Simple Garage System for

Realistic Car Controller Pro

Thank you for purchasing and using Simple Garage System.

Content

Importing and Installing

Let's get started by importing the package to the project. Please use **Unity 2021.3.2f1** or later versions for the best compatibility. Importer will ask you which assets should be imported to the project, be sure to select all assets. After importing the package, editor window will search for Realistic Car Controller Pro in the project. If it can't find it, it will ask you to import the latest version of Realistic Car Controller Pro into the project.

Simple Garage System won't work without Realistic Car Controller Pro, project will still compile but you won't be able to use the system. Simple Garage System supports all versions of Realistic Car Controller from **V1.37**. It's recommended to use the latest version.

When Simple Garage System finds the Realistic Car Controller Pro in the project, you are ready to go. Go ahead and play the demo scenes first to understand how the system works. Demo scenes are in the scenes folder. There is one main menu scene, and three gameplay scenes. You can test all features of the system and be sure to check everything.

TextMeshPro

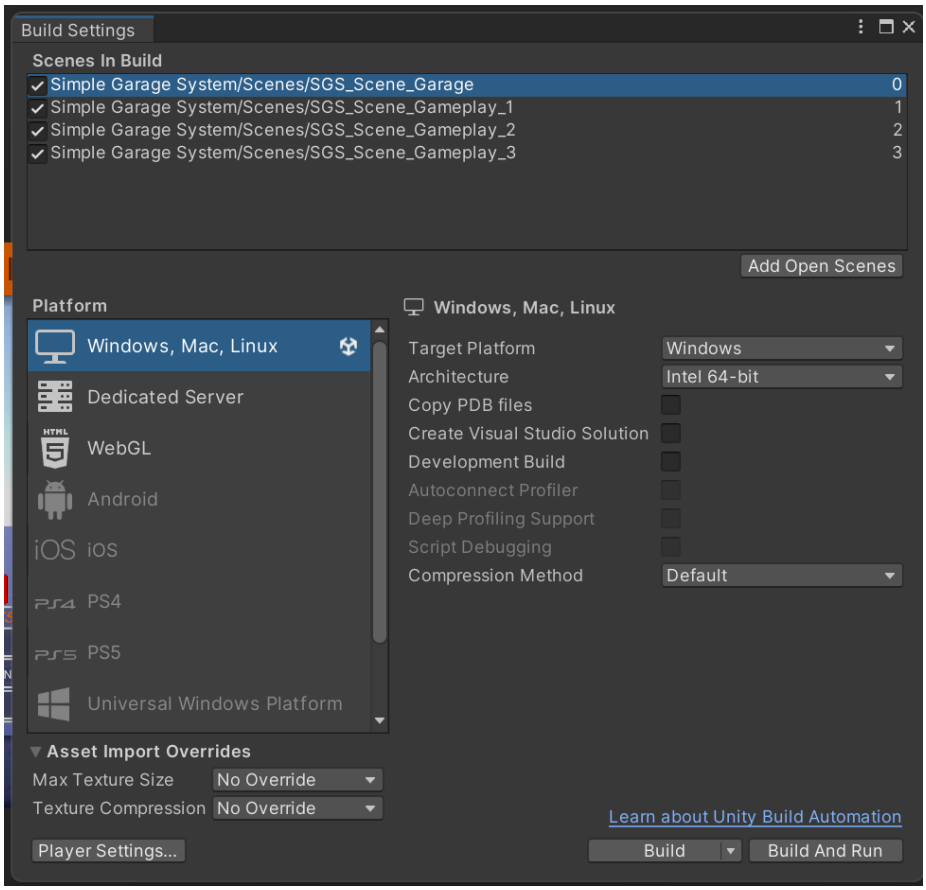
This project and RCCP are using **TextMeshPro**. Editor will ask you to install it when you open the demo scene, please install it. You can also install other resources of the **TextMeshPro** as well.

Overview

Let's take a look how the system works basically. **SGS_SceneManager** is typically brain of the system. Main menu scene must have this manager. It contains spawnable player vehicles, spawned ones, storing them, UI management, and taking control of all sub managers. No worries, system is very simple. When the player interacts with any items, **SGS_SceneManager** will be used. Each item and upgrade buttons have a simple script that communicates with the scene manager. Scene manager checks if the current vehicle has this item or upgrade. If it's not owned, adds it to the cart. Scene manager also checks the player money before purchasing the items in the cart. Same system is for vehicle purchase. Scene manager checks the current vehicle on the scene that if player owns the vehicle or not. As I said, **SGS_SceneManager** takes major part of the system. You might want to check the variables, fields, enums, and methods in this script. And when player is ready to play the target scene, the selected scene will be saved and loaded. That's basically how the system works.

Demo Scenes and Build Settings

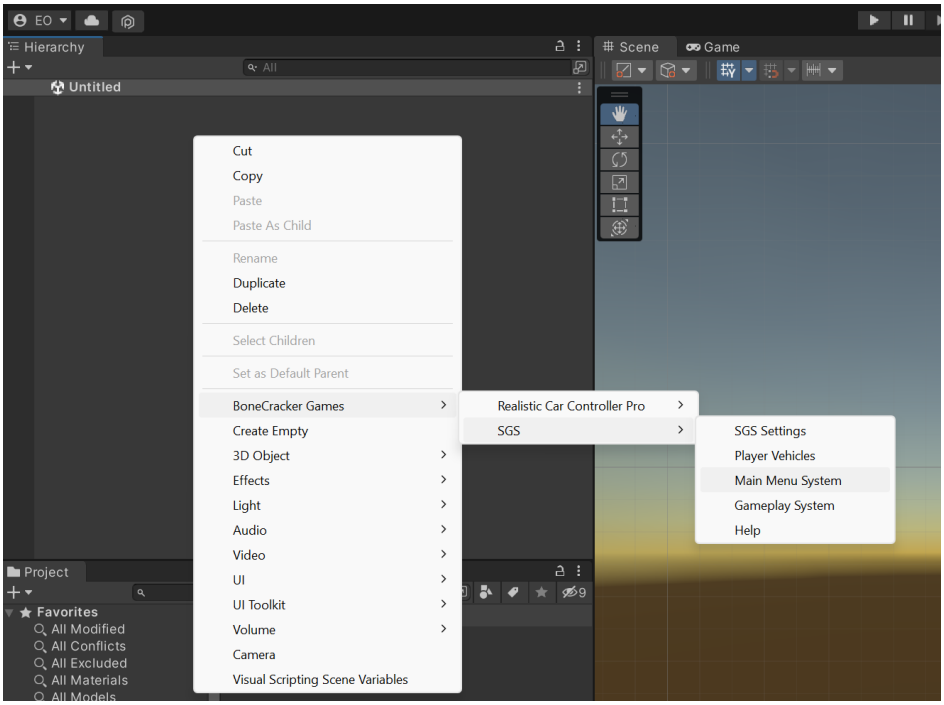
All demo scenes are in the scenes folder. Be sure all these scenes have been added to the build settings. To do that, open the **Build Settings** from **File → Build Settings**, and add the demo scenes by drag and drop. Otherwise, main menu can't load the target scene. When you're going to use your own scenes for the gameplay, be sure your Build Settings has them in the list.



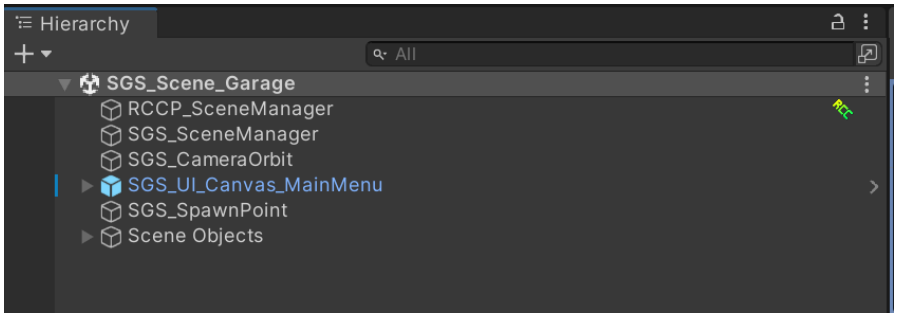
Scene selection UI buttons in the main menu are using "SelectScene", and "StartScene" methods in the SGS_SceneManager, and they have target scene indexes.

How to Create a New Main Menu Scene

Creating and using the system in a new scene is very simple. SGS has automated installation. Right click to the hierarchy panel and create the main menu system by **Tools → BCG → SGS → Create Main Menu System**. This will add all necessary systems to the scene. Of course, it will be using the default settings and resources. We'll edit and change them later. Once you create the system, play and test the system to ensure everything is working fine in this new scene.

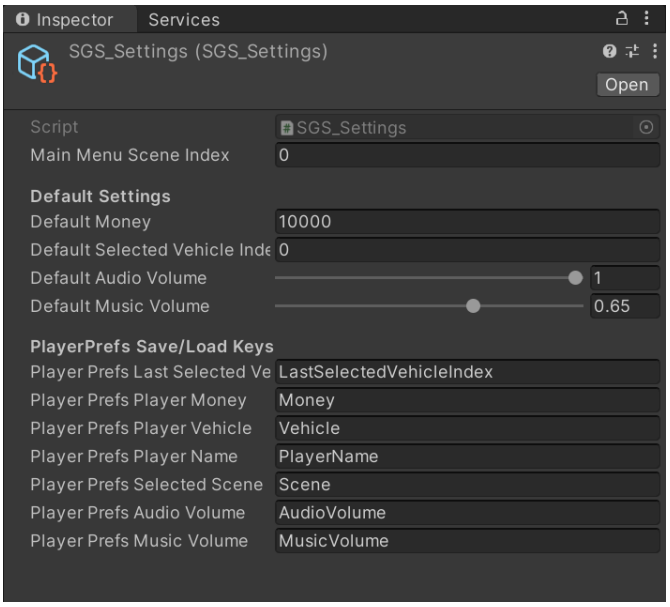


SGS Main Menu System requires these (All of them will be created and initialized automatically)



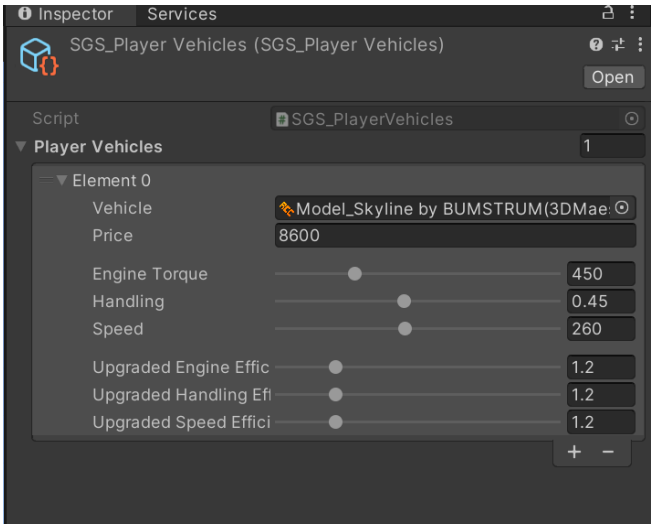
Settings and Resources (Scriptable objects)

As you well know, system is using the default settings and resources. But how to configure and manage them? What are the default settings and resources? How can I access them? How can I ask more questions? How can I... These are the scriptable objects which can be found in the resources folder of the SGS. Also, you can access them from **Tools → BCG → SGS**.



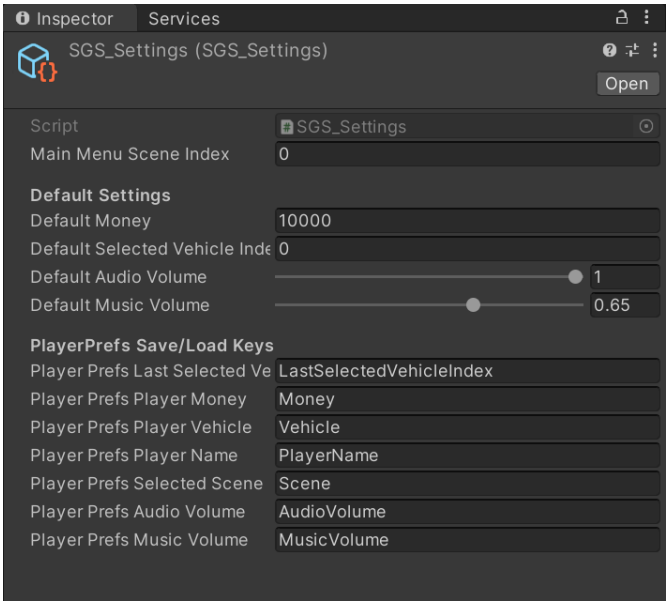
Player Vehicles (SGS_PlayerVehicles)

All selectable player vehicles have been stored in this scriptable object. You can add your own vehicles to this list as prefabs. Currently it's using the RCCP's demo vehicles. You'll be using your own vehicles. **SGS_SceneManager** will be using this to spawn and manage the player vehicles.



Settings (SGS_Settings)

All generic settings have been stored in this scriptable object. You can change initial and default values of some settings. Also, you can change the playerprefs keys as well.



These scriptable objects are very useful, because you don't need to edit the scripts one by one. Please don't change names and locations of the scriptable objects. Otherwise, instance of them won't work.

Price of the Vehicles

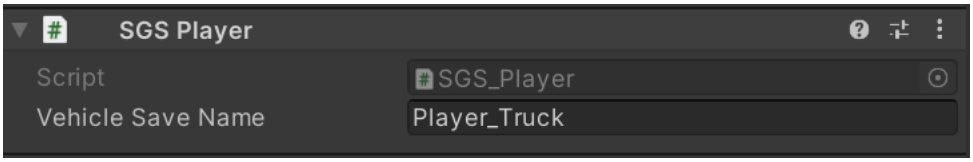
Price of the vehicles can be edited from the SGS_PlayerVehicles (**Tools → BCG → SGS → Player Vehicles**). 0 means it's unlocked by default.

Price of the Items and Upgrades

Each button has **SGS_UI_PurchaseItem** script. This script has a target item type, save string, and price. It checks the current vehicle save data if the vehicle owns the item or not. If it's owned, enables the upgrader script attached to the same button. You can directly change the price, name, and other settings of the corresponding item by selecting the UI button. For the upgrader buttons such as engine, brake, steering, and speed, they have **SGS_UI_PurchaseUpgrade** script instead of **SGS_UI_PurchaseItem**.

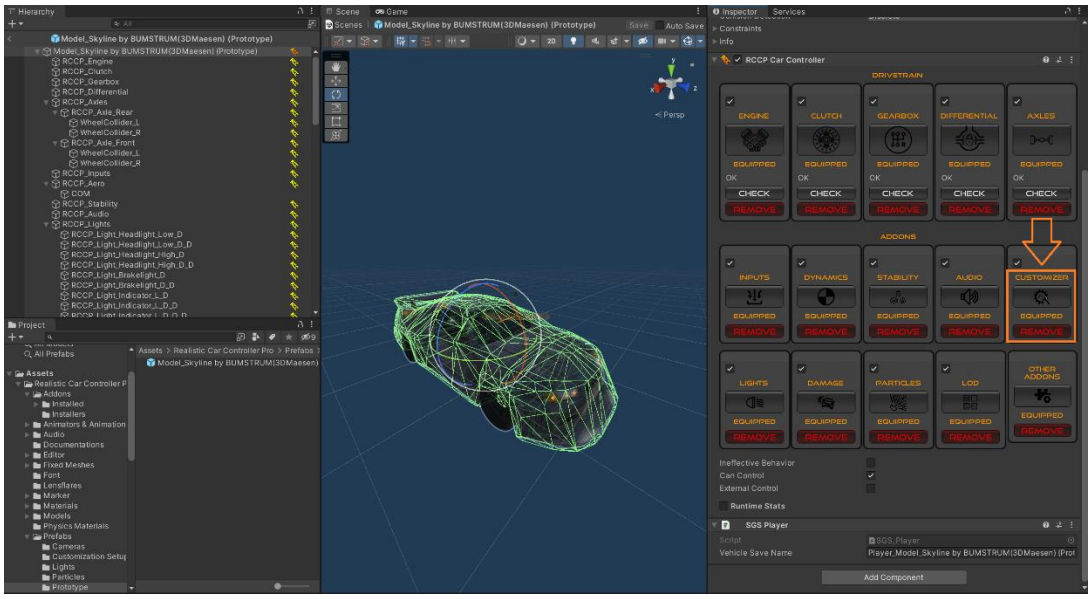
SGS_Player

All selectable player vehicles must have this component. SGS will check the vehicles in the SGS_PlayerVehicles and adds the SGS_Player script to them automatically. This script simply includes a save file name as a string. As you well know, all vehicles must have unique save names. This save name will be used with PlayerPrefs to lock, unlock, and select vehicles. You can use this script to collect points or coins in the game, because all vehicles will have this script.



Communicating with the Vehicle (RCCP_Customizer)

RCCP_Customizer will be used to modify the selected vehicle. **RCCP_Customizer** must be added to the vehicle in order to modify it, otherwise customization buttons will be disabled for the selected vehicle. Also, editor will inform you about this.



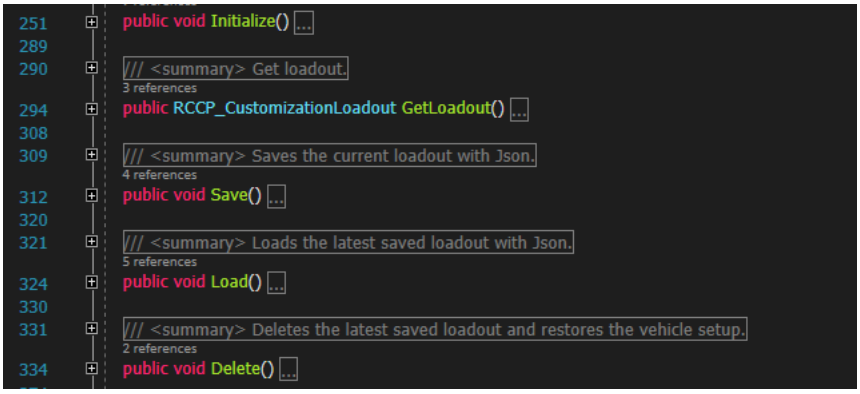
Once the player attempts to purchase or attach an item or upgrade, **RCCP_Customizer** on the vehicle will be used. This customizer has several types such as spoilers, neons, decals, upgrades

for engine, steering, and speed, wheels, mechanical configuration, paints, etc... To use all these features, vehicle's customizer component must have all of them. More info about the **RCCP_Customizer** will be explained is a separate documentation.

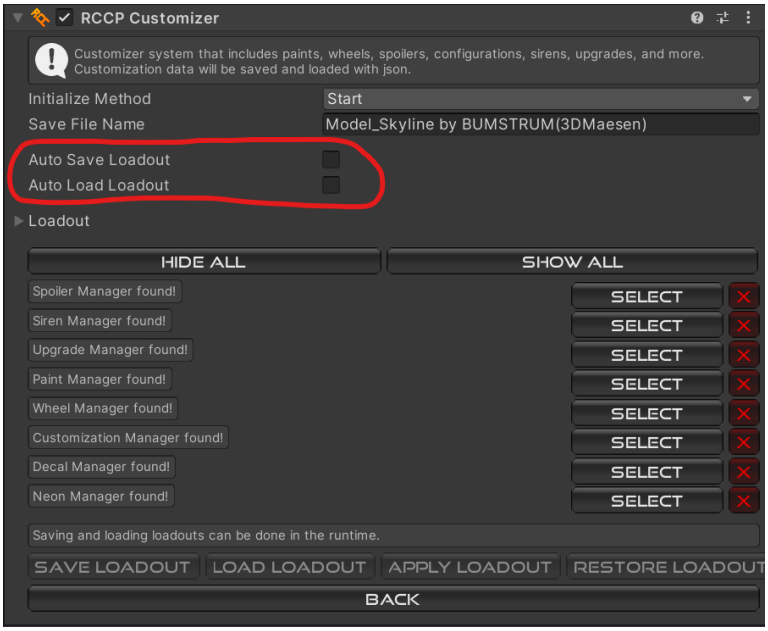
You can exclude or include only specific customizers for a vehicle. E.g. your vehicle model shouldn't use any spoiler, just don't use the spoiler modification on your vehicle. Spoiler UI button in the main menu canvas will be disabled for this vehicle. You're not restricted to use all of them.

Saving, Loading, Restoring, and Applying the Loadout

Saving, loading, restoring, and applying the customization loadout data have been managed by the **RCCP_Customizer** on the vehicle. It has a custom class named loadout. **RCCP_Customizer** has **Save**(), **Load**(), **Restore**(), and **Apply**() methods. Once the player purchases an item, it won't save directly. Once the player purchases the cart, it saves the latest loadout. If player cleans the cart, restores the latest loaded loadout and applies. **RCCP_Customizer** is responsible for saving and loading the customization loadout data. Each vehicle must have unique save name, be sure your vehicles have unique save names. You can check the save name directly from the **RCCP_Customizer** component.



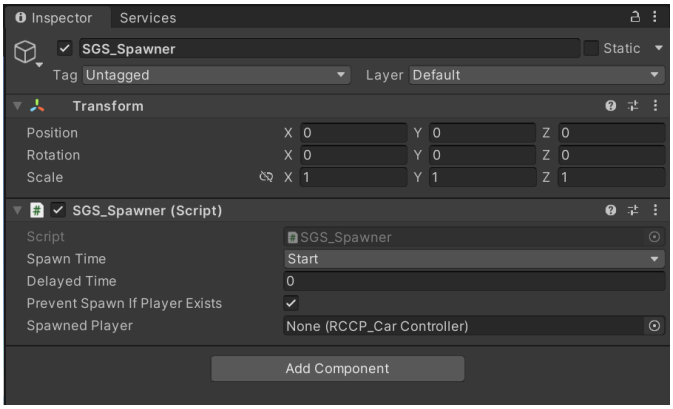
RCCP_Customizer has automatic save and load feature, but SGS is disabling the autosave feature because we don't want to own the items while previewing them. Saving method will be called by **SGS_SceneManager** only.



Gameplay Scenes

Gameplay scenes are just the same prototype scene as RCCP's. These gameplay scenes represent unique levels, so when the player selects a scene in the main menu, corresponding scene will be loaded and started. Currently project has three gameplay scenes. Each scene has an index target in the build settings. To find the indexes, open the **Build Settings** from **File → Build Settings**. You'll be able to see the included scenes in the build. As you can see, all scenes have unique indexes. When the player selects a scene, target index value will be used. Therefore, you must use the proper index value on scene selection buttons in the main menu. Each scene selection button is using a method named "**SelectScene**" in the **SGS_SceneManager**. This is where the player's scene selection is saved.

Gameplay scenes have **SGS_Spawner** to spawn the latest selected vehicle. Vehicles will be spawned at this position.



Loading the Target Scene

Each scene selection button is using a method named "**SelectScene**" in the **SGS_SceneManager**. This method only saves the selected scene, not loads it. There is another method named "**StartScene**" in the **SGS_SceneManager**. This method starts the selected scene. From this point, new level will be loaded, and the main menu scene will be unloaded.

Vehicle Selection and Purchasing

Vehicle selection UI panel has a few buttons interacting with **SGS_SceneManager**. These are next, previous, select, and purchase buttons. Next button's method is "**NextVehicle()**", previous is "**PreviousVehicle()**", select is "**SelectVehicle()**" and purchase is "**PurchaseVehicle()**". Select button will be deactivated if the current vehicle is not owned by the player. In this case, purchase button will be enabled, and text of the price will be updated.

UI Panels

Main menu UI canvas has several UI panels such as vehicle selection, scene selection, customization, settings, main, etc. When enabling an UI panel in the canvas, "**OpenPanel()**" method of the **SGS_SceneManager** will be used. This method disables all other panels and enables only the target panel. You can add your own UI panels to the **SGS_SceneManager**. When you're going to enable your own UI panel with a button, be sure this button is using the method in the **SGS_SceneManager**. Be sure to select the target UI panel as well.

UI Title Text

Main menu UI canvas has a title text at left top corner. This text will be updated accordingly when the player switches to the other panels. When the player pushes a button such as vehicles, or customization, it will use "**OpenPanel()**" method in the **SGS_SceneManager**. But also using "**SetPanelTitleText()**" method in the **SGS_SceneManager**. You can change the title text directly on your button.

Testing Panel

There is a small and useful testing panel in the main menu UI canvas. You can use it for testing purposes such as adding money, unlocking the vehicles, and resetting the game. But be sure to disable or delete the panel from the UI canvas before releasing the game. It's located in the Menu_Main gameobject.

Displaying and Updating the Texts (TextMeshPro)

All dynamic texts such as money, price, items, cars, etc. will be controlled by the **SGS_SceneManager**. They are not just regular text components, they are **TextMeshPro's** texts. Editor should ask you to import it when you open the demo scene. If you don't import it to the project, you'll not be able to see the texts.

All dynamic texts can be found here. You can check the **SGS_SceneManager** to inspect how the script is managing them at the runtime.

Using the SGS_SceneManager in Your Own Scripts

SGS_SceneManager is a singleton class. You can directly access to the instance by;

SGS_SceneManager.Instance;

All methods have detailed comments in the script. These methods have been listed below. Methods in the image may be changed in later versions, so please check the script.

```
194
195     /// <summary>
196     /// Spawns all selectable player vehicles once.
197     /// </summary>
        1 reference
198     private void SpawnAllPlayerVehicles() ...
247
248     /// <summary>
249     /// Gets the latest selected vehicle as int index.
250     /// </summary>
251     /// <returns></returns>
        1 reference
252     public int GetVehicleIndex() ...
257
258     /// <summary>
259     /// Saves the current vehicle as selected player vehicle.
260     /// </summary>
        0 references
261     public void SelectVehicle() ...
266
267     /// <summary>
268     /// Enables next vehicle and disables all other ones.
269     /// </summary>
        0 references
270     public void NextVehicle() ...
280
281     /// <summary>
282     /// Enables previous vehicle and disables all other ones.
283     /// </summary>
        0 references
284     public void PreviousVehicle() ...
294
295     /// <summary>
296     /// Enables the current vehicle and disables all other ones.
297     /// </summary>
        4 references
298     public void EnableVehicle() ...
322
323     /// <summary>
324     /// Purchases the current vehicle.
325     /// </summary>
        0 references
326     public void PurchaseVehicle() ...
353
354     /// <summary>
355     /// Selects the target scene.
356     /// </summary>
357     /// <param name="sceneIndex"></param>
        0 references
358     public void SelectScene(int sceneIndex) ...
363
364     /// <summary>
365     /// Starts the target scene.
366     /// </summary>
        0 references
367     public void StartScene() ...
374
        1 reference
375     public void OpenPanel(GameObject activePanel) ...
384
385     /// <summary>
386     /// Checks upgradable item purchased or not. If not purchased, add to the cart, remove otherwise.
387     /// </summary>
388     /// <param name="newItem"></param>
        1 reference
389     public void CheckUpgradePurchased(SGS_CartItem newItem) ...
404
```

```
404
405     /// <summary>
406     /// Checks purchasable item. If not purchased, add to the cart, remove otherwise.
407     /// </summary>
408     /// <param name="newItem"></param>
        1 reference
409     public void CheckItemPurchased(SGS_CartItem newItem) ...
417
418     /// <summary>
419     /// Adds a new item to the cart. Cart can't have items with same type.
420     /// </summary>
421     /// <param name="newItem"></param>
        2 references
422     public void AddItemToCart(SGS_CartItem newItem) ...
439
440     /// <summary>
441     /// Removes an item from the cart. Cart can't have items with same type.
442     /// </summary>
443     /// <param name="newItem"></param>
        2 references
444     public void RemoveItemFromCart(SGS_CartItem newItem) ...
461
462     /// <summary>
463     /// Clears the cart and restores the player vehicle back to the last loadout.
464     /// </summary>
        0 references
465     public void ClearCart() ...
485
486     /// <summary>
487     /// Purchases all items in the cart and saves the player vehicle loadout..
488     /// </summary>
        0 references
489     public void PurchaseCart() ...
540
541     /// <summary>
542     /// Updates all items in the cart list.
543     /// </summary>
        1 reference
544     public void UpdateCartItemsList() ...
590
591     /// <summary>
592     /// Saves the current loadout.
593     /// </summary>
        2 references
594     public void SaveCustomization() ...
604
605     /// <summary>
606     /// Loads the latest loadout.
607     /// </summary>
        1 reference
608     public void LoadCustomization() ...
616
617     /// <summary>
618     /// Applies the loaded loadout.
619     /// </summary>
        1 reference
620     public void ApplyCustomization() ...
628
629     /// <summary>
630     /// Adding money for testing purposes.
631     /// </summary>
        0 references
632     public void Testing_AddMoney() ...
637
638     /// <summary>
639     /// Unlocking all vehicles for testing purposes.
640     /// </summary>
        0 references
641     public void Testing_UnlockAllCars() ...
646
647     /// <summary>
648     /// Deletes the save data and restarts the game for testing purposes.
649     /// </summary>
        0 references
650     public void Testing_ResetSave() ...
```

Examples;

```
// Selects the current vehicle on the scene.
SGS_SceneManager.Instance.SelectVehicle();

// Attempts to purchase the current vehicle on the scene.
SGS_SceneManager.Instance.PurchaseVehicle();

// Attempts to purchase all items in the cart.
SGS_SceneManager.Instance.PurchaseCart();
```

SGS.cs

SGS class has ben used by the **SGS_SceneManager** and other related scripts as well. SGS is not attached to the any gameobject in the scene, you can use any method in the SGS directly, because this class has static methods. All static methods of the SGS have been listed below. Methods in the image may be changed in later versions, so please check the script.

```
/// <summary>
/// General API class.
/// </summary>
19 references
public class SGS {

    /// <summary>
    /// Gets the money.
    /// </summary>
    /// <returns></returns>
    4 references
    public static int GetMoney() ...

    /// <summary>
    /// Changes the player money. It can be positive or negative.
    /// </summary>
    /// <param name="amount"></param>
    3 references
    public static void ChangeMoney(int amount) ...

    /// <summary>
    /// Gets the latest selected player vehicle.
    /// </summary>
    /// <returns></returns>
    1 reference
    public static int GetVehicle() ...

    /// <summary>
    /// Sets the selected vehicle as player vehicle.
    /// </summary>
    /// <param name="vehicleIndex"></param>
    1 reference
    public static void SetVehicle(int vehicleIndex) ...

    /// <summary>
    ///  Unlocks the target vehicle.
    /// </summary>
    /// <param name="vehicleIndex"></param>
    4 references
    public static void UnlockVehicle(int vehicleIndex) ...

    /// <summary>
    /// Deleting save key for the target vehicle and locks it.
    /// </summary>
    /// <param name="vehicleIndex"></param>
    1 reference
    public static void LockVehicle(int vehicleIndex) ...

    /// <summary>
    /// Purchases and unlocks all vehicles.
    /// </summary>
    /// <param name="vehicleIndex"></param>
    1 reference
    public static void UnlockAllVehicles() ...

    /// <summary>
    /// Deleting save key for all vehicles and locking them.
    /// </summary>
    /// <param name="vehicleIndex"></param>
    0 references
    public static void LockAllVehicles() ...

    /// <summary>
    /// Is this vehicle owned by the player?
    /// </summary>
    /// <param name="vehicleIndex"></param>
    /// <returns></returns>
    1 reference
    public static bool IsOwnedVehicle(int vehicleIndex) ...
```

```
124
125     /// <summary>
126     /// Sets the target scene to load.
127     /// </summary>
128     /// <param name="sceneIndex"></param>
        1 reference
129     public static void SetScene(int sceneIndex) ...
134
135     /// <summary>
136     /// Gets the latest selected scene.
137     /// </summary>
138     /// <returns></returns>
        1 reference
139     public static int GetScene() ...
144
145     /// <summary>
146     /// Loads the latest selected scene.
147     /// </summary>
        1 reference
148     public static void StartGameplayScene() ...
153
154     /// <summary>
155     /// Restart the game.
156     /// </summary>
        0 references
157     public static void RestartGame() ...
162
163     /// <summary>
164     /// Back to the main menu.
165     /// </summary>
        0 references
166     public static void MainMenu() ...
171
172     /// <summary>
173     /// Sets the volume of the audiolistener.
174     /// </summary>
175     /// <param name="volume"></param>
        1 reference
176     public static void SetAudioVolume(float volume) ...
182
183     /// <summary>
184     /// Sets the music volume.
185     /// </summary>
186     /// <param name="volume"></param>
        1 reference
187     public static void SetMusicVolume(float volume) ...
192
193     /// <summary>
194     /// Get volume of the audiolistener.
195     /// </summary>
196     /// <returns></returns>
        1 reference
197     public static float GetAudioVolume() ...
202
203     /// <summary>
204     /// Get volume of the music.
205     /// </summary>
206     /// <returns></returns>
        1 reference
207     public static float GetMusicVolume() ...
212
213     /// <summary>
214     /// Resets the game by deleting the save data and reloading the scene again.
215     /// </summary>
        1 reference
216     public static void ResetGame() ...
222
223 }
224
```

Examples;

// Returns the money as int.

SGS.GetMoney();

// Changes the player money with given amount.

SGS.ChangeMoney(int changeAmount);

// Restarts the game.

SGS.RestartGame();

UI Button Animations

UI buttons have scripted animations such as highlighting the text color and sliding effect. Check the UI buttons in the main menu, they have **SGS_UI_TextHighlighter** and **SGS_UI_SliderHighlighter** scripts. First script is changing the color of the text when hover and click. Second script is animating an image in the UI button from left to right when hover. You don't need to use them on your UI buttons, but if you want to use, they're easy. **SGS_UI_TextHighlighter** script needs a target textmesh text, and **SGS_UI_SliderHighlighter** script needs an image to move from left to right. That's it. Also, there is another script for popping the text scale when it changes. **SGS_UI_TextPopperOnTextChange** script can be used for this. It needs a target textmesh text, and that's it.

Contact

Please include your invoice number while contacting me. I usually respond within a day. I may not respond on the weekend.

Email: bonecrackergames@gmail.com