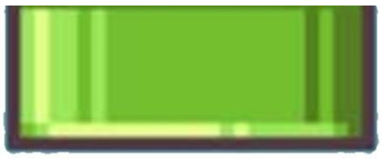
THE

CLONY BIRD MANUAL

AHMADNASER



A Step-by-Step Guide
to Develop A Complete
ClonyBird Unity 2d Game

unity

Clony Bird

A UNITY GUIDE

Authors:

- Rami Asia
- Ahmad A.Naser

THE CLONY BIRD MANUAL

A STEP BY STEP GUIDE TO DEVELOP A COMPLETE UNITY2D CLONY BIRD GAME

Rami Asia

Ahmad A.Naser

Get the unity3d development and design video training

Series here: AhmadNaser.com

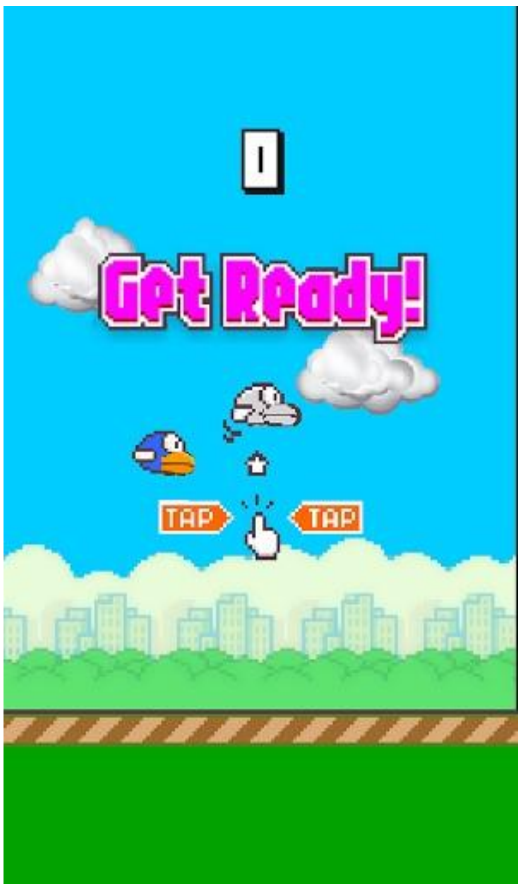
EARNINGS DISCLAIMER

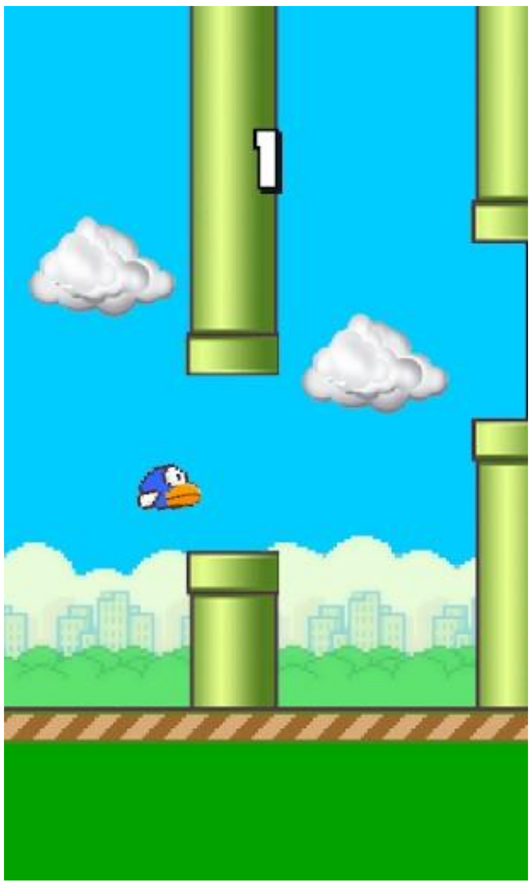
Table of Contents

1. Introduction

Welcome, programmer! Get ready to delve yourself into the vast world of game programming with Unity©. By the end of this document, you should have your very first game built using the Unity engine. The name of the game? Clony Bird.

This game is essentially a Flappy Bird clone. To those not familiar with Flappy Bird, it is a 2D side scrolling game where you are a flying bird soaring through a wall made of pipes with a small gap somewhere in the middle. The objective is to pass through as many pipes as you possibly can.





2. Where do I get started

In order to get all the required tools and software, we organized all the required content in a single page which contains all the links and materials you want, we need the following in order to process forward with this manual:

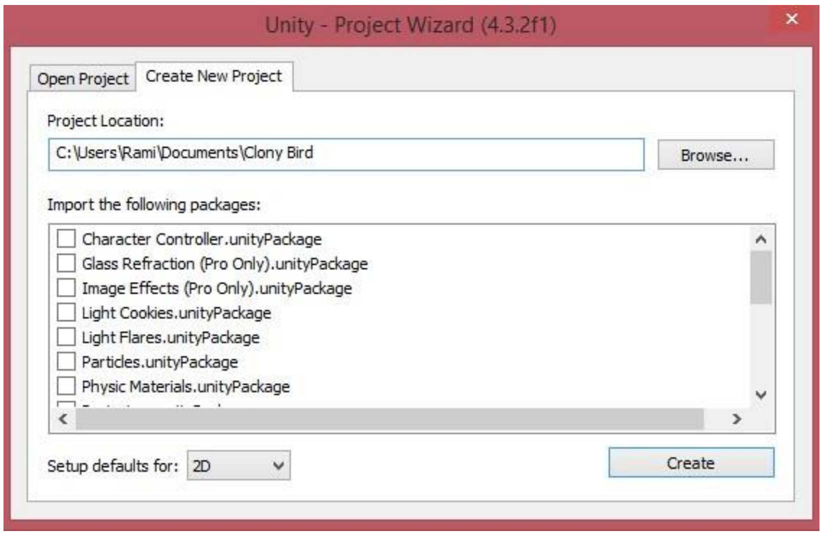
- Unity3d 4.5.0
- Photoshop
- The Sprites of the game(images)

In order to get all these links all what you have to do is visiting this link and follow the instructions:

http://ahmadnaser.com/get-clony-bird-resources/

3. Creating Our Project

To begin our project, we first need to create a new project in the Unity program. In order to do so, open Unity, and click on "File". Then, click on "New Project…" and give your project a name. In this case, we will name our new project "Clony Bird". At the bottom-left of the window, switch the default settings to "2D". Click on the "Create" button.


Unity - Project Wizard (4.3.2f1)

Open Project | Create New Project

Project Location: C:\Users\Rami\Documents\Clony Bird

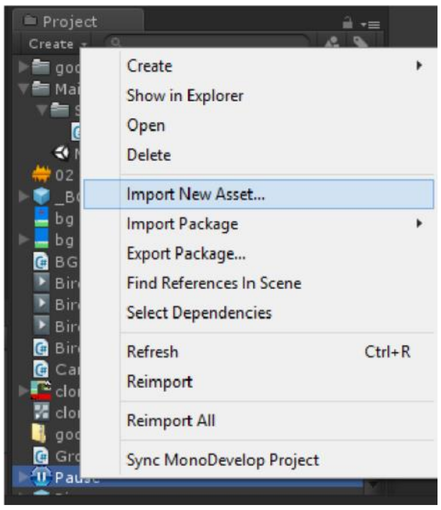
Import the following packages:
- [ ] Character Controller.unityPackage
- [ ] Glass Refraction (Pro Only).unityPackage
- [ ] Image Effects (Pro Only).unityPackage
- [ ] Light Cookies.unityPackage
- [ ] Light Flares.unityPackage
- [ ] Particles.unityPackage
- [ ] Physic Materials.unityPackage

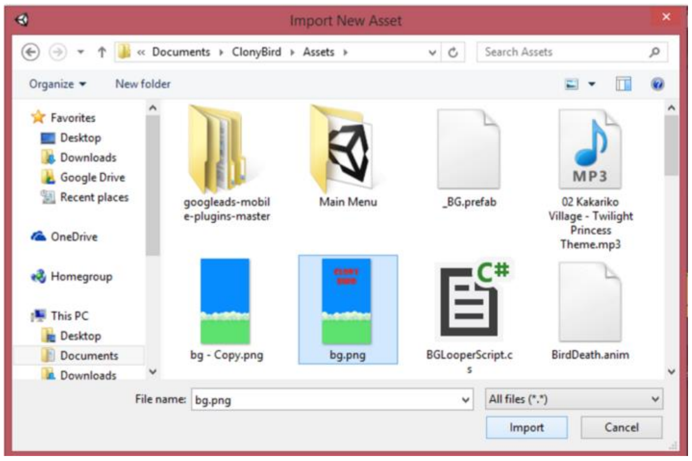
Setup defaults for: 2D

4. Getting Our Sprites Ready

Obtaining Our Sprites

Unsurprisingly, a vital part of any 2D video game is the set of sprites used. The sprites include the scenery, the characters, and everything you see while playing the game. To optimize sprite processing, we will holding all of our sprites in a single PSD image file. The name of this PSD file will be clony_bird_sprites.psd. The PSD file is recognized as an "asset" in our Unity project. As the name implies, an "asset" in our project is something useful or valuable in the creation of our project. In order to add the asset, right click on the area in the "Project" tab and click on "Import New Asset…" Then navigate to the location of the "clony_bird_sprites.psd" and click on the "Import" button.



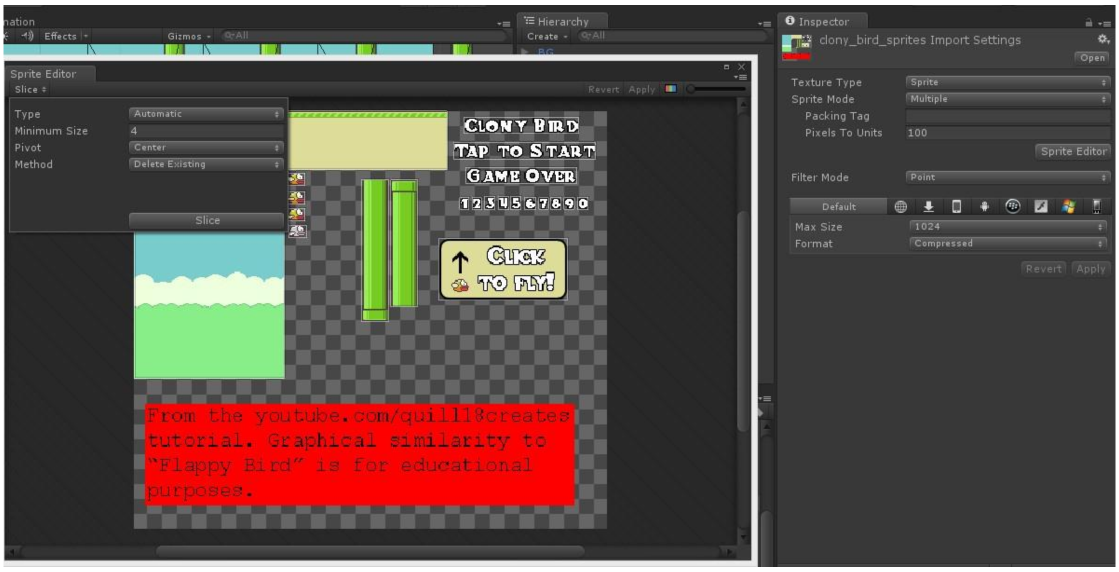

Import New Asset

« Documents › ClonyBird › Assets ›

File name: bg.png

All files (*.*)

Now that we have the asset added, we need to obtain the actual sprites from the file. In order to do this, we click once on the newly added PSD file listed in the "Project" tab. Next, under the "Inspector" tab, click on the button labeled "Sprite Editor". Once the sprite editor opens, you will find a small label on the top-left of the window with the word "Sprite" written. Click on this label and you will receive a small window with options for obtaining the sprites. For our convenience, for "Type" select "Automatic", and then click on "Slice" at the bottom of the menu. The result is that Unity has already divided the image into individual sprites for you! Make any corrections, if necessary, and click on the "Apply" button located at the top-right of the sprite editor. You will now see a list of the individual sprites under the "Project" tab. Viola! We now have our sprites ready for us!



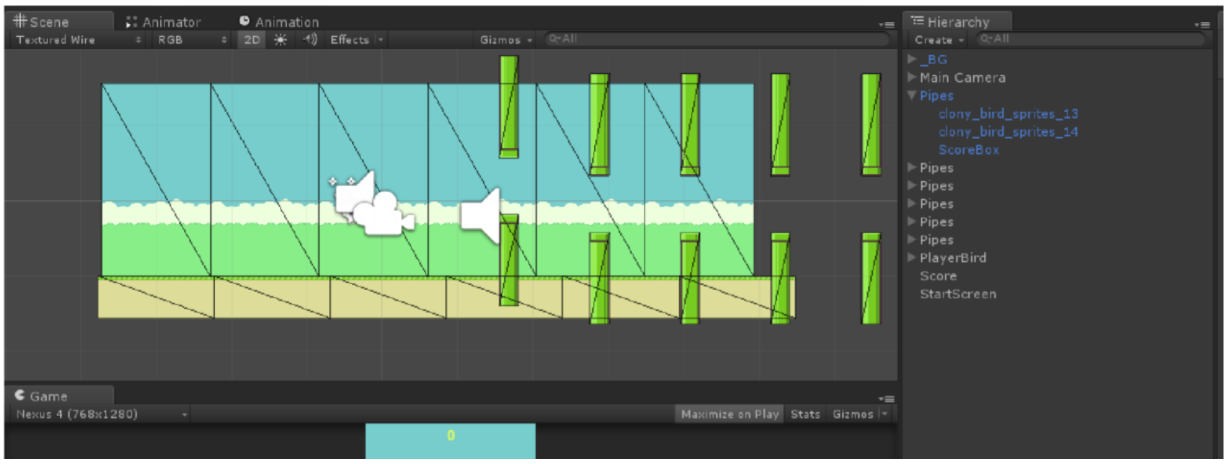
Type: Automatic
Minimum Size: 4
Pivot: Center
Method: Delete Existing

Texture Type: Sprite
Sprite Mode: Multiple
Packing Tag:
Pixels To Units: 100
Filter Mode: Point
Max Size: 1024
Format: Compressed

From the youtube.com/quill18creates tutorial. Graphical similarity to "Flappy Bird" is for educational purposes.

Positioning Our Sprites

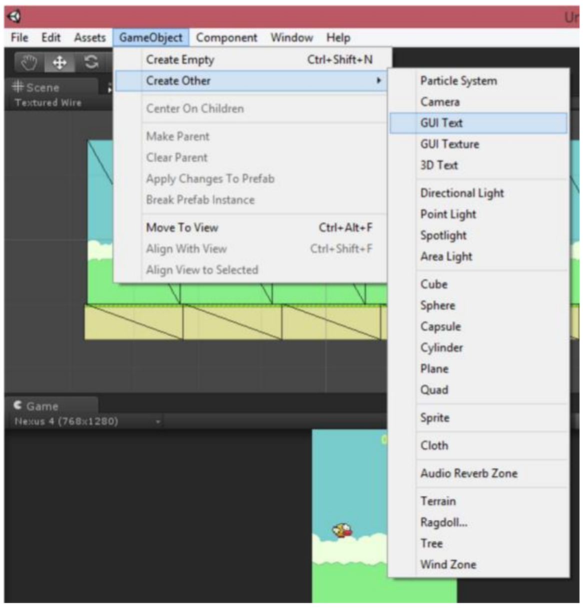
Now that we have our sprites in the "Project" tab, we now want to position them correctly in our scene. If you have not noticed it yet, the "Scene" tab is what holds the scene of our game. Notice how we have four different sprites for our bird. While holding CTRL, select the sprites that are colore d. Drag the selection into the "Scene" tab. Also, drag in the sky background and the ground. Now select the pipes and drag them under the "Scene" tab as well. Position them in a way so there is a vertical gap between them. Now, select both of them, and under the "Inspector" tab on the right, make sure they have the same value for their location on the X-axis. Finally, add the sprite labeled "Click to Fly!" to the scene and position it in a way that has the bird on the sprite exactly behind the actual bird s prite. For now, this is all the positioning we will do for our sprites until we add RigidBodies and BoxColliders, which are both explained later.



_BG
Main Camera
Pipes
clony_bird_sprites_13
clony_bird_sprites_14
ScoreBox
Pipes
Pipes
Pipes
Pipes
Pipes
PlayerBird
Score
StartScreen

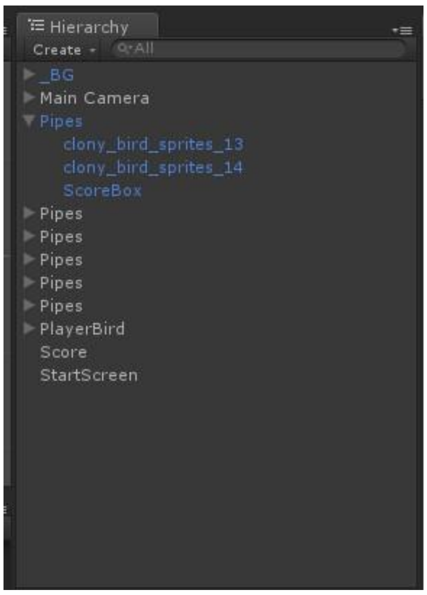
Keeping Score

We need a Text object in our game to show the player what their score is and what the high score is. We will add a new GameObject by clicking "GameObject" on the toolbar at the top of the screen, then selecting "Create Other" and then selecting "GUI Text". Name this GameObject "Score".



Hierarchies

Now that we have our main sprites in the scene, we want to organize our sprites better. Unity's solution for organizing assets is using hierarchies of objects in the game to divide them neatly. We will be dividing our objects into 4 sections: _BG (which holds our background objects), Main Camera (which is responsible for the camera and view of the player), Pipes (which will hold items concerning the handling of the pipes in the game), and PlayerBird (the bird character the player plays as).
In order to create the sections, at the top of the Unity program, select "GameObject" and select "Create Empty". Do this four times and give each new GameObject one of the names we previously mentioned for the sections. Now, in the "Hierarchy" tab, drag the pipe sprite objects into the newly created "Pipes" GameObject. Then drag the background objects into the new "_BG" GameObject. Finally, drag the bird sprite into the "PlayerBird" GameObject. Finally, we are done organizing our objects for now.



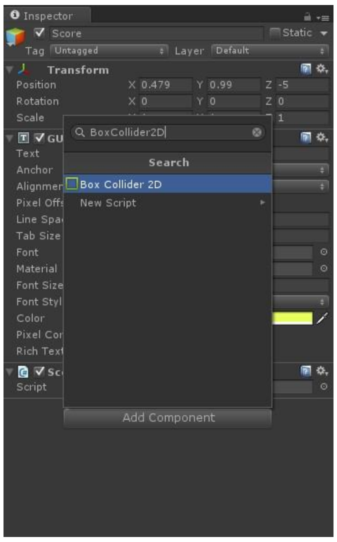
```
▶ _BG
▶ Main Camera
▼ Pipes
    clony_bird_sprites_13
    clony_bird_sprites_14
    ScoreBox
▶ Pipes
▶ Pipes
▶ Pipes
▶ Pipes
▶ Pipes
▶ PlayerBird
  Score
  StartScreen
```

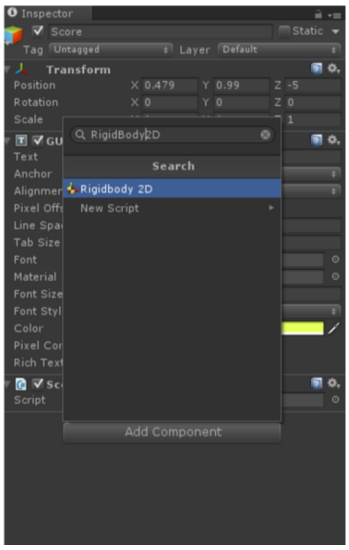
5. BoxColliders and RigidBodies

BoxColliders and Rigidbodies are vital to our game. They are what allow us to interact with the objects in our game by putting them under the effect of the physics engine. To be more precise, BoxColliders allow us to collide into objects and have such collisions handled. RigidBodies allow objects to be fully under the control of the physics engine of the game. We will be using the RigidBody2D and BoxCollider2D GameObjects, as those are more suited for 2D games like those that we are currently developing. Now we are going to add them where necessary.

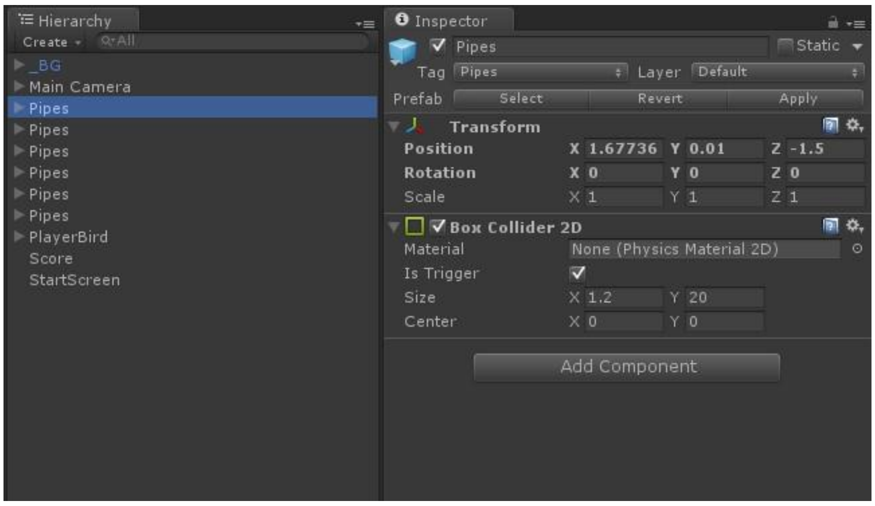
Adding BoxColliders and RigidBodies

The first step towards benefiting from BoxColliders and RigidBodies is adding them to the desired objects. Fortunately, it is incredibly simple to do so. To add either one to a GameObject, just click on the GameObject and then press on "Add Component" under the "Inspector" tab. Now you can simply type in the search bar the desired component you wish to add, and then select it. Type "BoxCollider2D" or "RigidBody2D" to add the desired one to your GameObject.





Now that we know how to add BoxColliders and RigidBodies, let us see what object will be getting each. The sprite for the ground will be getting a BoxCollider2D and a RigidBody2D with the option "is Kinematic" enabled. It is enabled because we do not want the physics engine to control it. The sky sprite will receive a BoxCollider2D with the option "is Trigger" enabled, since we will want this to trigger code later on, and a RigidBody2D with the option "is Kinematic" enabled as well. Our PipesGameObject has a BoxCollider2D that is a trigger, though the pipe sprites in it will have BoxCollider2Ds added that are not triggers. Finally, our PlayerBird will have a BoxCollider2D that is not a trigger and a RigidBody2D that is not kinematic.



```
Hierarchy                         Inspector
Create  All                       ✓ Pipes                          Static
_BG                               Tag Pipes        Layer Default
Main Camera                       Prefab  Select  Revert  Apply
Pipes                             Transform
Pipes                             Position  X 1.67736  Y 0.01  Z -1.5
Pipes                             Rotation  X 0        Y 0     Z 0
Pipes                             Scale     X 1        Y 1     Z 1
Pipes                             ✓ Box Collider 2D
Pipes                             Material    None (Physics Material 2D)
PlayerBird                        Is Trigger  ✓
Score                             Size        X 1.2    Y 20
StartScreen                       Center      X 0      Y 0
                                  Add Component
```

6. Continuing Our Scene

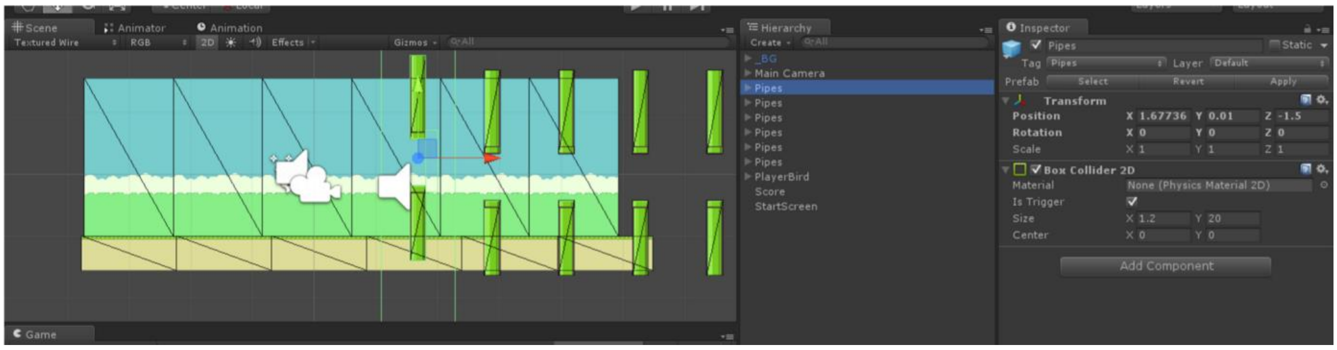
Now that we have most things in place for our game, let us get our scene more developed. We want the background to repeat as we move throughout the game. The way we will accomplish this is by having multiple sky sprites and ground sprites back-to-back and have each sprite reposition itself after we pass it to give off the feeling that we are infinitely scrolling past.

Replicating the Background

The first thing to do is click on the sky sprite, and press CTRL-D. We have now created a copy of the sprite. Now, looking at the BoxCollider2D details under the "Inspector" tab look at how wide our object is. The X-axis of the "Size" field informs us of the width of the object. Add that value to the X value of the "Position" field of the "Transform" section in the "Inspector" tab. You will now have two sky sprites back-to-back. Repeat this on the last sprite until you have six sprites in a line. Then do the same for the ground sprite.

You will also repeat the previous steps for the Pipes GameObject, but before you do that, a few extra steps must be taken. First, click on the Pipes GameObject and take a look at the BoxCollider settings under the Inspector tab. For the Size field, change the Y value to 20. This makes it impossible to miss the BoxCollider. Next, add a new GameObject called "ScoreBox". Resize it to be as tall as the gap between the two pipes, with a width of 0.2. Then, add a BoxCollider2D to it and enable the "is Trigger" option. Finally, replicate the Pipes GameObject just as you did the sky and ground sprites.

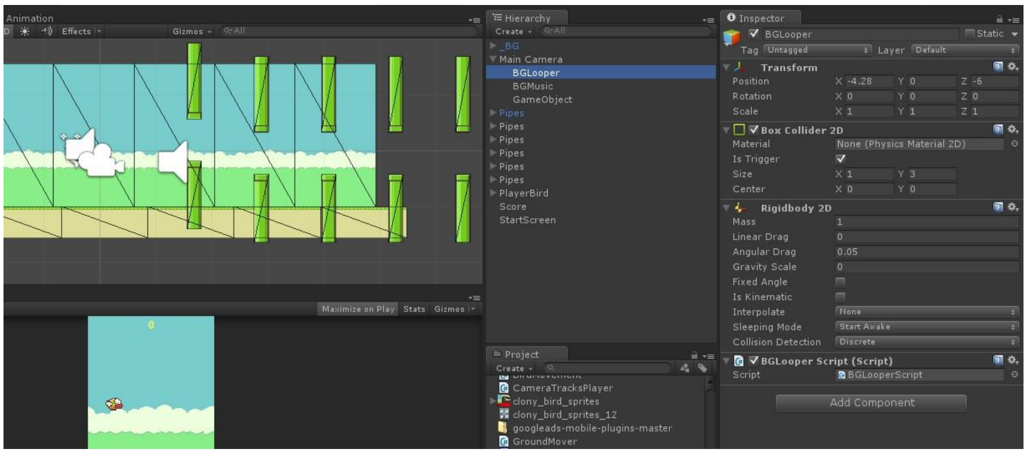
Next, we will want to position our bird. Ideally, put the bird in the middle of the third sky sprite. Then position the Main Camera to have the bird a little to the left of the center. This combination allows us to change the aspect ratio of the game while maintaining the scene. We are getting pretty close to completing our Clony Bird game.



```
Hierarchy                         Inspector
Create  All                       ✓ Pipes                          Static
_BG                               Tag Pipes        Layer Default
Main Camera                       Prefab  Select  Revert  Apply
Pipes                             Transform
Pipes                             Position  X 1.67736  Y 0.01  Z -1.5
Pipes                             Rotation  X 0        Y 0     Z 0
Pipes                             Scale     X 1        Y 1     Z 1
Pipes                             ✓ Box Collider 2D
Pipes                             Material    None (Physics Material 2D)
PlayerBird                        Is Trigger  ✓
Score                             Size        X 1.2    Y 20
StartScreen                       Center      X 0      Y 0
                                  Add Component
```

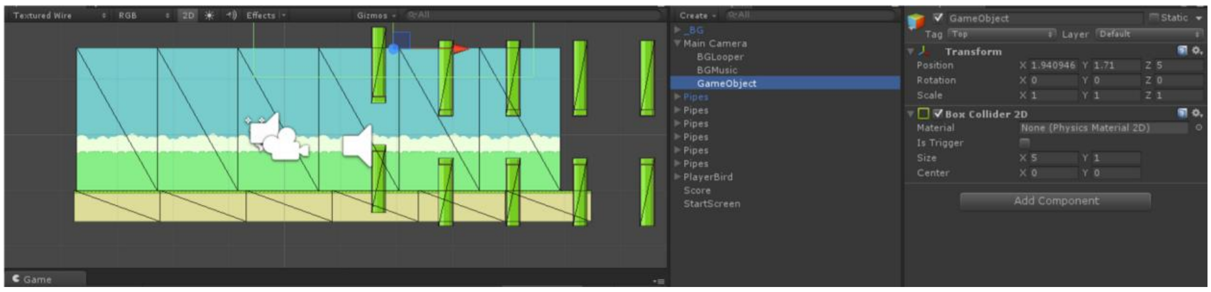
BGLooper

We now need an object that will follow the bird and reposition each background sprite that leaves the view of the Main Camera. For this, we will add a new GameObject called "BGLooper". Once created, position it so that is behind the first ground sprite. Give the new BGLooper a BoxCollider2D with the "is Trigger" option enabled and a RigidBody2D. Now, under the Hierarchy tab, drag the BGLooper object into Main Camera, thus keeping our hierarchy organized.



```
BGLooper                         Static
Tag Untagged        Layer Default
Transform
Position   X -4.28  Y 0  Z -6
Rotation   X 0      Y 0  Z 0
Scale      X 1      Y 1  Z 1
Box Collider 2D
Material   None (Physics Material 2D)
Is Trigger ✓
Size       X 1      Y 3
Center     X 0      Y 0
Rigidbody 2D
Mass               1
Linear Drag        0
Angular Drag       0.05
Gravity Scale      0
Fixed Angle
Is Kinematic
Interpolate        None
Sleeping Mode      Start Awake
Collision Detection Discrete
BGLooper Script (Script)
Script     BGLooperScript
Add Component
```

Next, we will want to add a GameObject that will block the bird from flying over the view of the Main Camera. Add a new GameObject and give it any name. Make it wide and position it over the bird and above the scope of the Main Camera. Now, add a BoxCollider2D to it. It will not be a trigger.



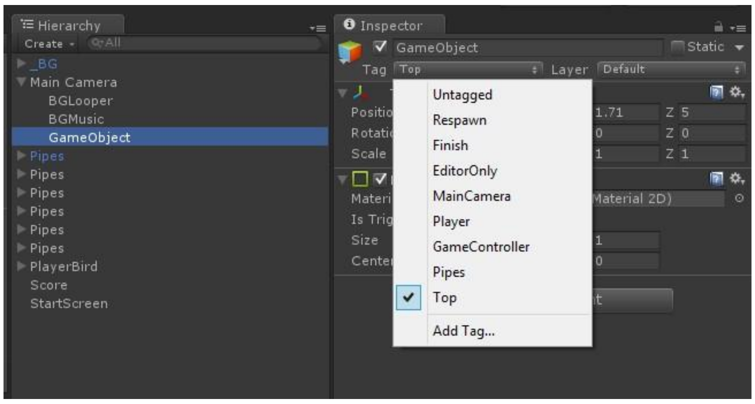
```
GameObject                       Static
Tag Top             Layer Default
Transform
Position   X 1.940946  Y 1.71  Z 5
Rotation   X 0         Y 0     Z 0
Scale      X 1         Y 1     Z 1
Box Collider 2D
Material   None (Physics Material 2D)
Is Trigger
Size       X 5         Y 1
Center     X 0         Y 0
Add Component
```

7. Layering

Now that we have the elements for our game in place in terms of interface, we want to layer things properly so that no conflicts happen. To do this, we will use tags and layers. The tags we will be using for our objects will be "Player" and "Pipes", as only the player object and pipes will require tags in our code later. The layers we will be using are Default, Background, and Player. Background sprites will fall under the Background layer, the bird will go into the Player layer, and everything else will be default.

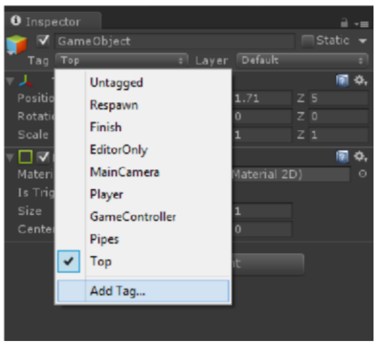
Setting Tags

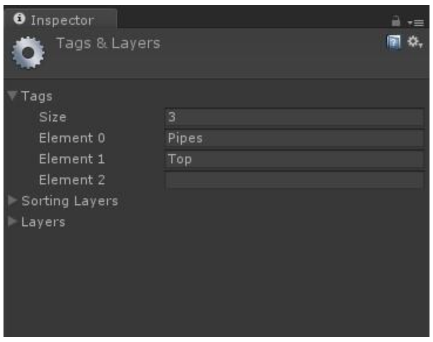
Setting tags for a GameObject is pretty simple. All you need to do is select the GameObject you want to tag, and then look at the Inspector tab. You will see a field labeled "Tag" with options in a dropdown menu. Select your desired tag.



Adding Custom Tags

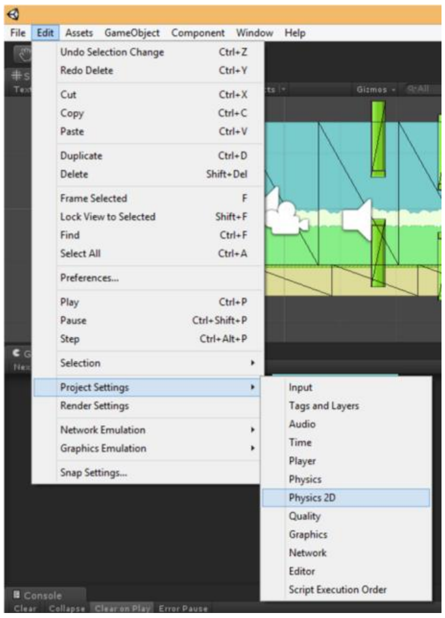
Sometimes you might want to add your own tag that is unavailable in the predefined list of Tags. You can add your own. Repeat all the steps above, but instead of selecting a preset tag, click on the last option, "Add Tag…". The Inpector tab then changes to a new tab with a list of your custom tags. Type in your desired tag and then repeat the steps of adding a tag as normal.

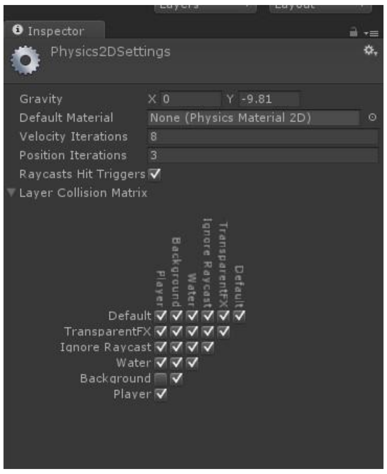




Collision Matrix

We can now decide which layers can collide with other layers, which is definitely convenient. In Clony bird, we want the Player layer to be able to collide with anything except for the Background layer, and vice versa. To accomplish this, we must go to the toolbar located at the top of the screen and click on "Edit", "Project Settings", and then "Physics 2D". Now, where the Inspector tab was located you will find the Physics2DSettings panel open. In the center is the collision matrix, which determines what collides with what. On the Player column, uncheck the Background row.




Inspector

Physics2DSettings

Gravity X 0 Y -9.81
Default Material None (Physics Material 2D)
Velocity Iterations 8
Position Iterations 3
Raycasts Hit Triggers ✓
Layer Collision Matrix

	Player	Background	Water	Ignore Raycast	TransparentFX	Default
Default	✓	✓	✓	✓	✓	✓
TransparentFX	✓	✓	✓	✓	✓	
Ignore Raycast	✓	✓	✓	✓		
Water	✓	✓	✓			
Background	☐	✓				
Player	✓					

8. Animations and Score Keeping

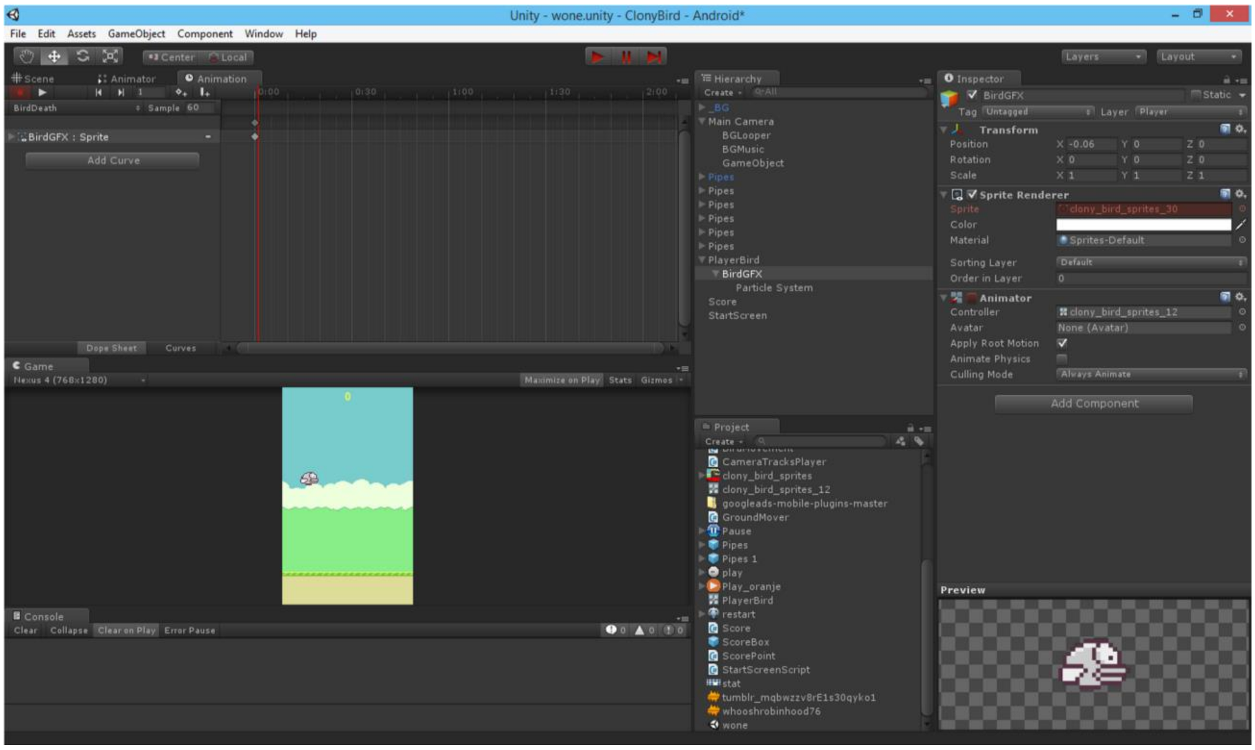
At this point, we have our interface mostly set up. Now, we require animations for our magical bird. As it flies, we want it to flap when we press our button for it to flap and we want it to turn grey when we lose. We also want to keep score as we progress through the game.

The Flying Animation

To add our flying animation, we want to open a tab labeled "Animation". This done by pressing CTRL6 on your keyboard. Now click on the bird sprite in the Hierarchy tab to let the Animation tab know the animations we are going to make will be for that sprite. Now, we already have the flapping animation, but we want the bird to be idle until we press our button for it to flap. We also want the bird to turn gray when we lose.

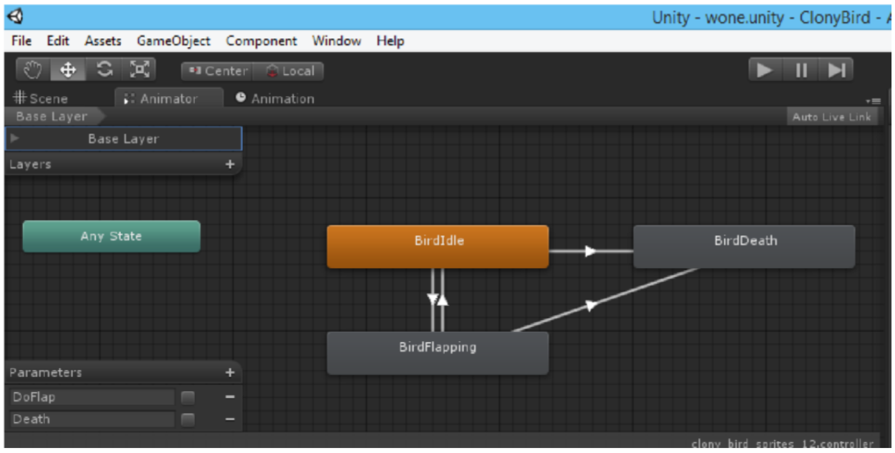
Flap When I Say So

Now we have our Animation tab open, and it knows we want to manage the animations for our bird. Drag the sprite that shows an idle bird in the Project tab to the Animation tab. Now you will see a prompt window to save the new Animation. Name it "BirdIdle". Now, looking at the Animation tab again, under the play button in the animation tab is a list of the animations available. Last in the list is "Create New Clip". Click on this to create our last animation. Now, drag the sprite with a gray bird into the Animation tab. Name this animation "BirdDeath". Rename our first animation to "BirdFlap".



At this point, we have our animations. However, Unity does not know when to use which animation. To remedy this, go to the Animator tab to the left of the Animation tab. Now we see our animations all by each other. Right-click the BirdIdle animation and select "Make Transition". Drag the transition to BirdFlap. At the bottom-left of the Animator tab, there is a small panel titled "Parameters" with a '+' sign by it. Select the '+' sign and select "Trigger" to add a trigger. We will add two: "DoFlap" and "Death". Now click on the white arrow. Representing the transition from BirdIdle to BirdFlap. At the bottom of the Inspector tab, there is a list of conditions for this transition to occur. The condition will be DoFlap, so select DoFlap in the list of possible conditions for the transition. Create a new transition from BirdFlap to BirdIdle with the condition being "Exit". This will have the BirdFlap animation transition to BirdIdle by default after a small duration of time.

Now we will create a transition from BirdIdle to BirdDeath. The condition will be the "Death" trigger. Also, create a transition from BirdFlap to BirdDeath with the same "Death" trigger as the condition. We now have our animations prepared.



9. Coding

Now that we have everything in place, it is time for the most vital part of creating the game: the code that makes it work. In this game, we have seven different scripts written in C#. Those scripts will be named: BirdMovement.cs, CameraTracksPlayer.cs, BGLooperScript.cs, GroundMover.cs, Score.cs, ScorePoint.cs, and StartScreenScript.cs.

Creating and Associating the Scripts

In order to create the scripts, right-click anywhere in the Project tab and select "Create" and then select "C# Script". Do this for each of our scripts. Once that is done, we will want to add each script to the object that needs it as a component. Here is which objects need what script:

- Each sky sprite will need the GroundMover script added
- The MainCamera object requires the CameraTracksPlayer script added
- The BGLooper object requires the BGLooperScript script to be added
- The ScoreBox object will need the ScorePoint script added
- The PlayerBird will need the BirdMovement script added
- The Score text object needs the Score script added
- The StartScreen object will need the StartScreenScript to be added

BirdMovement.cs

In our BirdMovement script, we will require a few parameters. The first is a boolean value called didFlap defaulting at false, which tells us if the bird is flapping. The second parameter is a float value flapspeed, which will hold the speed of each flap the bird flaps. The default value will be 175f. Also declared is an Animator object called animator. This will hold our animation and manipulate it. Another Boolean value will be called "dead", and will default to false. This will denote if we have lost the game or not. Finally, we will have a float parameter called deathCoolDown, which will allow the game to not restart immediately.

When the game starts, it will call the Start() method. All it will do initially is get the animation for us. It should look like this:

```
void Start () {
                animator = GetComponentInChildren<Animator>();
}
```

Next is our Update() method, which handles the graphics and input updates. In this method, we want to set didFlap to true if we pressed on the screen. Also, we want to subtract the deltaTime from deathCoolDown and reload the game when we press the screen. Our code will look like this when done:

```
        void Update() {
    if (dead) {
        deathCooldown -= Time.deltaTime;
        if (Input.GetKeyDown(KeyCode.Space) || Input.GetMouseButtonDown(0)) {
            Application.LoadLevel(Application.loadedLevel);
        }

    }
    if (Input.GetKeyDown(KeyCode.Space) || Input.GetMouseButtonDown(0)) {
        didFlap = true;
    }
}
```

Next up us the FixedUpdate() method. This method is in charge of handling physics updates in the game. It will handle our bird's speed and rotation in the game. We want the flapSpeed to add to the vertical speed of the bird, and have that force added to the bird. We also want the forward speed to be added to the bird's velocity. The code is as follows:

```
void FixedUpdate() {
     if (dead)
         return;
     rigidbody2D.AddForce(Vector2.right * forwardSpeed);
     if (didFlap) {
         rigidbody2D.AddForce(Vector2.up * flapSpeed);
         animator.SetTrigger("DoFlap");
         didFlap = false;
     }
     if (rigidbody2D.velocity.y > 0)
         transform.rotation = Quaternion.Euler(0, 0, 0);
     else {
         float angle = Mathf.Lerp(0, -90, (-rigidbody2D.velocity.y / 3 f));
         transform.rotation = Quaternion.Euler(0, 0, angle);
     }
 }
```

The last method in this script is OnCollisionEnter2D(). This handles situations when the bird collides with anything, which means death. In this method, we want to set the deathCoolDown value to half a second, which is 0.5f. We also want to set the "Death" trigger and set the boolean value of "dead" to true. This is represented by:

```
 void OnCollisionEnter2D(Collision2D collision) {
            if (godMode)
                return;

            animator.SetTrigger("Death");
            dead = true;
            deathCooldown = 0.5 f;
        }
```

BGLooper.cs

Next up in our line of scripts is BGLooper. This script will take care of looping the background repeatedly as we move through the game. We need to define three variables here: an integer numBGPanels, a float pipeMax, which is the maximum height of the Pipe objects, and float pipeMin, which holds the lowest height possible for the Pipe objects. The default value for numBGPanels is six, as we have six copies for each background sprite.

When we start the game, we want to set the pipes in a random fashion. This is done by taking each Pipe object, randomizing the position of its height within the bounds of pipeMax and pipeMin, and assign the new position to the Pipe object. The code is as follows:

```
        void Start() {
    GameObject[] pipes = GameObject.FindGameObjectsWithTag("Pipes");

    foreach(GameObject pipe in pipes) {
        Vector3 pos = pipe.transform.position;
        pos.y = Random.Range(pipeMin, pipeMax);
        pipe.transform.position = pos;
    }
}
```

Our second and last method for this script is OnTriggerEnter2D(), which, as mentioned before, handles when the GameObject the script is a component of collides with something. In the case of our background, we want to handle cases when the BGLooper object collides with the ground objects and Pipe objects. When they collide, the object that collided with BGLooper will be displaced to the right a length six times its width. Pipe heightswill also be changed. The code is as follows:

```
        void OnTriggerEnter2D(Collider2D collider) {
    Debug.Log("Triggered: " + collider.name);
    float widthOfBGObject = ((BoxCollider2D) collider).size.x;
    Vector3 pos = collider.transform.position;

    pos.x += numBGPanels * widthOfBGObject;
    if (collider.tag == "Pipes") {
        pos.y = Random.Range(pipeMin, pipeMax);
    }
    collider.transform.position = pos;
}
```

CameraTracksPlayer.cs

In this script, we handle how the main camera moves along the scene while the game plays. As the game starts, the Start() method only finds the offset between the Main Camera object and the bird sprite. The code is as follows:

```
 void Start() {
            GameObject player_go = GameObject.FindGameObjectWithTag("Player");
            if (player_go == null) {
                Debug.LogError("Couldn't find an object with tag 'Player'!");
                return;
            }
            player = player_go.transform;
            offsetX = transform.position.x - player.position.x;
        }
```

At each update, the Update() method sets the camera to the correct position by assigning its position the value of the player's position with the offset between the objects. The simple code is as follows:

```
void Update() {
    if (player != null) {
        Vector3 pos = transform.position;
        pos.x = player.position.x + offsetX;
        transform.position = pos;
    }
}
```

Score.cs

Score will keep track of the number of pairs of pipes the player passes by. There are 4 variables in this script: score, which is the score of the player currently, highScore, which is the player’s high score, an instance of the Score class named instance, and a BirdMovement object called bird.

When the program starts, we want to set instance to the current instance of Score. We also want to obtain the BirdMovement instance and assign it to bird, get our recorded high score, and set the current score to 0. The code is as follows:

```
void Start() {
            instance = this;
            GameObject player_go = GameObject.FindGameObjectWithTag("Player");
            bird = player_go.GetComponent < BirdMovement > ();
            score = 0;
            PlayerPrefs.GetInt("highScore", 0);
        }
```

When the game is over we want to set the instance value to null and record our high score. The OnDestroy() method takes care of this.

```
void OnDestroy(){  instance = null;

            PlayerPrefs.SetInt("highScore", highScore);

        }
```

As we update each frame, we want to update the Score GameObject to report the current score and high score:

```
void Update() {
            guiText.text = score + "\n" + highScore;
        }
```

We will also have a public method that will allowother

GameObjects to update the score when necessary.

The high score will also be increased if the score is of

a higher value:

```
public static void AddScore() {
    if (instance.bird.dead)
        return;
    score++;
    if (score > highScore)
        highScore = score;
}
```

ScorePoint.cs

This small script will be used by ScoreBox GameObjects to update the store. The script will increment the current score by one and update the high score if necessary by calling on the AddScore() method in the Score script.

```
void OnTriggerEnter2D(Collider2D collider) {
            if (collider.tag == "Player") {
                Score.AddScore();
            }
        }
```

StartScreenScript.cs

StartScreenScript will handle the start screen that we see when starting the application. It will get the sprite that instructs the user to click to fly. If the SpriteRenderer is inactive, the Start() method will make the sprite visible, and pause the game until initiated. The variable in this script is sawOnce, a boolean value that ells if we can see the starting sprite. The Start() method is as follows:

```
        void Start() {
            if (!sawOnce) {
                GetComponent < SpriteRenderer > ().enabled = true;
                Time.timeScale = 0;
                sawOnce = true;
            }
        }
```

The Update() method will only make sure the initial sprite instructing the user to click to fly is not enabledand hidden as soon as the user starts the game. The code is as follows:

```
void Update() {
            if (Time.timeScale == 0 && (Input.GetKeyDown(KeyCode.Space) ||
                    Input.GetMouseButtonDown(0))) {
                Time.timeScale = 1;
                GetComponent < SpriteRenderer > ().enabled = false;
            }
        }
```

GroundMover.cs

Contrary to the misleading name, the GroundMover script will make the sky move in a fashion thatwill give off a parallax effect when playing the game by making the sky move slower than the ground. The only variable is the RigidBody2D variable “player”. The Start() method will assign the player’s bird object to the value of player, as follows:

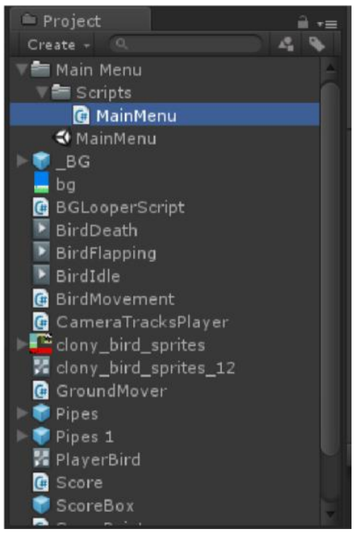
```
        void Start() {
            GameObject player_go = GameObject.FindGameObjectWithTag("Player");
            if (player_go == null) {
                Debug.LogError("Can't find object with tag 'Player'!");
                return;
            }
            player = player_go.rigidbody2D;
        }
```

The FixedUpdate() method will take care of the sky’s velocity relative to the player’s velocity. Thistwoline code is as follows:

```
void FixedUpdate() {
            float vel = player.velocity.x * 0.75 f;
            transform.position = transform.position + Vector3.right * vel *
Time.deltaTime;
        }
```

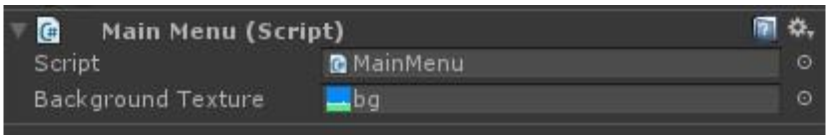
10. Adding a Main Menu

At this point, we now have a functioning game. It is time to add the icing on the cake to make it feel like a finished product. Our first step is to create a new Scene. Title it "MainMenu". Now, under the Projects tab, right click and select "Create" and then click on "C# Script". Name the new script "MainMenu". You should now have a new C# asset named "MainMenu".



Open the script and remove the Update() and FixedUpdate() methods already there. We will now declare a new Texture object called backgroundTexture right at the beginning of the MainMenu class. The declaration will look as follows: public Texture backgroundTexture;

Now, back at the Unity program, drag our new MainMenu script over the Main Camera GameObject. The script has now been added to the MainCamera. If we click on the Main Camera object, we will see our script at the bottom of the Inspector tab. In the small section dedicated to the MAinMenu script, there is an option to select the image to be used as the background texture. You can add any image as an asset and use that image by clicking on the small circle to the right of the Background Texture information section and choosing the image desired as the texture.


Main Menu (Script)
Script MainMenu
Background Texture bg

The OnGUI() Method

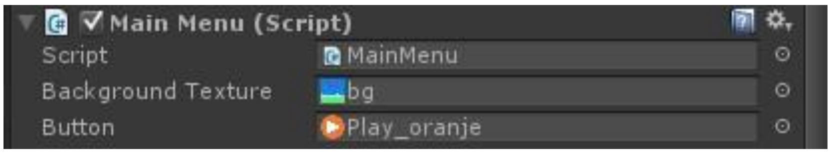
Going back to our MainMenu script, we now want to add the method that starts with our GUI, the OnGUI method. When we star the application, we want the image we specified as the background texture earlier to be displayed. The method that draws the texture is called "GUI.DrawTexture", which takes the shape to draw, and the texture it is to draw. Our code will look as follows:

```
void OnGUI(){
        GUI.DrawTexture(new Rect( 0 , 0 , Screen.width, Screen.height),
backgroundTexture);
}
```

Now that we have our background set up, we now want to add the button that will start the game. First, we must import an image to be used as the texture for a button. Import any image you would like to use as an asset. Then go back to our MainMenu script and in the beginning of the class, declare the Texture object that will be our image for the button. The declaration for a Texture object (which we will call "button") is written:

public Texture button;

After declaring the button, we need to specify which texture we want to use as our texture. Return to the Unity window. Under our MainMenu script in the Inspector tab, we should see a variable labeled "Button". Drag the image you want to use for the button over to that variable under MainMenu in the Inspector tab. Now Unity knows that the texture is supposed to use that image.


Main Menu (Script)
Script: MainMenu
Background Texture: bg
Button: Play_oranje

Creating the Button to Start Playing

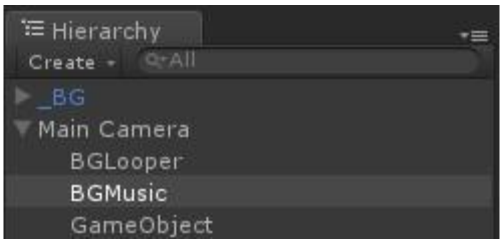
Finally, we have our texture and background image set up for our main menu. All that is left to complete the main menu is to have our image actual start the game when pressed. We do this by creating a Button object and pass a Rect (rectangle) object to draw out, with the "button" texture we created earlier as the image to use for our button, and an empty string. When our button is pressed, we want to load the scene we created with the actual game in it. This is done by going back to our MainMenu script, and adding the following code to our OnGUI() method:

```
if(GUI.Button(new Rect(Screen.width*0.35f, Screen.height*0.45f, Screen.width*0.3f,
Screen.height*0.3f),button ,"")){
               Application.LoadLevel("wone");
        }
```

Note: "wone" is the name of the scene where I created the level of the game.

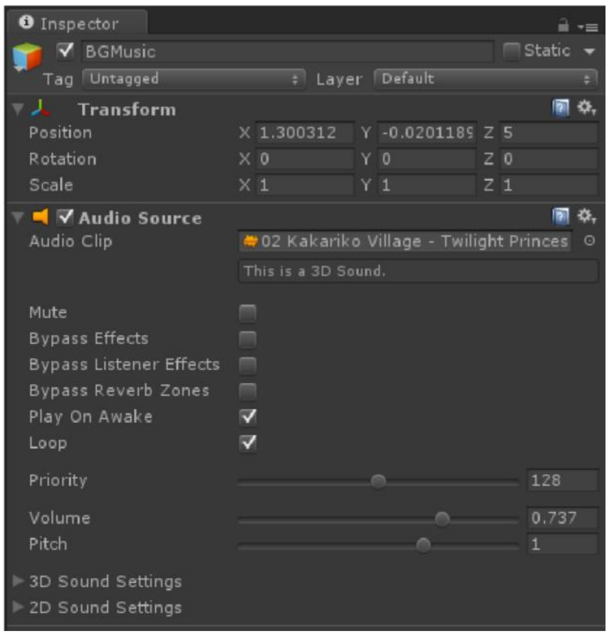
11. Adding Background Music

To give the users a better playing experience, we want them to delve into the game more emotionally. Music is a tool that can help accomplish that. We start by adding a new empty GameObject called "BGMusic". Drag the BGMusic object and drop it under the "Main Camera" hierarchy.



Select our BGMusic object under the Hierarchy tab and click on "Add Component" under the Inspector tab. We now want to add an Audio Source. At this point, we have an AudioSource component with our BGMusic object. You will find a few options in the AudioSource section of the Inspector tab. Put a check next to the "Loop" option to have our song loop when finished.

Right-Click anywhere under the Project tab and select "Import New Asset". Now add the song you would like to play in the background. Now, once the asset is added, click on the BGMusic object under Main Camera. Drag the newly added audio clip asset and drop it on the "Audio Clip" field of the "Audio Source" section under the Inspector tab. We now have background music! Apply the same steps to the Main Menu if you desire for background music there as well.


Inspector
BGMusic — Static
Tag: Untagged — Layer: Default

Transform
Position: X 1.300312, Y -0.0201189, Z 5
Rotation: X 0, Y 0, Z 0
Scale: X 1, Y 1, Z 1

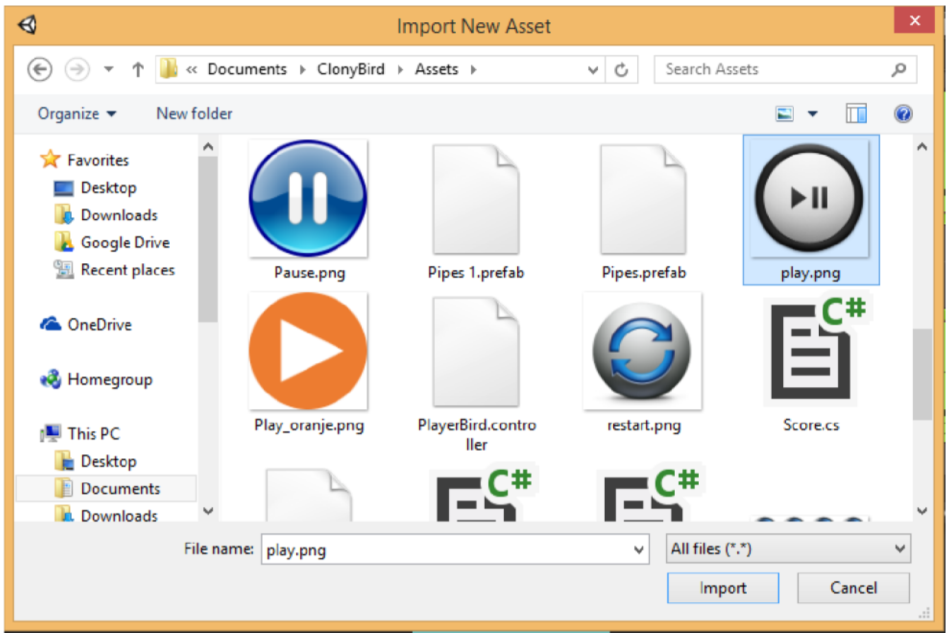
Audio Source
Audio Clip: 02 Kakariko Village - Twilight Princes
This is a 3D Sound.
Mute: unchecked
Bypass Effects: unchecked
Bypass Listener Effects: unchecked
Bypass Reverb Zones: unchecked
Play On Awake: ✓
Loop: ✓
Priority: 128
Volume: 0.737
Pitch: 1
3D Sound Settings
2D Sound Settings

12. Adding Pause and Reset Buttons, Sound Effects, and Particle Systems

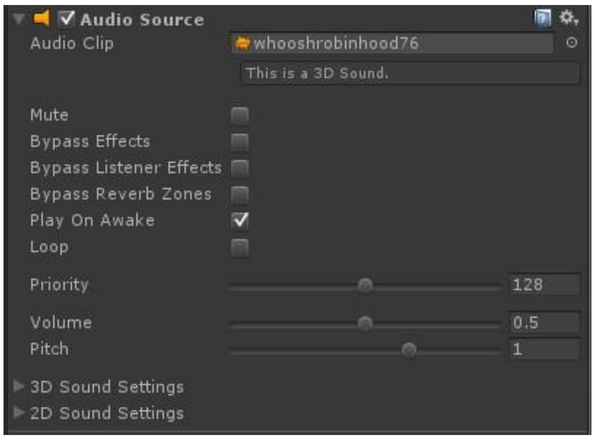
We will now be exploring how to add a Pause button to pause the game, a Reset button to reset the game and start over, sound effects whenever the bird flaps, and Particle Systems to display some video effect when the screen is touched. This will be divided into three sections: getting our objects in place, the code to implement them, and extra steps to bind the objects and the code.

Getting Our Objects in Place

To get our Reset and Pause buttons set up, the first step is to import the images we will be using as those buttons. Right-click under the Project tab and select "Import New Asset". Import any image you would like to use as your Pause button. Then redo the last step to add the image you will be using as your Reset button.

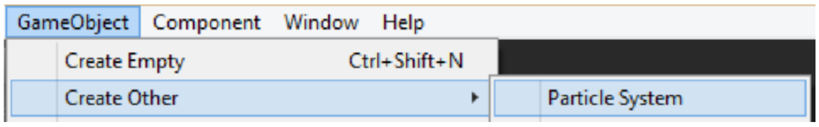


To begin adding our sound effects, click on the "PlayerBird" GameObject under the Hierarchy tab. Now select "Add Component" at the bottom of the Inspector tab. Now select "Audio Source". Now import a new Asset under the Project tab and select the audio clip you want to be your sound effect. Now drag that newly imported sound clip asset to the "Audio Clip" field of the "Audio Source" section under the Inspector tab of the PlayerBird object.



Audio Source
Audio Clip: whooshrobinhood76
This is a 3D Sound.
Mute: ☐
Bypass Effects: ☐
Bypass Listener Effects: ☐
Bypass Reverb Zones: ☐
Play On Awake: ✓
Loop: ☐
Priority: 128
Volume: 0.5
Pitch: 1
3D Sound Settings
2D Sound Settings

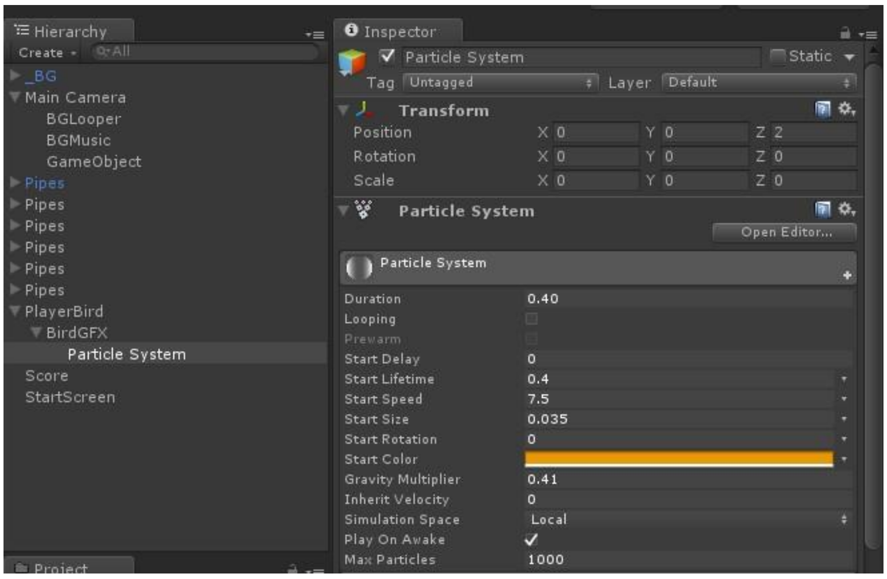
To add video effects when the screen is touched, we want to begin by adding a new Particle System GameObject. We start by clicking on "GameObject" in the toolbar, selecting "Create Other", and then selecting "Particle System". Now, we want to add it as a component of the BirdGFX object, hence we drag the "Particle System" object to the BirdGFX object under the PlayerBird object hierarchy.



We now have our Particle System object. However, as you might have noticed, it is always running and does not stop. We want it to play only when we touch the screen for a short amount of time. The "Looping" option must be unchecked to stop the effect from replaying forever, but other options are left totally up to you. Some of my options are as follows:

- Duration: 0.40 (seconds)
- Looping: Unchecked
- Start Lifetime: 0.4 (seconds)

- Start Size: 0.035
- Gravity Multiplier: 0.41
- Rate: 50 (50 particles emitted at once)



Particle System
Tag Untagged Layer Default

Transform
Position X 0 Y 0 Z 2
Rotation X 0 Y 0 Z 0
Scale X 0 Y 0 Z 0

Particle System

Duration	0.40
Looping	
Prewarm	
Start Delay	0
Start Lifetime	0.4
Start Speed	7.5
Start Size	0.035
Start Rotation	0
Start Color	🟧
Gravity Multiplier	0.41
Inherit Velocity	0
Simulation Space	Local
Play On Awake	✓
Max Particles	1000

Changes to the Code to Handle our Objects

After adding these new objects and components, two of our scripts must be changed to reflect the objects added.

13. Exporting the Game to Mobile Devices

Now that we have created the game, it is time to make it playable on a mobile device such as IPhone or Android© devices! In the Unity3D Professional 2D Game Development course, we give you the complete guide to publish and monetize your game on IOS, Android with a complete stepbystepguide to cover all the steps and practices. More detail.

In order to export the game for the simplest platform in our opinion "Android" , we must first have a scene file ready, which holds all the information of the scene of our game. Simply pressing CTRL-N will bring the prompt to save the current scene. Save it with whatever name you would like.

Once we have our scene, we must go to File and select "Build Settings…" A list of platforms in a new window is brought to you. Now, select Android from the list, and click on the "Player Settings" button on the bottom of the window. Now, under the Inspector tab you will find a plethora of settings to customize. Navigate to the field "Bundle Identifier" and identify your bundle in the form "com.Company.ProductName". In my case, I typed in com.ahmadnaser.ClonyBird for this field. Now go back to the Build Settings window, select Android from the list of platforms, and select "Build". A new prompt will arise and ask where you want to save the file and by what name. Name the apk file whatever you would like to name it and save. Viola! You have successfully created you first Android game with Unity!

CONNECT WITH AHMAD

Thank you so much for taking the time to read this book. I'm excited for you to start your path to creating the life of your dreams as a game developer. If you have any questions of any kind, feel free to contact me at www.ahmadnaser.com/contact
You can follow me on Twitter: @ahmadahammad and connect with me on Facebook: https://www.facebook.com/dev.ahmadnaser

Join me At Social Media Sites:

Subscribe :	http://bit.ly/1gz92ua
Website :	http://www.ahmadnaser.com/
Udemy :	http://bit.ly/ahmadnaseracademyudemy
Asset Store:	http://bit.ly/anunityassetstore
PlayStore:	http://bit.ly/anplaystore
AppleStore:	http://apple.co/1RAcuvB
Facebook :	http://on.fb.me/1k6THXg
Twitter :	http://bit.ly/1k6TVO3
Google :	http://bit.ly/1htkXul
Flickr :	http://bit.ly/1bIuuQa

I'm wishing you the best of health, happiness and success! Here's to you!
Ahmad A.Naser

About The Authors

Ahmad A.Naser is a one stop shop in web, mobile and 2d games development and design. He is a professional technical trainer at many institutions and organizations such as HRD, Galaxy, PITA and many others. He earned his Bachelor's degree in Computer Science from Birzeit University.

Ahmad loves educating and inspiring other entrepreneurs to succeed and live their dreams and build their own careers with his in-depth technical knowledge.

- 3+ Years' Experience in Mobile & Unity2d Development.
- 5+ Years' Experience in Web Development & Design.
- 3+ Years' Experience in online marketing and CRM systems.
- Designed his first educational game in 2005.
- Published 5 Online Courses on 2D Professional Games Development at Udemy.com

Learn more at https://www.udemy.com/u/ahmadnaser

Get the unity3d development and design video training

Series here: AhmadNaser.com

Rami Asia is an active software engineer, game developer, and researcher in Arabic natural language processing. He earned a Bachelor's Degree in Computer Science from Birzeit University. Raised in California and living in Palestine, he shares a love for the world, entrepreneurialism, and of course video games.

To connect with Rami, please email rami.t.asia@gmail.com

LinkedIn: https://www.linkedin.com/pub/rami-asia/58/a0a/3aa

Other Courses Published By the Authors

You can discover in "Unity + Unity5 Professional 2D Game Development Course" exactly how I give three years of experience in unity 2d development, publishing, monitization both for android and ios with over $400 free assets, Storyboard UI Kit and complete projects at https://www.udemy.com/unity3d-professional-2d-game-development-from-a-to-z/?couponCode=LearnFromAhmad1412

Unity3D Professional 2D Game Development From A to Z

Ahmad Naser, Technical Manager, Cloud & Web Specialist, Technica...

★★★★★ (8)

$499

~~$499~~ $14 97% off

You can see and learn how to build your clone of "Unity3d Alphabet Board Game Step by Step" one of my successful games both on apple store and playstore with over 50000+ download just within the first few months of release at https://www.udemy.com/unity3d-alphabet-board-game-step-by-step/?couponCode=LearnFromAhmad1412

19 lectures
2 hours video

Unity3D Alphabet Board Game Step By Step

Ahmad Naser, Technical Manager, Cloud & Web Specialist, Technica...

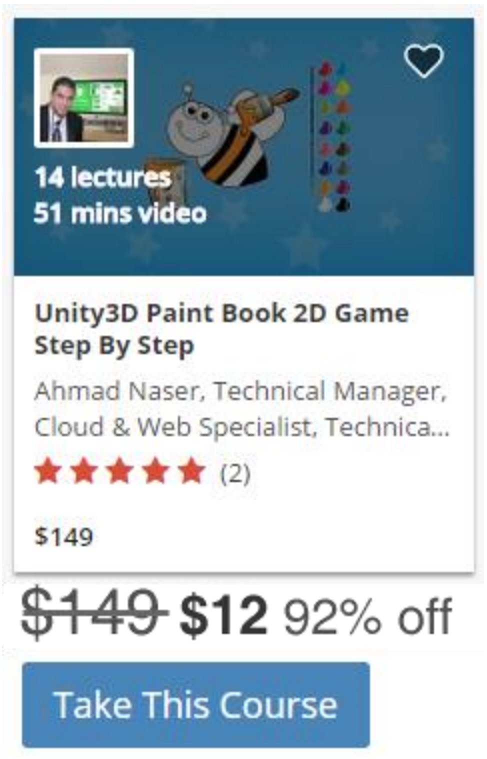
★★★★★ (4)

$199

~~$199~~ $12 95% off

You can use the following two step by step courses in order to build your complete 2d game with a lot of inspiration in coding and design using adobe illustrator and unity4.5+ and unity5.

https://www.udemy.com/unity3d-paint-book-2d-game-step-by-step/?couponCode=LearnFromAhmad1412



Unity3D Paint Book 2D Game Step By Step

Ahmad Naser, Technical Manager, Cloud & Web Specialist, Technica...

★★★★★ (2)

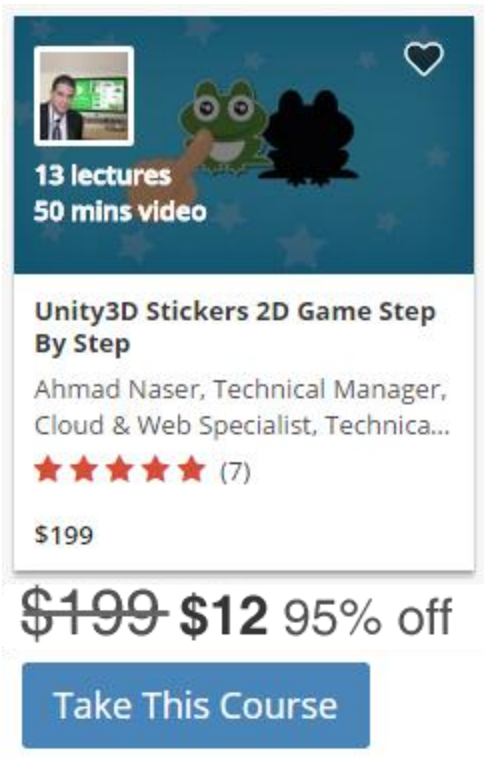
$149

~~$149~~ $12 92% off

Take This Course

https://www.udemy.com/unity3d-stickers-2d-game-step-by-step/?couponCode=LearnFromAhmad1412

.



Unity3D Stickers 2D Game Step By Step

Ahmad Naser, Technical Manager, Cloud & Web Specialist, Technica...

★★★★★ (7)

$199

~~$199~~ $12 95% off

Take This Course

You can learn how to build a complete multilingual android application and publish it on Google play store with this complete course at

https://www.udemy.com/android-multilingual-applications-development/?couponCode=LearnFromAhmad1412

Android Multilingual Applications Development Fro...

Ahmad Naser, Technical Manager, Cloud & Web Specialist, Technica...

★★★★★ (5)

$149

~~$149~~ $12 92% off

ONE LAST THING...

I would like everyone who helped me to write this book, special thanks for Rami Asia and Baraa Nasser.

Special Thanks for quill18creates for the video tutorials

If you enjoyed this book or found it useful I'd be very grateful if you contacted me. Your support really does make a difference and I read all the emails personally so I can get your feedback and make this book even better.