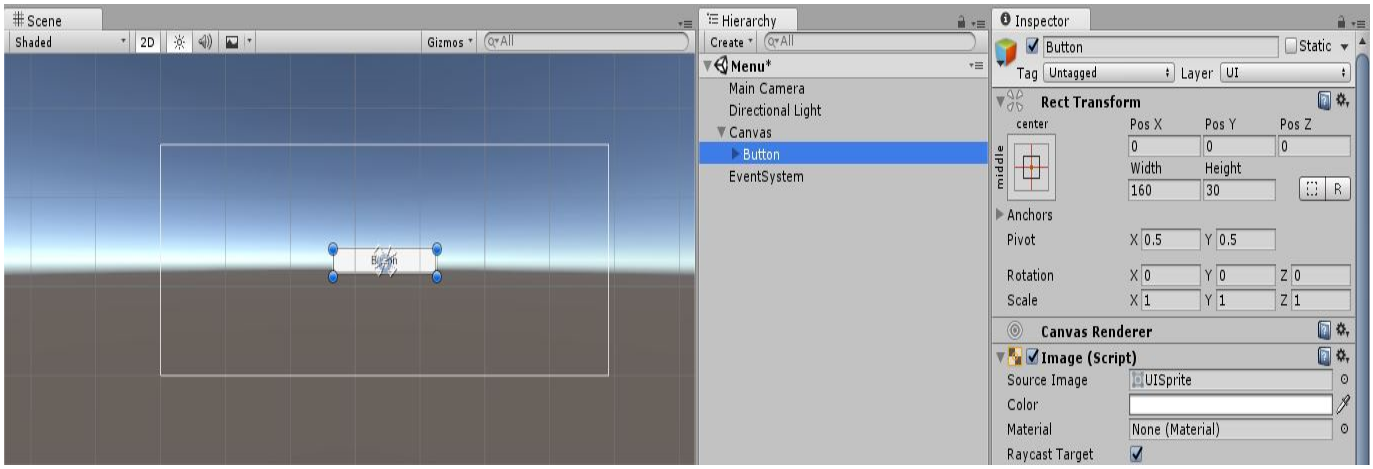
Pacman Tutorial

Open Unity3d and create a 3D project. On the project section, right click on Assets and create 4 folders, one named “Scripts”, one named “Levels”, one named “Prefabs” and one named “Materials”. Save your scene (File> Save Scene) under the “Levels” folder and name it “Menu”.

Go to GameObject> UI> Button and create a button. From the Scene section switch to 2D view and locate the button within the “Canvas” which is its parent object. Drag and drop the button right to the middle of the canvas, or simply change its transform to Pos X: 0 and Pos Y: 0.



Locate the Button object in the Hierarchy. Expand it and find the Text component. If you expand the “Text (Script)” section you can change the appearing name from “Button” to whatever you like. For now, change the name to “Play”. We are trying to make an initial Play Button for the player to press and commence the game. Of course all these actions are only making an interface for a button. If you start the game at this point the Play button will appear but it is neither clickable nor functional in general. In order to work, it needs a script attached.

In the Scripts folder created at the beginning, right click Create> C# Script and name it “Menu”. Erase completely the Start and Update functions, we do not need them as this will be a script not used in every frame of the game. Your new script should be the following:

```
using UnityEngine;
using System.Collections;
using UnityEngine.SceneManagement;

public class Menu : MonoBehaviour {

    public void buttonClick()
    {
        SceneManager.LoadScene (1);
    }
}
```

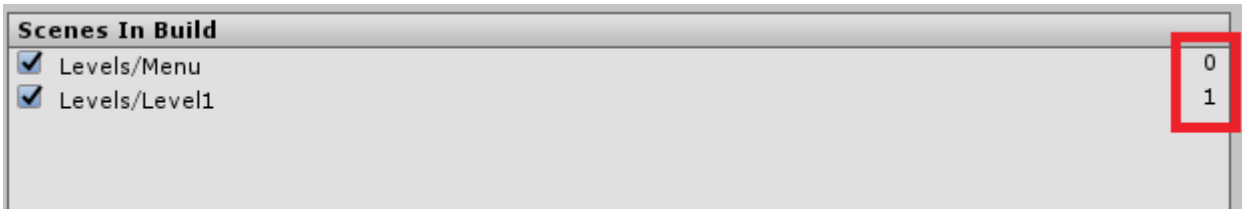
We have created a function that loads the game level when the button is pressed. But to complete the button action, we still have to make some other arrangements within the game engine.

Navigate to the Canvas in the Hierarchy section. Drag the Menu script in its inspector and then expand it to find the Button child. Scroll all the way down until you see an “On Click ()” subsection stating “List is Empty”. Click on the + symbol on the bottom right and you can now see a list with 3 tabs, “Runtime Only”, “No Function” and “None (Object)”. Click on the third one (on the small circle it has on its right end) and from the popup window choose “Canvas”. Now the second tab should be clickable.

From this new dropdown, click on Menu> buttonClick(). This way you assign an appropriate clicking behavior on the button you created earlier.

Go to File> Build Settings, click “Add Open Scenes” and close the window. Save this Scene one more time, and create a new Scene via File> New Scene under the “Levels” folder. Name it “Level1”. Go once more at the Build Settings and click again to “Add Open Scenes”. You can now notice that in our script we added the line: `SceneManager.LoadScene (1);`

So, why did we tell to the scene manager to load the scene named “1” and not the scene named “2”? How was this number chosen? While you are at the Build Settings, notice these numbers:



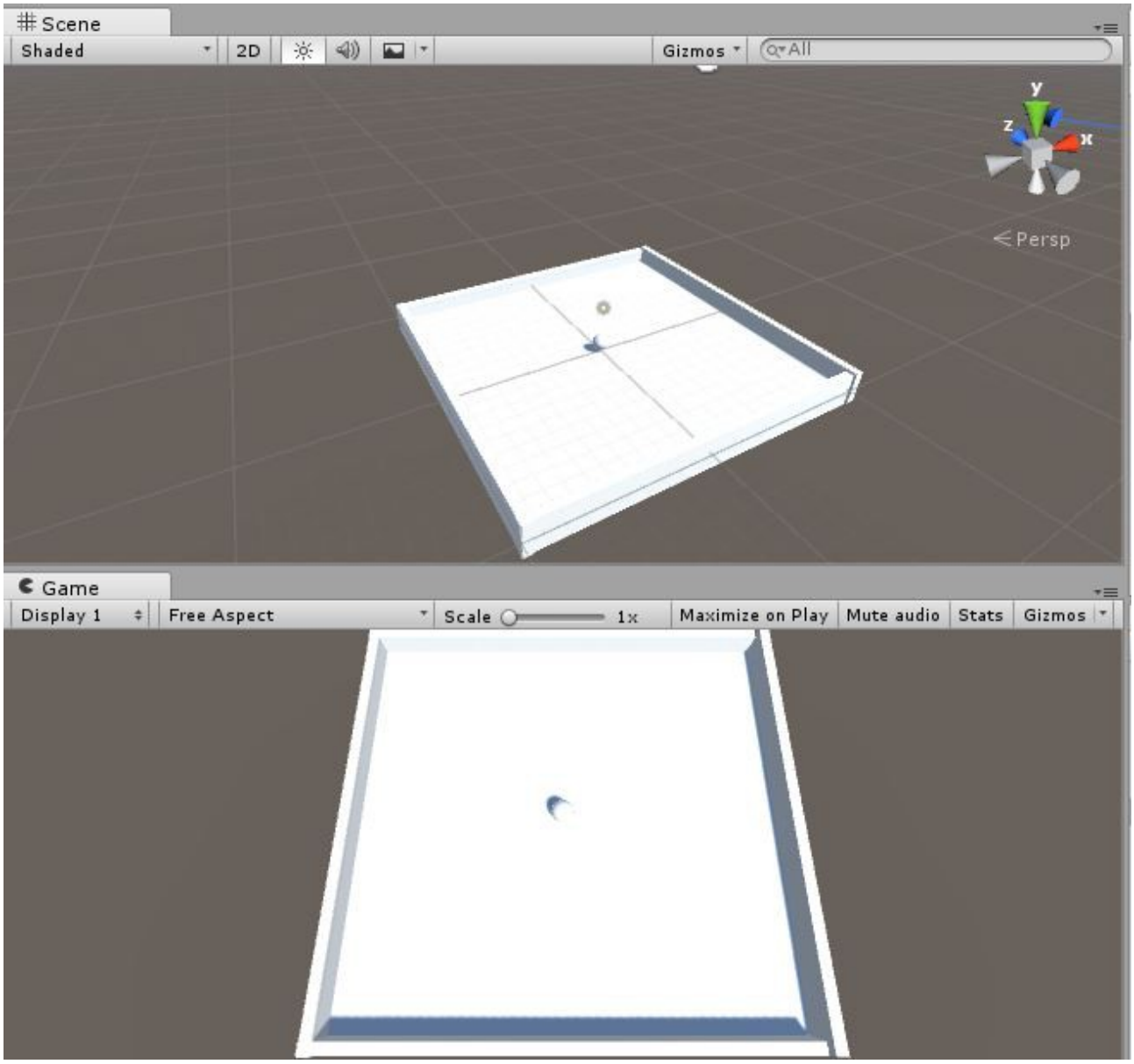
What we did with this process was to make the button in scene 0 to load us the scene 1. You can always look at the build settings to figure each scene's index.

Now let's get back to our Level1. Make sure you have saved all of your progress so far!

Unclick the 2D view. Hierarchy> Create> 3D Object> Plane. Zoom out in the scene and make sure you have a view of the whole plane. Make sure “Static” is ticked right next to Plane's name in the Inspector. Scale it to X: 2, Y: 1, Z: 2. Go to Window> Navigation and at the new window that appeared next to the Inspector, make sure that the Navigation Area is set to “Walkable”. This plane is going to be the main arena. Now time to make 4 walls. Hiearchy> Create> 3D Object> Cube. Change its scale: X: 20, Y: 2, Z: 0.5 and then proceed to duplicate it 3 times and twitch all the

duplicated walls accordingly in order to make a fence around the plane. This might take a while – don't give up. After all they don't have to be perfectly alligned!

Now time to make some characters. Hierarchy> Create> 3D Object> Sphere. On its transform section, click the small cogwheel and select “reset”. Pick it up just right over the plane we created. Go to Assets> Import Package> Characters and wait for the package to load. At the new “Import Unity Package” popup, click “Import” at the bottom right corner. You should be able to see at the Project section, under the Assets a new folder named “Standard Assets”. Navigate to RollerBall> Scripts under Standard Assets and drag the two scripts named “Ball” and “BallUserControl” on the sphere we created. Click on the sphere and Add Component> Physics> Rigidbody. Rename the sphere to “Player” and change its tag to “Player” as well. At this point play with the Main Camera's position in order to have an adequate view of the floor and the sphere when you hit play. It should look like this:



We made the player. Now time for some ghosts! Which of course cannot be common spheres, but it is probably better for them to be represented as Capsules. Hierarchy> Create> 3D Object> Capsule.

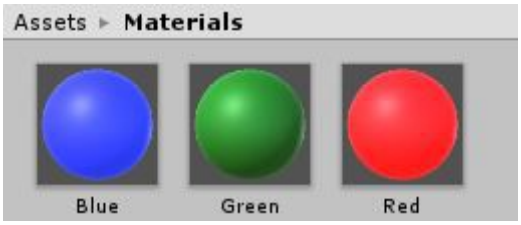
Add Component> Character Controller and add another component, Nav Mesh Agent. Under the Scripts folder, create a new C# script and name it “Ghost”. Make it look like this:

```
using UnityEngine;
using System.Collections;
using UnityEngine.SceneManagement;

public class Ghost : MonoBehaviour {

    public GameObject target;     //ghost's target (the player)
    NavMeshAgent agent;            //self reference to the ghost game ob
ject

    // Use this for initialization
    void Start () {
        agent = GetComponent<NavMeshAgent> ();
        if (target == null)
            target = GameObject.FindGameObjectWithTag ("Player");

    }

    // Update is called once per frame
    void Update () {
        //Update the target's location
        agent.destination = target.transform.position;
    }

    public void OnCollisionEnter(Collision collision)
    {
        if (collision.gameObject.tag == "Player")    //When hosts collid
es with player, load the initial menu
            SceneManager.LoadScene ("Menu");
    }
}
```

The OnCollisionEnter function is used to decide what to do when the ghost collides with the player. Back to Unity3d, on the Navigation tab (Window> Navigation), on the Bake tab, click the Bake button on the bottom right corner. Hit play. Watch the capsule chase away the player (you). In every frame the ghost is informed about the current player's position and acts accordingly (aka follows the player).

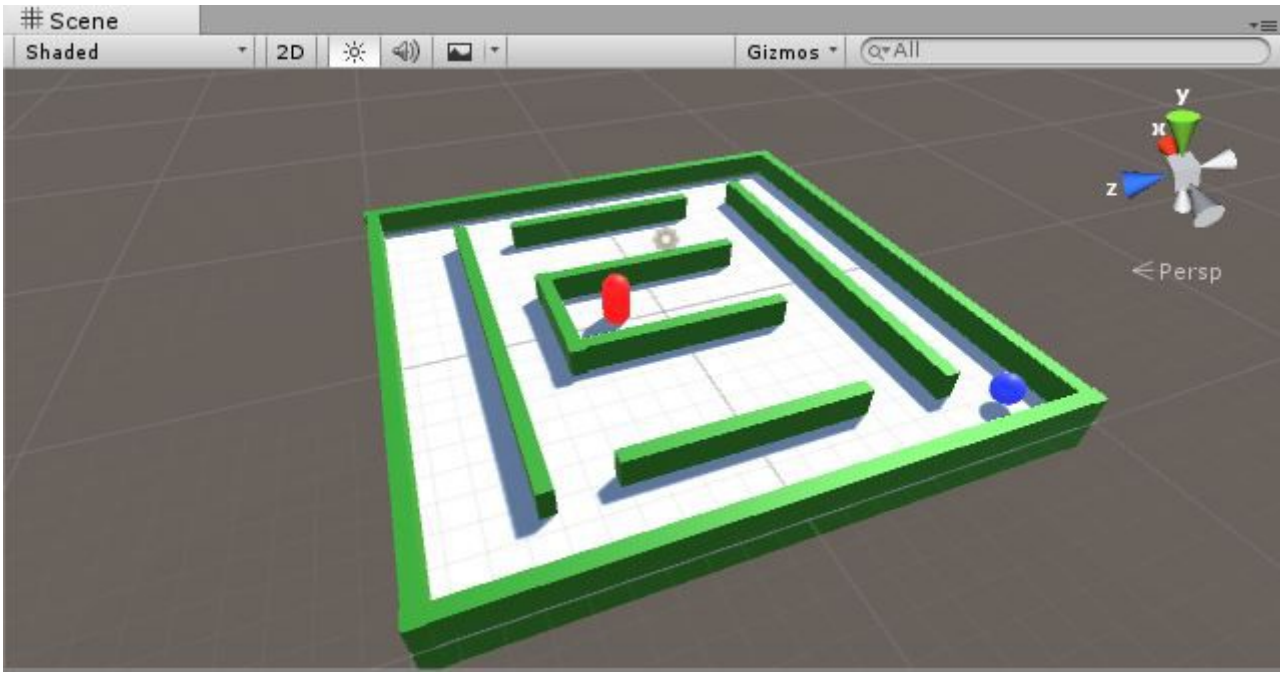
Time to make the map a bit more complex. But before doing that, we should give some colour to the player, the ghost and the walls so as to make everything distinguishable. Under the Materials folder we created at the beginning of this tutorial, right click and Create> Material. Name it Blue.

Twitch the “Albedo” selection to any blue you like (or any other colour you want, just rename the material afterwards to something more appropriate). Do the same for two more materials, for example red and green.



Drag and drop each material to each different category of objects. For instance, drag green to all the walls, blue to the player and red to the ghost.

Back to making the map more complex. Duplicate one of the walls into four more and scale them down to X: 15. Duplicate one more and scale it down to X: 5. At this point these are just some examples of a sample map, feel free to make your own.



Now click on all the walls and from the Inspector select Static> Navigation Static. Go to the Navigation window and click bake on the bottom right corner. By making the walls navigation static is making them unpassable obstacles. The ghost or the player now collide with them. Try to play the game and check out the ghost's pathfinding abilities!

Of course only one ghost is not making this pacman version very challenging. We need more ghosts, and this is why we will need a Ghost Prefab.

“The prefab acts as a template from which you can create new object instances in the scene. Any edits made to a prefab asset are immediately reflected in all instances produced from it but you can also override components and settings for each instance individually. “

Drag the Ghost object from the Hierarchy under the Prefabs folder under Assets. You will see the Ghost in the Hierarchy being blue now. That means it is a prefab copy. Duplicate the ghost and

make two more. Scatter them on the map. Go to a random ghost's inspector and on the Nav Mesh Agent section and under Steering change Speed to 2. Hit “Apply” on the top right at the inspector and all of the ghosts will now have this new steering speed. This is why we got ghost prefabs in the first place after all!

In the Π shape right in the middle of the arena there will be the place the player has to reach in order to load the next level. Hierarchy> Create> Create Empty and reset its transform. Add Component> Box Collider and edit the collider manually to make it occupy the whole Π area. You are basically resizing a transparent cube. Make a new C# script under the scripts folder and name it “NextLevel”. Edit it to be as following:

```
using UnityEngine;
using System.Collections;
using UnityEngine.SceneManagement;

public class NextLevel : MonoBehaviour {

    public void OnTriggerEnter(Collider collider)
    {
        if (collider.tag == "Player")
            SceneManager.LoadScene (SceneManager.GetActiveScene ().buildI
ndex + 1); //load the next level on collision
}
```

The OnTriggerEnter just like the OnCollisionEnter function is used to manage the actions following a collision with the item we added the collider to (the empty game object). Drag the script on the NextLevel object.

Save the scene and then right click on it (fromt he Assets) and Edit> Duplicate. Automatically the new Scene is going to be named “Level2”. Open the new scene and File> Build Settings> Add Open Scenes. Level2 will have the number 2 assigned by default.

The game is ready to run! If you feel that the ghosts are too fast, you can always make them slower. Add more obstacles or resize the whole map, do not forget though to change the Navigation settings each time.

Enjoy!