Welcome to the Racing Game Starter Kit documentation!

Thank you for choosing RGSK. This document will guide you on how to use RGSK effectively to create your own racing games!

For any questions, please see Support and Community

Attention: Always back up your project before updating! Any modified assets may be overwritten during the update process.

Table Of Contents

Introduction

Overview

The Racing Game Starter Kit (RGSK) is an all-in-one solution for developing racing games in Unity. It provides you with many features that can be used to create various types of racing games. RGSK is designed with simplicity in mind so everything required to get started should be straightforward, even for beginners.

Let's get started!

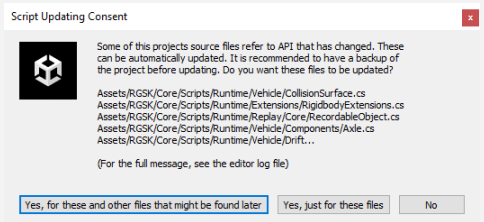
Installation

It is recommended to use Unity **2021 LTS or 2022 LTS** and to import RGSK into a new project.

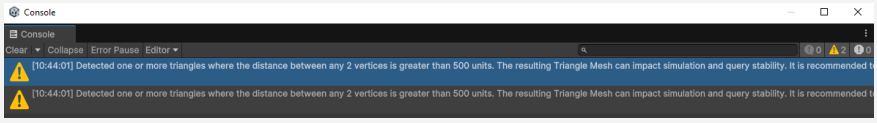
Unity 6

RGSK should work normally in Unity 6, however, it hasn't been extensively tested. If you are using Unity 6, please follow the steps below:

1. During import, click "**Yes, for these and other files...**" on the prompt:



2. These mesh warning messages can be safely ignored:



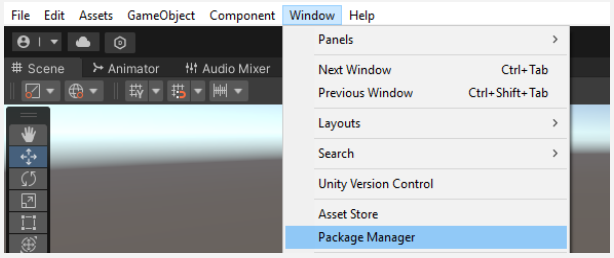
Installing Packages

Packages should automatically be installed after importing the asset, but in case they are not, please follow the guide below:

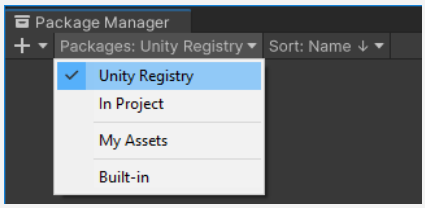
RGSK is dependent on the following packages:

- Cinemachine
- Input System
- Animation Rigging
- Post Processing
- Newtonsoft Json

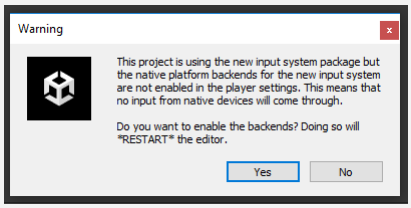
Open the Package Manager window (**Window/Package Manager**):



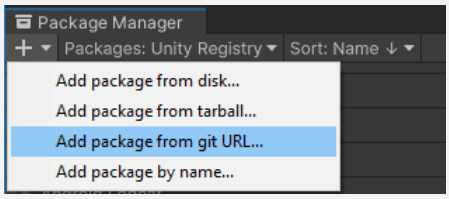
Select **Packages: Unity Registry**:

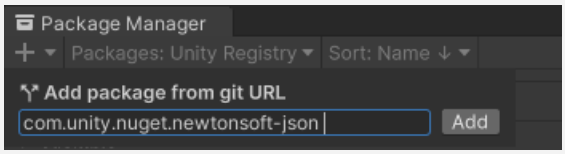


1. Find and install “**Cinemachine**” in the packages list.
2. Find and install “**Input System**” in the packages list. After installation, click “**Yes**” in the pop up window:



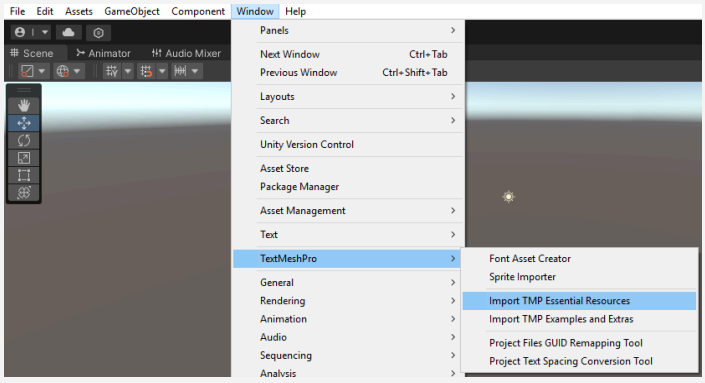
3. Find and install “**Animation Rigging**” in the packages list
4. Find and install “**Post Processing**” in the packages list
5. Click on the **+** button at the top left corner of the window and select “**Add package from git URL**”. Copy and paste **com.unity.nuget.newtonsoft-json** into the input field and click on **Add**





Add TMP Essential Resources

Select **Window/TextMeshPro/Import TMP Essential Resources**

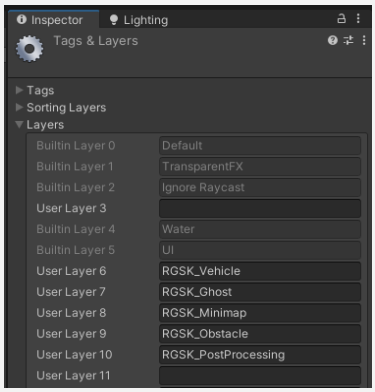


Click "**Import**" in the window that pops up.

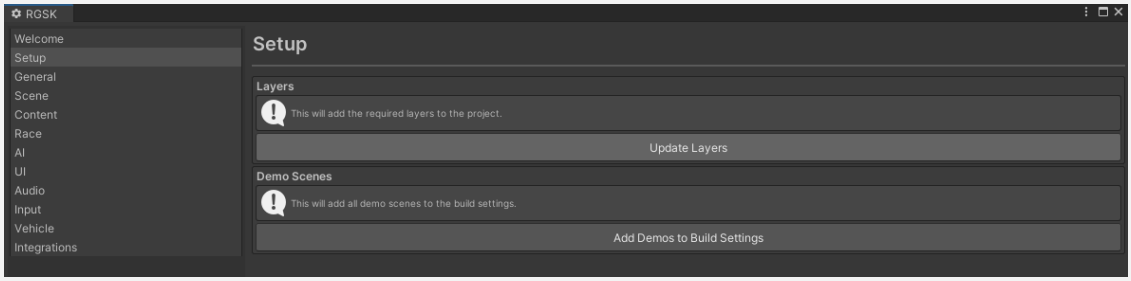
Add Layers

Layers should automatically be added after importing the asset, but in case they are not, please follow the guide below:

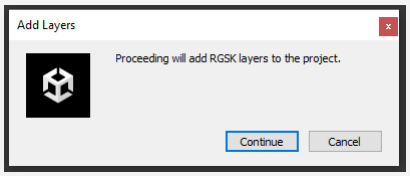
RGSK uses the following layers from **User Layer 6** to **User Layer 10**:



To update your project's layers, open the RGSK Menu (**Window/RGSK/Menu**) and under the **Setup** tab, click on the "**Update Layers**" button:



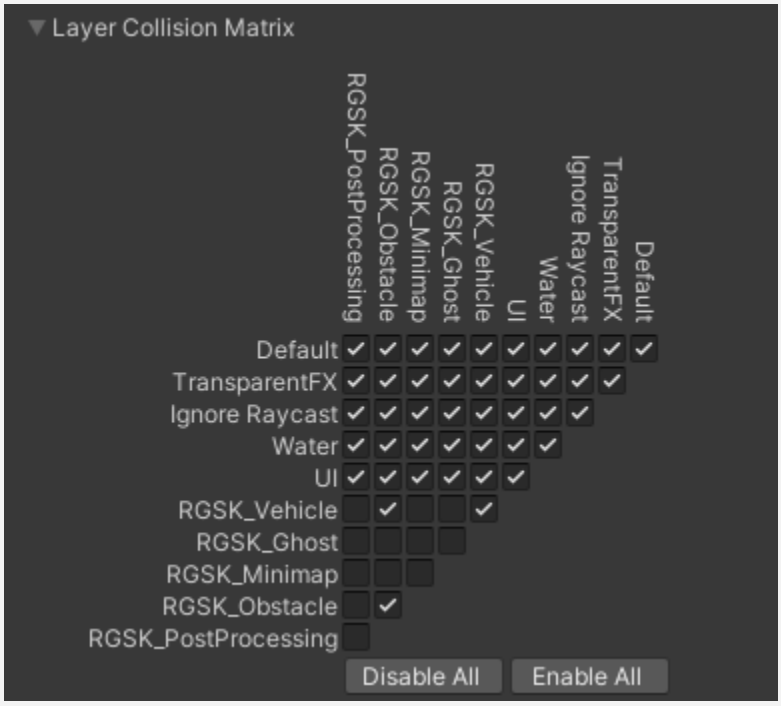
Click "**Continue**" in the prompt window:



Layer Collision Matrix

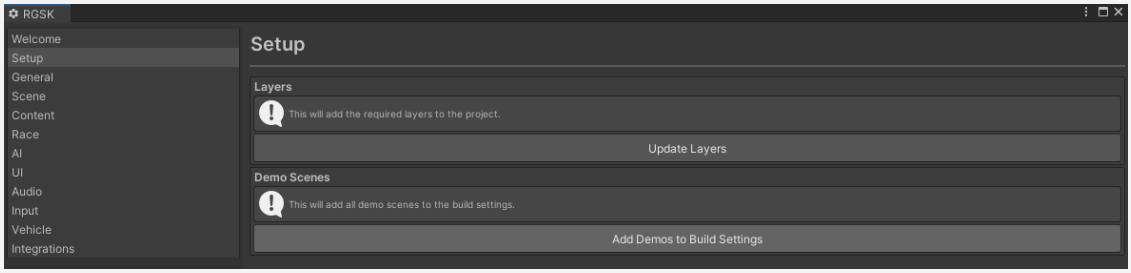
Layer collisions should automatically be setup after importing the asset, but in case they are not, please follow the guide below:

Open the project physics settings by going to **Edit/Project Settings** and selecting the "**Physics**" tab. Scroll down to the Layer Collision Matrix and ensure the "RGSK_" layer checkboxes look as follows:

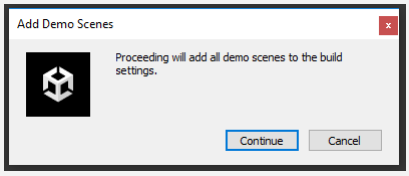


Running the Demos

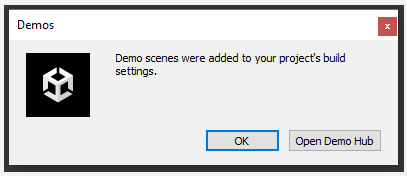
In order to run the demo scenes, they must first be added to your project's build settings. Open the RGSK Menu (**Window/RGSK/Menu**) and under the **Setup** tab, click on the **Add Demos To Build Settings** button



Click "**Continue**" in the prompt window:



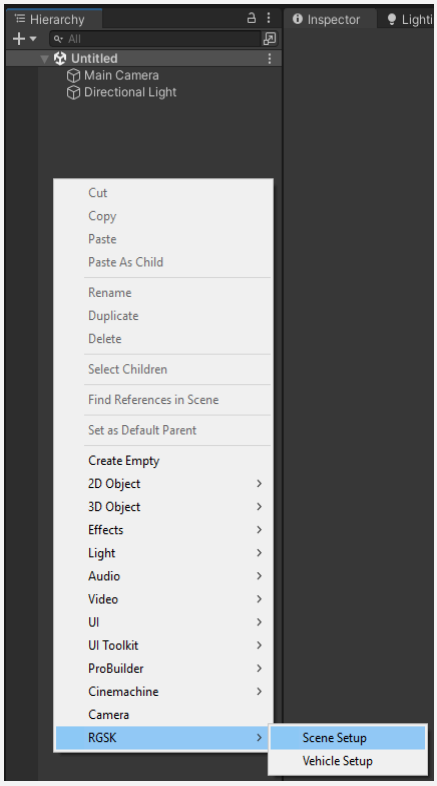
Another window should pop up, click on "**Open Demo Hub**" button and enter play mode.



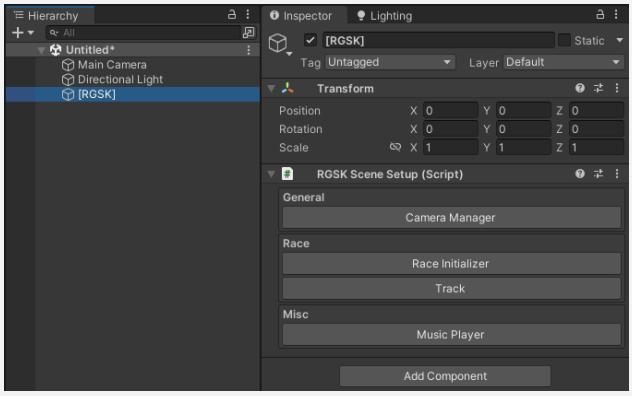
Quick Start Guide

Scene Setup

Right-click on the hierarchy and select **RGSK/Scene Setup**

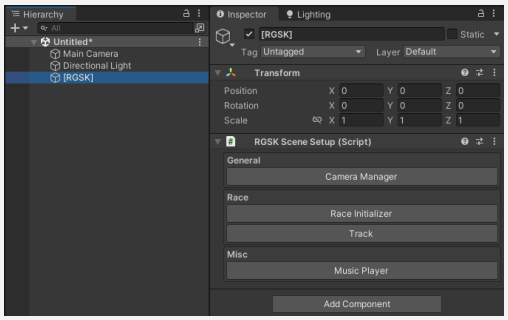


This will create a **"[RGSK]**" object. This is the **center of every scene** and handles the creation of core scene components.

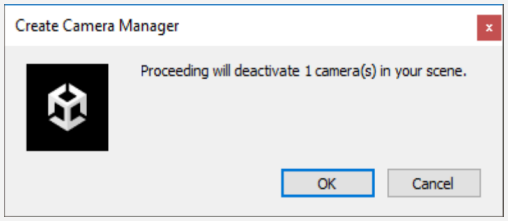


Creating the Camera Manager

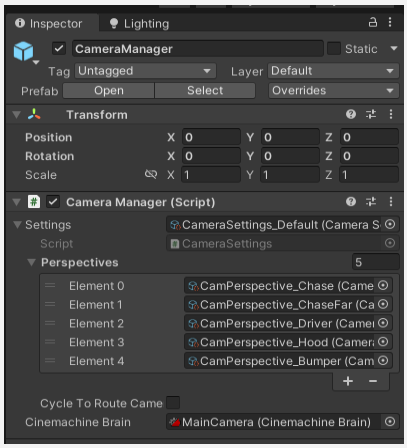
Select “[RGSK]” in the hierarchy and click on **Camera Manager**. The Camera Manager is an **essential part of every scene**.



If there are any active cameras in your scene, this window will pop up. Click on the “OK” button.

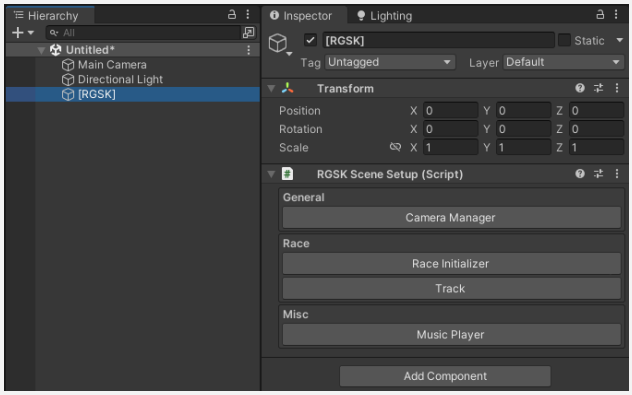


A “CameraManager” object will be created in the scene. The camera manager is responsible for creating and managing cameras in a scene. For more information on the camera manager, please go to Camera System.

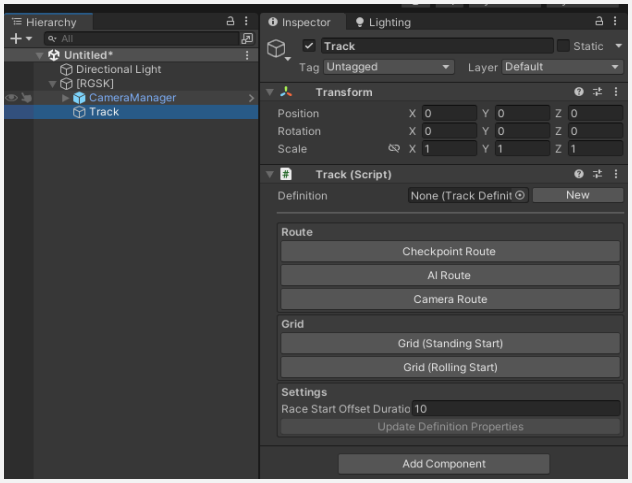


Creating a Race Track

Select “[RGSK]” in the hierarchy and click on **Track**



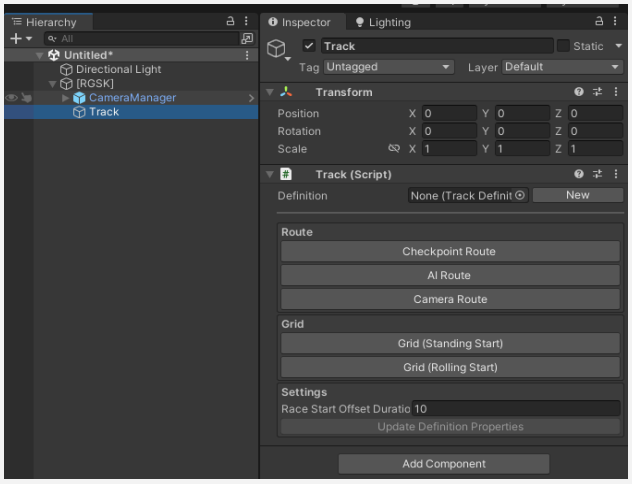
This will create a “Track” object. This is responsible for creating everything that makes up a race track.



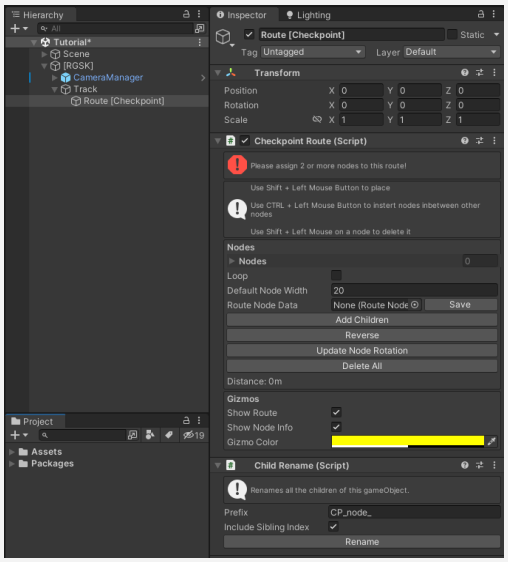
Race Start Offset Duration - how long the AI should maintain the starting offset before merging to the racing line after the race starts

Creating a Checkpoint Route

Select “Track” and click on **Checkpoint Route**.



This will create a **Checkpoint Route**. A checkpoint route defines your track’s layout from start to finish. Each checkpoint must be passed to complete a lap/race.



Default Node Width - the width of each checkpoint node. Every new node will be created with that width.

Placing Checkpoints

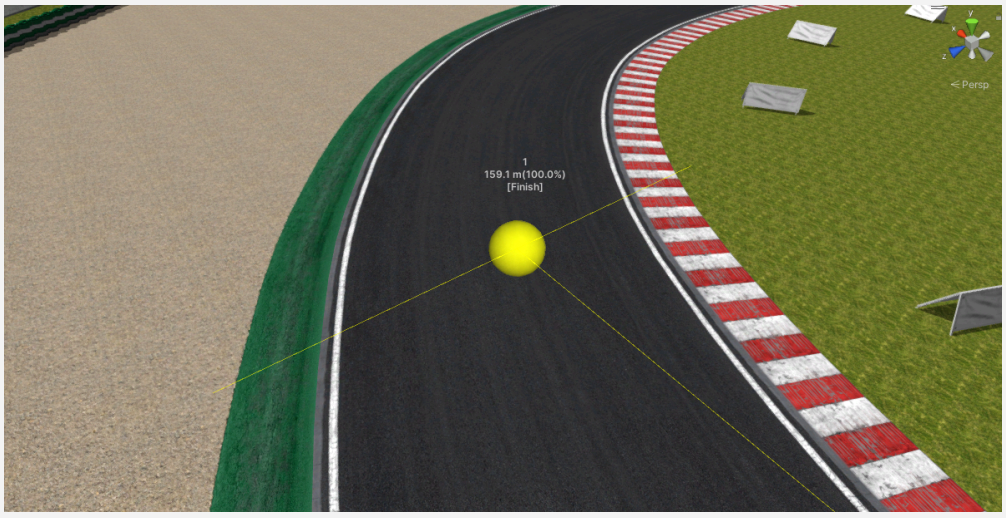
Before proceeding, please ensure that **Scene Gizmos** are enabled:

Use **Left Shift + Left Mouse Button** to place checkpoints around your track.

In a circuit race, the first checkpoint will be the start/finish line.

In a point-to-point race, the first checkpoint will be the first checkpoint racers must pass.

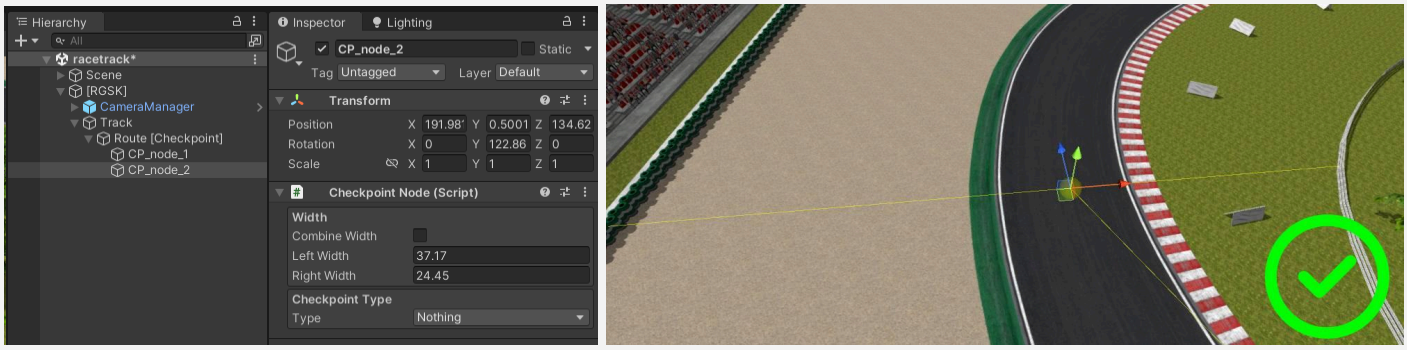
The lateral lines show the width and rotation of the checkpoint.



Ensure that the width and rotation match the desired width and the direction of your track.

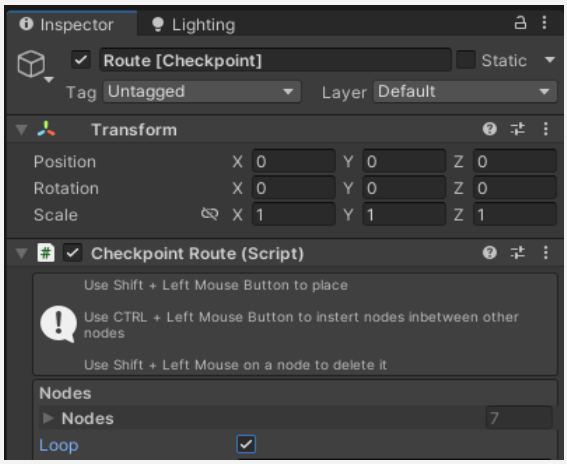
To **adjust the rotation**, click on the checkpoint node and rotate the transform to the desired angle.

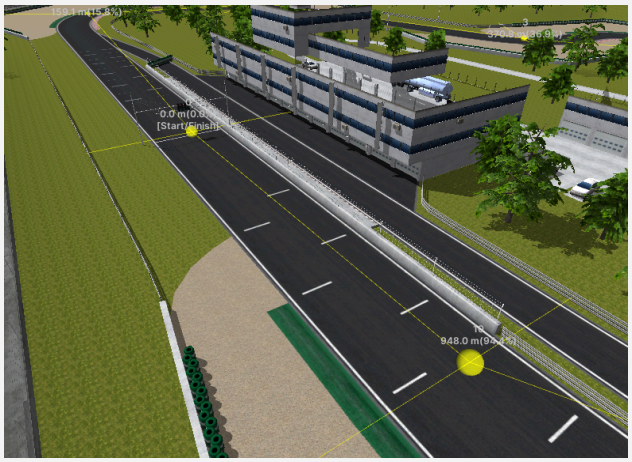
To **adjust the width**, click on the checkpoint node and edit the **Combined Width** value, or uncheck the “Combine Width” checkbox and edit the left/right widths independently.



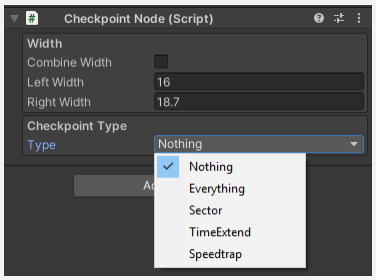
Continue to place each checkpoint along the track and adjust their width/rotation accordingly.

After placing all the checkpoints, set "**Loop**" to true if your track is a **circuit**.





Checkpoint Types



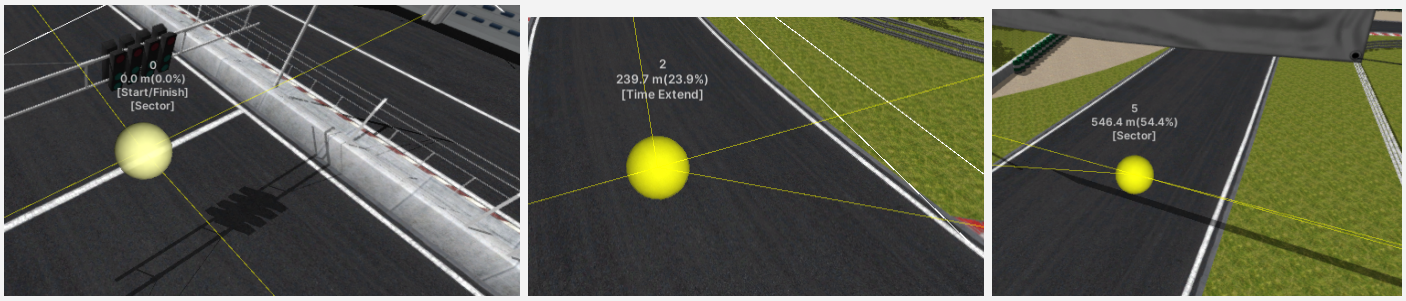
Each checkpoint can be assigned multiple types. There are 3 types of checkpoints:

Sector - records and stores your fastest lap time at this checkpoint each lap and shows you whether you were slower or faster. Typically the start/finish checkpoint and another checkpoint in the middle of the track would be sectors.

Time Extend - this is used in “Checkpoint” race types or similar where the competitor’s time is extended as they pass through checkpoints.

Speedtrap - this is used in “Speedtrap” race types or similar where the competitor's total speed is added up as they pass through checkpoints.

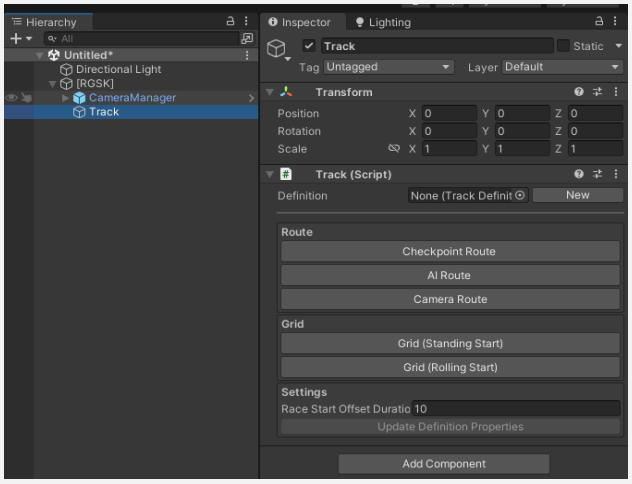
The checkpoint type will be displayed by the node info label:



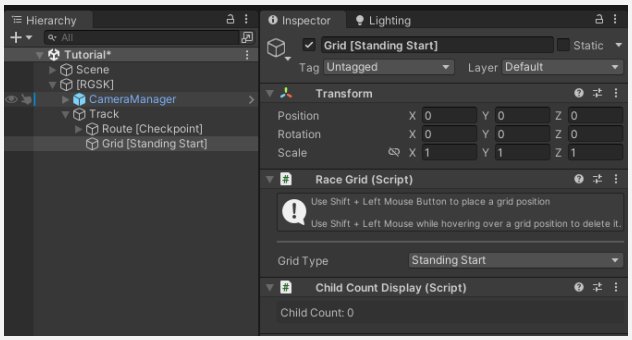
Creating a Race Grid

A Grid defines where the race competitors will spawn. There are 2 types of grids - Standing Start & Rolling Start.

Select “Track” and click on **Grid (Standing Start)**.



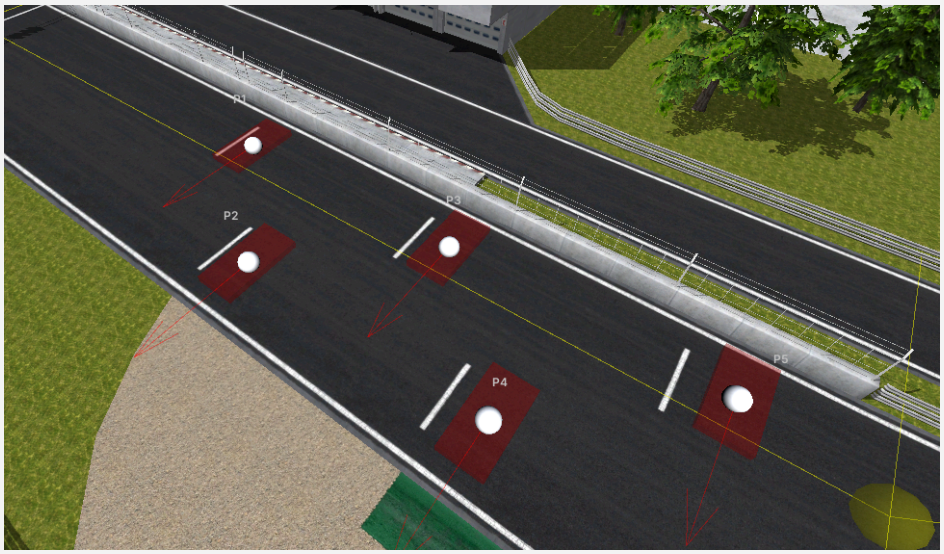
This will create a Standing Start grid.



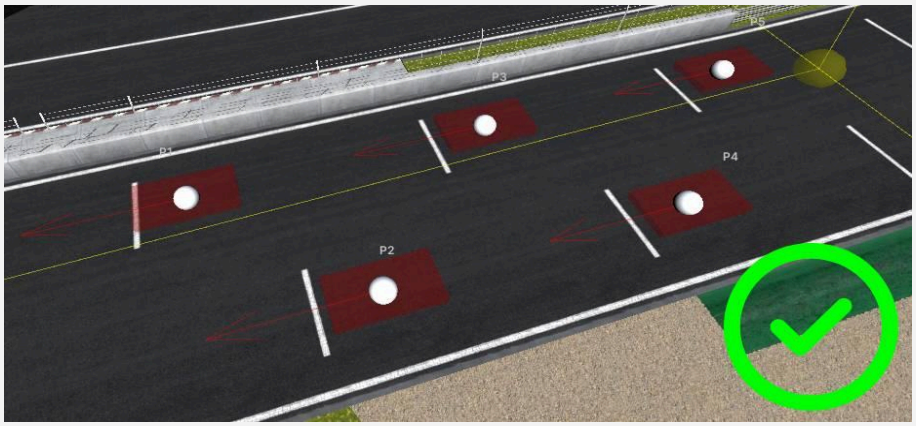
Placing Grid Positions

Before proceeding, please ensure that **Scene Gizmos** are enabled:

Use **Left Shift + Left Mouse Button** to place grid positions on your track.



The arrow points in the direction that the race competitor will spawn in. Adjust the grid position's rotation by selecting it and rotating the transform to the desired angle.

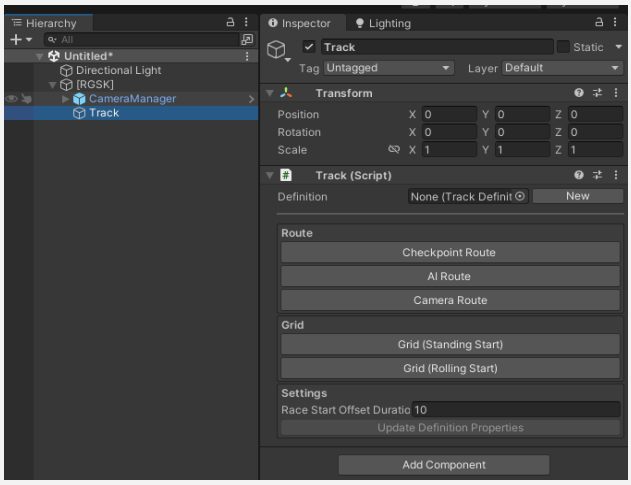


If you would like your track to also support rolling starts, repeat the steps above, but select **Grid (Rolling Start)** instead.

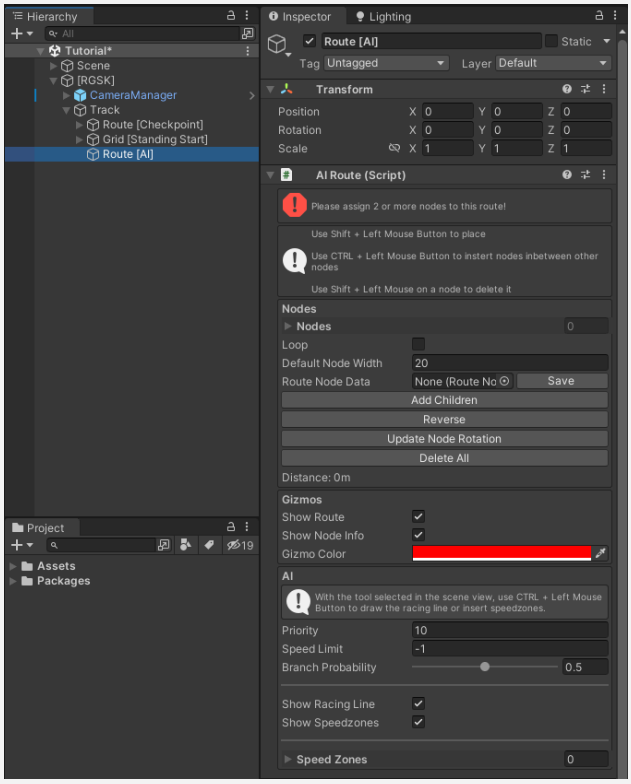
Creating an AI Route

An AI route defines the path that the AI vehicles will follow around your track.

Select “Track” and click on **AI Route**.



This will create an AI route.



Default Width - every new node will be created with that width

Priority - the route with the highest priority will be the one that the AI follow on start

Speed Limit - how fast the AI can go on this route

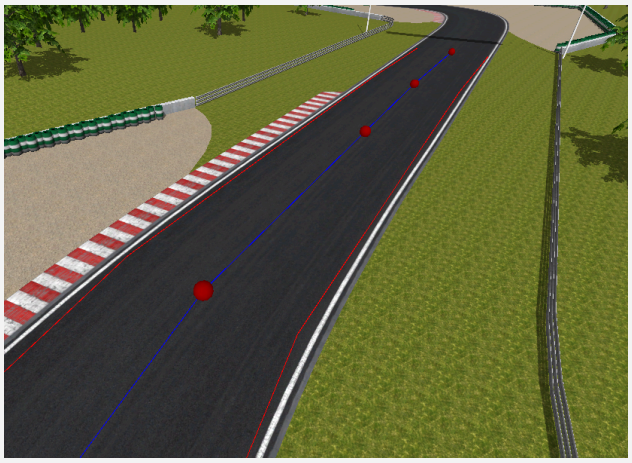
Branch Probability - the chance that the AI will choose this route at a branch

Placing AI Route Nodes

Before proceeding, please ensure that **Scene Gizmos** are enabled:

Use **Left Shift + Left Mouse Button** to place AI route nodes around your track. It doesn't matter where the first node is as the AI will look for the closest node at runtime.

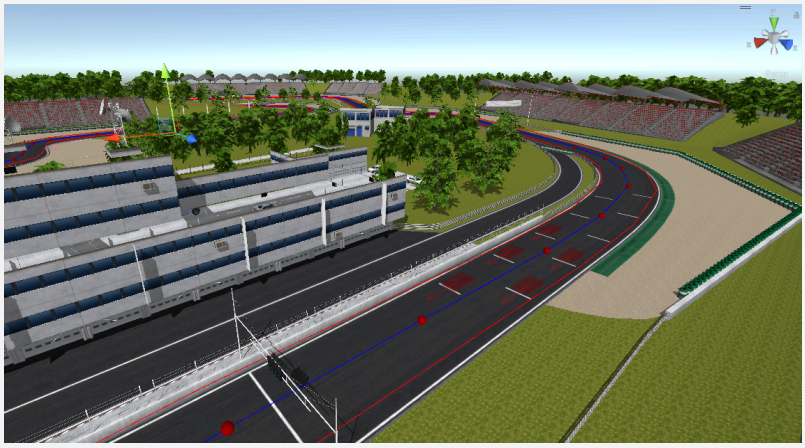
Place each node along the center of your track:

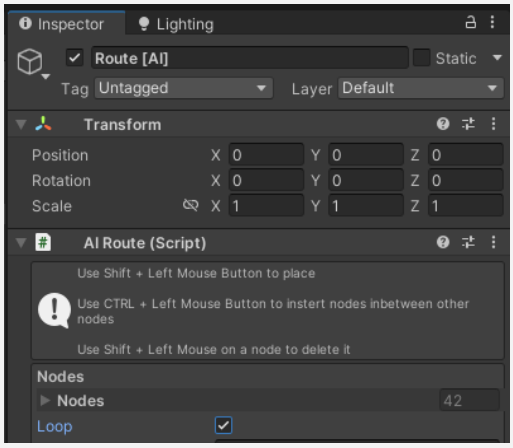


The red lines are the width of the track. The AI will keep within these bounds.

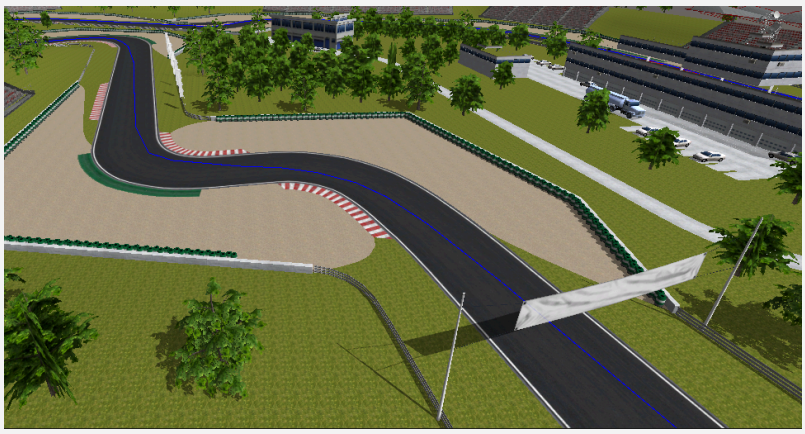
The blue line is the racing line. See Racing Line for more information.

Continue to place the nodes along your track. After placing all the nodes, set "**Loop**" to true if your track is a **circuit**.

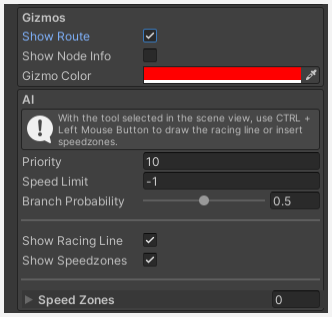




Racing Line



Ensure that “Show Route” is checked to be able to adjust the racing line:



Each node has a racing line offset that the AI will attempt to follow. To set the offset of each node, **select the “Racing Line” tool**:

With the racing line tool selected, use **Left Control + Left Mouse Button** near a node to adjust its racing line offset.

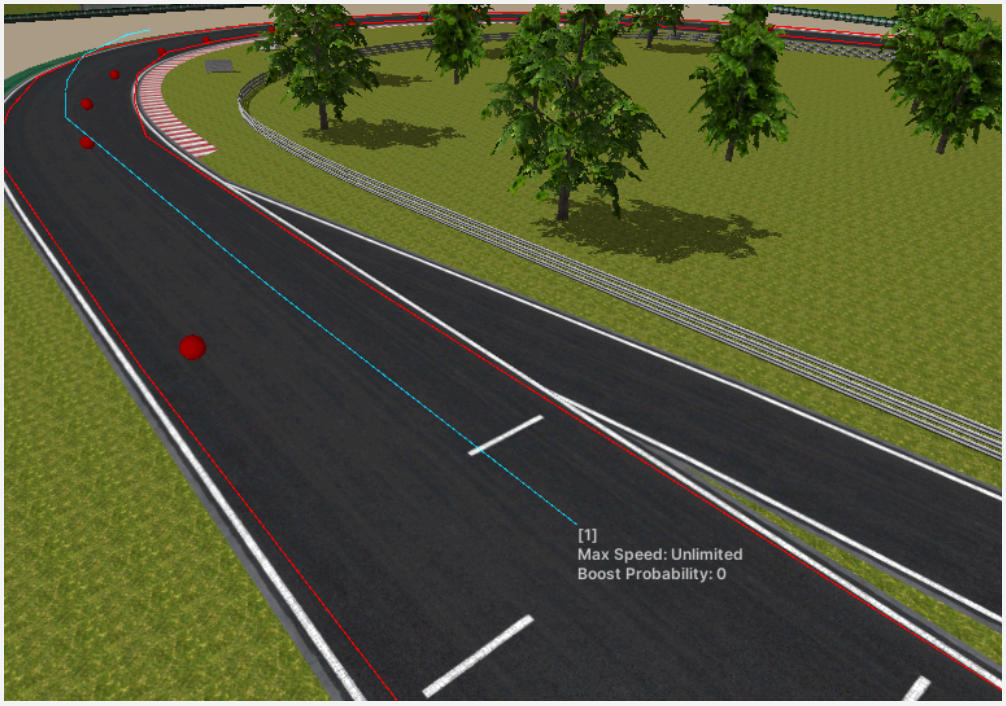
Speed Zones

Speed zones are regions of the route (defined by start and end distances) that tell the AI what speed they should be going. They are useful for creating braking zones and for nitrous zones.

If the route's **Speed Limit** is set to -1, the AI will always try to go as fast as they can until they are in a speed zone.

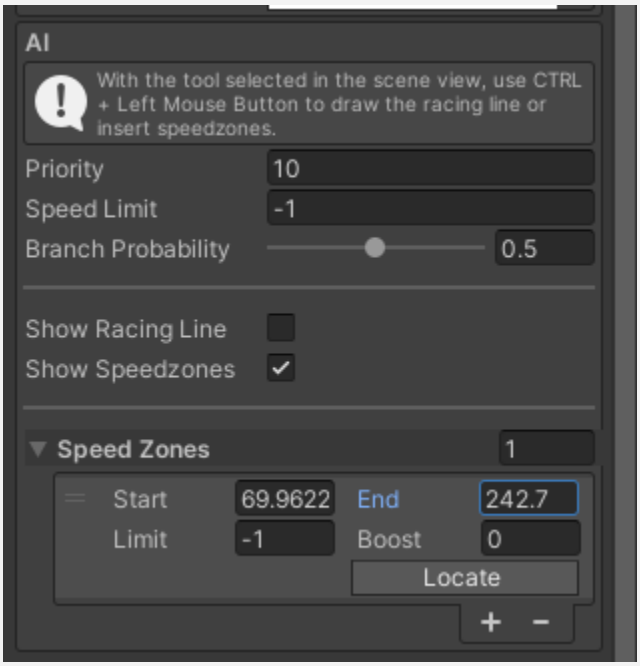
To place a new speedzone, **select the "Speedzone" tool**:

Using **Left Control + Left Mouse Button**, click somewhere on the route where you would like to create a speed zone:



This will create a new speedzone. The start distance will be at the point you clicked and the end distance will be 100 meters ahead.

Using the index of the speedzone, you can tweak it's properties from the AI route component::



Clicking and dragging left/right over the Start/End distances is a quick way to adjust the values.

Start - the distance on the route where this speedzone begins

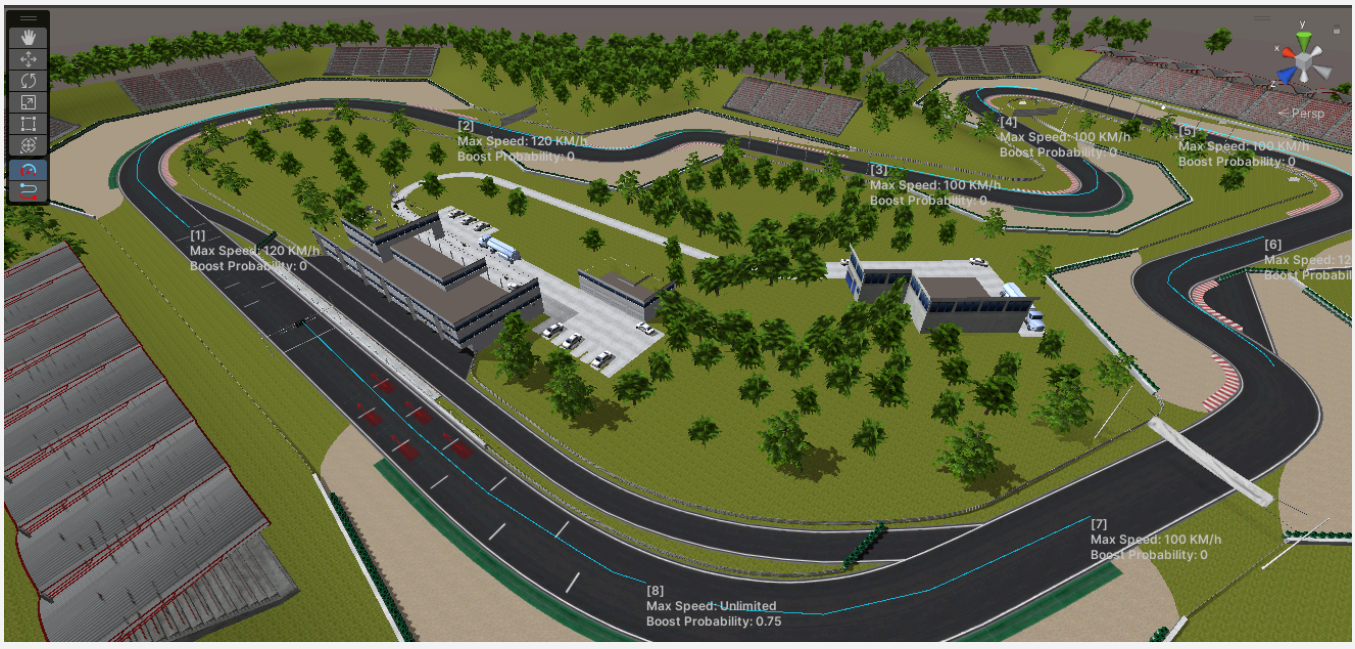
End - the distance on the route where this speedzone ends

Limit - the maximum speed that the AI can go while in this speedzone. Leave at -1 for no speed limit.

Boost - the probability from 0 to 1 that the AI will use nitrous while in this speedzone

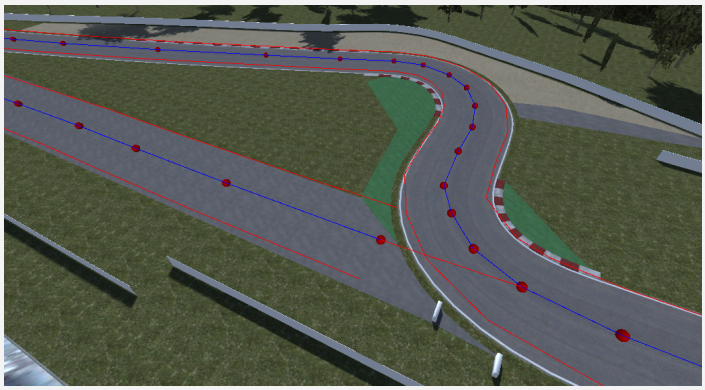
Locate - takes you to the speedzone start point using the scene view camera

Continue to insert speedzones at key areas of your track. This part requires a lot of fine tuning to get right so when testing your AI, you may need to make several changes to the speedzone values.

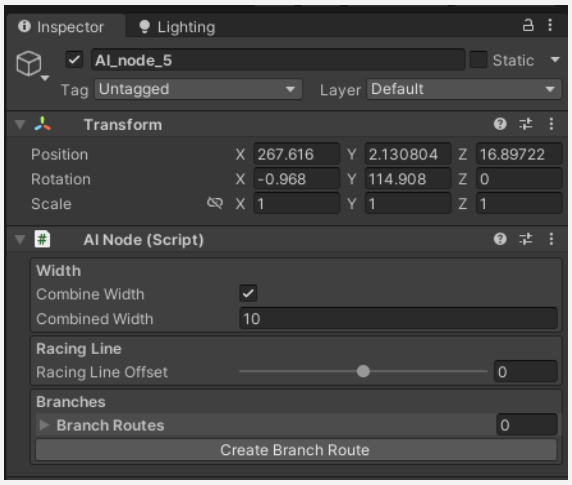


Branches

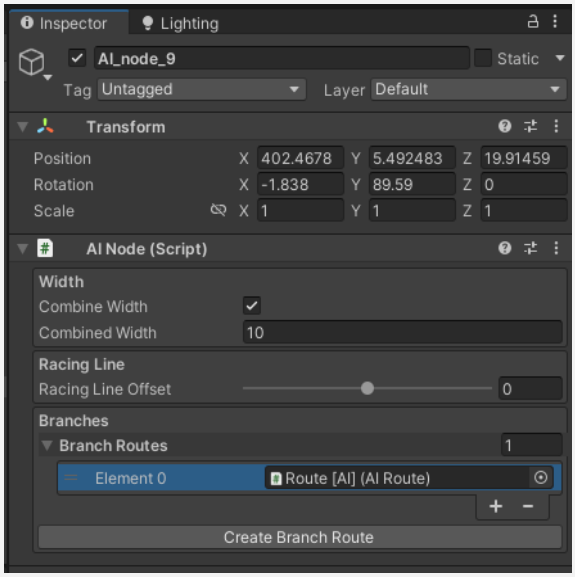
A branch is a new route that the AI can choose to follow upon reaching a node in the route.



To create a new branch, select the node that you wish the AI to branch out at and click on the **Create Branch Route** button.



This will create a new route. Place nodes for the new branch route and when you're done, select the last node of the new branch route and add the main route to the **Branch Routes** list. A line will be drawn to visualize the closest node that the AI will attempt to merge back into.

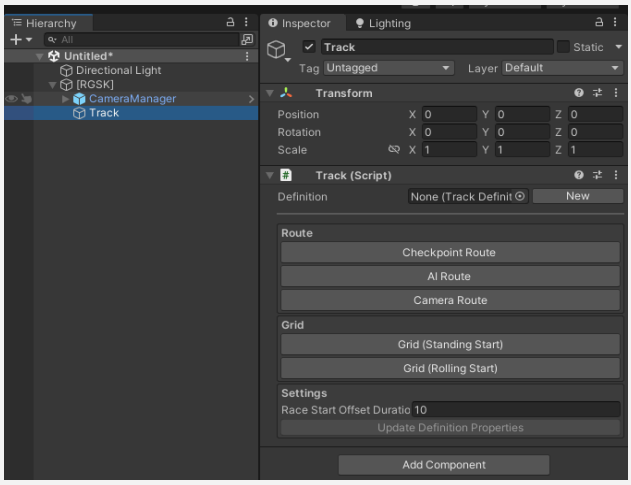


See the AI showcase demo (**Assets/RGSK/Demos/_CommonScenes/Showcase/AI.unity**) for an example.

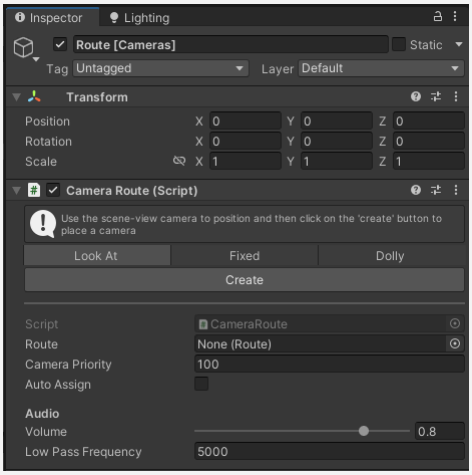
Creating Route Cameras

Route cameras are cinematic cameras placed around the track.

Select “Track” and click on **Camera Route**.



This will create a camera route.



Route - the route used to determine which camera to activate. **If this is left null, it will use worldspace coordinates**. It is recommended to assign the **AI Route** or the **Checkpoint Route** here.

Camera Priority - the priority given to the active virtual camera.

Auto Assign - whether this should be auto assigned to the camera manager. Leave this to **false** if this is part of a raceable track.

Volume - how loud the audio should be when route cameras are active.

Low Pass Frequency - the low pass to be used when route cameras are active.

Placing Route Cameras

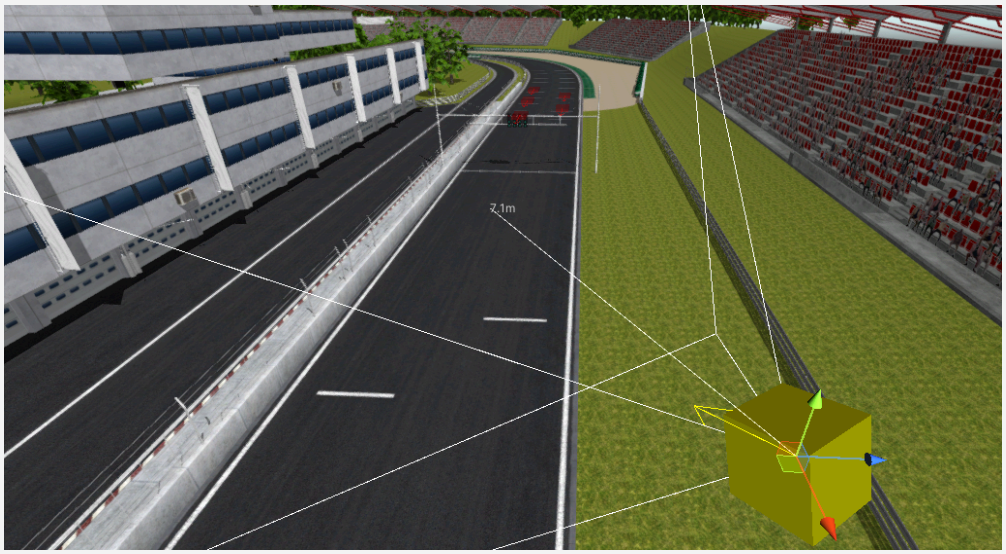
There are 3 types of route cameras:

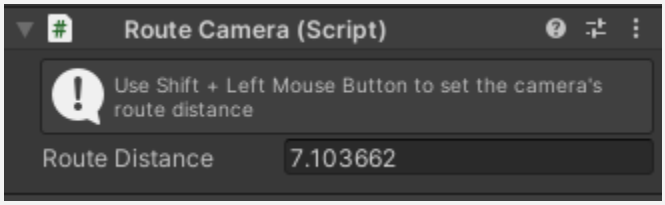
Look At - This camera looks at the target and adjusts its FOV based on the distance from the target

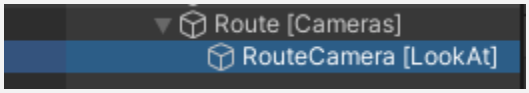
Fixed - This camera is fixed in place

Dolly - This camera looks at and moves along a predetermined route with the target

To create a route camera, select a camera type, position the scene view camera to where you would like to place the camera, and click on the "**Create**" button.





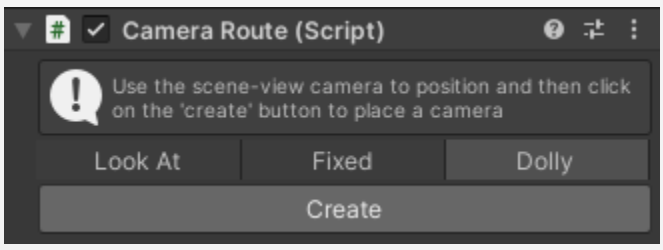


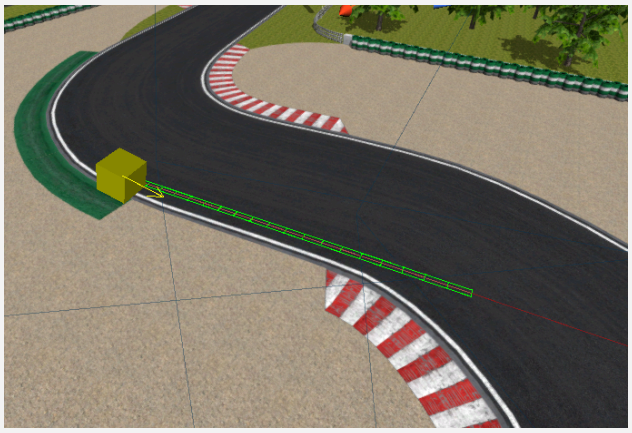
If the Route was assigned in the Camera Route component, you will see a line pointing to the route. This indicates the distance at which this camera will be activated.

To change the distance, select the camera (in this case, "Route Camera [LookAt]") and use **Left Shift + Left Mouse Button** to where where you would like the new distance to be on the route.

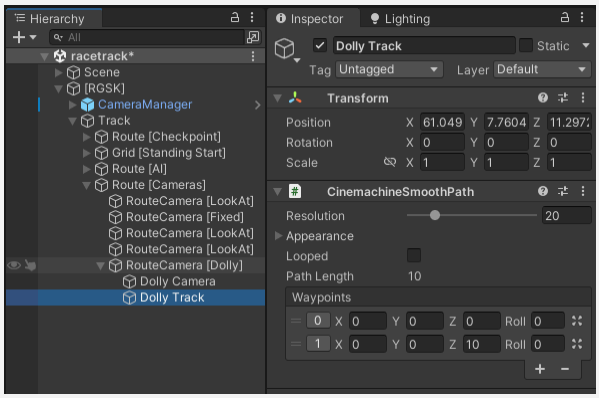
Creating a Dolly Camera

Dolly cameras look at the target while following a predetermined route. Select "**Dolly**" and click on the Create button.

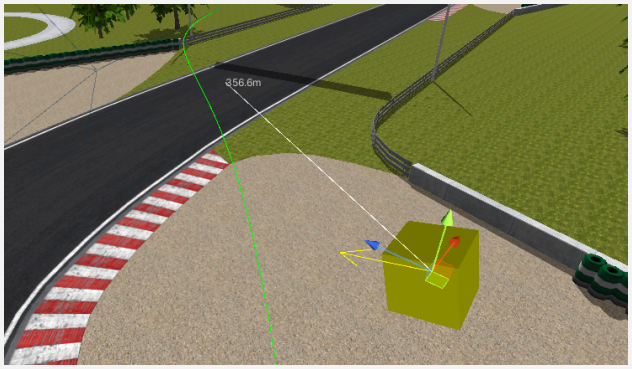




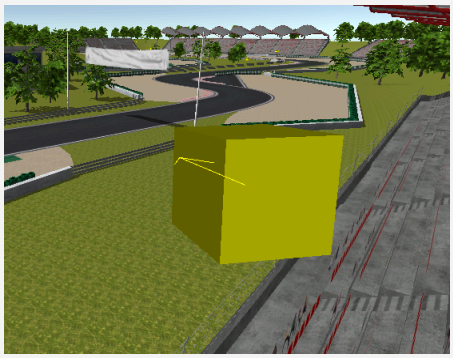
Select the “Dolly Track” and create the path. For more information, please reference the [Cinemachine Dolly Track manual]

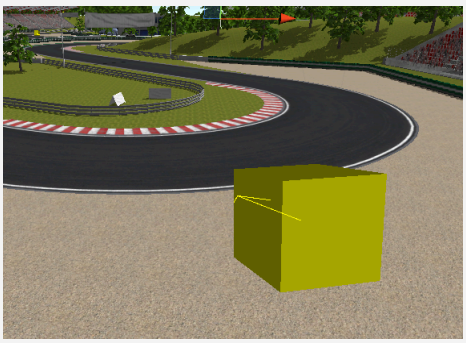


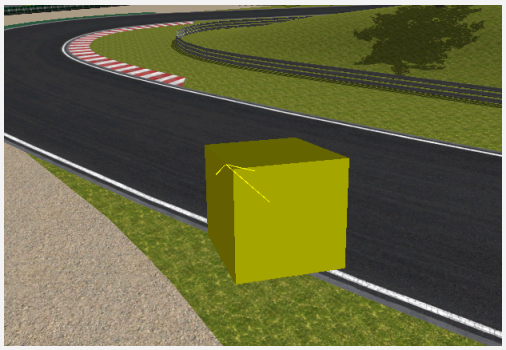
To modify the activation distance, select the “Dolly Camera” and use **Left Shift + Left Mouse Button** as explained above.



Continue to place route cameras all around your track to provide great cinematic shots. Get creative and play around with the distance values to ensure each camera is activated precisely where you would like it to be.





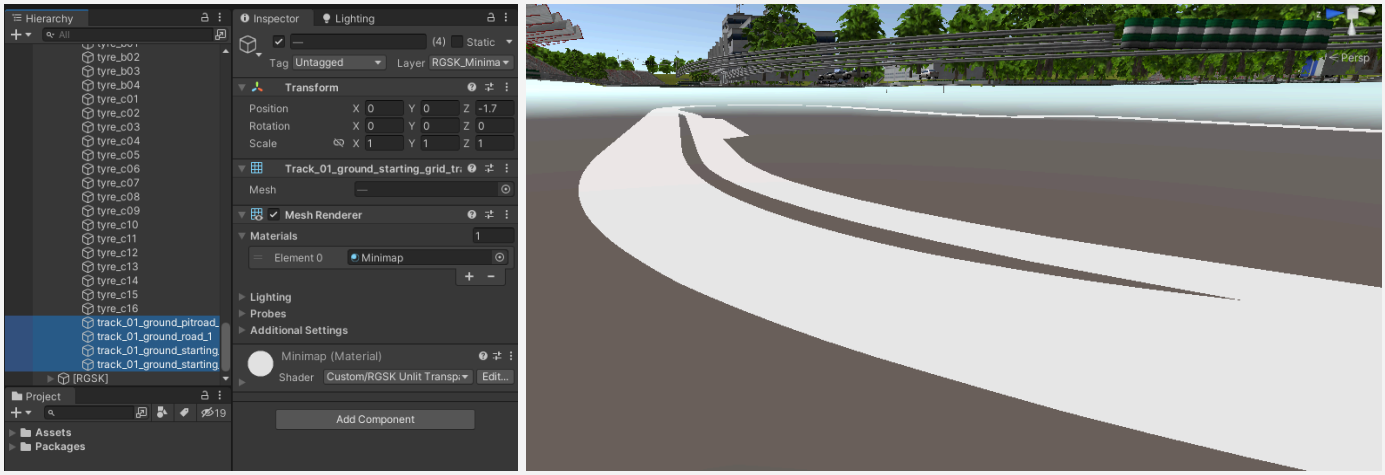


Creating a Track Minimap

To create a minimap all that is required is a mesh that is in the "**RGSK_Minimap**" layer.

One way to do this would be to:

- Duplicate your track mesh and place it below the surface of your track.
- Remove any colliders from the mesh
- Assign the "Minimap" material to the mesh (**Assets/RGSK/Demos/_CommonAssets/Art/Materials/Minimap.mat**)
- Set the objects to the "**RGSK_Minimap**" layer



Testing the Track

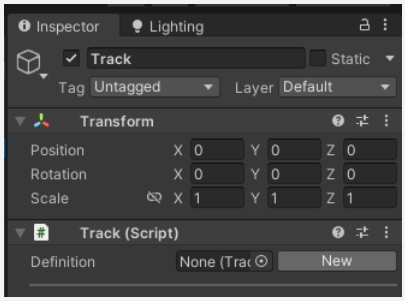
Your track is now ready to test! Please skip to Creating the Race Initializer to test your track.

Please note that not all track components are necessary for a track. The only 2 essentials are the checkpoint route and grid positions. The rest can be ignored if you do not need them.

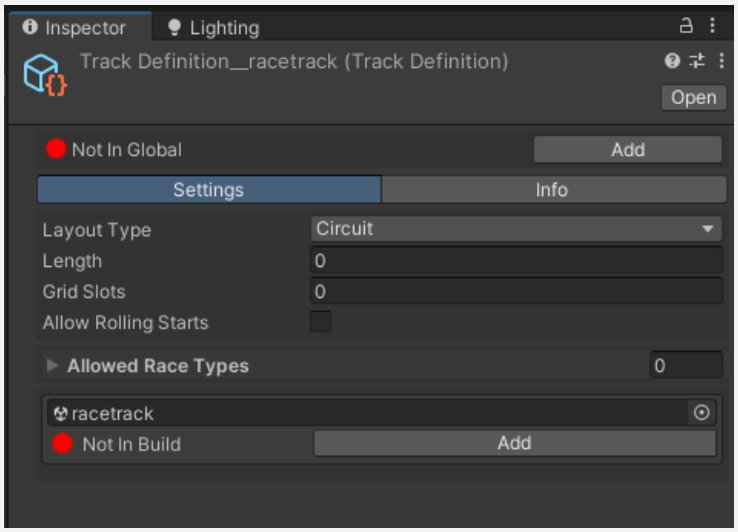
Adding the Track to Content Settings

Adding the track to the content settings will make it selectable.

In the Track component, click on the “New” button:

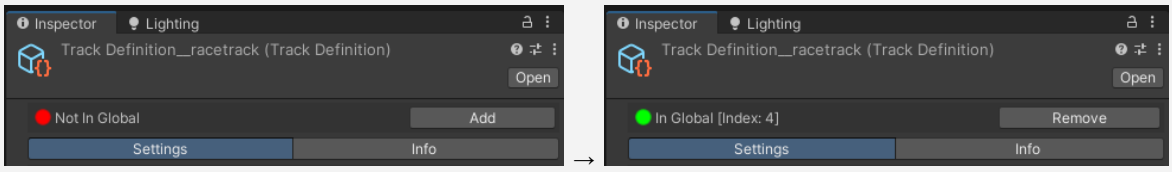


Save the asset to a path. The path must be anywhere under your “Assets” folder (e.g Assets/Data/TrackDefinitions)

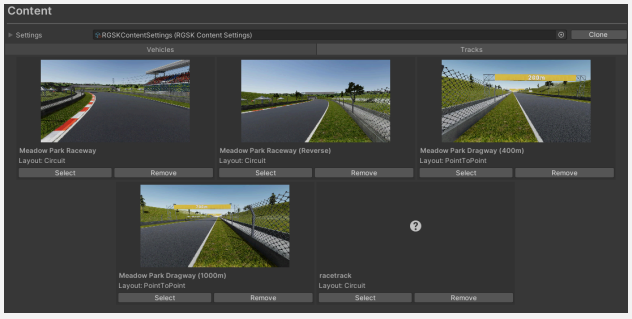


Add the scene to the Build Settings by clicking on the “**Add**” button below to the scene reference:

Finally, add the track to the Content Settings by clicking on the “**Add**” button at the top:



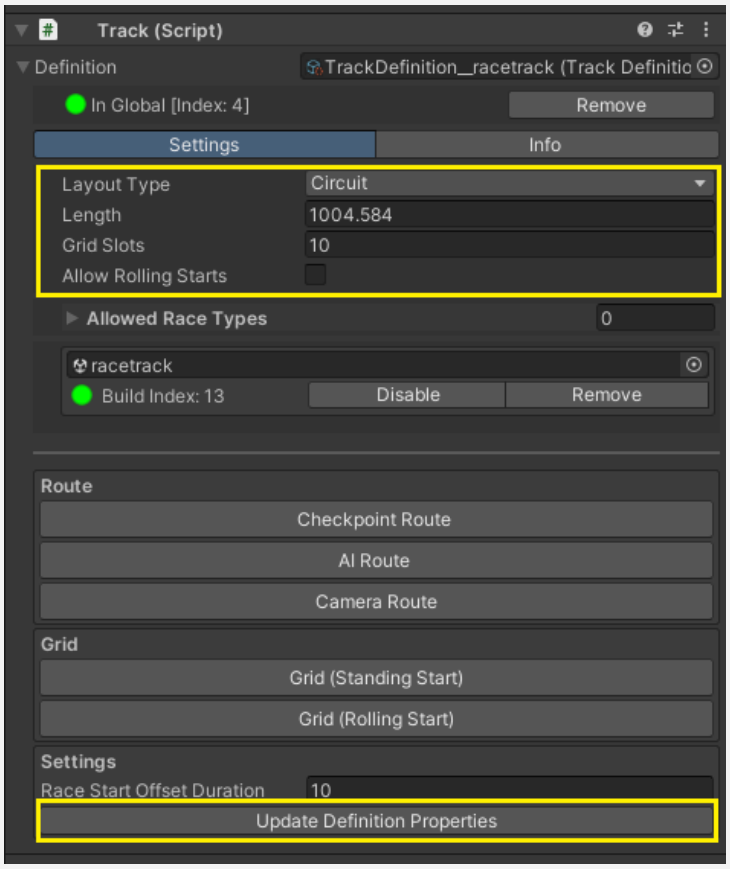
The track should now appear in the content settings:



For more information about Track Definitions, please see Track Definition.

Auto-Updating Track Properties

When a track definition is assigned, you can auto-update some of the properties by using the “**Update Definition Properties**” button:



Creating a Vehicle

Before proceeding, please ensure that your vehicle's axes are properly set. With the “Tool Handle Rotation” set to “Local”, ensure that:

Blue arrow points in the forward direction of the vehicle

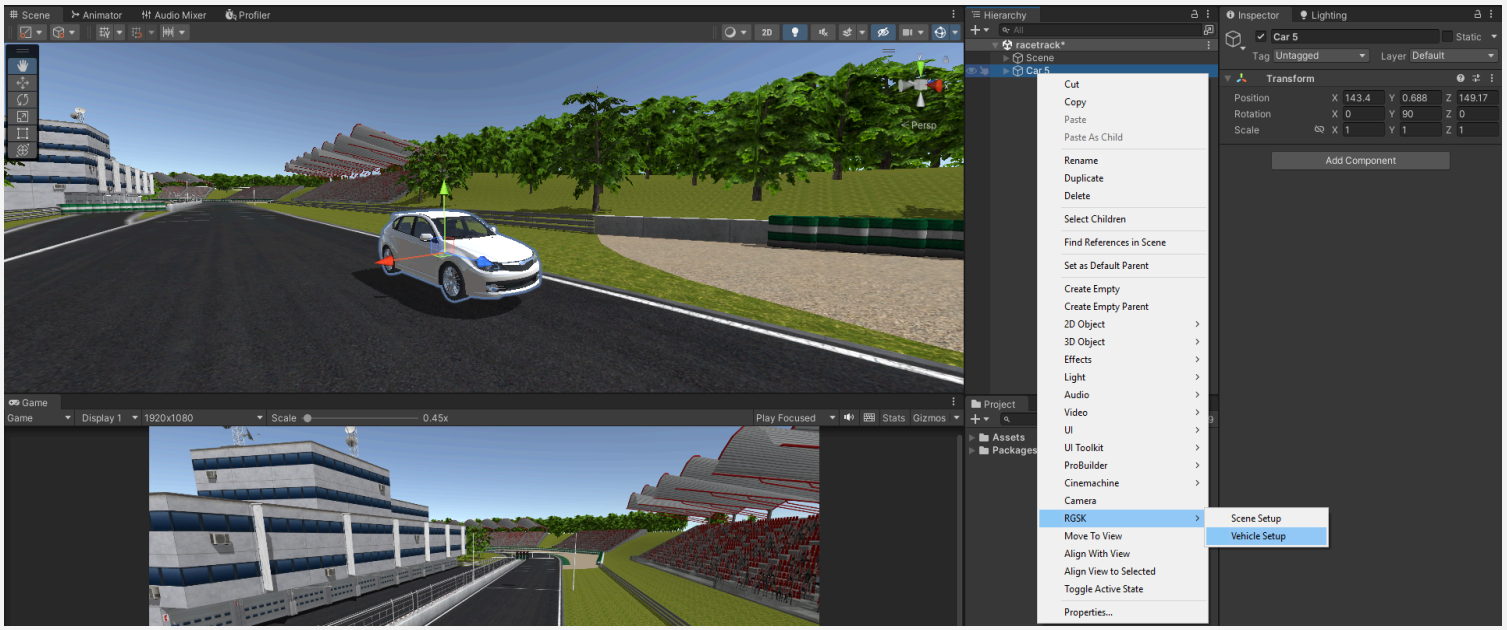
Red arrow points in the right direction of the vehicle

Green arrow points in the upward direction of the vehicle

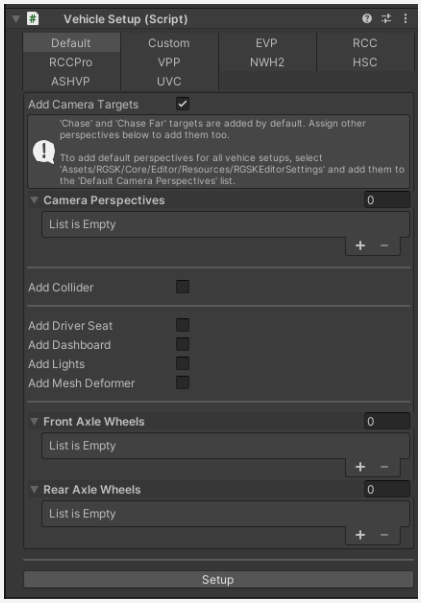
Correct vehicle/wheel axes:

Vehicle Setup

Select the vehicle in the hierarchy, right-click it, and select **RGSK/Vehicle Setup**



This will add a vehicle setup component to the vehicle:



If you are using a 3rd-party integration, select the corresponding tab and proceed as normal. Some options such as Wheels and Lights will not be available when using integrations as these should be handled by the integration itself.

Add Camera Targets - whether to add camera targets to this vehicle.

Camera Perspectives - by default, “chase” and “chase far” camera perspectives will be added. If you want to add more, assign them to the list (see Camera Perspectives)

Add Collider - this option will add a convex mesh collider to a mesh filter on the vehicle. Only select this option if you do not already have a collider for the vehicle. **If your vehicle already has a mesh collider ensure that it is set to “Convex”**

Add Dashboard - this option will add a world space UI dashboard to the vehicle. Useful if you would like a cockpit UI

Add Mesh Deformer - add a mesh deformer for collisions

Add Lights - add headlights, taillights & reverse lights to the vehicle

Front/Rear Axle Wheels - the wheel transforms that belong to the front/rear axle

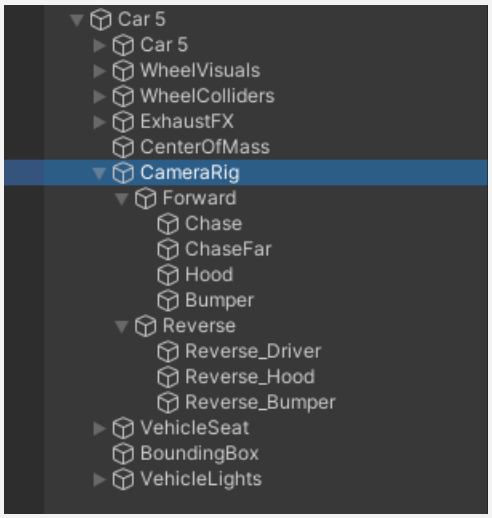
Once done, click on the “**Setup**” button

The vehicle should now have some added components to it.

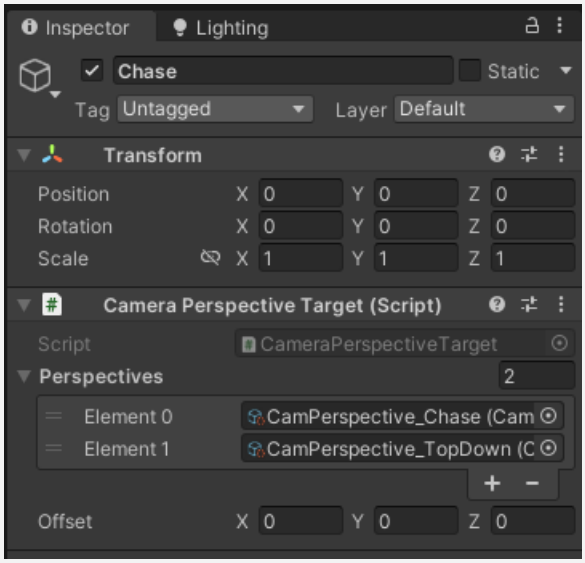
The wheel collider radius and bounding box size should be auto-calculated, however, if they are not, you will have to manually adjust the wheel collider's radius and bounding box size.

Camera Perspectives

Camera perspective targets are stored under the “CameraRig” object.



The objects under “Forward” are **Camera Perspective Targets**. The camera sets this object as its target.

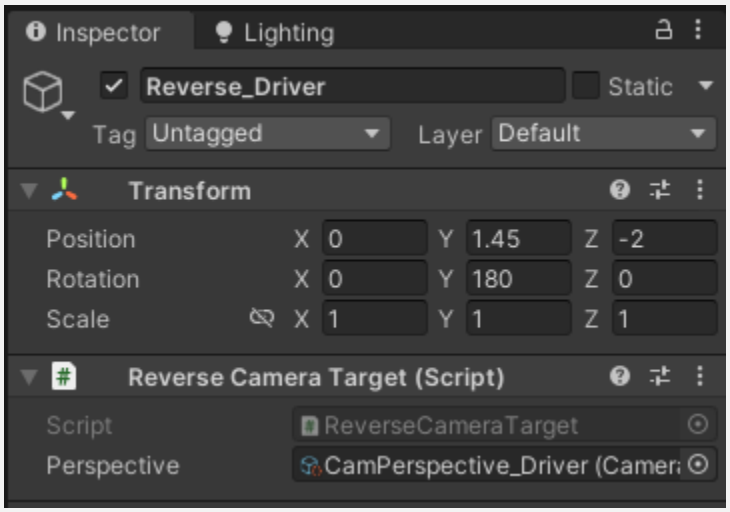


Perspectives - a list of the perspectives that will be used by this object

Offset - the offset applied to the Cinemachine virtual camera. This can be used for larger vehicles or in cases where the camera should be further (e.g. for the “ChaseFar” camera)

“Chase” and “Chase Far” don't need to be moved, but other perspectives such as “Hood”, “Driver”, “Bumper”, etc should be moved to different areas of the vehicle.

The objects under “Reverse” are **Reverse Camera Targets**. The reverse camera is set to the position of this object.

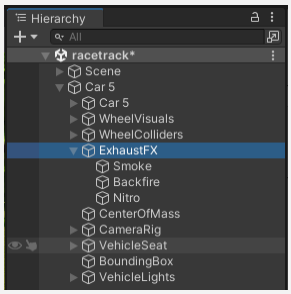


You can move them around to get a different rear view when looking back in these perspectives.

Exhaust Effects

Skip this step if you are using a 3rd-party integration

Exhaust effects are created toward the rear of the vehicle.

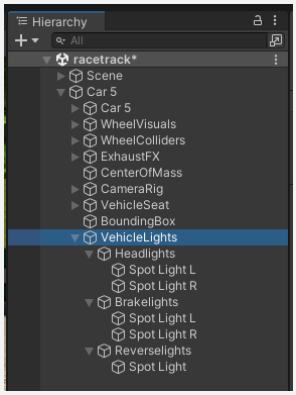


Move the “Smoke”, “Backfire” & “Nitro” particles to your vehicle’s exhaust. Duplicate the particles if your vehicle has multiple exhausts.

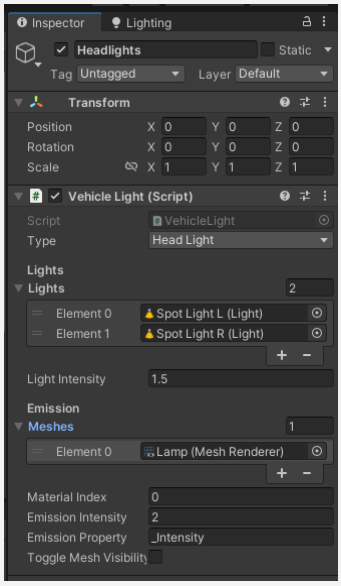
Lights

Skip this step if you are using a 3rd-party integration or if the “Add Lights” option was unchecked

Lights are created for the vehicle. Position the “Spot Lights” to where your vehicle’s headlights, taillights, and reverse lights are.



The “Headlights”, “Brakelights”, and “Reverselights” all have a **VehicleLight.cs** component attached. This script is responsible for controlling the light sources and changing mesh emission properties.



Type - the type of light. This value is auto-configured.

Lights - the list of all light sources associated with this light

Light Intensity - the light source intensity when the light is on

Meshes - the list of all meshes associated with this light

Material Index - the index of the material in the meshes that controls the emission

Emission Intensity - the value set to the emission property of the material

Emission Property - the shader's emission property. It is recommended to use the RGSK Lights Emissive shader found under **Custom/RGSK Lights Emissive**

Toggle Mesh Visibility - whether the meshes will be activated/deactivated based on whether the light is on/off

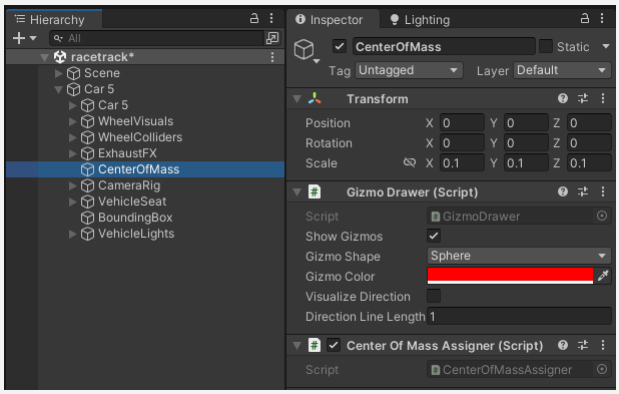
If set up correctly, the lights should be fully functional now:

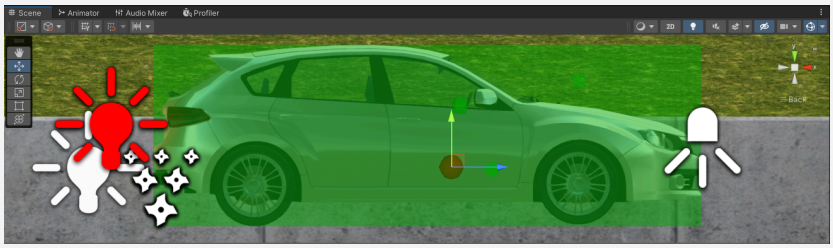
Adjusting the Center of Mass

Skip this step if you are using a 3rd-party integration

The center of mass is a crucial part of the vehicle and greatly affects the handling.

To adjust the center of mass, select the “Center of Mass” object and move it to the vehicle's center of mass position.





Tuning the vehicle

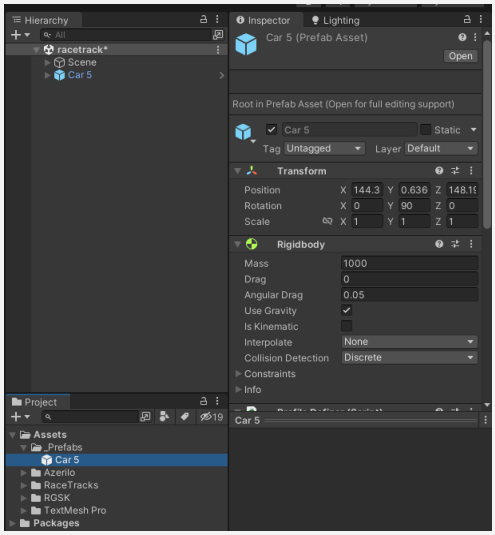
Skip this step if you are using a 3rd-party integration

All tuning happens through the **VehicleController.cs** component. Select the root of the vehicle (in this case, “Car 5”) and adjust the values to give your car a unique feel. See Vehicle Controller for more information.

Testing the Vehicle

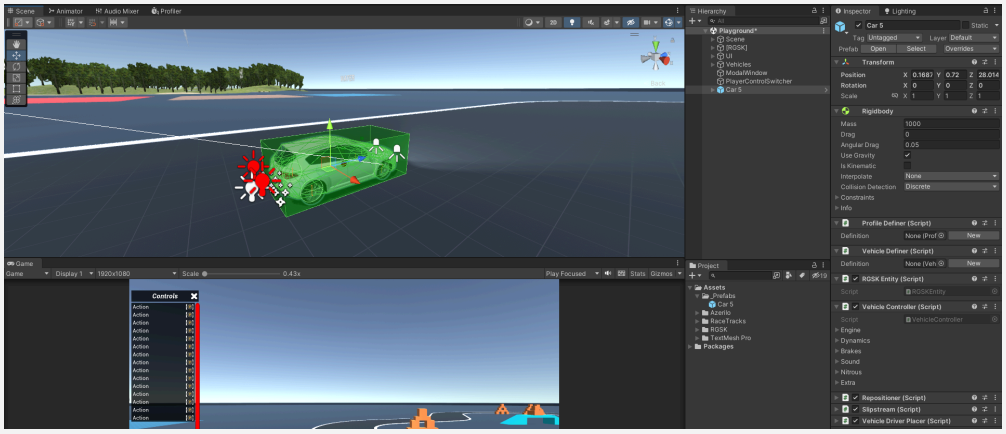
At this point, the vehicle is ready to be tested.

Make the vehicle a prefab by dragging it to the Project Tab:



Open the Playground scene (**Assets/RGSK/Demos/_CommonScenes/Showcase/Playground.unity**)

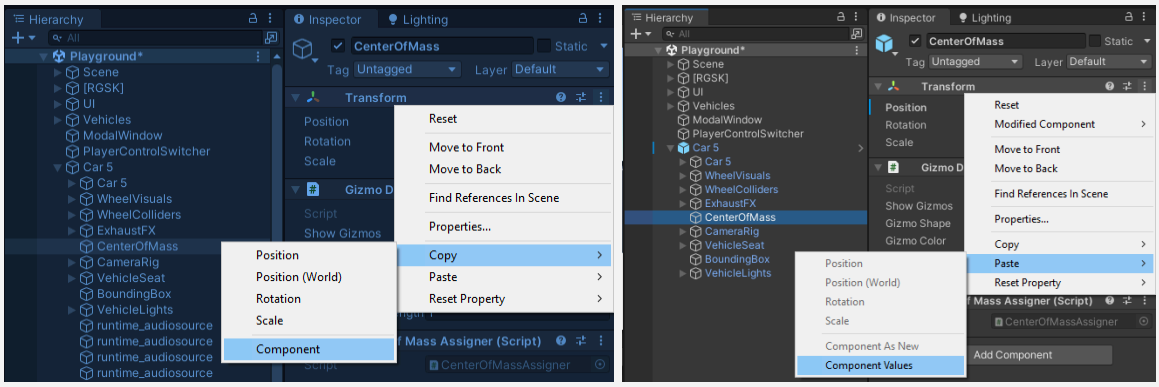
Drag and drop the prefab into the scene:



Press play, and use the **+ and - keys** to switch vehicles to your new vehicle.

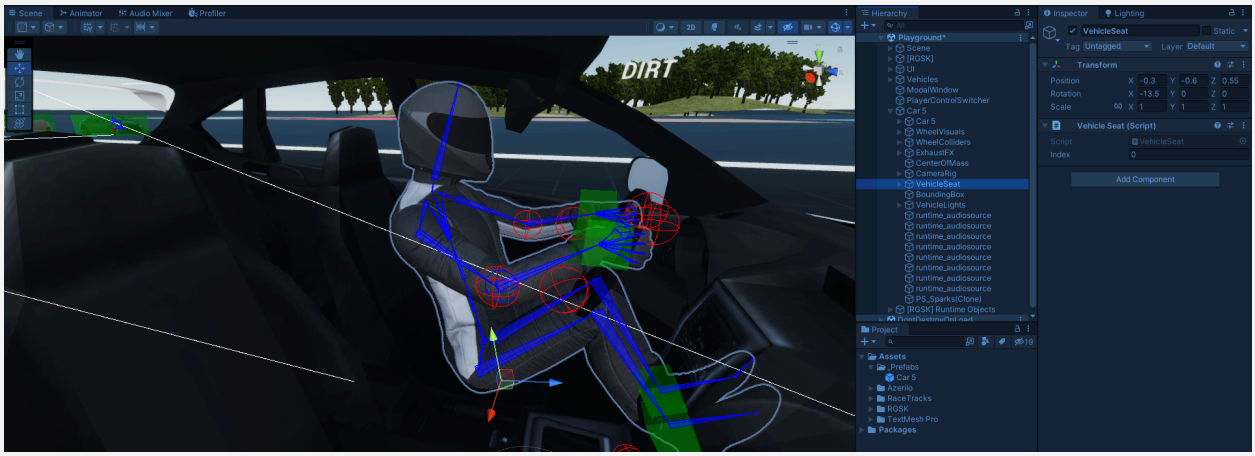
This is where you will have to take some time to tweak all the values to your liking. If the vehicle does not handle well, try moving the center of mass.

To save your values when you exit play mode, you can copy the component in play mode and paste it in edit mode:

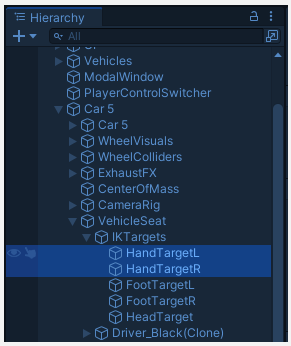


Driver

In play mode, select the “VehicleSeat” object and move it to the vehicle's driver seat. Copy/Paste the transform component as described in the technique above.



The driver's hands are controlled by the “HandTarget” IK targets:



To make the driver's hands move with the steering wheel, make “HandTargetL” and “HandTargetR” a child of your steering wheel transform. You may need to open the prefab in prefab mode to move these objects.

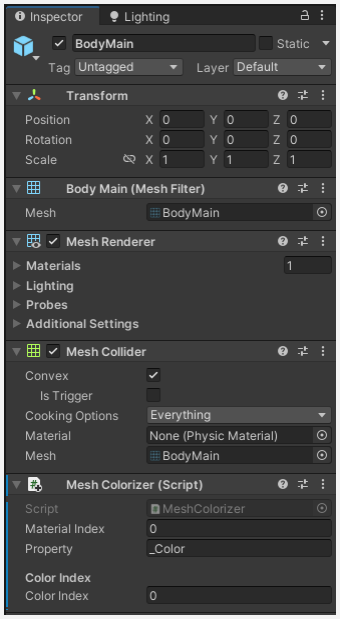
After moving the IK targets, ensure that the steering wheel is assigned in the vehicle controller:

Adjust the IK target's rotation and position to match your wheel using the copy/paste technique described above.

Color

A mesh colorizer allows the vehicle to change colors at runtime.

Add the MeshColorizer.cs component to the meshes that will support color changing at runtime:



Material Index - the index of the material to change color

Property - the shader property that controls color

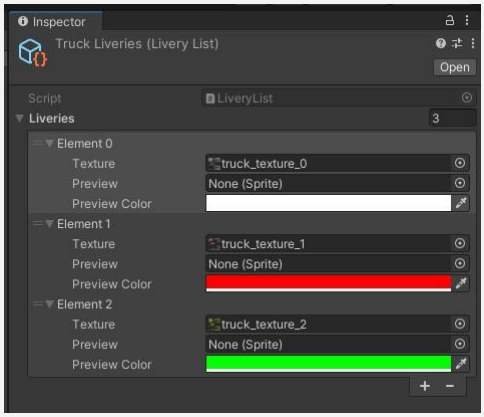
Color Index - a unique ID for this color. 0 can be used for primary colors, 1 can be used for secondary colors, 2 can be used for rim colors, etc.

Livery

A vehicle livery is a texture applied to a mesh. They can be used if you want to apply textures rather than colors to your vehicle.

To add liveries for a vehicle:

1. Create a new Livery List (**Assets/Create/RGSK/Misc/Livery List**)

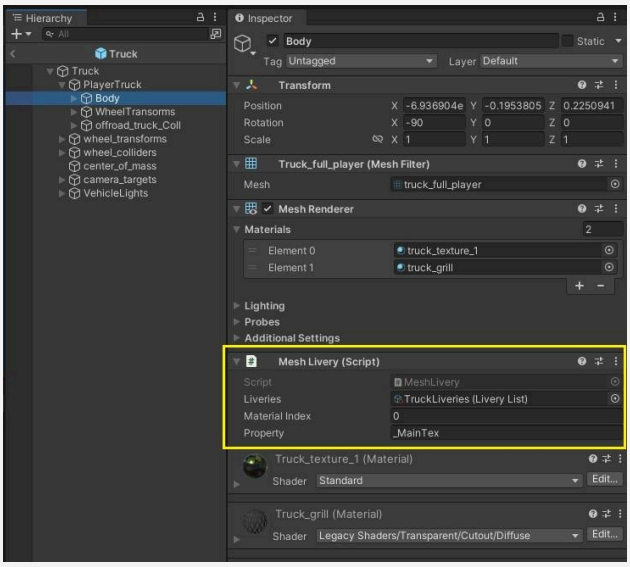


Texture - the texture applied to the mesh

Preview - the preview image used by the UI when selecting this livery

Preview Color - the preview color used by the UI when selecting this livery. Only applicable if Preview is not assigned.

2. Select the meshes on your vehicle that will apply the livery and add the MeshLivery.cs component and assign the new livery list to the "Liveries" field:

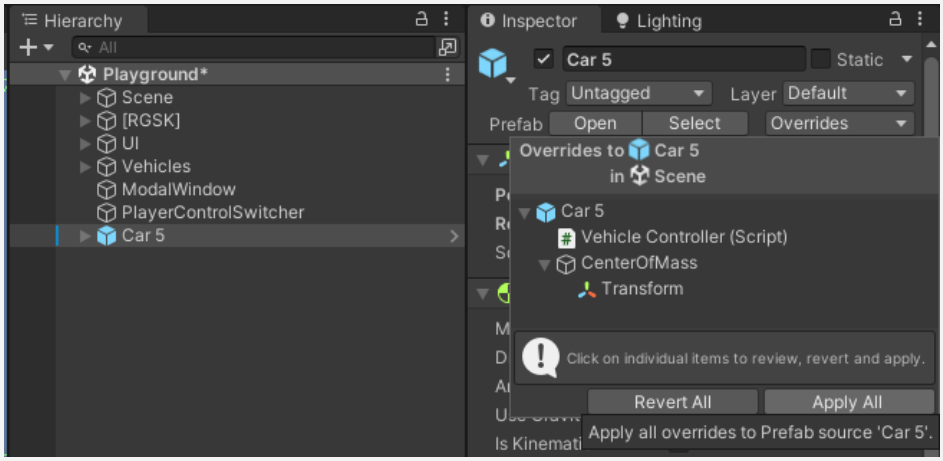


Material Index - the index of the material to change color

Property - the shader property that controls the main texture

Finalizing the Vehicle

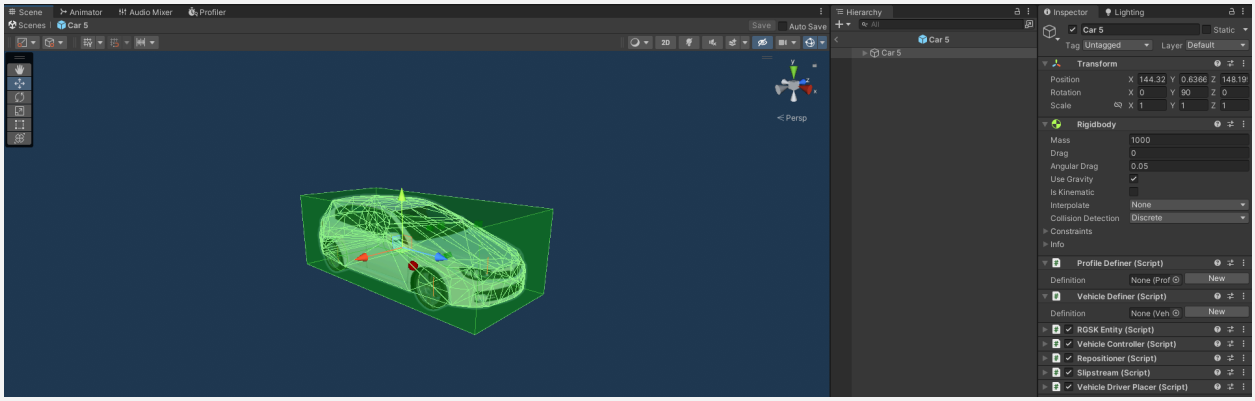
Once you are happy with your new vehicle, make sure you update the prefab by applying overrides (**Overrides/Apply All**):



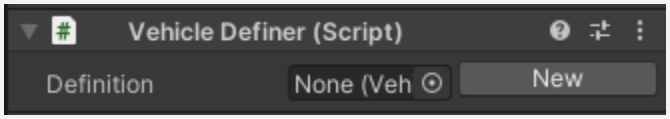
Adding the Vehicle to Content Settings

Adding the vehicle to the content settings will make it selectable.

Open the prefab in prefab mode by double-clicking the prefab in the project view.

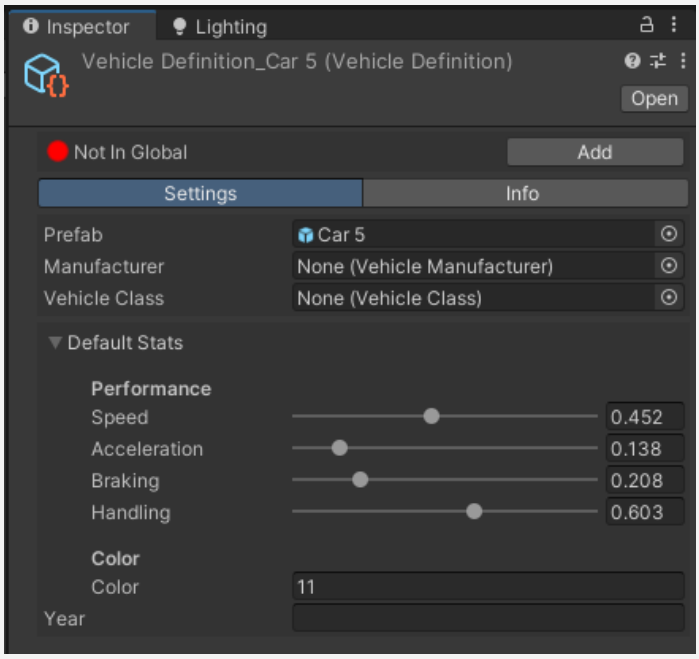


Create a new Vehicle Definition by clicking on the “New” button in the Vehicle Definer component:

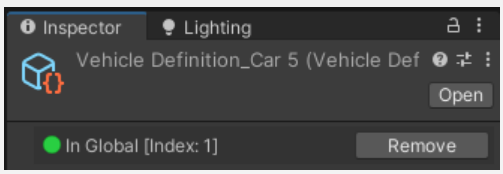


Save the asset to a path. The path must be anywhere under your “Assets” folder (e.g Assets/Data/VehicleDefinitions)

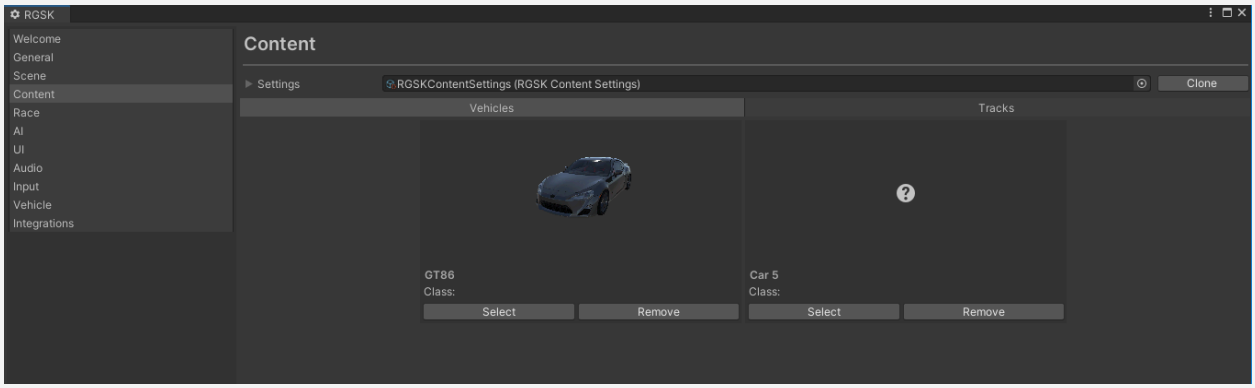
Select the newly created Vehicle Definition file and drag and drop the vehicle prefab into the “Prefab” field. Fill in the other fields to your preference. For more information about the vehicle definition, see Vehicle Definition.



Lastly, click on the "**Add**” button at the top to add this vehicle to the global content settings:



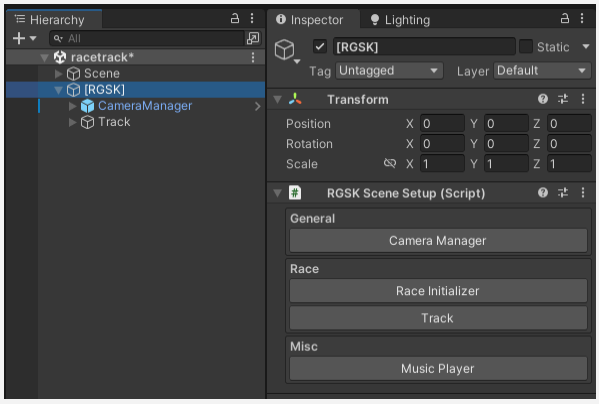
The vehicle should now appear in the content settings:



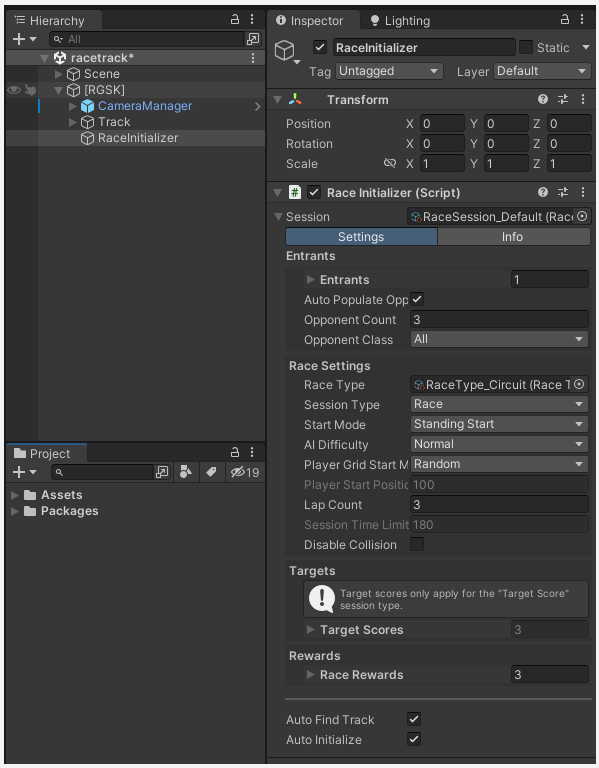
To add an item thumbnail for it, assign a sprite to the “Preview Photo”. This can quickly be done using the UI Thumbnail Creator at **Assets/RGSK/Demos/_CommonScenes/Misc/UIThumbnailCreator.unity**

Creating the Race Initializer

Select “[RGSK]” in the hierarchy and click on **Race Initializer**



This will create a Race Initializer. This component is responsible for initializing a race. Please see Race Initializer for more information.



By default, it will be created with the default RGSK session assigned, but you can either assign your own session (see Race Sessions for more information) or modify the existing settings.

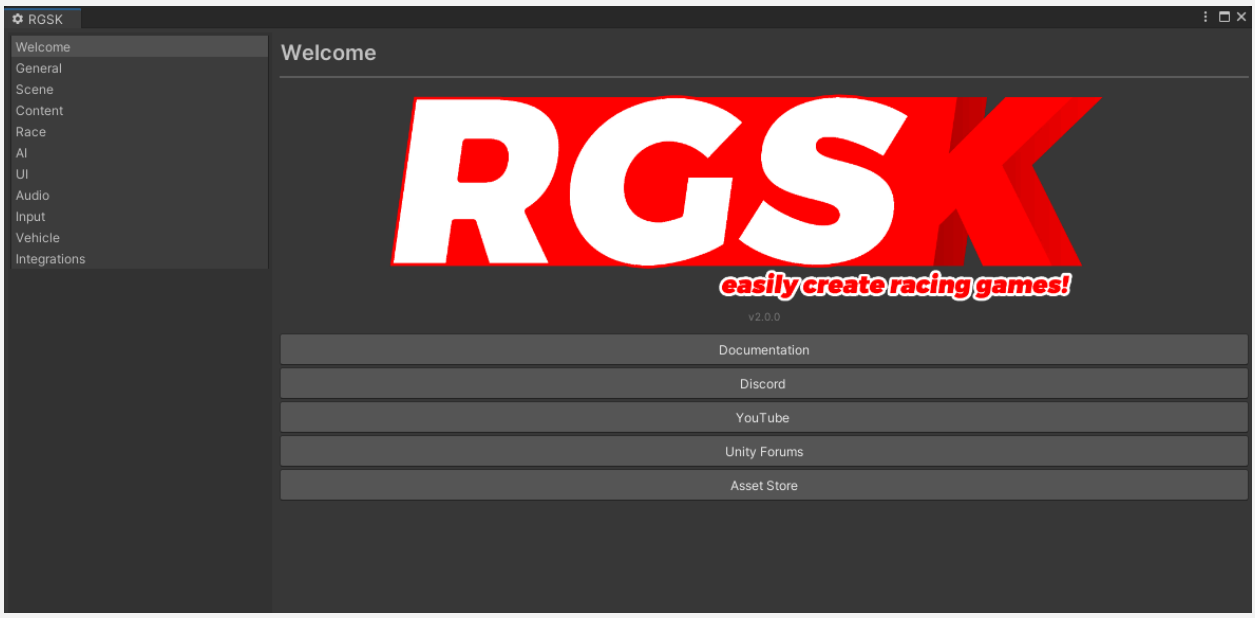
If your track has already been set up, click on the play button to begin testing.

RGSK Menu

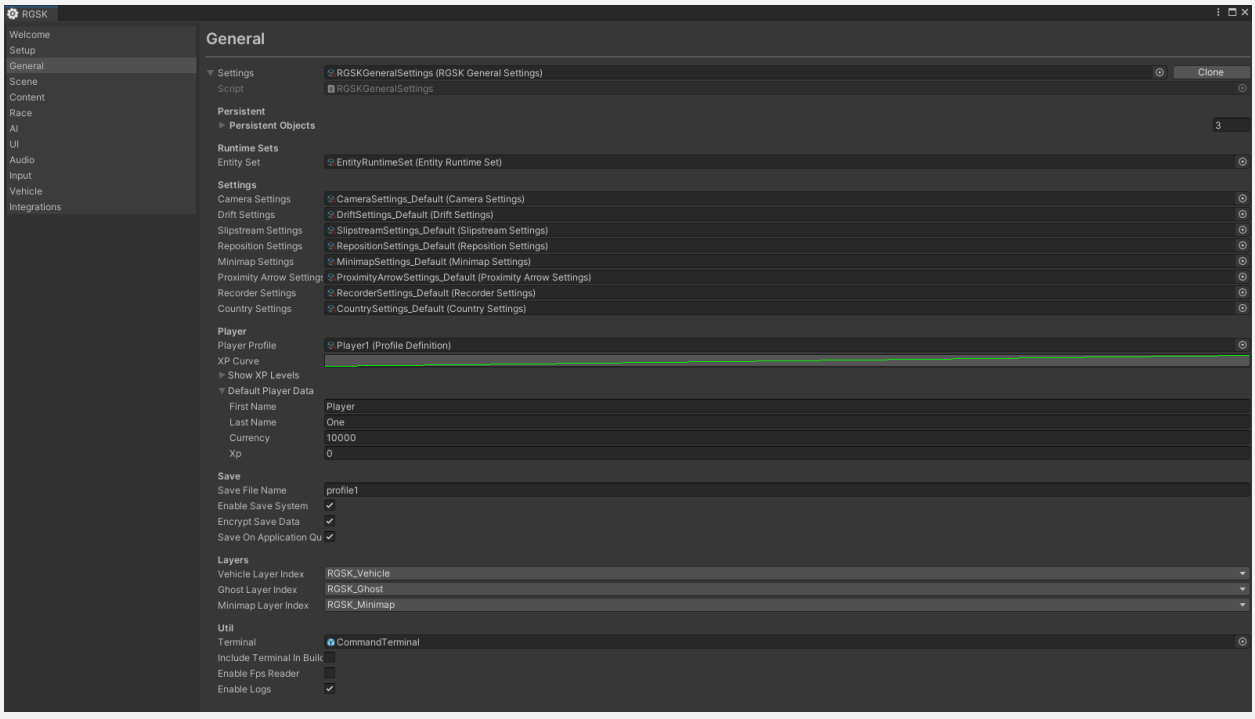
The RGSK Menu is the central window where all project settings are managed.

To open the RGSK Menu, select **Window/RGSK/Menu**

It is recommended to clone the scriptable objects and to use the cloned version instead of the default ones. This ensures that your settings won't get overwritten if you ever update the asset in the future. This can be done by clicking the “Clone” button in each tab. Other scriptable objects such as “Camera Settings”, “Drift Settings”, “Minimap Settings”, etc should be duplicated in the project tab and reassigned.



General



Persistent Objects - a list of objects that are created at runtime. **It is important to not remove "PersistentManagers" from this list as they are the backbone of RGSK**. Persistent Managers can be found under **Assets/RGSK/Core/Prefabs.**

Entity Set - this is a runtime set responsible for adding RGSK Entities to a shared list that is read from by the Race Manager, Camera Manager, and other managers.

Camera Settings - the camera settings used by the Camera Manager. If the settings is left in the Camera Manager, these settings will be used

Drift Settings - the drift settings used by the drifting system

Slipstream Settings - the settings used by the slipstream component

Reposition Settings - the settings used by the repositioner component

Minimap Settings - the settings used by the Minimap System

Recorder Settings - the settings used by the replay system. See Recorder Settings.

Country Settings - the settings used to store **Country Definitions**.

Player Profile - the player's details

XP Curve - the XP curve used by the player leveling system. To preview the Level/XP expand the "Show XP Levels" foldout

Default Player Data - the default values assigned to the player on a new save file

Save - see Save System

Vehicle/Ghost Layers - the layers that are used by vehicles and ghosts.

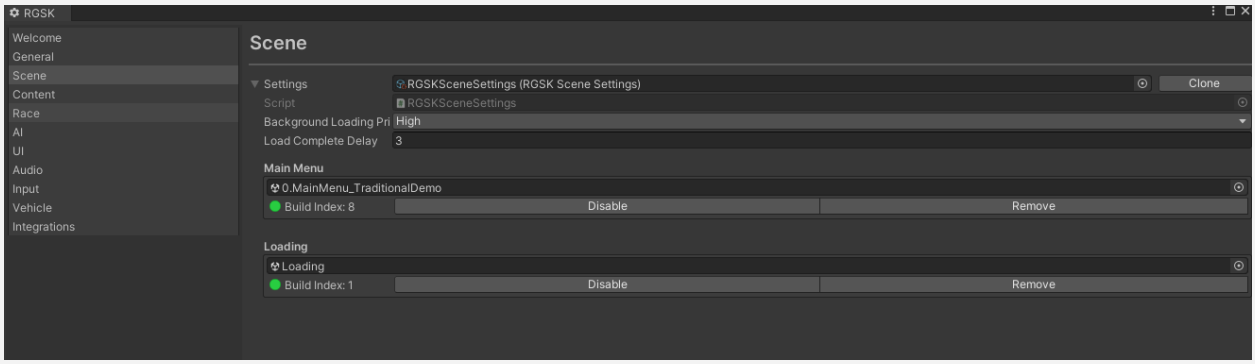
Terminal - the terminal used for development commands

Include Terminal in Builds - whether the terminal should be a part of builds. This option should be set to **false** when you want to ship your game.

Enable FPS Reader - show the FPS at runtime. This value is overridden by the Save Manager at runtime based on the value set in the settings screen.

Enable Logs - whether the logging system should be enabled. This is helpful for debugging.

Scene



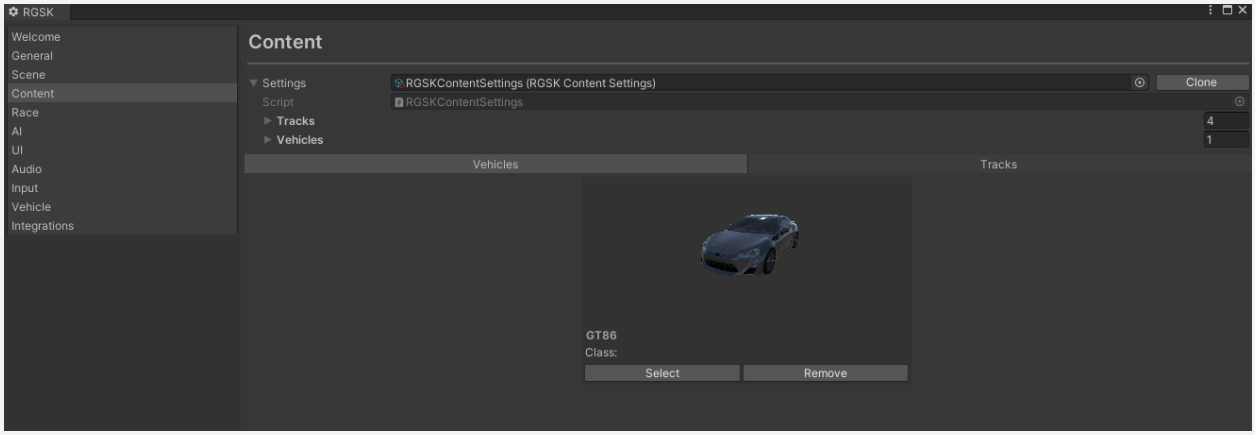
Background Loading Priority - loading priority. Leave this to **High**

Load Complete Delay - the amount of time to wait after a scene has been loaded to activate it

Main Menu - the main menu scene. This is the scene loaded whenever a scene is exited

Loading - the scene used to load other scenes. Please reference **Assets/RGSK/Demos/_CommonScenes/Loading.unity**

Content

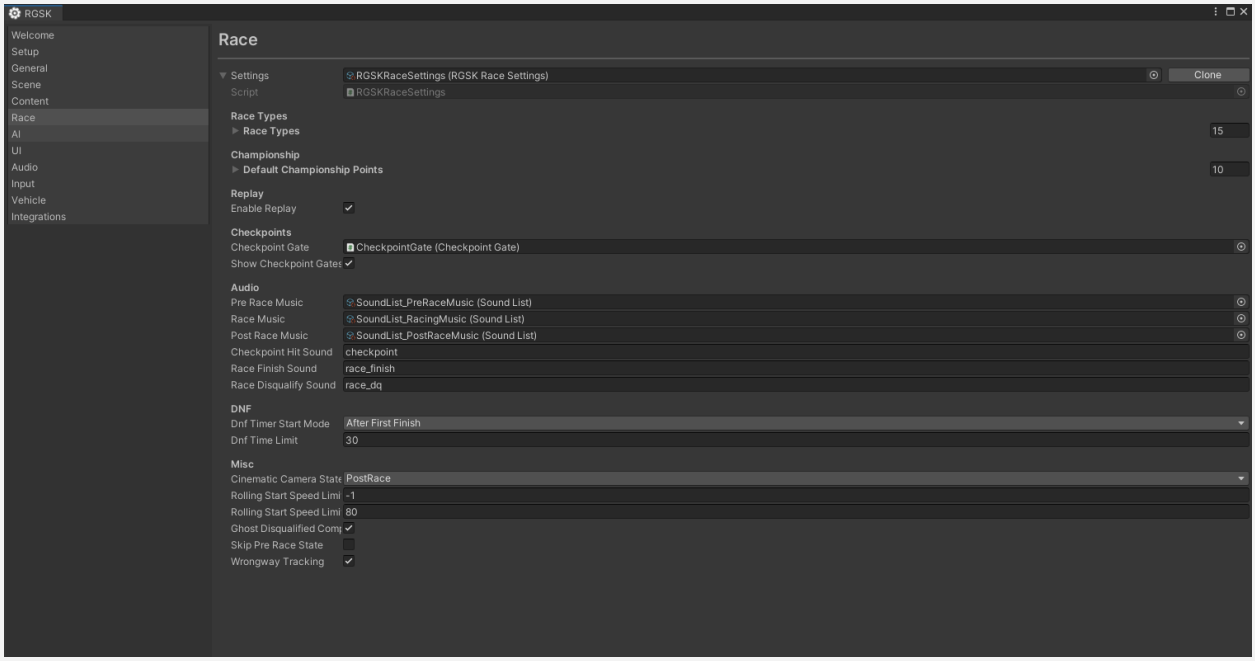


The content settings are where vehicles and tracks are stored and managed.

Vehicles are added through **Vehicle Definitions** and tracks are added through **Track Definitions**

See Adding the Vehicle to Content Settings and Adding the Track to Content Settings for more information

Race



Race Types - the list of race types

Checkpoints - see Checkpoint Gates

Pre-Race/Race/Post-Race Music - the music played in the corresponding race state. These are Sound Lists that contains the music playlist. See Audio System for more information.

Checkpoint Hit Sound - the sound played when a checkpoint is passed. This sound is fetched from the **Sounds** list. See Audio Settings

Race Finish/Disqualify Sound - the sound played when the player finishes or is disqualified. These sounds are fetched from the **Sounds** list. See Audio Settings

DNF Timer Start Mode - when the DNF timer should start

DNF Time Limit - how long the DNF timer will last for

Enable Replay - whether the replay system will run during races.

Default Championship Points - the points given per finishing position in a championship. [0] is 1st, [1] is 2nd, and so on.

Cinematic Camera States - the race states that the route cameras will be automatically enabled on

Rolling Start Speed Limit Solo - The speed limit (in KM/h) that the vehicles will aim to go during rolling starts where there is only 1 competitor. Set to -1 for no limit

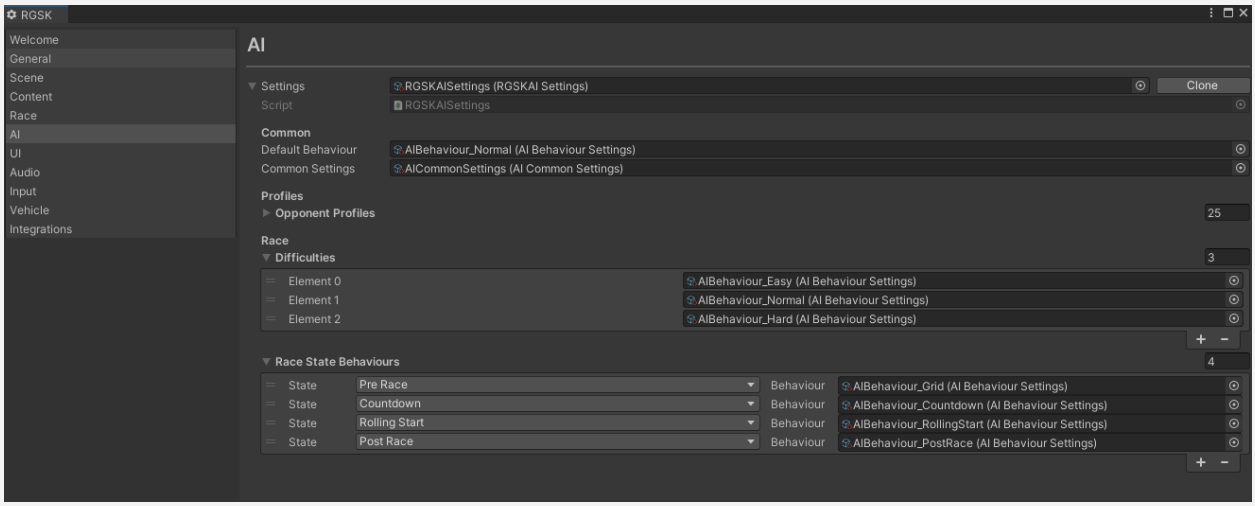
Rolling Start Speed Limit Multiple - The speed limit (in KM/h) that the vehicles will aim to go during rolling starts where there are multiple competitors. Set to -1 for no limit

Ghost Disqualified Competitors - whether disqualified competitors will be ghosted

Skip Pre-Race State - if true, the race will start in the countdown state, skipping any pre race UI.

Wrongway Tracking - whether to track if a competitor is going the wrong way.

AI



Default Behavior - the behavior assigned to AI by default

Common Settings - the common AI settings used

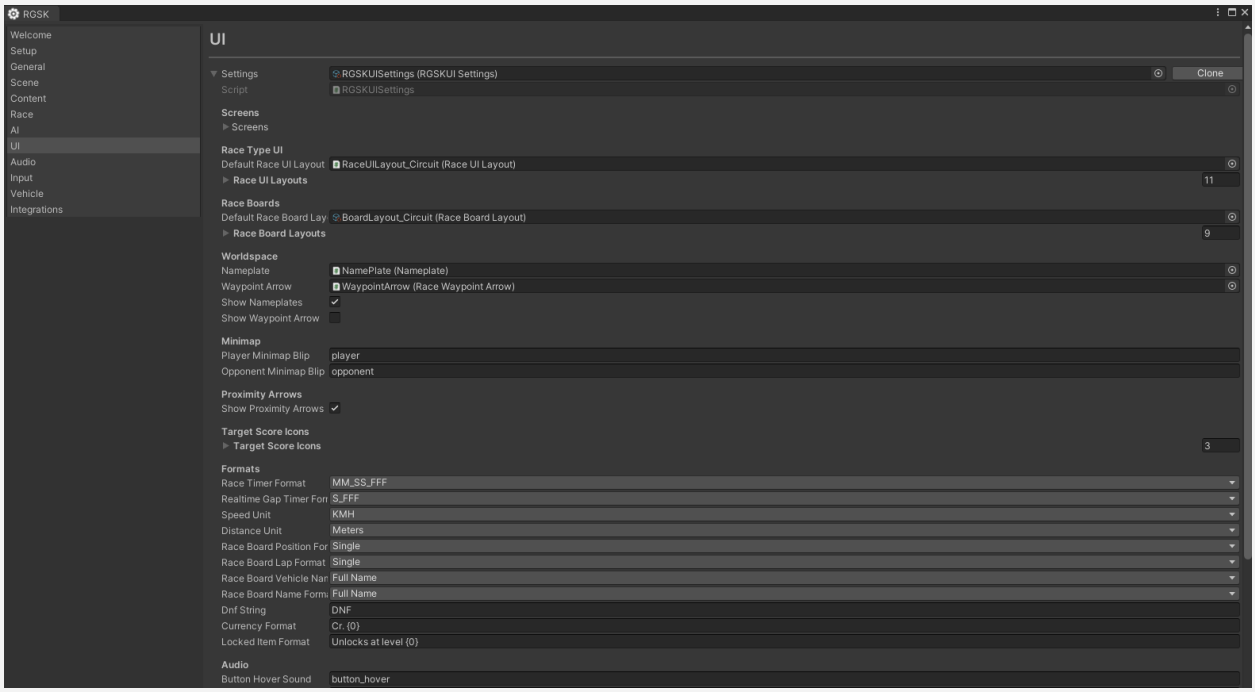
Opponent Profiles - a list of profiles that will be assigned to AI competitors during a race. To create a profile, select **Assets/Create/RGSK/Core/Profile** or use the tool at **Window/RGSK/Tools/Profile Definition Creator**

Difficulties - a list of AI Behaviors that will be selectable as difficulties in Race Sessions

Race State Behaviors - the AI behavior given to the AI at the corresponding Race State.

See AI System for more information.

UI



Screens - the screen IDs for the corresponding screens. See UI System for more information.

Default Race UI Layout - the default race UI layout used when one is not found for the active race type

Race UI Layouts - a list of Race UI Layout prefabs that are used to display varying UIs for different race types. See Race Type Specific UI for more information.

Default Board Layout - the default layout used for boards when one is not found for the active race type

Race Board Layouts - a list of all the board layouts that are used for different race types. See Race board Layouts for more information.

Button Hover/Click Sounds - the sounds played on button hover and click. These sounds are fetched from the **Sounds** list. See Audio Settings.

Modal Windows - the modal windows used by the Modal Window Manager to display modals. See Modal Windows for more information.

Modals - the properties used to display the corresponding modal windows.

Nameplate - the prefab used for nameplates. See Nameplates for more information.

Waypoint Arrow - the prefab used for the waypoint arrow. See Waypoint Arrow for more information.

Player/Opponent Minimap Blip - the blip used for player/opponents. See Minimap System for more information.

Race Timer Format - the format used for race timers in the UI

Real-time Gap Time Format - the format used to display time gaps in race boards

Speed Unit - the unit used to display speeds

Distance Unit - the unit used to display distances

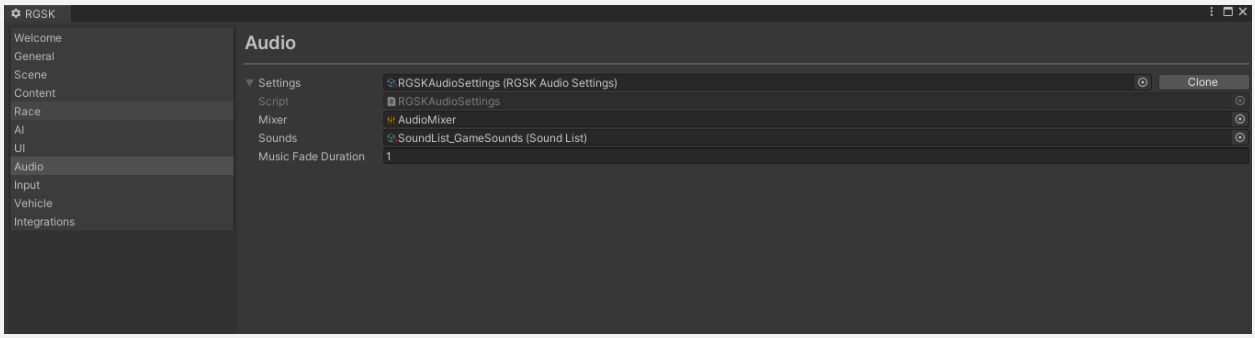
DNF String - the string used to show that a competitor is disqualified

Currency Format - how the currency string is displayed.

Locked Item Format - the string displayed by the UI for locked items set to the “XP Level” unlock mode

Target Score Icons - the icons used to display target scores. These are also shared by the built-in menu system’s race event/championship screens to display the player's record.

Audio

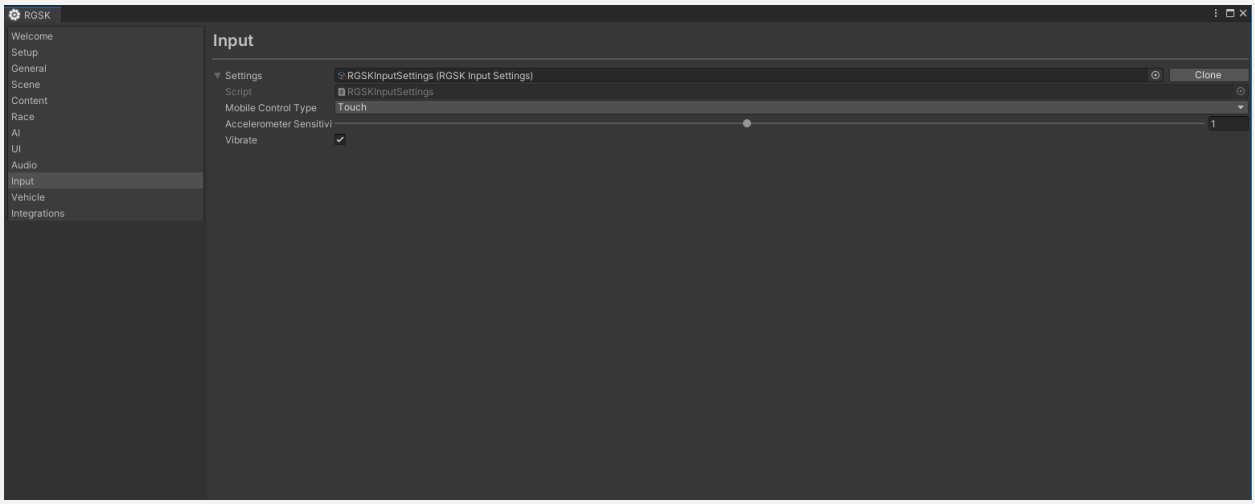


Mixer - the audio mixer used for all audio. The audio mixer needs to have certain groups and exposed values so it is best to use the default one located at **Assets/RGSK/Core/Audio**

Sounds - the Sound List (see Audio System) that contains all the sounds used at runtime.

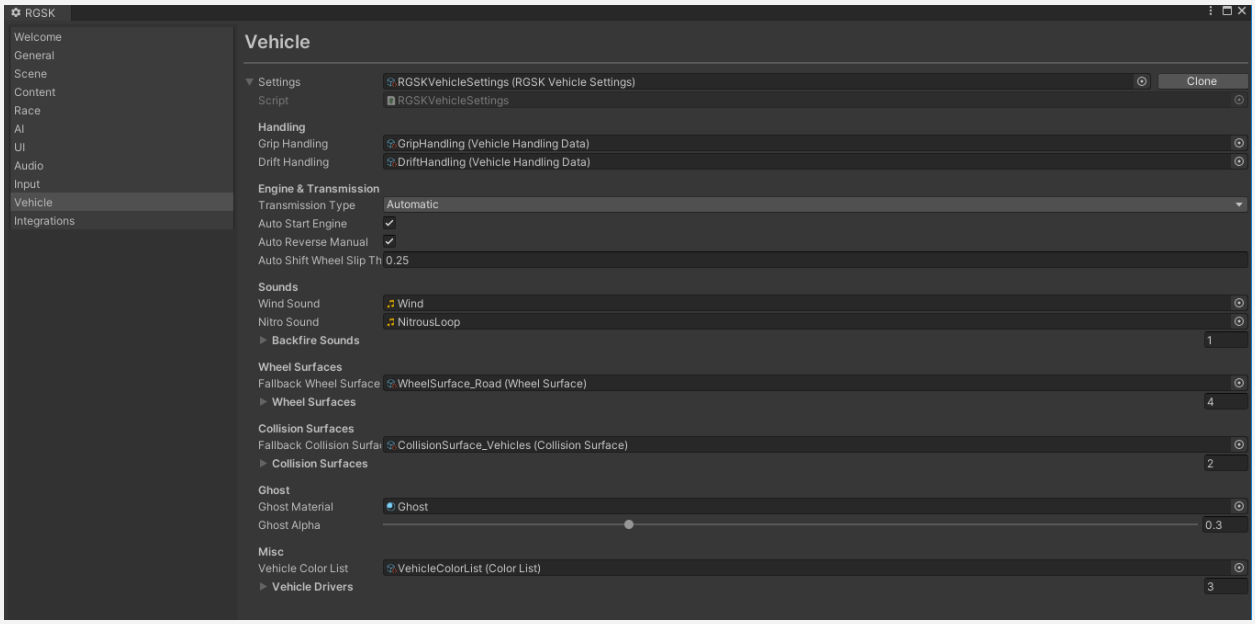
Music Fade Duration - how long the Music Manager (see Audio System) takes to fade music out

Input



See Input System

Vehicle



Grip/Drift Handing - the handing data used for the corresponding handling mode

Transmission Type - whether gear shifting will be done automatically or manually

Auto Start Engine - whether the vehicle engine will be started automatically

Auto Reverse Manual - whether the vehicle should automatically reverse when brake is held down. This only applies for manual transmission

Auto-Shift Wheel Slip Threshold - if the forward wheel slip is above this value, the vehicle will not shift up. This option only applies for automatic transmission

Wind Sound - the wind sound used by the built-in vehicle physics

Nitro sound - the sound used by the built-in vehicle physics for nitrous.

Backfire Sounds - the list of sounds used by the built-in vehicle physics when selecting a backfire sound to play

Fallback Wheel Surface - the wheel surface used when no matching surface was found by the wheel

Wheel Surfaces - the list of all wheel surfaces used by the Surface Manager. See Surface Manager for more information

Fallback Collision Surface - the collision surface used when no matching surface was found by the collision

Collision Surfaces - the list of all collision surfaces used by the Surface Manager. See Surface Manager for more information

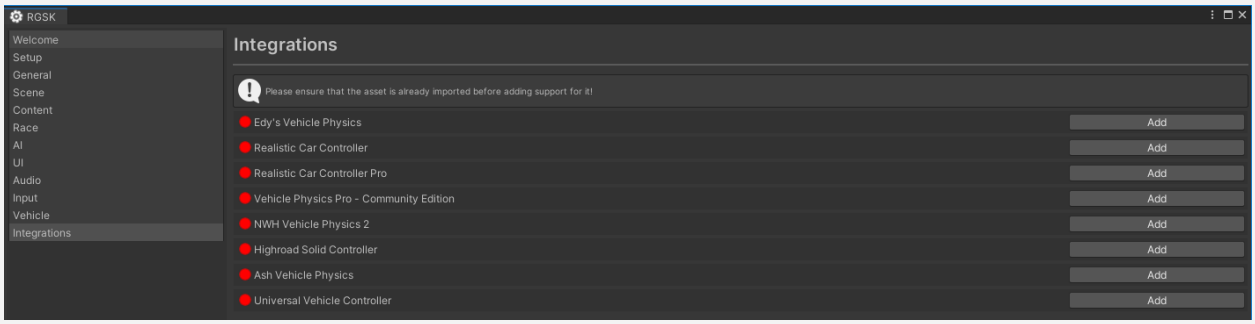
Ghost Material - the material used when a vehicle is ghosted

Ghost Alpha - the transparency of the ghosted vehicle

Vehicle Color List - the list of colors used when selecting vehicle colors

Vehicle Drivers - the list of driver prefabs that are used when creating drivers for vehicles. See Vehicle Setup for more information.

Integrations



All 3rd-party Integrations can be found here. See 3rd-party Integrations for more information.

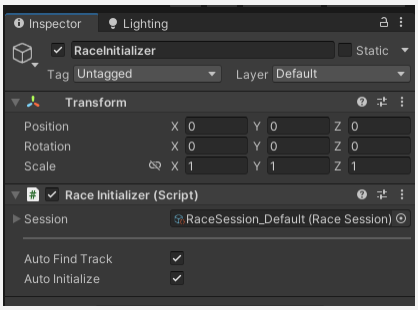
Racing System

The Racing System mostly runs in the background. It is initialized by a Race Initializer.

Race Initializer

The race initializer is responsible for initializing a race session.

Please see Creating the Race Initializer for a guide on how to create one.



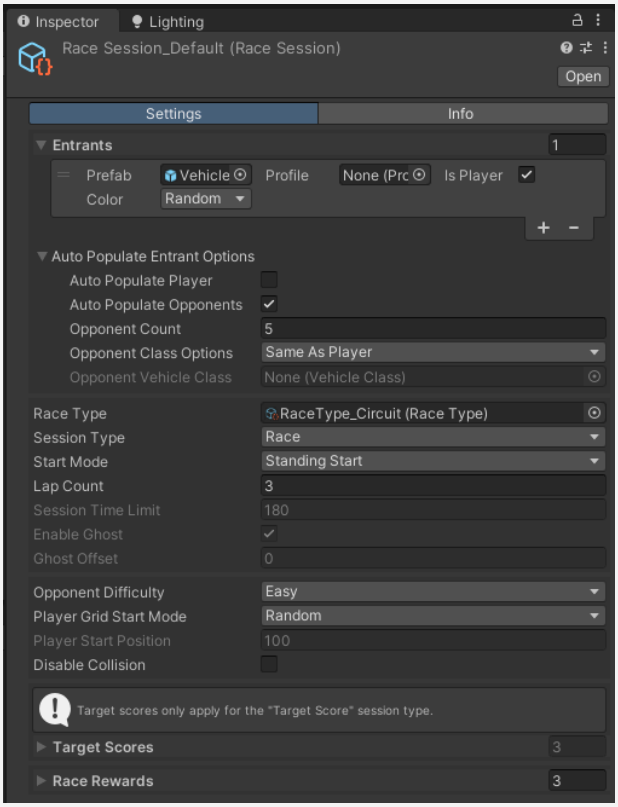
Session - the Race Session to be initialized

Auto Find Track - whether the track will be found automatically. Uncheck this to assign a track manually.

Auto Initialize - whether the race should be initialized once the scene is loaded.

Race Session

To create a race session, select **Assets/Create/RGSK/Race/Race Session**



Entrants - a list of each entrant in the race.

Auto Populate Entrant Options

Auto Populate Player - whether the player entrant should be added automatically at runtime. This option will load the vehicle assigned to RGSKCore.runtimeData.SelectedVehicle;

Auto Populate Opponents - whether the AI be added automatically when the race is initialized.

Opponent Count - how many AI opponents to add.

Opponent Class - the vehicle class the AI opponents will be added from.

Race Type - the race type that will be used in the session.

Session Type - this can either be “Race” or “Target Score”. In “target score”, the objective is to achieve targets that are defined under the “Target Scores” list (see below).

Start Mode - whether the race will start as a standing start or a rolling start.

AI Difficulty - the AI behavior that will be used by the AI during the session. See AI Behaviors for more information.

Player Grid Start Mode - how the player’s starting position is determined. When set to “Selected”, the start position can be set in **Player Start Position**

Lap Count - the number of laps. This option is only available for lap-based race types.

Session Time Limit - the duration of the session. This option is only available for time-based race types.

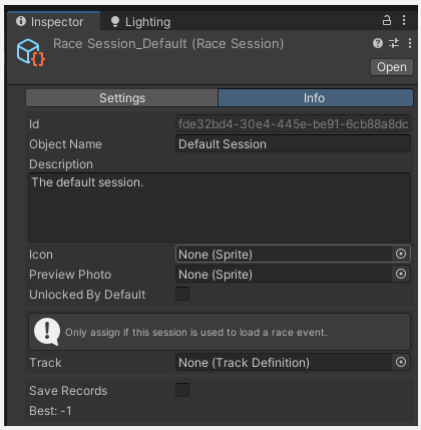
Enable Ghost - whether the ghost vehicle will be used. This option is only available for race types that allow ghosts.

Ghost Offset - how far ahead (in seconds) the ghost will start infront of the player.

Disable Collision - whether the competitors will be ghosted during the race.

Target Scores - a list of scores to be achieved. This option is only available for target score session types (see above).

Race Rewards - a list of the rewards assigned per finishing position. Element [0] is 1st, Element [1] is 2nd, and so on. See Reward System for more information.



ID - an automatically generated guid for this item.

Object Name - the name of this item.

Description - the description of this item

Icon - the icon of this item used by the UI.

Preview Photo - the photo of this item used by the UI.

Unlocked by Default - whether this item is unlocked by default.

Track - the track associated with this session. This should only be assigned if the session is being used to load a custom race event.

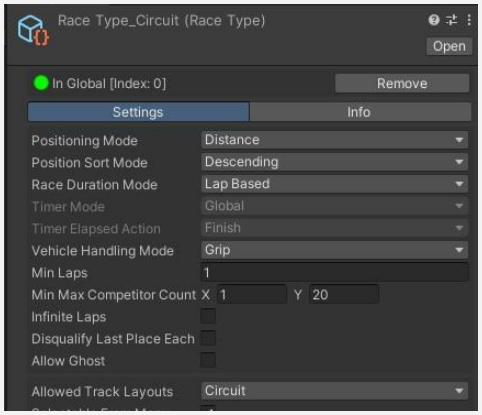
Save Records - whether the player’s best finishing position should be saved.

Best - the player’s best finishing position.

Race Types

To create a race type, select **Assets/Create/RGSK/Race/Race Type**

A race type defines the core settings of a race session.



In Global - whether this race type is added to the RGSK Race Settings. Use the Add/Remove buttons to toggle.

Positioning Mode - the value used to determine race positions.

Position Sort Mode - how the value of Position Mode (see above) will be ordered. Descending will order from highest to lowest.

Race Duration Mode - how the race end is determined.

Timer Mode - whether the session timer will affect all competitors (Global) or individually (Per Competitor).

Timer Elapsed Action - what happens when the global session timer elapses. This option is only available when “Timer Mode” is set to “Global”.

Vehicle Handing Mode - the handling mode that will be applied to the competitor’s vehicles.

Min Laps - the minimum amount of laps allowed.

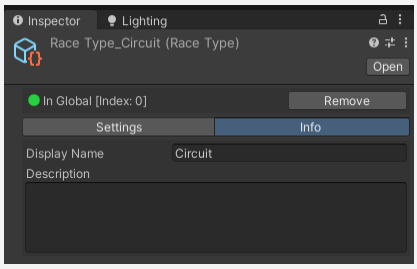
Min Max Competitor Count -the minimum and maximum amount of competitors that can participate.

Disqualify Last Place Each Lap - whether last place should be disqualified at the end of each lap.

Allow Ghost - whether ghosts are allowed to be used for this race type.

Allowed Track Layouts - which kind(s) of tracks this race type is allowed. This option is used by the built-in menu code (RaceSettingsScreen.cs).

Selectable From Menu - whether this race type will be selectable in the main menu. This option is used by the built-in menu code (RaceSettingsScreen.cs).



Display Name - the name of the race type

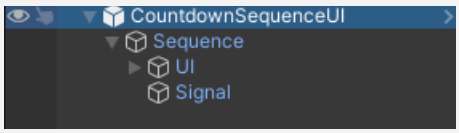
Description - a description of the race type

Race Countdown

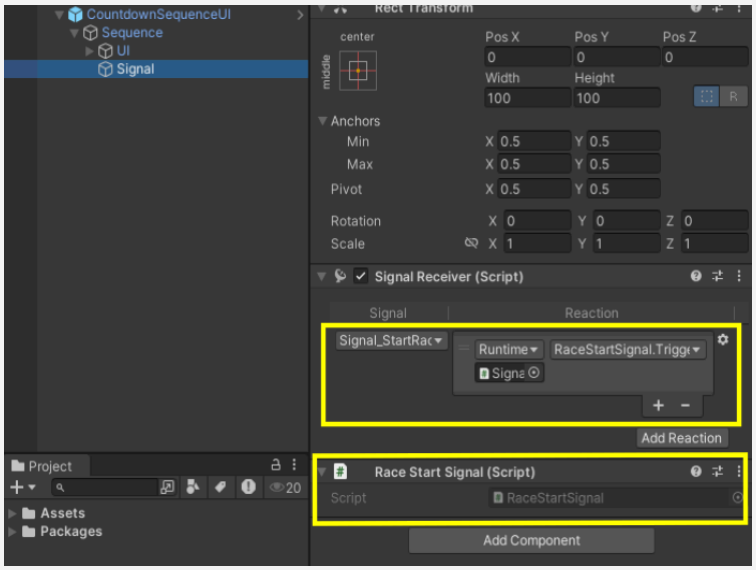
The race countdown is responsible for triggering the race start so it is **crucial for any race scene**. It works using Timeline and Timeline Signals.

Typically, it should be part of the UI (see "**Screen_Race**" under **Assets/RGSK/Demos/_CommonAssets/Prefabs/UI/Screens**), but you can have it anywhere in your scene.

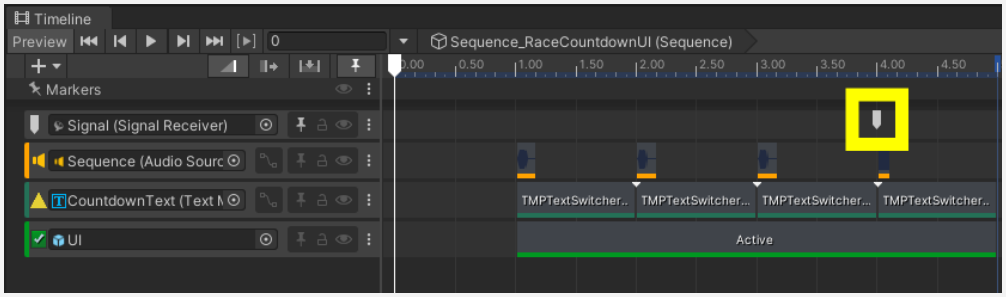
To set up a new countdown sequence, you can use the "**CountdownSequenceUI"** prefab as a starting point (**Assets/RGSK/Demos/_CommonAssets/Prefabs/UI/Elements/CountdownSequenceUI.prefab**):



The "**Signal**" is the key part. It triggers the race start at a given point in the Timeline sequence. The signal receiver calls the Trigger() method in the **Race Start Signal** component



In the Timeline window, the signal is set to trigger once the "GO" text appears:



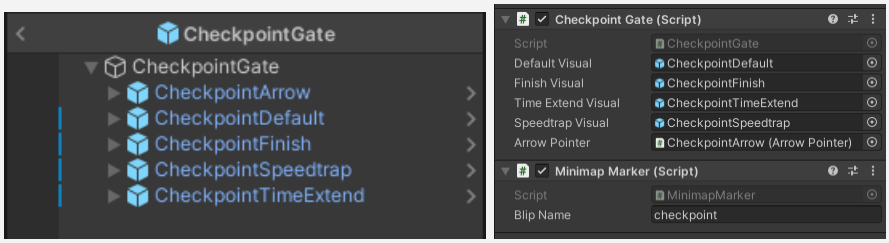
Following this structure, you can create your own race countdown sequences. It's important to ensure that the sequence is present in the scene before the Race Manager enters the countdown state so that the RaceCountdown.cs component can respond to the race state changed event.

Checkpoint Gates

Checkpoint gates are a visual representation of a checkpoint. An instance is instantiated on race start and it moves to the next checkpoint while changing its visuals based on the type of the next checkpoint.

For an example of one, open the asset at:

Assets/RGSK/Demos/_CommonAssets/Prefabs/Checkpoint/CheckpointGate.prefab



Default Visual - the object enabled for normal checkpoint types

Finish Visual - the object enabled for the final checkpoint of the race

Time Extend Visual - the object enabled for “Time Extend” checkpoint types

Speedtrap Visual - the object enabled for “Speedtrap” checkpoint types

Arrow Pointer - an arrow that points to the next checkpoint

A **Minimap Marker** script is also added and is responsible for creating a minimap blip for this object. See Minimap System for more information.

Checkpoint gate settings can be found in the RGSK Menu under “Race Settings”

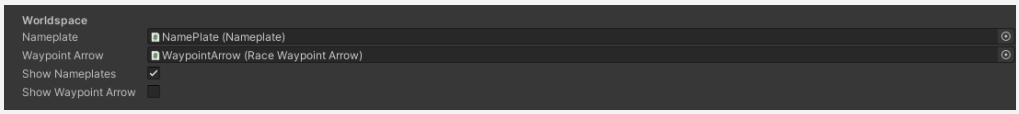
Waypoint Arrow

The waypoint arrow points to the next checkpoint. It is useful for open-world racing where the next checkpoint may not immediately be clear to the player.

For an example of one, open the asset at:

Assets/RGSK/Demos/_CommonAssets/Prefabs/UI/WorldSpace/WaypointArrow.prefab

Waypoint arrow settings can be found in the RGSK Menu under “UI Settings”



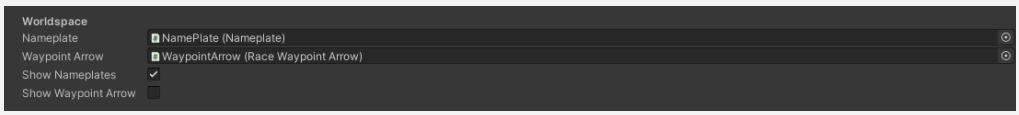
Nameplates

Nameplates are small UI panels above each race competitor that display competitor information (such as position, name, etc).

For an example of one, open the asset at:

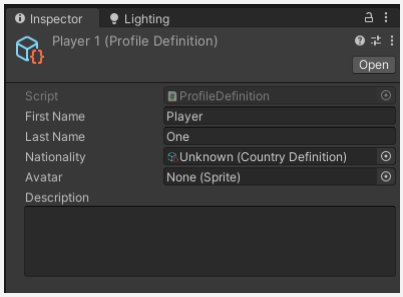
Assets/RGSK/Demos/_CommonAssets/Prefabs/UI/WorldSpace/NamePlate.prefab

Nameplate settings can be found in the RGSK Menu under “UI Settings”



Profiles

Profiles define the details of a competitor. To create a new profile, select **Assets/Create/RGSK/Core/Profile,** or to create multiple profiles from a text file use the tool at **Window/RGSK/Tools/Profile Definition Creator**



First/Last Name - the names of the competitor

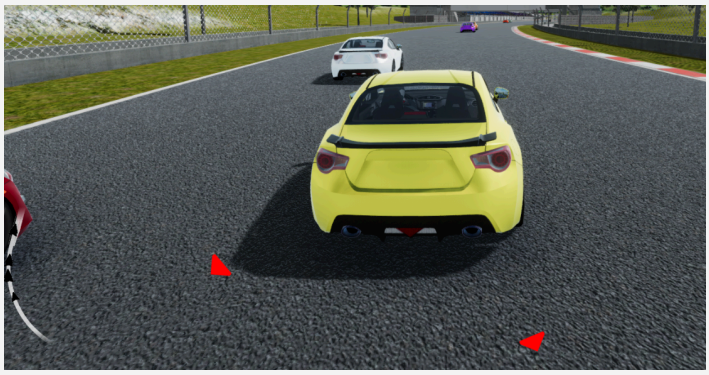
Nationality - the country of the competitor

Avatar - the avatar for this competitor

Description - a description of the competitor

These values can be displayed in the UI using **ProfileDefinitionUI.cs**

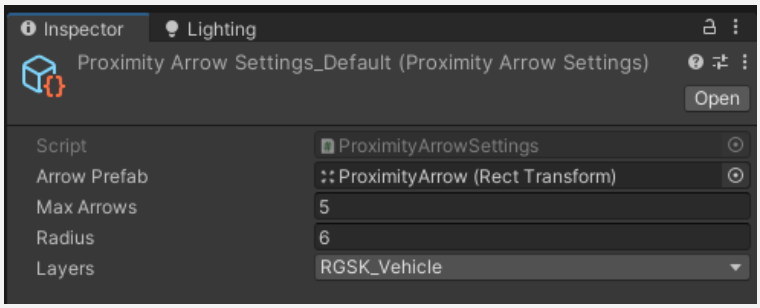
Proximity Arrows



Proximity arrows are UI indicators of surrounding objects. They work by:

1. Adding **ProximityArrowUI.cs** component to a UI Canvas
2. Adding **ProximityArrowTarget.cs** component to detectable objects

Proximity Arrow Settings are stored in the RGSK Menu under the "General" tab:



Arrow Prefab - The UI prefab of the arrow. For an example, see **Assets/RGSK/Demos/_CommonAssets/Prefabs/UI/Elements/ProximityArrow.prefab**

Max Arrows - The maximum number of arrows that can be on screen

Radius - The radius of the sphere that detects surrounding objects

Layers -The layers to detect surrounding objects from

Proximity arrows can be toggled in the RGSK Menu under “UI Settings”

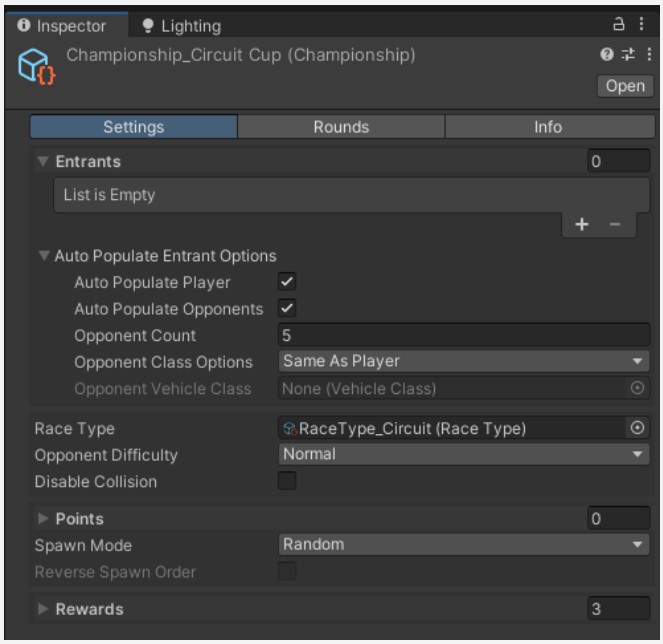
Championship System

Championships are a series of race sessions, where competitors earn points based on their finishing position in each round. The competitor with the highest total points at the end is the winner.

Creating a Championship

To create a new championship, select **Assets/Create/RGSK/Race/Championship**

Settings



Entrants - a list of each entrant in the race.

Auto Populate Entrant Options

> **Auto Populate Player** - whether the player entrant should be added automatically at runtime. This option will load the vehicle assigned to RGSKCore.runtimeData.SelectedVehicle;
>
> **Auto Populate Opponents** - whether the AI be added automatically when the race is initialized.
>
> **Opponent Count** - how many AI opponents to add.
>
> **Opponent Class** - the vehicle class the AI opponents will be added from.

Race Type - the race type that will be used by each race in the championship.

Opponent Difficulty - the AI behavior that will be used by the AI during the session. See AI Behaviors for more information.

Disable Collision - whether the competitors will be ghosted during each race.

Points - the points assigned per finishing position. If no points are assigned, the championship points will be read from the "**Default Championship Points**" in the Race Settings.

Spawn Mode - how competitors will be spawned in each round

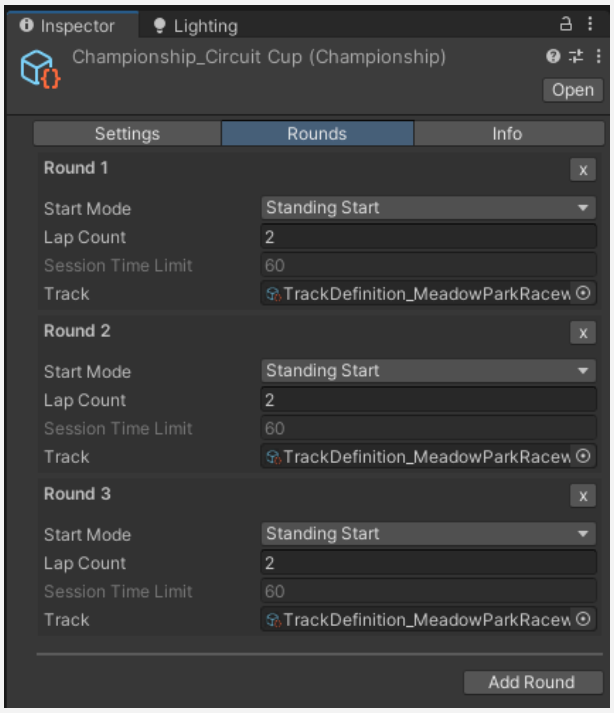
> **Random** - the competitors will spawn in random order
>
> **Total Points** - the competitors will spawn based on their total points
>
> **Previous Round Position** - the competitors will spawn based on the position they finished in the previous round

Reverse Spawn Order - whether to reverse the spawn order (only applicable if Spawn Mode is set to "Total Points" or "Previous Round Position")

Rewards - a list of the rewards assigned per finishing position. Element [0] is 1st, Element [1] is 2nd, and so on. See Reward System for more information

Rounds

To add a new round, select the “Rounds” tab and click on the "Add Round" button. To remove a round, click on the “x” button at the top-right corner of each round.



Some settings will be unavailable depending on the race type.

Start Mode - whether the race will start as a standing start or a rolling start.

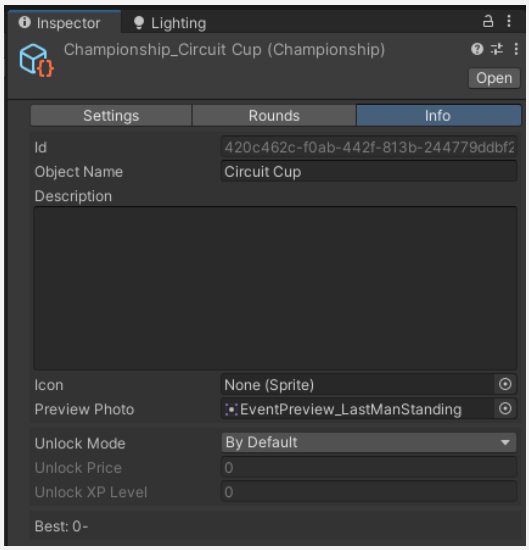
Lap Count - the number of laps. This option is only available for lap-based race types.

Session Time Limit - the duration of the session. This option is only available for time-based race types.

Track - the track used for this round.

Info

The championship info (such as the name) can be set in the “Info” tab. See Items for more information.



Starting a Championship

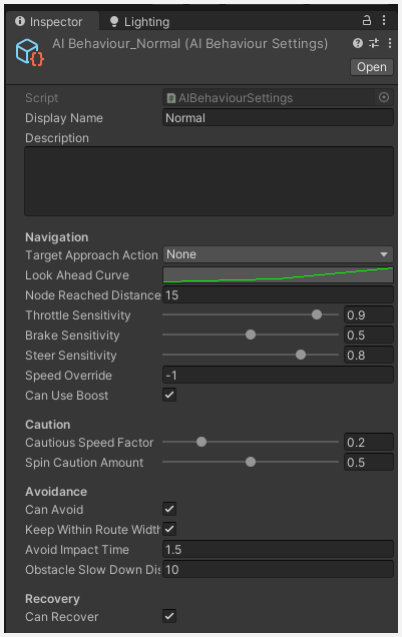
Championships can be started by calling the **ChampionshipManager.Instance.StartChampionship();** method in code or by adding them to the Championship UI Screen

AI System

The AI system works through the **AIController.cs** script. It is responsible for managing the AI's behavior.

Behaviors

AI Behaviors are stored as scriptable objects. To create a new AI behavior, select **Assets/Create/RGSK/AI/AI Behavior**



Display Name - the name of this behavior (mostly used for UI)

Description - a description of this behavior

Target Approach Action - how the AI will manage its speed based on the distance to the follow target

Look Ahead Curve - the curve that defines how far the follow target will be at a given speed (km/h)

Node Reached Distance - how far the AI has to be from a node to move on to the next

Throttle/Brake/Steer Sensitivity - how responsive the AI will be in giving these inputs

Speed Override - the speed that the AI will aim to travel at. Leave at - 1 to use the route's speed values

Can Use Boost - whether the AI will be capable of using boost input

Cautious Speed Factor - the % of current speed that the AI will slow down to when fully cautious. 0 = full slow down, 1= no slow down

Spin Caution Amount - how cautious the AI will be in detecting spin-outs

Can Avoid - whether the AI will be able to avoid obstacles

Keep Within Route Width - whether the AI will keep within route bounds when attempting to avoid an obstacle

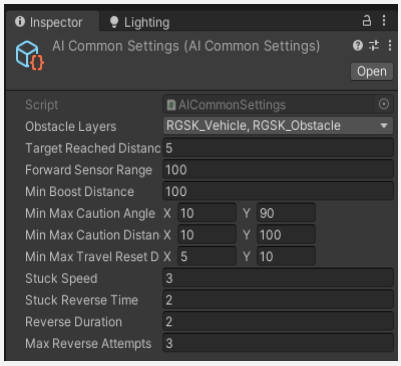
Avoid Impact Time - the time before impact that the AI should attempt to avoid an obstacle

Obstacle Slow Down Distance - the distance at which the AI will slow down if it gets too close to an obstacle

Can Recover - whether the AI will attempt to recover when stuck

Common Settings

The AI shares common settings that are stored in an **AI Common Settings** scriptable object. To create a new AI Common Settings, select **Assets/Create/RGSK/AI/AI Common Settings**



Obstacle Layers - the layers that the AI will check for obstacles in

Target Reached Distance - the distance from the target that the AI should stop at. This option only applies when the behavior's "target approach action" is set to "slow down and stop"

Forward Sensor Range - the range of the forward sensor used to detect obstacles ahead

Min Boost Distance - how far the AI must travel before being able to use boost

Min/Max Caution Angle - the angles from the follow target at which the AI will be minimally and maximally cautious

Min/Max Caution Distance - the distances at which the AI will begin to slow down. This only applies to the "slow down" & "slow down and stop" target approach actions

Min/Max Travel Reset Duration - how long after attempting to avoid an obstacle should the AI wait before returning to the racing line

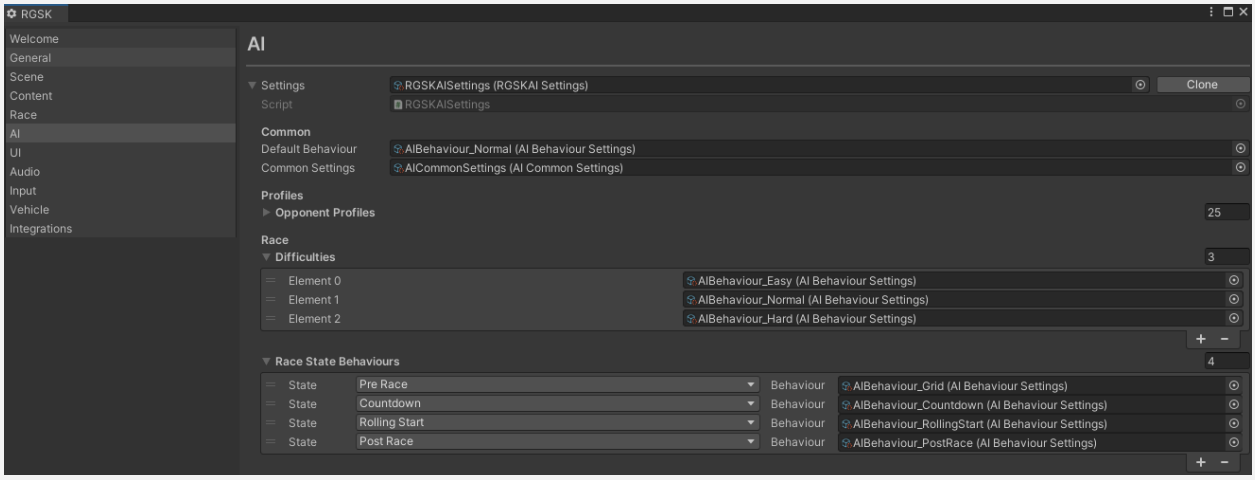
Stuck Speed - the speed that the AI will consider itself stuck and begin a recovery attempt

Stuck Reverse Time - how long to wait when stuck before attempting to reverse

Reverse Duration - how long to reverse when attempting to recover

Max Reverse Attempts - how many recovery attempts before forcing a respawn

All the AI Settings are stored in the RGSK Menu, see AI for more information.



Camera System

The camera system is powered by Cinemachine. It works through a central manager called the Camera Manager.

Camera Manager

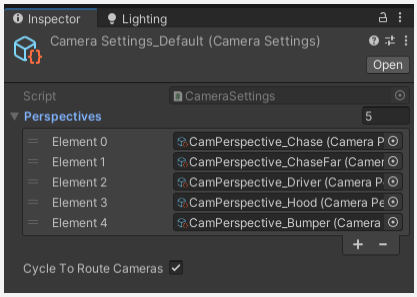
The Camera Manager is responsible for creating and managing all cameras in a scene. It is an essential part of every scene and so should be included in every scene.

Please see Creating the Camera Manager for a guide on how to create one.

Camera Settings

To create new camera settings, select **Assets/Create/RGSK/Settings/Camera Settings**

To use the new camera settings, either assign it directly to the camera manager or assign it in the RGSK Menu under “General Settings”.

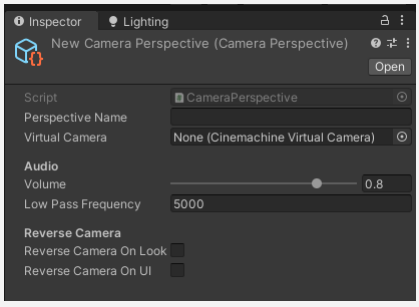


Perspectives - a list of the camera perspectives that will be used. The order in which they are arranged is the order in which they will be activated.

Cycle to Route Cameras - activate route cameras after all perspective cameras have been cycled through. See Creating Route Cameras for a guide on how to create them.

Camera Perspectives

To create a new camera perspective, select **Assets/Create/RGSK/Camera/Camera Perspective**



Perspective Name - the name of this perspective

Virtual Camera - the camera that will be used by this perspective. See **RGSK/Demos/_CommonAssets/Prefabs/Camera** for sample prefabs.

Volume - how loud the audio should be when this camera perspective is being used

Low Pass Frequency - the low pass to be used. Lower values should be used for interior cameras and higher values for exterior cameras.

Reverse Camera On Look Back - whether the reverse camera should be activated when looking back

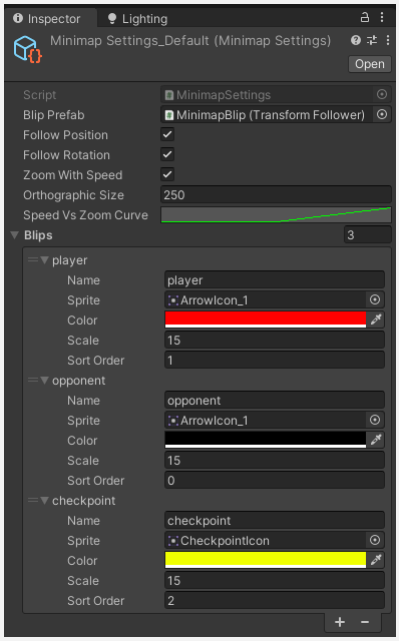
Reverse Camera On UI - whether the reverse camera UI element should be active when this perspective is being used

Once you are happy with the settings, assign the perspective to the camera settings to be used at runtime.

Minimap System

The minimap system works by having an orthographic camera (see Minimap Camera under the Camera Manager object) that renders objects that are in the “RGSK_Minimap” layer. The camera renders its output to a Render Texture which is then used by the UI to display the minimap. A Minimap Manager script runs in the background that is responsible for creating minimap blips.

Minimap Settings can be found in the RGSK Menu under “General Settings”



Blip Prefab - the prefab used for minimap blips

Follow Position/Rotation - whether the minimap camera should follow the position/rotation of the target

Zoom with Speed - whether the orthographic size of the minimap should change based on the speed of the target

Orthographic Size - the default orthographic size of the camera

Speed vs Zoom Curve - how the camera should zoom in relation to the target’s speed. This option only applies if “zoom with speed” is true.

Blips - a list of blip information that is used by the Minimap Manager to create blips.

Replay System

The replay system works through a persistent "Recorder Manager" created at runtime. This object stores and manages all recorders.

To make a scene object recordable, add the **RecordableSceneObject** component to it and the replay system will add it to the list of recordable objects. Open the replay showcase scene to see it in action (**Assets/RGSK/Demos/_CommonScenes/Showcase/Replay.unity**)

Recorder Settings

Recorder settings are shared settings used by the replay system. They are stored in the RGSK menu under "General Settings":

Recording Time Limit - the maximum length (in seconds) that a recording can last. Leave at -1 for no limit.

Max Playback Speed - the maximum fast forward/rewind replay speed

Slow Motion Time Scale - the timescale used when a replay is in slow motion

Ghost System

The ghost system works in a similar way to the replay system.

To make a ghostable object, add the **RecordableGhostObject** component to it. A clone will be created at runtime that will mimic the original object on demand. Open the ghost showcase scene to see it in action (**Assets/RGSK/Demos/_CommonScenes/Showcase/Ghost.unity**)

Vehicle Physics

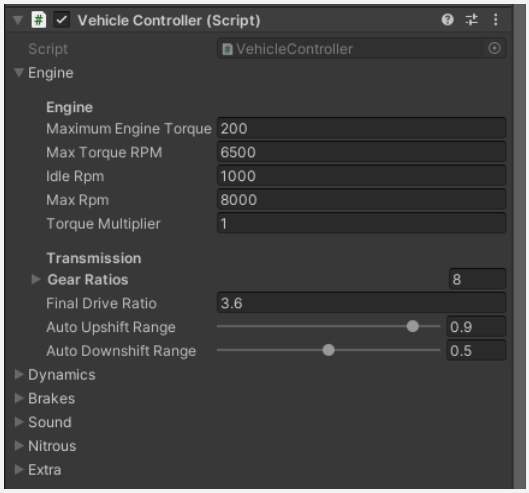
The built-in vehicle physics offers simple and reliable vehicle physics.

To set up a new vehicle, please see Vehicle Setup

Vehicle Controller

The vehicle physics are comprised of multiple components that work together. Each of the component's values can be tweaked to achieve different results for different vehicles.

Engine



Maximum Engine Torque - the maximum torque the engine will produce

Max Torque RPM - the RPM at which Maximum Engine Torque is produced

Idle RPM - the RPM that the engine idles at

Max RPM - the maximum RPM the engine can rev to

Torque Multiplier - the torque is multiplied by this value

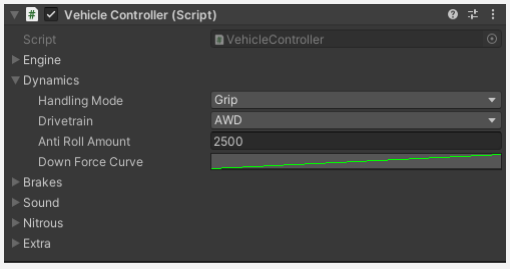
Gear Ratios - the number of gears and their ratios. Element [0] is reverse gear, Element [1] is neutral, and the rest are the forward gears.

Final Drive Ratio - the final drive ratio

Auto Upshift Range - the % of Max RPM to shift up. This only applies for automatic transmission.

Auto Downshift Range - the % of Max RPM to shift down. This only applies for automatic transmission.

Dynamics



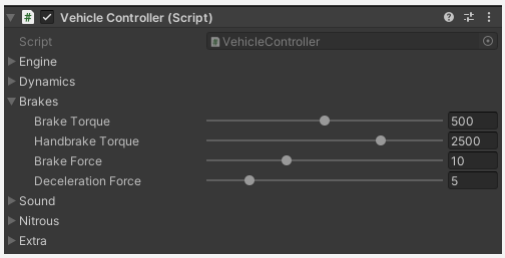
Handling Mode - the handling mode used.

Drivetrain - how the wheels are powered

Anti Roll Amount - the amount of force that helps keeps the vehicle from rolling over.

Downforce Curve - the amount of downward force added as speed increases.

Brakes



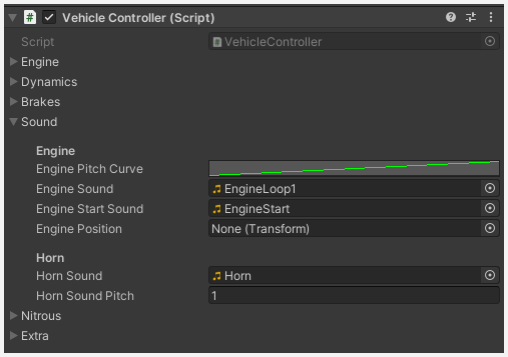
Brake Torque - the amount of brake torque added to the wheels when brake input is maxed out.

Handbrake Torque - the amount of brake torque added to rear axle wheels when handbrake is used

Brake Force - the amount of force added to the rigidbody when braking

Deceleration Force - the amount of force added to the rigidbody when coasting (i.e. when no throttle input is given)

Sound



Engine Pitch Curve - the way the pitch will increase as RPM increases.

Engine Sound - the sound of the engine

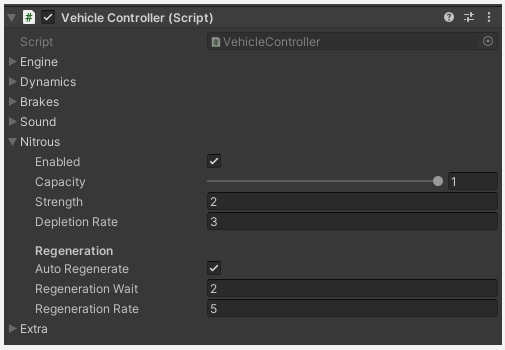
Engine Start Sound - the sound the engine will make when starting up

Engine Position - the position of the engine. The engine sound will come from this position

Horn - the sound of the horn

Horn Sound Pitch - the pitch of the horn sound

Nitrous



Enabled - whether this vehicle will have nitrous

Capacity - the amount of available nitrous

Strength - the strength of the nitrous

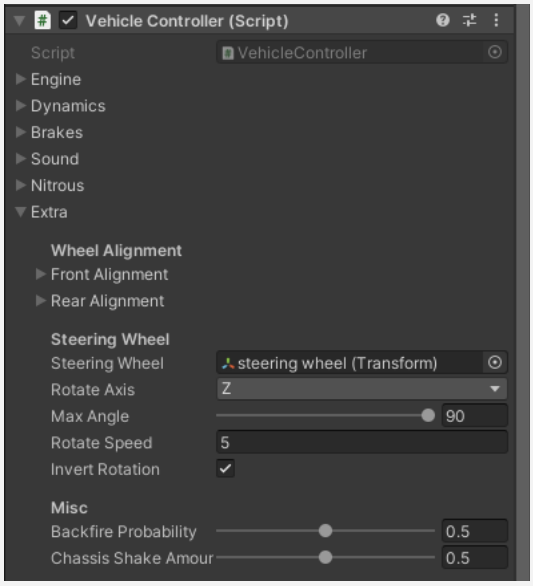
Depletion Rate - how long (in seconds) it takes to deplete all the nitrous

Auto Regenerate - whether the nitrous will regenerate after being used

Regeneration Wait - how long (in seconds) to wait before regeneration starts

Regeneration Rate - how long (in seconds) it will take to fully regenerate the nitrous.

Extra



Front/Rear Alignment - the alignment of the front/rear axle wheels

Steering Wheel - the steering wheel transform of the vehicle

Rotate Axis - the axis the steering wheel will rotate on

Max Angle - the maximum angle the steering wheel can rotate

Rotate Speed - how quickly the steering wheel rotates

Invert Rotation - whether the steering wheel rotation should be inverted

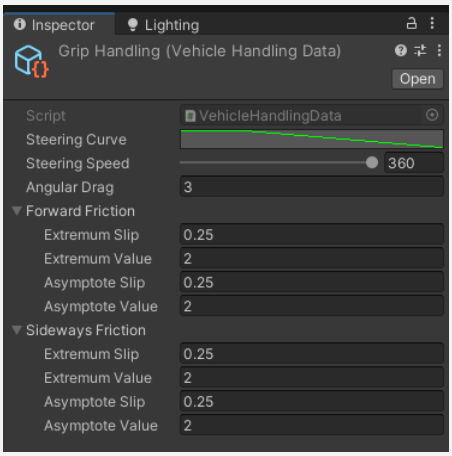
Backfire Probability - the chance of a backfire occurring

Chassis Shake Amount - the amount of chassis shake when stationary revving

Vehicle Handling Data

Vehicle Handling Data defines a vehicle's handling mode. There are 2 kinds of handling modes: Grip and Drift. Handling modes are stored in the Vehicle Settings.

To create a new handling data, select **Assets/Create/RGSK/Vehicle/Vehicle Handling Data**.



Steering Curve - the amount of steer angle at a given speed

Steering Speed - how quickly steer input occurs

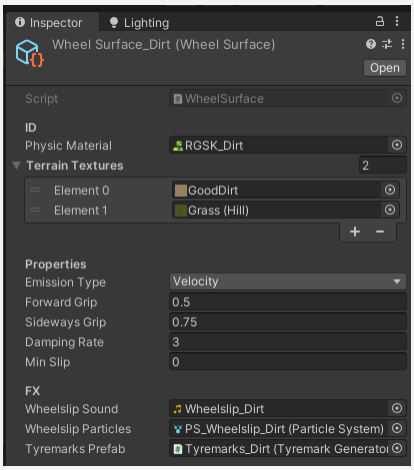
Angular Drag - the angular drag added to the rigidbody. This can help with stability.

Forward/Sideways Friction - the forward/sideways friction curves assigned to the wheel colliders.

Wheel Surfaces

Wheel Surfaces define how a vehicle's wheel will interact with a mesh collider or terrain.

To create a new wheel surface, select **Assets/Create/RGSK/Surface/Wheel Surface** and add it to the Vehicle Settings in the RGSK Menu by adding it to the "Wheel Surfaces" list.



Physic Material - when on a mesh collider, the physic material that identifies this surface

Terrain Textures - when on a terrain, the textures that identify this surface

Emission Type - how FX emission is determined, either by velocity or wheel slip

Forward/Sideways Grip - the wheel collider's forward/sideways stiffness value when on this surface

Damping Rate - how quickly torque is delivered to the wheels on this surface. Low values (e.g. 0.25) are good for asphalt and high values (e.g. 3.0) are good for dirt.

Min Slip - the minimum wheel slip required to generate FX

Wheelslip Sound - the sound produced when the wheel is slipping on this surface

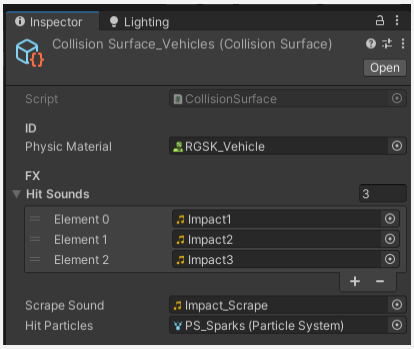
Wheelslip Particles- the particles produced when the wheel is slipping on this surface

Tyremark Prefabs - the tyremarks produced when the wheel is slipping on this surface. For examples, please reference the assets at **Assets/RGSK/Demos/_CommonAssets/Prefabs/FX/Tyremarks**

Collision Surfaces

Collision surfaces determine the FX that are produced when a vehicle collides with something.

To create a new collision surface, **Assets/Create/RGSK/Surface/Collision Surface** and add it to the Vehicle Settings in the RGSK Menu by adding it to the "Collision Surfaces" list.



Physic Material - the physical material that identifies this surface

Hit Sounds - the sounds played on a collision

Scrape Sound - the sound plays when scraping against this surface

Hit Particles - the particles produced on a collision

UI System

The UI system uses uGUI. It works through a persistent central UI Manager that is created automatically at runtime.

UI Manager

The UI Manager is responsible for creating and managing UI screens, Modals, and handling cursor states. It also has an Event System attached to it so you will not need one present in your scenes.

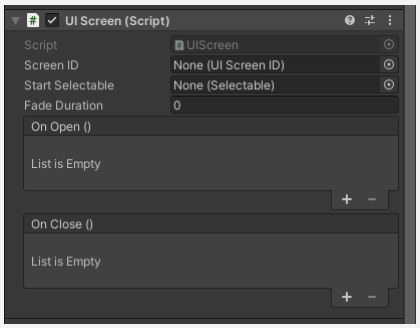
Core

The UI system is comprised of a few sub-systems.

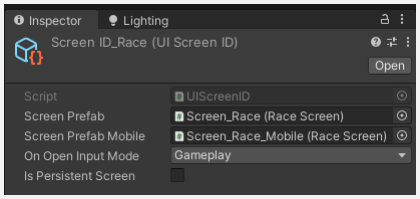
Screens

A screen is a canvas that is managed by the UI Manager. Only 1 screen can be active at a time.

To create a new screen, create a canvas in the hierarchy by selecting **GameObject/UI/Canvas**, and add a **UIScreen.cs** component using the "Add Component" button in the inspector.



Screen ID - this is what is used to identify the screen. To create a new ID, select **Assets/Create/RGSK/UI/UI Screen ID**



> **Screen Prefab** - if the UI Manager attempts to open this screen and it is not found in the scene, it will instantiate this prefab.
>
> **Screen Prefab Mobile** - the screen prefab used on mobile platforms. If null, it will use "Screen Prefab".
>
> **On Open Input Mode** - the input mode used when this screen is opened.
>
> **Is Persistent Screen** - whether this screen will persist through scene loads.

Start Selectable - the button (or other selectable type) to be selected when the screen is opened

Fade Duration - the amount of time to fade in the screen when opened. Leave it to 0 for no fading.

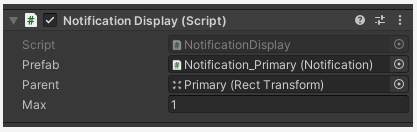
On Open/Close events - Unity event listeners for when the screen is opened or closed.

Notifications

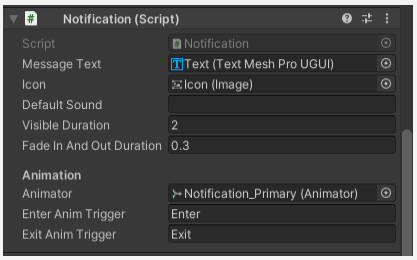
A notification is a message box that appears on the screen for a duration of time.

Notifications work in 2 parts:

Notification Display - this creates and manages the notification (see below)



Notification - this is the prefab used by the Notification Display.

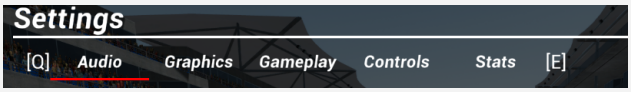


For examples of how notifications are used, reference the following assets:

Assets/RGSK/Demos/_CommonAssets/Prefabs/UI/Notifications/

Assets/RGSK/Demos/_CommonAssets/Prefabs/UI/Screens/Screen_Race.prefab

Tab System



The tab system works in 3 parts

Tab Controller - manages the activation of tabs

Tab Button - handles which tab to activate when clicked or selected

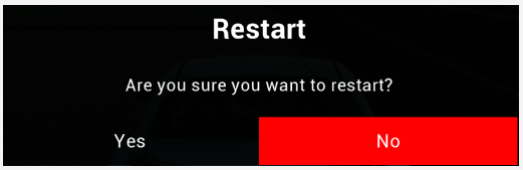
Tab - this is the game object that contains the tab's content and is activated/deactivated by the tab controller.

For examples of how the tab system is used, reference the following assets:

Assets/RGSK/Demos/_CommonAssets/Prefabs/UI/Screens/Screen_Settings.prefab

Assets/RGSK/Demos/_CommonAssets/Prefabs/UI/Misc/TabController.prefab

Modal Windows



Modal windows are managed by the Modal Window Manager which is attached to the UI Manager.

Modal window prefabs are stored in the RGSK Menu under "UI Settings".

For examples of how modal windows are used, reference the following assets:

Assets/RGSK/Demos/_CommonAssets/Prefabs/UI/Modal/

Displaying Entity Data

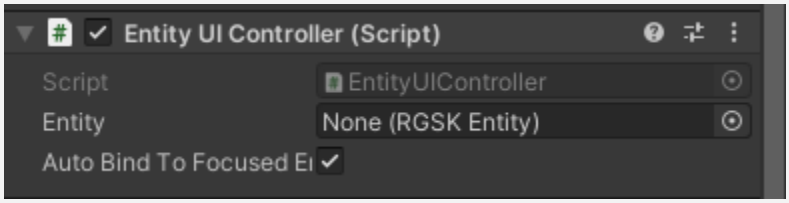
Entity data (such as race position, vehicle speed, lap time, drift points, profile name, nationality, etc) is displayed by an **Entity UI Controller** and an **Entity UI Component**.

An Entity UI Controller is what manages each child Entity UI Component and so should be at the root of your UI and the Entity UI Components should be under it as follows:

–EntityUIController

—EntityUIComponent (e.g CompetitorUI.cs, DriftUI.cs VehicleUI.cs, and ProfileUI.cs)

Entity UI Controller



Entity - the entity to read data from.

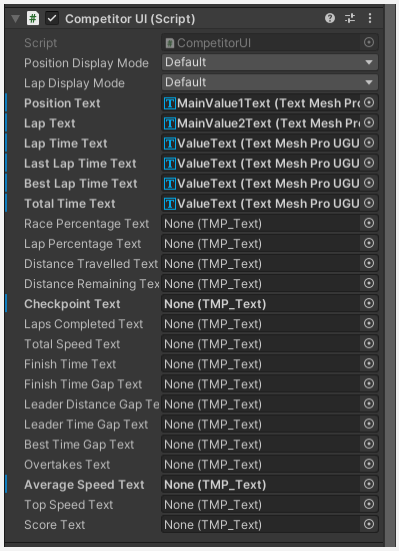
Auto Bind To Focused Entity - whether the child EntityUIComponents should display data of the currently focused entity by the Camera Manager. This should be set to true for shared UI such as the race screen.

Entity UI Components

These are the scripts that inherit from EntityUIComponent.cs. They include CompetitorUI.cs, DriftUI.cs VehicleUI.cs, and ProfileUI.cs. These scripts can be combined to show different kinds of data all within a single panel.

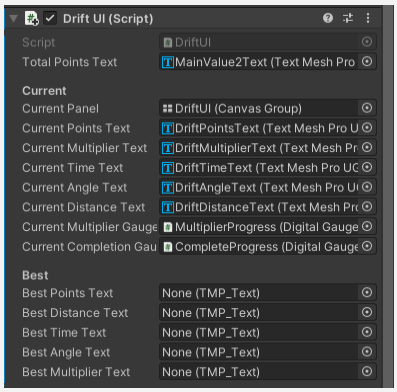
Competitor UI

Competitor.cs is responsible for displaying all race competitor information such as race position, lap time, and more.



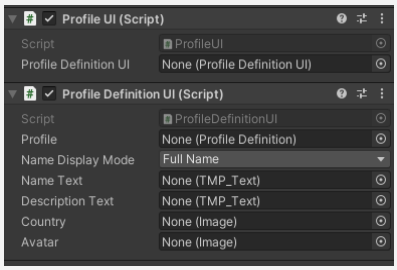
Drift UI

DriftUI.cs is responsible for displaying all drift information.



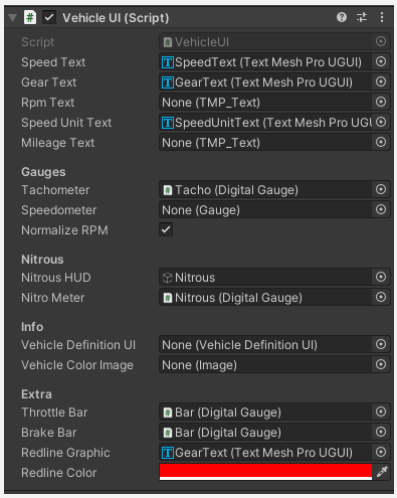
Profile UI

ProfileUI.cs is responsible for displaying all profile information. It works with **ProfileDefinitionUI.cs** which handles displaying the information.



Vehicle UI

VehicleUI.cs is responsible for displaying all vehicle information.



For examples of vehicle UI prefabs, reference the following assets:

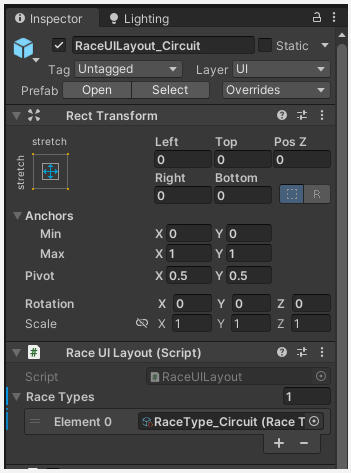
Assets/RGSK/Demos/_CommonAssets/Prefabs/UI/Elements/VehicleUI_Analog.prefab

Assets/RGSK/Demos/_CommonAssets/Prefabs/UI/Elements/VehicleUI_Compact.prefab

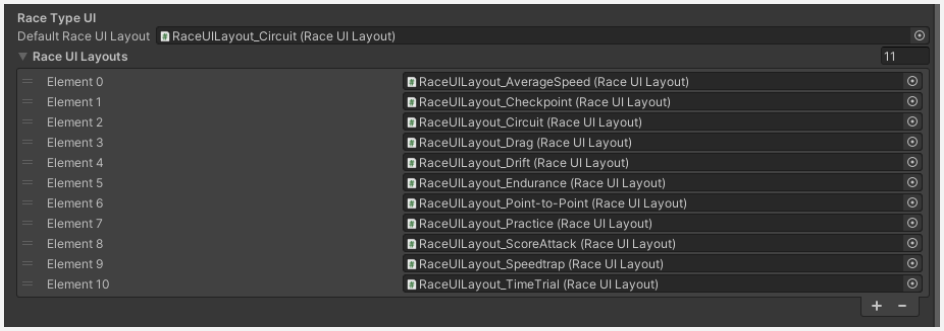
Assets/RGSK/Demos/_CommonAssets/Prefabs/UI/Elements/VehicleUI_Radial.prefab

Race Type Specific UI

You may require different race types to display different UI layouts. This is the responsibility of the RaceUILayout component:



Attach the **RaceUILayout.cs** script to your UI prefab, assign what **Race Types** it will be used in, and then assign it to the list in the RGSK Menu under “UI Settings”. This is where all race UI layouts should be stored:



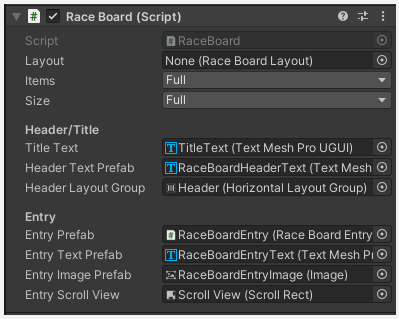
For examples of how race UI layouts are used, reference the following assets:

Assets/RGSK/Demos/_CommonAssets/Prefabs/UI/RaceTypeLayouts/

Raceboards

Race boards are responsible for showing important race data. A race board is made up of **2 main components**:

1. Raceboard



The Race board is responsible for creating and ordering entries within the board.

Layout - the race board layout used by this board. This is automatically assigned by [Race Screens] depending on the race type

Items - which set of items defined in the board layout will this board display

Size - “Full” will show all entries. “Mini” will show 4 entries - the top one will show 1st place, and the others will show the player and neighboring competitors.

Title Text - the text that shows the title of the board set in the race board layout.

Header Text Prefab - the text prefab used for the headers

Header Layout Group - the layout group that the header texts will be instantiated in.

Entry Prefab - the race board entry used for this board (see below).

Entry Text Prefab - the text prefab used for cells in the entry to show text

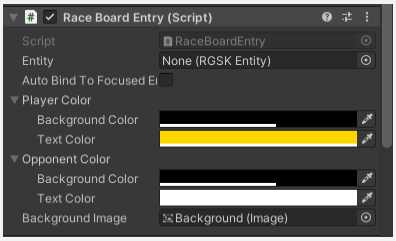
Entry Image Prefab - the image prefab used for cells in the entry to show images. These are shared by the headers.

Entry Scroll View - the scroll view that the entries will be instantiated in.

2. Raceboard Entry

A race board entry is simply an empty prefab with a **RaceboardEntry.cs** component and a horizontal layout group. It is auto-populated at runtime.

See the prefab under **Assets/RGSK/Demos/_CommonAssets/Prefabs/UI/Boards/Race/RaceBoardEntry.prefab** for an example.



Entity - the entity to read data from. This is auto-assigned.

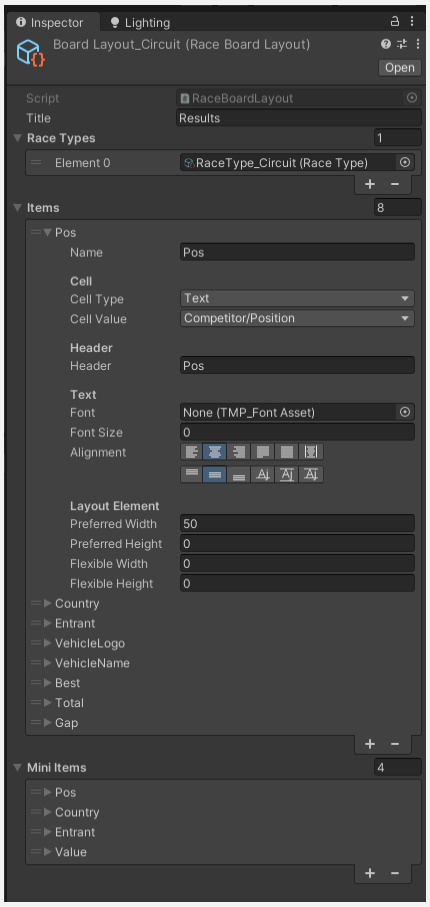
Auto Bind To Focused Entity - this is auto-set to false at runtime.

Player/Opponent Color - the colors used by player/opponent entries.

Background Image - the image that is affected by the **Player/Opponent** background colors

Board Layouts

A board layout defines how a race board will be drawn. To create a new layout, select **Assets/Create/RGSK/UI/Race Board Layout**



Title - the text displayed by the board's title text (see Raceboards)

Race Types -the race type(s) that this board layout will be used in

Items - the items that will be displayed on the board. Each item has its own properties

- **Cell type -** the type of cell, either text or image
- **Cell value** - the value displayed by this cell
- **Header** - the text displayed by the header text for this item
- **Text/Layout Element** - the item's text and layout properties

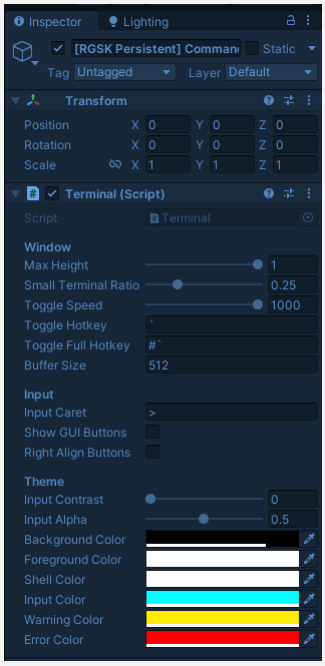
Mini Items - the items used for "Mini" item race boards

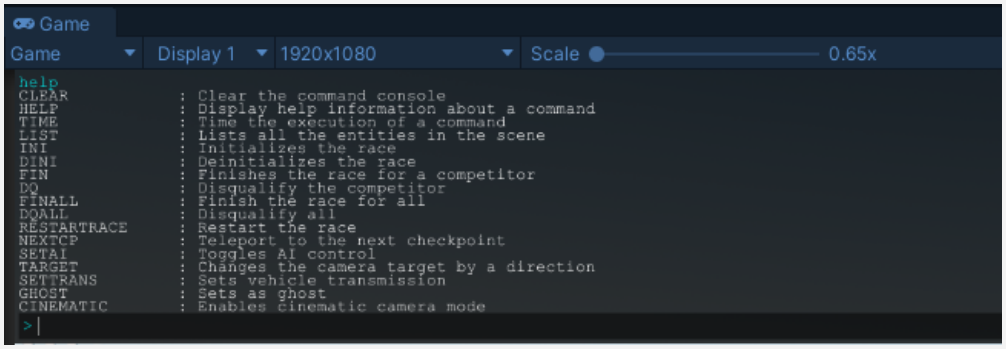
For examples of how race board layouts are used, reference the following assets:

Assets/RGSK/Core/ScriptableObjects/UI/RaceBoardLayouts

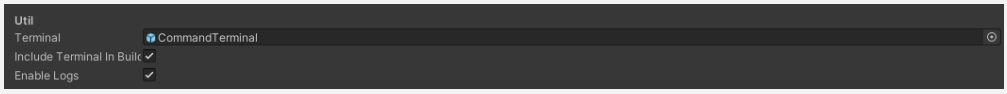
Terminal

The terminal is used to give commands that may help to speed up development. Press “~” to open/close the terminal, and type “help” to see all available commands.





Terminal Settings can be found in the RGSK Menu under “General Settings”



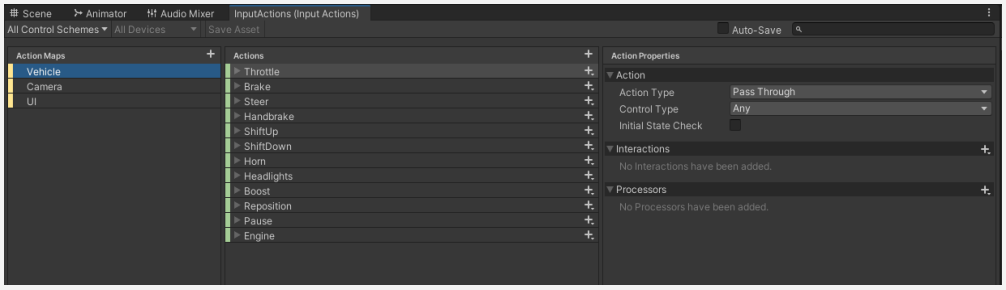
The terminal is provided by [stillwwater’s command terminal] under the MIT license.

Input System

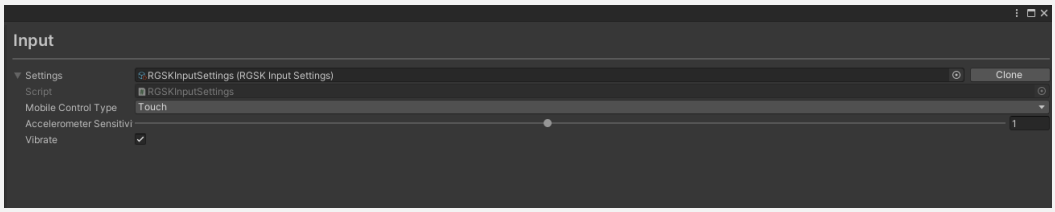
The input system uses the Unity Input System. It works through a persistent central Input Manager that is created automatically at runtime.

The input actions can be found under:

Assets/RGSK/Core/Input/InputActions/InputActions.inputactions



Some input settings can be found in the RGSK Menu

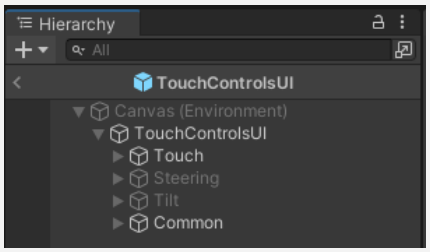


Mobile Control Type -the control type to use on mobile. The MobileUIController will automatically switch out UI controls to match.

Accelerometer Sensitivity - how sensitive the accelerometer will be when using the “Tilt” mobile control type.

Vibrate - enables rumble on gamepads.

Mobile Controls



Mobile controls mainly work through the Unity Input System’s **OnScreenButton.cs**

The **MobileUIController.cs** is responsible for activating sub-panels to match the Mobile Control Type (see above).

Please reference the prefab at **Assets/RGSK/Demos/_CommonAssets/Prefabs/UI/Elements/TouchControlsUI.prefab** to see how mobile controls are set up.

Audio System

The audio system works through a central Audio Manager that is created automatically at runtime.

Sound List

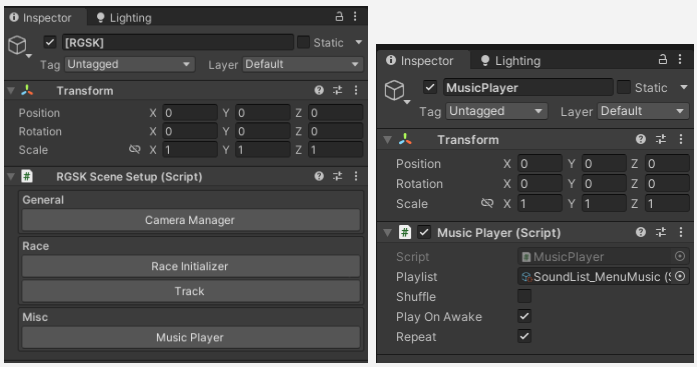
A sound list is where sounds are stored. To create a sound list, select **Assets/Create/RGSK/Sound List**

Music Manager

The music manager is responsible for playing music. It works by adding a Music Player to the scene or by directly assigning a playlist through code.

Music Player

To create a music player, select **Music Player** in the RGSK Scene Setup.



Playlist - the Sound List of music tracks

Shuffle - if true, the playlist will play in a random order

Play On Awake - whether music should play once the scene loads

Repeat - whether the playlist should loop

Scene Management

Scenes are loaded through a central scene manager class (SceneLoadManager.cs)

To load a new scene, call the **RGSK.SceneLoadManager.LoadScene()** method. All scenes must be loaded using this method because it invokes events that handle the unloading of certain persistent objects, which maintains proper scene management and resource handling.

When building your project, please ensure that the "Main Menu" and "Loading" scenes (configured in Scene Settings) are assigned and part of your build settings.

The Loading Scene

This is the scene which is responsible for transitioning to other scenes. It is configured in Scene Settings and must be assigned for scene managment to work correctly.

This scene must contain a "SceneLoadManager.cs" component to work. Please reference the scene at "**Assets/RGSK/Demos/_CommonScenes/Loading.unity**" for an example of how to setup your own loading scene.

When a call is made to load a new scene, the following steps occur:

1. The UI system opens the loading screen (configured in UI Settings)
2. The loading scene is loaded
3. The loading scene's "SceneLoadManager.cs" component handles the transition to the new scene.

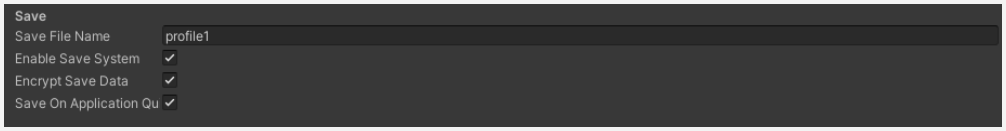
The Main Menu Scene

This is the scene that is loaded when a call to **RGSK.SceneLoadManager.LoadMainScene()** is made. It is configured in Scene Settings and must be assigned for scene managment to work correctly.

Save System

The save system uses Newtonsoft JSON. It works through a persistent Save Manager that is created automatically at runtime.

The save system settings can be found in the RGSK Menu under “General Settings”



Save File Name - the name of the save file. The path will be under the *application persistent data path/SaveData/Profiles/*

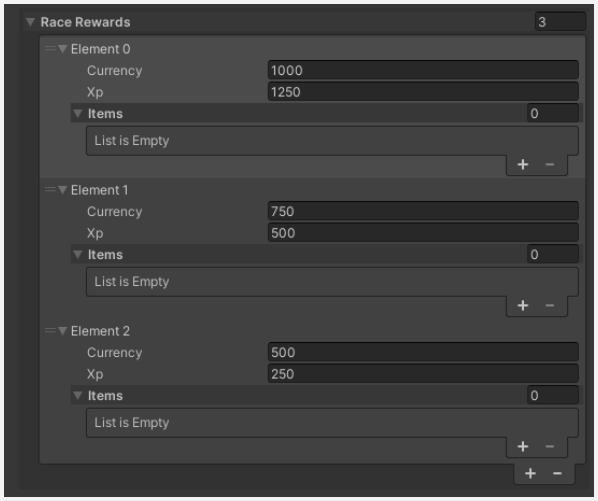
Enable Save System - whether the save system should be enabled

Encrypt Save Data - if true, the JSON save file will be encrypted and no longer be human-readable

Save on Application Quit - whether the save system should perform a save on application quit

Reward System

The reward system works by giving items, currency, and XP rewards per finishing position. Rewards are assigned in a Race Session.



A reward is given per finishing position. Rewards for finishing **1st will be Element 0**, **2nd will be Element 1**, and so on.

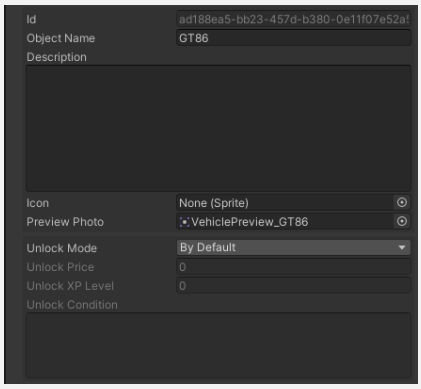
Currency - the amount of currency given

XP - the amount of XP gained

Items - the items that will be unlocked

Items

Items are unique scriptable objects that can be unlocked by the Reward System.



Each item has the following fields:

ID - an automatically generated guid for this item.

Object Name - the name of this item.

Description - the description of this item

Icon - the icon of this item used by the UI.

Preview Photo - the photo of this item used by the UI.

Unlock Mode - how the item is unlocked:

- **None** - the item will be unlocked by manually
- **By Default** - the item will be unlocked by default
- **Purchase** - the item will be unlocked upon purchase
- **XP Level -** the item will be unlocked upon reaching a specified XP level

Unlock Price - how much the item costs

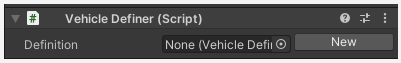
Unlock XP Level - the XP level at which the item will be unlocked

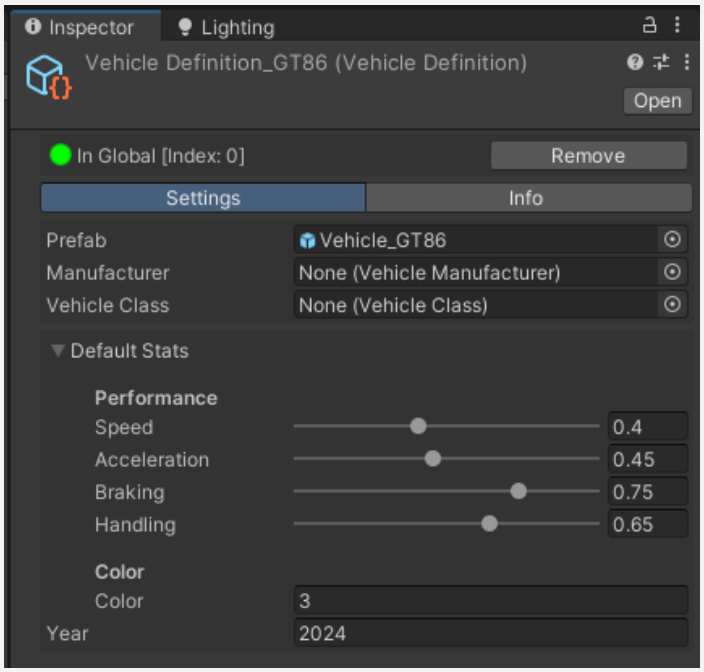
Unlock Condition - how the item will be unlocked. This string is displayed by the UI if it's not left empty and if **Unlock Mode** is set to "None"

Vehicle Definition

A vehicle definition is a scriptable object that holds a vehicle's data.

To create one, either select **Assets/Create/RGSK/Vehicle/Vehicle Definition** or use the Vehicle Definer attached to each vehicle:





Prefab - the prefab of this vehicle

Manufacturer - the Manufacturer of this vehicle. To create a new manufacturer, select **Assets/Create/RGSK/Vehicle/Vehicle Manufacturer**

Vehicle Class - the class of this vehicle. To create a new vehicle class, select **Assets/Create/RGSK/Vehicle/Vehicle Class**

Default Stats - the default stats of this vehicle.

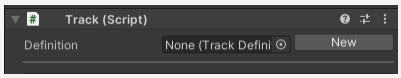
Color - the default color/livery index of this vehicle. This is the index of the color in the vehicle color list or texture in a livery list. See Vehicle Settings

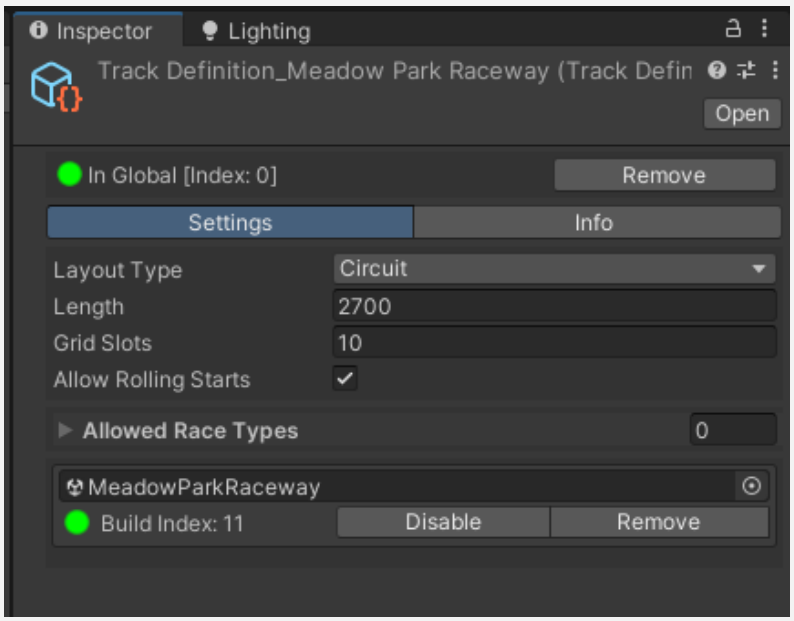
Year - the year of this vehicle. Used by the UI when displaying the full vehicle name (e.g. Toyota GT86 '24). This only applies when the Manufacturer is assigned.

Track Definition

A track definition is a scriptable object that holds a track's data.

To create one, either select **Assets/Create/RGSK/Track Definition** or use the Track component attached to each track:





Scene - the scene this track in

Country - the country the track is in. Used by the UI when displaying track info

Minimap Preview - a preview of the minimap. Used by the UI when displaying track info

Layout Type - whether the track is a circuit or point to point

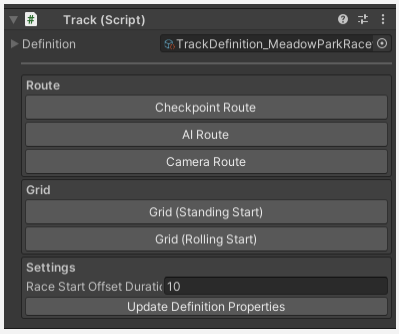
Length - the length of the track

Grid Slots - how many grid positions are available in this track

Allow Rolling Starts - whether this track has a rolling start grid assigned

Allowed Race Types - the race types that are allowed on this track. Leave empty to allow for all race types.

Layout Type, Length, Grid Slots & Allow Rolling Starts can be automatically set by using the "Update Definition Properties" button in the Track component:

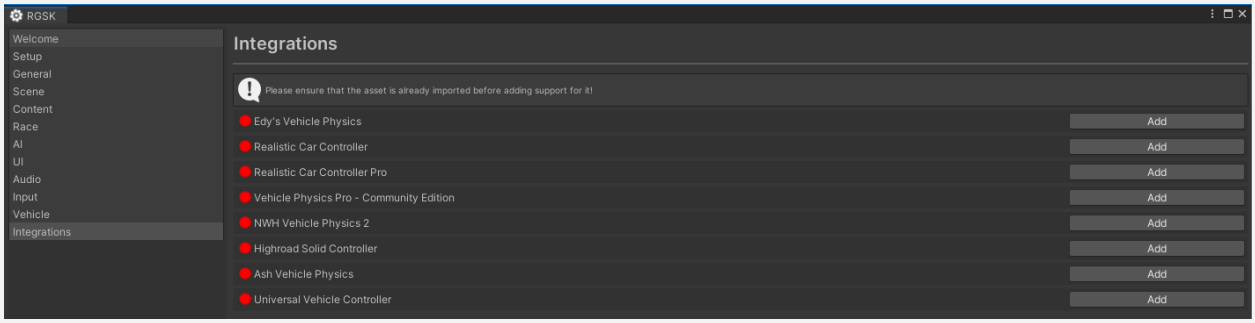


3rd-party Integrations

Please note that the integrations may not be perfect, and some loss of functionality is possible due to each integration using RGSK systems (e.g. Input and Camera systems). The AI behavior may also require adjustments to ensure compatibility with the new physics integration.

Before proceeding with any integration, please ensure the corresponding asset is imported to your project!

Open the RGSK Menu and select the “Integrations” tab and use the “Add” button to add support for the available integrations.



After clicking the “Add” button, please wait until script compilation is finished:

When compilation is finished, confirm that the integration was successful by checking if the ⚫red icon has turned green.

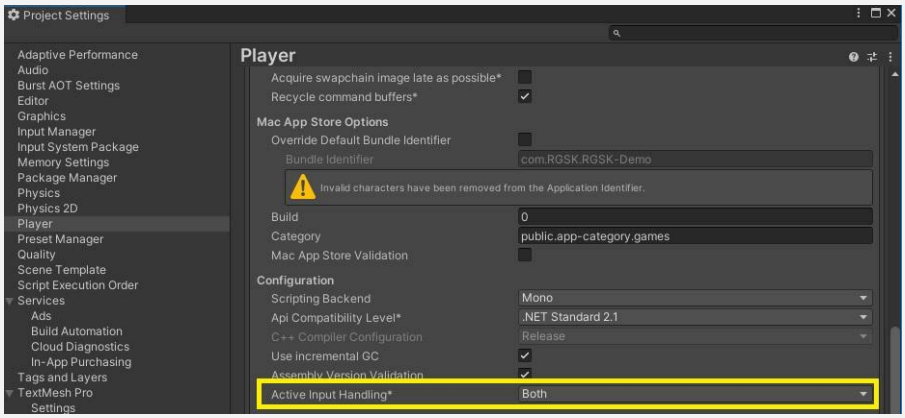
Custom Vehicle Physics

If you are using custom vehicle physics that are not part of the integration list, please follow the steps below.

1. Create a new script and copy/paste the contents from CustomVehiclePhysicsTemplate.cs (**Assets/RGSK/Core/Scripts/Runtime/Integrations/_Custom**) and make all neccessary modifications to suit your vehicle physics.
2. Drag the vehicle into the playground scene (**Assets/RGSK/Demos/_CommonScenes/Showcase/Playground.unity**)
3. Follow the instructions under Vehicle Setup, selecting the "**Custom**" tab
4. Add the integration script created in step (1) to the vehicle.
5. Test the vehicle. See Testing the Vehicle for more information.

Edy’s Vehicle Physics

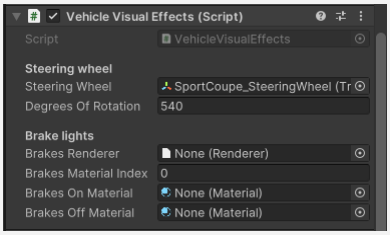
1. Enable “Edy’s Vehicle Physics” by clicking on the “Add” button in the integration menu.
2. Set the **Active Input Handling** to **Both** in the project settings (**Edit/Project Settings/Player**):



3. Drag your EVP vehicle into the playground scene (**Assets/RGSK/Demos/_CommonScenes/Showcase/Playground.unity**)
4. Follow the instructions under Vehicle Setup, selecting the "**EVP**" tab.
5. Test the vehicle. See Testing the Vehicle for more information.

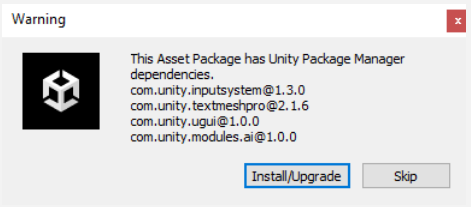
Known Issues

- The Vehicle Visual Effects component causes visual issues due to RGSK's rutime manipulation of materials. To resolve, either remove the component or unassign the material fields:



Realistic Car Controller

1. RGSK already uses these packages, so you can safely skip:

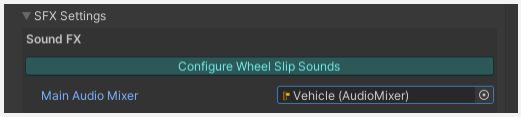


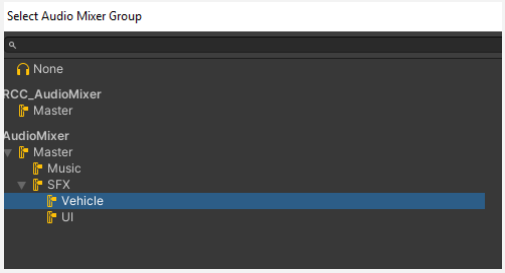
2. Enable “Realistic Car Controller” by clicking on the “Add” button in the integration menu.
3. To prevent RCC inputs from running alongside RGSK inputs, open RCC_CarControllerV4.cs and add “**|| overrideInputs**” to **all the if statements in the event methods** from **RCC_InputManager_OnTrailerDetatch()** to **RCC_InputManager_OnStartStopEngine()**. In RCC v4.0, these methods should be on lines 2303 to 2402:

```
private void RCC_InputManager_OnTrailerDetach()
{

    if (!canControl || externalController || overrideInputs)
        return;

    DetachTrailer();

}

private void RCC_InputManager_OnGearShiftDown()
{

    if (!canControl || externalController || overrideInputs)
        return;
```

```
private void RCC_InputManager_OnInteriorLights()
{

    if (!canControl || externalController || overrideInputs)
        return;

    interiorLightsOn = !interiorLightsOn;

}

private void RCC_InputManager_OnStartStopEngine()
{

    if (!canControl || externalController || overrideInputs)
        return;
```

4. Open RCC Settings and set **Main Audio Mixer** to "Vehicle":





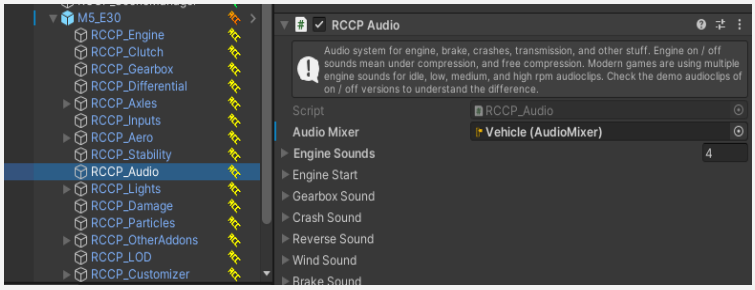
5. Drag the vehicle into the playground scene (**Assets/RGSK/Demos/_CommonScenes/Showcase/Playground.unity**)
6. Follow the instructions under Vehicle Setup, selecting the "**RCC**" tab.
7. Test the vehicle. See Testing the Vehicle for more information.

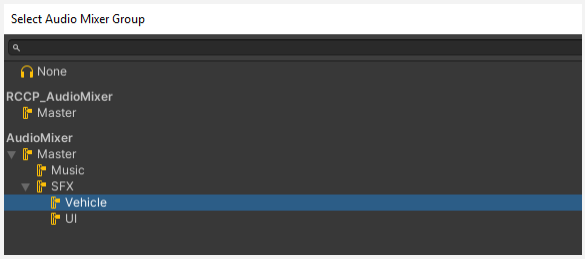
Known Issues

- The RCC demo vehicles may clip the ground on spawn, causing physics bugs. To resolve, raise all spawn points (i.e menu vehicle spawn point and race grid positions) by roughly 0.5.

Realistic Car Controller Pro

1. Enable “Realistic Car Controller Pro” by clicking on the “Add” button in the integration menu.
2. Drag the vehicle into the playground scene (**Assets/RGSK/Demos/_CommonScenes/Showcase/Playground.unity**)
3. Follow the instructions under Vehicle Setup, selecting the "**RCCPro**" tab.
4. Set the vehicle’s audio mixer group to "Vehicle":





5. Test the vehicle. See Testing the Vehicle for more information.

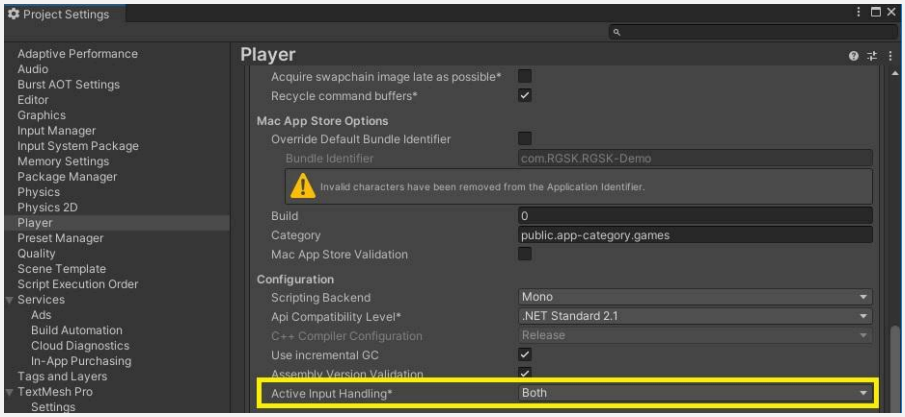
Known Issues

- The RCCP demo vehicles may clip the ground on spawn, causing physics bugs. To resolve, raise all spawn points (i.e menu vehicle spawn point and race grid positions) by roughly 0.5.
- During replay playback, the vehicle rigidbodies are set to kinematic as a workaound for performance issues. This disables collision effects and causes wheel rotation issues.

Vehicle Physics Pro - Community Edition

Please note that the community edition only supports 1 vehicle per scene. This will cause issues if you try to use multiple VPP vehicles, including ghost vehicles

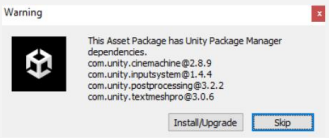
1. Enable “Vehicle Physics Pro - Community Edition” by clicking on the “Add” button in the integration menu.
2. Set the **Active Input Handling** to **Both** in the project settings (**Edit/Project Settings/Player**):



3. Drag the vehicle into the playground scene (**Assets/RGSK/Demos/_CommonScenes/Showcase/Playground.unity**)
4. Follow the instructions under Vehicle Setup, selecting the "**VPP**" tab.
5. Test the vehicle. See Testing the Vehicle for more information.

NWH Vehicle Physics 2

1. RGSK already uses these packages, so you can safely skip:



2. Enable “NWH Vehicle Physics 2” by clicking on the “Add” button in the integration menu.
3. In order for the RGSK audio system to work well with NWH, open EngineRunningComponent.cs (**Assets\NWH\Vehicle Physics 2\Scripts\VehicleController\Sound**) and comment out the following line in the VC_Update() method as follows:

```
125             _distortion = Mathf.SmoothDamp(_distortion, newDistortion, ref _distortionVelocity, smoothing);
126             //source.outputAudioMixerGroup.audioMixer.SetFloat("engineDistortion", _distortion);
```

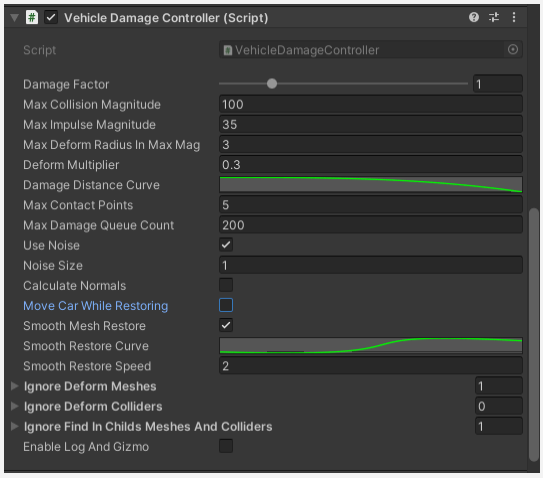
4. Drag the vehicle into the playground scene (**Assets/RGSK/Demos/_CommonScenes/Showcase/Playground.unity**)
5. Follow the instructions under Vehicle Setup, selecting the "**NWH2**" tab.
6. Test the vehicle. See Testing the Vehicle for more information.

Known Issues

- NWH2 dashboard GUI only works with the active car in the scene which causes the readings to be incorrect in other vehicles. To fix this, use the RGSK vehicle dashboard instead.
- The vehicle dashboard UI sometimes freezes, causing the TMP text to no longer update visually.
- There are some spatial audio issues when multiple cars are in the scene.

Universal Vehicle Controller

1. Enable “Universal Vehicle Controller” by clicking on the “Add” button in the integration menu.
2. Drag the vehicle into the playground scene (**Assets/RGSK/Demos/_CommonScenes/Showcase/Playground.unity**)
3. Follow the instructions under Vehicle Setup, selecting the "**UVC**" tab.
4. Uncheck "**Move Car While Restoring**" for all vehicles. Leaving this checked will cause critical issues in race scenes.



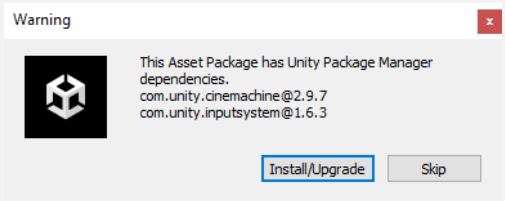
5. Test the vehicle. See Testing the Vehicle for more information.

Known Issues

- During replay playback, the vehicle rigidbodies are set to kinematic as a workaound for performance issues. This disables collision effects.
- During replay playback, nitrous effects will not always play.

Ash Vehicle Physics

1. RGSK already uses these packages, so you can safely skip:



2. Enable “Ash Vehicle Physics” by clicking on the “Add” button in the integration menu.
3. Drag the vehicle into the playground scene (**Assets/RGSK/Demos/_CommonScenes/Showcase/Playground.unity**)
4. Follow the instructions under Vehicle Setup, selecting the "**ASHVP**" tab.
5. Test the vehicle. See Testing the Vehicle for more information.

Sim-Cade Vehicle Physics

1. Enable "Sim-Cade Vehicle Physics" by clicking on the “Add” button in the integration menu.
2. Drag the vehicle into the playground scene (**Assets/RGSK/Demos/_CommonScenes/Showcase/Playground.unity**)
3. Follow the instructions under Vehicle Setup, selecting the "**SCVP**" tab.
4. Test the vehicle. See Testing the Vehicle for more information.

Known Issues

- The NitroBoost component is not supported due to incompatible input handling.
- The demo vehicles may clip the ground on spawn, causing physics bugs. To resolve, raise all spawn points (i.e menu vehicle spawn point and race grid positions) by roughly 1.0 on the Y-axis.

Highroad Solid Controller

1. Enable “Highroad Solid Controller ” by clicking on the “Add” button in the integration menu.
2. Open SoundManager.cs (**Assets\HighroadEngine\Common\Scripts\Managers\SoundManager.cs**) and add "**audioSource.outputAudioMixerGroup = RGSK.Helpers.AudioHelper.GetAudioMixerGroup(RGSK.AudioGroup.SFX.ToString());**" in the PlaySound() and PlayLoop() methods as follows:

```
114     // we set the audio source volume to the one in parameters
115     audioSource.volume = SfxVolume;
116     audioSource.outputAudioMixerGroup = RGSK.Helpers.AudioHelper.GetAudioMixerGroup(RGSK.AudioGroup.SFX.ToString());
```

```
81     // we set the audio source volume to the one in parameters
82     audioSource.volume = SfxVolume;
83     audioSource.outputAudioMixerGroup = RGSK.Helpers.AudioHelper.GetAudioMixerGroup(RGSK.AudioGroup.SFX.ToString());
```

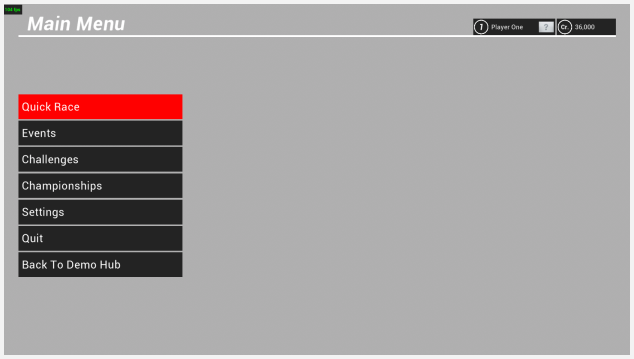
3. Drag the vehicle into the playground scene (**Assets/RGSK/Demos/_CommonScenes/Showcase/Playground.unity**)
4. Follow the instructions under Vehicle Setup, selecting the "**HSC**" tab.
5. Test the vehicle. See Testing the Vehicle for more information

Main Menu System

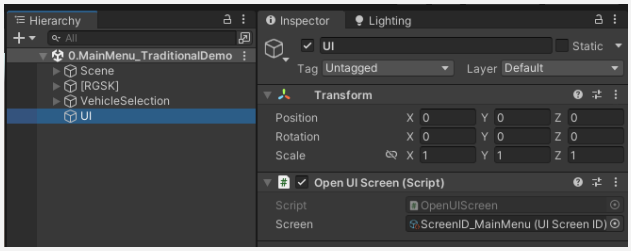
The main menu system offers you a good starting point for your game. To see the main menu system in action, please reference the **Traditional Demo** (at path **Assets/RGSK/Demos/CompleteDemos/Traditional/**)

The scripts used for the menu screens can be found under **Assets/RGSK/Core/Scripts/Runtime/UI/Screens/Menu/**

Main Screen

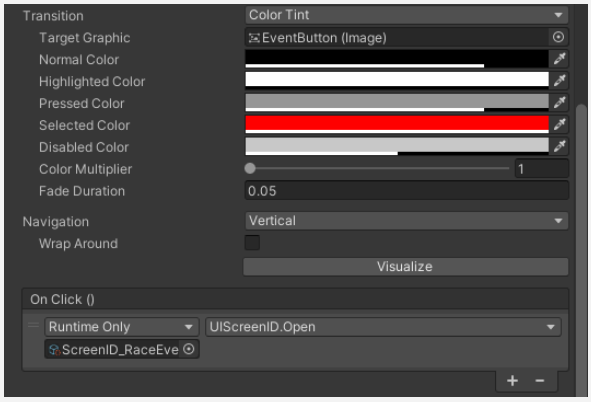


This is the screen that is first loaded. It is loaded by the OpenUIScreen.cs component that is attached to the “UI” object in the hierarchy. The UI Manager is then responsible for creating the screen **if it is not already in the scene**.



The “Quick Race” button makes a call to **GeneralHelper.OpenVehicleSelectScreenWithCallback** which opens the Vehicle Select Screen with a callback that opens the Track Select Screen when a vehicle is selected which then opens the Race Setting Screen when a track is selected.

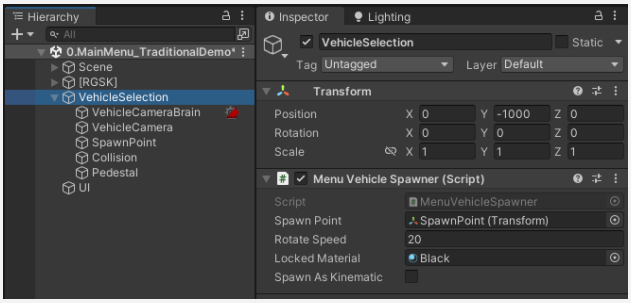
The “Events”, “Challenges”, “Championships”, and “Settings” buttons make calls UIScreenID.Open() in their OnClick method to open the desired UI Screen:



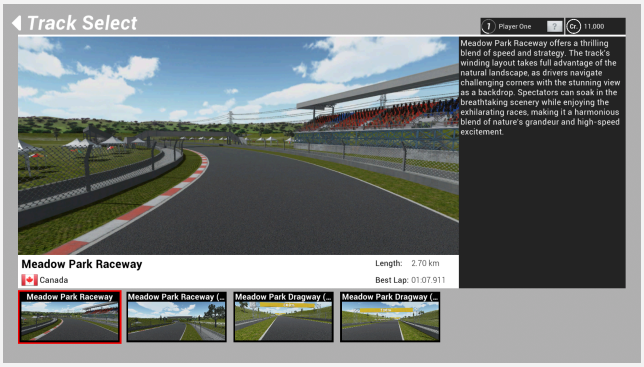
Vehicle Select Screen

The vehicle select screen loads all vehicles from the Content Settings and displays them. The screen has an “OnSelected” Unity event which determines what happens when a vehicle is selected.

The vehicle is instantiated by a Menu Vehicle Spawner component, which works hand-in-hand with the UI Screen code. The vehicle is displayed on the UI using a Render Texture that the “VehicleCameraBrain” outputs. Below is the structure:



Track Select Screen

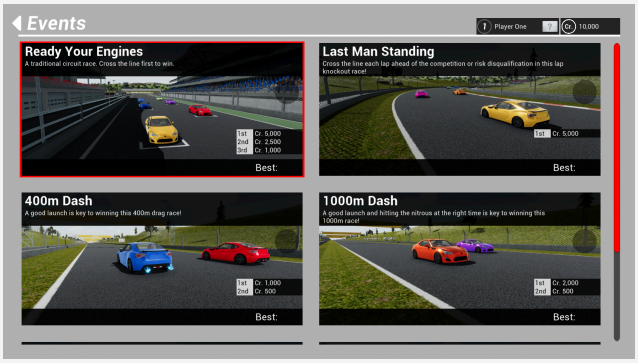


The track select menu loads all tracks from the Content Settings and displays them. The screen has an “OnSelected” Unity event which determines what happens when a track is selected.

Race Settings Screen

The race settings allow you to create custom settings for the race session. The settings available will vary based on the selected track or race type.

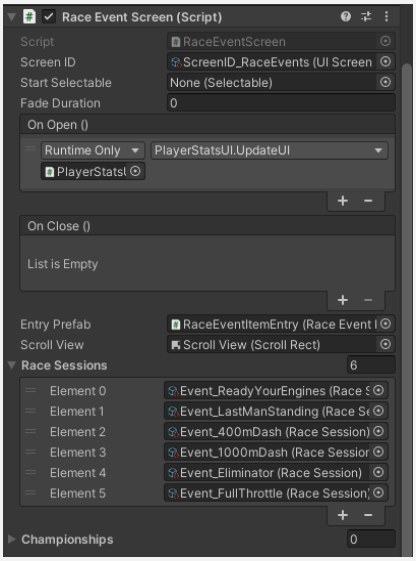
Events Screen



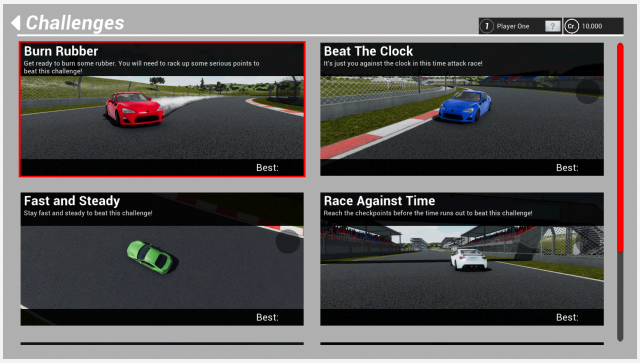
The events screen displays and loads predefined race sessions.

These race sessions allow for the player to select a custom vehicle because “Auto Populate Player” is enabled in each of the race sessions. This causes the UI code to make a call to **GeneralHelper.OpenVehicleSelectScreenWithCallback()** which opens the Vehicle Select Screen with a callback that starts the event when a vehicle is selected.

The displayed events are stored in the “Race Sessions” list in the UI Screen prefab at **Assets/RGSK/Demos/_CommonAssets/Prefabs/UI/Screens/Screen_RaceEvents.prefab**

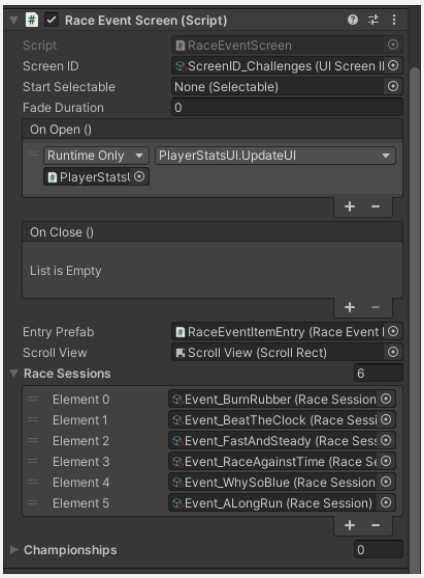


Challenges Screen

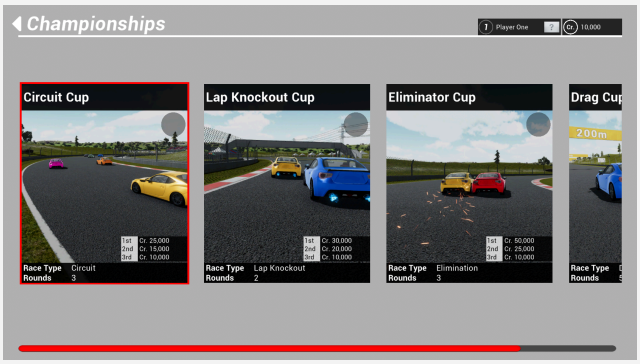


The challenges screen displays and loads predefined race sessions. These race sessions restrict the player from selecting a custom vehicle because “Auto Populate Player” is disabled in each of the race sessions.

The displayed challenges are stored in the “Race Sessions” list in the UI Screen prefab at **Assets/RGSK/Demos/_CommonAssets/Prefabs/UI/Screens/Screen_Challenges.prefab**



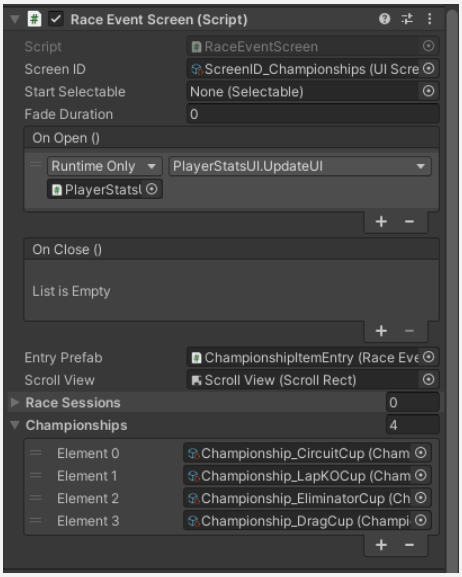
Championship Screen



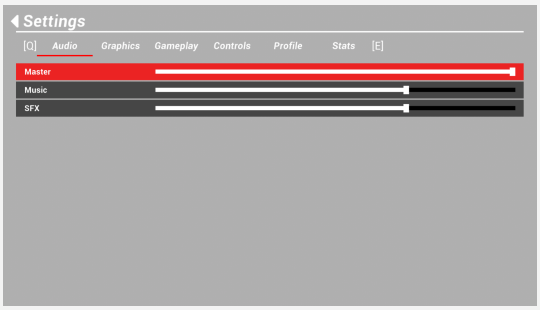
The championship screen displays and loads championships.

Each championship allows for the player to select a custom vehicle because “Auto Populate Player” is enabled in each of the championships. This causes the UI code to make a call to **GeneralHelper.OpenVehicleSelectScreenWithCallback()** which opens the Vehicle Select Screen with a callback that starts the championship when a vehicle is selected.

The displayed challenges are stored in the “Championships” list in the UI Screen prefab at **Assets/RGSK/Demos/_CommonAssets/Prefabs/UI/Screens/Screen_Championships.prefab**



Settings Screen



The settings screen provides a way to customize gameplay experience, including audio, graphics, and other settings. All values are saved/loaded by the Save Manager.

Support and Community

For any questions, please refer to the links below:

Email Contact | **Unity Forum**

Join the **Discord Server** to connect with other users, share what you're working on, get support, provide feedback, submit bug reports & feature requests, and keep up-to-date with the development of RGSK.

Check out the **YouTube** channel for tutorials and other video content about RGSK.

FAQ

Why are the demo scenes so dark when playing in the Unity Editor?

This is because lighting hasn't been generated for the scenes. Lighting can be generated through **Window/Rendering/Lighting**.

Why am I getting poor performance on mobile devices?

This is most likely caused by the default post-processing effects. To remove them, open the RGSK Menu, go to the "General" tab, and remove "PostProcessing_Default" from the "Persistent Objects" list.