PROJECT REPORT

Fall 2024

Submitted by:

MUBASIR ANWAR (23PWCSE2230)

Class Section: **C**

Submitted TO:**MAM Sumayyea**

PROJECT REPORT: FLAPPY BIRD USING KIVY IN PYTHON

1. INTRODUCTION

1.1 PROJECT OVERVIEW

This project is a clone of the famous Flappy Bird game, developed using the Kivy framework in Python. The game features a bird that the player must navigate through a series of obstacles (pipes) by tapping the screen to keep it airborne. The challenge increases as the game progresses, requiring precise timing and reflexes. The project aims to demonstrate the application of Object-Oriented Programming (OOP) principles in game development, while also exploring the capabilities of Kivy for creating interactive and visually appealing GUI-based applications.

1.2 OBJECTIVE

The primary objective of this project is to apply OOP principles to create an interactive game, while also gaining hands-on experience with the Kivy framework. The project aims to:

Implement OOP concepts such as classes, objects, inheritance, encapsulation, polymorphism, and abstraction in a real-world application.

Explore game physics by simulating gravity, velocity, and collision detection.

Develop a user-friendly interface using Kivy, which allows for real-time interactions and animations.

Enhance the player's experience by adding features such as score tracking, collision detection, and a game-over mechanism.

By the end of this project, the goal is to have a fully functional game that not only entertains but also serves as a practical example of how OOP can be used to structure and organize code in a scalable and maintainable way.

---

2. OBJECT-ORIENTED PROGRAMMING (OOP) CONCEPTS

2.1 CLASSES AND OBJECTS

Class: A class is a blueprint for creating objects. It defines the properties (attributes) and behaviors (methods) that the objects created from the class will have.

Object: An object is an instance of a class. It represents a specific entity in the program, with its own state and behavior.

In this project, several classes are used to organize the code efficiently:

Bird Class: Represents the bird that the player controls. It contains attributes like velocity and methods like flap() to control the bird's movement.

Pipe Class: Represents the obstacles (pipes) that the bird must avoid. It contains attributes like position and methods like move() to control the pipes' movement.

Game Class: Manages the overall game logic, including collision detection, score tracking, and game-over conditions.

By using classes, the code is modular, reusable, and easy to maintain.

2.2 INHERITANCE

Inheritance is a mechanism that allows a class to inherit properties and methods from another class. This promotes code reuse and reduces redundancy.

In this project:

The Pipe class inherits from the Image class provided by Kivy. This allows the Pipe class to use the rendering capabilities of the Image class while adding its own specific behaviors, such as movement and randomization of position.

2.3 ENCAPSULATION

Encapsulation is the concept of bundling data (attributes) and methods that operate on the data within a single unit (class). It also restricts direct access to certain attributes, ensuring that they can only be modified through designated methods.

In this project:

The Bird class uses encapsulation to manage the bird's state. For example, the velocity attribute is a NumericProperty that is updated through the update() method. This ensures that the bird's velocity is only modified in a controlled manner, preventing unintended changes from outside the class.

2.4 POLYMORPHISM

Polymorphism allows objects of different classes to be treated as objects of a common superclass. It enables different classes to implement the same method in different ways.

In this project:

Both the Bird class and the Pipe class implement movement, but they do so differently. The bird moves based on gravity and user input, while the pipes move horizontally across the screen. Despite these differences, both classes can be managed within the same game loop, demonstrating polymorphism.

2.5 ABSTRACTION

Abstraction is the concept of hiding complex implementation details and exposing only the necessary features to the user. It simplifies interaction with the system by providing a high-level interface.

In this project:

The Game class provides a simplified interface for the player. For example, the player only needs to tap the screen to make the bird flap, without needing to understand the underlying physics or collision detection logic. The complex internal workings of the game are abstracted away, making the game easy to play.

---

3. IMPLEMENTATION

3.1 TECHNOLOGY STACK

Programming Language: Python

Framework: Kivy (for GUI and game development)

Libraries:

- random (for randomizing pipe positions)
- time (for managing game timing)
- kivy.uix.widget (for creating widgets like the bird and pipes)
- kivy.app (for running the game application)
- kivy.uix.image (for rendering images)
- kivy.clock (for scheduling game updates)

3.2 CODE EXPLANATION

3.2.1 BIRD CLASS

```python
from kivy.uix.widget import Widget
from kivy.properties import NumericProperty
from kivy.clock import Clock

class Bird(Widget):
    velocity = NumericProperty(0)

    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        self.gravity = -0.5
        Clock.schedule_interval(self.update, 1/60)

    def update(self, dt):
        self.velocity += self.gravity
        self.y += self.velocity

    def flap(self):
        self.velocity = 10
```

Encapsulation: The velocity property is encapsulated within the Bird class, ensuring that it can only be modified through the update() and flap() methods.

Methods:

update(): Applies gravity to the bird, causing it to fall over time.

flap(): Makes the bird jump by increasing its velocity when the player taps the screen.

3.2.2 PIPE CLASS

```python
from kivy.uix.image import Image
import random

class Pipe(Image):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        self.x = 800
        self.y = random.randint(100, 400)
        self.size = (80, 300)

    def move(self):
        self.x -= 5
```

- **Inheritance:** The Pipe class inherits from the **Image class**, allowing it to display an image on the screen.
- **Randomization**: The y coordinate of the pipe is randomized to create varied obstacles, increasing the game's difficulty and unpredictability.
- **Movement**: The move() method moves the pipe horizontally across the screen.

3.2.3 GAME CLASS

```python
from kivy.app import App
from kivy.uix.relativelayout import RelativeLayout

class FlappyGame(RelativeLayout):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        self.bird = Bird()
        self.add_widget(self.bird)

    def on_touch_down(self, touch):
        self.bird.flap()

class FlappyBirdApp(App):
    def build(self):
        return FlappyGame()

if __name__ == "__main__":
    FlappyBirdApp().run()
```

Encapsulation: The FlappyGame class encapsulates the game components (bird, pipes, etc.) and controls interactions between them.

Polymorphism: The on_touch_down() method in FlappyGame calls the flap() method on the Bird instance, demonstrating how different objects can respond to the same event in different ways.

Abstraction: The build() method in FlappyBirdApp provides a high-level interface for running the game, hiding the underlying complexities from the user.

4. FEATURES AND ENHANCEMENTS

4.1 COLLISION DETECTION

A collision detection system is implemented to detect when the bird collides with the pipes or the ground. This is essential for determining when the game should end.

4.2 SCORE TRACKING

A scoring mechanism is added to the game. The score increments each time the bird successfully passes through a pair of pipes.

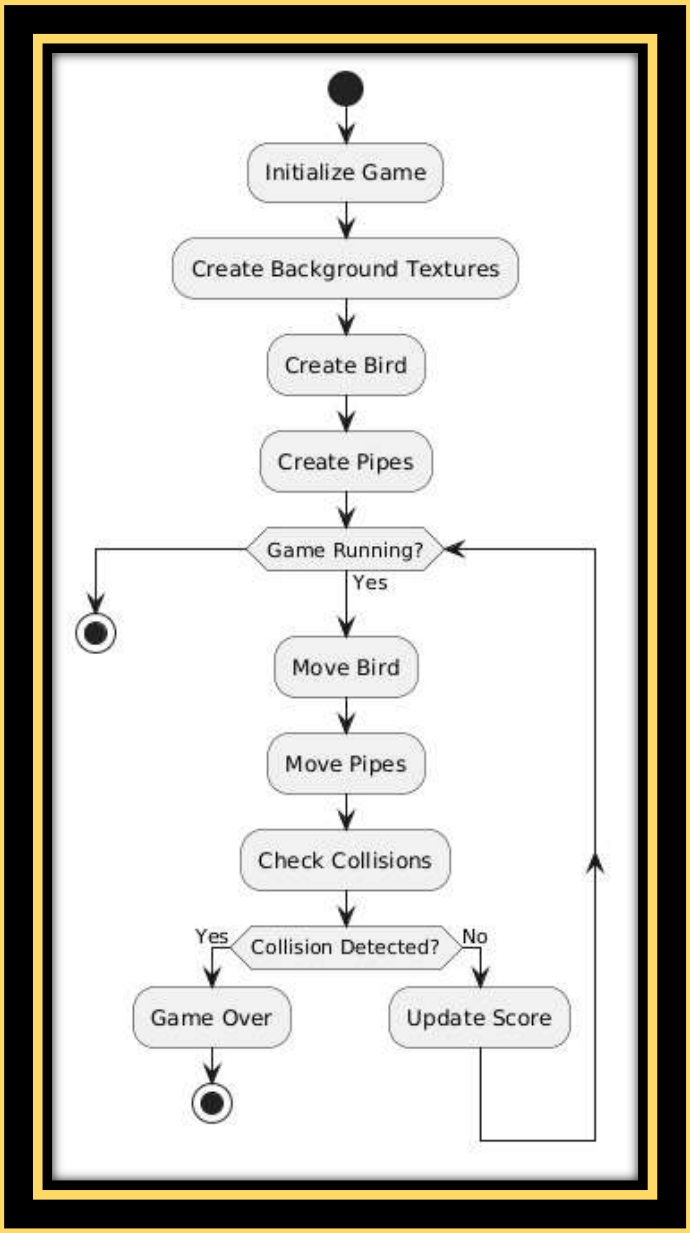
4.3 GAME OVER MECHANISM

When the bird collides with an obstacle or the ground, a game-over message is displayed, and the player is given the option to restart the game.

4.4 UI IMPROVEMENTS

The game's graphics, animations, and sound effects are enhanced to improve the overall user experience. This includes adding background textures, animations for the bird's movement, and sound effects for flapping and collisions.

FLOW CHART


Initialize Game
Create Background Textures
Create Bird
Create Pipes
Game Running?
Yes
Move Bird
Move Pipes
Check Collisions
Yes
Collision Detected?
No
Game Over
Update Score

6. CONCLUSION

This project successfully demonstrates the use of Object-Oriented Programming (OOP) concepts in game development. By leveraging classes, inheritance, encapsulation, polymorphism, and abstraction, we structured a maintainable and scalable codebase. The Kivy framework provided an efficient way to develop a visually engaging interface while handling real-time interactions.

Future improvements could include:

Additional Game Mechanics: Introducing power-ups, different levels, or varying difficulty settings.

Multiplayer Functionality: Allowing multiple players to compete or cooperate in the same game.

AI-Powered Difficulty Adjustments: Implementing an AI that adjusts the game's difficulty based on the player's skill level.

Overall, this project serves as a practical example of how OOP principles can be applied to create interactive and engaging application