F-GEAR

Extras

These components are not actually a part of vehicle physics but complementary tools that were used while making the demo scenes. You can use them for prototyping or learning purposes but they are far from being robust solutions and not recommended for production usage.

AIController : This is a simple ai driver component. The inputs of the system is a spline curve for the vehicles to follow and a speed/angle curve to limit vehicles speed around corners. If the spline control points are well placed and the speed/angle curve is tuned, the ai can achieve good lap times. Check the “race” demo to see it in action.

- First a target point on the spline is found. This point gets further as the vehicle goes faster.
- After that 5 rays are casted to look for potential collision targets. If an opponent is nearby, the target point is shifted to left or right to avoid collision. This target point determines the steering input.
- For throttle/brake inputs the tangent angle of the spline and the speed/angle curve is used. The x-axis of the speed/angle curve is the rotation angle in degrees and y-axis is the maximum speed value in kmh. The maximum speed allowed is retriewed from the speed/angle curve and throttle/brake inputs are set accordingly.
- Finally if the vehicle is stuck for 5 seconds, it is reset back to a position on the spline.

CarManager : This class has a couple of functionality:

- First of all it stores a list of vehicles which should be provided by the user. By using its ui you can switch between these vehicles, start/stop engine and activate/deactivate ai component if available.
- If a *CountDown* value is provided, all vehicles stay frozen at start and a count down text is displayed on screen.
- If a spline is provided and *ShowRankings* value is set then the rankings of the vehicles are calculated and displayed on top right of the screen. *ShowRankings* value must be set at the start or it may not work properly.
- Finally if user presses R button the vehicle is reset to a position on spline.

Effects : All the visual and sound effects are put together in this class. Engine sounds, particle effects, skidmarks, brake lights, muzzle effects... all are handled here.

- First a gameobject called “Effects” is created in the scene. It is found in resources folder and it contains the particle and skidmark effects. For the muzzle effect a gameobject must be provided by user.

- For the sound effects the user should provide the AudioClips. If clips are provided, the AudioSources are created automatically.
- For a simple brake light effect user should provide a renderer(s), a material index and a color. If target material is found, its emissive color is set when the vehicle brakes.
- Skidmarks, smoke particles and skid sound effects are triggered by using wheels slip values.
- Engine sound is effected by the provided parameters and the engine rpm.
- Muzzle effect and sound is triggered by a basic calculation based on rpm accelerations.
- A crash sound effect is played when a collision occurs.
- You can define surface types based on physic materials and change vehicles behavior and the effects according to the current hit surface. In *physicsSurfaceUpdate* function each wheels hit collider is checked for its PhysicMaterial. Then wheels friction values are changed according to the material. Also the drag value of the rigidbody is effected from the Roughness value(provided by the user) which aims to limit speed if vehicle goes out of the road.
- If *TerrainBasedSurface* is true then physic material based system is disabled, current hit terrains texture is used instead(*terrainSurfaceUpdate* function). You should define surface types but you do not need to set physics materials for surfaces. The index of the terrain texture corresponds to the surface index. Check *varying_grip.scene* for a demonstration.

ForceFeedback : *To use this class get the latest logitech gaming sdk, include the c# wrapper class LogitechGSDK and put the LogitechSteeringWheelEnginesWrapper.dll in the project. To test it in the race.scene first set Controller mode to "WHEEL" in input settings then activate ForceFeedback script(should already be attached) on player car.*

This one uses the logitech gaming sdk to generate forcefeedback effects on your wheel gear. Currently only tested with a logitech G29. At start the logitech steering wheel sdk is initated. It finds the available devices and get the first connected wheel id and sets its operating range. The default & maximum operating range for G29 is 900 degrees but it is set to 450 degrees for this script. There are 8 effect types used and some of these effects do not appear to work alongside with others, for ex. we need to disable spring effect when surface effect is active. Some of the effects uses AnimationCurves for calculations. If you have checked the *UseDefaultCurves* option(on by default) then you do not have to define these curves. If you want to modify the curves uncheck *UseDefaultCurves* and setup the curves manually. This also shows an optional gui that displays the power of each effect as percentages.

- **Led Effect** : Animates the led lights on the wheel according to the engine rpm.
- **Spring Effect** : Uses the *AlignCurve* to calculate a spring force that tries to align the wheel to the direction that the vehicle travels. Scale this effect with the *AlignForce*(0-100) coefficient.
- **Constant Effect** : Applies a constant opposite force on the wheel based on the *ConstantCurve*. *ConstantCurve*'s x-axis input is the vehicles speed. For ex. when the vehicle is not moving constant force should be high, it relaxes as the vehicle moves. Scale this effect with the Constant*Force*(0-100) coefficient.
- **Damper Effect** : A damper effect makes it difficult to rotate the wheel. The effects force is calculated based on *DamperCurve* and wheel load. Scale this effect with the Damper*Force*(0-100) coefficient.
- **Airborne Effect** : This effect is played when the front wheels are not touching the ground. It has no parameters.

- **Surface Effect** : This effect creates a vibration on the wheel to give the feeling of rough road conditions. Scale this effect with the Roughness*Force*(0-100) coefficient. To activate the effect getSurfaceRoughness function is called. This method checks the surface physic material type and returns positive numbers if bad road is detected. It is recommended to write your own version of this method based on your needs.
- **Frontal/Side Collision Effect** : When a collison occurs we use the impact size and the direction of the impact to calculate a force coefficient in the OnCollisionEnter function for both frontal and side collisions. Then these coefficients are applied to the wheel sdk as an effect parameter and they are interpolated to zero in a short time.

JoystickVibration : This is a basic example that vibrates the joystick when the vehicle hits an obstacle. First it tries to find the first connected device and stores its id. When a collision occurs the gamepad motors are activated via XInputs api method. *To use this class get the XInput.NET from https://github.com/speps/XInputDotNet/releases (tested with v2017.04-2)*

GaugeUI : Shows rpm, speed, gear, steering, pedal states, traction and suspension states. The *UICanvas* object in Resources folder is added to the scene and the ui components are updated according to the values fetched from the vehicle.

MinimapTool : This helps to create a minimap on the screen. Attach this to a gameobject and automatically a camera and a line renderer is also created. To see vehicles in minimap a list of vehicles should also be provided. Check the "race" demo to see it in action.

- First a *Resources/minimapQuad* object is created for each vehicle and attached to the vehicle.
- If a spline is provided a line is generated from the spline. This will represent the track when seen from the minimap camera.
- To make it work you need to add a new layer called minimap. The parent gameobject of the line renderer and the minimapQuads must be in this layer. After that remove minimap layer from main cameras culling mask and set the minimap cameras culling mask to only minimap layer. Finally position the minimap camera above the track, set its projection type to orthographic. At this point you can play the game and see how it fills the screen, change viewport rect, projectin size etc. according to your needs and you are done. *Warning : Demo scenes are using the layer called "minimap" and if you do not have this layer, it will probably use the 8th layer which defaults to post process layer, this may not be a problem but it is recommended to add this layer if you want to use this feature.*

MobileInput : Shows basic ui controls to drive the vehicle. First a MobileUICanvas object is added to the scene and the ui components event callbacks are set. Vehicle inputs are set according to the ui buttons states. Check the "mobile_race" demo to see it in action.

OrbitCamera : A simple camera script that orbits around a given target transform. You can rotate the camera by holding down the right mouse button and zoom in/out with the mouse scroll. If the spring parameter is non zero then the yaw rotation is interpolated towards the targets yaw. If DriftMode is enabled this time the yaw rotation is interpolated towards the targets movement vector.

RewindReplay : This script is a sample rewind replay implementation(check rewind_replay.scene). You define a time range to store the history of the vehicles and when you call *rewind* function all vehicles that are sampled will animate back in time. Calling *replay* will play from past to now and calling *play* will cancel the state.

SplineTool : This forms a list of points from the child gameobjects and creates a curve object out of these points. The underliyng curve implementation is a CubicBezierCurve from Tristan Grimmer. This spline is used in various places like ai, minimap generation and ranking calculations. You can see the constructed curve in the scene if you have added at least 2 child gameobjects. Check the "race" demo to see it in action.

Statistics : Calculates some performance stats and displays them. 0-100, 0-200, 100-0 and drifting stats are calculated and displayed. If a spline is provided the last lap time is also calculated and displayed.

RuntimeSample : Attach this to an empty gameobject and it creates a simple vehicle from code, check runtime.scene to see it in action. This also includes a couple of examples to modify vehicles in runtime. Warning : Just like the tutorial scene, you may get "Input Axis is not setup" errors if you have modified the default values of the unity input axis list.

Suspension Constraint : This script can be used to prevent tire/ground or tire/vehicle penetrations. Adds a configurable joint to each wheel at start and then updates their parameters in real time so that the vehicle body stays in an acceptable transform. Warning : This component sets vehicle's rigidbody's "solverVelocityIterations" value to be 2 as minumum in order to behave better when limits are wildly violated. Check the *constraint.scene* to see it in action.

ArcadeAssists : Driving assists in the core module mostly uses a realistic approach but they are sometimes limited in efficiency. This script contains a couple of effective assists that can be used to get more controllable vehicle setups. Check "easydrive" scene for a demonstration.

- **Under Steer Recover :** This one applies external torque to the vehicle body for avoiding understeer. This is a cheap way to battle understeering but combined with the over steer recover, you can easily get arcade style driving behaviour.
- **Over Steer Recover :** Applies external torque to the vehicle body for avoding over steer.
- **Anti Roll Bar(Front/Rear) :** This is the legacy antirollbar implementation that applies external forces to avoid body roll during fast cornerings. Front antirollbar helps to reduce oversteering and rear antirollbar helps to reduce understeering. This only works for 4 wheel vehicles.
- **Drift Assist Front** : As the vehicle goes sideways this value effects front wheels friction. Positive values result better grip as the oversteer angle increases and negative values result less grip as the oversteer angle increases. Giving negative values for both front and rear wheels will make the vehicle go sideways easily but you may need to use positive values for rear if the vehicle oversteers a lot. Tune it to get the desired behaviour. This only works for 4 wheel vehicles.
- **Drift Assist Rear** : The same effect ratio for rear wheels. This only works for 4 wheel vehicles.

- **Traction Assist** : The tire model has a flaw that at low speeds the tires do not behave as expected and may loose traction. This assist helps to boost traction up to %200 at low speeds. Be aware that this modifies the lateral/longitudinal friction parameter of wheels.
- **Traction Assist Max Speed** : Traction assist is activated below this speed in kmh. The traction boost is scaled as the vehicle goes slower and reaches the maximum value when the vehicle stops.
- **Torque Splitter Ratio** : If this is activated the torque share between front and rear axles are handled dynamically depending on their rotational speeds(still based on the the initial torque shares). Faster rotating axle will receive less torque and slower one will receive more even if the initial torque share is zero. This ratio is the percentage of torque share transfer. If this is %100 and your vehicle is rwd then in the extreme case all torque can be transferred to the front axle.
- **Torque Splitter Response Time** : Insant torque share transfer might be undesirable. This is the transfer time in miliseconds. The transfer will not happen instantly but will be interpolated in this duration.
- **setCruiseSpeed method** : This is not a setting but a function that you can call to set the vehicles instant speed. This could be usefull for rolling starts.

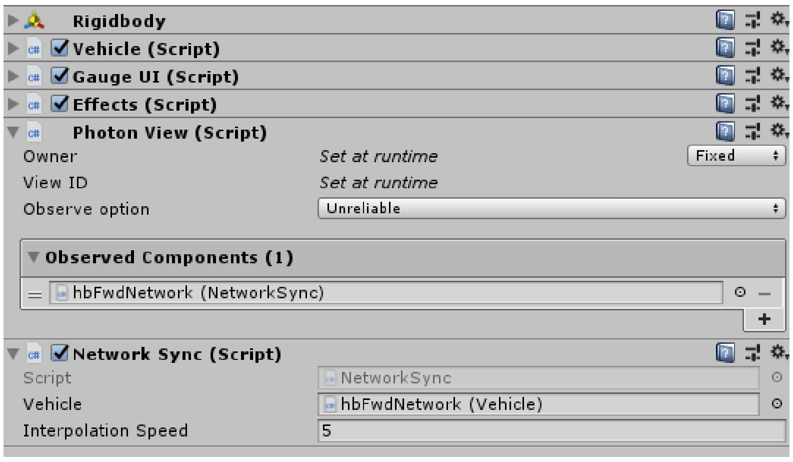
BikeHelper : A helper script that used for faking a motorcyle setup(check bikes.scene). Normally there is no way to make two wheel vehicles with fgear but we can put the left & right wheels at center and use this script to prevent falling over. This may result jittery behavior with low physics update rates so a minumum of 100hz is recommended.

BikeHelper2 : An alternate helper script for faking bikes. This behaves better in some cases.

NetworkSample : This script is only used in “multiplayer.scene”, it uses the photon unity network library to establish a connection to the photon server and instantiates a network enabled vehicle after connecting. You need to install PUN2 in order to run this sample.

- **SendRate** : Defines how many times per second PhotonNetwork should send a package. Directly effects the synchronization quality and the bandwidth usage.
- **VehiclePrefabName** : The name of the prefab that is instantiated upon connection which should be available in Resources folder. There is a sample network enabled vehicle prefab called “hbFwdNetwork” in resources folder.

NetworkSync : This script is attached to a vehicle gameobject to make it synchronized over network. There are many ways to do it, this script implements a very basic client authoritative network model. First of all this script requires a PhotonView component and you need to add this component to the list of “observed components” of PhotonView.



Rigidbody
Vehicle (Script)
Gauge UI (Script)
Effects (Script)
Photon View (Script)

Owner	*Set at runtime* Fixed
View ID	*Set at runtime*
Observe option	Unreliable

Observed Components (1)
hbFwdNetwork (NetworkSync)

Network Sync (Script)

Script	NetworkSync
Vehicle	hbFwdNetwork (Vehicle)
Interpolation Speed	5

The script starts with disabling input updates for the vehicle. Your own inputs are calculated, applied and send to the other clients. On the receiving end the last received inputs are applied to the vehicles. After setting the input options, an OrbitCamera component is attached to the vehicle and the 3d name TextMesh component is also added. Warning : This component is designed to work in "multiplayer.scene" only, some changes might be required to use in different scenes/scenarios. There are also no optimizations about bandwidth.

InterpolationSpeed : This is the interpolation speed towards latest known transform info. Higher values may result jitter and lower values increase visible latency. This is currently a constant value but a dynamic value depending on the ping time could also be useful.