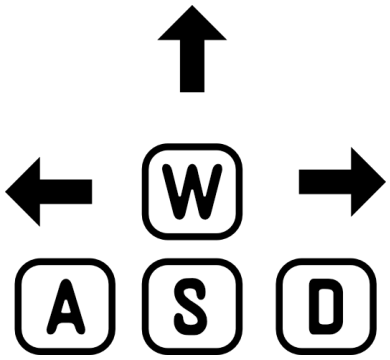
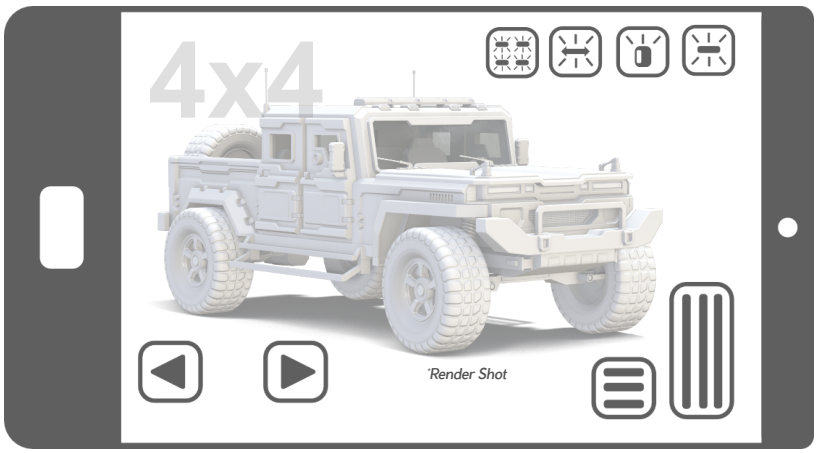
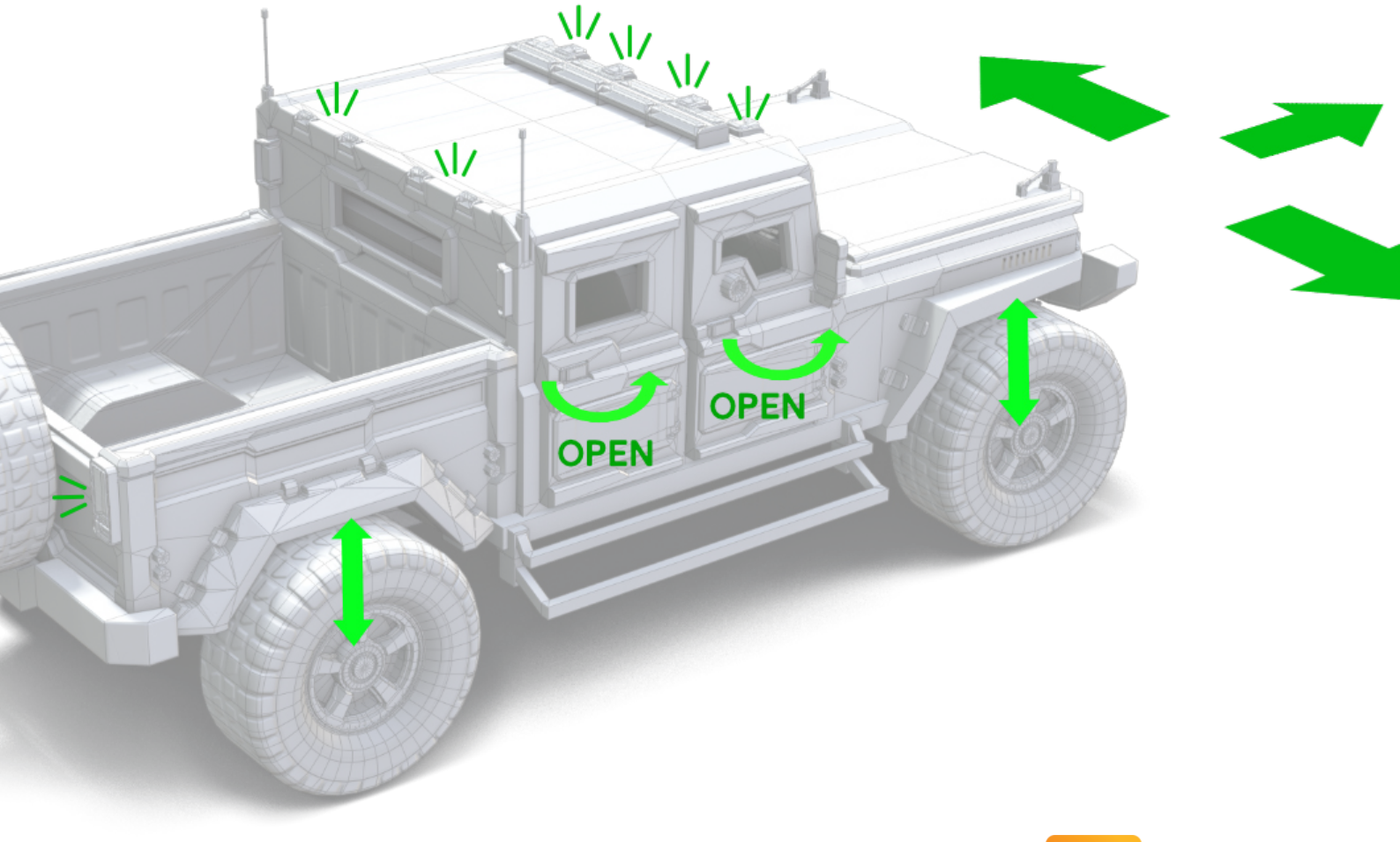
JRS VEHICLE PHYSICS

CONTROLLER

With Keyboard and On-screen Mobile Inputs







*Render Shot

JERMESA STUDIO

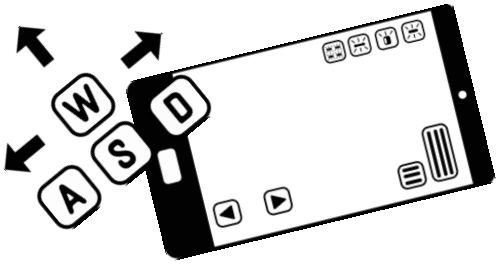
Table of Contents

JRS Vehicle Physics Controller

Introduction:

The JRS C# scripts is a powerful tool that allows you to create realistic and responsive vehicle behavior in your game or simulation. These script handles various aspects of vehicle physics, including steering, acceleration, braking, vehicle light system and even dust particle effects. In this guide, we'll explore the different parameters available in the Inspector of each script and how to use them to customize your vehicle's behavior.

This guide include a step-by-step explanation of the JRS C# scripts for the Vehicle, including the parameters in the Inspector and how to use them effectively.



Getting Started:

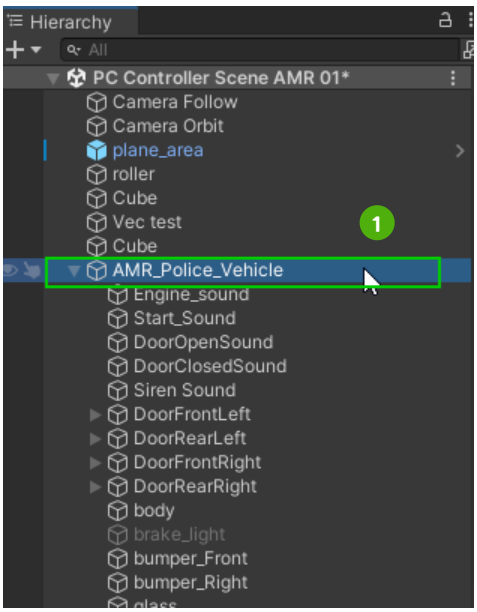
1

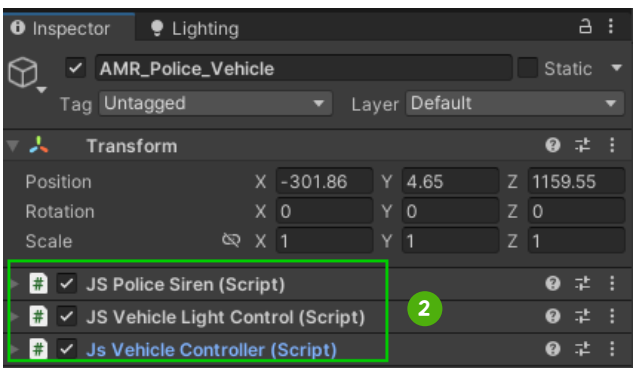
Attach the Scripts to Your Vehicle:

Ensure that the required scripts is attached to the GameObject representing your parent vehicle object in the Unity scene.

Here I have 'AMR_Police_Vehicle' as the parent object of my vehicle and lets attach the following three script in the inspector panel:

- JS Police Siren (Script)
- JS Vehicle Light Control (Script)
- Js Vehicle Controller (Script)





2 Adding Rigidbody and Box Collider:

- Under the Inspector panel of your parent object (AMR_Police_Vehicle) click Add Component Select Rigidbody with the following parameters

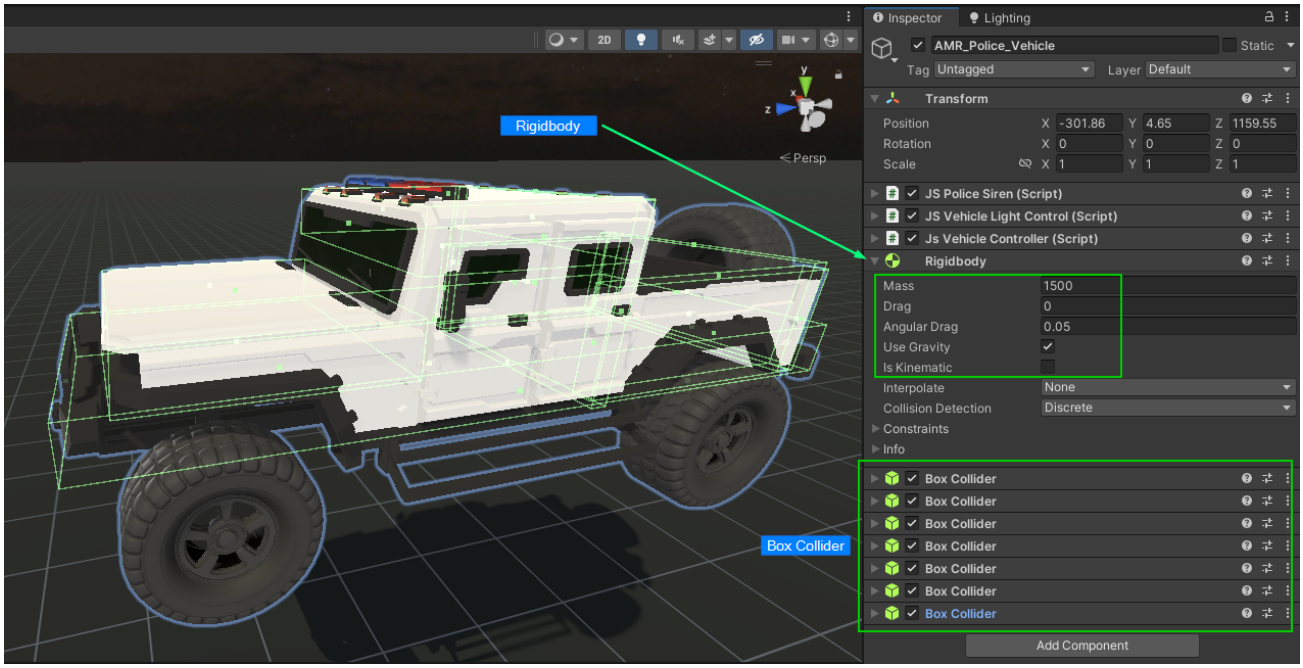
  Mass = 1500

  Drag = 0

  Angular Drag = 0.05

  Use Gravity = Enable

- Next we need to create Box Collider for our Vehicle. We can add as many Box Colliders as we needed to different parts of the vehicle. You can check out the sample files included in this pack.



UNDERSTANDING THE PARAMETERS

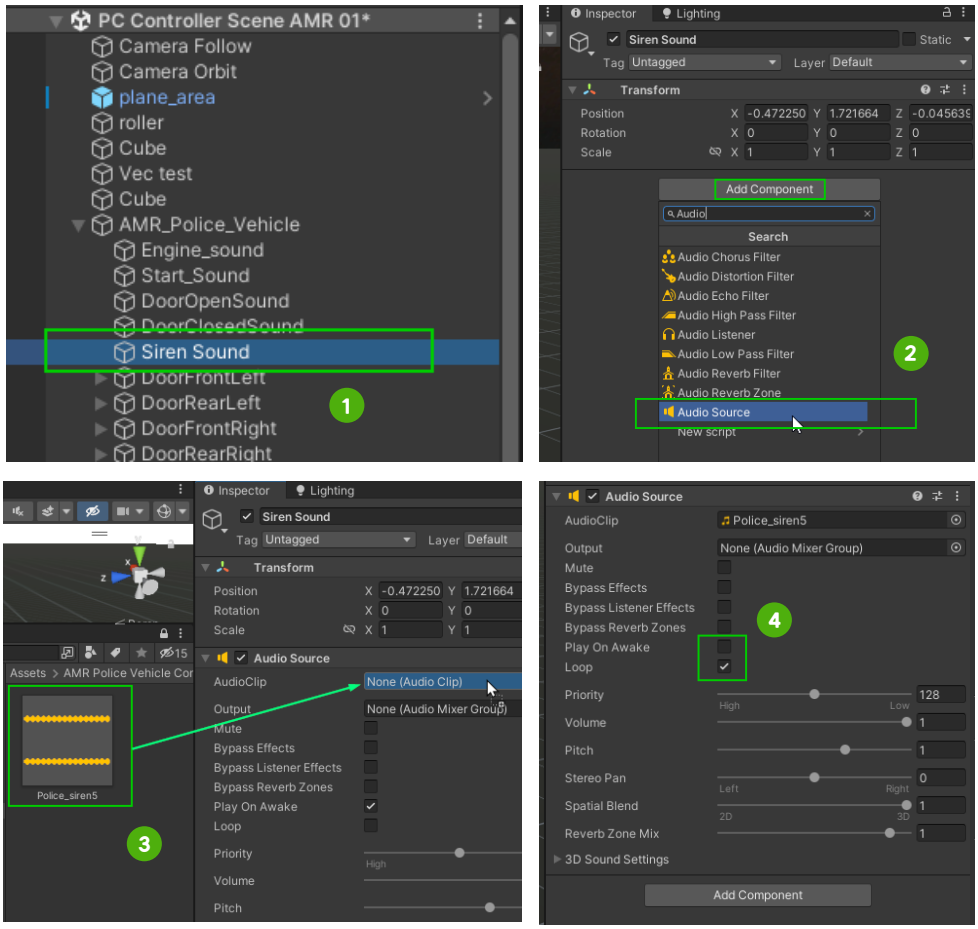
In every script, you'll find various parameters that control your vehicle's behavior. Let's understand them one by one.

JS Police Siren (Script)

Siren Sound - Drag and drop an Audio Source component here to be used as the police siren sound.

How to make an Audio Source Component?

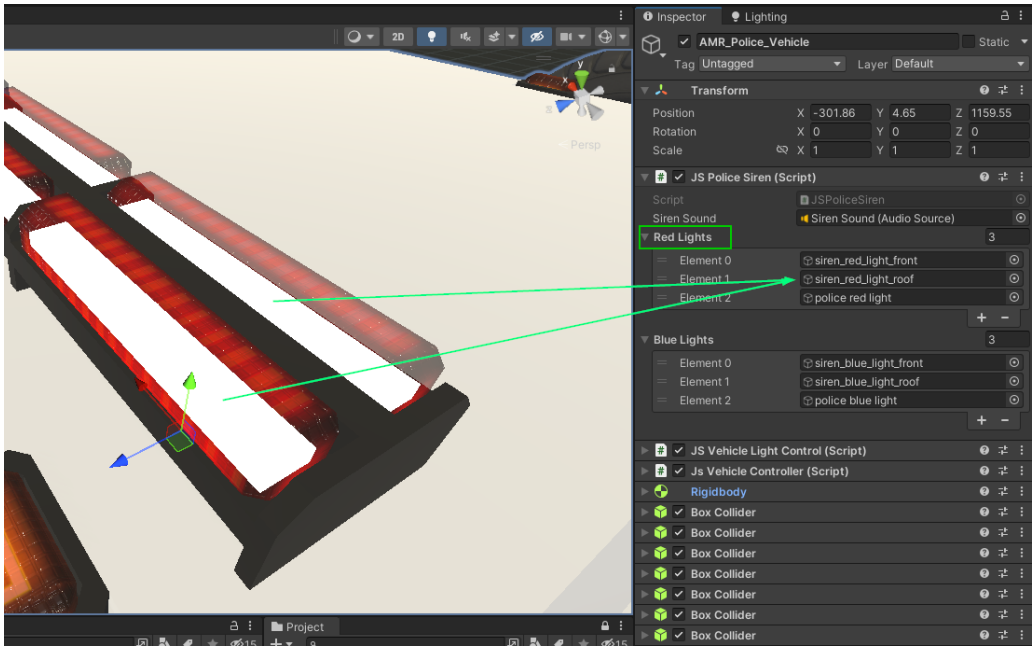
- First, create an *Empty GameObject* by clicking on *GameObject > Create Empty* and rename it to *"Siren Sound."*
- In the *Inspector panel*, click on *Add Component* and select *Audio Source*. Now, drag and drop your sound clip under *AudioClip* parameter.
- Finally, assign this *"Siren Sound"* GameObject to the *JS Police Siren (Script)* in the Inspector panel under *Siren Sound* parameters.



Red Lights - These parameters allow you to assign game objects or lights that represent the red lights of the police siren that will flicker. You can add as many objects or lights as you wish by clicking the '+' icon.

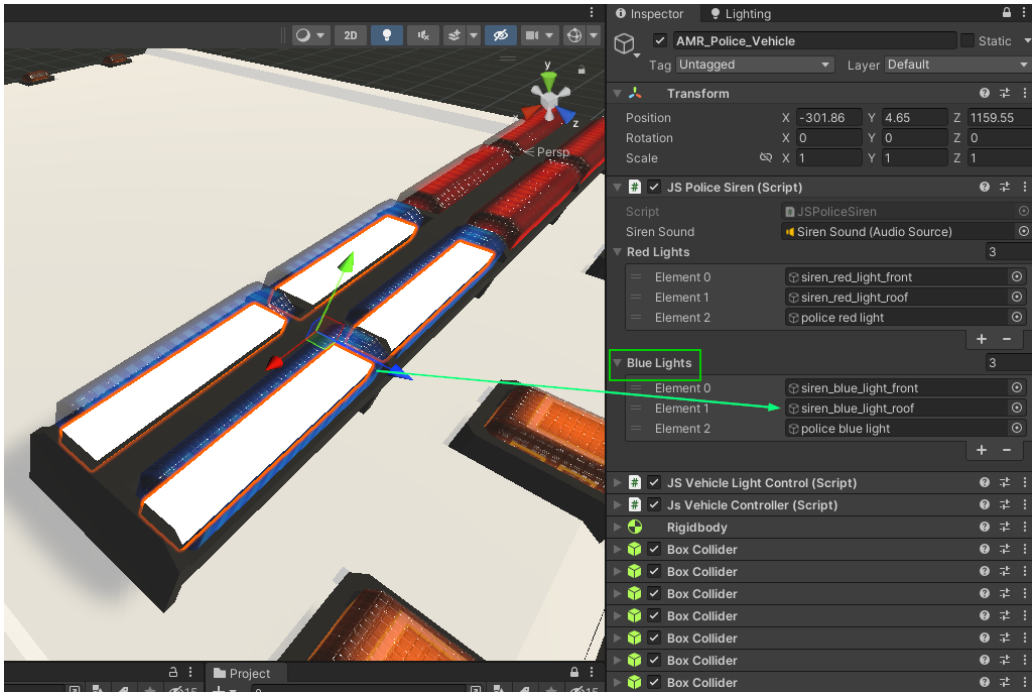
For instance I have a glow emissive object and light for the red signal, I can assign them in this Parameters so that they will activete when I switch on the Police siren.

> Remember to disable any lights glowing object in the inspector panel so that they will only be activated when we press on the desired Key.



Blue Lights - These parameters allow you to assign game objects or lights that represent the blue lights of the police siren that will flicker. You can add as many objects or lights as you wish by clicking the '+' icon.

Same here also have a glow emissive object and light for the blue signal, I can assign them in this Parameters so that they will activete when I switch on the Police siren.



Input Key

By default you can press *'P'* on the keyboard to toggle *ON/OFF* police siren. You can also customize the key by editing the *KeyCode* inside the *Update()* function in the *JSPoliceSiren.cs Script.* For Mobile input you can check out how to set-up *On-screen buttons.*

```
private void Update()
    {
        if (mobileInputController == null)
        {
            mobileInputController = FindObjectOfType<JSInputController>();
        }
        if (Input.GetKeyDown(KeyCode.P) || mobileInputController.sirenButton.IsButtonClicked())
        {
            isSirenOn = !isSirenOn;
            ToggleSirenLights();
            ToggleLightsVisibility(redLights, isSirenOn);
            ToggleLightsVisibility(blueLights, isSirenOn);
        }
    }
```

JS Vehicle Light Control (Script)

> In the *Inspector* window of your parent object *(AMR_Police_Vehicle)* and under *JS Vehicle Light Control (Script)*, you will see several parameters that you can use to control your vehicle lights

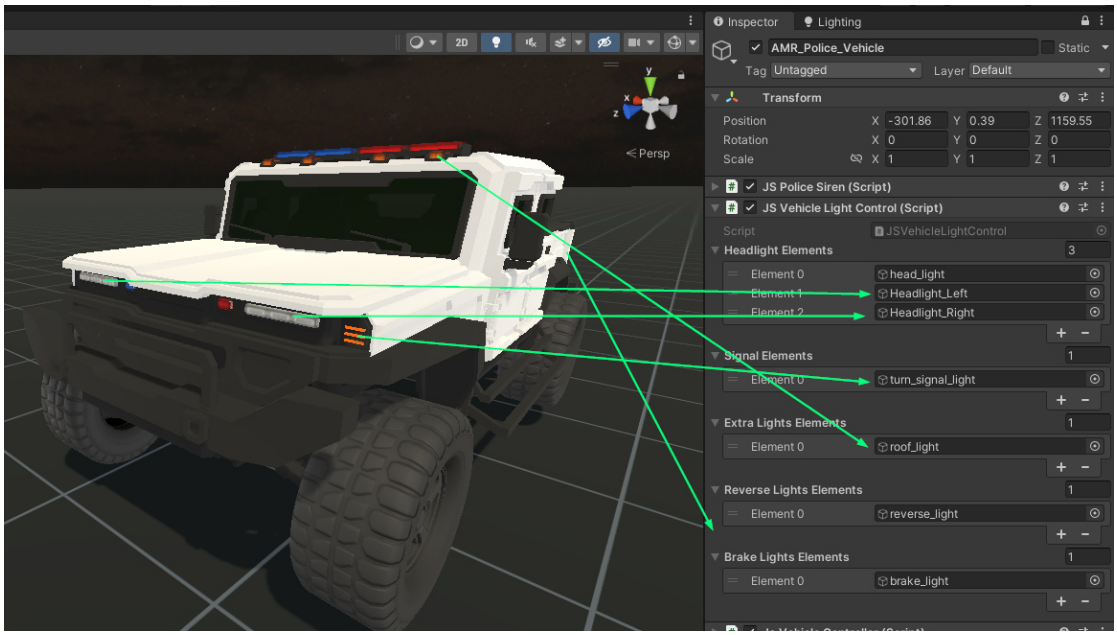
Headlight Elements - Here you can assign GameObjects/Lights representing the headlights of the vehicle.

Signal Elements - Here you can assign GameObjects/Lights representing the Signal lights of the vehicle.

Extra Lights Elements - Here you can assign GameObjects/Lights representing any extra lights you might have in your of vehicle.

Reverse Lights Elements - Here you can assign GameObjects/Lights representing the reverse lights of the vehicle.

Brake Lights Elements - Here you can assign GameObjects/Lights representing the brake lights of the vehicle.



Input Key

By default you can press the following key on the keyboard to toggle *ON/OFF* lights for your vehicle.

H - Toggle the headlights ON/OFF.

T - Toggle the signal lights ON/OFF.

E - Toggle the extra lights ON/OFF.

S - Reverse lights will be activated when you reverse your vehicle.

Spacebar - Brake lights will be activated when you apply brake to your vehicle.

You can also customize the key by editing the *KeyCode* inside the *Update()* function in the *JSVehicleLightControl.cs Script*. For Mobile input you can check out how to set-up *On-screen buttons*.

```
void Update()
    {
        if (mobileInputController == null)
        {
            mobileInputController = FindObjectOfType<JSInputController>();
        }


        if (Input.GetKeyDown(KeyCode.H) || mobileInputController.headLightsButton.IsButtonClicked())
        {
            lightsOn = !lightsOn;
            ToggleLights();
        }

        if (Input.GetKeyDown(KeyCode.T) || mobileInputController.signalLightsButton.IsButtonClicked())
        {
            signalOn = !signalOn;
            ToggleSignal();
        }

        if (Input.GetKeyDown(KeyCode.E) || mobileInputController.extraLightsButton.IsButtonClicked())
        {
            extraLightsOn = !extraLightsOn;
            ToggleExtraLights();
        }

        if (Input.GetKey(KeyCode.S))
        {
            ToggleReverseLights(true);
        }

        if (Input.GetKeyUp(KeyCode.S))
        {
            ToggleReverseLights(false);
        }


        if (Input.GetKey(KeyCode.Space))
        {
            ToggleBrakeLights(true);
        }

        if (Input.GetKeyUp(KeyCode.Space))
        {
            ToggleBrakeLights(false);
        }

    }
```

JS Vehicle Controller (Script)

> This is a basic step-by-step guide on how to use the "Js Vehicle Controller" script for Unity:

Understanding the Variables:

> In the ***Inspector*** window of your parent object ***(AMR_Police_Vehicle)*** and under ***JS Vehicle Controller (Script)***, you will see several parameters that you can use to control your vehicle lights

Motor Force:
This is the force that your vehicle's engine will produce. Larger numbers mean a more powerful engine. Normally for an SUV vehicle I would set it between 400 - 500.

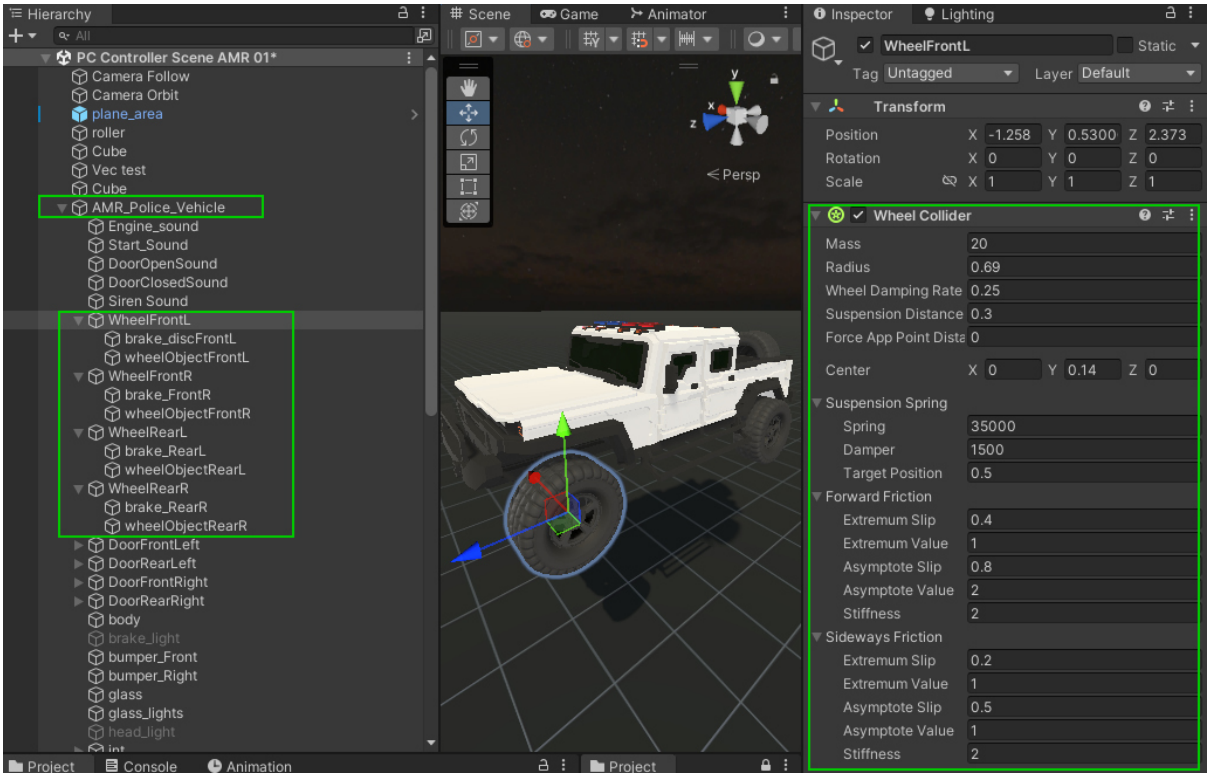
Max Steer Angle:
This variable determines how sharply your vehicle can turn. High values will allow for sharp turns, low values for gradual ones. For Normal four wheel vehicle we can set this value = 30

Wheel Collider:

These variables represent the wheels of the vehicle. You will need to assign these by dragging and dropping the corresponding wheel collider objects from your Hierarchy into these fields in the inspector.

- **Front Left Wheel** : Assign Front Left Wheel (Collider)
- **Front Right Wheel** : Assign Front Right Wheel (Collider)
- **Rear Left Wheel** : Assign Rear Left Wheel (Collider)
- **Rear Right Wheel** : Assign Rear Right Wheel (Collider)

> ### How to Create Wheel Collider?
>
> - On Unity click on **GameObject** > **Create Empty**
> - Rename your **GameObject** as **WheelFrontL (for wheel front left)**
> - Now with **WheelFrontL** selected click on **Add Component** in the **Inspector** panel, and select **Wheel Collider**.
> - Repeat the above steps to create all your Wheel Colliders. In the Hierarchy panel, add your corresponding wheel/brake object as a child under each Wheel Collider.



Wheel Collider Components:

Mass - This determine the mass of the Wheel (usually we set it at 20)

Radius - You need to adjust this value to fit the radius of your wheel

Wheel Damping Rate - Represents the damping of the wheel. This could be seen as an additional friction force. A high damping value will result in a slow wheel that needs more force to spin.

Suspension Distance - The suspension distance is the maximum possible compression of the spring in meters. Raise this to soften the suspension.

Force App Point Distance - This determines where the physics engine applies the forces of propulsion and braking to the wheel.

Center - A Vector3 representing the position of the wheel relative to the object's center of mass. It's usually fine to leave these values at 0 if the collider is centered on the object.

Suspension Spring:

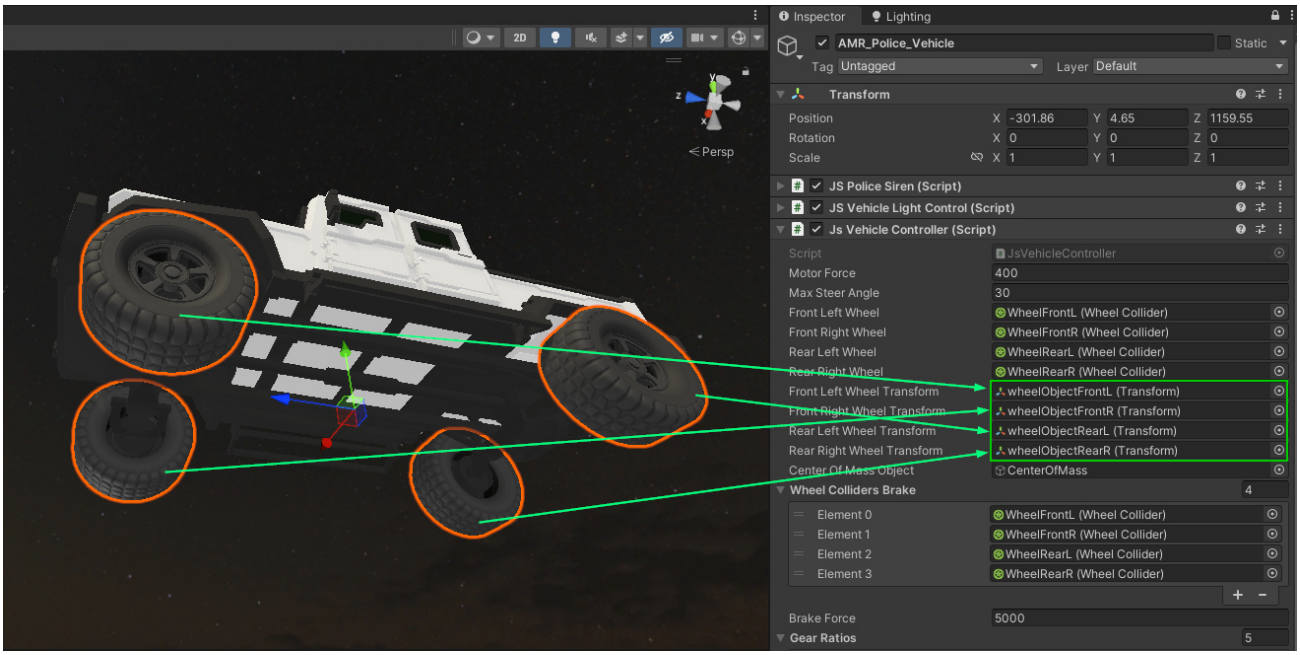
- Spring: The strength of the spring. Increase this to make the body stronger.
- Damper: The rate at which the spring returns to its neutral state. Increase this to make the spring return slower.
- Target Position: The rest state of the spring. Set at 0.5 will attempt to rest the spring halfway inside its limits.

Forward Friction / Sideways Friction (Stiffness) - Stiffness of the tire friction. High values make the tires grip the road, while low values make them slip.

Wheel Transform:

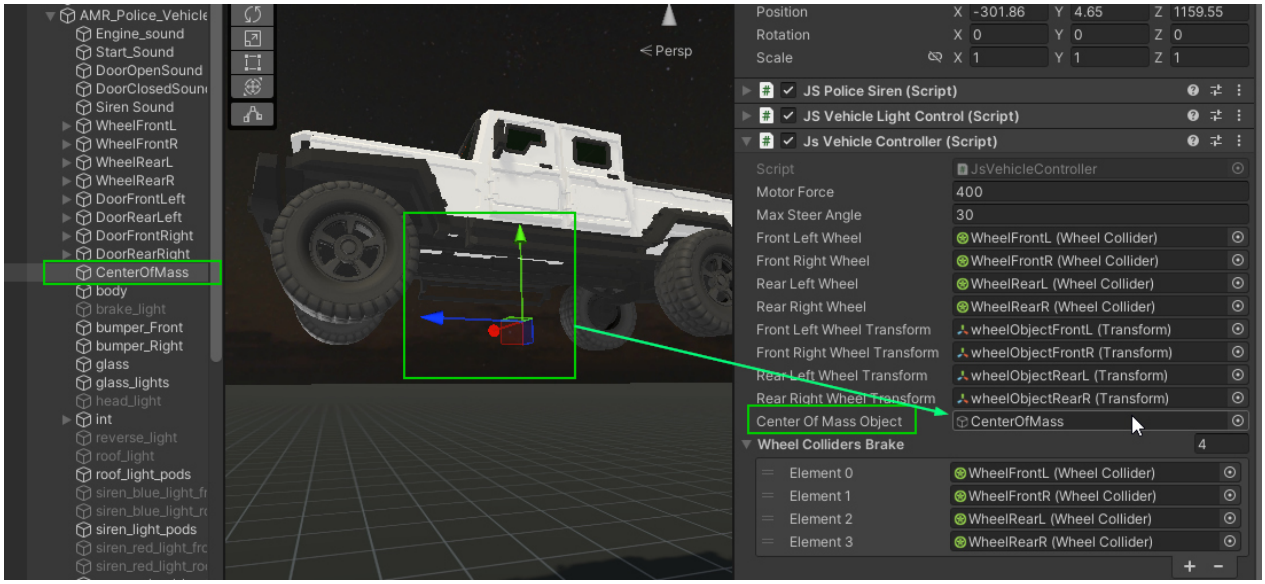
These components are references to the Transform components of the respective wheel game objects. They are used to update the visual position and rotation of the wheels based on the physics simulation.

- **Front Left Wheel Transform** : Assign Front Left Wheel (game object)
- **Front Right Wheel Transform** : Assign Front Right Wheel (game object)
- **Rear Left Wheel Transform** : Assign Rear Left Wheel (game object)
- **Rear Right Wheel Transform** : Assign Rear Right Wheel (game object)



Center Of Mass Object (GameObject):

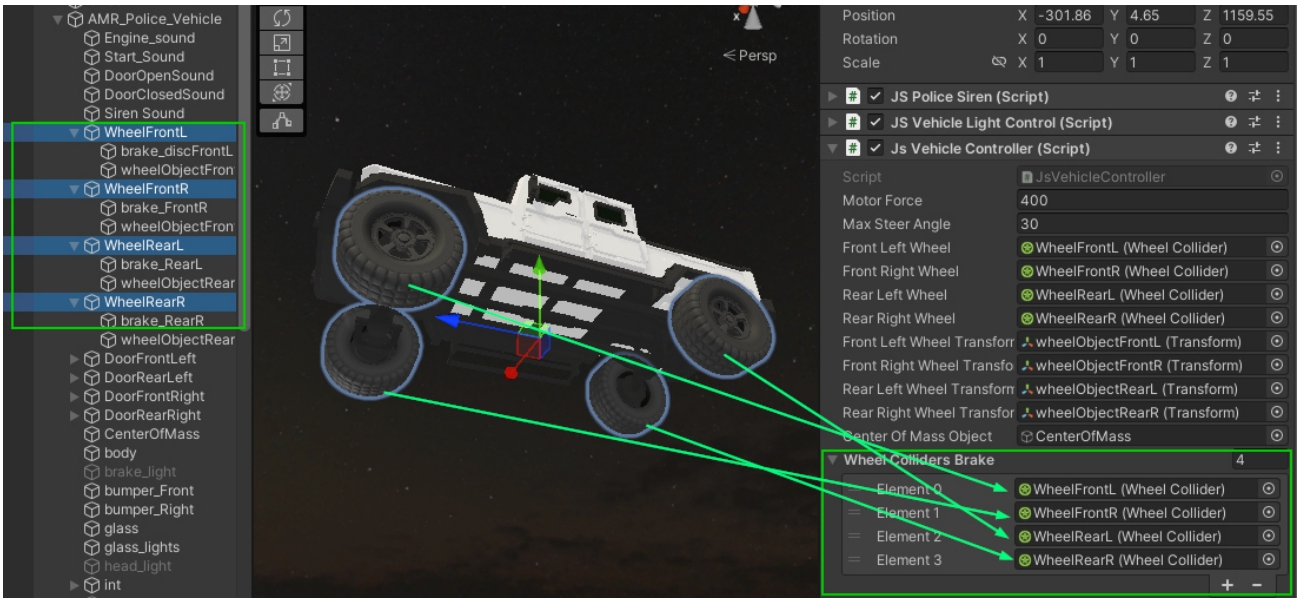
This represents the center of mass of your vehicle's stability. It should be an empty game object placed under your main vehicle (e.g., AMR Police Vehicle) in the Hierarchy. You can simply move it around in the editor to find the point that is the center of your vehicle.



Wheel Colliders Brake:

This represents the wheel colliders that will have braking force applied to them. You can click the '+' icon to add as many as you want. Remember, any wheel collider you add here—whether front wheel, rear wheel, or both—will have brake force applied when you brake your vehicle.

If you want the brake to be applied only to your front wheels, you can do this by adding only their wheel colliders to these parameters.



Brake Force:

This determines how powerful your brakes are. You can play around with the value I usually set this to 5000.

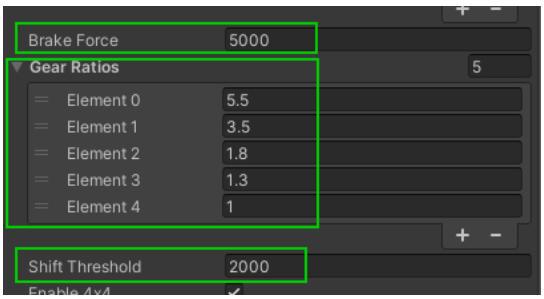
Gear Ratios:

These are the ratios that determine how much torque is transferred to your wheels at each gear. Gear ratios control the power and speed of the car.

Gear ratios determine the relationship between the engine RPM and the wheel speed. You can add how many gears you want and adjust these values to control the vehicle's acceleration and top speed in each gear.

Shift Threshold:

The RPM value at which the car will shift to the next gear. When the current RPM exceeds this threshold, the script will automatically shift to the next gear. I usually set this to 2000 if I have 5 gears.



Enable 4x4:

When enabled, this feature activates the vehicle's 4-wheel drive. When disabled, it reverts to 2-wheel drive.

Dust Particle System:

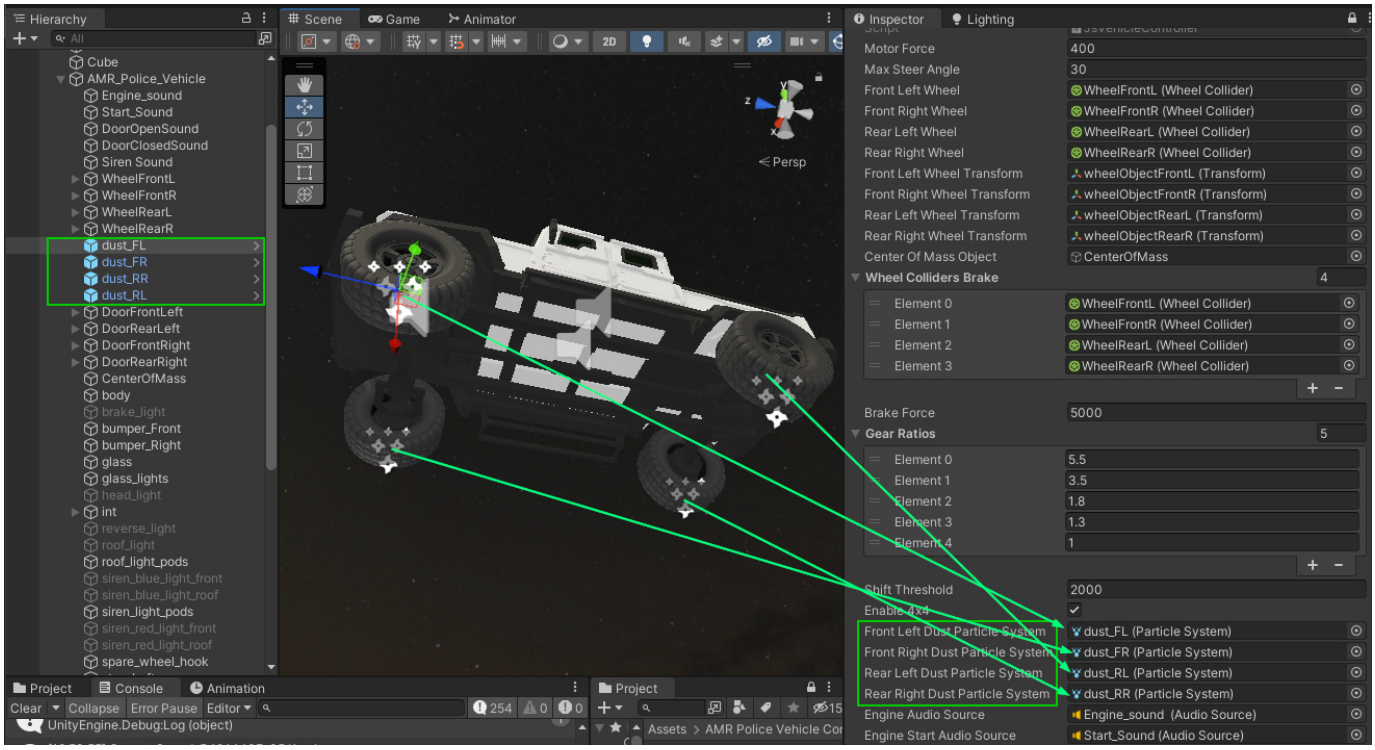
Reference to the dust particles system for each wheel. They are used to enable or disable the particle systems based on wheel slip, drift, and braking.

Front Left Dust Particle System : Assign Front Left Dust Particle System.

Front Right Dust Particle System : Assign Front Right Dust Particle System.

Rear Left Dust Particle System : Assign Rear Left Dust Particle System.

Rear Right Dust Particle System : Assign Rear Left Dust Particle System.

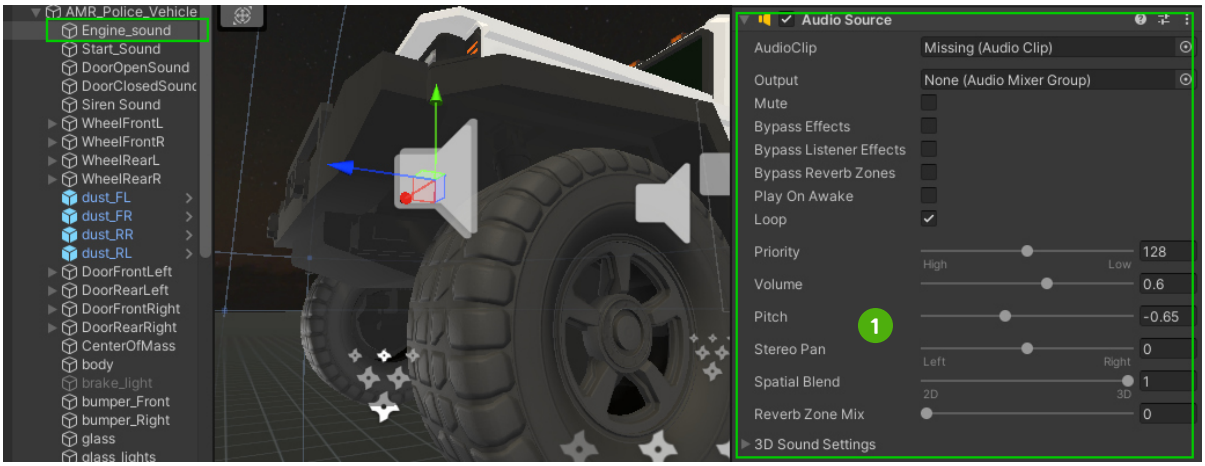


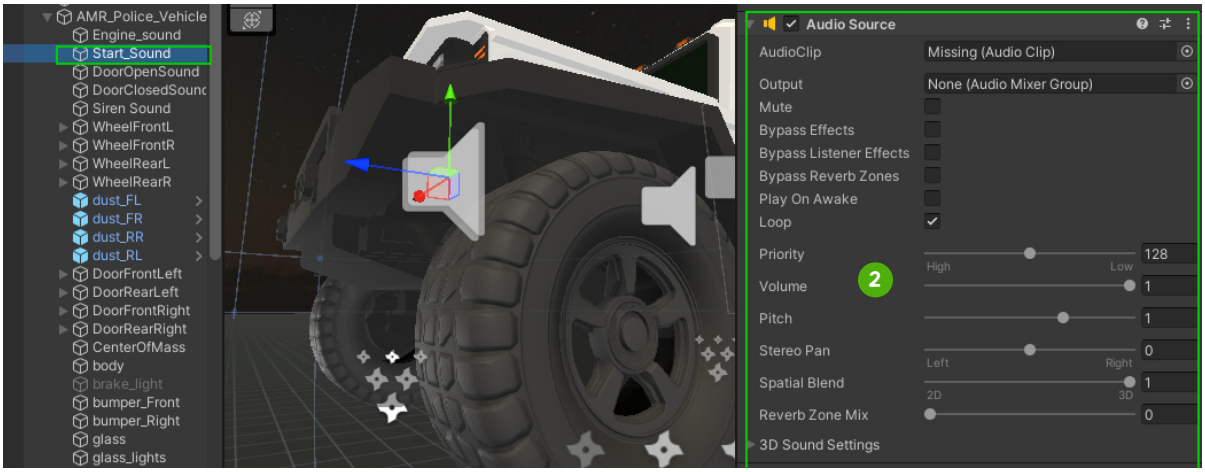
Audio Source:

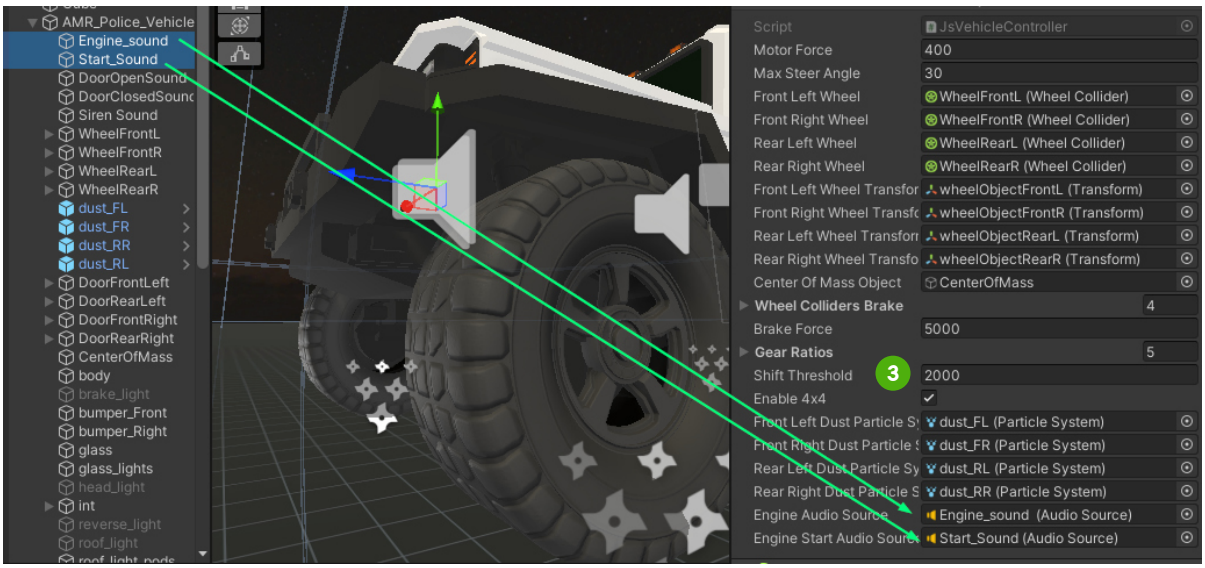
Engine Audio Source - This is the AudioSource component that will play the engine sound for the vehicle.

Engine Start Audio Source - This is the AudioSource component that will play the engine start sound when the vehicle begins moving.

Note: You can refer to this *How to make an Audio Source Component?*







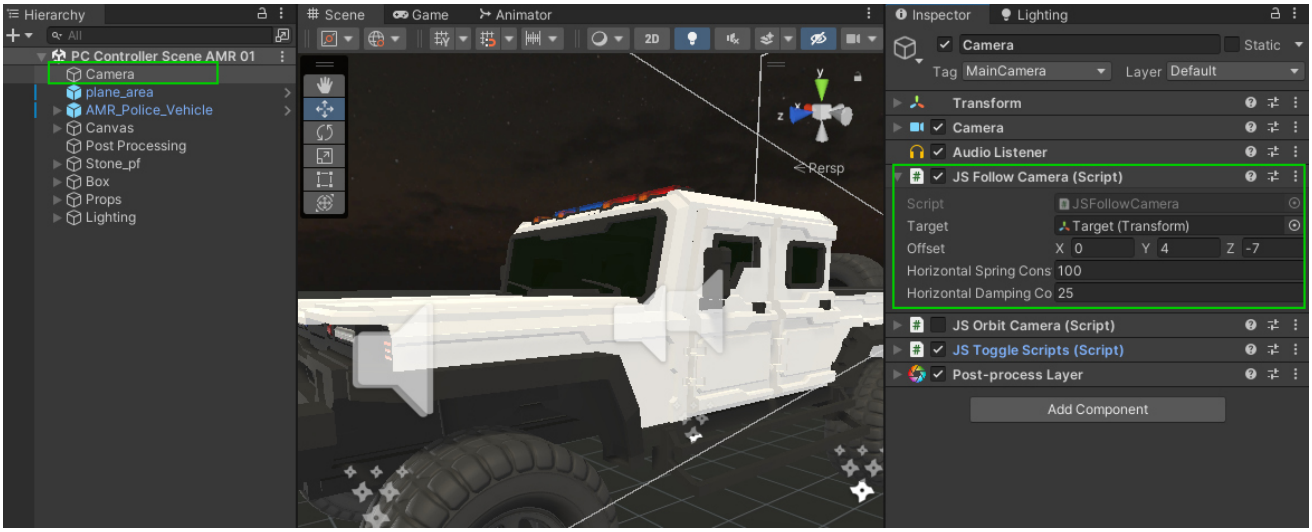
Camera Set-up

In these steps, you will learn how to set up and adjust the camera's parameters in the inspector. You can customize the behaviour of the Camera scripts to achieve the desired camera movement and zooming effects in your project.

JS Follow Camera (Script)

Understanding the parameters:

- **Target** - Assign an empty GameObject to the "Target" field in the inspector. This is the object that the camera will follow. Drag and drop the GameObject you want the camera to follow onto this field in the Inspector.
- **Offset** - This is the offset from the target that the camera will maintain. You can adjust the X, Y, and Z values to position the camera relative to the target. (eg 0,4,-7)
- **Horizontal Spring Constant** - This parameter controls how much the camera will spring back to its desired position horizontally. Adjust this value to change the camera's responsiveness to horizontal movement. (eg 100)
- **Horizontal Damping Constant** - This parameter controls how much the camera will dampen horizontal movement. Adjust this value to control how smooth or jerky the camera's movements are. (eg 25)

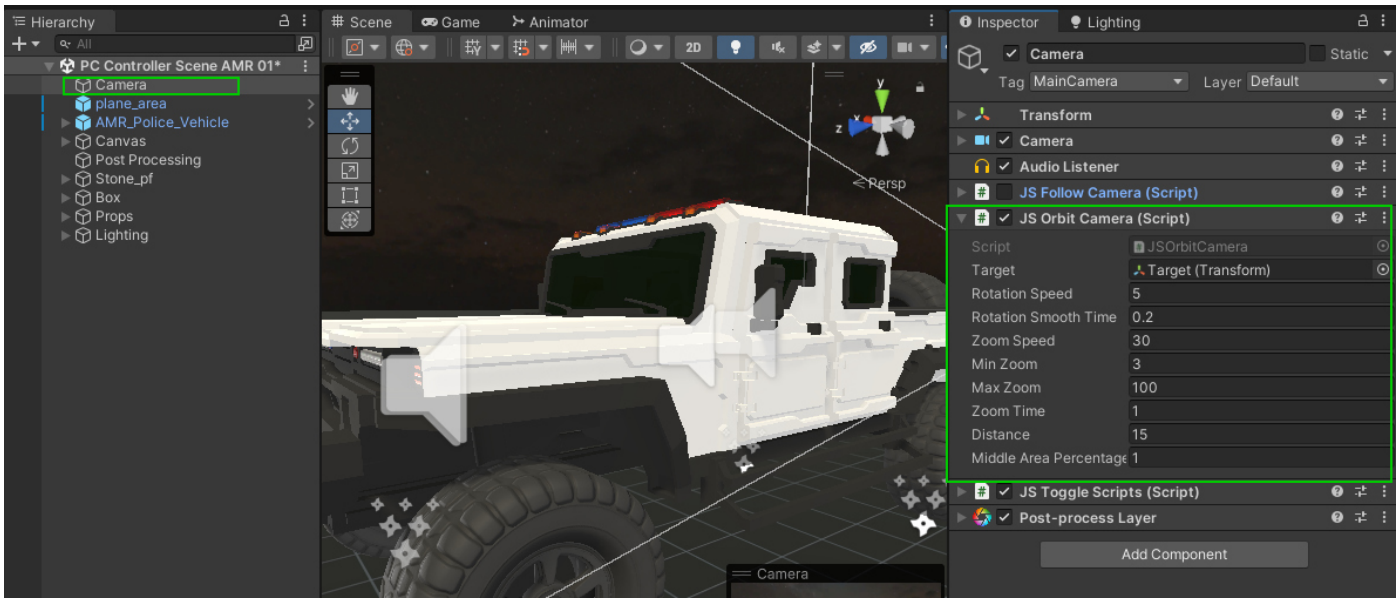


JS Orbit Camera (Script)

Understanding the parameters:

- **Target** - Assign an empty GameObject to the "Target" field in the inspector. This will be the object that the camera will orbit around.
- **Rotation Speed** - To control how fast the camera rotates around the target.
- **Rotation Smooth Time** - To control the smoothness of the camera rotation.
- **Zoom Speed** - To control how fast the camera zooms in and out.
- **Min Zoom** - To define the minimum zoom distances.
- **Max Zoom** - To define the maximum zoom distances.
- **Zoom Time** - To control the duration of the zoom animation.
- **Middle Area Percentage** - To define the size of the middle area on the screen where mouse and touch input will be captured or active in order to rotate the camera around the given target.

You can play the scene and interact with the camera using mouse or touch input within the designated middle area to rotate the camera around the target and zoom in and out.



Switch Camera View Using the Toggle Script

We need to attach both scripts (JS Follow Camera and JS Orbit Camera) to our camera so that we can switch between them using the JS Toggle Script. The ***JS Toggle script*** enables one script while disabling all other scripts.

JS Toggle (Script)

Let's start by looking at the parameters in the Inspector:

- **Script:** This is an elements of Script Toggles, where each element consists of a script and a toggle key. The script is a reference to a MonoBehaviour script in Unity.
- **Toggle key:** This is the keyboard key that will trigger the toggle action.

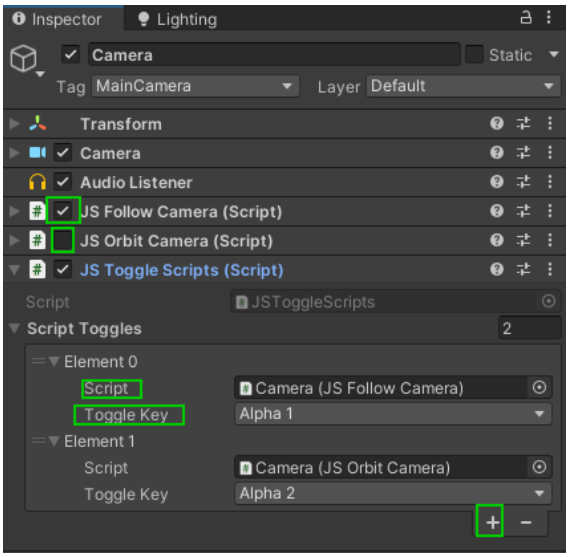
Now, when you run your Unity scene, you can press the assigned toggle key for each script to enable or disable it. Pressing the toggle key for a specific script will enable that script and disable all others.

How to use the toggle script to switch between cameras?

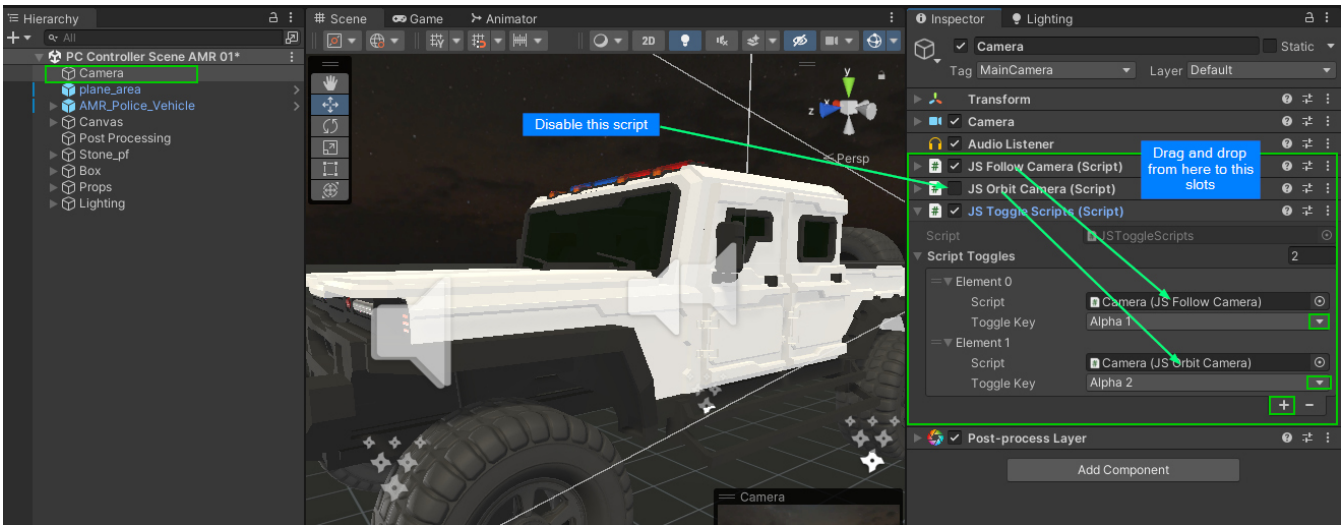
In this example, we will see how to switch between camera views using an assigned key. For camera switch using Mobile input please refer to [Camera Switch with On-Screen input]

Let's attach three scripts to this camera (Camera Follow): the ***JS Follow Camera script***, the ***JS Orbit Camera script***, and the ***JS Toggle script***.

Important : *Only one camera script should be enabled; all other camera scripts should be disabled in their checkboxes. In our case, we will enable only the JS Follow Camera script, so when we play the game, it will initially show this camera view.*



- he Inspector, you will see a section called ***Script Toggles***. Click on the **"+"** button to add a new element to the array.
- Since we have just two cameras lets add two elements by clicking on **"+"**
- **Inside Element 0**, you will see two fields: ***Script*** and ***Toggle Key.***
- **Script :** drag and drop the ***JS Follow Camera Script*** (drag it from within the inspector panel of Camera Follow not from your script folder).
- **Toggle Key :** specify the keyboard key that will trigger the toggle action lets select ***Alpha 1*** Inside ***Element 1.***
- **Script :** drag and drop the ***JS Orbit Camera Script*** (drag it from within the inspector panel of Camera Follow not from your script folder).
- **Toggle Key :** specify the keyboard key that will trigger the toggle action lets select ***Alpha 1***



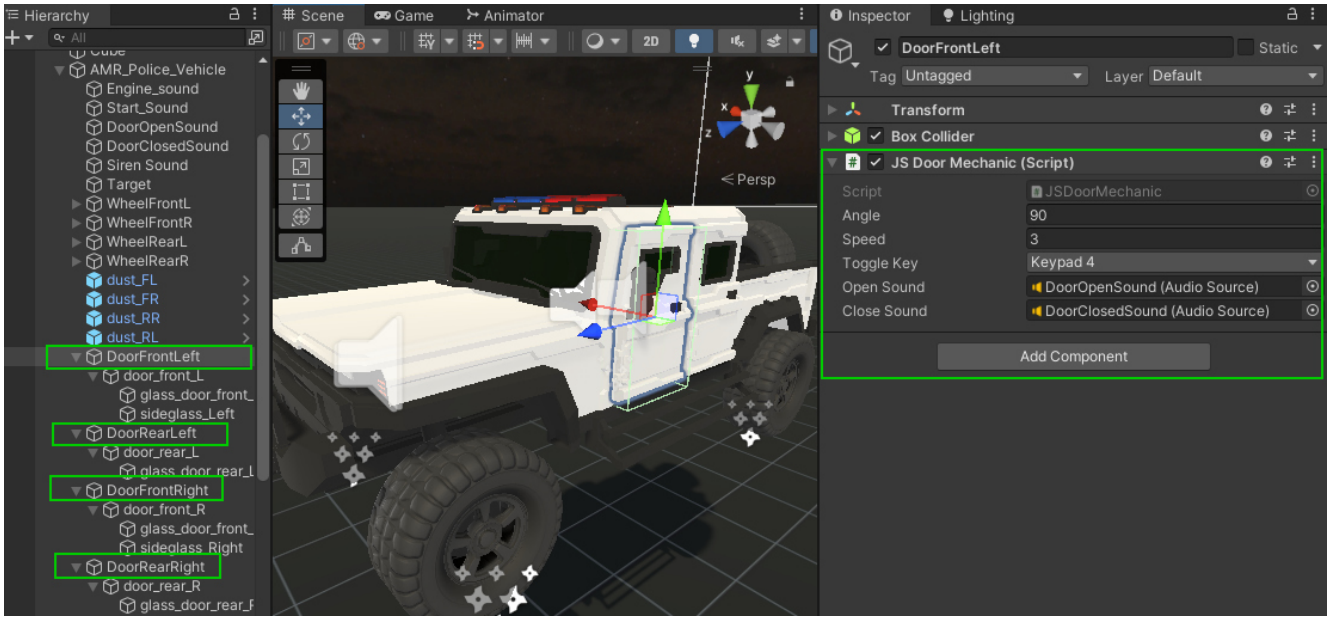
JS Vehicle Door (Script)

1. First, lets attach the script to the ***GameObject*** that represents the vehicle door you want to control.

2. Once the script is attached, you will see several parameters in the Inspector window that you can adjust:

- **Angle:** This parameter determines how wide the door opens. You can change it to adjust the angle at which the door should open.
- **Speed:** This parameter controls how fast the door opens and closes. You can increase or decrease it to change the speed of the door movement.
- **Toggle Key:** This parameter allows you to set a keyboard key that will toggle the door open and close. Pressing this key in play mode will trigger the door movement.
- **Open Sound:** This parameter lets you assign an AudioSource for the sound played when the door opens.
- **Close Sound:** This parameter lets you assign an AudioSource for the sound played when the door closes.

To use the script, simply press the assigned Toggle Key (default is Space key) while in play mode in the Unity Editor. This will open or close the door based on its current state.



JS Lock Rotation (Script)

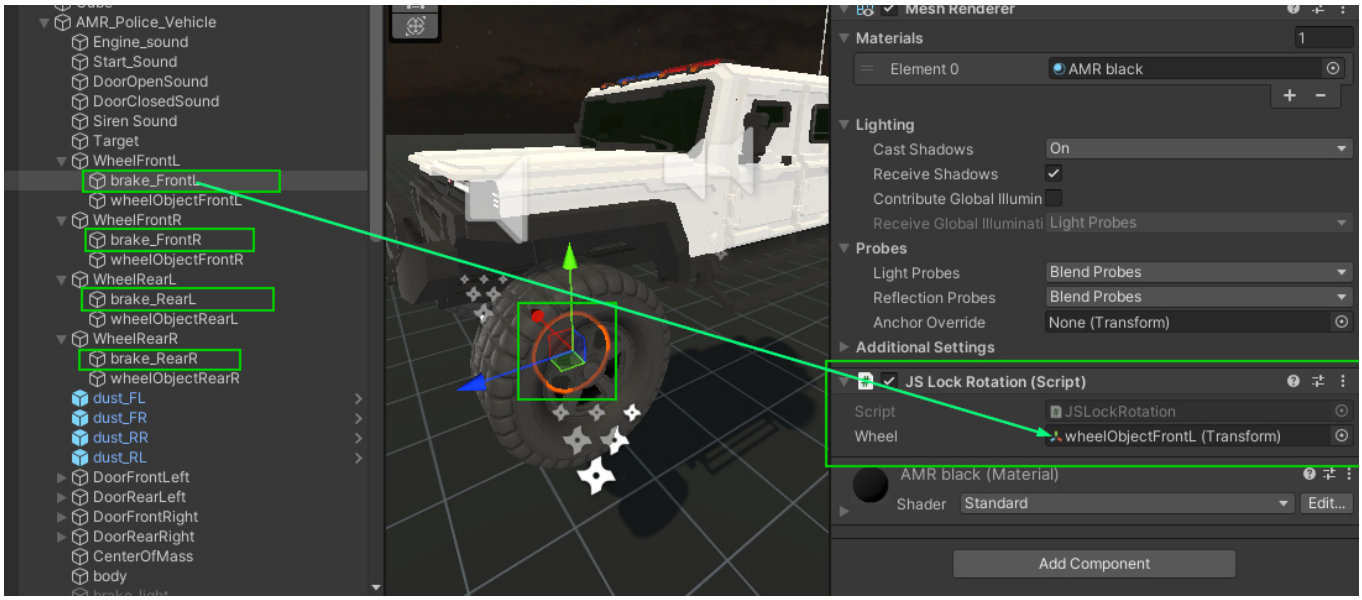
This script is used to sync the position of a given object (wheel) while locking its rotation. I used it for brake pads inside the wheel since I don't want the brake pads to rotate with the wheel but only move along with it.

Configure the Script Parameters

Wheel: This parameter requires a Transform object that represents the wheel. Drag and drop the wheel object from the Hierarchy window to this slot in the Inspector.

1. Press the Play button in the Unity Editor to run the scene.

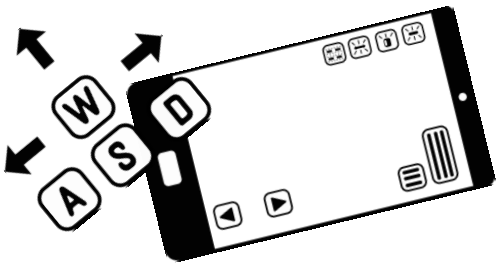
2. Observe how the object's rotation is locked but synchronized with the wheel's movement.



JS Input Controller (Script)

The Input Controller script is a crucial component in creating a realistic and responsive vehicle simulation within your Unity project. In this guide, we'll explore the different parameters and functionalities of the Input Controller Script to help you get started.

This script allows you to control the movement and various functions of your vehicle using both custom on-screen buttons (for Mobile) and keyboard inputs (for PC).



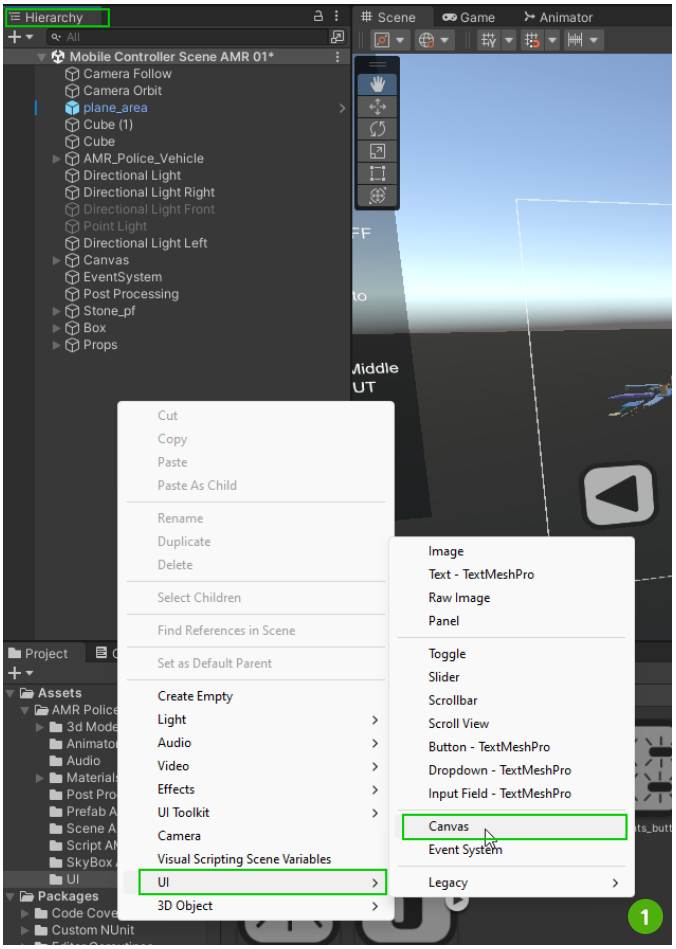
How to use the Vehicle Input Controller in your Unity project?

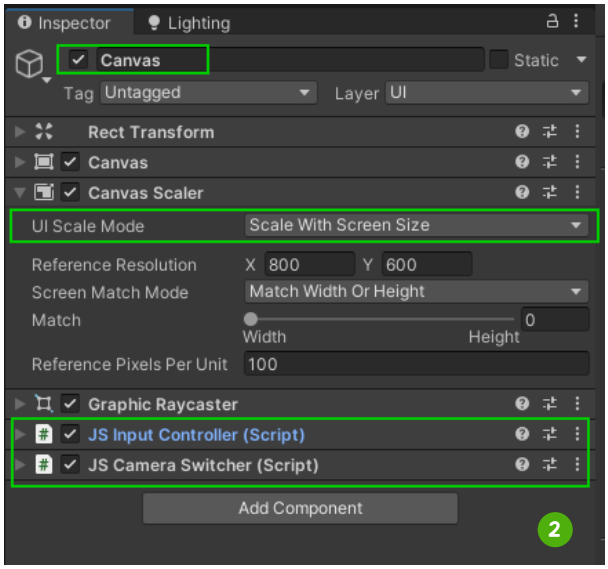
➡ Follow this Steps

On-screen buttons (for Mobile)

1

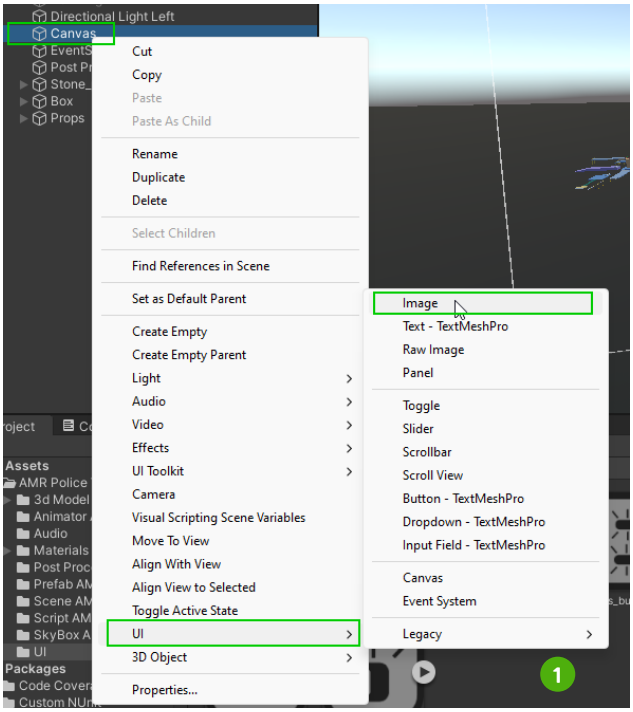
- First create a Canvas by *right-click* inside the Hierarchy panel goto *UI > Canvas*
- Now attach the *JSInputController (Script)* and *JS Camera Switcher (Script)* to the Canvas that you have just created.

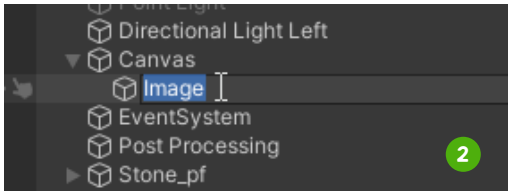


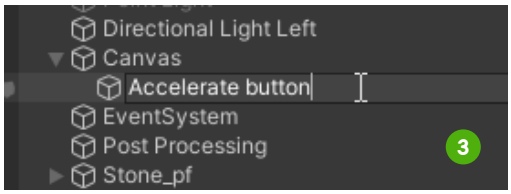


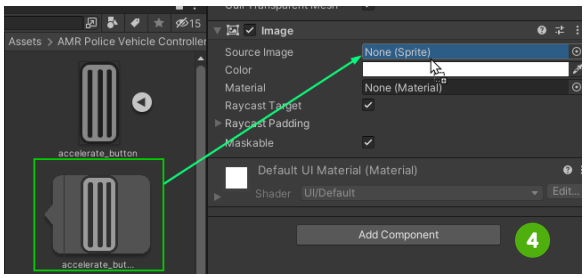
2

- Now we need to create an ***image button*** for each input. ***Right click*** on the ***Canvas***
- ***Goto UI > Image*** (this will create an image UI as a Canvas's child)
- This ***image*** will be our ***Accelerate button*** so lets rename as ***'Accelerate button'***
- Under this ***image (Accelerate button)*** in the ***Source Image*** we need to add an image for the accelerate button



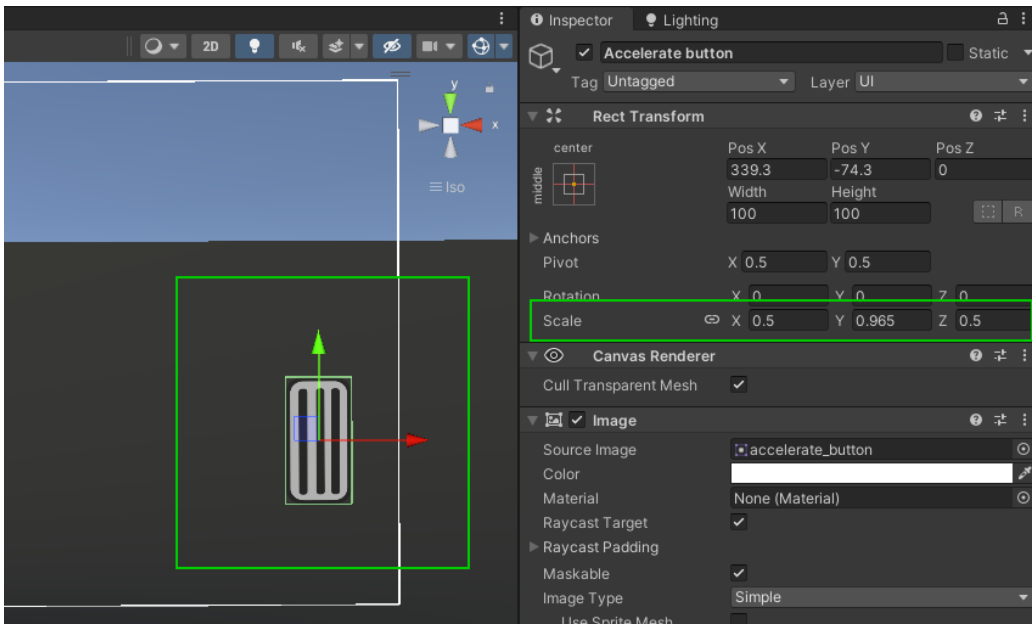






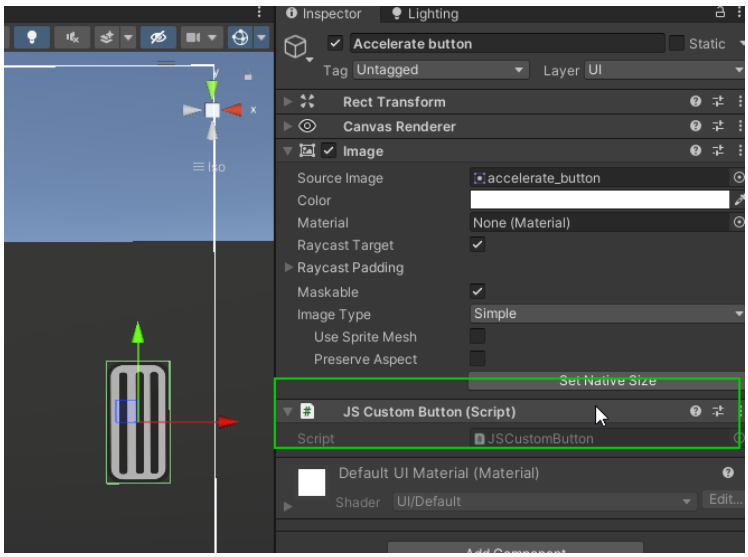
3

- Now under the ***Inspector panel*** adjust the ***scale*** of your button accordingly.
- In the ***Unity Editor*** you can position your button where you want to place them



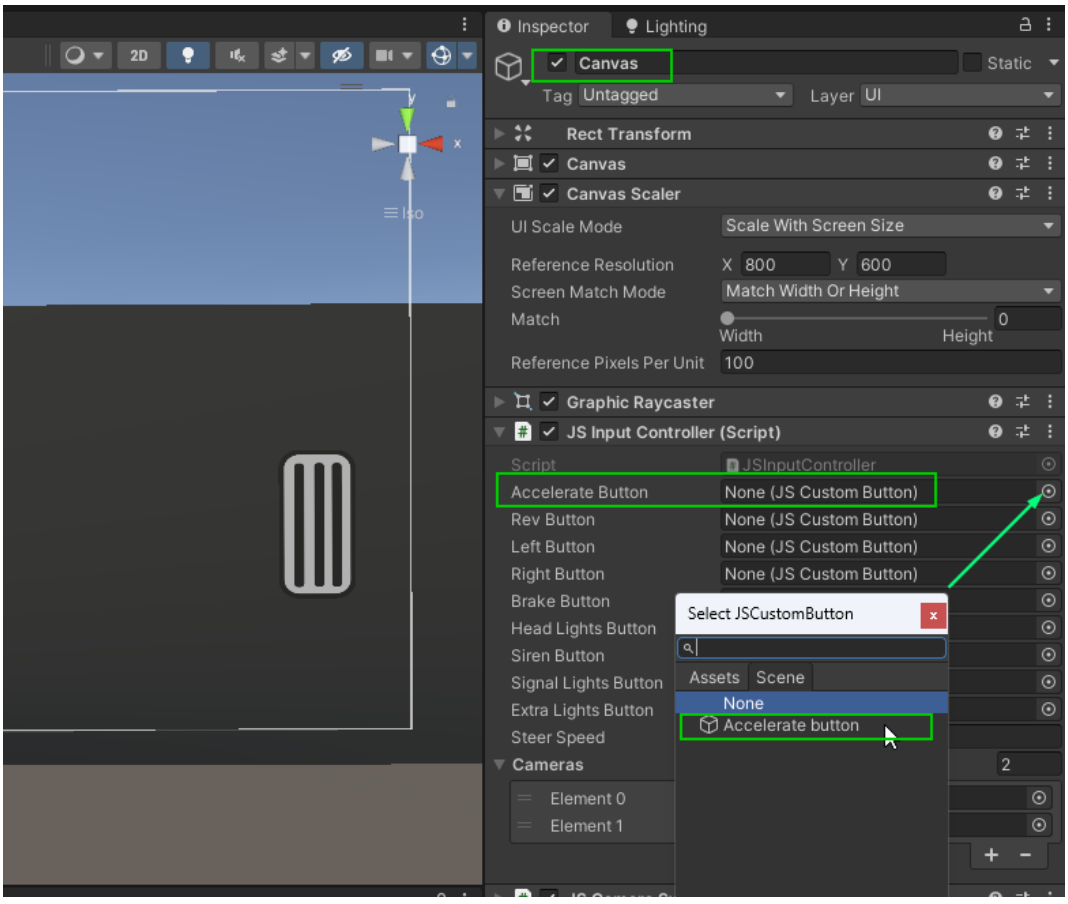
4

- To make ***Accelerate button*** function as a button, you need to attach the ***JSCustomButton*** script to it.



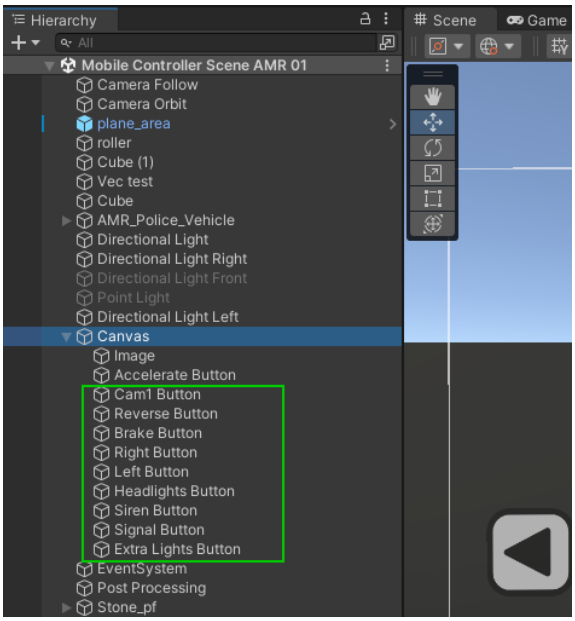
5

- Now select the ***Canvas*** from your ***Hierarchy*** and Under the ***JS Input Controller (Script)*** we need to assign each button accorgingly.
- To assign the ***Accelerate Button*** click on the ***small circle icon*** and then select ***Accelerate button***.

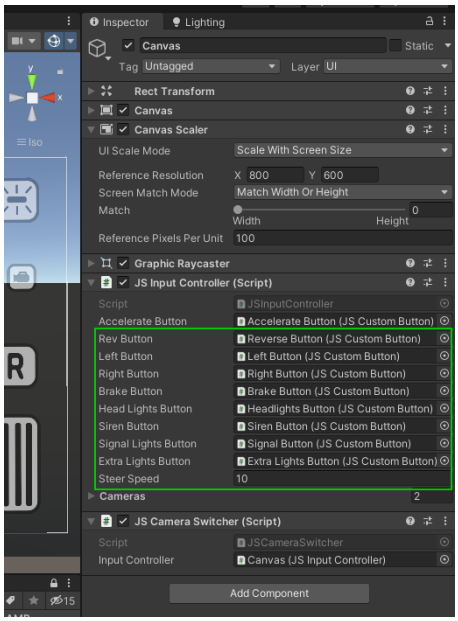


6

- We can repeat the same step to create all the required buttons.
- *Steer Speed:* Adjusting this value will affect how quickly the vehicle turns when you press the left or right buttons.



1



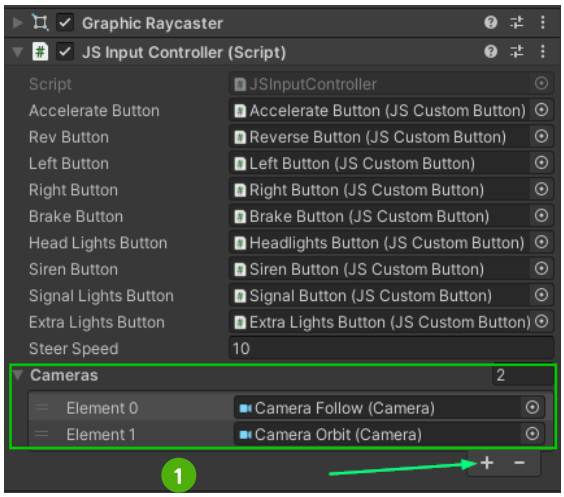
2

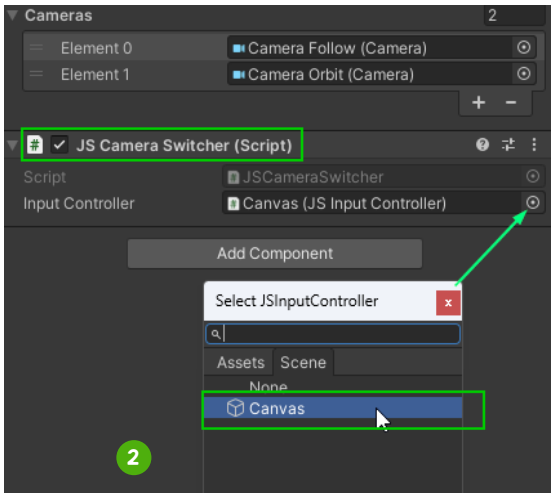
7

Camera Switch with On-Screen input

- In this final step we need to set our *cameras*. We will add two camers in this example, in case

  If you have more than two cameras you can add more by clicking on the "+" icon
- With you *Canvas* selected, inside the *inspector panel* you will see the *Cameras* parameter
- For Element 0 = Assign *Camera Follow* by cliking on the *small circle button*.
- For Element 1 = Assign *Camera Orbit* by cliking on the *small circle button*.
- Under *JS Camera Switcher (Script)* for the *Input Controller* we will assign our *Canvas* (since our Input controller Script was attached to our Canvas.)

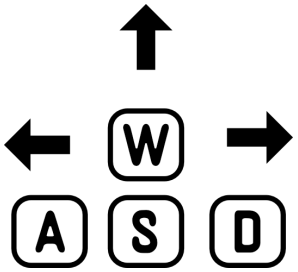




Keyboard Inputs (for PC)

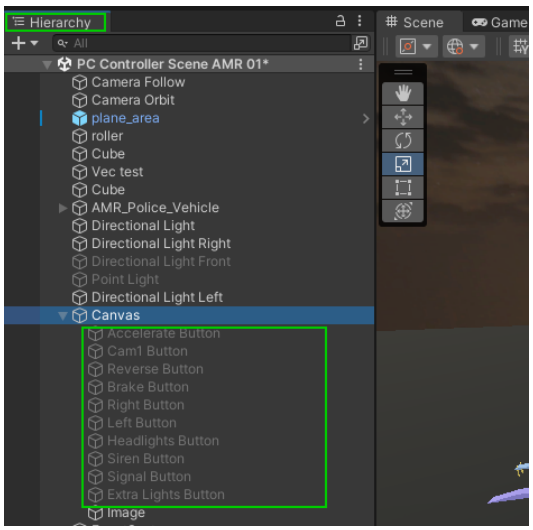
1

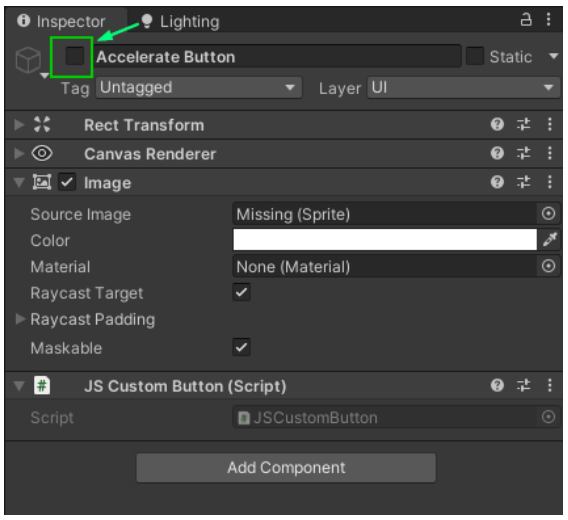
- First create a Canvas by ***right-click*** inside the Hierarchy panel goto ***UI > Canvas*** similar to what we did for the Mobile inputs.
- Now attach the ***JSInputController (Script)*** to the Canvas that you have just created.



2

- Since we don't need any button on the screen we will simply disable them.
- To disable, select the button from the Hirerchy and uncheck the checkbox in the Inspector panel





3

- Now we can use W/A/S/D key from the keyboard to drive the vehicle
- We can also customize the keys if needed by editing the JSInputController script with your desire KeyCode inside the Update() function.

```
0 references
private void Update()
{
    // Reset input values
    verticalInput = 0f;

    // Handle acceleration and braking
    if (Input.GetKey(KeyCode.W) || accelerateButton.IsButtonPressed())
    {
        verticalInput = 1f;
        //Debug.Log("Accelerate: verticalInput = " + verticalInput);
    }
    else if (Input.GetKey(KeyCode.S) || revButton.IsButtonPressed())
    {
        verticalInput = -1f;
        //Debug.Log("Brake: verticalInput = " + verticalInput);
    }
```

JS Input Controller (Script) Components:

The Vehicle Input Controller script consists of several key components, each with its own purpose and functionality. Let's dive into each of them:

1. Acceleration Button (accelerateButton): 

- This button is responsible for controlling the acceleration of the vehicle.
- When the button is pressed, the vehicle will start accelerating forward.

2. Reverse Button (revButton): 

- This button is responsible for controlling the reverse or backward movement of the vehicle.
- When the button is pressed, the vehicle will start moving in reverse.

3. Left Button (leftButton): 

- This button is responsible for controlling the left steering of the vehicle.
- When the button is pressed, the vehicle will start turning to the left.

4. Right Button (rightButton): 

- This button is responsible for controlling the right steering of the vehicle.
- When the button is pressed, the vehicle will start turning to the right.

5. Brake Button (brakeButton): 

- This button is responsible for controlling the braking of the vehicle.
- When the button is pressed, the vehicle will start slowing down and eventually come to a stop.

6. Headlights Button (headLightsButton): 

- This button is responsible for controlling the headlights of the vehicle.
- When the button is pressed, the headlights of the vehicle will turn on or off.

7. Siren Button (sirenButton): 

- This button is responsible for controlling the siren of the vehicle.
- When the button is pressed, the siren of the vehicle will turn on or off.

8. Signal Lights Button (signalLightsButton):

- This button is responsible for controlling the signal lights of the vehicle.

- When the button is pressed, the signal lights of the vehicle will turn on or off.

9. Extra Lights Button (extraLightsButton):

- This button is responsible for controlling any additional lights on the vehicle.

- When the button is pressed, the extra lights of the vehicle will turn on or off.

10. Steer Speed (steerSpeed):

- This parameter controls the speed at which the vehicle's steering responds to input.

- Adjusting this value will affect how quickly the vehicle turns when you press the left or right buttons.

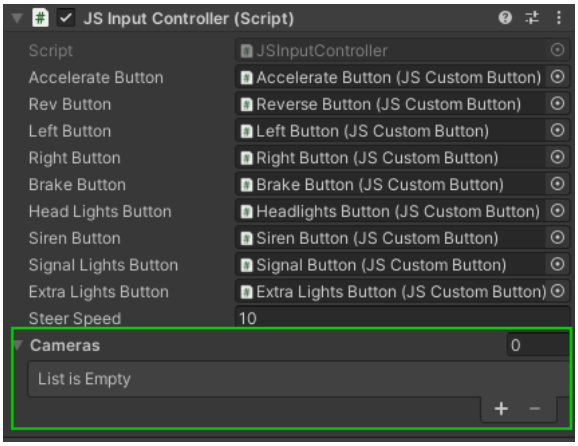
11. Cameras (for Mobile On-screen input only):

- This parameter allows you to assign multiple cameras to the Vehicle Input Controller, enabling camera switching for mobile input only. For PC inputs, we don't need to add any cameras here, as they were already set in the camera's Inspector panel. If we want to make any changes to the key, we can set it from the camera's Inspector panel.

- These cameras can be used to provide different views of the vehicle, such as a third-person perspective or a first-person view.

For Mobile Inputs

For PC Inputs (leave Cameras list empty)

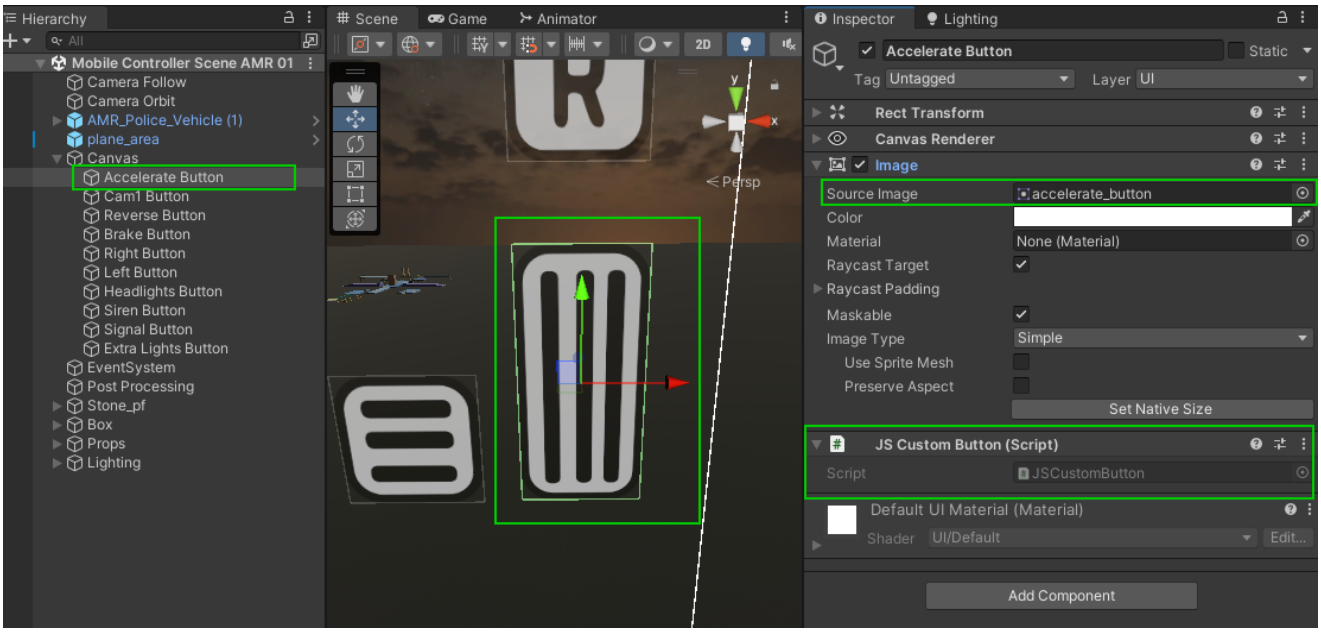


JS Custom Button (Script)

To use the JSCustomButton script in Unity, you need to attach it to a UI image or any other UI element that you want to function as a button.

In our case we need to create an image button as a child of the Canvas and attach the JSCustomButton script to it. Please refer here How to create an image button (Steps 2-4)

By attaching the ***JSCustomButton*** script to a UI image, you can create custom buttons with specific behaviour such as detecting when the button is pressed, released, or clicked.



JS Exit Application (Script)

The JSExitApplication script allows the player to exit the application, toggle between fullscreen and windowed mode, and resize the window on Windows platforms. It provides additional functionality for controlling the game window's display settings.

Now, let's explain what the script does:

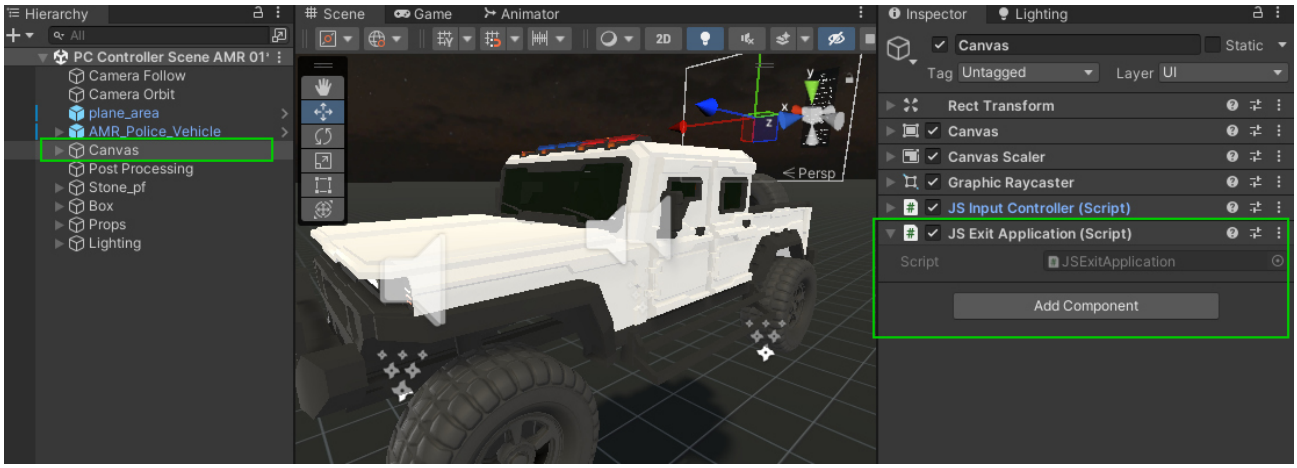
- If the player presses the ***Escape*** key, the game will either stop playing in the Unity Editor or exit the application in standalone mode.

- Pressing the ***PageUp*** key toggles between fullscreen and windowed mode.

- Pressing the ***PageDown*** key resizes the window to a specific size (for Windows platforms only).

- The ToggleFullscreen method changes the game's display mode between fullscreen and windowed based on the isFullscreen variable.

- The WindowResize method uses Windows API functions to resize and show the game window to a specific size.



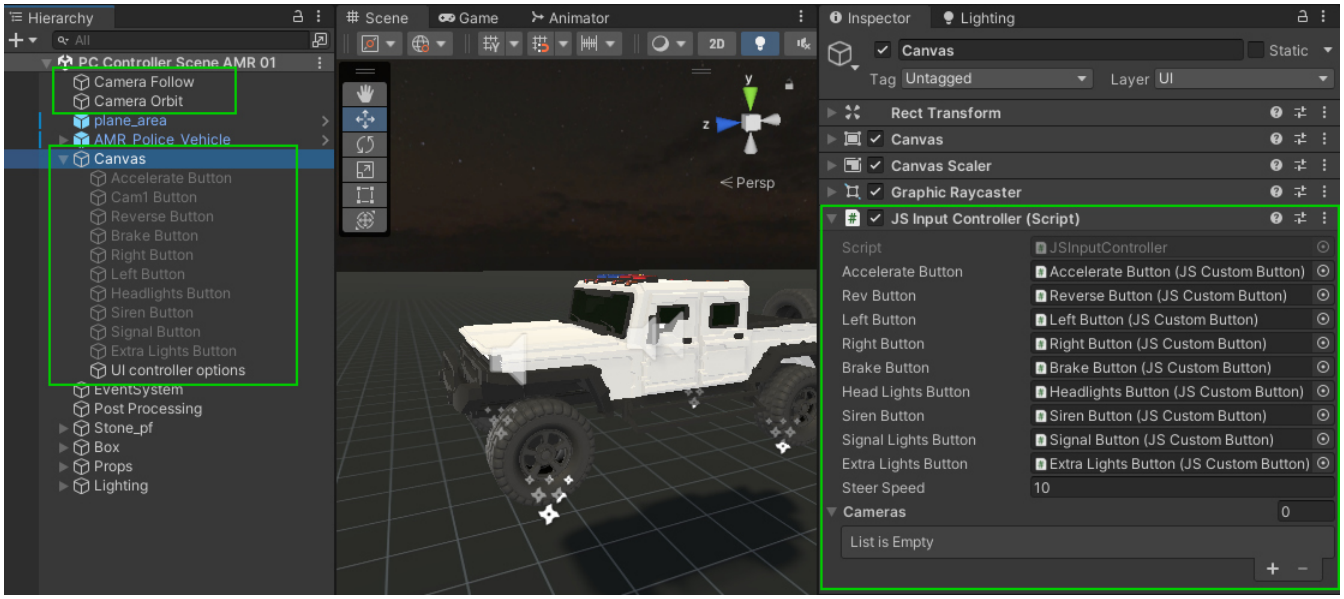
Trouble Shooting

Hierarchy Structure Error:

If you encounter any errors regarding ***NullReferenceException***, please check your Hierarchy Structure or for GameObjects missing from your Hierarchy.

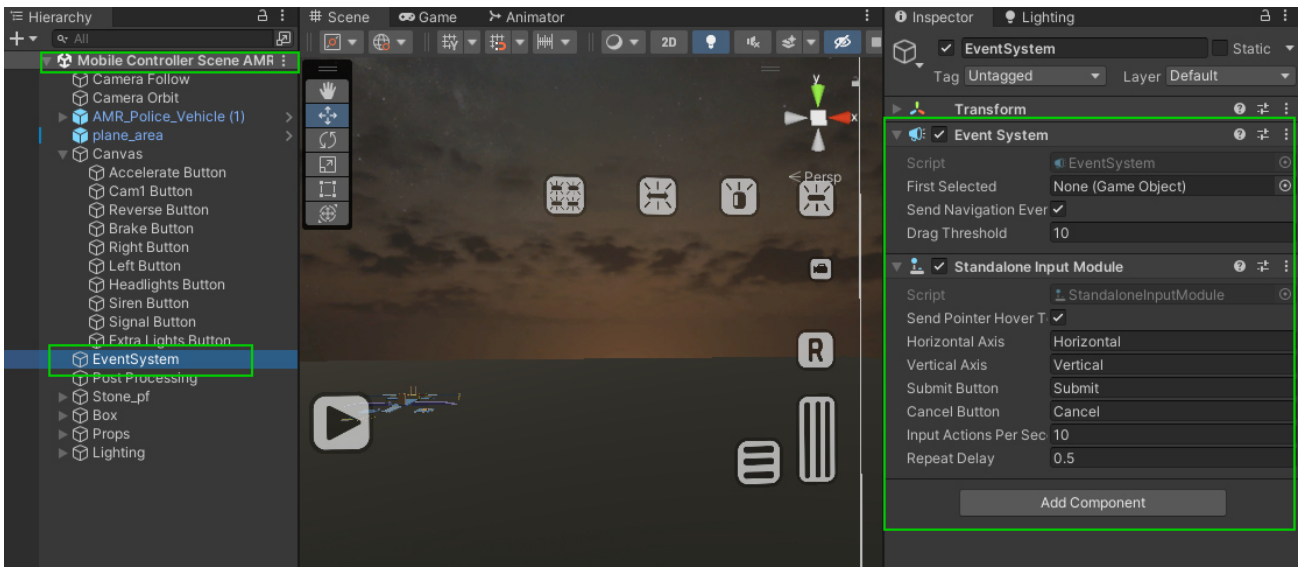
✓ ***Here are some of the most important objects from your hierarchy that should not be missing:***

- **Camera:** Your scene should have two Cameras ***Camera Follow*** and ***Camera Orbit***.
- **Canvas:** Since we're using the same script function for both keyboard input and on-screen input, we need to have the canvas with all the buttons as a child. *Even if we're making a game for desktop and need to disable them, they still need to be present in our hierarchy.*



- **EventSystem:** This is required for On-scree Mobile input. The EventSystem in Unity is responsible for managing input events, such as clicks from the mouse or touches on a touch screen.

  When you dont have EventSystem, it means that Unity is no longer listening for these input events, which is why your on-screen button may not work.

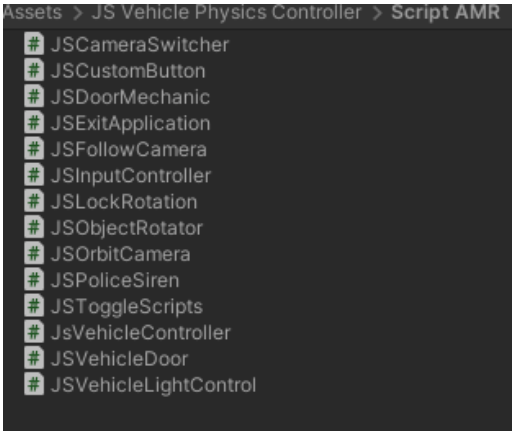


Missing Scripts and Unassigned Object Reference:

Scripts:

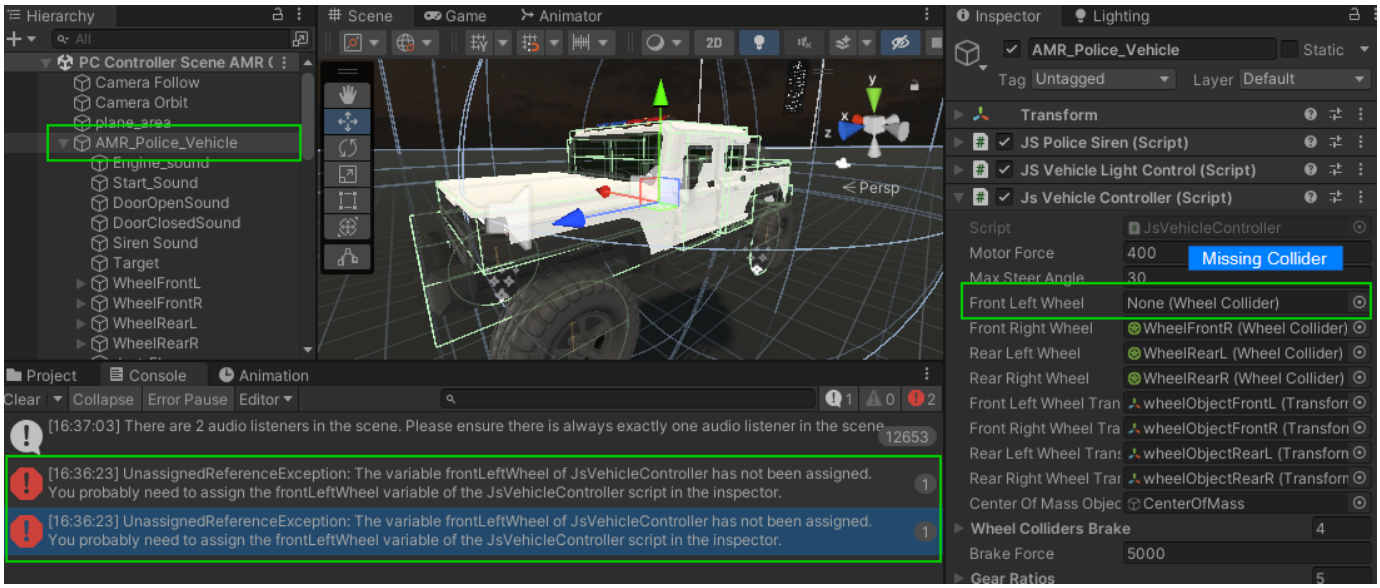
Please check that all the scripts below are present and have been attached accordingly.

JSCameraSwitcher, JSCustomButton, JSDoorMechanic, JSFollowCamera, JSOrbitCamera, JSInputController, JSLockRotation, JSPoliceSiren, JSToggleScripts, JsVehicleController, JSVehicleDoor, JSVehicleLightControl, JSExitApplication



Unassigned Objects:

If you encounter any errors regarding UnassignedReferenceException, please check that all objects have been assigned within your script. There should not be any missing objects from any component of your script.



Light System not function for On-Scree (Mobile input)

Since we're using the same script function for both keyboard input and on-screen input, we might encounter some problem with the Brake and Reverse light not function when using the On-Screen input in Mobile

➡To Fix this we need to Uncomment the lines of code related to mobile input controller for this functionality to work. Open the JSVehicleLightControl.cs and uncomment the below line of codes inside the Update() function

➡ *For Reverse Button*

```
// Uncomment the below function so that the the reverse light for mobile will work
    if (mobileInputController.revButton.IsButtonPressed())
    {
        ToggleReverseLights(true);
    }

    if (mobileInputController.revButton.IsButtonReleased())
    {
        ToggleReverseLights(false);
    }
```

➡ *For Brake Button*

```
//Uncomment the below function so that the the brake light for mobile will work
    if (mobileInputController.brakeButton.IsButtonPressed())
    {
        ToggleBrakeLights(true);
    }

    if (mobileInputController.brakeButton.IsButtonReleased())
    {
        ToggleBrakeLights(false);
    }
```

Please Check out..

our [Asset Store Page](#) if you need any 3d Vehicles! for your projects!

Asset Store Link: https://assetstore.unity.com/publishers/26267

Contact US

Mail: support@jermesa.com

Website: www.jermesa.com

Facebook: https://www.facebook.com/jermesastudio/

YouTube : https://www.youtube.com/channel/UCqvdhTGWLh0xo9epibCmpqA

WhatsApp: https://wa.me/message/UFORWWO2D45BJ1

Instagram: https://instagram.com/jermesa_studio

Twitter: https://twitter.com/Jermesastudio

Discord: https://discord.gg/4jC5BnzJvT

Asset Store: https://assetstore.unity.com/publishers/26267

LinkedIn: https://www.linkedin.com/company/jermesa-studio/

'Render Shot