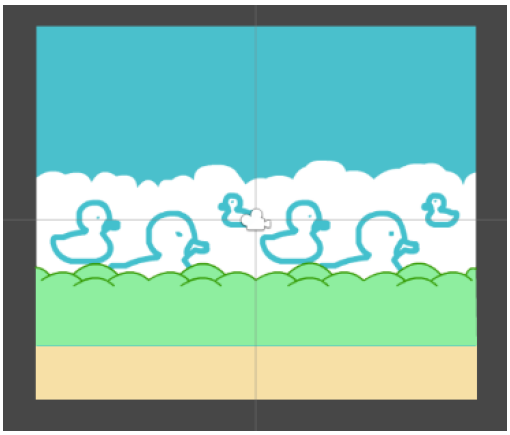
Quacky Ducks

1. Create Gameplay

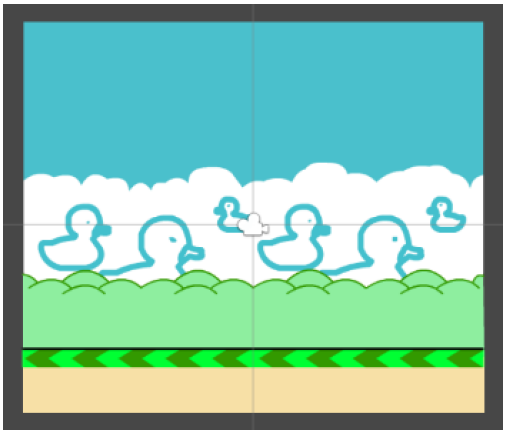
a) Setup Background

Drag the background sprite into the scene view and scale and position it so that it fits the window as shown below:



b) Add animating floor

Multi select the 5th and 6th sprites in the Assets folder (the ones that have green arrow marks on them) And drag them into the scene to create an animation. Name this ‘GrassAnimation’ and rename it to ‘Grass’ in the Hierarchy view. You will want to scale and position the grass so that it looks like the image below:

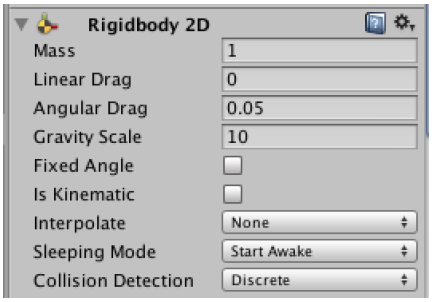


c) Add Character

In the assets folder multi select the two duck sprites and drag them into the scene. When prompted name the animation ‘PlayerAnimation’ and rename it in the Hierarchy to ‘Player’.

You will want to position the player somewhere to the left of the camera so that it leaves enough room between it and the pipes later on.

We want the duck to be constantly falling down until the user clicks the screen to make it go higher. The easiest way to do this is to attach a Rigid Body component to it and use the gravity parameter to achieve the falling down part. Select the ‘Player’ object, choose ‘Add Component’ and add a ‘RigidBody2D’ to it. Try setting the ‘Gravity Scale’ to 10 and press Play to watch it quickly fall out of the sky.

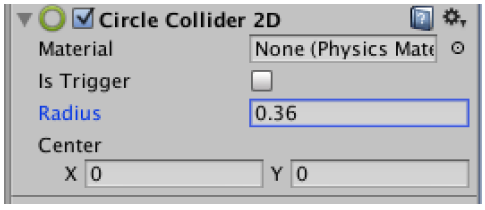


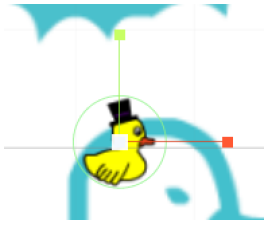
Now to make the player fly higher whenever we click the screen. Add the ’PlayerController’ script from ‘Assets > Scripts’ onto the player object. You will see that this script has 2 parameters: ‘flapDistance’ and ‘flapDropoff’. Distance controls how high the character will rise when the mouse gets clicked and Dropoff is how quickly it will lose that height gain and start to fall.

The script is quite simple, The update function checks to see if the user has pressed the mouse button down, If so then the y velocity gets increased and this gets set to the Rigid Body on the FixedUpdate function. The Update function also constantly reduces the y velocity by the dropoff value but always ensures it doesn’t go below 0 otherwise the combined effect of that and gravity would make the player fall too fast.

Now press play and see how every time you click the screen the duck rises higher!

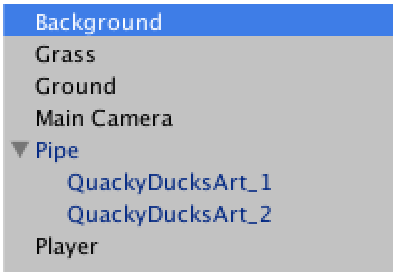
The final piece for the Player is to add a collider to it so that we can detect for collisions! This time we’re going to add a ‘Circle Collider’ to the player. You can adjust the ‘Radius’ parameter of the collider and see the size of the collider change in the scene view (shown by the green circle).



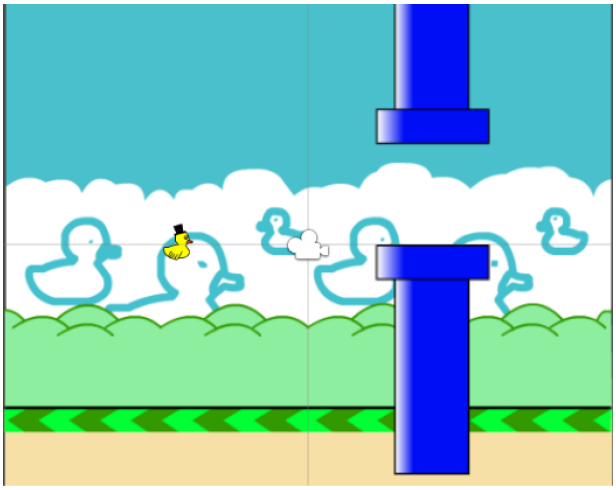


d) Add pipes

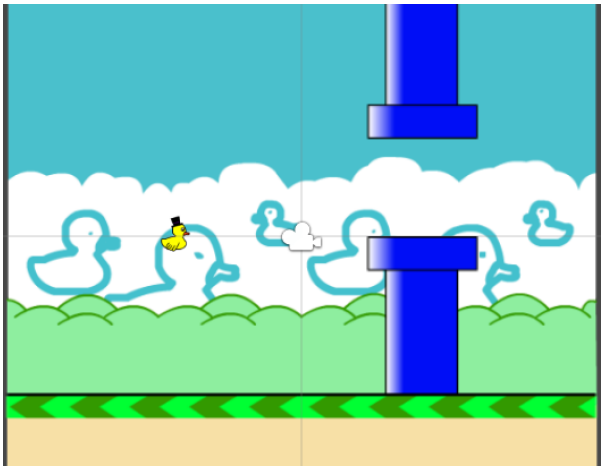
It's time to add some obstacles to the game. Create an empty GameObject in the scene (MenuBar > GameObject > Create Empty) and name it 'Pipes' in the Hierarchy. Select the two pipe sprites from the Assets folder and drag them so they are nested under the 'Pipes' object.



Position the sprites in the scene view as follows:



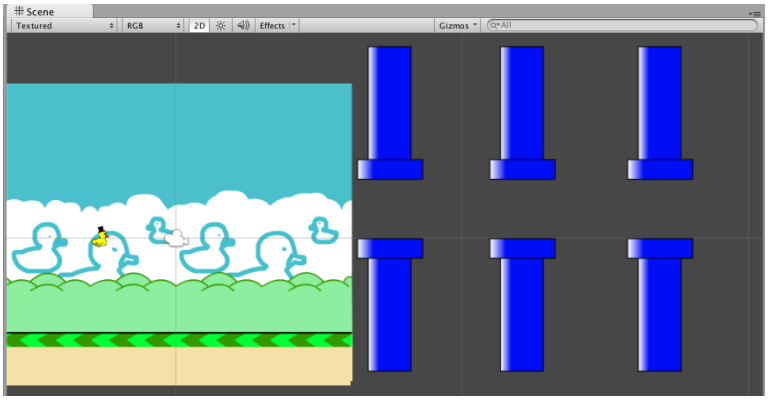
You will notice that the pipes are a bit longer than the scene, this is because they will change position vertically during the game, to make it look better we will move the grass animation in front of the pipes and add the ground sprite (QuackyDucksArt_7) to cover it too. Place all 3 objects in the 'Midground' sorting layer and set the 'Order in Layer' of the pipes to 0, the Ground to 1 and the Grass to 2.



Now it's time to set up the pipes so that when the player collides with them, the game ends. To both of the pipe sprites add a 'BoxCollider2D' for the collisions and to the parent Pipes object add a RigidBody2D component. You will want to set the 'Gravity Scale' parameter of this to 0 to prevent the pipes from falling. Finally add an 'Obstacle' script from the Scripts folder to the parent 'Pipes' object. You will see that the script has 3 assignable values: Speed - controls how fast the pipes move across the screen, highestPosition - the highest y value the pipes should show at and lowestPosition - the lowest value that the pipes will show at. To calculate the highest and lowest values I recommend positioning the Pipes object so that the top of the object is level to the top of the screen and copying it's y value to the highestPosition parameter and then moving the object so that the bottom of the pipes is just resting on the grass animation and copy it's y position to the lowestPosition parameter. Now if you test the scene you will see that each time the pipes reach the left side they reappear at the right but with a different vertical position. This is all handled by the script!

The script sets the Rigid Bodys x velocity to the speed value which gets updated every frame and in the Update function we check to see if the position is less than a hard coded value for the screens width. If it goes beyond the screens width it calls a function to reposition the object to the other side of the screen and to randomly set it's y position so that it is somewhere between lowestPosition and highestPosition.

Now we've got functional Pipes but we need a bunch of them moving across the screen. Drag the Pipes object into your 'Assets > Prefabs' folder to turn it into a prefab, the version in your hierarchy should turn blue to show you this has worked. Prefabs are useful for when you want to have multiple copies of an object. If you decide to change something in one prefab object you can apply the changes to all instances of that object too! Create 2 duplicates of the 'Pipes' object (cmd + d) and position them to the left of the scene so they start out of view, similar to as shown below:



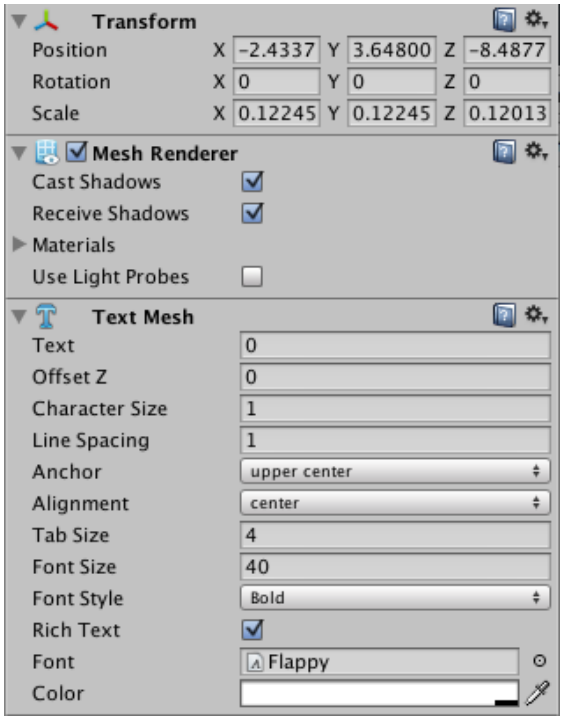
Now when you click ‘Play’ you will see all of the pipes moving and repeating to create a constant stream of obstacles!

2. Showing the score

Ok now it’s time to add some dynamic text so we can show the players score. In the current version of Unity (4.3.3) adding text to a 2D game is a bit of a nightmare. Currently the best way we have to do this is to use 3D text. Create a new 3D Text object (GameObject > Create Other > 3D Text). Rename the object to ‘Score’ and then in the Inspector a few properties will need to be changed:

> Set the Font to the ‘Flappy’ font that was included in the assets.
> The text will appear but will be blurry, set the Font Size to around 40.
> The text will now be sharper but will be huge so set the scale down to about 0.1.
> You can also set the Anchor to ‘upper center’ so that the score is always in the center.

The properties should look similar to this:



The text should now look ok, unfortunately though this text system is not compatible with the sorting layers of our 2D game and so it will be appearing behind our scene. Luckily there is a hidden scriptable way to add any component that has a Renderer to these layers. So if you add the 'TextRenderer' Script from 'Assets > Scripts' to this object and press Play it will fix the problem! Unfortunately the text will still render behind the sprites when the game isn't playing but you should still see the text outlines to know where it is.

Try and position the text so it's around the top centre of the screen and hit 'Play' to check it's working.

To actually perform the scoring, we need a way to check if the player has passed through the center of the pipes. to do this we can create a 'BoxCollider2D' on the parent Pipes object. You will want to adjust the size and position of the Collider so that it fits in the gap between the pipes. You will also want to tick the 'Is Trigger' option. This will still fire an event when another collider touches it however it will not prevent the other collider from passing through it which makes it perfect for scoring.
Now go back to the 'PlayerController' script and uncomment the following lines:

```
public TextMesh scoreText;

private int _score = 0;
private static int _bestScore = 0;
private Collider2D _lastColliderHit;
```

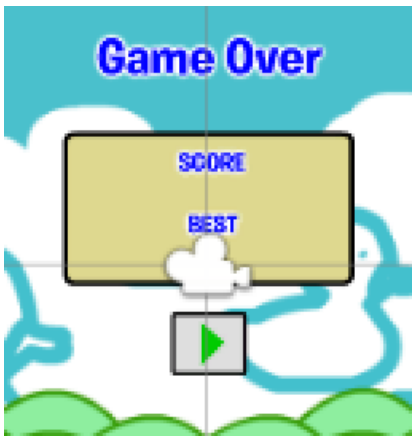
```
void OnTriggerEnter2D(Collider2D collider)
    {
        if(_lastColliderHit != collider)
        {
            _score++;
            scoreText.text = _score.ToString();
            _lastColliderHit = collider;
        }
    }
```

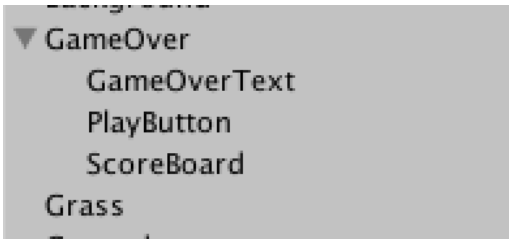
Now because we added a public variable we need to go back to Unity and assign it. Select the 'Player' object and set the 'Score' parameter to the Score text we just added. Now when you test the scene you should see the score increase every time you pass through some pipes!

By setting the _bestScore variable to be static we are ensuring that it never gets reset. Normally Application.LoadLevel will reset all variables but by setting this to static it will not reset.

3. Add game over screen

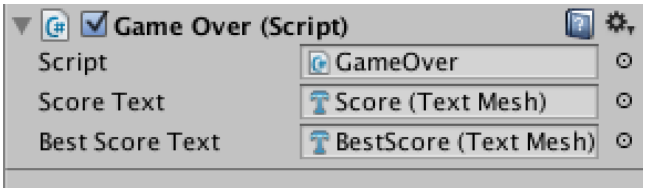
Create a new Empty Game Object and call it 'GameOver'. To this object drag the 'Game Over', 'Score Board' and 'Play Button' sprites and arrange them so they look like below. You will need to place each of the sprites in the 'Foreground' Sorting Layer.





Now we need to add some more dynamic text, so duplicate the previous score clip and drag the duplicate into the GameOverObject and position the text so that it sits under the 'SCORE' text on the scoreboard, and then duplicate the object, call it BestScore and position it under the 'BESTSCORE' text as follows:

Now that the text is in place add the ‘GameOver’ script from the Scripts Folder to the parent GameOver object in the Hierarchy. In the Inspector you will see two values that need assigning for Score Text and Best Score Text, so drag the corresponding objects you just created to them. It should look like this:



The game over script is very simple. It contains one function that sets the values of both the score and best score text objects. This function is declared as public so that other scripts can access it.

Okay now the Game Over component is ready but we don’t want to use it yet, so let’s turn it into a prefab to use later! Select the ‘GameOver’ object and drag it into the Assets folder to create the prefab and then delete it from the scene.

Open the ‘PlayerController’ script and uncomment the following lines:

```
public GameObject gameOverScreen;

private int _score = 0;
private int _bestScore = 0;

void OnCollisionEnter2D(Collision2D collision)
 {
      if(_score > _bestScore)
      {
         _bestScore = _score;
      }
```

```
        GameObject gameOverScreen = Instantiate(gameOverPrefab) as GameObject;
        gameOverScreen.GetComponent<GameOver>().setScore(_score,_bestScore);
    }
```

Then in the Inspector panel for the ‘Player’ object set the ‘Game Over Prefab’ to the prefab of GameOver that we just created in the assets folder.

Now when you hit play and crash into one of the pipes the Game Over screen will appear.

Unfortunately the world is still spinning around after we’ve lost! Luckily unity has a nice feature to prevent this. In the OnCollisionEnter2D function, uncomment the following line:

```
Time.timeScale = 0;
```

This will freeze time for any physics calculations and animations, so when you crash the game will freeze appropriately!

Unfortunately when we restart the game this effect will carry over so we need to restore time during the ‘Start’ function, so uncomment the following line:

```
Time.timeScale = 1;
```

This will ensure that time starts moving again when we choose to replay the game!

3. Add SFX

Adding SFX to the game is quite easy. Firstly select the ‘Player’ object and using ‘Add Component’ add an ‘Audio Source’ component to it, as this will be the source of all the sounds we want to play.

If you open up the PlayerController script again, it’s time to uncomment out some more code:

```
public AudioClip clickSound;
public AudioClip scoreSound;
public AudioClip crashSound;
```

These are the values that will appear in the inspector so that we can assign the sounds to the script.

```
private AudioSource _audioSource;
```

_audioSource = GetComponent<AudioSource>(); (In the Start function)

This stores a reference to the component we created so that we can use it quickly later on.

And finally these 3 lines of code to play each sfx:

_audioSource.PlayOneShot(clickSound); (in the update function)

_audioSource.PlayOneShot(scoreSound); (in the OnTriggerEnter2D function)

_audioSource.PlayOneShot(crashSound); (in the OnCollisionEnter2D function)

Now because we added some public variables we need to go back to the Inspector and assign them. In the 'Assets > Sounds' folder you will find the 3 sounds and each one will be named the same as the variable they need to be associated with - 'Click','Score' and 'Crash'. Assign them and click 'Play' and enjoy the result!

4. Challenges

a) Add a pause button to freeze gameplay

b) Add a start menu to the game

c) Increase the speed of the pipes every time they get re-initialized.

Acknowledgements:

Audio SFX were obtained from http://www.soundbible.com and are available under the Creative Commons Attribution 3.0 Licence.

Mario Jumping was renamed to Click (http://soundbible.com/1601-Mario-Jumping.html)

Blop was renamed to Score (http://soundbible.com/2067-Blop.html)

Strong Punch was renamed to Crash (http://soundbible.com/1773-Strong-Punch.html)