Pong in Unity a basic Intro

This tutorial recreates the classic game Pong, for those unfamiliar with the game, shame on you what have you been doing, living under a rock?! Go google it. Go on.

For those that now know the game, this tutorial assumes a basic working knowledge of Unity and the editor and some programming, although it will explain as it goes.

Create a new 2D project in Unity, this sets some basic things up for us but we need to make some simple changes to the Camera.

Select the Main Camera and review the settings, I'll point out the important ones, first, note that the Z position settings are -10, that means that the camera is on that "line", anything behind it won't be shown by that camera, anything above will be. The Background is just the colour, pick a background colour and set it. Projection needs (and should) be set to Orthographic, this is a 2D camera, a way of rendering 3D objects in 2D. See Figure 1 for an example, depending on the angle you view the stairs from it could be any one of the surrounding pictures. Size is pretty much how much the camera can see. Clipping planes is how far the camera can see. There are two values: Near and Far. Near is where it starts rendering, and Far is where it stops rendering.

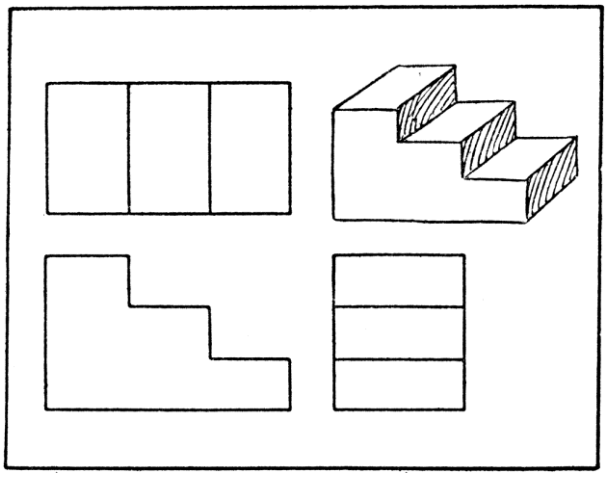
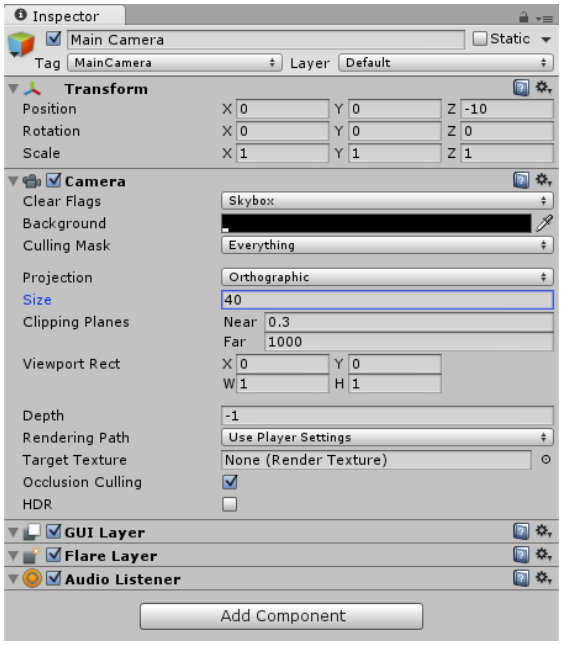


Figure 1 Source: Thomas E. French and Carl L. Svensen Mechanical Drawing For High Schools (New York: McGraw-Hill Book Company, Inc. , 1919)



We will use the following sprites from opengameart as our walls to ensure the ball doesn't escape, you can right click them from within here and save them to your assets folder but the easiest way is just to download them from the asset bundle on moodle.

When you import it make sure to set the Texture type to sprite, and drop the pixels per unit to 1,

*A **Pixels Per Unit** value of **1** means that **1 x 1** pixels will fit into one unit in the game world. We will use this value for all our textures, because the ball will be **1 x 1** pixels later on, which should be exactly one unit in the game world.*

Pong in Unity a basic Intro

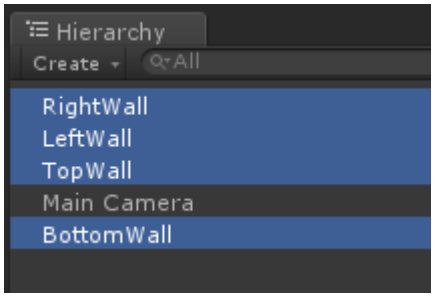
Position the walls so that they roughly match the image below, you'll need to rotate and adjust the vertical ones. You'll need to scale it suitably, rotating Z by 90 and Scaling X by 0.7 should produce a nice result. Make sure the camera is roughly in the middle, use the camera view to ensure you're positioned correctly.

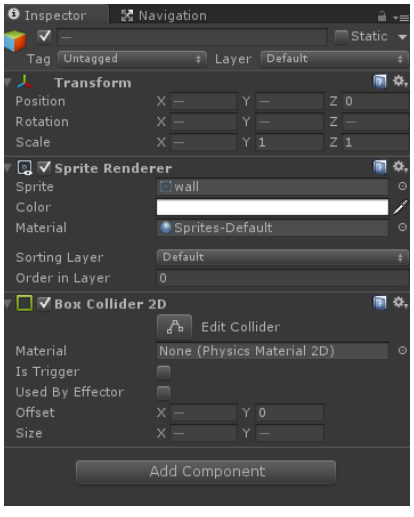
Currently the walls are there but not there, anything you fire through would just pass right through. We want them to be real walls so the racket and ball will collide with them. So, we add something called **Colliders.** This tells the built in physics engine that these are supposed to be solid and lets us specify ways they behave.

Right now we can see the walls in the game, but they aren't real walls yet. They are just images in the game world, a purely visual effect.

We want the walls to be real walls so that the Rackets and the Ball will collide with them instead of just going right through them.

So, select all the walls in the hierarchy, then in the Inspector go to Add component -> Box Collider 2D. Make sure to select the 2D version, adding 3D means we have a whole other dimension to calculate physics in, adding a performance overhead.





If you zoom into the scene you should also see a pale green line surrounding each wall, this shows the collider surrounding that wall.

You'll now want to copy and paste in the dotted line that shows each player's "half" of the board.

Import it as before, set the Pixels per Unit to 1 and drag it so it's roughly in the middle. Don't worry about adding a collider, we don't want the ball to collide with the net.

This time, import, rotate and position the two paddles (red and blue) as before set the PPU to 1. Add a 2D Box Collider to the paddles, later we'll want to move the paddles up and down but have it stop

when they collide with the walls. Note: as a rule of thumb, everything physical that moves through the game world will need a Rigidbody.
To add a Rigidbody to our Rackets we just select both of them in the Hierarchy again, then take a look in the Inspector and press Add Component->Physics 2D->Rigidbody 2D. We then modify the Rigidbody 2D to set Gravity to 0 (because there is no gravity in a pong game) and make it always have a Fixed Angle (the rackets should never rotate):

Racket Movement

Let's make it so players can move their paddles, we need to create a new C# Script, call it PaddleScript (capitals count).

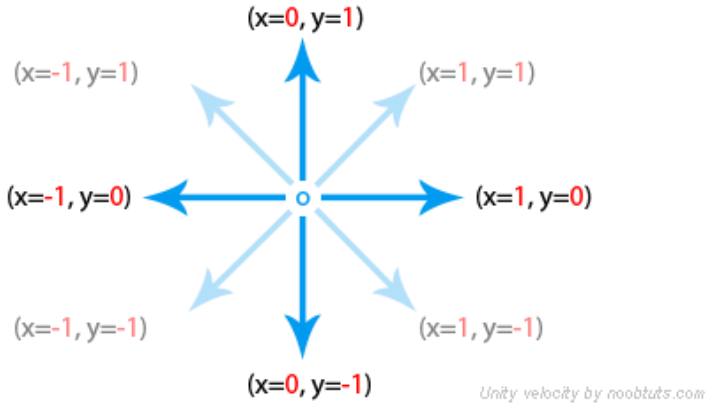
If you open it you won't see a lot,

The Start function is automatically called by Unity when starting the game. The Update function is automatically called over and over again, roughly 60 times per second (each time a frame renders).

But there is another Update function, it's called FixedUpdate. This one is also called over and over again, but in a fixed time interval. Unity's Physics are calculated in the exact same time interval, so it's usually a good idea to use FixedUpdate when doing Physics stuff (we want to move Paddle that have Colliders and RigidBodys, hence Physics stuff).

So let's remove the Start and Update functions and create a FixedUpdate function instead, the rackets have a Rigidbody component and we will use the Rigidbody's velocity property for movement. The velocity is always the movement direction multiplied by the speed.

The direction will be a Vector2 with a x component (horizontal direction) and a y component (vertical direction). The following image from noobtuts shows a few Vector2 examples:



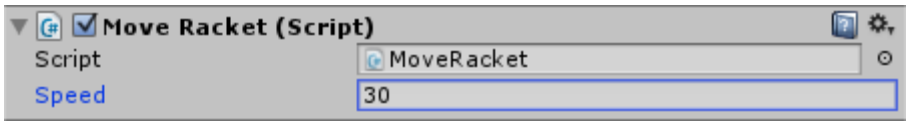
The rackets should only move upwards and downwards, which means that the x component will always be 0 and the y component will be 1 for upwards, -1 for downwards or 0 for not moving.

The y value depends on the user input to get this we will use Unity's GetAxisRaw function, we use GetAxisRaw to check the vertical input axis. This will return 1 when pressing either the W key, the UpArrow key, or when pointing a gamepad's stick upwards by default and -1 when it moves the other way.

Now we can use GetComponent to access the racket's Rigidbody component and then set its velocity:

```
1 using UnityEngine;
2 using System.Collections;
3
4 public class PaddleScript : MonoBehaviour {
5
6     void FixedUpate()
7     {
8         float v = Input.GetAxisRaw ("Vertical");
9         GetComponent<Rigidbody2D>().velocity = new Vector2(0, v);
10    }
11 }
12
```

We will also add a speed variable to our Script, so that we can control the racket's movement speed. We make the speed variable public so that we can always modify it in the Inspector without changing the Script. Now we can modify our Script to make use of the speed variable:




```
1 using UnityEngine;
2 using System.Collections;
3
4 public class PaddleScript : MonoBehaviour {
5
6     public float speed = 30;
7
8     void FixedUpate()
9     {
10        float v = Input.GetAxisRaw ("Vertical");
11        GetComponent<Rigidbody2D>().velocity = new Vector2(0, v)*speed;
12    }
13 }
```

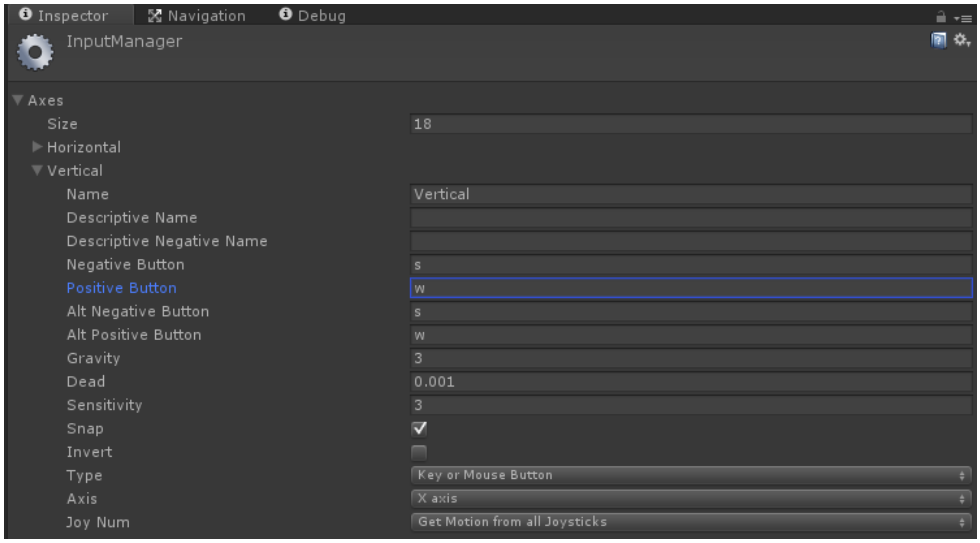
Note: we set the velocity to the direction multiplied by the speed, which is exactly the velocity's definition.

If we save the Script and press Play then we can now move the rackets, but they move together, there is just one problem, we can't move the rackets separately yet.

Right now, both our Scripts check the "Vertical" Input axis for the movement calculations. We could change this to use one specific keyset (e.g. W for player1, P for Player2) but instead let's create a new Axis so that we can change the Input Axis in the Inspector:

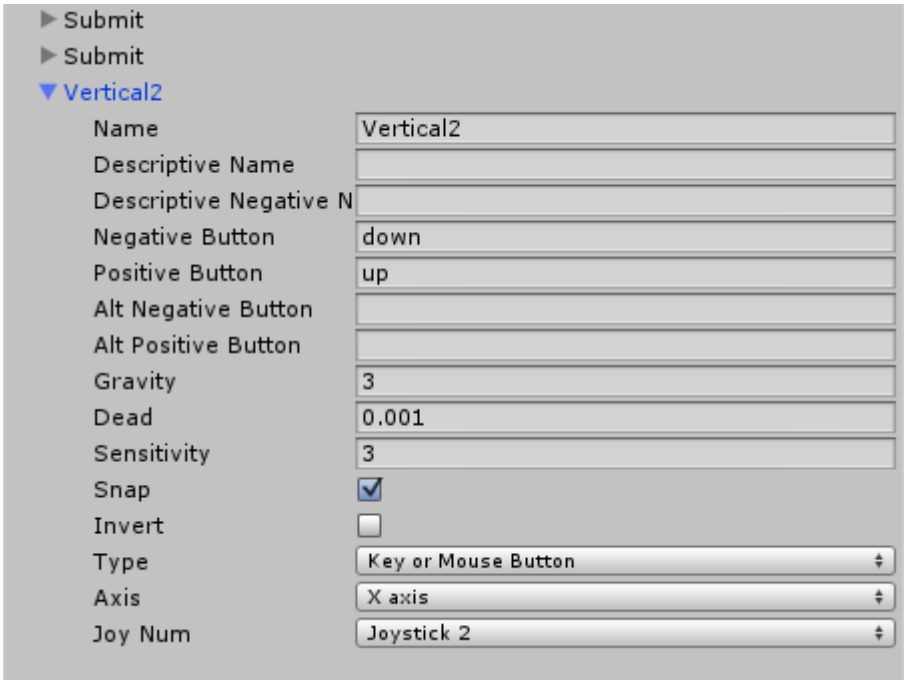
Pong in Unity a basic Intro

Select Edit->Project Settings->Input from the top menu, here we can modify the current Vertical axis so that it only uses the W and S keys. We will also make it use only Joystick 1:



Now we will increase the Size by one in order to add a new axis:

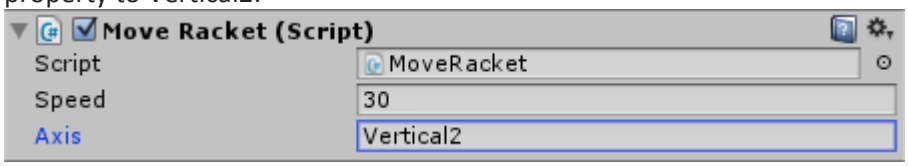
We will name it Vertical2 and modify it accordingly:



Finally, update the script to use a new “axis variable”

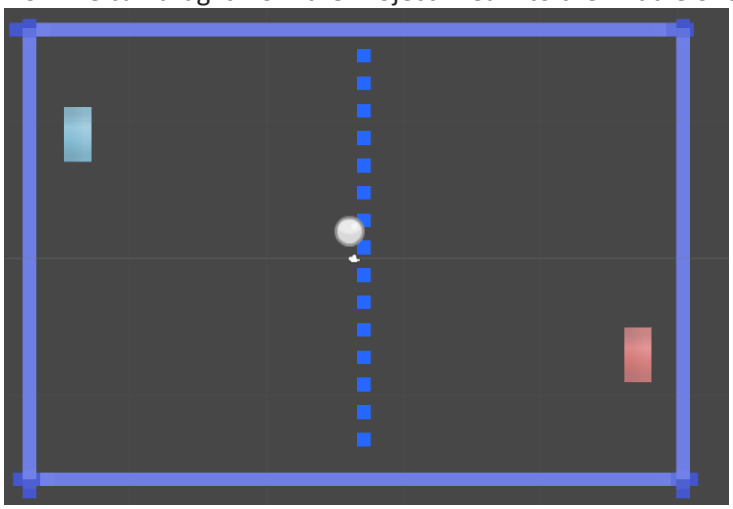
```
1 using UnityEngine;
2 using System.Collections;
3
4 public class PaddleScript : MonoBehaviour {
5
6     public float speed = 30;
7     public float v;
8     public string axis="Vertical";
9
10    void Start()
11    {
12        print ("start");
13    }
14
15    void FixedUpdate()
16    {
17        print ("one");
18        //float v = Input.GetAxisRaw ("Vertical");
19        v = Input.GetAxisRaw (axis);
20        GetComponent<Rigidbody2D>().velocity = new Vector2(0, v)*speed;
21    }
22
23 }
```

Afterwards we will select the paddleRed GameObject and change the MoveRacket Script's Axis property to Vertical2:



If we press Play then we can now move the rackets separately. Yay!

Okay so now we need a Ball, import it and configure it using the same Import Settings but a PPU of 10.
Now we can drag it from the Project Area into the middle of the Scene:

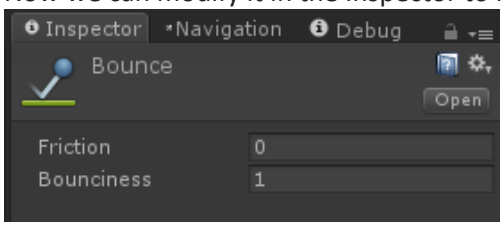


Our Ball should make use of Unity's Physics again, so let's select **Add Component**->**Physics 2D**->**Box Collider 2D** to add a Collider:

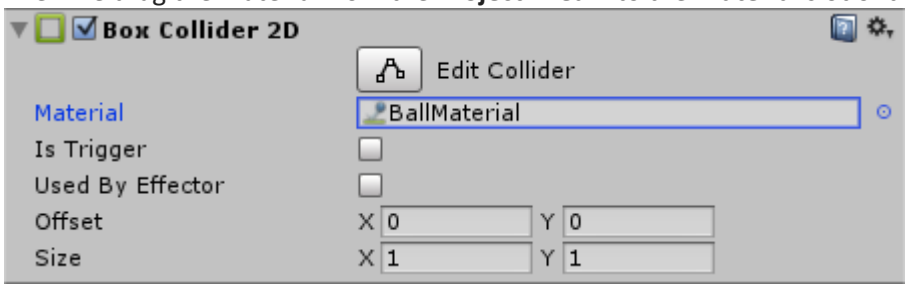
Pong in Unity a basic Intro

Our ball should bounce at the angle it hits the wall at, so it hits at 45, bounces off at -45, etc. this sounds like some complicated math that could be done with Scripting. But since we are lazy, we will just let Unity take care of the bouncing off thing by assigning a **Physics Material** to the Collider that makes it bounce off things all the time.

At first we right click in our Project Area and selected **Create**->**Physics2D Material** which we will name **Bounce**:
Now we can modify it in the Inspector to make it bounce off:



Then we drag the material from the **Project Area** into the **Material** slot of the Ball's Collider:



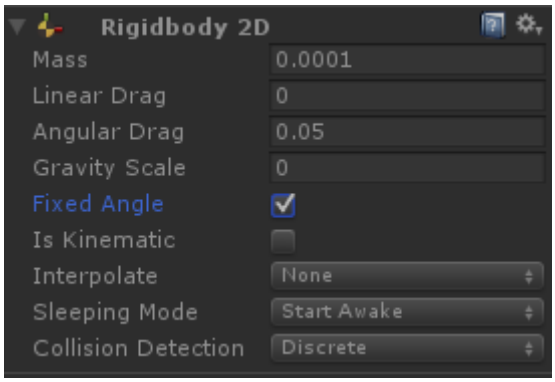
And that's it. Now the ball will bounce off in case it collides with things in the game, except it doesn't move. Well that's a good game isn't it.

In order to make our ball move through the game world, we will add a Rigidbody2D to it again by selecting Add Component->Physics 2D->Rigidbody 2D.
We will modify the Rigidbody component in several ways:

We don't want it to use Gravity

We want it to have a very small Mass so it doesn't push away the Rackets when colliding

We don't want it to rotate



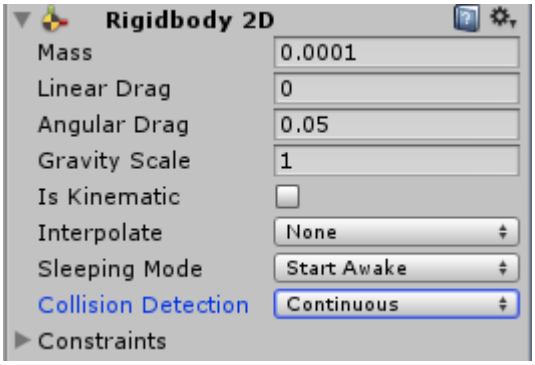
Okay so there is one more thing to do before we see some cool ball movement. We will select **Add Component**->**New Script**, name it **Ball** and select **CSharp** for the language.

Open it and get ready to again change the contents, we will use the **Start** function to give the ball some initial velocity. Yet again we will use a **direction** multiplied by a **speed**:

```
using UnityEngine;
using System.Collections;

public class Ball : MonoBehaviour {
    public float speed = 30;

    void Start() {
        // Initial Velocity
        GetComponent<Rigidbody2D>().velocity = Vector2.right * speed;
    }
}
```

Now if we press play, we can see the Ball bounce off the walls and paddles; there's a few more things to do but you can give the game a go. One issue you might find is that the ball can sometimes when going very fast bounce through the walls. Obviously this isn't particularly effective.

This is because the collision detection on the ball is set to discrete, this means that, if the object moves very fast, sometimes it might "miss" the collision. To solve this, change it from discrete, to continuous, meaning it will always be checking. On a normal game this might mean a massive performance hit, but in a simple one like pong, it's fine.



Secondly, if you notice, the ball bounces flat on, there is no change of angle like a proper game of pong or air hockey. Let's change our BallScript to make it change when it collides,

So we need work out the velocity based on where the ball hits, this is harder mathematically easy(ish) via code. So, we need two values for a Vector2, x and y, the **x** value is easy, it's **-1** in case it bounces off the right racket and it's **1** if it bounces off the left racket. What we really need to think about is the **y** value, which will be somewhere between **-1** and **1**. All we really need to calculate is this:

|| 1 <- at the top of the racket
||
|| 0 <- at the middle of the racket
||
|| -1 <- at the bottom of the racket

Or in other words: we just have to divide the ball's **y** coordinate by the racket's **height**. Here is our function:

```
float hitFactor(Vector2 ballPos, Vector2 racketPos,
                float racketHeight) {
    // ascii art:
    // ||  1 <- at the top of the racket
    // ||
    // ||  0 <- at the middle of the racket
    // ||
    // || -1 <- at the bottom of the racket
    return (ballPos.y - racketPos.y) / racketHeight;
}
```

And our final OnCollisionEnter2D function;

```
25      void OnCollisionEnter2D(Collision2D col) {
26          // Note: 'col' holds the collision information
27
28          // Hit the left Paddle?
29          if (col.gameObject.name == "paddleBlu") {
30              // Calculate hit Factor
31              float y = hitFactor(transform.position,
32                                  col.transform.position,
33                                  col.collider.bounds.size.y);
34
35              // Calculate direction, make length=1 via .normalized
36              Vector2 dir = new Vector2(1, y).normalized;
37
38              // Set Velocity with dir * speed
39              GetComponent<Rigidbody2D>().velocity = dir * speed;
40          }
41
42          // Hit the right Paddle?
43          if (col.gameObject.name == "paddleRed") {
44              // Calculate hit Factor
45              float y = hitFactor(transform.position,
46                                  col.transform.position,
47                                  col.collider.bounds.size.y);
48
49              // Calculate direction, make length=1 via .normalized
50              Vector2 dir = new Vector2(-1, y).normalized;
51
52              // Set Velocity with dir * speed
53              GetComponent<Rigidbody2D>().velocity = dir * speed;
54          }
55      }
```

We check using if statements which panel we hit, then use the hitFactor position to work out the direction and speed. Read through comments and make sure you understand what's happening, in particular, it may well be worth googling and explanation of normalisation.

If we press play, we can now influence the ball's bouncing direction depending on where we hit it. The game will now play like a rather basic pong game, to round it off we'll add a simple scoring system. When the ball hits the bit behind the paddle, we'll increase the score.

So, we need

- Something to hold the player's scores
- A bit of code to detect when the ball hits the walls
- Some logic that says when the ball hits the walls, update the scores

Once again, with Unity's built in tools this is pretty simple, the ball script already checks what the ball has collided with, so we'll add our code there. Open it up and;

Add a variable to hold each player's score of type int to the top under speed;

Then in the OnColission2DEnter() method, check to see if you've hit the LeftWall and if you have, add one to score. Then do the same to the RightWall. The two new bits are shown below.

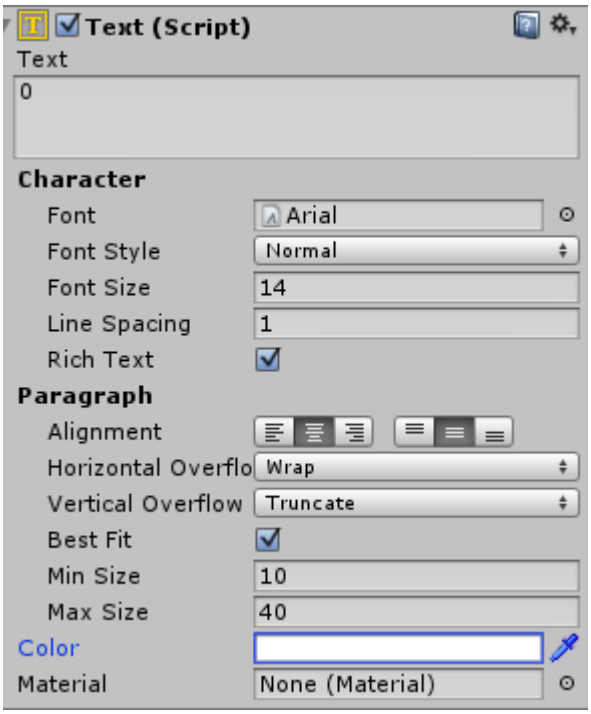
```
public int bluescore;
public int redscore;
```

```
//check which wall hit, update the score and redisplay the score.
if (col.gameObject.name == "LeftWall") {
    redscore++;

}
if (col.gameObject.name == "RightWall") {
    bluescore++;

}
```

Pong in Unity a basic Intro

Now all that's needed is a way to display it. This isn't going to turn into a Unity UGUI tutorial but it will cover some of the basics.

Add two new UI Gameobjects of type Text, position them at a sensible point and set the Text to 0, the colour to something that stands out and tick Best Fit (which will fit the text size to the size of the box.



Now we need to link those boxes to the code we wrote earlier. Again, Unity makes this simple.

Back to the ball code, (don't worry I'll post a full scipt listing shortly), add a line right at the top "using UnityEngine.UI". This simply says we're going to use the code associated with the new UI features.

Then add two public Text fields that will be used to hold the two Text objects for scores we created earlier.

```
1  using UnityEngine;
2  using UnityEngine.UI;
3  using System.Collections;
4
5  public class Ball : MonoBehaviour {
6      public float speed = 30;
7      public int bluescore;
8      public int redscore;
9      public Text redTextScore;
10     public Text blueTextScore;
```

and then where you increase the score you want to replace the text value of those objects with the current score. (Note: Text is the type text object, text is the actual text [e.g. 0]). Try it.

```
47          // Hit the right Paddle?
48          if (col.gameObject.name == "paddleRed") {
49              // Calculate hit Factor
50              float y = hitFactor(transform.position,
51                                  col.transform.position,
52                                  col.collider.bounds.size.y);
53
54              // Calculate direction, make length=1 via .normalized
55              Vector2 dir = new Vector2(-1, y).normalized;
56
57              // Set Velocity with dir * speed
58              GetComponent<Rigidbody2D>().velocity = dir * speed;
59          }
60          //check which wall hit, update the score and redisplay the score.
61          if (col.gameObject.name == "LeftWall") {
62              redscore++;
63              redTextScore.text=""+redscore;
64          }
65          if (col.gameObject.name == "RightWall") {
66              bluescore++;
67              blueTextScore.text=""+bluescore;
68          }
69      }
```

Pong in Unity a basic Intro

Challenges

Add a Trail Effect

Add the old Pong Sound that we all love

Increase the Ball's speed over time

Add a menu and a credits screen