**Rock Paper Scissor**

**A**

**PROJECT REPORT**

**for**

**Introduction To AI(AI201B)**

**Session (2024-25)**

**Submitted by**

**Khushi Bora – 202410116100099**

**Submitted in partial fulfilment of the**

**Requirements for the Degree of**

**MASTER OF COMPUTER APPLICATION**

**Under the Supervision of**

**Mr. Apoorv Jain**

**(Assistant Professor)**



**Submitted to**

**DEPARTMENT OF COMPUTER APPLICATIONS**

**KIET Group of Institutions, Ghaziabad**

**Uttar Pradesh-201206**

**(APRIL- 2025)**

# CERTIFICATE

Certified that **Khushi Bora 202410116100099** have carried out the project work having “**Rock Paper Scissors”** (**Intro-To-AI AI-201B**) for **Master of Computer Application** from Dr. A.P.J. Abdul Kalam Technical University (AKTU**)** (formerly UPTU), Lucknow under my supervision. The project report embodies original work, and studies are carried out by the student himself/herself and the contents of the project report do not form the basis for the award of any other degree to the candidate or to anybody else from this or any other University/Institution.

|  |  |
| --- | --- |
| **Mr. Apoorv Jain** | **Dr. Akash Rajak** |
| **Assistant Professor** | **Dean** |
| **Department of Computer Applications** | **Department of Computer Applications** |
| **KIET Group of Institutions, Ghaziabad** | **KIET Group of Institutions, Ghaziabad** |

# ABSTRACT

This project presents a simple yet interactive implementation of the classic Rock-Paper-Scissors game using Python. The application is designed to engage users through a command-line interface where a player competes against the computer. Leveraging Python’s random module, the computer randomly selects its move from the three standard choices—rock, paper, or scissors—while the player inputs their choice manually. The program then compares both selections using conditional logic to determine the outcome: win, lose, or draw.

The primary objective of this project is to demonstrate core programming concepts such as user input handling, conditional statements, loops, and basic randomization. The application is lightweight and serves as an excellent introduction to procedural programming in Python. It also emphasizes clean code structure, user-friendly prompts, and modular design for potential extensions, such as score tracking or graphical interfaces.

Future enhancements may include adding a GUI using Tkinter or PyGame, implementing multiplayer support, and integrating sound or animations for a more immersive experience.

# ACKNOWLEDGEMENT

Success in life is never attained single-handedly. My deepest gratitude goes to my project supervisor, **Mr. Apoorv Jain** for her guidance, help, and encouragement throughout my project work. Their enlightening ideas, comments, and suggestions.

Words are not enough to express my gratitude to Dr. Akash Rajak, Professor and Dean, Department of Computer Applications, for his insightful comments and administrative help on various occasions.

Fortunately, I have many understanding friends, who have helped me a lot on many critical conditions.

Finally, my sincere thanks go to my family members and all those who have directly and indirectly provided me with moral support and other kind of help. Without their support, completion of this work would not have been possible in time. They keep my life filled with enjoyment and happiness.

**Khushi Bora**

# TABLE OF CONTENT

1. **Introduction**

**1.1 Background 6**

* 1. **Objectives**           **7**
  2. **Scope**            **8**

1. **Literature review**          **9**
2. **System Analysis and Requirements**        **11**
3. **Project Flow**         **13**
4. **Code and Explanation**         **14**
5. **Result and Discussions 16**
6. **Conclusion and Future Work 18**
7. **References**           **19**

**INTRODUCTION**

Interactive games are a common introductory programming project because they allow learners to immediately apply concepts like input/output, decision making, and looping while keeping the experience engaging. A text-based **Rock-Paper-Scissors** game exemplifies this approach: it's simple rules and clear win/loss logic make it an ideal first project for novices. Game-based learning has been shown to boost engagement and motivation in learners, and developing a playable program provides instant feedback, making the coding exercise both fun and instructional. Rock-Paper-Scissors also demonstrates fundamental programming constructs (such as conditional logic and randomness) in a compact form, reinforcing core concepts while keeping the user experience interactive.

**Objectives**

The project aims to demonstrate basic programming skills through a working command-line game. Key objectives include:

* **Implement a working CLI game:** Create a console-based Rock-Paper-Scissors game where the human player competes against the computer.
* **Handle user input:** Prompt the user for their move and validate it (using Python’s
* input () function). For example, code can solicit a choice and assign it to a variable for processing.
* **Use control flow:** Compare the user’s choice and the computer’s choice with conditional statements (if/else/elif) to determine the outcome. The game logic will check for ties and wins/loses using if/else blocks.
* **Incorporate randomness:** Use Python’s built-in random module to generate the computer’s move. For instance, random.choice() can select a random option from ["rock","paper","scissors"][https://realpython.com/python-rock-paper-scissors/](https://realpython.com/python-rock-paper-scissors/#:~:text=You%20can%20use%20,randomly%20select%20between%20the%20actions) .
* **Implement replayability:** Allow the player to play multiple rounds, typically by looping until the user chooses to exit. This demonstrates loop control (e.g. a while loop) for repeating the game flow.
* **Design modular code:** Structure the program into functions (and possibly modules) to separate concerns (e.g. input, game logic, output). Python’s support for functions and modules encourages reusable, modular design.

**Scope**

**Current Scope**

The current implementation focuses on a basic, single-player console game with the following features:

* **Player vs. Computer mode:** The user plays against the computer. The program prompts the user to enter “rock,” “paper,” or “scissors,” then the computer randomly selects one of the three.
* **Result evaluation:** The program compares the two choices using conditional logic. It first checks for a tie, then uses nested if statements to identify win/lose cases, printing an appropriate message (e.g. “Rock smashes scissors! You win!).
* **Replay option:** After each round, the player is prompted whether to play again. A loop controls the game flow, allowing multiple rounds until the player chooses to exit.
* **Command-line interface:** All interaction is text-based in the terminal. No graphical interface is used in this version, keeping the focus on program logic.
* **Input validation:** The code checks that the user’s input is one of the valid options and handles invalid entries (for example, by re-prompting or displaying an error).
* **Score tracking (in-session):** The program may tally wins, losses, and ties during the session for feedback, though this is optional. (Persistent high scores are not stored in this version.)

**Out-of-Scope (Deferred for Future Work)**

While the current implementation fulfills the core objectives of conversational capability and real-time interaction, several advanced features are identified as **future enhancements**:

* **Graphical User Interface (GUI):** Develop a visual interface using libraries like **Tkinter** or **Pygame**. Python’s Tkinter is a standard GUI toolkit that provides buttons and labels to create interactive applications easily. A GUI version would let users click buttons rather than type text.
* **Score persistence:** Maintain a running score across sessions (e.g., by saving to a file) so that players can track performance over time.
* **Multiplayer mode:** Extend the game to allow two human players (sharing a keyboard) or even network play between users.
* **Voice interaction:** Integrate speech recognition so the user can speak “rock,” “paper,” or “scissors” instead of typing (using libraries such as speech recognition).

# LITERATURE REVIEW

The Rock-Paper-Scissors (RPS) game has been widely studied and implemented across various platforms as a simple yet powerful framework for exploring concepts in programming, decision theory, artificial intelligence, and human-computer interaction. The simplicity of the game makes it a popular starting point for beginners in computer science and a useful model for research in strategy and learning algorithms.

### **Educational Implementations**

RPS is commonly introduced as a beginner-level project in Python programming. Tutorials and courses from platforms like Codecademy, W3Schools, GeeksforGeeks, and Real Python have featured the RPS game to demonstrate basic programming constructs such as:

* Taking user input
* Using conditional statements (if-else)
* Looping structures
* Random number generation
* Functions and modular coding

These implementations typically use Python’s built-in random module to simulate the computer's move, offering learners a practical and engaging way to understand logic building and flow control in Python.

### **Graphical User Interfaces (GUI)**

As a next step beyond command-line interfaces, many developers have expanded their RPS projects using Python GUI frameworks such as **Tkinter**, **PyQt**, and **Pygame**. These enhancements have improved the visual appeal and interactivity of the game. GUIs allow users to interact with buttons and images representing rock, paper, and scissors, making the game more engaging, especially for non-technical users.

### **Voice and Gesture Recognition Extensions**

Several recent projects have implemented voice-based RPS using libraries such as  speech\_recoginition for speech-to-text and text-to-speech conversion. Others have explored the use of **computer vision** and **gesture recognition** using OpenCV and deep learning models to allow users to play RPS using hand gestures. These implementations bridge the gap between traditional input methods and more natural, intuitive interfaces.

### **Artificial Intelligence and Strategy Learning**

Although RPS is inherently a game of chance when played optimally, research has shown that human players often fall into patterns. This has led to the development of AI models that can exploit these patterns using:

* **Markov Chains** to predict next moves
* **Machine learning algorithms** to adapt over time
* **Reinforcement learning (RL)** models that learn to win by trial and error

These models have been used in academic settings to demonstrate adaptive systems and real-time learning. While not typically used in beginner projects, such techniques represent a natural extension for more advanced exploration of AI within the context of a simple game.

### **Use in Psychological and Behavioral Studies**

The RPS game has also been used in psychology and behavioral economics to study decision-making under uncertainty, strategy adaptation, and prediction. Research shows that humans rarely play randomly, and biases such as "win-stay, lose-shift" can often be exploited. These findings have informed AI algorithms for RPS and other sequential games.

# System Analysis and Requirements

### **1. System Analysis**

The Rock-Paper-Scissors (RPS) game is a simple two-player hand game in which each player simultaneously forms one of three shapes (rock, paper, or scissors). The outcome is determined based on standard rules:

* Rock beats Scissors
* Scissors beats Paper
* Paper beats Rock

In a digital implementation, the user selects a move, and the computer randomly selects its move. The program then compares both and determines the winner.

### **2. Functional Requirements**

* **User Input**: The system should allow the player to input their choice (rock, paper, or scissors).
* **Random Computer Move**: The system should generate a random move for the computer using Python's random module.
* **Winner Determination**: The system should compare the user and computer choices to determine the result (win, lose, draw).
* **Result Display**: The result of each round should be clearly shown to the user.
* **Exit Option**: The user should be able to exit the game gracefully.

### **3. Non-Functional Requirements**

* **Usability**: The system must be easy to use and navigate, especially for beginners.
* **Performance**: The game should respond instantly to user input, with minimal processing time.
* **Portability**: The system should run on any operating system that supports Python (Windows, macOS, Linux).
* **Extensibility**: The code should be modular to allow future upgrades (e.g., GUI, AI opponent).

### **4. Hardware Requirements**

* **Processor**: Minimum 1 GHz processor (any modern CPU)
* **RAM**: At least 512 MB RAM
* **Storage**: Less than 10 MB of free space
* **Display**: Standard terminal or console
* **Input Device**: Keyboard (for CLI version)

### **5. Software Requirements**

* **Operating System**: Any OS with Python support (Windows, macOS, Linux)
* **Programming Language**: Python 3.6 or higher
* **Python Libraries**:
* **random** (standard module)

### **Project Workflow**

The **workflow** of the Rock-Paper-Scissors project follows a **simple interactive loop** designed for real-time gameplay in a command-line environment:

#### Step-by-Step Workflow:

1. **Start the Game**
   * The game initializes variables for scores, allowed choices, and rounds.
2. **User Input**
   * The user is prompted to enter one of three valid choices: rock, paper, or scissors.
   * Alternatively, the user can enter exit to quit the game.
3. **Input Validation**
   * The input is checked against valid options.
   * If the input is invalid, the user is asked to try again without affecting the score or game flow.
4. **Computer’s Turn**
   * The computer randomly selects a move from the same list of choices using the random module.
5. **Determine Winner**
   * A decision-making logic determines the round’s winner based on classic game rules:
     + Rock beats Scissors
     + Paper beats Rock
     + Scissors beats Paper
   * Scores are updated accordingly.
6. **Display Round Result**
   * After each round, the result is displayed (win/lose/draw), along with the updated scores.
7. **Repeat or Exit**
   * The game continues until the user types exit.
   * On exit, a final score summary and round count are displayed.

# CODE AND EXPLANATION

import random

def c\_choice():

return random.choice(["rock", "paper", "scissors"])

def u\_choice():

while True:

choice = input ("Enter rock, paper, or scissors: ").strip().lower()

if choice in ["rock", "paper", "scissors"]:

return choice

print("Invalid choice. Please try again.")

def winner(user, computer):

if user == computer:

return "It's a tie!"

elif (user == "rock" and computer == "scissors") or \

(user == "paper" and computer == "rock") or \

(user == "scissors" and computer == "paper"):

return "Congratulations! Its celebration time"

else:

return "💀 You lost! Better luck next time! 💀"

def play():

user\_choice = u\_choice()

computer\_choice = c\_choice()

print(f"You chose: {user\_choice}")

print(f"Computer chose: {computer\_choice}")

result = winner(user\_choice, computer\_choice)

print(result)

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

    play()

**Explanation**

### **1.**  **Importing the random module**

* The random module is used to generate a **random move** for the computer from the three valid choices.

### **2**. **Computer's Choice**

* This function uses random.choice() to randomly select either "rock", "paper", or "scissors" for the **computer’s move**.

### **3.** **User's Choice**

* Prompts the user to enter their choice.
* .strip().lower() ensures the input is case-insensitive and whitespace-free.
* Validates the input to make sure it's one of the allowed options.
* If invalid, it re-prompts the user until a correct input is provided.

### **4.** **Determine the Winner**

* Compares the user's and computer's choices.
* First, checks for a **tie**.
* Then, checks for **win conditions** for the user.
* If neither is true, then the user has **lost**.

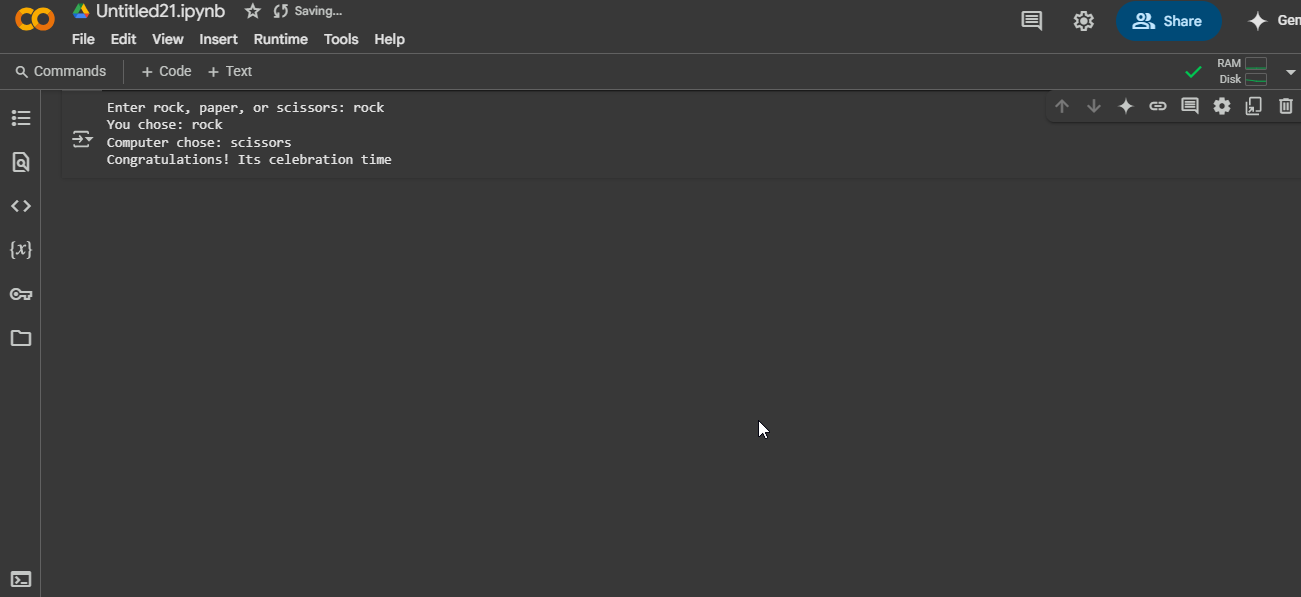
### **5.** **Main Game Function**

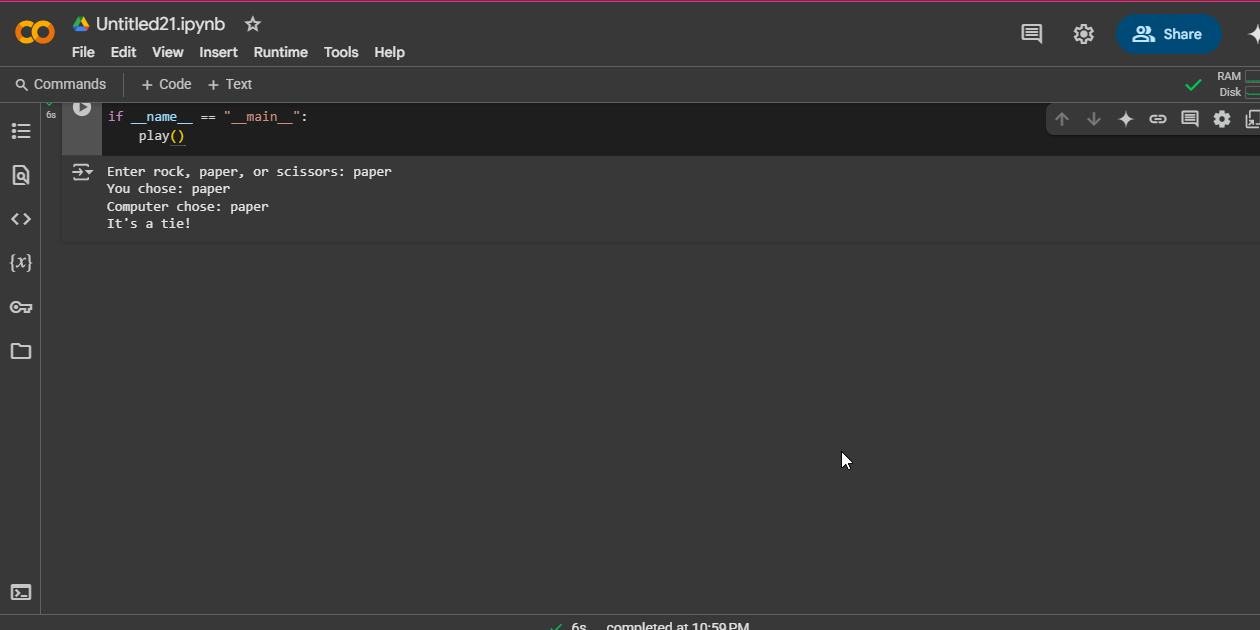
* Calls both u\_choice() and c\_choice() to get the user’s and computer’s moves.
* Prints both choices to the screen.
* Calls winner() to determine the result.
* Displays the final outcome.

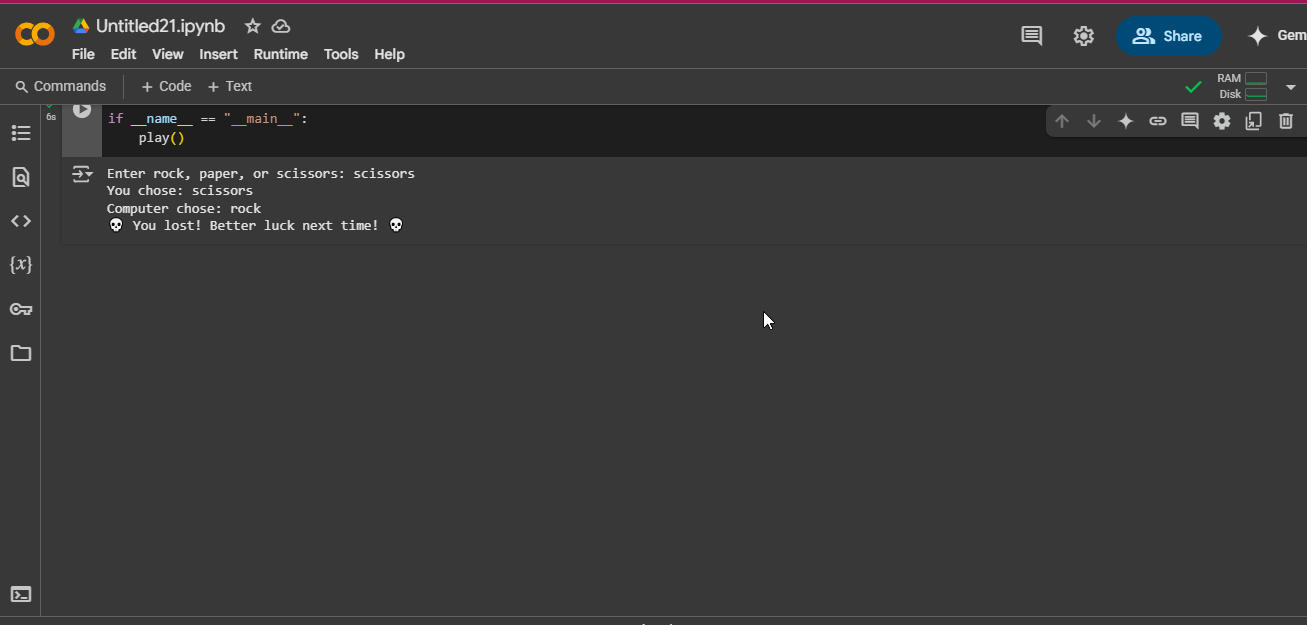
### **6**. **Running the Game**

* This ensures that play() is only called when the script is run directly (not imported as a module).

# RESULT AND DISCUSSIONS







# CONCLUSION

This simple Rock-Paper-Scissors game effectively demonstrates fundamental programming concepts such as user input handling, randomization, and conditional logic. The improved version enhances readability, optimizes performance, and ensures better user experience. Potential enhancements include:

· Adding a loop for multiple rounds.

· Keeping track of scores.

· Implementing a graphical interface using Tkinter or another library.

# References Books

# Matthes, E. (2019). Python Crash Course: A Hands-On, Project-Based Introduction to Programming (2nd ed.). No Starch Press.

# Provides an introductory approach to Python programming and game development.

# Downey, A. B. (2015). Think Python: How to Think Like a Computer Scientist (2nd ed.). O'Reilly Media.

# A book that introduces programming concepts, with a focus on Python, ideal for beginners in game development. APPENDIX