Flappy Bird

Mini-Project
Submitted in partial fulfilment of the requirements
Of University of Mumbai for the Degree of Engineering
(Computer Engineering)

by

Aditya Nitesh Singh (A-10)	TU3F1819005
Pritam P Nikalaje (A-12)	TU3F1819007
Tejas Vijay Adhav (A-13)	TU3F1819008

Under the Guidance of
DHAMELE SONALI JAIDEEP

Department of Computer Engineering

TERNA ENGINEERING COLLEGE

Plot No. 12, Sector-22, Phase-II, Opp. Railway Station,
Nerul (W), Navi Mumbai-400706.

University Of Mumbai
April - 2020

TERNA ENGINEERING COLLEGE

NERUL, NAVI MUMBAI

CERTIFICATE

This is to certify that

Aditya Nitesh Singh (A-10)	TU3F1819005
Pritam P Nikalaje (A-12)	TU3F1819007
Tejas Vijay Adhav (A-13)	TU3F1819008

has satisfactorily completed the requirements of the DSP Mini-Project

entitled

Flappy Bird

as prescribed by ***University of Mumbai*** *under the guidance of*

DHAMELE SONALI JAIDEEP

TABLE OF CONTENTS

1. Abstract

This mini project of the subject OSTL gives the implementation of a gaming program.
This report consists of the implementation program, design and structure of the code and program, snapshot representation of the output for the game and technical details that were considered while developing the program.

2. Introduction

Python is a great object-oriented, interpreted and interactive programming language. It has modules, classes, exceptions, very high dynamic data types and dynamic typing. There are interfaces to many system calls and libraries as well as to various windowing systems.

This project shows an implementation of a game developed using Python programming language making a considerable use of Pygame. Other libraries and modules are also put to useful implementation in the program. This program has made use of computer graphics and sound libraries which are designed to be used with the Python Programming Language.

The game developed using Python programming language with Pygame is a game named Flappy Bird. The game accepts the specific keyboard strokes to reflect in the game. The gameplay of Flappy Bird is described below:

Flappy Bird:

Flappy Bird is a side-scrolling game featuring 2D retro style graphics. The objective of the game is to direct a Flappy Bird which moves continuously in the right direction between the sets of Mario like pipes. They are green pipes which block the path of the Flappy Bird from below and above. The user has to direct the Flappy Bird through the gap between the pipes. The gap between the pipes can be at different positions, at different height from the ground level of

the game window. The gap position depends on the respective lengths of the pipes positioned at below and above in the gaming window.

If the Flappy Bird touches any of the pipes the player loses. The Flappy Bird briefly flaps upward each time the player taps the spacebar or up arrow button on the keyboard. If the spacebar or the up-arrow button is not tapped the Flappy Bird falls under the influence of gravity. If the Flappy Bird falls under gravity and touches the ground, the player loses the game.

Each time the user is able to direct the Flappy Bird between the green pipes, the user earns a point. The points are added for a user and each count is displayed by logging in the terminal. The score count is also displayed in the gaming window while the player is playing the game.

If the player loses the game by hitting the Flappy Bird in any of the pipes or when the Flappy Bird touches the ground, the screen comes to a standstill. When the player taps the spacebar and up-arrow key the screen resets to the original position and starts the game again.

This game is developed with Python Programming Language with use of libraries and modules which are described in detail below:

The modules used in the program are Pygame, ramdom, sys.

Pygame:

Pygame is a cross-platform set of Python modules designed for writing the video games. It includes the computer graphics and sound libraries designed to be used with Python Programming Language.

Pygame makes the use of Simple DirectMedia Layer (SDL) library, with the intention of allowing the real-time computer game development without using the low-level mechanics and derivatives. This is based on the assumption that the most expensive functions inside the games can be extracted from the game logic and making it possible to use a high level programming language such as Python to structure the game.

Applications using pygame can run on Android phones and tablets with support of sound, vibration, keyboard and accelerometer are supported.

Game Structure:

In the Flappy Bird game the pygame functions and some other functions are created for the implementation. The game is set with a Frames per second as 32. The screen width and screen height is set equal to fixed value. Then the code "pygame.display.set_mode()" is used to initialize a window or the screen for display. Display is a module and set_mode is a function in that module. It creates an instance of pygame.Surface class and returns that will display a Surface.

Pygame will register all events from the user into an event queue which can be received by the code 'pygame.event.get()'. Every element in this queue is an

Event object and they will all have the attribute type, which is an integer of what kind of event is it. This function is used for all events in the game like closing the game when the window is closed and when the player taps the spacebar or the up-arrow button.

The function ‘pygame.display.update()’ will run when no input is given, when no events takes place. It updates the entire surface only if no arguments are passed.

Each list is created for upper and lower pipes which are displayed in the game. Each list consists of two dictionaries with inputs regarding the pipes. The velocities of pipes moving in left direction, player velocity, player maximum descending and ascending velocities of Flappy Bird are given constant values. The velocities are calculates and changed when a new event is encountered.

A crashTest function is used to get to know if the player is crashed or not. The function returns true if the bird has crashed. A function always checks for the score.

The pipes are constantly moved to the left and a new pipe is added when the first pipe is about to cross the leftmost part of the screen. The moment the pipe is out of the screen it is removed. Then all the components are displayed for the game and the audio is added which is used with every event assigned to it.

In the main function the game starts and it is where all pygame modules are initialized. In the main function all the images and game sounds are implemented and the functions are called to finally implement the game.

3. Design and Implementation

```
import random
import sys
import pygame
from pygame.locals import *


FPS = 32
SCREENWIDTH = 289
SCREENHEIGHT = 511
SCREEN = pygame.display.set_mode((SCREENWIDTH,
SCREENHEIGHT))
GROUNDY = SCREENHEIGHT * 0.8
GAME_SPRITES = {}
GAME_SOUNDS = {}
PLAYER = 'gallery/sprites/bird.png'
BACKGROUND = 'gallery/sprites/background.png'
PIPE = 'gallery/sprites/pipe.png'

def welcome():
    playerx = int(SCREENWIDTH/5)
    playery = int((SCREENHEIGHT -
GAME_SPRITES['player'].get_height())/2)
    basex = 0
    while True:
        for event in pygame.event.get():
```

```
        # if user clicks on cross button, close the game
        if event.type == QUIT or (event.type==KEYDOWN and event.key
== K_ESCAPE):
            pygame.quit()
            sys.exit()

        # If the user presses space or up key, start the game for them
        elif event.type==KEYDOWN and (event.key==K_SPACE or
event.key == K_UP):
            return
        else:
            SCREEN.blit(GAME_SPRITES['background'], (0, 0))
            SCREEN.blit(GAME_SPRITES['player'], (playerx, playery))
            SCREEN.blit(GAME_SPRITES['base'], (basex, GROUNDY))
            pygame.display.update()
            FPSCLOCK.tick(FPS)

def mainGame():
    score = 0
    playerx = int(SCREENWIDTH/5)
    playery = int(SCREENWIDTH/2)
    basex = 0

    # Create 2 pipes for blitting on the screen
    newPipe1 = getRandomPipe()
    newPipe2 = getRandomPipe()
```

```
    # my List of upper pipes
    upperPipes = [
        {'x': SCREENWIDTH+200, 'y':newPipe1[0]['y']},
        {'x': SCREENWIDTH+200+(SCREENWIDTH/2),
'y':newPipe2[0]['y']},
    ]
    # my List of lower pipes
    lowerPipes = [
        {'x': SCREENWIDTH+200, 'y':newPipe1[1]['y']},
        {'x': SCREENWIDTH+200+(SCREENWIDTH/2),
'y':newPipe2[1]['y']},
    ]

    pipeVelX = -4 #Velocity of pipes moving towards left side

    playerVelY = -9 #Player's velocity along y-axis while flapping
    playerMaxVelY = 10 #Max velocity of Player along Y-axis i.e max
descending speed
    playerMinVelY = -8 #Min velocity of Player along Y-axis i.e max
ascending speed

    playerAccY = 1 #Player's downward acceleration

    playerFlapAccv = -8 # velocity while flapping
    playerFlapped = False # It is true only when the bird is flapping
```

```
    while True:
        for event in pygame.event.get():
            if event.type == QUIT or (event.type == KEYDOWN and event.key
== K_ESCAPE):
                pygame.quit()
                sys.exit()
            if event.type == KEYDOWN and (event.key == K_SPACE or
event.key == K_UP):
                if playery > 0:
                    playerVelY = playerFlapAccv
                    playerFlapped = True
                    GAME_SOUNDS['wing'].play()


        crashTest = isCollide(playerx, playery, upperPipes, lowerPipes) # This
function will return true if the player is crashed
        if crashTest:
            return

        #check for score
        playerMidPos = playerx + GAME_SPRITES['player'].get_width()/2
        for pipe in upperPipes:
            pipeMidPos = pipe['x'] + GAME_SPRITES['pipe'][0].get_width()/2
            if pipeMidPos<= playerMidPos < pipeMidPos +4:
                score +=1
                print(f"Your score is {score}")
                GAME_SOUNDS['point'].play()
```

```
        if playerVelY <playerMaxVelY and not playerFlapped:
            playerVelY += playerAccY

        if playerFlapped:
            playerFlapped = False
        playerHeight = GAME_SPRITES['player'].get_height()
        playery = playery + min(playerVelY, GROUNDY - playery -
playerHeight)

        # move pipes to the left
        for upperPipe , lowerPipe in zip(upperPipes, lowerPipes):
            upperPipe['x'] += pipeVelX
            lowerPipe['x'] += pipeVelX

        # Add a new pipe when the first is about to cross the leftmost part of
the screen
        if 0<upperPipes[0]['x']<5:
            newpipe = getRandomPipe()
            upperPipes.append(newpipe[0])
            lowerPipes.append(newpipe[1])

        # if the pipe is out of the screen, remove it
        if upperPipes[0]['x'] < -GAME_SPRITES['pipe'][0].get_width():
            upperPipes.pop(0)
            lowerPipes.pop(0)
```

```
        # Lets blit our sprites now i.e display all the components
        SCREEN.blit(GAME_SPRITES['background'], (0, 0))
        for upperPipe, lowerPipe in zip(upperPipes, lowerPipes):
            SCREEN.blit(GAME_SPRITES['pipe'][0], (upperPipe['x'],
upperPipe['y']))
            SCREEN.blit(GAME_SPRITES['pipe'][1], (lowerPipe['x'],
lowerPipe['y']))

        SCREEN.blit(GAME_SPRITES['base'], (basex, GROUNDY))
        SCREEN.blit(GAME_SPRITES['player'], (playerx, playery))
        myDigits = [int(x) for x in list(str(score))]
        width = 0
        for digit in myDigits:
            width += GAME_SPRITES['numbers'][digit].get_width()
        Xoffset = (SCREENWIDTH - width)/2

        for digit in myDigits:
            SCREEN.blit(GAME_SPRITES['numbers'][digit], (Xoffset,
SCREENHEIGHT*0.12))
            Xoffset += GAME_SPRITES['numbers'][digit].get_width()
        pygame.display.update()
        FPSCLOCK.tick(FPS)

def isCollide(playerx, playery, upperPipes, lowerPipes):
    if playery> GROUNDY - 25  or playery<0:
        GAME_SOUNDS['hit'].play()
```

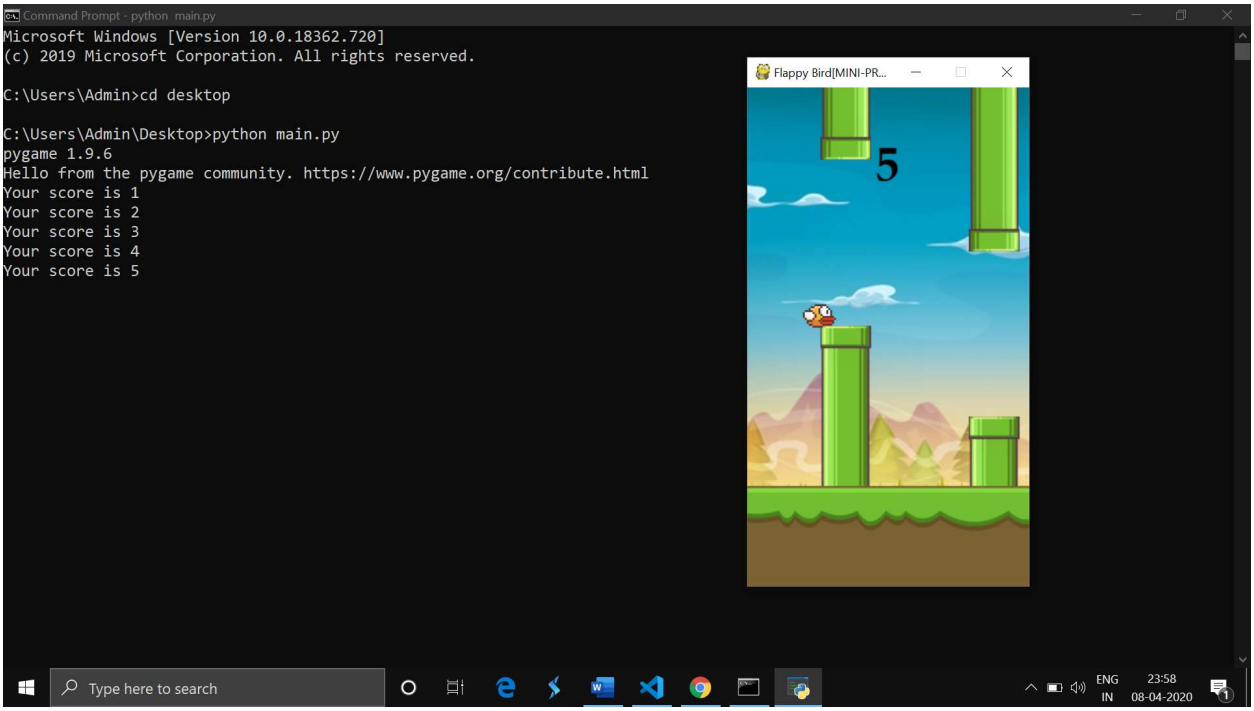
```
        return True

    for pipe in upperPipes:
        pipeHeight = GAME_SPRITES['pipe'][0].get_height()
        if(playery < pipeHeight + pipe['y'] and abs(playerx - pipe['x']) <
GAME_SPRITES['pipe'][0].get_width()):
            GAME_SOUNDS['hit'].play()
            return True

    for pipe in lowerPipes:
        if (playery + GAME_SPRITES['player'].get_height() > pipe['y']) and
abs(playerx - pipe['x']) < GAME_SPRITES['pipe'][0].get_width():
            GAME_SOUNDS['hit'].play()
            return True

    return False

def getRandomPipe():

    pipeHeight = GAME_SPRITES['pipe'][0].get_height()
    offset = SCREENHEIGHT/3
    y2 = offset + random.randrange(0, int(SCREENHEIGHT -
GAME_SPRITES['base'].get_height()  - 1.2 *offset))
    pipeX = SCREENWIDTH + 10
    y1 = pipeHeight - y2 + offset
    pipe = [
        {'x': pipeX, 'y': -y1}, #upper Pipe
```

```
        {'x': pipeX, 'y': y2} #lower Pipe
    ]
    return pipe




if __name__ == "__main__":
    # This will be the main point from where our game will start
    pygame.init() # Initialize all pygame's modules
    FPSCLOCK = pygame.time.Clock()
    pygame.display.set_caption('Flappy Bird[MINI-PROJECT]')
    GAME_SPRITES['numbers'] = (
        pygame.image.load('gallery/sprites/0.png').convert_alpha(),
        pygame.image.load('gallery/sprites/1.png').convert_alpha(),
        pygame.image.load('gallery/sprites/2.png').convert_alpha(),
        pygame.image.load('gallery/sprites/3.png').convert_alpha(),
        pygame.image.load('gallery/sprites/4.png').convert_alpha(),
        pygame.image.load('gallery/sprites/5.png').convert_alpha(),
        pygame.image.load('gallery/sprites/6.png').convert_alpha(),
        pygame.image.load('gallery/sprites/7.png').convert_alpha(),
        pygame.image.load('gallery/sprites/8.png').convert_alpha(),
        pygame.image.load('gallery/sprites/9.png').convert_alpha(),
    )
```

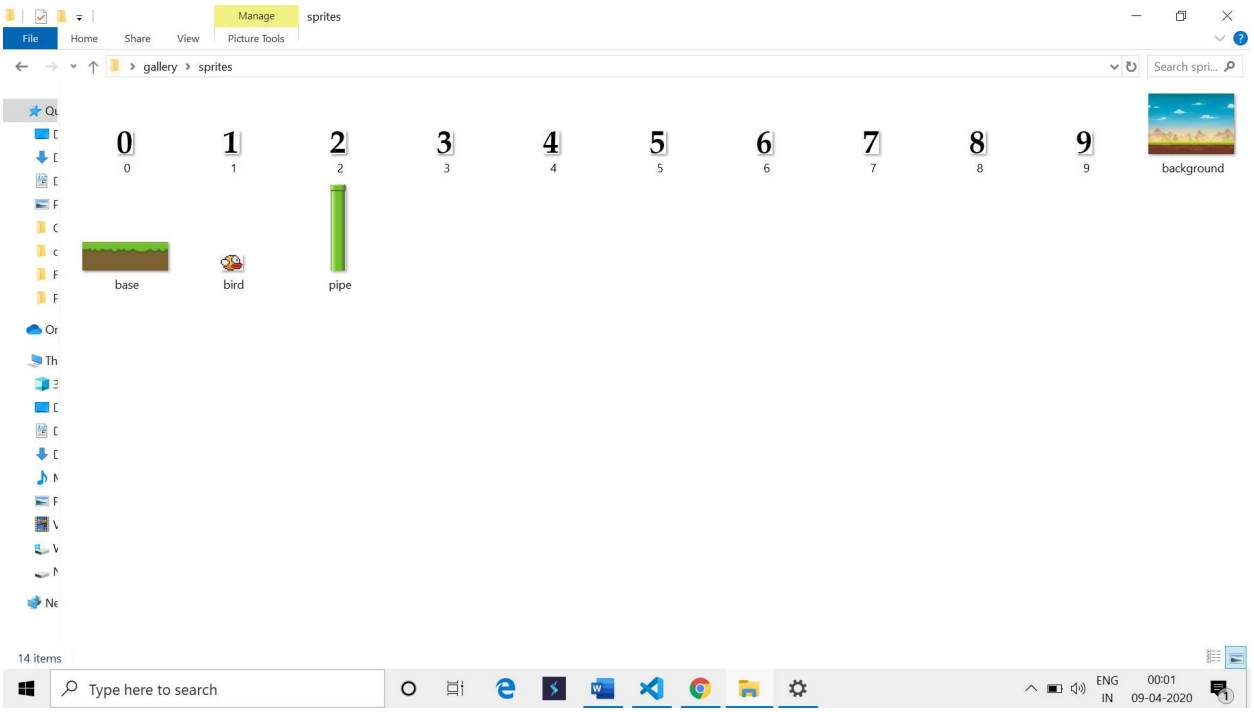
```
    GAME_SPRITES['base']
=pygame.image.load('gallery/sprites/base.png').convert_alpha()
    GAME_SPRITES['pipe'] =(pygame.transform.rotate(pygame.image.load(
PIPE).convert_alpha(), 180),
    pygame.image.load(PIPE).convert_alpha()
    )

    # Game sounds
    GAME_SOUNDS['die'] = pygame.mixer.Sound('gallery/audio/die.wav')
    GAME_SOUNDS['hit'] = pygame.mixer.Sound('gallery/audio/hit.wav')
    GAME_SOUNDS['point'] =
pygame.mixer.Sound('gallery/audio/point.wav')
    GAME_SOUNDS['swoosh'] =
pygame.mixer.Sound('gallery/audio/swoosh.wav')
    GAME_SOUNDS['wing'] =
pygame.mixer.Sound('gallery/audio/wing.wav')

    GAME_SPRITES['background'] =
pygame.image.load(BACKGROUND).convert()
    GAME_SPRITES['player'] =
pygame.image.load(PLAYER).convert_alpha()

    while True:
        welcome() # Shows welcome screen to the user until he presses a
button
        mainGame() # This is the main game function
```

4. Snapshots


Flappy Bird[MINI-PR...


Command Prompt - python main.py
Microsoft Windows [Version 10.0.18362.720]
(c) 2019 Microsoft Corporation. All rights reserved.
C:\Users\Admin>cd desktop
C:\Users\Admin\Desktop>python main.py
pygame 1.9.6
Hello from the pygame community. https://www.pygame.org/contribute.html
Your score is 1
Your score is 2
Your score is 3
Your score is 4
Your score is 5
Flappy Bird[MINI-PR...
5
Type here to search
ENG
IN
23:58
08-04-2020

The pictures that are used in the game implementation.



5. Technical Consideration

Pygame must be installed using the Python Package Installer.
Other modules used are random and sys.

Pygame is used to load images and the audio files for the game. The audio for the bird hitting pipe or the ground, the audio of the bird flapping, the audio of a point are also loaded.
'pygame.mixer.Sound' is used to load the sound files in the game.
'pygame.image.load' is used to load the image files in the game.

6. References