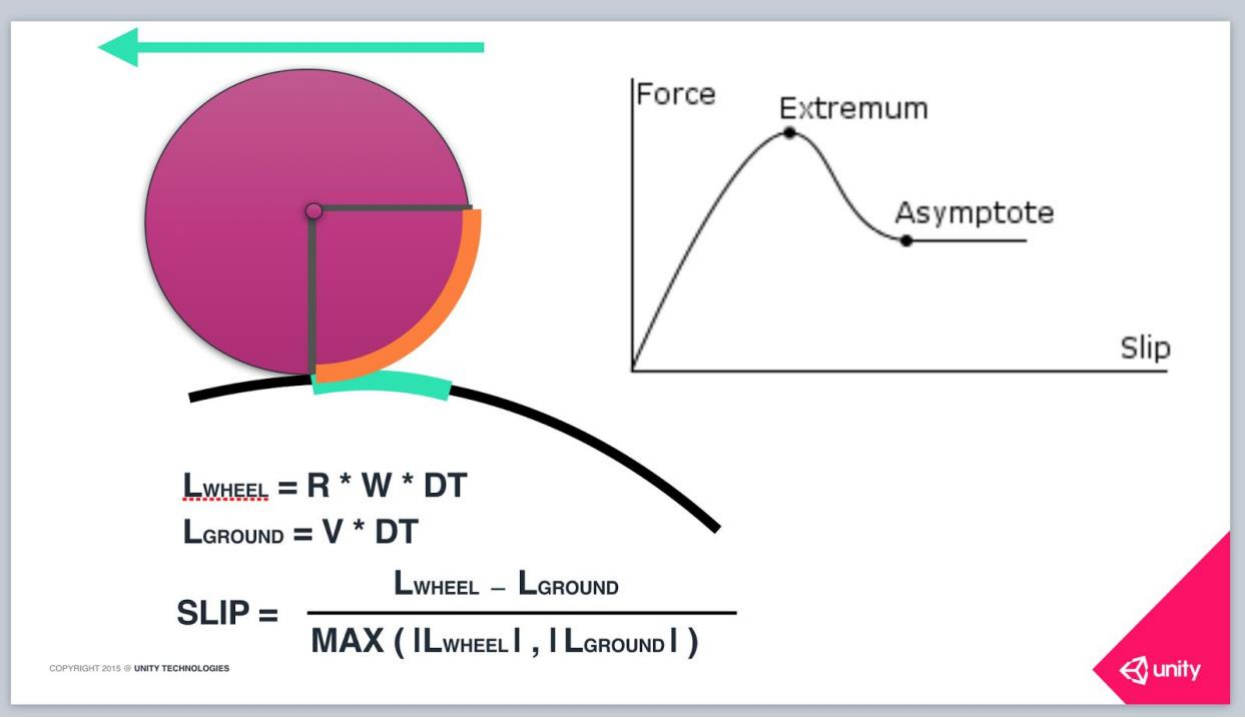
Unity uses the slip-based friction model to calculate how much longitudinal and lateral force should be applied to a vehicle that uses WheelColliders. Those forces determine how fast the vehicle reacts to acceleration as well as steering. In order to compute the forces, the physics engine works out the longitudinal and lateral slip values for each wheel first. Thus, all the computational inaccuracies that occur during the slip computation, have a direct impact on the forces produced by wheels, and, in the end, on the handling of your vehicle. The slips are computed from values like the angular speed of the wheel that are obtained an iterative integration process, and are sensitive to the amount of simulation sub-steps, and the natural frequency of the spring attached to the wheel. If it happens that there are not enough sub-steps for this given natural frequency of the spring, then one can observe a wheel oscillating instead of rotating smoothly. This is clearly visible on the slip graph shown by the GraphOverlay component, as explained below.

The slip-based friction model simulates a simplified version of the processes that take place in a real tyre. When a wheel starts turning, the rim of the wheel slips relative to the ground as the rubber of the tyre stretches and produces momentum. The more torque one applies to the wheel, the faster it rotates and the bigger slip it produces. Bigger slip stretches the rubber even more, producing more grip. This continues until the extremum slip is reached, after which the tyre loses the stiff contact with ground at the contact patch. At this point, the rubber stretch gets relaxed and thus produces less momentum. After the slip becomes bigger than the asymptote slip, it no longer matters how high the slip is: the rubber of the tyre reaches the state where the remaining slight contact with the surface would no longer let it compress further, and thus the momentum produced by the wheel remains constant no matter how much torque is applied.

As an example, let’s look at how the longitudinal slip works.

The picture below shows a wheel rolling to the left. Once the angular momentum (acceleration due to the engine; shown in cyan) is applied, the wheel starts spinning, and slips a little as the rubber gets compressed. Because of that, the distance along the surface travelled by the centre of the wheel will be different from the distance the tyre travelled along the surface (shown in orange). The physics engine computes the longitudinal slip as the difference between those two distances over the one that has the biggest absolute value. The result is normalised to be in [-1;1].

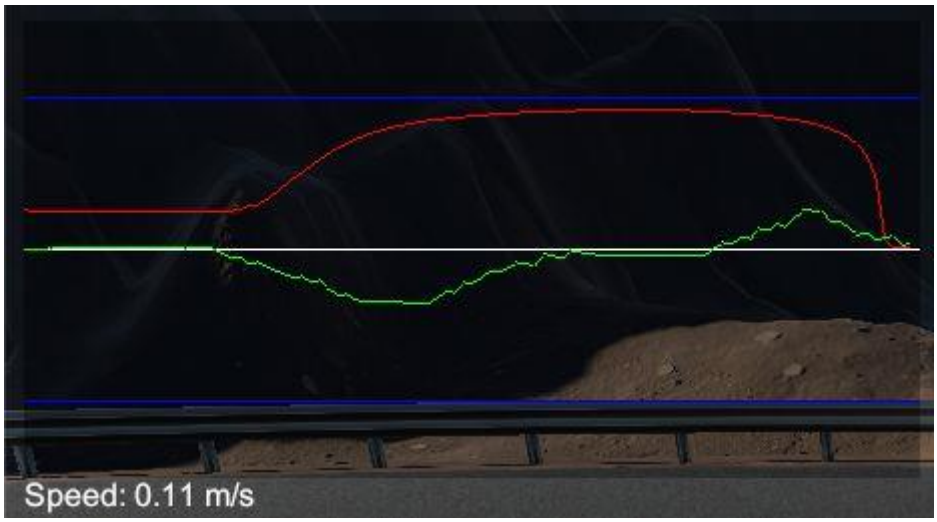
The lateral slip is computed in a similar way.



How to use the GraphOverlay component

- Add the GraphOverlay to any GameObject in your Scene
- Drag the carRoot GameObject of the car that you wish to debug into the Vehicle field of the GraphOverlay component
- Enter Play Mode and observe the graph of wheel slips in the Game view
- Accelerate the car (for example, from 0 to 100 km/h) and observe the changes in the graph

Interpreting the graph generated by the GraphOverlay component

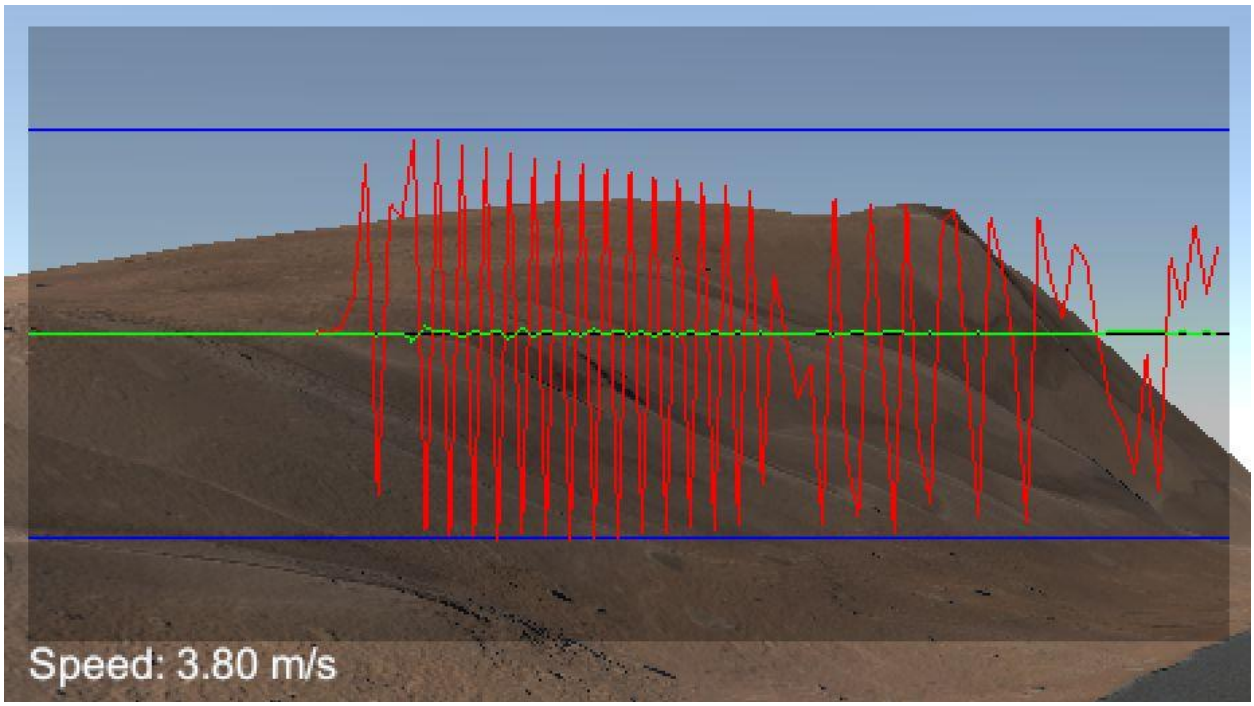


The x axis of the graph represents time and the y axis represents the slip value. The red and the green lines represent the longitudinal and lateral friction over time, respectively.

The blue lines on the graph represent the corridor [-1, 1]. The white line at the centre corresponds to the slip of 0.

Ideally, both curves should be smooth and fall close to 0, right where the value doesn't yet reach the extremum.

Consider the following graph, produced by a test car at low speed.

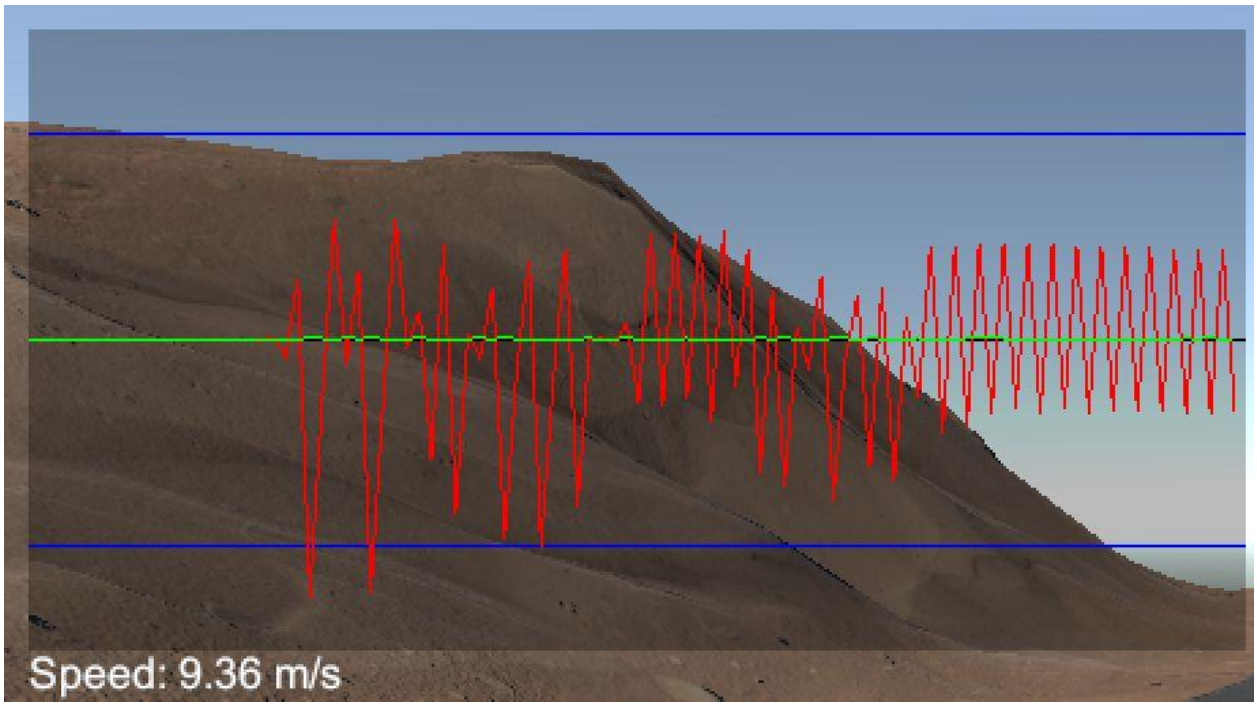


At these low speeds, the slip (the red line) repeatedly goes above and below zero (the green line). The oscillations are a direct indication that not enough substeps are taking place. This means that the code fails to work out the longitudinal slip, which results in

force being applied in opposing directions. This results in the vehicle struggling to accelerate.

In this case, the Critical Speed value for the WheelDrive component is set to the default value of 5 m/s. When the speed of the vehicle is below the Critical Speed value, the number of substeps is determined by the Steps Below value. As this problem occurs speeds below the Critical Speed, increasing the Steps Below value of the WheelDrive component is advised.

Now consider the following graph, taken from a test car at a high speed.



In this case, the slip does not jitter at above and below 0 moderate speeds. However, when the vehicle reaches a speed of 5m/s the slip began to jitter above and below 0.

Again, the Critical Speed value for the WheelDriver component is set to the default value of 5 m/s. Once the speed of the vehicle reaches the Critical Speed, the number of substeps is determined by the Steps Above value. In this case, increasing the value of Steps Above is advisable.

Further reading about vehicle

Learn more about why natural frequency and damping ratio is better than having to specify the raw spring values here:
https://www.youtube.com/watch?v=WGvuP6vV6j4&feature=youtu.be&t=17m

Having a working skeleton of a car is just one step from having a full vehicle working reasonably well. Learn about the steps involved in adding dynamics to any graphics model over here:

https://www.youtube.com/watch?v=xQcJAa6Ooa4

More in-depth information about how the simulation code works internally:

https://www.docdroid.net/vd5x/wc-info-pdf