Girls' Programming Network

2017

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: GPN Sydney and GPN Canberra

Girls' Programming Network - Sydney

Girls' Programming Network - Canberra

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

Writers

Renee Noble
Courtney Ross
Joanna Elliott
Amanda Meli
Adele Scott

Testers

Emily Aubin
Samantha Rampant
Leanne Magrath

Remember the mobile game Flappy Bird, and how it went viral on the internet? Remember how hard it was?

Let's make our own version of Flappy Bird, which can be just as hard as we want!

Part 0: Getting set up!

Goal: Let's get IDLE and our file setup before we start!

INTRO

Task 0.1: Set Up the File

Create a file where we are going to write the code for our game:

1) Create a new folder on the desktop called `flappy-bird`.
2) Download the `bg.png`, `bird.png` and `pipe.png` files to your new folder.
3) In your Python 3 IDLE click ***File*** and create a ***New File***.
4) Start by saving your file and calling it ***game.py***. Make sure it's saved to the folder you created on the desktop!

Now when you want to run your code, just click ***Run*** (or press ***F5*** on the keyboard)! Great! Now we're ready to code!

Task 0.2: Import pygame!

We're going to be using a python module named **pygame** today. It's going to help us create our game! We need to import this into our code:

1) `import` the pygame module into your code
2) Initialise pygame using `pygame.init()`

Try running your code. Nothing should happen, but now we're ready to get started!

Part 1: Displaying our first image!

To start off, let's get the game working with a background displayed!

Goal: Display the screen

Task 1.1: Displaying the screen

First we need to decide how big our game window is going to be. Then we can display the game window! We're going to make our game window 800 pixels wide, and 600 pixels tall.

1) Create a variable called `size_x` that stores the width of the window (800).
2) Create a variable called `size_y` that stores the height of the window (600).
3) Create the game window and a variable called `screen` that stores the game window.
   Hint: *If you're stuck, take a look at your pygame cheat sheet!*

Run your code! It should display the game window.

Note: You will not be able to close the game window by pressing the 'x' in the corner. Close the shell to close the game window.

A black screen is pretty boring! Let's make it interesting.

Goal: Display an image as the background.

Task 1.2: Displaying the background

In your file **game.py** write some code that will:

1) Load the background image (you downloaded it earlier, the file is called `bg.png`) and store it in a variable called `background_image`.
2) Draw the background image to the game window.
3) Update the game window.

Hint: *If you're stuck, take a look at your pygame cheat sheet!*

Part 2: The Bird is the word!

Now that we have a background, your flappy bird game needs a flappy bird!

Goal: Draw the flappy bird!

Task 2.1: Flappy Bird!

Let's load the bird and draw it to the screen!

1) Load the image of flappy bird (it's called `bird.png`!) and store it in a variable called `bird_image`.
2) Create two variables called `bird_x` and `bird_y` to store the x and y position of the flappy bird and give them values of 200 and 300.
3) Draw the flappy bird to the game window.
4) Update the game window.

Try playing around with placing the bird in different positions.

Now that we have the bird on the screen, we want to make the bird do something! First though, we need to create the game loop.

Goal: Create a game loop to keep updating the game

WHILE

Task 2.2: Create the game loop!

Let's create a while loop that keeps updating our game:

1) Create a variable called `game_running` that has a value of `True`.
2) Create a variable called `moving` and set it to an empty string.
3) Create the `while` loop that keeps going while `game_running` is true.
4) Update the game window inside the `while` loop.

Note: *Nothing new will happen currently!*

Now that the game is constantly looping, let's capture some input from the user.

Goal: Create an if statement that checks to see if the up or down arrow has been pressed.

IF

Task 2.3: Capturing events!

An event is some action that happens in the game. For example, pressing a key would be an event.

1) Get the list of events that are currently happening in pygame.
2) Create a `for` loop that checks every event.
3) Check to see if the event is for a pressed key.
4) If a key was pressed, check to see if it was the down arrow. If the down arrow was pressed, set the value of the variable `moving` to `"down"`. If the up arrow was pressed, set the value of `moving` to `"up"`.
5) Check to see if a key was released.
6) If a key was released, check to see if it was the down or up arrow. If it was either of those two keys, set the value of `moving` to be an empty string.
7) Print out the value of `moving` in each game loop.

Your program should now print 'up', 'down' or nothing, depending on what key was pressed!

Now that we're tracking to see if the up or down arrows were pressed, we actually want to move the flappy bird up and down!

Goal: Create an if statement that moves the bird up or down.

Task 2.4: Updating the bird's position

Now that we know where the user wants the bird to move, let's make the bird do it!

1) Create an if statement that checks if `moving` is equal to `"up"`, and if so decreases the `bird_y` value by 1.
2) Create an `elif` statement that checks if `moving` is equal to `"down"`, and if so increases the `bird_y` value by 1.
3) Draw the bird to the screen.

Hints:

- *Remember that* `pygame.display.update()` *always needs to be inside the game loop*
- *Remember that (0, 0) are the coordinates of the top left of the window and they increase to the right and down. So coordinates (x, y) represent x pixels to the right, and y pixels down.*

So our bird is now moving, but the game looks terrible! We're drawing way too many birds on the screen! This is known as 'trailing images'.

Goal: Make the bird only appear once on the game window

Task 2.5: Reducing trailing

We're going to fix the trailing images issue by re-drawing the background to the screen every time the game loops.

1) Draw the background to the screen again inside the game loop before drawing the bird.

Phew! That fixed it!

Part 3: The pipes!

Now that we have our flappy bird, and our flappy bird moves up and down, we need some obstacles to avoid.

Goal: Let's create a list of pipes that are spread out across the screen!

Task 3.1: Create a list of pipes

We're going to create a list of dictionaries to draw our pipes! Each dictionary is going to store the x and y positions of one pipe.

We need to create ten pipes. They should be be 200 pixels apart, and the first pipe should be 500 pixels from the left edge of the window. Also, each pipe should be 500 pixels from the top edge of the window. This means that our x positions for the ten pipes are [500, 700, 900, 1100, …] etc., and the y positions for the ten pipes are all 500.

1) Load the pipe image (it's called `pipe.png`) and store it in a variable called `pipe_image`.
2) Create a list called `pipe_x_positions` and set it equal to the ten x locations of the pipes, e.g. `[500, 700, 900, 1100, ...]`.
3) Create an empty list called `pipes`.
4) Create a `for` loop that loops through all ten `pipe_x_positions`.
5) Inside the for loop, create an empty dictionary called `new_pipe`.
6) Set `new_pipe['x']` to the x position of the pipe.
7) Set `new_pipe['y']` to the y position of the pipe.
8) Append `new_pipe` to the `pipes` list.

But we still need to display our pipes on the game window.

Goal: Draw the pipes to the game window

Task 3.2: Draw the pipes

1) Inside the game loop, create a `for` loop that draws all the pipes to the screen.

Check out your pipes! Don't they look awesome? Try increasing or decreasing the number of pipes that are created.

Our pipes are all the same size, which makes them pretty easy to avoid. Not to mention boring! Let's change how big they are.

Goal: Randomly decide how tall the pipes are

Task 3.3: Random sizes!

We're going to use the random module to randomly select how big the pipes are.

1) `import` the `random` module.
2) Use `random.randint()` to randomly select the y position of the pipe between 250 and 500.
3) Assign the randomly selected number to `pipe['y']`.

Now all our pipes should be different size! *Try running your program a few times and checking if the sizes are truly random!*

Our pipes are just sitting there though! Let's make them move!

Goal: Make the pipes move across the screen

Task 3.4: Make the pipes move

Let's make the pipes move across the screen!

1) Inside the game loop, after drawing the pipes, reduce the x value of each pipe by 1.

Now our pipes should be moving to the left!

We can still avoid all of the pipes just by moving flappy bird to the top of the screen. Let's make the game harder by making some of the pipes come from the top of the screen!

Goal: Randomly decide which pipes will be upside down!

★BONUS 3.5: Flip the pipe!

We're going to update the `for` loop where we created all the pipes with a new variable that stores whether the pipe was flipped or not.

1) At the end of the `for` loop where we created all the pipes, create a variable called `new_pipe['flipped']`, and use `random.randint()` to randomly give it a value of 0 or 1.
2) Still in the for loop, if the pipe should be flipped, update `new_pipe['y']` to be equal to the height of the game window minus the old value of `new_pipe['y']` minus the height of the pipe image.
3) Now inside the game loop, when drawing the pipes first check to see if it is flipped. If it is, transform the image to be upside down, then draw it.
4) If the pipe is not flipped, just drawit normally.

Run your code! How does the game look now?

Can you figure out why we updated the y position like that in step 2?

WHILE

Part 4: Collision Alert

Now that we have our flappy bird, and we have all the obstacles, we need to detect when our flappy bird hits them!

Goal: Detect when the flappy bird has collided with a pipe.

Task 4.1: Detect a Collision!

Get the position of the bird and pipes, and check to see if they intersect!

1) In the game loop, create a `for` loop that goes through the list of pipes.
2) Get the bounding rectangle of the pipe.
3) Get the bounding rectangle of the bird.
4) Check to see if the bird has collided with the pipe.
5) If the bird has collided with a pipe, print `"Oh no, you lose."`.

Your game should now print `"Oh no, you lose."` every time the flappy bird collides with a pipe.

Task 4.2: Stop the game!

Now that we know the game is over, let's update the value of `game_running` so that our game ends!

1) When a collision is detected, update the `game_running` variable to `False`.

The game should immediately pause!

★BONUS 4.3: Close the game!

Once the game is over, the user will probably want to close the game window. Create a loop that exits the game when the user presses 'q'.

1) At the top of your code, import the `sys` module.
2) After the game loop, create a new while loop that always runs.
3) Inside that while loop, check to see if the 'q' key was pressed.
4) If the 'q' key was pressed:
    a) Close the display window
    b) Call sys.exit()

The game should close!