Racing Game Starter Kit

Version 1.1.0a

This documentation will guide you on how to use the Racing Game Starter Kit to make your own cool racing games in Unity.

You can check out video tutorials from this link. Please note that the video tutorials may not always be up-to-date with the current version.

For any questions, suggestions or complaints feel free to contact me at ian.izzy94@gmail.com and I will respond promptly.

Let's Get Started!

TABLE OF CONTENTS

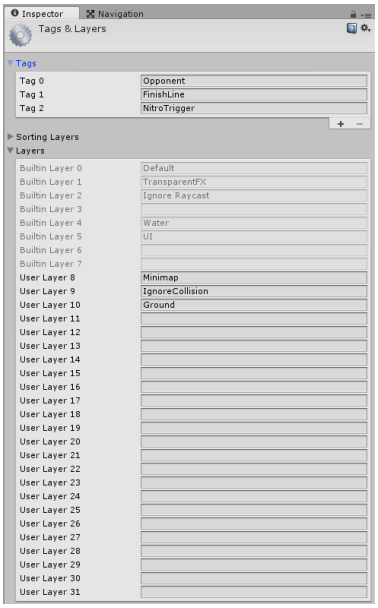
GETTING STARTED

Before using the Racing Game Starter Kit, you may need to make some changes to your Project's

Tags & Layers, **Input** and **Physics** settings.

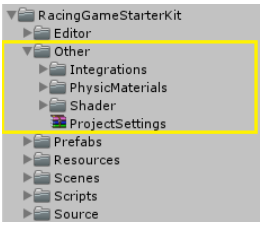
This may be done automatically when importing the Racing Game Starter Kit from the asset store.

Go to your Tags & Layers (*Edit/Project Settings/Tags and Layers*)



If your Tags & Layers don't contain the above, then please do the following:

1) Go to the RacingGameStarterKit / Others folder:



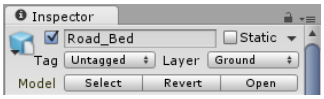
2) Extract all the files in the "ProjectSettings .zip" file into your Project's "ProjectSettings" folder. Replace all files with the already existing ones.

1 RACE TRACK SETUP

This section will go through how to setup your race track. This includes the Waypoints, Finish Line, Checkpoints, etc.

1.1 RACE PATH

Before setting a path for your race track, make sure that your race track (or its collision mesh) is set to the "Ground" layer as follows:



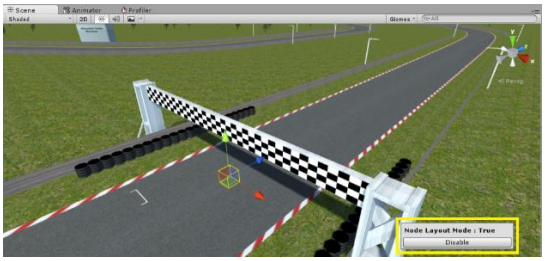
Creating the Race Path

To create a path, go to Window/Racing Game Starter Kit/Create/Create Race Path

This will create a Path Creator component in your scene. The path creator will help you visually create a path around your track.

Position the Path parent (i.e. the small yellow cube) at your track's start / finish line.

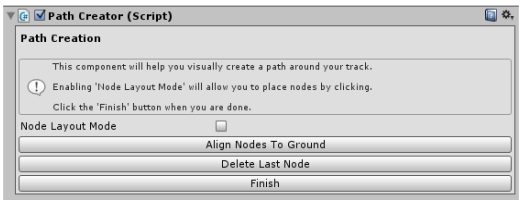
Next, enable **Node Layout Mode** by clicking the button in the Scene View Window:



Node Layout Mode lets you create your race path by clicking on where you want a path node placed. Make sure your track has a collision mesh for this to work.

Important: Ensure that the first path node is in front of your finish line and the last path node is behind your finish line.

After you have created a complete path around your track, click on the "**Finish**" button on the Path Creator component.



1.2 RACE TRIGGERS

There are 4 types of triggers that are used within a race:

- Finish Line Trigger
- Checkpoint Triggers
- Nitro Triggers (for AI)
- Brakezones (for AI)

1.2.1 Creating the Finish Line

To create a finish line for your race track, create a Box Collider trigger and:

- Give it the **"Finish" or "FinishLine"** tag.

- Set it to the "**Ignore Raycast**" layer.

Note: You can create a Box Collider trigger by simply creating a Cube then removing its Mesh Renderer and checking the "trigger" box on its Box Collider component.

1.2.2 Creating Checkpoints

Checkpoint triggers are only used in the "SpeedTrap" and "Checkpoints" race types.

To setup checkpoints for your race track, go to Window/Racing Game Starter Kit/Create/Create Race Checkpoints

This will create a Checkpoint Container component with a checkpoint within it. Position and duplicate the checkpoint trigger around your track.

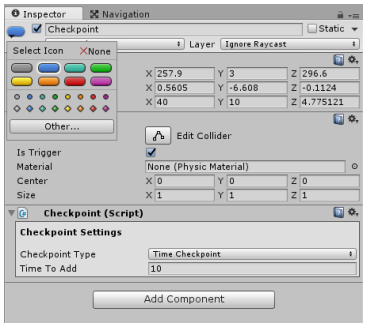
Time Checkpoints

Time checkpoints are used in the “Checkpoints” Race Type to increase a racers time. The amount of time added for passing the checkpoint is set by the “Time to Add” variable.

Speed Trap Checkpoints

Speed Trap checkpoints are used in the “SpeedTrap” Race Type to capture a racers speed.

Note: If you have lots of checkpoints, giving them Gizmo Icons makes them more manageable.



1.2.3 Creating Nitro Triggers

Nitro triggers are used to tell the Ai when to use nitro. When Ai enter this trigger the “nitro probability” value in the OpponentControl.cs will determine the chance of using nitro.

To set up nitro triggers for your race track, simply create a trigger and set its tag to “NitroTrigger” and the layer to “IgnoreRaycast”.

1.2.4 Creating AI Brakezones

Brakezones are used to enforce braking for the Ai, when in a brakezone the Ai will try to reach the “target speed” while in the trigger.

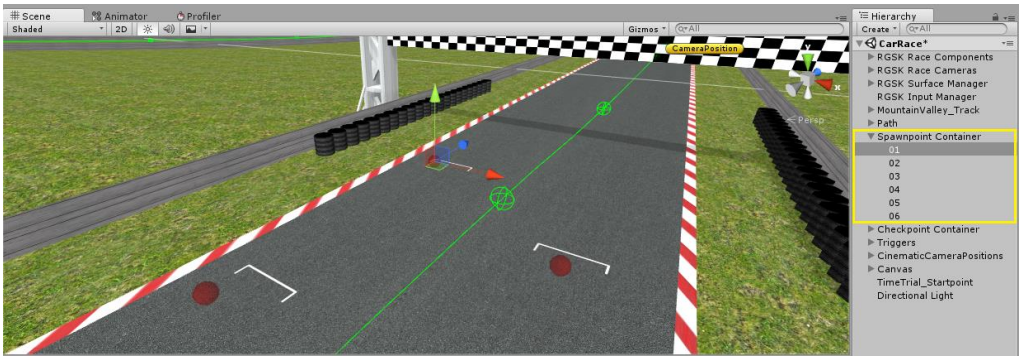
To set up brakezones for your race track, simply create a trigger and add the Brakezone.cs script and set its layer to “IgnoreRaycast”.

1.3 RACE SPAWN POINTS

Spawn points define where the racers will initially spawn.

To create spawn points, go to Window/Racing Game Starter Kit/Create/Create Race Spawnpoints

This will create a new Spawnpoint Container component. Position the spawn point child objects to where you want the racers to spawn.



Keep in mind that racers will spawn facing in the spawn point's direction so ensure the spawn point's rotation is how you want it.

Racers will also spawn according to the spawn point hierarchy arrangement. For example, in the picture above, "01" will be position 1, "02" will be position 2 and so on.

2 RACE COMPONENT SETUP

This section will go through how to setup race components. Race components includes the Race Manager, Race UI, Sound Manager, Input Manager etc.

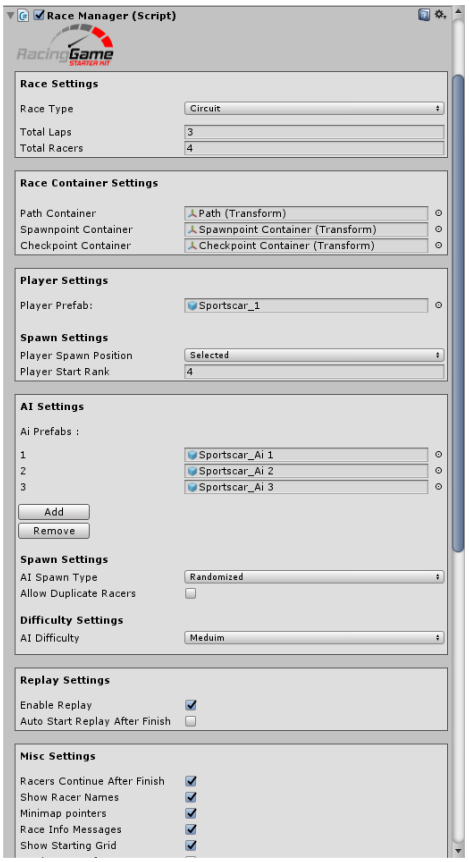
To create the Race Components go to Window/Racing Game Starter Kit/Create/Create Race Components

This will create a new “RGSK Race Components” game object that contains the Race Manager, Race UI, Sound Manager, Camera Manager & Data Loader.

2.1 RACE MANAGER

The Race Manager is the main race component, it handles mostly all race logic.

In the Race Manager you can set values such as the Race Type, Laps, Opponents, Spawn options, Racer names, Mini-map pointers, Race rewards and lots more.



2.1.1 Race Settings

Race Types

- **Circuit Race** is a normal race where racers have to complete a certain number of laps.

- **Lap Knockout** is a last man standing type race. Every lap the racer in last place is knocked out.

- **Time Trial** is a solo race where you aim to set a track best time. You must set the "**StartPoint**" object to determine where the vehicle will begin driving from. Use the "**Auto Drive**" option to determine whether Ai should drive you to the race start point. You can also set a Ghost vehicle in this mode by toggling "**Use Ghost Vehicle**". The Ghost vehicle will follow the path you took allowing you to race against yourself and set the best possible track time. The ghost vehicle's materials depend on what you set. If "**Set Material**" is checked, the ghost will use one material of your choice. If unchecked, the ghost will use its current material. Please ensure that you assign a Shader (preferably Transparent/Diffuse).

- **Speed Trap** is a race whereby the racer with the highest total speed wins. Speed is captured using Speed Trap Checkpoints.
- **Checkpoints** is a race whereby the time is reduced every second and the racers must get to the next checkpoint before the time runs out. The "**Initial Time"** is the time the racers have to get to the first checkpoint.

- **Elimination** is a race whereby the time is reduced every second and once the time runs out, the racer in last place is knocked out and the timer is refreshed. The "**Elimination Time"** is where the timer counts down from.

- **Drift** is a race whereby the player has to achieve a number of drift points. Checking the "**Use Time Limit**" option will give the player a limited amount of time to achieve the target scores. The amount of limited time is set in the "**Time Limit**" field - note that this is in seconds. The Drift Race Type does not work with motorbike physics.

The **Total Laps** are the number of laps the racers have to go to complete the race. On a Lap knockout race, this value is automatically set by the number of total racers

The **Total Racers** are total number of racers that will race (player included). Make sure that your spawn points are not less than this value.

2.1.2 Race Container Settings

Path container is the gameObject that contains the path. The race manager will automatically prompt you to create or assign one if null.

Spawnpoint container is the gameObject that contains the spawn points. The race manager will automatically prompt you to create or assign one if null.

Checkpoint container is the gameObject that contains the checkpoints. The race manager will automatically prompt you to create or assign one if null.

2.1.3 Player & Ai Settings

Player

Player Prefab is the player car.

Player Spawn Position is how player spawning should be handled. "**Randomized**" will spawn the player randomly. "**Selected**" will allow you to choose what position the player will spawn in.

AI

Ai Prefabs are the opponent racers. You can set as many as you want. The race manager will spawn them at random.

AI Spawn Type is how AI spawning will be handled. "**Randomized**" randomizes the spawn. "**Order**" spawns them in order according to how you arrange them in the "Ai Prefabs" list.

Allow duplicate racers is an option you get if you choose "Randomized" spawn type. Checking it will allow duplicate racers. AI spawn settings will be automatically set if your values are inconsistent (e.g. if you have more total racer value than ai racers)

Ai Difficulty allows you to set the Ai difficulty level. Once spawned, the AI difficulty will be set to this value. Set this to **Custom** if you don't want any changes to your Ai difficulty.

2.1.4 Replay Settings

Enable Replay lets you choose whether you want to enable replay features or not. By default this is set to true because replays are awesome.

Auto Start Replay after Finish if checked, the replay will automatically start a few seconds after the race is completed.

2.1.5 Misc Settings

Racers continue after finish - Should all the racers keep driving after finishing the race?

Show Racer Names - Should opponent racer names appear on top of them? Make sure you set a racer name prefab under **Racer Names Settings** below if set to true. The racer name prefab must contain the RacerName.cs component.

Show Mini-map Pointers - Should there be pointers on top of the racers? Make sure you set the pointers under **Mini-map Settings** below if set to true. The mini-map pointers must contain a RacerPointer.cs component.

Race Info Messages will show useful race messages such as, “Final Lap”, “New best time” & more if set to true. A “Race Info” text is required in RaceUI component for this to work.

Show Starting Grid will give way to show the starting grid in Race UI.

Load Race Preferences will load values set in the menu or other scenes into the race manager when the scene begins.

Force Wrongway Respawn will respawn a racer if they go the wrong way for a few seconds.

Start countdown from is the number the countdown will start from. A default value of 3 has been set but you can always change it.

Race countdown delay is how long to wait (in seconds) before starting the countdown.

2.1.6 Racer Names

Player Name is the name given to the Player. If PlayerPrefs has a “PlayerName” key, this string will be used as the player name.

Opponent Names are generated from a .txt file located in the Resources folder. To change or add the names, open the “RacerNames.txt” in the resources folder.

Racer Name Prefab is the gameObject that appears on top of other racer’s heads. This prefab has to contain the RacerName.cs component.

To create a new Racer Name component,

- Create an empty gameObject and add the RacerName.cs component to it.
- Create a Text Mesh (GameObject/3D Object/3D Text)
- Make this Text Mesh a child of the empty gameObject
- Assign the **Racer Position**, **Racer Name** or **Racer Distance** depending on what you want to see.

If you have any issues creating one, please reference the demo Racer Names under the RacingGameStarterKit/Prefabs/Misc/Race Components folder.

2.1.7 Mini-map Settings

Player Pointer is the mini-map pointer for the player

Opponent Pointer is the mini-map pointer for the opponent.

Mini-map pointers must contain the RacerPointer.cs component.

2.1.8 Race Rewards

Race rewards give the player rewards after completing a race. Available rewards are currency, vehicle unlock & track unlock.

To add a new Reward Position, click on the “Add Reward Position” button and fill in the fields.When a player finishes in this reward position, the rewards will be given to them.

Currency is given through a “Currency” PlayerPref that is set in the PlayerData.cs AddCurrecny() function.

Vehicle & Track unlocks are handled by setting the vehicle / track reward string to a PlayerPref value of 1.

2.2 RACE UI

Race UI handles displaying all the UI in the race. You will need to create a Canvas and place all your UI elements within it. Check out the UI tutorials if you are new to this. If you have any issues creating your UI elements for the Race UI component, please reference the demo scenes or contact me for assistance.

2.2.1 Starting Grid UI

The starting grid UI is displayed if “Show Starting Grid” is set to true in the Race Manager.

Starting Grid Panel is the UI Panel containing all the starting grid UI elements.

Starting Grid is a list of the RaceUI.RacerInfoUI class. Each element is a slot in the grid.

Assign the texts for the info you want shown in the starting grid (preferably Position, Name & Vehicle Name).

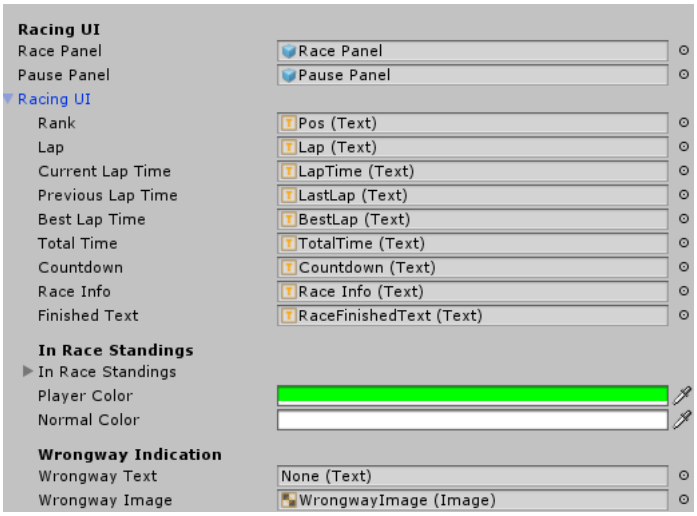
2.2.2 Racing UI

The racing UI contains all the UI elements shown within a race.

Race Panel is the UI Panel containing all the race UI elements. This includes everything in the RacingUI, DriftUI & VehicleUI foldout.

Pause Panel is the UI panel containing the pause menu.

2.2.2.1 Racing UI

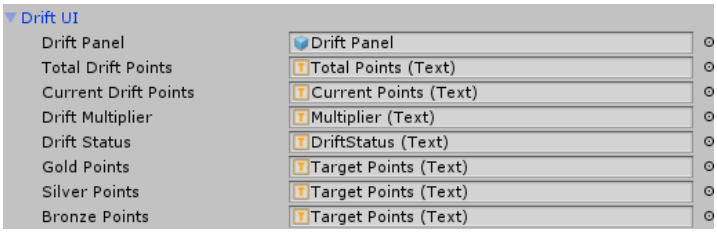


Most of the RacingUI is self-explanatory. You don't have to assign every individual variable slot. Only assign the texts that you want shown.

In Race Standings display the racer positions within the race. It's important to arrange them in order e.g. Position 1 should be element [0], position 2 should be element [1] etc. Position 1 should be element [0], position 2 should be element [1] etc.

Preferably, only assign the Position & Name texts for an in race standing element.

2.2.2.2 Drifting UI

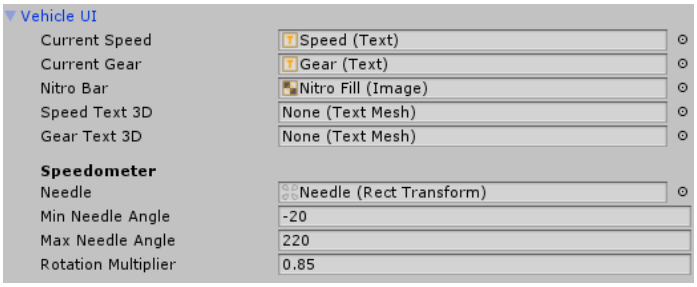


The DriftUI is only used in the Drift Race Type.

Drift Panel is the panel that contains all the texts below it.

The Drift Panel should be a child of the **Race Panel**

2.2.2.3 Vehicle UI



Nitro Bar shows your vehicles nitro capacity. The Nitro Bar should be a Filled image.

Speed Text 3D & **Gear Text 3D** are the 3D text meshes that you may want to display speed through. Place these Text Meshes in your vehicle and name them "3DSpeedText" and "3DGearText" respectively.

Needle is the speedometer needle.

Min Needle Angle is the angle the needle should be when the vehicle is not moving

Max Needle Angle is the angle the needle should be when the vehicle is moving at top speed.

Rotation Multiplier is the amount of rotation added to the needle. This is used for precision.

2.2.3 Fail Race UI

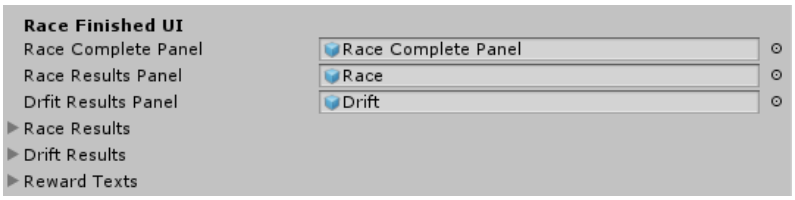
Fail Race UI is used for when the player is knocked out, eliminated or fails a race. This applies for Checkpoint, Lap Knockout & Elimination Races.

Fail Race Panel is the UI Panel that contains all the Fail Race UI elements.

Fail Title is the the title given to the fail panel. For example, on a knockout race this would be set to "Knocked Out".

Fail Reason is the text that shows the reason why the player failed. For example, in a checkpoint race the reason would be "You ran out of time".

2.2.4 Race Finished UI



The Race Finished UI is the UI that is displayed at the end of a race.

Race Complete Panel is the UI Panel containing the **Race Results Panel** & **Drift Results Panel.**

Race Results are a list of texts that can be displayed at the end of the race, just like on the Starting Grid. Make sure that you arrange them in order e.g the texts in element [0] of the list should correspond to the racer that finishes first, texts in element [1] should correspond to the 2nd place racer, and so on.

Race Results should be a child of the Race Results Panel.

Drift Results are the results shown at the end of a Drift Race. They should all be children of the Drift Results Panel.

Reward Texts show the player's winnings. These include currency, vehicle unlock and track unlock.

2.2.5 Replay UI

Replay Panel is the UI Panel containing all your Replay UI elements.

Progress Bar is the replay progress bar Image. This image must be a “Filled” Image.

Replay function buttons such as Fast Forward, Rewind & Pause can be added by:

1) Creating these buttons under your “Replay Panel”

2) Assigning corresponding functions to the buttons :
    - Pause/Play - ReplayManager.PauseReplay()
    - FastForward – ReplayManager.AdjustSpeed (2)
    - Rewind – ReplayManager.AdjustSpeed (-2)
    - Exit Replay – ReplayManager.ExitReplay()
    - Change Camera – CameraManager.SwitchBetweenReplayCameras()

2.2.6 Screen Fade

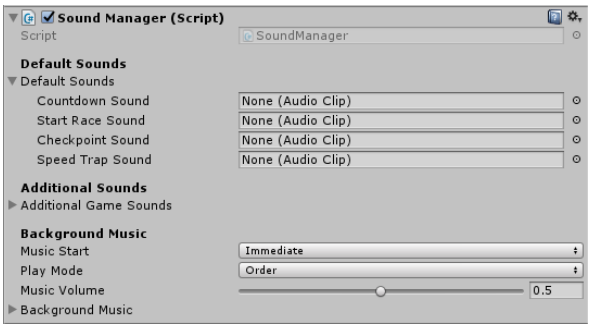
ScreenFade is the image that handles the screen fading.
Important Note: Assign a “CanvasGroup” component to it and uncheck all the checkboxes. This is to ensure that it doesn’t block any of the users clicks.

FadeSpeed is the rate at which fade occurs. Setting this value to 1 would take the screen 1 second to fade.

FadeOnStart/Exit determines whether the screen should fade when the scene loads and when it is exited or restarted.

2.3 SOUND MANAGER

The Sound Manager handles playing sounds and music in the race.



Default Sounds are the sounds played by default during a race. These include the countdown sound, race start, passing a checkpoint & passing a speed trap.

Additional Game Sounds can be added in the List. Each sound has a name & a corresponding AudioClip.

To play one of these sounds call the SoundManager.instance.PlaySound() function. It takes 2 parameters:

1) string name - the name of the sound in the list
2) bool sound2D – should the sound's spatial blend be set to 2D or not

Background Music

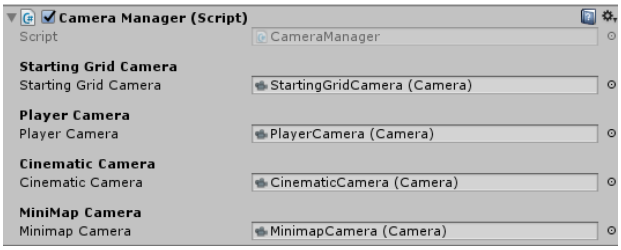
Music Start allows you to set when the the music should start. Immediately, Before Countdown or After Countdown.

Play Mode allows you to set whether playback should be in order or randomized.

Background Music Array is where you add all your race music.

2.4 CAMERA MANAGER

The Camera Manager handles managing & switching between race cameras. Player camera position switching is not handled be the Camera Manager.

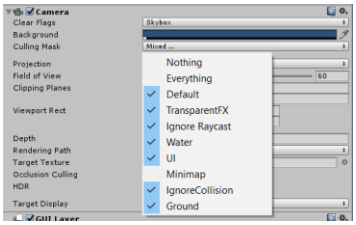


2.4.1 Setting up Race Cameras

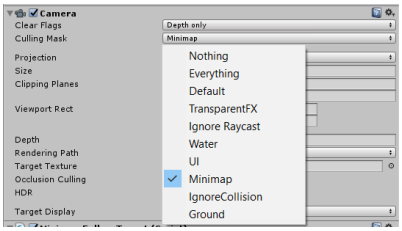
Race Cameras can easily be setup by going to Window/Racing Game Starter Kit/Create/Create Race Cameras

This will create a “RGSK Race Cameras” gameObject containing all the cameras. The cameras will not be fully configured so make sure you make the following changes:

For the Player Camera, Cinematic Camera & Starting Grid Camera set the Culling Mask to all but the Minimap Layer as follows:



For the Minimap Camera, set the Culling Mask to Minimap Layer only:



After creating the Race Cameras assign them to the Camera Manager. Only assign the Cameras that you want but please note that the Player Camera should not be left blank!

2.4.2 Setting up the Cinematic Camera

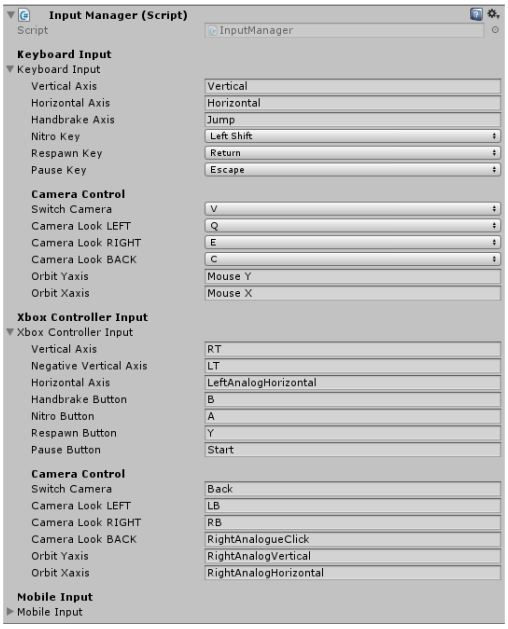
The cinematic camera is mainly used for replays. Setting up the cinematic camera is fairly easily – assign the “Camera Positions Parent” variable.

The “Camera Positions Parent” variable is the parent gameObject that contains all the cinematic camera positions along the race track.

Create an empty gameObject and create child gameObjects within it. Place the child objects around the track where you want the cinematic camera to be able to go. Assign the parent gameObject to the “Camera Positions Parent” variable in the CinematicCamera component.

The cinematic camera will find the closest position to the player and move there.

2.5 INPUT MANAGER



The Input Manager makes it easy for key / button bindings for Keyboard & Xbox Controllers.

To set up an Input Manager go to Window/Racing Game Starter Kit/Create/Create Input Manager

There should only be one instance of the Input Manager in a scene. Preferably, you would only use one in your whole game. Selecting **DontDestroyOnLoad** ensures that the Input Manager is not destroyed on Load.

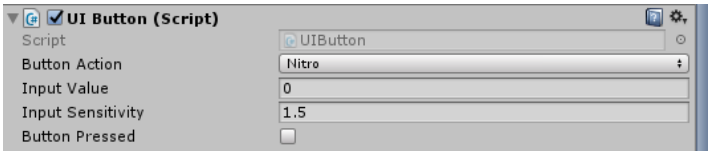
The input manager is referenced and used by the PlayerControl.cs script.

The Mobile Input is automatically assigned by an active MobileControlManager in your scene.

2.5.1 Mobile Input

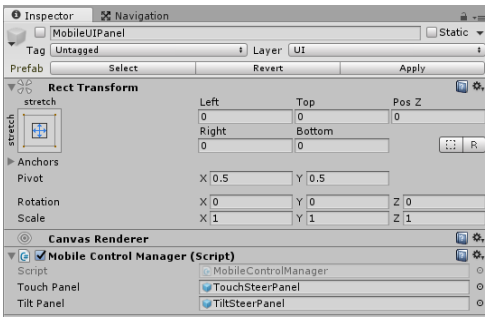
To setup mobile controls, simply drag the “MobileUI” prefab form RacingGameStarterKit/Prefabs/UI folder to your Canvas (under the Race Panel).

The mobile buttons have the UIButton.cs component attached.



The Button Action enum makes it easy for you to set what the button does. This comes in handy when creating your Mobile UI from scratch.

2.5.1.1 Mobile Control Manager



The Mobile Control Manager is attached to the parent of the MobileUI Panel. The main purpose of it is to assign the UIButtons to the Input Manager and handle switching between Touch and Tilt controls.

2.6 MINI-MAP SETUP

To set up a mini-map:

1) Duplicate your track and place it directly below the original
2) Set the duplicate track to the “Minimap” Layer.

3) Give the duplicate track separate materials, preferably the minimap material used in the demo scenes minimap.
4) Position the Minimap Camera using the transform component & Viewport rects to where you want it.

2.7 WAYPOINT ARROW

The waypoint arrow is used to point in the direction of the next path node.

To Setup the waypoint arrow:

1) Drag the "WaypointArrow" from the RacingGameStarterKit/Source/Models folder to your scene.
2) Position the arrow wherever you want – preferably as a child of the Player Camera.

Leave the name of the arrow as "WaypointArrow" because the WaypointArrow.cs script finds it by this name.

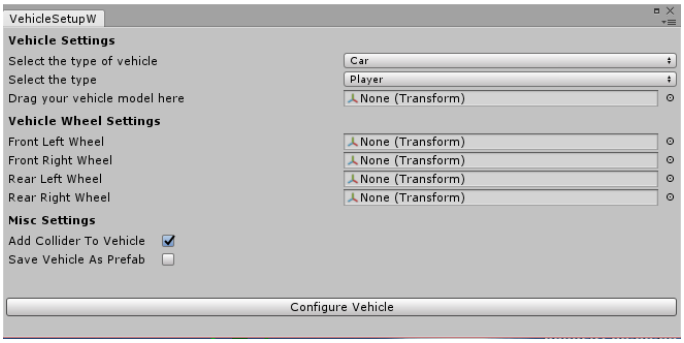
Note: Ensure your vehicle has the WaypointArrow.cs component attached. This is done automatically by the Vehicle Setup Wizard.

3 VEHICLE SETUP

This section goes through vehicle setup. This includes vehicle configuration, vehicle input (player & Ai), wheel effects, external vehicle physics, the IK controlled racer and more.

3.1 VEHICLE CONFIGURATION

3.1.1 Vehicle Setup Wizard



The Vehicle Setup Wizard helps you configure your vehicles in seconds.

To configure a vehicle, go to Window/Racing Game Starter Kit/Vehicle Configuration/Vehicle Setup Wizard

Before filling in any of the fields, make sure that your vehicle pivots and axes are correctly setup i.e.

X axis should face right

Y axis should face up

Z axis should face forward

Correct vehicle pivot / direction.

Correct wheel pivot / direction.

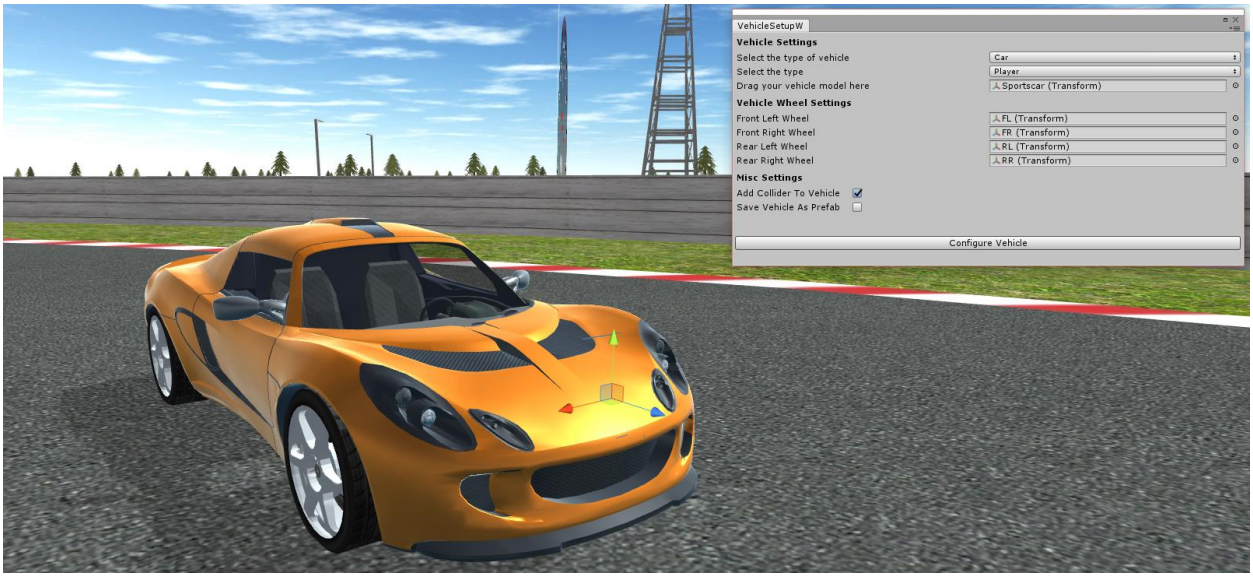
Fixing Incorrect Vehicle Pivot / Direction

If your vehicle pivot & directions are not properly configured, you can fix this within the Editor by doing the following:

1) Select your wheel transform and focus on it using the “F” key or double-clicking it.
2) Create a new GameObject (CTRL + SHIFT + N) or GameObject/CreateEmpty
3) Make the wheel a child of this gameObject
4) Use the new gameObject as your wheel

Another way to fix this is by going into your 3D Editor Software and resolving the issue from there.

When ready, fill in the Vehicle Setup Wizard fields:



Type Of Vehicle - is the vehicle a Car or a Motorbike.

Type - is the vehicle a Player or AI

Add Collider to Vehicle - if this is checked a Box Collider will be added to the vehicle

Save as Prefab - will basically save your configured Player vehicle as a prefab in a new “**Rescources/PlayerVehicles**” folder. This folder is the default for where player vehicles are loaded from. An Ai vehicle will be saved in a new “**Prefabs/AIVehicles**” folder.

Once all the fields are assigned, click the "**Configure Vehicle**" button.

After configuring the vehicle, it will have all the necessary components to race. Some components may need tweaking such as the collider and the Wheel Colliders.

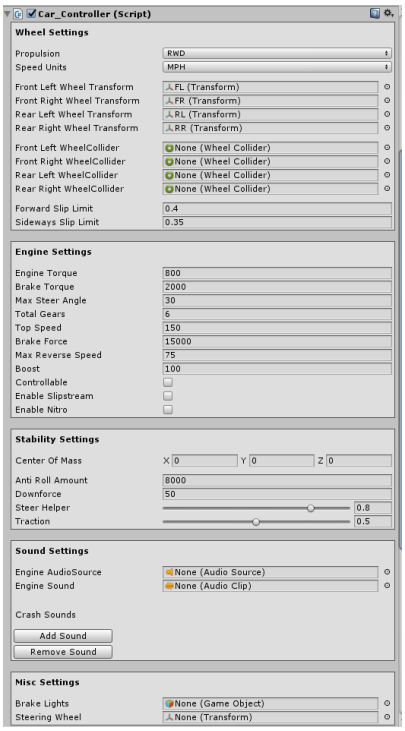
At this point, you can set "controllable" to true on the Car_Controller or Motorbike_Controller component and test drive your vehicle with the ***RaceManager gameObject disabled.***

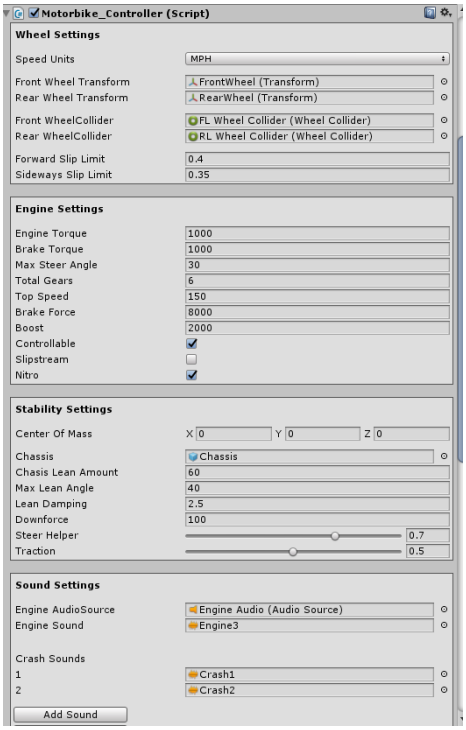
Tip: You can quickly set up an AI racer directly from your player vehicle by going to Window/Racing Game Starter Kit/Vehicle Configuration/External Vehicle Physics/Add Required AI Components

3.2 VEHICLE COMPONENTS

3.2.1.1 Controllers

Your vehicles will have a Car or Motorbike Controller component attached. These define the vehicle's properties. Tweak the settings to your liking:





The Vehicle Setup Wizard will assign the important values for you.

Most of the settings are self-explanatory but I will go through a few:

Enable Slipstream - determines whether the vehicle will be able to slipstream other cars. If set to true, the vehicle will get a speed boost when driving behind other racers. After checking this bool, scroll down to *Slipstream Settings* and make the necessary changes.

Enable Nitro - determines whether the vehicle will be able to use nitro. After checking this bool, scroll down to *Nitro Settings* and make the necessary changes:

- *Nitro Group is the gameObject containing your nitro ParticleSystems. You can find a pre-made prefab at RacingGameStarterKitPrefabs/Misc/NitroGroup.prefab*
- *Nitro Regeneration Rate is how fast nitro is regenerated*
- *Nitro Depletion Rate is how fast nitro is depleted*

Brake Force – This is the amount off added force to stop the vehicle – a higher value will bring the vehicle to a stop quicker

Boost – This is the amount of added forward speed given to the vehicle. If you want a faster vehicle increase this value rather than the **Engine Torque** (keep the engine torque to a maximum of about 1500)

Anti-roll Amount – The amount of force that keeps the vehicle from flipping

Down force - The amount of downward force added to the vehicle as it speeds up

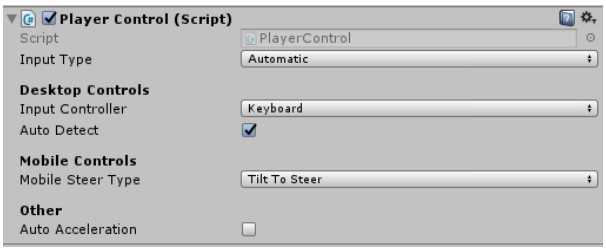
Chassis Lean Amount (Motorbike) - The amount of lean applied to the bike chassis

Max Lean Angle (Motorbike) - The maximum angle the bike will lean while making turns

Lean Damping (Motorbike) - how fast it will reach the max lean angle

3.2.1.2 Player Input

Player Input is handled by the PlayerControl.cs component:



Input Type is the platform on which input will be received. Setting this to “Automatic” will automatically set the Input Type based on the platform.

Note: To test mobile controls in the editor, set this to “Mobile”.

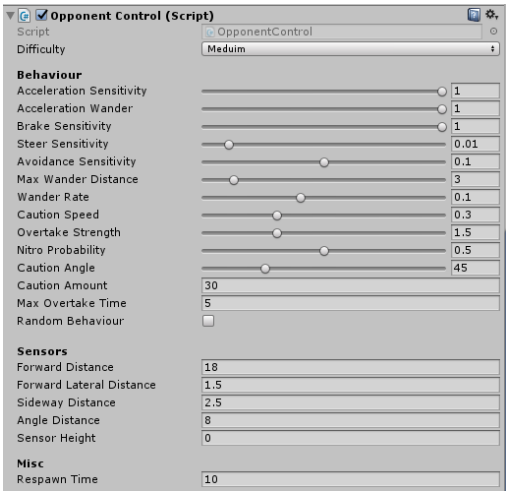
Input Controller is whether a Keyboard or Xbox Controller is being used for input.

Auto Detect will automatically detect whether a Keyboard or Xbox Controller is being used.

Mobile Steer Type is how mobile steering is handled. “Tilt to Steer” uses your devices accelerometer to steer whereas “Touch Steer” uses on screen UIButtons.

3.2.1.3 AI Input

Ai Input is handled by the OpponentControl.cs component:



You can achieve different AI behaviors by tweaking the behavior settings. I will go through what each one does:

Difficulty – here you can choose between 3 difficulty levels. Note that this will make changes to the behavior settings when the AI is spawned. Setting this to “Custom” will not change any of the behavior settings.

Acceleration Sensitivity is how sensitively this AI will use the accelerator.

Acceleration Fluctuation is how frequent the throttle will fluctuate (*0 = faster Ai , 1 = slower Ai)*

Brake Sensitivity is how sensitively this AI will use the brake.

Steer Sensitivity is how sensitively this AI will steer to reach the next waypoint.

Avoidance Sensitivity is how sensitively this AI will steer to avoid another racer or obstacle. When tweaking this value consider checking the **Sensors settings** too.

Max Wander Distance is how far the AI can travel (laterally) from the path. Don’t set this value larger than your track’s width

Wander Rate is how often the AI will begin to wander

Caution Speed is the percentage of the top speed that the AI will try to reach when it is fully cautious (e.g. at a sharp corner)

Overtake Strength is how much the AI will wander to overtake a racer

Nitro probability is the chance of the Ai using nitro when it enters a Nitro Trigger.

Caution Angle is the angle to treat as cautious. Default set to 45.

Caution Amount is how cautious the AI will be around corners.

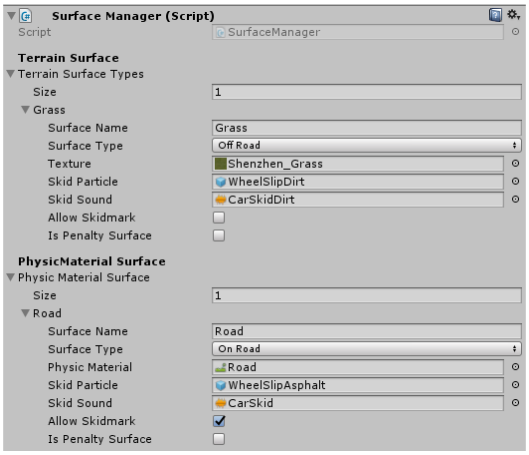
Max Overtake Time is the maximum amount of time the Ai can stay in an overtaking state

Random Behavior will give the AI a random wander behavior.

3.3 VEHICLE WHEEL EFFECTS & SKID MARKS

3.3.1 Surface Manager

The Surface manager handles vehicle wheel particle effects & skid marks.



To create the Surface Manager go to Window/Racing Game Starter Kit/Create/Create Surface Manager

Terrain Surfaces

This is a list of terrain texture based surface detection. When the vehicle wheels are over a Terrain, this list will be referenced.

Physic Material Surfaces

This is a list of Physic Material based surface detection. When the vehicle wheels are over a Mesh Collider that has a Physic Material, this list will be referenced.

Surface Type - the surface of this texture on road or off road

Texture (*Terrain Surface*) **-** the terrain texture on which the *Skid Particle* & *Skid Sound* will be generated.

Physic Material (*Physic Material Surface*) **-** the Physic Material on which the *Skid Particle* & *Skid Sound* will be generated.

Skid Particle - the particle emitted when the vehicle exceeds its forward / sideway slip limits

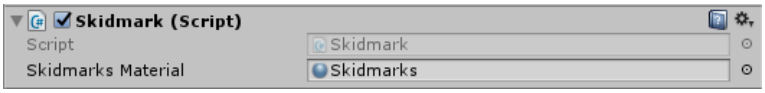
Skid Sound - the sound played when the vehicle exceeds its forward / sideway slip limits

Allow Skid mark - should skid marks be emitted on this surface?

Is Penalty Surface - if set to true, drift points will not be calculated on this surface.

3.3.2 Skid marks

The Surface Manager will be created with a Skidmark component as a child gameObject.



Assign a material to the “Skidmarks Material” variable.

3.4 MULTIPLE CAMERA VIEWS

To set up multiple camera views for your vehicle:

2) Create an empty gameObject and make it a child of your vehicle

3) Add the ChildCameraPosition.cs component to the gameObject

4) Position the gameObject where the camera should be

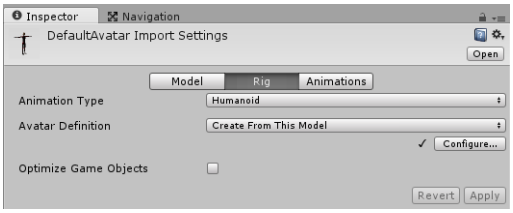
5) Set the **Mode** in the ChildCameraPosition component to either “Fixed” or “First Person” depending on what you want the camera to be

When you switch cameras the PlayerCamera will move to these positions and set its Camera Mode according to the mode set in the ChildCameraPosition.cs component.

3.5 IK RACER SETUP

The IK controlled racer works with any humanoid character *with a correctly configured Avatar*. The IK racer only works with cars but can be setup for motorbikes provided you have the necessary animations.

Before setting up, make sure that your character model animation type is set to Humanoid and the Avatar is correctly configured:



3.5.1 Character Setup

1) Drag your character from your Project tab to the hierarchy and make it a child of your vehicle

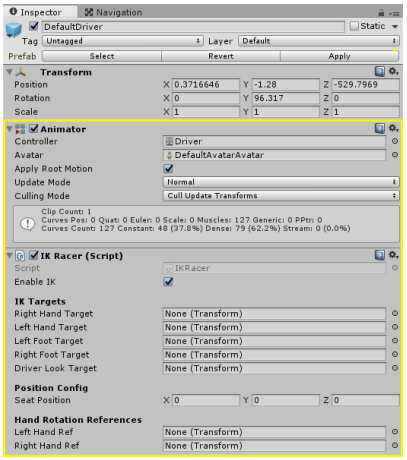
2) Add the IKRacer.cs script to your character

   Racing Game Starter Kit/Scripts/Race/Others/IKRacer.cs

3) Assign the “Driver” Animator Controller as the Animator’s Controller

   Racing Game Starter Kit/Source/Models/Driver/Animator/Driver

4) Your character’s inspector should now look like this :



3.5.2 IK Targets

The IK targets are empty game Objects where the hands & feet will be placed.

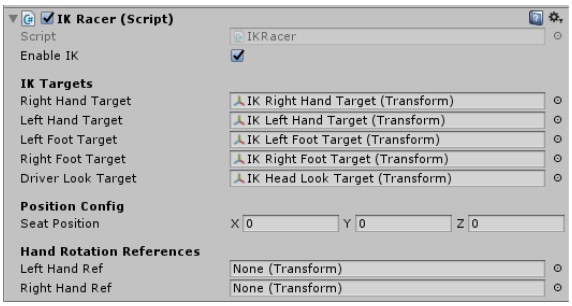
The **Hand Targets** should be children of the steering wheel so that they move together.

The **Foot Targets** should just be direct children of your vehicle.

The **Head Look Target** is where the driver will look based on steer input.

Hierarchy example:

After creating the IK Targets, assign them to your character's IKRacer.cs component:



The IK Target positions and rotations will probably need tweaking when testing.

3.5.3 IK Racer Position Configuration

Now that the IK targets have been set up, we need to set the sitting position of the IK Racer.

1) Enter play mode and change the "Seat Position" values until you get the right values that suit your vehicle. You may also need to adjust your character's rotation.

2) Copy the IKRacer component values then exit play mode. Next, Paste the component values back to the IKRacer component.

   If this step is confusing, please watch this Unity video to clarify things:
   https://www.youtube.com/watch?v=FPFFMwr2W9E

3.5.4 Hand Rotation References

By Default, your character won't grip the steering wheel. You will probably have something that looks like this:

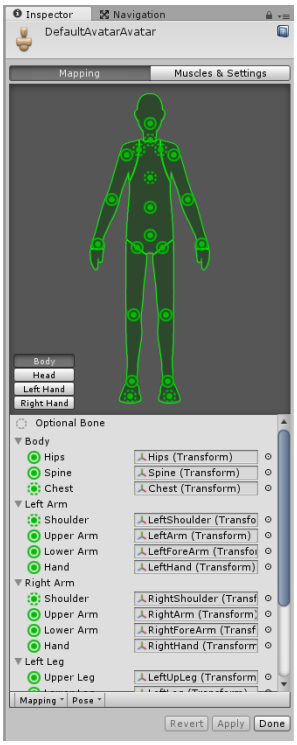
The Hand Rotation References are basically duplicated hands in a grip pose that the real hands use as a reference.

I've found that the fastest way to set this up is to:

1) Select your character model in the Project Tab

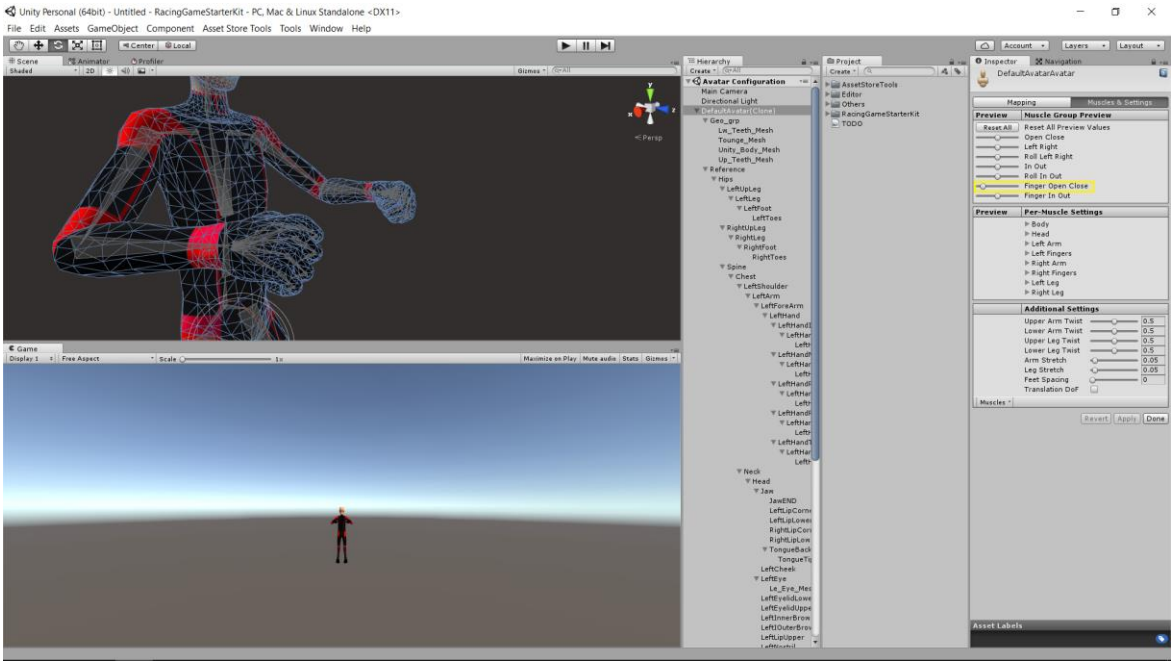
2) Under the "Rig" options, click on the "Configure" button

3) You should now be in a new scene with an inspector that looks like this :

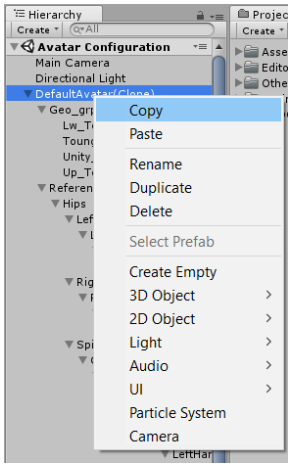


4) Select “Muscles & Settings”

5) Lower the “Finger Open Close” value till you get a good hand grip pose:



6) Right click your character in the hierarchy and select copy then go back to the previous scene to finalize the IKRacer:

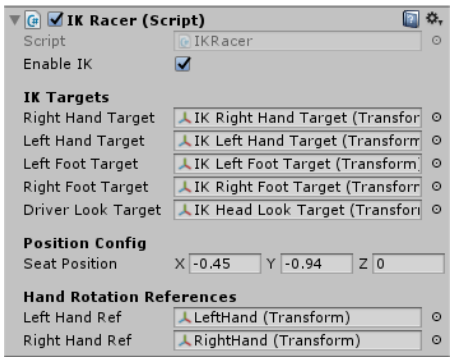


7) Right click an empty area of the hierarchy in your scene and select Paste. This will paste your character from the Avatar Configuration scene into your current scene.

8) Detach the new character's hands by dragging them completely off:

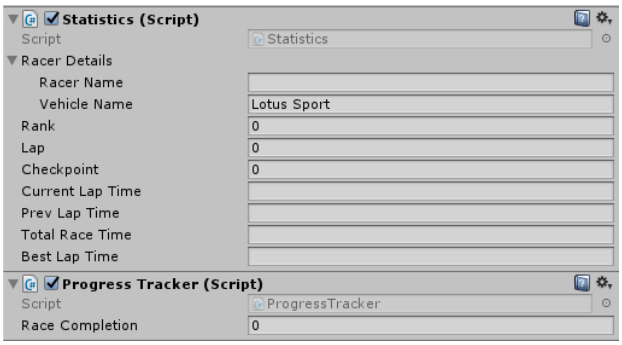
9) Drag them over to where you placed the IK Hand Targets, as children of your vehicle's Steering Wheel. You can add the **IKBoneViewer.cs** script to the hands to view its skeleton for editing.

10) Assign them to your character's IKRacer component :



3.6 RACING COMPONENTS

A configured vehicle will have a Statistics.cs & ProgressTracker.cs component:



3.6.1 Statistics

Statistics.cs handles keeping track of the racer's rank, name, vehicle name, lap, race times, race state, saving best times, wrong way detection etc.

Racer Details allow you to assign a racer name & a vehicle name to the racer. The Race Manager will overwrite the racer name if the *"Assign AI Names"* or "*Assign Player Name"* bools are checked in the Race Manager's **Racer Name Settings**.

3.6.2 Progress Tracker

The progress tracker keeps track of a racers progress along the race track. It works together with the RankManager.cs to set a racers position / rank.

3.7 CUSTOM VEHICLE PHYSICS

3.7.1 Edy's Vehicle Physics

To use Edy's Vehicle Physics, go to RacingGameStarterKit/Other/Integrations/EVP Support and import the EVPSupport.unitypackage file.

This will make the necessary changes to the scripts to support EVP. All Car Configurations will now have to be done via EVP and not the Vehicle Setup Wizard.

Setup EVP Player

- With the vehicle selected in the hierarchy, go to Window/RacingGameStarterKit/VehicleConfiguration/External Vehicle Physics/Add Required Player Components

- Disable the car's VehicleStandardInput.cs component.

Setup EVP Ai

- With the vehicle selected in the hierarchy, go to Window/RacingGameStarterKit/VehicleConfiguration/External Vehicle Physics/Add Required AI Components

Setup EVP Drift

3.7.2 Realistic Car Controller

To use RCC, go to RacingGameStarterKit/Other/Integrations/RCC and import the RealisticCarController.unitypackage file.

This will make the necessary changes to the scripts to support RCC. All Car Configurations will now have to be done via RCC and not the Vehicle Setup Wizard.

Setup RCC Player

- With the car selected in the hierarchy, go to Window/RacingGameStarterKit/VehicleConfiguration/External Vehicle Physics/Add Required Player Components

- Set **CanBeControllable** to false in the car's RCC_CarControllerV3 component

Setup RCC Ai

- With the vehicle selected in the hierarchy, go to Window/RacingGameStarterKit/VehicleConfiguration/External Vehicle Physics/Add Required AI Components

- Set **CanBeControllable** to false in the car's RCC_CarControllerV3 component

Setup RCC Drift

To setup the drift point controller to work with RCC, open the RCCControllerV3.cs script:

1) Add the RGSK namespace at the top of the script (*using RGSK;*)
2) In the DriftVariables() function, add the following:

```
1 reference
void DriftVariables(){

    WheelHit hit;
    RearRightWheelCollider.wheelCollider.GetGroundHit(out hit);

    if(speed > 1f && driftingNow)
        driftAngle = hit.sidewaysSlip * .75f;
    else
        driftAngle = 0f;

    if(Mathf.Abs(hit.sidewaysSlip) > .25f)
        driftingNow = true;
    else
        driftingNow = false;

    //RGSK DRIFT
    if (GetComponent<DriftPointController>())
        GetComponent<DriftPointController>().drifting = driftingNow;

}
```

3.7.3 Unitycar 2.2

To use Unitycar, go to RacingGameStarterKit/Other/Integrations/Unitycar2.2 and import the UnityCar2.2.unitypackage file.

This will make the necessary changes to the scripts to support Unitycar. All Car Configurations will now have to be done via Unitycar and not the Vehicle Setup Wizard.

IMPORTANT: Open the Drivetrain.cs script and add the RGSK namespace (*using RGSK;*) to the script then enter the following code in the Start () function:

```
        CalcIdleThrottle();
        DisengageClutch();
        StartEngine();

        if (RaceManager.instance)
            automatic = false;
    }
```

Setup Unitycar Player

- With the car selected in the hierarchy, go to Window/RacingGameStarterKit/VehicleConfiguration/External Vehicle Physics/Add Required Player Components

Setup Unitycar Ai

- With the vehicle selected in the hierarchy, go to Window/RacingGameStarterKit/VehicleConfiguration/External Vehicle Physics/Add Required AI Components

- Remove any input scripts such as :
    - MobileCarController.cs
    - MouseCarController.cs
    - AxisCarController.cs

Setup Unitycar Drift

3.7.4 Randomation Vehicle Physics 2.0

To use Randomation Vehicle Physics, go to RacingGameStarterKit/Other/Integrations/Randomation and import the RVP.unitypackage file.

This will make the necessary changes to the scripts to support Randomation Vehicle Physics 2.0.

All Car Configurations will now have to be done via RVP 2.0 and not the Vehicle Setup Wizard.

IMPORTANT: This only works with the C# scripts of RVP. JS scripts are excluded from this integration.

Setup RVP Player

- With the car selected in the hierarchy, go to Window/RacingGameStarterKit/VehicleConfiguration/External Vehicle Physics/Add Required Player Components
- Disable any vehicle input scripts i.e. BasicInput.cs or MobileInput.cs

Setup RVP Ai

- With the vehicle selected in the hierarchy, go to Window/RacingGameStarterKit/VehicleConfiguration/External Vehicle Physics/Add Required AI Components
- Remove any vehicle input scripts i.e. BasicInput.cs or MobileInput.cs

Ensure that you have the “GlobalControl” & “TimeMaster” prefabs in your race scene.

Setup RVP Drift

3.7.5 Using Your Own Vehicle Physics

The racing system isn't fully tied to vehicle physics so integrating your own won't be a hassle.

1) Open RaceManager.cs and scroll down to the StartRace() function

```
public void StartRace()
{
    //enable cars to start racing
    Statistics[] racers = GameObject.FindObjectsOfType(typeof(Statistics)) as Statistics[];

    foreach (Statistics go in racers)
    {
        if (go.GetComponent<Car_Controller>())
            go.GetComponent<Car_Controller>().controllable = true;

        if (go.GetComponent<Motorbike_Controller>())
            go.GetComponent<Motorbike_Controller>().controllable = true;

        if (go.GetComponent<MyVehiclePhysics>())
            go.GetComponent<MyVehiclePhysics>().enableInput = true;

        if (_raceType == RaceType.Elimination)
            eliminationList.Add(go);
    }
```

As seen above, either set your vehicle physics to enabled or activate a "enable input" bool to allow all the racers to give input and start racing.

2) Open OpponentControl.cs and make the following changes

- Add your Vehicle Physics as a variable:

```
private MyVehiclePhysics myVehiclePhysics;
```

- Assign it in the Awake function:

```
if (GetComponent<MyVehiclePhysics>())
    myVehiclePhysics = GetComponent<MyVehiclePhysics>();
```

- FeedInput in the FeedInput() method:

```
if (myVehiclePhysics)
{
    //motor value
    myVehiclePhysics.motorInput = m;

    //brake value
    myVehiclePhysics.brakeInput = b;

    //handbrake value (always 0 for AI)
    myVehiclePhysics.handbrakeInput = 0;

    //steer value
    myVehiclePhysics.steerInput = s + avoidanceSteer;
}
```

3) Open Statistics.cs and make the following changes

- Enable player control & Ai control in the PlayerMode() & AiMode() functions. Replace "PlayerControl" with your vehicle input script.

- In the RegisterCheckpoints() function, scroll down to the Speedtrap case and add your vehicle physics current speed as the “speed”:

```
//add to the racers total speed
float speed = 0;

if (GetComponent<Car_Controller>())
    speed = GetComponent<Car_Controller>().currentSpeed;

if (GetComponent<Motorbike_Controller>())
    speed = GetComponent<Motorbike_Controller>().currentSpeed;

if (GetComponent<MyVehiclePhysics>())
    speed = GetComponent<MyVehiclePhysics>().currentSpeed;

speedRecord += speed;
```

4) Open RaceUI.cs and make the following changes

- In the VehicleGUI() function, add your vehicle’s current speed and gear

```
//Speed
if (vehicleUI.currentSpeed)
{
    if (player.GetComponent<Car_Controller>())
        vehicleUI.currentSpeed.text = player.GetComponent<Car_Controller>().currentSpeed + player.GetComponen

    if (player.GetComponent<Motorbike_Controller>())
        vehicleUI.currentSpeed.text = player.GetComponent<Motorbike_Controller>().currentSpeed + player.GetCo

    if (player.GetComponent<MyVehiclePhysics>())
        vehicleUI.currentSpeed.text = player.GetComponent<MyVehiclePhysics>().currentSpeed;
}

//Gear
if (vehicleUI.currentGear)
{
    if (player.GetComponent<Car_Controller>())
        vehicleUI.currentGear.text = player.GetComponent<Car_Controller>().currentGear.ToString();

    if (player.GetComponent<Motorbike_Controller>())
        vehicleUI.currentGear.text = player.GetComponent<Motorbike_Controller>().currentGear.ToString();

    if (player.GetComponent<MyVehiclePhysics>())
        vehicleUI.currentGear.text = player.GetComponent<MyVehiclePhysics>().currentGear;
}
```

- Set the “fraction” of the Speedometer to your vehicles current speed:

```
float fraction = 0;

if (player.GetComponent<Car_Controller>())
{
    fraction = player.GetComponent<Car_Controller>().currentSpeed / vehicleUI.maxNeedleAngle;
}

if (player.GetComponent<Motorbike_Controller>())
{
    fraction = player.GetComponent<Motorbike_Controller>().currentSpeed / vehicleUI.maxNeedleAngle;
}

if (player.GetComponent<MyVehiclePhysics>())
{
    fraction = player.GetComponent<MyVehiclePhysics>().currentSpeed / vehicleUI.maxNeedleAngle;
}
```

5) Open ReplayManager.cs and make the following changes

- Add your vehicle physics as one of the vars in the Racer class.

- In the GetRacersAndStartRecording() function, get your vehicle physics:

```
for (int i = 0; i < racers.Count; i++)
{
    racers[i].racer = allRacers[i].transform;

    if (racers[i].racer.GetComponent<Car_Controller>())
    {
        racers[i].carController = racers[i].racer.GetComponent<Car_Controller>();
    }

    if (racers[i].racer.GetComponent<Motorbike_Controller>())
    {
        racers[i].motorbikeController = racers[i].racer.GetComponent<Motorbike_Controller>();

        if (racers[i].motorbikeController.chassis)
            racers[i].motorbikeChassis = racers[i].motorbikeController.chassis.transform;
    }

    if (racers[i].racer.GetComponent<MyVehiclePhysics>())
    {
        racers[i].myVehiclePhysics = racers[i].racer.GetComponent<MyVehiclePhysics>();
    }
}

replayState = ReplayState.Recording;
```

- In the Record() function, add your vehicle physics and record it's values

```
if (racers[i].myVehiclePhysics)
{
    racers[i].vehicleState.Add(new VehicleState(racers[i].racer.position, racers[i].racer.rotation, racers[i].racer.GetComponent<Rigidbody>().velocity, racers[i].myVehiclePhysics.motorInput, racers[i].myVehiclePhysics.br
}
```

- Scroll down to the SetRacerStateFromReplayKey() function and playback the values:

```
if (rigid.transform.GetComponent<MyVehiclePhysics>())
{
    rigid.transform.GetComponent<MyVehiclePhysics>().motorInput = Throttle;

    rigid.transform.GetComponent<MyVehiclePhysics>().brakeInput = Brake;

    rigid.transform.GetComponent<MyVehiclePhysics>().handbrakeInput = Handbrake;

    rigid.transform.GetComponent<MyVehiclePhysics>().steerInput = Steer;
}
```

6) Open GhostVehicle.cs and make the same changes as the ReplayManager.cs however without getting an element in an array.

7) Open IKRacer.cs and make the following changes

- Add your vehicle physics as a variable
- Assign it in the Start function
- Scroll down to the HeadLook() function and set "steer" as your vehicle physics steer input.

3.8 USING REWIRED

Rewired is an advanced input system that makes handling input so much better in Unity:

Check it out here:

Rewired

To use Rewired with the Racing Game Starter Kit, import the **Rewired.unitypackage** found in the RacingGameStarterKit/Other/Integrations/Rewired folder.

This package comes with an edited PlayerControl.cs that works with Rewired and a Rewired Input Manager prefab. This prefab has been setup in a fairly simple way so it should be easy to extend to suit your racing game's input requirements. Make sure you have this input manager present in your race scenes.

If you are new to Rewired, I would recommend watching this video for the full overview of Rewired:

Rewired Overview

4 FINALIZING THE RACE SCENE

A race scene should consist of:

1) A fully configured Race Track i.e. Path, Spawn points, Triggers (*a finish line must be included*).

2) Race Components i.e. the "RGSK Race Components", "RGSK Race Cameras" & the "RGSK Input Manager" gameObjects.

3) Configured Race Vehicles (Player & Ai)

Before running the scene to test your first complete race, make sure to add Player and Ai Vehicles to the Race Manager's Player & Ai Settings.

If you experience any issues with anything please reference the demo scenes or contact me for assistance.

5 RACE DATA

All Data in the Racing Game Starter Kit is stored using PlayerPrefs.

5.1 DATA LOADER

The **DataLoader.cs** is responsible for loading data and assigning it to the Race Manager. The Data Loader loads the following values:

6) Race Type
7) Player Vehicle
8) Player Name
9) Laps
10) Total Racers
11) AI Difficulty

To load your preferences, please ensure that you have an active Data Loader in your scene and check "Load Race Preferences" in the Race Manager.

5.2 BEST TRACK LAP TIMES

Your best time for each track scene is stored in a PlayerPref string.

To access the player's best time for a track simply load it by:

PlayerPrefs.GetString("BestTime"+ "SceneName");

5.3 PLAYER DATA

5.3.1 Currency

Player currency is saved & loaded in the **PlayerData.cs** script.

A default currency of 100,000 is set.

To change the default currency, open the **PlayerData.cs** script, change the "startingCurrency" value then go to Window/RacingGameStarterKit/Utility/Clear Player Data

5.3.2 Vehicle & Track Unlock

PlayerData.cs handles unlocking vehicles and tracks by simply setting the vehicles / track name to a PlayerPref integer value of 1.

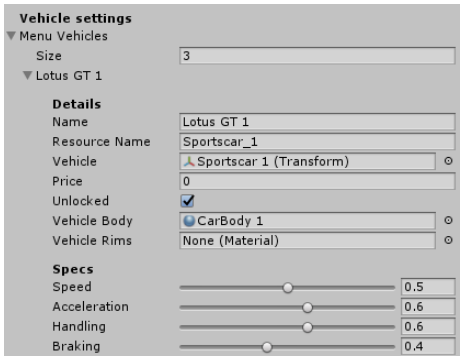
The MenuManager.cs then checks whether the PlayerPref is 0(locked) or 1(unlocked).

6 USING THE MENU SYSTEM

The menu system (MenuManager.cs) handles vehicle selection, track selection, settings and more. Please note that the menu system was **intended for demo purposes** and will probably have to be extended to meet your racing game's menu requirements.

6.1 ADDING PLAYER VEHICLES

1) Drag your model from the Project View into your scene.
2) Add a new element in the "Menu Vehicles" List :



3) Assign the values accordingly:

   Name is the vehicle name- this will be displayed on the vehicle selection screen.

Resource Name is the name of the vehicle within a Resources/PlayerVehicles folder. This is where vehicles are loaded from by the DataLoader.cs component.

Vehicle the vehicle gameObject in the hierarchy.

Price is how much this vehicle will cost if you want to allow the player to be able to purchase it.

Unlocked is the vehicle unlocked or not. Setting this to true on start will pre-unlock the vehicle.

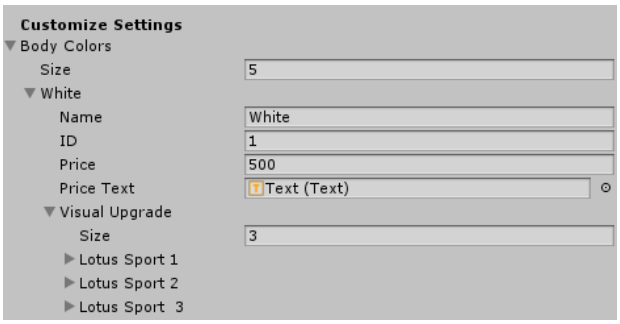
Vehicle Body is the vehicle's body material. Used in Color customizations.

Vehicle Rims is the vehicle's rims material. Used in Rim customizations.

Vehicle Specs are the visual representations of the vehicle's performance values.

6.2 VEHICLE CUSTOMIZATION

6.2.1.1 Body Color



Body color customizations work by adding a new element in the "Body Colors" array and filling in the details.

Each element in the array is one individual color.

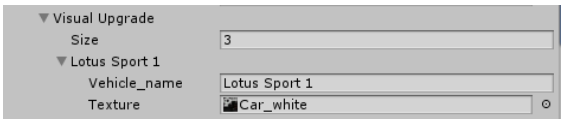
Name is the color's name. This makes it easier to manage all you colors within the array.

ID is the color's ID, this must be a unique number between all your colors

Price is how much this color costs

Price Text is the text that displays this colors price

Visual Upgrade is a list of that applies the texture of the color to a vehicle.



- These must be arranged in the same order that your MenuVehicle's List is arranged

Texture is this color's corresponding texture. This texture is applied to the vehicle's "Vehicle Material"

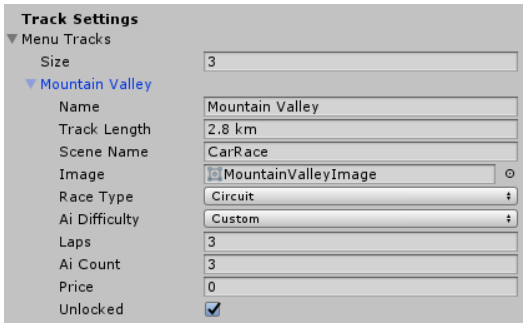
6.2.1.2 Rims

Rim customizations work the same way as Body Color customizations. Please read above.

6.3 ADDING RACE TRACKS

To add a new race track:

1) Add a new element to the "MenuTrack" list and fill in the details:



Name is the name of the track that will be displayed in the menu selection screen

Track Length will also be shown in the menu selection screen. Logging the Race Manager's "`raceDistance`" value will give you the approximate length of your track.

Scene Name is the name of the track's scene. Make sure this matches the name of the corresponding scene in the Build Settings.

Image is the track's image displayed in the menu selection scene

Race Type is this track's race type.

Ai Difficulty is how hard the Ai will be on this track

Laps / Ai Count are the number of laps and opponents in the track

Price is how much this track will cost if you want to allow the player to be able to purchase it

Unlocked is this track unlocked or not. Setting this to true on start will pre-unlock the track.

6.4 MENU UI SETUP

The Menu UI setup will be fairly simple if you are familiar with UGUI. A prefab of the menu canvas can be found in the RacingGameStarterKit/Prefabs/UI folder.

Assign only the variables that you need to the MenuManager component

If you are unsure of anything, please reference the demo scene's MenuManager or contact me for assistance.

Congratulations!

You've set up a complete racing environment. Hit play and race!

If you encounter any problems or errors, please reference the demo scene, leave a post on the forum or contact me directly at ian.izzy94@gmail.com

Thank you for your support!

CREDITS

Background Music – Unity3.x Car Tutorial Demo (https://www.assetstore.unity3d.com/en/#!/content/10)