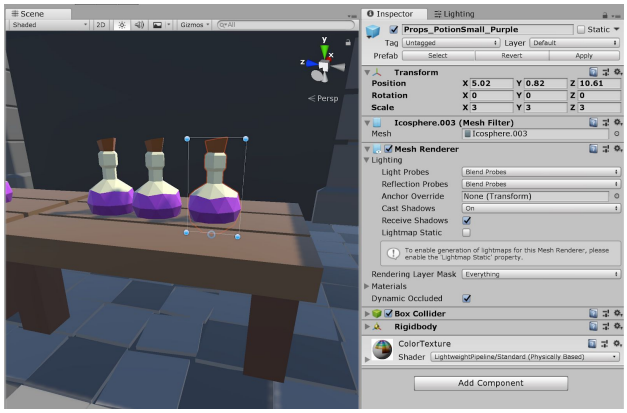
CSE 490j
Unity Scripting Assignment

The Object-Component System

To understand Unity scripting, first it is important to understand Unity notion of Object Component Relationships. All things in your scene are game objects. Each game object has its own set of components that help define certain properties such as physics, colliders, position information, etc.

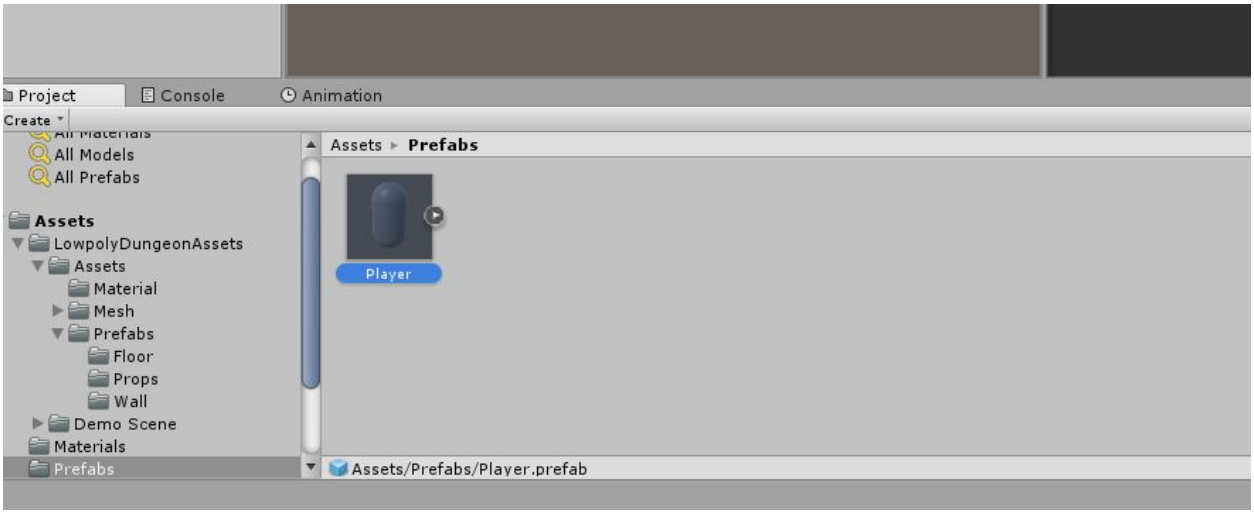
These components can be viewed by selecting the object and looking at the inspector on the far right.



As you can see, this object has a transform component, a mesh filter, a mesh renderer, a box collider and a rigidbody. From the inspector, you can see these properties update as you modify them in the scene as well as modify most of these

Setting up the scene

- Delete your current Main Camera
- Go to the project Window > Prefabs > Player



- Drag the prefab into your scene where you want the player to start

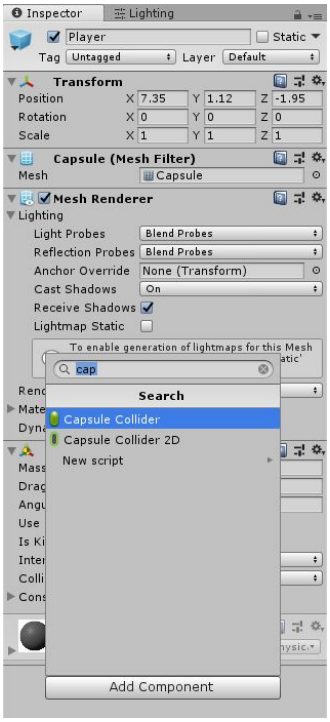
If you set this up right, your game view will now show you the position of you player. If you click on the player, you will see it has a number of components attached to it.

- Transform - contains information about position, rotation, and scale
- Capsule (Mesh Filter) - controls the mesh of the objects
- Mesh Renderer - controls properties such as shadows and materials
- Rigidbody - controls the physics properties of the object such as gravity and mass

Try to mess around with some of these properties (the easiest would probably be the transform properties), see what happens. A lot of getting used to Unity is learning about these components and what they do.

If you hit play at this point, your player should fall through the floor. This is because the rigidbody component has added gravity to your player. However, we want the player to collide with the floor. In order to do that, we need to add colliders to certain objects.

- Click on Player
- Scroll down and click “Add Component”
- Search for and click on “capsule collider”



Collider components add collider data to objects. Specific shape colliders utilize different shapes. All colliders will automatically try to conform to the shape game object.

You can modify a ton of the collider settings in the inspector. However, the most useful one is the edit collider button. This will highlight the collider in the scene view (outlined in green) and you can click on the green nodes to modify the shape.

Now if you hit play you will see … that your player still falls through the floor… This is because you haven’t added colliders to the floor. This would be tedious to do one by one, but we can actually do it all at once.

- Select all of your floor objects in your scene (through the scene view or the inspector)
- Go into the inspector, and add a box collider.

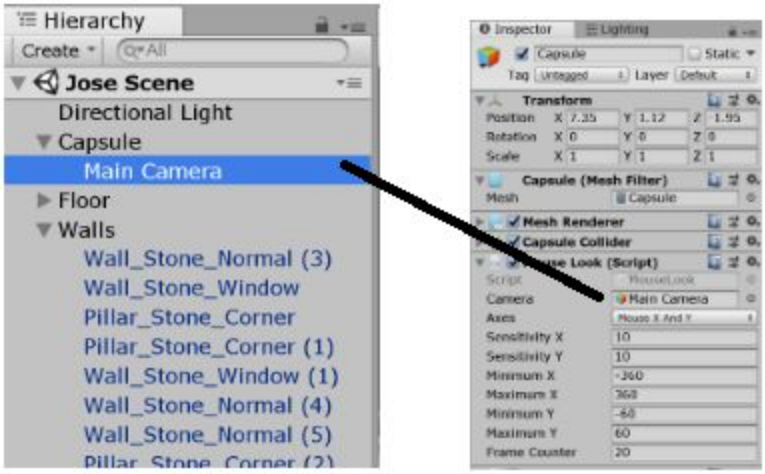
Now all of your floor objects should have colliders on them, and if you hit play, your player should stop when it collides with the floor.

Now go into each of your other objects (walls, props, etc.) and add colliders to them. Experiment with different shapes to see what fits best.

Add Script to the Player

Scripts in Unity function the same way as other components, in that you can add them to Gameobjects in your scene. We will be adding a mouse look script to your player, so that you can look around freely while you play.

- Go to project panel > scripts
- Click and drag the MouseLook.cs onto your player gameobject
    - Note: you can actually also do this by clicking on your player, then in the inspector “add component” and searching “MouseLook” just like any other component
- In the Mouse look component you will see a property, camera, that currently says “none”
- In the inspector, you should see below the player a gameobject that says “Main Camera”
    - Note: gameobjects and even components can be properties of other components

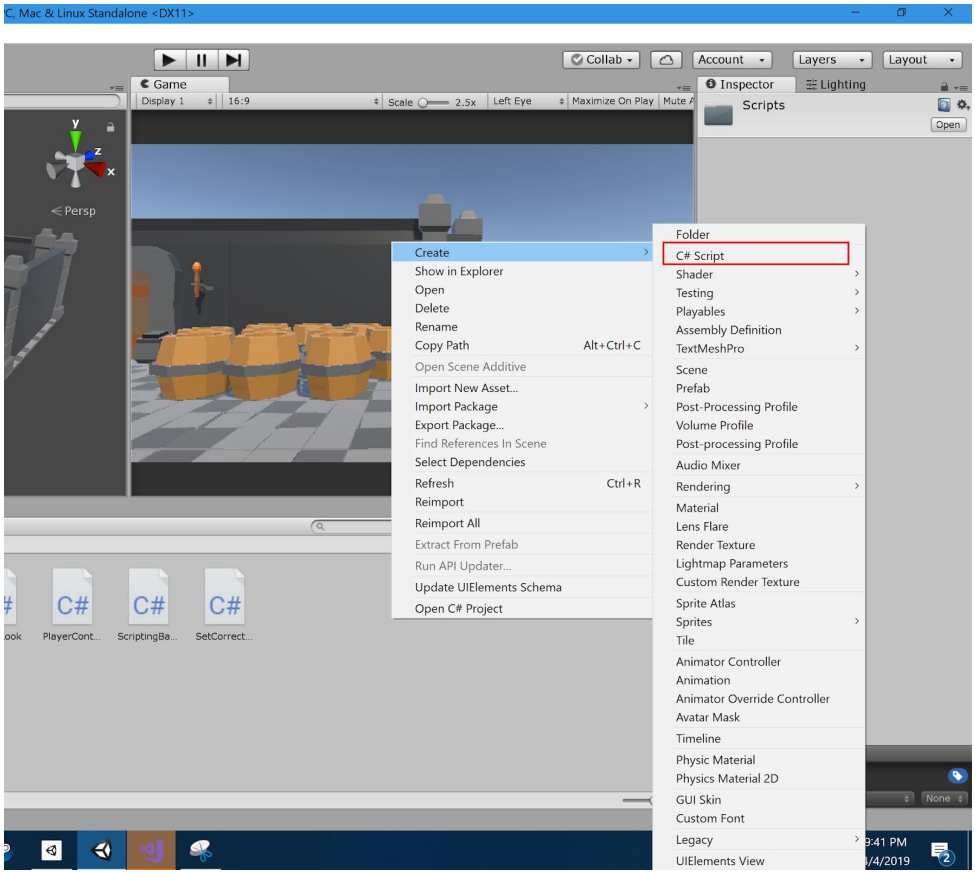


Now if you hit play, you should be able to look around the world with your mouse. Take a look at the mouse look settings and try playing around with them. Get the sensitivity of the mouse to feel good to move around. If you are curious about what is going on behind the scenes, feel free to click on the script in the project tab and take a look at the underlying code.

So now that we have learned how to use and interact with scripts in the editor, lets learn how to write some. If you haven’t already, take a look at the “Scripting Basics” page (that I am still currently writing) to get a general sense of how Unity scripting works.

Player Movement

First let us create a new C# script

- Go to your projects tab and open the scripts folder
- Right click > Create > C# Script
- 

- Name the script “PlayerController”
- Then add that script to the Player Game Object (either by dragging it or going to “Add Component” in the Inspector)
- Double click on the on the script to open up the scripting editor

Alright so now that we are here, let's write some code. When you open it up, you should see something like this:

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

public class PlayerController : MonoBehaviour {

        // Use this for initialization
        void Start () {

        }

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

        }
}
```

As the comments suggest, any code put in between the brackets of the Start function will run once at the beginning of play, and any code put in Update will one once every frame. In our case, we don't need Start, so feel free to delete it. Next we are going to establish some variables.

```
public class PlayerController : MonoBehaviour {

    public float speed = 5;
    public Rigidbody rigid;

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

    }
}
```

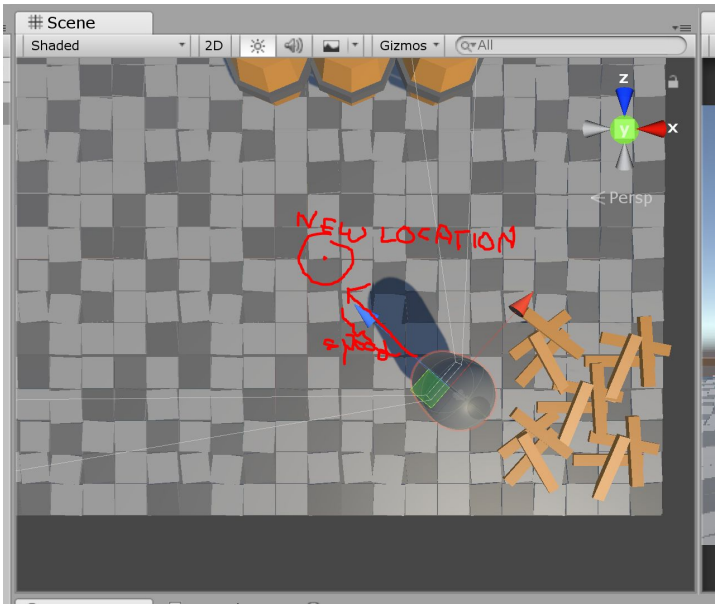
The speed variable is how we will control how fast the player will move when we hit the input. The rigidbody variable is will be a reference to a rigidbody component in the editor. Note that, since they are marked public will appear in the editor as a property that you can manage without having to go back into the code.

Next, let's tackle forward and back movement.

```
    // Update is called once per frame
    void Update () {
        if (Input.GetKey(KeyCode.W))
        {
            rigid.MovePosition(transform.position + this.transform.forward * speed * Time.deltaTime);
        }
        else if (Input.GetKey(KeyCode.S))
        {
            rigid.MovePosition(transform.position + -this.transform.forward * speed * Time.deltaTime);
        }
    }
```

Alright, there is a lot going on here, so let's unpack this:

- Input.GetKey() is a Unity function that takes in a "KeyCode" (which has settings for virtually any key on the keyboard) and returns true if the key is pressed down
- rigid.MovePosition takes a (x,y,z) coordinate (or a Vector3), and places the object at that location
- transform.position refers to the position property of the transform component of the gameobject in 3d space.
    - Note: you can also grab transform.eulerAngles and transform.scale for the scale and rotation properties respectively.
- This.transform.forward in this case refers to the direction is currently looking in
- Time.deltaTime is a built in unity value that grabs how long each frame takes
- By setting the position as the current position + the direction we want to travel



- to * the speed, we are essentially moving the player forward (or backward)

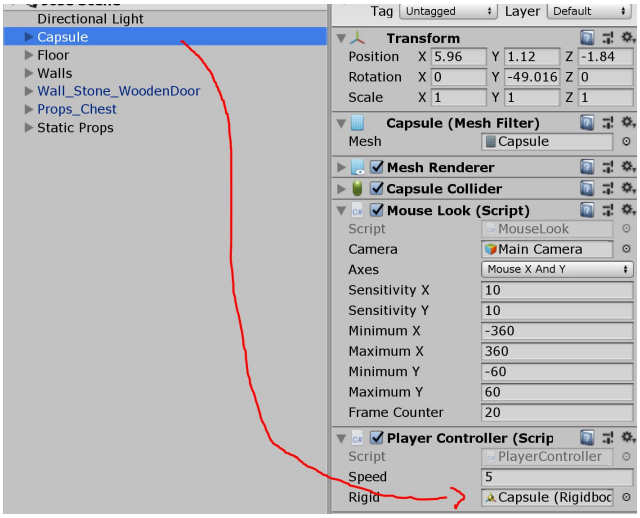
- We flip the forward in the next one, so that we move backwards

Then to do left and right movement, we essentially do the same thing but change to the appropriate keys and direction

```
// Update is called once per frame
void Update () {
    if (Input.GetKey(KeyCode.W))
    {
        rigid.MovePosition(transform.position + this.transform.forward * speed * Time.deltaTime);
    }
    else if (Input.GetKey(KeyCode.S))
    {
        rigid.MovePosition(transform.position + -this.transform.forward * speed * Time.deltaTime);
    }

    if (Input.GetKey(KeyCode.A))
    {
        rigid.MovePosition(transform.position + -transform.right * speed * Time.deltaTime);
    }
    else if (Input.GetKey(KeyCode.D))
    {
        rigid.MovePosition(transform.position + transform.right * speed * Time.deltaTime);
    }
}
```

Congratulations, you have written your first script. Now, go back in to the editor. In the inspector of the player, go to PlayerController script, and where it asks for a rigid, drag the capsule gameobject itself in. Make sure to set the speed value higher than 0 as well.



And there ya go. If you hit play, you player should move in the direction that he is facing.

Even though this code works, it prioritizes clarity for efficiency. There are a number of ways we can make this shorter and easier to use. Here are some extra challenges if you feel up to it:

- Rather than having an if else block,using Input.GetAxis() function and some clever math to see how you can execute that chun in a single line.

- There are tons of different functions to actually move the player, try to find some (such as rigid.velocity). Each come with their own trade offs so explore them.
- Since we always know that the rigidbody we want will be on the object itself, look into the GetComponent<>() function and place it in the Start function.

EXTRA CHALLENGE (or maybe requirement or just heavily suggested I dunno yet): Open the door

So in my scene, there is a door, blocking off the building from the player. Try to write a script that will automatically open the door when the player gets a certain distance to it. Here are some hints.

- Try making a “public GameObject player” variable that you can use to refer to the player in the script.
- Look into Vector3.Distance(), to check the distance between two 3d points
- Use transform.eulerAngles(x,y,z) to manipulate the rotation of the door