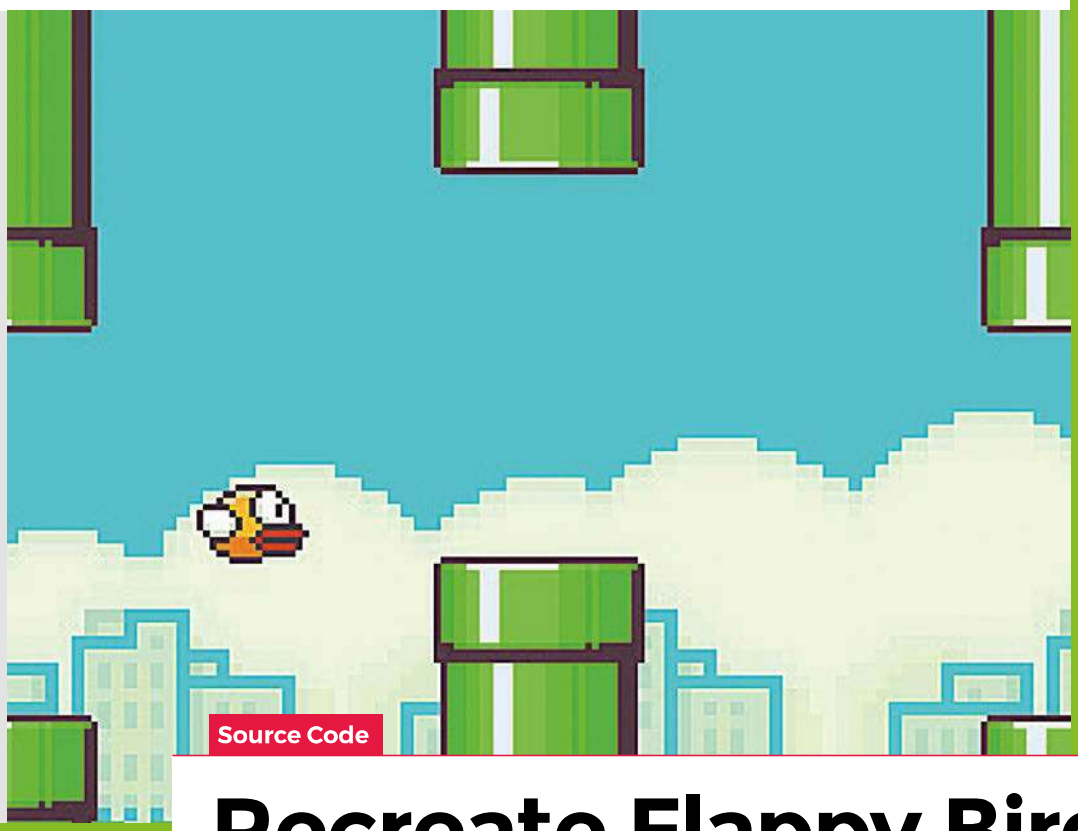


Flappy Bird: ridiculously big in 2014, at least for a while.

Source Code

Recreate Flappy Bird's flight mechanic

Learn how to create your own version of the simple-yet-addictive side-scroller

AUTHOR
RIK CROSS

Flappy Bird was released by programmer Dong Nguyen in 2013, and made use of a straightforward game mechanic to create an addictive hit. Tapping the screen provided 'lift' to the main character, which is used strategically to navigate through a series of moving pipes. A point is scored for each pipe successfully passed. The idea proved so addictive that Nguyen eventually regretted his creation and removed it from the Google and Apple app stores. In this article, I'll show you how to recreate this simple yet time-consuming game, using Python and Pygame Zero.

The player's motion is very similar to that employed in a standard platformer: falling down towards the bottom of the screen under gravity. See the article, *Super Mario-style jumping physics* in Wireframe #7 for more on creating this type of movement. Pressing a button (in our case, the **SPACE** bar) gives the player some upward thrust by setting its velocity to a negative value (i.e. upwards) larger than the value of gravity acting downwards. I've adapted and used two different images for the sprite (made by Imaginary Perception and available on **opengameart.org**), so that it looks like it's flapping its wings to generate lift and move upwards.

Sets of pipes are set equally spaced apart horizontally, and move towards the player slowly each frame of the game. These pipes are stored as two lists of rectangles, `top_pipes` and `bottom_pipes`, so that the player can attempt to fly through gaps between the top and bottom pipes. Once a pipe in the `top_pipes` list reaches the left side of the screen past the player's position, a `score` is incremented and the top and corresponding bottom pipes are removed from their respective lists. A new set of pipes is created at the right edge of the screen, creating a continuous challenge for the player. The y-position of the gap between each newly created pair of pipes is decided randomly (between minimum and maximum limits), which is used to calculate the position and height of the new pipes.

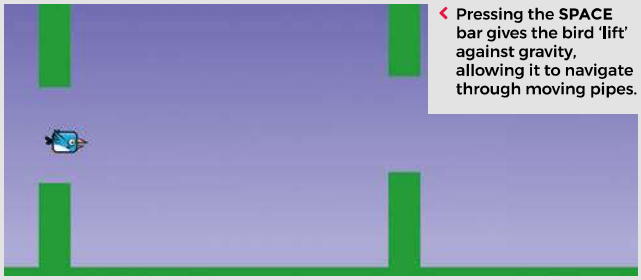
The game stops and a Game Over message appears if the player collides with either a pipe or the ground. The collision detection in the game uses the `player.colliderect()` method, which checks whether two rectangles overlap. As the player sprite isn't exactly rectangular, it means that the collision detection isn't pixel-perfect, and improvements could be made by using a different approach. Changing the values for `GRAVITY`, `PIPE_GAP`, `PIPE_SPEED`, and `player.flap_velocity` through a process of trial and error will result in a game that has just the right amount of frustration! You could even change these values as the player's score increases, to add another layer of challenge.

A flapping bird in Python

Here's Rik's code, which recreates *Flappy Bird*'s avian mayhem in Python. To get it running on your system, you'll need to install Pygame Zero – you can find instructions at **wfmag.cc/pgzero**

Download the code from GitHub: **wfmag.cc/wfmag29**

```
from random import randint

WIDTH = 1000
HEIGHT = 600

# pipes are dark green, move 2 pixels per frame and
# have a gap of 150 pixels between top and bottom pipes
PIPE_COLOUR = (38,155,29)
PIPE_SPEED = 2
PIPE_GAP = 150

GRAVITY = 0.2

# create top and bottom pipes, with a gap in between
top_pipes = [
    Rect((500,0),(50,200)),
    Rect((1000,0),(50,300))
]

bottom_pipes = [
    Rect((500,200 + PIPE_GAP), (50,HEIGHT - 200 - PIPE_GAP)),
    Rect((1000,300 + PIPE_GAP), (50,HEIGHT - 300 - PIPE_GAP))
]

player = Actor('player-down',(100,400))
# define initial and flap velocities
player.y_velocity = 0
player.flap_velocity = -5
player.score = 0

playing = True

def update():

    global playing
    if playing:

        # space key to flap
        if keyboard.space and player.y_velocity > 0:
            player.y_velocity = player.flap_velocity

        # acceleration is rate of change of velocity
        player.y_velocity += GRAVITY
        # velocity is rate of change of position
        player.y += player.y_velocity

        # player image depends on velocity
        if player.y_velocity > 0:
            player.image = 'player-down'
        else:
            player.image = 'player-up'
        for pipe_list in top_pipes, bottom_pipes:
            for pipe in pipe_list:
                pipe.x -= PIPE_SPEED
                if pipe.x < -50:
                    pipe_list.remove(pipe)

        # create new pipes
        if len(top_pipes) < 2:
            player.score += 1
            h = randint(150,350)
            top_pipes.append(Rect((1000,0),(50,h)))
            bottom_pipes.append(Rect((1000,h + PIPE_GAP),(50,
HEIGHT - h - PIPE_GAP)))

        # game over if player collides with a pipe...
        for p in top_pipes + bottom_pipes:
            if player.colliderect(p):
                playing = False

        # ...or touches the ground
        if player.y > (HEIGHT - 20):
            playing = False

def draw():

    if playing:

        screen.clear()

        screen.blit('background', (0,0))

        for pipe in top_pipes + bottom_pipes:
            screen.draw.filled_rect(pipe, PIPE_COLOUR)

        screen.draw.text(str(player.score), (20, 20),
fontsize=40, color="white")

        player.draw()

    else:

        screen.draw.text('Game Over!', (420, 200),
fontsize=40, color="white")
```



Pressing the **SPACE** bar gives the bird 'lift' against gravity, allowing it to navigate through moving pipes.