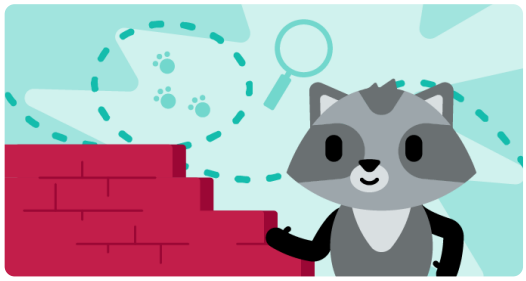
Explore a 3D world

Create a 3D world in Unity and explore it with an animated 3D player character



Step 1 Introduction

Create a 3D world in Unity and explore it with an animated 3D player character.

This project is brought to you with generous support from **Unity Technologies** (https://unity.com/). These **projects** (https://projects.raspberrypi.org/en/pathways/unity-intro) offer young people the opportunity to take their first steps in creating virtual worlds using Real-Time 3D.

> **Unity** is a development environment for making games, virtual environments, visual novels, digital animations, and more. You can use Unity to develop 2D and 3D cross-platform games for PCs, consoles, mobile devices, and the internet.

You will:

- Understand the Unity environment including the Scene editor
- Add 3D objects with materials as scenery
- Create a player character with animations that can move around

Recommended previous experience

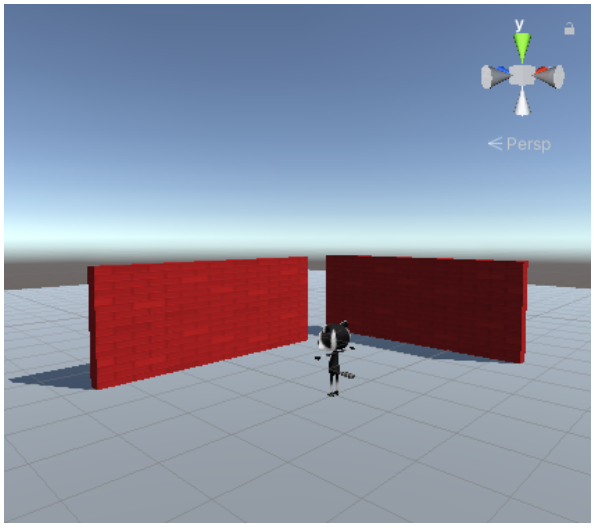
This project is suitable for learners with previous coding experience, such as **Scratch** (https://projects.raspberrypi.org/en/pathways/scratch-intro) or **Python** (https://projects.raspberrypi.org/en/pathways/python-intro), who are comfortable typing text-based code.

You will need

This project requires the Unity Editor, which can be installed from the Unity Hub. This is a large download and install, so we recommend you install it before starting this project.

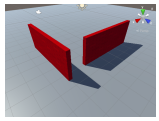
You can follow our **Unity guide** (https://projects.raspberrypi.org/en/projects/unity-guide) to install Unity Hub and the Unity Editor for your operating system.

You will also need to download the **Unity starter package** (https://rpf.io/p/en/explore-a-3d-world-go) of assets before starting.


y
Persp

Step 2 Set the 3D scene

Your 3D world, or 'map', needs a floor and walls.



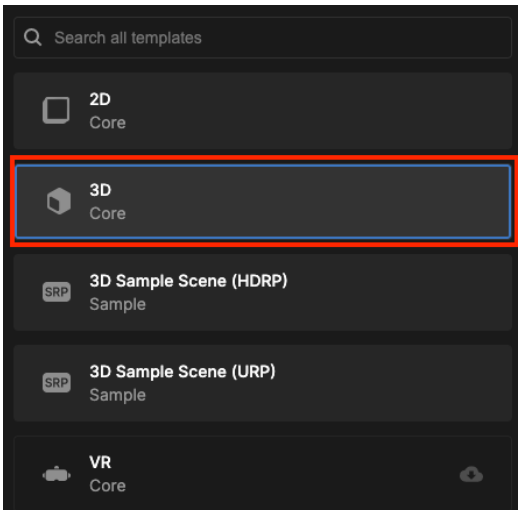
> People are spending more time in **online virtual environments**. As well as playing games, people relax, explore, socialise, learn, and participate in interactive entertainment. Some people call the future of these environments the **metaverse**. Being able to design 3D worlds is an important skill.

A Unity project needs graphics and sound 'Assets'.

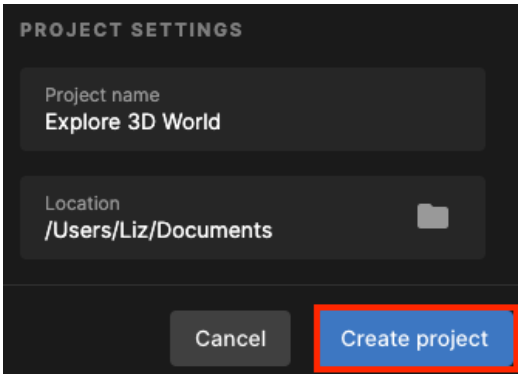
Download and unzip the **Unity starter package** (https://rpf.io/p/en/explore-a-3d-world-go) to your computer. Choose a sensible location such as your Documents folder.

Launch the Unity Hub and click **Projects** then select **New project**:

From the list choose **All templates** then select **3D Core**:

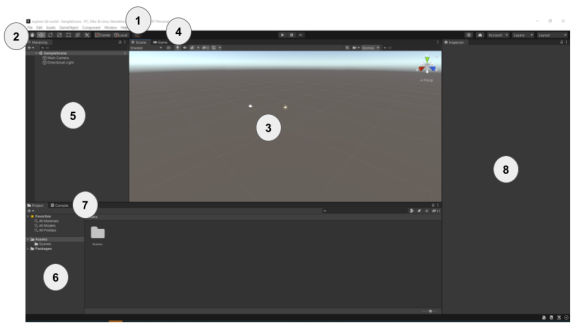


Edit the project settings to give your project a sensible name and save it to a sensible location. Then click **Create project**:



Your new project will open in the Unity Editor. It may take some time to load.

The Unity Editor looks like this:



The Unity Editor windows and views

1. **The Unity menu** is used to import, open, and save scenes and projects. You can amend your Unity Editor preferences and add new GameObjects and components.

2. **The Toolbar** contains tools for navigating in the Scene view, controlling play in the Game view, and customising your Unity Editor layout.

3. **The Scene view** is used to navigate and edit your Scene. You can select and position GameObjects including characters, scenery, cameras, and lights.

4. **The Game view** can be accessed by clicking on the **Game** tab. It shows the scene as it looks through the lens of your cameras. When you click on the **Play** button to enter Play mode, the Game view simulates your scene as it would be viewed by a user.

5. **The Hierarchy window** shows all the GameObjects in your Scene and the structure between them. Here, you can add and navigate the GameObjects in your project. GameObjects can have 'child objects' that move with them.

6. **The Project window** shows a library of all the files included in your project. You can find the Assets you want to use here.

7. **The Console window** can be accessed by clicking on the **Console** tab. It shows important messages. This is where you can see compiler errors (errors in your Script) and messages that you print using `Debug.Log()`.

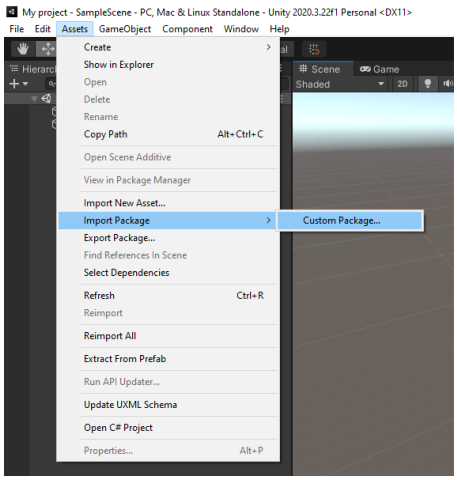
8. **The Inspector window** allows you to view and edit the properties of GameObjects. You can add other components to your GameObjects and edit the values they use.

The Unity starter package you downloaded contains a number of Assets for you to use in your project.

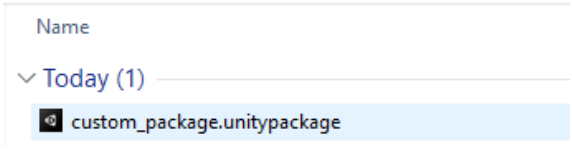
To import them into your new project, click on the **Assets** menu and select **Import package** > **Custom Package…** then navigate to the downloaded **Unity starter package**.

Import a package

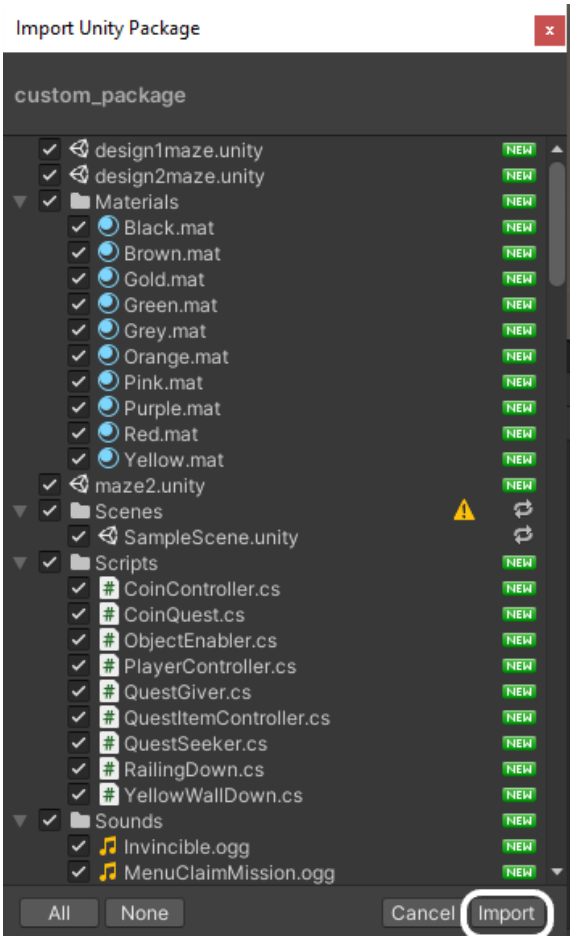
Import your downloaded package.



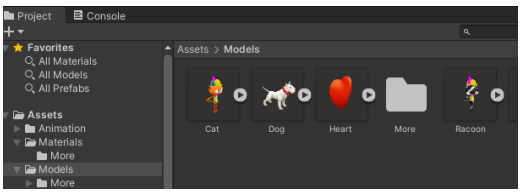
Select the location of your downloaded package.



Click on the **Import** button to import all of the package.

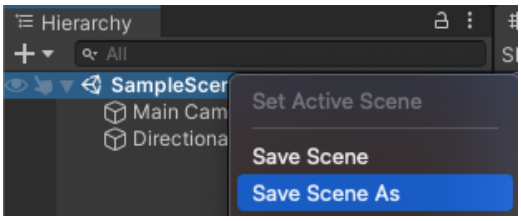


The **Project window** is where you can see all the files included in your project. Click on the **Models** folder in the Assets folder to see the models you have imported.

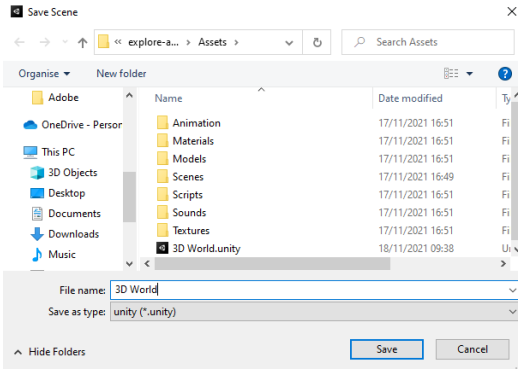


In Unity, a **Scene** contains GameObjects. A Unity project with multiple game levels might have one scene per level.

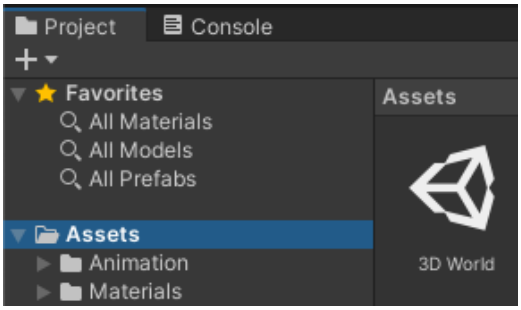
Right-click on **SampleScene** in the Hierarchy and choose **Save Scene As**.



In the pop-up window, name your Scene `3D World`:

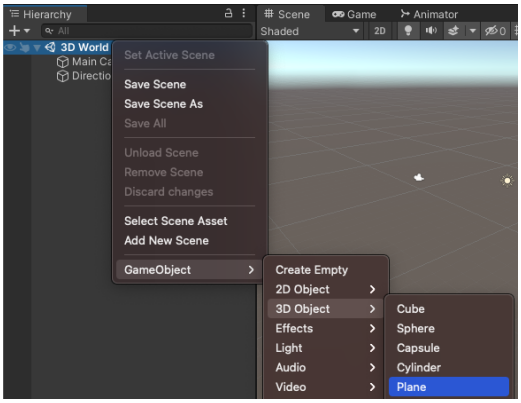


A new file will appear in the Assets folder in the Project window:



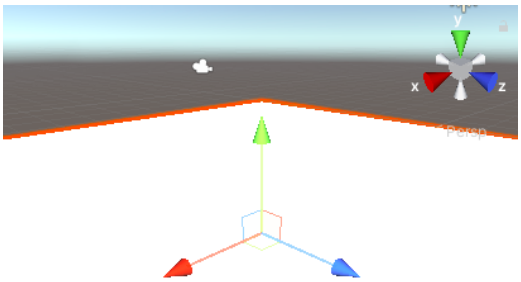
First, your world needs some ground.

Right-click on your scene (name 3D World) in the Hierarchy window and choose **GameObject** > **3D Object** > **Plane**:



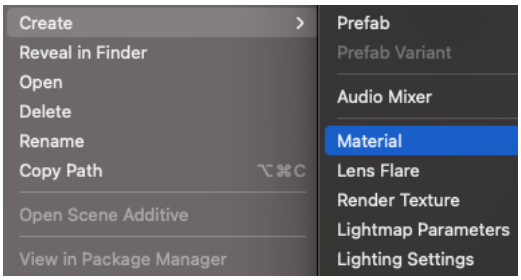
This will create a ground for your world.

The default size for the plane is 10m × 10m. Unity uses metres as the unit of measurement.



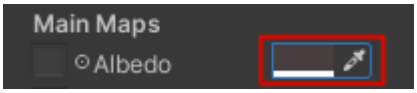
The **material** of a GameObject controls how it looks. Give the plane a different colour material.

In the Project window, right-click on the **Materials** folder and choose **Create** > **Material**.

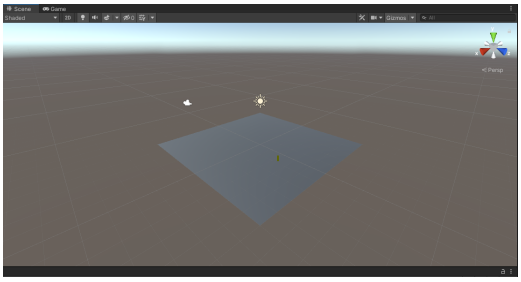


A new material should appear in the Materials folder. Decide what colour you will use for your floor and name your new material:

Click on the colour next to 'Albedo' in the Inspector window and choose a colour for your material (we used grey):



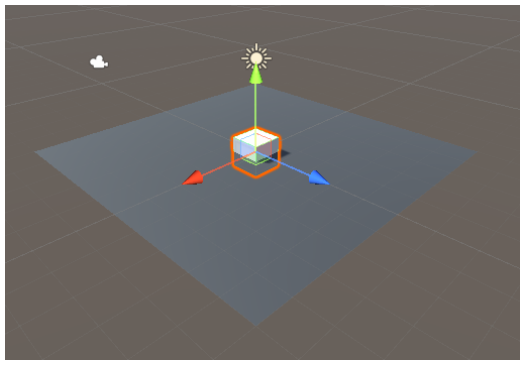
Drag your new material from the Project window to your plane in the Scene view:



You can create objects from 3D shapes.

Right-click on your **3D World** scene in the Hierarchy window and choose **GameObject** > **3D Object** > **Cube**.

This will create a cube at the centre of the scene, at (0, 0, 0).



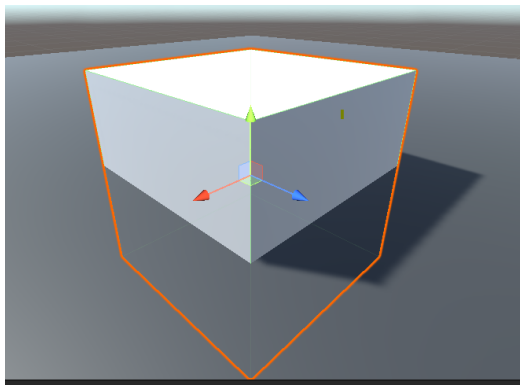
You can see the cube in the Scene view. This is the behind-the-scenes view of your game where you set everything up.

Tip: Click on the **Scene** tab to make sure you can see the Scene view.

Click on the cube in the Scene view or Hierarchy window to select it.

Use `Shift`+`F` (hold down the `Shift` key and tap `F`) to focus on the cube.

You can also use the scroll wheel on the mouse, or the up and down arrow keys, to zoom in and out:

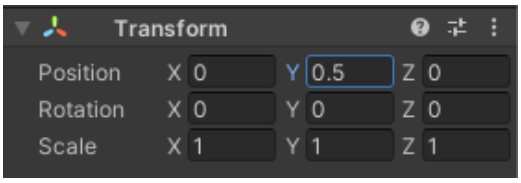


You need to get the cube to sit on the plane.

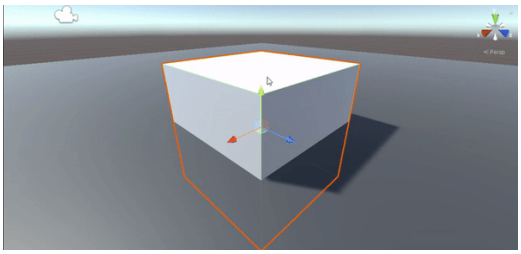
Click on the cube in the Scene view or Hierarchy window to select it.

Choose
You can either:

- Change the y position in the Inspector window to 0.5 (half the height of the cube):

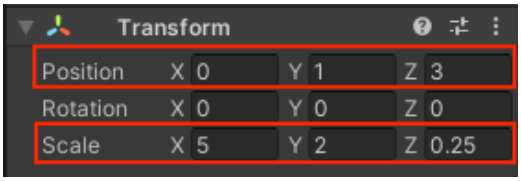


- Use the Move tool to drag the green arrow up until the cube sits on the plane:



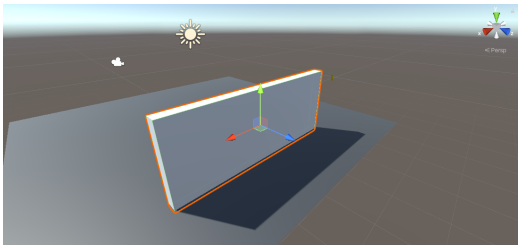
Tip: If you make a mistake in the Unity Editor, you can use `Ctrl`+`Z` (or `Cmd`+`Z`) to **undo** your last action.

Now change the cube into a wall with the following Position and Scale settings:



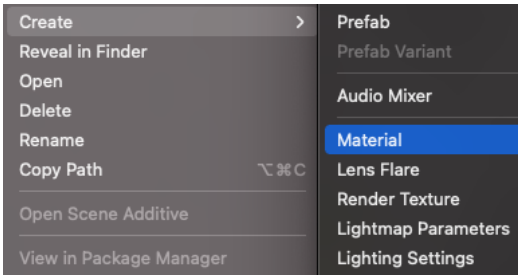
You can either enter the values in the Transform component for the cube or click on the Scale tool and drag the handles in the Scene view (this will update the Transform values.)

Zoom out to see your wall:



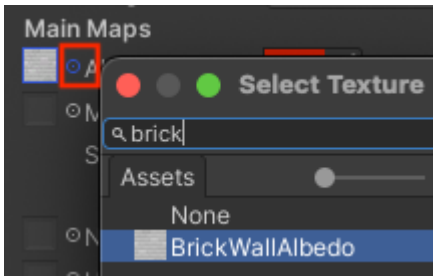
A material can have a colour and a texture and there are lots of properties that you can adjust to get different effects. A **texture** is a 2D image that can be created in an image editor.

In the Project window, right-click on the **Materials** folder and choose **Create** > **Material**. You are going to create a coloured brick wall. Give the material a descriptive name:



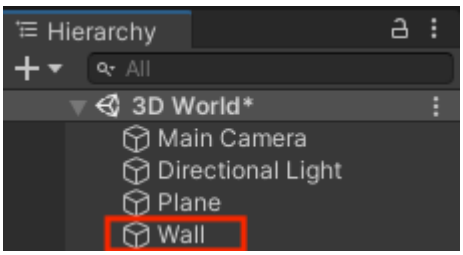
Click on the colour next to 'Albedo' in the Inspector window and choose a colour for your material:

Add a texture by clicking on the circle to the left of 'Albedo' and selecting **BrickWallAlbedo** texture from the list:



Drag your new material from the Project window to your wall in the Scene view:

In the Inspector window, right-click on your cube, choose **Rename** from the menu and rename your object from `Cube` to `Wall`:



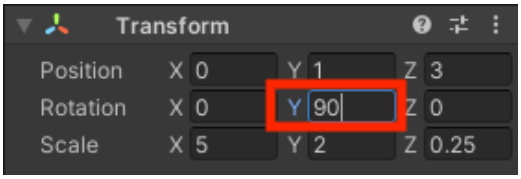
Tip: You can name a new GameObject in the Hierarchy window when you create it and you can change the name in the Inspector window.

To create a copy of your wall, you can either:

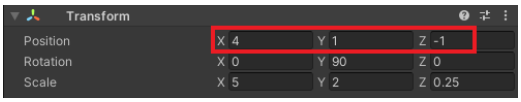
- Right-click on your Wall object in the Hierarchy window and choose **Duplicate**
- Select your wall in the Scene view and use `Ctrl+D` (or `Cmd+D`) to duplicate

Your new wall will be in exactly the same place as your first wall.

Change the y rotation of the new wall to `90`:

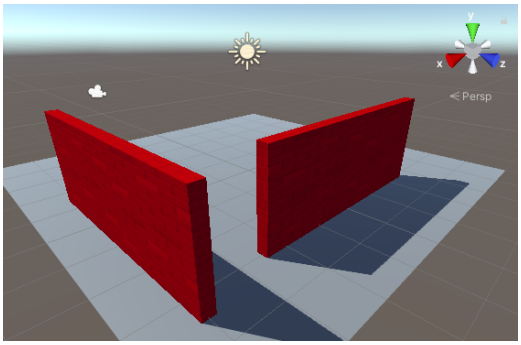


Reposition the new wall to the following position: x = 4, y = 1, z = -1.

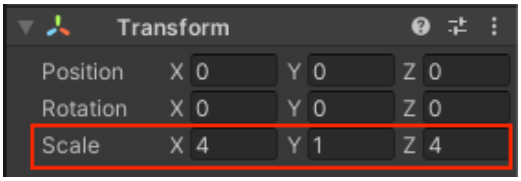


You can either enter the values in the Inspector window or drag the arrows in your Scene — it doesn't matter if the position is exact.

Your Scene should look like this:



Click on your Plane. Change the Scale settings on the Plane to make it bigger so you have more room:



Think of a 4×4 plane as 40 metres by 40 metres in the real world: plenty of room for your character to move around.

When you have unsaved changes, you will see a '*' next to your Scene in the Hierarchy window.

Save your Scene by clicking **File** > **Save**. Or use `Ctrl+S`.

Also, save your project by clicking **File** > **Save Project**.

Unity does not normally autosave changes, but your starter project contains a script to autosave your project every 60 seconds.

You can navigate around your scene to see it from different angles. If you get lost, just click on a wall in the Hierarchy window and then use `Shift+F` to focus and then zoom out:

Moving in the Scene view

To use flythrough mode, hold down the **right** mouse button **and use**:

- WASD to move around
- QE for up and down
- Shift to go faster
- The mouse to rotate the camera

Use the scroll wheel on your mouse to zoom in and out. Your trackpad may also have a scroll motion.

To move the scene around, hold the `ALT` key and the middle mouse button and then drag to move. You can also use the arrow keys to move around.

To focus on an object, click on the GameObject in the Scene view and then click `F`.

You can also click on an object in the Hierarchy window and then click `Shift+F` to focus on that GameObject in the Scene view.

Tip: If you get lost, click on your Player in the Hierarchy window and then `Shift+F` to focus on the Player. Then you can use the scroll wheel to zoom out.

Remember, if you navigate around then you will be looking at your scene from a different perspective so your view won't look exactly the same as our examples.

Save your project

Step 3 Add a player character

The player in your world will be a Cat or Raccoon character.

Click on the **Models** folder in the Project window. A model describes what a 3D object looks like and can be created using 3D modelling tools such as Blender. We have included some models that you can use.

Choose either the `Cat` or `Raccoon` model and drag it from the Project window to the Scene view:

Tip: If you have accidentally added 'CatBase' or 'RaccoonBase' models, or if you want to change your character at this point, you can delete the model from the scene. Right-click on the model GameObject in the Hierarchy window and select 'Delete'.

Your character will appear in the Scene view in a T-pose.

> The **T-pose** is the default position for a game character before it has been animated.

Click on your character in the Scene view and tap the `F` key.

Tip: If you get lost in the Scene view, you can click on your character (or another GameObject) in the Hierarchy window and then click `Shift+F` to focus on your character in the Scene view.

Hmm, your character is wearing multiple accessories.

Click on your character in the Hierarchy window. This will open the settings for the GameObject in the Inspector window.

Click on the arrow next to your character in the Hierarchy window to see the 'child objects'. Click on **ConstructionGearMesh** and uncheck the box next to its name in the Inspector window. This will hide the hard hat and high-vis vest:

Hide the other accessories for your character in the same way, or just keep one active.

Tip: GameObjects that are not active appear greyed out in the Hierarchy window:

The player will see the game through the 'Main Camera', which is shown as a video camera icon in the Scene. Select the camera in the Hierarchy window to see the embedded camera view:

The Game view shows what your project will look like to a player.

Click on the Game view tab. Your character will be in whatever position you dragged it to in the Scene view (you might not be able to see it).

If you have enough room on your screen, then it's really useful to see the Scene view and the Game view at the same time.

Drag the Game view tab to the right so that it appears next to the Scene view:

Unity uses x, y, and z coordinates to position GameObjects in 3D space:

Units and 3D coordinates in Unity

Unity uses x, y, and z coordinates to position objects in 3D space.

A unit, the length of a grid square, corresponds to 1 metre in the real world.

If you add a plane and don't rotate it, then x and z give the coordinates on the plane and y gives the up and down coordinates from the plane.

Coordinates are given as a 'Vector3' or three numbers. The centre or default position of the world is (0, 0, 0).

(0, 1, 0) is a position at the centre and 1 metre up.

(1, 0, 1) is a position on the plane (y = 0) and one unit away in the x and z directions.

Select your character (in the Hierarchy window or Scene view) and then change its 'Transform' settings so the 'Position' is (0, 0, 0) — the centre of the world:

Your character will move to the centre in the Scene view and the Game view:

Rename your character to ‘Player’ in the Inspector window. This will make it easy to find if you add more GameObjects.

Save your project

Step 4 Add character movement

Your player will move with the WASD or arrow keys.

> Unity uses the **C#** (pronounced C sharp) programming language, which is used by professional software developers. C# is an object-oriented language with **classes** that define behaviour for similar objects and **methods**, which are functions that belong to a class. In Unity, a **script** defines a class with variables and methods. You can add the same script to multiple GameObjects if they need the same features.

Click on the **Player** GameObject in the Hierarchy window or Scene view so you can see its properties in the Inspector window.

Tip: Make sure you have the **Player** selected and not one of its child objects.

Click **Add Component** and start to type `character` in the Search box, then click on the **Character Controller** component when it appears:

The Character Controller component adds new features to your Player GameObject including a `SimpleMove` method and a **collider**. Colliders can be used to stop your character walking through solid objects and to detect when collisions take place.

> A **collider** is a shape that is used to detect when a GameObject collides, or intersects, with another GameObject. It's much quicker for a computer to check for collisions with a simple collider shape than the complex shape of a GameObject. A **hitbox** is a kind of collider.

The Character Controller collider has a height of `2` and a centre at `0, 0, 0`; this means it is positioned half above and half below the plane:

Your character has a height of `1`, meaning their centre on the y-axis is at `0.5`. Change the value in the Character Controller y-axis centre to `0.5` and the height to `1` to match the character:

Your character needs a script so that the player can move it around. You'll need a code editor installed on your computer to edit this script.

Install and use Visual Studio with Unity

Unity and Visual Studio

Many different code editors can be used with Unity, but Visual Studio Community Edition is probably the easiest to install and use.

In Unity Hub, select **Installs** from the menu on the left, and then click on the gear icon to the right of your Unity version and select **Add modules**.

Make sure the checkbox next to Microsoft Visual Studio Community has been checked, and then click the **Continue** button.

Read the licensing terms and then if you agree, check the box and click the **Install** button.

Once the installation of Visual Studio has completed you will need to **restart your computer**, then open your Unity project. Click on **Edit** and then choose **Preferences** from the menu.

In the menu on the left, select **External Tools** and in the drop-down menu for **External Script Editor** choose **Visual Studio 2019**.

Go to the Inspector window for the Player and click on the **Add Component** button. Type `script` and select **New Script**. Name your new script `PlayerController`, then press `Enter`.

The new script will be saved in the Assets folder:

Double-click on **PlayerController** in the script component in the Inspector window. The script will open in a separate code editor and have this code:

PlayerController.cs

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

public class PlayerController : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {

    }
}
```

Debug: Check that the name after 'class' is `PlayerController` and this matches the name of your script file: if you rename the file after creating it, then you will need to change the class name in the script.

The Start method is called once when you play your scene. Add code to print the message `Player started` when your project starts running.

Use the `Debug.Log()` method to print a message when the `Start` method is called for the Player GameObject. The message will appear in the bar at the bottom of the Unity Editor and in the Console window:

PlayerController.cs - Start()

```
// Start is called before the first frame update
void Start()
{
    Debug.Log("Player started");
}
```

Tip: The lines starting with // are comments that explain the code. You don't need to type them.

Save your PlayerController script in your code editor, using `Ctrl+S` (or `Cmd+S`), then return to the Unity Editor. The Unity Editor will load your script to get it ready to run; this may take a few seconds.

Click on the Console window tab to bring it to the front:

Test: Go to the Toolbar and click once on the **Play** button to put your scene into Play mode. This will simulate your scene as it would be viewed and interacted with by a user:

Unity takes a few seconds to start up, then you should see the `Debug.Log()` 'Player started' output in the Console.

Debug: Your scene won't play if there are errors in your code. Check the Console window for information. You may see:

- `; expected` – check for a semicolon `;` at the end of each line of code.
- `Newline in constant` – you missed a quote `"` from the end of a text string.
- `} expected` – you should have a pair of open and close curly brackets `{}` around each method and around the class. Check that your curly brackets match.
- `) expected` – make sure there's a closing `)` at the end of each method call, before the semicolon.
- `Debug` does not contain a definition for 'log' – C# is case sensitive, so it needs to be `Log` with a capital `L`.

Compare your code with the example code and make sure everything is exactly the same.

Click once on the **Play** button again to exit Play mode and the debug output will stop.

Tip: Changes made in Play mode are lost when you exit Play mode. Make sure you exit Play mode when you finish testing.

Unity creates the effect of movement by quickly drawing images to the screen. Each image is a **frame**. The `Update` method is called once every frame.

You will be able to use the WASD or arrow keys (players on a mobile or console can use different inputs without you changing your code.)

`Input.GetAxis("Vertical")` takes input from the `W` and `S` keys or the up and down arrow keys, and returns a number between 1 and -1, which it uses for forwards and backwards movement.

PlayerController.cs - Update()

```
14  void Update()
15  {
16      float speed = Input.GetAxis("Vertical");
17
18      if (speed != 0) // Player moving
19      {
20          Debug.Log(speed);
21      }
22  }
```

A `float` is a decimal number.

Save your PlayerController script in your code editor, using `Ctrl+S` (or `Cmd+S`), then return to the Unity Editor.

Tip: You might finding it quicker to use `Alt+Tab` (or `Cmd+Tab`) to switch between your web browser with the project instructions, the Unity Editor, and your code editor.

Test: Go to the Toolbar and click on the **Play** button to put your scene into Play mode.

Place your **mouse pointer in the Game view** and press keys `W` and `S`. Look at the values logged in the Console window as you press the keys. Each time you press `W` a positive number is logged, when you press `S` a negative number is logged.

The numbers range between -1.0 and 1.0 and correspond to movement from the vertical controls on the keyboard (or a game controller). You can also use the up and down arrow keys.

Tip: The output also appears in the bar at the bottom of the Unity Editor.

Click the **Play** button again to exit Play mode and the debug output will stop.

It's easy to forget whether your game is playing or not. A Play mode colour tint makes it easier to tell when your scene is playing:

To set a tint, go to the **Edit Menu** (or Unity Menu) and select **Preferences**. Choose the **Colours** menu and find the property called **Playmode tint**.

Click on the existing colour to see a colour wheel where you can choose a colour and opacity level:

Tip: Try a light colour so that you can still clearly see the text in the editor when the scene is running.

Return to the Unity Editor and press the **Play** button to see your new tint in action. When you are happy with the tint you have chosen, press the **Play** button again to exit Play mode.

The Character Controller component provides a `SimpleMove` method.

Add code to use the vertical input value to move the Player each frame.

You can **remove** the Debug code.

A Unity `Vector3` is used to store 3D points or directions. The `forward` variable stores the direction that the Player is facing:

PlayerController.cs - Update()

```
14  void Update()
15  {
16      float speed = Input.GetAxis("Vertical");
17
18      // Forward is the forward direction for this character
19      Vector3 forward = transform.TransformDirection(Vector3.forward);
20
21      // You need the Character Controller so you can use SimpleMove
22      CharacterController controller = GetComponent<CharacterController>();
23      controller.SimpleMove(forward * speed);
24  }
```

Test: Click **Play** to enter Play mode and try out your code. Use the `W` and `S` keys or the up and down arrow keys to glide forwards and backwards.

Debug: Remember to check the Console window for helpful messages. Check brackets, semicolons, and capital letters in your code carefully.

Tip: Make sure your mouse pointer is in the **Game view**.

Try and walk through the wall. The `SimpleMove` method from the Character Controller component stops you from being able to walk through GameObjects that have a collider. A collider is automatically added when you create a 3D shape as you did for the wall.

You can pan around in the Scene view by holding your right mouse button and dragging. Pan to get a better view of the wall as your character walks into it:

To move your Player, move the mouse pointer back to the **Game view**.

Click the **Play** button again to exit Play mode.

Add another line so your character can `Rotate` when the player presses the `A` and `D` keys or the left and right arrow keys:

PlayerController.cs - Update()

```
14  void Update()
15  {
16      float speed = Input.GetAxis("Vertical");
17
18      // Rotate around y-axis
19      transform.Rotate(0, Input.GetAxis("Horizontal"), 0);
20
21      // Forward is the forward direction for this character
22      Vector3 forward = transform.TransformDirection(Vector3.forward);
23
24      // You need the Character Controller so you can use SimpleMove
25      CharacterController controller = GetComponent<CharacterController>();
26      controller.SimpleMove(forward * speed);
27  }
```

Save your code and switch back to the Unity Editor. Unity will load your updated script.

Test: Click **Play** to enter Play mode and try out your code. Use the `A` and `D` keys or the left and right arrow keys to rotate.

Debug: If you are still seeing output to the Console and movement isn't working, then make sure you have saved your script in the code editor.

Click the **Play** button again to exit Play mode.

You can also control the speed of movement and rotation.

Open your PlayerController script and add variables for the `moveSpeed` and `rotateSpeed`.

PlayerController.cs

```
5  public class PlayerController : MonoBehaviour
6  {
7      public float moveSpeed = 4.0f; //The f at the end of the number says it is a floating-point number
8      public float rotateSpeed = 1.5f;
9
10     // Start is called before the first frame update
11     void Start()
12     {
```

Update the code to `Rotate` and `SimpleMove` your character to multiply them by the new variables:

PlayerController.cs - Update()

```
21 // Rotate around y-axis
22 transform.Rotate(0, Input.GetAxis("Horizontal") * rotateSpeed, 0);
```

and

PlayerController.cs - Update()

```
27 // You need the Character Controller so you can use SimpleMove
28 CharacterController controller = GetComponent<CharacterController>();
29 controller.SimpleMove(forward * speed * moveSpeed);
```

Test: Play your scene and check if you are happy with the speed settings.

Make changes to `moveSpeed` and `rotateSpeed` in your script until you are happy.

Tip: You can mask the `Debug.Log()` lines by putting `//` at the beginning of the line.
You can also mask multiple lines using `/*` and `*/`:

```
/*if (speed != 0) // Player moving
{
    Debug.Log(speed);
}*/
```

Click the **Play** button again to exit Play mode.

Save your project

Step 5 Animation and camera position

Your Player is moving around, but at the moment, it's stuck in a T-pose position. You can improve this by using animations.

Drag the **IdleWalk** animator from the **Animation** > **Animators** folder in the Project window to the Controller property of the Animator component of your character:

This will add Idle and Walk animations to your character with a `forward` Boolean parameter that you can use to control which animation plays.

Test: Play your project and make sure you can see the Idle animation:

Add code to the `Update` method of your script so that when the character is moving forward it uses a walking animation, otherwise it uses an idle animation:

PlayerController.cs - Update()

```
void Update()
{
    float speed = Input.GetAxis("Vertical");

    //Set animations
    Animator anim = gameObject.GetComponent<Animator>();

    if (Input.GetAxis("Vertical") > 0) // Forwards
    {
        anim.SetBool("forward", true);
    }
    else // Idle
    {
        anim.SetBool("forward", false);
    }

    // Rotate around y-axis
    transform.Rotate(0, Input.GetAxis("Horizontal"), 0);
```

Test: Play your project and make sure you can see the animation change to walk when you move forward and switch to idle when you are not moving forward:

In games, the camera often follows the Player.

The placement of a **virtual camera** in a 3D environment is key for creating the right perspective for users. Visibility levels from the camera lens affect the difficulty level and influence the atmosphere of a game.

In the Hierarchy window, drag the **Main Camera** to the Player GameObject; it will become a 'child' of the Player and will follow the Player around.

Test: Play your project. The camera will now follow your character, but it's a bit far away and walls often come between the Player and the camera.

You can adjust the position and rotation of the camera in the Scene view or the Inspector window.

Exit Play mode and select the **Main Camera** in the Hierarchy window. Adjust its Transform settings to get a third-person view of your Player, looking down from behind and above your Player:

You can position the camera in the Scene view using the Transform and Rotate tools if you prefer:

Test: Play your project. The camera will now follow your character with the camera just behind and above your character and looking down at an angle.

Adjust the camera settings until you are happy with them.

Tip: You can try settings out in Play mode but you need to exit Play mode and update the settings to keep them.

What happens if you go off the edge of the plane? Don't worry your character will go back to the centre next time you enter Play mode:

Save your project

Upgrade your project

Now you can make your 3D world just the way you want it.

You can:

- Add more 3D objects to your world, try Spheres and Capsules
- Try different colours and materials
- Change the Scale settings or the Transform values for your character to make it bigger or smaller — you'll need to change all three values to keep it in proportion
- Enable and disable different accessories in the Inspector window to get the look you want
- Adjust the Player movement and rotation speed
- Adjust the position of the camera

You can download the basic and extended projects as packages along with the WebGL projects from the zipped solutions directory at **https://rpf.io/p/en/explore-a-3d-world-get** (https://rpf.io/p/en/explore-a-3d-world-get)

Completed project

You can download the **completed project here** (https://rpf.io/p/en/explore-a-3d-world-get).

You can view the completed project below.

Save your project

What next?

If you are following the **Introduction to Unity** (https://projects.raspberrypi.org/en/raspberrypi/unity-intro) pathway, you can move on to the **Star collector** (https://projects.raspberrypi.org/en/projects/star-collector) project. In this project, you will make a minigame where you collect stars that are hidden around a 3D map.
