**Copyright Submission Form** (Computer Software)

**Title of the Software/Application:** Ping Pong

**Name of authors:** Satya Vardhan Ande, Dr Bhaveshkumar Choithram Dharmani

**Name of Student/Scholar:** Satya Vardhan Ande

**Registration Number:** 12409122

**Program Name:** B. Tech. in ECE

**Name/ UID of Supervisor:** Dr Bhaveshkumar Choithram Dharmani (23409)

**School:** School of Electronics and Communication Engineering

**Name and UID of the all co-authors/Co-Supervisor:** Dr Bhaveshkumar Choithram Dharmani (23409)

**Affiliation details (with Pin code) of co-authors/Co- Supervisor (if External):**

**Language of the Work:** Python

**Summary, Uniqueness and Utility of work:**

**Summary of the Ping Pong Game in Python**

A Ping Pong game (also known as Pong) is a simple two-player or single-player arcade-style game where players control paddles to hit a ball back and forth. In Python, it can be implemented using libraries like pygame that provide tools for creating graphical interfaces, handling user input, and updating game states.

The game usually involves the following components:

1. **Paddles:** Controlled by players to hit the ball.
2. **Ball:** Moves across the screen and bounces off walls and paddles.
3. **Game Area:** The rectangular field where the game takes place, with boundaries for paddles and the ball to interact.
4. **Score System:** Keeps track of points for each player.
5. **Game Logic:** Handles ball movement, paddle collision detection, scoring, and game state (e.g., restart or end game).

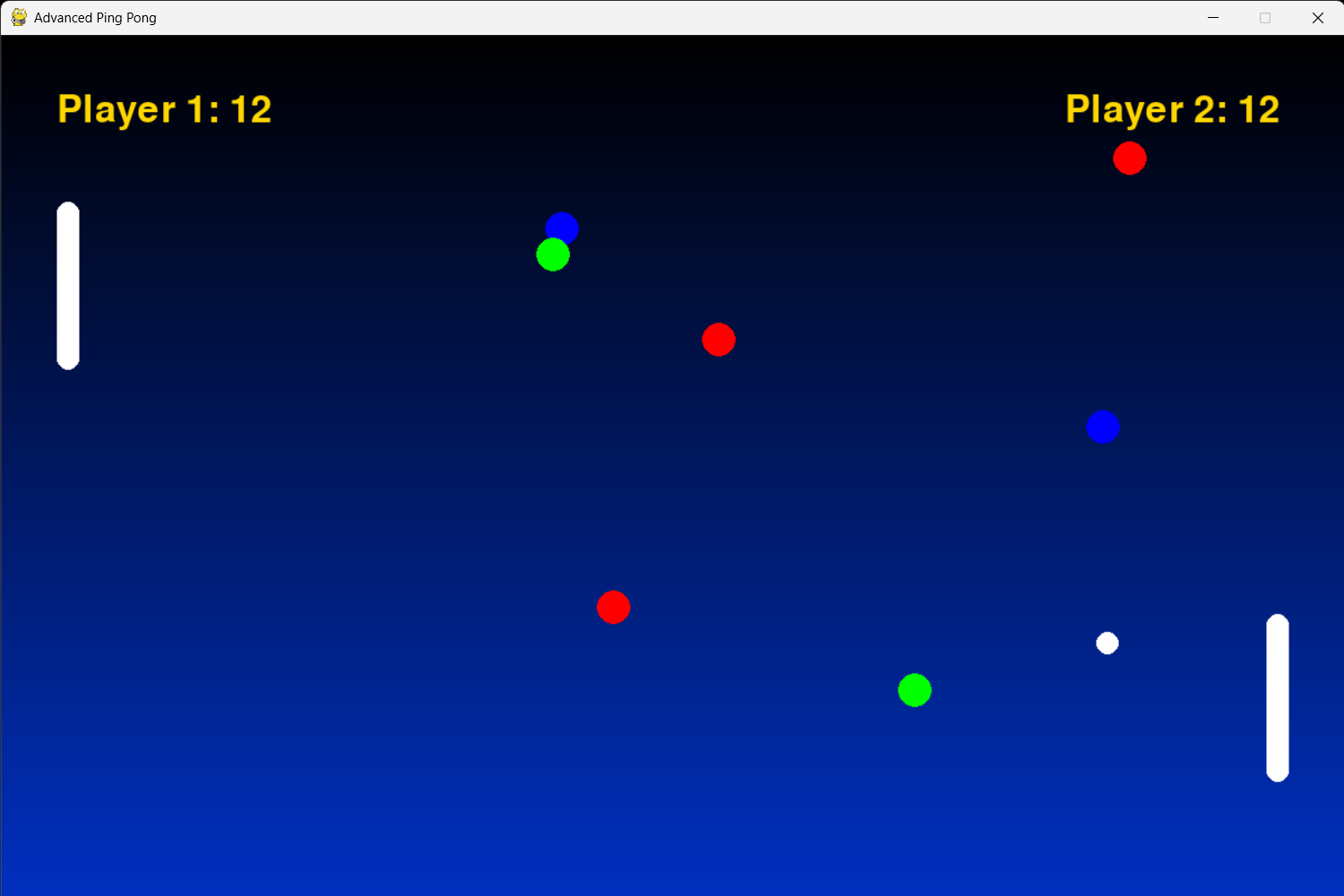
**Uniqueness of the Python Ping Pong Game**

The uniqueness of a Python-based Ping Pong game lies in the flexibility and customization it allows. Python provides a rich ecosystem of libraries and tools that can be used to enhance the game in unique ways:

* **Customization:** You can tweak game elements such as paddle speed, ball trajectory, or scoring rules.
* **Graphics and Sound:** Integration with libraries like pygame allows for custom visual effects and sounds, creating a personalized user experience.
* **AI Integration:** If a single-player mode is desired, Python allows the inclusion of artificial intelligence (AI) to control the opposing paddle.
* **Interactivity:** Python supports dynamic interaction through the keyboard or mouse, enabling players to control the paddles with ease.

**Utility of the Ping Pong Game in Python**

1. **Learning Tool:** The game is an excellent project for beginner programmers. It helps them understand core concepts like event handling, object-oriented programming (OOP), graphics rendering, and game loops.
2. **Game Development Practice:** Building the game helps reinforce skills used in more complex game development, such as collision detection, sprite animation, and real-time user input handling.
3. **Entertainment:** The game can serve as a lighthearted and engaging way to practice programming while also providing enjoyment for users.
4. **Extendable Framework:** The code can serve as a base for creating more advanced games. Developers can expand it by adding features like multiplayer over a network, difficulty levels, or advanced AI algorithms.
5. **Code Reusability:** Once the basic game framework is created, it can be reused for other game projects, reducing the need to start from scratch every time.

**FlowDiagram: **

1. **Start Game**: The game begins when the user launches it. Initialization happens, setting up variables like the ball position, paddles, scores, and game settings.
2. **Initialize Game State**: This involves setting the initial conditions of the game:
   * Ball starting position (center of the screen).
   * Paddle positions (typically at the left and right edges).
   * Score initialization (often starting at zero).
3. **Display Game Screen**: The graphical elements (background, paddles, ball, and score) are rendered onto the screen. This is done continuously to create the illusion of movement.
4. **Handle User Input**: This is where player interactions are captured. For example:
   * Player 1 uses the up and down arrow keys to move their paddle.
   * Player 2 uses the W and S keys, or a mouse for paddle control.
   * If there’s AI, it automatically moves the paddle based on the ball’s position.
5. **Move Ball**: The ball's movement is calculated based on its velocity, bouncing off walls and paddles. Its position is updated frame-by-frame.
6. **Check for Ball Collision**: Each frame, the ball’s new position is checked:
   * If it collides with a paddle, the ball’s direction is reversed.
   * If the ball hits the top or bottom wall, its direction is also reversed.
   * If the ball passes the left or right wall, it scores a point for the opponent.
7. **Update Score & Game**: When the ball passes a player’s paddle (scoring event), the opponent gains a point, and the ball is reset to the center.
8. **Check for Game Over**: After each score update, the game checks if either player has reached the winning score or if a quit condition is met (e.g., player exits the game).
9. **Continue Game Loop or End Game**:
   * If the game is still active (no winner yet), it loops back to the **Display Game Screen** step.
   * If a player reaches the winning score or the game ends, the final score is displayed, and the game ends.

**Optional Enhancements**

* **AI**: If implementing a single-player mode, the computer can control the opposing paddle. A simple AI might use ball prediction to follow the ball’s trajectory and position the paddle accordingly.
* **Difficulty Levels**: You can introduce varying difficulty levels by changing the speed of the ball or the AI’s reaction time.
* **Sound Effects**: Add sounds for ball bounces, scoring, or game over events to enhance user experience.
* **Multiple Players (Online or Local)**: For multiplayer functionality, a networking layer can be added for online play, or multiple controllers for local play.

***Source Code/Object Code*:**

**pip install pygame**

import pygame

import random

import sys

class PowerUp:

def \_\_init\_\_(self, x, y, power\_type):

self.x = x

self.y = y

self.type = power\_type

self.width = 20

self.height = 20

self.color = {

'speed': (255, 0, 0), # Red

'size': (0, 255, 0), # Green

'freeze': (0, 0, 255) # Blue

}[power\_type]

self.duration = 300 # 5 seconds at 60 FPS

def draw(self, screen):

"""Draw power-up with a glowing effect"""

glow\_color = (self.color[0], self.color[1], self.color[2], 128)

pygame.draw.circle(screen, glow\_color, (self.x + self.width // 2, self.y + self.height // 2), self.width // 2 + 5)

pygame.draw.circle(screen, self.color, (self.x + self.width // 2, self.y + self.height // 2), self.width // 2)

class PingPongGame:

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

pygame.init()

# Screen Setup

self.width = 1200

self.height = 800

self.screen = pygame.display.set\_mode((self.width, self.height))

pygame.display.set\_caption("Advanced Ping Pong")

# Colors

self.WHITE = (255, 255, 255)

self.BLACK = (0, 0, 0)

# Game Objects

self.paddle\_width = 20

self.paddle\_height = 150

self.ball\_size = 20

# Paddle Positions

self.player1\_x = 50

self.player1\_y = self.height // 2 - self.paddle\_height // 2

self.player2\_x = self.width - 70

self.player2\_y = self.height // 2 - self.paddle\_height // 2

# Ball Physics

self.ball\_x = self.width // 2

self.ball\_y = self.height // 2

self.ball\_dx = 7 \* random.choice((-1, 1))

self.ball\_dy = 7 \* random.choice((-1, 1))

# Game State

self.player1\_score = 0

self.player2\_score = 0

self.clock = pygame.time.Clock()

self.font = pygame.font.Font(None, 50)

# Power-up System

self.power\_ups = []

self.active\_power\_ups = {'player1': None, 'player2': None}

self.power\_up\_spawn\_timer = 0

# Movement Speed

self.player1\_speed = 8

self.player2\_speed = 8

# Difficulty Progression

self.difficulty\_multiplier = 1.0

def spawn\_power\_up(self):

"""Randomly spawn power-ups on the screen"""

power\_types = ['speed', 'size', 'freeze']

x = random.randint(200, self.width - 200)

y = random.randint(100, self.height - 100)

return PowerUp(x, y, random.choice(power\_types))

def apply\_power\_up(self, player, power\_up):

"""Apply power-up effects to players"""

if player == 'player1':

if power\_up.type == 'speed':

self.player1\_speed \*= 1.5

elif power\_up.type == 'size':

self.paddle\_height \*= 1.5

elif power\_up.type == 'freeze':

self.player2\_speed = 2

else:

if power\_up.type == 'speed':

self.player2\_speed \*= 1.5

elif power\_up.type == 'size':

self.paddle\_height \*= 1.5

elif power\_up.type == 'freeze':

self.player1\_speed = 2

def reset\_power\_ups(self):

"""Reset power-up effects"""

self.player1\_speed = 8

self.player2\_speed = 8

self.paddle\_height = 150

def handle\_events(self):

"""Handle keyboard events for player movement"""

keys = pygame.key.get\_pressed()

# Player 1 Controls (W/S)

if keys[pygame.K\_w] and self.player1\_y > 0:

self.player1\_y -= self.player1\_speed

if keys[pygame.K\_s] and self.player1\_y < self.height - self.paddle\_height:

self.player1\_y += self.player1\_speed

# Player 2 Controls (UP/DOWN Arrow)

if keys[pygame.K\_UP] and self.player2\_y > 0:

self.player2\_y -= self.player2\_speed

if keys[pygame.K\_DOWN] and self.player2\_y < self.height - self.paddle\_height:

self.player2\_y += self.player2\_speed

def update\_ball(self):

"""Update ball movement and handle collisions"""

self.ball\_x += self.ball\_dx

self.ball\_y += self.ball\_dy

# Wall collisions

if self.ball\_y <= 0 or self.ball\_y >= self.height - self.ball\_size:

self.ball\_dy \*= -1

# Paddle collisions

if (self.ball\_x <= self.player1\_x + self.paddle\_width and

self.player1\_y <= self.ball\_y <= self.player1\_y + self.paddle\_height):

self.ball\_dx \*= -1.1

if (self.ball\_x + self.ball\_size >= self.player2\_x and

self.player2\_y <= self.ball\_y <= self.player2\_y + self.paddle\_height):

self.ball\_dx \*= -1.1

# Scoring

if self.ball\_x < 0:

self.player2\_score += 1

self.reset\_ball()

if self.ball\_x > self.width:

self.player1\_score += 1

self.reset\_ball()

def reset\_ball(self):

"""Reset ball position and speed"""

self.ball\_x = self.width // 2

self.ball\_y = self.height // 2

self.ball\_dx = 7 \* random.choice((-1, 1))

self.ball\_dy = 7 \* random.choice((-1, 1))

self.reset\_power\_ups()

def draw\_gradient\_background(self):

"""Draw a gradient background"""

for i in range(self.height):

r = int(0 \* (i / self.height))

g = int(50 \* (i / self.height))

b = int(200 \* (i / self.height))

pygame.draw.line(self.screen, (r, g, b), (0, i), (self.width, i))

def draw\_game(self):

"""Render game objects"""

self.draw\_gradient\_background()

# Draw Paddles with rounded edges

pygame.draw.rect(self.screen, self.WHITE,

(self.player1\_x, self.player1\_y,

self.paddle\_width, self.paddle\_height), border\_radius=10)

pygame.draw.rect(self.screen, self.WHITE,

(self.player2\_x, self.player2\_y,

self.paddle\_width, self.paddle\_height), border\_radius=10)

# Draw Ball with glow effect

pygame.draw.circle(self.screen, (255, 255, 255), (self.ball\_x + self.ball\_size // 2, self.ball\_y + self.ball\_size // 2), self.ball\_size // 2)

# Draw Scores with glow effect

score1 = self.font.render(f"Player 1: {self.player1\_score}", True, (255, 215, 0)) # Gold color

score2 = self.font.render(f"Player 2: {self.player2\_score}", True, (255, 215, 0)) # Gold color

self.screen.blit(score1, (50, 50))

self.screen.blit(score2, (self.width - 250, 50))

# Draw Power-ups

for power\_up in self.power\_ups:

power\_up.draw(self.screen)

pygame.display.flip()

def run(self):

"""Main game loop"""

running = True

while running:

for event in pygame.event.get():

if event.type == pygame.QUIT:

running = False

self.handle\_events()

self.update\_ball()

# Power-up spawning

self.power\_up\_spawn\_timer += 1

if self.power\_up\_spawn\_timer > 600: # Spawn every 10 seconds

self.power\_ups.append(self.spawn\_power\_up())

self.power\_up\_spawn\_timer = 0

self.draw\_game()

self.clock.tick(60) # 60 FPS

pygame.quit()

sys.exit()

# Run the game

if \_\_name\_\_ == "\_\_main\_\_":

game = PingPongGame()

game.run()