Lesson 4: Introduction-to-unity-and-creating-your-first-scene

- Unity is a cross-platform game engine developed by Unity Technologies, first announced and released in June 2005 at Apple Worldwide Developers Conference as a Mac OS X game engine.
- The engine has since been gradually extended to support a variety **of desktop, mobile, console, augmented reality, and virtual reality platforms.** It is particularly popular for iOS and Android mobile game development, is considered easy to use for beginner developers, and is popular for indie game development.
- Unity is the market leader when it comes to enthusiasts wanting to create their own games, whether it's **2D and 3D**. It's a great place to start for beginners. And it's free!
- Before downloading Unity, though, it's a good idea to get a coding editor installed, as Unity doesn't come with one built in. Our lesson use Visual Studio Community edition, which is by Microsoft. **If you've been doing C# or VB courses, then you already have VS installed.** In which case, you can go straight to the Unity download. If you haven't, then you can download VS here:

Get Visual Studio Community

- It's a hefty download, however. A great alternative is Visual Studio Code, also by Microsoft, and available for Windows, Linux and Mac. You can get it here:

Get Visual Studio Code

- Once you download your coding editor, you need Unity itself. Go to this page and grab yourself the software.

Get Unity

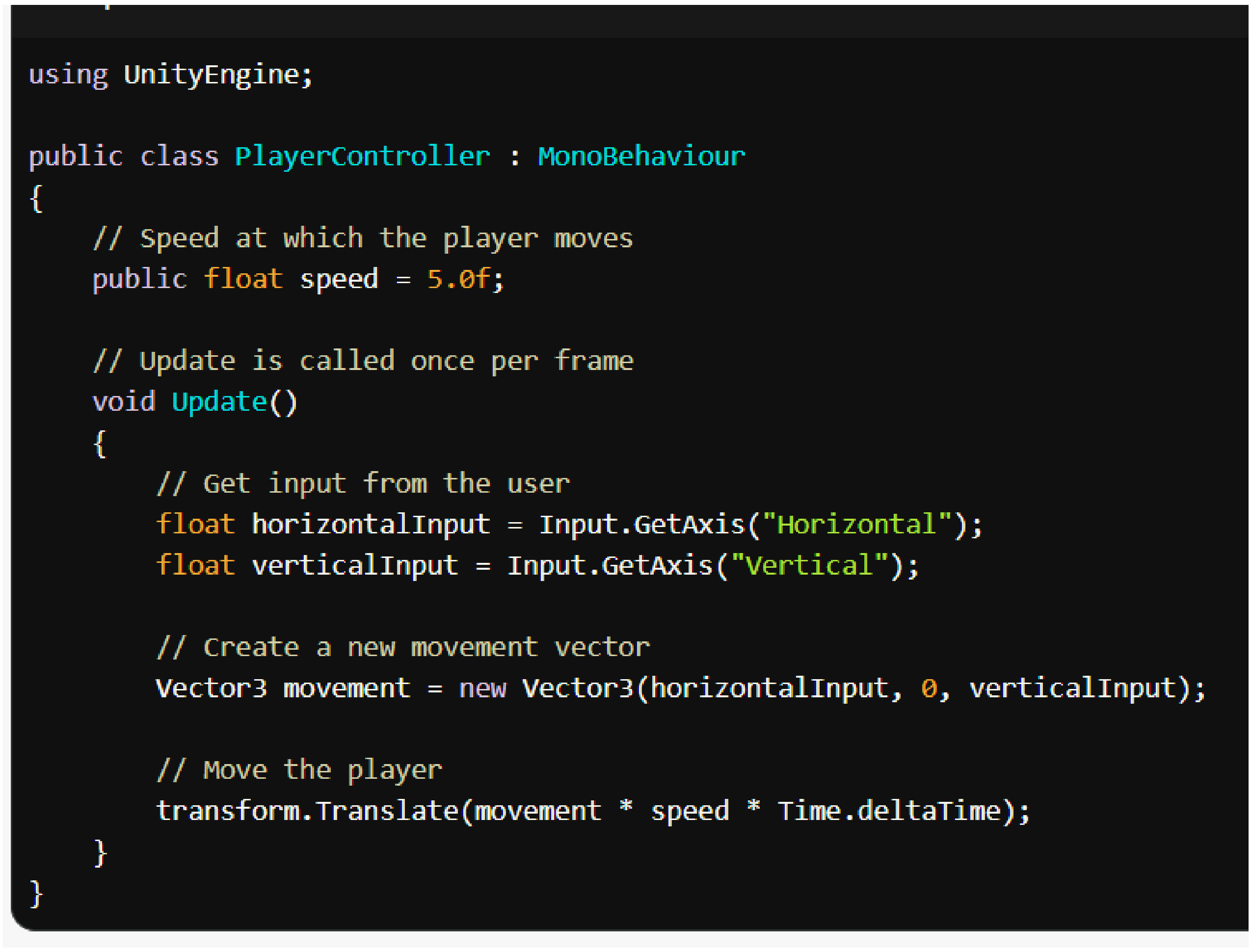
Unity Engine Fundamentals: A Beginner's Guide

Creating and Manipulating Game Objects

Game objects are the fundamental elements of any Unity game. **These objects can be anything from players, enemies, to environmental elements.** Learn how to create game objects by right-clicking in the Hierarchy Panel or by going to the Game Object menu. Once created, you can **manipulate these objects by moving, rotating, and scaling, utilizing the tools located in the toolbar**. Familiarizing yourself with these basic operations will allow you to start building your game's environment and characters.

Scripting in Unity

Scripting is where much of your game's magic happens. Unity uses the C# programming language, which offers a comprehensive environment to code game behavior, interactions, and mechanics. Even if you're new to coding, Unity's community and wealth of online tutorials can get you started. **Learn the basics of variables, methods, and classes to script simple actions, like making a character jump or respond to input**. Unity's scripting API provides a wealth of pre-built functions that perform various tasks, making the development process more intuitive.



```
using UnityEngine;

public class PlayerController : MonoBehaviour
{
    // Speed at which the player moves
    public float speed = 5.0f;

    // Update is called once per frame
    void Update()
    {
        // Get input from the user
        float horizontalInput = Input.GetAxis("Horizontal");
        float verticalInput = Input.GetAxis("Vertical");

        // Create a new movement vector
        Vector3 movement = new Vector3(horizontalInput, 0, verticalInput);

        // Move the player
        transform.Translate(movement * speed * Time.deltaTime);
    }
}
```

Explanation of the Code

1. **Namespaces**:

using UnityEngine;

- This line imports the UnityEngine namespace, which contains classes and functions necessary for Unity scripting, including MonoBehaviour, Input, and Vector3.

2. **Class Definition**:

public class PlayerController : MonoBehaviour

- MonoBehaviour is a base class in Unity that provides core functionality for scripts attached to GameObjects. It is essential for creating interactive and dynamic behavior in Unity applications. Here's a detailed overview of what MonoBehaviour means and its key features:
    1. Inheritance
- Base Class: All scripts that you want to run in Unity must inherit from the MonoBehaviour class. This allows your scripts to leverage Unity's built-in functionalities.
- This defines a new class called PlayerController that inherits from MonoBehaviour. In Unity, all scripts that need to be attached to GameObjects must inherit from MonoBehaviour.

3. **Public Variable**:

```
public float speed = 5.0f;
```

- This line declares a public variable speed that can be adjusted in the Unity Inspector. It controls how fast the player moves. The default value is set to 5.0f.

4. **Update Method**:

```
void Update()
```

- The Update method is called once per frame. This is where you typically handle input and update the GameObject's state.

5. **Input Handling**:

```
float horizontalInput = Input.GetAxis("Horizontal");

float verticalInput = Input.GetAxis("Vertical");
```

- Input.GetAxis("Horizontal") and Input.GetAxis("Vertical") retrieve player input from the keyboard (e.g., arrow keys or WASD). The values range from -1 to 1, where 0 means no input, -1 indicates left/down, and 1 indicates right/up.

6. **Movement Vector**:

```
Vector3 movement = new Vector3(horizontalInput, 0, verticalInput);
```

- A Vector3 object named movement is created, which represents the direction and magnitude of movement. The y component is set to 0 since we're only moving in the horizontal plane.

7. **Moving the Player**:

```
transform.Translate(movement * speed * Time.deltaTime);
```

- transform.Translate moves the GameObject this script is attached to. The movement vector is multiplied by speed and Time.deltaTime to ensure smooth and frame rate-independent movement. Time.deltaTime is the time taken to complete the last frame, which leads to consistent movement speed regardless of the frame rate.

Using the Script in Unity

1. **Create a New GameObject**:
   - In the Unity Editor, create a new GameObject (e.g., a 3D Cube) by right-clicking in the Hierarchy and selecting 3D Object > Cube.
2. **Attach the Script**:
   - Create a new C# script named PlayerController, copy the provided code into it, and save it. Drag the script onto the Cube GameObject in the Hierarchy.
3. **Adjust Speed**:
   - With the Cube selected, you can see the PlayerController component in the Inspector. Adjust the speed variable as desired.
4. **Play the Game**:
   - Press the Play button in the Unity Editor. Use the arrow keys or WASD to move the Cube around the scene.

Conclusion

- This simple PlayerController script demonstrates the basics of Unity scripting, including input handling, movement, and the use of MonoBehaviour**. As you learn more, you can expand on this script to include features like jumping, animations, or interactions with other GameObjects.**

Importing Assets and Using Asset Store

Assets are the building blocks of your Unity game. These include the likes of **textures, models, animations, audio files, and prefabs.** You can import assets into your project by simply dragging and dropping them into the Unity Editor or by using the Import New Asset option under the Assets menu. The Unity Asset Store is an invaluable resource where you can buy or download free assets created by Unity developers around the world. From entire environment packs to pre-animated characters, it's a great way to enhance your project without starting from scratch.

- **Prefabs and Assets in Unity: Importance of Learning**

In Unity, **prefabs** and **assets** are fundamental concepts that significantly enhance the efficiency and flexibility of game development. Understanding these elements is crucial for creating effective and maintainable projects. Here's why they are important:

1. Prefabs

Definition: A prefab is a reusable GameObject stored in the project folder. It can contain various components, including scripts, textures, and other GameObjects.

Importance:

- **Reusability**:
  - Prefabs allow developers to create a GameObject once and reuse it throughout the project. This is especially useful for repeated elements like enemies, items, and environment objects.
- **Consistency**:
  - By using prefabs, you ensure that all instances of a GameObject have the same properties and behaviors. This consistency helps maintain the quality of the game.
- **Efficient Workflow**:
  - Changes made to a prefab automatically propagate to all instances in the scene. This reduces the time spent on adjustments and ensures that updates are uniformly applied.
- **Dynamic Instantiation**:
  - Prefabs can be instantiated at runtime using scripts, allowing for dynamic creation of GameObjects based on game events (e.g., spawning enemies).

Example: If you create a "Health Pack" prefab, you can place it in multiple locations in your game. If you need to adjust its properties (like healing amount), you can do so once in the prefab, and all instances will update automatically.

2. Assets

Definition: Assets are files used in your project, including textures, models, audio clips, animations, scripts, and more.

Importance:

- **Diverse Resources**:
  - Assets provide the building blocks for your game. Understanding how to manage and utilize them effectively enhances the quality and richness of your game environments.
- **Organization**:
  - Learning how to organize assets in the Unity project structure helps maintain a clean and manageable project. This is crucial as projects grow in size and complexity.
- **Importing and Exporting**:
  - Familiarity with asset formats and how to import/export them allows you to integrate third-party tools and resources, enhancing your game's functionality and aesthetics.

- **Optimization**:
    - Knowing how to use assets efficiently can lead to better performance. For example, using sprite atlases for 2D games can reduce draw calls and improve rendering performance.

Example: If you have a collection of sound effects, understanding how to import and manage these audio assets will allow you to create a more immersive experience by adding appropriate sounds to game events.

Conclusion

Learning about prefabs and assets in Unity is crucial for efficient game development. They enable developers to create reusable components and manage resources effectively, leading to more organized, consistent, and high-quality games. Understanding these concepts not only improves your development workflow but also enhances the overall player experience.

Game Physics and Colliders

Understanding physics is necessary to create a more immersive game**. Unity provides a built-in physics engine that simulates real-world physics to make your game realistic. Learn about Rigid body components, which can be added to game objects to give them physical properties like** mass, drag, and gravity**.** Use colliders to define the shape of an object's physical boundaries for collision detection, which is crucial for creating interactions between the player and the environment. **Experimenting with these components will help you manage complex simulations like vehicle dynamics, character movements, or any event driven by physical interactions.**

Cameras and Lighting

Cameras in Unity determine what the player sees. By understanding camera positioning and effects, you can create compelling scenes and guide the player's focus. Explore various **camera controls and components, such as the Camera Follow script,** to make your camera track the player or any other object smoothly. Lighting also plays a crucial role in visual aesthetics and mood-setting. Master the different types of lights available in Unity, such as directional, point, and spotlights, and familiarize yourself with lightmapping techniques to enhance your game's look and performance.

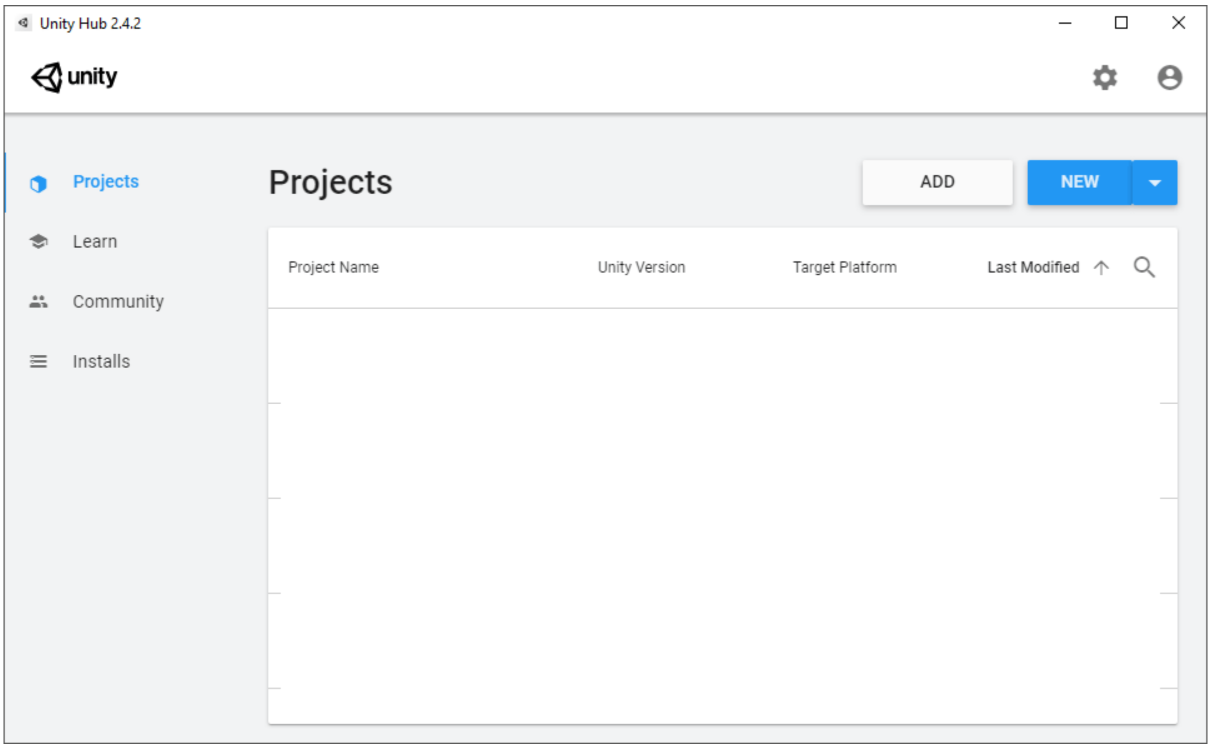
Building and Deploying a Game

Once your game is ready, it's time to share it with the world. Under the File menu, select Build Settings to open the build options. Unity allows you to build your game for various platforms, including Windows, macOS, iOS, and Android. Choose your target platform, configure the player settings, and click Build to compile your game into an executable package. With Unity's cross-platform capabilities, deploying your game on multiple platforms is relatively straightforward.

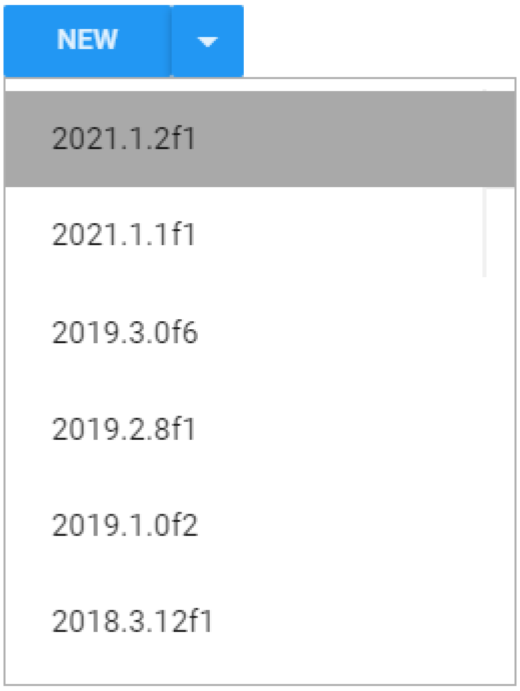
Unity is a comprehensive tool that can seem daunting at first, but with patient learning and consistent practice, you'll find it an incredibly rewarding experience. Keep exploring the various functions, engage with the community, and remember that every expert was once a beginner. Have fun on your journey to becoming a Unity developer!

Getting Started with Unity 3D

Now that you've downloaded the software, let's make a start. In this section, you'll find your way around Unity and get used to the software as we create a very basic driving game. So, from your computer's Start Menu, launch Unity Hub. You'll see this screen:

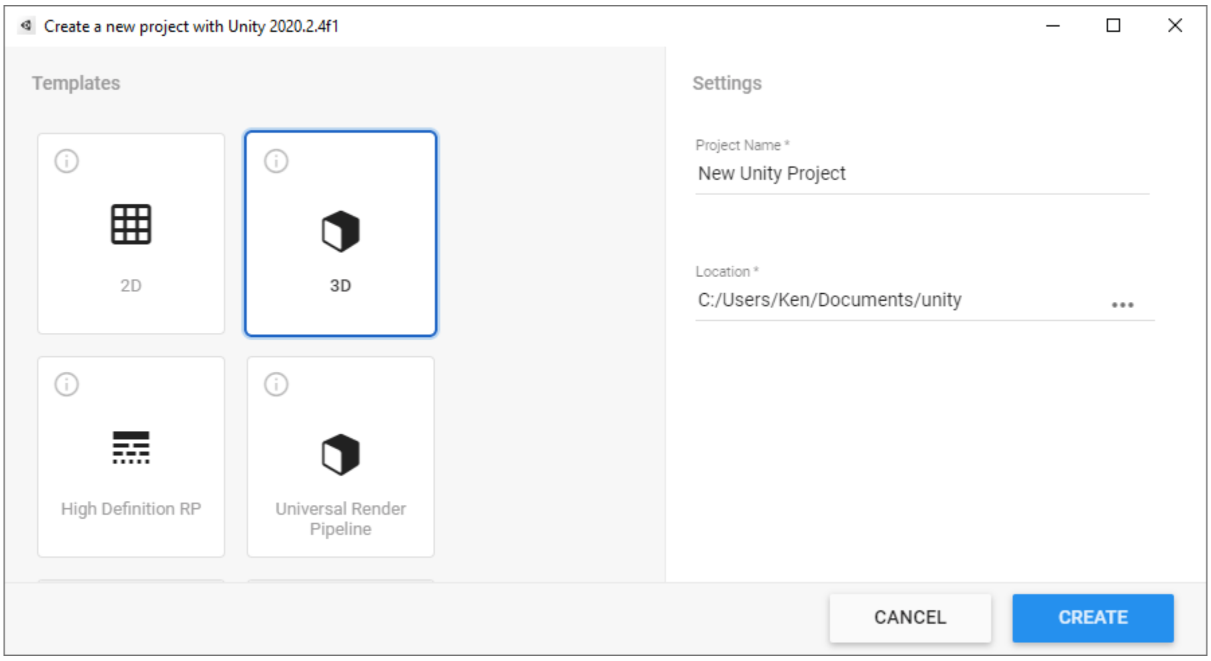


Make sure the Projects tab is selected on the left. In the top right, there's a New button with a white arrow beside it. You can create Unity projects using different versions. Click the white arrow to show which software versions you have:



Unity nags you a lot about new versions. However, it doesn't update the version you have. It installs a new version and displays which version of the software you have on this list.

But click the New button or a version on the list to see this screen:

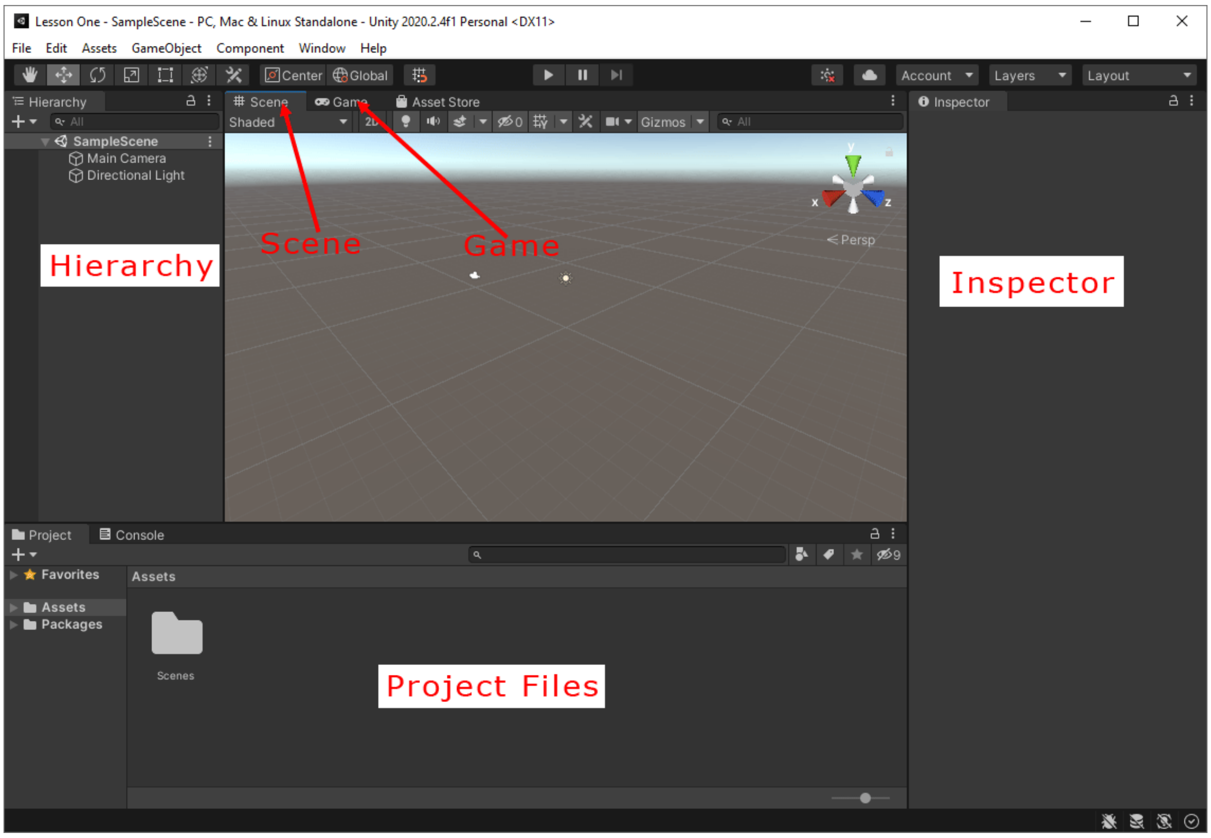

Create a new project with Unity 2020.2.4f1
Templates
2D
3D
High Definition RP
Universal Render Pipeline
Settings
CANCEL
CREATE

Project Name *
New Unity Project

Location *
C:/Users/Ken/Documents/unity

Make sure the 3D item is selected on the left. On the right, under Settings, type a name for your project. Call it anything you like. Then click the Create button at the bottom.

The software itslef can take a while to loa. But when it does, you'll see this screen:



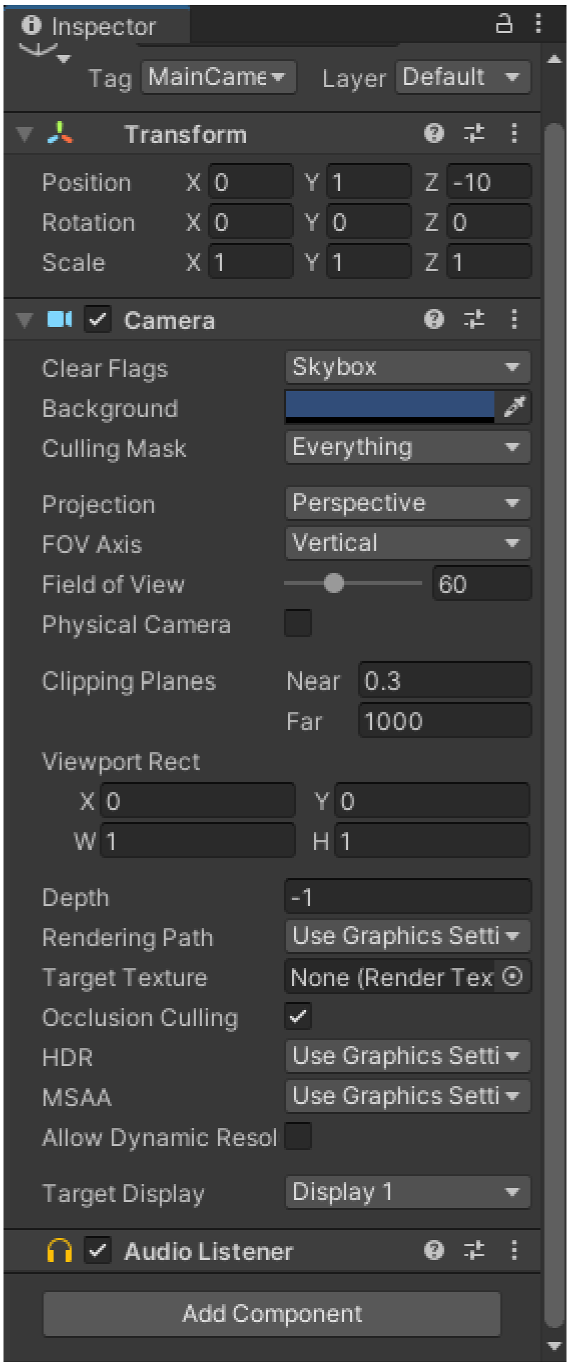
There are five important areas highlighted in the image above:

- Hierarchy
- Inspector
- ProjectFiles
- Scene
- Game

Let's go through them.

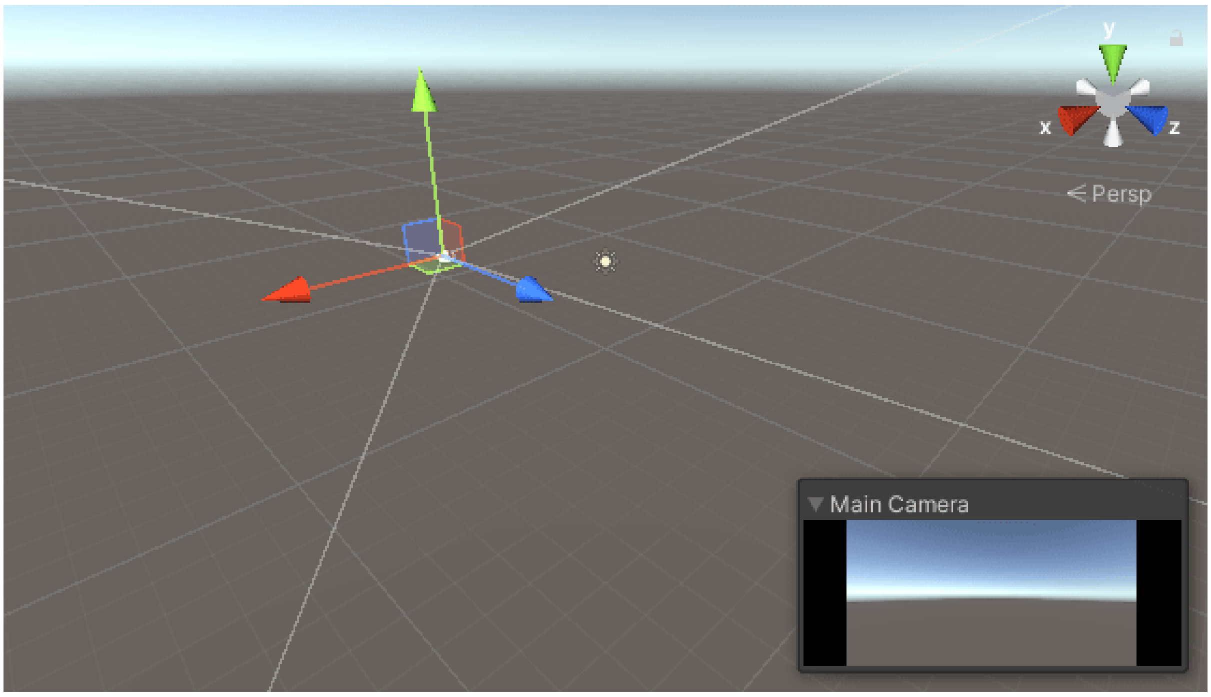
1. **Hierarchy**
   This is a list of all the game objects that are in your scene. When you first create a project, a camera and a light will be added by default.
2. **Inspector**
   When you select a game object, you'll see a list of components that are attached to that game object. Each component will have lots of settings you can change.
3. **ProjectFiles**
   All the files in your project.
4. **Scene**
   This is where you create your game.
5. **Game**
   This tab lets you see what your game looks like when played.

In the hierarchy on the left, click the **Main Camera** item to select it. In the Inspector ont he right, you should see a list of all the components attached to the Main Camera:


```
Inspector
Tag MainCame        Layer Default

Transform
Position     X 0    Y 1    Z -10
Rotation     X 0    Y 0    Z 0
Scale        X 1    Y 1    Z 1

Camera ✓
Clear Flags          Skybox
Background
Culling Mask         Everything

Projection           Perspective
FOV Axis             Vertical
Field of View        60
Physical Camera

Clipping Planes      Near 0.3
                     Far  1000

Viewport Rect
    X 0      Y 0
    W 1      H 1

Depth                -1
Rendering Path       Use Graphics Setti
Target Texture       None (Render Tex
Occlusion Culling    ✓
HDR                  Use Graphics Setti
MSAA                 Use Graphics Setti
Allow Dynamic Resol

Target Display       Display 1

Audio Listener ✓

Add Component
```

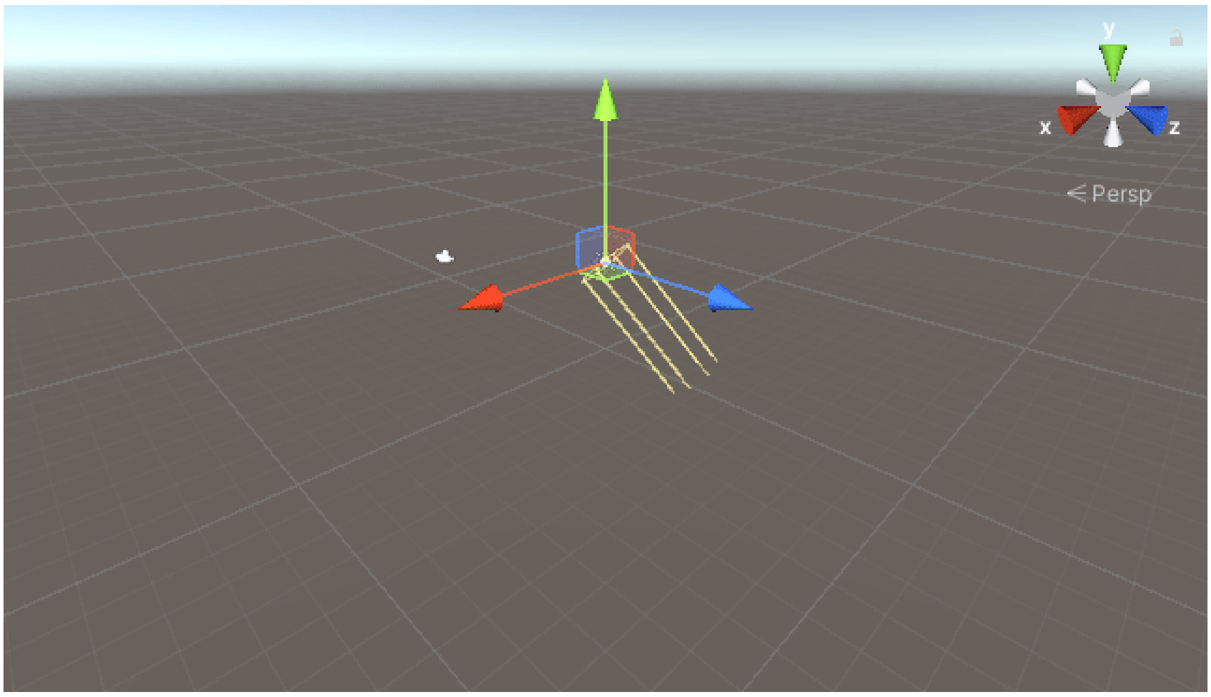
Despite how big the list is, there are only three components attached to the Main Camera: a Transform, a Camera, and an Audio Listener. Notice how many settings you can change.

When you select the camera in the hierarchy, you'll also see the camera component selected in the scene. You'll also see what the camera is looking at. This is shown in a box in the bottom right:



Notice the three coloured arrows when you select the camera. These are directional arrows and appear whenever a game object is selected in your scene. The green arrow is for the Y (up and down direction), the red arrow is for the X direction (left and right), and the blue arrow is for the Z direction (near and far away).

In the hierarchy, click the Directional Light object to select it. The camera view in the bottom right will disappear. But notice we still have the coloured arrows:

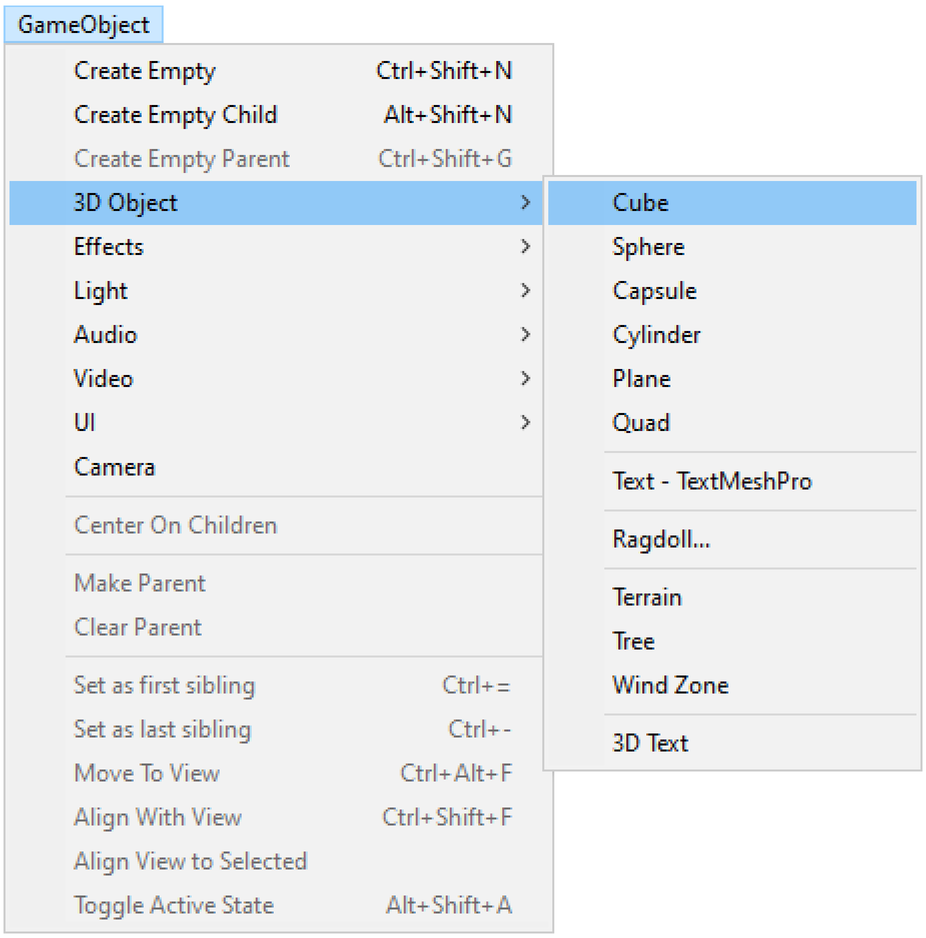


The four lines are showing you in which direction the light is shining. If you look in the Inspector window on the right, you'll see that there are two components attached to the light: a Transform and a Light. (All game objects have a Transform component.)

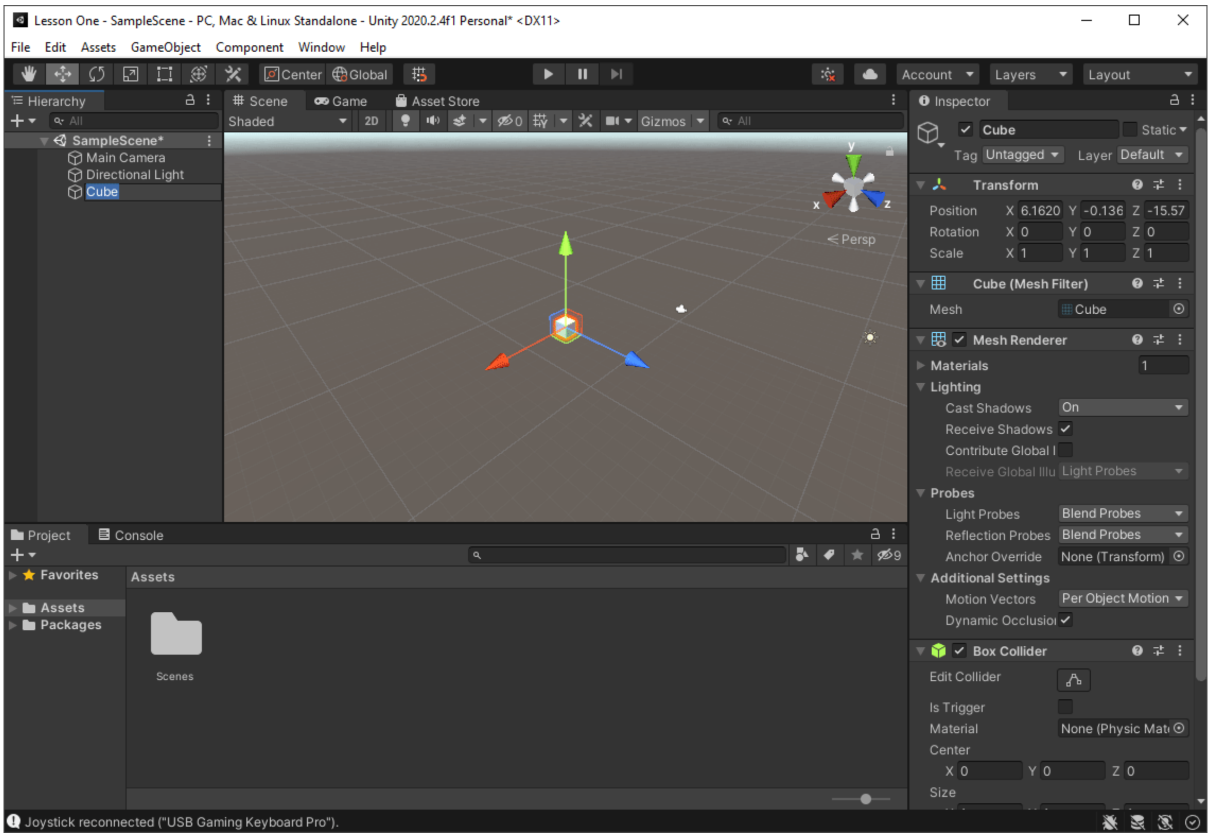
Now let's see how to move around in a Scene.

First, let's add a game object. We'll add a cube to the scene.

At the top of Unity, you'll see a menu bar with items for File, Edit, Assets, Game Objects, Component, Window, and Help. Click the **Game Object** item. From the menu that appears, select **3D Object > Cube**:



When you select the Cube item, two things happen: one, a cube appears in the Scene window and, two, a new item with the default name of Cube appears in the Hierarchy window:



Cube — Static
Tag: Untagged — Layer: Default

Transform
Position X 6.1620 Y -0.136 Z -15.57
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1

Cube (Mesh Filter)
Mesh: Cube

Mesh Renderer
Materials: 1
Lighting
Cast Shadows: On
Receive Shadows: ✓
Contribute Global I
Receive Global Illu: Light Probes
Probes
Light Probes: Blend Probes
Reflection Probes: Blend Probes
Anchor Override: None (Transform)
Additional Settings
Motion Vectors: Per Object Motion
Dynamic Occlusion: ✓

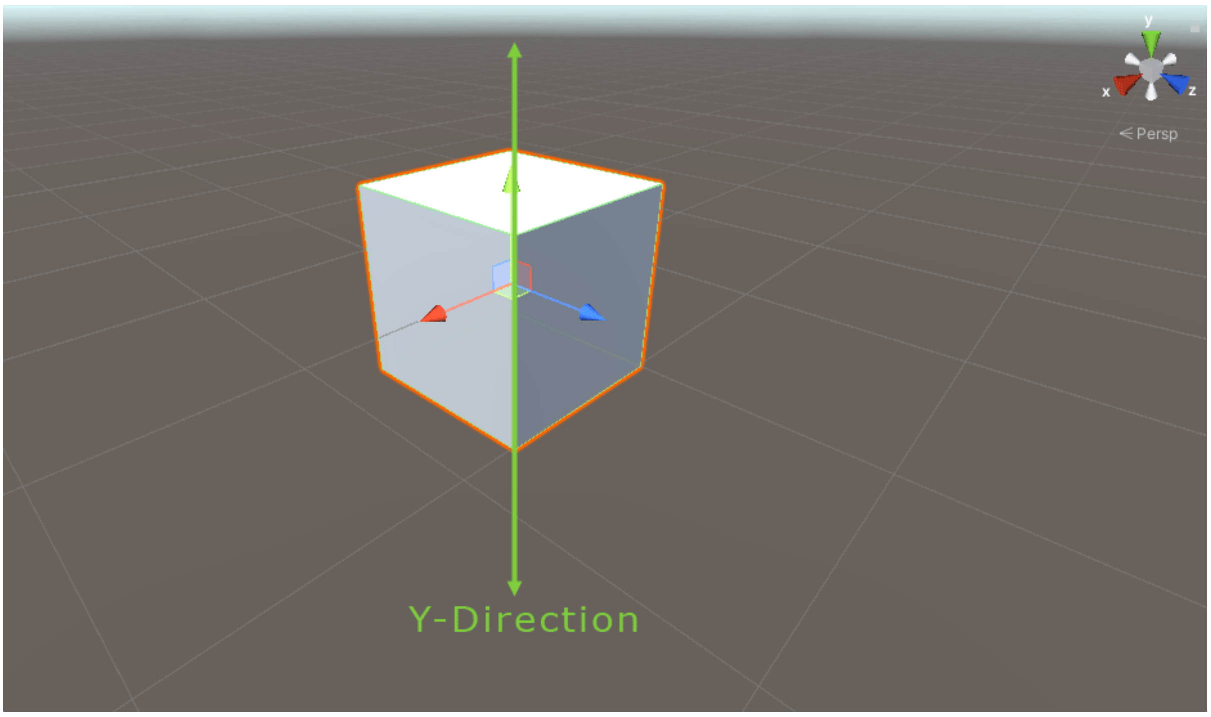
Box Collider
Edit Collider
Is Trigger
Material: None (Physic Mat
Center X 0 Y 0 Z 0
Size

Joystick reconnected ("USB Gaming Keyboard Pro").

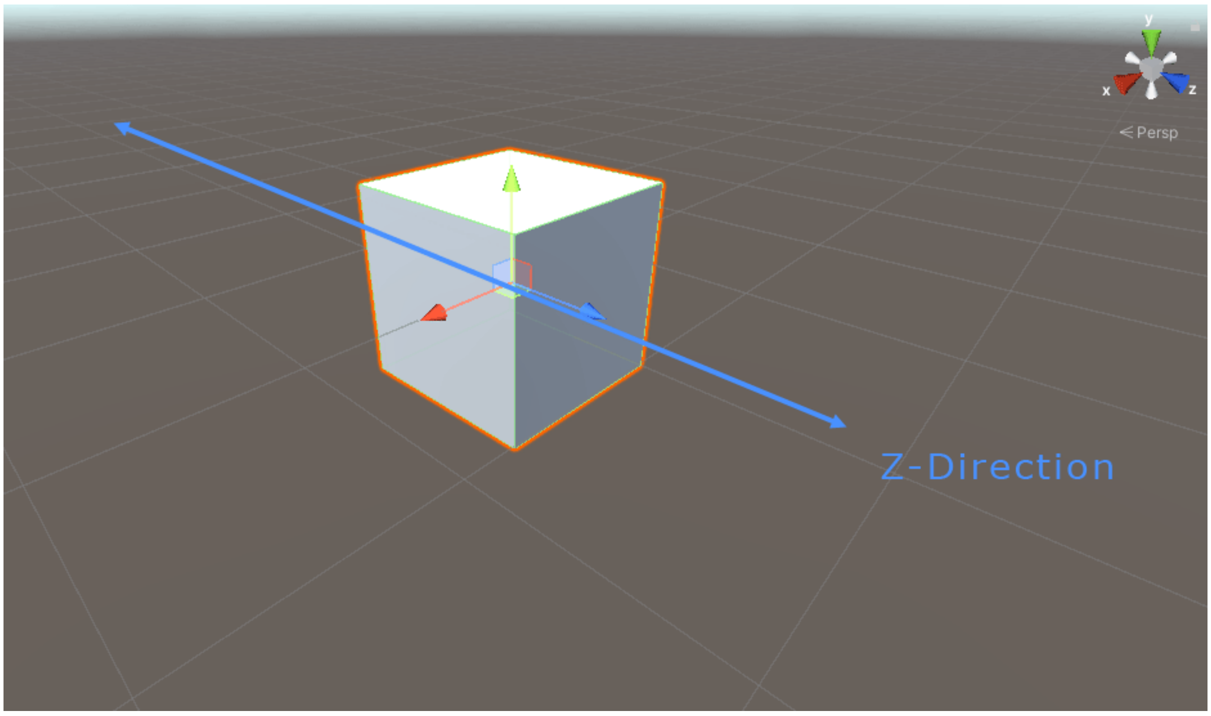
With the cube selected, take a look at the Inspector window on the right. The cube has four components attached: a Transform, a Cube (Mesh Filter), A Mesh Renderer, and a Box Collider. We'll concentrate on the Transform component. Notice it has three items: a Position, a Rotation, and a Scale. Let's play around with these, see what they do.

In your Hierarchy window, double click your Cube item. This will zoom you in on the cube. In the Inspector window, notice that the position in the image above is 6.1620 for X, -0.136 for Y, and -15.57 for Z. Now hold your mouse over the cube's red arrow in the Scene view. Hold down your left mouse button on the red arrow. Keep it held down and drag. Watch the X value change in the Inspector window as you drag. The numbers will change. Now do the same for the green and blue arrows. Watch what happens in the Scene view to your cube and its Y or Z values in the Inspector.

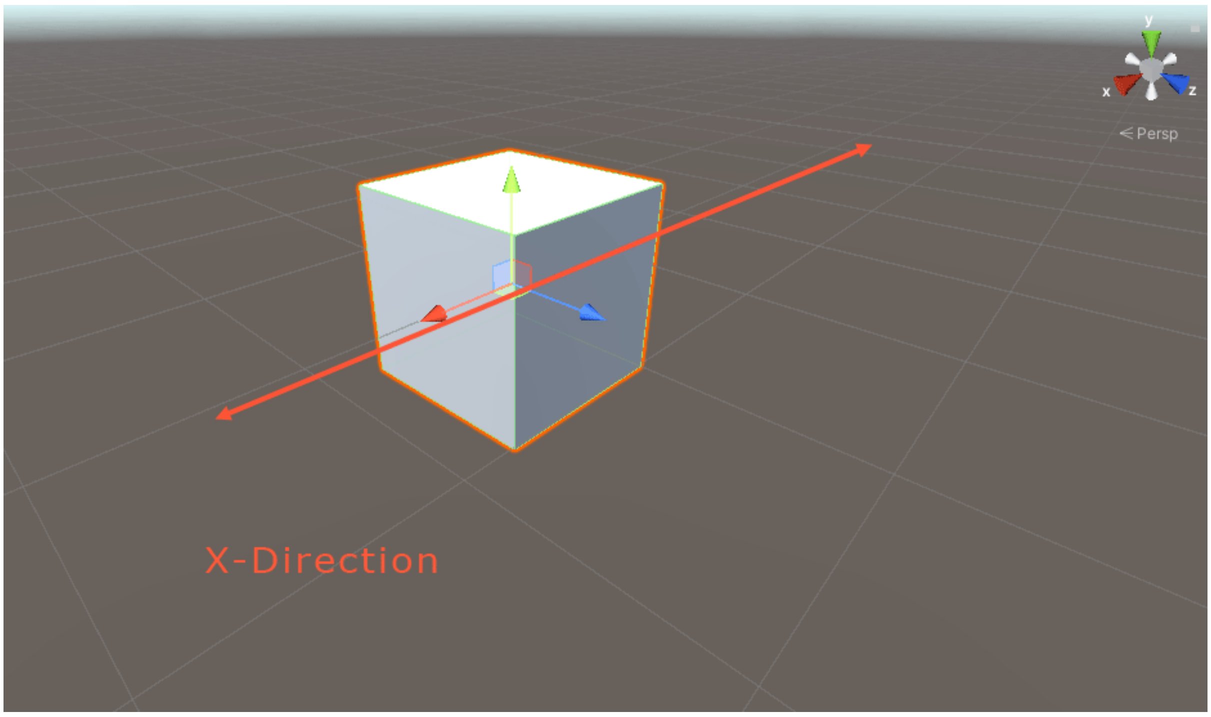
You should notice that the cube moves up and down when you drag the blue arrow:



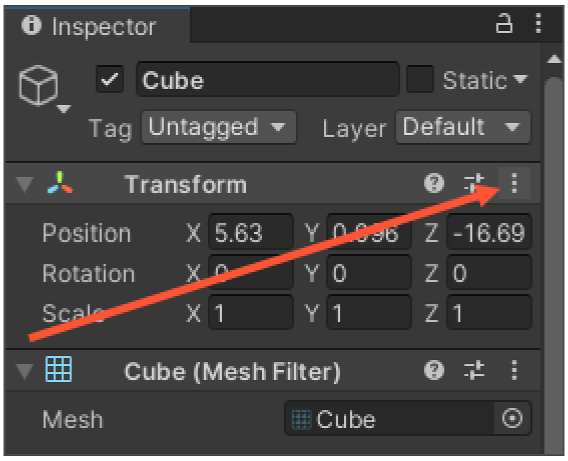
It moves near and farther away when you drag the blue arrow:



And the cube moves from left to right when you drag the red arrow:



If you want to move your cube to the centre of the scene, you can reset its position quite easily. Click the three dots to the right of the Transform heading in the Inspector:

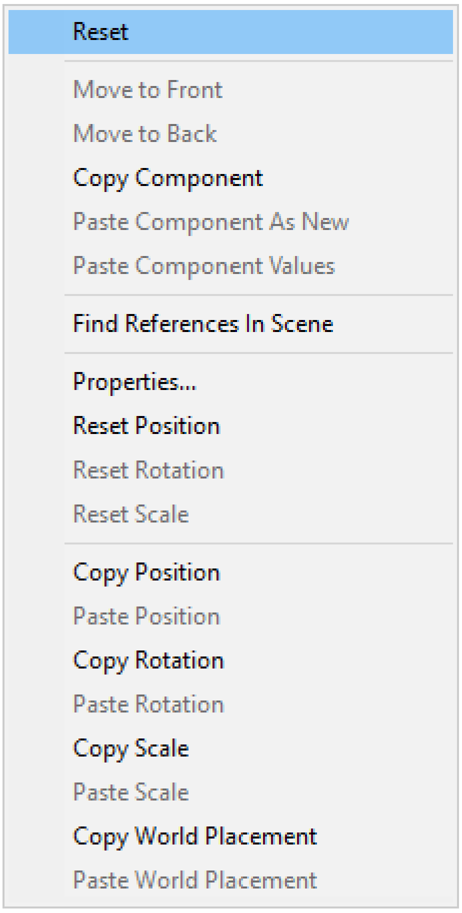


Cube
Tag Untagged Layer Default

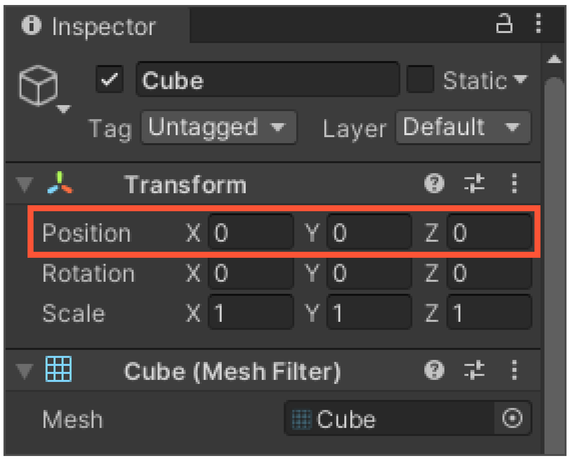
Position	X 5.63	Y 0.096	Z -16.69
Rotation	X 0	Y 0	Z 0
Scale	X 1	Y 1	Z 1

Mesh Cube

A menu will appear. Select **Reset**:



The X, Y and Z values in the Inspector will now all be 0:



Cube

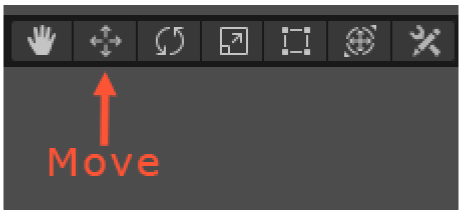
Position X 0 Y 0 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1

Mesh Cube

(If you suddenly can't see your cube in the Scene window, simply double-click it again in the Hierarchy.)

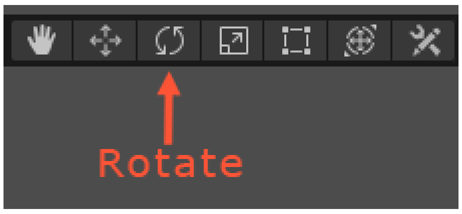
Note that you can also click inside of the text boxes where the numbers are. Delete the old value and type a new one. (Another way to change the values is to hold your left mouse button over the X, the Y or the Z. Keep it held down and drag left or right.)

You can also use the toolbar at the top of Unity to get the position arrows. Just click the Move icon to see the coloured arrows:

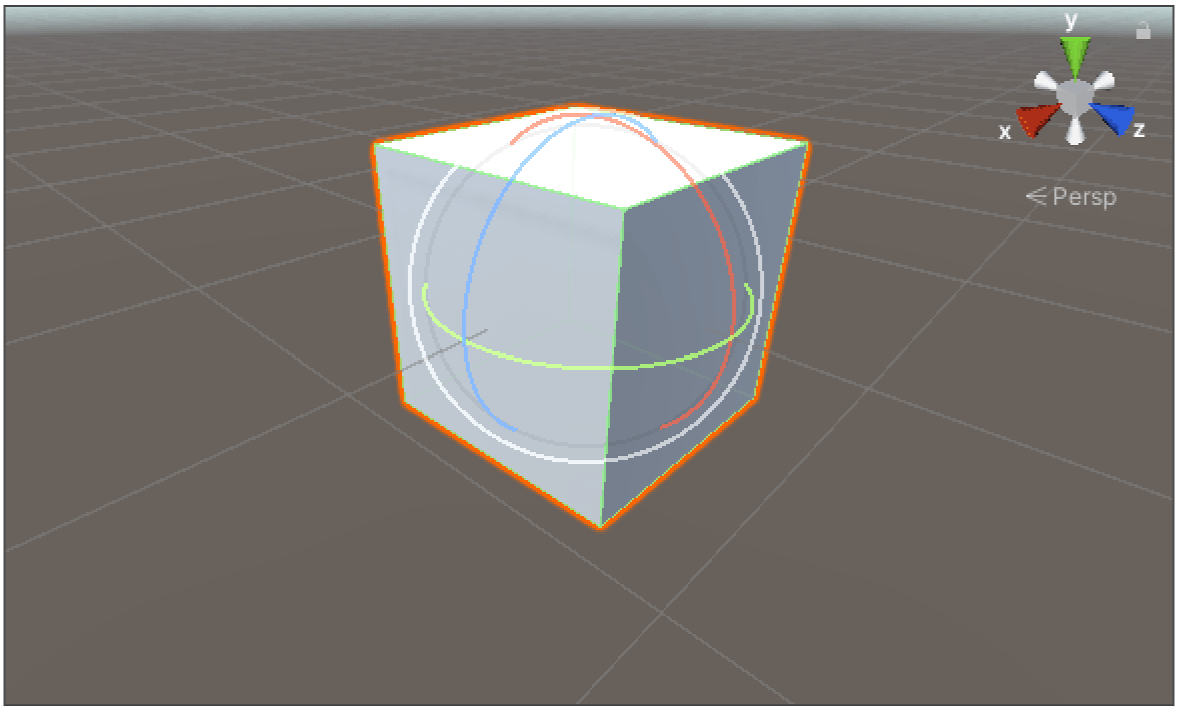


Rotate

Just under the Position transform in the Inspector is the Rotation transform. To see how that works, and with your cube selected, click the rotation icon in the toolbar at the top of Unity:

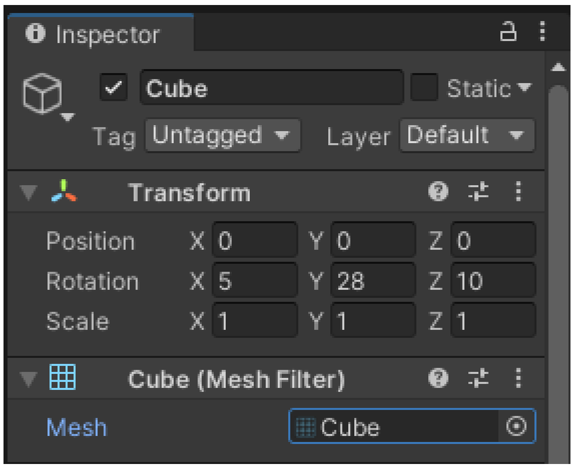


You will then see coloured circles instead of coloured arrows appear on your cube:



Hold your left mouse button down on one of the coloured circles. Keep it held down and drag. Notice which way the cube moves as you drag. Try all the coloured circles and note which ones are the X, the Y, and Z axis. Does the cube rotate the way you expected? If not, bear in mind that it's rotation on an axis. For example, the Y axis goes up and down. The cube will, therefore, rotate from left to right (or vice versa) .

You can, of course, simply type a value in the text boxes. In the image below, we've rotated the cube 5 degrees on the Y axis, 28 on the Y axis, and 10 on the Z:


Inspector

Cube Static

Tag Untagged Layer Default

Transform

Position	X 0	Y 0	Z 0
Rotation	X 5	Y 28	Z 10
Scale	X 1	Y 1	Z 1

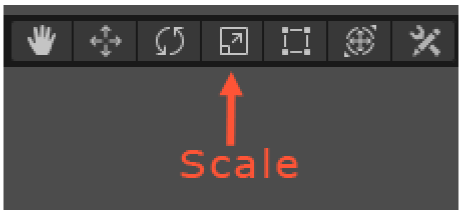
Cube (Mesh Filter)

Mesh Cube

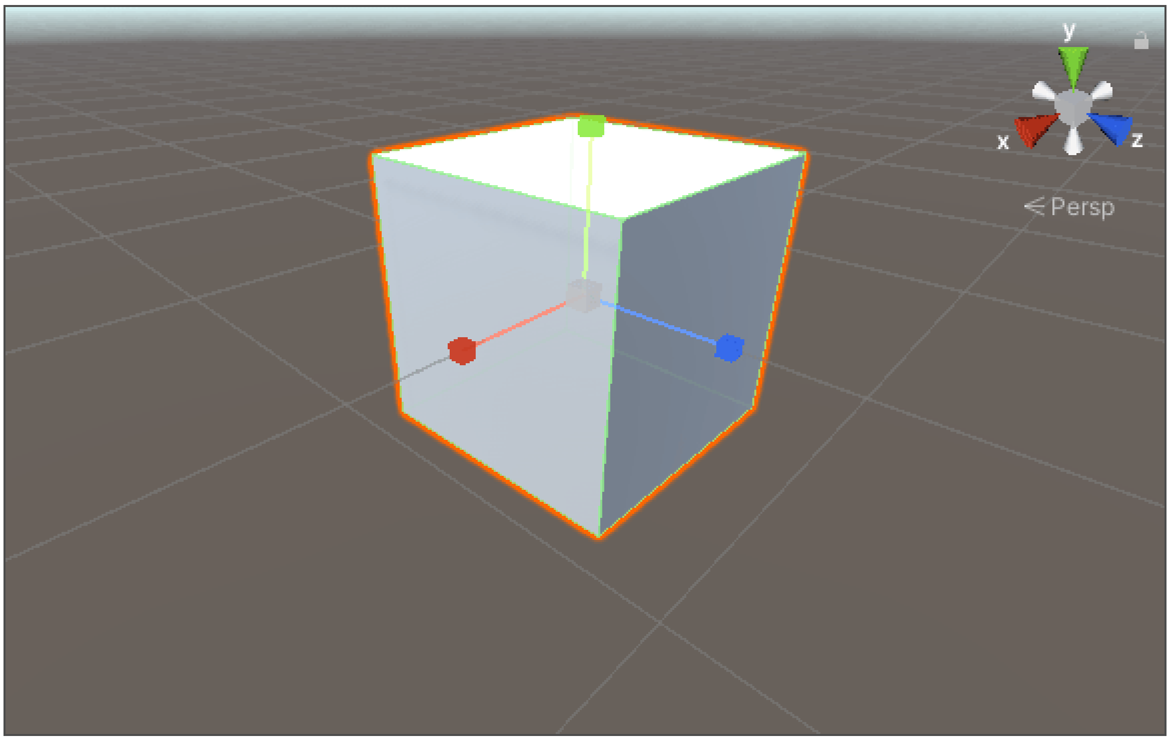
To get your cube back in the rotation it was before, you can reset it, just like you did with the Position. Clicking reset from the menu resets the Position, the Rotation, and the Scale.

Scale

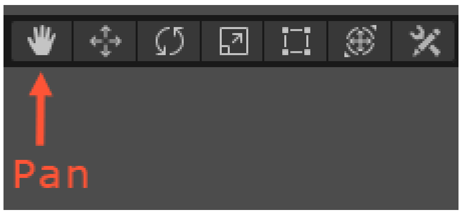
You can also scale the cube. You can type new values into the Scale textboxes. Or, you can use the Scale arrows. To do that, click the Scale icon on the Unity toolbar:



When you click the Scale icon, you'll see arrows appear on your cube. This time, the end points of the arrows are squares:



Mouse Movement
You can move around the Scene view itself with the aid of your mouse. To zoom in and out, make sure your mouse is somewhere over the Scene. Now scroll your middle mouse button. To pan round, hold down your middle mouse button and drag. Alternatively, click the hand icon in the Unity toolbar at the top:



With the hand tool selected, hold down your left mouse button and drag to pan around the scene.

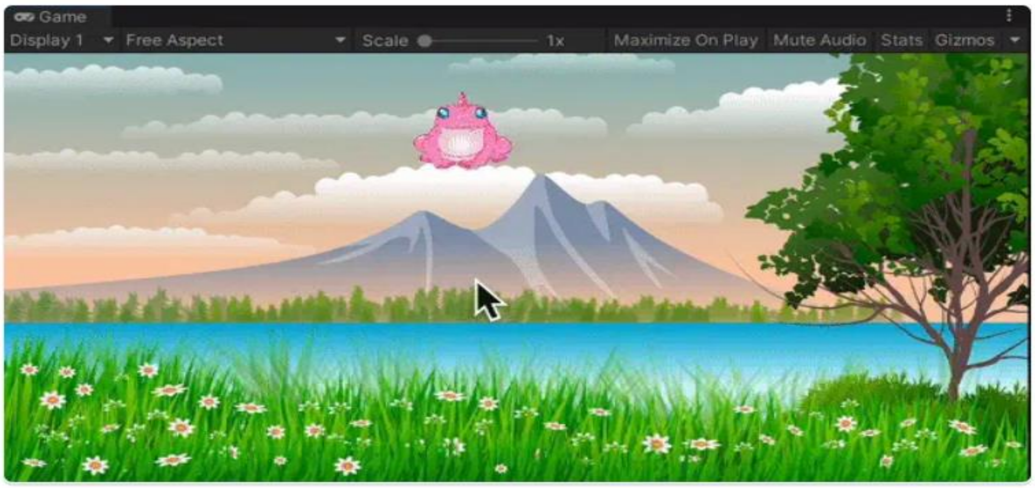
If you hold down your right mouse button anywhere in the scene, you can rotate the scene itself to give you a better view of things.

So, to sum up:

1. **Mouse Wheel**: scroll your mouse wheel to zoom in and out.
2. **Mouse Wheel**: Hold down the mouse wheel to pan.
3. **Left Mouse Button**: Select the hand icon from the toolbar, hold down your left mouse button and drag to pan around your scene.
4. **Right Mouse button:** Hold down your right mouse button to rotate your scene.

Class practical Revision Exercise in groups

Deliverable output: after stages of developing a game using frog object you will be required to write C# script when you point the frog it gets killed.



Requirements / instructions step by step follow the blog link below

https://blog.sentry.io/unity-tutorial-developing-your-first-unity-game-part-1/

https://blog.sentry.io/unity-tutorial-developing-your-first-unity-game-part-2/