# ROCK PAPER SCISSORS

**A PROJECT REPORT**

***Submitted by***

Arka Banerjee(23BCS11474)

Anmol Kumar(23BCS11187)

Ayush Agnihotri(23BCS11800)

***in partial fulfillment for the award of the degree of***

**BACHELOR OF ENGINEERING**

**IN**

ELECTRONICS ENGINEERING


**Chandigarh University**

JUNE 2023


# BONAFIDE CERTIFICATE

Certified that this project report **"ROCK PAPER SCISSORS"** is the bonafide work of "Arka, Anmol and Ayush" who carried out the project work under my/our supervision.

| | |
|---|---|
| _____ | _____ |
| **SIGNATURE** | **SIGNATURE** |
| | Mrs. Jyoti Arora |
| **HEAD OF THE DEPARTMENT** | **SUPERVISOR** |
| Computer Science Engineering | Computer Science Engineering |

Submitted for the project viva-voce examination held on_____

| | |
|---|---|
| **INTERNAL EXAMINER** | **EXTERNAL EXAMINER** |

# List of Figures



## TABLE OF CONTENTS

# ABSTRACT

The Rock Paper Scissors AI Game is a modern reinterpretation of the traditional hand-gesture-based game, implemented using Java Swing and enhanced with predictive artificial intelligence capabilities. The core objective of the project is to offer an interactive, visually appealing, and intelligent game environment where the computer opponent is not just a random entity, but one that adapts to the player's behavioral patterns over time. This combination of graphical user interface (GUI) design and frequency-based AI prediction introduces both recreational value and educational insight into event-driven programming, UI development, and elementary machine learning concepts.

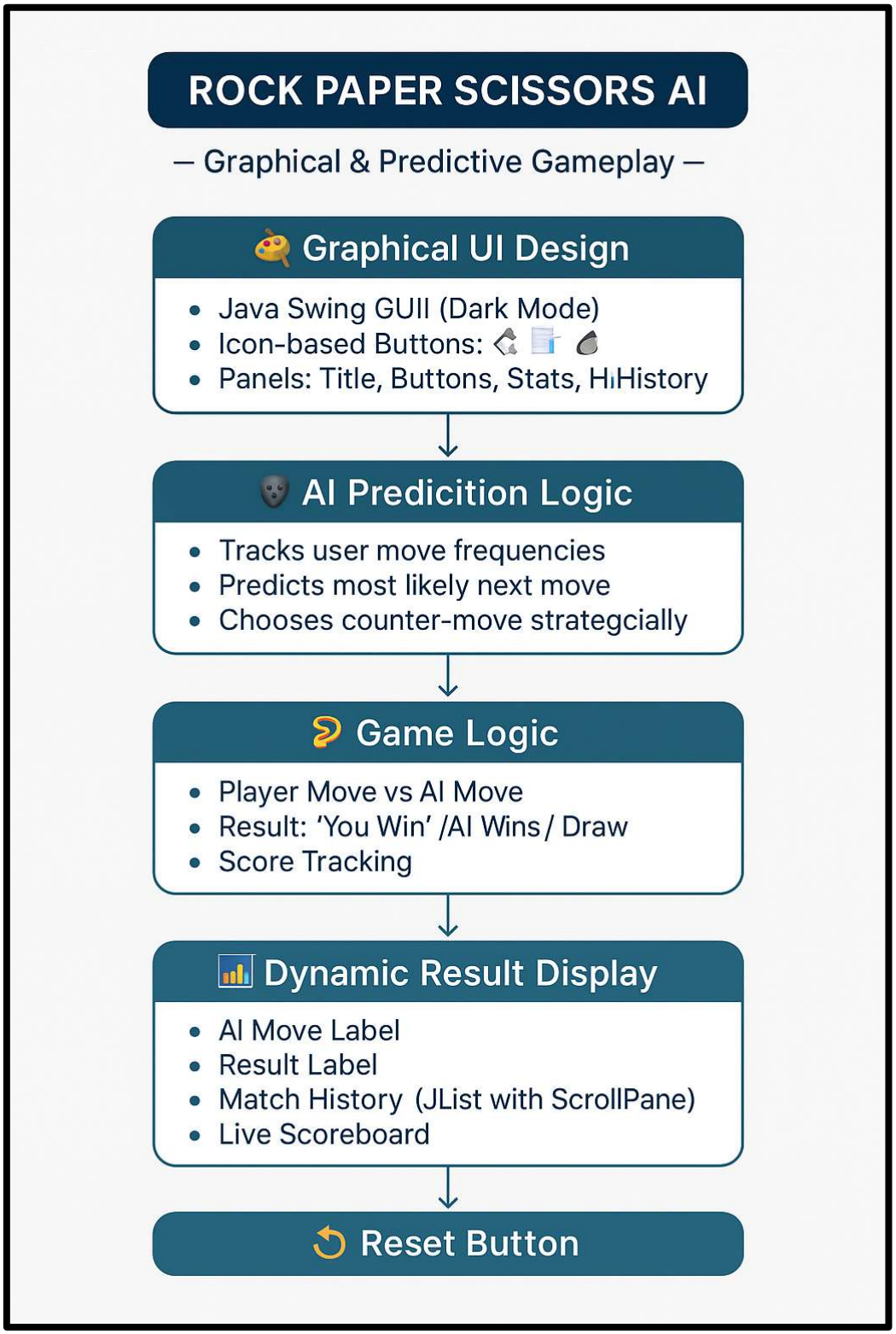
The application's interface has been carefully crafted using Java Swing components arranged through various layout managers to ensure clarity, accessibility, and responsiveness. It features a dark-themed GUI with high-contrast labels contributing to an enjoyable and modern user experience. The interface includes a structured arrangement of panels: a title header, button panel, result display section, and a dedicated match history sidebar. Each move triggers real-time updates, showcasing the AI's choice, round outcome, and updated statistics of wins, losses, and draws.

Unlike traditional Rock Paper Scissors implementations, where the computer's move is generated randomly, this version introduces a basic form of AI intelligence. The AI keeps track of how often the player chooses each of the three moves. It then predicts the user's next move based on the highest frequency and counters it accordingly. For example, if the user has most often chosen "rock," the AI is more likely to choose "paper" in response. This predictive behavior adds strategic depth and prevents monotonous gameplay, encouraging users to vary their choices and think tactically.

To enhance engagement, the application includes a match history feature using a scrollable list (JList), which logs each round's player move, AI move, and result. Additionally, a scoreboard is dynamically updated after every match. A reset button is provided to allow users to start fresh by clearing the history, resetting all scores, and reinitializing the prediction logic.

This project adheres to key software development principles, including modular coding, event-driven interaction, and logical separation of concerns. Each component — from button styling to AI prediction — is encapsulated within dedicated methods, allowing for easy maintenance and future expansion, such as integrating sound effects, multiplayer functionality, or advanced statistical analysis.

# GRAPHICAL ABSTRACT



# CHAPTER 1.

# INTRODUCTION

## 1.1. Client Identification/Need Identification/Identification of relevant Contemporary issue

The increasing popularity of interactive applications and basic AI-driven systems has created a demand for software that not only entertains but also educates. Rock Paper Scissors, a universally recognized and simple game, offers an effective platform to implement and demonstrate foundational artificial intelligence concepts.it's a problem that someone needs resolution (consultancy problem)

Several studies and education reports have shown that gamified learning enhances cognitive engagement, especially among early programmers and AI enthusiasts. In traditional implementations of the game, the opponent (computer) chooses moves randomly, which lacks realism and challenge. This creates a need for AI-driven models that adapt to player behavior and simulate intelligent response patterns.Relevant contemporary issue documented in reports of some agencies

This project aims to fulfill that need by building a desktop-based Rock Paper Scissors game using Java Swing, with an AI that learns user behavior over time through frequency-based prediction. The goal is to provide an accessible and demonstrative system that introduces basic machine intelligence in an engaging way.

## 1.2. Identification of Problem

Despite the simplicity and widespread recognition of the Rock Paper Scissors game, most computer-based implementations fail to integrate intelligence into gameplay. The majority of available versions rely solely on random move generation, which does not adapt to user patterns or behavior. This results in a non-engaging and static experience that lacks realism or strategy.

The absence of behaviorally adaptive logic in casual games represents a missed opportunity to demonstrate learning AI systems. Furthermore, there is no educational example in many introductory programming resources that uses basic learning algorithms within a graphical game environment. This project addresses this gap.

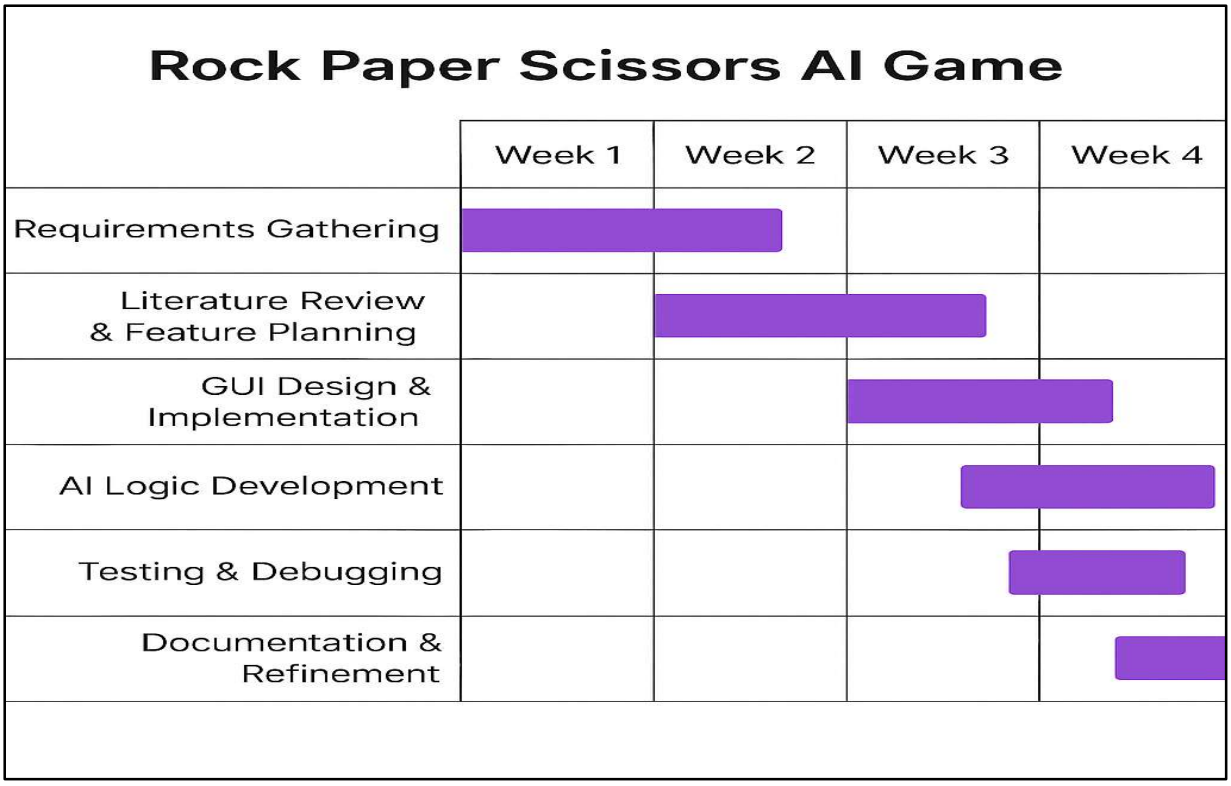
## 1.3. Identification of Tasks

To solve the identified problem and realize the goal of an intelligent and interactive Rock Paper Scissors game, the following tasks were identified:

1. **Requirement Analysis**
   Understand the features necessary for gameplay, user interaction, and AI logic.
2. **GUI Design and Development**
   Create a clean, user-friendly graphical interface using Java Swing.
3. **AI Logic Implementation**
   Develop a frequency-based system to predict and counter user moves intelligently.

4. **Integration of Components**
   Ensure the user interface, game logic, and statistical tracking work seamlessly together.
5. **Testing and Debugging**
   Perform thorough validation of the game flow, prediction accuracy, and UI responsiveness.
6. **Enhancement Features**
   Add user experience features like a reset button, move history, and win/loss counters.

## 1.4. Timeline




## 1.5. Organization of the Report

This report is organized into five chapters as follows:

- **Chapter 1: Introduction**
  Introduces the background, identifies the problem, justifies the need, defines the tasks and timeline, and outlines the structure of the report.
- **Chapter 2: Literature Survey**
  Provides a review of existing solutions, techniques used in similar projects, and lays the foundation for the adopted approach.
- **Chapter 3: Design Flow / Process**
  Discusses the design methodology, specification selection, constraints, and implementation

plan with algorithm and flow diagrams.

- **Chapter 4: Results Analysis and Validation**
  Presents the system outputs, testing methodology, interpretation of results, and performance validation.
- **Chapter 5: Conclusion and Future Work**
  Summarizes the outcomes, highlights deviations, and proposes enhancements and further development directions.

# CHAPTER 2.
# LITERATURE REVIEW/BACKGROUND STUDY

## 2.1. Timeline of the reported problem

The problem of creating adaptive AI in Rock Paper Scissors was recognized as early as the 2000s. Researchers found that human players often follow patterns, making their choices predictable. Over time, various studies explored AI strategies to exploit this predictability. Early versions used randomness, but by the 2010s, statistical and learning models like frequency analysis and Markov chains were applied to improve AI performance in such games. These approaches have since been adopted in AI learning experiments and simple game AI models worldwide.

## 2.2. Proposed solutions

Multiple solutions have been proposed to enhance the AI's intelligence in Rock Paper Scissors:

- **Random Play:** The computer selects moves randomly; this method lacks adaptability.
- **Pattern Detection:** AI looks for repeating user patterns to predict the next move.
- **Markov Chains:** Predicts the next move based on past transitions between moves.
- **Reinforcement Learning:** AI improves its choices based on previous results (more complex).
- **Frequency Analysis (Used Here):** Tracks which move the user plays most and counters it. This method is simple, efficient, and effective for real-time desktop applications.

## 2.3. Bibliometric analysis

A review of literature reveals the following insights:

- **Key Features:** Most approaches use statistical prediction (frequency, pattern recognition, or Markov models) to simulate intelligent gameplay.
- **Effectiveness:** Frequency-based prediction models are lightweight and quick to respond, making them ideal for real-time applications like desktop games.
- **Drawbacks:** More advanced techniques like reinforcement learning require large datasets and training time, which are impractical for small games. Simpler models may struggle with erratic or random user behavior.

## 2.4. Review Summary

The literature review highlights that adaptive AI in simple games improves user engagement. Research supports the idea that users often develop predictable patterns, which AI can exploit. Given the project's scope, using frequency-based prediction aligns well with existing studies and ensures efficient real-time behavior without heavy computation.

## 2.5. Problem Definition

To develop a Rock Paper Scissors desktop game that provides an adaptive AI experience by predicting and countering the user's most frequent move.

- **To Be Done:** Design a GUI in Java, implement a frequency-based AI model, and add gameplay tracking.
- **How:** Use Java Swing for GUI and maintain a move-frequency map to drive AI decisions.
- **Not To Be Done:** Use complex learning models like neural networks or long-term data storage, which exceed project scope.

## 2.6. Goals/Objectives

- Build a Java Swing-based GUI for user interaction.
- Develop an AI opponent using frequency analysis of user input.
- Track and display wins, losses, and draws in real-time.
- Maintain a visual match history list.
- Add a reset function to restart gameplay at any time.
- Ensure the project is simple, lightweight, and understandable for educational purposes.

# CHAPTER 3.
# DESIGN FLOW/PROCESS

## 3.1. Evaluation & Selection of Specifications/Features

The foundation of this project lies in creating an interactive and intelligent version of the Rock-Paper-Scissors game. Based on prior implementations and examples found in educational tools, gaming tutorials, and open-source projects, several recurring features were identified. These include a basic game loop, player-versus-computer mechanics, win/loss/draw tracking, and sometimes a minimalistic graphical interface.

For this solution, a deeper evaluation was conducted to determine which features would best support a rich user experience and educational value. After analyzing the existing patterns and potential enhancements, the following features were shortlisted for inclusion:

- Dark-themed, modern Graphical User Interface (GUI)
- AI prediction logic based on player's historical move frequency (frequency analysis)
- Match history tracking with list view
- Real-time updates of game statistics (win/loss/draw)
- Buttons with icons and styled user interface
- Reset feature to restart the session
- Use of Java's built-in Swing and AWT libraries for cross-platform compatibility

## 3.2. Design Constraints

While the project is software-based and lightweight, certain design constraints still apply. From a regulatory and environmental standpoint, there are no major concerns, as the application does not interact with external systems, databases, or sensitive data.Economically, the design is constrained to utilize only free, open-source tools. Java SE and its Swing library meet this criterion by offering a robust framework without incurring any cost. The development avoids third-party frameworks or dependencies, ensuring maximum portability and ease of deployment.

From a manufacturability perspective, the solution must be easy to replicate and run across different operating systems. Java's cross-platform compatibility ensures that the application functions uniformly on Windows, macOS, and Linux.

Ethical, social, or political considerations are minimal given the non-sensitive nature of the game. However, the AI's design avoids manipulative behavior and ensures fairness. Health-wise, the game interface is designed to be simple, clean, and dark-themed, reducing eye strain, especially during extended use. Finally, the overall design keeps memory and CPU usage low to support smooth operation even on low-end machines

| Constraint Type | Consideration |
|---|---|
| **Regulatory** | No regulatory impact since it's a game. |
| **Economic** | Low-cost: Built with Java SE & Swing, no third-party paid tools used. |
| **Environmental** | N/A – Digital project; no physical footprint. |

| Constraint Type | Consideration |
|---|---|
| **Health/Safety** | GUI ensures ease of use; no physical harm. |
| **Manufacturability** | Easily replicable and extendable in any IDE supporting Java. |
| **Professional/Ethical** | Promotes fair play; includes AI logic but not manipulative. |
| **Social/Political** | N/A; non-sensitive content. |
| **Cost** | Fully open-source and offline-capable; zero operational cost. |

## 3.3. Analysis and Feature finalization subject to constraints

After considering the constraints, an analysis of the proposed features was undertaken. Some features were retained without modification, some were enhanced, while others were removed due to limitations or scope relevance. The AI logic, which was initially planned as a random move generator, was replaced with a frequency-based prediction model to make the gameplay more dynamic and competitive. This adds an educational layer by demonstrating a simple machine learning concept. The GUI was upgraded from plain buttons to styled ones with emoji icons, improving usability and visual appeal. However, features like sound effects and multiplayer capabilities were set aside for now to keep the codebase lightweight and avoid adding unnecessary complexity. Match history and statistics tracking were included as essential components for user engagement. These additions enhance replay value and provide feedback on the player's performance over time. Through this refinement process, the final set of features emerged as both achievable and user-focused, striking a balance between functionality, simplicity, and performance

| Feature | Action Taken (Keep/Modify/Remove) | Reason/Constraint Reference |
|---|---|---|
| Random AI | Removed | Replaced with frequency-based prediction for the challenge. |
| Plain Buttons | Modified | Styled with icons and UI enhancements for better UX. |
| Sound Effects | Removed (Optional) | Not implemented to keep dependencies zero and file size low. |
| Multiplayer Mode | Removed | Not needed for scope; AI vs. player only. |
| Console-only display | Removed | Replaced with a GUI using Swing. |
| Statistics and History | Kept and enhanced | Shown in a sidebar, scrollable and clear. |

| Feature | Action Taken (Keep/Modify/Remove) | Reason/Constraint Reference |
|---|---|---|
| Cross-platform compatibility | Ensured | Java Swing provides OS independence. |

## 3.4. Design Flow

**Design Option 1: Basic Random AI**

- Uses **Math.random()** to generate the AI's move.
- No learning or adaptation — AI behaviour is purely random every round.
- Implements a **very simple GUI** with only three basic buttons: Rock, Paper, and Scissors.
- No visual styling, icons, or coloured themes — minimal user interface.
- Does **not maintain match history** — players cannot view previous outcomes.
- **No statistics panel** — win/loss/draw count is not tracked or displayed.
- Easy to implement — suitable for learning basic Java Swing and game logic.
- Lacks depth, strategy, and long-term user engagement.
- Not suitable for modern gaming experiences where feedback and interactivity matter.

**Design Option 2: Predictive AI with GUI**

- AI uses **frequency analysis** to predict the player's most common move.
- Selects the **counter-move** strategically to improve AI's chance of winning.
- Features a **fully styled GUI** with:
  - ❖ Dark mode background
  - ❖ Modern fonts and color-coded labels
- Includes a **Match History Panel** to display all previous rounds.
- Real-time **Statistics Panel** shows updated counts of wins, losses, and draws.
- Adds a **Reset button** to clear all stats and restart the game smoothly.
- Encourages **strategic gameplay** by adapting to the player's patterns.
- Offers a significantly more **interactive and visually appealing** user experience.
- Designed for scalability — allows easy addition of features like sound, animations, or multiplayer mode.

## 3.5. Design selection

After critically evaluating both options, Design Option 2 was selected. This decision was based on its advanced AI behavior and more engaging user interface. Unlike the first design, which offers a purely random and repetitive gameplay experience, the second design simulates a form of adaptive intelligence. Furthermore, the second design makes the application visually appealing and user-friendly. It enables players to analyze their gameplay patterns via the match history and adjust strategies accordingly. This turns the game into not just entertainment but also an interactive learning experience for understanding logic, GUI design, and simple AI mechanics. The presence of a reset button

allows for experimentation without needing to restart the application. These features collectively make the second design more suitable for the project's goals and audience.

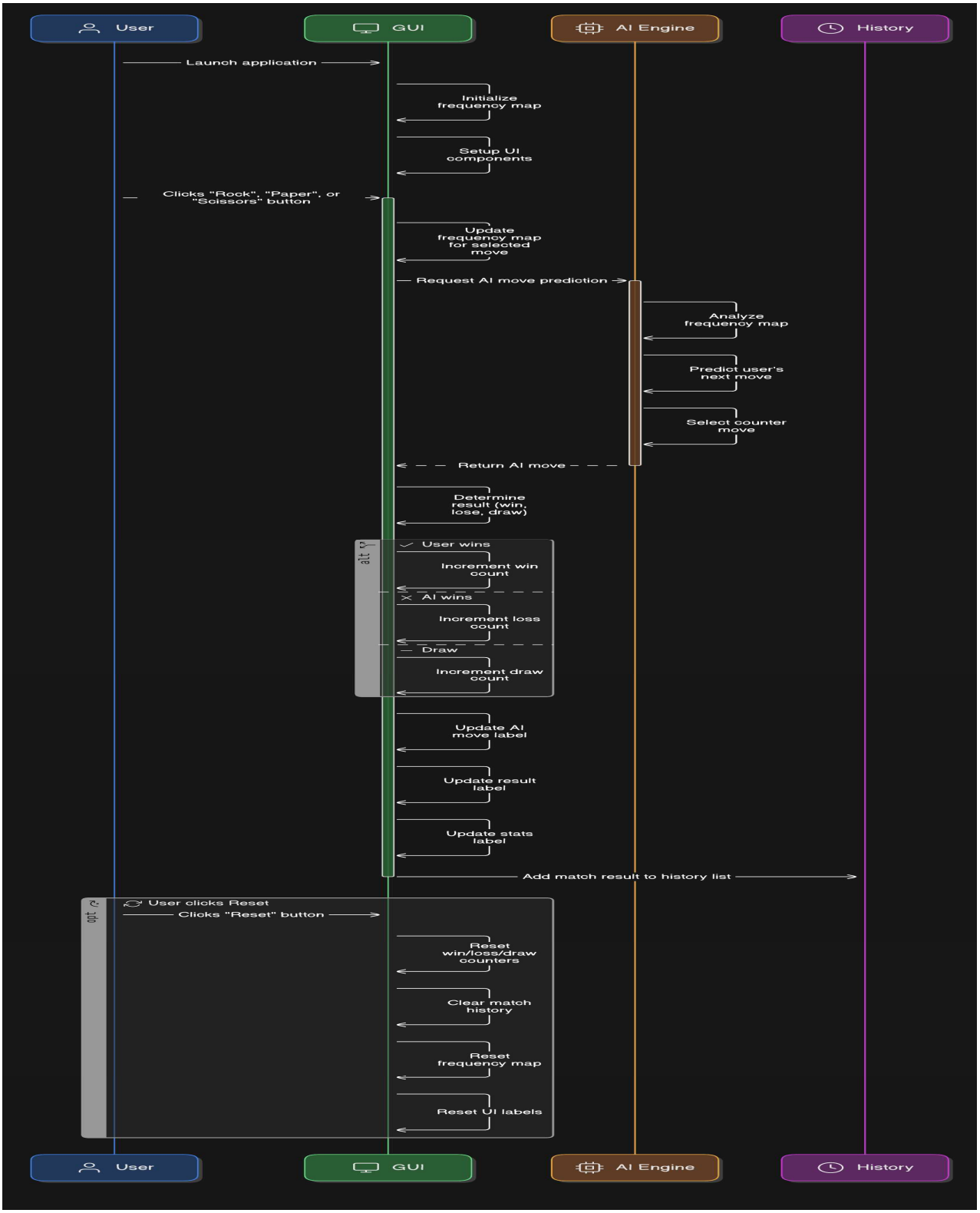
| Criteria | Option 1 (Random AI) | Option 2 (Predictive AI + GUI) |
|---|---|---|
| **AI Intelligence** | Purely random | Predictive logic via frequency analysis |
| **User Interface** | Basic UI | Rich GUI, icon-based, dark mode |
| **User Engagement** | Low | High – history + live stats |
| **Educational Value** | Less logic exposure | Demonstrates pattern prediction |

**Final choice**: Option 2 is chosen for its rich user experience, educational value, and practical AI integration.

## 3.6. Implementation plan/methodology

The implementation of the Rock Paper Scissors AI Game was carried out in a structured and modular manner using Java Swing for the graphical interface. The GUI was designed first, incorporating buttons, labels, and panels arranged using layout managers to ensure a clean and responsive interface. Color themes and icons were added for better user engagement. The core game logic was developed to determine outcomes based on standard rules. An AI component was integrated using **frequency-based prediction**, which analyzes the player's move history to counter likely choices, making the game more intelligent and dynamic. Event listeners were attached to buttons to handle user interactions, and the match history was displayed using a dynamic list model. Statistics for wins, losses, and draws were updated in real time. A reset function was added to restart the game at any time, clearing all data. The code was modularized for clarity and future scalability.

## 3.7. Flowchart



# CHAPTER 4.
# RESULTS ANALYSIS AND VALIDATION

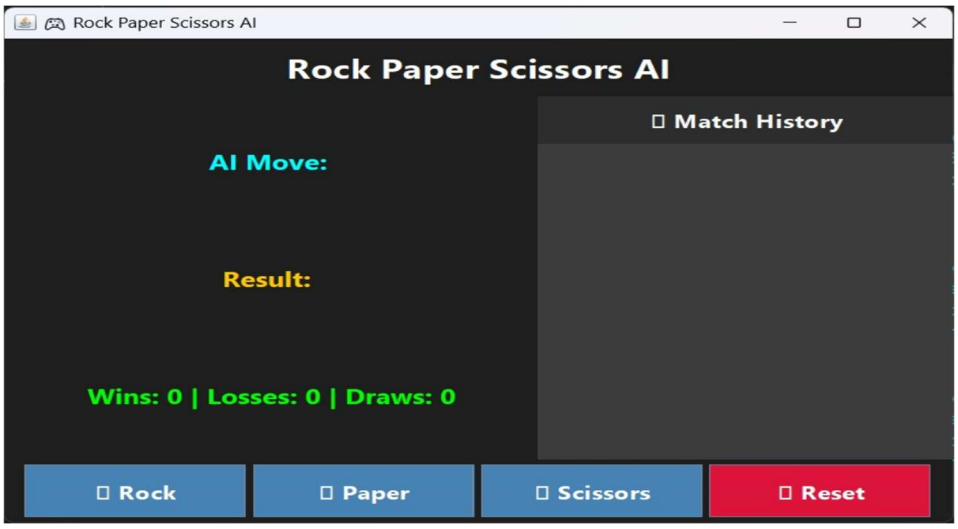
## 4.1. Implementation of solution

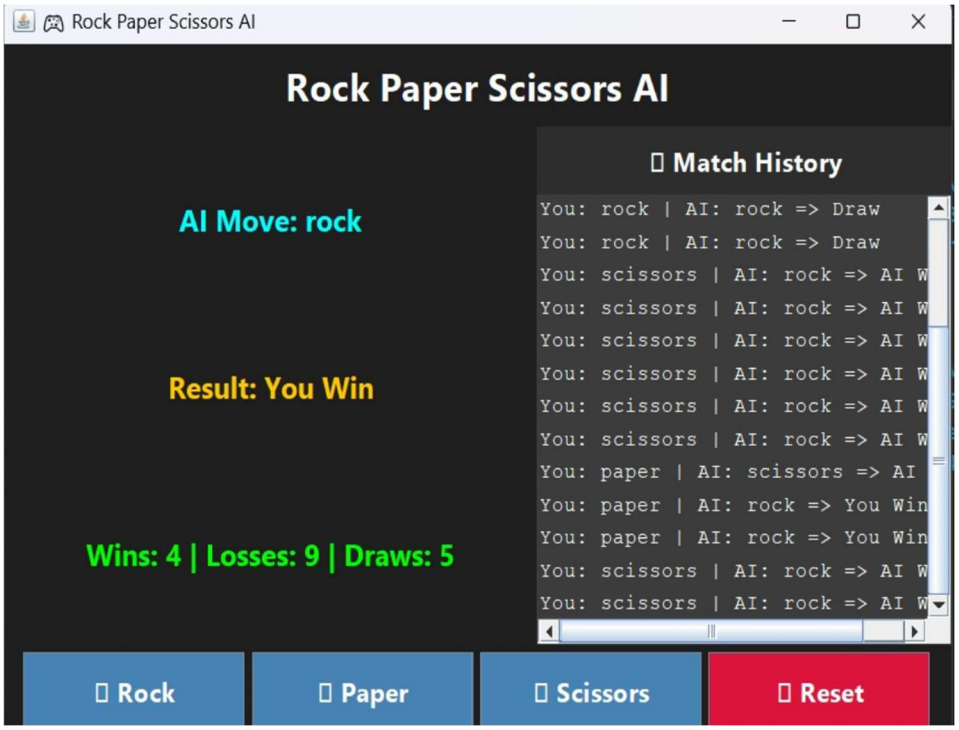
The Rock Paper Scissors AI Game was implemented using a structured development workflow, incorporating modern tools and practices:

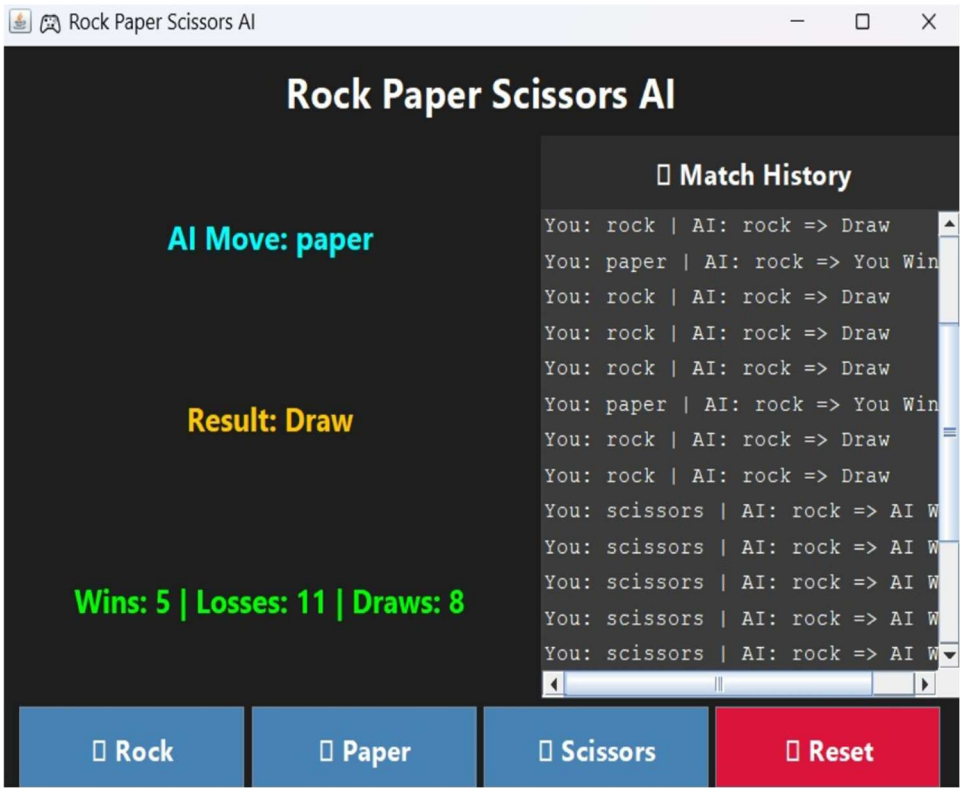
- **Analysis:** Requirements were identified and analyzed to define the system behavior, gameplay logic, and user interaction flow.
- **Design Drawings/Schematics:** The system design was represented using flowcharts and modular structure planning. UI components were laid out using Swing's layout managers (e.g., BorderLayout, FlowLayout).
- **Report Preparation:** Documentation was created using Microsoft Word and LaTeX-based formatting guidelines as per university standards. UML and Gantt chart templates were used where applicable.
- **Project Management and Communication:** Task breakdown was tracked using checklists and version control. Code was managed with Git, ensuring organized development and testing cycles.
- **Testing / Characterization / Interpretation / Data Validation:**
    - Each game feature (button click, AI prediction, stats update) was tested individually (unit testing).
    - The AI was validated by comparing predicted moves against expected user behavior trends.
    - The reset feature was tested to confirm a complete state refresh without restarting the application.
    - UI behavior was tested across different screen sizes to ensure responsiveness and consistency.

The solution was implemented efficiently using **Java SE (JDK 17)** and modern IDEs such as **IntelliJ IDEA and VS CODE** for development and testing.

## 4.2. Testing and Validation




Rock Paper Scissors AI
Rock Paper Scissors AI
Match History
AI Move: rock
You: rock | AI: rock => Draw
You: rock | AI: rock => Draw
You: scissors | AI: rock => AI W
You: scissors | AI: rock => AI W
You: scissors | AI: rock => AI W
You: scissors | AI: rock => AI W
You: scissors | AI: rock => AI W
You: scissors | AI: rock => AI W
You: paper | AI: scissors => AI
You: paper | AI: rock => You Win
You: paper | AI: rock => You Win
You: scissors | AI: rock => AI W
You: scissors | AI: rock => AI W
Result: You Win
Wins: 4 | Losses: 9 | Draws: 5
Rock
Paper
Scissors
Reset


Rock Paper Scissors AI
Rock Paper Scissors AI
Match History
AI Move: paper
You: rock | AI: rock => Draw
You: paper | AI: rock => You Win
You: rock | AI: rock => Draw
You: rock | AI: rock => Draw
You: rock | AI: rock => Draw
You: paper | AI: rock => You Win
You: rock | AI: rock => Draw
You: rock | AI: rock => Draw
You: scissors | AI: rock => AI W
You: scissors | AI: rock => AI W
You: scissors | AI: rock => AI W
You: scissors | AI: rock => AI W
You: scissors | AI: rock => AI W
Result: Draw
Wins: 5 | Losses: 11 | Draws: 8
Rock
Paper
Scissors
Reset

## 4.3. Code Snapshots

```
import java.awt.*;
import java.util.*;
import javax.swing.*;

public class RockPaperScissorsGUI2 extends JFrame {

    private static final String[] MOVES = {"rock", "paper", "scissors"};
    private Map<String, Integer> frequencyMap = new HashMap<>();
    private DefaultListModel<String> historyModel = new DefaultListModel<>();
    private int wins = 0, losses = 0, draws = 0;

    private JLabel aiMoveLabel, resultLabel, statsLabel;
    private JList<String> historyList;

    public RockPaperScissorsGUI2() {
        initializeFrequencyMap();
        setupUI();
    }

    private void setupUI() {
        setTitle("🎮 Rock Paper Scissors AI");
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setSize(700, 500);
        setLocationRelativeTo(null);
        setLayout(new BorderLayout());
        getContentPane().setBackground(new Color(30, 30, 30));

        // Title Label
        JLabel titleLabel = new JLabel("Rock Paper Scissors AI",
SwingConstants.CENTER);
        titleLabel.setFont(new Font("Segoe UI", Font.BOLD, 26));
        titleLabel.setForeground(Color.WHITE);
        titleLabel.setBorder(BorderFactory.createEmptyBorder(10, 0, 10, 0));
        add(titleLabel, BorderLayout.NORTH);

        // Bottom Panel (Buttons)
        JPanel buttonPanel = new JPanel(new FlowLayout());
        buttonPanel.setBackground(new Color(30, 30, 30));

        JButton rockBtn = createStyledButton("✊ Rock", "rock");
        JButton paperBtn = createStyledButton("✋ Paper", "paper");
        JButton scissorsBtn = createStyledButton("✂ Scissors", "scissors");
```

```
        JButton resetBtn = new JButton("🔄 Reset");
        resetBtn.setFont(new Font("Segoe UI", Font.BOLD, 18));
        resetBtn.setBackground(new Color(220, 20, 60));
        resetBtn.setForeground(Color.WHITE);
        resetBtn.setFocusPainted(false);
        resetBtn.setCursor(new Cursor(Cursor.HAND_CURSOR));
        resetBtn.setPreferredSize(new Dimension(160, 50));
        resetBtn.addActionListener(e -> resetGame());

        buttonPanel.add(rockBtn);
        buttonPanel.add(paperBtn);
        buttonPanel.add(scissorsBtn);
        buttonPanel.add(resetBtn);
        add(buttonPanel, BorderLayout.SOUTH);

        // Center Panel (Results)
        JPanel centerPanel = new JPanel(new GridLayout(3, 1, 0, 10));
        centerPanel.setBackground(new Color(30, 30, 30));

        aiMoveLabel = createStyledLabel("AI Move: ", Color.CYAN);
        resultLabel = createStyledLabel("Result: ", Color.ORANGE);
        statsLabel = createStyledLabel("Wins: 0 | Losses: 0 | Draws: 0",
Color.GREEN);

        centerPanel.setBorder(BorderFactory.createEmptyBorder(10, 30, 10, 30));
        centerPanel.add(aiMoveLabel);
        centerPanel.add(resultLabel);
        centerPanel.add(statsLabel);

        add(centerPanel, BorderLayout.CENTER);

        // Right Panel (Match History)
        JPanel rightPanel = new JPanel(new BorderLayout());
        rightPanel.setBackground(new Color(45, 45, 45));
        rightPanel.setPreferredSize(new Dimension(300, 0));

        JLabel historyTitle = new JLabel("🕒 Match History",
SwingConstants.CENTER);
        historyTitle.setFont(new Font("Segoe UI", Font.BOLD, 18));
        historyTitle.setForeground(Color.WHITE);
        historyTitle.setBorder(BorderFactory.createEmptyBorder(10, 0, 10, 0));

        historyList = new JList<>(historyModel);
        historyList.setFont(new Font("Monospaced", Font.PLAIN, 14));
```

```
        historyList.setBackground(new Color(60, 60, 60));
        historyList.setForeground(Color.WHITE);

        JScrollPane scrollPane = new JScrollPane(historyList);
        scrollPane.setBorder(BorderFactory.createLineBorder(Color.DARK_GRAY));

        rightPanel.add(historyTitle, BorderLayout.NORTH);
        rightPanel.add(scrollPane, BorderLayout.CENTER);
        add(rightPanel, BorderLayout.EAST);

        setVisible(true);
    }

    private JButton createStyledButton(String text, String move) {
        JButton btn = new JButton(text);
        btn.setFont(new Font("Segoe UI", Font.BOLD, 18));
        btn.setBackground(new Color(70, 130, 180));
        btn.setForeground(Color.WHITE);
        btn.setFocusPainted(false);
        btn.setCursor(new Cursor(Cursor.HAND_CURSOR));
        btn.setPreferredSize(new Dimension(160, 50));
        btn.addActionListener(e -> play(move));
        return btn;
    }

    private JLabel createStyledLabel(String text, Color color) {
        JLabel label = new JLabel(text, SwingConstants.CENTER);
        label.setFont(new Font("Segoe UI", Font.BOLD, 20));
        label.setForeground(color);
        return label;
    }

    private void play(String userMove) {
        frequencyMap.put(userMove, frequencyMap.get(userMove) + 1);
        String aiMove = getAIMove();
        String result = getResult(userMove, aiMove);

        aiMoveLabel.setText("AI Move: " + aiMove);
        resultLabel.setText("Result: " + result);

        switch (result) {
            case "You Win": wins++; break;
            case "AI Wins": losses++; break;
            case "Draw": draws++; break;
        }
```

```
        statsLabel.setText("Wins: " + wins + " | Losses: " + losses + " | Draws:
" + draws);
        historyModel.add(0, "You: " + userMove + " | AI: " + aiMove + " => " +
result);
    }

    private void resetGame() {
        wins = losses = draws = 0;
        statsLabel.setText("Wins: 0 | Losses: 0 | Draws: 0");
        historyModel.clear();
        initializeFrequencyMap();
        aiMoveLabel.setText("AI Move: ");
        resultLabel.setText("Result: ");
    }

    private void initializeFrequencyMap() {
        frequencyMap.clear();
        for (String move : MOVES) {
            frequencyMap.put(move, 0);
        }
    }

    private String getAIMove() {
        String predicted = "rock";
        int max = -1;
        for (Map.Entry<String, Integer> entry : frequencyMap.entrySet()) {
            if (entry.getValue() > max) {
                max = entry.getValue();
                predicted = entry.getKey();
            }
        }
        return getCounterMove(predicted);
    }

    private String getCounterMove(String move) {
        switch (move) {
            case "rock": return "paper";
            case "paper": return "scissors";
            case "scissors": return "rock";
            default: return "rock";
        }
    }

    private String getResult(String user, String ai) {
```

```
        if (user.equals(ai)) return "Draw";
        if ((user.equals("rock") && ai.equals("scissors")) ||
            (user.equals("paper") && ai.equals("rock")) ||
            (user.equals("scissors") && ai.equals("paper"))) {
            return "You Win";
        }
        return "AI Wins";
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(RockPaperScissorsGUI2::new);
    }
}
```

# CHAPTER 5.
# CONCLUSION AND FUTURE WORK

## 5.1. Conclusion

The Rock Paper Scissors AI Game was successfully implemented using Java Swing and a frequency-based prediction model. The primary objective—to develop an interactive game with basic adaptive AI—was met. The AI could effectively counter the user's most frequent moves, providing a more engaging gameplay experience compared to random selection.

Expected Results:

- AI adapting to user patterns
- Clear tracking of wins, losses, and draws
- Interactive and responsive GUI

Deviations from Expected Results:
In cases where the user played moves randomly or inconsistently, the AI's frequency-based prediction had reduced accuracy. This limitation is inherent to the simplicity of the algorithm and highlights areas for potential improvement.

## 5.2. Future work

To enhance the functionality and learning capabilities of the system, the following improvements are proposed:

- **Advanced AI Techniques:** Incorporate Markov chains or basic reinforcement learning for better move prediction.
- **Data Persistence:** Store historical data across sessions to improve long-term predictions.
- **Multiplayer Mode:** Allow local or network-based two-player games.
- **Difficulty Levels:** Add game modes where the AI uses different strategies (random, frequency-based, learning-based).
- **Mobile/Desktop Deployment:** Convert the application into a cross-platform executable using JavaFX or frameworks like Gluon.

These extensions would make the game more robust, intelligent, and suitable for broader use cases including teaching, gaming, and AI demonstration.

# REFERENCES


1. **GeeksforGeeks.** (n.d.). *Java Swing Tutorial*. Retrieved from https://www.geeksforgeeks.org/java-swing/
   – Used for understanding the basics of Java Swing components and layout managers.
2. **W3Schools.** (n.d.). *Java Tutorial*. Retrieved from https://www.w3schools.com/java/
   – Provided syntax support and examples on Java fundamentals and control structures.
3. **JavaTPoint.** (n.d.). *Java Swing Tutorial*. Retrieved from https://www.javatpoint.com/java-swing
   – Helped in structuring GUI panels and adding event listeners.
4. **Oracle Java Documentation.** (n.d.). *Java Platform SE 8 API Specification*. Retrieved from https://docs.oracle.com/javase/8/docs/api/
   – Official documentation used to understand `javax.swing`, `java.awt`, and event handling classes.
5. **ProgrammingKnowledge.** (2014). *Java GUI Tutorial - Make a GUI in Java*. YouTube. Retrieved from https://www.youtube.com/watch?v=Kmgo00avvEw
   – Used to understand how to build basic Java GUIs using Swing.
6. **Bro Code.** (2020). *Java Rock Paper Scissors Game*. YouTube. Retrieved from https://www.youtube.com/watch?v=ND4fd6yScBM
   – Helped in structuring the basic Rock-Paper-Scissors logic and integrating it into the GUI.
7. **Code With Harry.** (n.d.). *Java Tutorial in Hindi*. YouTube. Retrieved from https://www.youtube.com/playlist?list=PLu0W_9lII9ajLcqRcj4PoEihkukF_OTzA
   – Offered easy-to-understand explanation of Java concepts in Hindi for beginners.

# APPENDIX

## Appendix A: Source Code Snapshot

The following is the primary Java class used to build the Rock Paper Scissors AI game with GUI and predictive AI logic.

- File Name: RockPaperScissorsGUI2.java
- Language: Java SE (Swing GUI)

**Key Components Included:**

- JFrame-based window layout with dark theme.
- Styled JButtons for move selection.
- JLabel for displaying results, AI moves, and statistics.
- JList and DefaultListModel for match history tracking.
- Frequency-based AI decision-making algorithm.
- Reset mechanism to clear history and scores.

## Appendix B: Tools & Technologies Used

| Technology/Tool | Purpose |
|---|---|
| Java SE (JDK 17+) | Core programming language and development kit |
| Java Swing | GUI creation and layout |
| IntelliJ IDEA / Eclipse | IDE used for Java GUI development |
| Java AWT | For layout management and UI component handling |
| Operating System | Windows 10/11 (tested), cross-platform support |

## Appendix C: AI Prediction Logic (Pseudocode)

**Initialize frequencyMap for "rock", "paper", "scissors" with 0**
**On each user move:**
**Increment frequencyMap[userMove]**
**Predict most frequent user move**
**Choose AI move that beats the predicted move**
**Determine round result**
**Update stats and history list**

## Appendix D: GUI Layout Flow

**Main Window:**

- North: Rock Paper Scissors Using AI(TITLE)
- South: Move Buttons + Reset Button
- Center: AI Move, Result, and Statistics Labels
- East: Match History List (Scrollable)