

# ULTIMATE ENDLESS RUNNER KIT

## USER MANUAL



iOS



WEB



POWERED BY

unity



## Table of Contents

|   |    |
|---|----|
| Getting Started.....                                | 3  |
| Frequent References .....                           | 3  |
| Player .....  | 3  |
| Patch.....  | 3  |
| Spline.....   | 3  |
| CPs .....   | 3  |
| Audience .....                                      | 3  |
| Project Architecture .....                          | 4  |
| Physics.....  | 4  |
| Menus.....  | 4  |
| Collisions .....                                    | 4  |
| Sounds .....  | 4  |
| Adding an Obstacle.....                             | 4  |
| Replacing Player Character.....                     | 9  |
| Adding a Sound.....                                 | 12 |
| Adding a Patch .....                                | 16 |
| Adjusting CPs of a Straight Patch .....             | 21 |
| Adjusting CPs of a Curved Patch.....                | 21 |
| Adding a Menu .....                                 | 25 |
| Replacing the Enemy Character .....                 | 32 |
| How to Switch between Gyro and Swipe Controls ..... | 33 |
| Adding a Power-up.....                              | 35 |

# Introduction

---

## Getting Started

1. Import the unity package into an empty project.
2. Go to Project > Scenes > Scene 1.
3. Click “Don’t Save” if prompted to save the current scene. (The project only uses a single scene file)
4. Press the play button to get an idea of the menu flow and gameplay.

## Frequent References

The documentation will frequently refer to some elements that are the core of the project:

### Player

The *Player* is the prefab that contains the primary character of the game. This prefab also contains various scripts. The word “Player” will not be used to refer to the user in this document. The *Player* behavior is controlled by *ControllerScript.js*.

### Patch

Patches are the environments/ surroundings of the game. They are generated by *PatchesRandomizer.js* during run time. Each patch has the default displacement of 3000 meters.

At any particular time, two patches are always active, the one the *Player*, is on and the immediate next one. Every time the *Player* passes a patch completely it is destroyed and the next one is generated.

### Spline

The spline is a line passing over each patch, which defines the path of the *Player* and the location where obstacles will be generated.

### CPs

The spline is manipulated using CPs, which is short for Check Points. If the patch is completely straight, only two CPs are needed to draw a straight spline. However, if the path has bends and curves, multiple CPs are needed to be arranged to draw a spline that has exactly the same shape and centered on the path of the patch. Take a look at the sample patches to acquaint yourself with how CPs are related to the spline.

## Audience

The user guide is composed assuming that the reader/ developer has an understanding of game development in Unity and Javascript programming for Unity.

## Project Architecture

The project doesn't use many game objects, physics, GUI components etc. provided by Unity to maximize performance on mobile platforms.

### Physics

For instance the character/ player physics has been defined as needed in the *ControllerScript.js*.

### Menus

In the same way menus do not use the default GUI components and the custom implementation is defined in the *MenuScript.js*.

### Collisions

There are two kinds of collisions detected during gameplay. The first and the most common one is the front collisions with obstacles detected and handled by the *PlayerFrontCollider*.

The second type of collisions is called "stumble" which are detected by *PlayerSidesCollider* if the *Player* bumps into an obstacle sideways. This triggers the *Enemy* (police car by default) to chase the *Player* for a unit time (5 seconds by default) and then move out of the camera. If the player stumbles (collides sideways) into another obstacle while the *Enemy* is chasing it, game over routine is called and the death scene is played.

### Sounds

All the sounds used in the scene are 2D sounds. All the audio sources have been added to the *SoundManager* prefab located in the scene. A script also named as *SoundManager* has been added to the prefab and is responsible for playing or stopping a sound as needed. Exposed variables are used to hold the references of the audio sources.

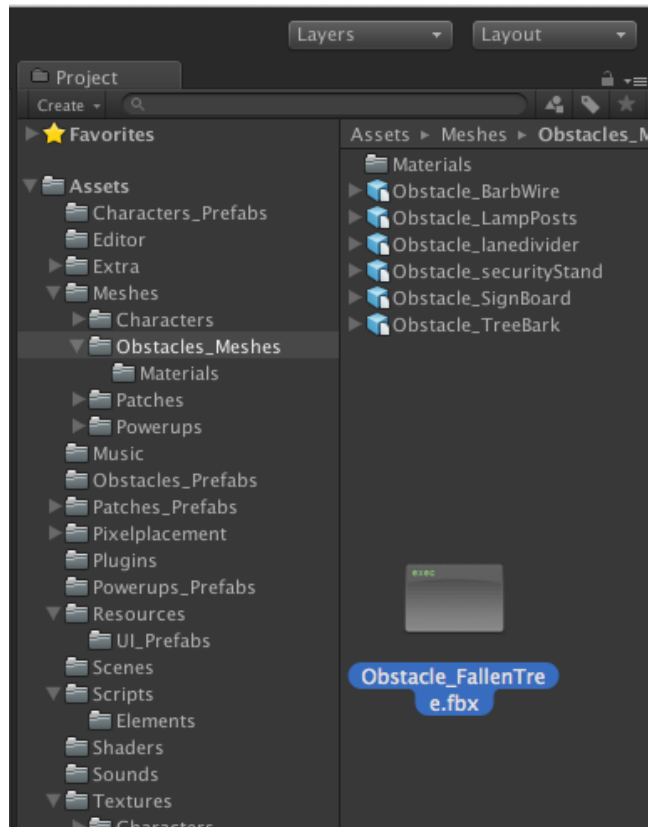
## Personalizing the Template

---

### Adding an Obstacle

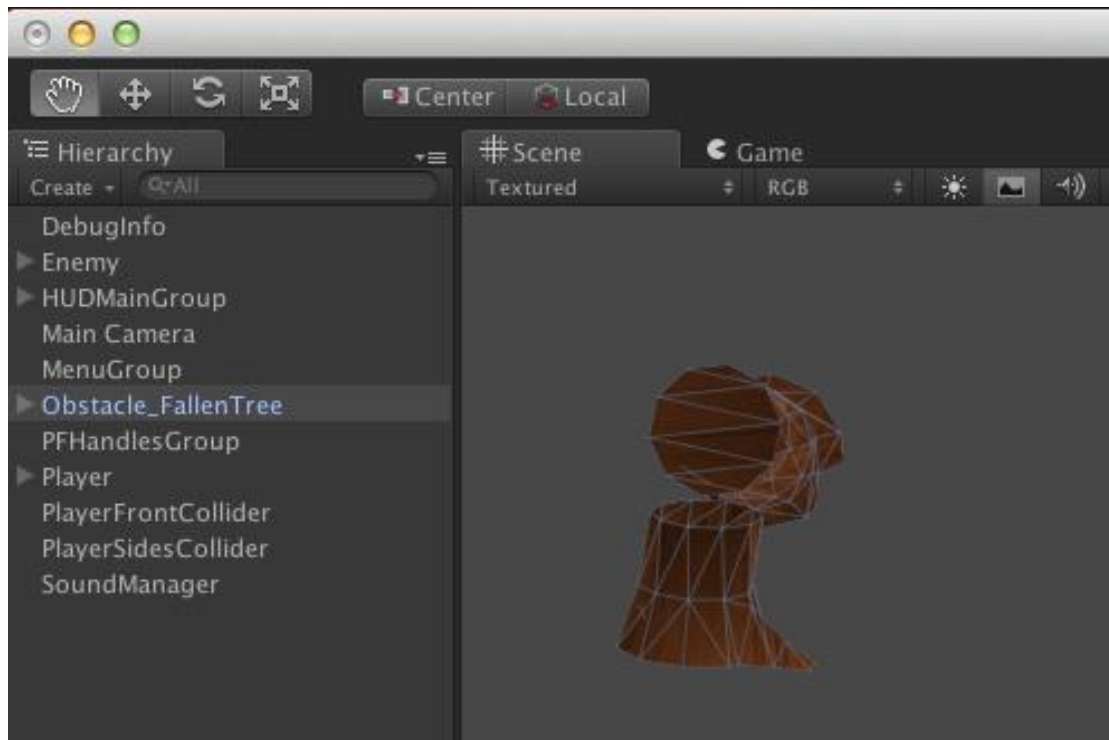
1. Import the .fbx of the asset that you will use as an obstacle.



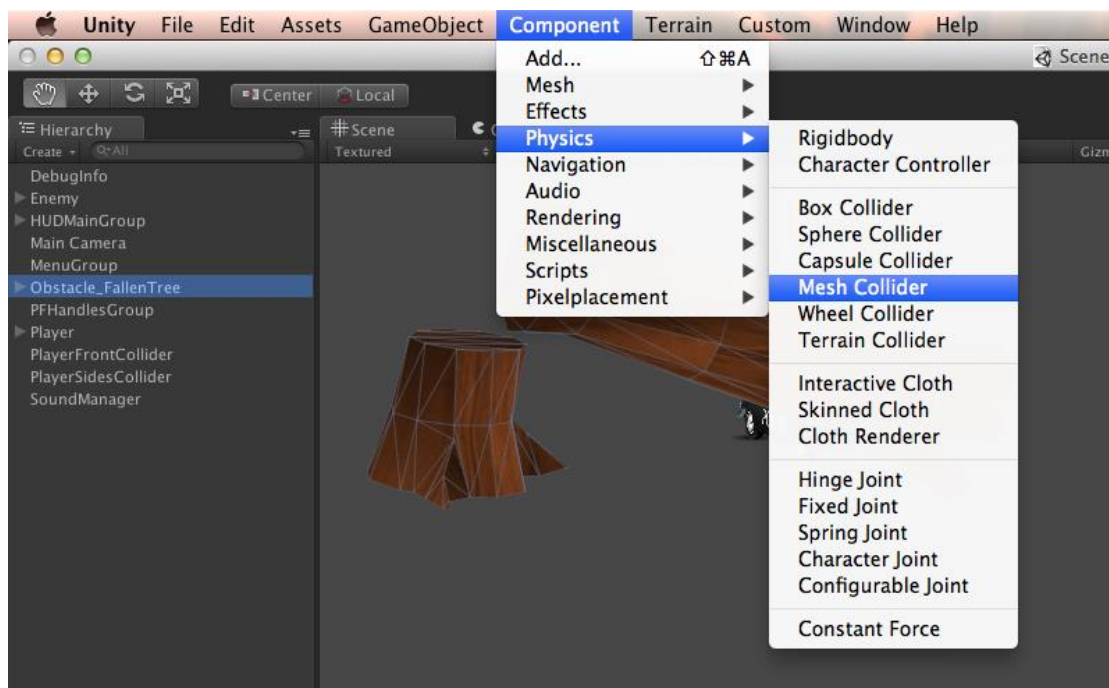


### Optimization Notes:

- Use Model > Scale Factor to adjust the scale.
  - Use Model > Mesh Compression to compress the imported asset as much as you can without compromising its appearance.
  - Select "None" in Rig > Animation Type if your asset isn't rigged.
  - Uncheck the Import Animation option in Animations > Import Animation if your asset isn't animated.
2. Drag the imported asset in "Hierarchy" tab to create a prefab.



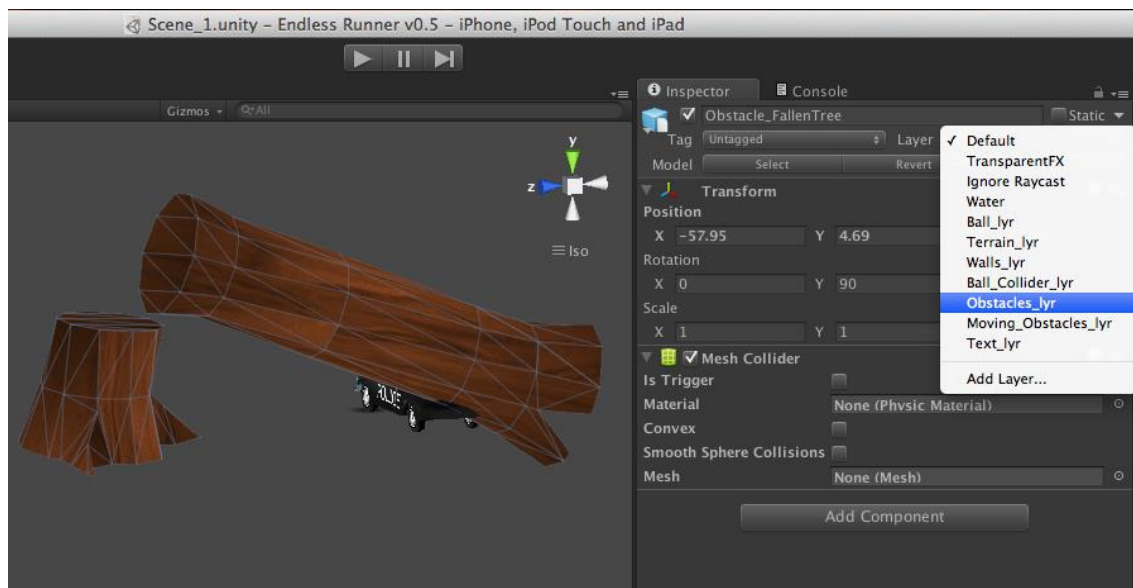
3. Select Component > Physics > Mesh Collider to add a mesh collider to your obstacle asset.



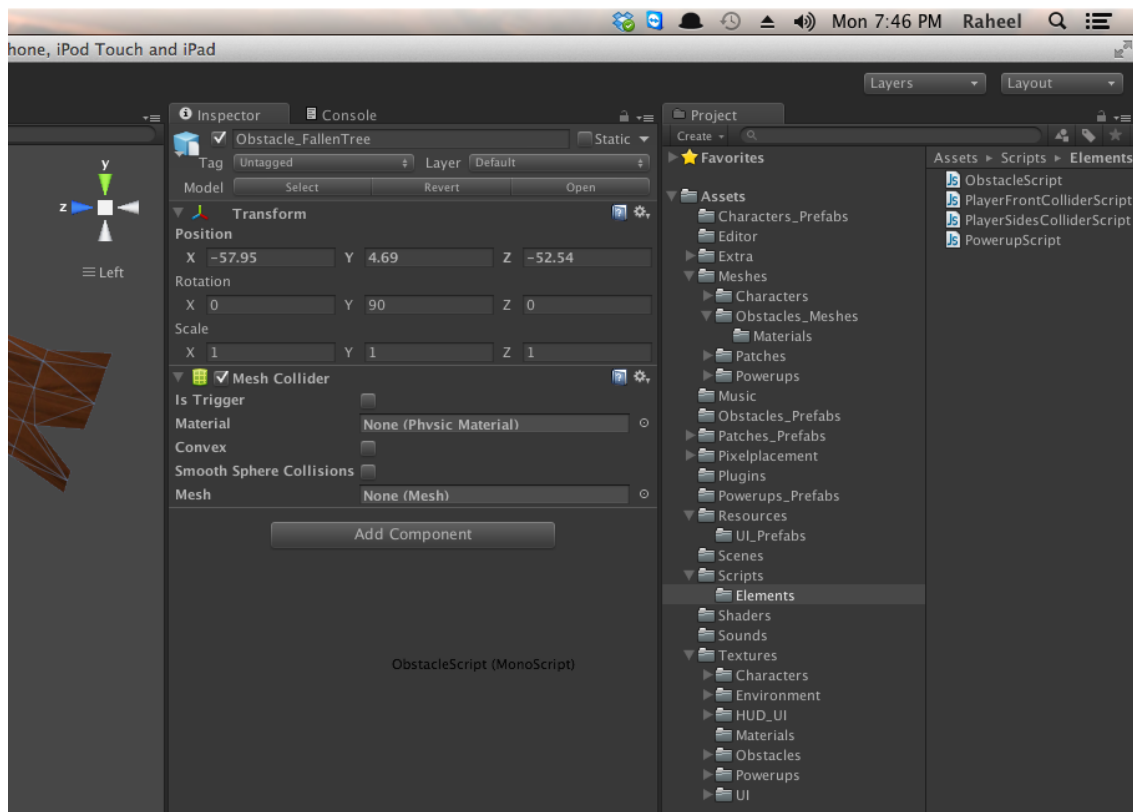
### Optimization Notes:

- Use primitive colliders such as Sphere, Box etc. for better performance during gameplay.
  - Do not scale the collider component on your prefab for optimizing object pooling.
4. Select Layer > Obstacles\_lyr in the *Inspector* tab.

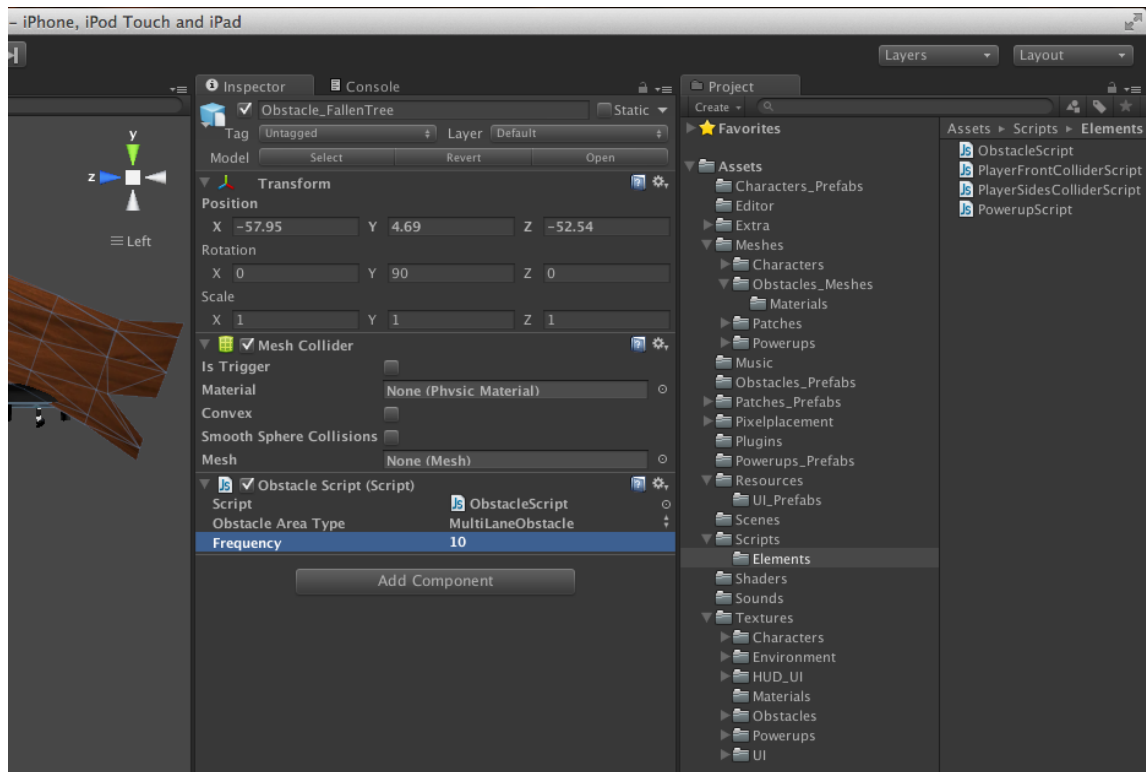
**Note:** Make sure the object with the collider has the layer “Obstacle\_lyr”. You can also change the layer of all the objects of an obstacle prefab to “Obstacle\_lyr” to keep things simple.



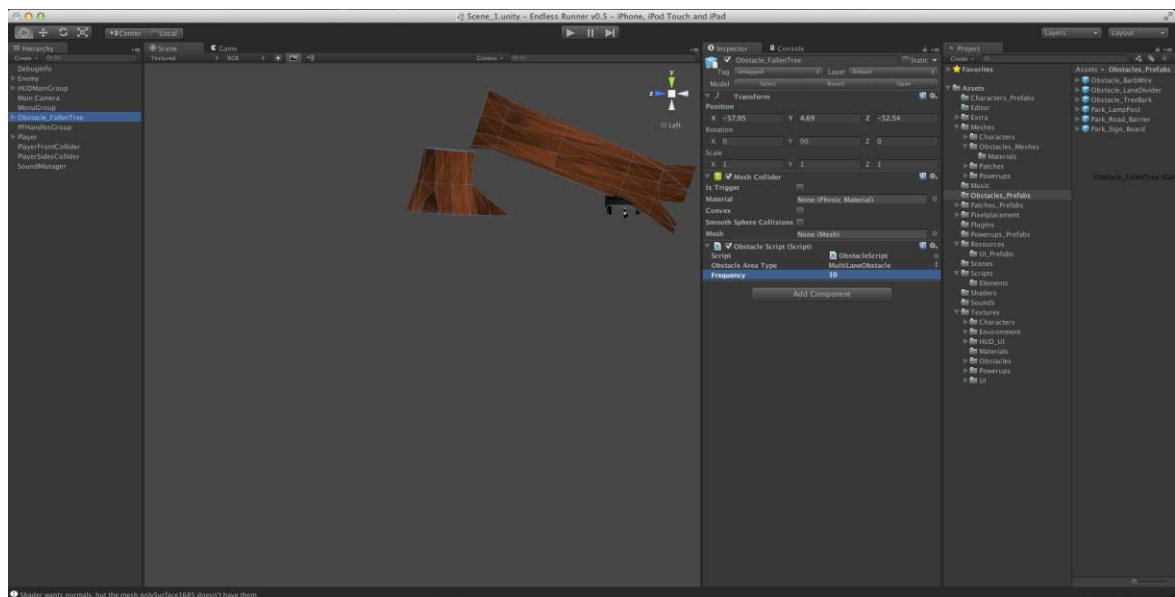
5. Go to Project > Assets > Scripts > Elements.
6. Add the “ObstacleScript.js” on your obstacle’s prefab.



7. Set the *Obstacle Area Type* on the *Obstacle Script* Component.
8. Set the Frequency. (Default Value is 10)

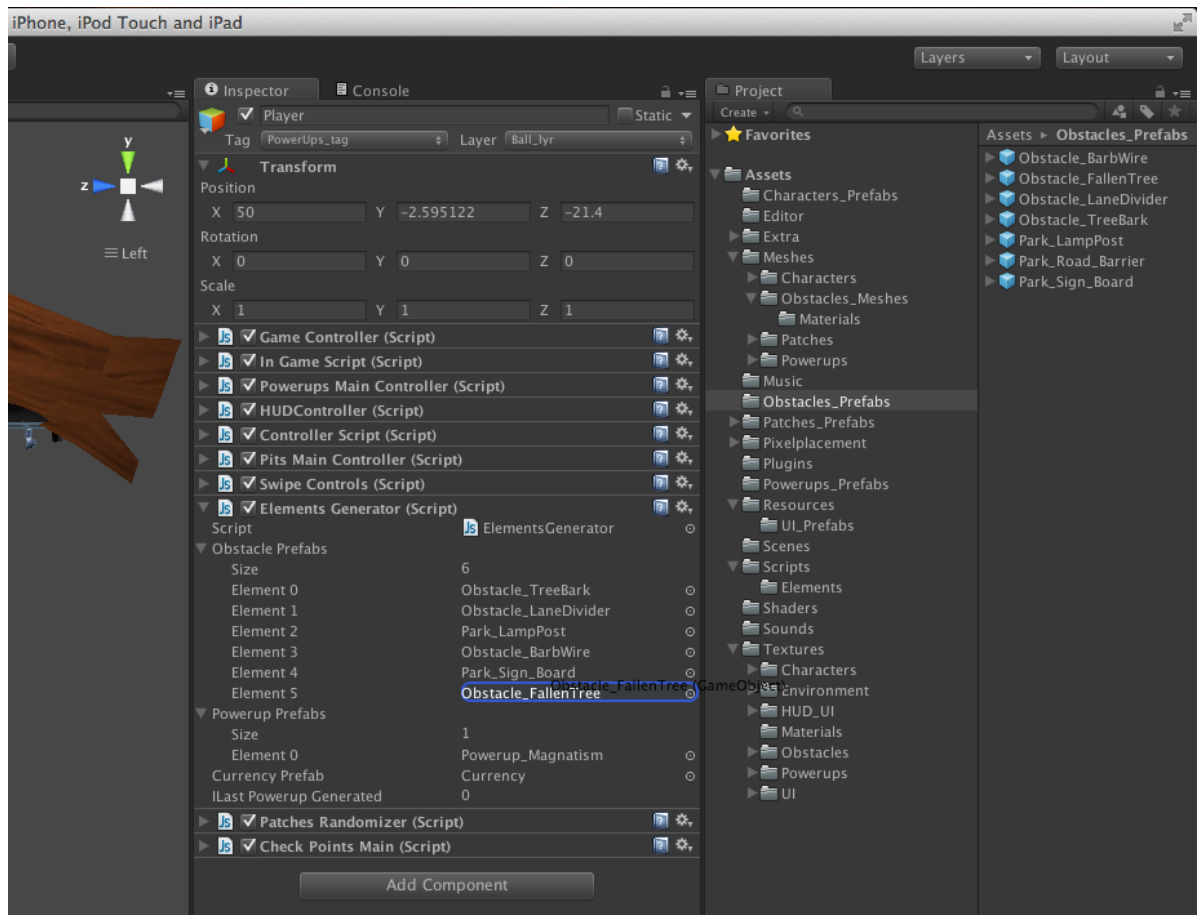


9. Drag the newly created prefab to *Obstacles\_Prefabs* or any other folder in the *Project* tab to create a prefab.



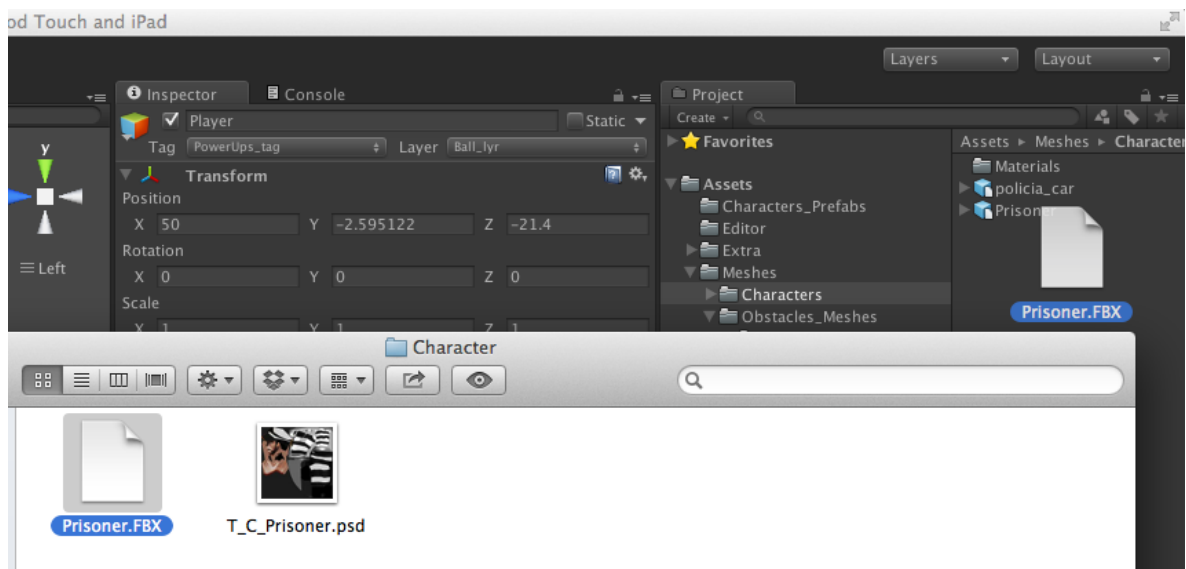
10. To make the created obstacle appear on the path during gameplay, you need to add it to Hierarchy > Player > Elements Generator > Obstacle Prefabs.
11. Remove the newly created obstacle from the Hierarchy tab before launching the game.





## Replacing Player Character

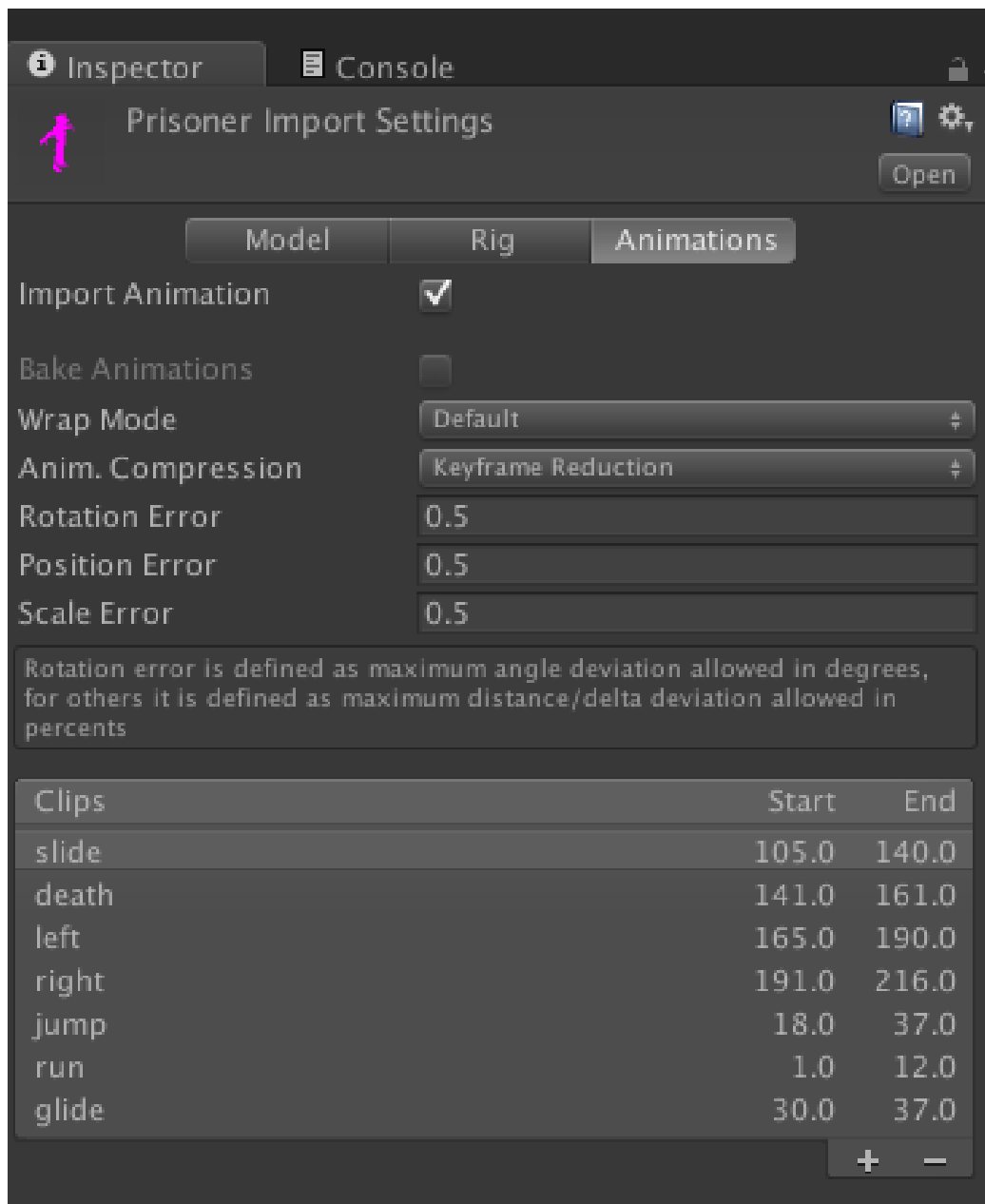
1. Import the player .fbx that will be used as the main character.



### Optimization Notes:

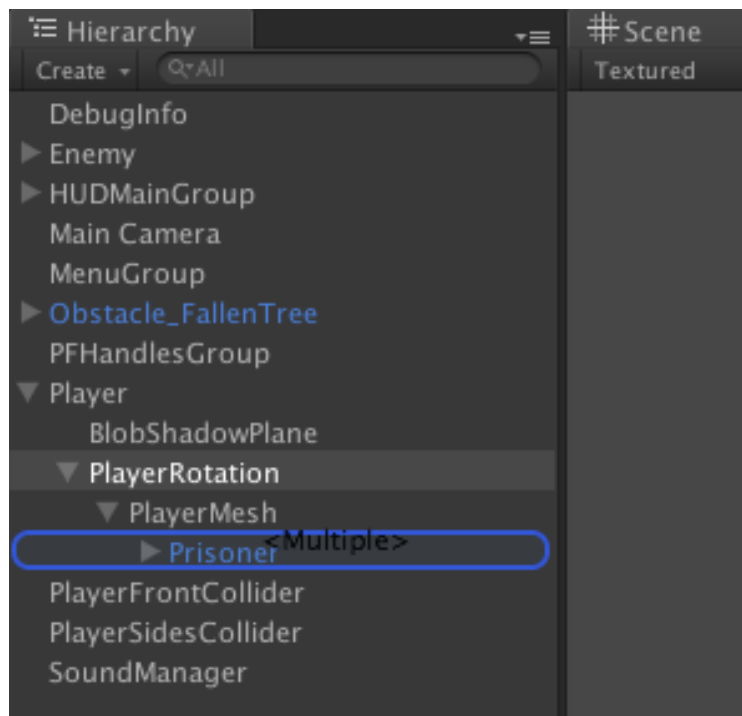
- Use Model > Scale Factor to adjust the scale.
- Use Model > Mesh Compression to compress the imported asset as much as you can without compromising its appearance.

2. Define the animation frames.

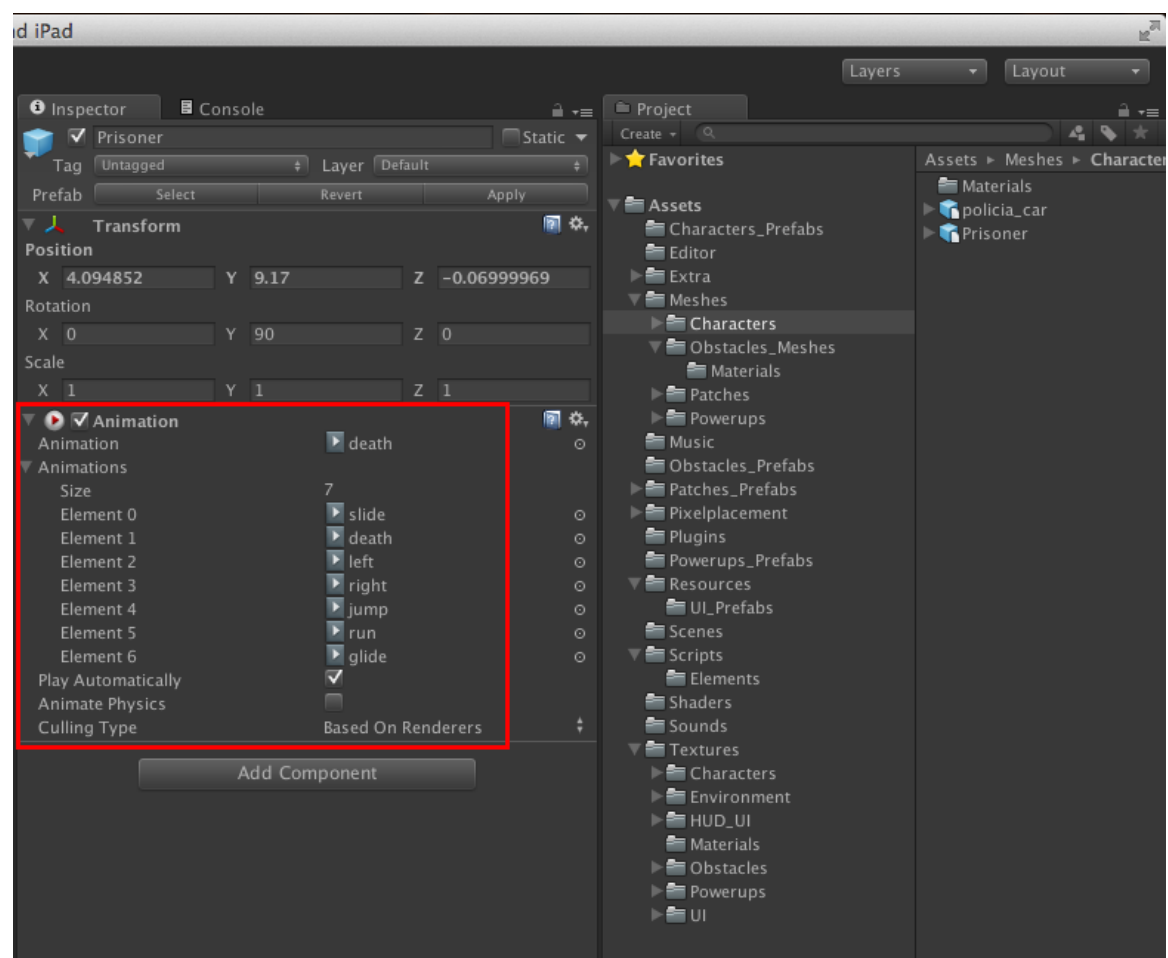


**Note:** The *ControllerScript.js* uses all the animations listed in the above screenshot. If a certain animation is not defined, an exception will be thrown when the script attempts to play that animation.

3. Drag the imported asset in Hierarchy > Player > PlayerRotation > PlayerMesh.



4. Make sure the *Animation* component of the character prefab has all the animations that will be used by the *ControllerScript.js*.



5. Change the name of the character to your player character in the *Start()* function of the *ControllerScript.js* to get the animations.

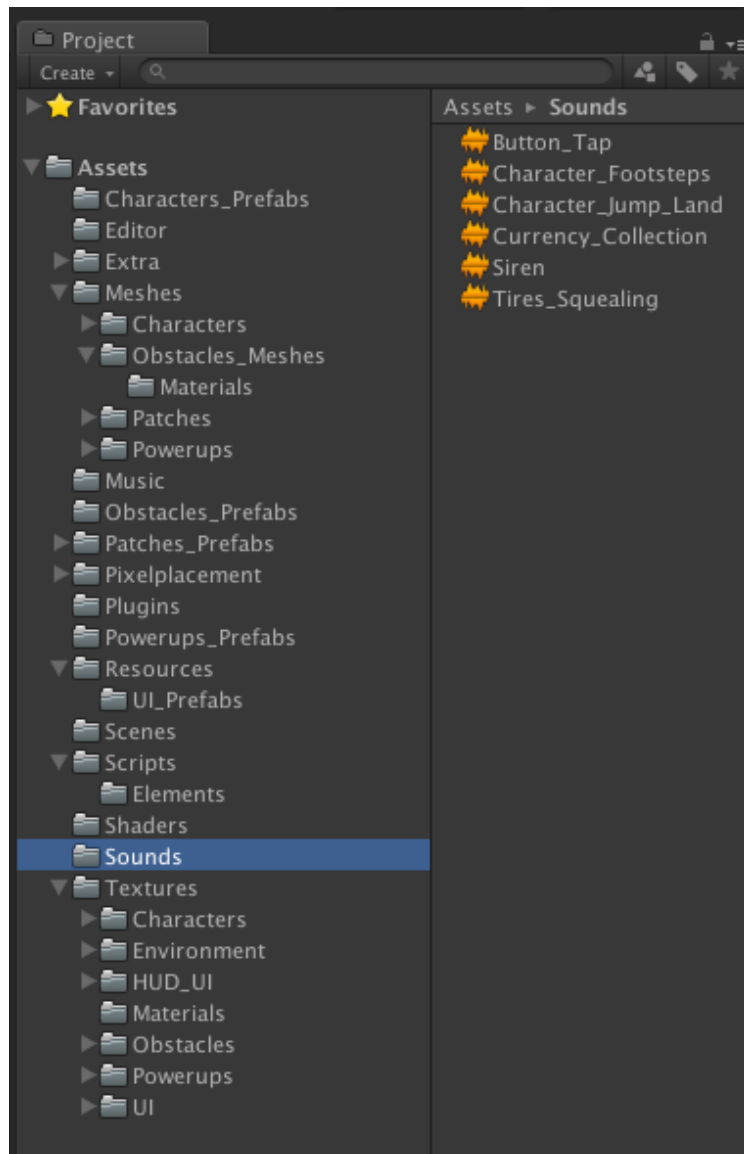
```
//get the animation component of the player character  
aPlayer = this.transform.Find("PlayerRotation/PlayerMesh/Prisoner").GetComponent(Anim  
tBlobShadowPlane = transform.Find("BlobShadowPlane");
```

## Adding a Sound

1. Go to Project > Scripts > SoundManager.
2. Add the sound's name in the relevant *enum*. Lets take the Siren sound played by the police car as an example.

```
1  #pragma strict  
2  
3  var asCharacterSounds : AudioSource[]  
4  var asPowerupSounds : AudioSource[];  
5  var asMenuSounds : AudioSource[];  
6  var asMusic : AudioSource[];  
7  var asEnemySounds : AudioSource[];  
8  
9  //Music and Sound Enums  
10 enum CharacterSounds  
11 {  
12     Footsteps = 0,  
13     JumpLand = 1  
14 }  
15  
16 enum PowerupSounds  
17 {  
18     CurrencyCollection = 0,  
19     PowerupCollection = 1  
20 }  
21  
22 enum MenuSounds  
23 {  
24     ButtonTap = 0  
25 }  
26  
27 enum EnemySounds  
28 {  
29     TiresSqueal = 0,  
30     Siren = 1  
31 }  
32
```

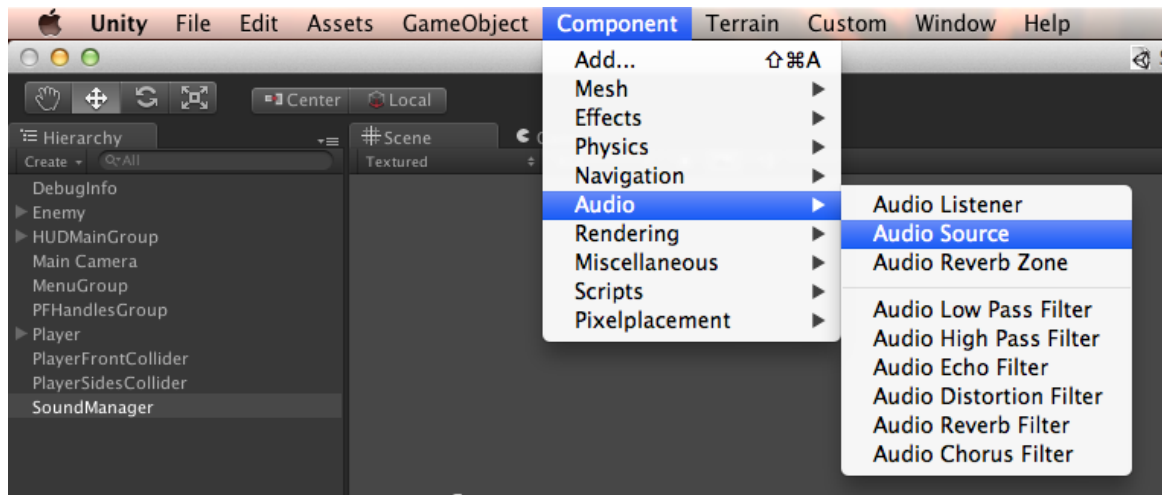
3. Import the sound. (Drag and drop the sound from the Finder/ Window to Project > Assets > Sounds)



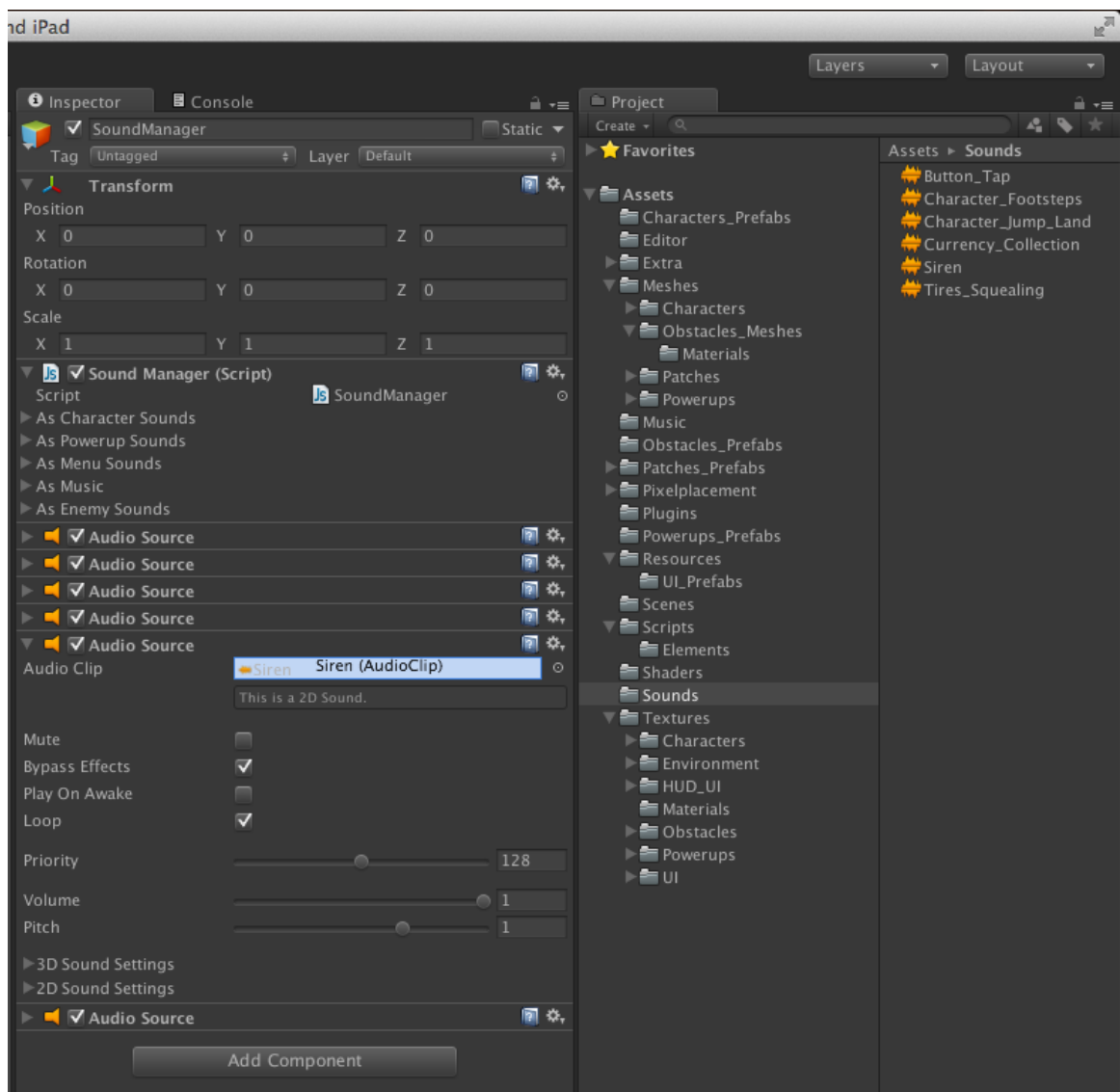
### Optimization Notes:

- Change the *Audio Format* and *Load type* as needed.
- Uncheck *3D Sound* as the project by default doesn't use 3D sounds.
- 4. Select Hierarchy > SoundManager.
- 5. Select Component > Audio > Audio Source. (This will add another Audio Source to your *SoundManager* prefab to be used later)

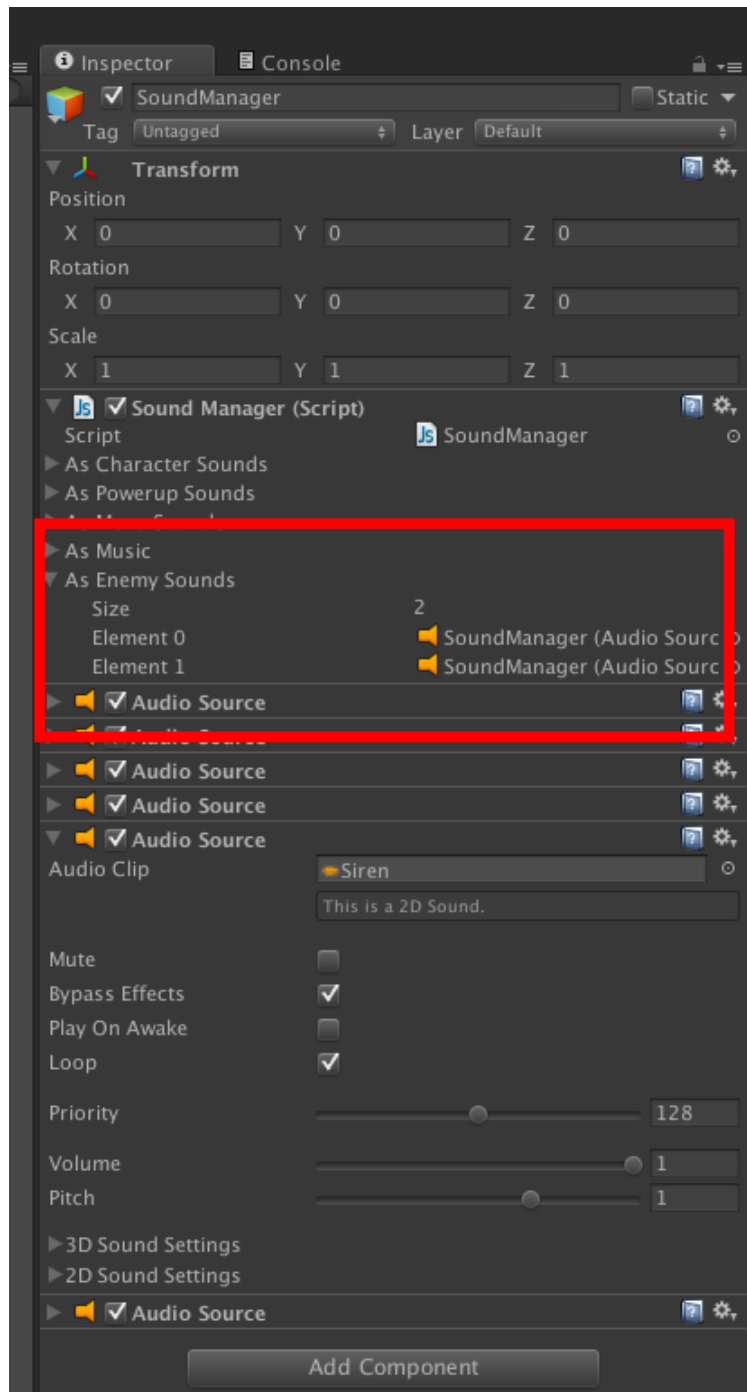




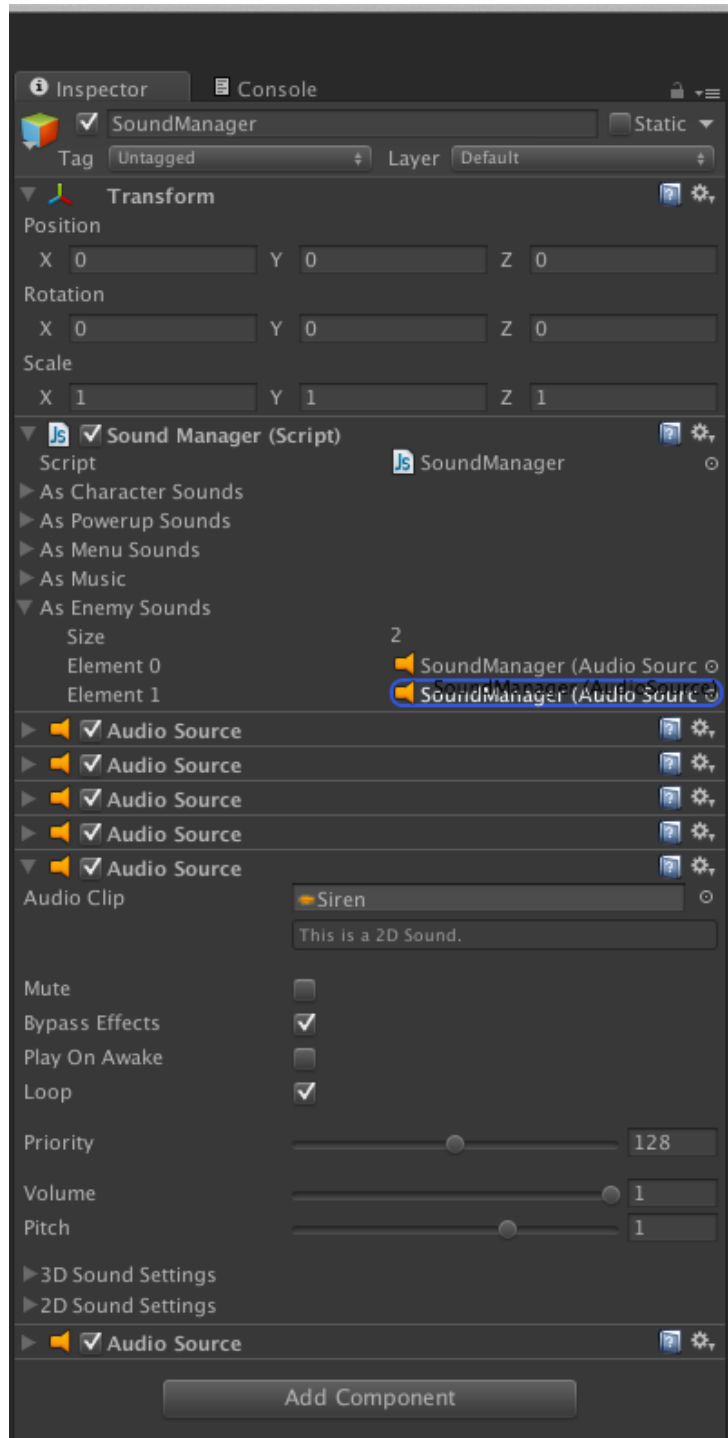
6. Drag the sound from Project > Sounds to Audio Source > Audio Clip of the newly created Audio Source Component.



7. Increase the array size of the relevant array.



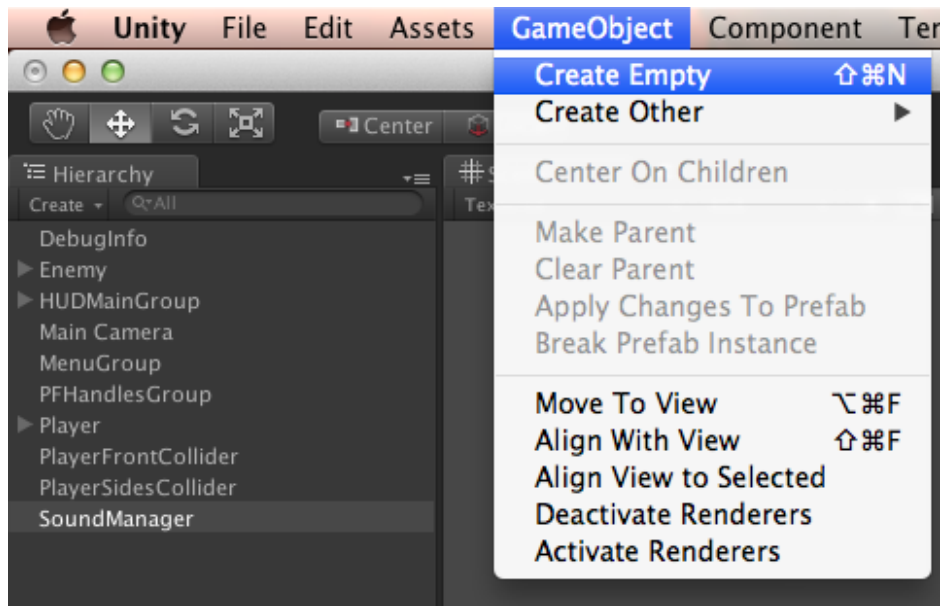
8. Drag the Audio Source and drop it as the last index of the array.



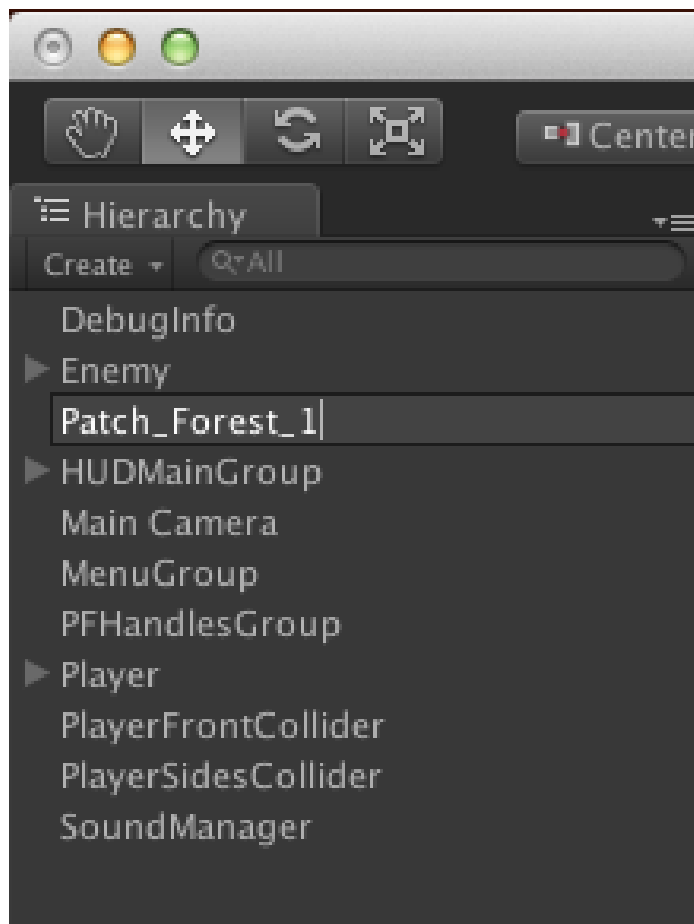
9. Use the “playSound(…)” function from SoundManager in the scripts where you need to use the newly added sound.

## Adding a Patch

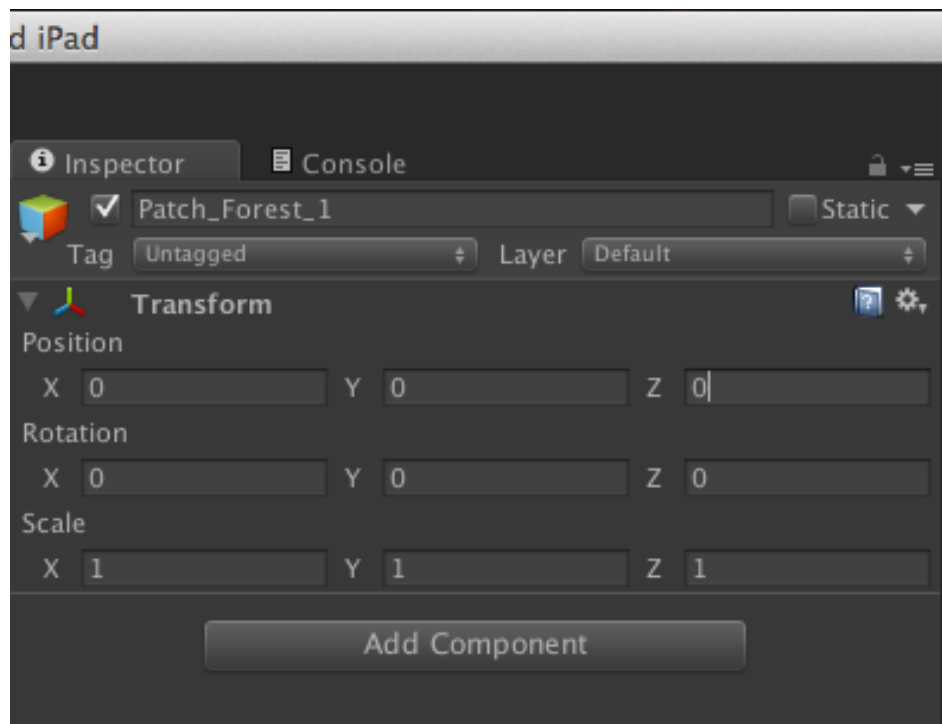
1. In the *Hierarchy* tab, add a new game object by selecting GameObject > CreateEmpty.



2. Name the empty game object.

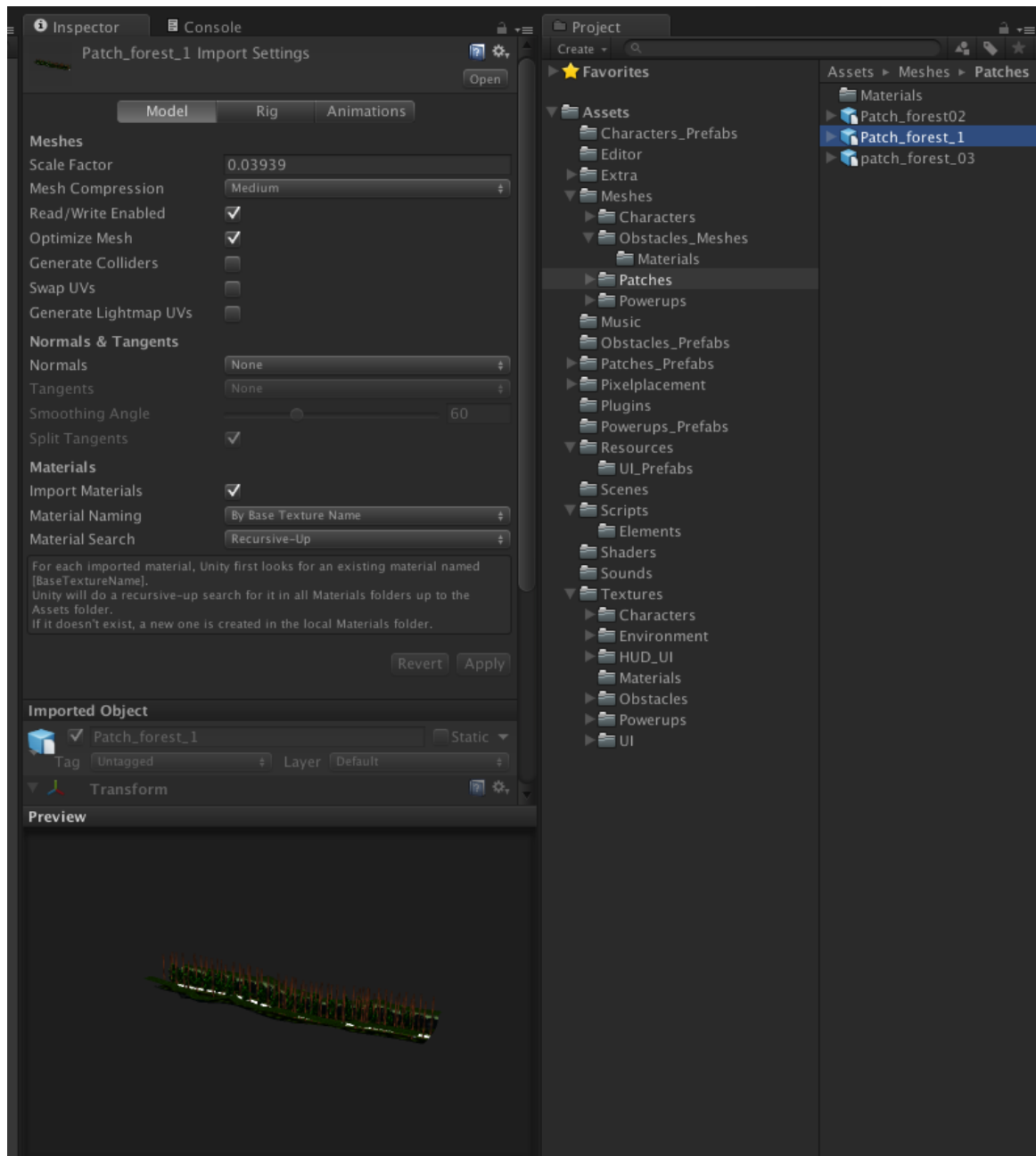


3. Set the game object position and rotation transforms to zero.



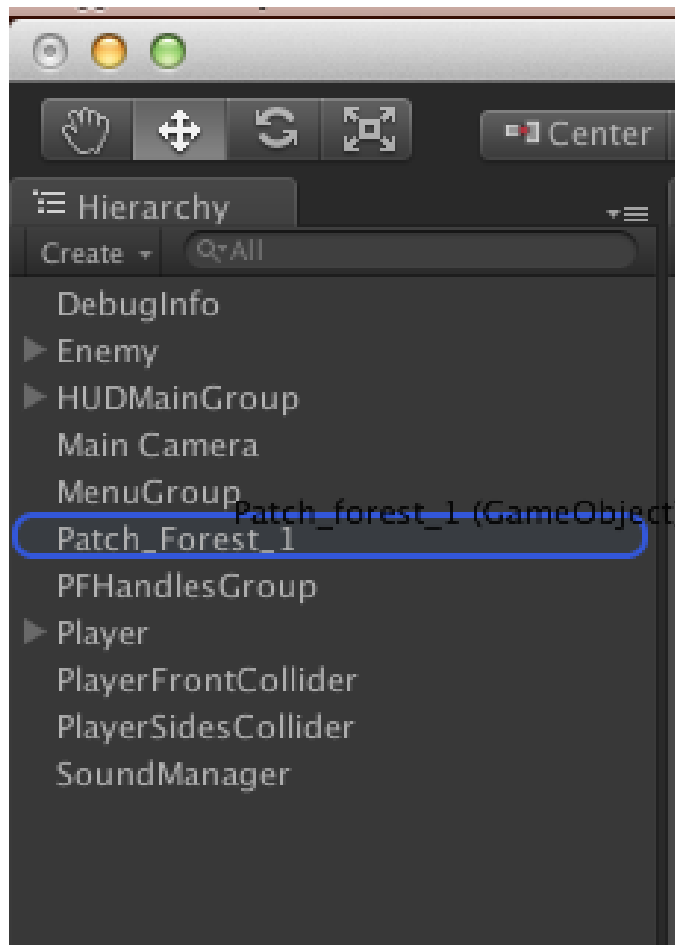
4. Import the patch .fbx.



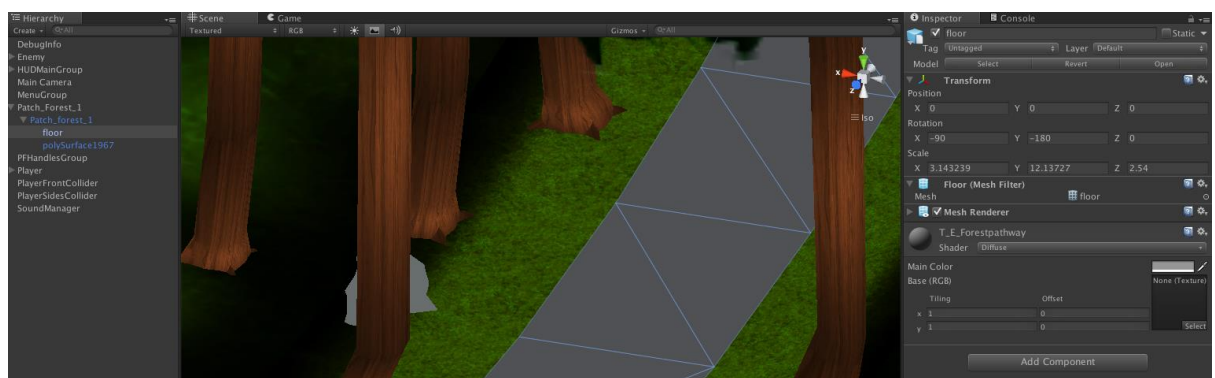


### Optimization Notes:

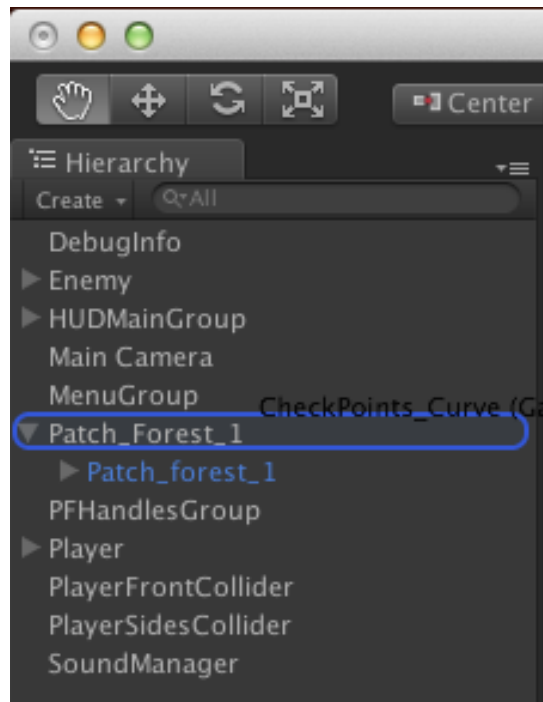
- Use Model > Scale Factor to adjust the scale.
  - Use Model > Mesh Compression to compress the imported asset as much as you can without compromising its appearance.
  - Select "None" in Rig > Animation Type if your asset isn't rigged.
  - Uncheck the Import Animation option in Animations > Import Animation if your asset isn't animated.
5. Drag the imported .fbx and drop it in the created empty game object.



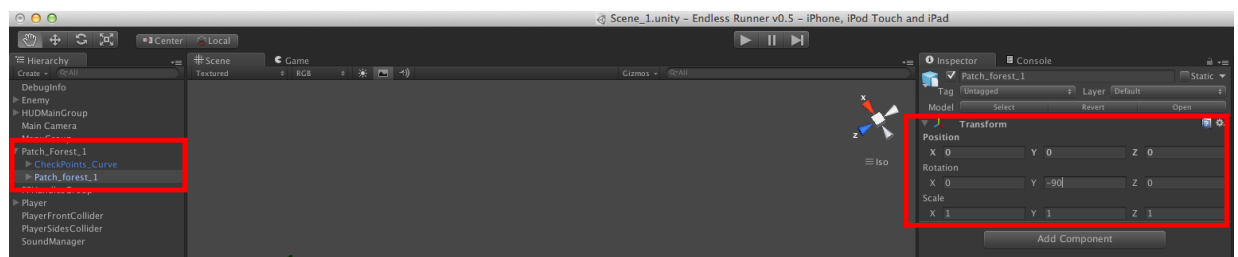
6. Select the floor of your patch where the character will run and perform two steps:
  - a. Select Component > Physics > Mesh Collider.
  - b. Select Layer > Terrain lyr.



7. Select Project > Patches Prefabs > Sample CPs.
8. Drag the “CheckPoints\_Straight” prefab into the patch prefab if your patch’s path is in a straight line. If the path is not in a straight line, drag the “CheckPoints\_Curve” instead.



9. Rotate the patch if it does not have the same orientation as the spline.



### Adjusting CPs of a Straight Patch

- Adjust the Model > Scale Factor of your .fbx and make sure the patch is exactly the size of the spline.

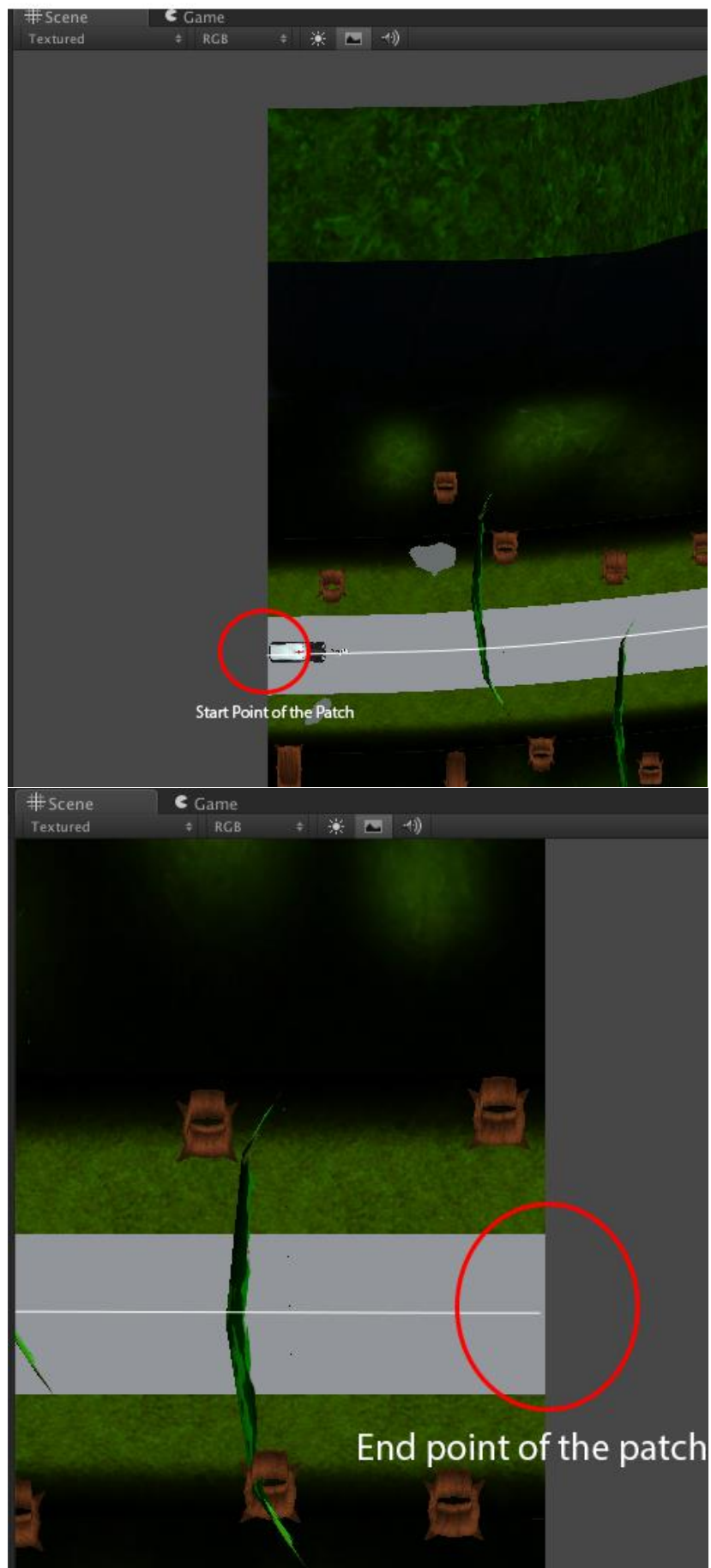
**Note:** Make sure the spline is centered on the path.

- Select Hierarchy > \*patch name\*.
- Click CheckPoints\_Straight.
- From the title bar, select Custom > Patch CP Generator. (This saves all the changes you have made to the spline)
- Select the \*patch name\*.
- Drag the patch to Project > Patches Prefabs to save the created prefab.

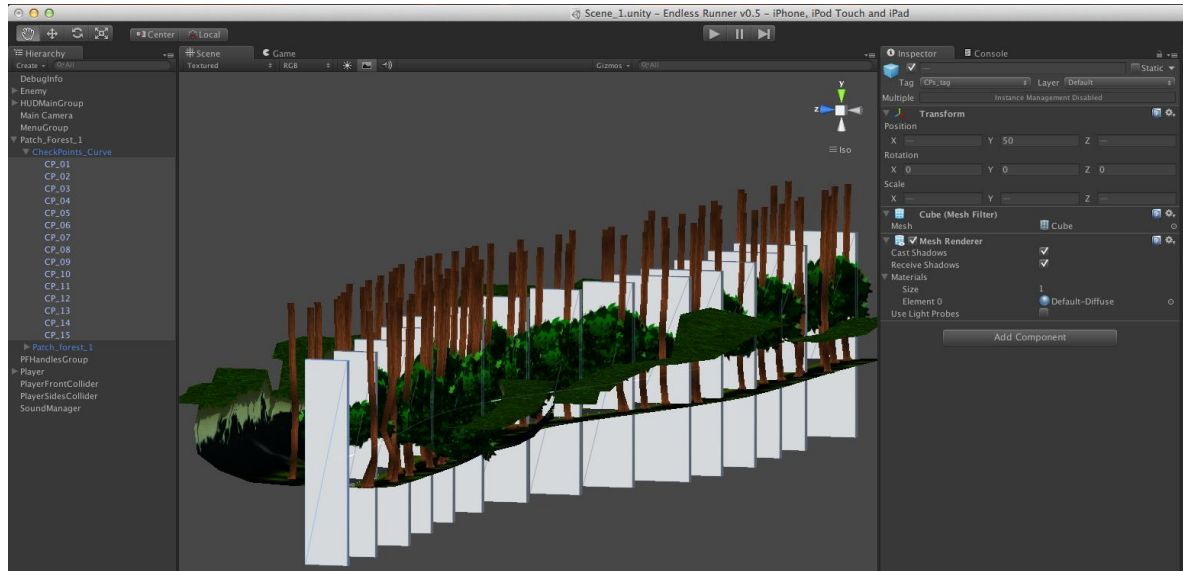
### Adjusting CPs of a Curved Patch

- Adjust the Model > Scale Factor of your .fbx and make sure the patch is exactly the size of the spline.

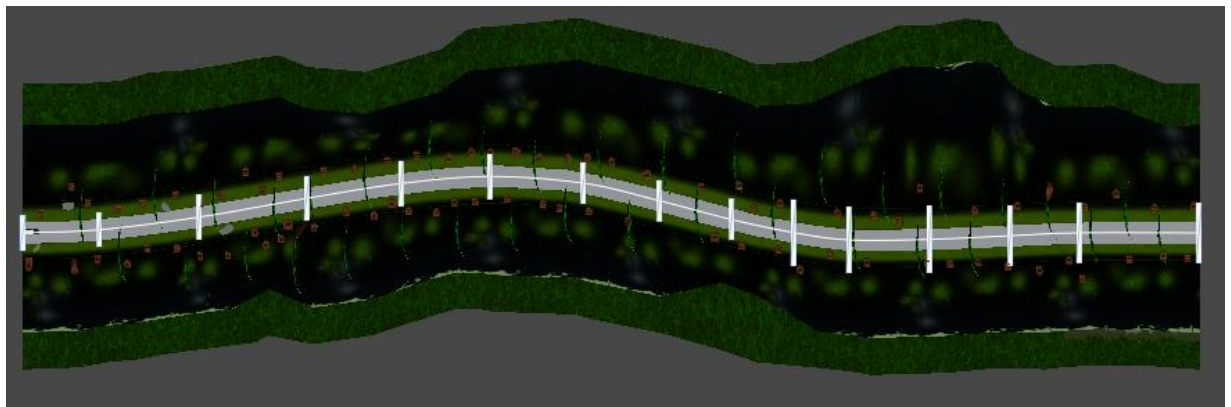
**Note:** The start and end points of the spline should be centered on the start and end of the path.



- b. Select Hierarchy > *\*patch name\** > CheckPoints Curve.
- c. Select all the CPs.
- d. Check the *Mesh Renderer* component from Inspector > Mesh Renderer.  
(This will display all the Check Points that shape the spline)



- e. Adjust the CPs so that the spline is centered on the path at every point on that path. You can only change the position of the CPs; changing their rotation or scale will have no effect on the spline.

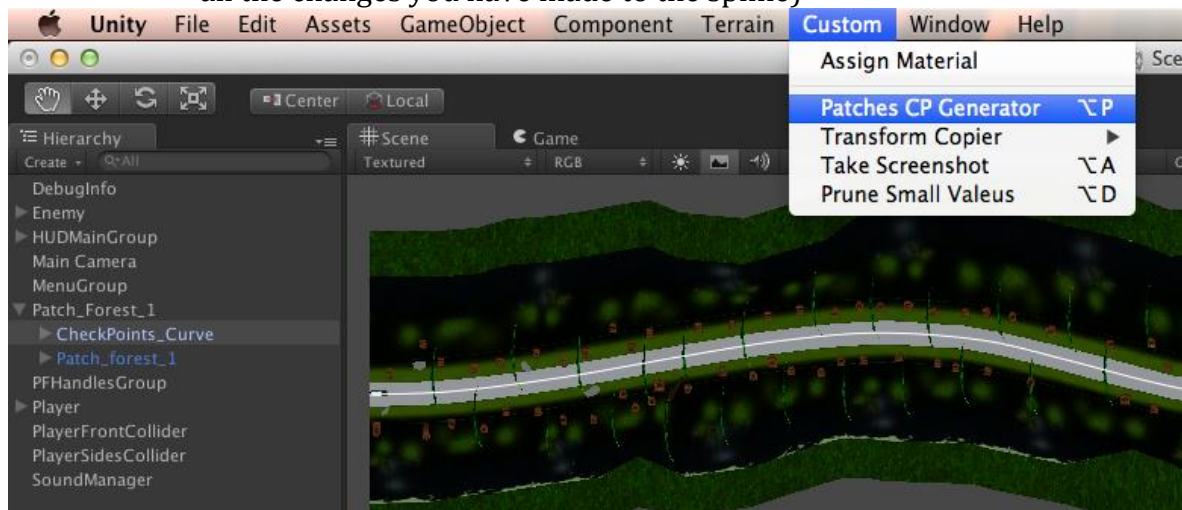


### Notes:

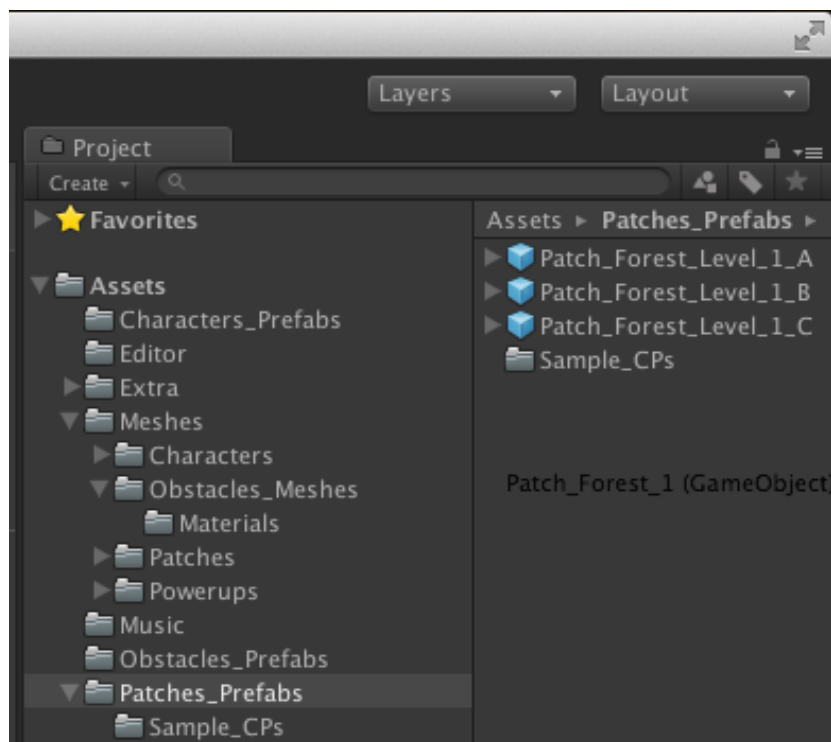
- Make sure your Editor's camera is in Isometric mode. (You can change the camera mode by clicking the cube in the top right corner of the screen.)
- Make sure the Editor is in top view.
- You can add or remove CPs as required but make sure not to miss a number in the sequence.
  - f. Check off the *Mesh Renderer* component from Inspector > Mesh Renderer when you're done adjusting the spline.
  - g. Click *CheckPoints\_Straight/ CheckPoints\_Curve* or whatever you have named the prefab containing the checkpoints.



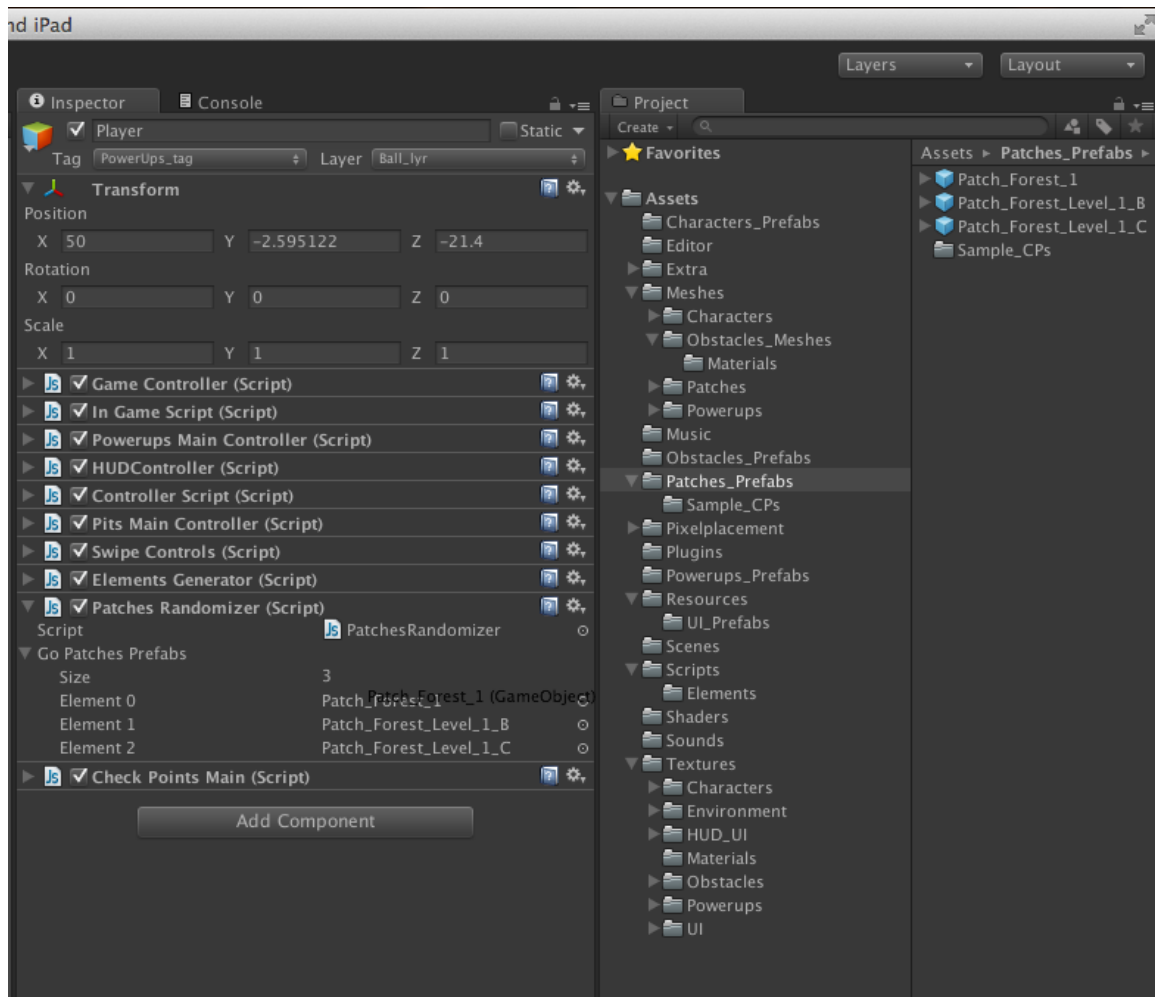
- h. From the title bar, select Custom > Patch CP Generator. (This saves all the changes you have made to the spline)



- i. Select the *\*patch name\**.
- j. Drag the patch to Project > Patches Prefabs to save the created prefab.



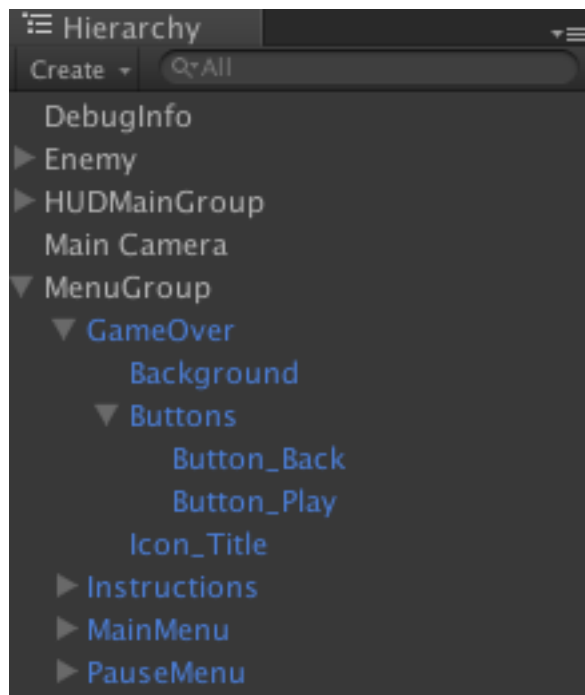
10. Once the patch has been saved, select Player > Patches Randomizer from the Hierarchy.
11. Increase the array size of "Go Patches Prefabs".
12. Drag and add the newly created patch from Project > Assets > Patches\_Prefabs, to the *Go Patches Prefabs* array.



13. Remove the patch from the *Hierarchy* tab.

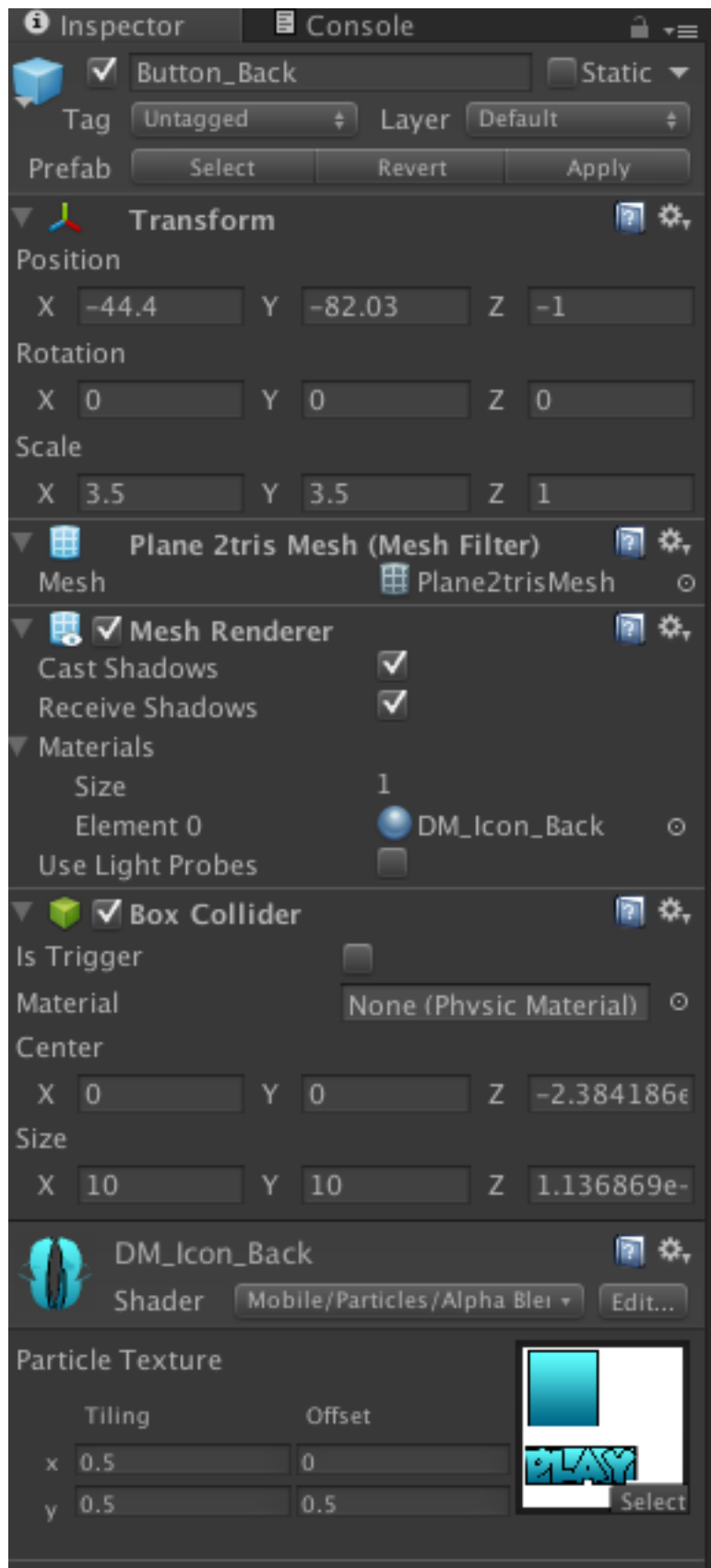
## Adding a Menu

The *MenuGroup* prefab in the Hierarchy tab is the parent of all the menu prefabs used. The menus are present in the scene at all times. They are displayed by bringing them in front of the HUD Camera by the *MenuScript.js*. The project uses the low-poly plane named *Plane2tirsMesh* located at [Project > Assets > Editor](#). As always, the lower the poly-count, the better the performance.



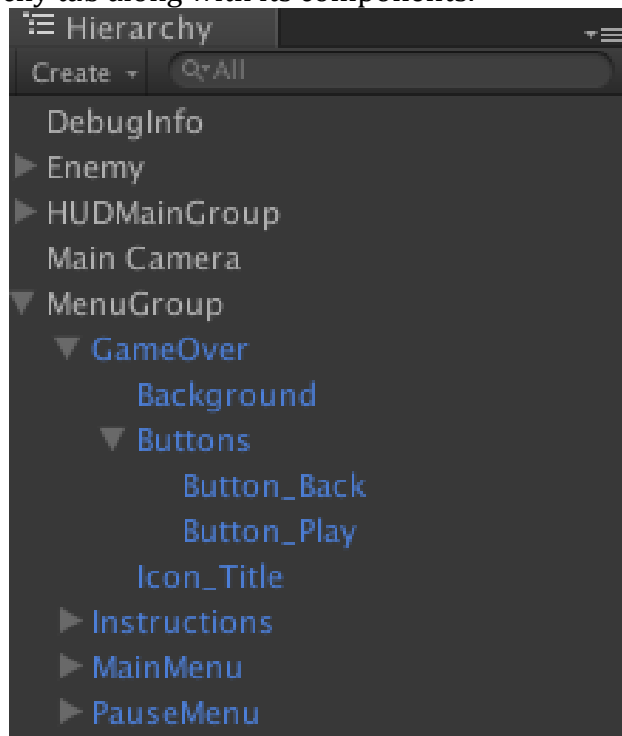
By convention in the project, an empty GameObject is created to contain the backgrounds, icons and buttons (*GameOver* is used as an example in the above screen shot). All of the buttons are enclosed in yet another empty GameObject named *Buttons*. Each button is a plane that also has a box collider as a component to detect raycasts (raycasts are used to detect taps on buttons).

In the following example the button named *Button\_Back* is shown with all its components. Notice the low-poly plane used instead of the default plane used by Unity. Secondly, observe that a box collider has been attached to the button to detect raycasts. A box collider can be added by selecting the plane and then selecting Component > Physics > Box Collider from the title bar.

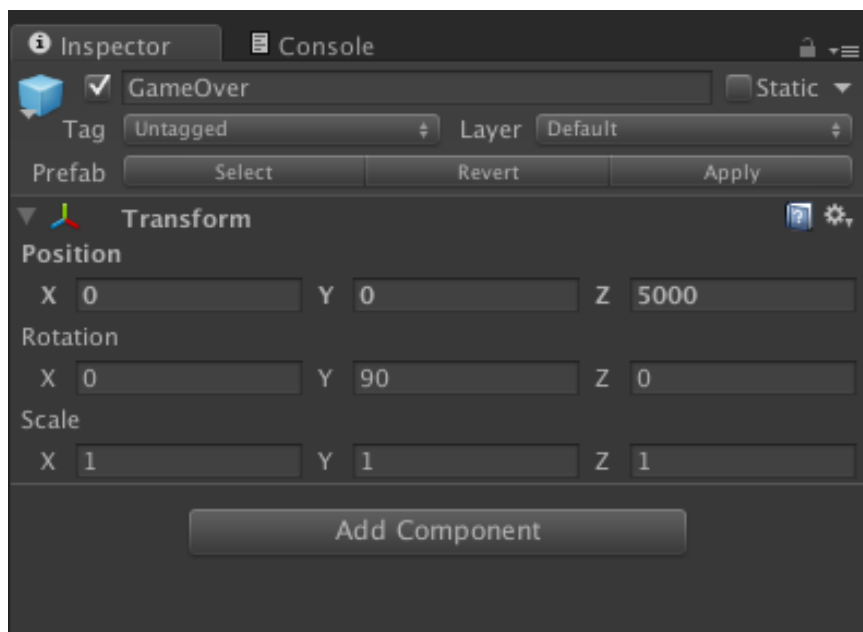


1. As mentioned before, make sure all components of the menu are enclosed in a single game object. (For the sake of example, let's call the menu "GameOver")

2. Place the *GameOver* prefab in the *MenuGroup* prefab located in the Hierarchy tab along with its components.

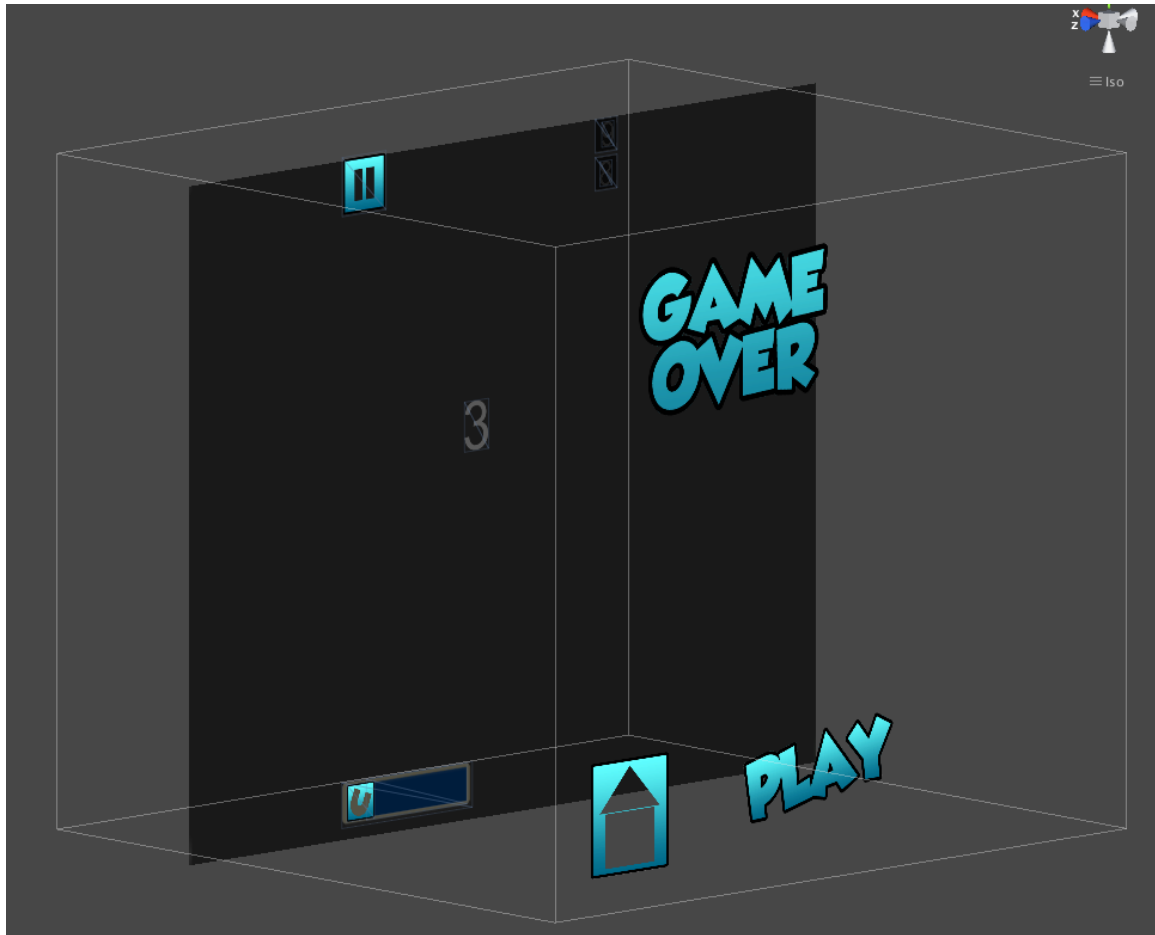


3. Set the position of the *GameOver* prefab at 0,0,5000. (This brings the prefab into the HUD Camera as the HUD Camera is located at (0,0,5000))



4. Double click on Hierarchy > HUDMainGroup to display it in the Editor and make sure the menu items are facing towards the negative x-axis. The screen shot shows the menu title, the buttons, the semi-transparent background and also the HUD elements in the background (ignore these while designing a menu).





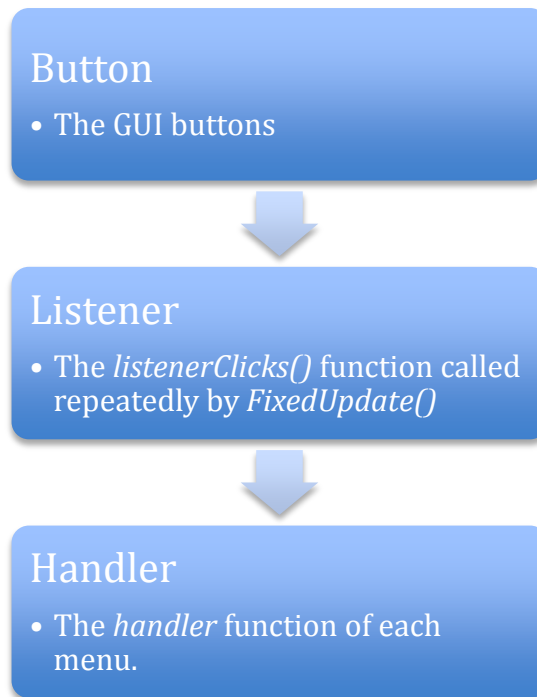
5. Open the *MenuScript.js* from Project > Assets > Scripts.

### **Menu Script Architecture:**

The architecture of the menu controller, which is the *MenuScript.js* script, relies on a three-tier architecture. The first layer consists of the buttons that in-turn consist of a plane, a material and a box collider.

The second layer is the listener that consists of a function named *listenerClicks()* in the *MenuScript.js*. This function is called by the *FixedUpdate()* function to listen for any interaction with the screen. If the user taps on the screen, the camera raycasts the menu in search of the button tapped. If the raycast finds a collider in its way, a relevant handler is called. By relevant, we mean the handler of the active menu.

Finally, the third layer consists of handler functions. Each menu has a single handler function that defines the implementation of all the buttons the menu has. When the listener function calls the handler function, it also passes the transform of the button tapped as a parameter. The handler function uses this transform and compares it with the array of buttons' transforms, to check which button's implementation to execute.



6. Add the name of the newly created menu in the *MenuIDs* enum.

```
15  
16 public enum MenuIDs  
17 {  
18     MainMenu = 0,  
19     PauseMenu = 1,  
20     GameOverMenu = 2  
21 }
```

7. Create a Transform type array to store the transforms of the buttons in the GameOver menu.
8. Create an integer type variable to store the number of button in the GameOver menu.

```
69 //Game Over Menu  
70 private var tGameOverButtons:Transform[];  
71 private var iGameOverButtonsCount:int = 2;
```

9. Store the *GameOver* menu's transform the *tMenuTransform* array.
10. Initialize the transform type array that holds the references of the buttons in the *GameOver* menu.

```
//game over menu initialization  
tMenuTransforms[MenuIDs.GameOverMenu] = tMenuGroup.Find("GameOver").GetComponent(Transform) as Transform;  
tGameOverButtons = new Transform[iGameOverButtonsCount];  
tGameOverButtons[0] = tMenuTransforms[MenuIDs.GameOverMenu].Find("Buttons/Button_Back");  
tGameOverButtons[1] = tMenuTransforms[MenuIDs.GameOverMenu].Find("Buttons/Button_Play");
```

11. Create a handler function for *GameOver* menu to define the implementation of its buttons. In our case, we have two buttons.

```
private function handlerGameOverMenu(buttonTransform : Transform)
{
    if (tGameOverButtons[0] == buttonTransform)//main menu button
    {
        hInGameScript.procesClicksDeathMenu(GameOverMenuEvents.Back);
        CloseMenu(MenuIDs.GameOverMenu);
        ShowMenu(MenuIDs.MainMenu);
        CurrentMenu = MenuIDs.MainMenu;
    }
    else if (tGameOverButtons[1] == buttonTransform)//play button
    {
        hInGameScript.procesClicksDeathMenu(GameOverMenuEvents.Play);
        CloseMenu(CurrentMenu);
    }
}
```

12. Add a new case in the *listenerClicks* function. This function will call the *GameOver* handler function if its button is pressed.

```
else if (CurrentMenu == MenuIDs.GameOverMenu)
    handlerGameOverMenu(hit.transform);
```

13. Add a new case in the *ShowMenu(...)* function to display the *GameOver* menu prefab when needed. Setting the prefab's y position to zero brings it in front of the HUD Camera.

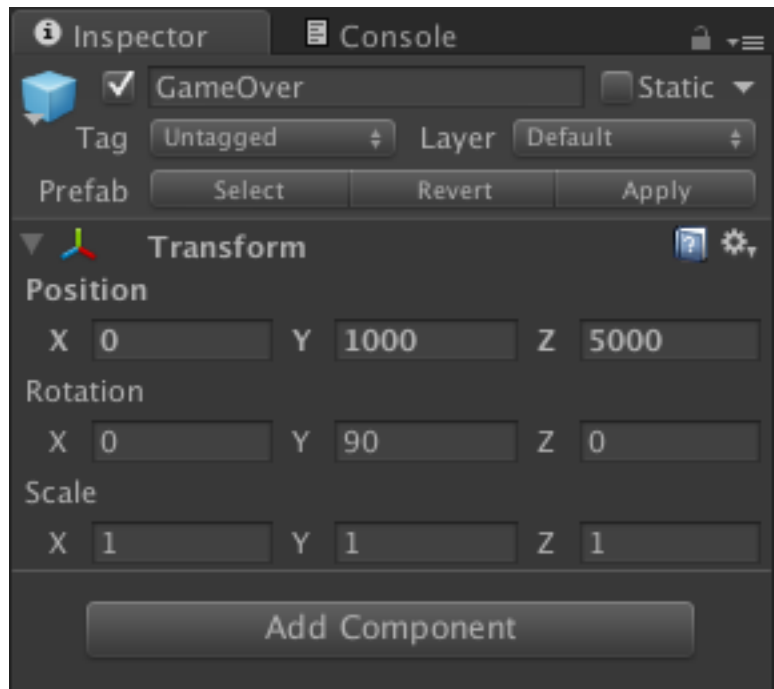
```
else if (MenuIDs.GameOverMenu == index)
{
    tMenuTransforms[MenuIDs.GameOverMenu].position.y = 0;
}
```

14. Add a case and its implementation in the *CloseMenu(...)* function as shown below. Setting the prefab's y position to 1000 moves it away from the HUD Camera.

```
else if (index == MenuIDs.GameOverMenu)
{
    tMenuTransforms[MenuIDs.GameOverMenu].position.y = 1000;
}
```

**Note:** Whenever you need to use the created menu, call the *ShowMenu(...)* function passing the appropriate parameter. Similarly, to close the menu, call the *CloseMenu(...)* function with the same parameter.

15. Set the position of the newly created *GameOver* menu prefab to (0,1000,5000). This will remove it from the HUD Camera in default conditions.

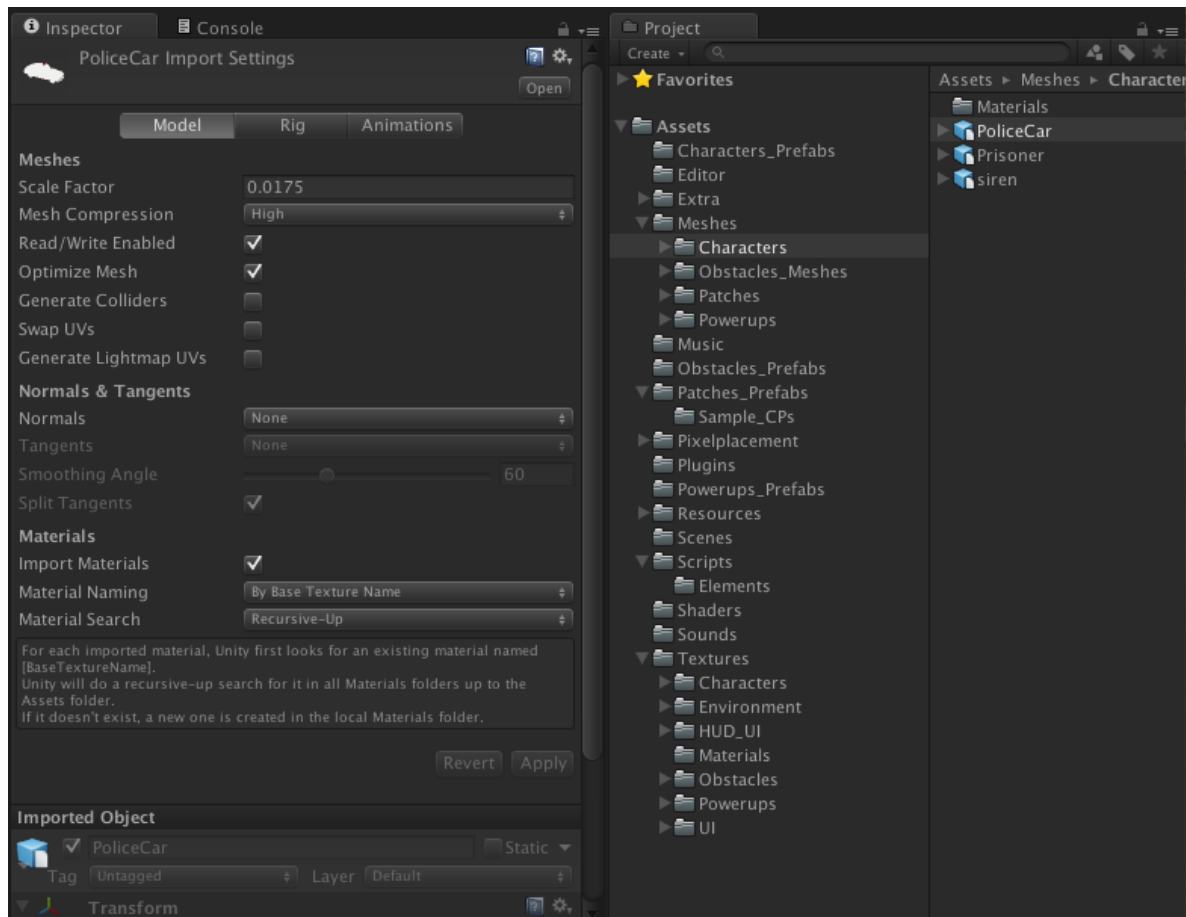


## Replacing the Enemy Character

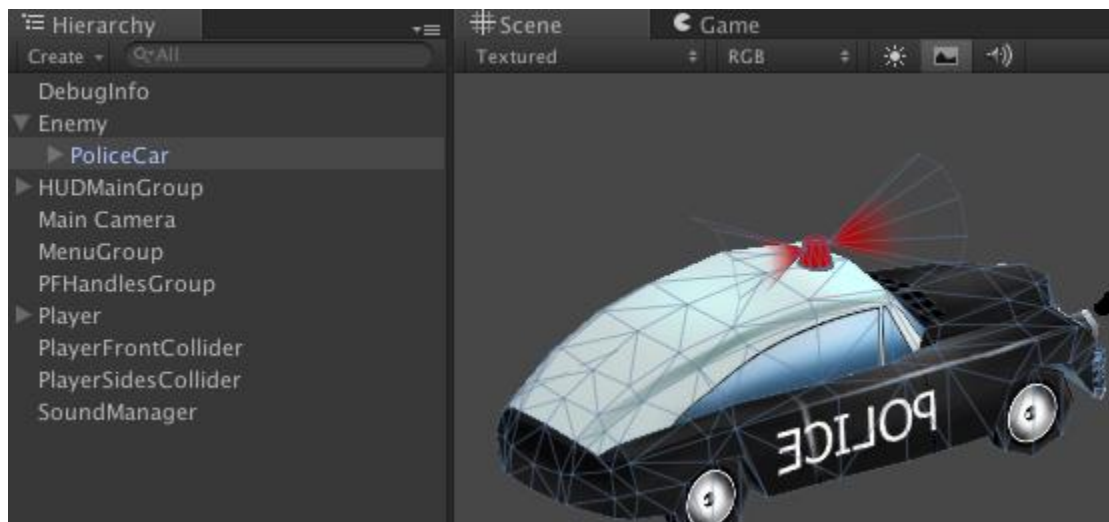
1. Import the player .fbx that will be used as the main enemy.

### Optimization Notes:

- Use Model > Scale Factor to adjust the scale.
- Use Model > Mesh Compression to compress the imported asset as much as you can without compromising its appearance.
- Select "None" in Rig > Animation Type if your asset isn't rigged.
- Uncheck the Import Animation option in Animations > Import Animation if your asset isn't animated.



2. Drag the imported asset in Hierarchy > Enemy.



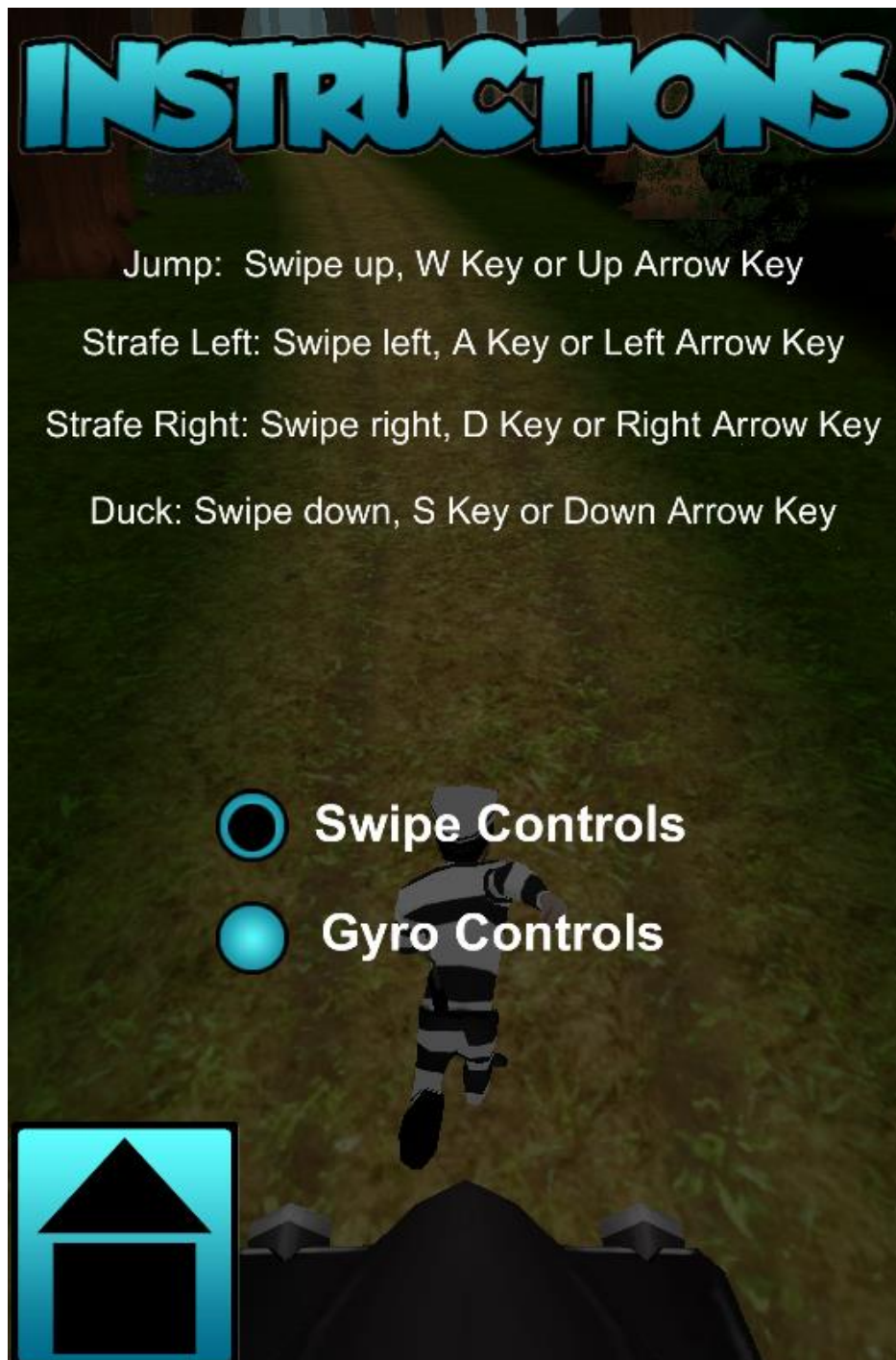
3. Change the implementation in the Project > Assets > Scripts > *EnemyController.js* as needed.

## How to Switch between Gyro and Swipe Controls

The type of controls can be switched from the information menu. Clicking the following link located on the main menu accesses the instructions menu:



The radio buttons in the in the instructions menu can be used to toggle between swipe and gyro controls.

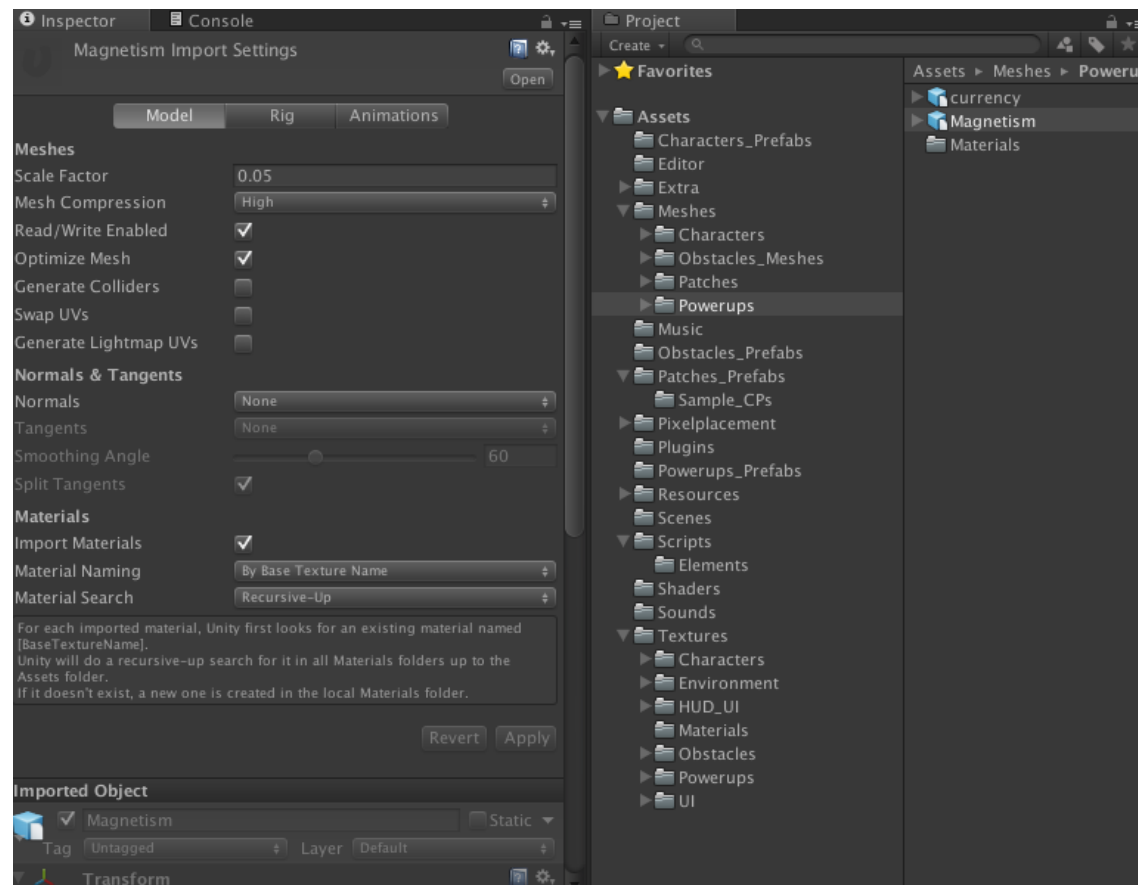


## Adding a Power-up

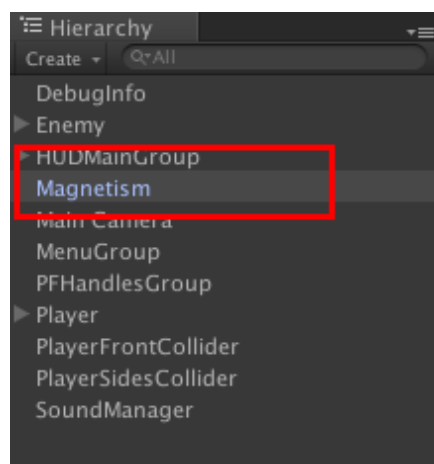
1. Import the .fbx of the asset that you will use as an obstacle.

### Optimization Notes:

- Use Model > Scale Factor to adjust the scale.
- Use Model > Mesh Compression to compress the imported asset as much as you can without compromising its appearance.
- Select “None” in Rig > Animation Type if your asset isn’t rigged.
- Uncheck the Import Animation option in Animations > Import Animation if your asset isn’t animated.

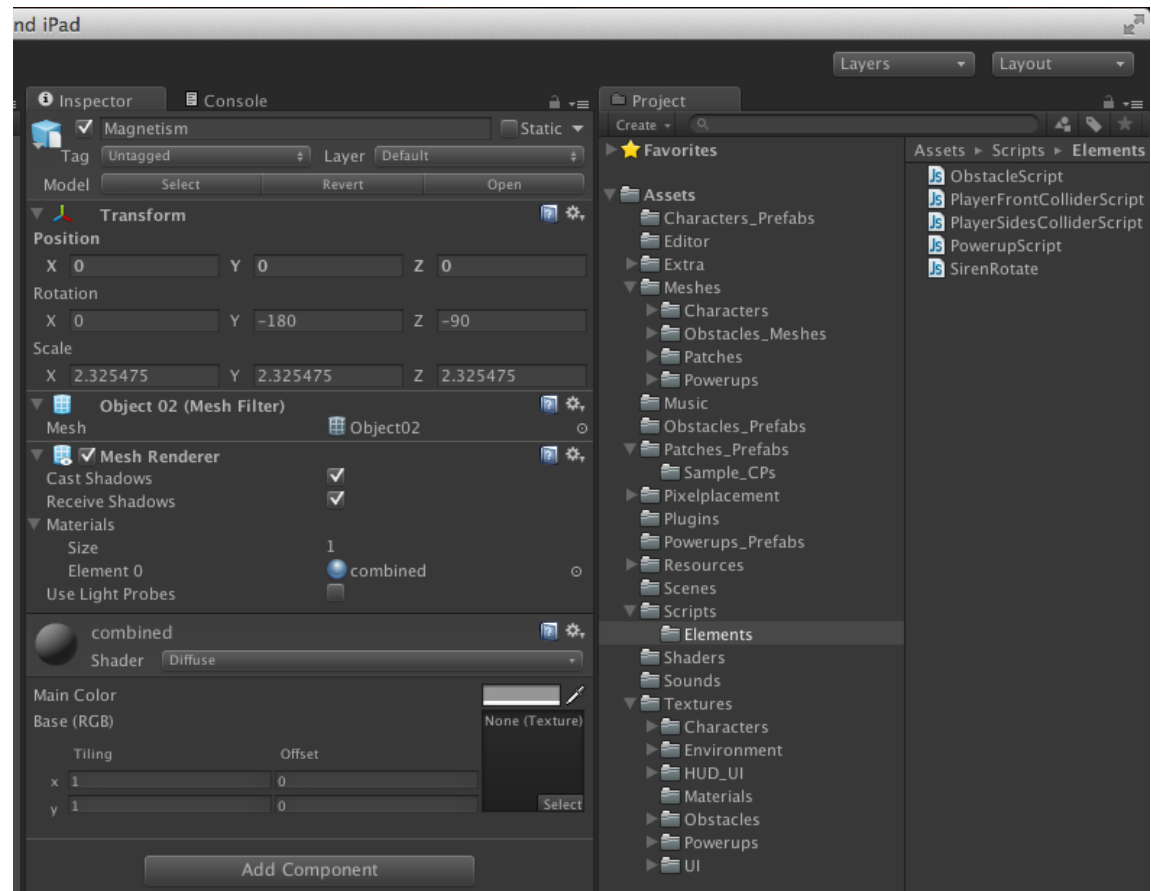


2. Drag the imported asset in “Hierarchy” tab to create a prefab.

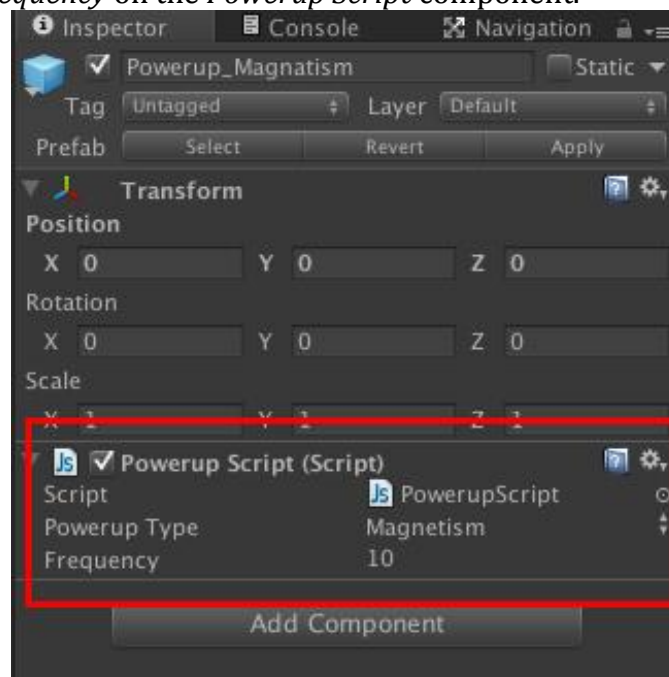




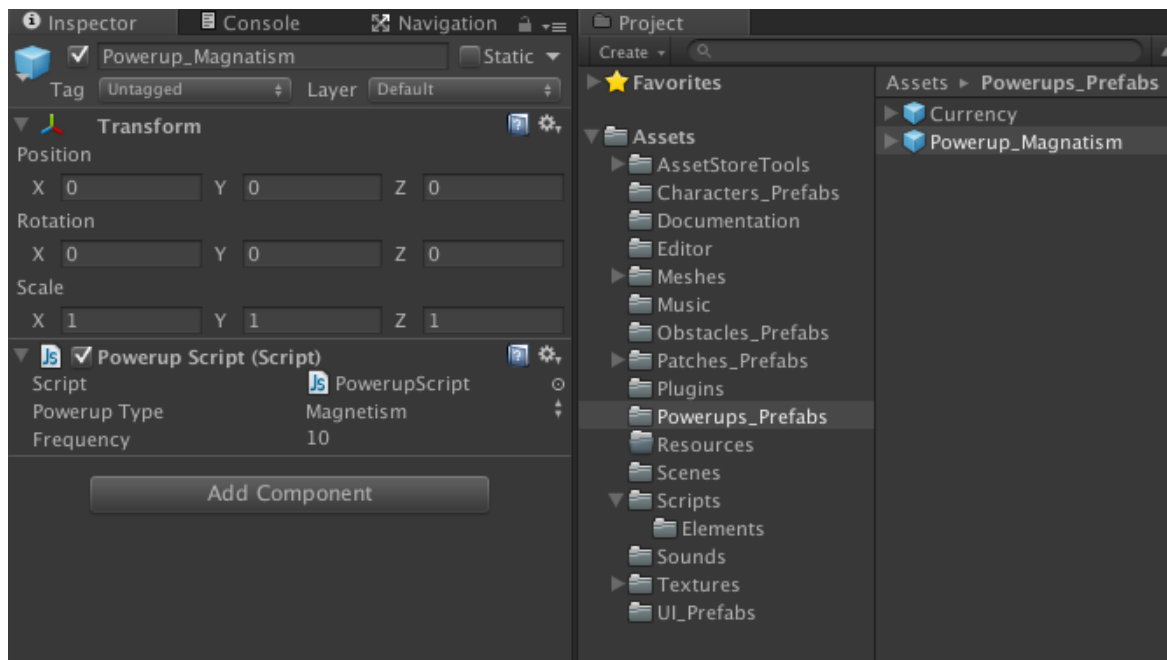
3. Go to Project > Assets > Scripts > Elements.



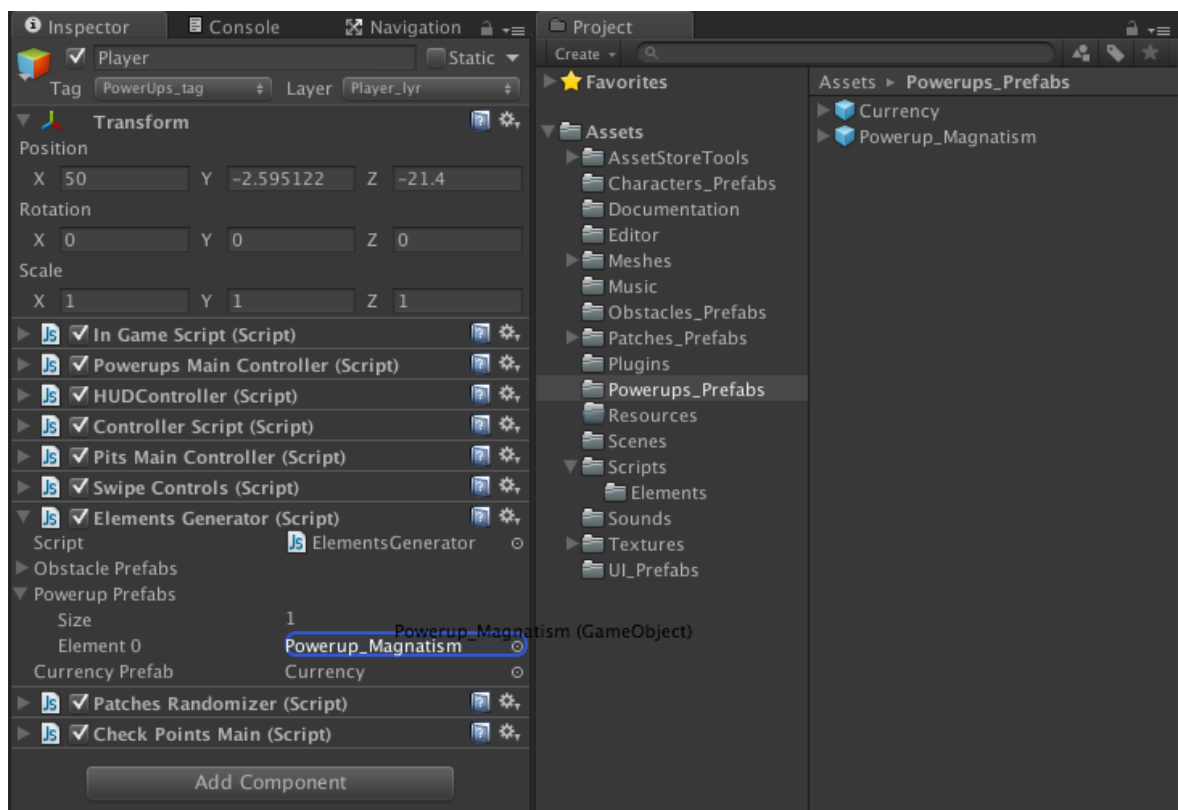
4. Add the "ObstacleScript.js" on your obstacle's prefab.
5. Set the *Powerup Type* on the *Powerup Script* component.
6. Set the *Frequency* on the *Powerup Script* component.



7. Drag and drop your newly created power-up prefab to Project > Powerups Prefabs.



8. Now, we have to tell the *ElementGenerator* script about our newly created power-up so that it will generate the power-up on the path during gameplay. To do that, simply drag and drop the saved prefab to the *ElementGenerator.js* script that's a part of the *Player* prefab. The prefab has to drop in the *Powerup Prefabs* hierarchy to take effect.



9. Finally delete the power-up prefab from the Hierarchy tab. It does not belong in the scene by default because it is generated programmatically during runtime.