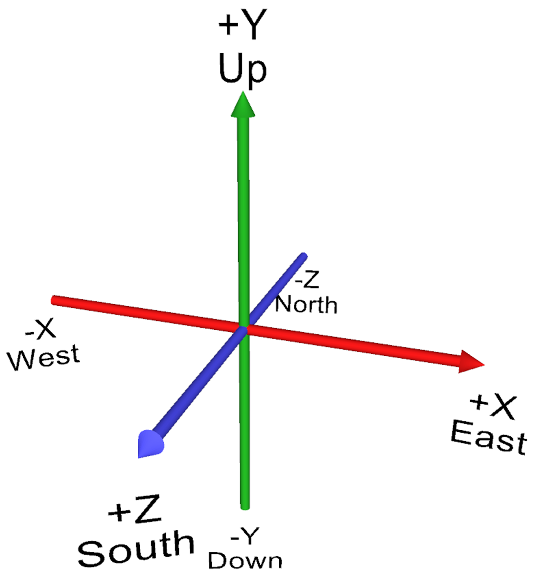
Chapter One: Introduction

Welcome to MetaUnite, the interactive Unity 3D tutorial in the Meta augmented reality experience. By the end of this tutorial, you will have learned the fundamentals of how Unity games are created. While you still may need to learn the nuances of the editor and how to navigate it, you should be very familiar with the way Unity scripts interact with GameObjects, how to write C# scripts for Unity, and how Unity physics work and how to apply it.

Unity World and Objects

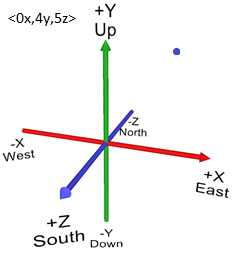
When Unity loads a virtual space, it is in structures known as ‘scenes’. Scenes separate spaces from other spaces and are loaded independently of each other. These spaces can contain many different sets of objects and can even share objects. Let’s load an empty scene.

(A demonstrational 3D axis will appear in the headset).



In this scene, we can see a grid showing our world space. Unity always begins this space at

<0, 0, 0>. This represents our X, Y, and Z axis. Let’s try plotting a point on this plane.



(A point appears on the 3D plane, and text appears which shows the point’s coordinates)

You can move this point around in the 3D space in front of you. Why don’t you try grabbing it and watching how it changes the coordinates? These coordinate values represent the point’s space in the world, which is called world position.

Unity will often map points in vectors, that appear in parentheses such as this:

Vector3 newCoord = new Vector3(x,y,z);

The x, y, and z here represent different values that can be plugged into this constructor. Variables and equations are also accepted in this field, so long as they can be evaluated as float values. Let’s give it some random values for now.

Vector3 newCoord = new Vector3(5, 3, 6.5);

All we’ve done so far is created the Vector3 variable; however, we haven’t applied it to anything yet. Let’s go back to our scene.

Unity loads objects into its world space, which are known as GameObjects. These GameObjects, even if they are empty, are always given a position in the world space. This is known as a transform. GameObjects can be assigned multiple properties known as components, which can include a wide variety of things, including scripts, 3D models, 2D images, physics properties, sounds, UI elements, and many more. First, let’s assign that Vector3 coordinate to the point GameObject. While we could just manually type in the coordinates we want for that point, assigning a Vector3 as the object’s transform position may let us change these values using other code functions down the road.

I’ll assign a script to this point GameObject. In programming terms, this is equivalent to creating a class. Scripts can also be assigned to multiple GameObjects. In our void Start() method, which is similar to a main method that is called upon loading the object, we’ll assign this vector to the object’s position.

public class Point : MonoBehaviour

{

Vector3 newCoord = new Vector3(1, 3, 4);

void Start()

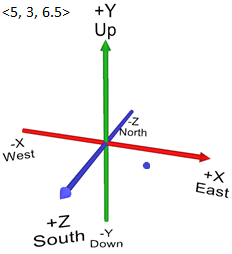
{

transform.position = newCoord;

}

}

Now, let’s see how that point’s position gets affected by this code once we reload the scene.



Amazing! Note that these changes are only visible because we reloaded the scene, and the GameObject was freshly created. Any changes made to these coordinates would not be visible live, but only if the GameObject were freshly loaded. In order to make changes to a GameObject after it’s been loaded while the game is still running , we need to either put it into a Unity method that is running alongside the game, such as Update(), FixedUpdate(), or LateUpdate(), or we need to put it into its own method that gets called whenever you wish to update that coordinate. Now, onto the next lesson.