Flappy Bird Improvements

This tutorial relies on a completed previous tutorial- that of flappy birds. It will show you how to smooth out, add to and generally improve your flappy birds experience.

Part 1- A stylish main menu

At the moment- the game just starts, what we want is to add a nice main menu that plays on start up.

Open the flappybirds tutorial project and create a new scene.

It's better to save it straight away, so, File- Save Scene as, name it Menu.

Now, import the asset file for this lesson into Unity (extract->Drag and drop in project folder), it should contain a number of Menu buttons and images. The assets for this bit were created by The Le and downloaded from OpenGameArt under a public domain license.

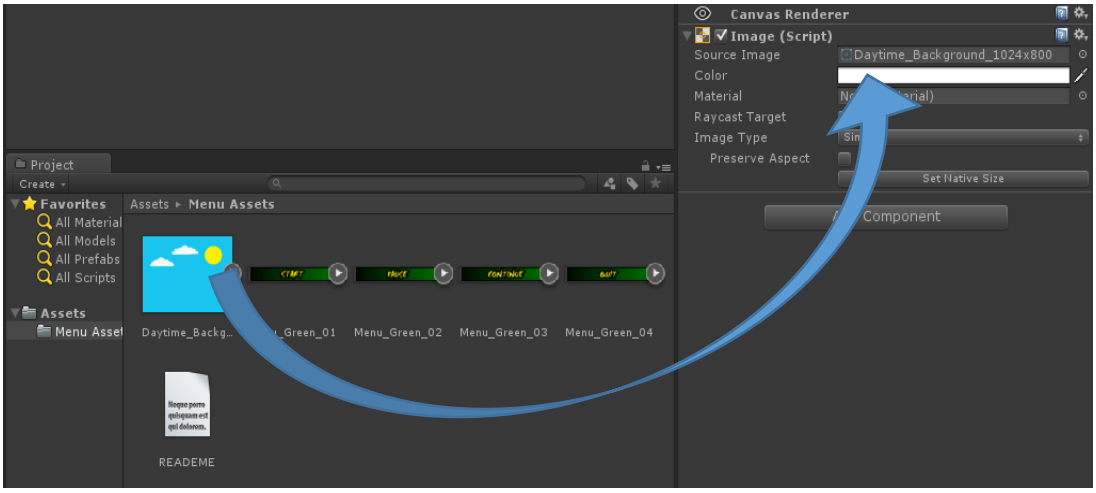
The Canvas

At the top menu go to GameObject->UI->Canvas to add a canvas to the screen.

The canvas is the collection that all UI elements are added too and built on, it has a number of options (shown right) that control how your UI is shown, some of which we'll look at here.

Like the canvas, the EventSystem deals with UI and is added automatically when you add any UI elements. It is used to handle input to objects within a scene particularly in UI elements, (things like clicking, dragging, etc.).

Add an image to the canvas and rename it background, this is going to be the background of our menu, we'll use kungfu4000's background- simple daytime, again public domain from OpenGameArt. Drag it from the project view to the image's Source image field.



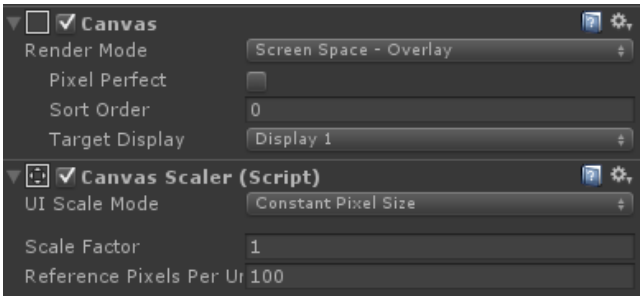
It's going to be tiny, so find the "Set Native Size" button, press it and it should set the image to 1024 by 800, more than enough.

Scaling the Canvas

You'll use the Canvas Scaler to adjust the way the background image displays, at the moment if the game view is too small, Unity chops off sections to make it fit.

If you were to run the game on a device with a large enough resolution or simply stretch the Game view to fit the whole image, you would see the entire background image.

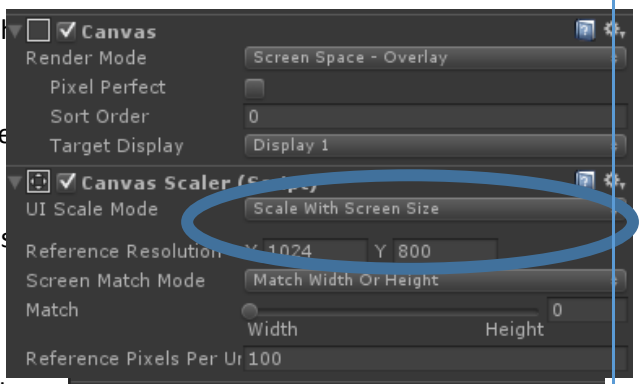
Designers use something called a reference solution to decide on all sorts and many games only use this resolution. What we are going to do is use a happy intermediate, the Canvas Scaler. You'll see it attached to every Canvas; click on canvas in the Inspector, you should see the Canvas Scaler component.



The Canvas Scalar has three modes:

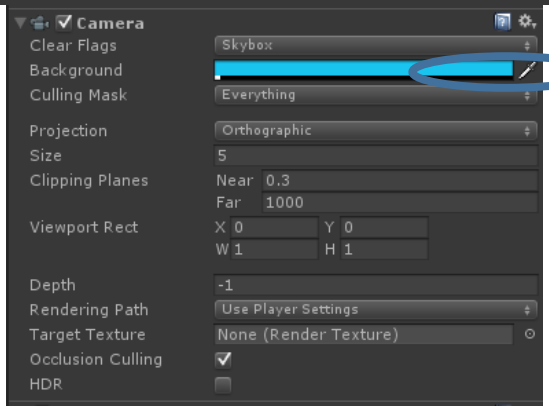
- Constant Pixel Size: Makes all the user interface elements retain the same pixel size, regardless of the screen size.
- Scale With Screen Size: User interface elements are sized and positioned in accordance to a referenced resolution. If the current resolution is larger than the referenced resolution, then the Canvas will maintain the reference resolution, wh
the target resolution.
- Constant Physical Size: Positions of the user interface
such as millimetres or points.

Change the component mode to Scale With Screen Size and s
800.



Just to make sure, we'll also set the Camera background to the same colour blue as the image, so it blends.

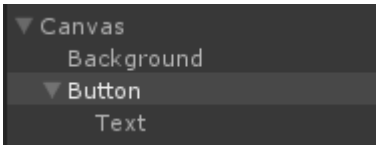
Click Main Camera-> and look for the background field. If you click the eyedropper and hover over the image, you can select the colour you'd like.



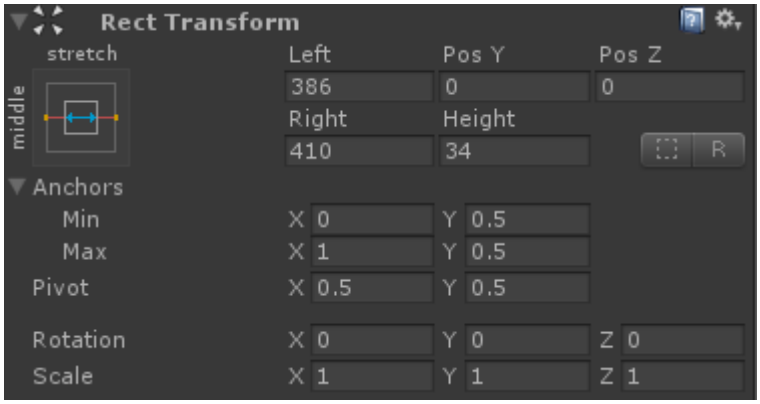
Adding Buttons

So, we have a nice background but no buttons or functionality in it, so, add two new buttons (Gameobject->UI->Button). You can see it appear automagically but it's a bit bland, luckily, we've got some made earlier. Drag the "Start" image from assets to the Source image field of the button, similar to how we did above. Set Native Size again to make it look a bit bigger and clearer.

Now we have a button that looks nice but does nothing. We're going to add some functionality but first, we're going to sort out the text overlaid. In the Hierarchy, find your button, click the little down arrow then select the text object attached. Now delete it, we won't be using it in this tutorial.



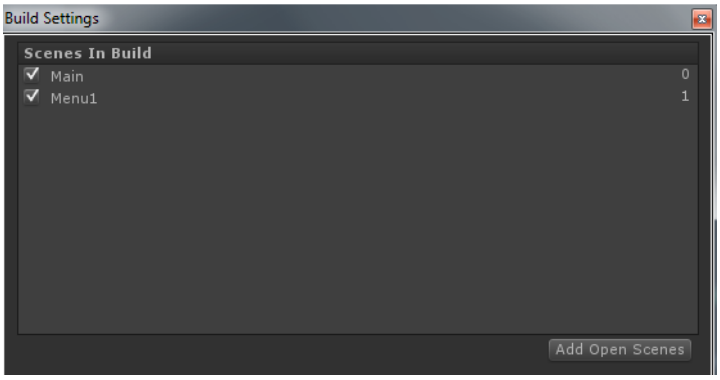
Rename the button StartBtn and set it's anchor to stretch middle and position etc. to the information below.



Now, add another button, set it up the same but use the Quit image, name it QuitBTN and position it at -100 Y.

Making it work

Now, a bit of code is needed, you need to add some code that says, when you press start, load the level, when you press quit, quit.



First of all though, we need to specify which scenes we're using in our game.

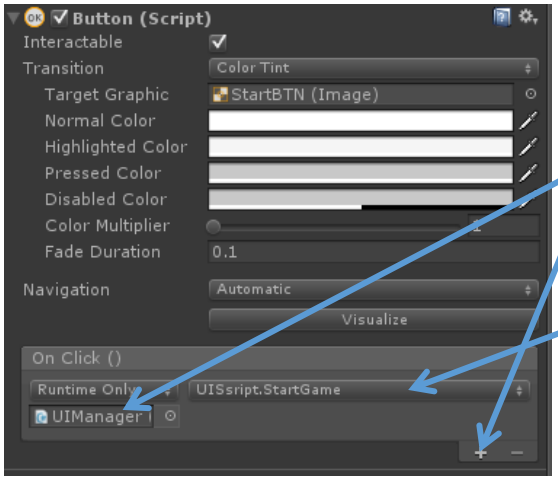
Click on File->Build settings.

A menu should appear with a lot of build settings, what we want is the bit at the top, drag from project view the scenes you want to include (probably main and menu) and it'll assign them a number.

Exit and create a new object called UIManager and a new script named UIScript then open it. The following is what needs to be included, each line is explained using the comments (//).

```
using UnityEngine;
using System.Collections;
//Tells Unity you're using it's new scenemanager
using UnityEngine.SceneManagement;

public class UISsript : MonoBehaviour {

    //This needs to be public so it can be seen by the UI, it will be the code that runs
    public void StartGame()
    {
        //tell the scenemanager which scene to run
        SceneManager.LoadScene("Main");
    }
}
```

Then you just need to link it to your buttons click.



1) Click the plus
2) Drag the UIManager object you created here.
3) Find the UIScript on that object and the StartGame function.

We're going to do something very similar to the Quit button, we'll write some code that says when pressed-> quit and run it.

In UISCript add the following under the } of StartGame and attach it to the quit button as above;

```
public void QuitGame()
{
    Application.Quit();
}
```

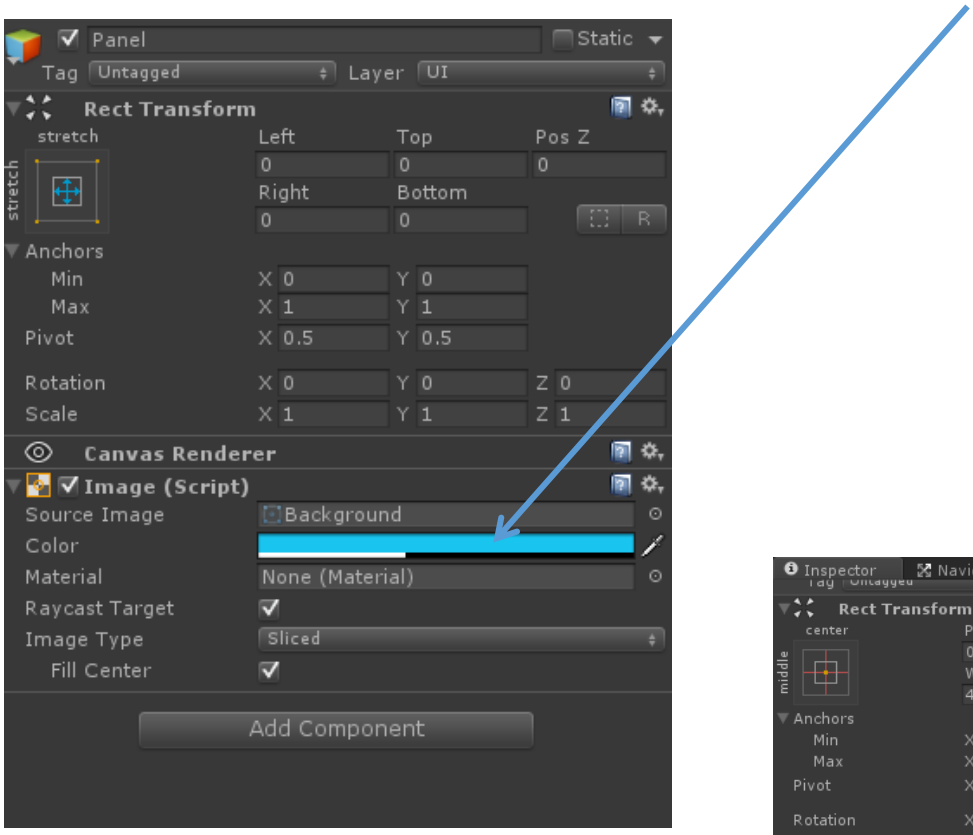
Now it's up to you to finish snazzing up your menu, maybe a logo and title would work. Try it out.

In game menus

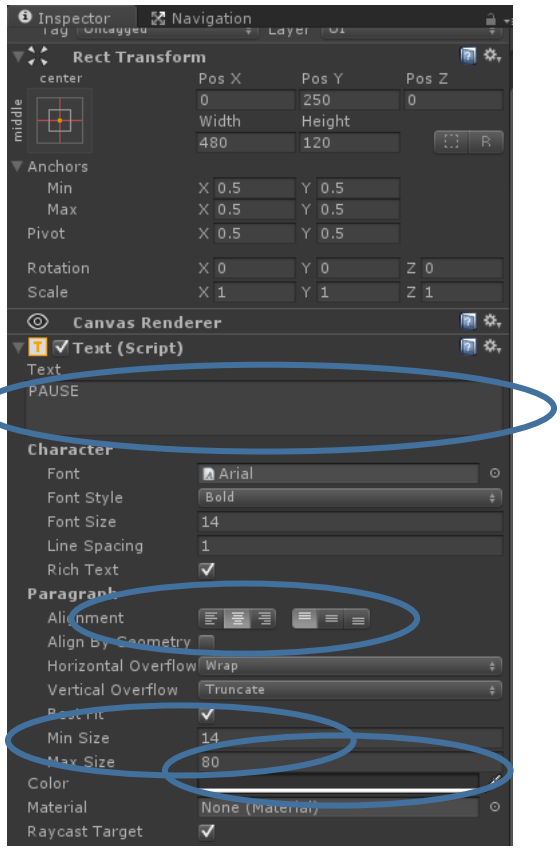
Now you have a main menu, perhaps an in game one is needed as well. We could tie this to a key on the keyboard if needed, but that's going to cause compatibility issues on mobile, instead, we'll add a virtual button and use that.

So, open your main scene, add a Canvas as we did before. This time we are going to add a panel, a panel is like a container for things within the canvas but can be seen. By default it's a semi transparent grey, but we're going to change it.

Add a panel and change the color to that of the background of the sky for instance.



Now, to the panel add a text element, set the settings as shown to the right. We change the text to say "Pause" (this will be our pause menu) and align it in the middle. We edit the width and height to make it more appropriate. Finally we click the Best fit sizing button, this makes sure our font resizes depending on the size of the box. You can change the colour of the writing too if you'd like. (This doesn't look too great, you may want to replace it with a suitable image or new font).



This time we're going to add two options, one to continue, one to quit.

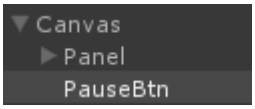
Add two new buttons as detailed above and assign one the continue image, the other the quit. Name them sensibly and read on for the code to link them.

If you play the game now you'll see that the menu appears over the game all the time. We need to stop this, we also need to make the buttons work.

To your panel add a "Canvas Group" component, this allows you to set a number of settings related to the canvas, such as making it appear, making it able to be clicked through, etc.

What we want is that when we pause the game the menu appears, when we unpause it disappears.

Create a new button NOT on the panel (see to the right, easiest way is to right click canvas and add it that way). Again, set it up as you did above.



Now, add a new GameObject, assign the script "UIScript" to it and open it up. We've already got a QuitGame function we can use but we want one that resumes the game. By association, we may want one that pauses the game too, we'll add these now.

Pausing the game

Pausing the game is simple, we want the gameplay to freeze and the pause panel to appear.

At the top of your UIScript add a new field to hold the panel (note we use the top type GameObject) and the CanvasGroup.

```
public class UISsript : MonoBehaviour {

    public GameObject panel;
    CanvasGroup settings;
```

Then we write a Start() section that gets the CanvasGroup and makes it disappear if the panel exists. We have to check because we're using the same script in two places.

```
void Start()
{
    if (panel != null)
    {
        settings = panel.GetComponent<CanvasGroup>();
    }
    if (settings != null)
    {
        disAppear();
    }
}
```

Finally, we write the appear and disappear helper methods along with some associated code;

```
public void pauseGame()
{
    Time.timeScale = 0;
}

public void unPauseGame()
{
    Time.timeScale = 1;
}
```

The first two are the ones that actually pause and unpause the game, we write them separate incase we need to use them elsewhere.

Setting Time.timeScale to 0 pauses the game (time no longer passes) and to 1 starts it again.

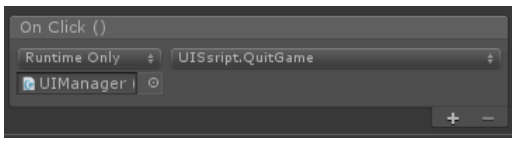
```
public void appear()
{
    pauseGame();
    settings.alpha = 1;
    settings.blocksRaycasts = true;
}

public void disAppear()
{
    unPauseGame();
    settings.alpha = 0;
    settings.blocksRaycasts = false;
}
```

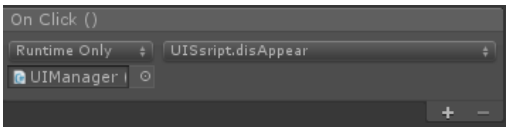
Then we write some code that makes the panel appear and dissapear, after that it will call the code we wrote before.

Remember, settings is the name we gave to our canvasGroup, this lets us access the alpha (how transparent it is) and whether raycasts are blocked (can you click through the panel).

Now, we assign these to buttons, drag the GameObject the code is attached to in and assign the correct function.

Continue gets disAppear and Quit gets QuitGame.





Finally, the separate pause button gets the appear code.

Stopping keyboard input whilst paused

Well, you should have a pause menu now, the problem occurs when we consider that clicking continue will cause the bird to jump, often into its death. Add the following to your UIScript

```
static bool paused;

public static bool isPaused()
{
    return paused;
}
```

We have a variable that checks if the game is paused or not and a public method that lets other classes check. Both are made static so we don't need a direct reference to the UIScript class we can just call it using its class name.

Now, edit your pause and unpause methods to set pause = true/ false and change the following in Bird. This just says only move the bird if the button is down AND the game isn't paused.

```
if (Input.GetMouseButtonDown(0)&&!(UISsript.isPaused())) {
```

Keeping Score

The final thing that we need to do is to keep score, we're going to do this in a very very simple way;- how long can you stay alive.

We do this by measuring time since start, which happily Unity provides us with an easy way of tracking.

We use the Time class again; open up the UIManager once more if it's closed and we'll make the following changes;

```
//Tells Unity you're using the new UI tools
using UnityEngine.UI;
```

Add this at the top to make sure Unity knows we're using the full suite of new UI tools.

```
public GameObject panel;
CanvasGroup settings;
static bool paused;
int score;
[SerializeField] Text scoreTxt;
```

Add the score variable to the bottom of your list. We'll make it an Integer whole number to keep it simple.

Add the scoreTxt variable as well that will be the Text box on the UI used to hold the score.

Then, the code to update the score;

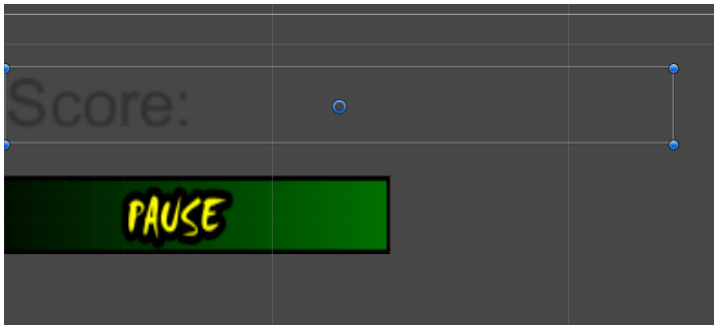
```
void Update()
{
    score = Mathf.FloorToInt(Time.timeSinceLevelLoad);
    scoreTxt.text = ""+score;

}
```

Mathf.FloorToInt() is a helper function that just rounds the number up or down to a whole number.

Time.timeSinceLevelLoad just returns the amount of time you've been playing that level for (remember, time pauses when the game is paused). This is reset when you reload.

scoreTxt.text=””+score – this just says, make the score a string (””+ makes an int a string) then set the text of scoreTxt to that value. Update runs every frame meaning we'll do this every frame, this isn't super-efficient but isn't a worry at such a simple game.

Now, add two new text boxes to the canvas, position them somewhere clear (top right is easy) and you can fiddle around with the text settings to your liking. Change the text of one to Score: name this one ScoreLbl and leave the other's text blank and name is ScoreTxt. Drag the second one (on the right) to the UI manager field for ScoreTxt.



Now try playing your game. You should have a working score system.

You've hopefully learnt a lot in this brief set of tutorials, about colliders and physics, the new UI system and controlling time. There's plenty to extend and change here and I highly encourage you to do so, it's one of the best ways to learn.