Drag Racing Framework

👋 Welcome

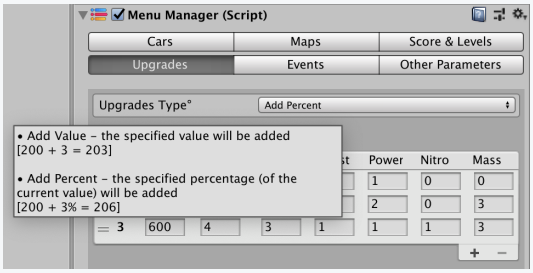
This is a manual for the 1.3.1 version

Here you can see the detailed documentation for the **Drag Racing Framework**. If you have any questions, feel free to write on the **Discord server**, or by email - **gercstudio@gmail.com** and we'll be happy to provide you with assistance

Tips

> ⓘ To know which version you're using, open the info menu by pressing
>
> **[Window -> Drag Racing Framework -> About]**

> ⓘ If you see a variable with this symbol °, hover over it to see a tooltip



🛠️ Get Started

There are 3 main components in the framework: the **MenuManager**, **CarController**, and **GameManager**. They are responsible for the **menu**, **cars**, and **levels**, respectively.

Best to start by creating a car:

Car Creation

Then customize the menu and create game levels:

Menu Manager

How to create a level

Builds

Your game will work the same on all platforms, so just set the target platform and click the "build" button.

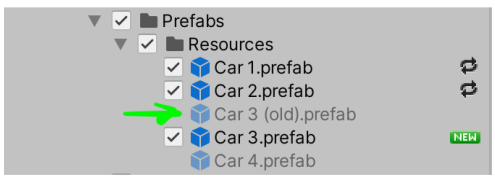
> If you're going to build a WebGL game, read this section first.

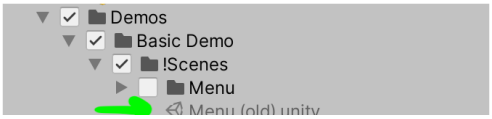
Migration Guide

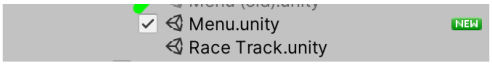
From the 1.2.2 to 1.3 version

> **Back up your project before upgrading!**

■ When you update the project, **all your prefabs and scenes will remain unchanged**, only the demo things will be overwritten. If you want to save them, just change their name





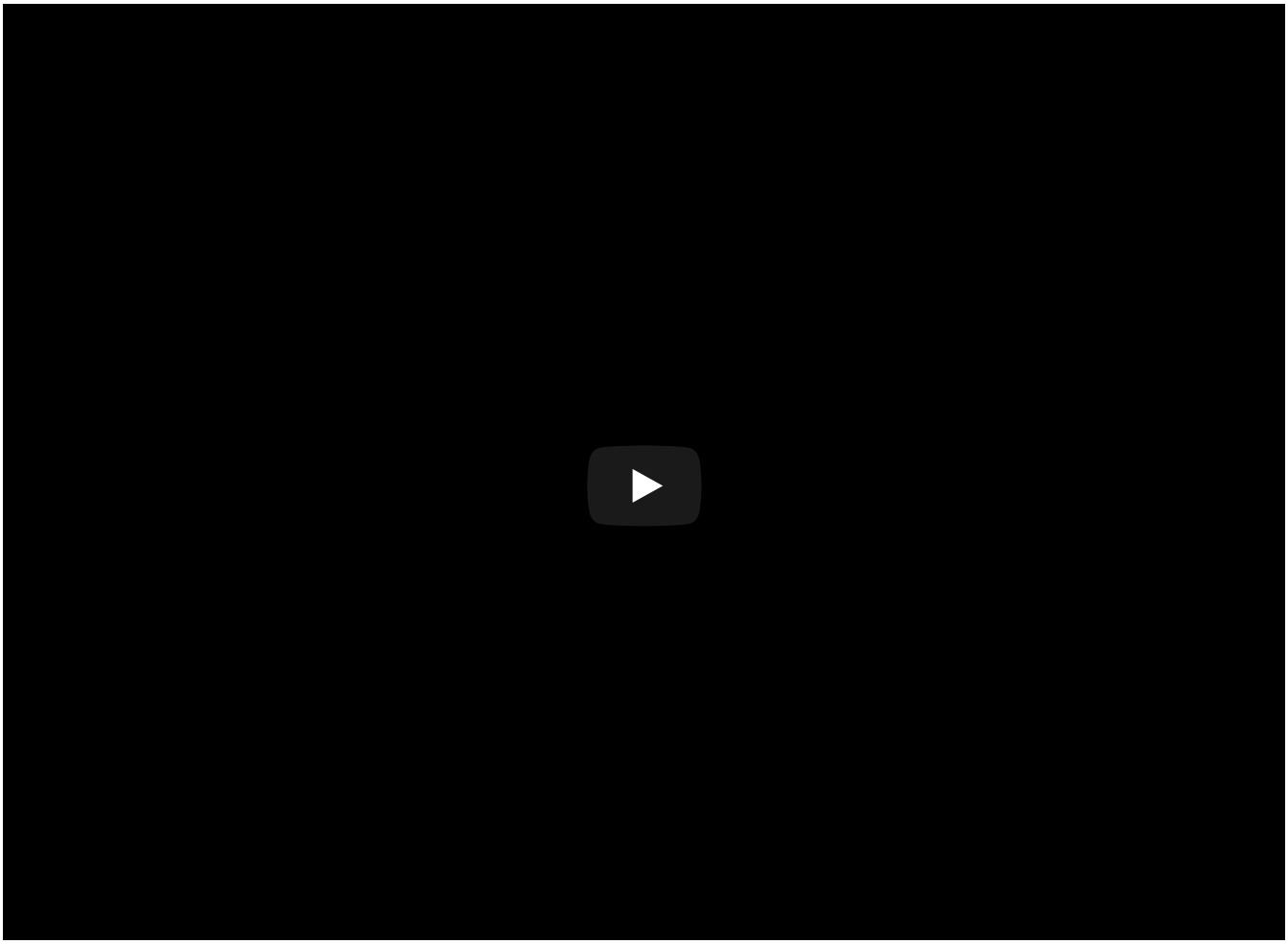


■ The framework now uses an **updated UI system**. To transfer your current UI to the new manager, do the following:

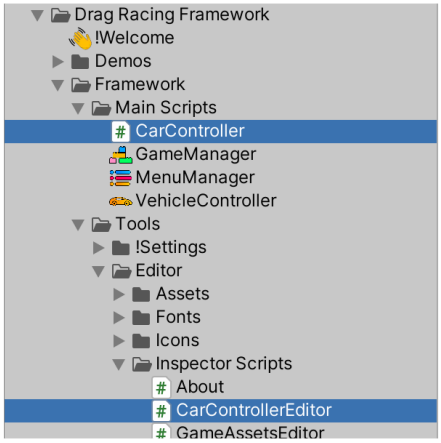
- Rename the UI Manager in your project

- Import the new DRF version
- Transfer the UI elements from the old input manager to the new one (below you'll see a video on how to do it)

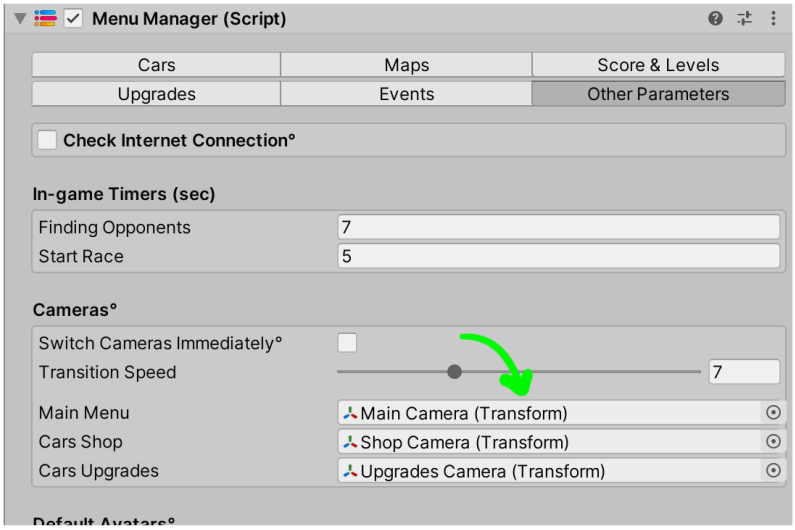
> After importing, when you activate the old manager, you may see many red errors, just ignore them
>
> 
>



■ After importing, you need to delete 2 old scripts. And then, recreate your cars using the new script **VehicleController**.



■ Also, you need to create additional cameras in the main menu.



■ In some cases, you'll need to set some parameters in scripts again (sounds, images, prefabs, etc)

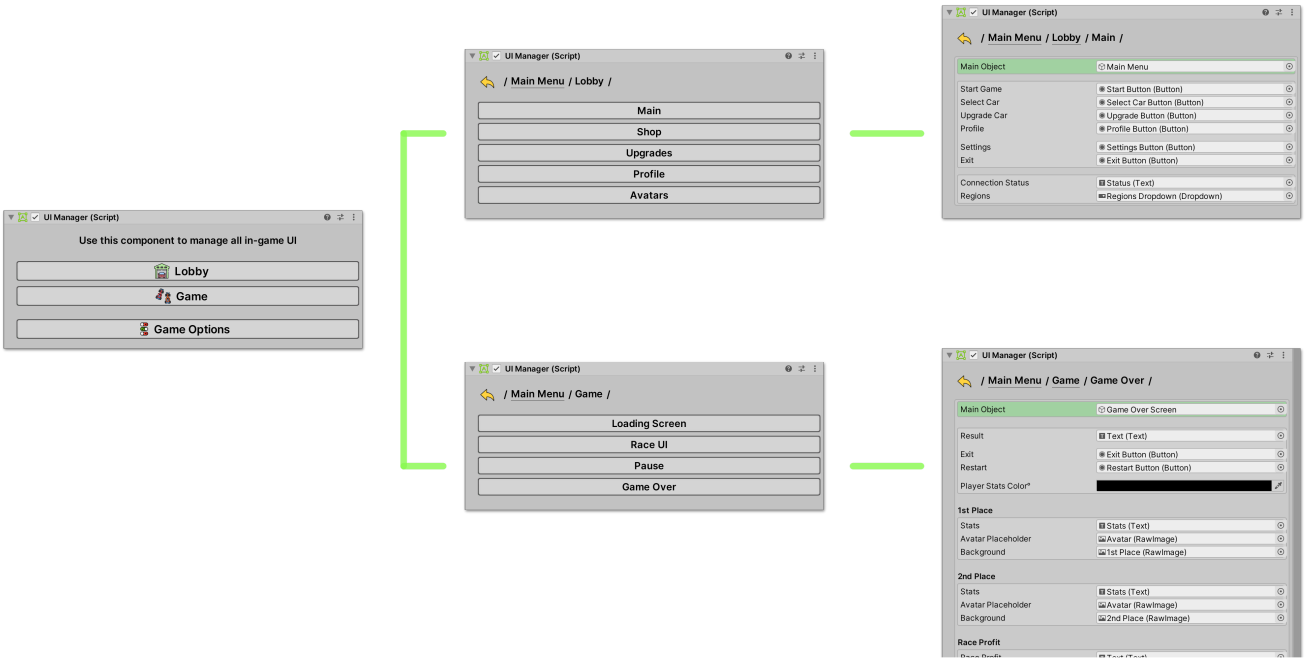
> If you have any problems, be sure to contact us:
> **gercstudio@gmail.com** | **https://discord.gg/fxmJSjD**

Project Settings

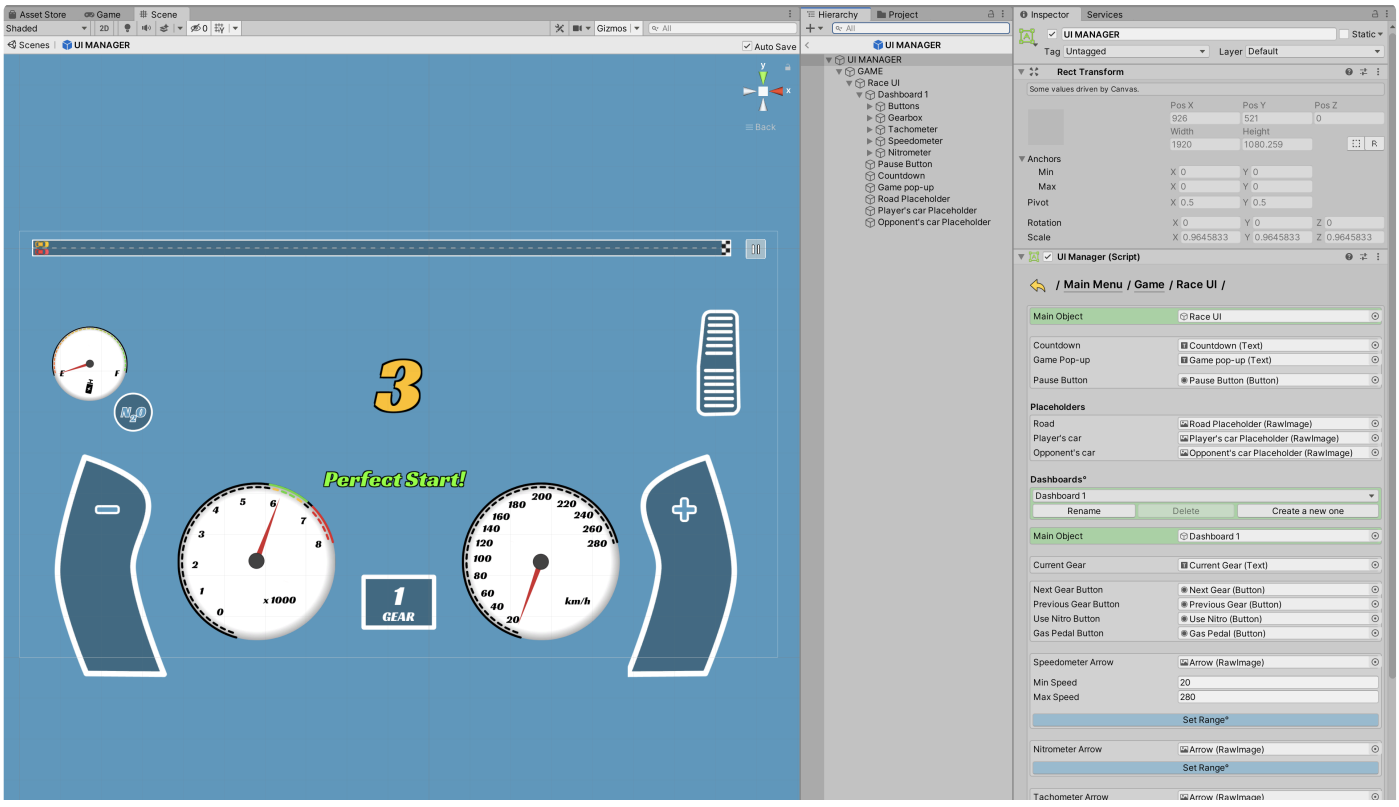
UI Manager

The UI Manager contains the entire user interface for the Menu and Game. To open it, press the **(Tools -> DRF -> Project Settings -> UI Manager)** or use the **Shift+U** hotkey.

- All UI elements are divided into groups to make it easier for you to interact with them.

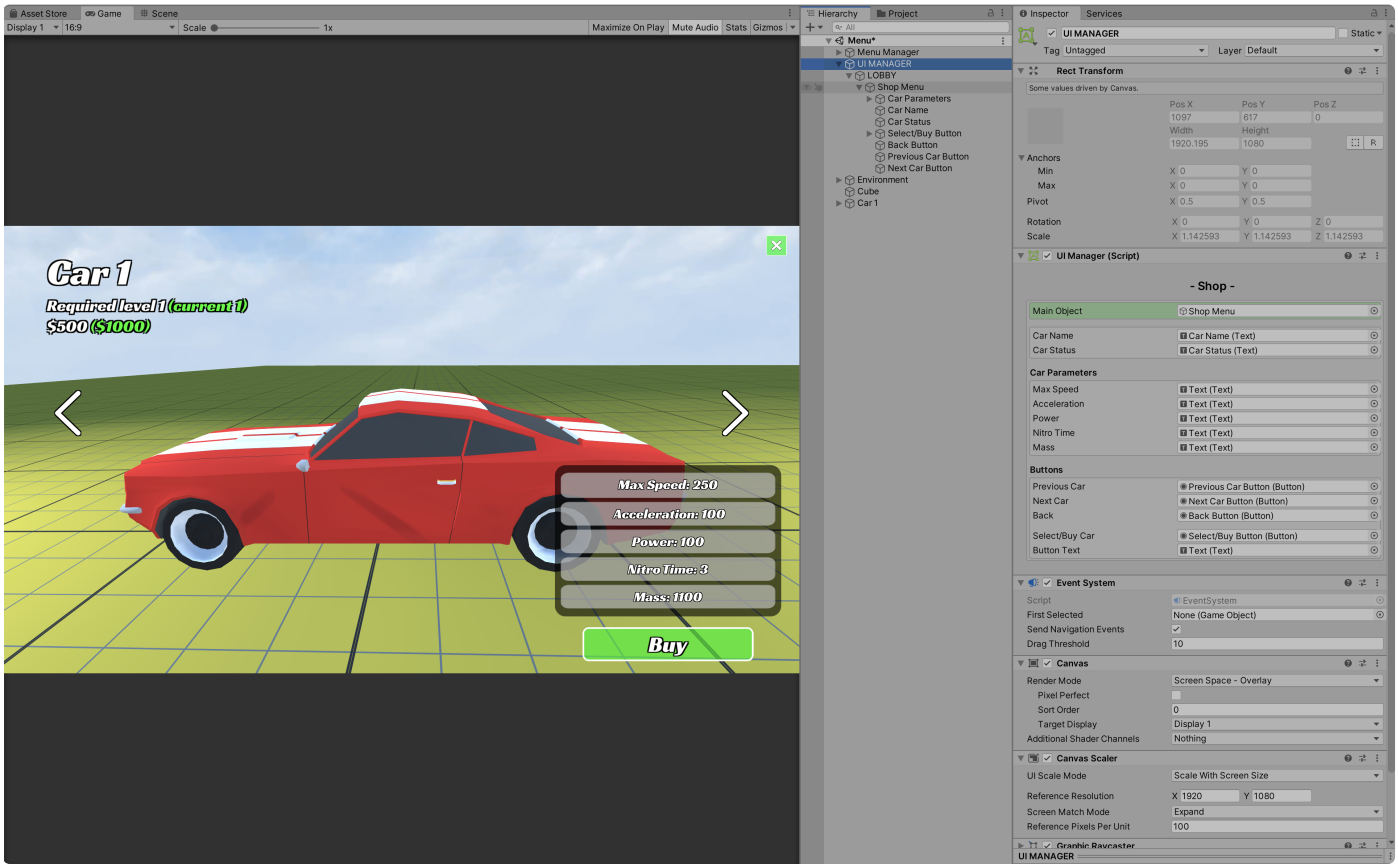


- When you open a section, the entire UI of that group is activated.



> **Please note:** don't delete or move the path items that are written in caps (for example, **UI Manager / LOBBY**), but you can safely move and delete everything else.

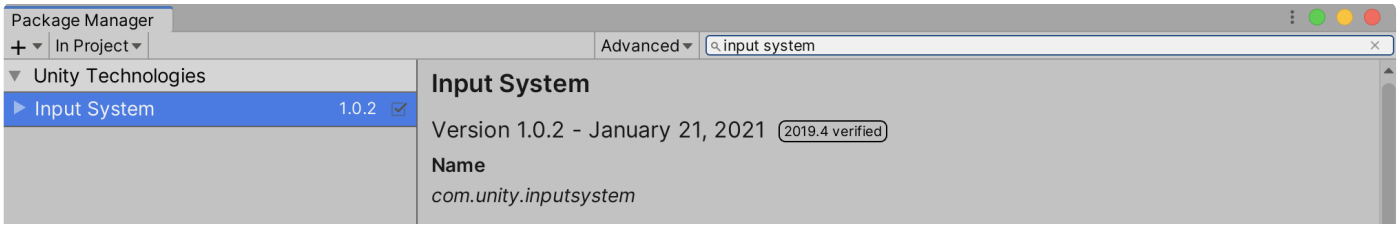
- During the game, when UI elements appear on the screen, then the corresponding menu opens in the UI Manager prefab (which is located in your game scene).

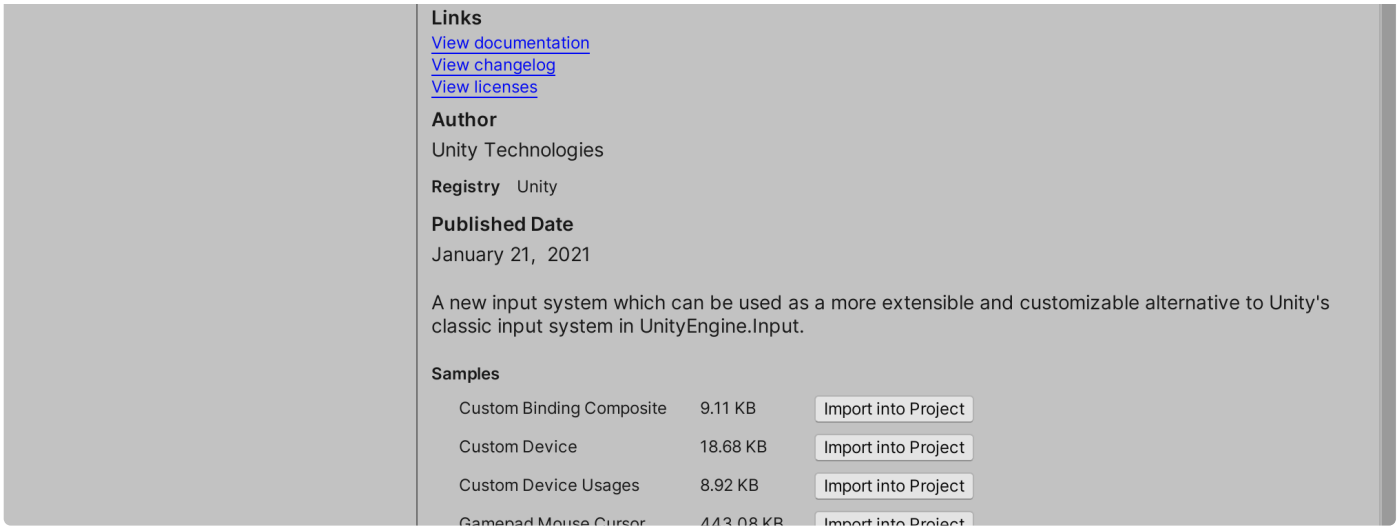


Input Settings

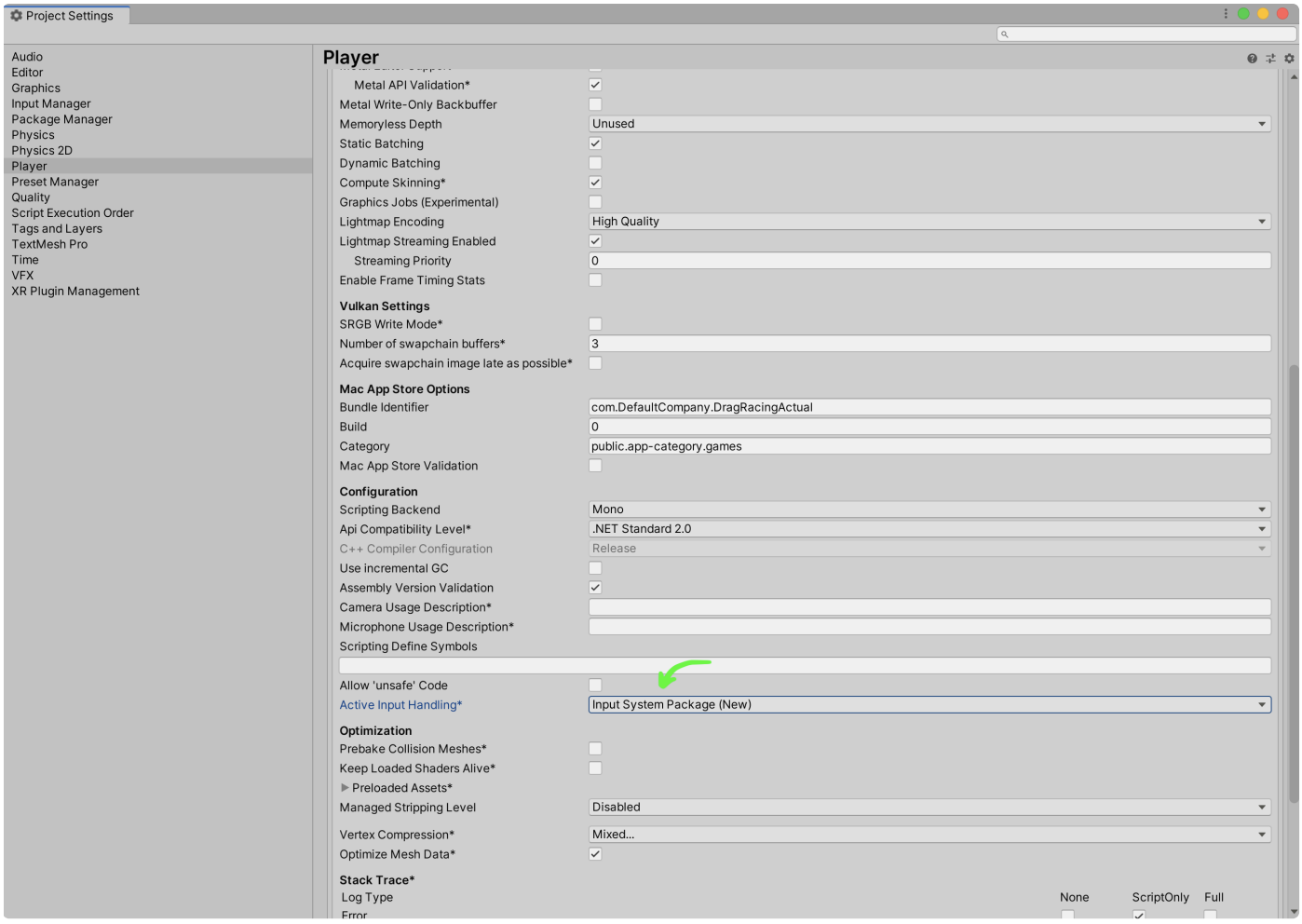
By default players can control the game with the UI buttons (by pressing them with mouse or touch). But you can also activate keyboard and gamepad inputs:

- Import the **Input System package** into your project.

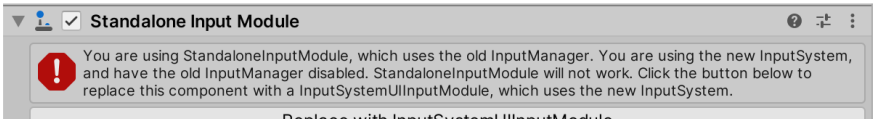


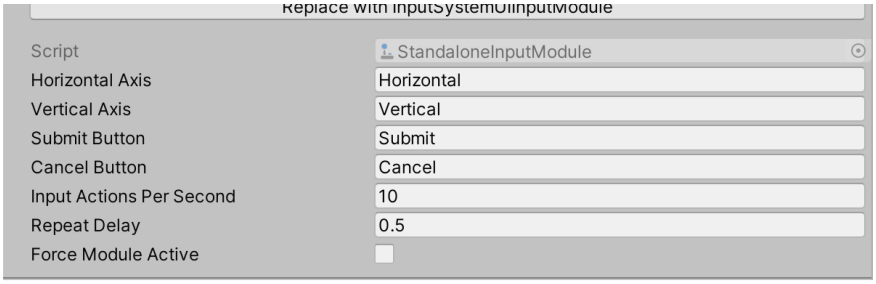


- After the package is installed, set the **Active Input Handling** to **Input System Package (new)** (Edit -> Project Settings -> Player).

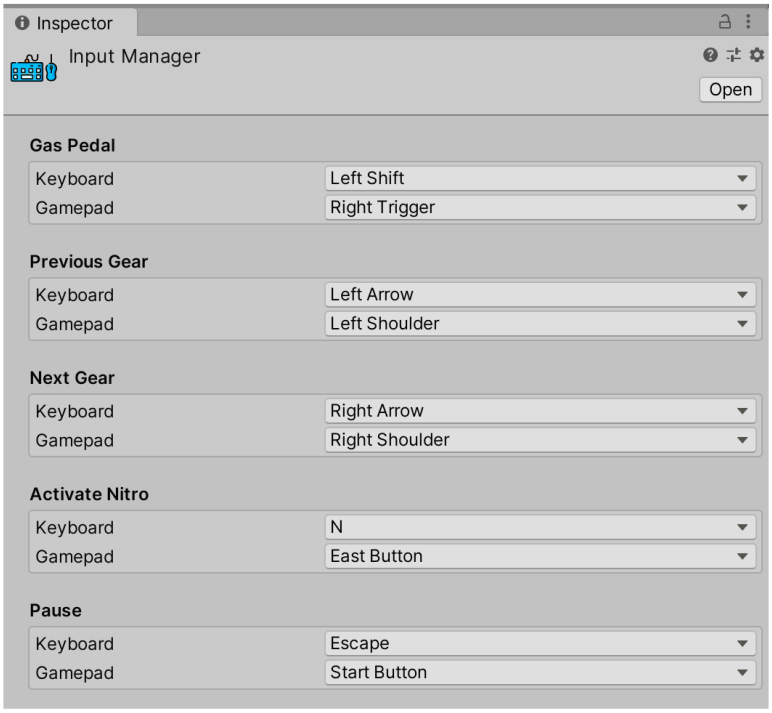


- Activate an integration by pressing **Tools -> DRF -> Integrations -> Input System**.
- Replace the **Standalone Input Module** which is located on the **[UI Manager]** prefab with the new one.





- Now you can adjust input methods in the **Input Manager (Tools -> DRF -> Project Settings -> Input Manager)**



Car Controller

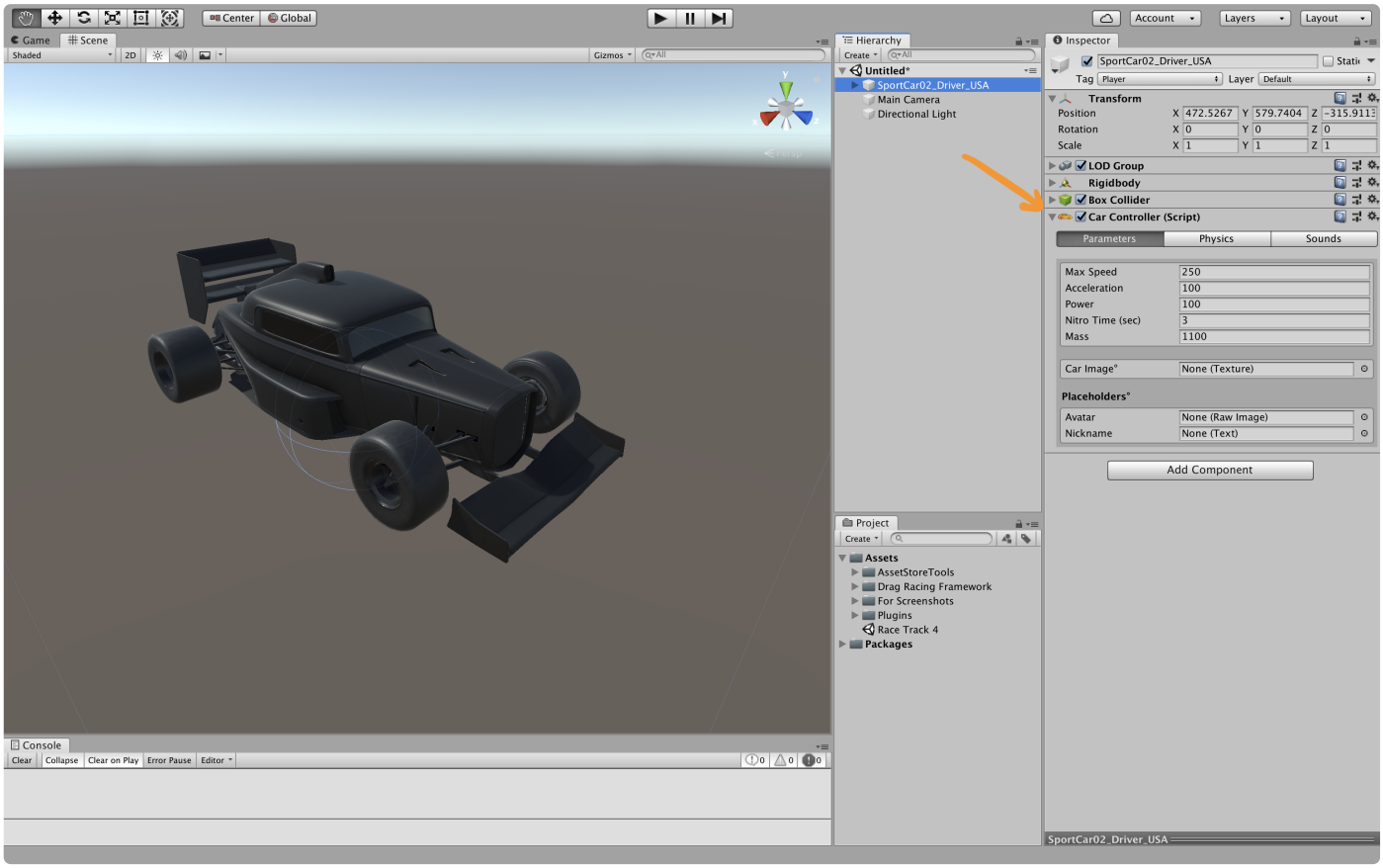
Car Creation

> Before you create a car, make sure that it complies with the recommendations:
>
> - **If the car has a complex structure, make the body and wheels separate.**
> - **Also, combine all body parts into one game object.**

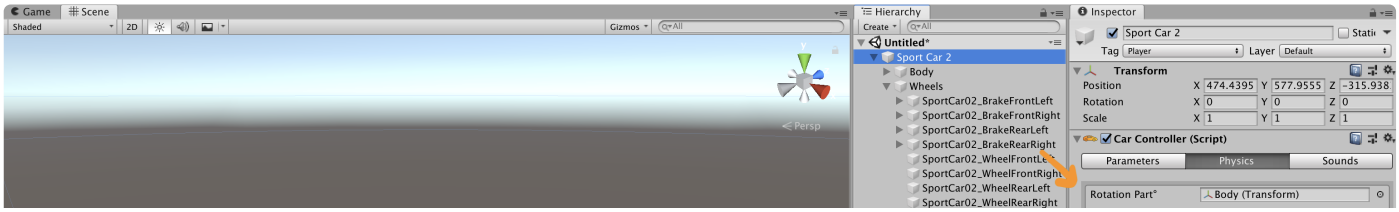
After creating a car, move the prefab to the **[any path/Resources]** folder.

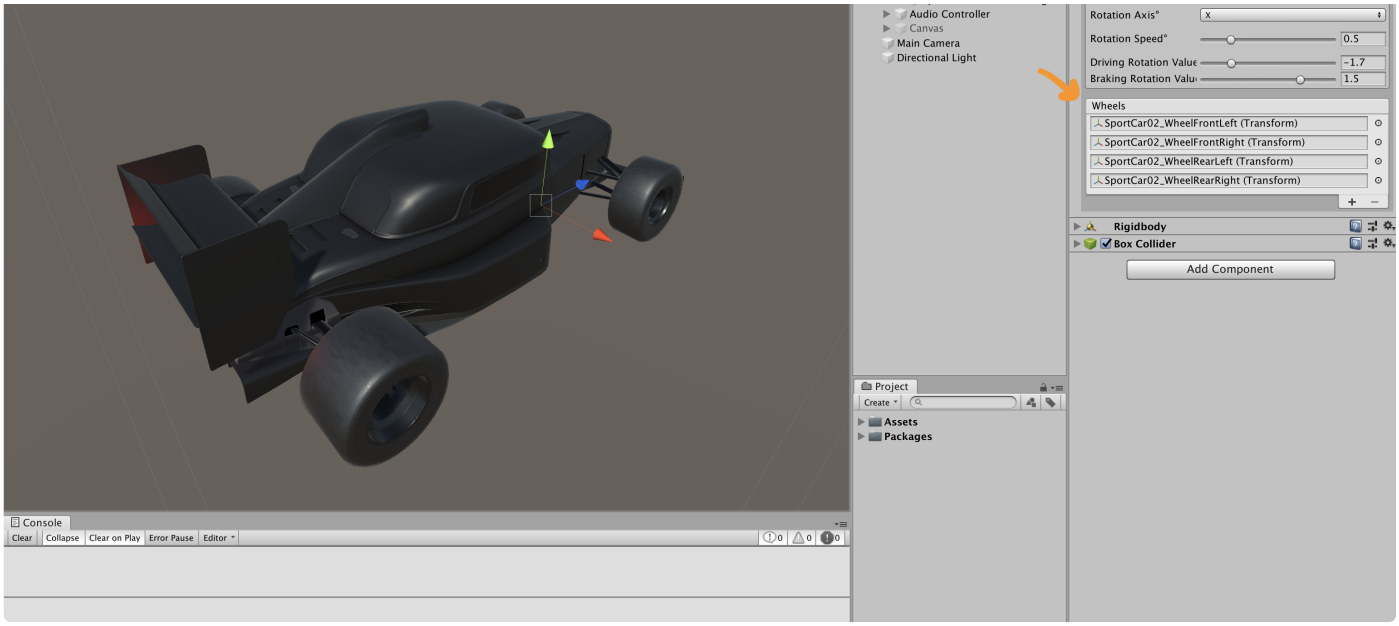
To create a car do the following:

1. Add the **CarController** component on a car model

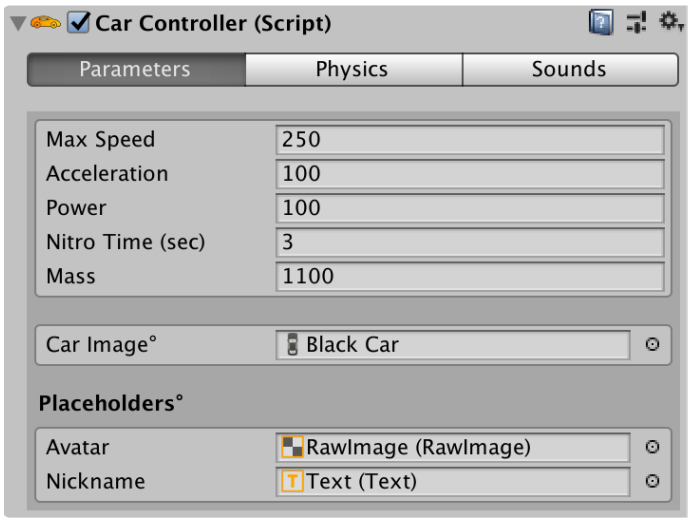


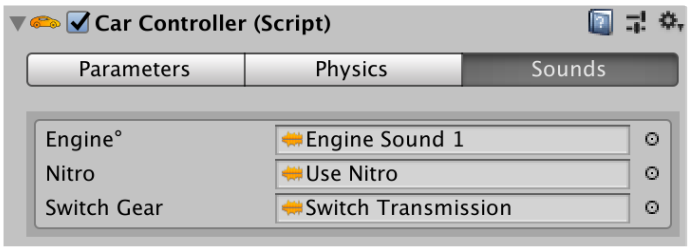
2. Set car wheels and body in the script





3. Adjust other parameters (sounds, images, UI placeholders, etc)

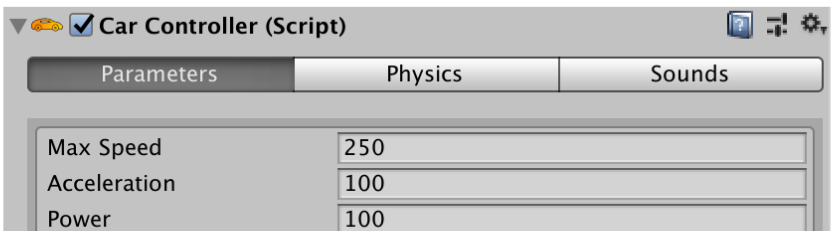




Parameters

Main Values

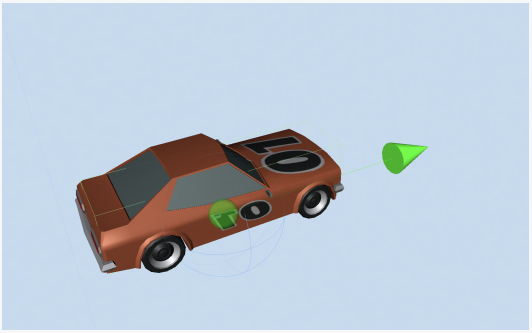
In the **Parameters** tab, you'll find all parameters, that affect the car movement. And also they can be upgraded in the game menu (read more here).

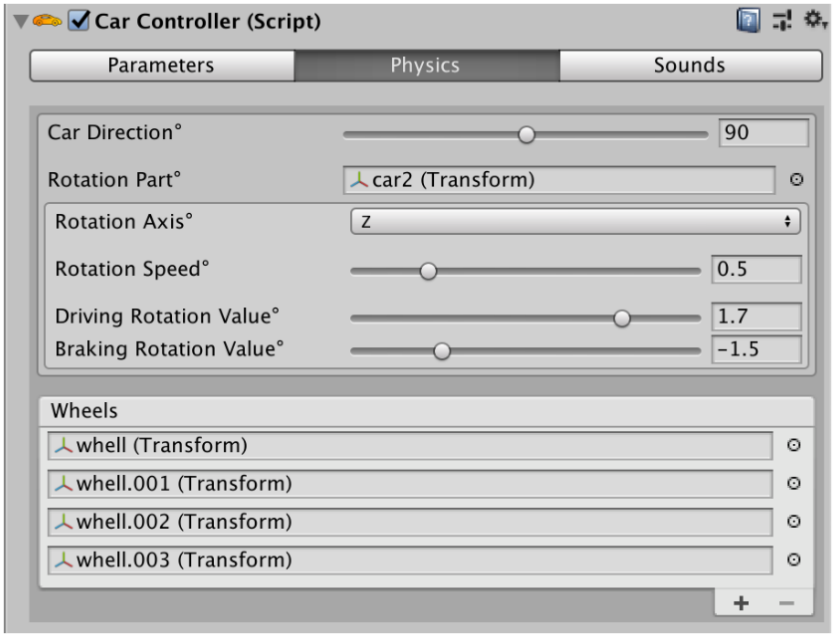


Physics

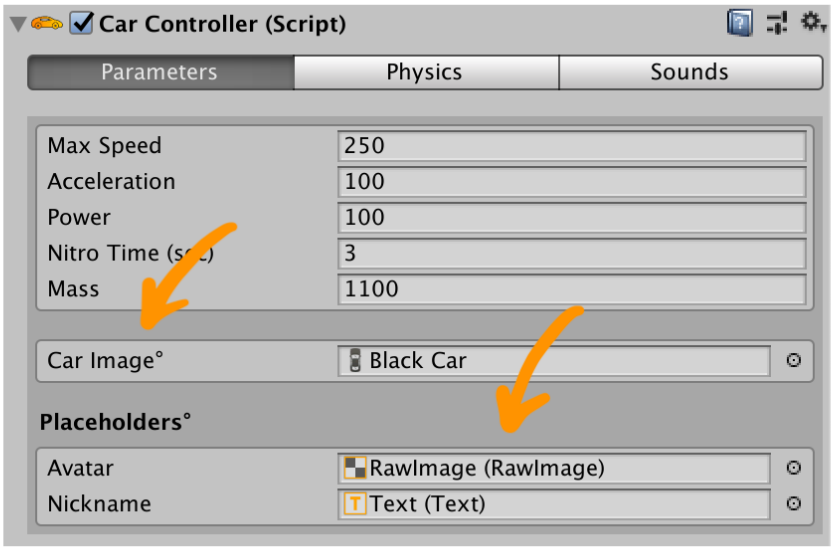
All body physics is generated automatically when driving. All you need to set is the base driving and braking rotations.

Wheels rotate automatically depending on the speed of the car.

> Don't forget to set the **Car Direction** value so that the **green arrow** on the car prefab looks straight ahead.
>
> 



Images

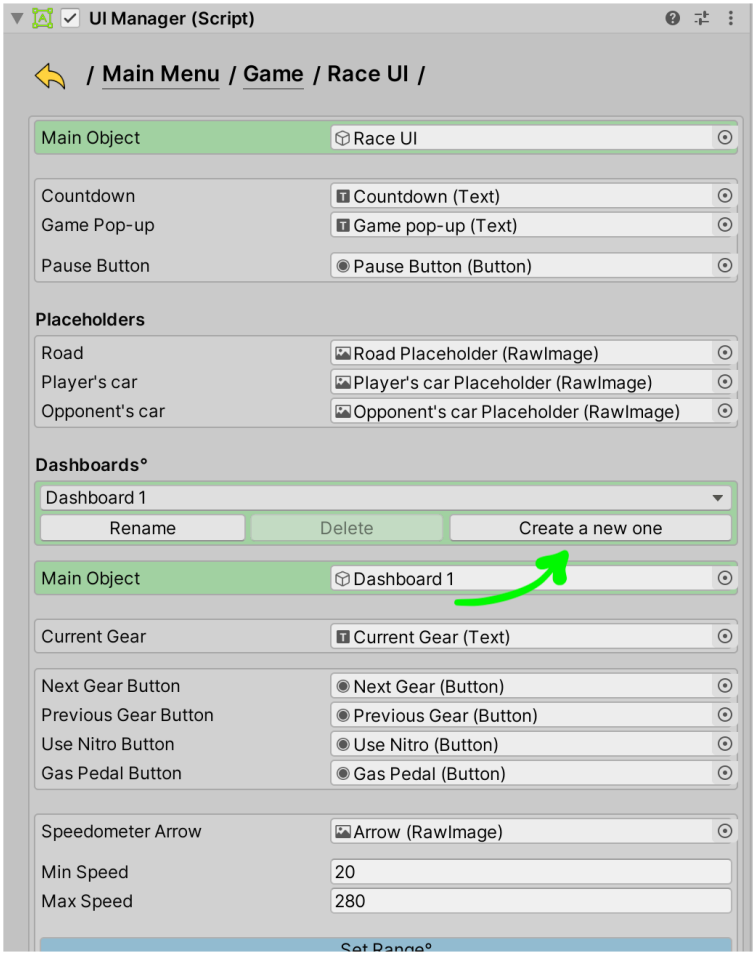


A **Car Image** will be displayed on the screen to show the position on the road.

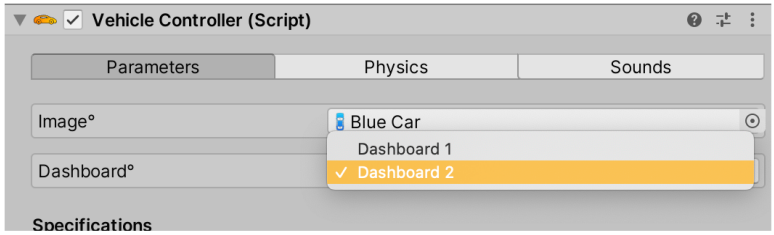
Placeholders will show the opponent's avatar and name (above the car).

Dashboards

- You can create several dashboards and then apply each of them to different cars. To create a new dashboard, go to **UI Manager / Game / Race UI**.

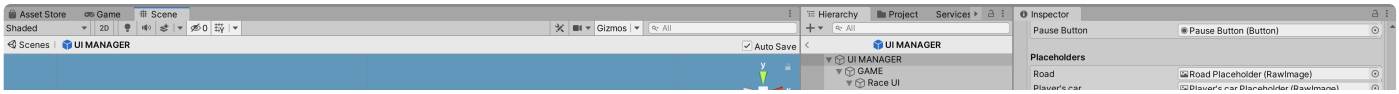


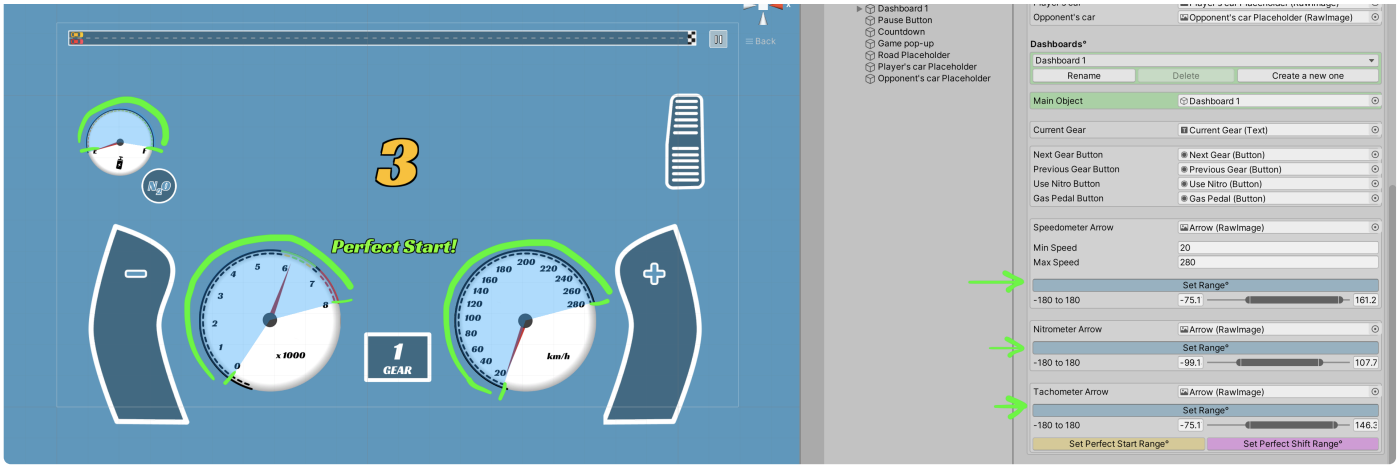
- To select a dashboard for the vehicle, open the **VehicleController** script and select one of the dashboards you created in the **UI Manager**.



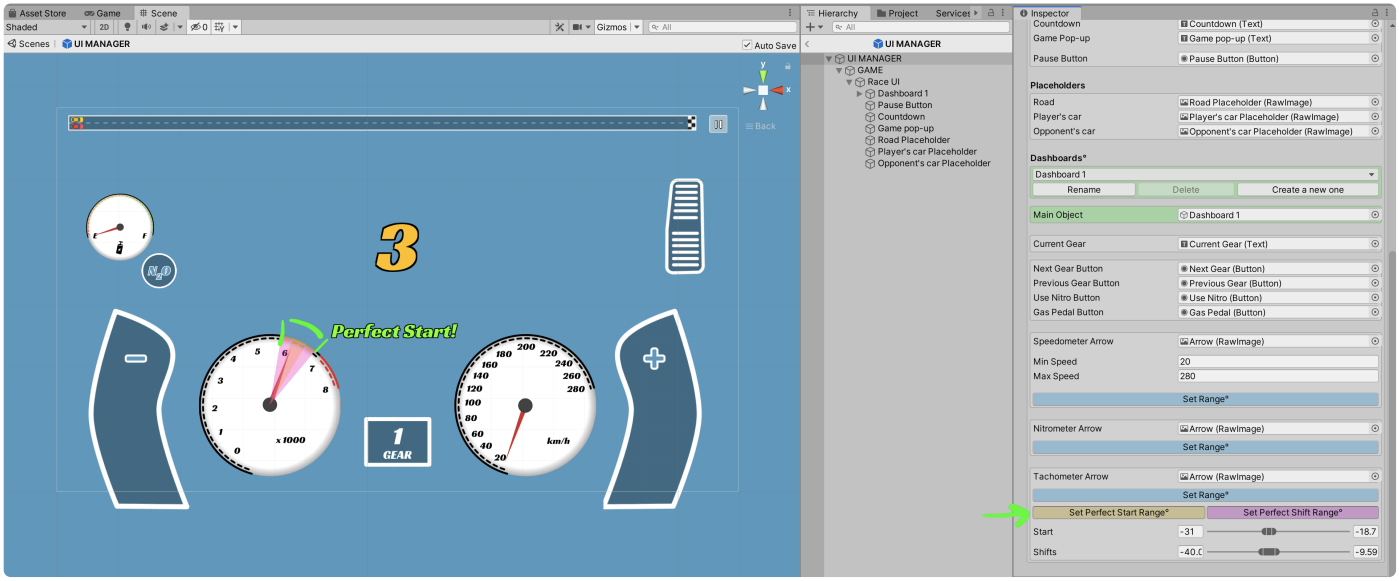
- After you set all images, you need to adjust **Ranges** for the speedometer, tachometer, and nitrometer.

> These limits are needed so that arrows moves within the values on the instruments





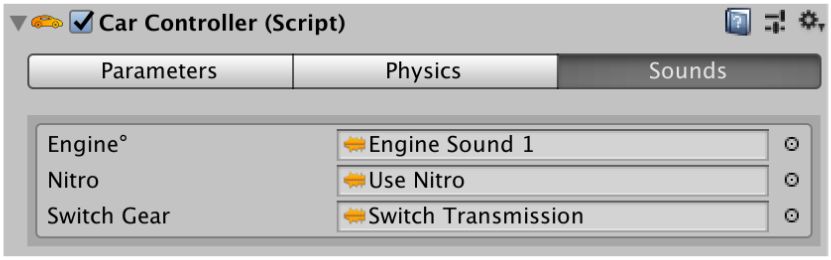
- Also, you should adjust the limits for a **perfect start** and **perfect gear switches**.



Audio System

There is a powerful audio controller in the **CarCntroller** script. With it, you need to set only 3 sounds: **Engine**, **Nitro**, and **Transmission Switching**.

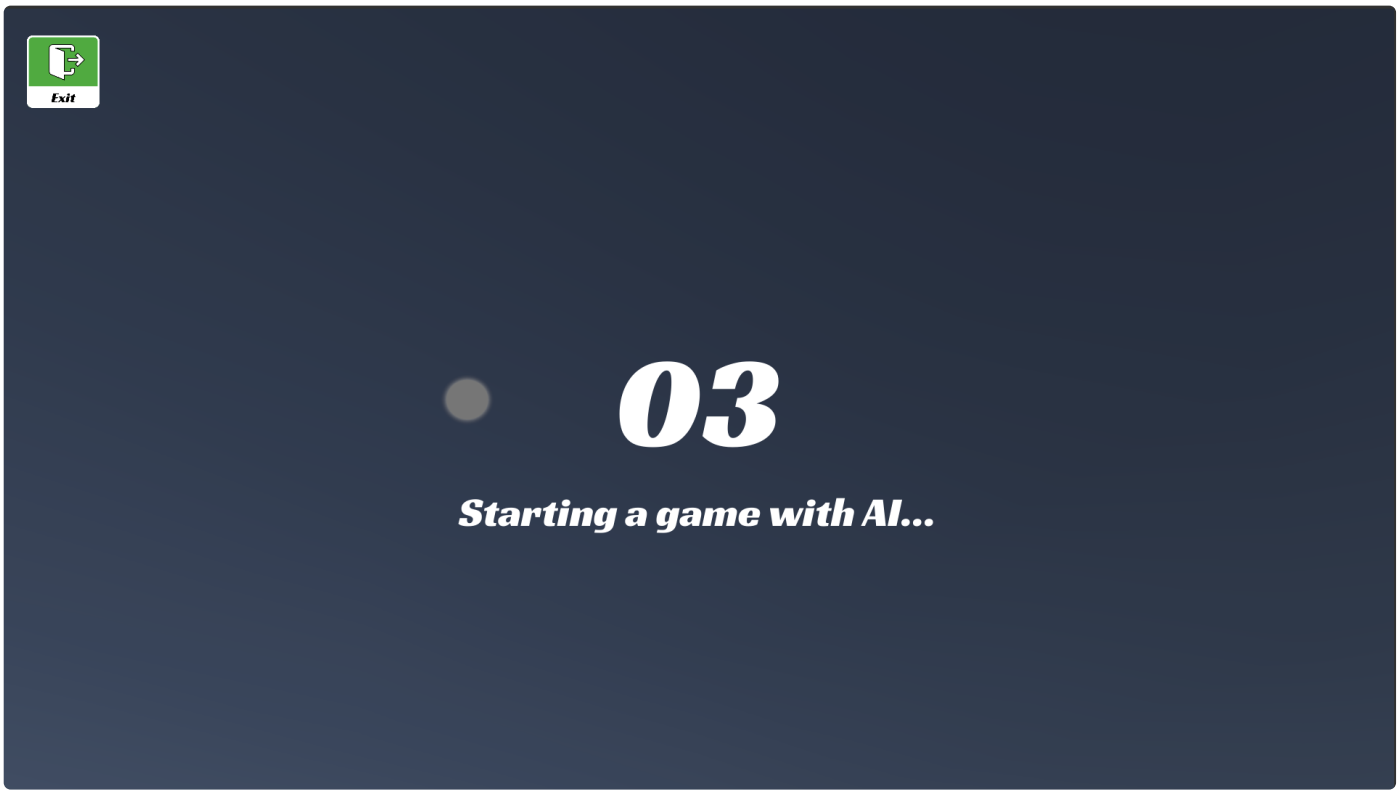
When a car is driving all sounds (like slippage, acceleration, braking, etc) are automatically generated from the Engine sound (using different pitches, volume, etc).



Built-in AI Controller

The AI controller is embedded in the **CarController** script. So, all cars you set in the **MenuManager** script (read more here) will be randomly used as opponents.

- The AI controller has events - perfect start, perfect shifts, using nitro (they will start randomly).
- (When the Random Opponents parameter is active) If a player installs a few upgrades on his car, then an opponent will have the same upgrades to match the player's car (in this case, players will always feel rivalry).



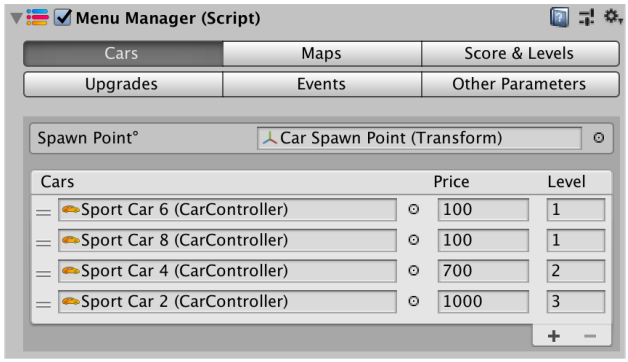
> ⓘ By default, players ride with AI, but you can enable a **multiplayer module** to play with live opponents (read more here).
>
> In this case, if there is an internet connection and a suitable opponent is found, then the player will race with him. If not, then the race will be with AI.

Menu Manager

Cars

Player Cars

Add cars in the [Cars] section and set their required levels and prices (read here on how to create a car).



> At least one car must be available for players at the beginning of the game, so set the 1st level and an affordable price for it.

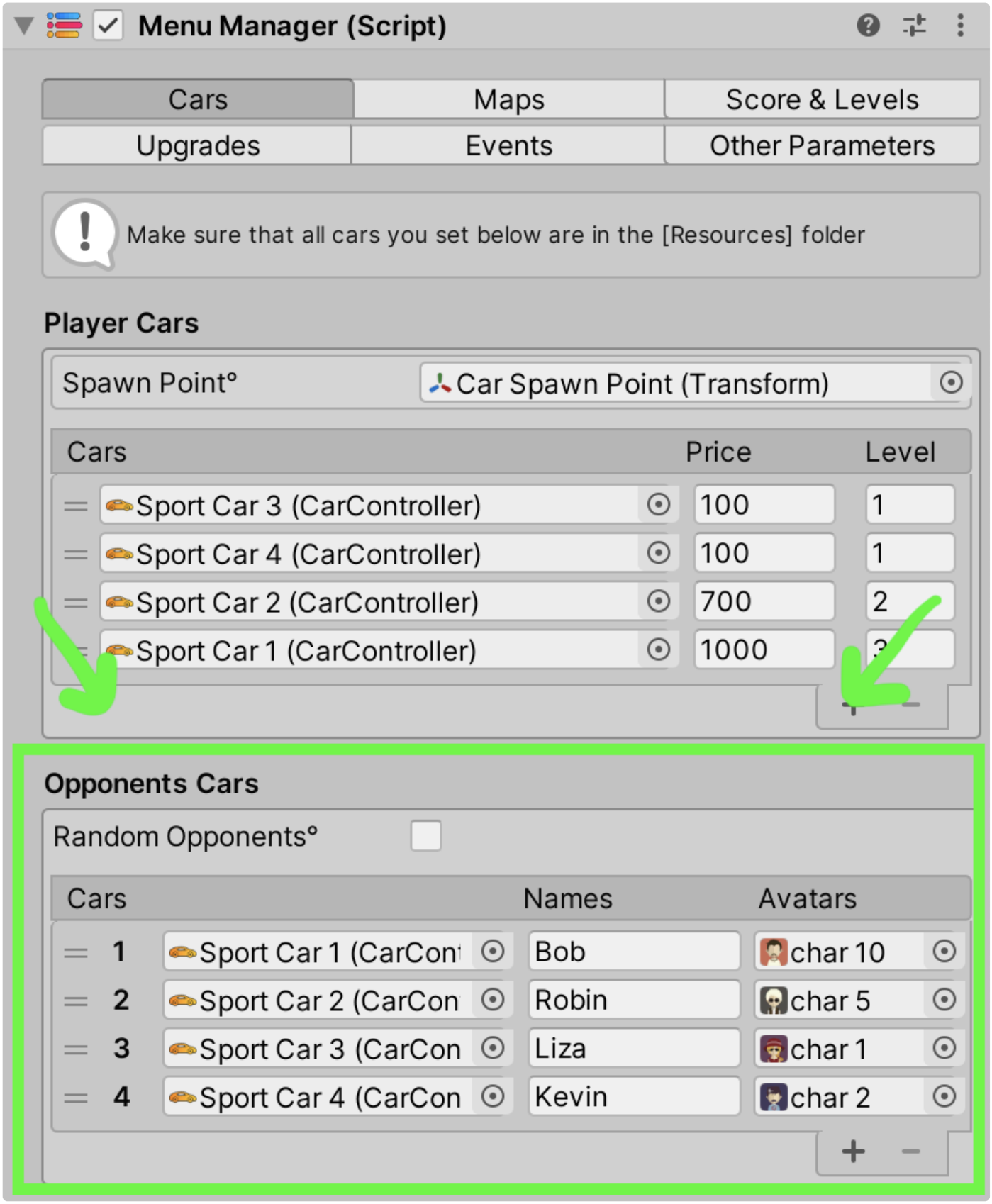
During the game, all cars from this section will be displayed in the game shop.

Opponents Manager

Below in this section, you can find the manager of opponents. Add opponents and cars there so that the player can race with them.

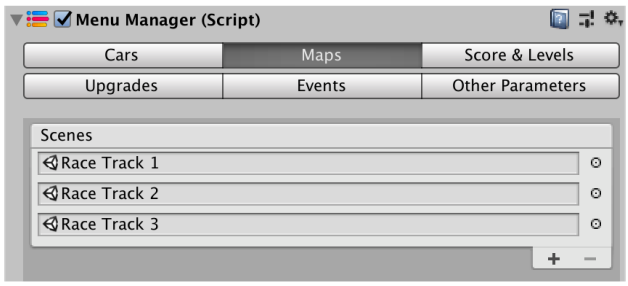
If the **Random Opponents** parameter **is active**, the opponent's car will be randomly selected.

If **it is not active**, the player has to win the opponents in turn - when he wins one, he goes to the next opponent (all progress is saved).

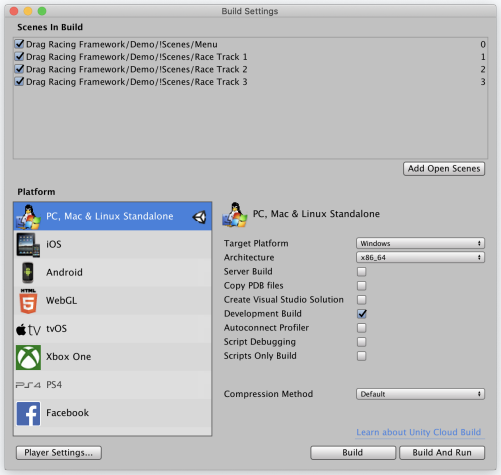


Race Tracks

Here you should set all race tracks (read here on how to create a race track). During the game, they will be selected randomly.



When you add or remove a level, the Build Settings menu is automatically updated. But before building a game, make sure all race tracks and the menu scene are there.



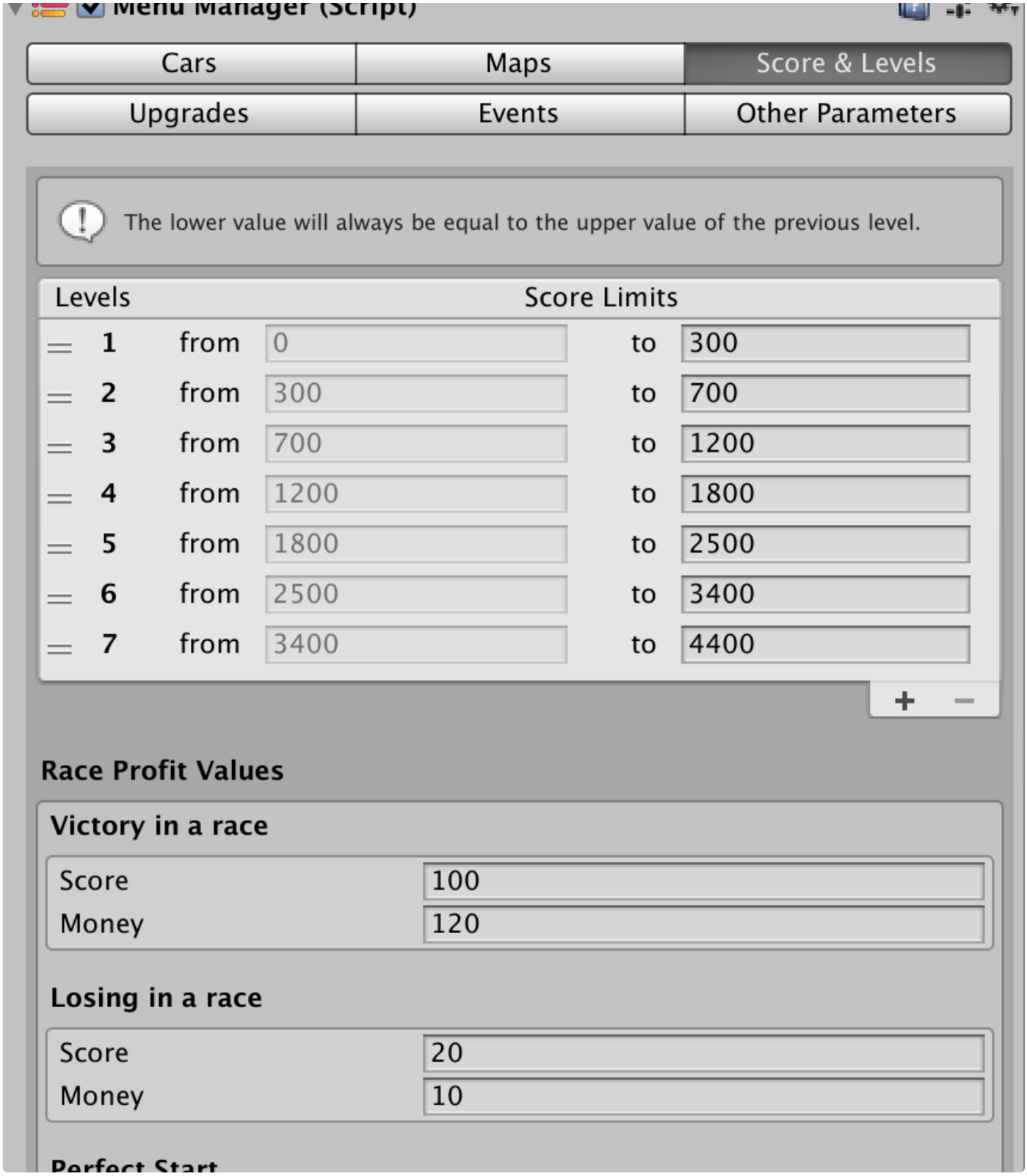
Progress System

Score & Levels Tab

This is a complete gaming mechanics:

- Players participate in races and **get profit** (money and points).
- The more points, the more the player's level.
- By increasing the level and earning money, players can **buy cars** (which you added here) and **upgrade them** (read more here).

In the **Score & Levels** section, you can easily customize all parameters for yourself.



Car Upgrades

In the **Upgrades** tab, you can create upgrades for the 5 units: **Engine**, **Turbo**, **Transmission**, **Nitro**, **Weight**. For each unit, you should add stages, set the values that each stage will increase, and also write the required levels and prices.

Cars | Maps | Score & Levels

Upgrades | Events | Other Parameters

Upgrades Type° : Add Percent

Engine Upgrades

№	Price	Level	Speed	Boost	Power	Nitro	Mass
1	100	2	1	1	1	0	0
2	500	3	1	1	2	0	3
3	600	4	3	1	1	1	3

Turbo Upgrades

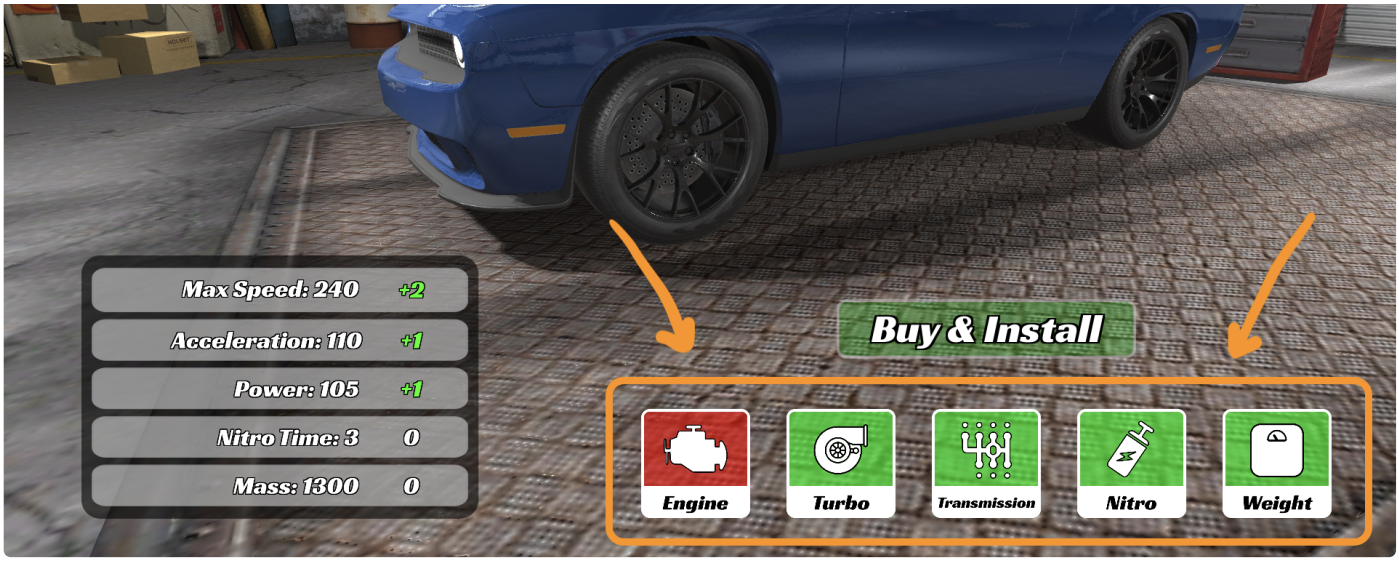
№	Price	Level	Speed	Boost	Power	Nitro	Mass
1	700	3	1	2	1	0	1
2	800	4	1	1	3	0	0
3	850	5	3	1	1	0	0

Transmission Upgrades

№	Price	Level	Speed	Boost	Power	Nitro	Mass
1	300	2	0	3	1	0	-2
2	510	4	0	5	1	0	-2
3	780	5	1	6	1	0	-3

Nitro Upgrades

These 5 units will be displayed in the game menu (in the upgrades shop)

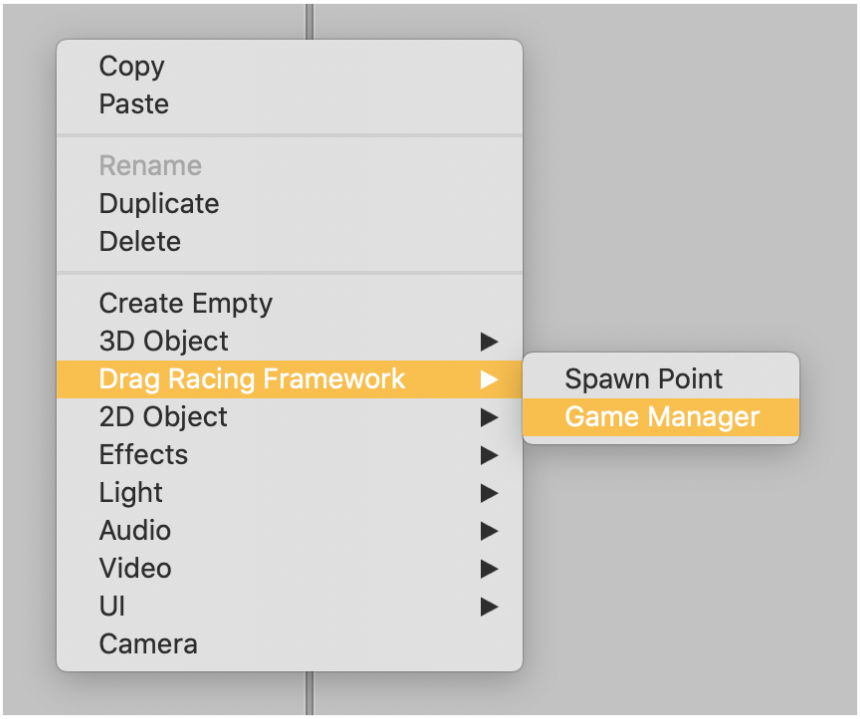


All stages and values will also be displayed in the menu

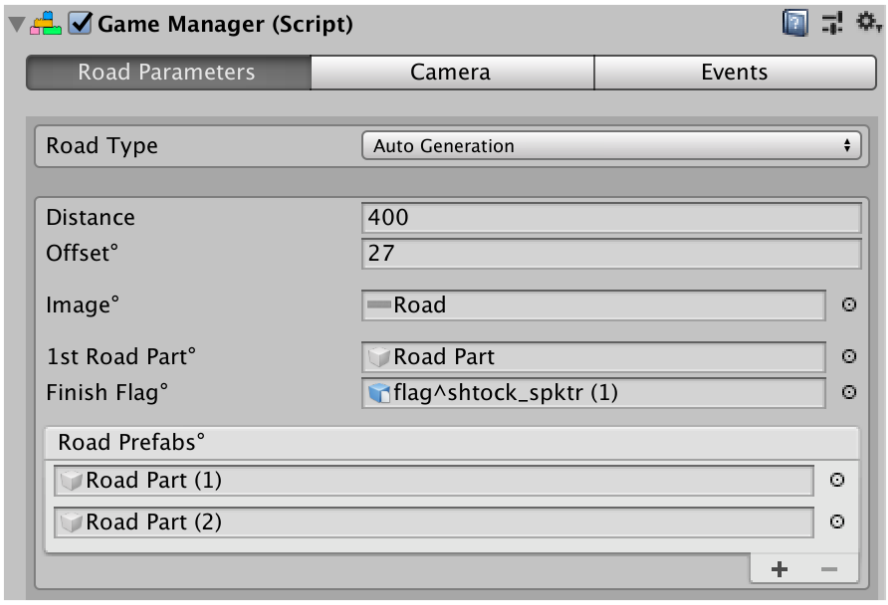
Game Manager

How to create a level

1) Create a **GameManager** by pressing **Right Mouse Button -> Drag Racing Framework -> Game Manager**

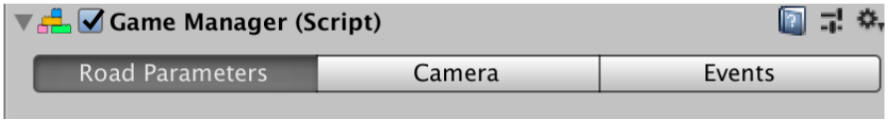


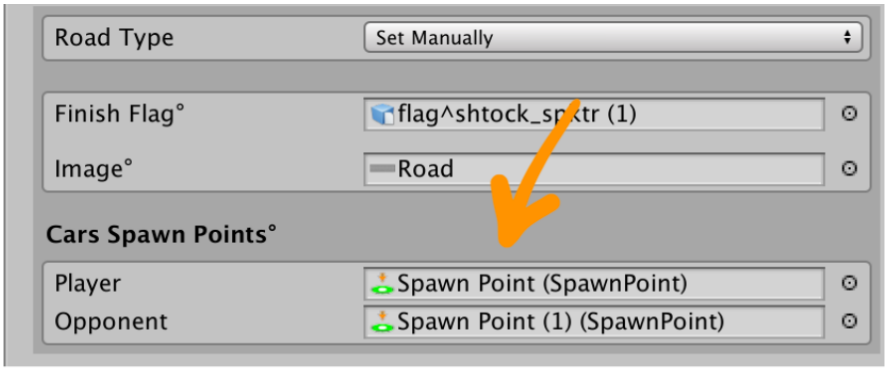
2) Choose a **Road Type** and set up parameters (read more about the road controller here)

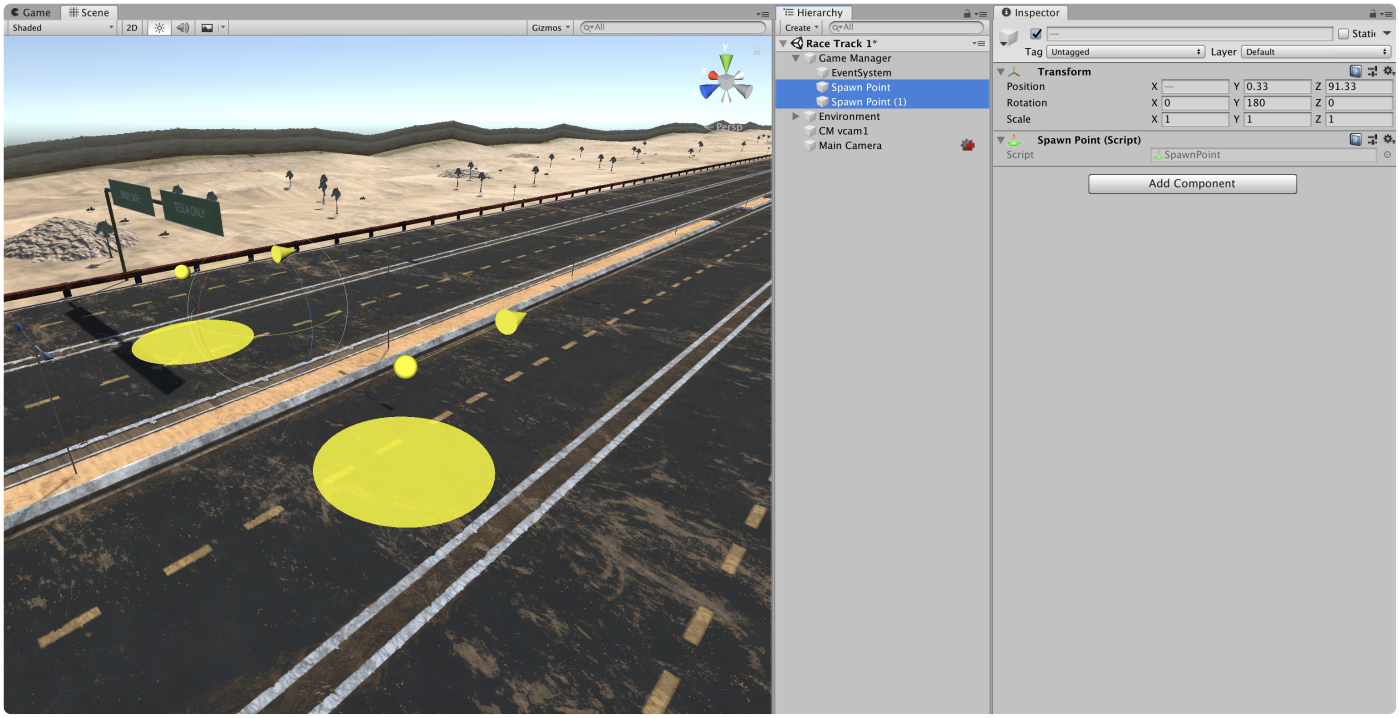


3) Set the spawn points and choose the race direction

> ⓘ To create a spawn point press **Right Mouse Button -> Drag Racing Framework -> Spawn Point**







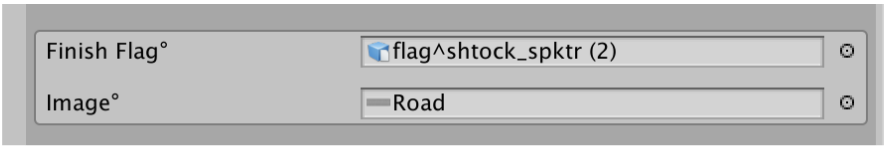
4) Set the main camera (read here about the main camera)

Road Controller

The road controller allows you to create a race track using 2 different ways. But in both cases, the race ends when the player crosses the finish flag.

Set Manually

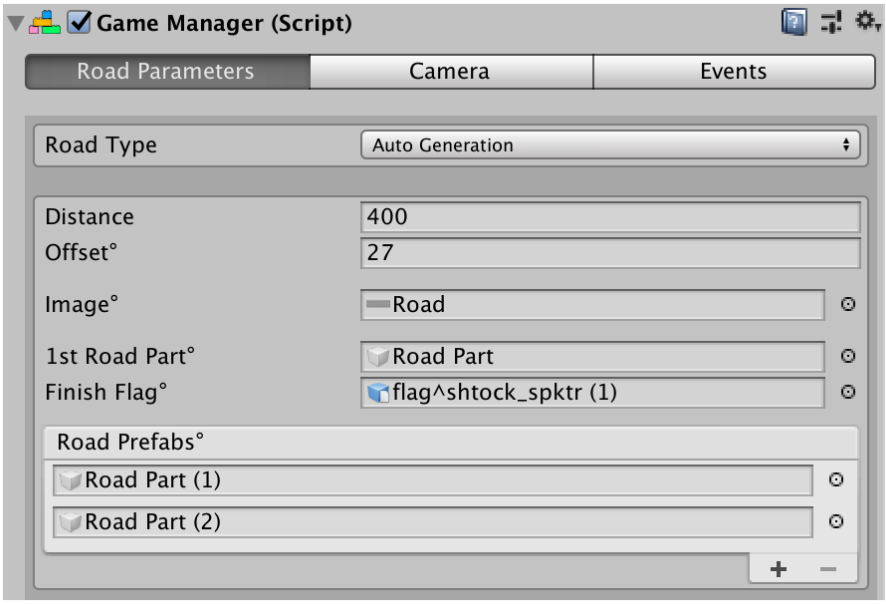
In this method, you need to add the whole track on a scene, then place a finish flag model at the end.



Auto Generation

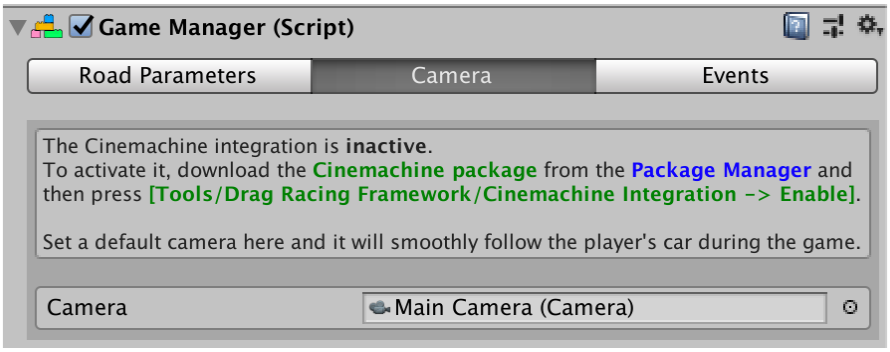
In this case, the road will be generated automatically. All you need is to set the **1st Road Part** (it will be the start of the race track), **Road Prefabs** (they will be randomly spawned), and the **Distance**.

Also, adjust the **Offset** parameter, so that each part of the road is generated immediately after the current.



Main Camera

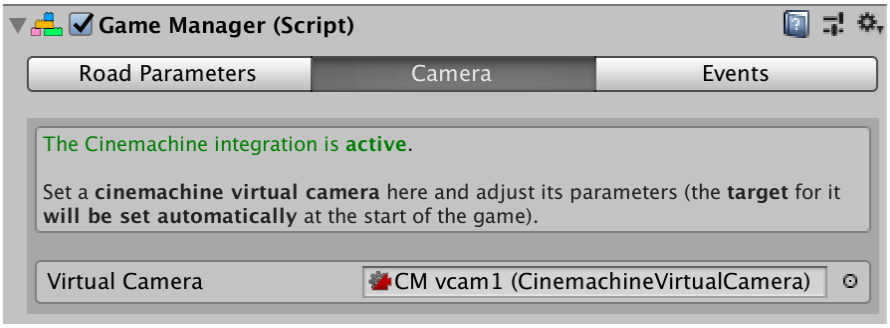
By default, a simple camera is used in the framework. You should place it anywhere in a scene and during the game, it will follow the player's car from that view.



But also you can enable the **Cinemachine** integration (which will allow you to create a virtual camera with effects) - read more about the Cinemachine and its features here.

To enable the integration do the following:

- Install the Cinemachne tools from the Package Manager.
- Press **Tools -> Drag Racing Framework -> Integrations -> Cinemachine Integration**.
- Create a virtual camera and add it to the GameManager script.



> ⓘ You don't need to set a **camera target** for the virtual camera manually because it will be chosen automatically during the game.

Other Components

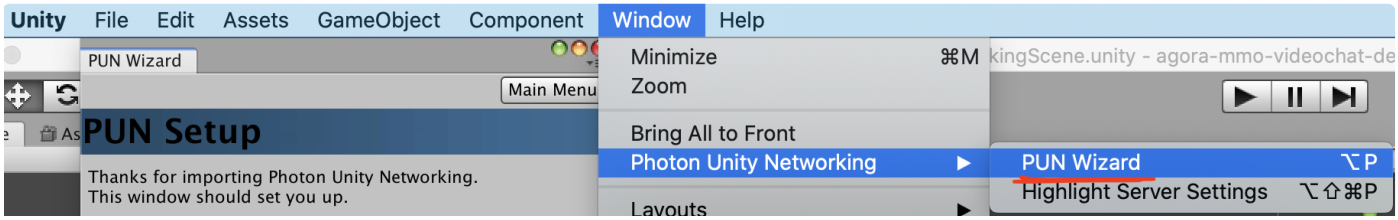
👥 Multiplayer Module

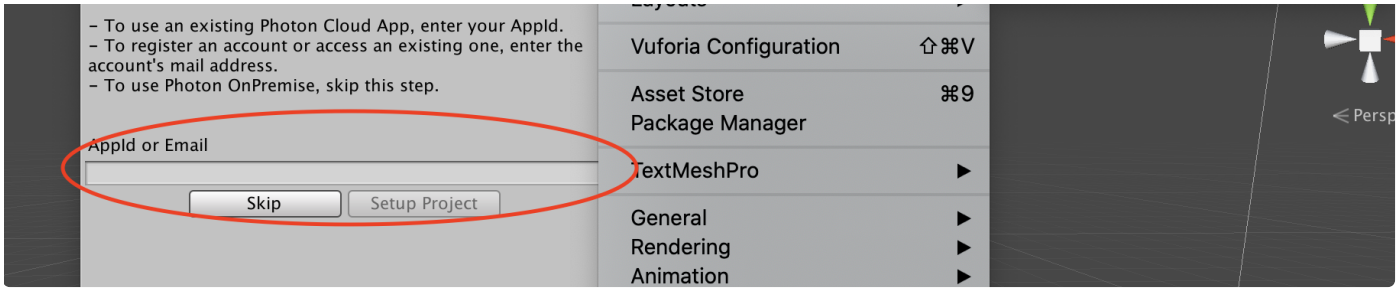
Activation

1) Import the PUN 2 from the Asset Store

2) Write your project ID in the **PUN Setup** window **[Window -> Photon Unity Networking -> PUN Wizard -> Setup Project]** (also this window will appear immediately after import).

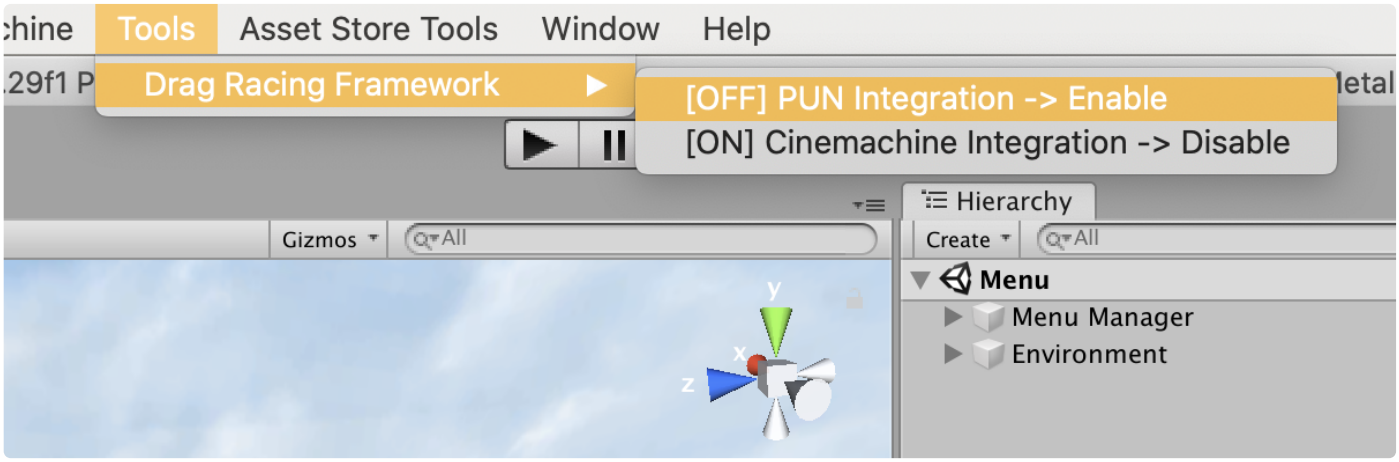
> ⓘ You can get an ID on the Photon site, or use our: **5f8e39db-2ea0-4a49-bfb6-0ce36e6e69b5**



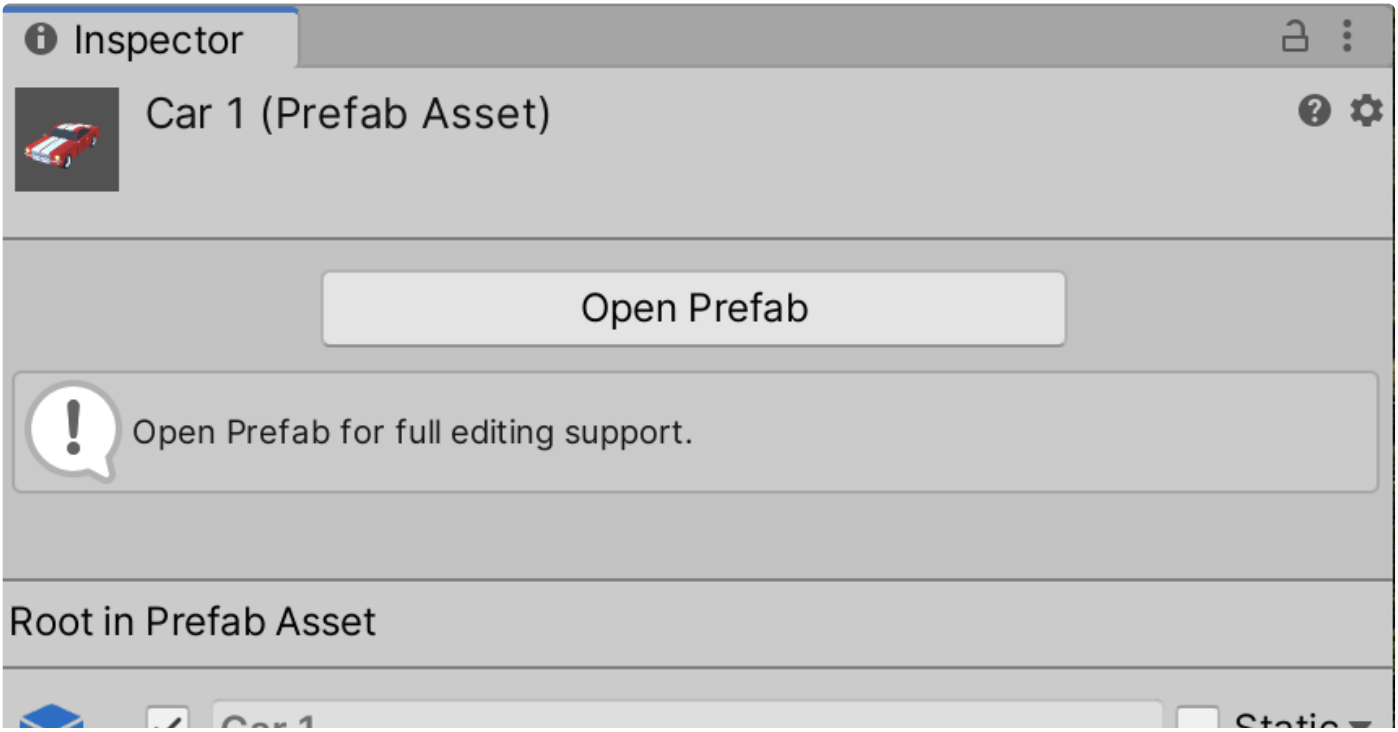


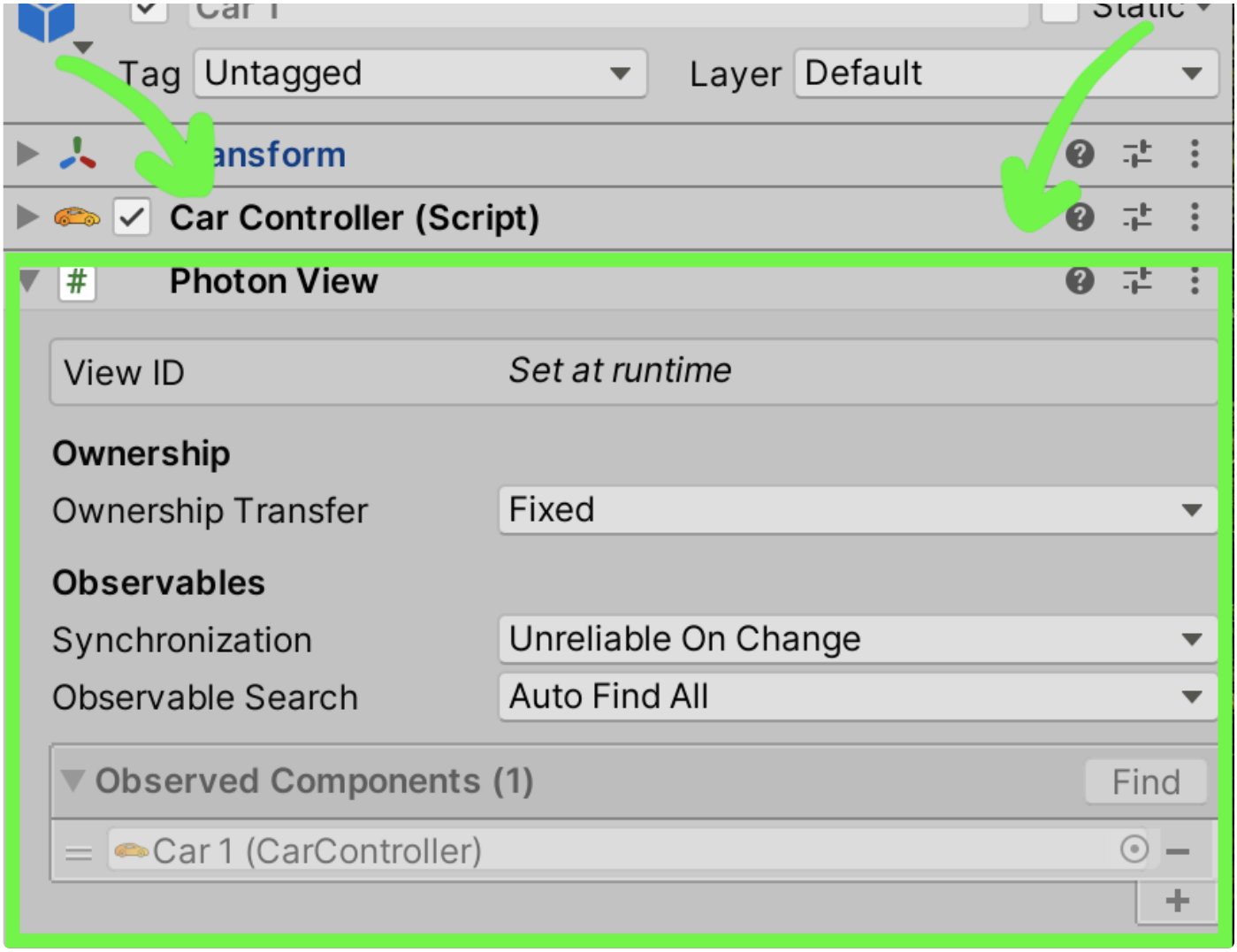
3) Enable the Multiplayer module by pressing **[Tools -> Drag Racing Framework -> Integrations -> PUN Integration]**

> ⓘ You can disable this module at any time by pressing there again.



4) Add the **PhotonView** script on all car prefabs.



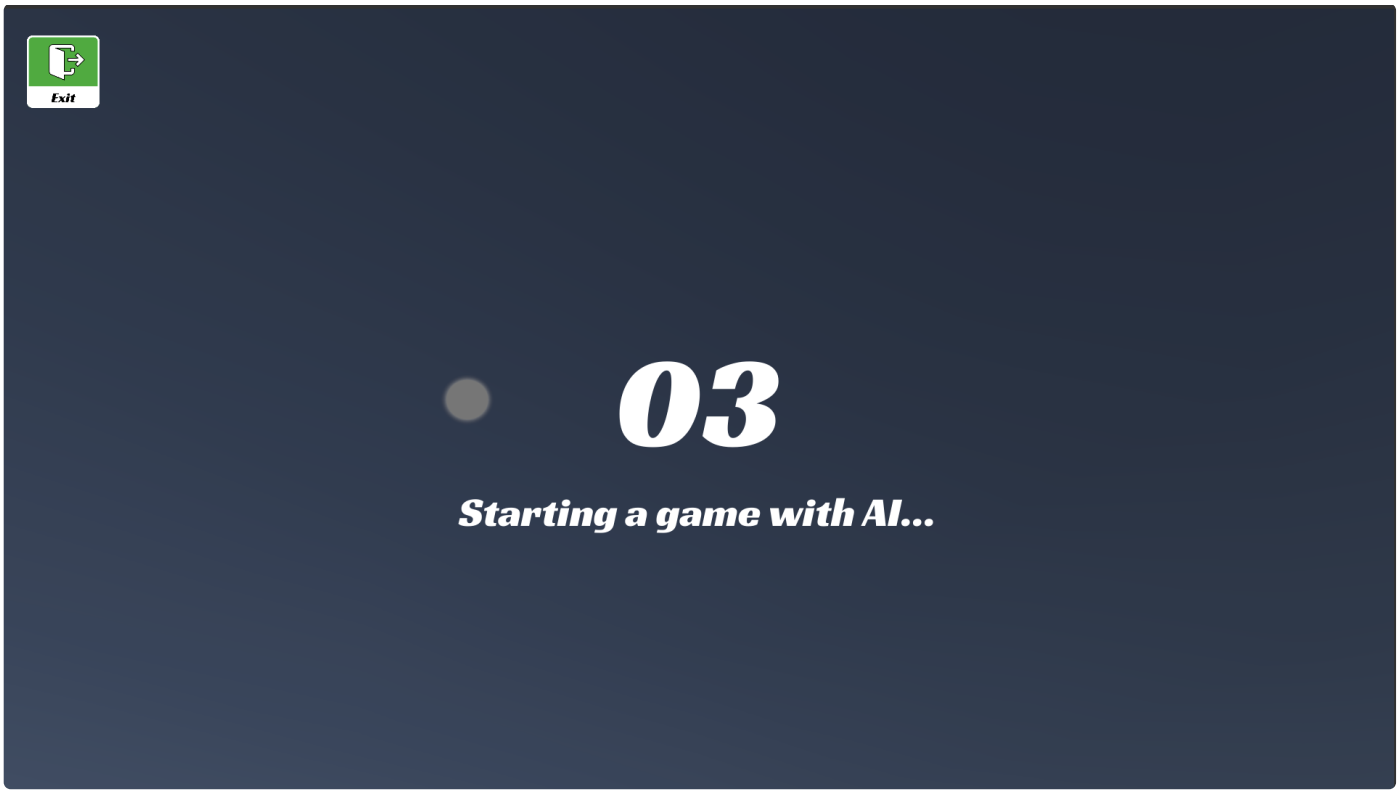


Game Logic

When a player starts the game, it connects to the photon server (using the best region, which you can see in the upper right corner)

When a player presses the "**Race**" button, the game begins to match opponents. If at the same moment someone also presses the "**Race**" button (in the same region), then the game will connect these 2 players and they will race.

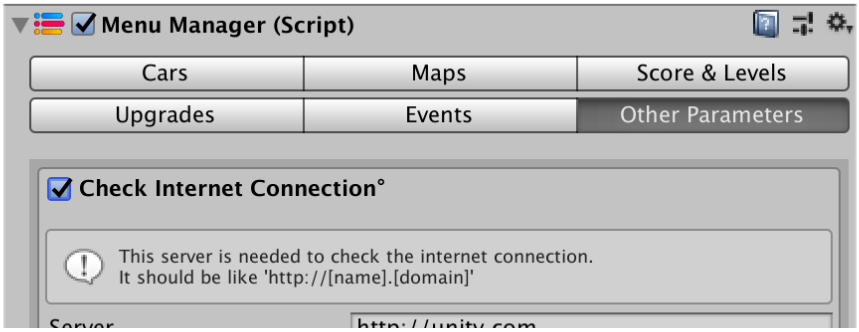
If the opponent is not found or there is no internet connection, then the player will race from the AI.



Check Internet Connection

If this bool is active, the **Menu Manager** will first check if there is an internet connection, and then connect to the Photon server.

To check the connection, the script uses the **Server** value (best to write big sites there - google, unity, etc)



This solution is good for Desktop and Mobile builds because in this case, players will see a notification if they have no network connection.

But you don't need to use it for the **WebGL** build, so in this case, **disable the bool**.

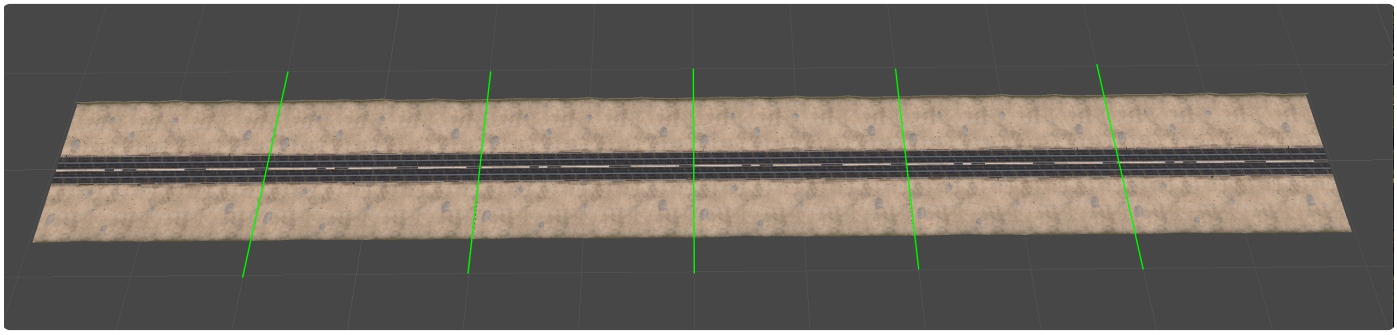
Correct Cars Position

We don't synchronize the position of the cars every frame to avoid lags. Instead, we synchronize the speed and the opponent's car position is calculated on the local client.

But with high ping, the synchronization of cars can be inaccurate (even 300ms is enough for positions to shift in such a dynamic game). Therefore, the script periodically corrects the position of the opponent's car.

How it works:

- At the beginning of the game, the entire race track is divided into several virtual sectors



- The time it took the player to drive each sector is synchronized, and based on it, the position of the cars is corrected

💾 Save System

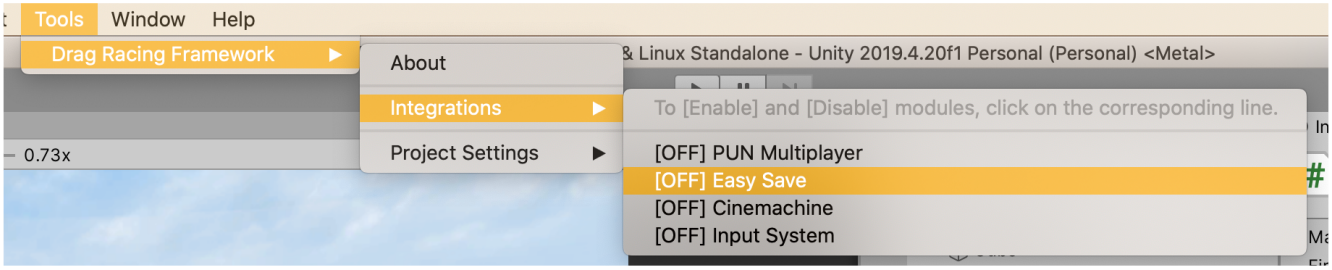
By default, all player data is saved to a json file (you can edit the path in the **GameAssets** script).

```
98          public static void LoadDataFromFile(ref ValuesToSave valuesToSave)
99          {
100
101     #if !DRF_ES3_INTEGRATION
102             var path :string  = Application.persistentDataPath + "/GameData.json";
103
```

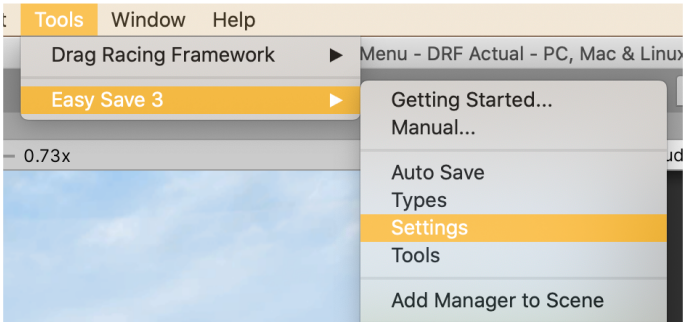
```
85          public static void SaveDataToFile(ValuesToSave valuesToSave)
86          {
87
88              var path :string  = Application.persistentDataPath + "/GameData.json";
89              var json :string  = JsonUtility.ToJson(valuesToSave);
```

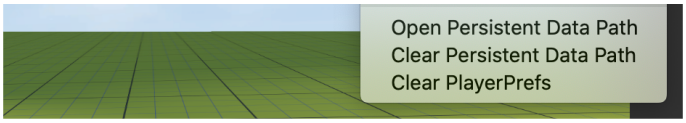
But if you need a more advanced save system, then you can activate the integration with **EasySave** framework:

1. Import the system into your project
2. Activate the integration by pressing ***Tools -> DRF -> Integrations -> Easy Save***



All other parameters (such as save path, format, etc) can be changed in the **Easy Save settings**





⚡ Events

In the **MenuManager** and **GameMangers** scripts, you will find many built-in events, they will help you to expand the gameplay.