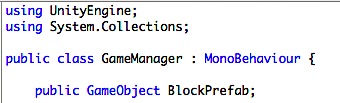
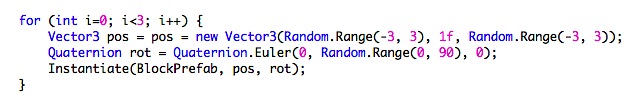
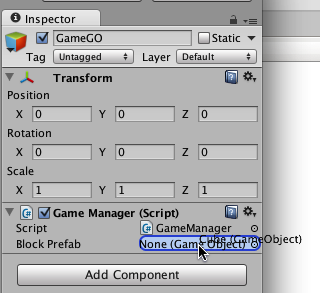
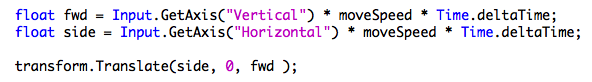
**Unity 3D Tutorial**

1. First, get the 3D Resources files from GitHub. You should have three files: tile.psd, block.psd, and GooglyEyes.unitypackage.
2. Open Unity. Create a new project: File > New Project. Don’t import anything, just pick a file location, give it a name, and click Create Project.
   1. Arrange your screen so that you can see all of the following panels: Game, Scene, Hierarchy, Project, Inspector.
3. Now save your scene (even though it doesn’t have anything in it yet): File > Save Scene As… Give it a name and save it in the Assets folder.
4. We’ll add a couple of assets. Make a new folder under Assets, called Textures. Drag in the two graphics files from GitHub Resources into the folder. You can do this either in the operating system or in Unity itself.
5. The first thing we’ll make is a plane that will be our floor. In the menu, go to GameObject > Create Other > Plane.
   1. Select the plane in the Scene tab, and look at the Inspector tab. The transform of the plane is displayed. Change the position to (0, 0, 0).
   2. It’s pretty small, only 10 x10. Let’s scale it up a little: Change scale to (2, 1, 2). Now it is 20x20 with coordinates ranging from -10 to 10 in the x and z directions.
   3. We will use the tile texture on the plane. Open the Textures folder, then select the plane so it is visible in the Inspector. Drag tile.psd onto the plane. This creates a Material and applies it.
   4. Currently, we can see the tile as one big image, but we want it to repeat it (tile it). In the Inspector, with the plane selected, go to the texture, and where it says Tiling, enter 20 for x and for y.
6. We need a good view of the action, so we’ll fix the camera angle and get more light.
   1. Camera first: Select Main Camera from the Hierarchy. In the Inspector, change its Y position to about 6, its Z position to about -12, and then change its X rotation to something that give you a good viewing angle (about 40).
   2. Now some light. Go to GameObject > Create Other > Directional Light.
   3. We’ll also add some ambient light, so shadows are not too dark. Go to Edit > Render Settings and click on the color block for Ambient Light. Set it to a mid-gray (change V slider value to about 120). This lightens the darkest areas.
7. We are going to make some blocks through code. First, we need to set up a “prefab” of the block that we will instantiate. Create a cube (GameObject > Create Other > Cube).
   1. Select the cube it in the Inspector and drag the block.psd texture onto it.
   2. Set the cube’s position to (0, 0.5, 0).
   3. Select it in the Hierarchy, and drag it over into the Project tab. This gesture just made the cube into a prefab, which we will use to instantiate multiple blocks.
   4. Go back to the Hierarchy, and delete the Cube there. We’ll make them from code instead.
8. Now we’ll write some code.
   1. First, we need a place to attach our scripts. It is common to use an empty GameObject for this. Go to GameObject > Create Empty. This places the empty GameObject (now visible in the hierarchy) into our scene. Rename it to GameGO (This is a convention that I use, to name empty GameObjects used for holding scripts with a “GO” suffix).
   2. Create a new folder called “Scripts” (Assets > Create > Folder). In that folder, create a new C# script. Rename it GameManager before editing.
9. Double-click the GameManager script to open it. It will open in MonoDevelop by default (it will take a minute the first time, because it has to open the program).
   1. When the script opens, you will see some stubs for Start (used for initialization) and Update (used for frame-based code).
   2. Create a public variable for the block prefab:
   3. In the Start function, we’ll make three blocks at random locations. The Instantiate function requires three parameters: a prefab reference, a Vector3 position, and a Quaternion rotation. Here’s our code:
   4. Save the file, and go back to Unity. Select GameGO and drag the GameManager script onto it. The script will appear in the Inspector, and the public variable BlockPrefab will be visible in the inspector, too. Note that it does not currently have a value (it says None). Give it a value by dragging the Cube prefab onto it, as shown in the image.
10. Save and play in Unity (CTRL + P or click the arrow at the top of the screen). Three lovely blocks will appear!
11. Next, we want to drive a character around. So we need a character. Find the GooglyEyes.unitypackage that you downloaded. Go to Assets > Import Package > Custom Package and find select the file. It will appear in the Project tab, in a folder called Prefabs. (You might as well put the cube prefab in there, too, just to be organized.)
12. We’ll instantiate the googlyeyed guy. In the GameManager script, add two more variables:
    1. GuyPrefab, also a public GameObject
    2. myGuy, a private GameObject
13. Give GuyPrefab a value by dragging the guy prefab onto the variable in the Inspector.
14. Instantiate the guy in the middle of the plane:
15. Run the program, and your guy should appear. Next we’ll make him move. Create a new C# script (put it in the Scripts folder) called PlayerController. Give it a public float variable: moveSpeed (initialize to 10).
16. In the Update function, we will get user input and use it to move the character. Input is controlled by an Input Manager, but by default we have reasonable values. Inputs tyally return a value between -1 and 1, so for example, input defined as from a “Vertical” controller (like up/down arrow keys or W/S) will give us -1 for W or up, and 1 for S or down. We’ll multiply that times our moveSpeed, then change our position. Similarly, we will get values from the “Horizontal” input, and map them to rotation. So here’s our Update() function:

Note that we multiply our values by Time.deltaTime. This is because we are calculating the speed and rotation per second, but Update runs several times per second. Because we can’t be certain of the exact delay between frames, Time.deltaTime returns that value for us, and lets us increment by the appropriate fraction for this time step.

1. Drag the PlayerController script onto the Guy prefab. Run the program. Move your guy!

OK, so maybe it’s not the best movement control you’ve ever experienced. But it’s a start. Odd, though – you can drive him right off the world. And he goes right through the boxes. This is because he has no physics applied to him. Let’s add some physics, just for fun.

1. Select the cube prefab. At the bottom of the inspector, there is a button that says Add Component. Click this and select Physics > Rigidbody.
2. Select the Guy prefab and do the same thing. Run the program. What happens?
3. Ok, that was weird. What happened? His eyeballs are too heavy!
4. We can fix this. In the guy prefab, twirl down the rigidbody to see its attributes. At the bottom is Constraints. Open that. We only want him to rotate along the Y-axis, so freeze the other rotations.