

Introduction to Unity 3D

<https://code.tutsplus.com/tutorials/introduction-to-unity3d--mobile-10752>

Tutorial

This tutorial explains in text the most basic game form Unity : Roll-a-ball

Create project - Step 1

- Create Scene(save it in the asset folder)
- Create Gameboard /Playfield(GameObject - 3D obj - plane/hierarchy - create- plane)
- Inspector - Transform - Reset (places the coordinates at 0-0-0)
- Grid option - #scene - gizmos - show/hide grid
- Adding Colour : Project - add materials - from inspector - pick albedo - drag it to player or background
- Change direction of light - hierarchy - directional light - change transforms

Moving the ball - Step 2

- Add a sphere
- Make it rigid - select player - component - physics - rigid body
- Add script - either project - create script and add then or better : add component in insector, add script, name it and createandadd

Open script editor - Visual Basic - .cs files - it is a script editing system. -

Update - for rendering the code

Fixed Update - called before any physics calculations

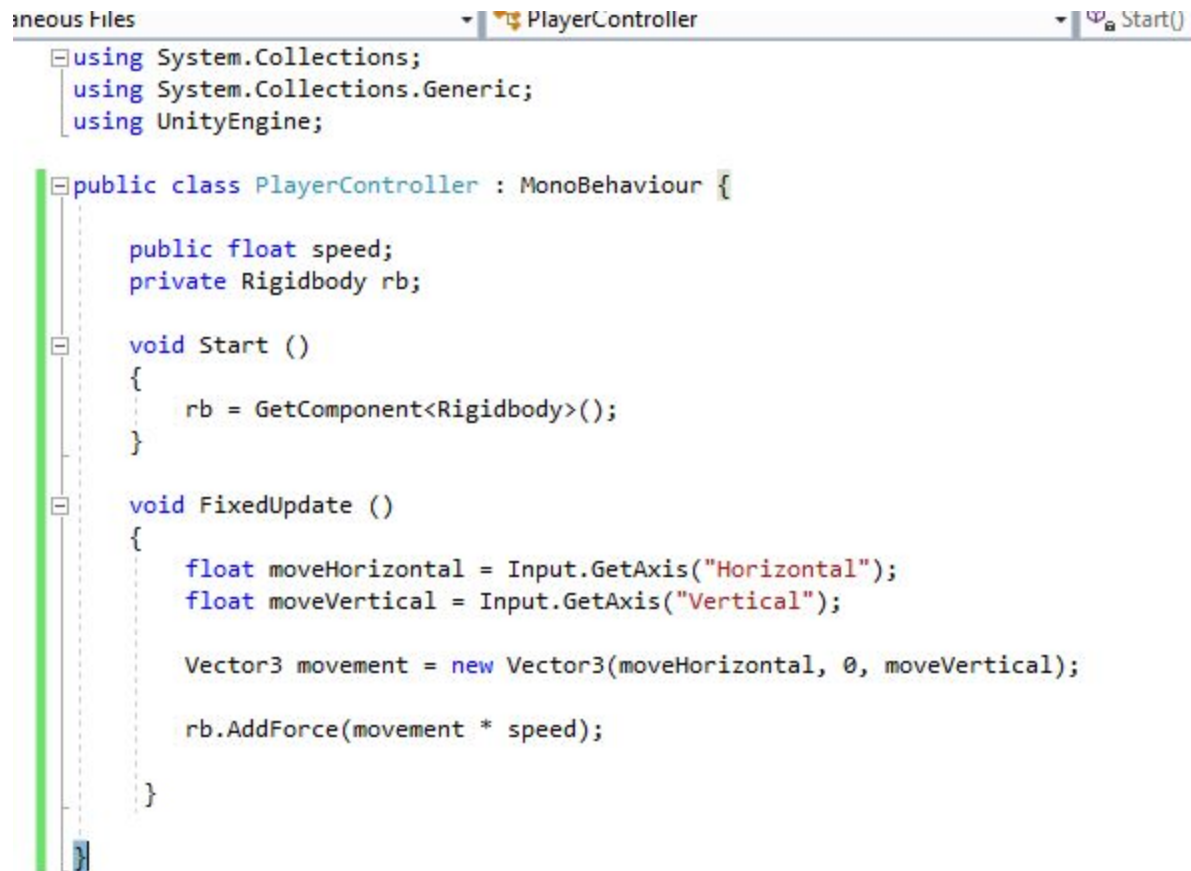
Input - All kinds of user input - accelerometer on mobile/Keyboards

Get.Axis

RigidBody

AddForce

Add Speed

The image shows a screenshot of a code editor window. At the top, there is a tab labeled 'PlayerController' with a small icon to its left and a 'Start()' button to its right. Below the tab, the code is written in C#. It starts with three 'using' statements: 'using System.Collections;', 'using System.Collections.Generic;', and 'using UnityEngine;'. Then, a class definition 'public class PlayerController : MonoBehaviour {' is shown. Inside the class, there is a 'public float speed;' and a 'private Rigidbody rb;'. Two methods are defined: 'void Start ()' which contains 'rb = GetComponent<Rigidbody>();', and 'void FixedUpdate ()' which contains logic for movement based on input axes 'Horizontal' and 'Vertical', creating a 'Vector3 movement' and adding a force to the 'rb' scaled by 'speed'. The code is color-coded: keywords in blue, comments in green, strings in red, and literals in black. A vertical green bar is on the left side of the code area, and a blue cursor is at the bottom left.

Moving the camera - Step 3

Linking the camera to the player's transform position

```

1  using System.Collections;
2  using UnityEngine;
3
4  public class CameraController : MonoBehaviour {
5
6      public GameObject player;
7
8      private Vector3 offset;
9
10     // Use this for initialization
11     void Start () {
12         offset = transform.position - player.transform.position;
13     }
14
15     // Update is called once per frame
16     void LateUpdate () {
17         transform.position = player.transform.position + offset;
18     }
19 }
20

```

Making a play area - Step 4

Add walls using add GameObject - cube

Specify dimensions - Transform moves it to exact place in play area

Make multiple of them and place them under hierarchy under a single heading



Reset their Position as soon as you build them for no errors

Duplicating objects - select them and choose duplicate under edit

After duplicating, use Position and scale to move objects around.

Make boundaries so object can't go out of the ground



Creating collectable objects - Step 5

Create a cube and name it as pickup

To rotate the collectible objects to make them more attractive

Writing script to transform the axes continuously

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class Rotator : MonoBehaviour {
6
7
8      // Update is called once per frame
9      void Update () {
10         transform.Rotate(new Vector3 (15, 30, 45) * Time.deltaTime);
11     }
12 }
13
```

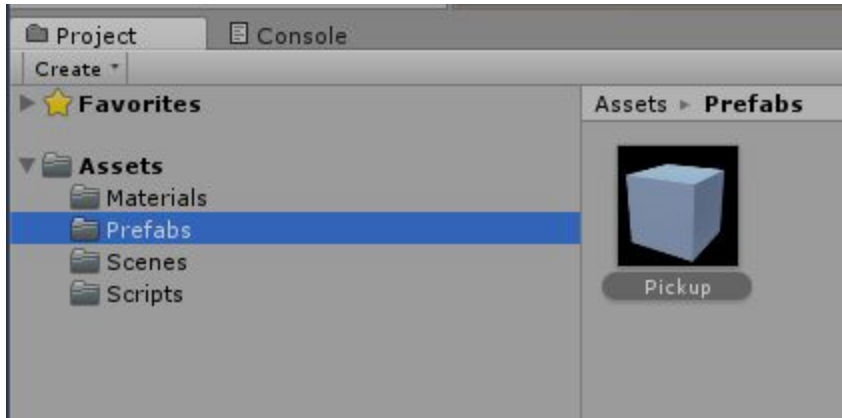
Make prefabs : how to

We need to make our pickup object into a prefab

'A prefab is an asset that contains a template or blueprint of a gameobject or game object family'

Create a folder to hold our prefabs

Make a prefab folder in project and keep pickup object (drag them) to prefab folder

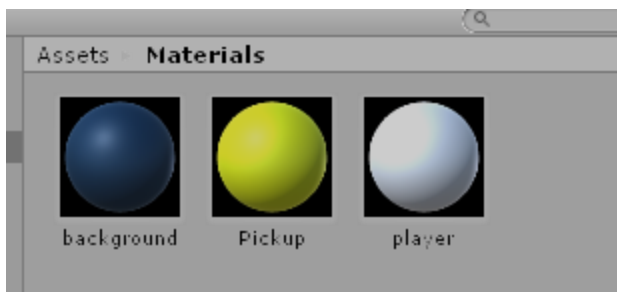


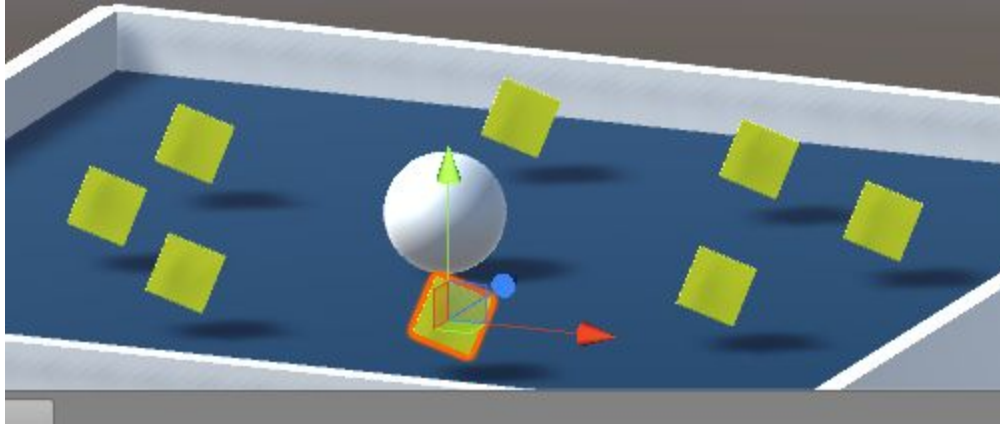
Create a game object 'Pickups' to hold our pickup's
Ensure pickup's position is reset to origin



Put all pick up objects in the gameobject 'Pickups'

Make pickups more attractive by adding colour to it. For this add material, change it colour, and apply it to prefabs by drag and drop





Counting points - Step 6

This means detecting collisions of player with pickups

Editing is done in Player controller script

[For finding help related to any command, you can do it by two ways :

In the script: select the term and press CTRL+' to open help directory, alternatively

In Unity: selected the small book with a question mark in the inspector to open help directory]

Use the command : Collider.OnTriggerEnter(Collider)

And copy the code:

```
'void OnTriggerEnter(Collider other) {  
    Destroy(other.gameObject);
```

This code gives us the ability to detect contact between player and object without actually colliding

Useful commands : open up 'GameObject'

.Tag ; tag allows us to identify a game object by tag value

.CompareTag : compare the tag of any game object with a string value

.SetActive : used to deactivate pickup objects

```

void OnTriggerEnter(Collider other)
{
    if (other.gameObject.CompareTag("Pick Up"))
    {
        other.gameObject.SetActive (false);
    }
}

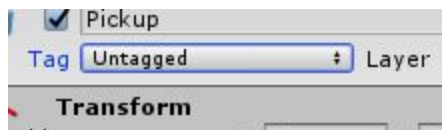
Destroy(other.gameObject);
if (other.gameObject.CompareTag("Player"))
    gameObject.SetActive(false);

```

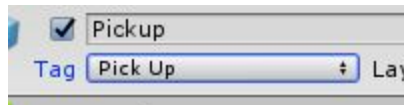
Use the highlighted part which is copied from the scripts of the above three useful commands to build our code(non-highlighted part)

Now delete the highlighted part(it was just to build our code)
Save the code and return to unity

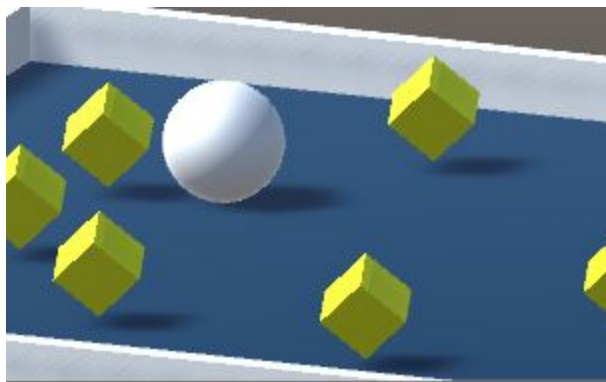
Under prefabs under pickup, in the inspector, we can see 'Untagged'
Select - add tag- name it same as in the script - 'Pick Up' (this is case sensitive)
Check the new Tag instead of Untagged



Changed to



We will now see that the player bounces off dynamic Game Objects (Pickup objects) but is not able to overlap



For this we need to do some modifications :

Select : prefab - Pickup - Box Collider - is Trigger - Check
(All pick up cubes should be checked automatically, if not do them manually)

After this, As the player enters the trigger, we pickup the objects



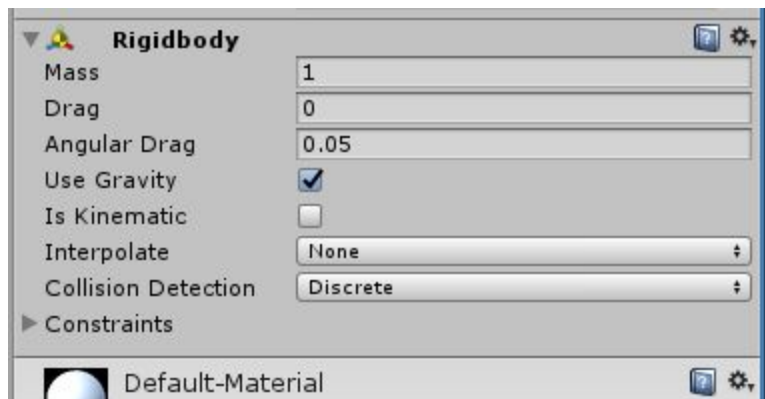
Making it Fast:
everytime we move, rotate, or hit the static collider,
Unity will recalculate all the static collider again,
And update the static collider cache
This takes resources

We can move, rotate the dynamic collider, and unity doesn't calculate any volumes

All rigid bodies make the program dynamic
Currently our pickup game collider has a box collider but no rigid body, so unity is calculating cache after every frame which makes it slow
Solution: Add a rigid body to pick up cubes. This makes them dynamic and doesn't have to recalculate cache again and again

Add this in the inspectors in

Prefabs - pick up - add component - physics - rigid body (enable IS KINEMATIC)



Display Score and Text - Step 7

Editing is done in player controller script

Add a private variable to hold the count

Private variables: available only in the script, not available in the inspector

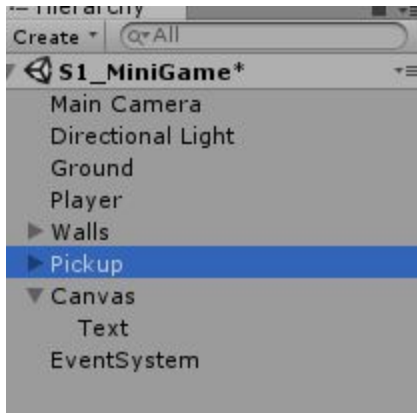
Adding a count

Add a private counter(integer) which will store the value of objects picked up

Add count 0 in start function to set initial value

Add count = count + 1 after the game object is set to false

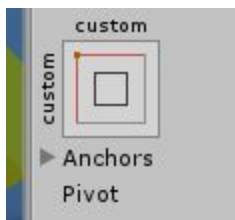
Displaying the count(using Unity's UI)



Add from hierarchy = UI - text(It creates a text and canvas and event system)

In the inspector for count text, you can change font, size, shape, colour and add text to it
'It uses a rect transform'

You can anchor the text to the top left corner by selecting this button on the inspector and then
Press ctrl and shift and select top left



This moves the text to top left

Now open player control script for editing

We need to add namespace UI to script

Add using UnityEngine.UI; to the top of the script

Add

```

void Start ()
{
    rb = GetComponent<Rigidbody>();
    count = 0;
    countText.text = "Count: " + count.ToString();
}

```

3rd line in start menu

Add the same line of code in the later part

```

12
13 void Start ()
14 {
15     rb = GetComponent<Rigidbody>();
16     count = 0;
17     countText.text = "Count: " + count.ToString ();
18
19
20 void FixedUpdate ()
21 {
22     float moveHorizontal = Input.GetAxis ("Horizontal");
23     float moveVertical = Input.GetAxis ("Vertical");
24
25     Vector3 movement = new Vector3 (moveHorizontal, 0.0f, moveVertical);
26
27     rb.AddForce (movement * speed);
28 }
29
30 void OnTriggerEnter(Collider other)
31 {
32     if (other.gameObject.CompareTag("Pick Up"))
33     {
34         other.gameObject.SetActive (false);
35         count = count + 1;
36         countText.text = "Count: " + count.ToString ();
37     }
38 }
39

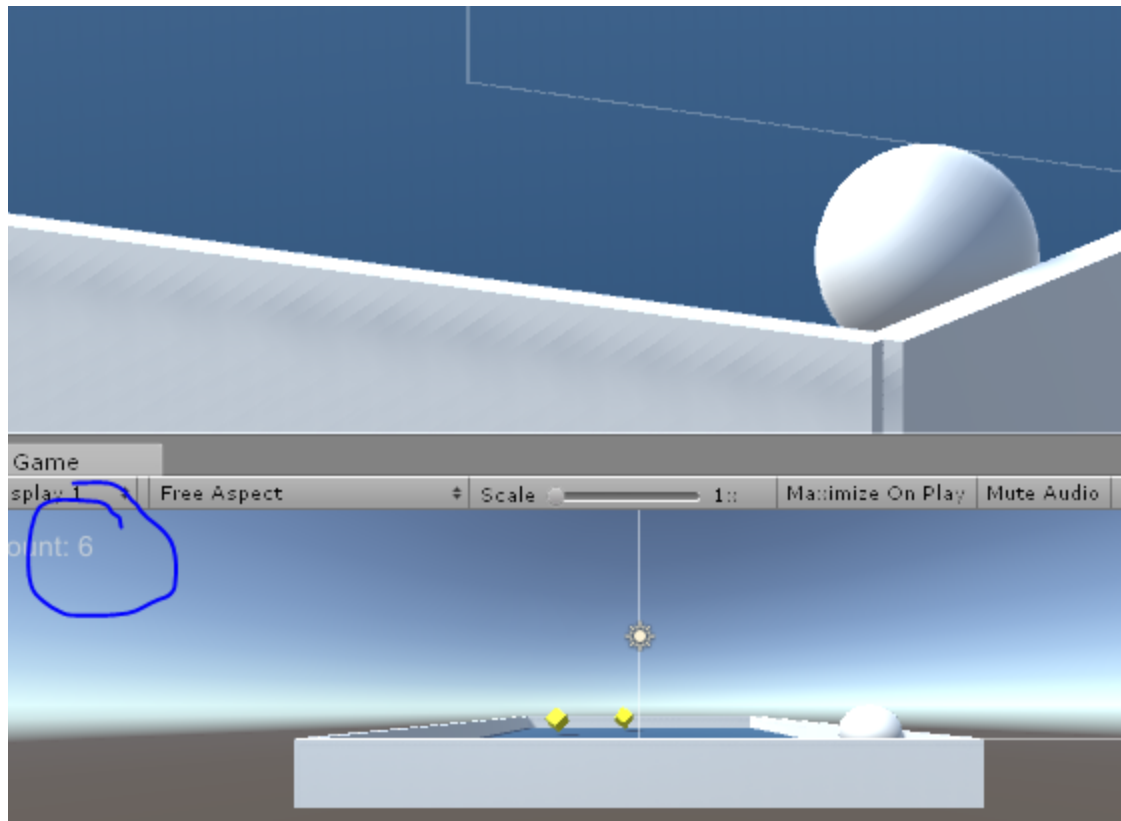
```

We can also call a function instead of adding same lines at both the places

Now we can see in the player controller script has a new text line in the inspector
 Drag 'Count Text' from hierarchy to the Text in player script

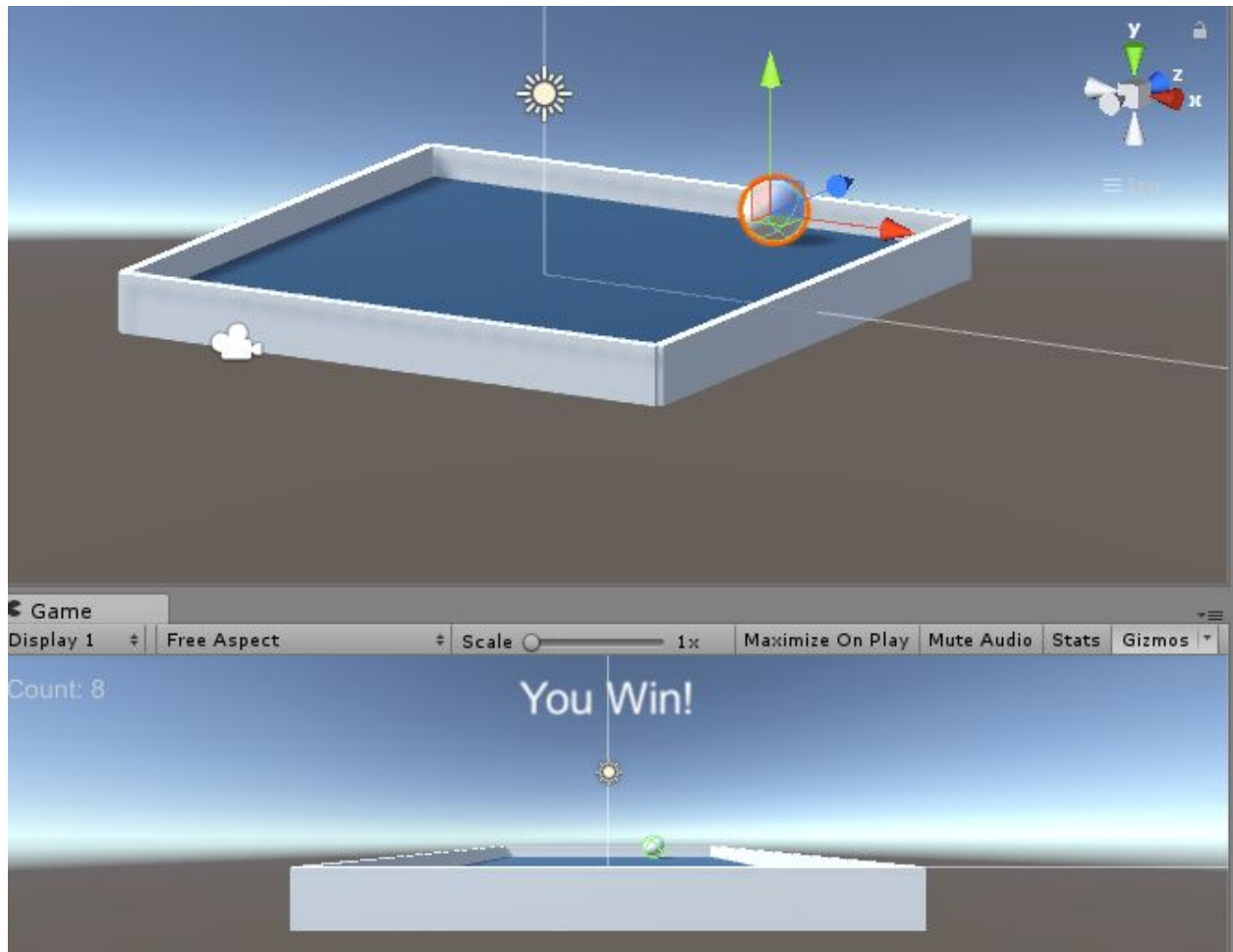


Save , enter play mode and see this happens



Similar procedure for adding a winning text, add from UI from hierarchy
Change the Y value of the rect transform to make it on a comfortable position

Make changes to player script to accommodate winning text
And then save changes and try to run



Step 8 - Building our game

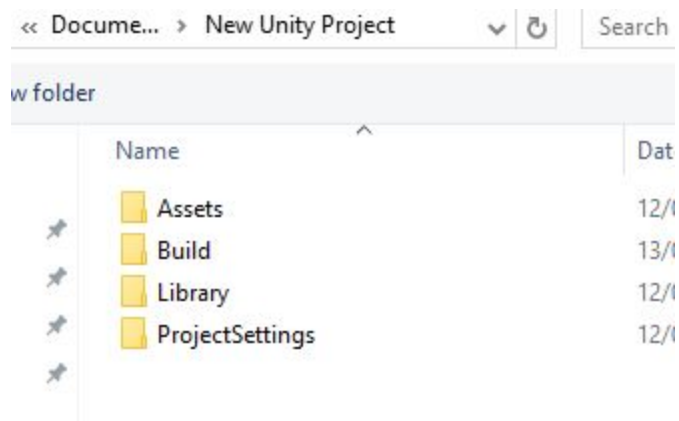
Open build settings from file menu

Choose platform

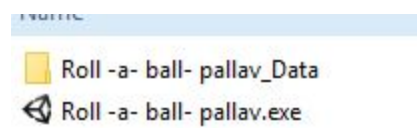
Add the scenes you want in the game

Click Build. It will be an exe file

Make sure to choose the same folder which contains the project files.



>Make a new folder named build and save the exe file inside it.



It makes an exe file which is ready to run
I win!