SKILL DEVELOPMENT

Project Report

DLithe Consultancy Services Pvt. Ltd.

DLithe™
Be Agile..Be in Demand

PROJECT REPORT ASSESSMENT

Student Name: Sukanya B Shankargouda

USN: 2JR23CS108

Assignment: Java

Organization: DLithe Consultancy Services Pvt. Ltd

Supervisor's Name: Archana SM

Submitted to

Signature of Training Supervisor

Date:

Signature of Students

Date

DLithe™
Be Agile..Be in Demand

TABLE OF CONTENTS

INTRODUCTION

The The Simon Says game is a classic memory skill game where players must repeat a sequence of colors or sounds in the correct order. With each successful round, the sequence becomes progressively longer and more challenging.

This Java-based implementation of Simon Says is designed to help students understand core programming concepts such as:

- Event-driven programming
- GUI development using Swing/AWT/JavaFX
- Timers and delays
- Data structures like arrays or ArrayLists
- Handling user interaction and game logic

In this project, the computer generates a random sequence of colored buttons, which light up in a specific order. The player must then replicate the sequence by clicking the correct buttons in the same order. If the player makes a mistake, the game ends. Otherwise, the sequence grows longer, and the game continues.

This project can serve both as an educational tool and an entertaining game, suitable for demonstrating how Java can be used to build interactive applications with graphical interfaces.

DLithe™
Be Agile..Be in Demand

SIMON SAYS GAME REQUIREMENT

Problem statement

The objective of this project is to design and implement a **Simon Says game** using the Java programming language. The game should simulate the behavior of the traditional memory-based game, where the system generates and displays a sequence of visual cues (such as button flashes or colors), and the player must replicate the exact sequence.

The core challenge is to handle the **increasing complexity of the sequence**, manage user interactions, and provide real-time feedback based on the player's input. The game should include the following functionalities:

- ## Objective

- **To develop a fully functional Simon Says game** that challenges the user's memory by displaying an increasingly long sequence of actions (colors, buttons, or sounds).

- **To implement an interactive graphical user interface (GUI)** using Java libraries such as **Swing** or **JavaFX** for user-friendly gameplay.

- **To utilize event-driven programming** to handle user input (button clicks or key presses) and provide real-time feedback.

- **To apply logical structures and control flow** to manage the game's sequence generation, timing, and scoring system.

- **To practice object-oriented programming (OOP) principles**, such as encapsulation, inheritance, and modular design, in organizing the game's component

- ## Functional requirements:

  - **Game Start:**
    - The system shall display a **"Start Game"** button to begin the game.
    - Once started, the game shall generate and display the first sequence step.
  - **Sequence Generation:**
    - The system shall **randomly generate a sequence** of button flashes (colors/sounds).
    - With each round, the sequence shall **increase in length by one**.
  - **Visual and/or Audio Feedback:**
    - The system shall **highlight each button** in the sequence one by one.

- Optionally, each button press may be accompanied by a **sound**.

User Interaction:

- The system shall allow the user to **click buttons** in the same order as shown.
- The system shall **record the user's input** and compare it with the generated sequence.

Input Validation:

- The system shall **validate the user's input** after each click.
- If the user input is correct, the game shall proceed to the next round.
- If the input is incorrect, the game shall end and display a **"Game Over"** message.

Score Tracking:

- The system shall **keep track of the number of rounds completed** (score).
- The score shall be displayed on the screen and updated after each successful round.

Game Over and Restart:

- Upon incorrect input, the system shall **end the game** and show the final score.
- The system shall allow the user to **restart the game** from the beginning.

User Interface (UI):

- The system shall display **colored buttons** representing the game's input.

• Non - Functional requirements

Usability:

- The game should have a **simple and user-friendly interface** that is easy to navigate for users of all ages.
- The buttons and messages should be **clearly visible** and labeled appropriately.

Performance:

- The game should respond to user actions (e.g., button clicks) **instantly without noticeable lag**.
- Sequence generation and playback should be **smooth and well-timed** to ensure a good user experience.

Reliability:

- The game should function correctly without crashing or freezing during normal gameplay.
- Input validation should be consistent and accurate throughout the game.

- **Maintainability:**

  - The code should be **modular and well-organized**, making it easy to understand, update, and debug.

  - Clear comments and documentation should be included to support future maintenance or enhancements

• Error handling

- **Invalid User Input Handling:**

  - The system should detect and handle **incorrect sequence input** by the player. When the user clicks the wrong button (not matching the current sequence), the game should **gracefully end the round**, show a **"Game Over"** message, and **prevent further input** until a new game starts.

• System Requirements

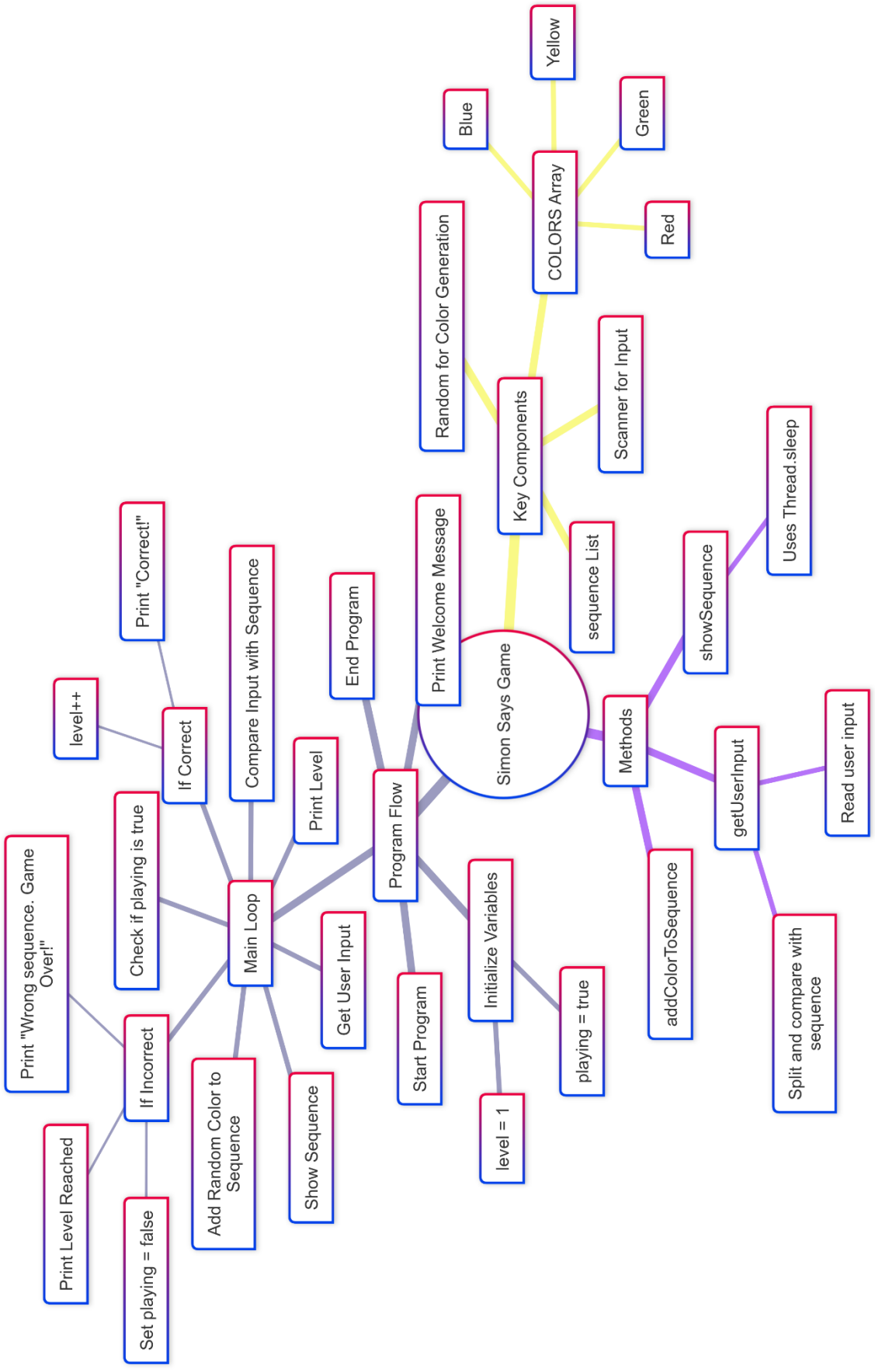
- **Hardware:** Minimum of **2 GB RAM**, **1.0 GHz processor**, and **50 MB free storage**.

- **Operating System: Windows 7**, **macOS 10.9**, or **Linux** with **Java support**.

- **Software: Java Runtime Environment (JRE) 8 or higher** and **Java Development Kit (JDK)** for development.

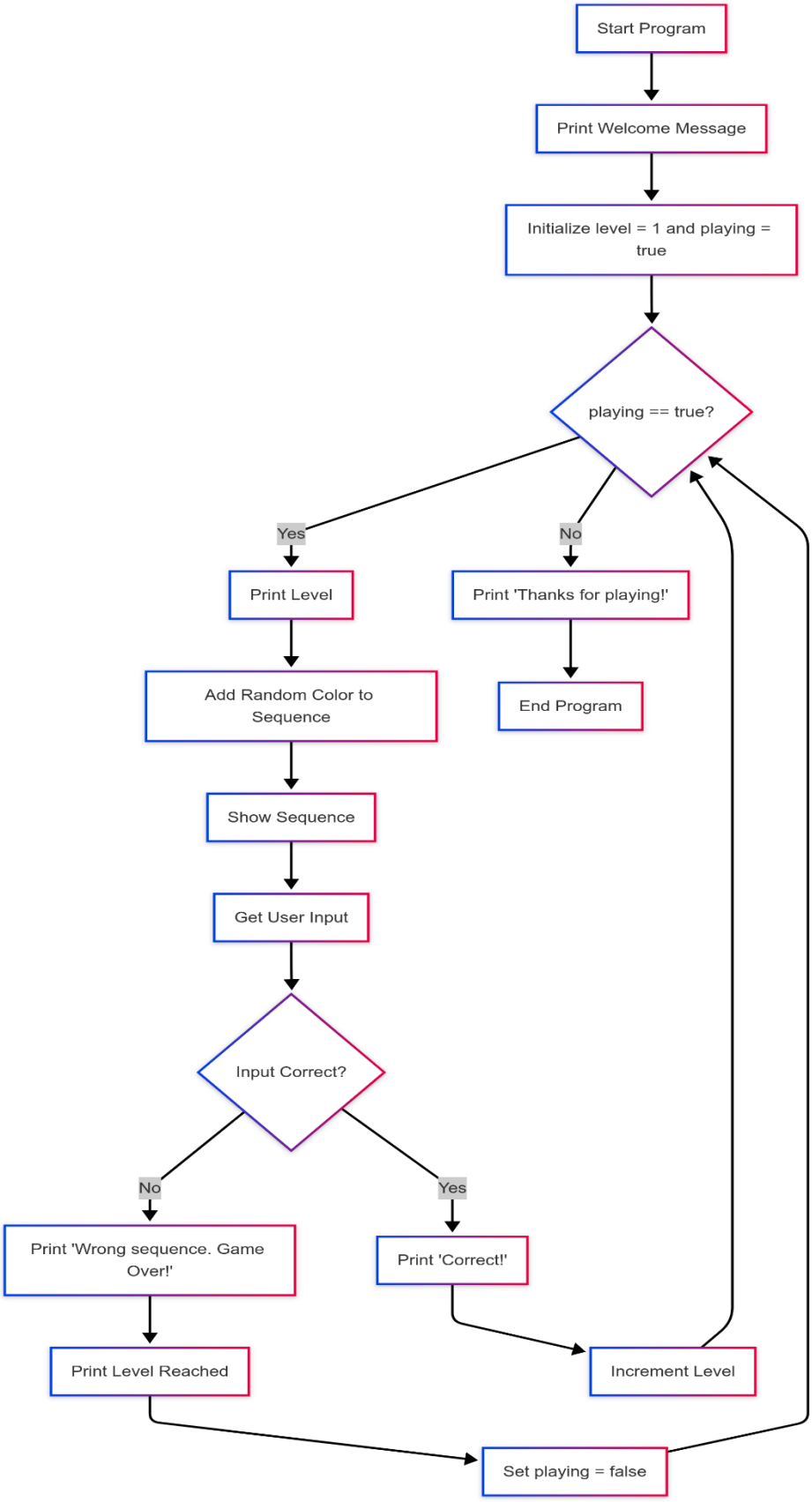
DLithe™
Be Agile..Be in Demand

MIND MAP



DLithe™
Be Agile..Be in Demand

FLOW CHART

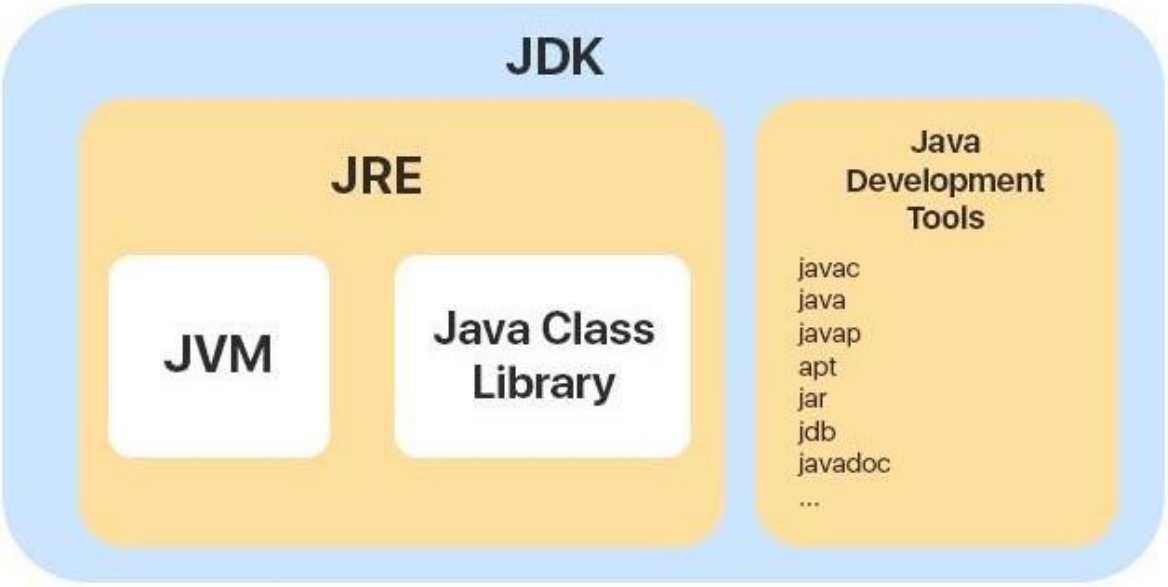


DLithe™
Be Agile..Be in Demand

PROJECT DEVELOPMENT IMAGES

```java
import java.util.*;
public class SimonSays {
    private static final String[] COLORS = {"Red", "Green", "Blue", "Yellow"};
    private static List<String> sequence = new ArrayList<>();
    private static Scanner scanner = new Scanner(System.in);
    private static Random random = new Random();
    Run | Debug
    public static void main(String[] args) {
        System.out.println(x:"*****Welcome to Simon Says!*****");
        System.out.println(x:"Repeat the color sequence. Type exactly as shown (case-sensitive).");
        boolean playing = true;
        int level = 1;
        while (playing) {
            System.out.println("\nLevel " + level);
            addColorToSequence();
            showSequence();

            if (!getUserInput()) {
                System.out.println(x:"Wrong sequence. Game Over!");
                System.out.println("You reached Level " + level);
                playing = false;
            } else {
                System.out.println(x:" Correct!");
                level++;
            }
        }

        System.out.println(x:"Thanks for playing!");
    }

    private static void addColorToSequence() {
        String nextColor = COLORS[random.nextInt(COLORS.length)];
        sequence.add(nextColor);
    }

    private static void showSequence() {
        System.out.print(s:"Simon says: ");
        for (String color : sequence) {
            System.out.print(color + " ");
        }
        System.out.println();
        try {
            Thread.sleep(millis:1000); // simulate time delay
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }
    }

    private static boolean getUserInput() {
        System.out.println(x:"Now, repeat the sequence (space-separated):");
        String input = scanner.nextLine();
        String[] userSequence = input.trim().split(regex:"\\s+");

        if (userSequence.length != sequence.size()) {
            return false;
        }

        for (int i = 0; i < sequence.size(); i++) {
            if (!sequence.get(i).equals(userSequence[i])) {
                return false;
            }
        }

        return true;
    }
}
```

TECHNOLOGIES USED



- **Programming Language:**
  Java: The core language used to implement the game logic. Java is platform-independent and supports object-oriented programming, making it ideal for simple interactive applications.

- **Integrated Development Environment (IDE):**
  Eclipse / IntelliJ IDEA / Visual Studio Code (VS Code): These IDEs were used for writing, editing, debugging, and running the Java code. They provide syntax highlighting, error checking, and code suggestions that improve productivity.

- **Deployment Environment:**
  - Console-based Execution:
    The game is deployed and executed using a command-line interface (CLI). This allows the game to be simple, lightweight, and easy to run on any system with a Java Runtime Environment (JRE).
  - Java Development Kit (JDK):
    A complete environment for Java development, including the compiler (javac) and Java Runtime Environment (JRE) necessary to compile and run the application.

DLithe™
Be Agile..Be in Demand

OUTPUT

```
*****Welcome to Simon Says!*****
Repeat the color sequence. Type exactly as shown (case-sensitive).

Level 1
Simon says: Green
Now, repeat the sequence (space-separated):
Green
 Correct!

Level 2
Simon says: Green Blue
Now, repeat the sequence (space-separated):
Green Blue
 Correct!

Level 3
Simon says: Green Blue Blue
Now, repeat the sequence (space-separated):
Green Blue Blue
 Correct!

Level 4
Simon says: Green Blue Blue Blue
Now, repeat the sequence (space-separated):
Green Blue
Wrong sequence. Game Over!
You reached Level 4
Thanks for playing!
```

DLithe™
Be Agile..Be in Demand

TRAINING EXPERIENCE

As part of my professional development, I actively engaged in an intensive training program focused on Java skill enhancement. The training covered both foundational and advanced concepts of Java programming, including object-oriented programming (OOP), exception handling, multithreading, collections framework, input/output streams, and Java Database Connectivity (JDBC).

The learning experience was highly interactive, involving practical coding sessions, real-world problem-solving exercises, and project-based assessments. I gained hands-on experience in building Java-based applications, understanding design patterns, and implementing efficient and reusable code. Emphasis was placed on writing clean, maintainable code and applying industry-standard best practices.

Through guided mentorship and collaborative projects, I also developed my understanding of software development lifecycles and tools such as Eclipse/IntelliJ, Maven, and Git. The training significantly strengthened my technical abilities and prepared me to take on Java-related development tasks with greater confidence and proficiency.

Overall, the Java skill development training was a transformative learning journey that deepened my understanding of core and advanced Java concepts, while also improving my problem-solving capabilities, code optimization skills, and readiness for real-time application development.

.

DLithe™
Be Agile..Be in Demand

KEY LEARNINGS

Strong Understanding of Core Java Concepts:

Gained in-depth knowledge of Java fundamentals such as data types, control structures, classes, objects, and methods.

Object-Oriented Programming (OOP):

Developed a clear understanding of OOP principles including encapsulation, inheritance, polymorphism, and abstraction, and their implementation in Java.

Exception Handling:

Learned how to write robust programs using try-catch blocks, throw/throws statements, and custom exceptions to manage runtime errors effectively

Collections Framework:

Explored Java's built-in data structures such as ArrayList, HashMap, HashSet, and their use in managing and organizing data efficiently.

File Handling and I/O Streams:

Acquired skills in reading from and writing to files using Java I/O and NIO packages for real-world data manipulation.

Java Database Connectivity (JDBC):

Learned to connect Java applications with databases using JDBC, execute queries, and handle result sets.

Integrated Development Environments (IDEs):

Gained proficiency in using IDEs like Eclipse or IntelliJ IDEA to write, debug, and test Java applications.

Version Control with Git:

Learned the basics of version control systems and how to use Git for collaborative development and code management

Code Optimization and Best Practices:

Improved skills in writing clean, efficient, and maintainable code following industry-standard practices and coding conventions.

Team Collaboration and Communication:

Worked on group projects which enhanced my ability to collaborate, share code, conduct peer reviews, and communicate effectively within a development team.

Problem-Solving and Logical Thinking:

Strengthened my ability to break down complex problems, implement algorithms, and write logic-based solutions using Java.

CHALLENGES

Input Validation:

- E**Sequence Generation and Memory Management:**
- **Challenge:** Ensuring that the sequence of colors/buttons is generated randomly but also stored and retrieved correctly as it increases in length with each round.
- **Solution:** Using arrays or ArrayLists to store the sequence and iterating over them correctly.
- **Timing and Synchronization:**
- **Challenge:** Managing the timing between sequence playback and user input. The sequence must be shown with proper delays, and the game needs to wait for the user's input without lag.
- **Solution:** Proper use of **timers** and **threads** to control timing and synchronize the sequence display with user actions.
- **Input Validation:**
- **Challenge:** Ensuring that the user's input is validated correctly, including detecting when they click a wrong button and handling it without causing the game to crash or become unresponsive.
- **Solution:** Implementing clear error handling for incorrect inputs and providing immediate feedback to the player.
- **User Interface Responsiveness:**
- **Challenge:** Making the game interface responsive and interactive, while handling visual and input changes in real-time.
- **Solution:** Using event listeners properly and ensuring the GUI components (buttons, labels) update smoothly based on user interaction.
- **Game State Management:**
- **Challenge:** Managing different game states (e.g., starting, playing, game over) and making sure the game transitions correctly between them.
- **Solution:** Using state variables to track the game state and implement proper transitions based on player progress.

- **Performance on Low-End Machines:**
- **Challenge:** Ensuring that the game runs smoothly on lower-end computers or systems with limited resources (RAM, processing power).
- **Solution:** Optimizing code and managing system resources effectively, especially during sequence playback and game state transitions

APPLICATIONS

Learning Tool for Beginners

- **Educational Tool for Memory Improvement:**
  - The game can be used as an educational tool to enhance **memory** and **concentration** in children and adults. It provides a fun way to exercise the brain and improve short-term memory by challenging players to remember and replicate increasingly complex sequences.
- **Cognitive Training for Elderly:**
  - This game can serve as a part of **cognitive training** for the elderly, helping to keep their memory sharp and potentially reduce the effects of cognitive decline by engaging them in memory-based exercises.
- **Interactive Entertainment:**
  - As a simple, yet addictive game, **Simon Says** can be used as an entertainment app on various platforms, appealing to casual gamers who enjoy quick, fun challenges.
- **Game Design and Development Learning:**
  - The project can be used as a **learning tool** for new developers to understand core programming concepts, such as **event-driven programming**, **GUI development**, **game logic**, and **error handling**.
- **Game Testing for UI/UX Practices:**
  - The game's design and user interaction mechanics can be used in **user experience (UX)** testing. Developers can test interface layouts, button placement, and user response times to refine and improve interactive designs.
- **Mobile and Web Versions:**
  - The project can be adapted for **mobile** (Android/iOS) or **web-based** platforms (using JavaScript or Java applets) to reach a wider audience and offer a portable gaming experience.
- **Customization for Cognitive Research:**

Researchers in the field of **cognitive science** could use the game to study memory retention, attention span, and other cognitive functions. Variations in difficulty or sequence types can be tailored to gather specific data.

- **Corporate Training Tool:**

  - The game can be adapted into a **corporate training** tool to improve memory recall and focus during training exercises, particularly for employees in fast-paced industries where quick decision-making and memory retention are key.

CONCLUSION

The Simon Says game project has provided an insightful and hands-on learning experience in Java programming, especially in areas such as event-driven programming, GUI development, and game logic implementation. By working through the creation of this memory-based game, we have gained valuable skills in both problem-solving and software development. Throughout the project, we successfully implemented features such as random sequence generation, input validation, and timing mechanisms, while ensuring a smooth user interface. We also learned to address various challenges, including managing the flow of the game, handling user interactions, and ensuring that the game performed well on different systems. This project not only reinforced core Java concepts like Object-Oriented Programming (OOP), data structures, and error handling, but also provided a platform for creativity and logic-based thinking. Additionally, it demonstrated how a simple idea, like a memory game, can be translated into a fully functional application. In conclusion, the Simon Says game has proven to be both an educational tool and an engaging interactive experience. This project has helped improve our technical skills and provided a foundation for future development in game design, Java development, and user interface design.