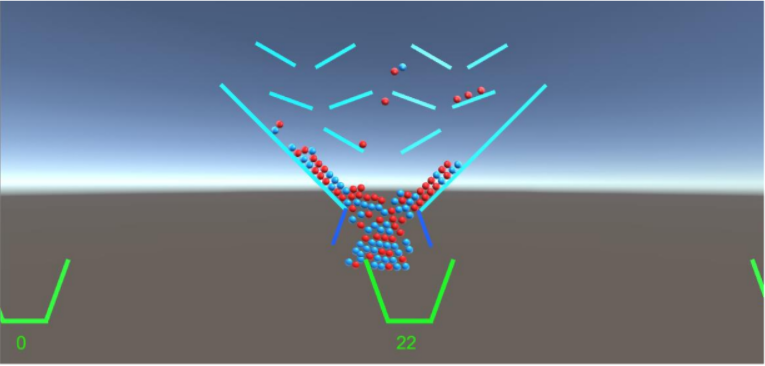
## Exercise - A Hundred Balls



In the past couple of sessions, we have covered various fundamental concepts of programming, including the C# language syntax, language constructs, algorithms and arrays. Most of what we have learned is part of the prerequisite knowledge required for making applications and games. Now that we have some of these basic skills, we can start improving them by practicing all that we have learned through making a game!

## Prerequisites

Before getting started, make sure you understand:

- How to [Navigate around Unity](https://www.youtube.com/watch?v=AMw3fm44ZK4).

- Unity Project to contain the project files

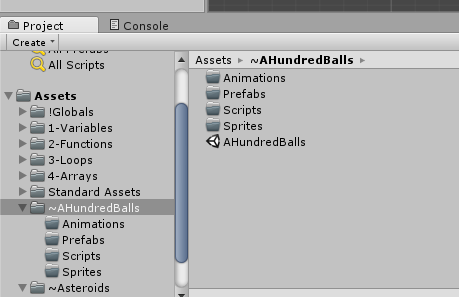
- C# language syntax: [C# Syntax Cheat Sheet](https://www.google.com/url?q=http://thebuildingcoder.typepad.com/github/Gytaco/RevitAPI/Handouts/Cs_Cheat_Sheet_THE_CODING_GUYS_1_of_2.pdf&ust=1504304040000000&usg=AFQjCNF0uqk0qmFiJKpZfYssYhwRtDqMEg&hl=en-US)

## Making A Hundred Balls

Let’s do this!

### Step 1 - Getting Started

We’re going to start by creating a folder inside of our project called “**AHundredBalls**”.

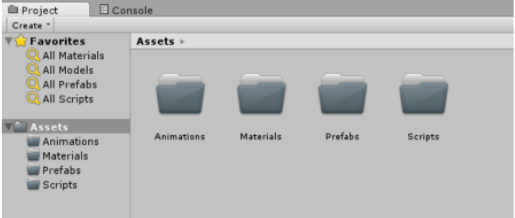


Once our project is created, let’s start setting our scene!

## 

Now, since this is going to be pretty big project, we’re going to make our files neater by creating folders inside our assets folder to house all of our game resources.

Create folders to store **Animations**, **Materials**, **Prefabs** and **Scripts**.



Great! Our project is going to be so much tidier now.

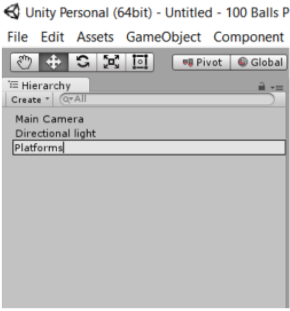
## 

### Step 2 – Lights, Camera, Platforms!

We need to ensure our project has a **Directional Light** so our objects will be visible on the screen as well as a **Main Camera** so we can actually see our game world.

Once we know that we have these, we’re going to start by creating our platforms for our balls to collide with.

Create a new **empty GameObject** and call it “*Platforms*”

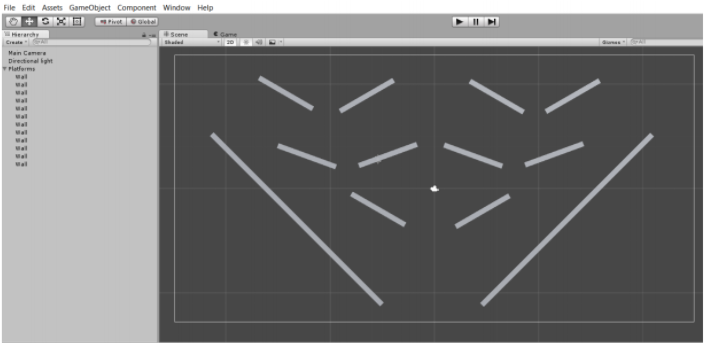


## 

## 

Now inside of this empty GameObject, we’re going to create **Cubes**.

Make sure to stretch and manipulate them using the necessary tools in Unity (i.e, The Move, Rotation and Scale tools) and set them up to look similar to this layout:



Make sure you highlight all of the cubes afterwards and name all of them to “*Wall*”.

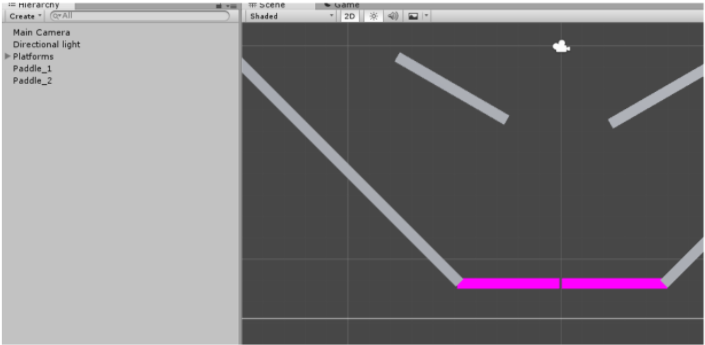
## 

## 

## 

### Step 3 – Paddles

We now need paddles to release our balls. To do this, we’re going to create **two(2)** new **Cubes** that will form the shape of our paddles. Then we’re going to place them in the correct position, like so:



Here, I have made their color pink, if you want to change the color of the GameObjects, **create a new Material** in the “*Materials*” folder, call it whatever you wish and drag that material over the GameObject you want to change color.

## 

## 

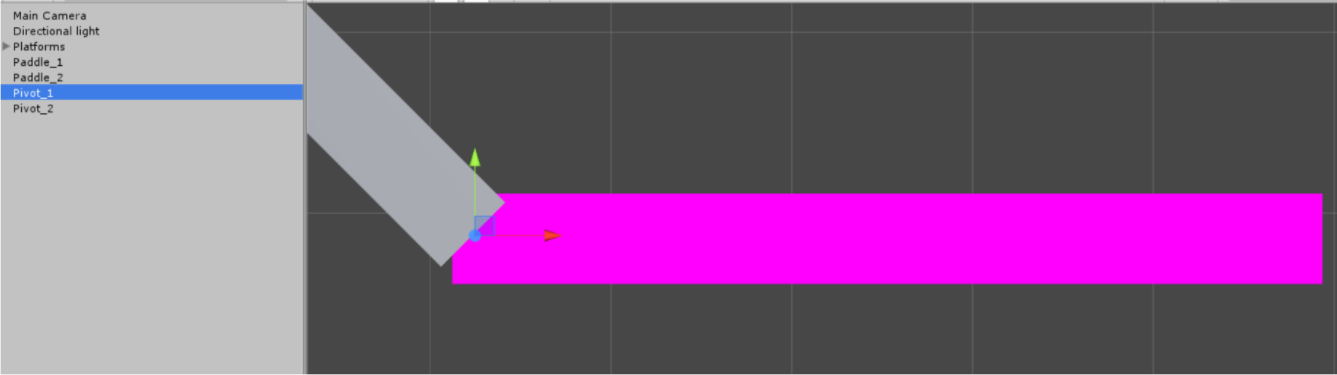
## 

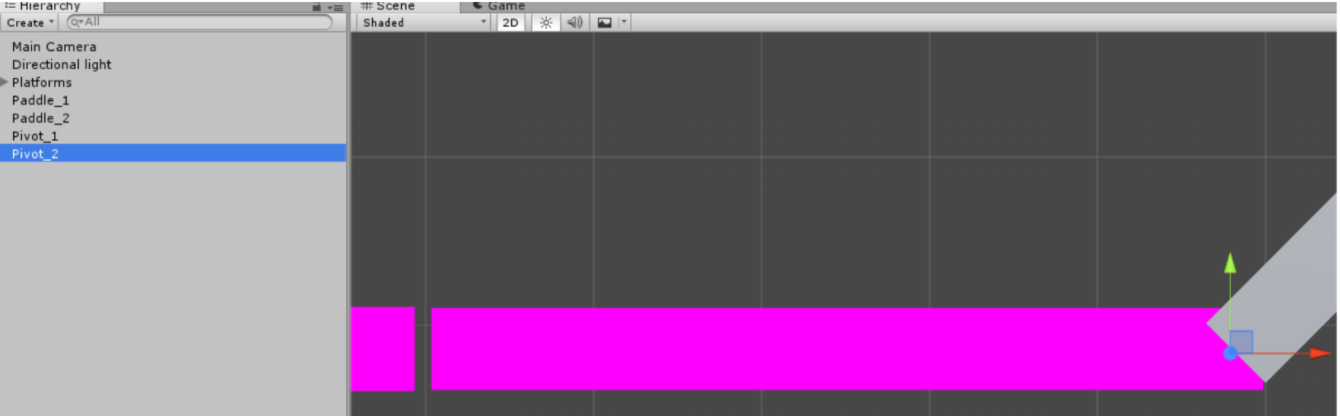
### Step 4 – Creating Animations

In order to animate the paddles, we’re going to use **Unity’s Animation system**. Note: If it’s not familiar to you, please be sure to read up on Unity’s animation system here: [Unity's Animation System Overview](https://www.google.com/url?q=http://docs.unity3d.com/Manual/AnimationOverview.html&ust=1504304760000000&usg=AFQjCNGPaFkcPc2iDm7dUBmaFdFNYMDCkQ&hl=en-US)

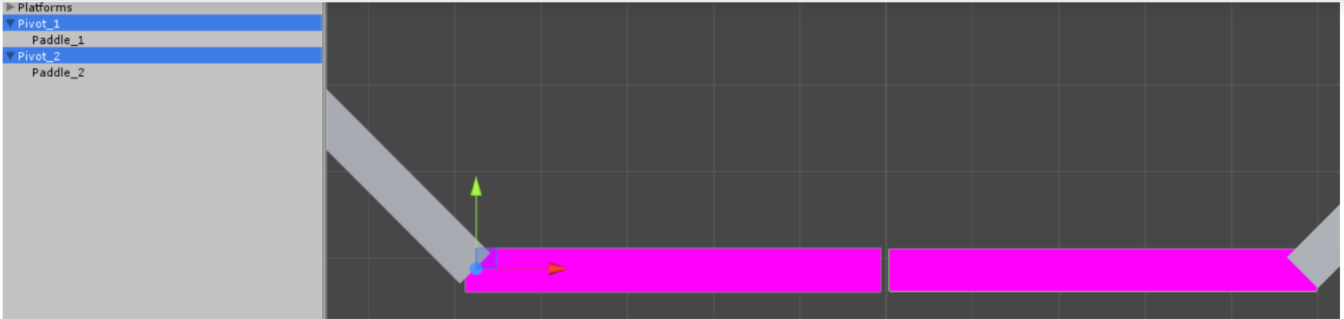
Firstly, let’s create pivot points that our paddles will swing from in the animation. **Create a new empty GameObject** and call it “*Pivot\_1*”.

Then set the newly created pivot point at the place where you want the Paddle to rotate around:

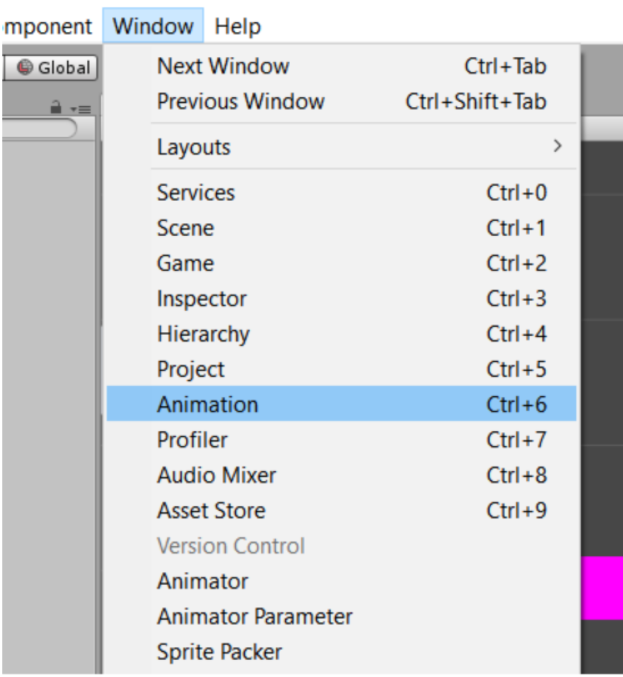


Then do the same for the other side:

Once you have done that, make **Paddle\_1** a child of **Pivot\_1** and so on:

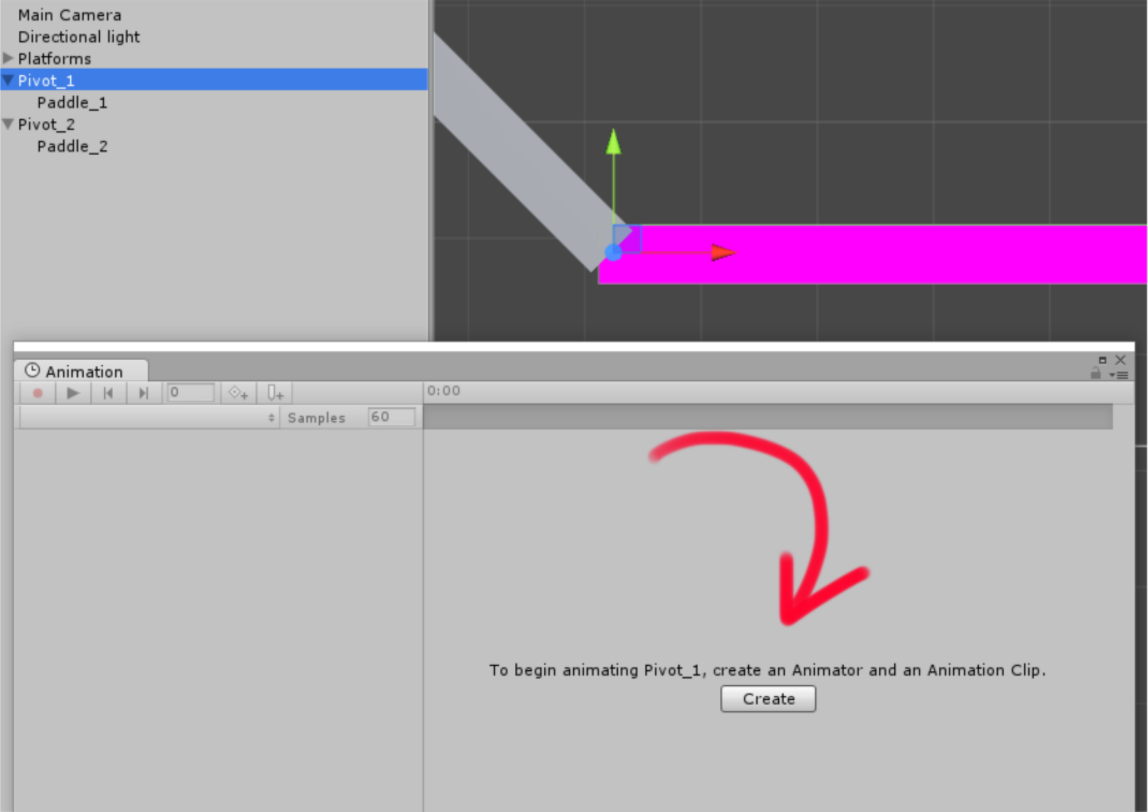


Next, we’re going to open Unity’s Animation window by clicking on **Window** > **Animation**.



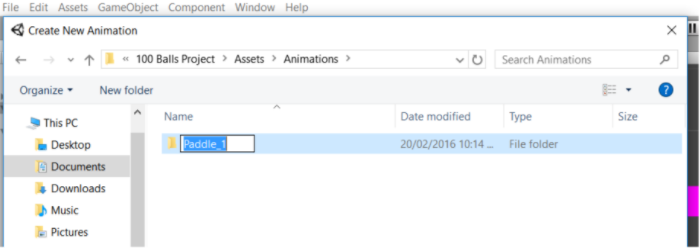
### Step 5 – Animating the Paddles

Now, with the **Animation** window opened, select the **Pivot** you wish to create an animation for and click on the **Create** button in this window

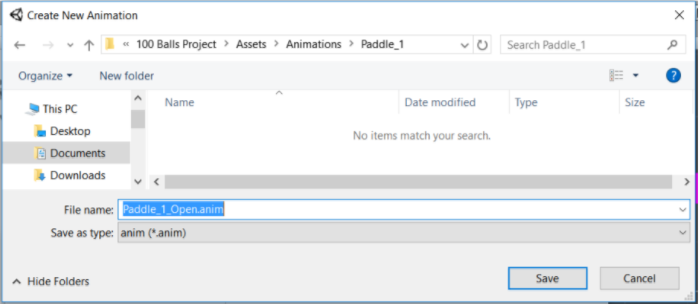


You will now be presented with the **File Explorer**. Navigate to your Project’s **Assets** folder and the **Animations** folder you created earlier.

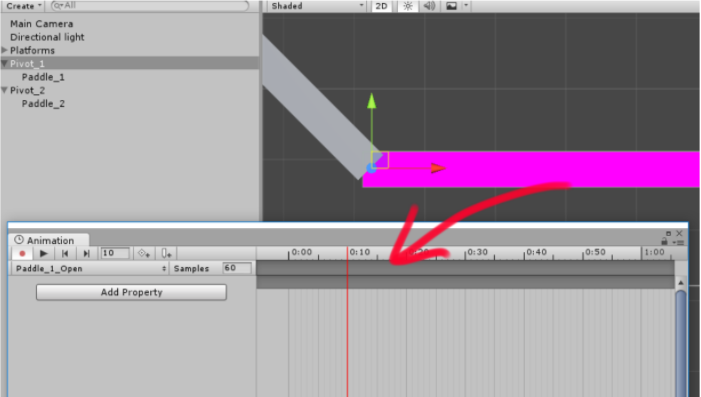
Now create a new folder and call it **Paddle\_1**



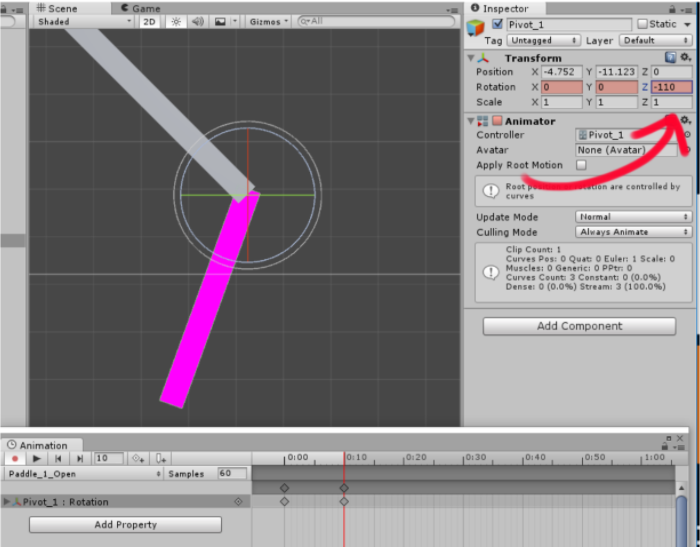
Then open up that folder and **save** the new **.anim** file with the name “**Paddle\_1\_Open.anim**”:



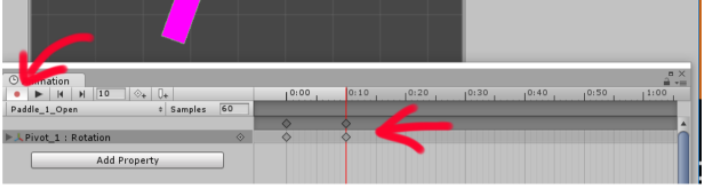
Going back to the Animation window, you should now see “*Paddle\_1\_Open*” selected in the drop down. Click on the “0:10” mark on the **animation timeline**.



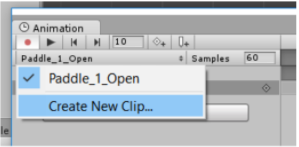
With this selected, **rotate** your Paddle around **-110** along the **Z**:



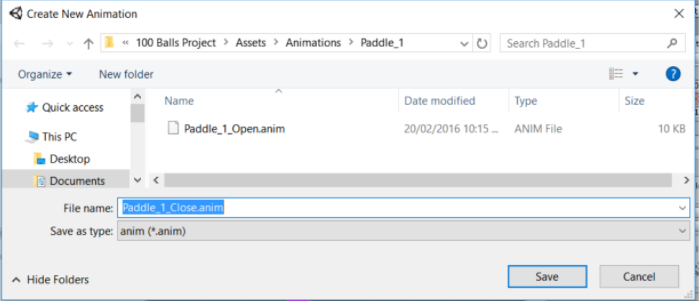
You should notice the recording button (red dot) is selected and the key frames will be created for you:



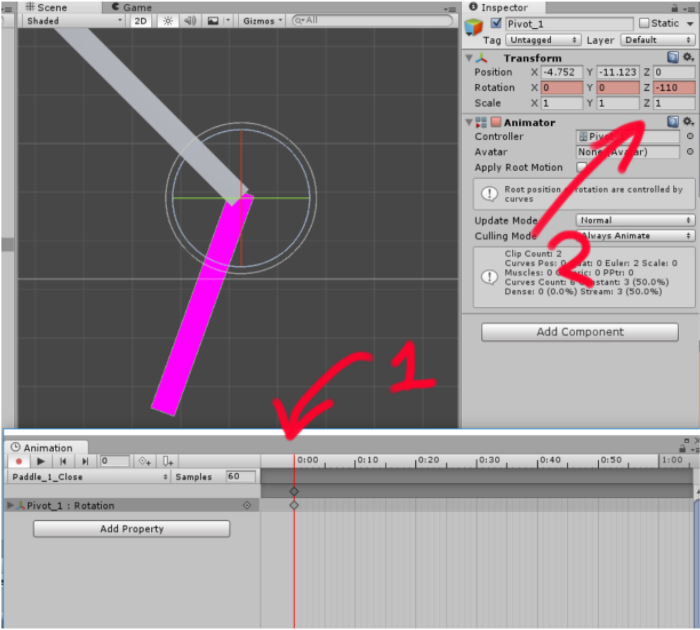
After we have finished with our Paddle\_1\_Open clip, we need to create a new clip:



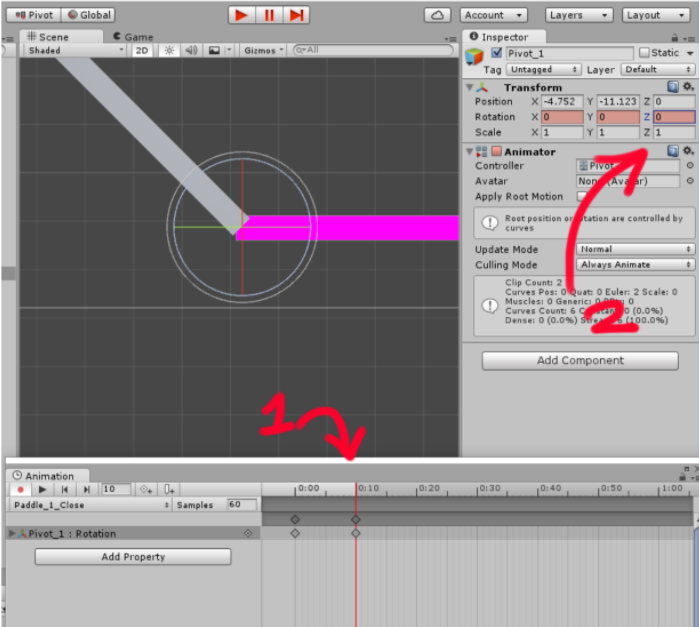
..and call it “**Paddle\_1\_Close**”:



This time, we’re going to start by selecting time **0:00** and rotating the paddle around **-110** along the **Z**:



Then select the **0:10** mark in the animation window and rotate the Pivot\_1 back to **0** along the **Z** axis:



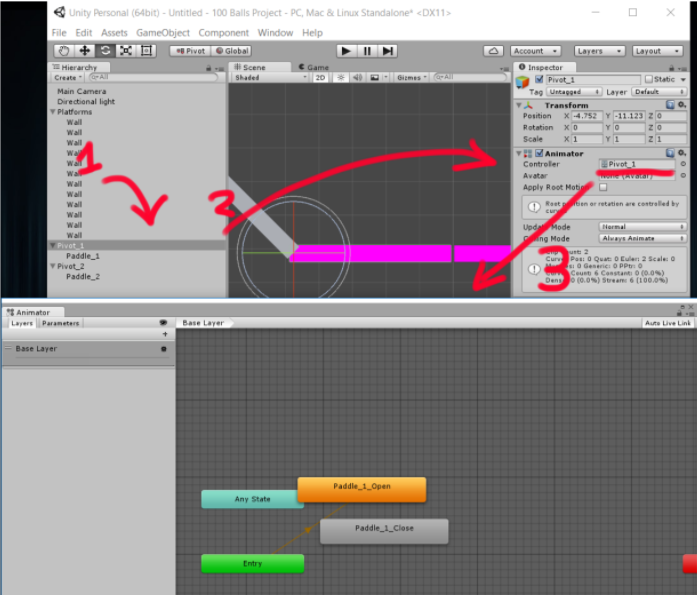
Great! Now our first Paddle is animated!

Now do the **exact same step** for Pivot\_2, remembering to rotate around **110** degrees along the **Z**.

### Step 6 – Setting up Animation States

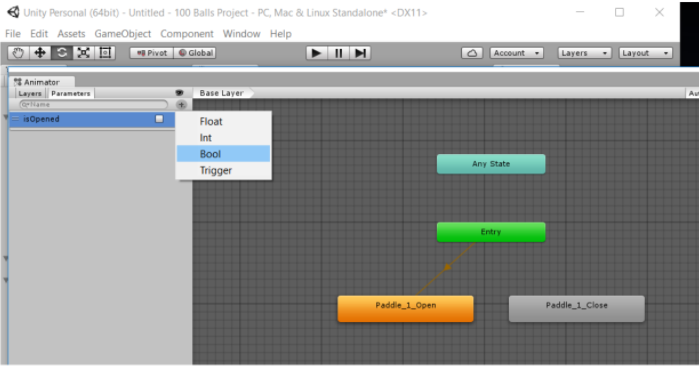
If we run our project, you’ll notice that our Paddles open once and they do not close. We need to set up **Animation States** and trigger them on a keypress event. The way we do this is using Unity’s Animator window. Note: If you don’t quite know what **Unity’s Animator** is and what it does, watch this: [The Animator Controller](https://www.google.com/url?q=https://unity3d.com/learn/tutorials/modules/beginner/animation/animator-controller&ust=1504305240000000&usg=AFQjCNEo6x2M4FQV-sC-bQYIHT7SkwTt2A&hl=en-US).

Open up Unity’s **Animator** by clicking the **Pivot\_1** GameObject and in the Inspector, double-click on the **Pivot\_1** Controller like so:



This is what’s known as a “Mecanim State Machine”. You’ll first notice that both of our animation clips we created are created as an **Animation State**. This is what we will use to jump between the **Paddle’s** Open and Close states in our game.

First off, select the **Parameters** tab located in the top-left of the window. Then click on the little ‘**+**’ symbol to add a new Parameter and select “**Bool**”. Call it “*isOpened*”:

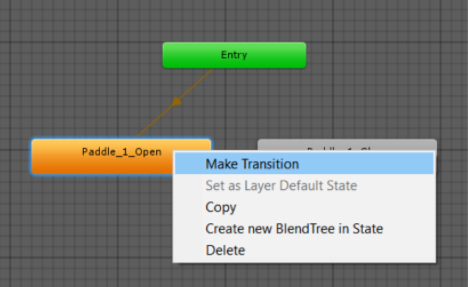


This is going to be the *bool* we access in our C# Script that will allow us to switch between the open and close states of our paddles.

## 

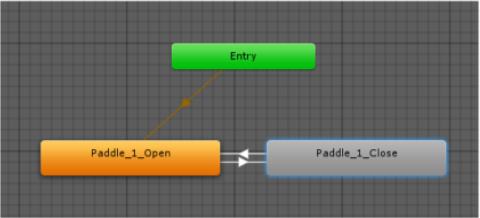
In order for our animation to switch between states using our newly created **Bool**, we need to create a transition between states.

Right-click on the **default state** (the orange state) and select **Make Transition**:



An arrow should come out and follow your cursor. Now select the **Paddle\_1\_Close** state. This makes a transition between states.

Now connect another transition from **Paddle\_1\_Close** > **Paddle\_1\_Open** and you should end up with this:

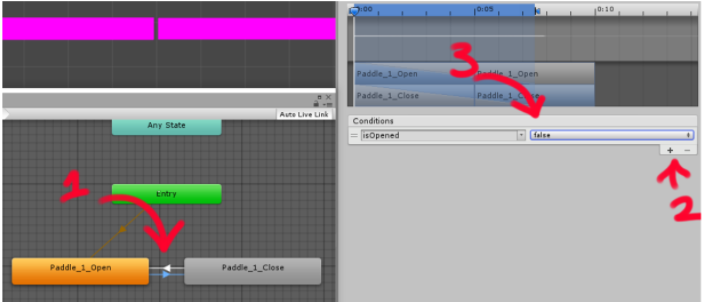


Okay great!

1. Now we’re going to select the transition going from **Paddle\_1\_Open** > **Paddle\_1\_Close** and you should notice the **settings** for it show up in the inspector of Unity.

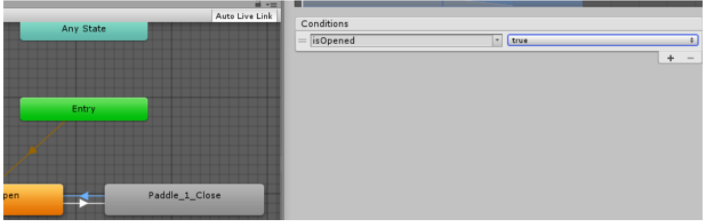
2. Underneath **Conditions**, add a new condition by clicking the ‘+’ symbol and you should see our parameter show up next to it.

3. Set **isOpened** to **false**:



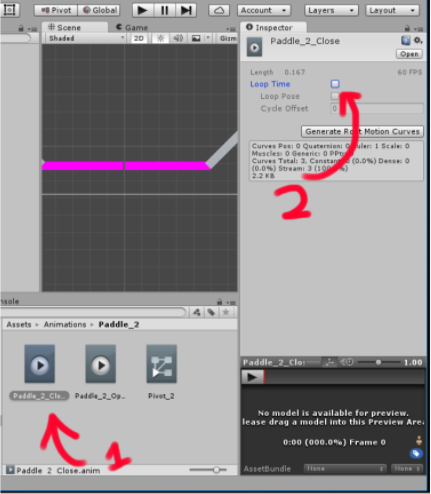
This way, in code, when we change the variable *isOpened*, it will switch between states.

Now do the same for the other transition, but set **isOpened** to **true**:



One last step, we need to make sure that all of our **Animation Clips** do not repeat (loop).

1. Click on your animation clips in your assets folder

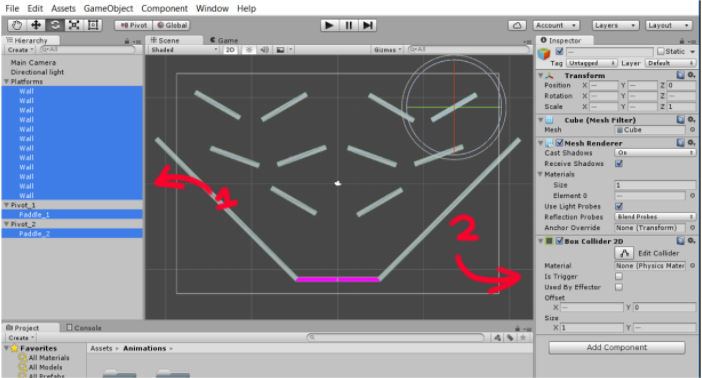
2. Make sure the **Loop Time** variable is **Unchecked**

### Step 7 – Creating a Ball Spawner

In this step, we’re going to create a spawner that actually spawns our Balls so that they can interact with our current world.

To start off, we’re going to have to swap out all of our **BoxColliders** for **BoxCollider2Ds**, otherwise, our GameObjects will interact with the 3D Physics engine.

1. Select all of the **Walls** as well as both of the **Paddles**.

2. Remove the “**BoxCollider**” component and replace it with **BoxCollider2D**

## 

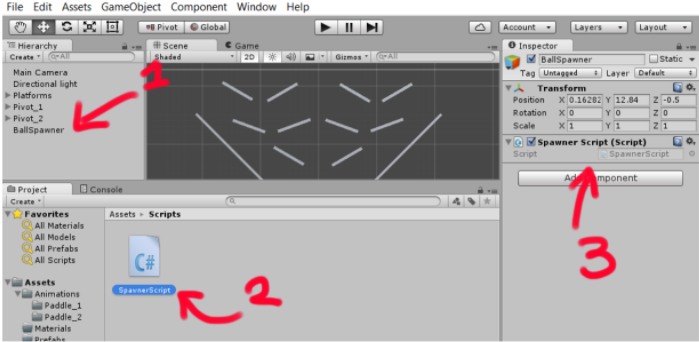
## 

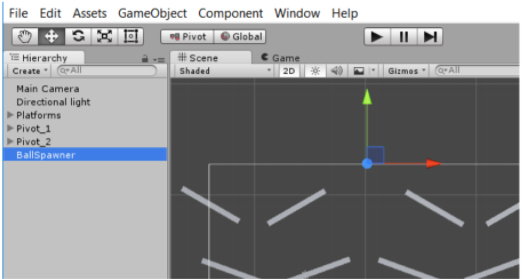
## 

1. Now, we’re going to **create a new empty GameObject** and call it “**BallSpawner**”.

2. Then we’re going to **create a new C# Script** in our Scripts folder and call it “**SpawnerScript**”

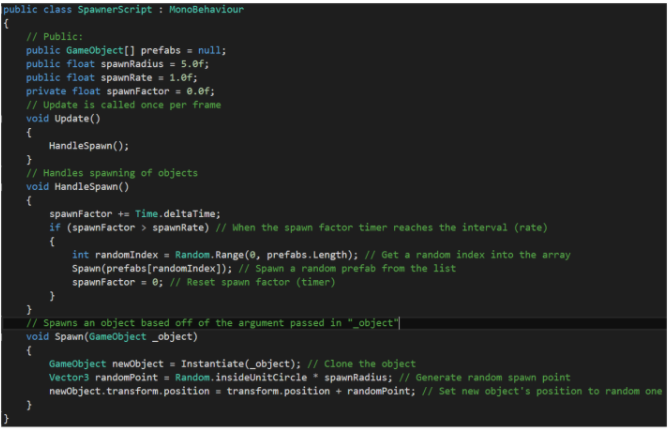
3. With this new script we create it, attach it to the **BallSpawner** GameObject



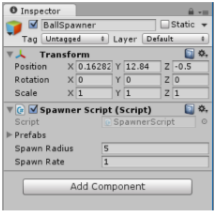
Make sure **BallSpawner** is set just above your platforms, that way they spawn in the correct position:

Here’s the code required for our spawner. Make sure you understand this code by commenting in the appropriate places like so:

*Note: If you do not like how this code is working, feel free to add your own functionalities!*



With this code written, go back to Unity and select the **BallSpawner** GameObject. You should see this in your inspector:

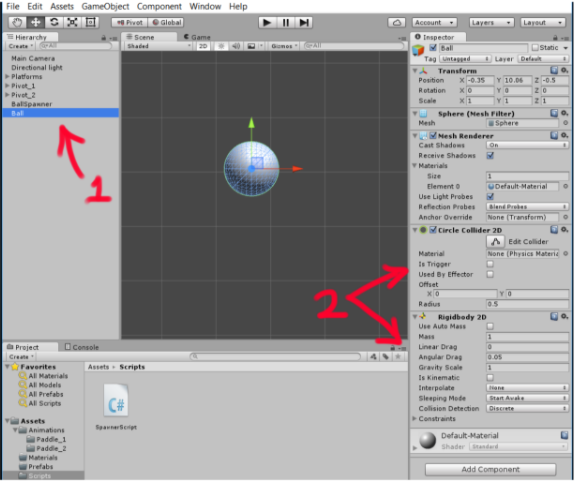


*Note: If you do not see this but you have written the code, check Unity’s Console for any outstanding Errors/Warnings that may need attention. Then look up a fix for it.*

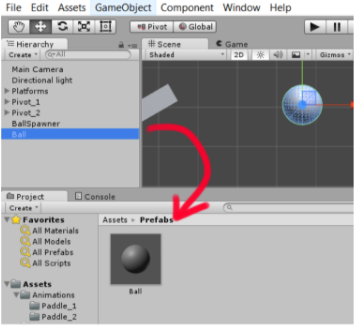
We now need to actually create a ball prefab for our **BallSpawner** to create. Otherwise, we won’t have any balls moving around the place, freely.

1. **Create a new Sphere GameObject** and call it “**Ball**”

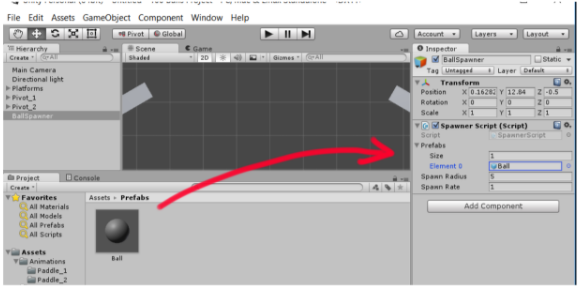
2. Swap the **SphereCollider** with a **CircleCollider2D**, otherwise it won’t interact with our platforms. Also add a **Rigidbody2D** component.

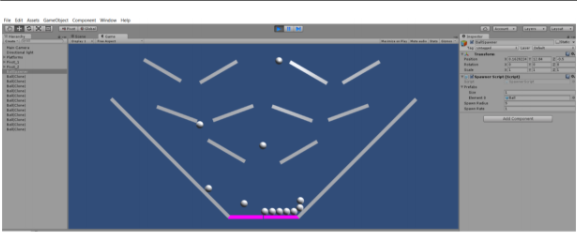


Next, create a prefab by clicking and dragging the **Ball** we just modified into the **Prefabs** folder in our Assets folder:

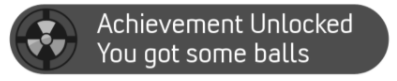


Next, click and drag the prefab into a new element of our Spawner Script:





Yay! You should now have some balls!

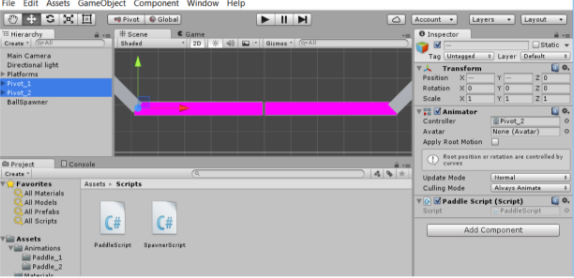


Now that we have done this amazing achievement! We can continue to make our balls drop!

### Step 8 – Moving the Paddles

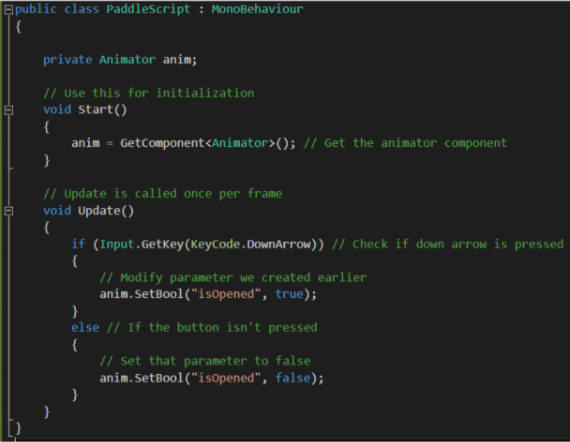
We’ve already done a lot in our project, now it’s time to add some interactivity by making our paddles open to drop our balls.

Create a new **C# Script** called “**PaddleScript**” and attach it to both of our **Pivot** GameObjects:

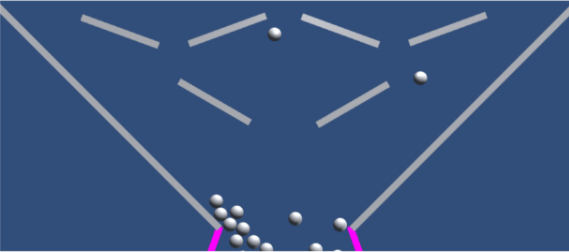


Next, we’re going to use the following code this code:

*Note: Remember to comment where necessary and you are able to modify this however you wish!*



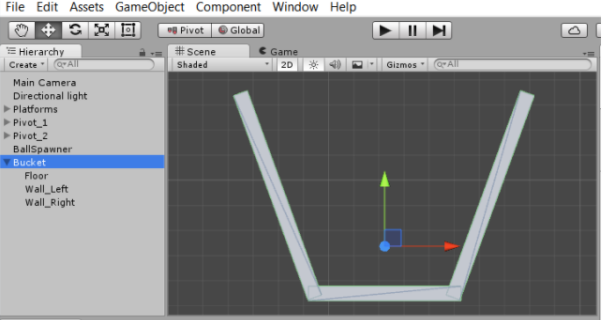
If we run our project, we should be able to drop our balls!



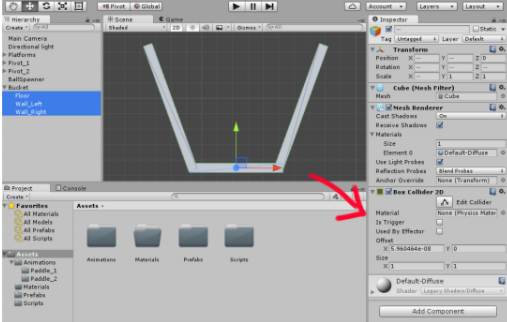
### Step 9 – Creating Buckets

We need to create buckets to catch our balls so we can deliver them safely to the other side of the screen for our final score.

**Create a new empty GameObject** and call it “*Bucket*”. Then create **Cubes** inside of it that are formed in the shape of this:



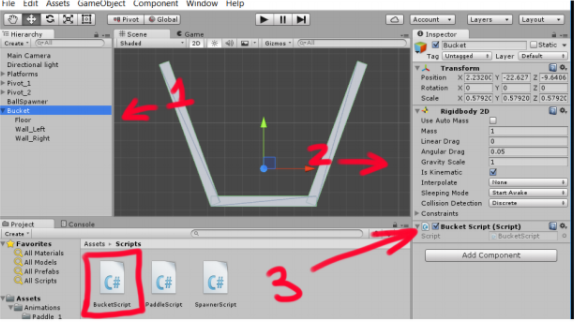
Remember to swap out the **BoxColliders** for **BoxCollider2Ds**

****

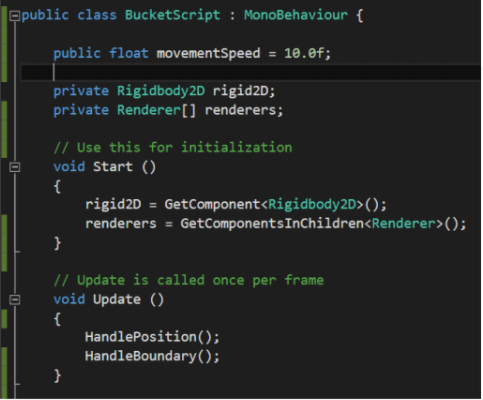
1. We’re going to select our **Bucket** parent

2. Attach a **Rigidbody2D** component to it

3. Create a **BucketScript** C# Script and attach it to the **Bucket**

****

Now, here is the **first half** of the code we need to create our Game:

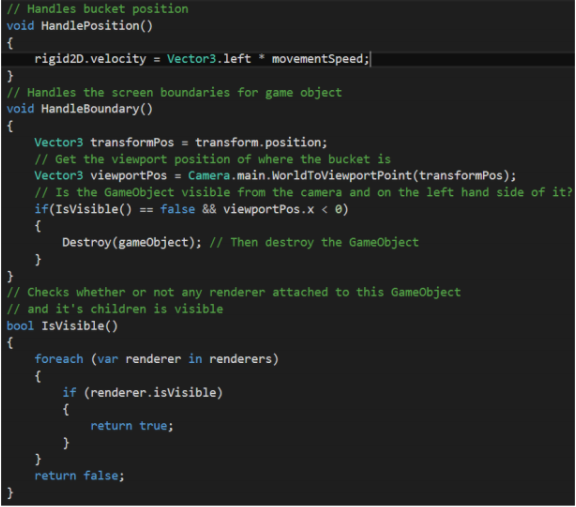
****

Notice how we are obtaining the **Rigidbody2D** component as a single variable and our **Renderer** as an array?

Well what you will also notice is we’re calling a function known as **GetComponentsInChildren** which what it does is get multiple components from its children that are of type ‘Renderer’ in this case.

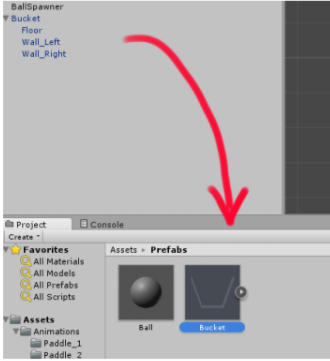
We’re going to be using this in the second half to keep a mental note of that.

Here is the second half of the code required:

****

Here we are using the **renderers** array by looping through and checking if any one of them is visible, that way we can destroy the GameObject if that is partly the case. The other case is if the Bucket flies off to the left of the screen.

We’re going to make a prefab of this bucket we created and use it later:



### Step 10 – Finishing up

Lastly, just like our BallSpawner that is using a **SpawnerScript** to spawn balls outside the screen, we’re going to create a spawner for our buckets:

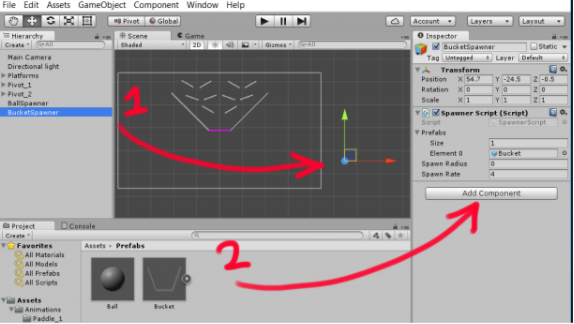
1. **Create a new empty GameObject** called “BucketSpawner” and

have it be placed just to the right-hand side of the screen where it will spawn

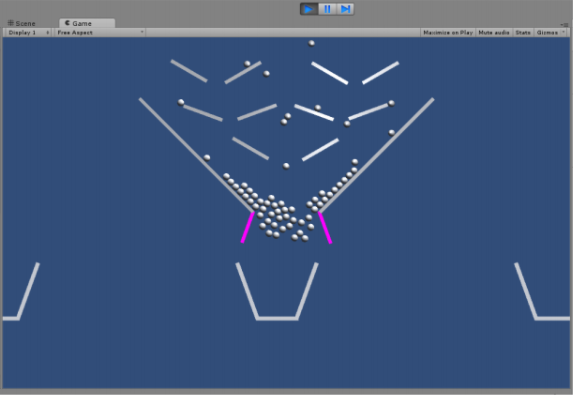
2. Then drag our **Bucket prefab** into the SpawnerScript’s prefab

array at **element 0**. Set **SpawnRadius** to **0** and **SpawnRate** to **4**.

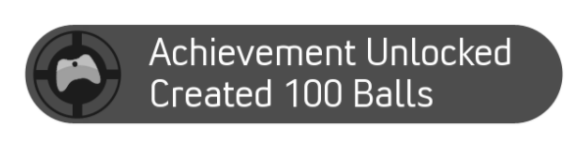
Note: Play around with these values if you wish.



If you run your code, you should see this:



There we have it! We created our ball project!



## Homework Tasks:

It looks as though we have done quite a sufficient amount of work trying to get basic gameplay in our game. But like any other grand projects, there are always ways to improve upon it and make it better!

Here are just a few things you can possibly do to improve it, but go ahead and add any other features you feel would make it better!

1. Try and spawn **Balls** and **Buckets** with their own random material

applied to it. Also try adding components such as **trail renderers** and other components that add interesting visual features to the balls.

2. Using **triggers**, modify the BucketScript to count how many balls are dropped inside of it and display the player’s Score using a Text object. Hint: [OnTriggerEnter2D](https://www.google.com/url?q=http://docs.unity3d.com/ScriptReference/MonoBehaviour.OnTriggerEnter2D.html&ust=1504311420000000&usg=AFQjCNFd8DfHFdBQGTOaxo09X31MAqk-VA&hl=en-US)

*Note: If you don’t know how to use Text to display the score, follow this tutorial from Unity:* [*Displaying the Score and Text in Unity*](https://unity3d.com/learn/tutorials/projects/roll-ball-tutorial/displaying-score-and-text)

## Extra Challenges:

1. Look up the “100 Balls” game play on YouTube: [100 Balls Gameplay](https://www.youtube.com/watch?v=zqzvfeuJiDw) Try and make the buckets fly around the screen and drop balls when it reaches the top. Or, you could make it so that the bucket rotates on key press when it reaches the top.