

# Unity Game Engine Practical Activity

## Student Workbook

**Workbook Title:**

Introduction to Programming Animated Characters in the Unity Game Engine.

**Workbook Description:**

In this activity, you will learn the basics of animated 3D models in Unity, and you will learn how to control animated 3D models using C# code. You will also learn further programming in the Unity games Engine and be able to prototype simple game elements.

**Please Read:**

- This student workbook will check that you have understood and can apply the knowledge you have gained during this week's talks.
- Please read each workbook section carefully and when required please type out all code. Attempting to copy and paste code *may* cause errors during compilation. Also writing out the code helps you understand how to program C# for Unity.

**Document Version:**

V4.0.

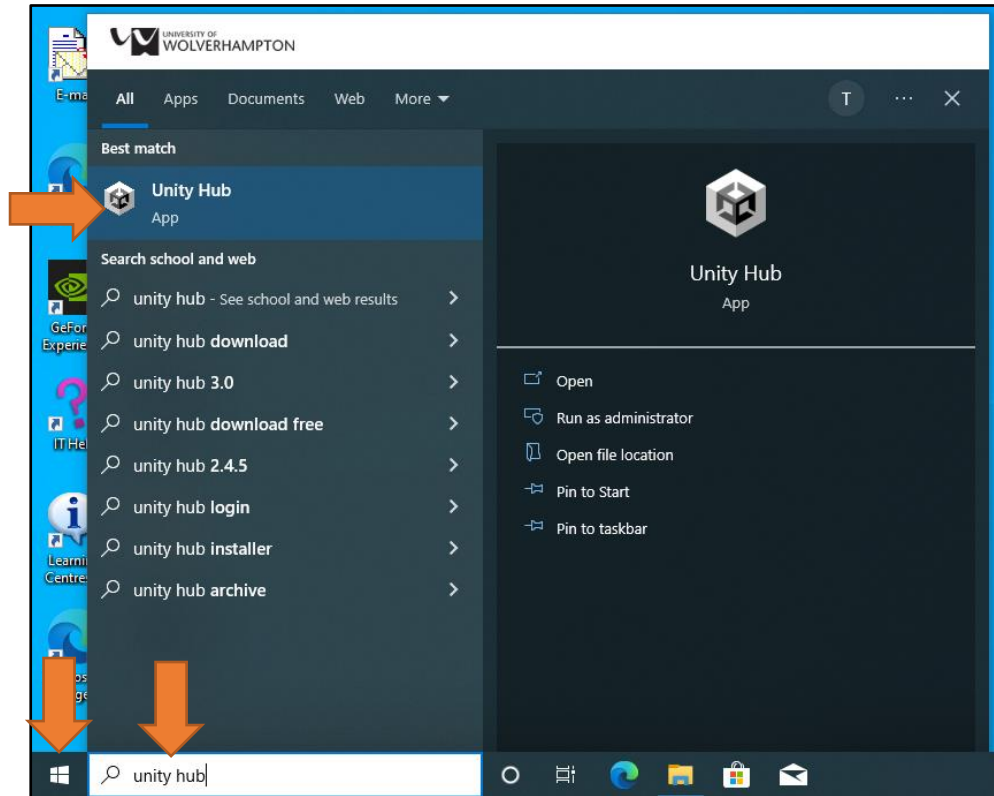
**Game Engine Version:**

This student workbook was checked using **Unity 2022.--f1**.

# 1. Start the Unity Hub and Create a New Project

Start the **Unity Hub** by clicking on the “Unity Hub” shortcut in the Windows start menu.

Do this by clicking the start menu and typing **Unity Hub** into the search box. Click on the Unity Hub icon. Below is a screenshot of the windows start menu and the “Unity Hub” shortcut icon.



**Image description:** The windows start menu and the “Unity Hub” shortcut. Click the start menu and type Unity Hub into the search box. Click on the Unity Hub icon.

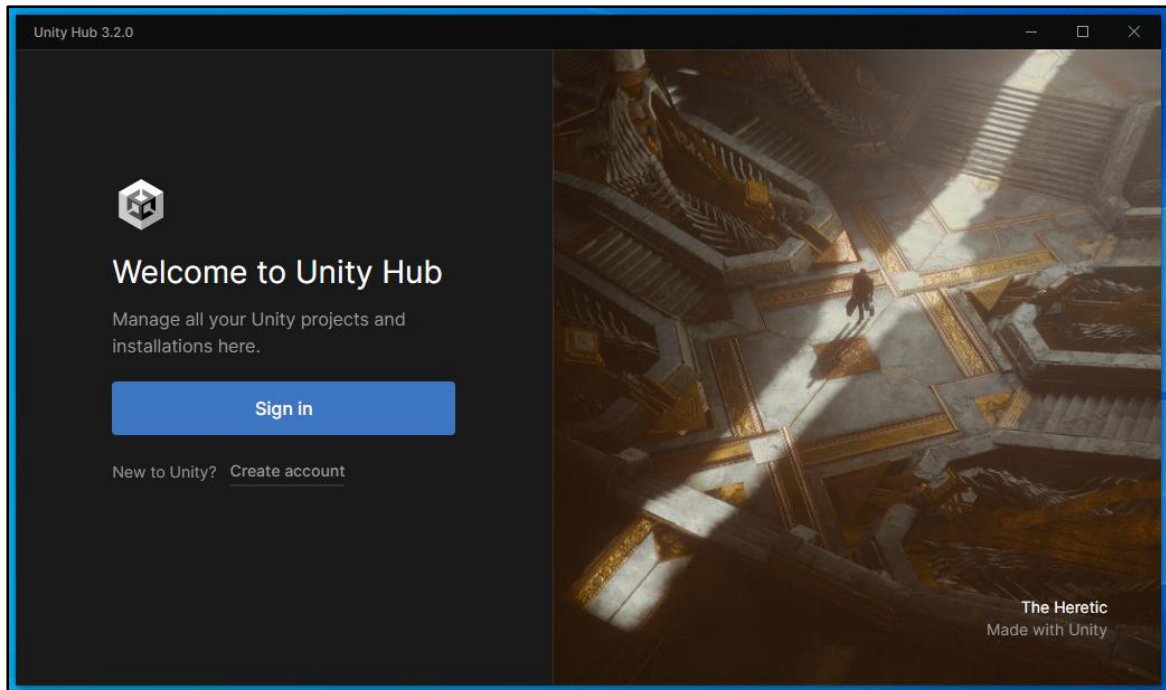
The Unity Hub icon looks like the screenshot below.



**Image description:** The Unity hub Windows icon.

**When the Unity Hub loads, you will probably need to sign in if you have not used Unity on the university PC you are currently using.**

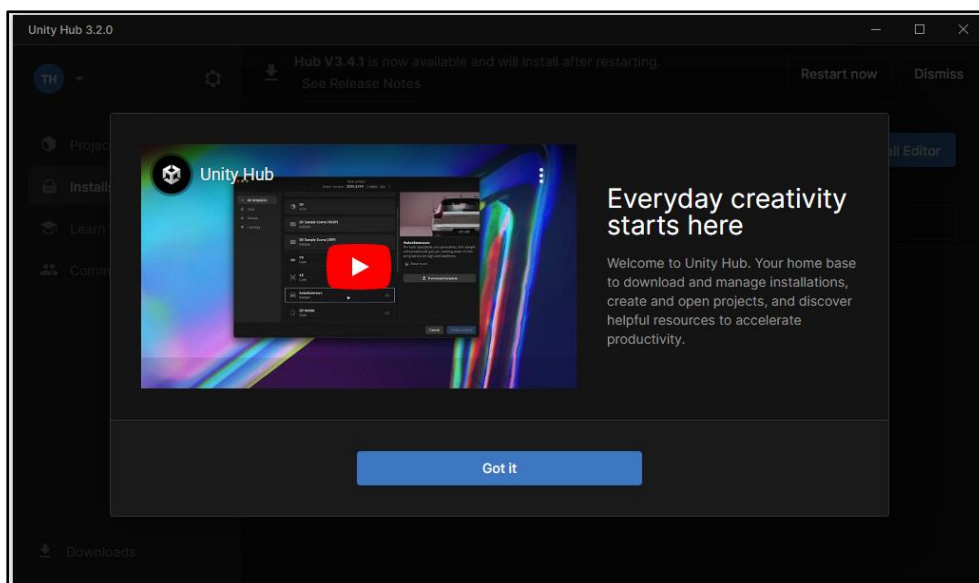
See the screenshot below for an example of what you might see when the Unity Hub loads on a university PC.



**Image description:** The Unity Hub window when it first loads, and you have not used Unity on the university PC you are currently using. Click Sign in if you have a Unity account OR click Create account if you do not.

If needed, click the Sign in button. A webpage will load. Sign in with your free Unity account OR if needed, create a free Unity account. If you do not have a free Unity account, you need to create one. You can use your university email address or a personal one.

Once you have signed in, you might see a message like this. Just click “Got it”.



**Image description:** The Unity Hub window when you have signed in for the first time. Just click “Got it”.

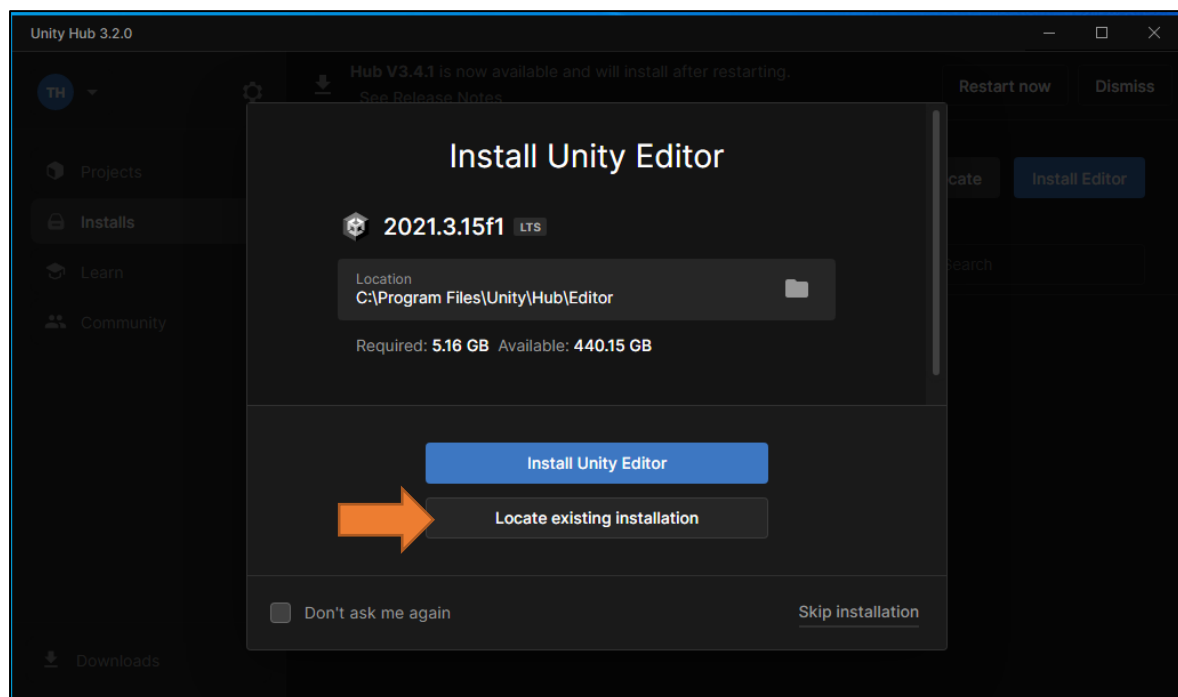
**Next, you might see a message like the screenshot below if you have not used Unity on this particular University PC before.**

- Unfortunately, the University PCs do not automatically know the location of Unity the first time you use it on that PC.

**We first need to locate the existing installation of Unity.**

- Remember, we cannot install any software on University PCs; therefore, we cannot install the Editor on University PCs. However, it is installed. We just need to locate it.

**Click on the “Locate existing installation” button.**

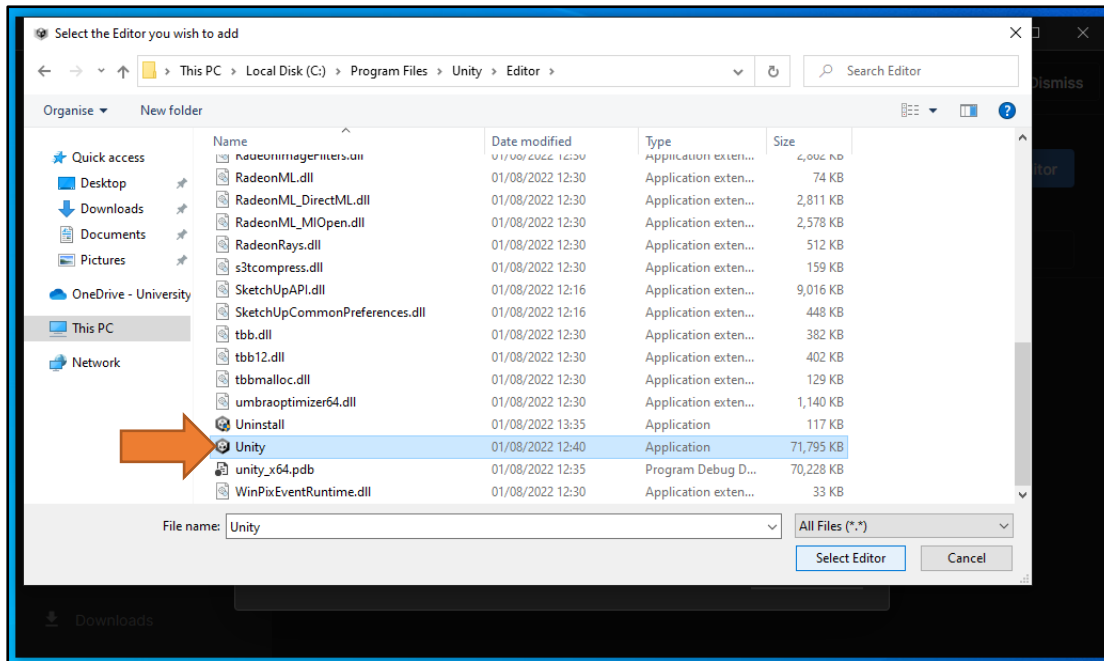


***Image description:*** The Unity Hub window asking for the location of the Unity Editor. Click on the “Locate existing installation” button to find the existing install.

**Next, navigate to the folder c:\Program Files\Unity\Editor**

**...and Select Unity from the list. Click the “Select Editor” button.**

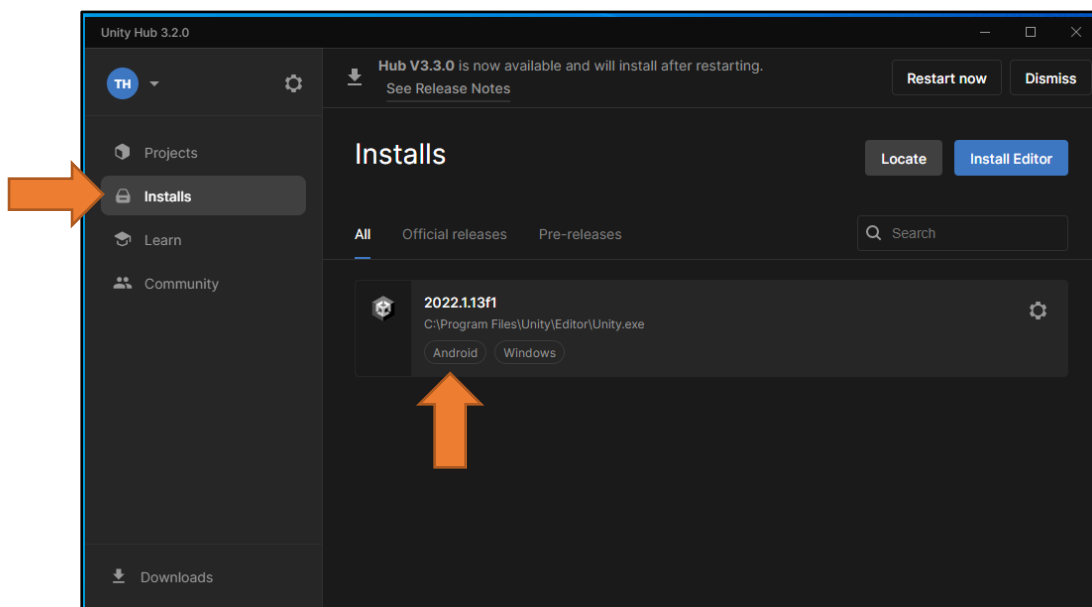
See the screenshot below for an example of what this looks like.



**Image description:** The Unity Hub window asking for the location of the Unity Editor. Next, navigate to the folder `c:\Program Files\Unity\Editor` and Select Unity from the list. Click the “Select Editor” button.

When you click the “Select Editor” button the “Select the Editor you wish to add” window should disappear. We now need to check the Unity Editor has been added.

**In the Unity Hub, click the Installs tab to confirm the correct version of Unity has been added.** See the screenshot below for an example.



**Image description:** The Unity Hub window. The Installs tab has been selected. We can see a version of the engine has been added to the installs tab.

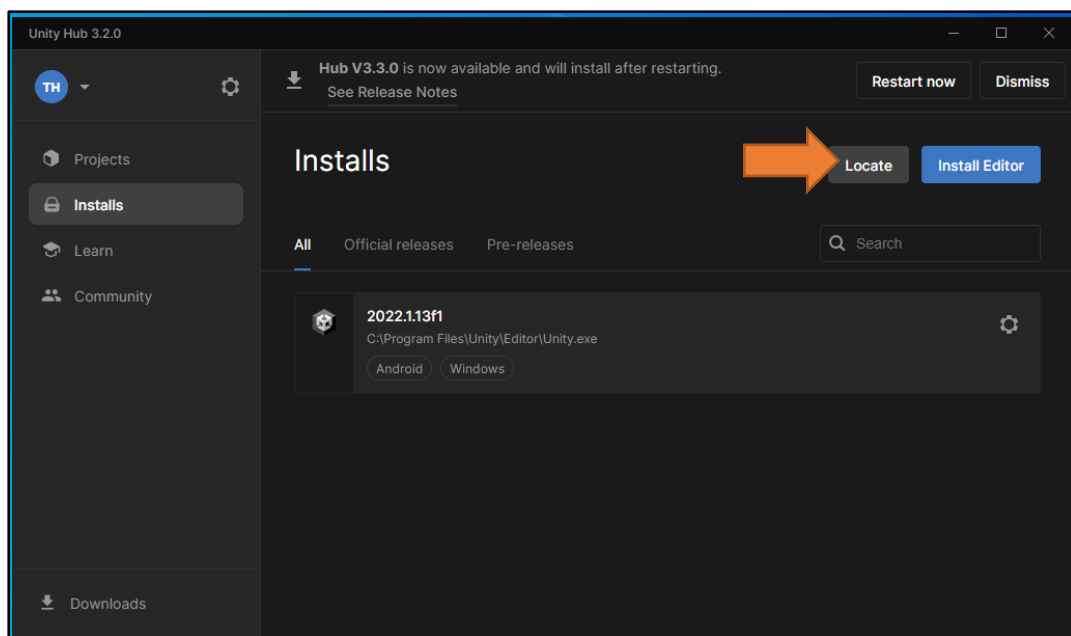
We can see in the screen shot above that a version of the engine has been added to the installs tab. Therefore, we can now use Unity.

**Top Tip:**

When you load Unity on a University PC you should always click the Installs tab to confirm the correct version of Unity has been added to the Unity Hub.

Sometimes the “Install Unity Editor” window with the “Locate existing installation” button does not appear. But you still might need to add a version of the Unity editor.

If there is no version of Unity in the Installs list, then click the “Locate” button in the Unity Hub to locate the Unity Editor. See the screenshot below for an example.

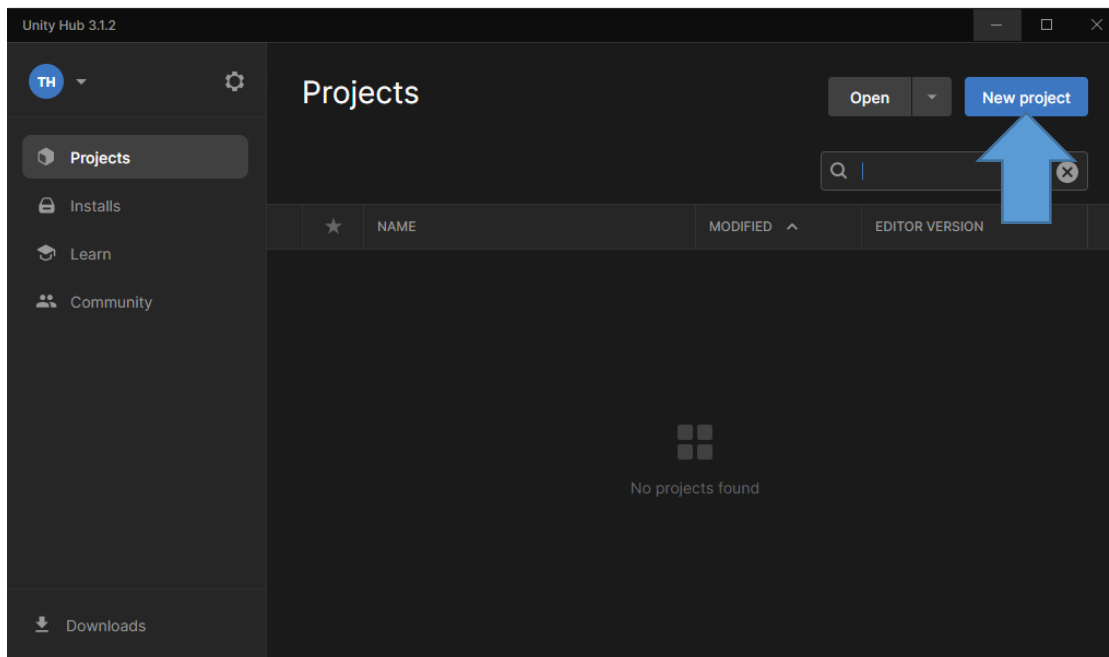


***Image description:*** The Unity Hub window. If an installation of the Unity Editor is missing, click the “Locate” button to locate the Unity Editor

Once the correct version of Unity has been added to the Unity Hub (see installs tab), you are ready to use Unity.

In the Unity Hub, go to the Projects tab and click the (blue) New Project button.

The Unity Hub window should look like the screenshot below. The blue New Project button is on the top right of the window.



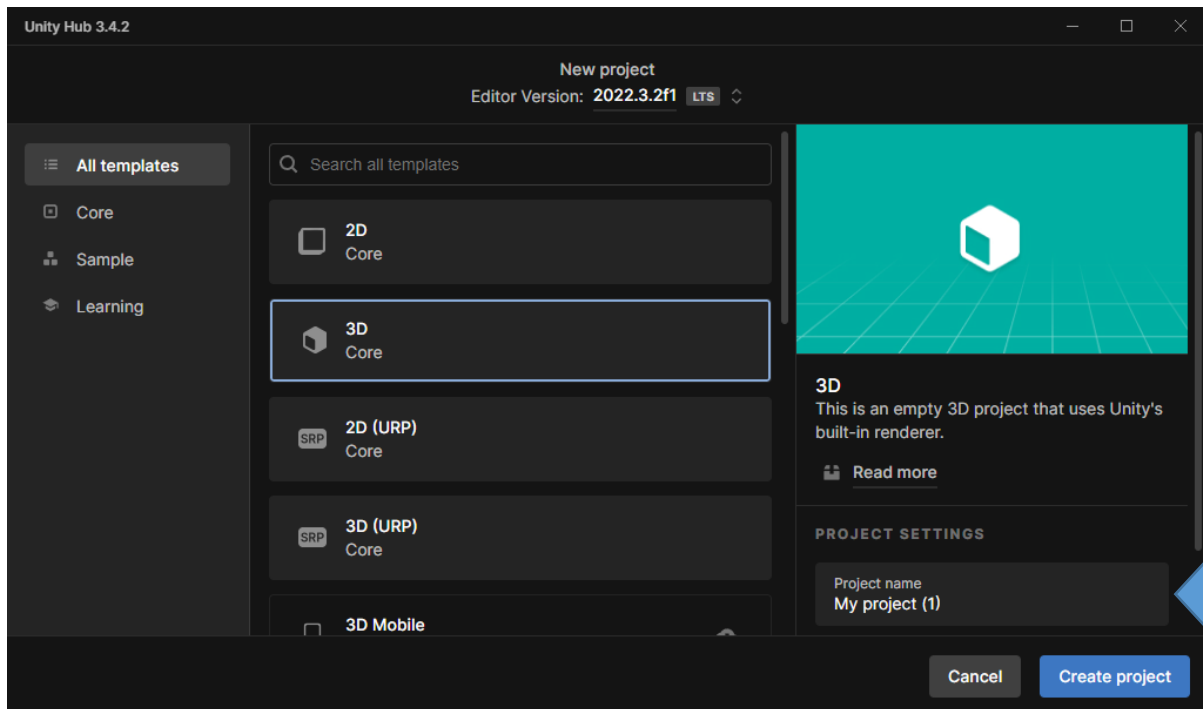
**Image description:** The Unity hub with an arrow pointing to the New Project button.

**When you click the New Project button a “New project” screen will appear.**

The window looks like the screenshot below.

Find the **3D Core** template.

Select the **3D Core** template.



**Image description:** The Unity hub “New project” screen.

Make sure you are creating a project in the correct version of the game engine. You can see the game engine version next to the “Editor Version:” text at the top of the Unity hub “New project” screen. **See the main Canvas page or the module assessment for the version of the game engine we are using.**

Set the following settings in the Unity hub “New project” screen.

**Template:** Select the **3D Core** template. This is a basic empty 3D template.

**Project Name:** Give your project the name **AnimationExample1**.

**Location:** Select a location to store your project. Below is some additional information about where to store your projects.

### **PLEASE READ - Important Unity Project Save Location Information**

**In general, Unity projects are large (e.g., over 500 mb). Therefore, you need to think about where you are going to store the projects and develop a workflow to manage your Unity projects.** This is especially important when working with Unity projects on university PCs.

**In general, I would recommend storing your Unity projects on a USB3 memory stick OR USB3 hard drive. You can then load your Unity projects from your memory stick.**

- **To add a Unity project to the Unity Hub simply click the drop-down menu next to the open button and click “Add Project from disk”.**



## 2. Importing Assets into your Project

We will now import some assets that we will use in this workbook.

Unity has an Asset Store that is home to thousands of free and priced assets. We will use **free** assets on the Unity Asset Store to help us create games or interactive 3D applications.

**We are going to add the following free assets to our project:**

- POLYGON Starter Pack - Low Poly 3D Art by Synty
- Robot Hero : PBR HP Polyart
- Starter Assets - First Person Character Controller

**For quickest import, add the assets in the order above.**

**Before you can add the assets to your Unity project you need to first add them to your Unity account.**

### **Step 1: Adding Assets to Your Unity Account**

**If needed, add the Unity assets to your Unity account. See the “Introduction to Unity” workbook for detailed guidance on how to add assets to your Unity account via the Unity asset store.**

In a web browser, go to the web site: <https://assetstore.unity.com/>

**Log into your Unity account. If you do not have a Unity account, you need to create one.**

Search for the each of the assets listed above, select the asset, and click the “Add to My Assets” button (it is a big blue button near the top right of the web page). Make sure you have logged into your Unity account at this point.

**When you have added the assets to your account, you can close the web browser.**

### **Step 2: Importing Assets into Your Unity Project**

In Unity you can import assets that have been added to your account. If you have not added the assets to your account, you need to add them before you proceed. See the previous step for more information.

**In the Unity Editor, go to the Window menu and select Package Manager.**

Select “My Assets” from the drop-down menu.

**You may need to sign in with your Unity account to view your assets.** If you are not signed-in, you should have the option to sign-in on the left panel of the Package Manager. Sign-in if needed.

A list of all your assets should appear.

**Import all the assets asset pack(s) outlined above.**

Follow these steps to import each asset pack:

- To import an asset pack, select it in the list.
- Next, you may need to click the **Download** button to download the asset to your computer.
- When the assets have been downloaded an Import button will appear. **Click the import button to import the asset.**
- **When you click the Import button a small “Import Unity Package” window will appear.**
- **Simply, click the Import button on the “Import Unity Package” window.**
- When you click the Import button the assets will be added to your project.

**When you have imported all the assets, close the Package Manager window.**

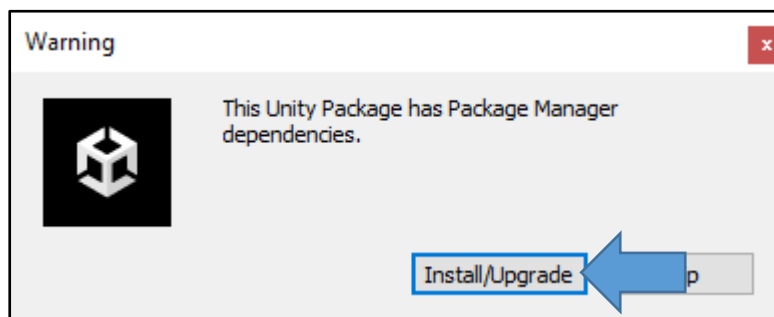
### **Step 2.1: Importing - Starter Assets - First Person Character Controller**

Remember, the **First Person Character Controller** has some additional import steps.

Select the **Starter Assets - First Person Character Controller** from the list. If you have a lot of assets, you may need to click the **Load Next** button at the bottom of the assets list.

**When you click the Import button, you will see this message.**

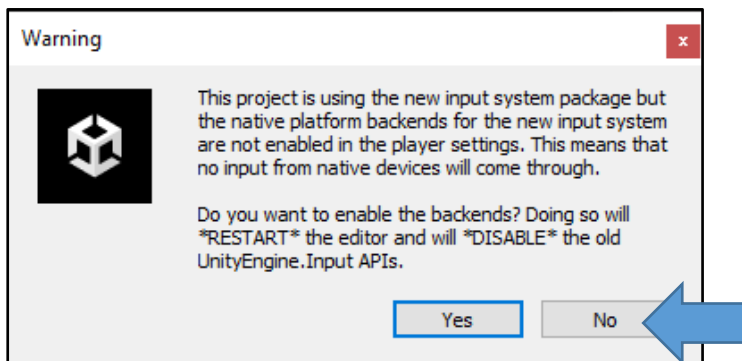
**Click the “Install/Upgrade” button.** See the screenshot below for an example.



***Image description:*** When you click the Import button, you will see this message. Click the “Install/Upgrade” button.

Wait for the package dependencies to be installed / upgraded.

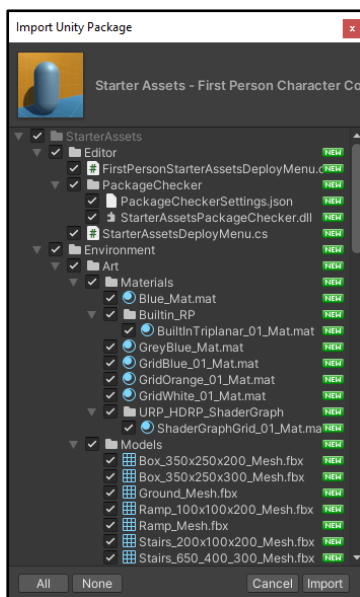
Next, the message in the screenshot below will be displayed. Click No.



**Image description:** A warning message stating that the assets being imported use the new input system package. Click no.

Wait while more importing happens.

Then, a small “Import Unity Package” window will appear. See the screenshot below for an example.



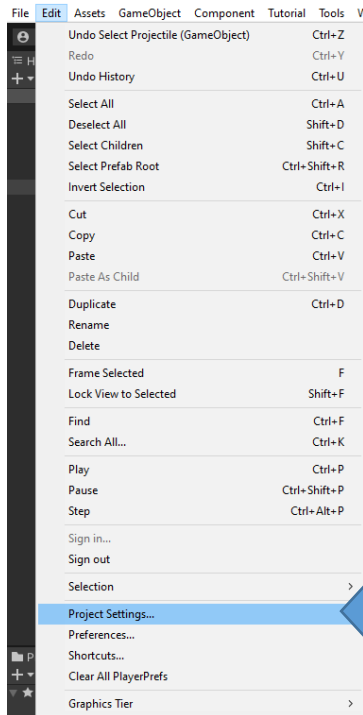
**Image description:** The “Import Unity Package” window. Click Import.

Simply, click the Import button on the “Import Unity Package” window.

When you click the Import button the assets will be added to your project. Wait while this happens.

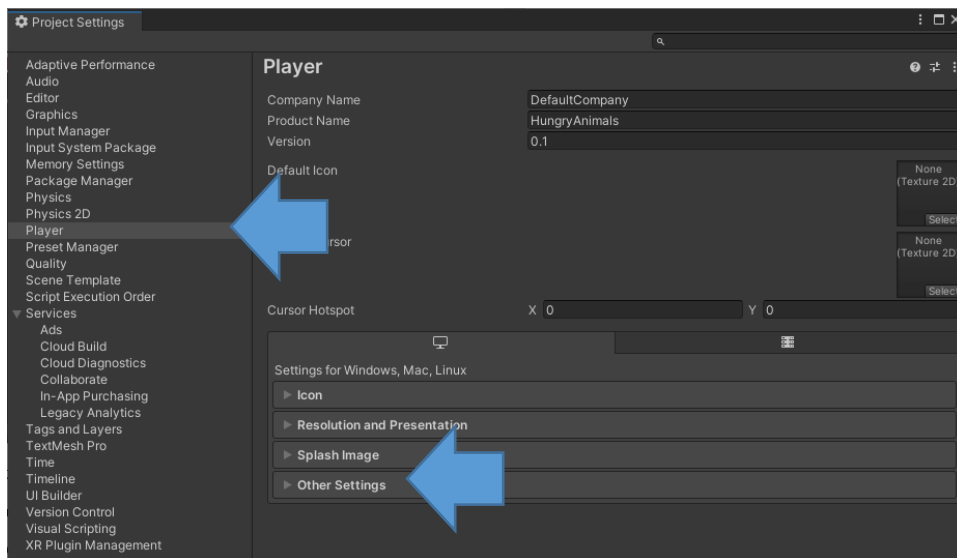
When everything has finished importing you should close the Package Manager.

In the Unity Editor, go to the Edit drop down menu and select Project Settings. See the screenshot below for an example.



**Image description:** Open the Project Settings window to set Unity project input type.

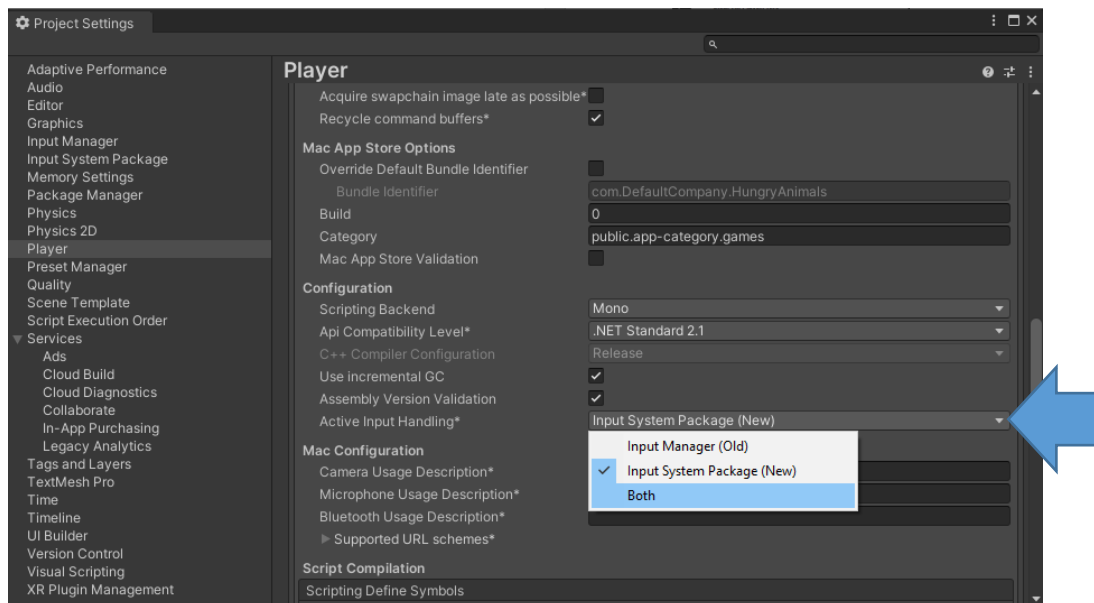
Select Player from the list on the left and expand the “Other Settings” option. See the screenshot below for an example.



**Image description:** Select Player from the list on the left and expand the “Other Settings” option.

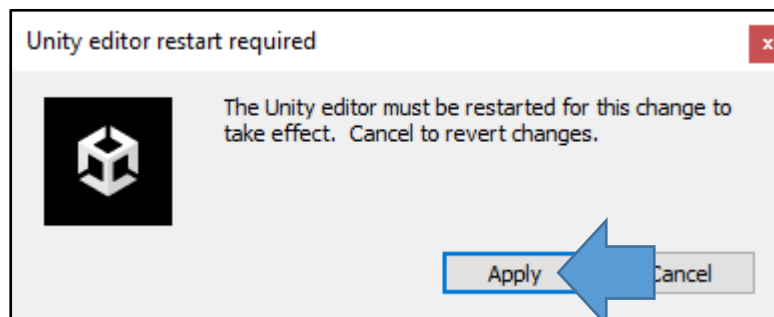
**Scroll down “Other Settings” until you find the “Active Input Handling” option. Click the drop-down menu and select Both.** See the screenshot below for an example.

**Note:** there are two input systems in Unity. An old one and a new one. In our workbooks we will use both because the old input system is very quick to get up and running.



**Image description:** Scroll down “Other Settings” until you find the “Active Input Handling” option. Click the drop-down menu and select Both.

A “Unity editor restart required” message box will appear. Click Apply.



**Image description:** A warning message stating that Unity must restart. Click Apply.

**Note,** at this point Unity will restart. This is fine. If needed, click Save to save your scene. Unity will now restart.

When Unity restarts, close the “Project Settings” window (if needed).

When Unity has restarted we can continue to import asset packs (if needed).

### 3. Set External Script Editor and Regenerate Project files (If needed)

When our Unity projects include code, we might need to set the external script editor to Visual Studio and click the Regenerate Project files button.

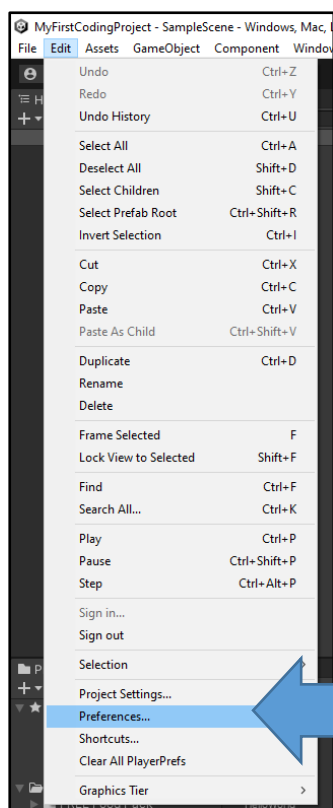
This is important if you are opening a Unity project on a computer for the first time and it contains code. Regenerating project files ensures that the Visual Studio intellisense will work properly.

Therefore, it is a good idea to check the external script editor is set to Visual Studio and click the Regenerate Project files button every-time you load your Unity project on a new computer.

**Tip:** At home you probably do not need to do this if you are using the same computer all the time. Things should just work. You only need to do this if you are opening your Unity project on a new computer.

Follow the steps below to set Visual Studio as the external script editor and regenerate project files.

In the Unity Editor, click the **Edit** drop-down menu, and select **Preferences**. It is the 5<sup>th</sup> option from the bottom of the menu. See the screenshot below for an example.



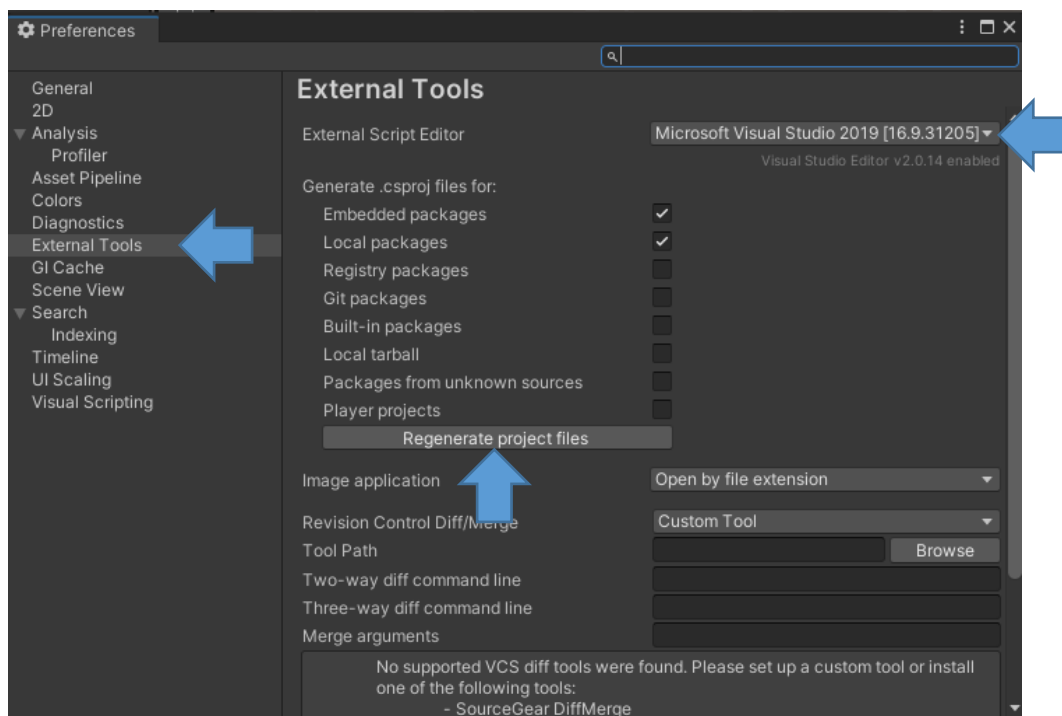
**Image description:** Open the Preferences window to set Visual Studio as the external script editor and regenerate project files.

A preferences window will load.

Select **External Tools** from the list on the left.

Then for the **External Script Editor** option and select the correct version of **Visual Studio**. If Visual Studio is already set, you do not need to do anything.

Then click the **Regenerate Project files** button.



**Image description:** The Preferences window. If needed, set the external script editor to Visual Studio. Then, click the regenerate project files button.

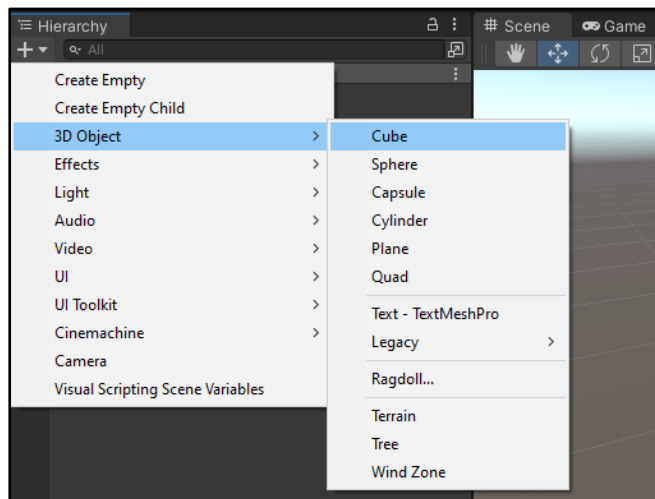
You can now close the preferences window.

## 4. Creating a Simple Scene with Primitive Shapes

In this section we will create a simple scene. This illustrates how you can create a simple scene in Unity using primitive shapes. This is an alternative to terrains or could be used in conjunction with terrains.

The first thing we will do is create a simple ground or floor to walk on.

Go to the Hierarchy Window and click the plus (+) menu. Select 3D Object -> Cube. See the screenshot below for an example.



**Image description:** The Hierarchy Window and the plus (+) menu. Cube has been selected in the menu.

A cube should be created in your scene.

At this point the cube should be selected in the hierarchy window and you should be able to enter a name for the cube. Give the cube the name **Ground**.

- **Tip:** If the cube is not selected in the hierarchy window. For example, you might have clicked on something else. Go to the Hierarchy Window and slowly single click twice on the Cube text to rename it. Or you can right click and select Rename.

Select the ground cube in the hierarchy window.

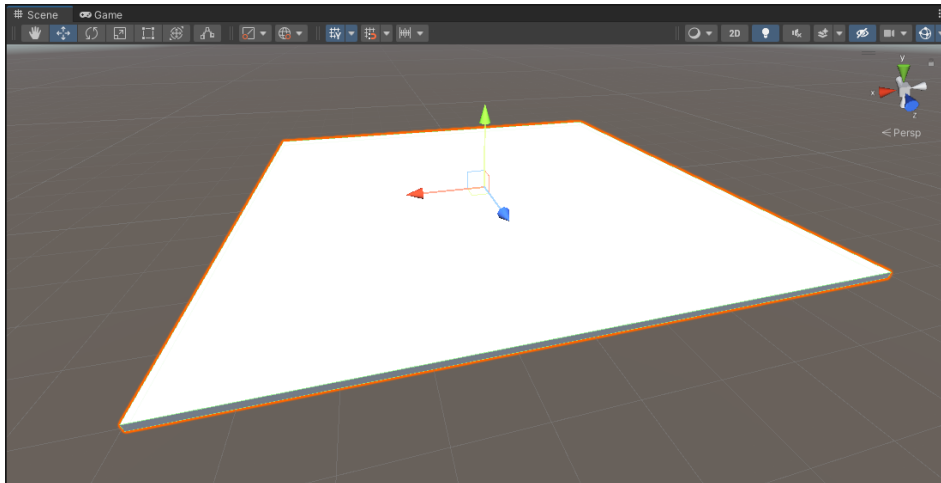
Go to the inspector. Scale the cube so that it looks more like a floor. I used the following settings:

X: 100  
Y: 1  
Z: 100

This creates a relatively small environment. You can use larger values for the x and z axes to create a bigger environment.

Your cube should now look like the screenshot below.





**Image description:** The ground cube in the scene view.

Next, we will give the ground some colour using a material.

Go to the Project Window. Select the Assets folder. Right click in the righthand panel of the Assets folder and select Create -> Folder.

When the folder is created give it the name **Materials**.

- **Tip:** It is a good idea to organise your assets in folders. You can make the folder structure simple or complicated. You should do what works for your project.

**Double click on the Materials folder to open it.**

**Right click in the Materials folder and go to Create -> Material.**

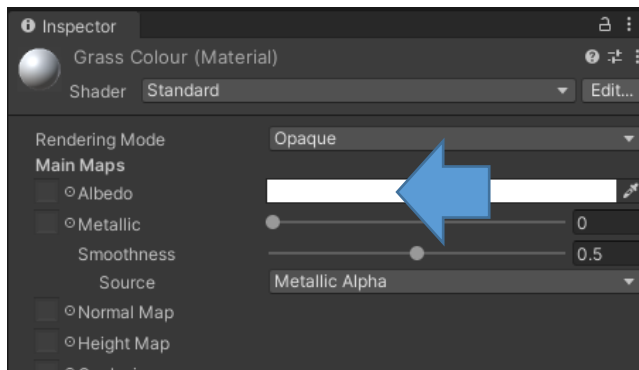
Give the Material the name **GrassColour**.

Select the material in the Project window. You should be able to see its properties in the inspector.

Go to the inspector, click on the block next to the Albedo property at top of the Inspector.

- **Information:** The Albedo property, pronounced, al-bee-dow, is the colour or texture map for the object.

See the screenshot below for an example.



**Image description:** The inspector window for the material. Click on the white box next to the Albedo property.

Clicking on the block next to the Albedo property at top of the Inspector will open a colour picker window.

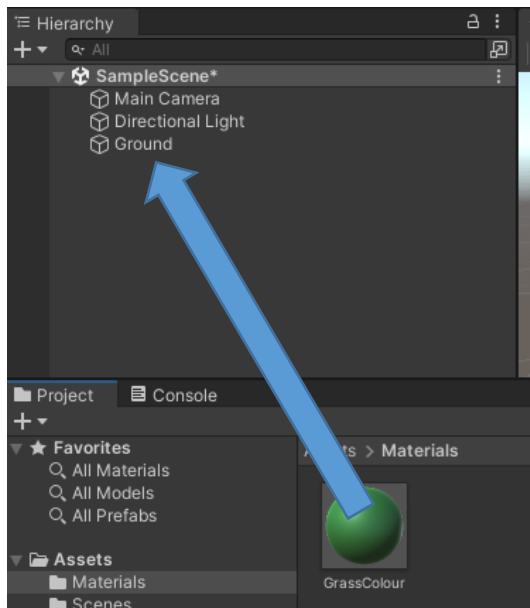
Select a green colour. I entered the hexadecimal value: 228C22.

See the screenshot below for an example.



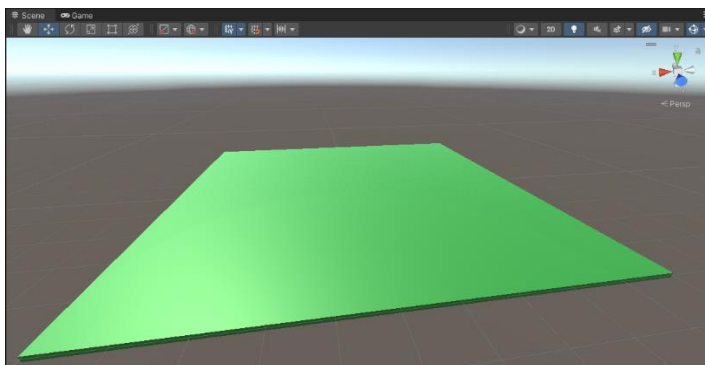
**Image description:** The material Albedo property colour picker. Close the colour picker window.

Next, we need to assign the material to the ground cube in the scene. To do this, drag the **GrassColour** material from the Project window and drop it onto the name of the ground object in the Hierarchy OR drag it onto the ground in the scene view.



**Image description:** Drag the material from the Project window to the ground GameObject.

Your updated ground should now look like it has a green grass colour applied to it. See the screenshot below for an example.



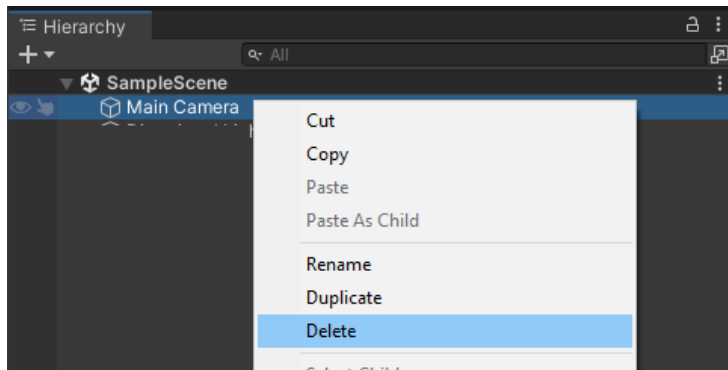
**Image description:** The ground cube in the scene view. A green material has been applied to it.

**Save your scene.** Go to the File menu and find the save option.

## 5. Adding a First-Person Camera to the Environment

We will now add a first-person camera to our scene. This camera will allow a player or user to move around our scene.

By default, Unity includes a camera in our scene. Go to the hierarchy, find the Main camera, right click on it, and select delete from the menu. See the screenshot below for an example.



**Image description:** Example of how to delete the “Main Camera” from the scene.

Once you have deleted the main camera from the scene you can add a first-person camera.

**Go to the Project window.**

**Navigate to the folder: StarterAssets -> FirstPersonController -> Prefabs.**

**You should find that some of the assets in this folder are coloured magenta (pink). This is because they are setup to use the universal render pipeline, instead of the built-in render pipeline, which this project is using.**

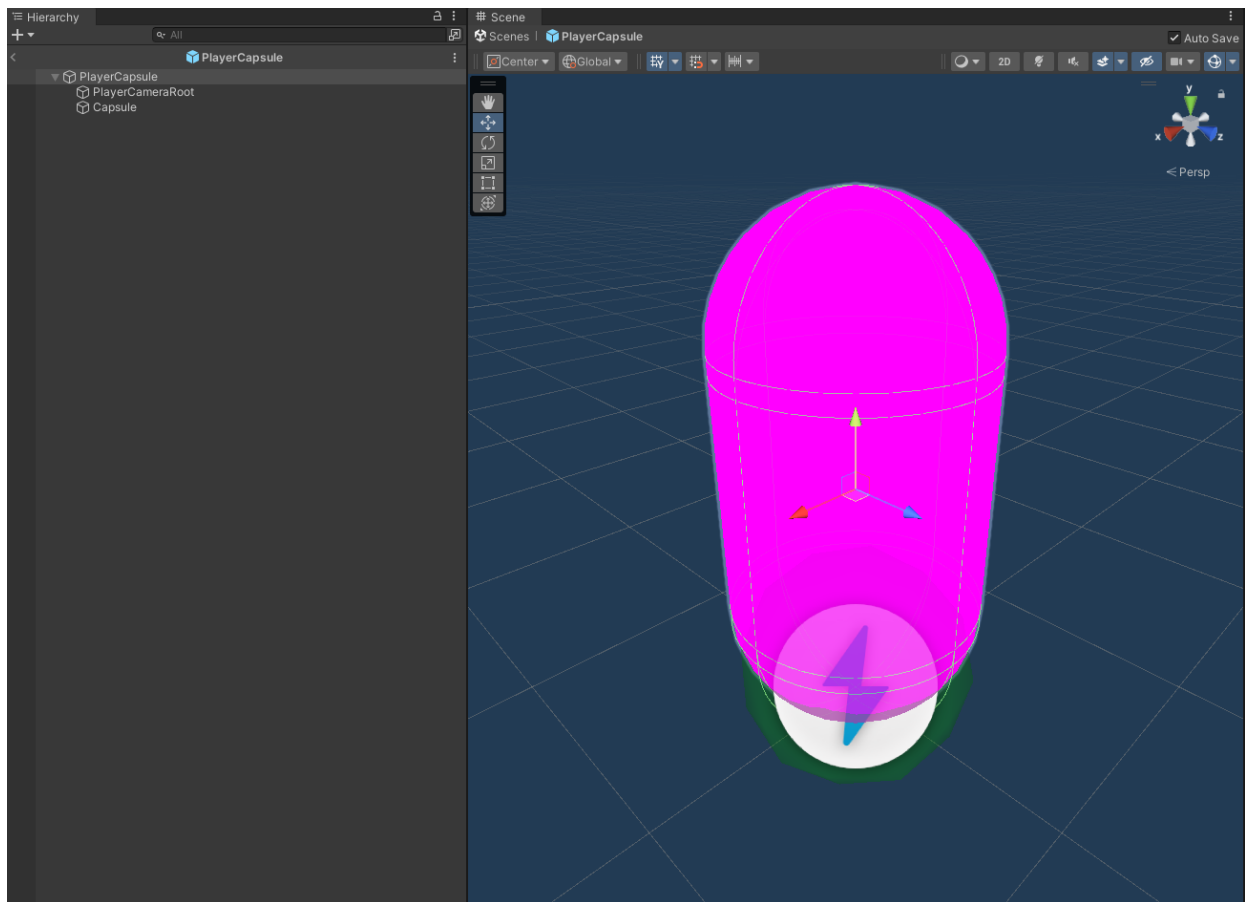
First, we need to change the **PlayerCapsule** prefab so that it uses the built-in render pipeline.

**Double click** on the **PlayerCapsule** prefab. This will open the prefab editor. See the screenshot below for an example.



**Image description:** some of the assets in the StarterAssets -> FirstPersonController -> Prefabs folder are coloured magenta (pink). This is because they are setup to use the universal render pipeline, instead of the built-in render pipeline, which this project is using. Double click on the PlayerCapsule prefab. This will open the prefab editor.

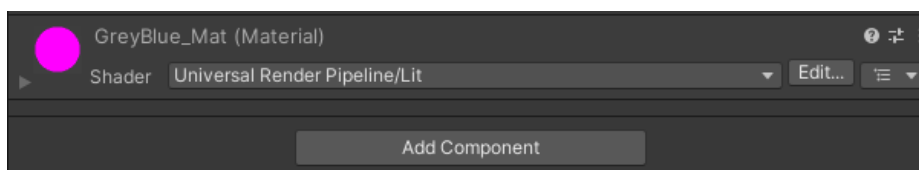
The prefab editor should open. See the screenshot below for an example.



**Image description:** The prefab editor with the PlayerCapsule prefab loaded.

**Click the Capsule gameObject in the Hierarchy.**

Go to the Inspector and find the material for the game object. See the screenshot below for an example.

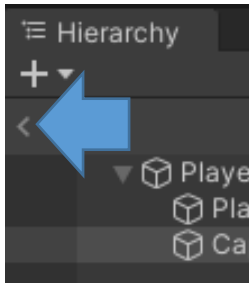


**Image description:** Go to the Inspector and find the material for the game object.

**Click the drop-down menu next to the word “Shader”. Select “Standard” from the drop-down menu.**

The capsule should now **not** be coloured magenta (pink).

Click the back arrow near the top of the Hierarchy to move back to your scene. See the screenshot below for an example.



**Image description:** Click the back arrow near the top of the Hierarchy to move back to your scene.

### **Save your scene.**

**Drag the PlayerCapsule prefab into the scene** and position it somewhere on the ground.

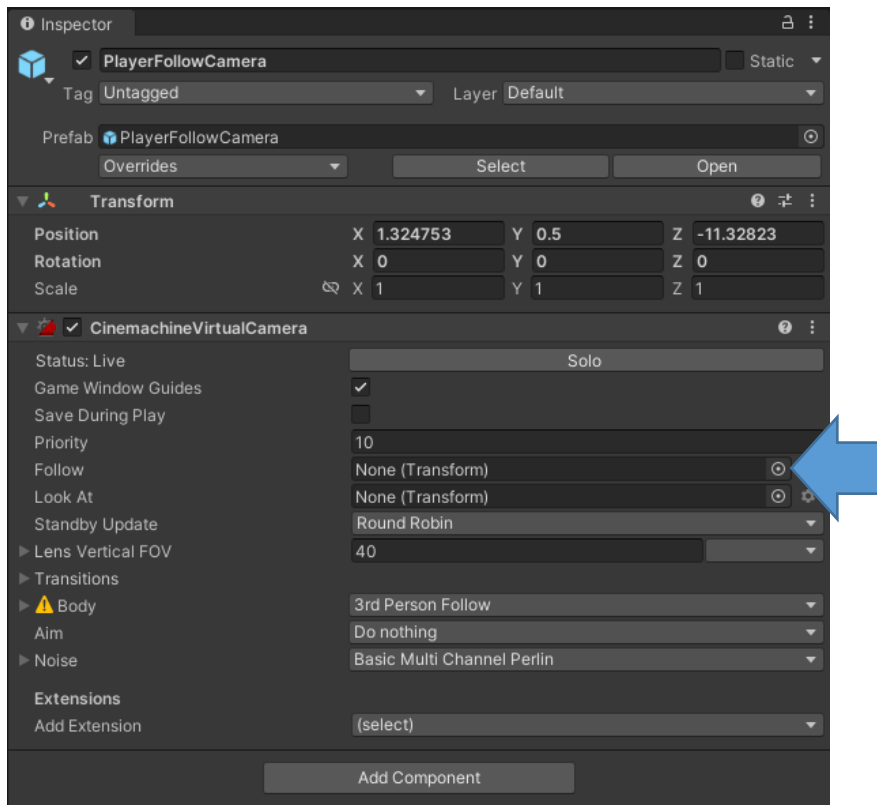
**Drag the PlayerFollowCamera prefab into the scene.** This prefab can be positioned anywhere; however, I would recommend you position it near the PlayerCapsule prefab.

**Drag the MainCamera prefab into the scene.** This prefab should automatically position itself at the PlayerFollowCamera prefab's position.

**Next, go to the hierarchy, and select the PlayerFollowCamera prefab.**

Go to the inspector and find the **CinemachineVirtualCamera** component.

Find the **Follow** property and click the bullet icon next to "None (Transform)". See the screenshot below for an example.



**Image description:** Go to the hierarchy and select the *PlayerFollowCamera* prefab. Go to the inspector and find the *CinemachineVirtualCamera* component. Find the follow property and click the bullet icon next to “None (Transform)”.

A “Select Transform” window will appear.

Double click on **PlayerCameraRoot** (in the Scene tab) to select it.

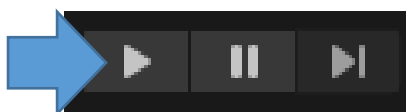
The follow property of the *CinemachineVirtualCamera* component should now have the value “*PlayerCameraRoot (Transform)*”.

**The first-person camera should now be setup.**

**Save the scene.**

**🎮 We are now ready to playtest the scene.**

Press the play button to playtest your scene. See the screenshot below for an example.



**Image description:** The game / scene play controls on the main toolbar. Click the play button.

Play Mode is a realistic test of your game.

- Note, when in Play mode you can adjust GameObjects via the Scene window. However, all adjustments made to GameObjects will be temporary and undone when play mode is stopped.

If the game view is behind the scenes view, click the Game view tab to select the Game view window.

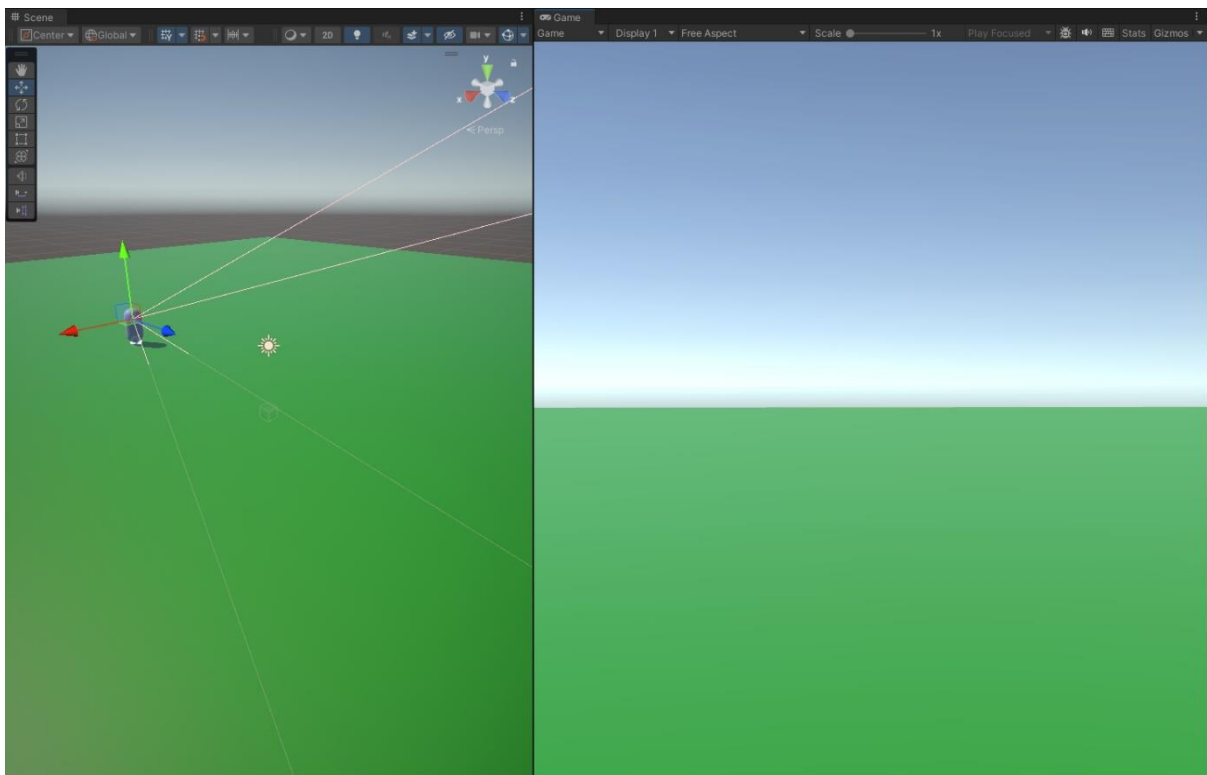
**You should be able to walk around the environment.**

The default controls:

- Arrow keys or WASD to move. Spacebar to jump. Hold down shift to sprint.
- Mouse look.

**To stop playtesting the game, click the play button again.**

See the screenshot below for an example. Note, in my setup I have the Scene and Game view side-by-side.



**Image description:** A playtest of the scene using the first-person controller. Note, in my setup I have the Scene and Game view side-by-side.



## 6. Mecanim Animation System

“Unity’s animation system is based on the concept of Animation Clips, which contain information about how certain objects should change their position, rotation, or other properties over time.

Each clip can be thought of as a single linear recording.

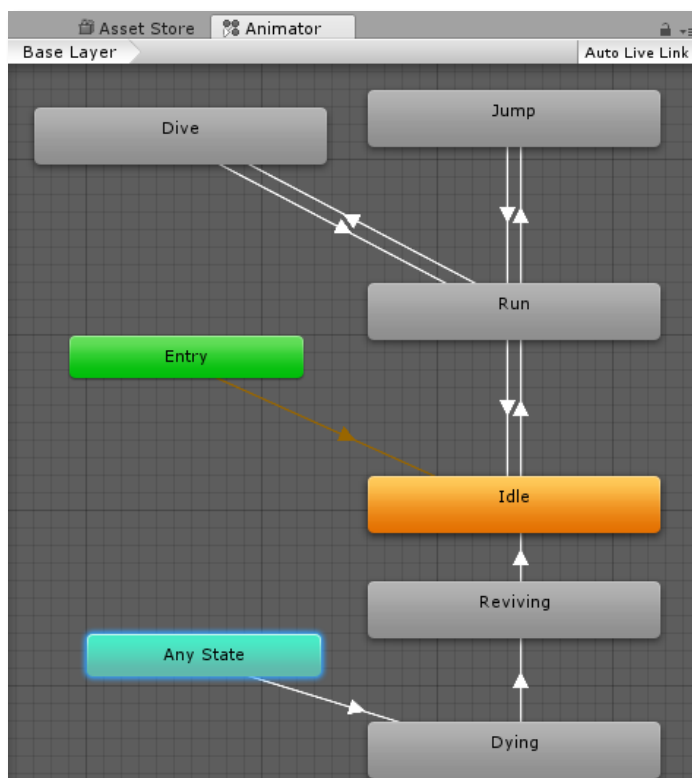
Animation clips from external sources are created by artists or animators with 3rd party tools such as Max or Maya or come from motion capture studios or other sources.

Animation Clips are then organised into a structured flowchart-like system called an Animator Controller.

The Animator Controller acts as a “State Machine” which keeps track of which clip should currently be playing, and when the animations should change or blend together.”

**Source:** <https://docs.unity3d.com/Manual/AnimationOverview.html>

See the screenshot below for an example of an **Animator Controller**.



**Image description:** An image of the animator controller.

**Let’s explore how to code and create behaviour for an animated character.**

We have imported some 3D virtual characters. It includes animations and all key assets.

**Go to the project window.**


Navigate to the folder: **SciFiWarriorPBRHPPolyart -> Prefabs.**

Drag the prefab **HPCharacter** into your scene. **Place the prefab on your ground.** See the screenshot below for an example.



***Image description:** The prefab HPCharacter in a scene.*

**Save the scene.**

 **We are now ready to playtest the scene.**

Press the play button to playtest your scene.

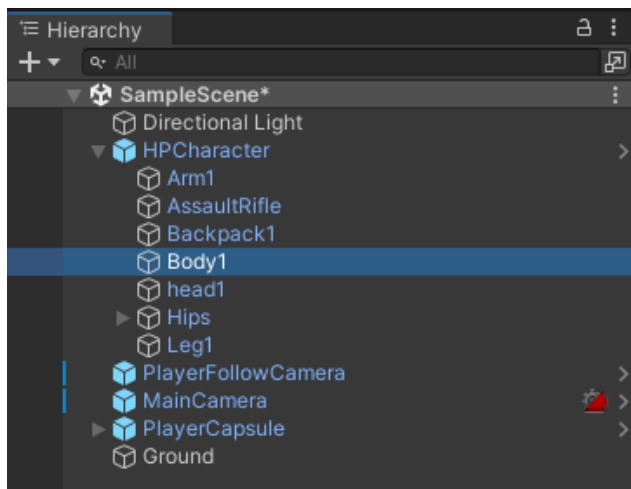
Move the first-person camera in front of the **HPCharacter** prefab. You should see the character play through all its animations. The **Animator Controller** included for this prefab just cycles through animations. We would like a controller that performs more useful behaviour, such as idle when the character is not moving and walking or running when the character is moving.

Let's create a new **Animator Controller** for the virtual character.

Click the **Play** button **again to stop playing.**

We will now update the **HPCharacter** prefab and create a new **Animator Controller** for the virtual character.

Go to the hierarchy and the **HPCharacter** prefab. Select the triangle next to the **HPCharacter** prefab and select the **Body1** prefab. See the screenshot below for an example.



**Image description:** The HPCharacter prefab selected in the hierarchy. The Body1 GameObject within the HPCharacter prefab has been selected.

Go to the Inspector, click the Add Component button and add a Capsule Collider component.

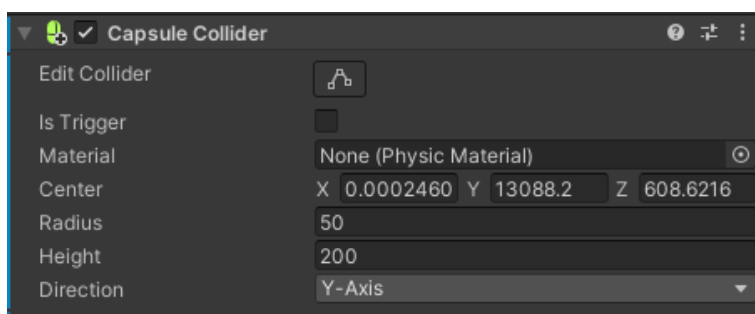
Set the Capsule Collider properties to the following setting:

**Radius:** 50.

**Height:** 200.

**Center:** Y: 13048.2

Your capsule collider component properties should look like the screenshot below.



**Image description:** The capsule collider component properties.

Your **HPCharacter** prefab should now have a green capsule collider around it. Your scene should look like the screenshot below.



**Image description:** The HPCharacter prefab. We can see a green capsule collider.

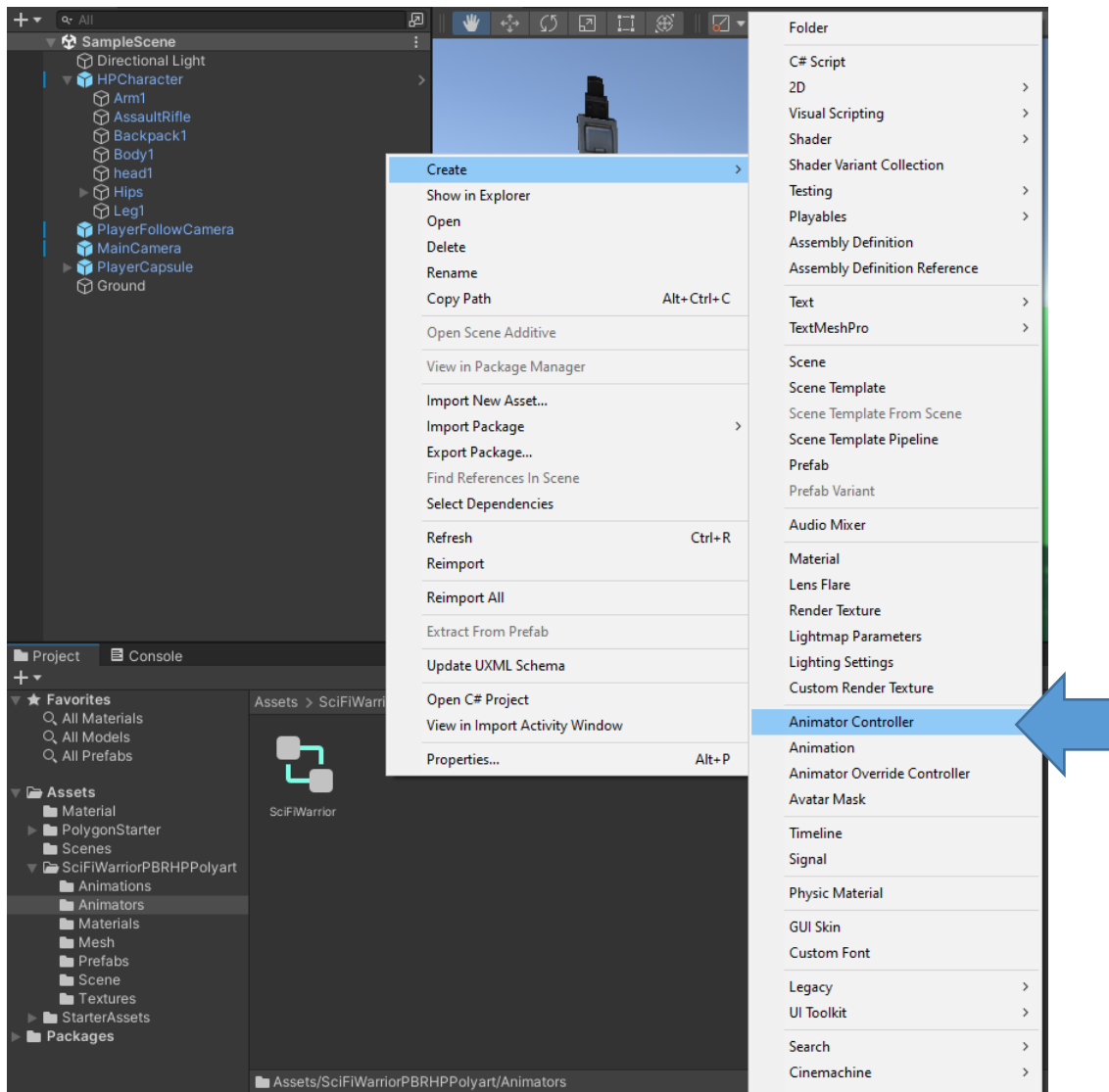
**General Note** – These values for Radius and height are quite large because the **HPCharacter** prefab is large and has been scaled down to fit with the rest of the model. See the scale property when you select **Body1** in the **HPCharacter** prefab. Other animated character will probably require smaller values for this.

**Save the scene.**

**Next, go to the project window.**

**Navigate to the folder: SciFiWarriorPBRHPPolyart -> Animators.**

Right click in the Animators folder, go to the Create menu and select **Animator Controller**. See the screenshot below for an example.



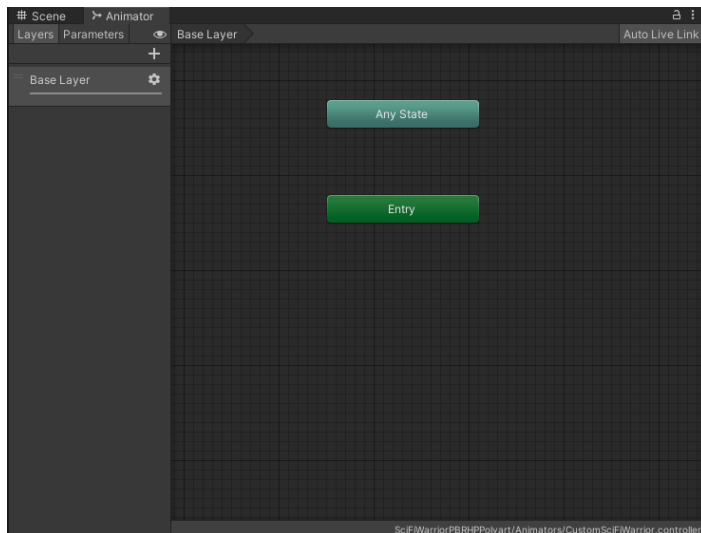
**Image description:** In the project window, Navigate to the folder: SciFiWarriorPBRHPPolyart -> Animators.Right. Click in the Animators folder, go to the Create menu and select Animator Controller.

Name the Animator Controller: **CustomSciFiWarrior**. See the screenshot below for an example.



**Image description:** The newly created Animator Controller, called CustomSciFiWarrior.

Double click on the **CustomSciFiWarrior** Animator Controller. The animator window should appear. See the screenshot below for an example.



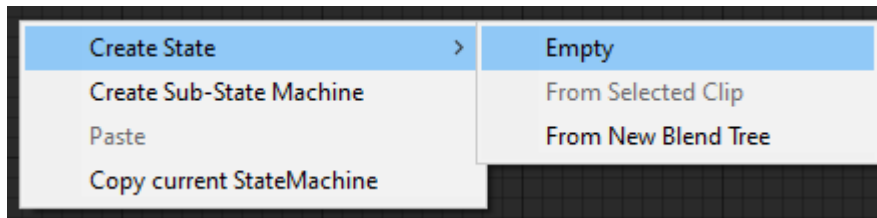
**Image description:** The animator window for a newly created Animator Controller.

We will now create an Animator Controller state machine for our **HPCharacter**.

**We will now add states to the Animator Controller.**

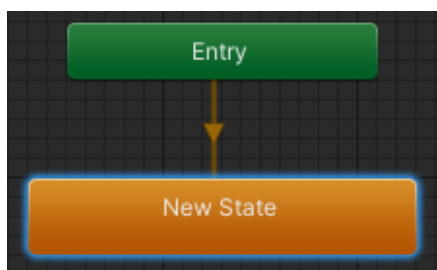
Right click on the checked dark grey area of the Animator window.

Select "Create State" -> Empty. See the screenshot below for an example.



**Image description:** Select "Create State" -> Empty. See the screenshot below for an example.

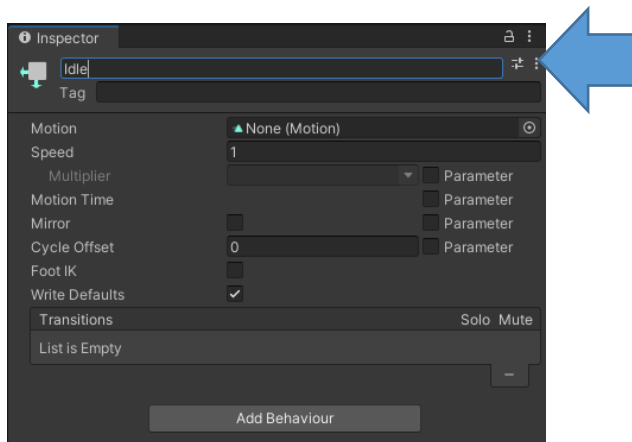
An Empty Animator state will be created. The first state you create will also result in a transition being created from the Entry state and New State. Your Animator window should look like the screenshot below.



**Image description:** The newly created state. The first state you create will also result in a transition being created from the Entry state and New State.

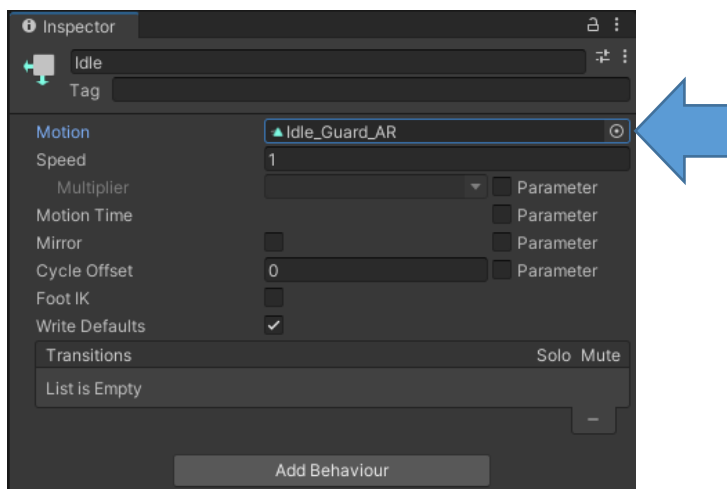
Click on the “New State”.

Go to the **inspector** and rename the new state to **Idle**. Press the enter key after you have written the new name. See the screenshot below for an example.



***Image description:** Give the new state the name Idle.*

Go to the Motion option, click the bullet icon, and select the **Idle\_guard\_ar** animation from the window that appears. The Inspector for the Idle state should look like the screenshot below.

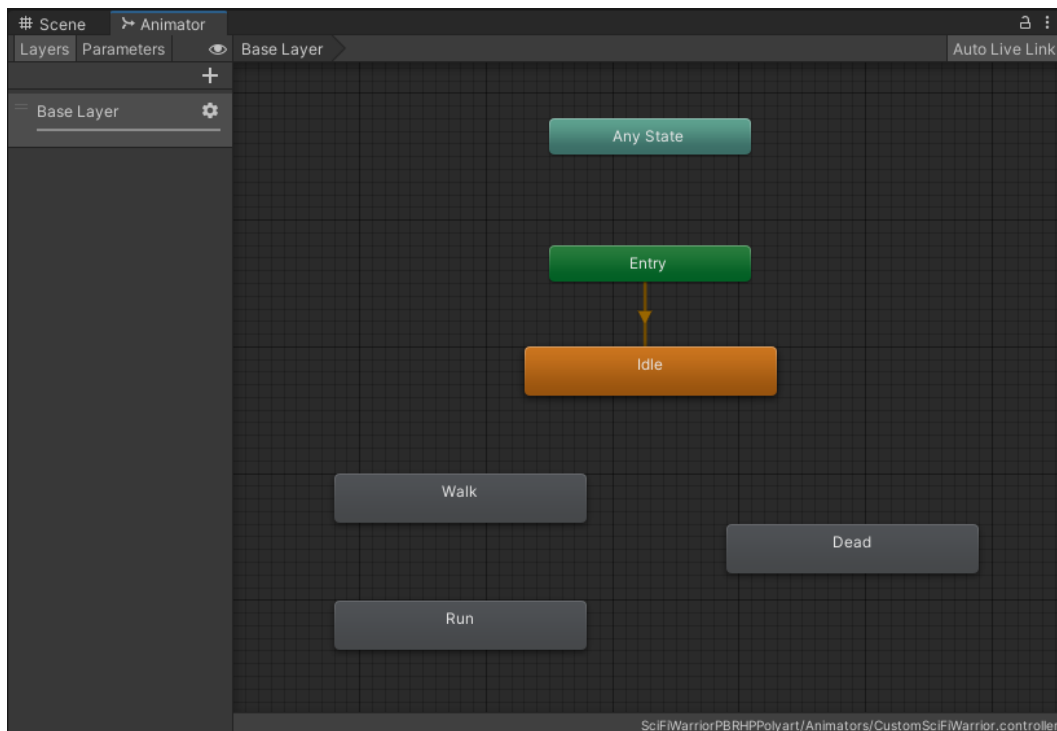


***Image description:** The Idle state inspector properties. Motion has been set to the Idle\_guard\_ar animation*

Next, add the following states and set the following animations.

State Name	Animation Name
Walk	WalkFront_Shoot_ar
Run	Run_guard_AR
Dead	Die

Your animator window should now look like the screenshot below.

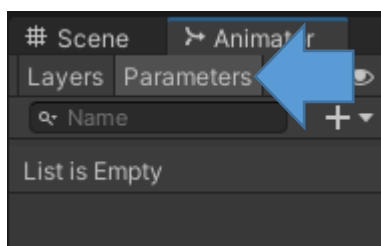


***Image description:** The animator window after you have added your states.*

We now need to add transitions to the Animator state machine.

However, first we need to add Parameters that the Animator can use.

Go to the Animator window and select the Parameters tab. See the screenshot below for an example.

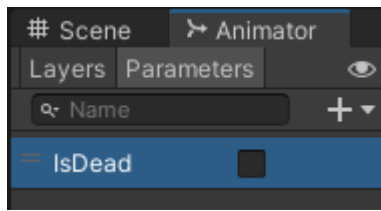


***Image description:** The Parameters tab in the Animator window.*

Click the small plus (+) button to add a parameter. From the submenu select **bool**.

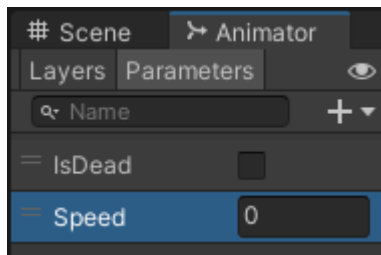
Name the new parameter / variable **IsDead**. Your parameter should look like the screenshot below.





**Image description:** The Parameters tab in the Animator window. An IsDead bool variable has been added.

Next, add a **float** parameter / variable called **Speed**. Your parameters should look like the screenshot below.



**Image description:** The Parameters tab in the Animator window. A Speed float variable has been added.

We now need to add transitions to the Animator state machine.

Right click on the Idle state and select **Make Transition**. Move the mouse and click on the Walk state. Your updated Animator state machine should look like the screenshot below.



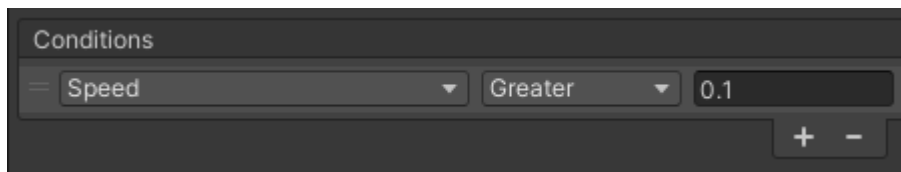
**Image description:** A transition has been added between the Idle state and the Walk state.

**Select the transition you just created and go to the inspector.**

Go to the Conditions section in the inspector.

Click the plus (+) button to add a condition.

Select the **Speed** variable from the dropdown menu, select **Greater** from the other dropdown menu and enter **0.1** in the text box. Your Inspector window should look like the screenshot below.



**Image description:** A condition has been added by clicking the plus (+) button. Select The Speed variable from the dropdown menu, select Greater from the other dropdown menu and enter 0.1 in the text box.

**Note** - If you wanted/need to add another condition you can by clicking the plus (+) button again. However, in this example we do not need any other conditions.

**Next, we need to create transitions and add conditions for all the states we have created.**

You should create the following transitions and add the following conditions.

### **Transition**

### **Condition**

Idle -> Walk  
*(We have already created this transition. You do not need to create it again.)*

Speed Greater 0.1

Walk -> Idle

Speed Less 0.1

Walk -> Run

Speed Greater 1

Run -> Walk

Speed Less 1.1

Idle -> Dead

Dead true

Walk -> Dead

Dead true

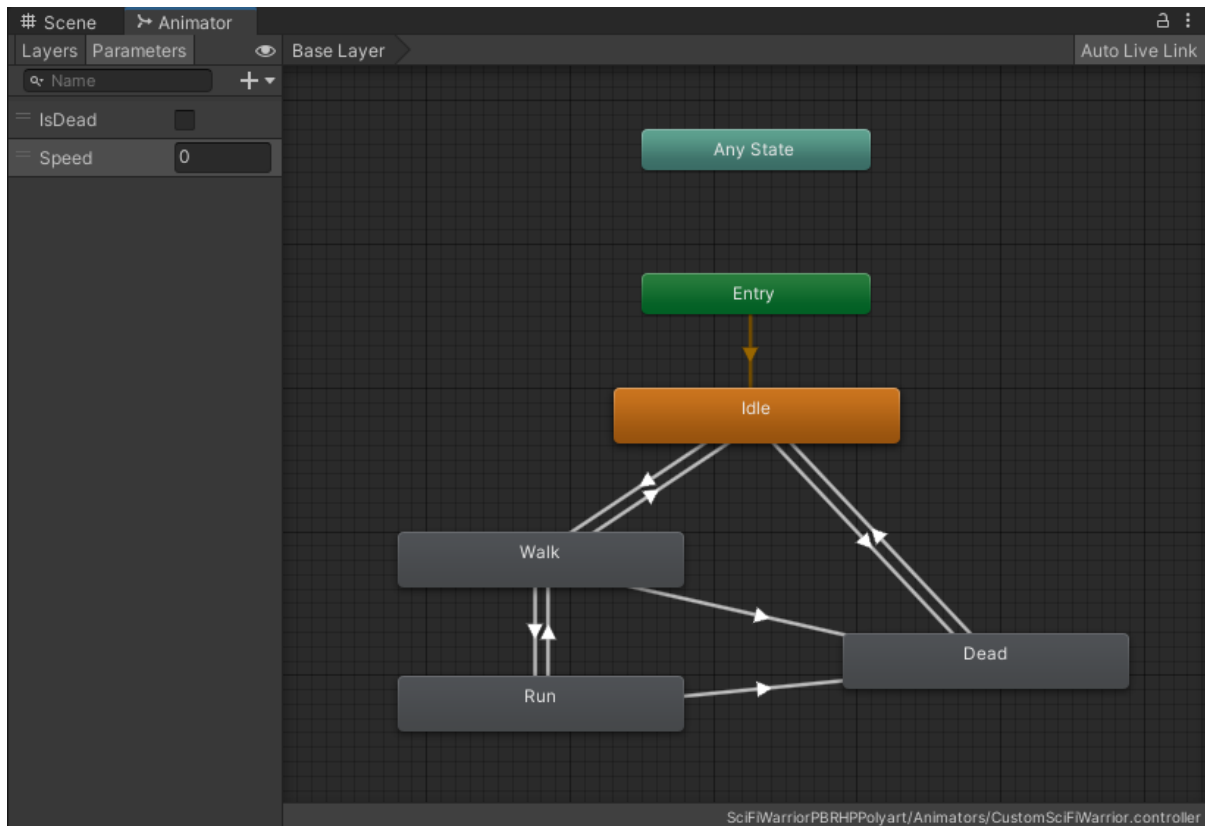
Run -> Dead

Dead true

Dead -> Idle

Dead false

Below is an example of the Animator state machine you should have created.



**Image description:** The Animator state machine you should have created. All the states now have transitions.

This state machine allows the animations of the player to move from Idle to walk to Run. It also allows the player to move to the dead state from any of its states. When in the dead state the animations can only move to the idle state.

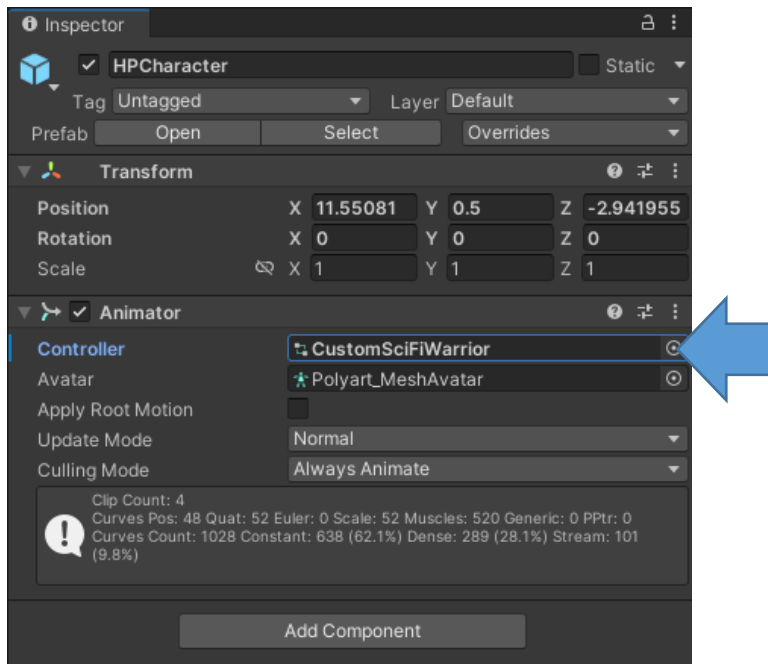
### Close the Animator window.

Go to the hierarchy window and select the **HPCharacter** prefab.

Go to the inspector and go to the Animator section.

Go to the Controller property and click on the bullet icon. A window will appear. Select the **CustomSciFiWarrior** Animator Controller from the window.

The Controller property should in the Animator section should look like the screenshot below.



**Image description:** The inspector for the HPCharacter prefab. The Controller property has been set to the CustomSciFiWarrior Animator Controller.

We have now created and set a new Animator Controller. Next, we will create a script to control the behaviour of the HPCharacter.

## 7. Scripting a Virtual Character

We will now create a C# script to control the behaviour of the **HPCharacter**.

Create a C# script. Go to the project window. Select the Assets folder. Right click in the righthand panel of the Assets folder and select Create -> Folder. When the folder is created give it the name **Scripts**.

**Double click on the Scripts folder to open it.**

Right click in the Scripts folder and go to Create -> C# Scripts.

When the script has been created give it the name **SciFiWarriorCON**.

Update the code in the script file to match the code below.

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class SciFiWarriorCON : MonoBehaviour
{
    // A reference to the player GameObject.
    [SerializeField]
    private GameObject player;

    // The speed of the character.
    [SerializeField]
    private float moveSpeed = 1.0f;

    // A reference to the animator.
    private Animator anim;

    // Store the current AnimatorStateInfo.
    AnimatorStateInfo currentStateInfo;

    // Hashes to animator parameters.
    private int speedHash = Animator.StringToHash("Speed");
    private int dyingHash = Animator.StringToHash("IsDead");

    // Start is called before the first frame update
    void Start()
    {
        anim = GetComponent<Animator>();
    }

    // Update is called once per frame
    void Update()
    {
        if (Vector3.Distance(transform.position, player.transform.position) < 7)
        {
            anim.SetFloat(speedHash, 0.7f);
            Vector3 tmp = player.transform.position;
            tmp.y = this.transform.position.y;

            transform.LookAt(tmp, Vector3.up);

            currentStateInfo = anim.GetCurrentAnimatorStateInfo(0);
            if (currentStateInfo.IsName("Walk"))
            {
                transform.Translate(Vector3.forward * moveSpeed * Time.deltaTime);
            }
        }
        else
        {
            anim.SetFloat(speedHash, 0);
        }
    }
}

```

The code above includes some class level variables to store a reference to the player GameObject and the animator for this GameObject. It also stores hashes to parameters in the animator. The Update method sets the animator parameter for speed to 0.7 if the player GameObject is closer than 7 units to the virtual character. Otherwise, the speed parameter is set to 0.

**Save the script in Visual Studio and go back to Unity.**

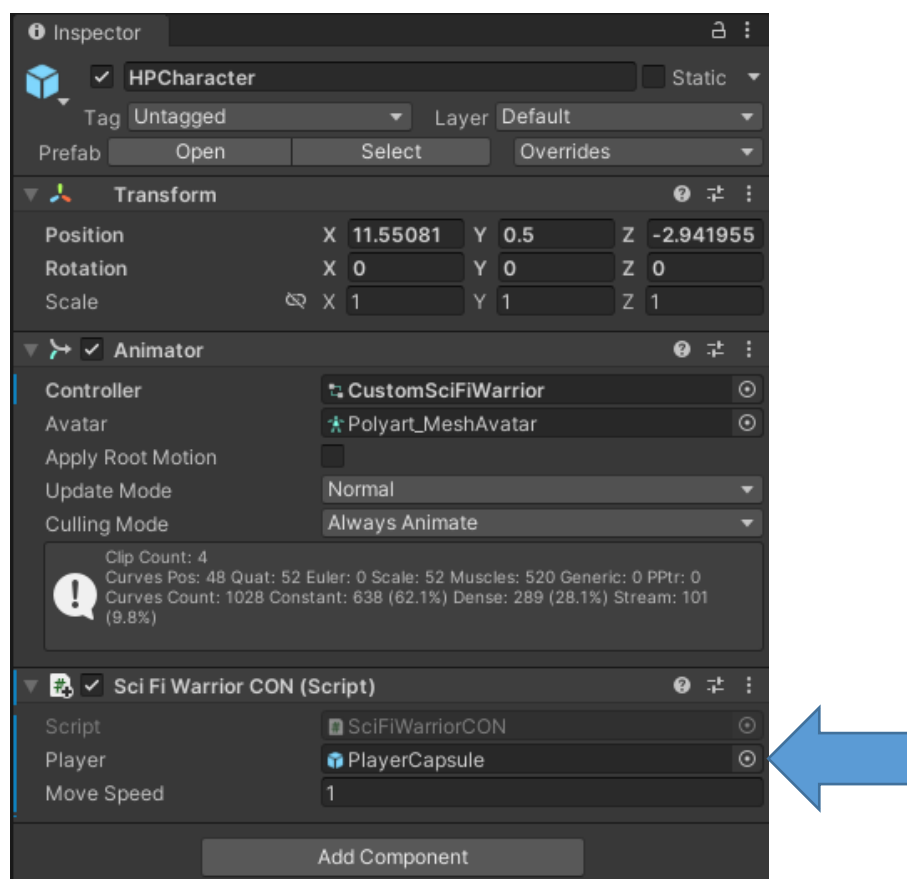
Next, we need to attach the script to our **HPCharacter** prefab.

In the Unity Editor, go to the hierarchy window, select the **HPCharacter** prefab.

Go to the inspector, click the “Add Component” button and search for the **SciFiWarriorCON** script. Click on the script to add it.

**Go to the hierarchy window and drag the PlayerCapsule prefab to the Player property of the SciFiWarriorCON script.**

Your **HPCharacter** prefab Inspector properties should look like the screenshot below.



**Image description:** The inspector for the HPCharacter prefab. The SciFiWarriorCON script has been added. The Player property of the SciFiWarriorCON script has been set to the PlayerCapsule.

**Save your scene.**

**🎮 We are now ready to playtest the scene.**

Press the play button to playtest your scene.

Use the W, A, S, D keys to move towards the **HPCharacter**. You should see the **HPCharacter** playing its idle animation. When you get close to the **HPCharacter** it should start walking towards the players camera. The **HPCharacter** should keep following the player until the player gets more than 7 units way. Your animated **HPCharacter** should look like the screenshot below.



**Image description:** The game window. The HPCharacter is walking towards the players camera.

## 8. Adjusting the Animation Properties

The **HPCharacter** prefab is a little slow when moving from idle to walking. We can adjust this speed.

Next, go to the project window.

**Navigate to the folder: SciFiWarriorPBRHPPolyart -> Animators.**

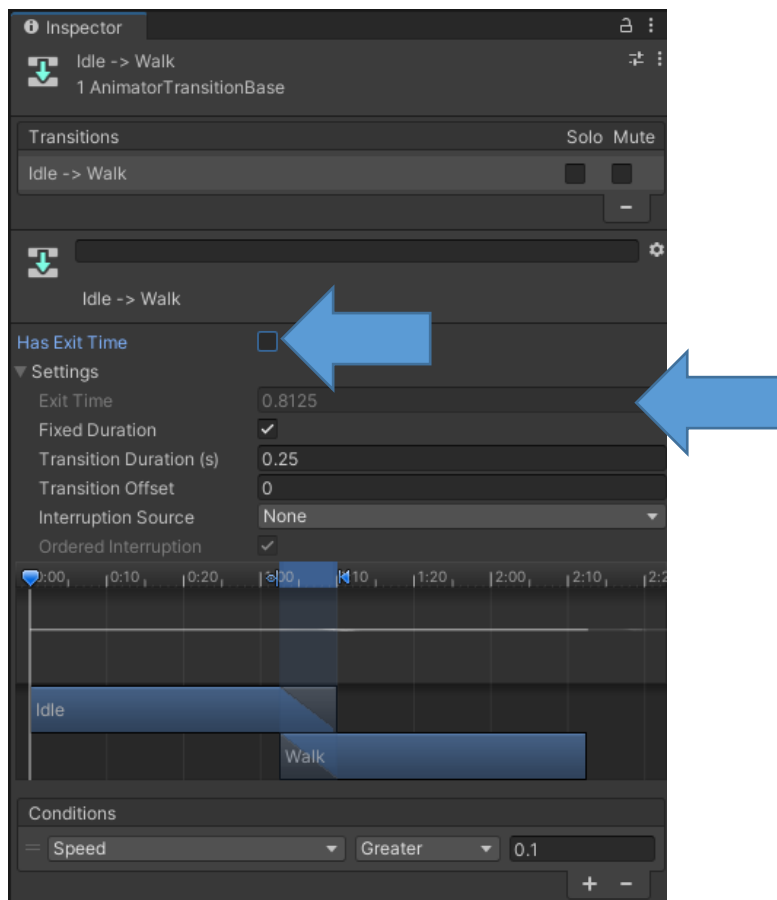
Double click on the CustomSciFiWarrior Animator Controller. The animator window should appear.

In the Animator window, select the Idle to Walk transition and go to the inspector.

**Find the Has Exit Time property and uncheck it.**

You can also click the grey triangle next to the setting text to reveal more of the Has Exit Time settings. See the screenshot below for an example.

In the screenshot below I have unchecked the Has Exit Time property. We can also see more settings for this property. For example, the Exit Time value. This value sets, as a percentage, how much of the animation must be played before it exits. In this case 81% or 0.81 because the number ranges from 0 to 1, must be played.



***Image description:*** The inspector for the Idle to Walk transition. The Has Exit Time property has been unchecked. The grey triangle next to the setting text has been clicked to reveal more of the Has Exit Time settings. The “Has Exit Time” and “Exit Time” properties have been highlighted.

**Repeat this process for all the transitions. Uncheck the “Has Exit Time” property for the following transition:**

Idle -> Walk

(We have already done this one. You do not need to create it again.)

Walk -> Idle



Walk -> Run  
Run -> Walk

Idle -> Dead  
Walk -> Dead  
Run -> Dead

Dead -> Idle

**Close the Animator window.**

**Save your scene.**

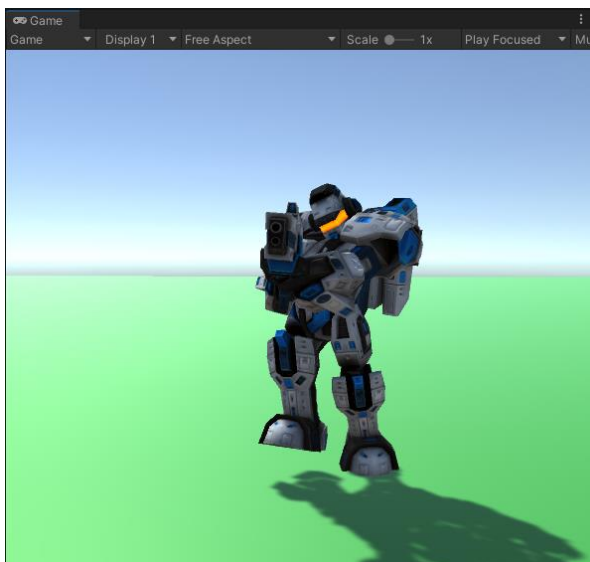
 **We are now ready to playtest the scene.**

Press the play button to playtest your scene.

Use the W, A, S, D keys to move towards the **HPCharacter**. You should see the **HPCharacter** playing its idle animation. When you get close to the **HPCharacter** it should start walking towards the players camera. The **HPCharacter** should keep following the player until the player gets more than 7 units way.

**In this playtest the animation transitions should appear more prompt.**

Your animated **HPCharacter** should look like the screenshot below.



**Image description:** The game window. The HPCharacter is walking towards the players camera.

## 9. Scripting a Virtual Character – Game Events

Unity passes control to a script intermittently by calling certain functions that are declared within it.

Once a function has finished executing, control is passed back to Unity.

These are event functions.

You have seen:

- Start.
- Update.

Others include:

- FixedUpdate - called just before each physics update.
- LateUpdate – called after update.
- OnCollisionEnter - Physics events. Called when a collision occurs.
- OnTriggerEnter – Called when another object collider enters the collider.

Let's explore the **OnTriggerEnter** event.

**First, we will add another collider that will check for overlapping events, rather than blocking another object. We will keep the collider we have already added. That will be still used for object blocking.**

Go to the hierarchy window and select the **HPCharacter** prefab.

**Go to the inspector window, click the Add Component button and add a Capsule Collider component.**

Set the Capsule Collider properties to the following setting:

Is Trigger     True.

Center        X: 0, Y: 1, Z: 0.

Radius        0.6.

Height        2.

Your **HPCharacter** prefab should now have **two** green capsule colliders around it. Your scene should look like the screenshot below.



**Image description:** The HPCharacter prefab. We can see two green capsule colliders.

Open the script **SciFiWarriorCON**.

Update the code in the script file to match the code below.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class SciFiWarriorCON : MonoBehaviour
{
    // A reference to the player GameObject.
    [SerializeField]
    private GameObject player;

    // The speed of the character.
    [SerializeField]
    private float moveSpeed = 1.0f;

    // A reference to the animator.
    private Animator anim;

    // Store the current AnimatorStateInfo.
    AnimatorStateInfo currentStateInfo;

    // Hashes to animator parameters.
    private int speedHash = Animator.StringToHash("Speed");
    private int dyingHash = Animator.StringToHash("IsDead");

    private float WaitTime = 3;
    private float WaitTimer = 0;

    // Start is called before the first frame update
    void Start()
    {
        anim = GetComponent<Animator>();
    }
}
```

```

// Update is called once per frame
void Update()
{
    currentStateInfo = anim.GetCurrentAnimatorStateInfo(0);
    if (currentStateInfo.IsName("Dead"))
    {
        // The GameObject is in the dead state.
        // Start the timer.
        WaitTimer += Time.deltaTime;
        if (WaitTimer > WaitTime)
        {
            anim.SetBool("IsDead", false);
            WaitTimer = 0;
        }
    }

    if (Vector3.Distance(transform.position, player.transform.position) < 7)
    {
        anim.SetFloat(speedHash, 0.7f);
        Vector3 tmp = player.transform.position;
        tmp.y = this.transform.position.y;

        transform.LookAt(tmp, Vector3.up);

        currentStateInfo = anim.GetCurrentAnimatorStateInfo(0);
        if (currentStateInfo.IsName("Walk"))
        {
            transform.Translate(Vector3.forward * moveSpeed * Time.deltaTime);
        }
    }
    else
    {
        anim.SetFloat(speedHash, 0);
    }
}

void OnTriggerEnter(Collider other)
{
    Debug.Log("OnTriggerEnter - " + other.gameObject.name);

    if (other.gameObject.tag.Contains("Projectile"))
    {
        anim.SetBool(dyingHash, true);
        Destroy(other.gameObject);
    }
}
}

```

In the code above we add two class level variables to manage a simple timer in the script. We add code to the update method to check if the GameObject is in the dead state. If this is the case a timer is started. When the timer finishes the Animator controller IsDead parameter is set to false.

We have also added a **OnTriggerEnter** method that is called when an object overlaps the GameObjects collider. In this function we check the overlapping objects tag. If the

tag is a projectile, we set the IsDead parameter for the animator controller and destroy the projectile object.

**Save the script in Visual Studio and go back to Unity.**

**In next sections we will create a simple projectile that will trigger the dying animation for the animated character.**

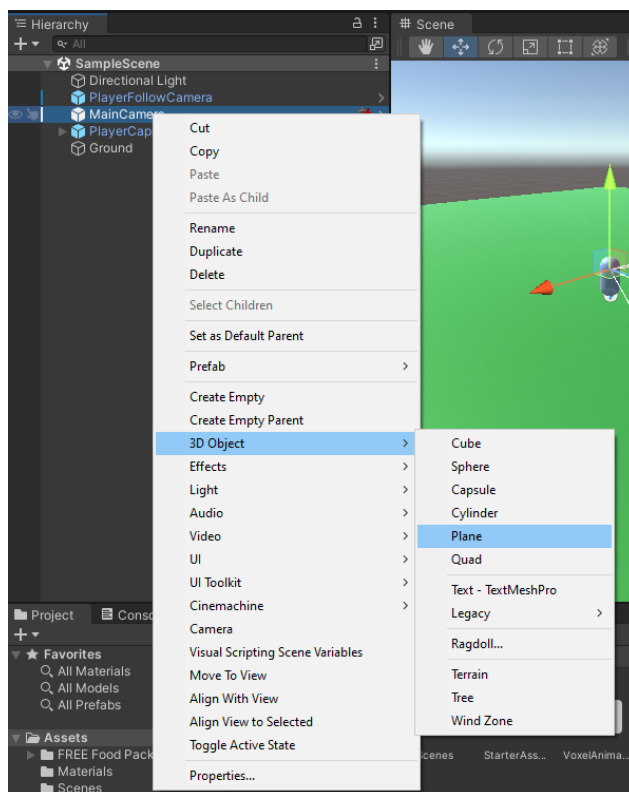
## 10. Adding a Crosshair (i.e., target) to the First-Person Camera

**We will now add a target to the first-person camera. This will allow a player to select things more easily in the scene or target when firing projectiles.**

Go to the Hierarchy Window.

**Right-click on the MainCamera asset. Go to the 3D Object submenu and select plane.**

See the screenshot below for an example.



***Image description:*** Right-click on the MainCamera asset. Go to the 3D Object submenu and select plane.

A plane should be created as a child object of the **MainCamera** asset.

**Note, in maths, a plane is a flat, two-dimensional (2D) surface.**

**Give the plane the name Crosshair.**

Rotate the plane -90 on the x axis, scale it to 0.008 on the x, y and z axis and set the z axis position to 1. Your transform component for the plane should now have the following position, rotation, and scale properties:

**Position:**

X: 0  
Y: 0  
Z: 1

**Rotation:**

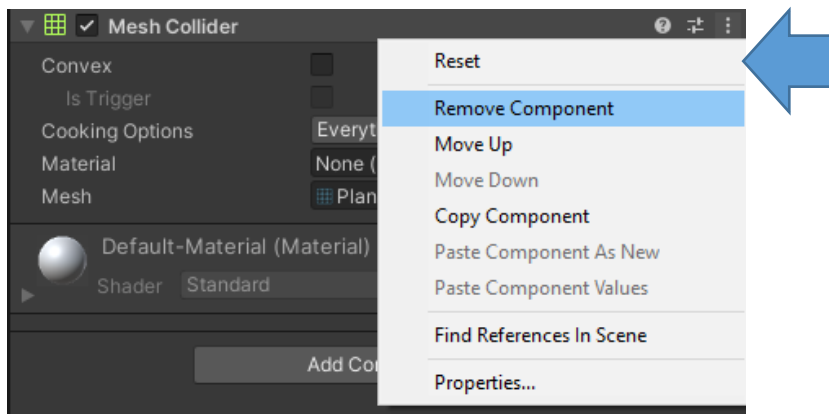
X: -90  
Y: 0  
Z: 0

**Scale:**

X: 0.008  
Y: 0.008  
Z: 0.008

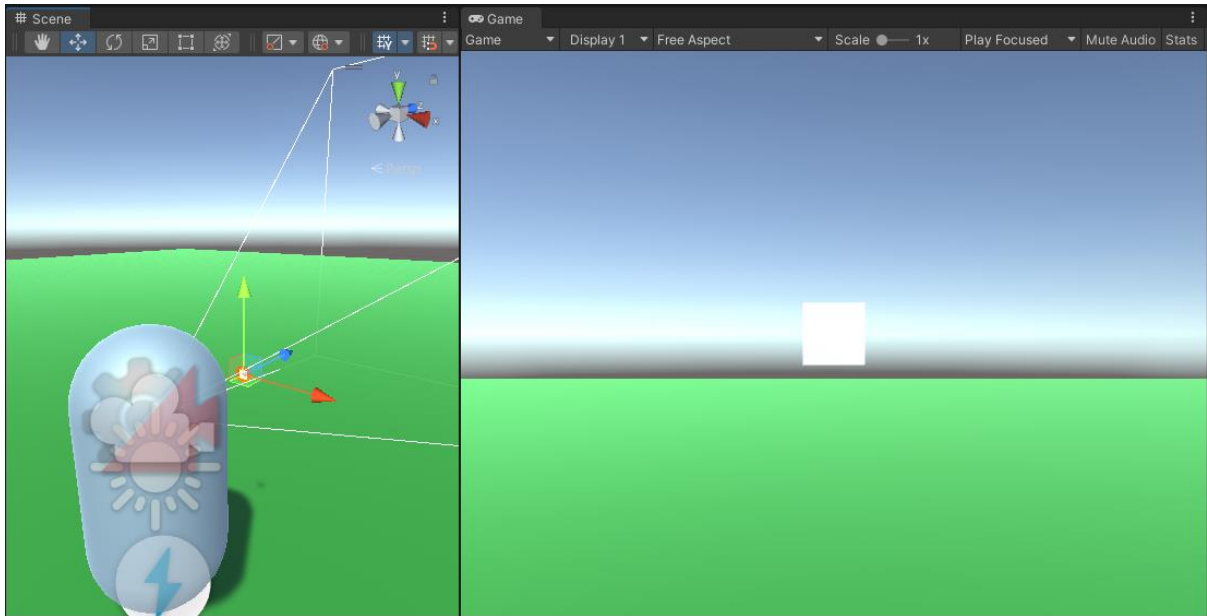
Also, remove the mesh collider component from the plane. Do this by going to the inspector and finding the Mesh Collider component.

Then, click the three dots on the right side of the mesh collider text. Then click Remove Component. See the screenshot below for an example.



**Image description:** Find the Mesh Collider component on the crosshair plane. Click the three dots on the right side of the mesh collider text. Then click Remove Component.

You should now have a plane in the middle of the screen that is facing the camera. See the screenshot below for an example.



**Image description:** A plane in the middle of the screen that is facing the camera. In the Scene window on the left of the image we can see the plane position in front of the camera and first-person capsule. The plane is selected in the scene.

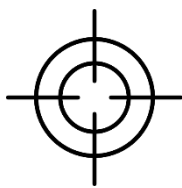
**Next, we need to add a crosshair texture.**

You can search the internet for a crosshair. In general, you need a crosshair with a transparent background.

**Here is the crosshair I found:**

[https://www.flaticon.com/free-icon/crosshair\\_865405](https://www.flaticon.com/free-icon/crosshair_865405)

*Crosshair icons created by Good Ware.*



I downloaded the 512px png.

**You can use this crosshair too or you can find your own on the internet.**

Download your crosshair texture.

Go to the Unity Editor.

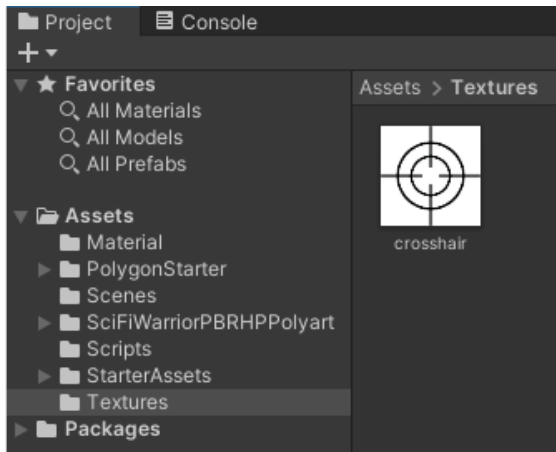
Go to the Project Window. Select the Assets folder. Right click in the righthand panel of the Assets folder and select Create -> Folder.

When the folder is created give it the name **Textures**.

**Double click on the Textures folder to open it.**

**Next, we will import the image you have downloaded into your Unity project by dragging it from the saved location to the Assets window (which should be open in the Texture folder).**

Your Texture folder should look like the screenshot below.

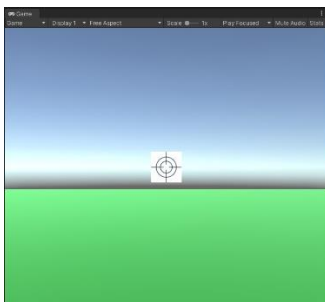


**Image description:** The project window. A Textures folder has been created and opened. A texture has been added to the folder.

**Drag your crosshair texture from the project window onto the crosshair plane in the hierarchy.**

When you drag your crosshair texture from the Project window onto the crosshair plane in the hierarchy Unity will automatically create a material for the crosshair in a Materials folder.

**The crosshair texture should now be applied to the crosshair plane.** See the screenshot below for an example.



**Image description:** The game window showing the crosshair texture applied to the crosshair plane.

You will probably have a white border around your crosshair. We want to remove this.

**Go to the hierarchy and click on the Crosshair plane.**

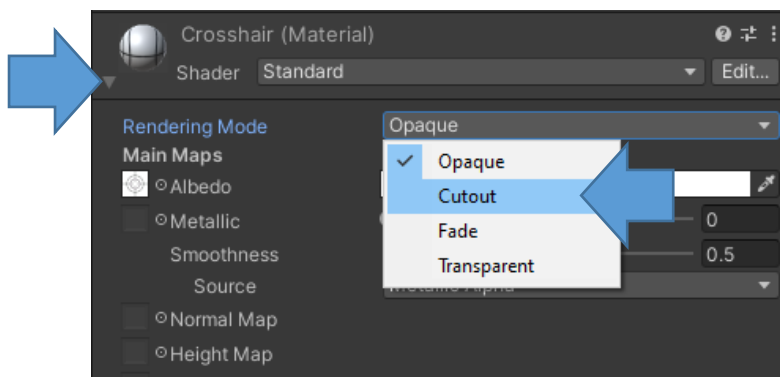


Go to the inspector and find the Crosshair material.

Click on the triangle on the left side of the word “Shader”. This will expose more material properties.

**Find the Rendering Mode dropdown list and select Cutout from the list.**

See the screenshot below for an example.



**Image description:** The material component for the crosshair plane. Click on the triangle on the left side of the word “Shader”. Find the Rendering Mode dropdown list and select Cutout from the list.

The background to the crosshair should now be transparent. See the screenshot below for an example.



**Image description:** The game window showing the crosshair texture applied to the crosshair plane. The background is now transparent.

**Save the scene.**

**🎮 We are now ready to playtest the scene.**

Press the play button to playtest your scene.

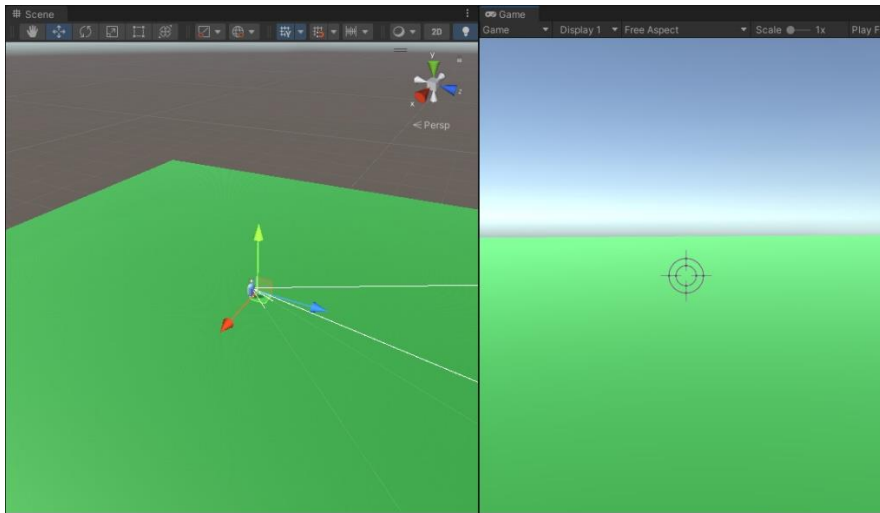
**You should be able to walk around the environment. You should have a crosshair at the centre of your camera in the game window.**

The default controls:

- Arrow keys or WASD to move. Spacebar to jump. Hold down shift to sprint.

- Mouse look.

See the screenshot below for an example.



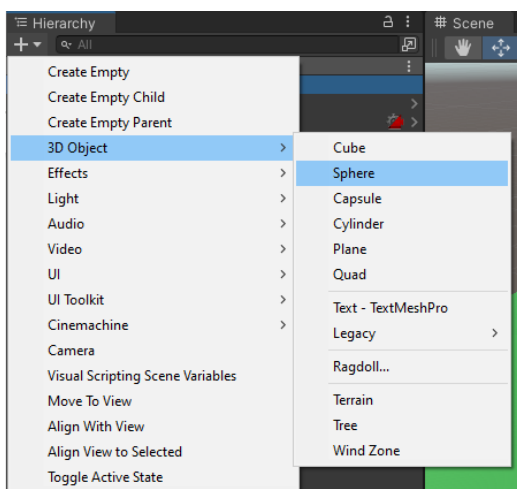
**Image description:** A playtest of the scene using the first-person controller. Note, in my setup I have the Scene and Game view side-by-side.

When you have finished walking around stop playtesting your scene.

## 11. Creating a Simple Projectile - Sphere

We will now add a projectile to the first-person controller. This will allow us to explore further coding in Unity.

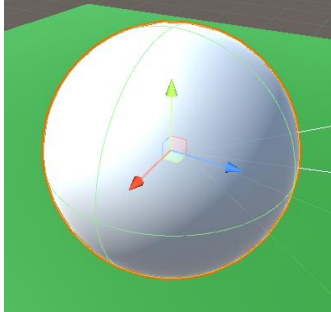
Go to the hierarchy window and click the plus (+) menu. Select 3D Object -> Sphere. See the screenshot below for an example.



**Image description:** The Hierarchy Window and the plus (+) menu. Sphere has been selected in the menu.

Call the sphere **Projectile**.

Select the sphere object in the hierarchy, move the cursor over the scene window and press F. This should focus your view on the sphere. See the screenshot below for an example.



**Image description:** The sphere projectile has been selected. I have pressed the F key to focus on the sphere. The scene view camera moves close to the sphere.

Next, we will add a material to the sphere.

**Go to the Assets -> materials folder.**

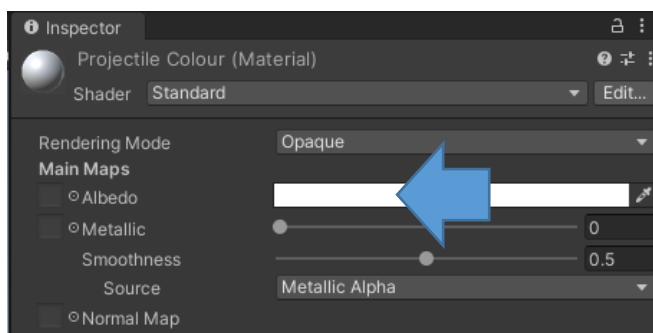
**Right click in the Materials folder and go to Create -> Material.**

Give the Material the name **ProjectileColour**.

Select the material in the Project window. You should be able to see its properties in the inspector.

Go to the inspector, click on the block next to the Albedo property at top of the Inspector.

See the screenshot below for an example.



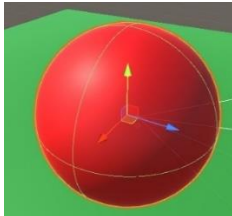
**Image description:** The inspector window for the material. Click on the white box next to the Albedo property.

**Clicking on the block next to the Albedo property at top of the Inspector will open a colour picker window.**

**Select a red colour. I entered the hexadecimal value: FF0000.**

To assign the material, drag the **ProjectileColour** material from the Project window and drop it onto the name of the sphere object in the hierarchy.

Your sphere should now match the colour of your material. See the screenshot below for an example.



***Image description:** The sphere projectile has been selected. The projectile colour material has been applied to the sphere.*

**Next, we will set the physics engine to control the projectile by adding a rigidbody component to the sphere.**

Select the Sphere in the hierarchy.

Go to the Inspector and click the **add component** button. Search for the rigidbody. Select rigidbody from the list to add the component.

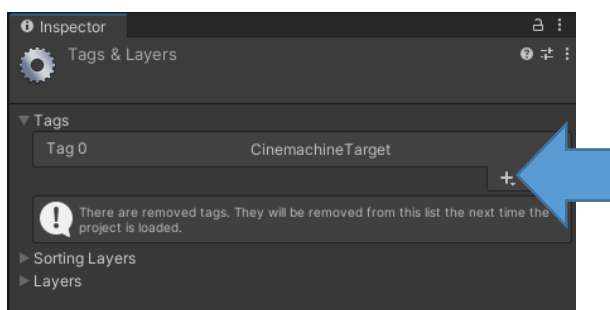
**Next, we will give the projectile a tag of Projectile.**

Make sure the projectile sphere is still selected in the hierarchy.

Go to the Inspector. Click the tag dropdown menu and select “Add Tag...”.

The Inspector will change to a “Tags & Layers” window.

Click the plus (+) icon in the Tags section. See the screenshot below for an example.



***Image description:** When you click the “Add Tag...” option the Inspector will change to a “Tags & Layers” list. Click the plus (+) icon in the Tags section.*

Enter the name **Projectile** and **click save**.

“Tags & Layers” window should now have the **Projectile** tag in it.

**Select the sphere projectile again in the Hierarchy.**

The inspector should now have the sphere’s properties again.

Go to the inspector. Click the tag dropdown menu and select the **Projectile** tag.

**Next, we want the projectile to be stored and instantiated when a key is pressed. We do not want the projectile in the scene by default.**

**Therefore, we need to store the object as a prefab and instantiate it when a key is pressed.**

Go to the Project Window. Select the Assets folder. Right click in the righthand panel of the Assets folder and select Create -> Folder.

When the folder is created give it the name **Prefabs**.

**Next, drag the sphere from the hierarchy and drop it into the Prefabs folder in the project window.**

**Save the scene.** Go to the File menu and find the save option.

**You can now delete the original sphere object in the scene / from hierarchy window. Do not delete the Projectile in the project window.**

**Save the scene.**

## 12. Creating a Projectile Launcher Script

Next, we will add a script to our Unity project to launch (e.g., fire) our projectile.

Select the Assets folder in the project window.

Go to the Assets -> Scripts folder.

Right click in the Scripts folder and go to Create -> C# Scripts.

When the script has been created give it the name **ProjectileLauncher**.

Double click on your **ProjectileLauncher** C# script to open it in Visual Studio.

Update the code in the script file to match the code below.

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class ProjectileLauncher : MonoBehaviour
{
    [SerializeField]
    private Rigidbody projectileRigidBody;
    [SerializeField]
    private float projectilePower = 1500;
    [SerializeField]
    private GameObject muzzle;

    [SerializeField]
    private float COOLDOWN_TIME = 0.5f;
    private float coolDown = 0;

    // Start is called before the first frame update
    void Start()
    {
    }

    // Update is called once per frame
    void Update()
    {
        if (coolDown <= 0)
        {
            if (Input.GetButtonUp("Fire1"))
            {
                coolDown = COOLDOWN_TIME;

                // Instantiate the projectile.
                Rigidbody aInstance = Instantiate(projectileRigidBody,
                    muzzle.transform.position, transform.rotation) as Rigidbody;

                // Add force.
                Vector3 forward = transform.TransformDirection(Vector3.forward);
                aInstance.AddForce(forward * projectilePower);

                // Destroy the object after X seconds.
                Destroy(aInstance.gameObject, 8);
            }
        }
        else
        {
            coolDown = coolDown - Time.deltaTime;
        }
    }
}

```

In the code above we add two class level variables to the script:

```
private Rigidbody projectileRigidBody;  
private float projectilePower = 1500;
```

These variables will store a reference to the GameObjects Rigidbody component and store a float value that represents the projectile power.

We also add another variable that stores the GameObject that represents the muzzle position of the weapon firing the projectile.

There are also two more variables that handle how quickly the player / user can fire a projectile. The time is set in seconds in the COOLDOWN\_TIME variable.

In the update method we add code to check if the cooldown counter is equal to zero or less than zero. If this is true, we check if the Fire1 button has been pressed. If this is true, we create an instance of the project using the Rigidbody component variable, then add a force to it to make it move. Finally, we tell it to destroy itself in 8 seconds. If the cooldown counter is not equal to zero or less than zero, we decrease it by the amount of time since the update method was last called.

**Save the script in Visual Studio and go back to Unity.**

Next, we need to attach the script to the camera.

**In the Unity Editor, go to the hierarchy window and select the MainCamera GameObject.**

Go to the Inspector. In the Inspector click the **Add Component button**, which is at the bottom of the window. Search for the **ProjectileLauncher** script and then double click on it to add it.

Next, we need to set some properties in the **ProjectileLauncher** component.

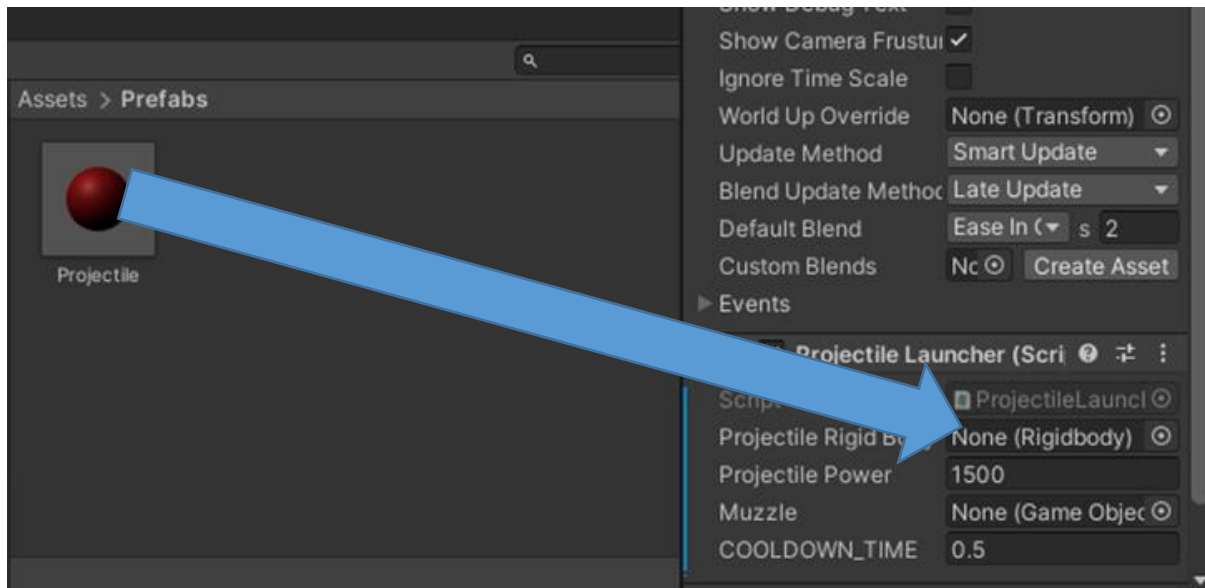
In the Inspector, find the **ProjectileLauncher** component.

We also need to assign the Projectile prefab to the script variable **projectileRigidbody** and we need to assign the **muzzle** GameObject.

To do this, drag the **Projectile prefab** from the project window and drop it onto the **projectileRigidbody** variable in the inspector.

See the screenshot below for an example.





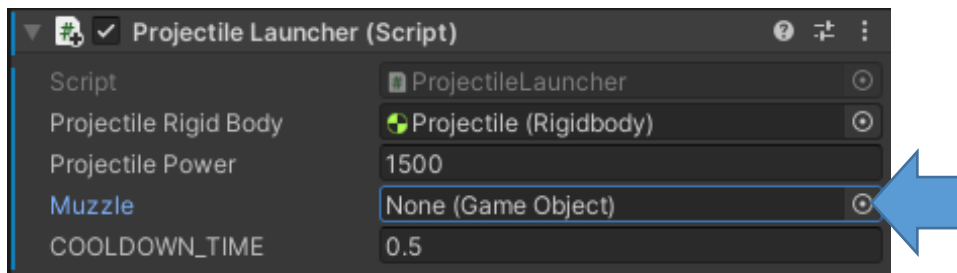
**Image description:** drag the Projectile prefab from the project window and drop it onto the projectileRigidbody variable in the inspector.

Next, we set the muzzle variable.

Go to the hierarchy, select the MainCamera GameObject.

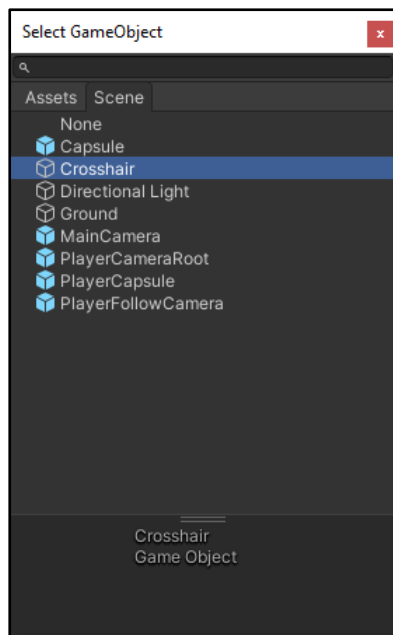
Go to the inspector and find the **Projectile Launcher** component.

Click the bullet icon next to the Muzzle variable. See the screenshot below for an example.



**Image description:** Click the bullet icon next to the Muzzle variable.

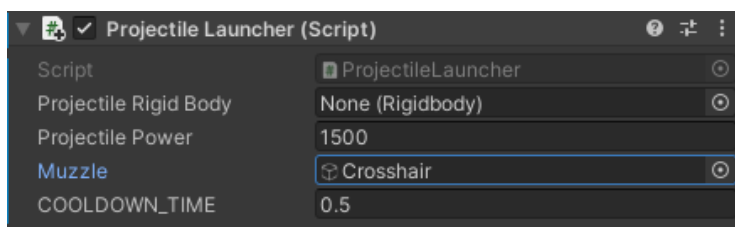
A "Select GameObject" window will appear. Click the Scene tab and select the Crosshair from the list. See the screenshot below for an example.



**Image description:** Click the Scene tab and select the Crosshair from the list. Do this by double clicking on the Crosshair GameObject to select it.

**Double click on the Crosshair GameObject to select it.**

The **Projectile Launcher** component for the **MainCamera** GameObject should now look like the screenshot below. The **projectileRigidbody** variable and **muzzle** variable have been set.



**Image description:** The Projectile Launcher component for the MainCamera GameObject.

**Save the scene.**

## 13. Firing a Projectile and Testing the Dying Animation

In this section we will see all the elements we have created come together. We will fire a projectile at the animated character. When the projectile hits the character, it will play the dying animation.

🎮 We are now ready to playtest the scene.

Press the play button to playtest your scene.

Use the W, A, S, D keys to move towards the **HPCharacter**. You should see the **HPCharacter** playing its idle animation. When you get close to the **HPCharacter** it should start walking towards the players camera. The **HPCharacter** should keep following the player until the player gets more than 7 units way.

Press the left mouse button to fire a projectile. You should now be able to fire a sphere projectile. The projectiles will disappear after 8 seconds.

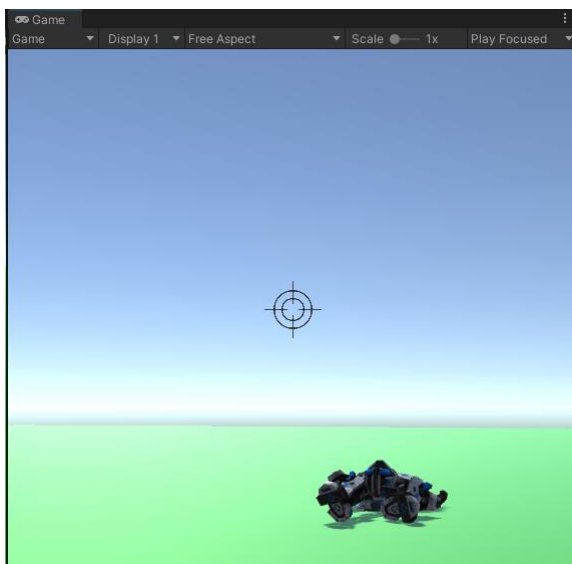
If a projectile hits the **HPCharacter GameObject** it should play the dying animation. This means it should fall over.

If it does not work, please review the previous steps and your code.

The default controls:

- Arrow keys or WASD to move. Spacebar to jump. Hold down shift to sprint.
- Mouse look.

See the screenshot below for an example.



**Image description:** A playtest of the scene using the first-person controller. The animated character has been hit by a projectile and the dying animation has been played and has finished.

## 14. Further Tasks

⚠ **ALERT - Useful Tip** ⚠ These tasks and any tasks beyond this point can be considered optional if you are running behind with things (e.g., it has taken you over a week to complete this workbook). If you are behind, I would recommend you stop this workbook here and move onto the next workbook. If you are not behind, I recommend you complete the remaining tasks as they will help improve your knowledge. Please speak to a member of the module team if you have any questions.

🚀 Please complete these further tasks:

1. Make the distance at which the **HPCharacter** starts charging the human player further away. Explore at least 3 different values to see what effect it has.
2. Update the code in the **SciFiWarriorCON** script so that the **HPCharacter** starts charging the player (i.e., running) for 2 seconds.
3. Update the **SciFiWarriorCON** script so that the **HPCharacter** respawns to idle after 5 second of being dead.
4. Repeat this workbook again using a different free asset that includes animated characters by the publisher “Dungeon Mason” on the Unity asset store. The steps in this workbook **should** still work, but some properties will be different (e.g., a Capsule Collider radius might need different values). Here is a link - <https://assetstore.unity.com/publishers/23554>

> **END OF STUDENT WORKBOOK** ■