**Report on Mini Project**

**TSEC**

**Subject :** Python

**Title of the Project** : Flappy Bird Game

**Details of the students :**

**Batch Roll Number Name**

**C31 2203153 Yash Shelke**

**C31 2203145 Tanmay Sarode**

**C31 2203144 Sushmit Sanyal**

**Prof In-charge : Dr. Gauri Shukla**

**Signature of the Batch In charge**

**PYTHON MPR REPORT**

**Typing Speed Test**

**Introduction:**

Welcome to the captivating realm of Flappy Bird, where simplicity meets addiction in a symphony of challenge and charm. In this timeless classic, players assume the role of a humble bird navigating a perilous maze of pipes, deftly maneuvering through narrow gaps with precision and skill. With each tap of the screen, the avian protagonist flutters upward, defying gravity's relentless pull, while the player grapples with the ever-present threat of collision with the unforgiving obstacles above and below. Despite its humble origins, Flappy Bird captivates with its intuitive gameplay and endless replayability, enticing players to test their reflexes and determination against an ever-mounting score. Join the millions who have succumbed to its addictive allure, and embark on a journey where every tap counts, and every triumph is earned. Welcome to Flappy Bird, where the sky's the limit, and the challenge is boundless.

**Concepts used.**

**Pygame Integration:** Pygame, a Python library for game development, was extensively used in this project. Key Pygame functions and classes employed include:

* **pygame.init()**: Initializing the Pygame library.
* **pygame.display.set\_mode()**: Creating a display surface for rendering graphics.
* **pygame.time.Clock()**: Managing the game's frame rate.
* **pygame.event.get()**: Retrieving events from the event queue, such as key presses or mouse movements.
* **pygame.sprite.Sprite**: A base class for creating game objects.
* **pygame.draw.rect()**: Drawing rectangles on the display surface.
* **pygame.mixer.Sound()**: Loading and playing sound effects.

**MySQL Integration:** The project integrated MySQL for storing and retrieving player data. The **mysql-connector-python** library facilitated interaction with the MySQL database. Key functions and methods utilized from this library include:

* **mysql.connector.connect()**: Establishing a connection to the MySQL server.
* **connection.cursor()**: Creating a cursor object to execute SQL queries.
* **cursor.execute()**: Executing SQL queries or commands on the database.
* **cursor.fetchall()**: Retrieving all rows from the result set of a query.
* **connection.commit()**: Committing the current transaction to make changes permanent.
* **connection.close()**: Closing the connection to the database.

**Integration Workflow:** The integration workflow involved several steps:

1. Importing the **mysql.connector** module.
2. Establishing a connection to the MySQL server using **mysql.connector.connect()** with necessary credentials.
3. Creating a cursor object using **connection.cursor()**.
4. Executing SQL queries and commands using **cursor.execute()**.
5. For INSERT and UPDATE queries, committing changes using **connection.commit()**.
6. For SELECT queries, retrieving the result set using **cursor.fetchall()** or **cursor.fetchone()**.

Closing the cursor and connection when finished using **cursor.close()** and **connection.close()**.

**Code Overview:**

* **game.py**: Implements the Flappy Bird game using Pygame. It includes game logic, user interface, and database integration for storing high scores.
* **bird.py**: Defines the Bird class representing the player-controlled bird in the game.
* **pipe.py**: Defines the Pipe class representing the obstacles in the game.

**Code:**

* 1. **game.py**:

import time

from tkinter import messagebox

from bird import Bird

from pipe import Pipe

import pygame as pg

import sys

import mysql.connector

import tkinter as tk

pg.init()

conn = mysql.connector.connect(

host="localhost",

user="root",

password="Dragon@04",

database="flappybirdgame"

)

class Game:

def \_\_init\_\_(self):

self.width = 600

self.height = 768

self.scale\_factor = 1.5

self.win = pg.display.set\_mode((self.width, self.height))

self.clock = pg.time.Clock()

self.move\_speed = 250

self.bird = Bird(self.scale\_factor)

self.is\_enter\_pressed = False

self.pipes = []

self.font = pg.font.Font("assets/font.ttf", 24)

self.score\_text = self.font.render("Score: 0 ", True, (0, 0, 0))

self.score\_text\_rect = self.score\_text.get\_rect(center=(100, 30))

self.restart\_text = self.font.render("Restart", True, (0, 0, 0))

self.restart\_text\_rect = self.restart\_text.get\_rect(center=(300, 700))

self.highScore\_text = self.font.render("High Score: 0 ", True, (0, 0, 0))

self.highScore\_text\_rect = self.highScore\_text.get\_rect(center=(400, 30))

self.score = 0

self.highScore = 0

self.start\_monitoring = False

self.pipe\_generate\_counter = 71

self.is\_game\_started = True

self.setUpBgAndGround()

self.root = tk.Tk()

self.root.title("New HighScore")

question\_label = tk.Label(self.root, text="Enter your username here:")

question\_label.pack()

self.question\_entry = tk.Entry(self.root, width=50) # Adjust width as needed

self.question\_entry.pack()

submit\_button = tk.Button(self.root, text="Submit", command=self.submit\_username)

submit\_button.pack()

self.gameLoop()

def submit\_username(self):

username = self.question\_entry.get()

cursor = conn.cursor()

sql = "INSERT INTO userinfo (user\_name, user\_highscrore) VALUES (%s, %s)"

cursor.execute(sql, (username, self.highScore))

conn.commit()

cursor.close()

self.root.destroy()

def gameLoop(self):

last\_time = time.time()

while True:

new\_time = time.time()

dt = new\_time - last\_time

last\_time = new\_time

for event in pg.event.get():

if event.type == pg.QUIT:

pg.quit()

sys.exit()

if event.type == pg.KEYDOWN and self.is\_game\_started:

if event.key == pg.K\_RETURN:

self.is\_enter\_pressed = True

self.bird.update\_on = True

if event.key == pg.K\_SPACE and self.is\_enter\_pressed:

self.bird.flap(dt)

if event.type == pg.MOUSEBUTTONUP:

if self.restart\_text\_rect.collidepoint(pg.mouse.get\_pos()):

self.restartGame()

self.updateEverything(dt)

self.checkCollisions()

self.checkScore()

self.drawEverything()

pg.display.update()

self.clock.tick(60)

def restartGame(self):

if self.score > self.highScore:

self.highScore = self.score

self.highScore\_text = self.font.render(f"High Score: {self.highScore} ", True, (0, 0, 0))

self.root.mainloop()

messagebox.showinfo("CONGRATS", f"Your new high score is {self.highScore}")

self.score = 0

self.score\_text = self.font.render("Score: 0 ", True, (0, 0, 0))

self.is\_enter\_pressed = False

self.is\_game\_started = True

self.bird.resetposition()

self.pipes.clear()

self.pipe\_generate\_counter = 71

self.bird.update\_on = False

def checkScore(self):

if len(self.pipes) > 0:

if self.bird.rect.left > self.pipes[0].rect\_down.left and not self.start\_monitoring and self.bird.rect.right < self.pipes[0].rect\_down.right and not self.start\_monitoring:

self.start\_monitoring = True

if self.bird.rect.left > self.pipes[0].rect\_down.right and self.start\_monitoring:

self.start\_monitoring = False

self.score += 1

self.score\_text = self.font.render(f"Score: {self.score} ", True, (0, 0, 0))

def checkCollisions(self):

if len(self.pipes):

if self.bird.rect.bottom > 568:

self.bird.update\_on = False

self.is\_enter\_pressed = False

self.is\_game\_started = False

if self.bird.rect.colliderect(self.pipes[0].rect\_down) or self.bird.rect.colliderect(

self.pipes[0].rect\_up):

self.is\_enter\_pressed = False

self.is\_game\_started = False

def updateEverything(self, dt):

if self.is\_enter\_pressed:

self.ground1\_rect.x -= self.move\_speed \* dt

self.ground2\_rect.x -= self.move\_speed \* dt

if self.ground1\_rect.right < 0:

self.ground1\_rect.x = self.ground2\_rect.right

if self.ground2\_rect.right < 0:

self.ground2\_rect.x = self.ground1\_rect.right

if self.pipe\_generate\_counter > 70:

self.pipes.append(Pipe(self.scale\_factor, self.move\_speed))

self.pipe\_generate\_counter = 0

self.pipe\_generate\_counter += 1

for pipe in self.pipes:

pipe.update(dt)

if len(self.pipes) != 0:

if self.pipes[0].rect\_up.right < 9:

self.pipes.pop(0)

self.bird.update(dt)

def drawEverything(self):

self.win.blit(self.bg\_img, (0, -300))

for pipe in self.pipes:

pipe.drawPipe(self.win)

self.win.blit(self.ground1\_img, self.ground1\_rect)

self.win.blit(self.ground2\_img, self.ground2\_rect)

self.win.blit(self.bird.image, self.bird.rect)

self.win.blit(self.score\_text, self.score\_text\_rect)

self.win.blit(self.highScore\_text, self.highScore\_text\_rect)

if not self.is\_game\_started:

self.win.blit(self.restart\_text, self.restart\_text\_rect)

def setUpBgAndGround(self):

self.bg\_img = pg.transform.scale\_by(pg.image.load("assets/bg.png").convert(), self.scale\_factor)

self.ground1\_img = pg.transform.scale\_by(pg.image.load("assets/ground.png").convert(), self.scale\_factor)

self.ground2\_img = pg.transform.scale\_by(pg.image.load("assets/ground.png").convert(), self.scale\_factor)

self.ground1\_rect = self.ground1\_img.get\_rect()

self.ground2\_rect = self.ground2\_img.get\_rect()

self.ground1\_rect.x = 0

self.ground2\_rect.x = self.ground1\_rect.right

self.ground1\_rect.y = 568

self.ground2\_rect.y = 568

game = Game()

* 1. **bird.py:**

import pygame as pg

class Bird(pg.sprite.Sprite):

def \_\_init\_\_(self,scale\_factor):

super(Bird,self).\_\_init\_\_()

self.img\_list=[pg.transform.scale\_by(pg.image.load("assets/birdup.png").convert\_alpha(),scale\_factor),pg.transform.scale\_by(pg.image.load("assets/birddown.png").convert\_alpha(),scale\_factor)]

self.image\_index=0

self.image=self.img\_list[self.image\_index]

self.rect=self.image.get\_rect(center=(100,100))

self.y\_velocity=0

self.gravity=10

self.anim\_counter=0

self.flap\_speed=250

self.update\_on=False

def update(self,dt):

if self.update\_on:

self.playAnimation()

self.applyGravity(dt)

if self.rect.y<=0:

self.rect.y=0

self.flap\_speed=0

elif self.rect.y>=0 and self.flap\_speed==0:

self.flap\_speed=250

def applyGravity(self,dt):

self.y\_velocity += self.gravity \* dt

self.rect.y += self.y\_velocity

def flap(self,dt):

self.y\_velocity=-self.flap\_speed\*dt

def playAnimation(self):

if self.anim\_counter==5:

self.image=self.img\_list[self.image\_index]

if self.image\_index==0: self.image\_index=1

else: self.image\_index=0

self.anim\_counter=0

self.anim\_counter+=1

def resetposition(self):

self.rect.center=(100, 100)

self.y\_velocity=0

self.anim\_counter=0

* 1. **pipe.py**:

import pygame as pg

from random import randint

class Pipe:

def \_\_init\_\_(self,scale\_factor,move\_speed):

self.img\_up=pg.transform.scale\_by(pg.image.load("assets/pipeup.png").convert\_alpha(),scale\_factor)

self.img\_down = pg.transform.scale\_by(pg.image.load("assets/pipedown.png").convert\_alpha(), scale\_factor)

self.rect\_up=self.img\_up.get\_rect()

self.rect\_down=self.img\_down.get\_rect()

self.pipe\_distance=200

self.rect\_up.y=randint(250,520)

self.rect\_up.x=600

self.rect\_down.y=self.rect\_up.y-self.pipe\_distance-self.rect\_up.height

self.rect\_down.x = 600

self.move\_speed=move\_speed

def drawPipe(self,win):

win.blit(self.img\_up,self.rect\_up)

win.blit(self.img\_down,self.rect\_down)

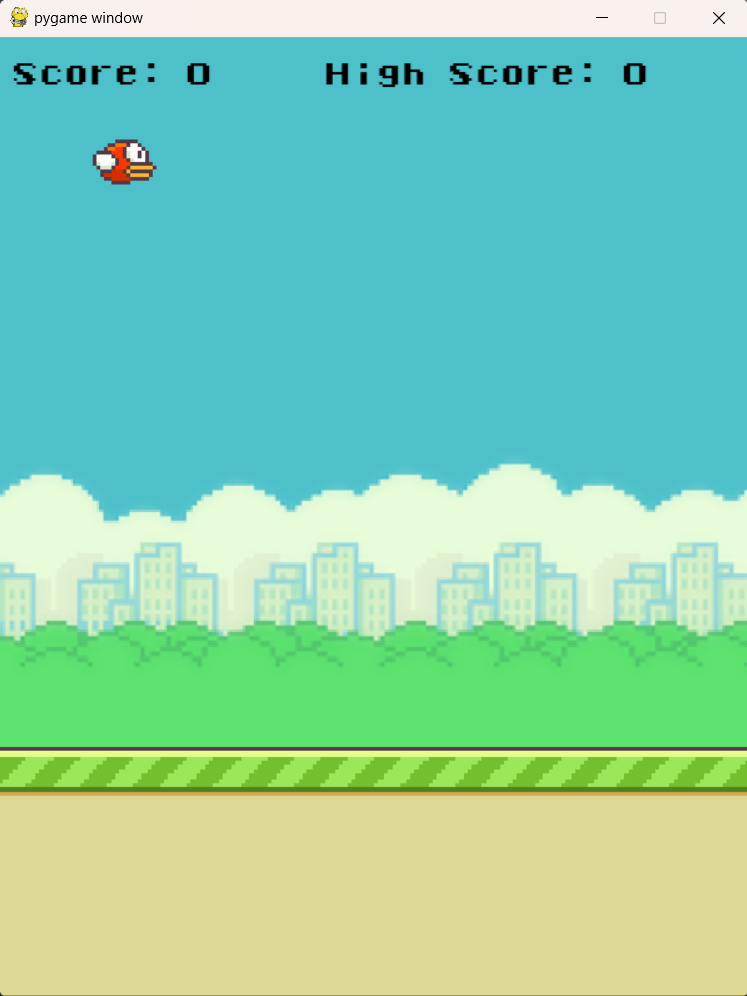
def update(self, dt):

self.rect\_up.x-=int(self.move\_speed\*dt)

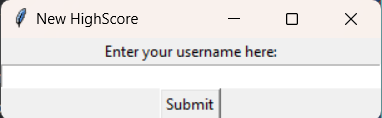
self.rect\_down.x-=int(self.move\_speed\*dt)

**Output:**

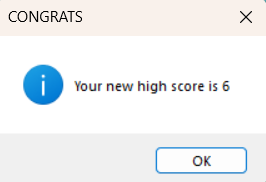
* 1. Main Game Frame

****

* 1. Username interface



* 1. New high-score interface



* 1. MySQL workbench database table

