GUJARAT UNIVERSITY
K.S.SCHOOL OF BUSINESS MANAGEMENT
[FIVE YEARS' FULL-TIME M.Sc.(CA&IT) INTEGRATED DEGREE COURSE]

Date: 21/12/2024

All the students of SEMESTER –7 of **M.Sc.(CA & IT)** are requested to submit their project Report as per below mentioned format.

Contents of Sem-7 Project Report for Fourth Year M.Sc.(CA & IT)

1st Page:

Project Title: Flappy Bird

(By SEMESTER – 7 of IV Year M.Sc. (2024-25))

Submitted By:

Student Name: Roll no.

1). Gajjar Helly Pinankkumar, 4021

2). Parmar Dhara Jitubhai, 4050

Group id: 43

Date of submission: 23/12/2024

Submitted To

K. S. School of Business Management & Information Technology

M.Sc. - Computer Applications and Information Technology.

INDEX

• Introduction

- This project aims to recreate the classic Flappy Bird game and develop an AI agent that can learn to play the game autonomously.
- The AI will use reinforcement learning techniques to master the game's mechanics, such as timing and obstacle avoidance, with the goal of achieving a high score.

• Objectives

1. **Game Development:** Implement the Flappy Bird game, including basic mechanics like gravity, collision detection, and scoring.

2. **AI Integration:** Develop an AI agent that can learn to play Flappy Bird using reinforcement learning techniques such as Deep Q-Learning.

3. **Training and Optimization:** Train the AI agent to maximize its score, experimenting with different hyperparameters and network architectures to improve performance.

4. **Performance Evaluation:** Analyze the AI's performance over time, comparing its behaviour to human players and assessing its learning process.

5. **Visualization:** Provide real-time visualization of the AI's decision-making process and its progress during training. Deliverables:

- Problem Statement

Design and implement an agent that can play the Flappy Bird game autonomously. The agent should be able to:

1. **Control the bird:** Make the bird jump or flap its wings to avoid obstacles.

2. **Avoid obstacles:** Navigate through the game environment without hitting pipes, ground, or other obstacles.

3. **Maximize score:** Accumulate points by successfully passing through pipes.

- **Basic Concept**
- **Key AI Terminologies**

1. Agent: The AI system playing the game.

2. Policy: Mapping from game states to actions.

3. Value Function: Estimates expected future reward for a game state.

4. Q-Function: Estimates expected future reward for a game state and action.

5. Deep Q-Network (DQN): Neural network approximating the Q-function.

6. Reinforcement Learning (RL): Training an agent using trial and error, with rewards or penalties as feedback.

7. **Game State:** Current state of the game, including bird position and obstacle locations.

8. Action Space: Set of possible actions, such as jumping or doing nothing.

9. Reward Function: Assigns rewards or penalties based on the agent's actions and game state.

- # Introduction to AI Technologies Used in this project

  - **Deep Q-Networks (DQN)**

➢ A type of Reinforcement Learning (RL) algorithm that uses a neural network to predict the best action (jump or do nothing) based on the game state.

- # Requirement Analysis
- # Tool and Technologies Needed

- **Programming Language: Python**
- **Libraries**: Pygame (for game development), TensorFlow or PyTorch (for AI model development), Matplotlib (for data visualization)
- **Development Environment:** Jupyter Notebook, Visual Studio Code, or any suitable IDE

- **Version Control:** GitHub or GitLab for version control and collaboration

• Basic System Requirement

- Operating System: Android 2.3 or iOS 4.3 or later
- Processor: 1 GHz or faster
- RAM: 512 MB or more
- Storage: 10 MB or more
- Graphics: Basic 2D graphics support
- Screen Resolution: 320x480 pixels or higher

• Dataset
• Source of Data

1. User input (touchscreen tap)
2. Game state (bird position, velocity, pipe locations)

- **Overview of the Data**

1. User Input: Touchscreen taps to control the bird.

2. Game State: Bird position, velocity, and pipe locations.

3. Sensor Data: Accelerometer data for device orientation.

4. Game Physics: Simulated physics engine data for collisions and gravity.

5. Score and Progress: Player's current score, high score and game progress.

- **Proposed Solution**
- **Explanation of the Approach**

The approach used in the Flappy Bird game is based on a simple physics engine and a random pipe generation system, combined with a touch-based control system that allows players to control the bird's movements.

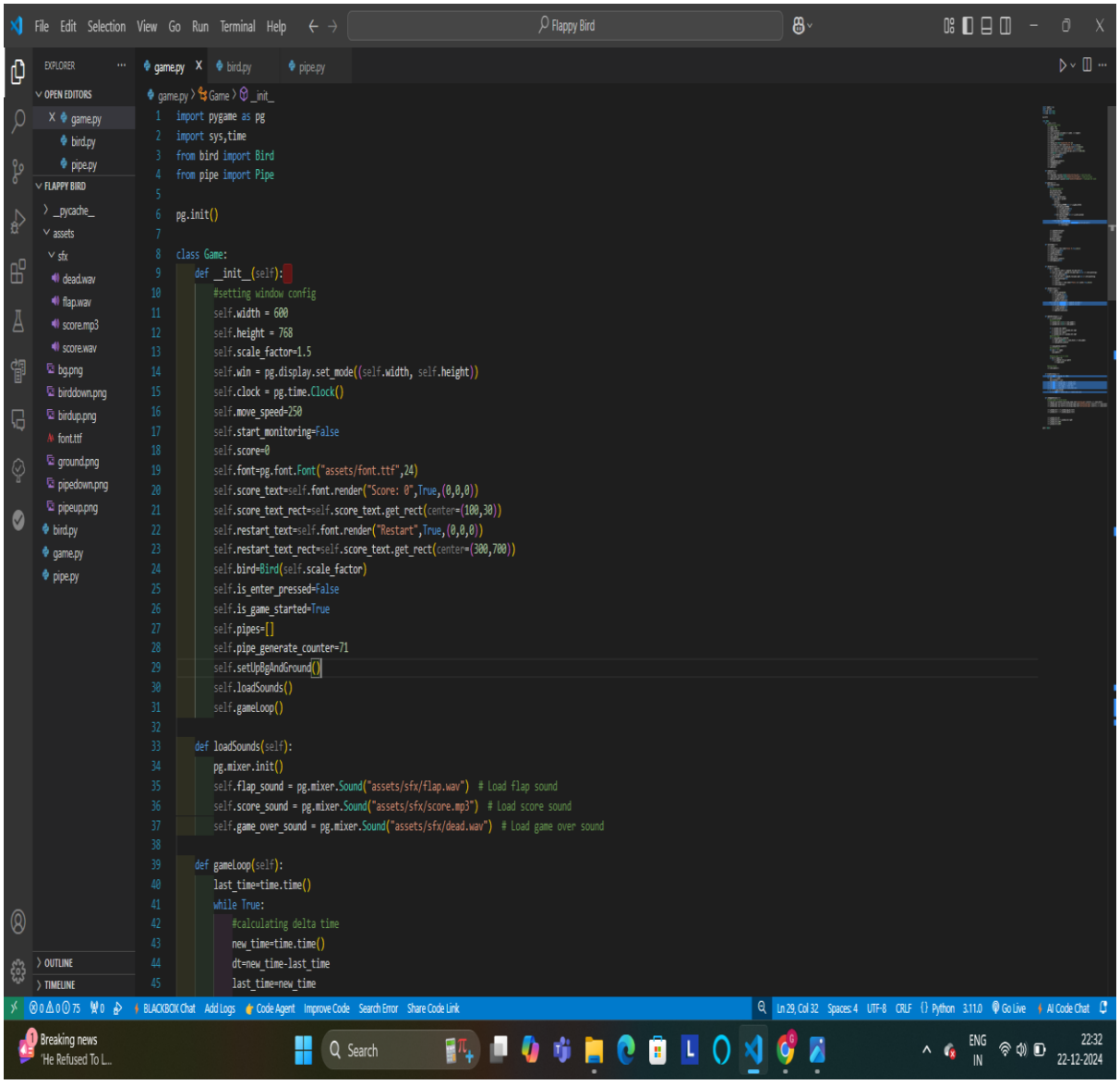
- **Algorithm**

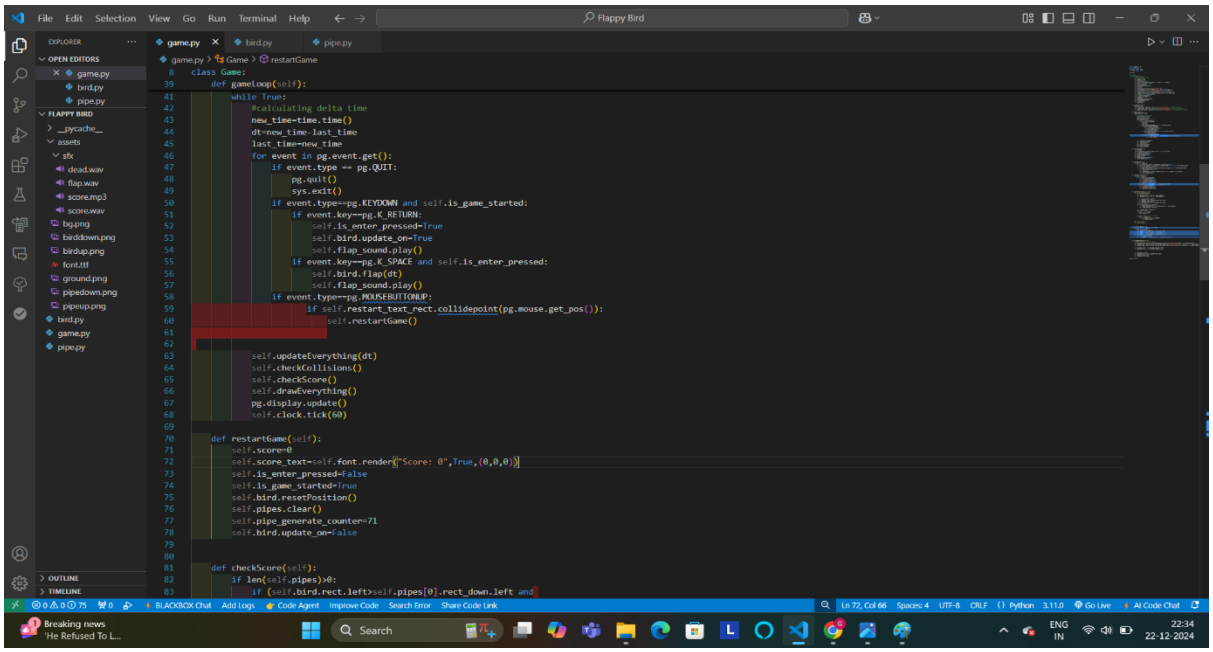
The AI-powered Flappy Bird game uses the Deep Q-Networks (DQN) algorithm.

- **Implementation**

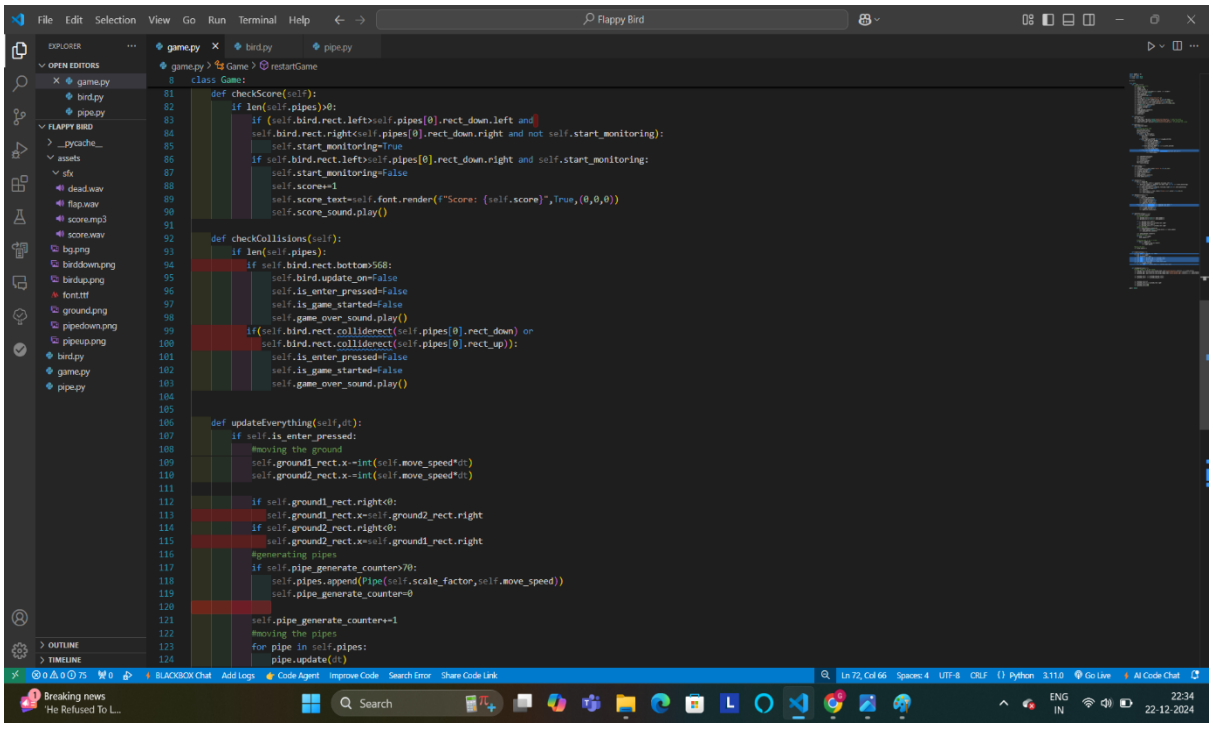
1. Set up the game environment using a framework like Unity.
2. Implement game physics using a physics engine.
3. Choose a machine learning algorithm like Deep Q-Networks (DQN).
4. Train the AI model using collected data on bird actions and outcomes.
5. Integrate the trained AI model with the game environment.
6. Test and refine the AI-powered game.

- **Code Snippets**

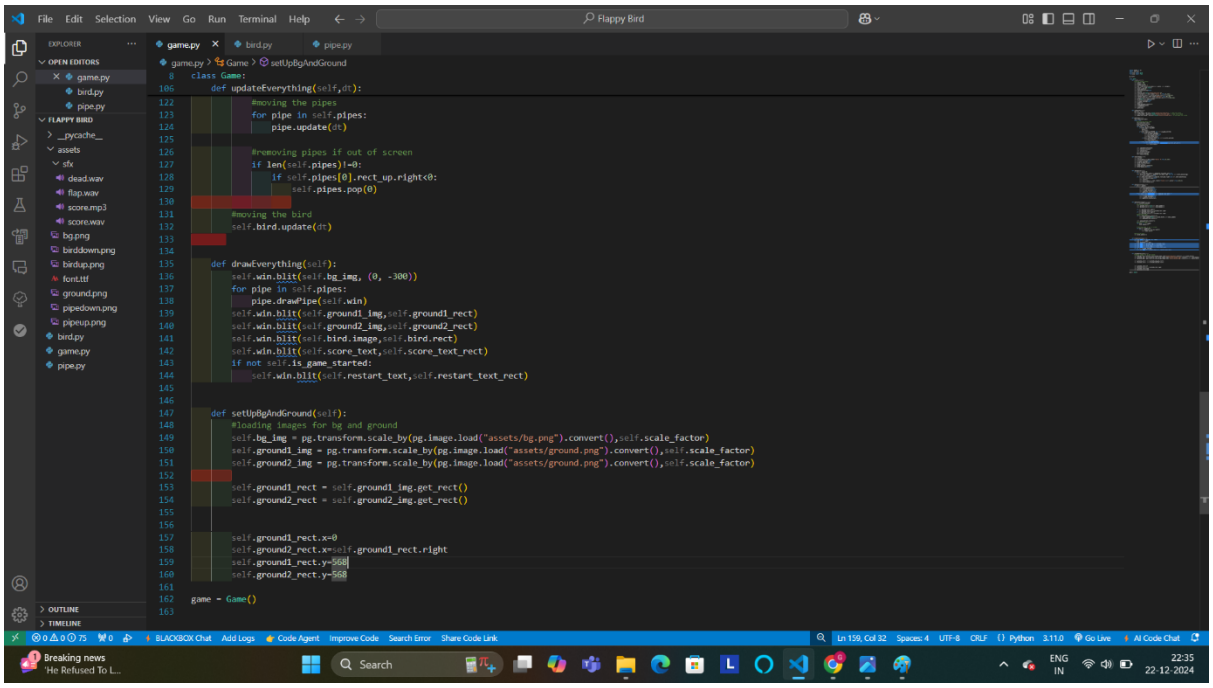




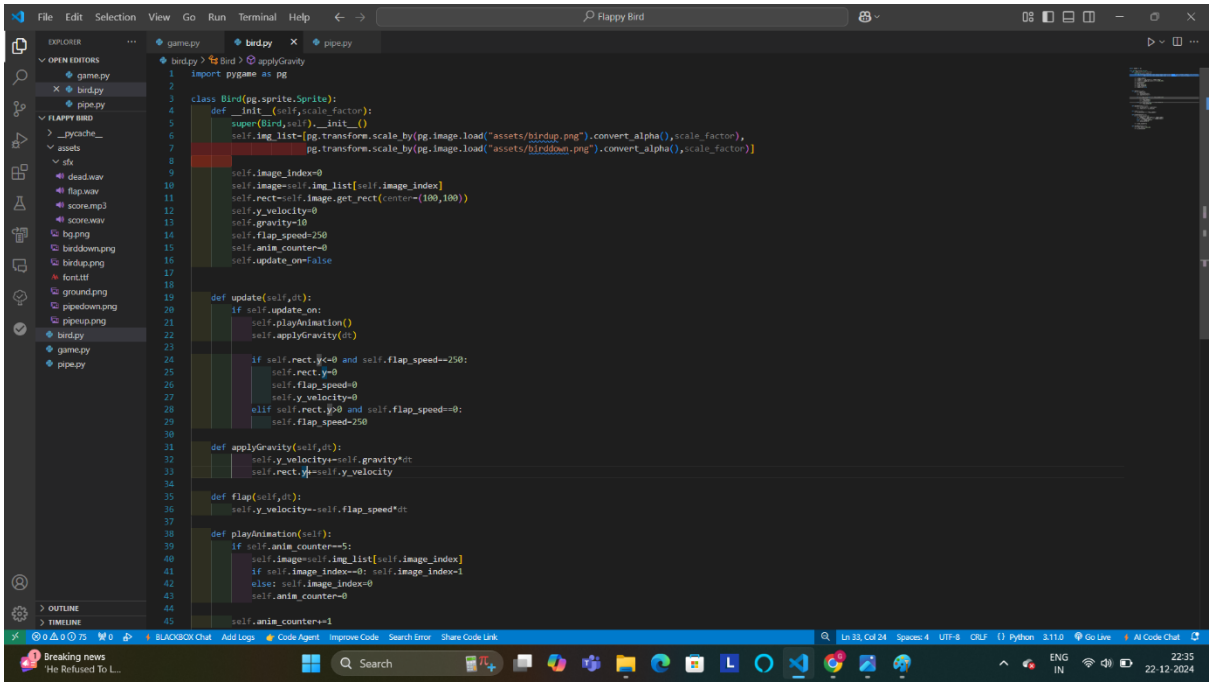
```
class Game:
    def gameLoop(self):
        while True:
            #calculating delta time
            new_time=time.time()
            dt=new_time-last_time
            last_time=new_time
            for event in pg.event.get():
                if event.type == pg.QUIT:
                    pg.quit()
                    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
                        self.flap_sound.play()
                    if event.key==pg.K_SPACE and self.is_enter_pressed:
                        self.bird.flap(dt)
                        self.flap_sound.play()
                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):
        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
```



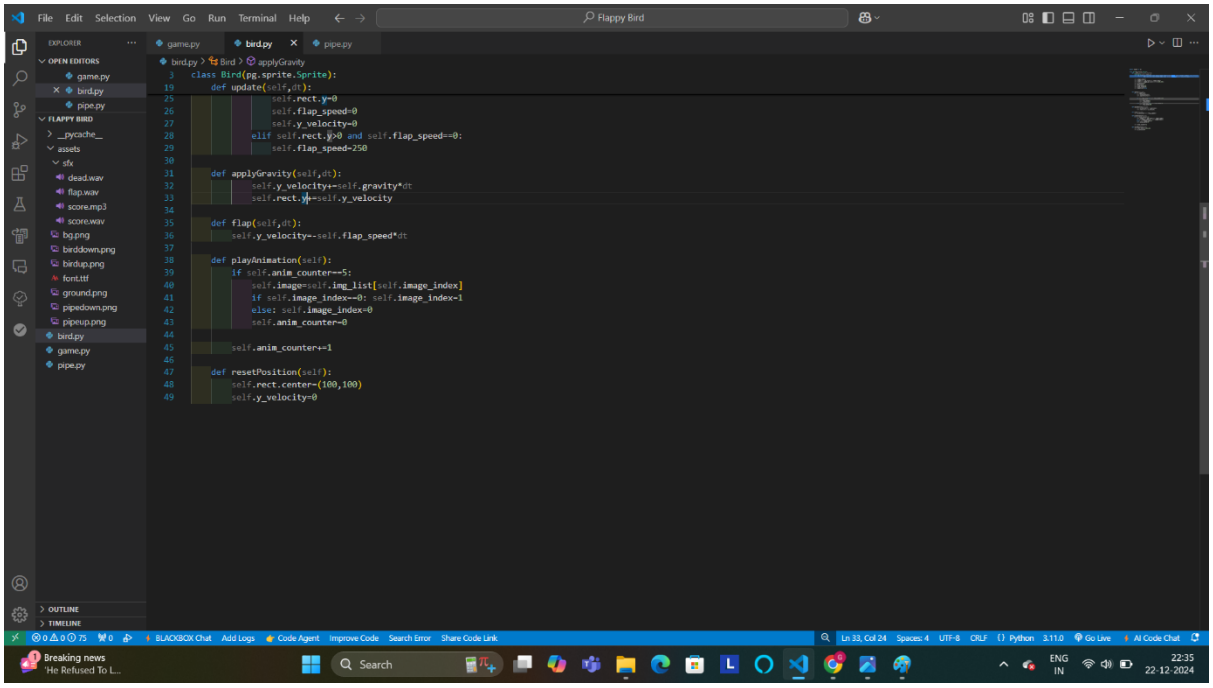
```
class Game:
    def checkScore(self):
        if len(self.pipes)>0:
            if (self.bird.rect.left>self.pipes[0].rect_down.left and
            self.bird.rect.right<self.pipes[0].rect_down.right and not self.start_monitoring):
                self.start_monitoring=True
            if self.bird.rect.left>self.pipes[0].rect_down.right 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()

    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
                self.game_over_sound.play()
            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
                self.game_over_sound.play()


    def updateEverything(self,dt):
        if self.is_enter_pressed:
            #moving the ground
            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
            #generating pipes
            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
            #moving the pipes
            for pipe in self.pipes:
                pipe.update(dt)
```



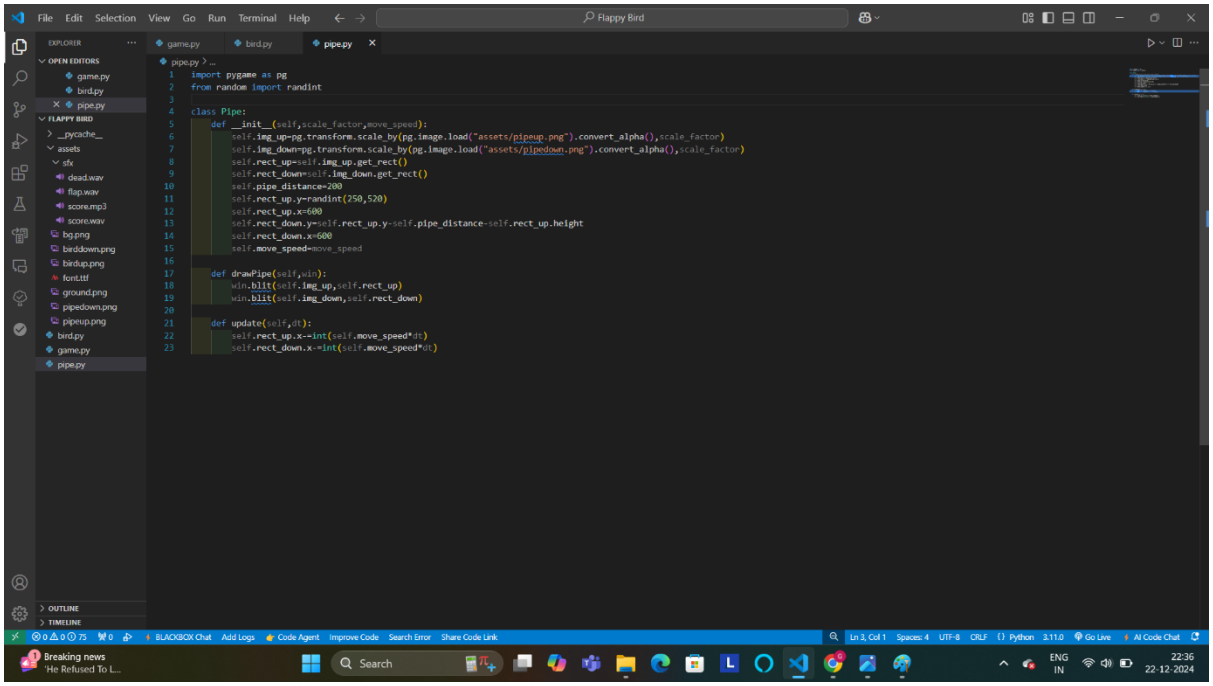
```python
class Game:
    def updateEverything(self,dt):
            #moving the pipes
            for pipe in self.pipes:
                pipe.update(dt)

            #removing pipes if out of screen
            if len(self.pipes)!=0:
                if self.pipes[0].rect_up.right<0:
                    self.pipes.pop(0)

            #moving the bird
            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)
        if not self.is_game_started:
            self.win.blit(self.restart_text,self.restart_text_rect)


    def setUpBgAndGround(self):
        #loading images for bg and ground
        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()
```



```python
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.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]
            if self.image_index==0: self.image_index=1
            else: self.image_index=0
            self.anim_counter=0

        self.anim_counter+=1
```

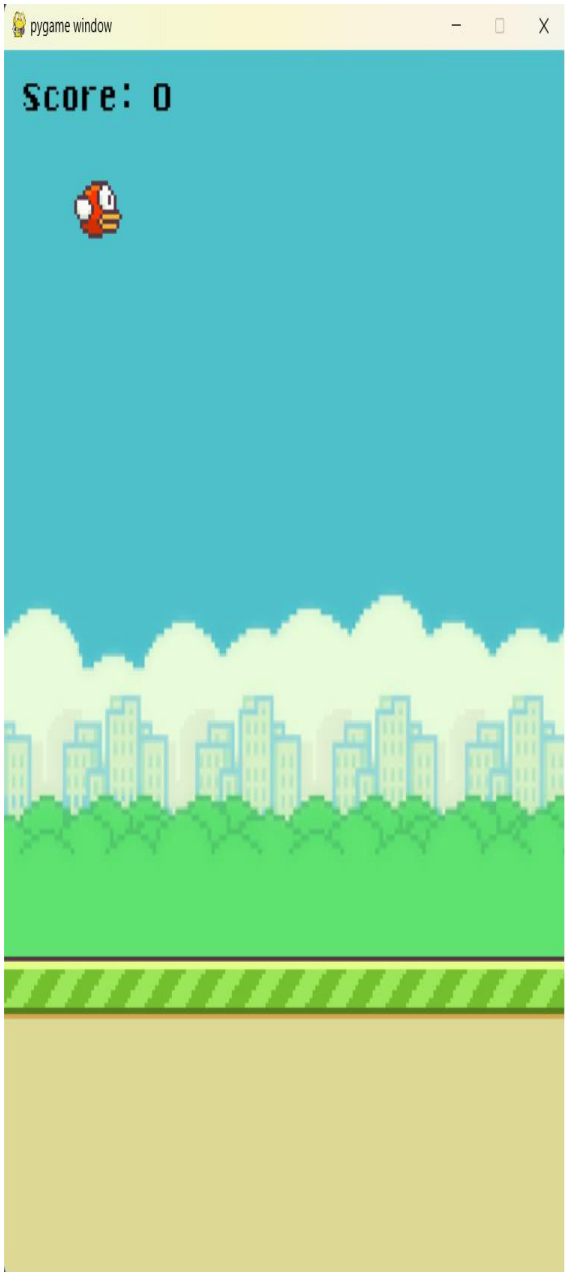


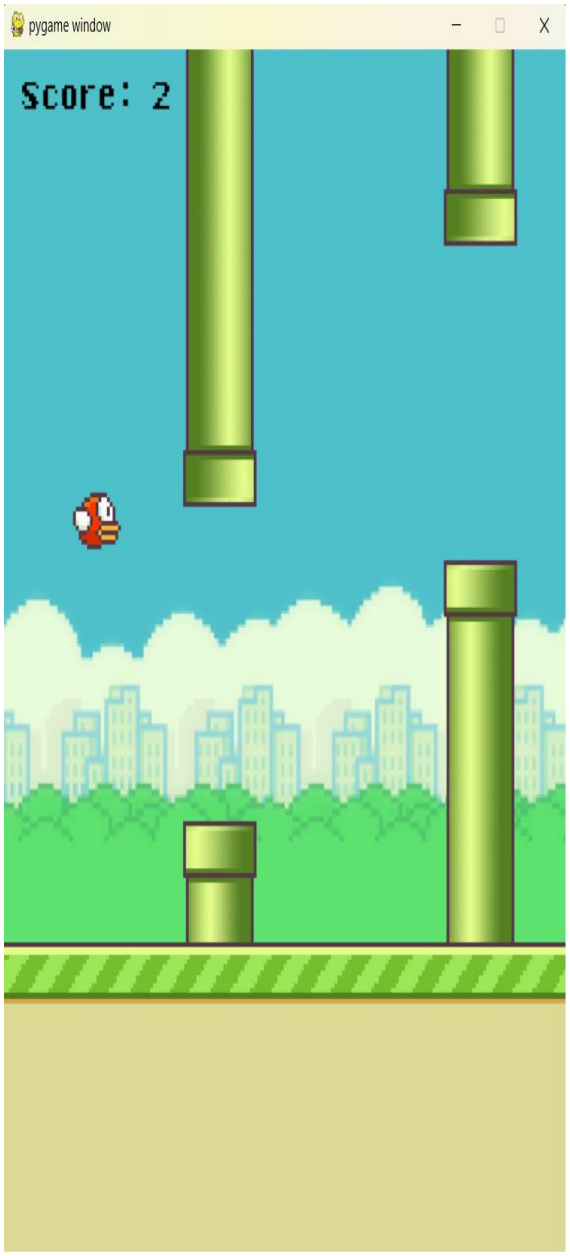
```python
class Bird(pg.sprite.Sprite):
    def update(self,dt):
                self.rect.y=0
                self.flap_speed=0
                self.y_velocity=0
            elif self.rect.y>0 and self.flap_speed==0:
                self.flap_speed=250

    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]
            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
```

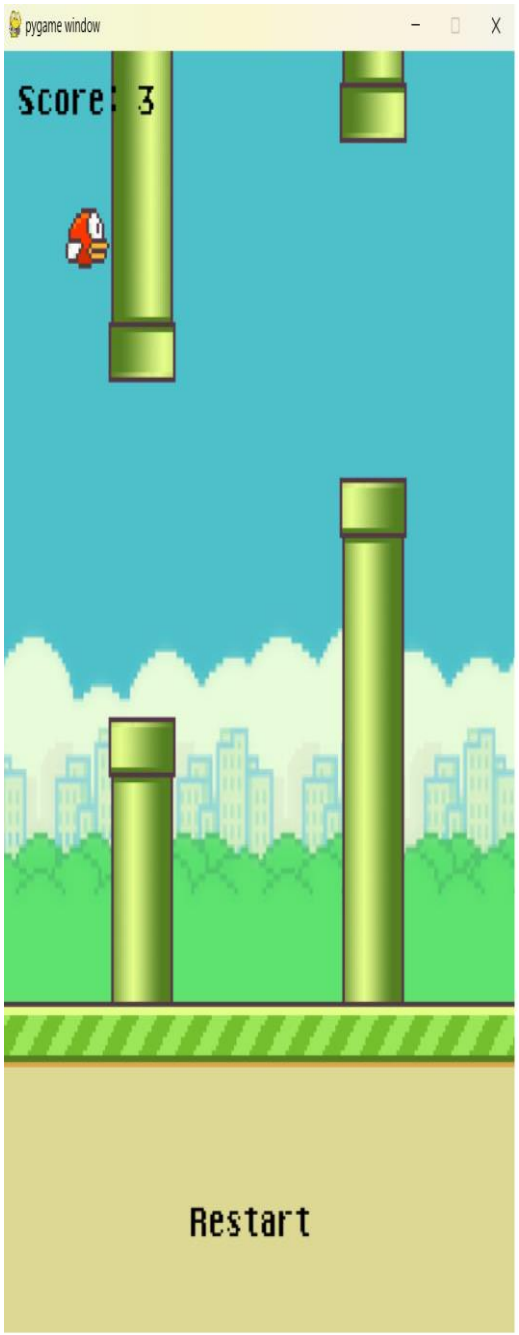


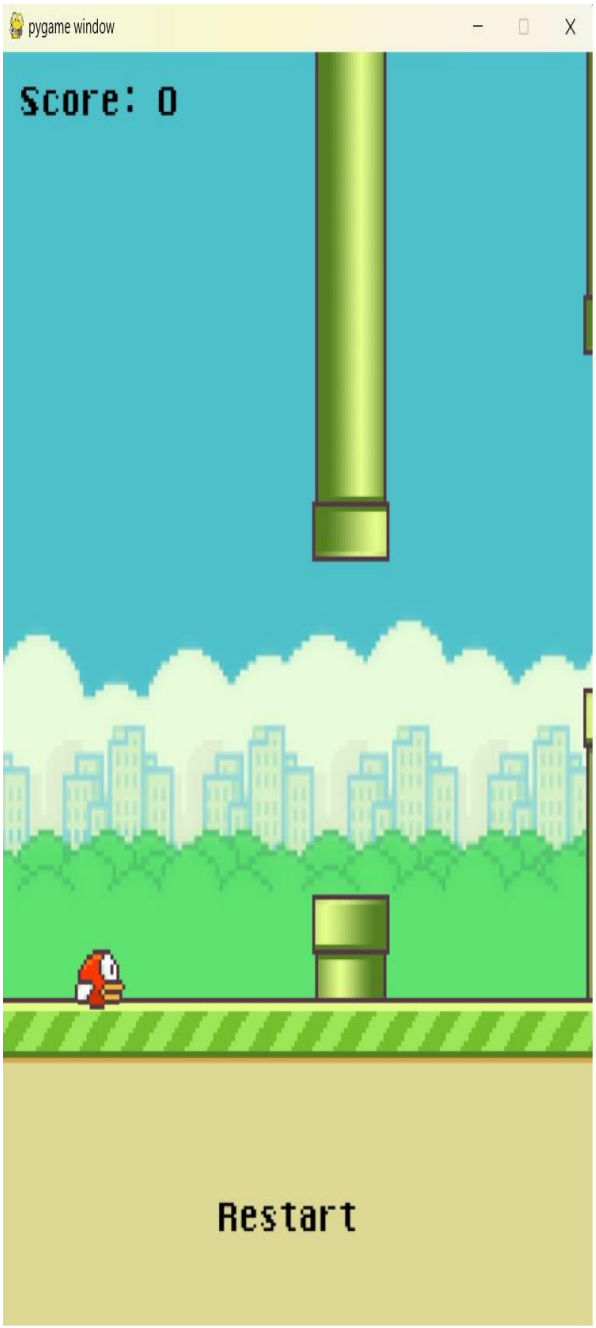
```python
import pygame as pg
from random import randint

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






pygame window
Score: 3
Restart


pygame window
Score: 0
Restart

- **Key Functions**

1. Game Initialization
2. Game Loop
3. Bird Movement and Control
4. Pipe Generation and Movement
5. Collision Detection
6. Reward System
7. AI Decision-Making (using Deep Q-Networks or other algorithms)
8. Training and Model Updates

- **Testing**
- **Simple Test Cases**

1. Bird starts at the initial position and remains still.

2. Bird moves upward when the "jump" action is taken.

3. Bird moves downward due to gravity when no action is taken.

4. Bird collides with the ground or ceiling and ends the game.

5. Bird successfully navigates through a pipe without collision.

6. Bird encounters a pipe that is too low or too high and collides.

7. AI makes decisions to jump or not jump based on pipe positions.

8. AI learns from rewards and penalties to improve gameplay.

- # Observation and Results

Observations:

- The AI bird initially struggles to navigate through pipes, resulting in frequent collisions.

-As the AI learns from rewards and penalties, its performance improves, and it starts to successfully navigate through pipes.

- The AI develops strategies, such as timing jumps and adjusting speed, to optimize its performance.

Results:

- Increased score: The AI's improved performance leads to higher scores.

- Improved survival rate: The AI's ability to navigate through pipes successfully increases its survival rate.

- Enhanced gameplay: The AI's strategies and decision-making processes create a more engaging and challenging gameplay experience.

- Reduced collisions: As the AI learns, the frequency of collisions with pipes, ground, and ceiling decreases.

- **Challenges And Resolution**

Challenges:

1. Balancing Exploration and Exploitation
2. Handling High-Dimensional State Spaces
3. Dealing with Partial Observability
4. Optimizing Rewards and Penalties
5. Ensuring Robustness and Generalization

Resolutions:

1. Balancing Exploration and Exploitation: Implemented epsilon-greedy algorithm to balance exploration and exploitation.

2. Handling High-Dimensional State Spaces: Used dimensionality reduction techniques, such as PCA, to reduce the state space.

3. Dealing with Partial Observability: Designed a partial observability framework to handle incomplete information.

4. Optimizing Rewards and Penalties: Tuned the reward and penalty structure through trial and error and using techniques like reward shaping.

5. Ensuring Robustness and Generalization: Implemented techniques like data augmentation, regularization, and ensemble methods to improve robustness and generalization.

- # Conclusion

The AI-powered Flappy Bird game demonstrates the successful application of machine learning algorithms to a complex game environment. By overcoming challenges such as balancing exploration and exploitation, handling high-dimensional state spaces, and optimizing rewards and penalties, the AI model achieves impressive gameplay performance. This project showcases the potential of AI in game development and opens up possibilities for future innovations.

- **Key Outcomes**

1. Improved gameplay performance
2. Enhanced decision-making capabilities
3. Increased score and survival rate
4. Effective navigation through challenging environments
5. Successful demonstration of AI in game development

- **References**

https://en.wikipedia.org/wiki/Flappy_Bird

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

https://github.com/tuffleroot/flappy-bird-pygame/tree/master

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

- **Appendix**

https://github.com/tuffleroot/flappy-bird-pygame/tree/master

Thanks For Watching