Lab 1

Roll-a-ball – Based on: http://unity3d.com/learn/tutorials/projects/roll-a-ball/introduction

Notes on labs:

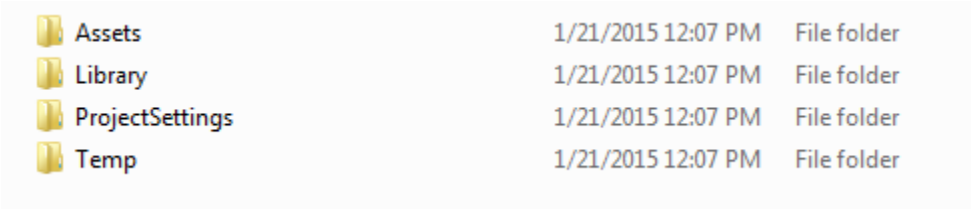
- Optional, although I will take attendance.

Task:

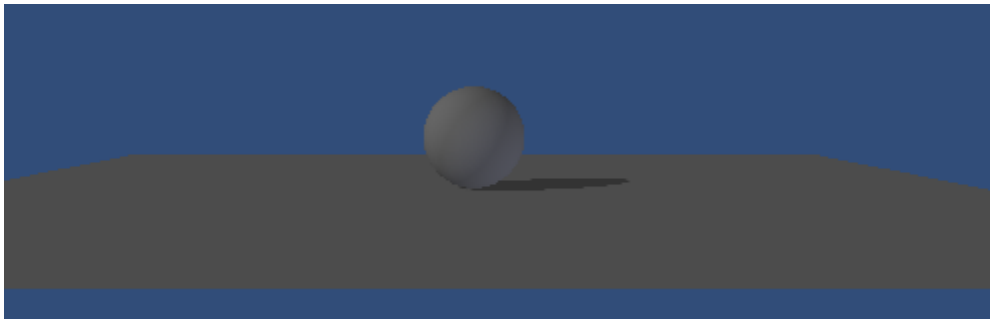
- Create Roll a Ball game that uses physics and forces
- Learn about keyboard input
- Learn about collision between objects.
- No, models, sounds, or animations for this project.

Setting up the Game

1. File-> New Project (Set the destination wherever you want, I recommend creating a folder called lab1)
2. Unity 3D will setup the following folders for you in your project.



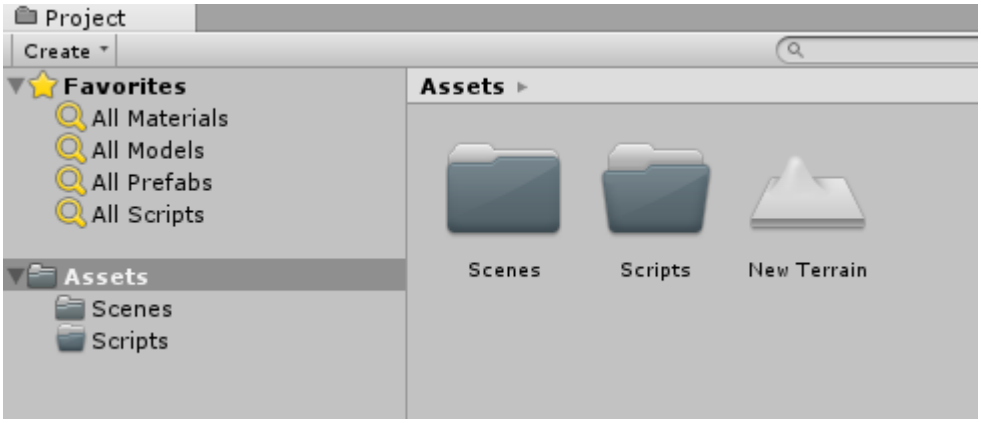
3. Next do a File->Save Scene, and create a new folder in the Assets called “Scenes”, and save the file as “scene1”
4. Now lets create a plane, as our surface to work on. We can either use a terrain, or a plane. I am going to choose to do a terrain.
    a. We then will want to rename the terrain. We can double click the object, and then in the hierarchy anel on the left, rename the object from ‘terrain’ t ‘GameTerrain’
    b. You will also notice, there are handles on the terrain. We can move it around, and transform I I t his way, or otherwise use the inspector panel, by changing the scale.
5. Now lets create our game object, which will be a ‘sphere’. Once again, do GameObject->Sphere
    a. Rename the object ‘Player’
    b. If you cannot find your object, highlight it in the hierarchy and press ‘f’ to focus on that object.
    c. Set the objects position to 0,1,0
6. Add a directional light now, so our scene can have some light. GameObject->Light->Directional light.
    a. Set the intensity to 0.1.
    b. Set the position to 0,5,0
    c. Set angle to 30,60,0
    d. Shadow Type -> Soft Shadows
    e. Resolution -> Very High Resolution
7. Add one more light by duplicating(Ctrl+D) our current Directional Light
    a. Rename it to fill light
    b. Shadowtype-> No Shadows
    c. Adjust the color to a blue-ish tint.
    d. Reverse the angles, so it should be -30,-60,0, so that the light shines up.



Save your scene.

Move the player

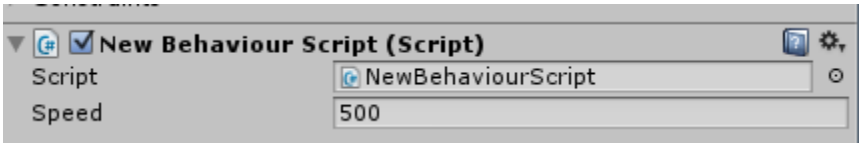
1. Select the Player. You will see in the inspector a way to “Add Component” Select Physics->Rigid Body
2. Now we are going to move the player by manipulating the physics forces. We will write a script to do this. Lets get organized though, and create a scripts folder. Go down to the Project Panel, and then right-click create new folder. Name it ‘scripts’.



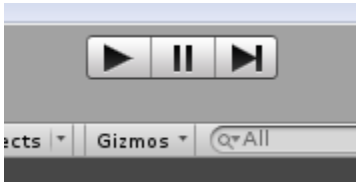
3. We then will ‘add component’ on our object again, which will be a C# Script. This will attach it to the object.
4. Double-click on the script, and then MonoDevelop will be loaded. MonoDevelop is Unity’s primary scripting engine.
    a. You will notice in your script 2 methods.
        i. Start() – Which is what happens during initialization (like a constructor)
        ii. Update() – What happens every frame of the game on this object.
    b. If we want we can delete Start, as we will not need it, or otherwise just leave it for later.
5. Start by typing the word ‘input’ in the ‘Update()’ method.
    a. If you then hold “Ctrl” and press the single quote key ‘, then help will be brought up for you. (Note if you are on Mac, Ctrl is replaced with Cmd)
    b. Here is the final script

```
public class NewBehaviourScript : MonoBehaviour {

    public float speed;

    // Update is called once per frame
    void Update () {
        // Record input from our player.
        float moveHorizontal = Input.GetAxis ("Horizontal");
        float moveVertical = Input.GetAxis ("Vertical");

        // Use the input we get to move the rigid body.
        Vector3 movement = new Vector3 (moveHorizontal, 0.0f, moveVertical);
        // Make a smooth movement by multiplying by Time.deltaTime.
        // Adjust our 'movement' vector by 'speed' to make game playable.
        rigidbody.AddForce (movement * speed * Time.deltaTime);
    }
}
```

c. Save your script in MonoDevelop. Unity is constantly compiling, so it will be ready to go.
d. Return t Unity3D. Now is a good time to save your Unity Scene. (In fact, you should constantly be saving!)
e. Notice if we select our 'Player' game object, it will have a property for speed. Lets set it to 500.



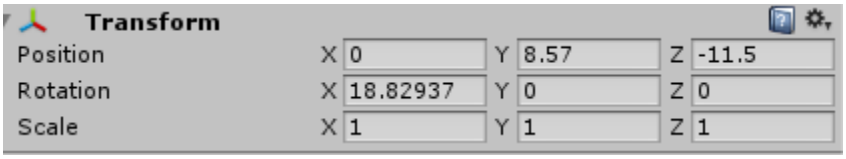
f. Test your game out by pressing the play button.



The Camera

Let us create a new camera, so that it is tied to the player.

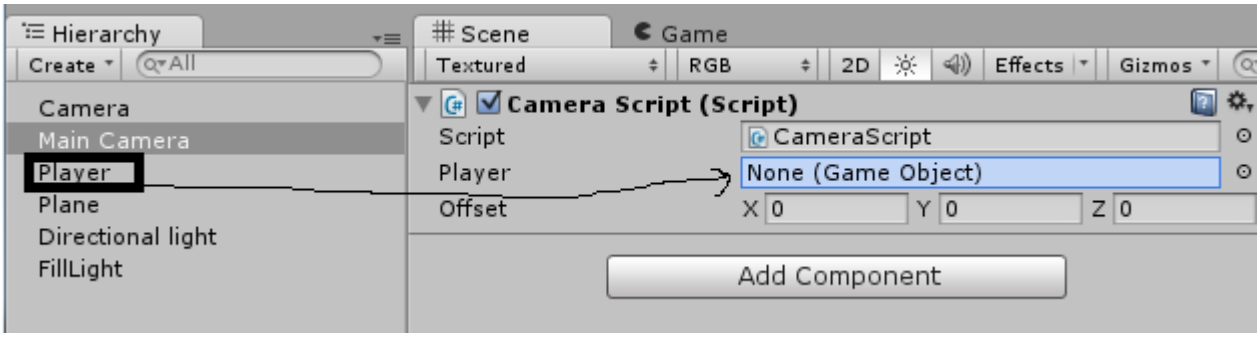
1. Game Object -> Camera (You can replicate my settings roughly by dragging around the camera, or typing them in here)



2. We wil then write a script (Add Component->Script->CSharp, and be sure to move the script into the Scripts folder)

```
public class CameraScript : MonoBehaviour {

    // Reference the player
    public GameObject player;
    // Position of our camera
    public Vector3 offset;

    void Start(){
        // Where we moved the camera in our scene.
        offset = transform.position;
    }

    // LateUpdate is called once per frame
    // But at the end of a frame.
    void LateUpdate () {
        // Our initial positin, plus the updated player position.
        transform.position = player.transform.position + offset;
    }
}
```

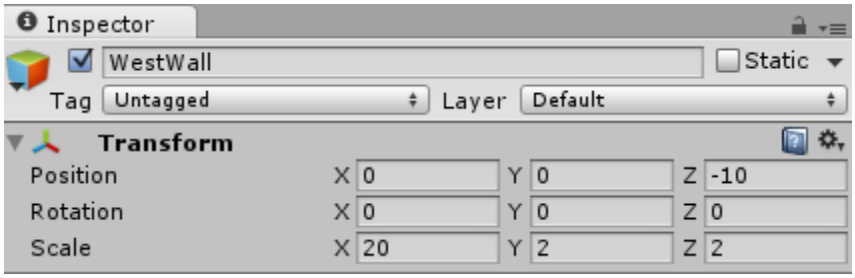
3. Drag the Player into the Camera Scripts's property for player. This creates a reference to that GameObject(in this case, our Sphere) GameObject in our script.



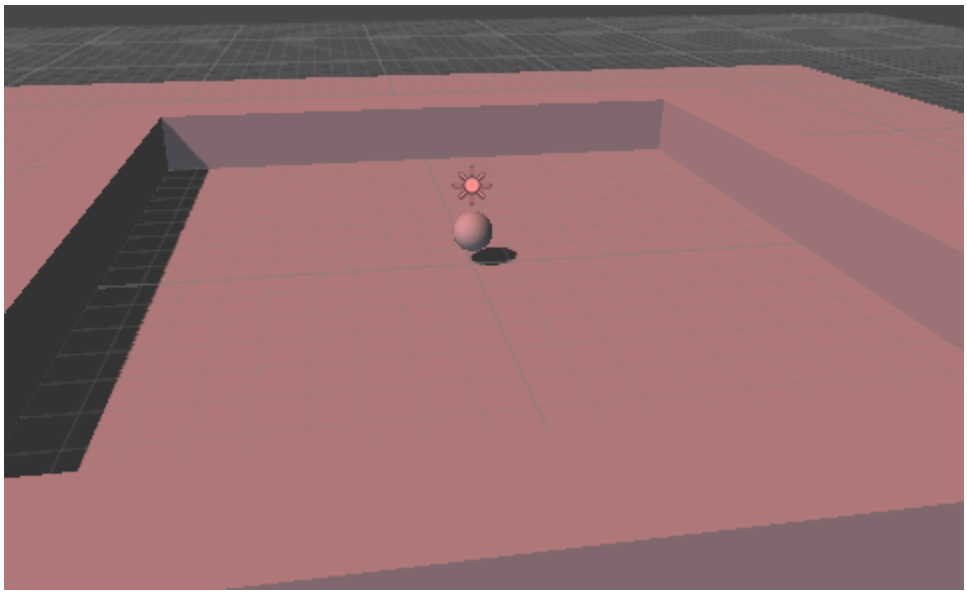
Create the Level

1. Create a new GameObject called Walls in our hierarchy. We will then create four cubes underneath it, and scale them to build walls.

2. The scale of the West Wall for example is this:

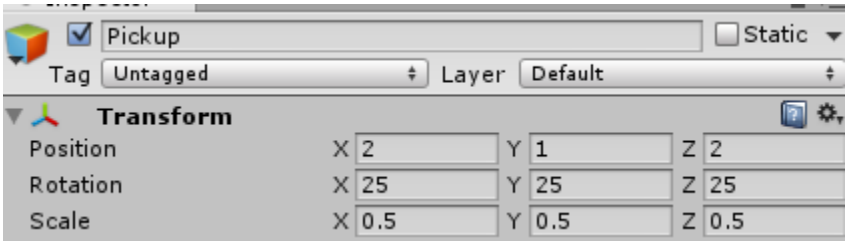


3. Your scene should roughly look like this



Create Items

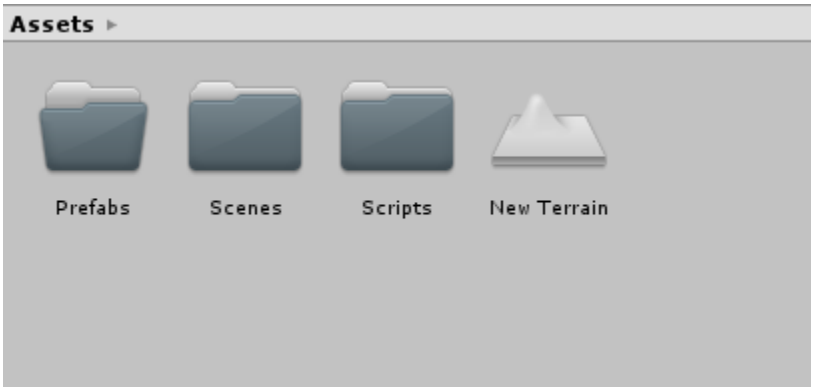
1. Create a new GameObject that is a cube called ‘pickup’
2. Scale it appropriately



3. Lets now create a script called ‘RotatingCube’ for the pickup object to make it spin. Lets now open it up in MonoDevelop
4. Here is the final script.

```
public class RotatingCube : MonoBehaviour {

    // Update is called once per frame
    void Update () {
        transform.Rotate (new Vector3 (15, 30, 45) * Time.deltaTime);
    }
}
```

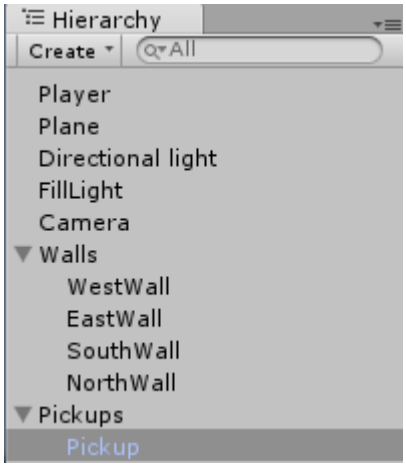
5.
6. Now that we have created an object, lets create some more of them. We are going to make these into ‘prefabs’. A ‘prefab’ is a reusable asset that acts as a blueprint for an object. We can use it in any scene in our project once it is created!
7. Lets create a new folder in our hierarchy called ‘prefabs’ to do this.



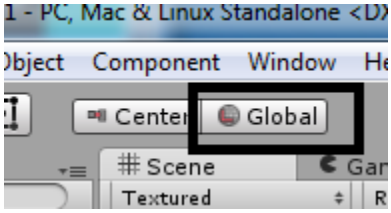
8. Then drag our object into the folder from our object hierarchy. And that’ it! You will notice our object is now highlighted in ‘blue’ in the hierarchy to indicate that it is a prefab.

9. 

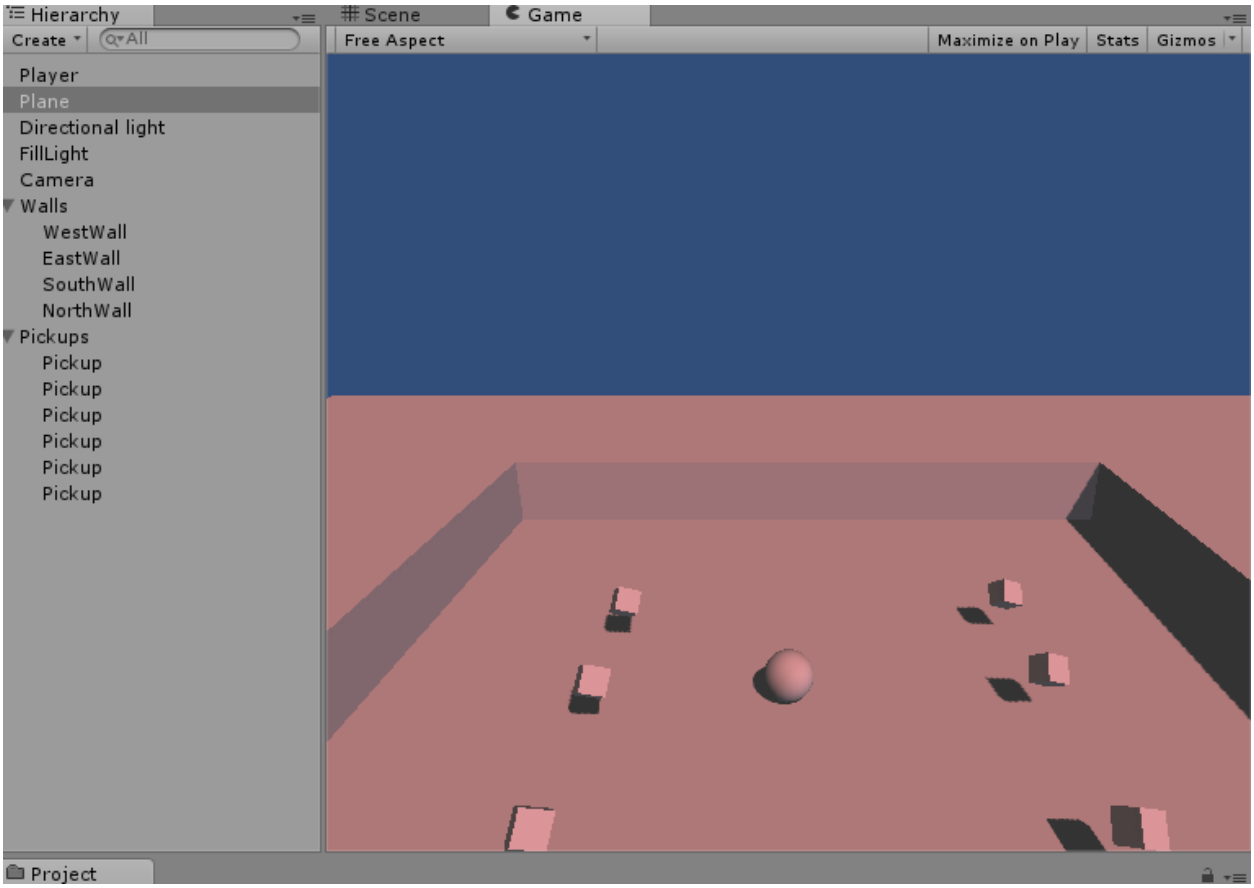
10. Now that it is a prefab, lets create an empty GameObject called ‘pickups’. Position this game object to the origin. Now move our pickup into the object, so that it is a child.



11. Now we want to duplicate this object, and move it around. To move it around in a global coordinate space, toggle the coordinate space to global.



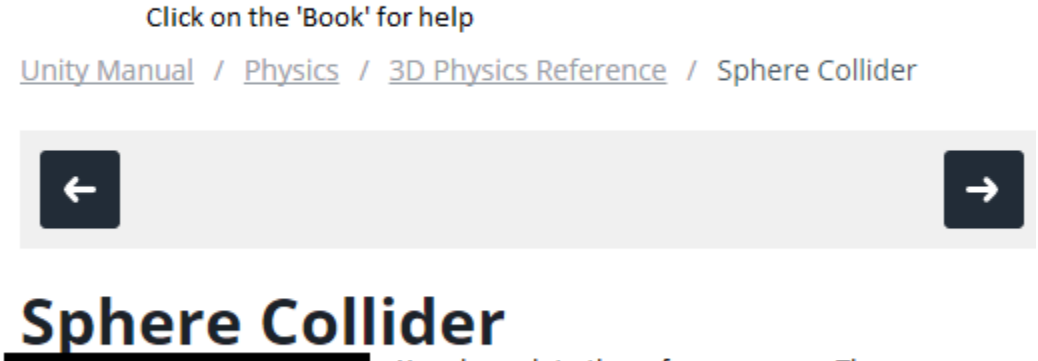
12. Now we manipulate the object, in regards to the world, instead of its own rotation, scale, and position.
13. Create several of them by duplicating them (note, since they are prefabs, they will already have scripts attached to them and be ready to go!)
14. You scene should look something like this:

15. 
Hierarchy
Scene
Game
Create
All
Free Aspect
Maximize on Play
Stats
Gizmos
Player
Plane
Directional light
FillLight
Camera
Walls
WestWall
EastWall
SouthWall
NorthWall
Pickups
Pickup
Pickup
Pickup
Pickup
Pickup
Pickup
Project

Collecting Objects

In order to win the game, the player will collide with the objects and pick them up. When they have picked up all the objects, the game is over. In order to do this, we will learn about colliding with the correct object, and picking it up.

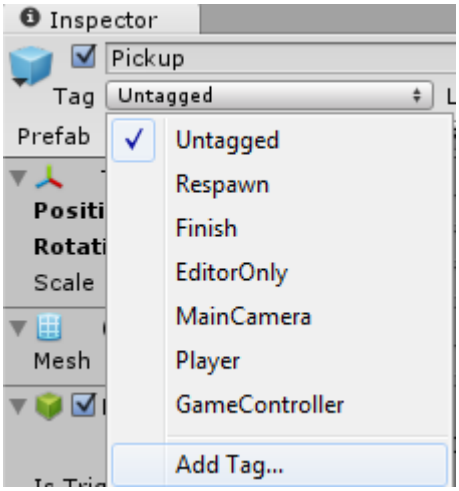
1. Open up our original player script that we wrote.
2. (Stepping back, this is how to learn about what methods are available)

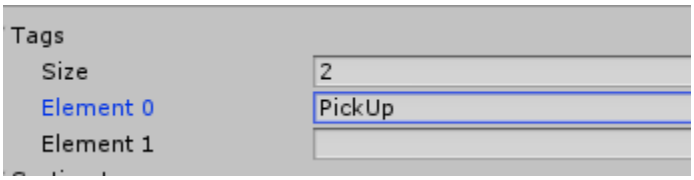


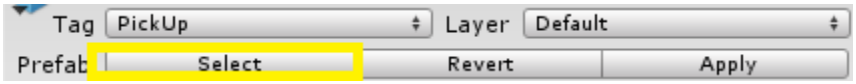
3.
4. In our script, add the following method.

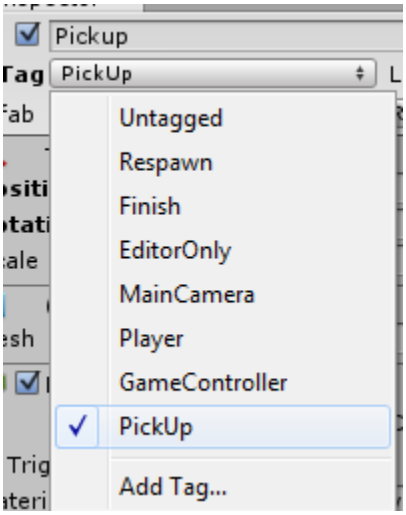
```
void OnTriggerEnter(Collider other){
    // Check if our object collides with a pickup object.
    if (other.gameObject.tag == "PickUp") {
        other.gameObject.SetActive(false);
    }
}
```

5.
6. Now we must ‘tag’ our object so that Unity knows about it. To do this, select one of our ‘pickup’objects and then find the ‘Tag’ item, and ‘Add Tag’

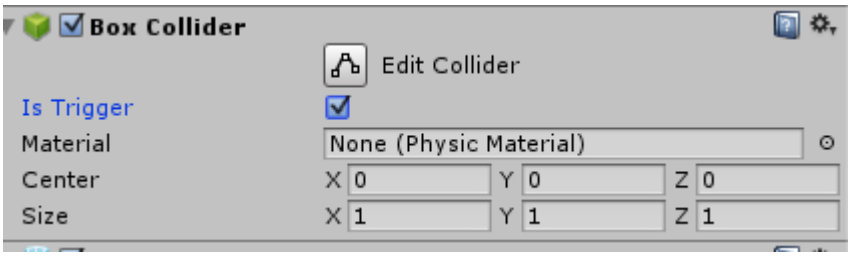
7. 

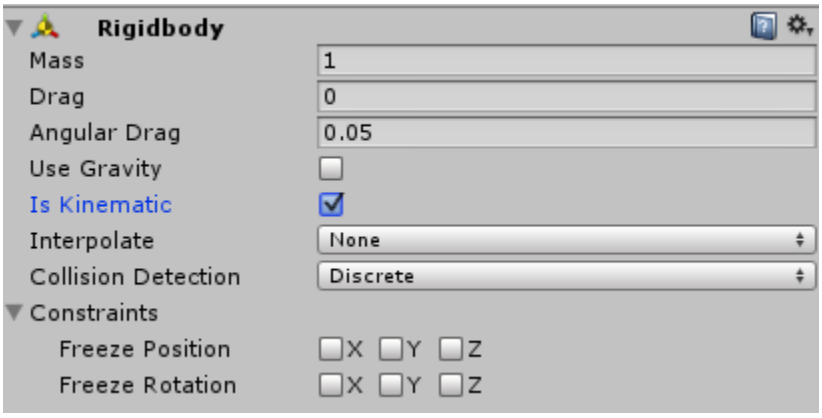
8. We then add the name of our tag (NOTE: This is case sensitive, and must exactly match what is in our code, because C# is a case sensitive language).
9. 

10. Then return to our 'pickup' object
11. We then hit 'select' for our prefab.
12. 

13. This ensures we make changes to every game object that is of this type.
14. Finally we tag our object.
15. 

16. The final step for our prefab, is to then make sure that it is a trigger. We want our objects geometry to act as a trigger, as opposed to physical collisions.

17. 

18. Side note: Small performance optimization to add a rigid body for our pickup object.
    a. Is Kinematic makes objects ignore physics simulations.
    b. Rigid body avoids excess computation for every frame on dynamic objects.
19. 

Winning the Game

1. When we have picked up all of the objects, we will have won the game. Lets make a private variable in our player script that gets updated every time we collide with an object.
2. Final code:

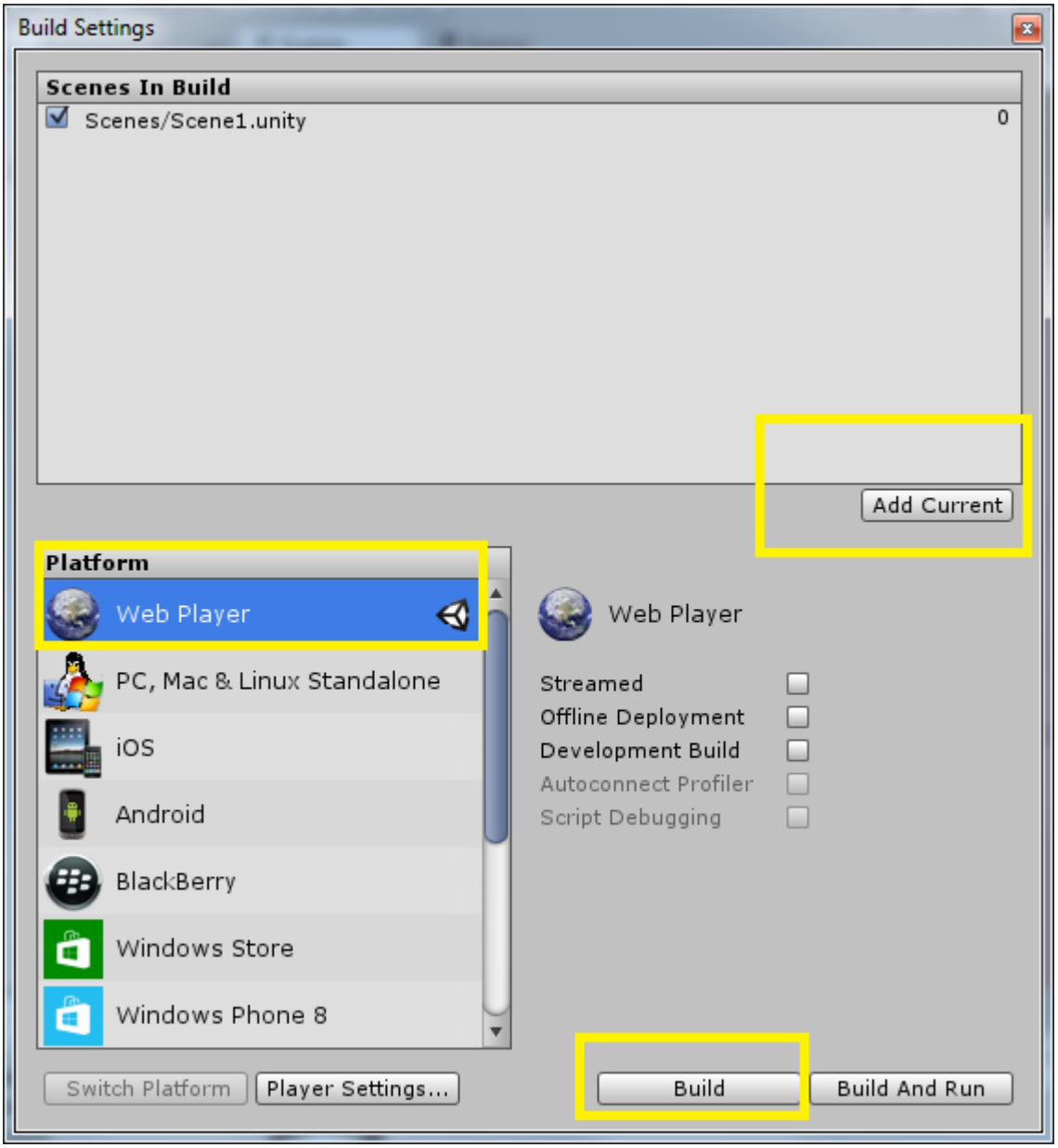
```
public float speed;
private int count;

void Start(){
    count = 0;
}

// Update is called once per frame
void Update () {
    // Record input from our player.
    float moveHorizontal = Input.GetAxis ("Horizontal");
    float moveVertical = Input.GetAxis ("Vertical");

    // Use the input we get to move the rigid body.
    Vector3 movement = new Vector3 (moveHorizontal, 0.0f, moveVertical);
    // Make a smooth movement by multiplying by Time.deltaTime.
    // Adjust our 'movement' vector by 'speed' to make game playable.
    rigidbody.AddForce (movement * speed * Time.deltaTime);
}

void OnTriggerEnter(Collider other){
    // Check if our object collides with a pickup object.
    if (other.gameObject.tag == "PickUp") {
        other.gameObject.SetActive(false);
        count += 1;
    }
}
```

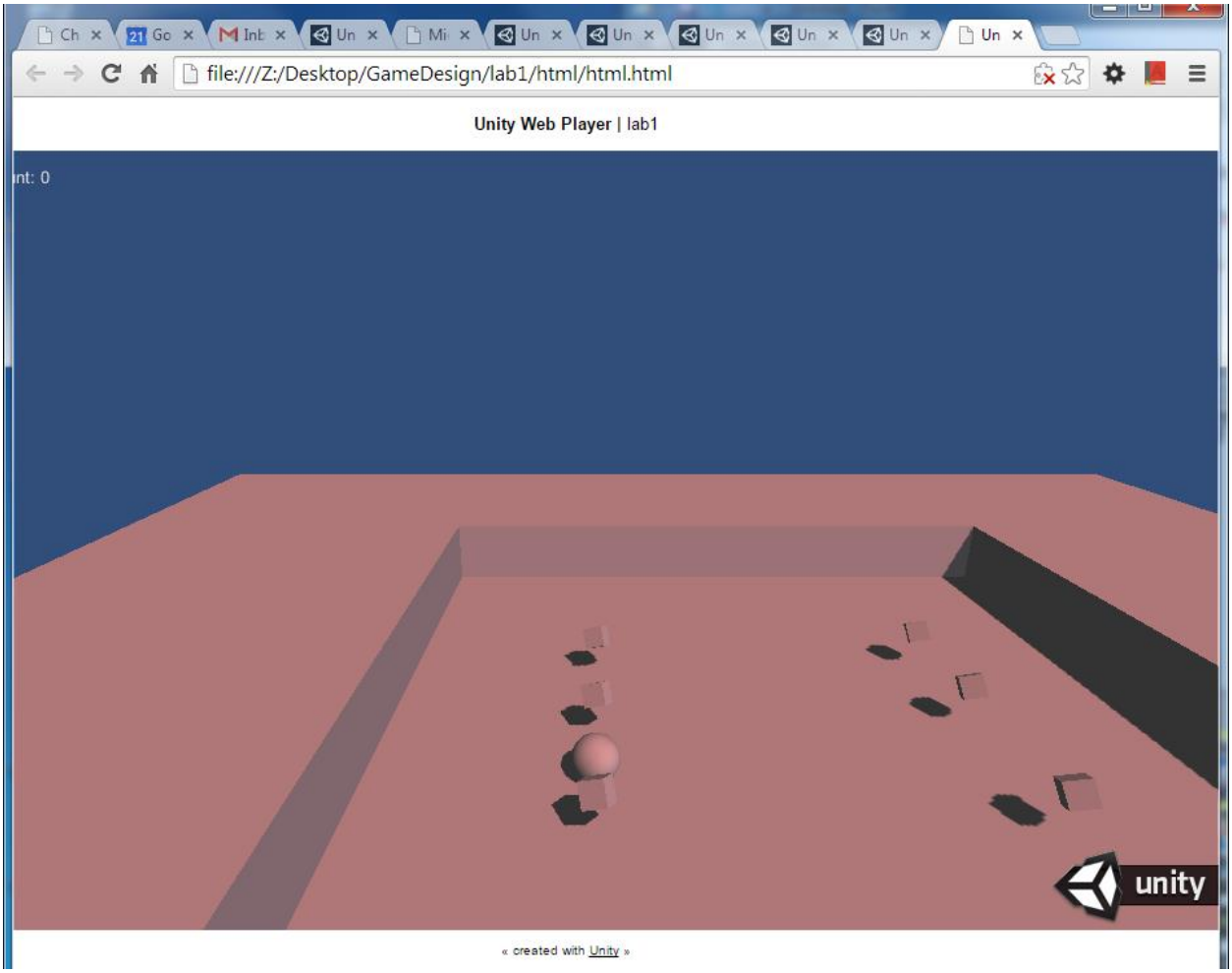
3. Lets now display the code for our user to see how well they are doing.

```
public float speed;
private int count;
public GUIText countText;

void Start(){
    count = 0;
    SetCountText ();
}

// Update is called once per frame
void Update () {
    // Record input from our player.
    float moveHorizontal = Input.GetAxis ("Horizontal");
    float moveVertical = Input.GetAxis ("Vertical");

    // Use the input we get to move the rigid body.
    Vector3 movement = new Vector3 (moveHorizontal, 0.0f, m
    // Make a smooth movement by multiplying by Time.deltaT
    // Adjust our 'movement' vector by 'speed' to make game
    rigidbody.AddForce (movement * speed * Time.deltaTime);
}

void OnTriggerEnter(Collider other){
    // Check if our object collides with a pickup object.
    if (other.gameObject.tag == "PickUp") {
        other.gameObject.SetActive(false);
        count += 1;
        SetCountText();
    }
}

void SetCountText(){
    countText.text = "Count: " + count.ToString ();
}

void OnGUI() {
    if (count == 6) {
        GUI.Label ((new Rect (10, 10, 100, 20)), "WAY TO GO! YOU WIN!")
    } else {
        GUI.Label ((new Rect (10, 10, 100, 20)), countText);
    }

}
```

4.
5. SAVE all of your scripts and your Unity3D scene.
6. Play your game!!

For an Extra Cool Factor (Show your friends, export to the web!)





Do go home and install the android SDK or if you're on a Mac, install the iOS XCode Tools and deploy!