Report on

ROCK PAPER SCISSORS – GAME

A Project Work Report

Submitted in the partial fulfilment for the award of the degree of

Master Of Computer Application

by
Prince Kumar (202410116100149)
Neha (202410116100131)
Samiksha teotia (202410116100178)
Rahul (202410116100157)

Session:2024-2025 (II Semester) Under the Supervision of

Mrs. Komal Salgotra (Assistant Professor)
KIET Group of Institutions, Delhi – NCR, Ghaziabad

Department Of Computer Applications
KIET Group of Institutions, Delhi – NCR, Ghaziabad Uttar Pradesh-201206
(2024 – 2025)

DECLARATION

I. The undersigned solemnly declare that the project report is based on my own work carried out during the course of our study under the supervision of **Mrs. Komal Salgotra**. I assert the statements made and conclusions drawn are an outcome of my work. I further certify that the work contained in the report is original and has been done by me under the general supervision of my supervisor.

II. The work has not been submitted to any other Institution for any other degree/diploma/certificate in this university or any other University of India or abroad.

III. We have followed the guidelines provided by the university in writing the report.

IV. Whenever we have used materials (data, theoretical analysis, and text) from other sources, we have given due credit to them in the text of the report and giving their details in the references.

index

INTRODUCTION

The **Rock Paper Scissors Game** is a simple yet engaging project that recreates the traditional hand game in a digital format. This project is built using **Python** making it an excellent choice for beginners and intermediate programmers who want to learn GUI development, event handling, and game logic implementation. The game allows users to compete against the computer, with the system determining the winner based on predefined rules.

Beyond just playing the game, this project includes user authentication, enabling players to sign up and log in to track their progress. It also features a **leaderboard system**, displaying top players based on their performance. The **game history** functionality records past matches, allowing users to review their choices and improve their strategy. Additionally, the interface is enhanced with **background images, animations, and multiple UI screens**, making the gameplay experience more interactive and visually appealing. The use of API integration helps store and retrieve leaderboard data, ensuring a dynamic and competitive environment for users.

The **Rock Paper Scissors Game** is a fun and interactive project that brings the classic hand game to a digital platform using **Python**. It allows users to play against the computer, incorporating a login and signup system to track individual progress. The game includes features like a leaderboard, game history, and dynamic UI enhancements with images and animations to create an engaging experience. By integrating APIs for storing scores and user data, this project not only tests a player's luck and strategy but also demonstrates key programming concepts like event handling, GUI development, and data management in a visually appealing way.

OBJECTIVES OF THE PROJECT

1. **Develop an Interactive Game Interface** – Create a user-friendly and visually appealing GUI using Tkinter, allowing users to seamlessly play the game.

2. **Implement User Authentication** – Include Login and Signup functionality to track individual users and their game history.

3. **Enhance User Experience with Multimedia** – Use images, sounds, and animations to make the game more engaging and interactive.

4. **Introduce a Leaderboard System** – Maintain a ranking system that displays the top players based on their performance.

5. **Store and Retrieve Game History** – Save users' previous game results and display them when required.

6. **Ensure Randomized and Fair Gameplay** – Utilize randomization techniques to allow fair and unpredictable outcomes against the computer.

7. **Integrate API for Score Management** – Use an API to store and fetch leaderboard scores, making the game dynamic and competitive.

8. **Provide Multi-Page Navigation** – Implement a **multi-page UI**, including Home, Game, Leaderboard, and User Profile pages for a structured experience.

9. **Allow Scalability and Future Enhancements** – Write modular and **well-structured code** that supports additional features like multiplayer mode or AI-based opponents.

10. **Encourage Logical Thinking:** The game challenges users to think strategically, improving their ability to predict outcomes based on past patterns.

11. **Introduce Artificial Intelligence (AI) Logic:** Implement an AI opponent that adapts its choices based on the player's previous moves, making the game more challenging.

12. **Enable Data Persistence:** Store user details, scores, and game history in a database or file system for future reference.

13. **Enhance Visual Appeal with Themes & Animations:** Allow users to select from different themes, apply animations, and improve UI aesthetics for a more immersive experience.

14. **Expand to Multiplayer Mode:** Add an option for users to **play against friends** or challenge an online opponent through a network-based system.

15. **Incorporate Learning & Fun Elements:** Provide insights or **fun facts** about game theory, probability, and strategy after each match to make it educational.

16. **Optimize Performance & Responsiveness:** Ensure that the game runs smoothly across different screen sizes and devices, maintaining a high FPS (frames per second) and quick response times.

17. **Allow Game Customization:** Provide settings where users can adjust difficulty levels, change background music, or modify gameplay rules for a personalized experience.

18. **Integrate Leaderboard Rewards:** Gamify the experience by introducing badges, achievements, or virtual rewards for top players to encourage replayability.

19. **Support Multiple Languages:** Expand the game's accessibility by offering a multi-language feature, allowing users to select their preferred language.

Project Requirements (Software/Hardware requirements)

1. **Software Requirements:**

- Programming Language: Python 3.x
- GUI Library: Tkinter (for graphical user interface)
- Image Processing: PIL (Pillow) for handling images
- Sound Effects: Winsound (for Windows) or Pygame (for cross-platform support)
- Database (Optional): SQLite or Firebase (for user authentication and leaderboard)
- API (Optional): Flask or Django (if using a web-based backend for storing scores)
- IDE/Text Editor: VS Code, PyCharm, or Jupyter Notebook
- Additional Libraries:
    - random (for game logic)
    - requests (for API integration)
    - os (for file handling)

2. **Hardware Requirements:**

- Processor: Minimum Intel Core i3 or equivalent
- RAM: At least 4GB RAM (8GB recommended for smooth execution)
- Storage: 500MB free space for storing game assets and user data
- Graphics: Integrated or dedicated graphics (for better UI performance)
- Display: Minimum 1280x720 resolution (for clear UI elements)
- Sound Card (Optional): For playing sound effects during the game

Implementation Details

The **Rock Paper Scissors Game** is a simple yet engaging project implemented using Python and the pygame library. This project provides an interactive graphical user interface (GUI) where players can select **Rock, Paper, or Scissors** and compete against the computer. The game incorporates **sound effects** to enhance user experience and maintains a **scoreboard** to track wins and losses. Additionally, it features a **match history section**, displaying the last five rounds played. The program utilizes **randomization** to generate the computer's moves and employs clear logic to determine the winner based on predefined rules. With its intuitive design and smooth animations, this project effectively demonstrates fundamental concepts of **event handling, graphics rendering, and game development** in Python.

Code –

```
import pygame
import random
import sys

# Initialize pygame and the mixer for sound
pygame.init()
pygame.mixer.init()

# Load sound files
win_sound = pygame.mixer.Sound("win.wav")
lose_sound = pygame.mixer.Sound("lose.wav")
draw_sound = pygame.mixer.Sound("draw.wav")

# Colors
DARK_ORANGE = (255, 140, 0)
WHITE = (255, 255, 255)
BLACK = (0, 0, 0)
RED = (255, 0, 0)
YELLOW = (255, 255, 0)
GREEN = (0, 255, 0)
BLUE = (0, 0, 255)
LIGHT_GRAY = (220, 220, 220)

# Screen settings
WIDTH, HEIGHT = 1000, 600
screen = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("Rock Paper Scissors - Ultimate Edition")

# Load images
rock_img = pygame.image.load("rock.png")
```

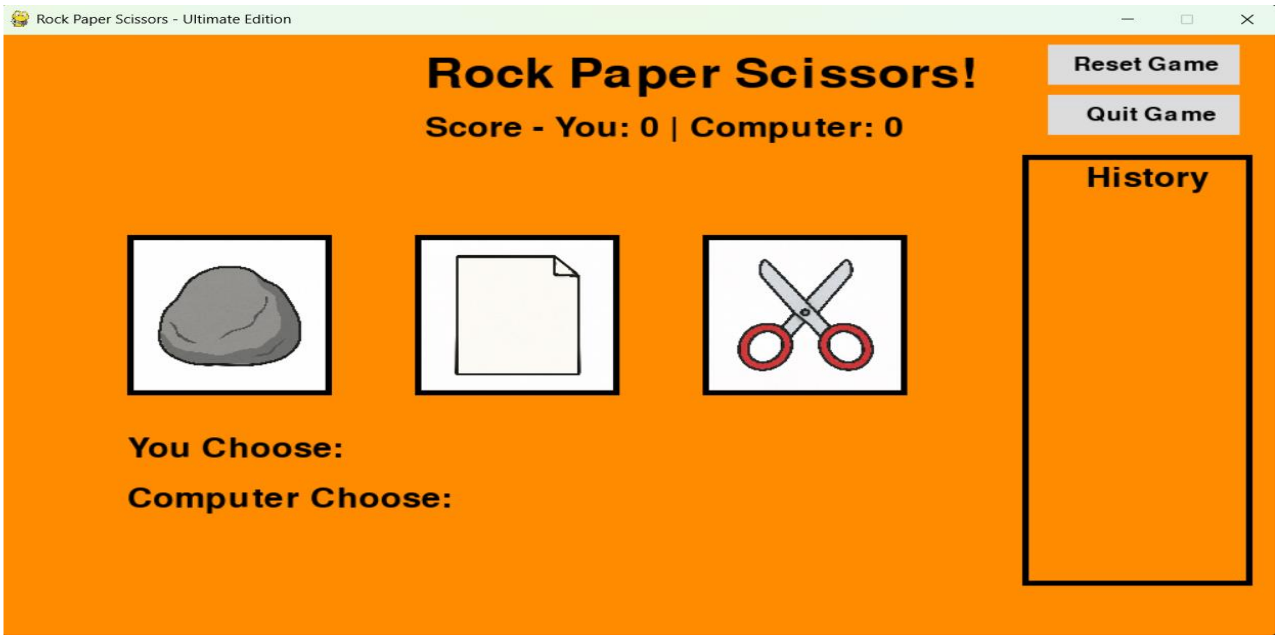
```
paper_img = pygame.image.load("paper.png")
scissors_img = pygame.image.load("scissors.png")

# Resize images
img_size = (150, 150)
rock_img = pygame.transform.scale(rock_img, img_size)
paper_img = pygame.transform.scale(paper_img, img_size)
scissors_img = pygame.transform.scale(scissors_img, img_size)

def add_border(image, border_color=BLACK, border_width=5):
    bordered_img = pygame.Surface((img_size[0] + 2 * border_width, img_size[1] + 2 *
border_width))
    bordered_img.fill(border_color)
    bordered_img.blit(image, (border_width, border_width))
    return bordered_img

# Add border to images
rock_img = add_border(rock_img)
paper_img = add_border(paper_img)
scissors_img = add_border(scissors_img)

# Fonts
font_large = pygame.font.Font(None, 60)
font_medium = pygame.font.Font(None, 40)
font_small = pygame.font.Font(None, 28)

# Game variables
score_player = 0
score_computer = 0
choices = ["rock", "paper", "scissors"]
player_choice = ""
computer_choice = ""
result_text = ""
result_color = YELLOW
history = []

# ✅ Buttons
reset_button = pygame.Rect(820, 10, 150, 40)
quit_button = pygame.Rect(820, 60, 150, 40)

running = True
while running:
    screen.fill(DARK_ORANGE)

    heading_text = font_large.render("Rock Paper Scissors!", True, BLACK)
    screen.blit(heading_text, (WIDTH // 3, 20))

    score_text = font_medium.render(f"Score - You: {score_player} | Computer:
{score_computer}", True, BLACK)
    screen.blit(score_text, (WIDTH // 3, 80))

    screen.blit(rock_img, (100, 200))
    screen.blit(paper_img, (325, 200))
    screen.blit(scissors_img, (550, 200))
    player_text = font_medium.render("You Choose:", True, BLACK)
    screen.blit(player_text, (100, 400))
    choice_text = font_medium.render(player_choice, True, YELLOW)
    screen.blit(choice_text, (280, 400))

    computer_text = font_medium.render("Computer Choose:", True, BLACK)
    screen.blit(computer_text, (100, 450))
    comp_choice_text = font_medium.render(computer_choice, True, YELLOW)
    screen.blit(comp_choice_text, (380, 450))

    result_display = font_large.render(result_text, True, result_color)
    screen.blit(result_display, (WIDTH // 3, 500))

    # History Box
    pygame.draw.rect(screen, BLACK, (800, 120, 180, 430), 5)
    history_title = font_medium.render("History", True, BLACK)
    screen.blit(history_title, (850, 130))

    y_offset = 170
    for round_result in history[-5:]:
        player_line = font_small.render(f"You: {round_result[0]}", True, BLACK)
        computer_line = font_small.render(f"Computer: {round_result[1]}", True, BLACK)
        screen.blit(player_line, (810, y_offset))
        screen.blit(computer_line, (810, y_offset + 20))
        y_offset += 40

    # ✅ Draw Reset and Quit Buttons
    pygame.draw.rect(screen, LIGHT_GRAY, reset_button)
    pygame.draw.rect(screen, LIGHT_GRAY, quit_button)
    screen.blit(font_small.render("Reset Game", True, BLACK), (reset_button.x + 20,
reset_button.y + 10))
    screen.blit(font_small.render("Quit Game", True, BLACK), (quit_button.x + 30,
quit_button.y + 10))

    pygame.display.flip()

    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

        elif event.type == pygame.MOUSEBUTTONDOWN:
            x, y = event.pos

            # ✅ Button Click Handling
            if reset_button.collidepoint(event.pos):
                score_player = 0
                score_computer = 0
                player_choice = ""
                computer_choice = ""
                result_text = ""
                history.clear()

            elif quit_button.collidepoint(event.pos):
                pygame.quit()
                sys.exit()
            # Choice click handling
            if 100 <= x <= 250 and 200 <= y <= 350:
                player_choice = "rock"
            elif 325 <= x <= 475 and 200 <= y <= 350:
                player_choice = "paper"
            elif 550 <= x <= 700 and 200 <= y <= 350:
                player_choice = "scissors"
            else:
                continue

            computer_choice = random.choice(choices)

            if player_choice == computer_choice:
                result_text = "It's a Draw!"
                result_color = BLUE
                draw_sound.play()
            elif (player_choice == "rock" and computer_choice == "scissors") or \
                 (player_choice == "paper" and computer_choice == "rock") or \
                 (player_choice == "scissors" and computer_choice == "paper"):
                result_text = "You Win!"
                result_color = GREEN
                win_sound.play()
                score_player += 1
            else:
                result_text = "You Lose!"
                result_color = RED
                lose_sound.play()
                score_computer += 1

            history.append((player_choice, computer_choice))
            if len(history) > 5:
                history.pop(0)
```

2nd file –

```
import pygame
import random

# Initialize pygame
pygame.init()

# Colors
DARK_ORANGE = (255, 140, 0)  # Dark Orange Background
WHITE = (255, 255, 255)
BLACK = (0, 0, 0)
RED = (255, 0, 0)  # RED for "You Win!" and "You Lose!"
YELLOW = (255, 255, 0)
GREEN = (0, 255, 0)
BLUE = (0, 0, 255)  # BLUE for "It's a Draw!"

# Screen settings
```

```
WIDTH, HEIGHT = 1000, 600  # Increased width for history section
screen = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("Rock Paper Scissors - Ultimate Edition")

# Load images
rock_img = pygame.image.load("rock.png")
paper_img = pygame.image.load("paper.png")
scissors_img = pygame.image.load("scissors.png")

# Resize images
img_size = (150, 150)
rock_img = pygame.transform.scale(rock_img, img_size)
paper_img = pygame.transform.scale(paper_img, img_size)
scissors_img = pygame.transform.scale(scissors_img, img_size)

# Function to add a black border to images
def add_border(image, border_color=BLACK, border_width=5):
    bordered_img = pygame.Surface((img_size[0] + 2 * border_width, img_size[1] + 2 *
border_width))
    bordered_img.fill(border_color)
    bordered_img.blit(image, (border_width, border_width))
    return bordered_img

# Apply black border to images
rock_img = add_border(rock_img)
paper_img = add_border(paper_img)
scissors_img = add_border(scissors_img)

# Font settings
font_large = pygame.font.Font(None, 60)
font_medium = pygame.font.Font(None, 40)
font_small = pygame.font.Font(None, 28)

# Game variables
score_player = 0
score_computer = 0
choices = ["rock", "paper", "scissors"]
player_choice = ""
computer_choice = ""
result_text = ""
result_color = YELLOW  # Default color for result text
history = []  # List to store history of last 5 rounds

running = True
while running:
    screen.fill(DARK_ORANGE)  # Background color changed to DARK ORANGE

    # Heading
    heading_text = font_large.render("Rock Paper Scissors!", True, BLACK)
    screen.blit(heading_text, (WIDTH // 3, 20))

    # Score Display (Color is BLACK)
    score_text = font_medium.render(f"Score - You: {score_player} | Computer:
{score_computer}", True, BLACK)
    screen.blit(score_text, (WIDTH // 3, 80))
    # Display Choices with Black Border
    screen.blit(rock_img, (100, 200))
    screen.blit(paper_img, (325, 200))
    screen.blit(scissors_img, (550, 200))

    # Display 'You Choose' and 'Computer Choose'
    player_text = font_medium.render("You Choose:", True, BLACK)
    screen.blit(player_text, (100, 400))
    choice_text = font_medium.render(player_choice, True, YELLOW)
    screen.blit(choice_text, (280, 400))

    computer_text = font_medium.render("Computer Choose:", True, BLACK)
    screen.blit(computer_text, (100, 450))
    comp_choice_text = font_medium.render(computer_choice, True, YELLOW)

    screen.blit(comp_choice_text, (380, 450))

    # Display Result
    result_display = font_large.render(result_text, True, result_color)
    screen.blit(result_display, (WIDTH // 3, 500))

    # Draw history box
    pygame.draw.rect(screen, BLACK, (800, 50, 180, 500), 5)  # Black border for history
section
    history_title = font_medium.render("History", True, BLACK)
    screen.blit(history_title, (850, 60))

    # Display last 5 rounds in history (in 2-line format)
    y_offset = 100
    for round_result in history[-5:]:  # Show only last 5 rounds
        # Split round_result into two lines for player and computer
        player_line = font_small.render(f"You: {round_result[0]}", True, BLACK)
        computer_line = font_small.render(f"Computer: {round_result[1]}", True, BLACK)

        screen.blit(player_line, (810, y_offset))
        screen.blit(computer_line, (810, y_offset + 20))

        y_offset += 40  # Adjust spacing for compact display
    pygame.display.flip()

    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
        elif event.type == pygame.MOUSEBUTTONDOWN:
            x, y = event.pos
            if 100 <= x <= 250 and 200 <= y <= 350:
                player_choice = "rock"
            elif 325 <= x <= 475 and 200 <= y <= 350:
                player_choice = "paper"
            elif 550 <= x <= 700 and 200 <= y <= 350:
                player_choice = "scissors"

            if player_choice:
                computer_choice = random.choice(choices)
            if player_choice == computer_choice:
                result_text = "It's a Draw!"
                result_color = BLUE  # BLUE for "It's a Draw!"
                history.append((player_choice, computer_choice))  # Add to history as tuple
            elif (player_choice == "rock" and computer_choice == "scissors") or \
                 (player_choice == "paper" and computer_choice == "rock") or \
                 (player_choice == "scissors" and computer_choice == "paper"):
                result_text = "You Win!"
                result_color = RED  # RED for "You Win!"
                score_player += 1
                history.append((player_choice, computer_choice))  # Add to history as tuple
            else:
                result_text = "You Lose!"
                result_color = RED  # RED for "You Lose!"
                score_computer += 1
                history.append((player_choice, computer_choice))  # Add to history as tuple

pygame.quit()
```

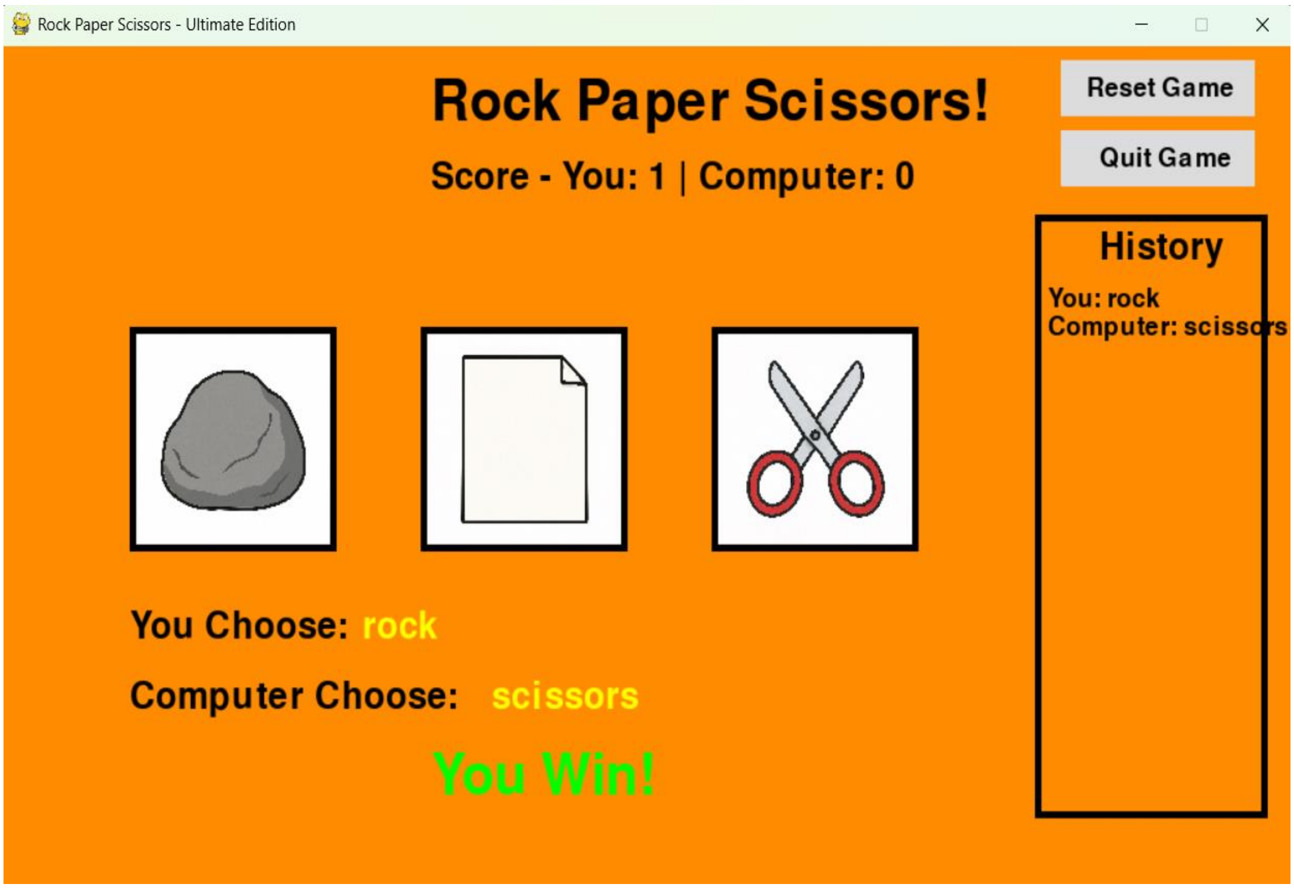
Output Analysis



Score - You: 0 | Computer: 0

You Choose:

Computer Choose:



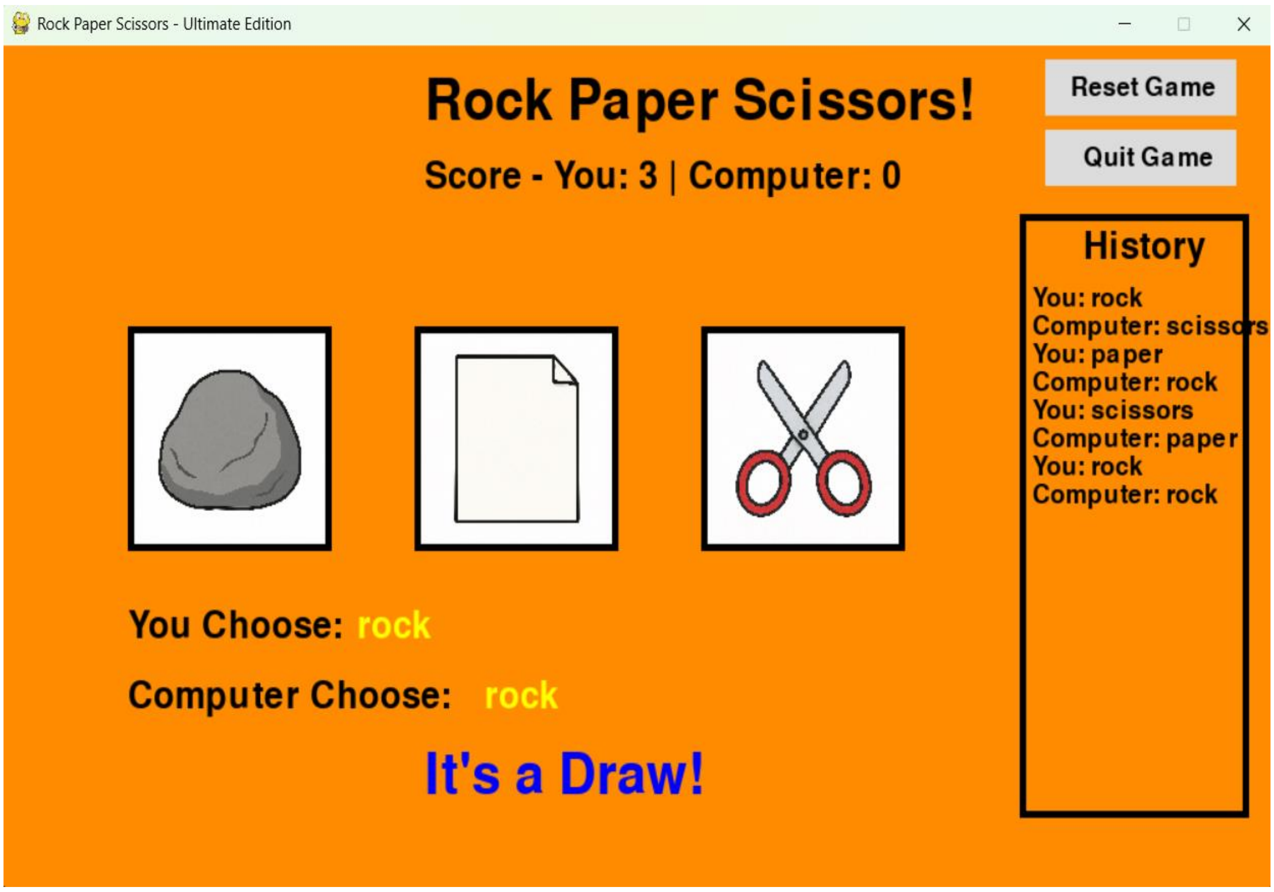
Score - You: 1 | Computer: 0

You: rock
Computer: scissors

You Choose: rock

Computer Choose: scissors

You Win!


Rock Paper Scissors - Ultimate Edition

Rock Paper Scissors!

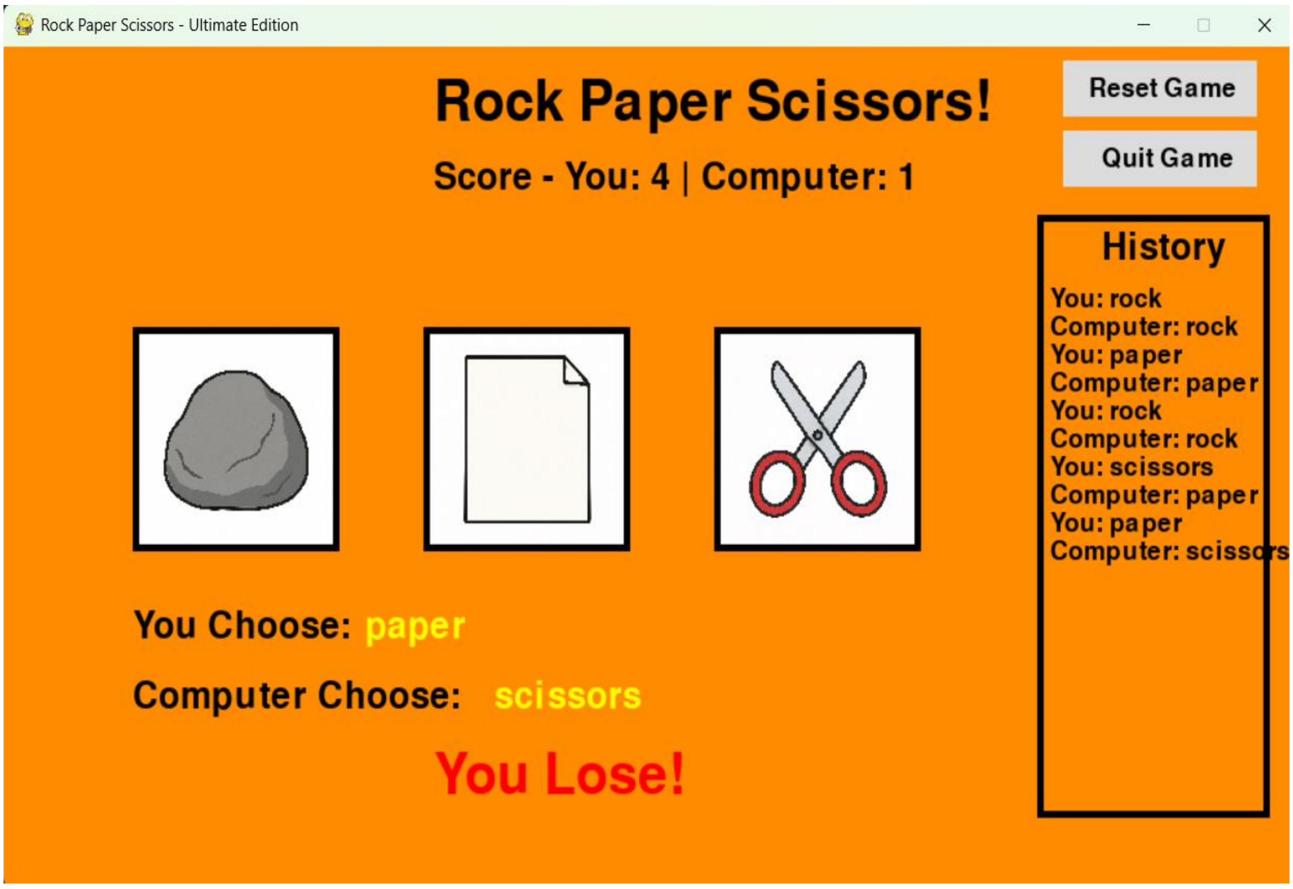
Score - You: 3 | Computer: 0

History
You: rock
Computer: scissors
You: paper
Computer: rock
You: scissors
Computer: paper
You: rock
Computer: rock

You Choose: rock

Computer Choose: rock

It's a Draw!


Rock Paper Scissors - Ultimate Edition

Rock Paper Scissors!

Score - You: 4 | Computer: 1

History
You: rock
Computer: rock
You: paper
Computer: paper
You: rock
Computer: rock
You: scissors
Computer: paper
You: paper
Computer: scissors

You Choose: paper

Computer Choose: scissors

You Lose!

Conclusion

The **Rock Paper Scissors Game** successfully demonstrates the implementation of an interactive GUI-based game using Python and the pygame library. This project not only enhances the player's gaming experience through sound effects, animations, and a match history feature but also reinforces key programming concepts such as event handling, randomization, and graphical rendering. The structured design and intuitive user interface make it easy to understand and play. Through this project, we explored the integration of multimedia elements, logic building, and user interaction in Python. It serves as a great example of how simple logic-based games can be transformed into engaging applications with visual and audio enhancements. Future improvements could include multiplayer mode**,** AI difficulty levels, and additional customization options, making it even more exciting and interactive.