Project Title: Flappy Bird Game using Pygame

Subject: Decision Modelling

Class: Msc.AI – Part I

Team Members and their student Id's:

Riya Bhattacharya (4870208)

Rishu Shrivastav (4919413)

Ketaki Sonawane (4804451)

Yogini Pawar (4735843)

1. Introduction

Aim

The aim of this project is to create a functional clone of the Flappy Bird game using Python and the Pygame library.

Problem Statement

Flappy Bird is a simple yet addictive game that requires precise timing and control. The challenge is to implement the game's core mechanics and ensure smooth gameplay.

Introduction

Flappy Bird, originally developed by Dong Nguyen, became a viral sensation due to its challenging gameplay.
It is a side-scrolling game with 2D graphics. The game has a classic style with simple shapes. The player's task is to control a bird that tends to fly downward. You need to continuously control this character to move through the gaps between sewer pipes appearing from both sides of the terrain. Even a small collision causes the round to end. Each time you press the key, the bird will seem to bounce a beat. Each pair of pipes the bird can fly through corresponds to a point in your round. Your goal is to try to get the highest score possible on this unlimited flight.This project recreates Flappy Bird using Pygame, focusing on the bird's movement, pipe generation, collision detection, and scoring system.

2. Scope

The scope of this project includes developing the game mechanics, implementing collision detection, and creating a scoring system. It does not include advanced features such as power-ups or multiplayer functionality.

3. Objectives

1. Implement bird movement and controls.
2. Create and manage pipe obstacles.
3. Implement collision detection.
4. Develop a scoring system.
5. Ensure smooth and responsive gameplay.

4. Motivation

The motivation for this project is to learn game development using Pygame and to recreate a popular game that provides a challenging yet rewarding experience. Additionally, the project aims to relive childhood memories, feeling the nostalgia and immense pride that comes from making a successful game. It is a great feeling to create something from scratch and watch it come to life, providing both a sense of accomplishment and a fun gaming experience.

5. Literature Review

References

- Pygame Documentation
  https://www.pygame.org/docs/

- Tutorials on Pygame Game Development
  https://www.youtube.com/watch?v=XON1IA8MxQw&list=PLz7mmheDeooW795HsNzkLS6TFQSa4Idtw

- Article on Flappy Bird
  https://www.geeksforgeeks.org/flappy-bird-game-in-javascript/amp/

6. Methodology

Pygame Library
Pygame is a set of Python modules designed for writing video games. It includes computer graphics and sound libraries.

Game Components
Bird: Controlled by the player, with physics-based movement.
Pipes: Generated at regular intervals, moving horizontally.
Collision Detection: Ensures the bird's interactions with pipes and ground are detected.
Scoring System: Increases the score as the bird successfully navigates between pipes.

7. Implementation

Code

The game is implemented in Python using the Pygame library. Below is a simplified version of the main game loop:

game.py

```
import pygame as pg # pygame is used for creating games
import sys, time # sys is used to handle system-level operations like exiting
the game.
from bird import Bird # time is used for managing the game loop timing.
from pipe import Pipe # Bird and Pipe are custom modules representing the bird
and pipes.

# Initializing Pygame and Pygame Mixer:
pg.init() #pg.init() initializes all Pygame modules.
pg.mixer.init() # pg.mixer.init() initializes the Pygame mixer for handling
sound.

 # Setting Up Basic Game Parameters:
class Game:
    def __init__(self):
        self.scale_factor = 1.5 # scale_factor adjusts the size of game
elements.
        self.width = 600   # width and height define the initial window size.
        self.height = 768
        self.win = pg.display.set_mode((self.width, self.height),
pg.RESIZABLE)
        # self.win creates a resizable Pygame window.

        self.clock = pg.time.Clock() # self.clock manages the frame rate.
        self.move_speed = 250 # move_speed controls how fast elements move on
the screen.
        self.start_monitoring = False # start_monitoring tracks whether the
bird is in scoring range.
        self.score = 0 # score keeps the player's score.
        '''font, score_text, score_text_rect handle the score display.'''
        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))

        '''restart_text, restart_text_rect handle the restart button
display.'''
        self.restart_text = self.font.render("Restart", True, (0, 0, 0))
        self.restart_text_rect = self.restart_text.get_rect(center=(300, 700))
```

```
        #Initializes the bird object.
        self.bird = Bird(self.scale_factor)

        self.is_enter_pressed = False
        self.is_game_started = True
        self.is_game_paused = False
        self.hit_sound_played = False
        self.pipes = [] # pipes list holds pipe objects.
        self.pipe_generate_counter = 71 # pipe_generate_counter helps with
timing pipe generation.

        '''setUpBgAndGround loads and positions background and ground images.
        loadSounds loads game sounds.
        gameLoop starts the main game loop.'''
        self.setUpBgAndGround()
        self.loadSounds()

        self.gameLoop()

    #Continuously runs until the game exits.
    def gameLoop(self):
        last_time = time.time()
        while True:
            new_time = time.time()
            dt = new_time - last_time
            last_time = new_time

        # Handles user inputs and window events.
        # Toggles pause with the K key.
        #Starts the game with the Enter key.
        #Makes the bird flap with the Space key, playing the flap sound.
            for event in pg.event.get():
                if event.type == pg.QUIT:
                    pg.quit()
                    sys.exit()
                if event.type == pg.KEYDOWN:
                    if event.key == pg.K_k:
                        self.togglePause()
                    if event.key == pg.K_RETURN and self.is_game_started and
not self.is_game_paused:
                        self.is_enter_pressed = True
                        self.bird.update_on = True
                    if event.key == pg.K_SPACE and self.is_enter_pressed and
not self.is_game_paused:
                        self.bird.flap(dt)
                        self.flap_sound.play()
                # Restarts the game if the restart button is clicked.
                if event.type == pg.MOUSEBUTTONDOWN:
```

```
                    if
self.restart_text_rect.collidepoint(pg.mouse.get_pos()):
                        self.restartGame()
                # Adjusts the game elements when the window is resized.
                if event.type == pg.VIDEORESIZE:
                    self.resizeWindow(event.w, event.h)

            # Updates game elements, checks score, and detects collisions if
the game is not paused.
            if not self.is_game_paused:
                self.updateEverything(dt)
                self.CheckScore()
                self.checkCollisions()

            # Draws all game elements and updates the display.
            #Maintains a frame rate of 60 FPS(Frame Per Second).
            self.drawEverything()
            pg.display.update()
            self.clock.tick(60)

   # Toggles the game's paused state.
    def togglePause(self):
        self.is_game_paused = not self.is_game_paused

    #Resizes the game window and adjusts game elements accordingly.
    def resizeWindow(self, width, height):
        self.width, self.height = width, height
        self.win = pg.display.set_mode((self.width, self.height),
pg.RESIZABLE)
        self.scale_factor = min(self.width / 600, self.height / 768)
        self.setUpBgAndGround()
        self.bird.scale(self.scale_factor)
        self.score_text_rect.center = (100 * self.scale_factor, 30 *
self.scale_factor)
        self.restart_text_rect.center = (self.width / 2, self.height - 100 *
self.scale_factor)

    # Resets the game to its initial state.
    def restartGame(self):
        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.hit_sound_played = False
        self.pipes.clear()
        self.pipe_generate_counter = 71
        self.bird.update_on = False
        self.bird.resetPosition()
```

```
    # Increases the score when the bird successfully passes through pipes and
plays the score sound.
    def CheckScore(self):
        if len(self.pipes) > 0:
            if self.bird.rect.left > self.pipes[0].rect_down.left 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))
                self.score_sound.play()

    # Checks if the bird collides with the ground or pipes and plays the hit
sound only once per collision.
    def checkCollisions(self):
        if len(self.pipes):
            if self.bird.rect.bottom > self.height - 200 * self.scale_factor:
                self.bird.update_on = False
                self.is_enter_pressed = False
                self.is_game_started = False
                if not self.hit_sound_played:
                    self.hit_sound.play()
                    self.hit_sound_played = True
            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
                if not self.hit_sound_played:
                    self.hit_sound.play()
                    self.hit_sound_played = True

    #Updates the ground position, generates pipes, updates pipes, and updates
the bird.
    def updateEverything(self, dt):
        if self.is_enter_pressed:
            self.ground1_rect.x -= int(self.move_speed * dt)
            self.ground2_rect.x -= int(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 < 0:
                    self.pipes.pop(0)

        self.bird.update(dt)

    # Draws the background, pipes, ground, bird, score, and restart text.
    def drawEverything(self):
        self.win.blit(self.bg_img, (0, 0))
        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)

        if not self.is_game_started:
            self.win.blit(self.restart_text, self.restart_text_rect)

    #Loads and scales the background and ground images, sets their initial
positions.
    def setUpBgAndGround(self):
        self.bg_img =
pg.transform.scale(pg.image.load("assets/bg.png").convert(), (self.width,
self.height))
        self.ground1_img =
pg.transform.scale(pg.image.load("assets/ground.png").convert(), (self.width,
int(200 * self.scale_factor)))
        self.ground2_img =
pg.transform.scale(pg.image.load("assets/ground.png").convert(), (self.width,
int(200 * 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 = self.height - 200 * self.scale_factor
        self.ground2_rect.y = self.height - 200 * self.scale_factor
```

```
    # Loads the sound files for flap, score, and hit actions.
    def loadSounds(self):
        self.flap_sound = pg.mixer.Sound("assets/sfx/flap.wav")
        self.score_sound = pg.mixer.Sound("assets/sfx/score.wav")
        self.hit_sound = pg.mixer.Sound("assets/sfx/dead.wav")

#Instantiates the Game class, starting the game.
game = Game()
```

bird.py

```
import pygame as pg

class Bird(pg.sprite.Sprite):
    def __init__(self, scale_factor):
        super(Bird, self).__init__()
        self.original_img_list = [
            pg.image.load("assets/birdup.png").convert_alpha(),
            pg.image.load("assets/birddown.png").convert_alpha()
        ]
        self.img_list = [pg.transform.scale_by(img, scale_factor) for img in
self.original_img_list]
        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.flap_speed = 250
        self.anim_counter = 0
        self.update_on = False

    def update(self, dt):
        if self.update_on:
            self.playAnimation()
            self.applyGravity(dt)

            if self.rect.y <= 0 and self.flap_speed == 250:
                self.rect.y = 0
                self.flap_speed = 0
                self.y_velocity = 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]
            self.image_index = 1 - self.image_index  # Toggle between 0 and 1
            self.anim_counter = 0
        self.anim_counter += 1

    def resetPosition(self):
        self.rect.center = (100, 100)
        self.y_velocity = 0
        self.anim_counter = 0

    def scale(self, scale_factor):
        self.img_list = [pg.transform.scale_by(img, scale_factor) for img in
self.original_img_list]
        self.image = self.img_list[self.image_index]
        self.rect = self.image.get_rect(center=self.rect.center)
```

pipe.py

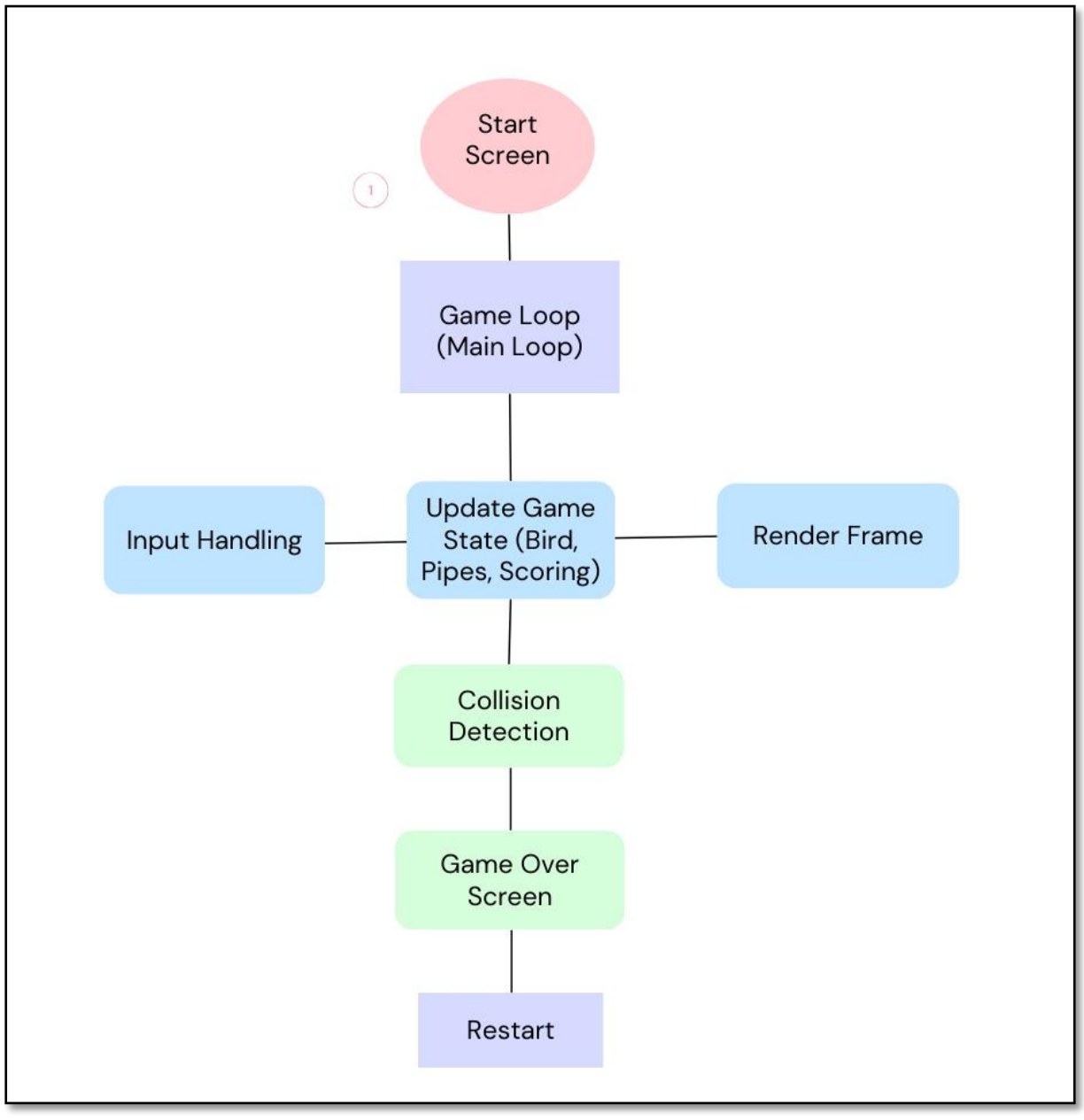
```
import pygame as pg
from random import randint
class Pipe:
    def __init__(self,scale_factor,move_speed):
        self.img_up=pg.transform.scale_by(pg.image.load("assets/pipeup.png").c
onvert_alpha(),scale_factor)
        self.img_down=pg.transform.scale_by(pg.image.load("assets/pipedown.png
").convert_alpha(),scale_factor)
        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)
```

Game Architecture

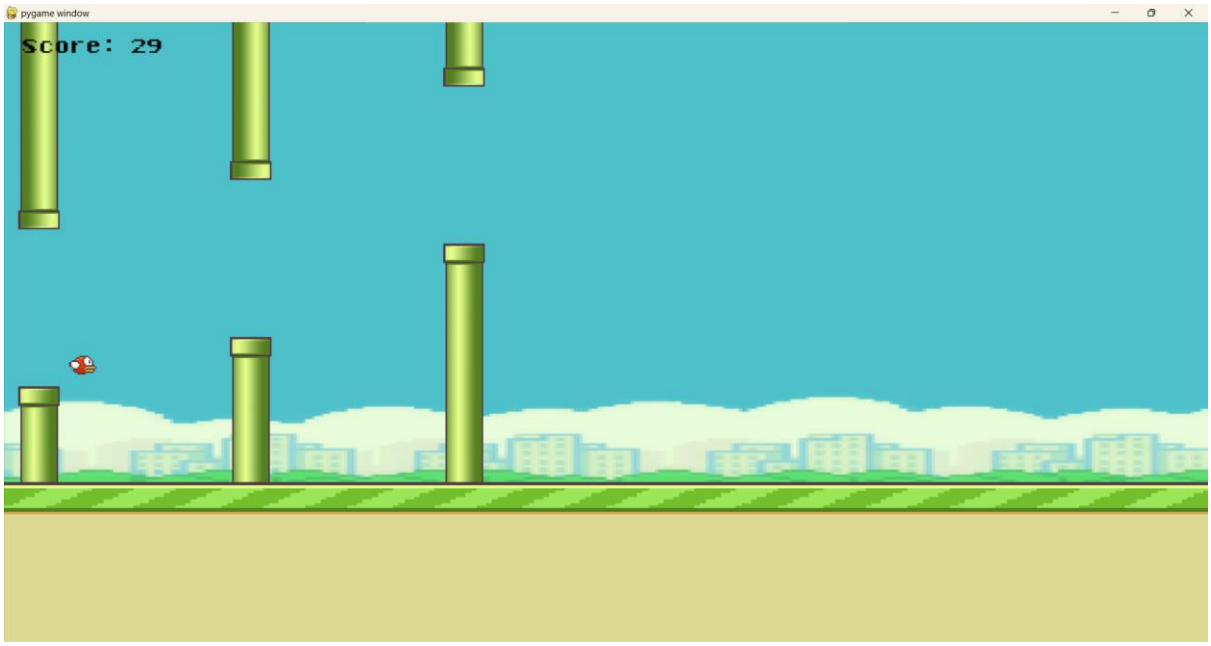
The game architecture includes the main game loop, event handling, game state management, and rendering. Key components like the bird, pipes, collision detection, and score are managed within this loop.

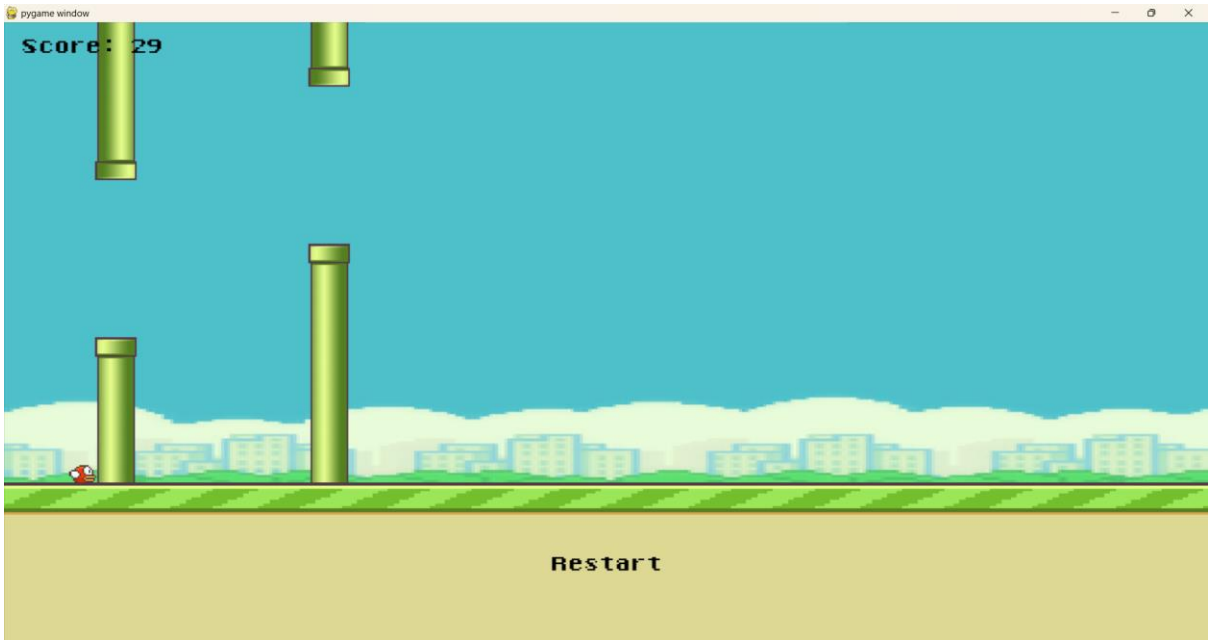


8. Results

Output Screenshot

The game successfully recreates the core mechanics of Flappy Bird. The bird moves smoothly, pipes are generated correctly, and the collision detection works as intended.





9. Discussion

The project successfully met its objectives. Challenges included fine-tuning the bird's physics and ensuring smooth pipe generation. The use of Pygame provided a solid foundation for game development, and the final product closely mimics the original game. Collaborative efforts contributed to a well-rounded and robust implementation.

10. Conclusion

The Flappy Bird game developed using Pygame demonstrates the capability of Python for game development. The project achieved its goals, providing a functional and engaging game experience. Collaboration enhanced the project, leading to a better outcome.

11. Future Work

Future enhancements could include adding sound effects, improving graphics, implementing power-ups, and creating a menu system. Additionally, exploring multiplayer functionality or AI-based enhancements could provide further learning opportunities. Integrating levels in the game, maybe easy, medium and hard mode or maybe creating different avatar and different teams and different reward system.

12. References

Pygame Documentation. (n.d.). Retrieved from https://www.pygame.org/docs/

https://www.geeksforgeeks.org/how-to-make-flappy-bird-game-in-pygame/

https://www.youtube.com/watch?v=XON1IA8MxQw&list=PLz7mmheDeooW795HsNzkLS6TFQSa4Idtw

13. GitHub Link

Github ID of all the team members

https://github.com/yoginipawar

https://github.com/riyabhatt0123

https://github.com/Rishushrivastav90/

https://github.com/ketaki-27

14. Decision Modeling in Game Design

Decision modeling in this game includes:

Game Design Decisions: Choosing the game's difficulty level by adjusting the speed of pipes and gravity.

Player Interaction: Deciding how the bird responds to user input.

Collision Detection: Implementing robust collision detection to ensure accurate game mechanics.

Scoring System: Designing a fair and rewarding scoring system to enhance player engagement.

Collaboration Decisions: Distributing tasks among team members, coordinating development efforts, and integrating contributions effectively.