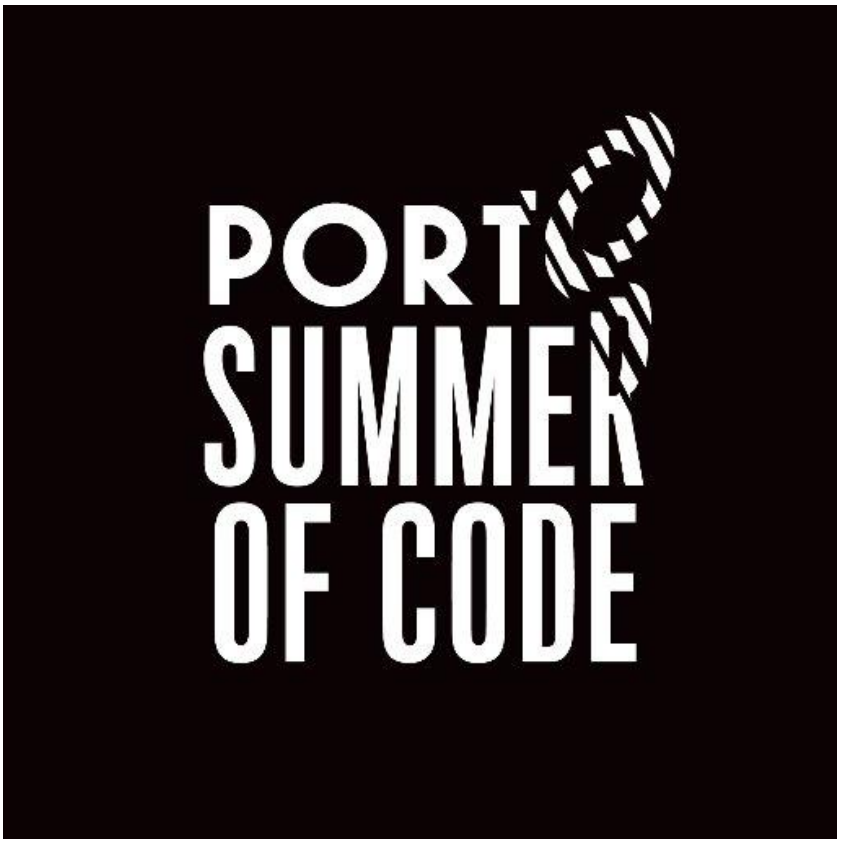
Super Angry Bowser

- An Angry Birds Clone



Introduction

This document will guide you through the process of creating the Super Angry Bowser game in the Unity3D WYSIWYG (What You See Is What You Get) game making tool.

Requirements

System Requirements for Unity Authoring

- Windows: XP SP2 or later; Mac OS X: Intel CPU & "Snow Leopard" 10.6 or later. Note that Unity was not tested on server versions of Windows and OS X.
- Graphics card with DirectX 9 level (shader model 2.0) capabilities. Any card made since 2004 should work.
- Using Occlusion Culling requires GPU with Occlusion Query support (some Intel GPUs do not support that).

System Requirements for Unity iOS Authoring

- An Intel-based Mac
- Mac OS X "Snow Leopard" 10.6 or later

System Requirements for Unity Android Authoring

In addition to the general system requirements for Unity Authoring

- Windows XP SP2 or later; Mac OS X 10.6 or later
- Android SDK and Java Development Kit (JDK)
- Android authored content requires devices equipped with:
    - Android OS 2.0 or later
    - Device powered by an ARMv7 (Cortex family) CPU
    - GPU support for OpenGLES 2.0 is recommended

System Requirements for Unity-Authored Content

- Windows XP or later; Mac OS X 10.5 or later.
- Pretty much any 3D graphics card, depending on complexity.
- Browser games require HTML5 (WebGL) support.

Please note that for this tutorial we will be using Unity 2017.1 . The game should also work in newer versions with minor or no changes.

Objectives

By the end of this document, you should be able to play Super Angry Bowser, a 2D Angry Birds clone game. This game encompasses the following features:

- 2D Super Mario Bros. animated sprites
- Original Super Marios Bros. Sound effects and soundtracks
- Physics based projectiles and objects
- Simple GUI
- Tap and fire mechanics, useful for mobile support

For the sake of briefdom, some other game essential features will not be included in this guide (score, leaderboards, persistency, etc). However, the bases will be set for you to explore Unity and add other things as you see fit. Maybe some powerups, enemies or whatever you can think of. Let your imagination run wild.

Assets

Assets that are needed for this Tutorial:

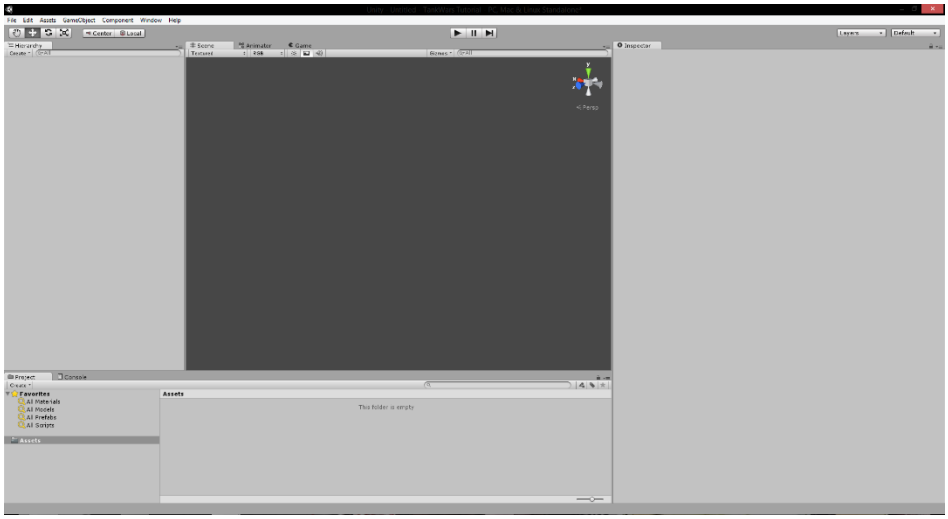
- **Assets Folder**,
- **Full Project (optional)**

Super Angry Bowser Tutorial

This chapter will be divided into muliple parts or chapters that encompass diferent parts of the game. At the end of each chapter, you should end up with something that is “playable” (as in, you can run it and check the results of what you’ve done in Unity’s Game window). Some suggestions are also given, as well as some food for thought.

Part I : Setting up Unity

Okay, you've downloaded Unity from their website, set up an account, logged in, created a new 2D project (probably named "Angry Bowser" or something) . And now what you see is this:



This is what you'll be looking at for the next hours. It is comprised of several views. The most important ones for you are:

- **Hierarchy view** – lists all the elements (Game Objects) of your current scene,
- **Scene view** – a 2D/3D view of all the game objects in your current scene,
- **Project view** – a tree of folders and assets that are used in your game,
- **Inspector view** – shows details of the current selected Game Object,
- **Sprite Editor** – allows for the creation of sprites from a sprite sheet.

You can take your time to locate these elements. You can also move them around, expand them, dock them or undock them to your heart's content. On the top right corner, you'll see a Layout combo box that probably says "Default". You can click on it and select a different layout (or you can create and save your own) if you don't feel like messing with the tool's GUI too much.

You will also notice three buttons at the centre-top of the window. These might look familiar to you:

- **Play button** – allows you to test your game and play the current selected scene
- **Pause button** – pauses the game
- **Step button** – advances one cycle in the game and re-pauses it.

So go ahead, press the play button! You should see an empty blue canvas. Notice that you've went from the "Scene" view to the "Game view". The game view is what you'd see in the game. You can still check and navigate around the Scene view even when in Play mode.

Now, **save your scene**. Either via **CTRL + S** or through **File -> Save Scene**. I called mine “Level1”, as it is the scene we will be working on (it will represent the first level).

Look at your hierarchy view. You notice that there is a “Main Camera” there. Click on it.

Your inspector view should show some information like this. You should also see a small window called “Camera Preview”, filled with blue, appear inside the Scene window. It represents what that camera sees of that scene.

Back to the inspector view of the Main Camera Game Object. Each thing in it is a **component** of the “Main Camera” game object. Let’s focus on these two:

- **Transform** – holds information regarding the position, rotation and scale of an object
- **Camera** – this component means that this Game Object is a Camera. It has a lot of parameters, most of which even if you tinkered with, you wouldn’t see any visual effects (as the scene is empty).

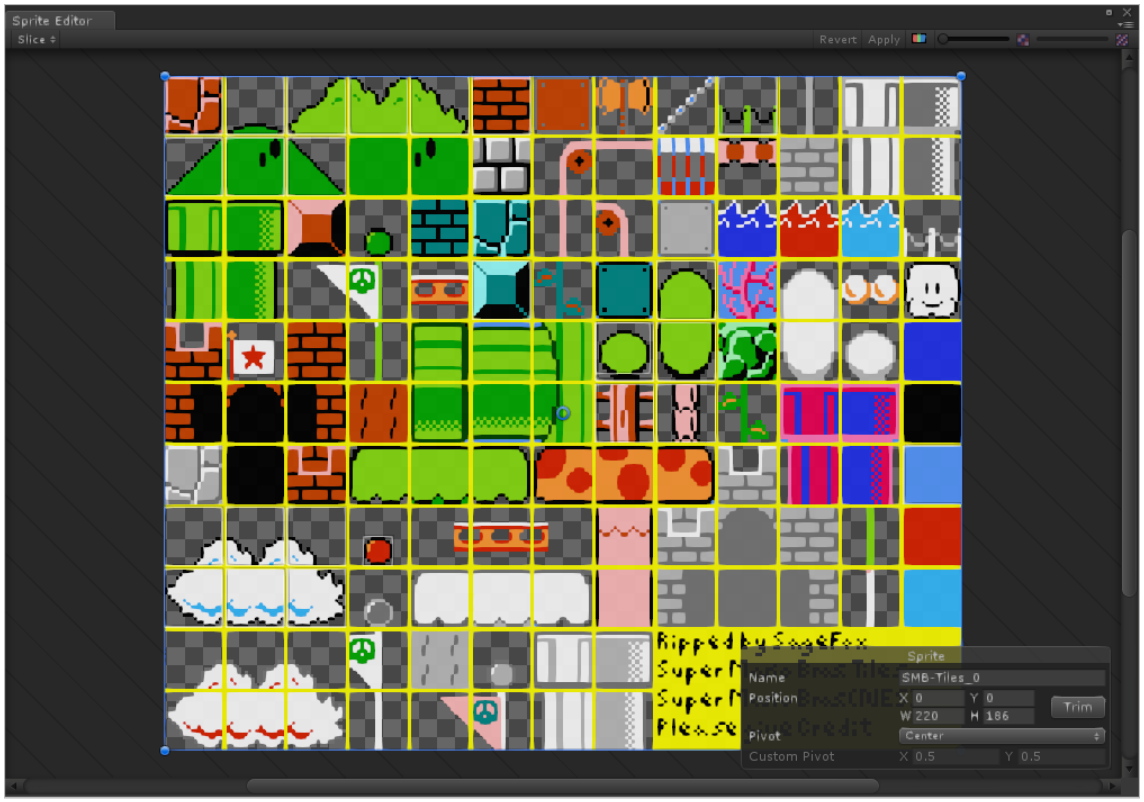
Please copy the above Camera settings, in the image, (no need to copy the Transform settings) and apply them to your own Main Camera. We will be using an orthographic camera (since it is a 2D game).

Now, lets import some assets into our project. Double click the **SuperAngryBowserAssets.unitypackage** to open the import window in Unity. Import all the files and folders. If it asks you something about migrating, go ahead and click on the lie : “I Made a Backup, Go Ahead” button! You should now have a few folders (Fonts, Scripts, Sounds, Sprites) in your project’s Assets folder.

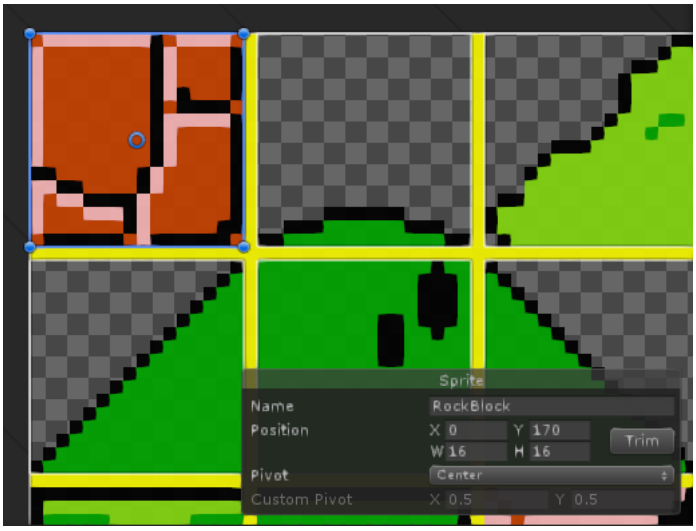
Okay, so let's add something to the scene. Let's start with creating our first sprite.

Go to **GameObject -> 2D Object -> Sprite**. A "New Sprite" game object should appear on the Scene window and hierarchy view. Click on it and notice the components that make him up. Note that it has a "Sprite Renderer", responsible for holding the graphics related information of the object. Now, we need to add a "Sprite" to the "Sprite Renderer".

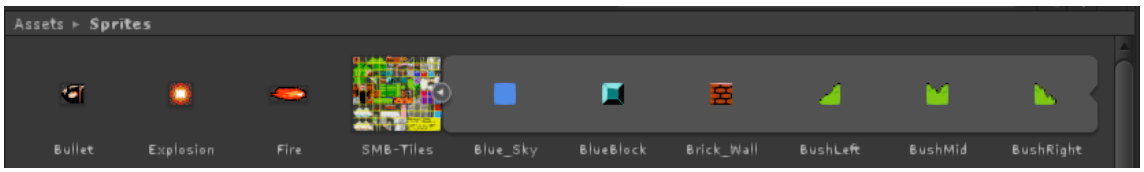
> Go to the Sprites folder on your project folder inside Unity. Notice it has quite a few Textures. Some of them, quite big as they comprise several sprites (they are called "sprite sheets"). Select the SMB-Tiles Texture. Now, on the Inspector window, you should see a button called "Sprite Editor" (note, this will only show up if the Texture is imported as a Sprite (2D\uGUI) **and** it's Sprite Mode is "Multiple". Click on it.



You should see the above window pop up. Now, we will be selecting our first sprite from it! Select the Rock sprite on the top left corner. The idea is creating a boundary around it (click and drag around it) and giving it a name (on the lower right menu), like so:

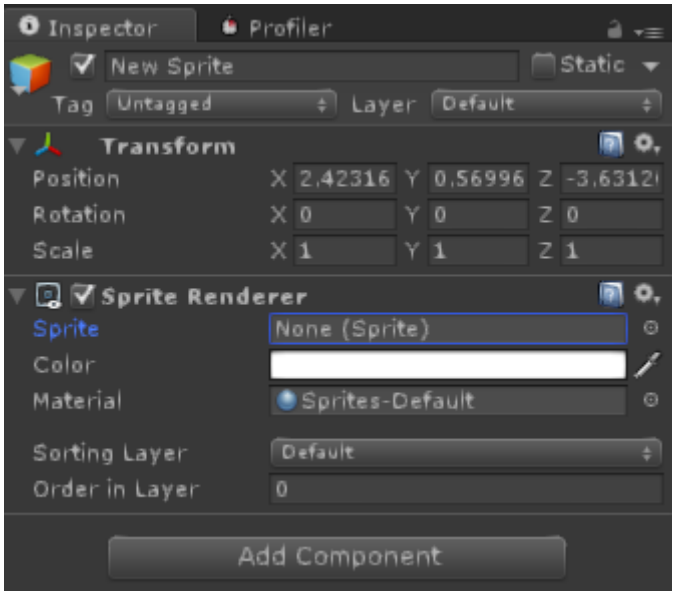


Now click “apply” on that window. Now notice how an arrow appears on that sprite sheet in the viewer. It contains the sprites you delimited in this editor.



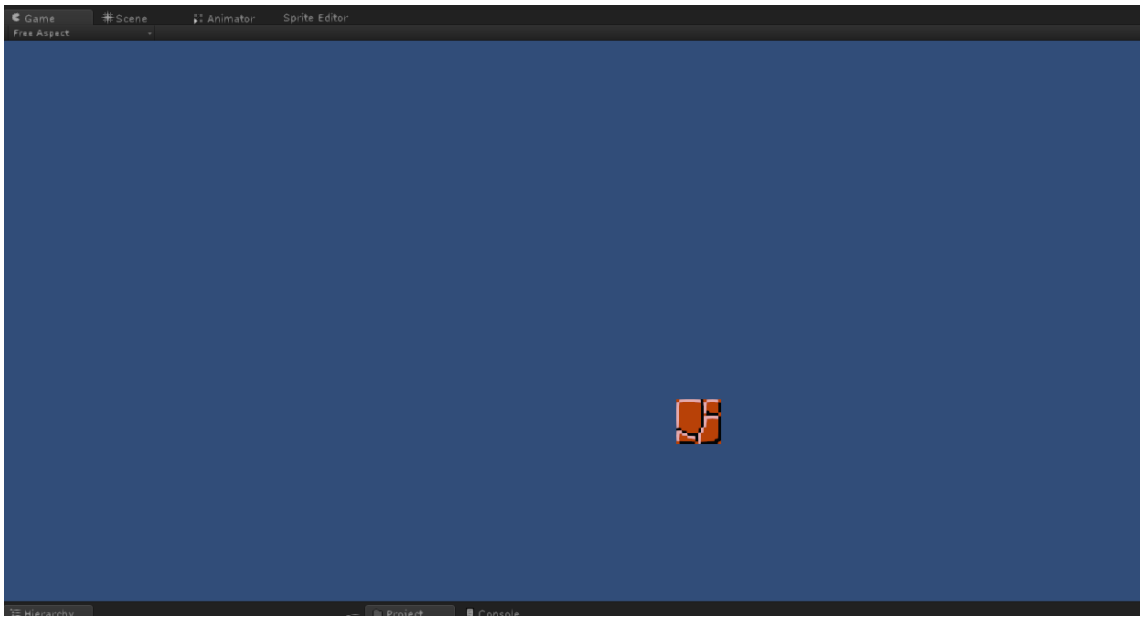
(Mine here contains already quite a few sprites. It’s your call how many and which ones to use).

Now, go to the New Sprite you created on your hierarchy view. Click on it and look at your Inspector window. Notice the Tag field that says “Untagged”. Let’s mark it as being a “Ground” block. Select AddTag, create a new tag named “Ground” and add it to the sprite. On the field “Sprite” under the “Sprite Renderer” section, click on the little ball to the right. It should pop up a new window asking you to provide a sprite for it to be used.



Yay, we have our first sprite (if you can’t see it, go to Edit->Frame Selected, it should show you your first sprite in the game world)! Now, press play. Chances are you are still seeing a bland

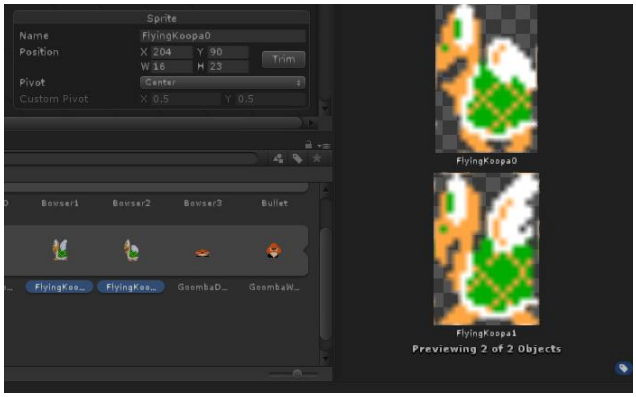
blue screen, right? If so, click on the Main Camera again now. It still shows an empty blue? Well, that means that the sprite isn't aligned with the camera. Go to the scene view and manipulate the scene view. If you get lost, just select the plane object in the hierarchy view and either press the **F** key (while having the mouse over the scene window) or go **Edit –> Frame Selected** to have the scene view aligned with the plane. Once you have a good view of the scene, select the Main Camera object and go to **GameObject -> Align with View**. Now your camera is aligned with the current view of your scene. You could, alternatively, manually align the camera with your sprite by manipulating the component transform of your sprite or camera object manually. If you hit play, now you are seeing your plane! Pretty static, right?



Let's make an animated **Koopa**!

Go to the Sprite Editor, this time for the SMBenemies Texture Sheet. Notice that it contains quite a few enemies, and the different frames for their different animations. To make it a bit easier, we will only consider a single animation for each enemy (note that if you are interested in multiple animations, they are already available in the asset folders of the full project). So, in the Sprite Editor window start creating single sprites as you did before for the Rock sprite, but this time, one for each state (two in total) of the flying koopa.

Once you have both those sprites created, you'll now learn a new way to create sprites and an animation at the same time! In the project view, simple select both these sprites, like so.



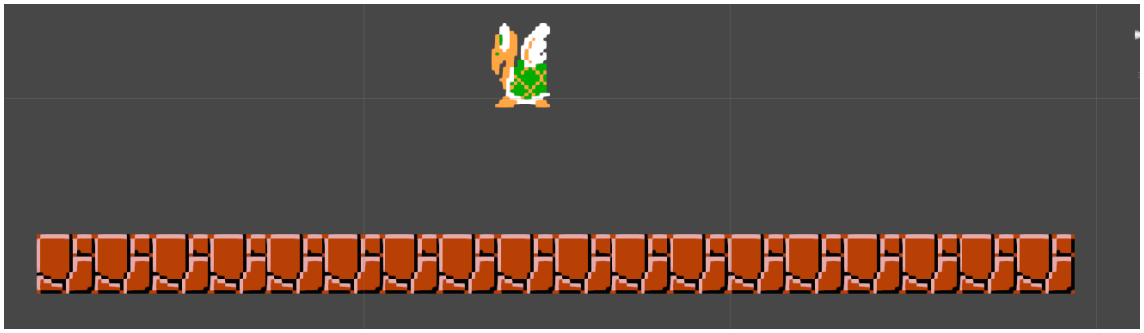
And now drag and drop it to the hierarchy view (Be sure that the game is not running anymore. If the “Play” button looks different from the other two ,Pause and Play Frame, it is still running. Hit Play again, and it will stop!) It will ask you where to save a new animation (I’d recommend a folder named “Animations”, otherwise things can get pretty messy from now on). Name it something like Flying Koopa.

Now we have a new game object on the scene, a new animation, and an animation controller (we won’t cover the animation controller for now). Place the new Game Object on top of the RockSprite, and hit play.

Well, it is moving. But too quickly, right? Then go to the animation controller (probably named FlyingKoopa.controller) and open it. It should open a new window, called animator. Select “Flying Koopa” and change it speed to something more reasonable (it shows 1 in Speed, meaning it is running at 1*60fps. Maybe go for about a third and put in 0.3 in the Speed value).

Go Back, hit play. It should play slower, and more like it would be in a Mario Bros Game. Now, let's add physics to the game.

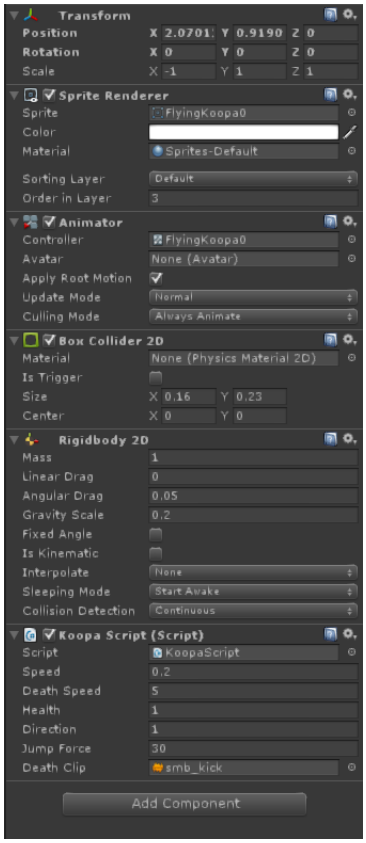
Select the Rock game object from the hierarchy view (always remember to stop "playing" the game when editing it. Otherwise, changes are temporary). Select Add Component -> Physics 2D -> Box Collider. This component allows us to check for collisions automatically. Now, do the same to the flying Koopa game object. Additionally, add another component, a RigidBody2d, by selecting **Add Component -> Physics 2D -> RigidBody2d**. Change the component's "Gravity scale" to 0.2. Now, move the turtle a bit over the block. Press play and see it fall onto the ground. Things are getting more dynamic now! It's time for us to create a scenario. Create a folder named "Prefabs". Drag the "Rock" game object from the Hierarchy window to it. It will create a prefab for the game object, i.e., a way for you to quickly create instances of it. Now, create a new Empty Game Object (**GameObject-> Empty Game Object**) and drag the Rock Game Object to it. It is now a child of that Empty Game Object. This allows us to truly create hierarchies. Copy paste a few rocks and move them around till you create a nice floor to your liking.



Now we start having something that could look like a simple level for our game. Let's go a bit further and add the script that controls the flying turtle's movement. Also, by now it becomes important to save the scene frequently (Unity can crash every now and then, and losing the scene you've set up can be frustrating). **File-> Save Scene** comes to the rescue. Or you can just spam **CTRL + S** every now and then like I do!

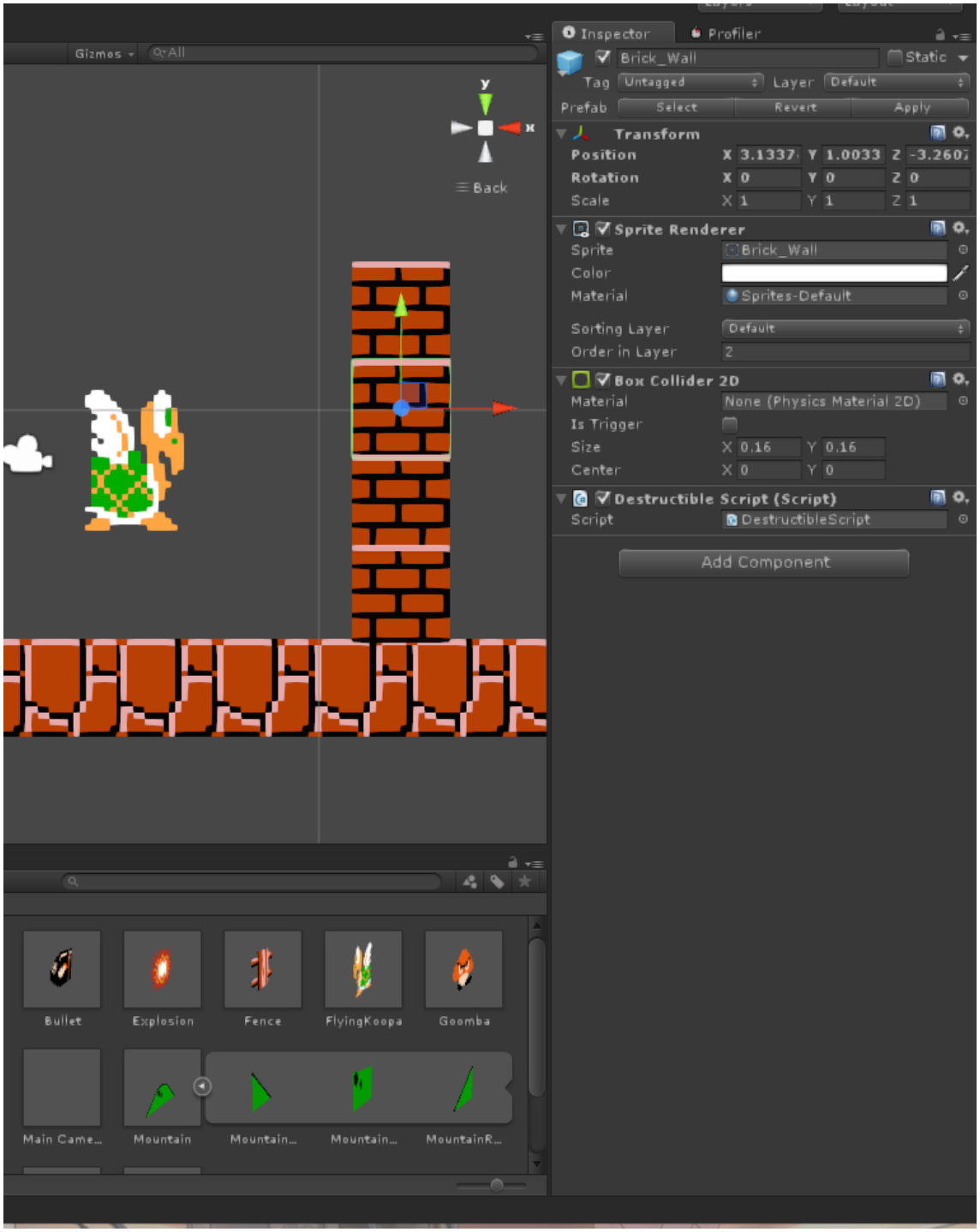
Part II : Start Scripting Enemy A.I.

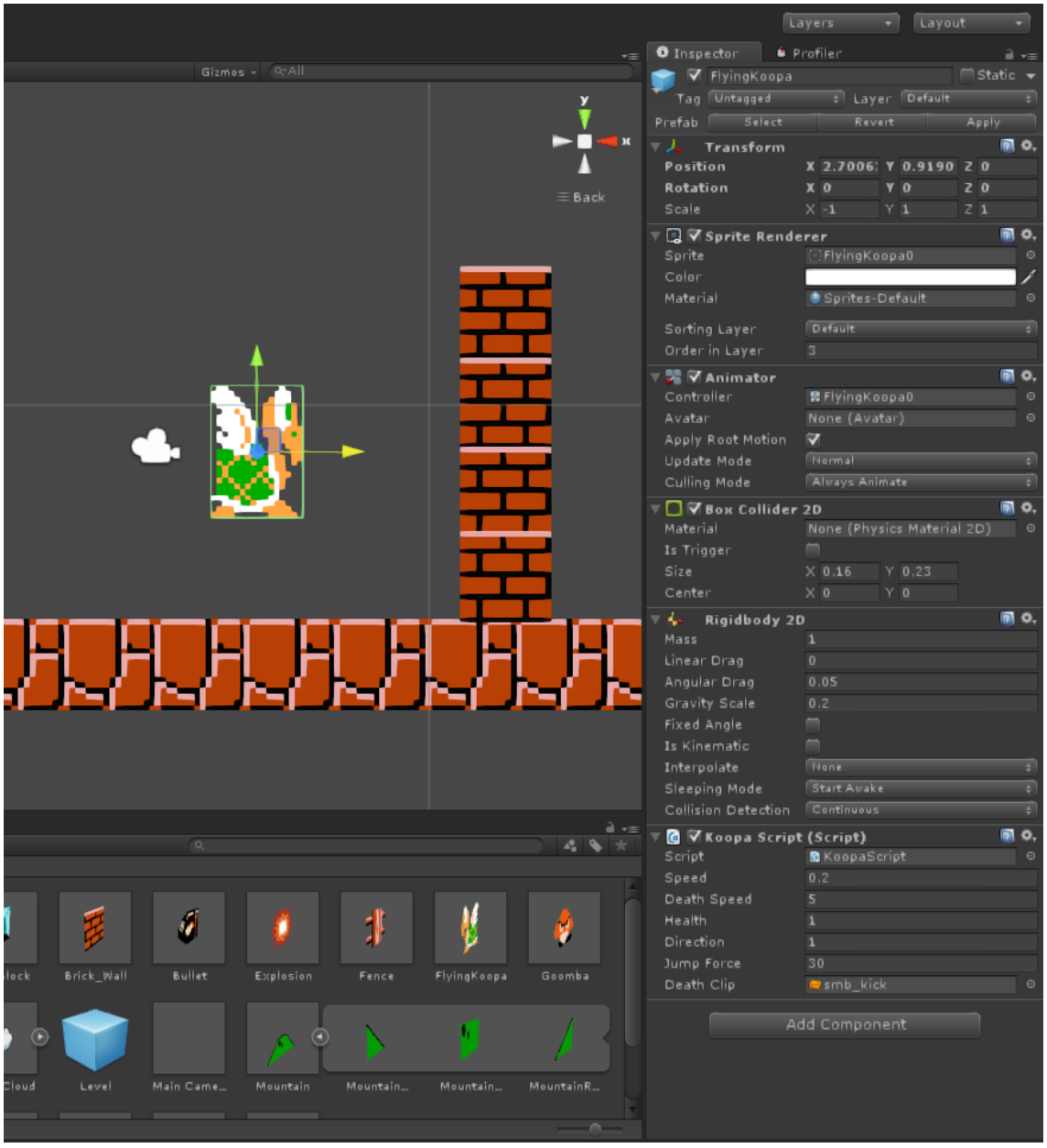
In the script folder there is a script called "**KoopaScript**". We're going to add it to our Flying Koopa. Either simple drag and drop it onto the turtle, or select "**Add Component-> Script**" from the respective Inspector View.



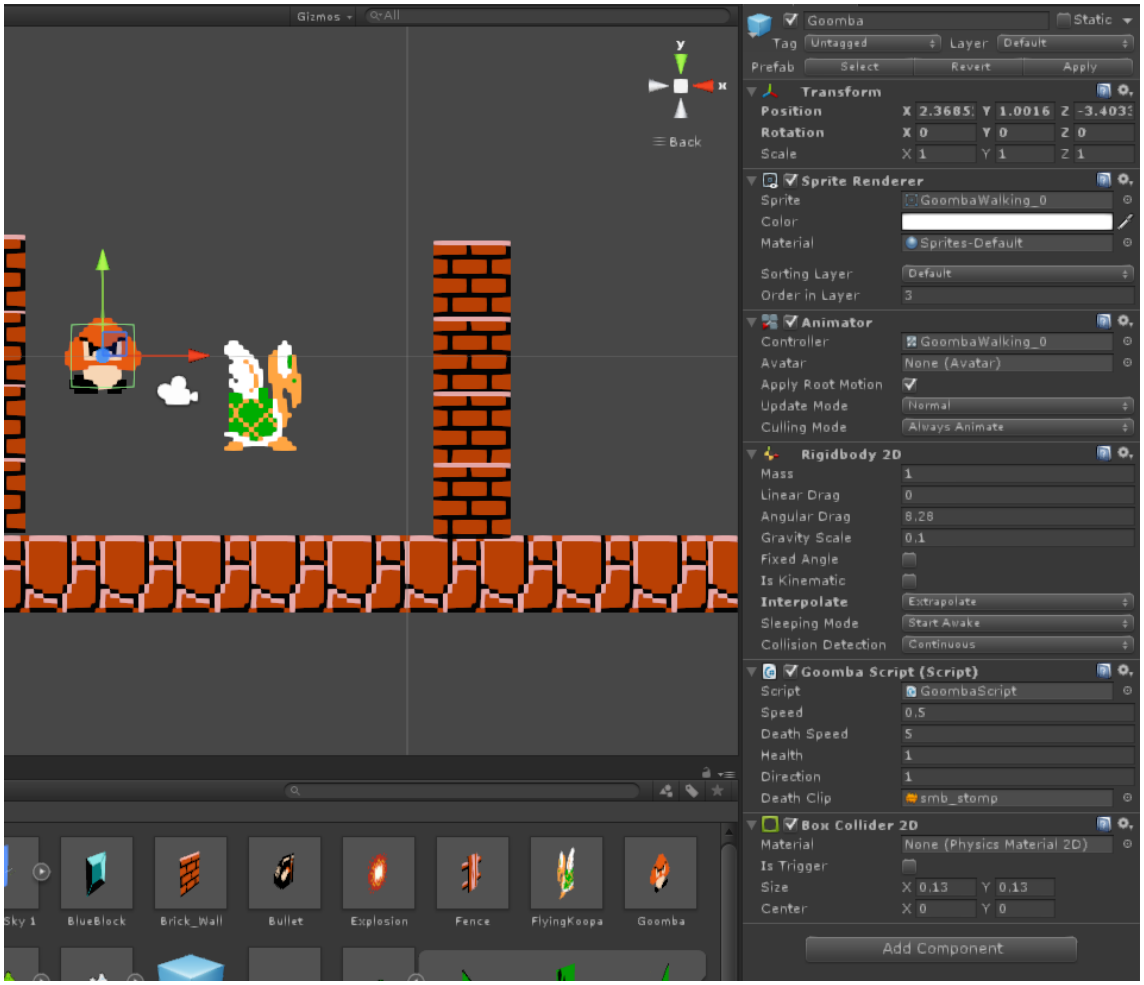
Copy the parameters shown above for the Koopa Script. Also, note the negative X scale in the Transform component. This is to ensure that our sprite is initially facing the X+ axis. Not changing that value will make the game object to move backwards.

Now press play. Your koopa should be happily jumping around. To its doom! Let's add a new type of block, the "Brick Wall". You should now know how to achieve this result with the knowledge you have been given up until now. This is what my Brick_Wall Looks Like:

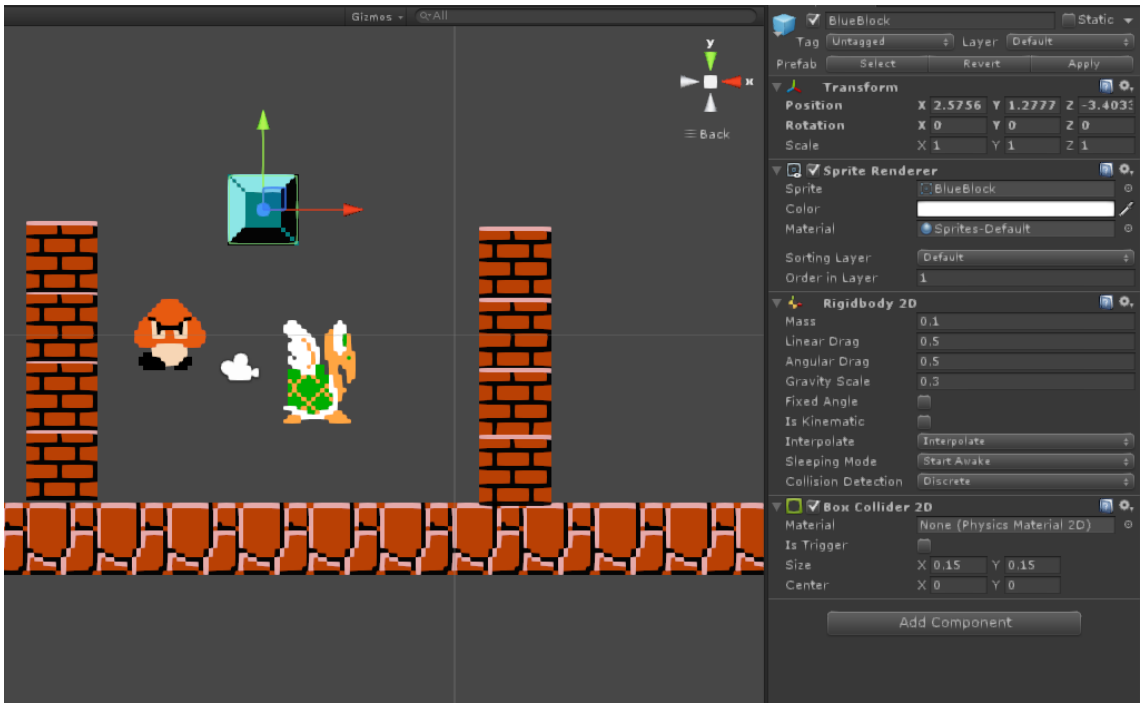

Gizmos
All
Back
Inspector
Profiler
Brick_Wall
Static
Tag
Untagged
Layer
Default
Prefab
Select
Revert
Apply
Transform
Position
Rotation
Scale
Sprite Renderer
Sprite
Brick_Wall
Color
Material
Sprites-Default
Sorting Layer
Default
Order in Layer
2
Box Collider 2D
Material
None (Physics Material 2D)
Is Trigger
Size
Center
Destructible Script (Script)
Script
DestructibleScript
Add Component
Bullet
Explosion
Fence
FlyingKoopa
Goomba
Main Came...
Mountain
Mountain...
Mountain...
MountainR...



Again, create a new prefab for it in your prefab folder. Now you have walls your koopa will bump with and change its direction. Let's add a new enemy, a Goomba. Go to the enemy sprite sheet, select the two goomba walking sprites for walking and create a new animation much like what you did with the Koopa game object. In fact, repeat the same process again, but this time, change the script added to it. Instead of adding the "Koopa Script", add the "GoombaScript". Should look something like this.



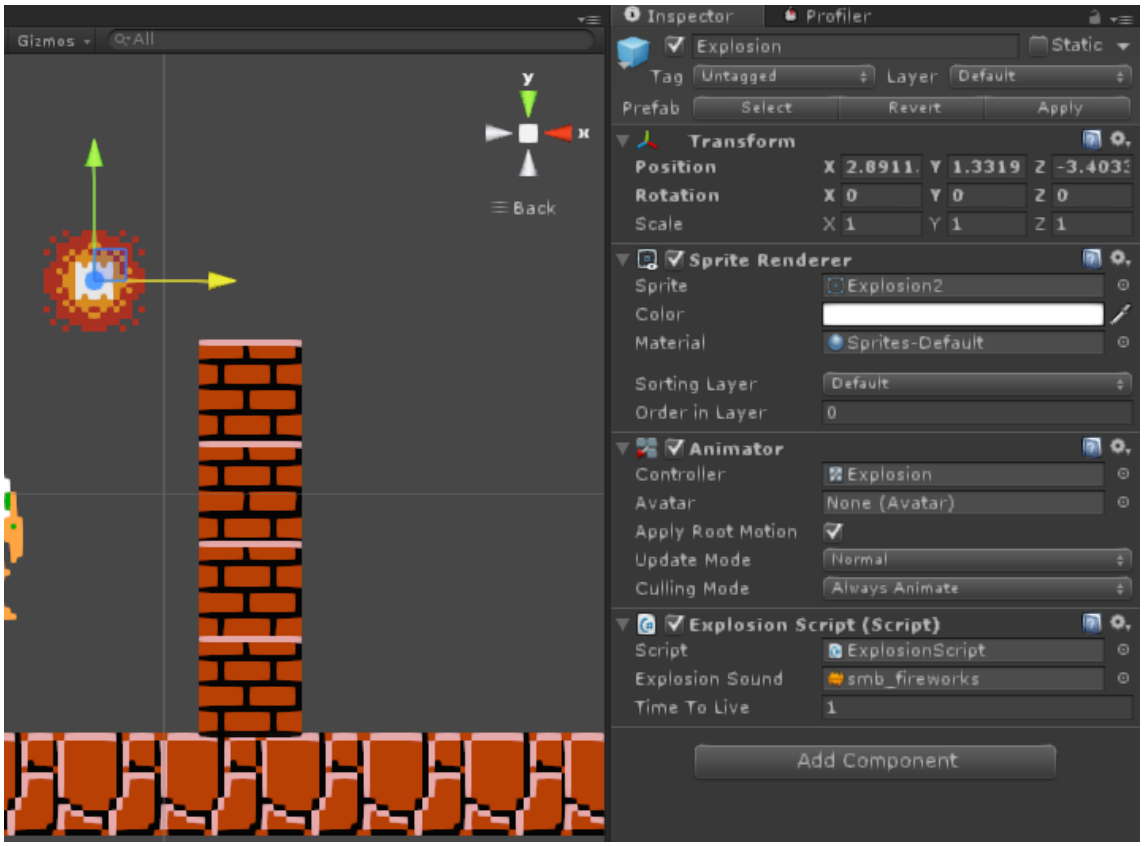
Also, you can add a new type of brick that is influenced by gravity. I created a blue block from the first sprite sheet that is indestructible but is affected by gravity. It looks like this:



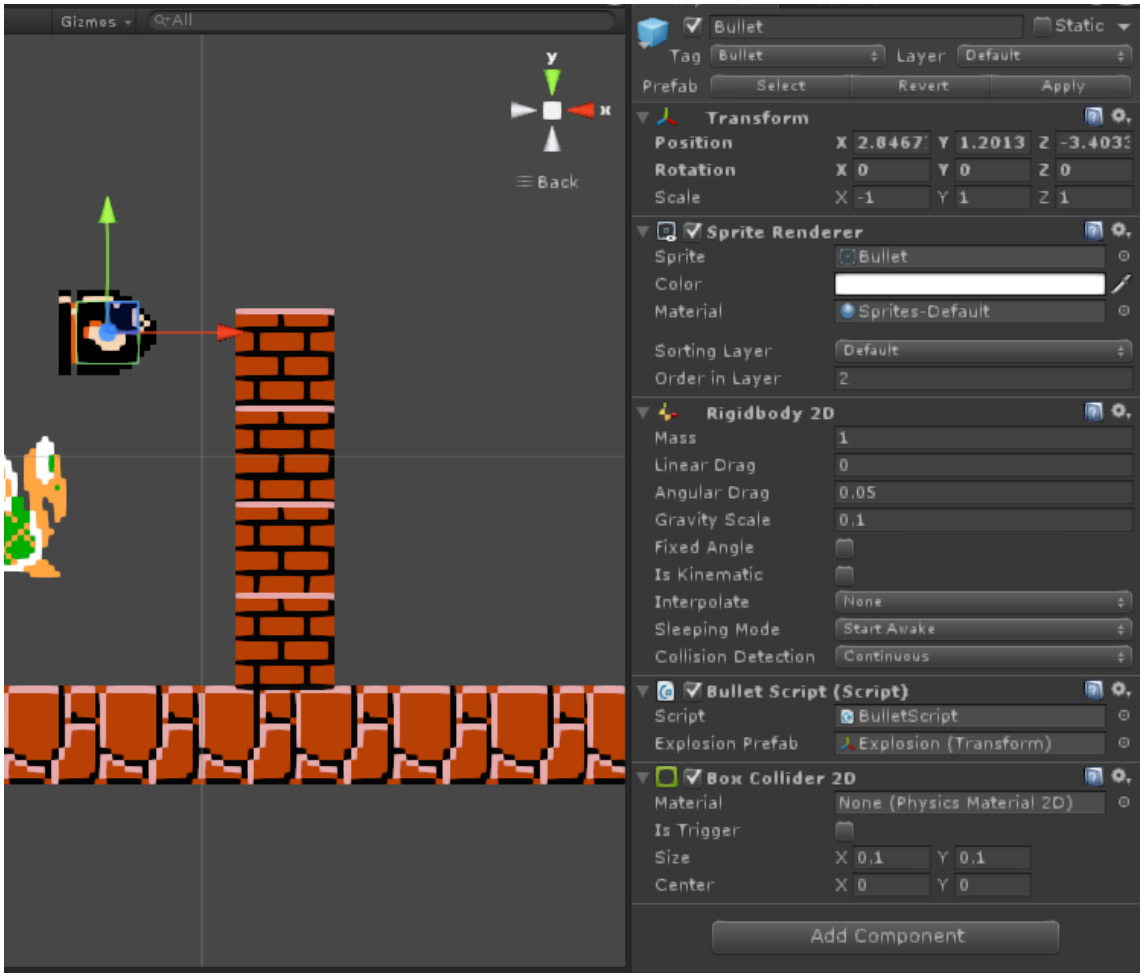
Don’t forget to transform these GameObjects as prefabs! Think of prefabs as “molds”, something you can use for easily replicating or creating similar Game Objects. It’s still not very interactive, is it? Let’s create the Cannon now. So that we can fire against these sprites.

Part III : Fire away!

We need to add what we are firing. Bullets! But first, we have to add Explosions (you will see why when you are creating the bullets). An Explosion is comprised of a 3 sprite animation. It is available in the SMBItems sprite sheet). Do the same as before for creating the Game Object and its animation. In the end, it should look like this.



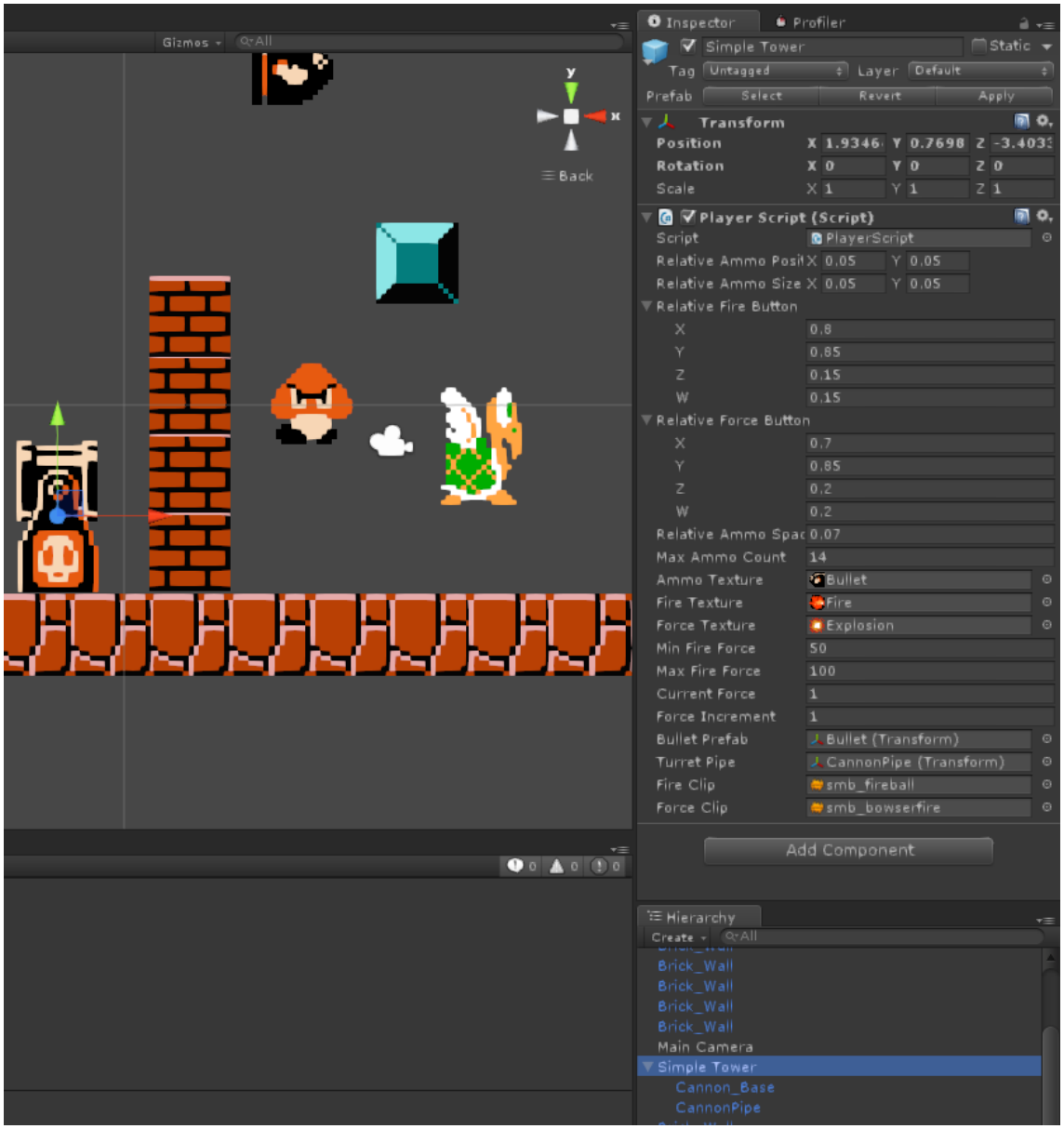
And now we add the bullet. The sprite is available in the SMBenemies sprite sheed. This is what a bullet is for our game:



It doesn't have any animation, it is comprised of a single sprite. A rigidbody and box collider handle the physics. And notice that the "BulletScript" needs a reference to an "Explosion Prefab". Create a prefab for the explosion and refer to it in the "Explosion Prefab" field. This basically means that it will instantiate a single explosion whenever the bullet hits anything. The bullet is destroyed and the explosion follows soon afterward. Also, notice how the camera follows the bullet trajectory horizontally, much like how it happens in Angry birds (You may not notice this effect as the bullet simple falls… for now.

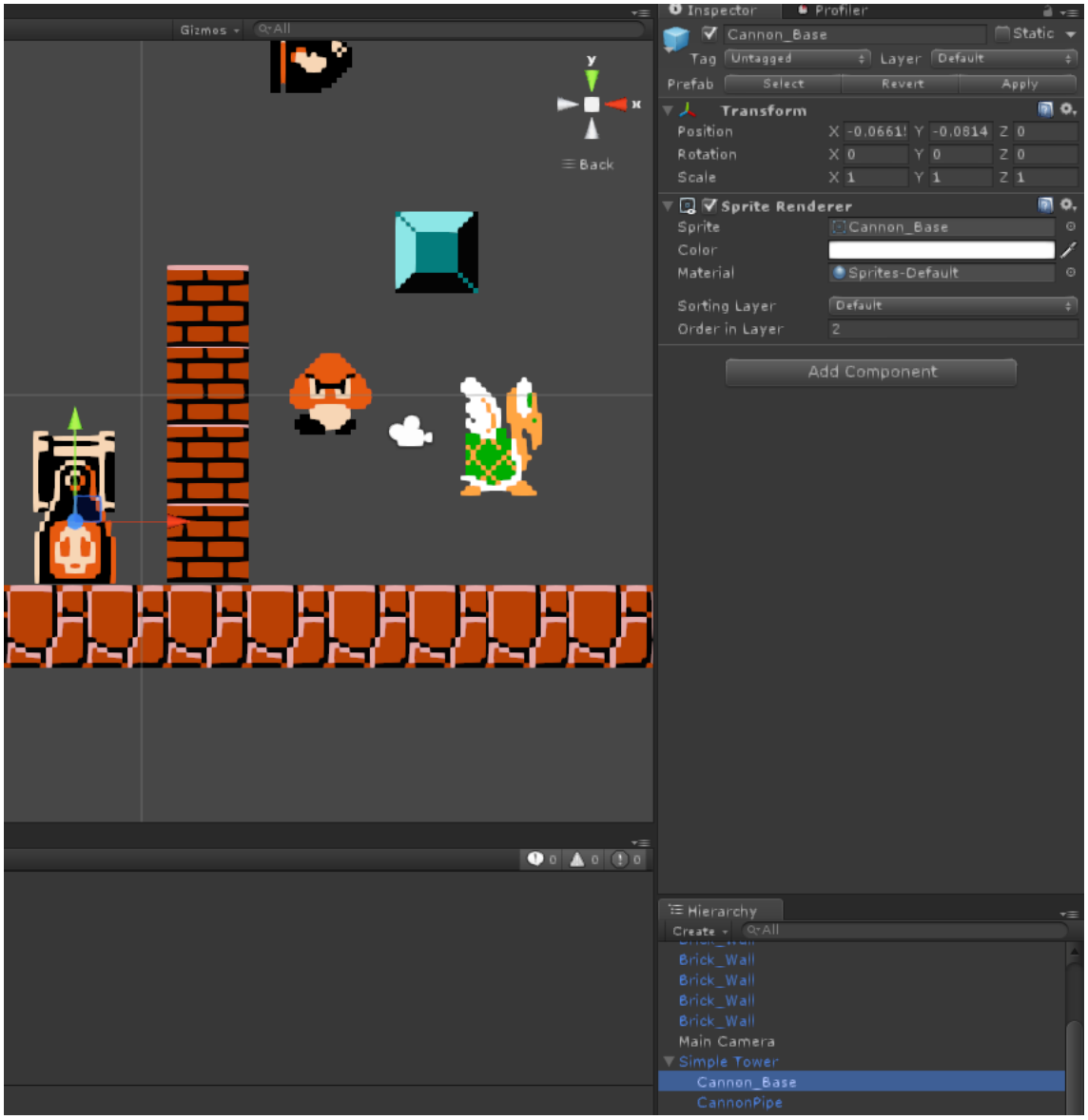
Now, onto creating our simple tower for firing. It is comprised of 3 Game Objects! The first one, Called Simple Tower will be the father of the other two, the Cannon Base and the Cannon Pipe.

Simple Tower (no sprite)



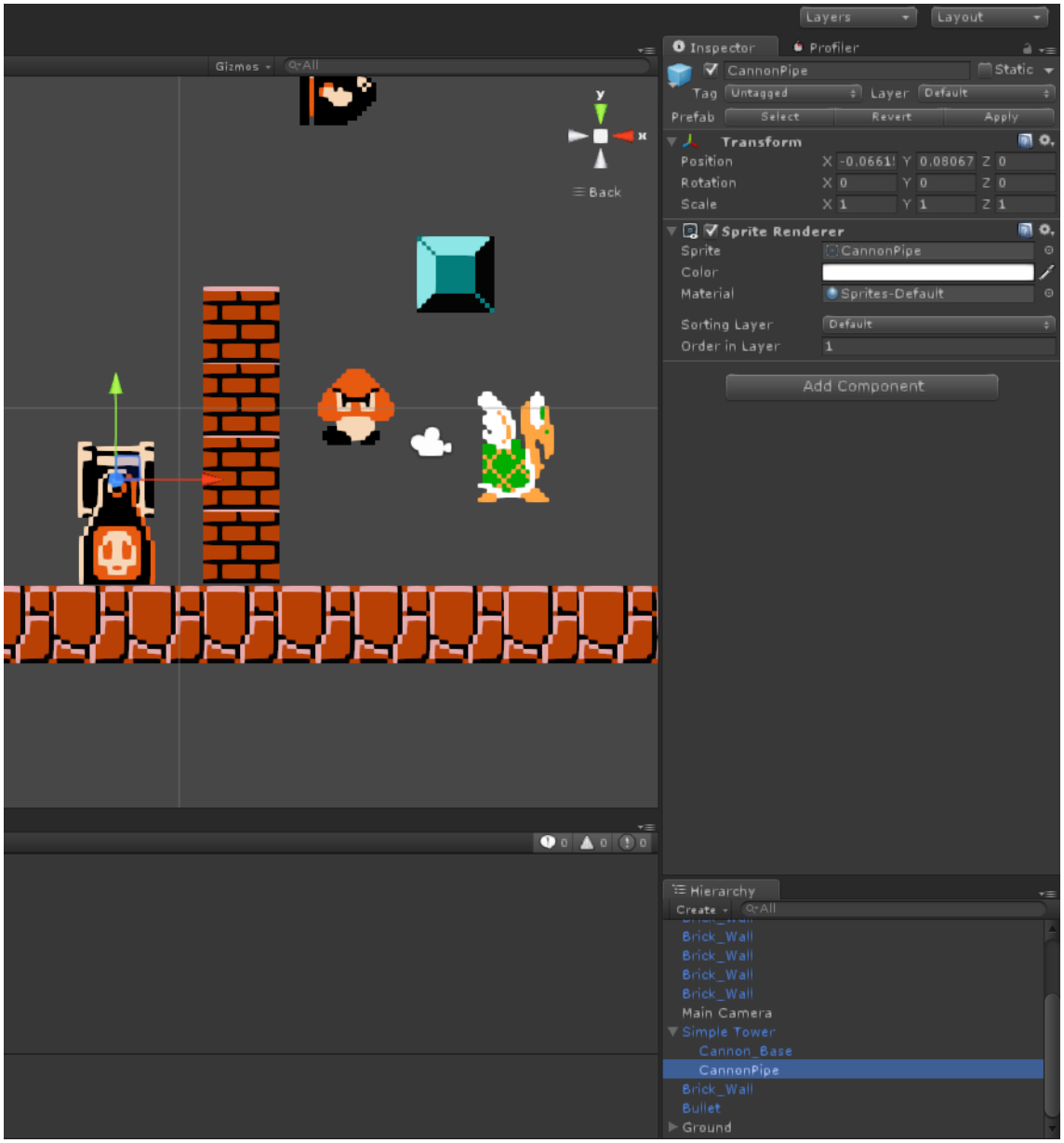
You will need to come back to the “Simple Tower” G.O. after you’ve created the Cannon Pipe G.O. as the field “Turret Pipe” refers to that G.O.. You can change the values of “Min Fire Force”, “Max Fire Force” and “Force Increment” to values that you deem appropriate for the cannon’s fire mechanics (ex: changing the “Force Increment” from 1 to 0.1 can make precise shooting a bit easier).

Cannon_Base:



The Cannon Base is just a simple sprite of the base of the cannon, nothing more.

CannonPipe:

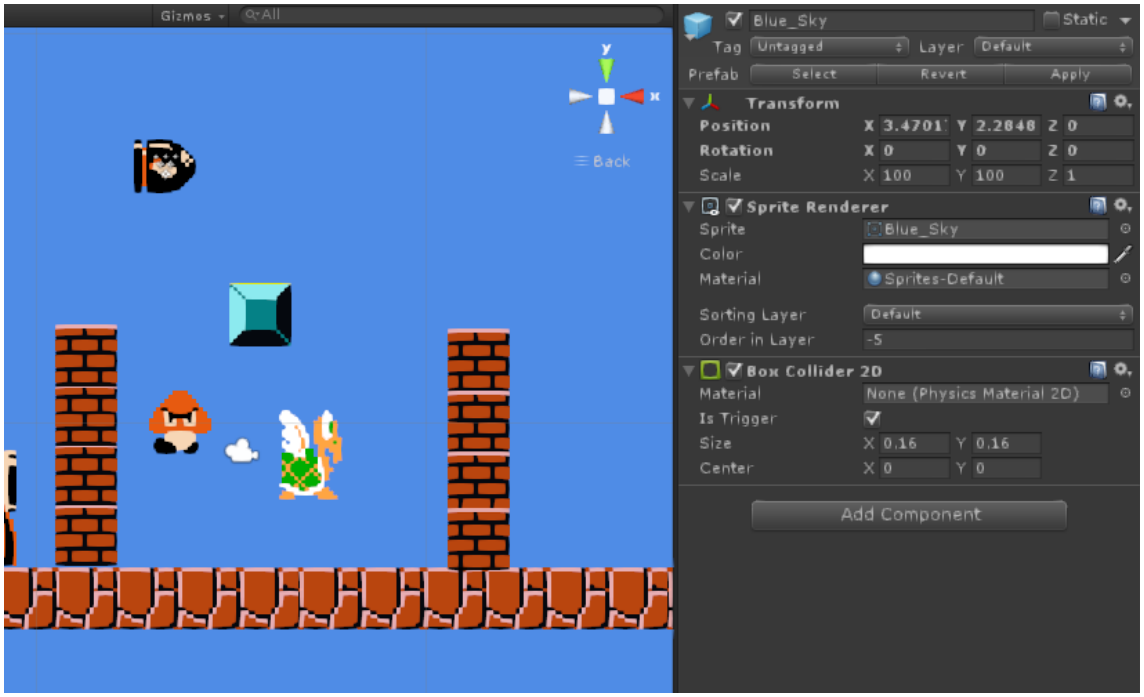


Ditto for the cannon pipe. However, now you can reference to it, in the respective field in the Simple Tower G.O. After doing that, you can create a “Tower Prefab”. And you are now ready to play!

Part IV : Extra Mile

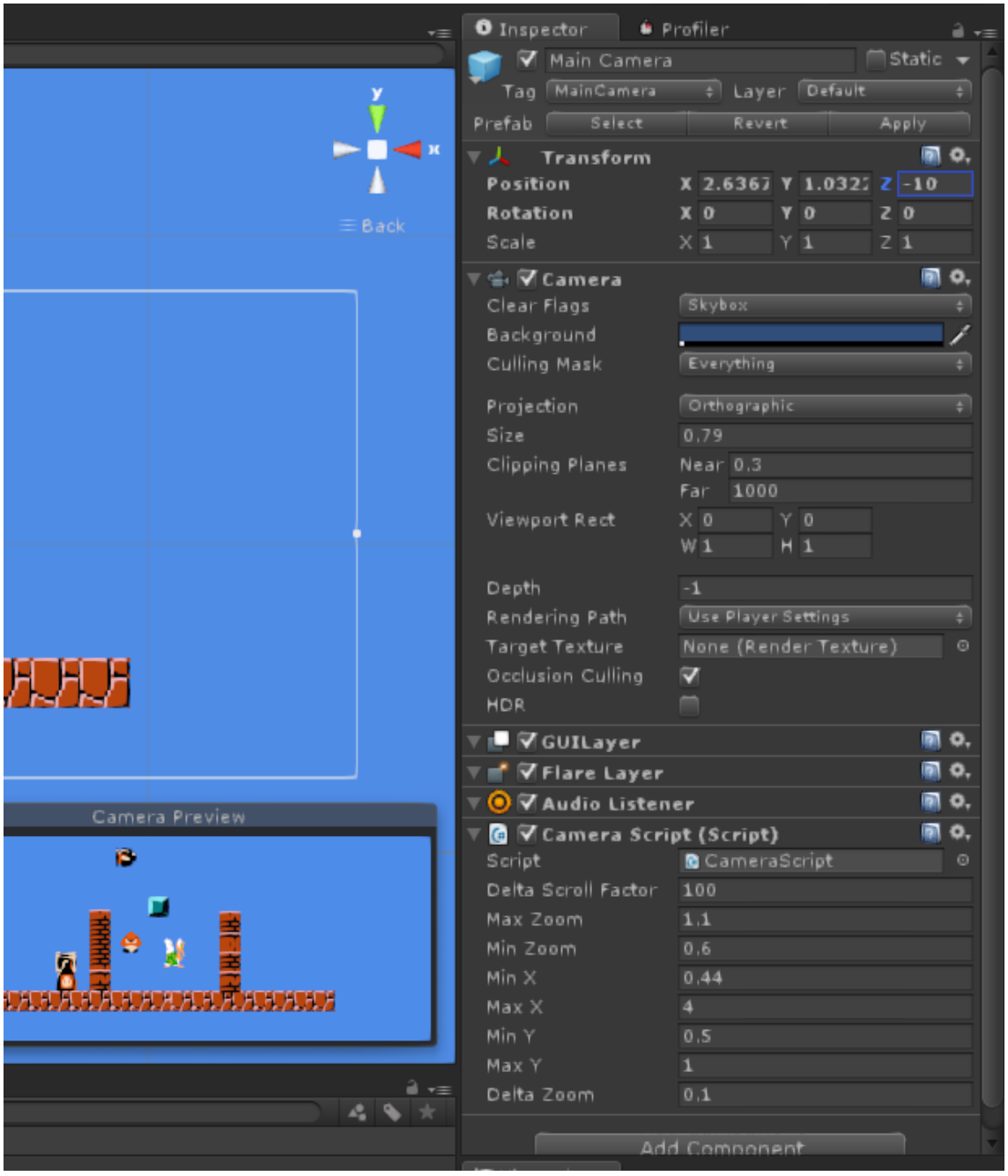
Yes, we have something that is playable, barely. But it is still missing a few things, such as interactive camera movement, level and a few other assets. Let’s start with adding a nice sky.

Select the blue square sprite from the SMB-Tiles sprite sheet. Create a G.O. from it and scale it up quite a bit (100x100) and add a box collider. Additionally, place it in the -5 “Order in Layer”. This ensures that the sprite is drawing first and thus, stays in the background.



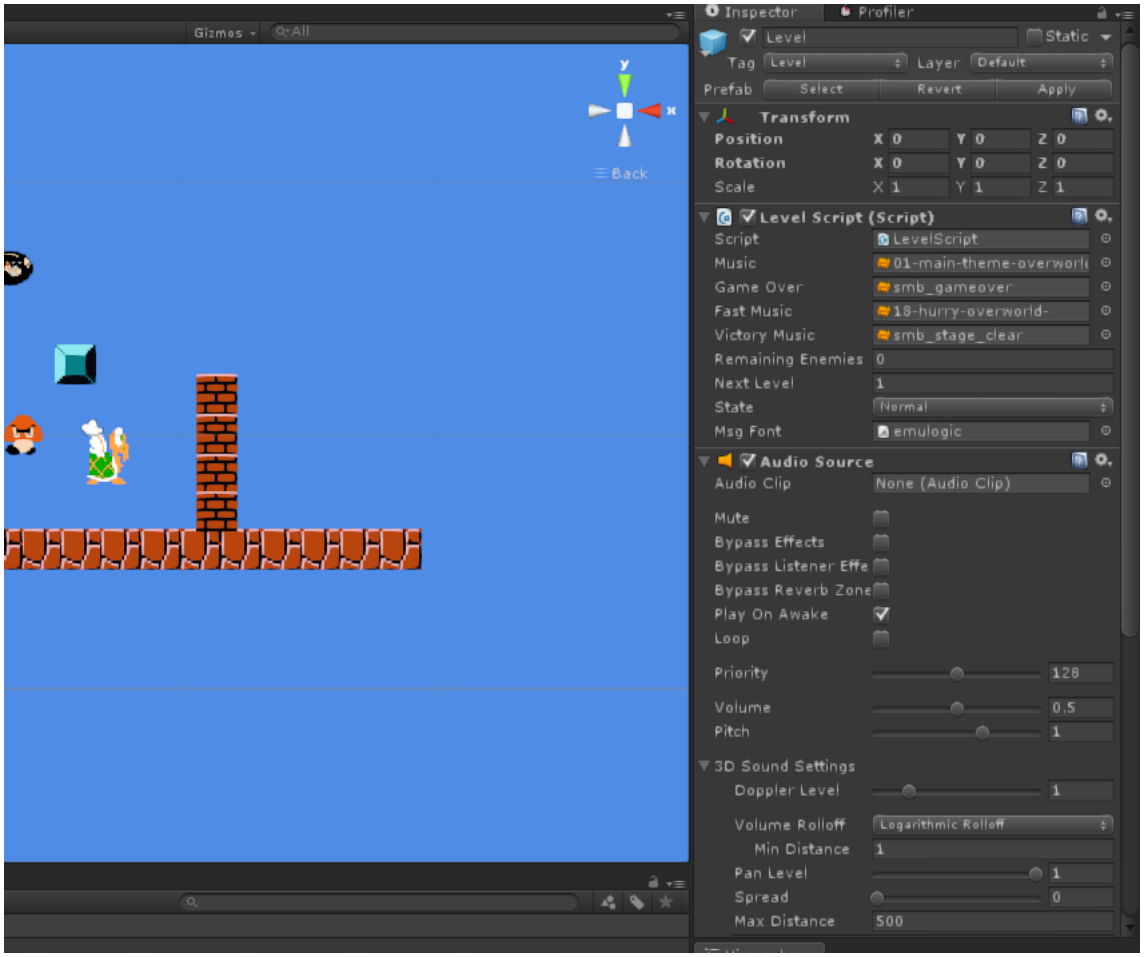
Things are looking up. And since we added a Box Collider 2D as a Trigger (very important!!!) we can also use it to freely aim our cannon (we no longer need to tap specific object to aim at).

The Main Camera also need a bit of a tune up. Add the Camera Script to it like so :



Now, you can “Right Click” to pan the camera around and use the scroll wheel to zoom in and out. Neat!

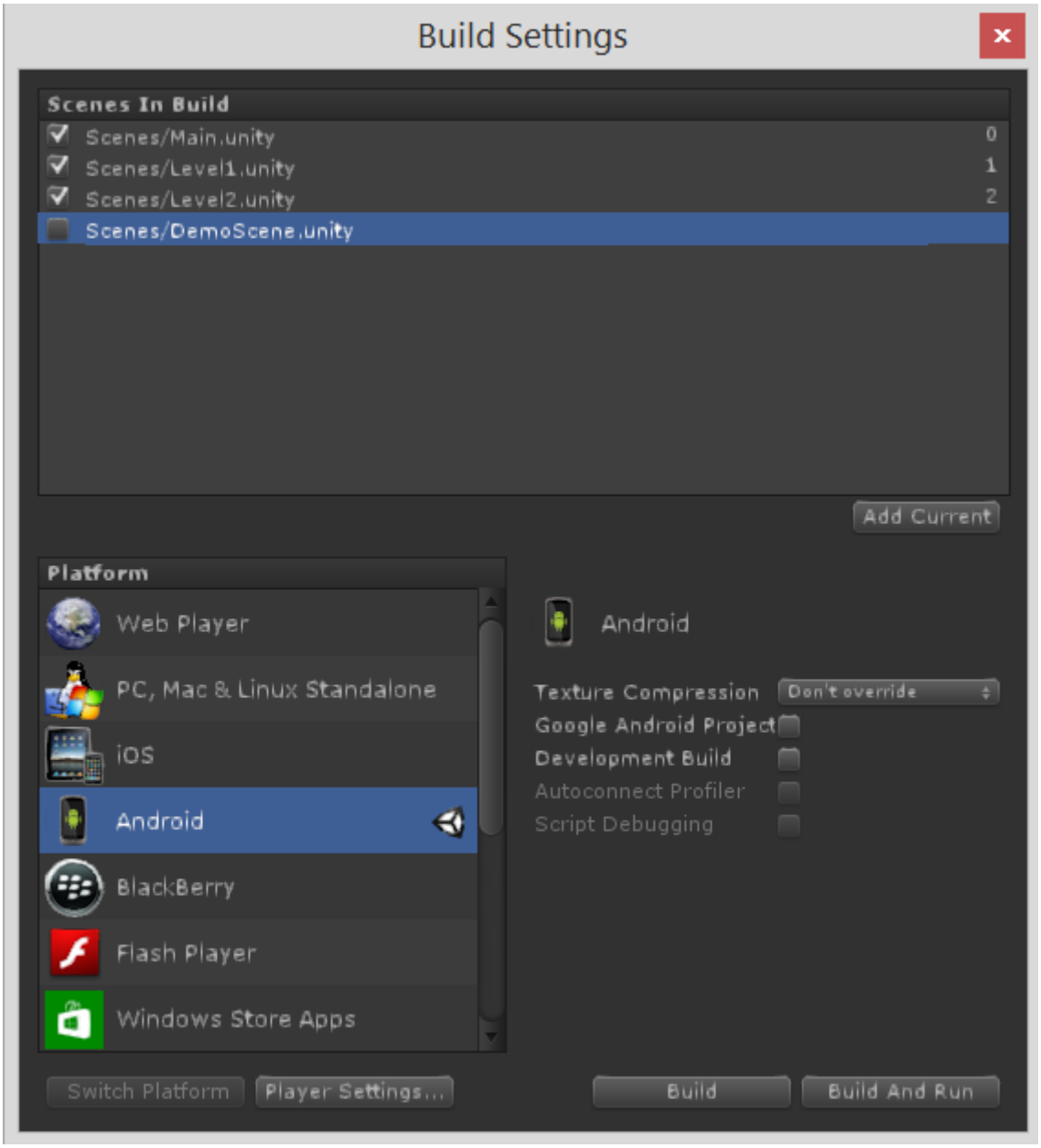
Finally, we create the concept of level. Create an empty G.O. and make it look like so:



It's basically a placeholder for the Level Script that plays the respective musics and keeps track of what level we are currently playing (current level is actually 0, so next level is 1). It also contains and "Audio Source" as it is responsible for playing the music. It will automatically try to advance to the next level whenever all enemies are defeated. It will also reset the level automatically if the player loses all ammo in the process. Note that it contains a new tag, the "Level" tag.

Now, let's add a new level. Save this scene first. And then, select save as and save it as a new scene. You now have 2 scenes (each representing a level) in your game. You can switch from one to another by double clicking it in Unity (In the Project View). Feel free to edit them around (add blocks, enemies, etc) and save them when you are done. For the scene that will act as a second level, don't forget to change the value of "Next Level" by one in the Level Script Component of the respective G.O..

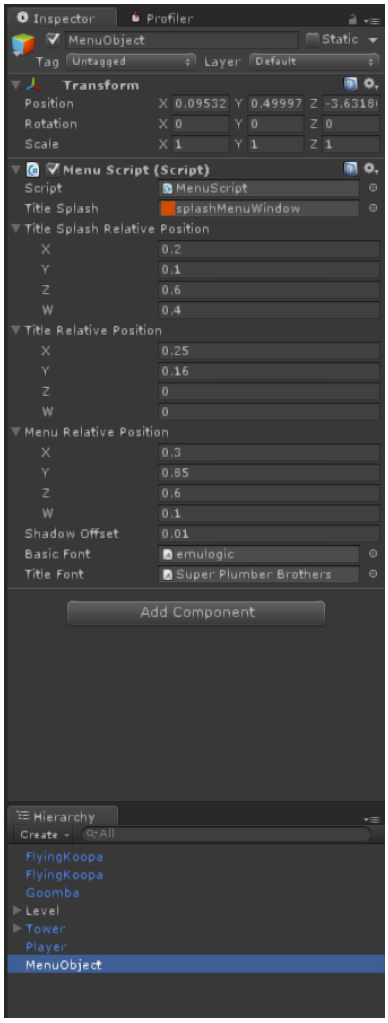
We still need to tell unity what is our default scene and what is the "order" of levels. To do so go to File->Build Settings. Add levels to your liking, but be sure that the order is more or less logic. Also, keep in mind that even though you may freely name your scenes, Unity internally assigns them a number. That is the number we are using in the "Next Level" field above. Unity will open the game in scene 0.



By now you've noticed that you can target your game to different platforms. Your current game should be fully functional in the following platforms:

- Web Player
- PC, Mac & Linux
- Android
- Flash Player

Other platforms will probably result in a somewhat broken or incomplete port. What's left now is a main menu, and adding a few other sprites to give it that extra nostalgia feeling. Rinse and repeat, but keep in mind that we only have two A.I scripts for enemy A.I. behaviour. For the main menu, create a new scene (empty or maybe even a copy of a level!), set it as the default Unity scene and add the following Empty Game Object to it:



Now you have a main menu! By clicking the text that appears on the center of the screen, the menu will attempt to load the scene that is defined as the scene “1” in the Build Settings menu. Now, since you have so many sprites lying around, why not add them to make a complex and beautiful Mario game?

Feel free to check out my full project of this tutorial. It also has a couple of features (such as parallax scrolling) that you might find interesting (and easy to implement via the ParallaxScript!).

Final Thoughts:

If everything went as planned, you should now have a 2D Angry birds clone. You can now add game modes, power ups, and everything you can think of in order to make a truly fun game! This is just a small peek into what Unity has to offer in terms of 2D game features. Also, check out the source code for each script. Most of it is somewhat self-explanatory, and can be useful for you in the future!