PROJECT 2 - FLAPPY BIRD GAME

By Disha Bhaglal

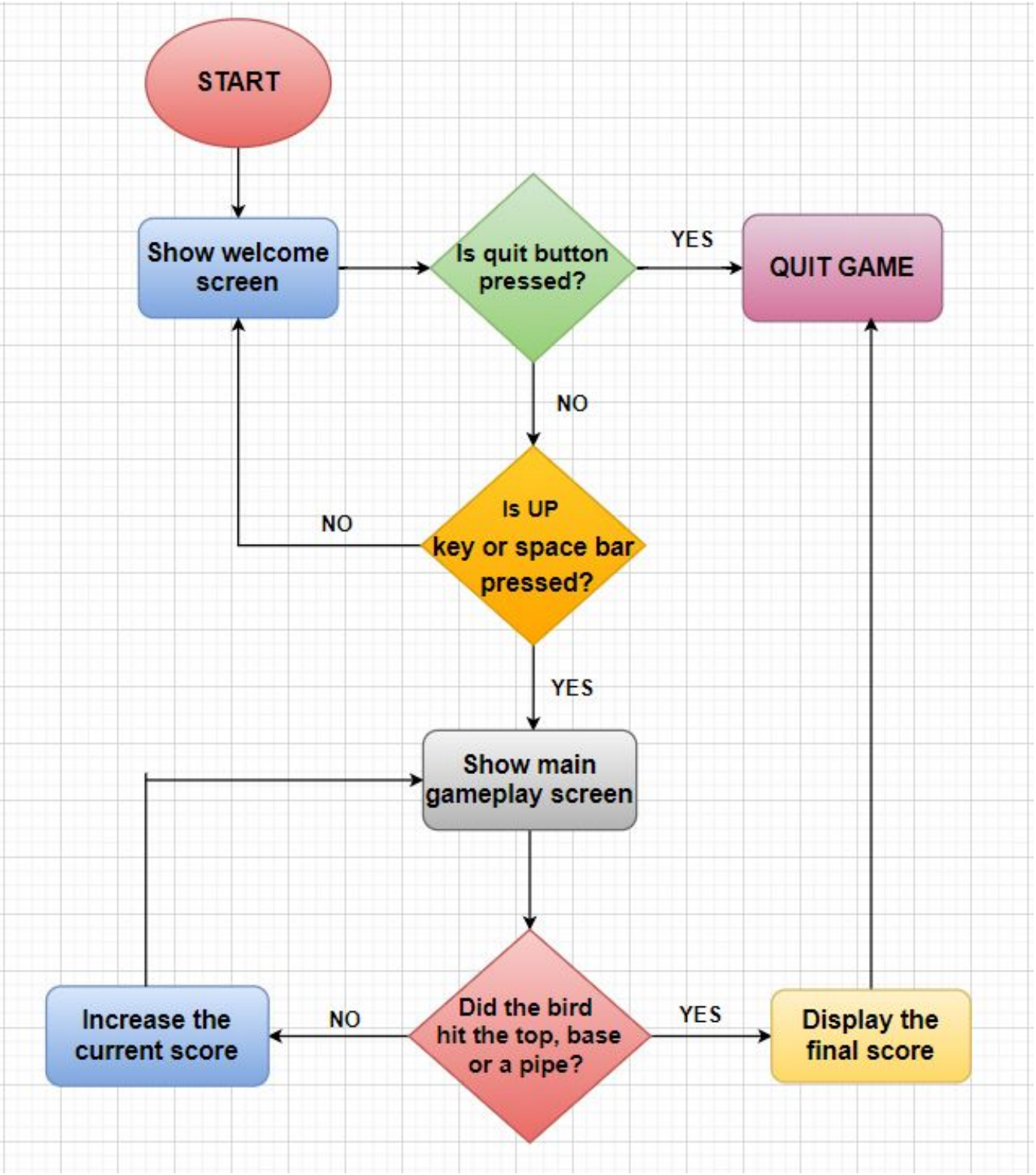
INTRODUCTION

Flappy bird was originally released as a mobile game where you tap the screen to make the bird fly. If the bird hits the pipes or the edges of the screen, the game ends and the player needs to restart. In this project, we have made a computer version of the game where the bird will be controlled using the up key or the space bar.

TECHNOLOGY USED

We will be using Python language for writing the code. We will also be using Pygame which is a cross-platform set of Python modules designed for writing video games. It includes computer graphics and sound libraries designed to be used with the Python programming language. Pygame is suitable to create client-side applications that can be potentially wrapped in a standalone executable. So, a prior knowledge of python and pygame is required for this project.

FLOW CHART


START
Show welcome screen
Is quit button pressed?
YES
QUIT GAME
NO
Is UP key or space bar pressed?
NO
YES
Show main gameplay screen
Increase the current score
NO
Did the bird hit the top, base or a pipe?
YES
Display the final score

CODE

The following is the link to the github repository of the code for this project:
https://github.com/DishaBhaglal/Flappy_Bird.git

EXPLANATION FOR THE CODE

Now, we will go through the code piece by piece.

```
1   import random
2   import sys
3   import pygame
4   from pygame.locals import *
5
```

Figure 1

In figure 1, we are importing the necessary modules. We will use random for generating random numbers for our game. sys.exit from the sys module will be used to exit the program. In line 3 and 4 we are importing Pygame and the basic Pygame imports respectively.

```
6   #Global variables for the game
7   fps = 32
8   screen_width = 289
9   screen_height = 511
10  screen = pygame.display.set_mode((screen_width,screen_height))
11  ground_y = screen_height*0.8
12  game_images = {}
13  game_sounds = {}
14  player = 'gallery/images/bird.png'
15  background = 'gallery/images/background.png'
16  pipe = 'gallery/images/pipe.png'
17  title = 'gallery/images/title.png'
18
```

Figure 2

Here, we are declaring various global variables for our game. In line 7-9, we set some value for fps(frames per second), screen_width and screen_height. Then we are creating the screen with screen_width and screen_height as an argument for the pygame.display.set_mode() function. Then from line 11-13 we create a ground-y variable which will give the y-coordinate for our base image, and 2 dictionaries game_images and game_sounds which will contain our various images and sounds used for the game. From 14-17, we store the images of the player (bird), background, pipe and the title in these variables by giving their paths.

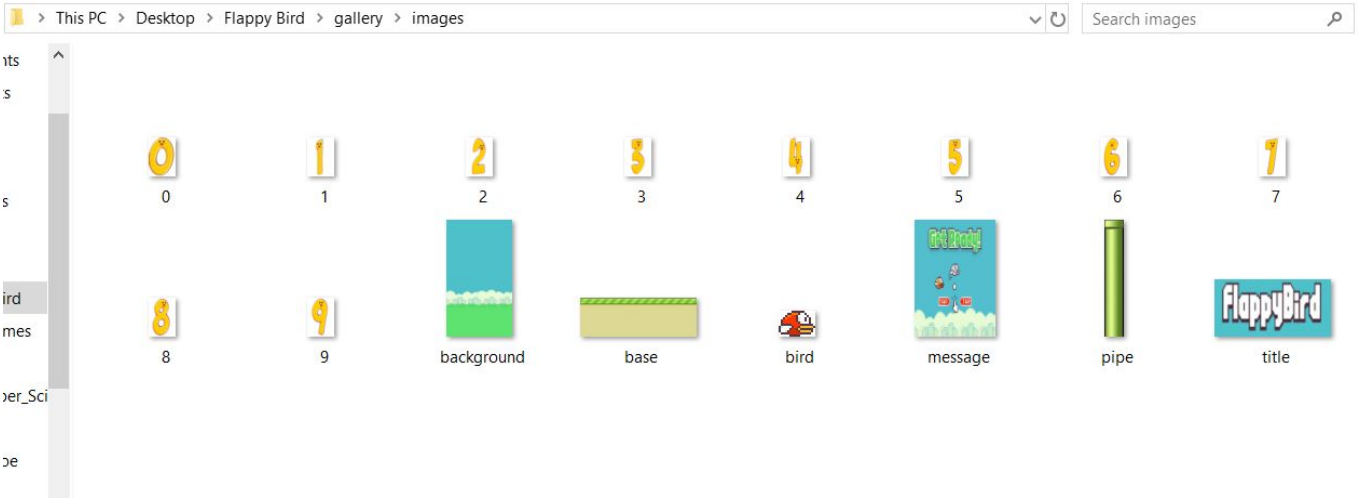


Figure 3

Figure 3 shows the images used in this project.

Figure 4

This figure shows the audios used for the project.

```
167  if __name__ == "__main__":
168      pygame.init()
169      fps_clock = pygame.time.Clock()
170      pygame.display.set_caption('Flappy Bird')
171      game_images['numbers'] = (
172          pygame.image.load('gallery/images/0.png').convert_alpha(),
173          pygame.image.load('gallery/images/1.png').convert_alpha(),
174          pygame.image.load('gallery/images/2.png').convert_alpha(),
175          pygame.image.load('gallery/images/3.png').convert_alpha(),
176          pygame.image.load('gallery/images/4.png').convert_alpha(),
177          pygame.image.load('gallery/images/5.png').convert_alpha(),
178          pygame.image.load('gallery/images/6.png').convert_alpha(),
179          pygame.image.load('gallery/images/7.png').convert_alpha(),
180          pygame.image.load('gallery/images/8.png').convert_alpha(),
181          pygame.image.load('gallery/images/9.png').convert_alpha()
182          )
183      game_images['message'] = pygame.image.load('gallery/images/message.png').convert_alpha()
184      game_images['base'] = pygame.image.load('gallery/images/base.png').convert_alpha()
185      game_images['pipe'] = (
186          pygame.transform.rotate(pygame.image.load(pipe).convert_alpha(), 180),
187          pygame.image.load(pipe).convert_alpha()
188          )
189      game_images['background'] = pygame.image.load(background).convert_alpha()
190      game_images['player'] = pygame.image.load(player).convert_alpha()
191      game_images['title'] = pygame.image.load(title).convert_alpha()
192
193      #Game Sounds
194      game_sounds['die'] = pygame.mixer.Sound('gallery/audio/die.wav')
195      game_sounds['hit'] = pygame.mixer.Sound('gallery/audio/hit.wav')
196      game_sounds['point'] = pygame.mixer.Sound('gallery/audio/point.wav')
197      game_sounds['swoosh'] = pygame.mixer.Sound('gallery/audio/swoosh.wav')
198      game_sounds['wing'] = pygame.mixer.Sound('gallery/audio/wing.wav')
199
200      while True:
201          welcomeScreen()
202          mainGame()
203
```

Figure 5

Let’s see this block outside any function first, From line 167, our main game will start and we will initialize all pygame modules using pygame.init(). We create fps_clock variable to help us track time at a moment using pygame.tick.Clock() function. Then we will give a title to our main game window in line 170. From line 171-182, we are storing the images in a tuple with first, which we are then assigning to the ’numbers’ key in the game_images dictionary. We use pygame.image.load() with paths of the images as arguments along with convert_alpha() to change the pixel format of an image including per pixel alphas. Similarly, from line 183-191 we have added the images of the message, base, pipe, background, player and the title, into the dictionary using various keys. For pipe, we also added an inverted pipe image by using pygame.transform.rotate() function and rotating by 180 degrees.

From line 193-198, we are adding the sounds to the game_sounds dictionary using various keys. It is similar to what we did for images but here we use pygame.mixer.Sound() function with the paths for various sounds as the

argument for storing the sounds. Then we start a loop calling the welcomeScreen() and mainGame() functions.

```
19  def welcomeScreen():
20      player_x = int(screen_width/8)
21      player_y = int((screen_height - game_images['player'].get_height())/2)
22      message_x = int((screen_width - game_images['message'].get_width())/2)
23      message_y = int(screen_height*0.2)
24      title_x = int((screen_width - game_images['message'].get_width())/2)
25      title_y = int(screen_height*0.04)
26      base_x = 0
27      while True:
28          for event in pygame.event.get():
29              if event.type == QUIT or (event.type == KEYDOWN and event.key == K_ESCAPE):
30                  pygame.quit()
31                  sys.exit()
32              elif event.type == KEYDOWN and (event.key == K_SPACE or event.key == K_UP):
33                  return
34              else:
35                  screen.blit(game_images['background'],(0,0))
36                  screen.blit(game_images['message'],(message_x,message_y))
37                  screen.blit(game_images['player'],(player_x,player_y))
38                  screen.blit(game_images['base'],(base_x,ground_y))
39                  screen.blit(game_images['title'],(title_x,title_y))
40                  pygame.display.update()
41                  fps_clock.tick(fps)
```

Figure 6

Here, we define our welcomeScreen() function which will display the welcome screen on starting the game. From line 20-25, we are assigning the values of the x-coordinate and y-coordinate for the player, message and title images. We have selected the arguments by trial and error and you can alter the values for exploring. We also give the x-coordinate of base here. Then in line 27 we start a while loop which will always be True and thus will start a loop which will not stop unless the programme is quit. Here we use make a for loop for analysing all the events taking place using pygame.event.get(). Then we check if we have quit the screen by performing a quit type of event or by pressing the escape key. If yes, pygame will quit and the programme will exit. Then we will check the next condition i.e whether we clicked the up key or the space button. If yes, we will return from the function. And if we click no key or button, we will display the welcome screen. For that we will blit (place) the background, message, payer, base and title images using screen.blit() function. Then we will update our window using pygame.display.update(). Then we update our clock variable with fps value as argument to show just 32 frames per second.

```
43  def mainGame():
44      score = 0
45      player_x = int(screen_width/8)
46      player_y = int(screen_height/2)
47      base_x = 0
48
49      newPipe1 = getRandomPipe()
50      newPipe2 = getRandomPipe()
51
52      upperPipes = [
53          {'x': screen_width+200, 'y': newPipe1[0]['y']},
54          {'x': screen_width+200+(screen_width/2), 'y': newPipe2[0]['y']}
55      ]
56
57      lowerPipes = [
58          {'x': screen_width+200, 'y': newPipe1[1]['y']},
59          {'x': screen_width+200+(screen_width/2), 'y': newPipe2[1]['y']}
60      ]
61
62      pipeVelX = -4
63
64      playerVelY = -9
65      playerMaxVelY = 10
66      playerMinVelY = -8
67      playerAccY = 1
68
69      playerFlapVel = -8
70      playerFlapped = False
71
72
73      while True:
74          for event in pygame.event.get():
75              if event.type == QUIT or (event.type == KEYDOWN and event.key == K_ESCAPE):
76                  pygame.quit()
77                  sys.exit()
78              if event.type == KEYDOWN and (event.key == K_SPACE or event.key == K_UP):
79                  if player_y > 0:
80                      playerVelY = playerFlapVel
81                      playerFlapped = True
82                      game_sounds['wing'].play()
83
84          crashTest = isCollide(player_x, player_y, upperPipes, lowerPipes)
85          if crashTest:
86              return
87
88          playerMidPos = player_x + game_images['player'].get_width()/2
89          for pipe in upperPipes:
90              pipeMidPos = pipe['x'] + game_images['pipe'][0].get_width()/2
91              if pipeMidPos<= playerMidPos < pipeMidPos + 4:
92                  score +=1
93                  print(f"Your Score is {score}")
94                  game_sounds['point'].play()
95
96          if playerVelY <playerMaxVelY and not playerFlapped:
97              playerVelY += playerAccY
```

Figure 7

In Figure 7, we start defining our mainGame() function. We initialize the variable score with 0, and also give the coordinates for player image and base again. Then we create 2 pipes for blitting on the screen using getRandomPipe() which we will define later. Then from line 52-60, we create a list of upper pipes (inverted ones) and lower pipes with their x and y coordinates. Again we have chosen values by trial and error. Then from line 61-68 we declare variables for velocities in different directions for the bird. We also provide an acceleration variable. playerFlapVel is the velocity while flapping and playerFlapped is set to false (which is true only if the bird flaps). Then again we check for events. First for exiting the game and exit the game if true. Then we check if the up key or spacebar is pressed. If yes,we check if the player is below the screen top and if yes, we make some updates and play the sound of the wing using .play(). After this, we check if we have crashed using the isCollide() function we will define soon. If true, we will return from the function. From line 88-94, we will check and update the scores. Using the player's, mid position and the positions of the pipes, we increase the score if we cross a pipe and print it in the console. Also we play the point sound for crossing each pipe. Then if the player velocity in y direction has not yet become max, we will provide the acceleration.

```
99          if playerFlapped:
100             playerFlapped = False
101         playerHeight = game_images['player'].get_height()
102         player_y = player_y + min(playerVelY, ground_y - player_y - playerHeight)
103
104         for upperPipe, lowerPipe in zip(upperPipes, lowerPipes):
105             upperPipe['x'] += pipeVelX
106             lowerPipe['x']  += pipeVelX
107
108         if 0<upperPipes[0]['x']<5:
109             newPipe = getRandomPipe()
110             upperPipes.append(newPipe[0])
111             lowerPipes.append(newPipe[1])
112
113         if upperPipes[0]['x'] < -game_images['pipe'][0].get_width():
114             upperPipes.pop(0)
115             lowerPipes.pop(0)
116
117         screen.blit(game_images['background'], (0, 0))
118         for upperPipe, lowerPipe in zip(upperPipes, lowerPipes):
119             screen.blit(game_images['pipe'][0], (upperPipe['x'], upperPipe['y']))
120             screen.blit(game_images['pipe'][1], (lowerPipe['x'], lowerPipe['y']))
121         screen.blit(game_images['base'], (base_x, ground_y))
122         screen.blit(game_images['player'], (player_x, player_y))
123
124         myDigits = [int(x) for x in list(str(score))]
125         width = 0
126         for digit in myDigits:
127             width += game_images['numbers'][digit].get_width()
128         Xoffset = (screen_width - width)/2
129
130         for digit in myDigits:
131             screen.blit(game_images['numbers'][digit], (Xoffset, screen_height*0.12))
132             Xoffset += game_images['numbers'][digit].get_width()
133         pygame.display.update()
134         fps_clock.tick(fps)
135
```

Figure 8

In line 99 we update playerFlpped. And then the position of the bird. In lines 104-107, we move the pipes to the left. Then from line 108-112, we add a new pipe when the first one is about to cross the leftmost part of the screen. From 113-116, we see if the pipe is out of the screen and if yes, we remove it. From line 113-134, we blit our pipes and the score on our screen and then update the display screen. For score we first access all the digits of the score (if more than 1 digit score) and blit the require images. We update our clock again.

```
136  def isCollide(player_x, player_y, upperPipes, lowerPipes):
137      if player_y>ground_y-25 or player_y<0:
138          game_sounds['hit'].play()
139          return True
140
141      for pipe in upperPipes:
142          pipeHeight = game_images['pipe'][0].get_height()
143          if (player_y < pipeHeight + pipe['y']) and (abs(player_x - pipe['x']) < game_images['pipe'][0].get_width() - 15):
144              game_sounds['hit'].play()
145              return True
146
147      for pipe in lowerPipes:
148          if (player_y + game_images['player'].get_height() > pipe['y']) and (abs(player_x - pipe['x']) < game_images['pipe'][0].get_width() - 15):
149              game_sounds['hit'].play()
150              return True
151
152      return False
153
```

Figure 9

In the isCollide() function, first we check if we have hit the top or the base in line 137-140. Then from 141-146, we look for collision with upper pipes, We compare the position of the bird with that of the pipe to check for the collision. Similarly we repeat for lower pipes. If any of the collision conditions are true, we play the hit sound and return True.

```
155  def getRandomPipe():
156      pipeHeight = game_images['pipe'][0].get_height()
157      offset = screen_height/3
158      y2 = offset + random.randrange(0, int(screen_height - game_images['base'].get_height() - 1.2*offset))
159      pipeX = screen_width + 10
160      y1 = pipeHeight - y2 + offset
161      pipe = [
162          {'x': pipeX, 'y': -y1},
163          {'x': pipeX, 'y': y2}
164      ]
165      return pipe
166
```

Figure 10

Here, we are defining our getRandomPipe() function, We store the height of pipe in pipeHeight variable, use offset variable to store one-third of screen_width, then we assign the values of the x and y coordinates for the pipes using random functions at equal distances, but with different sizes of upper an lower pipes. Then we store the coordinates in a list named pipe, and return it.