Girls' Programming Network

Flappy Bird!

This project was created by GPN Australia for GPN sites all around Australia!

This workbook and related materials were created by tutors at:

Sydney, Canberra and Perth

Girls' Programming Network

If you see any of the following tutors don't forget to thank them!!

Writers

Amanda Hogan
Isabella Hogan
Renee Noble

A massive thanks to our sponsors for supporting us!

ATLASSIAN

Part 0: Setting up

Task 0.1: Start Programming

Follow the instructions from the lecture to set up your flappy bird python file!

Hint

Don't forget to ask the tutors for help if you're not sure what to do next

Task 0.2: You've got a blank space, so write your name!

At the top of the file edit the comment to write your name!
Any line starting with # is a comment.

```
# This is a comment
```

Task 0.3: Some scaffolding

Now read through the other comments to see the different steps we'll be doing.

☑ CHECKPOINT ☑

If you can tick all of these off you can go to Part 1:

- ☐ You should have a file called flappy_bird.py
- ☐ Your file has your name at the top in a comment
- ☐ Run your file with F5 key and it does nothing!!

Part 1: Making a game

Intro to Pygame

Task 1.1: Print a welcome and the rules

Welcome the player and print the rules!

Use some print statements to make it happen when you run your code:

Put this under the print welcome comment

```
The game is about to start!
Click the mouse to "flap" upwards
Dodge the pipes and the floor
Good luck and have fun!
```

Task 1.2: Get Pygame Zero

PygameZero will give us the tools we need to write our game code. Let's import it!

1. At the top of you file, under the start modules comment, **import Pygame Zero**

2. At the bottom of your file, under the runs everything comments, **make Pygame Zero GO!**

Hint

To import and start Pygame Zero we can use this code:

```
import pgzrun
...
pgzrun.go()
```

Task 1.3: Make a screen

Before we make our game we need to set up the screen to display it on!

1. **Make a screen** that is 800 pixels wide and 600 pixels tall. This should be under the create constants comment

2. **Run** your code! Do you see a big dark box?

Hint

To make a screen that is 100 pixels wide and 200 pixels tall you could use this code

```
WIDTH = 100
HEIGHT = 200
```

☑ CHECKPOINT ☑

If you can tick all of these off you can go to Part 2:

- ☐ Print a welcome
- ☐ Print the rules
- ☐ Shows a dark screen that is the same size as the background image
- ☐ Try running your code!

Part 2: Images!

Task 2.1: Save all the images!

We need to load all the images that we need for our game!

1. **Download** the images onto our computer. Go to https://www.girlsprogramming.network/flappy-bird-images and download all the images there.

2. **Double click** on the zip folder to *unzip* the the folder to get to the images

3. **Move the images folder** into the "flappy_bird" folder

Task 2.2: Backgrounds are better!!

Let's start our game by adding the background to the screen! We just downloaded the pic! We'll be using a **function** to **draw** things to our screen.

1. Under the comment "make background" make a new Actor() called background with an image of "bg".

2. Directly under that make it so that the actor's x value is 400 and y value is 300

3. **Create a function** call `draw()` under the draw everything to screen comment

4. Inside your function, ***draw background to the screen*** This should be under the set background image comment.

5. Make sure the code you wrote in Step 4 is ***indented inside your function***.

Hint: How to function!

How do I indent things inside my function?
Use the Tab key to make the parts you want inside your function go inside!

```
def myFunction():
    # this line is indented, it's part of the function!
# this line is not indented, it isn't part of the function.
```

Hint: Make an actor!

To make an actor with an x value of 10 and a y value of 25, it should look like this;

```
myCharacter = Actor("characterImage")
myCharacter.x = 10
myCharacter.y = 25
```

Hint: Draw an actor!

We can use draw, since we already programmed the x, y position for our background Actor.

Here's an example:

```
myCharacter.draw()
```

☑ CHECKPOINT ☑

If you can tick all of these off you can go to Part 3:

- ☐ Run your code!
- ☐ Saved all the images to the same place as your code
- ☐ Made the background appear on the game screen

★ BONUS 2.4: Funky Backgrounds!!

Waiting for the next lecture? Try adding this bonus feature!!

We can use any image as a background! Go on the internet and find another image to use as a background! Save it in like we did with our flappy bird images, load it and use it instead of our regular background image

Note: If your new background image is a different size you might need to change the size of the image to fit the screen.

Events & If Statements

Part 3: The bird is the word!

Task 3.1: Where's that bird?

This game is meant to be Flappy Bird but there's no bird on our screen! Let's fix that.

1. **Check** that you have the **bird image file called bird.png** where you saved your code. This should be in your images folder.

2. Create an ***Actor*** using the bird image,
   Store it in a variable **named bird.** Do this after the make bird comment

3. After you make the bird set the **x value** to be **160** and the **y value** to be **300**

Task 3.2: Show me the bird!

Now we're ready to add the bird to the screen!

1. Go to your **draw function**

2. **After** the draw actors comment, ***draw*** your bird.
   Make sure this is indented inside your function.

Task 3.3: Make the bird fly

We need to make the bird move downwards on the screen

1. Make a new **function** called `update()` after the "updates everything" comment

2. You need to **increase** the bird's y value **by 1** by adding one to its value under the updating bird comment

Pygame zero does all the work for looping and frames so don't worry about that.

Hint

We need to save a new y value to bird. We can do this the same way we originally set it.

e.g.

```
myCharacter.y = myCharacter.y + 5
```

Task 3.4: Falling off the screen

When our bird touches the bottom of the screen it should end the game.

1. **Import** a new module called `sys` directly under where you `import pgzrun`

2. In the update function under the "bird hits bottom of screen" comment you need to test if the bird's y value is **more** than the **height** of the screen. You will need an **if statement** for this.

1. If it is, print `"Game Over!"` then put the line `sys.exit()` to immediately **stop the game**.

Task 3.5: Moving the bird

Our bird should "flap" upwards when the mouse is clicked.

1. Make a **new function** called `on_mouse_down()` after the moving comment

2. **Decrease** the bird's y value by 50

☑ CHECKPOINT ☑

If you can tick all of these off you can go to Part 4:

- ☐ There is a background still on the screen
- ☐ There is a bird on the screen and its falling
- ☐ When you click the bird moves back up
- ☐ Run your code!

★ BONUS 3.6: Funky Bird!

Waiting for the next lecture? Try adding this bonus feature!!

We can use any image as our bird!

1. Go on the **internet** to find a cool image that you want to be the **character** of the game. Must be a jpg,png, or gif

2. Save the image into your **images folder**

3. Where you've set the bird's image replace **"bird"** with the **name of your file**

Try and make sure the new image is a similar size so the game isn't too hard.

Classes

Part 4: Pipes!

We'll be using a class for the pipes to make them easier to deal with an eventually decrease the amount of code

Task 4.1: Make a pipe class

Let's make a class called Pipes with an initialise function. Don't pass anything in yet!

This class should be right under the make pipes comment

Hint

To make a class it should look like this:

```
class myClass():
    def __init__(self):
```

Task 4.2: Initialising it!

To make our pipes we will need some values. Each Pipes object will consist of two pipes - a top and a bottom. To make these pipes we'll need some information:

- The shared x value
- The central y value between the pipes
- The constant gap between the pipes

For now let's pass those variables in.

Hint

To pass variables into an **init method** it should look like:

```
def __init__(self, parameter1, parameter2, ...):
```

Task 4.3: Now what?

We now have some pipes but let's make them "actors" for pyGame to know what to do with them.

Inside the **init method**:

1. Make a top and bottom actor using the images "top" and "bottom".

2. Use the passed in information to set the x and y values for the two actors.
    - The top's y value is found with the formula:
        The centre value - (half the height of the pipes(300) + half the gap)
    - The bottom's y value is found with the formula:
        The centre value + half the height of the pipes(300) + half the gap

The top and bottom actors should be declared, so they can be changed by future methods. However, the parameters that have been passed in.

Hint

To pass variables into an init method and then declare them it should look like:

```
def __init__(self,parameter1,parameter2,...):
    self.parameter1 = parameter1
    ...
```

Task 4.4: Now what pt2.

Now that we have our new class plus some parameters set up, we need to make the pipes. We're going to start with three.

They should be made directly under the class. Once created, put each into a pipes list.

- The first pipes should have an x value of 266, and a central value of 400
- The second pipes should have an x value of 532, and a central value of 180
- The third pipe should have an x value of 798, and a central value of 415

The gap between the pipes should be 220.

Hint

To make an instance of an object it looks like this:

```
myObject1 = myClass(parameter1,parameter2,...)
```

Task 4.5: Draw those pipes!

In the draw function under the draw actors comment you need to loop though your pipes list and draw the top and bottom of each to the screen.

Hint: addressing an object's variables

Remember that to address a value of an object it should look like:

```
myObject1.parameter1 = 6
```

Hint: Looping through a list

To loop through a list you need to do something along the lines of:

```
for thing in myList:
    thing  = thing + 5
```

Task 4.6: Move those pipes!

In the update function under the update pipes comment you need to loop though your pipe list and decrease the x value of the top and bottom pipes by 1 for each of the pipe groups

Hint: addressing an object's variables

Remember that to address a value of an object it should look like:

```
myObject1.parameter1 = 6
```

Task 4.7: Wrapping around!

Run your code and see what happens.
Once the pipes are off the page they just disappear! Let's make them come back!

In your update function in the loop where the pipes are updated, check to see if their x value is less than -44. If it is, set it to 844 to send them back to the start!

☑ CHECKPOINT ☑

If you can tick all of these off you can go to Part 5:

- ☐ The game has some moving pipes
- ☐ The program ends when the bird hits the ground

Part 5: Game Over!

Task 5.1: Colliding!

Now our bird goes up and down and our pipes move right to left. We need to make some collisions! We need to check every pipe to see if the bird is hitting it. For now let's quit the program when the character dies.

Remember you'll need to check the top and bottom of each Pipes object.

Hint

To check if one actor has collided with another in pygame zero it looks like this:

```
if myObject1.colliderect(myObject2):
    print("Hit!")
```

Task 5.2: Points!

You need to make a variable in your create constants section that will store a score. Then every time a set of two pipes "falls" off the edge of the screen you should give the player a point. For now you can just print the score before every line that says `sys.exit()`

Remember that this variable will need to be global in the update function.

Hint

To make a variable global you need to write this at the top of a function

```
global myVariable
```

Task 5.3: Creating game over!

Now that we are detecting when the bird hits a pipe we need to make a global gameOver flag (True or False) that starts at False but is switched to True when the bird collides with a pipe or the bottom of the screen.

We'll need this to change our display to show they lost. You will also need to change where you are awarding points to the player to make sure they aren't getting points when they're already dead.

Hint: Making a Boolean

To set up the flag or boolean variable.

```
myFlag = True
```

Task 5.4: Now what?

Now that we are detecting when the game is over we need to change the screen when it is. First we'll start with a background fill. You can choose any colour for this. (Use google to find different colours in rgb format)

Hint

To fill the screen with a colour it should look like:

`screen.fill((0,0,0))` ← this is RGB colouring so it will display black

Task 5.5: Some text

The next step of our game over screen is to tell the player that it is game over. To do this we'll be displaying some text on the screen, including"Game Over!" and their score.

Hint

To draw some text on the screen it should look like this:

```
screen.draw.text("Some Text", center = (400,300), fontsize = 60)
```

Task 5.6: Random pipes!

Right now the pipes are staying at the same y values every time they loop so let's use some random numbers to fix that.

Now instead of passing in the initial centre y value and gap we'll make them with the random module.

1. First you need to import the random module under where you've imported sys and pgzrun. This needs to be a `from [module] import *` import.

2. Inside your __init__ function above everything else, you need to generate a random gap value. This needs to be between 160 and 260.

3. Nest instead of passing in a centre y value you need to generate one between a minimum and maximum.
   Minimum: -250 + half the height of the pipes(300) + half the generated gap
   Maximum: 850 - (half the height of the pipes(300) + half the generated gap)
   Make sure you use // to halve the gap, otherwise it might crash.

4. Now you need to change your top and bottom pipes' y values to use these values and stop passing the constant ones in.

5. You also need to make sure that you're doing this when the pipes wrap around so they get a new random y value

Hint

To generate a random integer between two given numbers it should look like:

```
randint(50,60)
```

Remember to import random at the top of the page!

☑ CHECKPOINT ☑

If you can tick all of these off you can go to Part 6:

- ☐ Your game ends after the bird collides with the wall or pipes
- ☐ Your game has some random points and generates a score
- ☐ Your game displays a game over screen

★ BONUS 5.7: Score on screen

Waiting for the next lecture? Try adding this bonus feature!!

We are storing a score. Now we can try to display the score on the screen during the game play in order to know how many points we're getting. The code should look something like:

```
screen.draw.text(f"My Number is:
{myNum}",topleft=(0,0),color=(0,0,0), fontsize = 30)
```

★ BONUS 5.8: Interesting fonts!

Waiting for the next lecture? Try adding this bonus feature!!

We have some basic text on a screen to show that the game is over, but now you can make it your own. Feel free to add your own images off the internet. You can also play around with the text styling. A good resource to help with that is the website; https://pygame-zero.readthedocs.io/en/stable/ptext.html

Part 6: Make some methods!

We're going to improve our class by using methods to cut down on the repetition required for updating and drawing our pipes.

Task 6.1: Making an update method!

We want to make an update method that updates both pipes in our pipeGroup objects.

We'll start by making the method function and then we'll copy across the needed code. However, since pygame zero has specific functions that it uses differently than normal we can't call our method just "update()" instead call it something like updatePipes()

Hint

Much like a normal function to create a new method it looks like this:

```
def myMethod(self,parameter1,...):
```

Task 6.2: Implementing the method!

Now you need to go back to where you originally updated the pipes. Use this code as a basis to your method.
The method should change the x values of its top and bottom actors by 1. It should wrap the pipe around when it moves off screen. When it wraps around it should generate new y and gap values. It should take in "bird" as a parameter and detect whether it has collided with either pipe, changing gameOver when necessary.

Once this is added to the method you will have to change your original function to just call this method for each object (you can use a loop here)

Task 6.3: Making a draw method!

This method needs to draw both the top and bottom pipes of the group to the screen. Once this is created you should replace your original draw code to call this method for each object.

Remember that because of Pygame zero having specific function names this method should be called something other than draw()

☑ CHECKPOINT ☑

If you can tick all of these off you can go to some extensions or other bonuses:

- ☐ You have a pipe class that has methods
- ☐ You are updating the values and displaying the objects and the gameplay still works
- ☐ You have a working game!