MALOKE GAMES

ASSETS

AUTOMOTIVE GAUGES

GUI & CODE

Essential Instruments

- **Contact: Maloke7-Games@yahoo.com.br**
- **Full Portfolio:** https://maloke.itch.io/
- **AssetStore:** https://assetstore.unity.com/publishers/26634

- **Quick Instructions:**

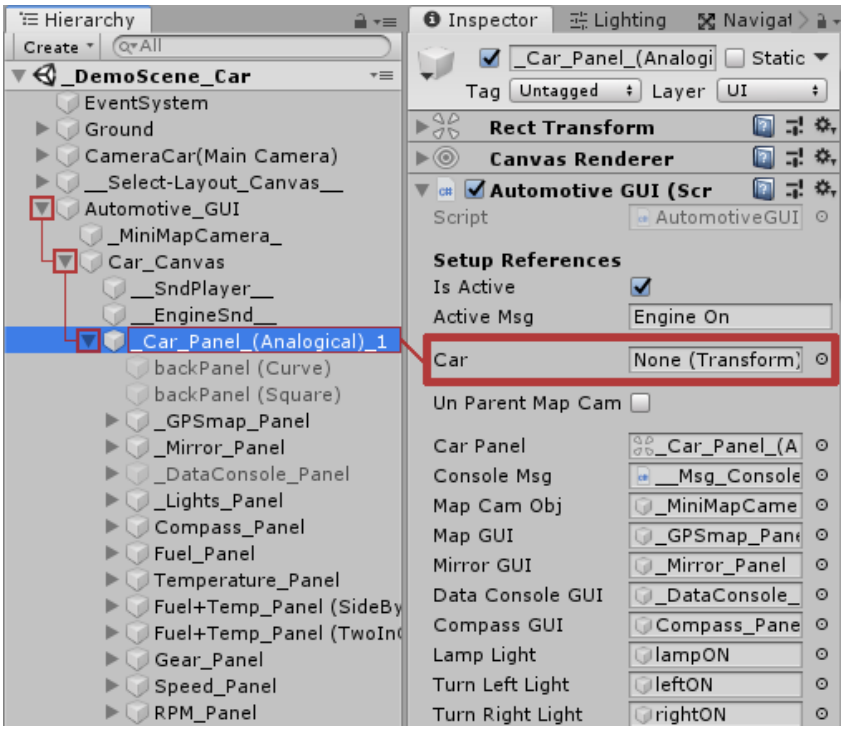
You can find a DemoScene with this asset configured and working straight away for 10 different layouts as examples ***(or you can create your own design by customizing any of the Templates prefabs!)***. The demo uses a very basic camera movement script applied to the main camera that emulates a car movement to help you visualize properly how it works.

If you want to use on your own project/car you ***just need to link a reference of your car's Transform to the main script***. If you leave these fields ***empty*** it will automatically look for the current ***MainCamera*** in the scene.

Just follow these 3 simple steps:

1- Drop on your scene any of the "**x_AutoGUI.prefab**" prefab.

2- Locate the main script "**AutomotiveGUI**" as shown in the image:

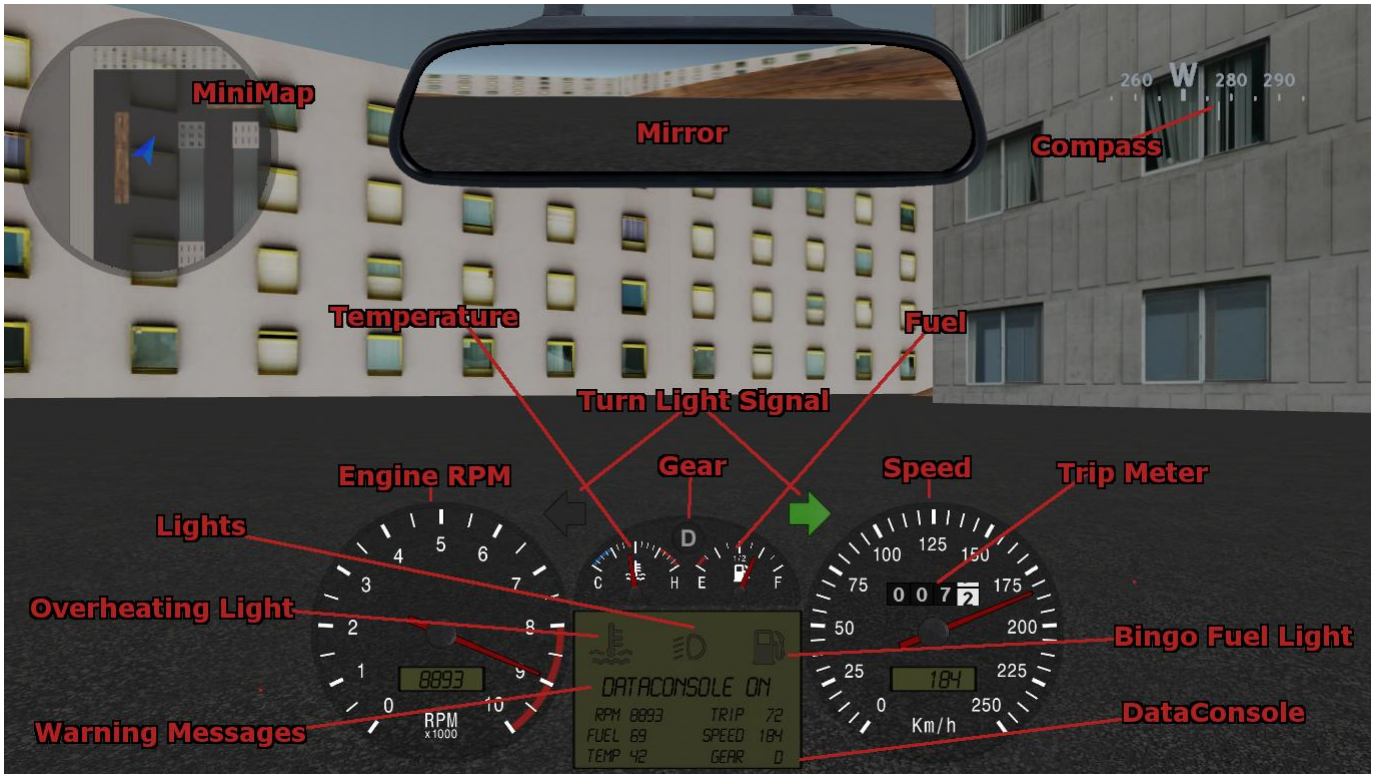


3- Then simply drag your car object to the field "**Car**" and you are ready to go!

To make use of the rear view mirror, just drop the prefab **"__RearMirrorCamera__"** inside your car's transform and position it where you feel it is convenient. This prefab is a Camera that will automatically send the image to the mirror's GUI.

If you want to know more about this asset and how to customize or tweak it, you can find extra information further on this document.

- **Instruments/Gauges Symbology and Overview:**



This asset can be used in **Automatic** or **Manual** mode.

On **Manual** mode, you can set the value of each instrument directly on the **External Controllers** section of the **AutomotiveHUD** inside Editor and set each light on/off from there too.
In manual mode, you can still make use of the **LowPassFilter** factor to smooth readings.

On **Automatic** mode you just have to link your vehicle ***transform*** to the script and all values will be calculated. This includes simple mechanics for *Engine Overheating*, *Fuel*, *Trip meter* measuring drove distance and *Engine Pitch* sound synchronized with RPM.

This asset includes ***6 Prefab Templates*** with variations inside each one. You can start with a prefab and then customize your own version and displacement of the instruments.

Also includes a ***3D prefab configured in WorldCoordinates***, so you can displace the instruments inside a **3D** car model for instance.

Besides the instruments, you can also find a simple topdown ***MiniMap with Zoom In/Out/Reset*** operations, a ***Rear View Mirror*** and a ***Compass Bar***.

- **Main Layouts:**

This asset comes with many example and variations layouts and you can customize it further by changing colors, positions and enabling/disabling instruments that you wish to use. We recommend that you break prefab instance before making modifications to avoid overwriting the original example.

*** Compact ***

*** Compact Transparent ***

*** Full GUI ***

*** Minimalistic GUI ***

*** Large Panel ***

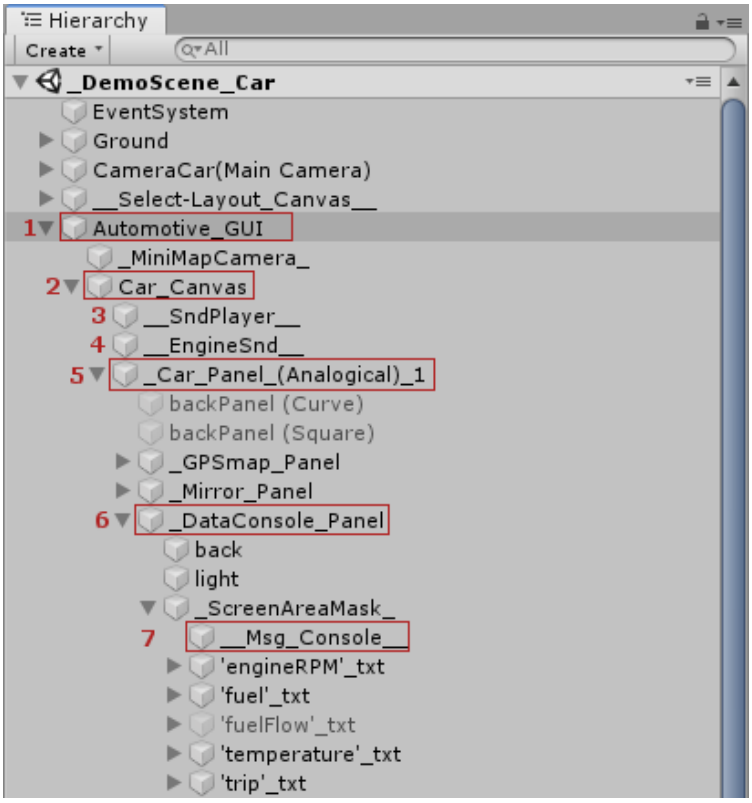
*** Panel with embedded GPS Map ***

* Square Panel *

* 3D Mode for inside Models *

*** Lights On/Off ***

- **Main Components:**



1- The Main root of the PreFab.

2- The MainCanvas of this Asset.

3- Contains script "**SndPlayer**" that manages the General GUI sounds.

4- Contains the **AudioSource** for the **Engine sound**.

5- The main script " **AutomotiveGUI**" which controls the whole Instruments, calculations and references.

6- The **DataConsole** which displays all variables at runtime and console messages. Can be used as an extra instrument or just as an extra help while tweaking values.

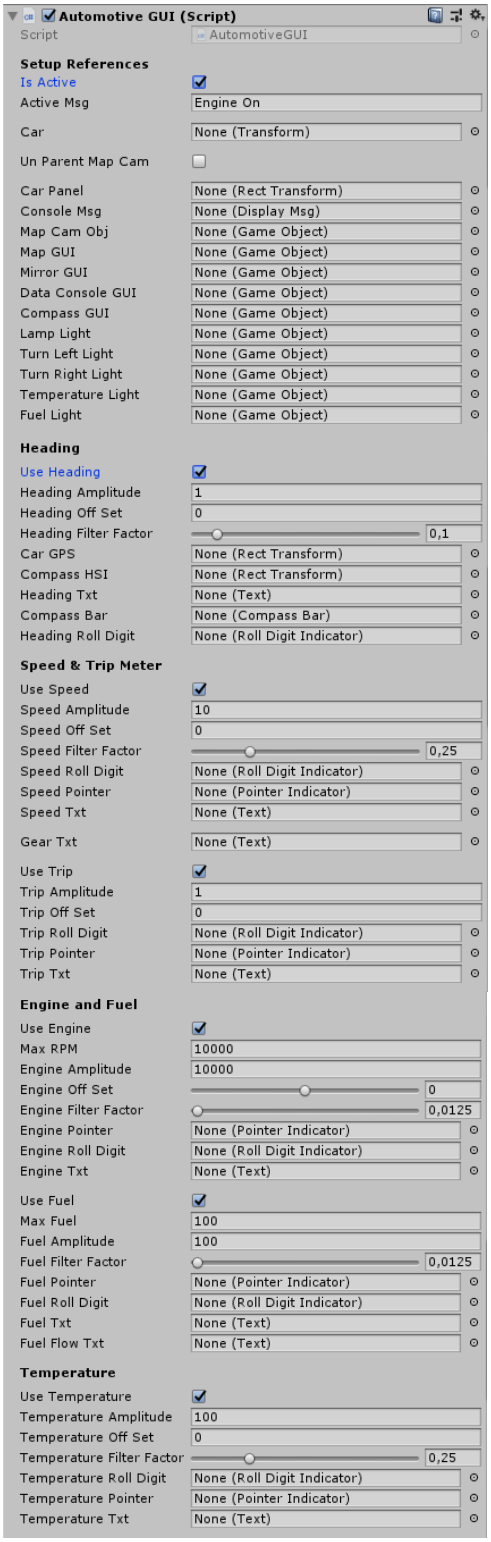
ATTENTION!! - DO NOT DELETE THE DATACONSOLE GAMEOBJECT!

If you do not wish to show it, then just disable the gameobject. Other instruments get some values from it and deleting it may create missing references. Other instruments can be deleted if not used but make sure to check the references for pointer indicators and digits from the main script.

7- Contains the script "**DisplayMsg**" which is responsible to send/show text messages to the console inside the DataConsolePanel.

***Some of these components are used "under the hood" by the asset and do not require any setup or configuration... but fell free to use them for extra purposes on your project if you like!*

➢ AutomotiveGUI - Structure in Editor:



- "**isActive**" determines if the script should be active.

- "**Car**"(***Transform***) is the references to your vehicle gameObject which will be used to calculate all the values displayed on the instruments.

- **ActiveMsg** is the *string* displayed on the console when this instruments panel is activated.

- The others are just references to each GUI element used internaly by the script and doesn't require any setup.

Each section below corresponds to a ***Car Variable*** and the following pattern applies to all of them:

-The *bool* values "**useXXX**" determine if that variable will be used by the script.

- The "**xxxAmplitude**" is a value multiplied to that variable after calculations and before showing it on the GUI. It can work as a ***Scale*** or as a ***unit conversion factor***.

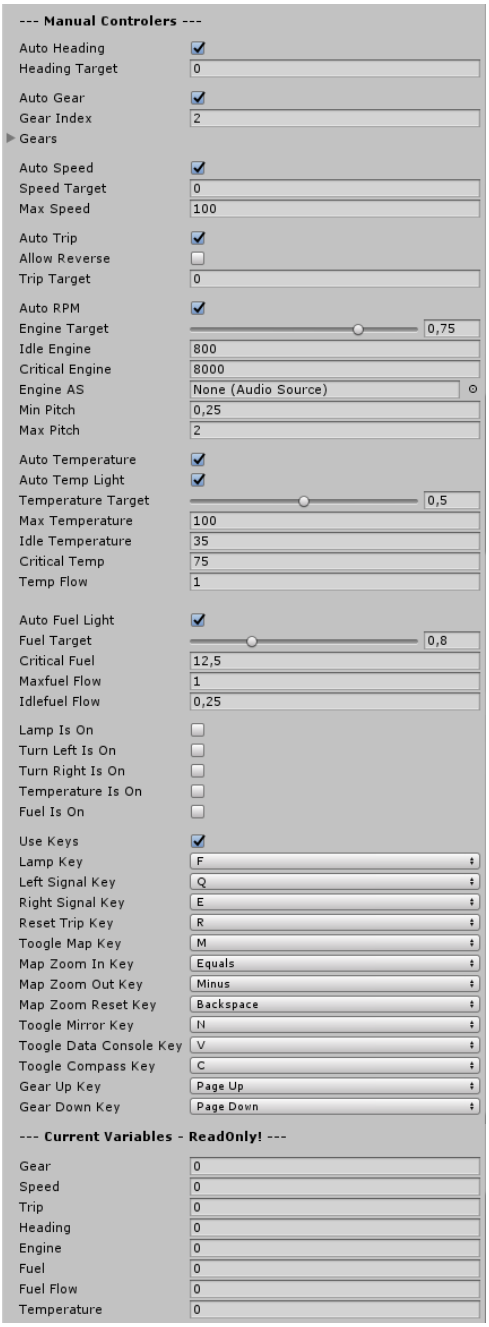
- The "**xxxOffSet**" is a value added to the variable after calculation. It can also be used *to* ***unit conversion*** or simple adjusts.

- All the "**xxxFilterFactor**" values are used to smooth the value shown *(works as a* ***lowpass filter*****)**. If set to **1** it will ***show direct value*** and will have no filtering at all. If set closer to **0** it ***will be smoothed and take more time to reach the final value***.

- The "**CompassBar**" is a reference to the script that controls the compass sliding position to indicate current heading.

- All "**xxxTXT**" are references to a ui *text* component that represent the variable in text format on the **DataConsole**.

- The "**XXPointer**" is a reference to the script that controls the visual indication of that value using the angle of the UI pointer instrument.



- On **the Manual Controllers** section you can set the instruments value manually or let the script calculate it automatically. Also, you can configure some of the critical values, lights and Keys as follow:

-The ***bool*** values "**AutoXXX**" determine if that variable will be calculated by the script or if you wish to set it manually, for instance, if you already have a car asset that calculates these values and you wish *only to use the GUI* to show that value. For the **lights**, it determines if you wish to set them **ON/OFF manually** using the lights **bools** or let the script **automatically** control when they are lit or not using the **critical values** references.

- The **"xxxTarget"** sliders are normalized between ***0-1 (except for the speed and heading)*** and let's you *set the desired value to be shown* by the instrument. In auto mode these values will be automatically controlled by the script.

- The **"EngineAS"** is a reference to the **AudioSource** used for the **engine sounds**. The **audio pitch** will automatically follow from **Min** to **MaxPitch** value depending on current engine RPM. If you already have an asset that controls engine sounds for you, then you can leave this field ***empty***.

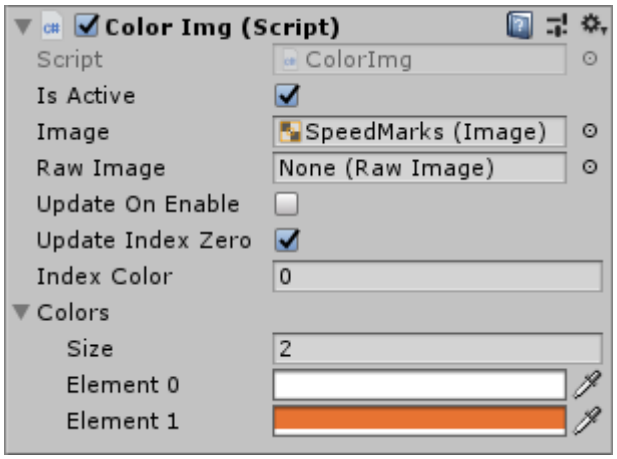
- The **IdleFuelFlow** and **MaxFuelFlow** determine how much ***% of the fuel is consumed for one minute***. The **IdleFuelFlow** determines *the minimum consumption*, while *Max* determines consumption ***when Engine RPM is at 100%*** and interpolates it in-between.

- The Keys sections lets you customize what keys can be used for manipulating the GUI instruments controls. Set the **UseKeys** to ***false*** if you do not want the player to change anything during gameplay.

- The last section shows **Current Variables** values for all the variables in real time inside the Editor. This is just for debug or tweeking and are **ReadOnly**! You can also visualize these variables during runtime using the **DataConsole**.

- **Custom Panel Lights Color (ColorImg Script):**

You can ***customize*** your own colors *for* ***all pointers and instruments*** (Both when lights are *On and OFF*). By default*,* ***pressing F will toggle the panel lights on/off*** using the amber color for lights, but you can customize it by editing the fields of the **ColorImg *script*** as shown below:



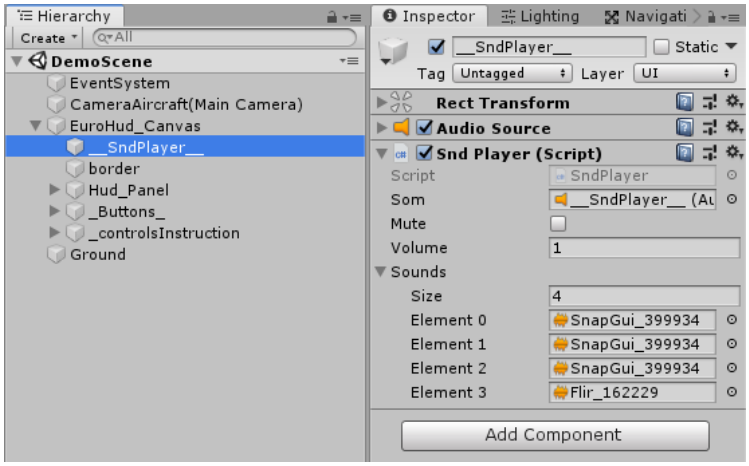
When you toogle Lights, the main script will automatically look for all **ColorImg *scrips*** currently active on the scene and will toogle the image component color for the **Element0 when lights are OFF** and **Element1 for lights ON**.

The option **"UpdateIndexZero"** will automatically allocate the current color used in the editor for the component to the **Element0** during runtime.

A ***quick way*** to change light colors of all components is to type **"ColorImg"** on the **Hierarchy search field** and then selet all the gameObjects containing it to edit then simultaneously to your desired color.

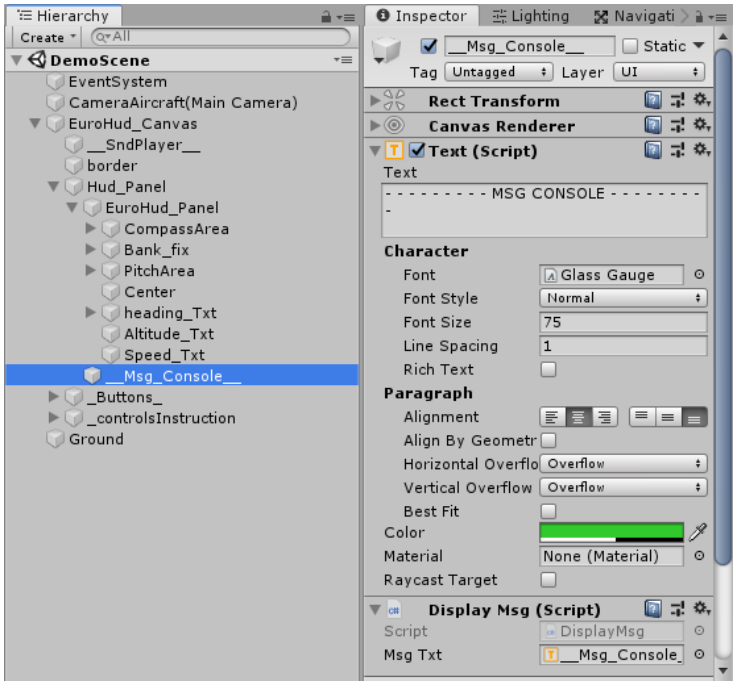
- **Secondary Components (Sound and Message Console) :**

If you wish extra functionalities, you can make use of these components by script calling statics methods:




```
-public static void play(int index);
-public static void play(AudioClip clip, float volume = 1f);
```

(Plays the sound listed on array Sounds with index position or the audioclip itself)

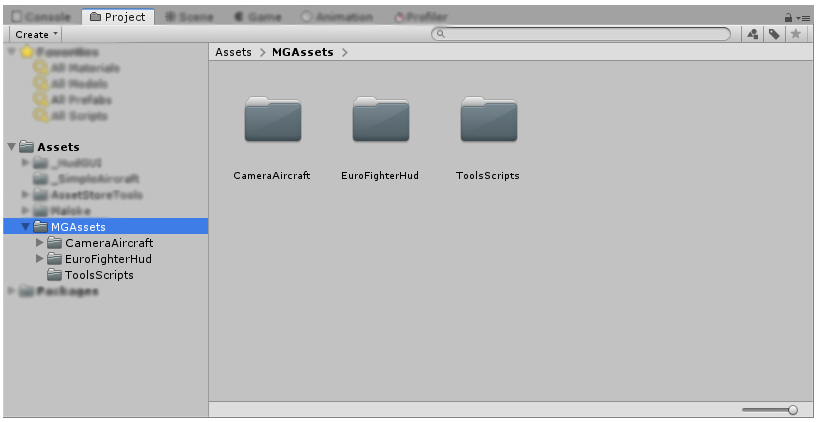



```
-public void displayMsg(string msg = "");
-public void displayQuickMsg(string msg = "");
-public static void show(string msg = "", float timed = 0);
```

Displays a string message on the bottom of the HUD for an amount of seconds (quick is 5s).

- **Asset's Folders Organization:**

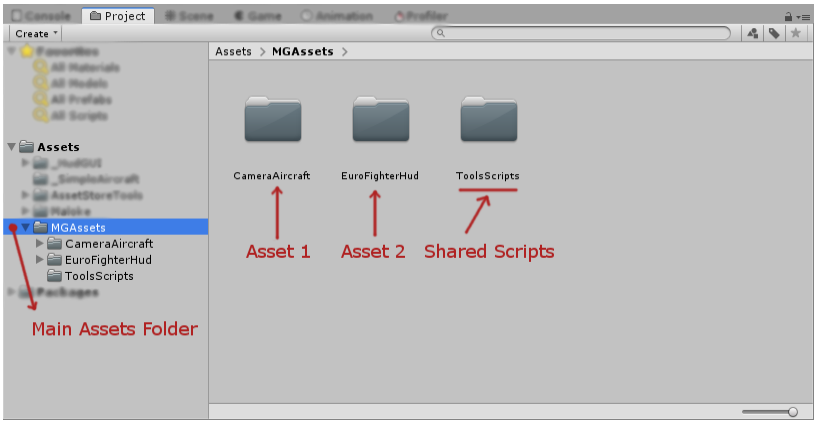
All assets and packages from MalokeGamesAssets will be downloaded/unpacked to a folder called **"MGAssets"** inside the Unity's **"Assets"** root:



Inside **"MGAssets"** you will find a separate folder for each asset package and all their specific resources (like data, scripts, textures, sprites, prefabs, demo scenes and so on...) will be found inside and organized in their respective subfolders.

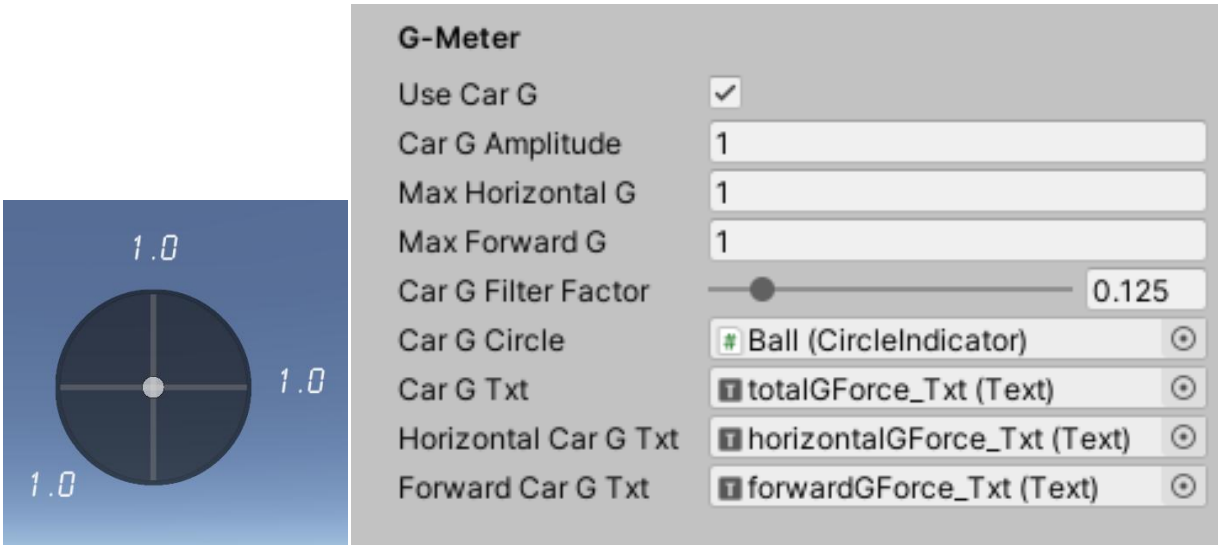
Notice that some assets may use "under the hood" some general scripts and shared funcionalities, so for this reason (and to avoid duplicity or accidental deletion) you will find all this shared tools inside a folder called **"ToolsScripts"**.

Feel free to explore and use them on your projects too, they are simple but handy!



- **Update 1.1 - Accelerometer / Car G-Force:**

This update added a new instrument to display the horizontal acceleration (both forward and lateral). Below is the explanation of its factors:



- The "**UseCarG**" bool will enable the script calculation for the Accelerometer.

- "**CarGAmplitude**" is a value multiplied by the calculated G-Force. It works as a Scale or as a unit conversion factor.

- "**MaxHorizontalG**" determines the value in which the ball indicator will touch the circle on the horizontal axis.

- "**MaxForwardG**" determines the value in which the ball indicator will touch the circle on the vertical axis.

- "**CarGFilterFactor**" is used to smooth the value shown (works as a lowpass filter). If set to 1 it will show direct value and will have no filtering at all. If set closer to 0 it will be smoothed and take more time to reach the final value.