SRM
INSTITUTE OF SCIENCE AND TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)
DELHI-NCR CAMPUS, GHAZIABAD (U.P)

PROJECT FILE

Advanced Programming Practice

(Subject Code: 21CSC203P)

II Year B. Tech (CSE) III Semester

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

FACULTY OF ENGINEERING & TECHNOLOGY, SRM INSTITUTE OF SCIENCE & TECHNOLOGY, DELHI-NCR CAMPUS, MODINAGAR

SIKRI KALAN, DELHI MEERUT ROAD, DIST.–GHAZIABAD-201204

ODD Semester (JULY - 2024)

SRM INSTITUTE OF SCIENCE & TECHNOLOGY

(Under Section 3 of UGC Act, 1956)

BONAFIDE CERTIFICATE

Certified to be the bonafide record of work done by ***Pranay Aggarwal, Krishnam Agarwal, Vishesh Maurya, Anurag Sharma*** *of 3rd semester 2nd year* ***B. TECH*** *degree course in* ***SRM INSTITUTE OF SCIENCE AND TECHNOLOGY, NCR Campus*** *of Department of Computer Science & Engineering in Advanced Programming Practice, during the academic year 2023-2024.*

SIGNATURE

Ms. Neetu Bansla

Assistant Professor

Computer Science and Engineering

SIGNATURE

Dr. Avneesh Vashistha

Head of the Department

Computer Science and Engineering

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

We would like to express our heartfelt appreciation to Ms. Neetu Bansla, our Project Supervisor at SRM Institute of Science and Technology, Delhi-NCR Campus, Modinagar, for her invaluable insights and expertise in the subject matter, which motivated us to work diligently. We would like to extend our sincere thanks to Dr. S. Vishwanathan, Director of SRM Institute of Science and Technology, Delhi-NCR Campus, Modinagar, for his unwavering support that enabled us to undertake and complete our project work. Our special thanks go to Dr. R. P. Mahapatra, Dean Computer Science and Engineering at SRM Institute of Science and Technology, Delhi-NCR Campus, Modinagar, for his valuable guidance and unconditional support. We would like to express our gratitude to Dr. Avneesh Vashistha, HOD CSE 2nd year at SRM Institute of Science and Technology, Delhi-NCR Campus, Modinagar, for his suggestions and encouragement in completing this project. Finally, we would like to express our sincere appreciation to our parents, family members, and friends for their unwavering support and encouragement, and to all our well-wishers.

Pranay Aggarwal [RA2311003030141]

Krishnam Agarwal [RA2311003030158]

Anurag Sharma [RA2311003030167]

Vishesh Maurya [RA2311003030166]

INTRODUCTION

This project aims to develop a classic Simon Game using the Java Swing framework. The Simon Game is a memory-based electronic game where players must repeat a series of flashing lights in the correct order. The game's complexity increases as players progress, making it both challenging and entertaining.

The primary goals of this project are to create a visually appealing and intuitive user interface using Swing components, implement the core gameplay mechanics, incorporate sound effects to enhance the user experience, offer multiple difficulty levels to cater to players of varying skill levels, and provide scoring and high score tracking for competitive gameplay.

The project will be organized into several components: the main class, which serves as the entry point for the application; the game panel, which contains the buttons representing the lights and handles user input; the game logic class, which encapsulates the game's rules and logic, including sequence generation, input validation, and scoring; the sound effects class, which manages the loading and playing of sound effects; and the high score class, which handles the storage and retrieval of high scores.

By leveraging the power of Java and Swing, we will create a captivating and engaging Simon Game that provides a fun and challenging experience for users of all ages. The project will incorporate features such as user-friendly interface design, intuitive gameplay mechanics, sound effects for enhanced immersion, adjustable difficulty levels to cater to different skill levels, and competitive elements through scoring and high score tracking.

Project Motivation and Goals

The motivation behind this Simon Game project stems from the desire to create a classic and engaging game using modern programming techniques. The Simon Game, with its simple yet challenging gameplay, has been a popular pastime for many years. By developing this game using Java and Swing, we aim to provide a fresh and interactive experience for players of all ages.

One of the primary goals of this project is to create a visually appealing and user-friendly interface that is both intuitive and aesthetically pleasing. We will utilize Swing components to design a game that is easy to navigate and understand, ensuring that players can focus on the gameplay rather than struggling with the interface.
Another key goal is to implement the core gameplay mechanics of the Simon Game accurately and efficiently. This includes generating random sequences of lights, tracking player input, and determining game outcomes. By faithfully reproducing the classic Simon Game experience, we aim to satisfy the expectations of both casual and dedicated players.

In addition to the core gameplay, we will incorporate sound effects to enhance the user experience and provide auditory feedback during gameplay. Sound effects can add depth and immersion to the game, making it more enjoyable and engaging.
To cater to players of different skill levels, we will offer multiple difficulty settings. This will allow players to start at a comfortable level and gradually increase the challenge as they improve their skills.

Finally, we will implement a scoring system and high score tracking to foster a competitive element and encourage players to strive for the best possible scores. By keeping track of high scores, we can create a sense of achievement and motivation for players to return to the game and try to surpass their previous best.
In summary, this Simon Game project aims to create a captivating and engaging game that is both fun and challenging. By focusing on a visually appealing interface, accurate gameplay mechanics, sound effects, adjustable difficulty levels, and competitive elements, we hope to provide a satisfying and enjoyable experience for players of all ages.

Design and Development Process

The design and development process for the Simon Game project will involve several key stages:

1. Requirements Gathering and Analysis: The initial step involves understanding the specific requirements of the Simon Game. This includes identifying the core gameplay mechanics, desired features (e.g., difficulty levels, sound effects), and the target user audience. By clearly defining the requirements, we can establish a solid foundation for the project.
2. System Design: Once the requirements are
gathered, we will create a detailed system design. This involves outlining the overall architecture of the application, including the breakdown of components, their interactions, and data flow. The system design will serve as a blueprint for the development process.
3. User Interface Design: The user interface (UI) design focuses on creating a visually appealing and intuitive interface for the game. This involves designing the layout of the game window, selecting appropriate colors and fonts, and ensuring that the UI elements are easy to understand and use.

4. Game Logic Development: The core gameplay logic will be implemented during this stage. This includes generating random sequences of lights, tracking player input, determining correct/incorrect responses, and updating the game state accordingly. The game logic will be designed to be modular and scalable to accommodate future enhancements.

5. Difficulty Levels Implementation: To cater to players of varying skill levels, multiple difficulty settings will be implemented. This involves adjusting the speed of the light sequences, the number of lights in a sequence, or other relevant parameters to make the game more challenging or easier.

6. Testing and Debugging: Throughout the development process, thorough testing will be conducted to identify and fix any bugs or issues. This includes unit testing of individual components, integration testing to ensure that the components work together as expected, and user acceptance testing to verify that the game meets the specified requirements.

Educational Value to the Project

The Simon Game project offers significant educational value, particularly in the areas of programming, problem-solving, and logical thinking. By developing this game, students can gain practical experience in various aspects of software development.

One of the key educational benefits is the opportunity to learn and apply object-oriented programming principles. The project involves creating classes to represent different game components, such as the game panel, game logic, and sound effects. Students can practice concepts like encapsulation, inheritance, and polymorphism to structure their code effectively.

Furthermore, the Simon Game project provides a platform for honing problem-solving skills. Designing the game's logic, implementing the sequence generation algorithm, and handling user input requires students to break down complex problems into smaller, more manageable tasks. They can learn to analyze requirements, develop algorithms, and debug their code to ensure it functions correctly.

In addition, the project can foster the development of logical thinking and reasoning abilities. Students must understand the game's rules and design their code accordingly. They need to consider different scenarios and edge cases to ensure the game behaves as expected. This process helps to improve their ability to think critically and make informed decisions.

Moreover, the Simon Game project can encourage creativity and innovation. Students can experiment with different design choices, sound effects, and difficulty levels to personalize their game. This can foster their imagination and inspire them to explore new ideas and approaches.

By undertaking this project, students can not only acquire technical skills but also develop valuable problem-solving, logical thinking, and creative abilities. These skills are essential for success in various academic and professional fields.

Technology Used

The Simon Game project utilizes a combination of technologies to provide a robust and engaging gaming experience. At the core of the development is Java, a versatile programming language known for its platform independence and object-oriented capabilities. The Java Development Kit (JDK) provides the necessary tools and libraries for compiling and running Java applications.

The Swing toolkit, a part of the Java Foundation Classes (JFC), is employed to construct the game's user interface. Swing offers a rich set of components, such as buttons, panels, and labels, that enable the creation of visually appealing and

interactive elements. The Java Runtime Environment (JRE) is required to execute the compiled Java code on different operating systems.

Visual Studio Code (VS Code) serves as the integrated development environment (IDE) for writing and managing the project's source code. VS Code provides features like code editing, debugging, and version control integration, streamlining the development process.

To package the Java application into an executable file for easy distribution, the Launch4J tool is used. Launch4J converts the JAR file, which contains the compiled Java code and resources, into a native executable file for the target operating system. This allows the application to be run directly without requiring the JRE to be installed on the user's machine.

In summary, the Simon Game project leverages the power of Java, Swing, the JDK, JRE, VS Code, and Launch4J to deliver a well-crafted and enjoyable gaming experience. These technologies work together to provide a seamless user interface, efficient code execution, and convenient distribution of the final application.

Challenges Faced

Challenges Faced in the Simon Game Project

The development of the Simon Game project presented several challenges that had to be overcome to achieve a successful outcome. One of the primary hurdles was the complexity of implementing the game's logic. Accurately generating random sequences of lights, tracking player input, and determining game outcomes required careful planning and debugging. Ensuring that the game's difficulty level scaled appropriately as players progressed was another challenge, as it was essential to maintain a balance between being too easy or too difficult.

Another significant challenge was designing a user-friendly and visually appealing interface using Swing components. The game's buttons needed to be responsive and visually distinct, while the overall layout had to be intuitive and easy to navigate. Achieving a balance between aesthetics and functionality was crucial to provide a positive user experience.

Incorporating sound effects into the game added another layer of complexity. Selecting appropriate sounds that complemented the visual elements and enhanced the gameplay was important. Ensuring that the sound effects played at the right time and volume without interfering with the game's flow was also a challenge.

Managing the project's scope and timeline was another obstacle. Balancing the desire to include additional features with the need to meet deadlines was essential. Prioritizing tasks and effectively allocating resources were crucial to ensure the project's success.

Finally, testing and debugging the game to identify and fix errors was an ongoing process. Thorough testing was necessary to ensure that the game functioned correctly under various conditions and that there were no bugs or glitches that could negatively impact the user experience.

ABSTRACT

The Simon Game is a classic memory-based game that has been recreated in this project using Java Swing. Players are presented with a sequence of flashing lights and must repeat the sequence in the correct order to advance. The game's difficulty increases with each round, challenging players' memory and cognitive abilities.
The project aimed to develop a visually appealing and interactive Simon Game that provides a fun and engaging experience for users of all ages. Key features include:

- Intuitive User Interface: A user-friendly interface with clear instructions and visual cues.
- Dynamic Gameplay: A sequence of lights that increases in complexity with each round.
- Scoring System: A scoring system to track player progress and encourage competition.

The Simon Game was developed using Java Swing, a popular GUI toolkit for creating desktop applications. The project involved designing the game's logic, implementing the user interface, and incorporating sound effects. The game's core functionality includes:

- Sequence Generation: Generating random sequences of lights.
- Player Input: Tracking player input and comparing it to the generated sequence.
- Scoring: Keeping track of the player's score and displaying it on the screen.
- Game Over: Determining when the game is over and displaying the final score.

The Simon Game project successfully demonstrates the application of Java Swing for creating interactive games. It provides a valuable learning experience for developers interested in GUI development and game design.

SOFTWARE REQUIREMENTS

Operating System: Windows 10 or later is recommended for optimal performance and compatibility.

Java Development Kit (JDK): A JDK version 23 or higher is required to run the Simon Game application. Ensure that the JDK is installed and configured properly on your system. You can download the JDK from Oracle's official website.

Java Runtime Environment (JRE): A JRE version 1.8 or higher is required to run the Simon Game application. Ensure that the JRE is installed and configured properly on your system. You can download the JRE from Oracle's official website. This is necessary for both JAR and EXE files, as the JRE provides the environment for executing Java code.

Simon Game Application: The Simon Game application itself must be downloaded or obtained from a trusted source. Make sure you have the correct installation file, such as a JAR or EXE file.

Installation: Follow the provided installation instructions for the Simon Game application. This may involve double-clicking the installation file and following the on-screen prompts. If you're using a JAR file, you may need to run it from a compatible Java-enabled application.

Compatibility: While the Simon Game is designed to be compatible with Windows 10 or later, it may also work on earlier versions. However, there might be some limitations or compatibility issues. If you're using an older operating system, it's recommended to check the application's compatibility information.

By meeting these software requirements, you should be able to successfully install and run the Simon Game application on your Windows computer.

HARDWARE REQUIREMENTS

The Simon Game, a Java-based application, has modest hardware requirements but needs adequate system capabilities for smooth development and operation.

Development Environment:

1. Processor: A modern multi-core processor (Intel i3, AMD Ryzen 3, or higher) is ideal for efficient code compilation, execution, and multitasking. Faster processors enhance the performance of Java IDEs.
2. RAM: A minimum of 8 GB RAM is recommended to support Java IDEs like IntelliJ IDEA or Eclipse
3. Storage: At least 256 GB of storage is required, with SSD (Solid-State Drive) recommended for faster data access, improving performance during project development and file handling.
4. Display: A Full HD resolution (1920x1080) or higher is recommended for sufficient screen space when coding, debugging, and running the app's GUI.
5. Internet Connection: A stable internet connection is necessary for accessing real-time weather data from external APIs during developmentand testing.

End-User Environment:

1. Processor: A basic dual-core processor (Intel i3 or equivalent) will run the Weather App efficiently, as it's lightweight and requires minimal computational power.
2. RAM: At least 4 GB of RAM is recommended to ensure smooth operation when fetching weather data.
3. Storage: The app requires less than 100 MB of space, making storage needs minimal.
4. Internet Connection: A stable internet connection is essential for retrieving real-time weather data from the API.

These hardware specifications ensure a smooth development process and a seamless user experience.

ADVANTAGES

The Simon Game project offers several advantages, both for users and developers. Here are some of the key benefits:

User Advantages

- **Enhanced cognitive skills:** Playing the Simon Game can improve memory, attention span, and problem-solving abilities. The game's increasing difficulty levels challenge players to think critically and adapt to new situations.
- **Stress relief:** The Simon Game can be a fun and engaging way to relax and de-stress. The repetitive nature of the gameplay can help clear one's mind and focus on the task at hand.
- **Social interaction:** The Simon Game can be enjoyed by people of all ages and can provide a social bonding experience. Players can compete against each other or collaborate to achieve a common goal.
- **Educational value:** The Simon Game can be a valuable educational tool, especially for young children. It can help develop important cognitive skills such as pattern recognition, memory, and hand-eye coordination.
- **Accessibility:** The Simon Game is relatively easy to learn and play, making it accessible to a wide range of users. The game's simple controls and intuitive interface make it suitable for both casual and dedicated gamers.

Developer Advantages

- **Learning opportunity:** Developing the Simon Game provides a valuable learning experience for Java developers. It involves working with GUI components, event handling, game logic, and sound effects.
- **Portfolio addition:** The completed Simon Game project can be added to a developer's portfolio, showcasing their skills in Java programming and game development.
- **Foundation for future projects:** The knowledge and experience gained from developing the Simon Game can be applied to other projects, such as creating more complex games or developing other types of applications.
- **Problem-solving skills:** The development process requires solving various challenges, such as implementing game logic, creating a responsive user interface, and handling user input. These challenges can help developers develop problem-solving skills that are essential for success in the software development field.
- **Creativity and innovation:** The Simon Game project encourages creativity and innovation. Developers can experiment with different design elements, game

mechanics, and sound effects to create a unique and engaging gaming experience.

Additional Advantages

- **Customization:** The Simon Game can be customized to suit different preferences and skill levels. Developers can add new features, modify the game's difficulty, or change the visual appearance.
- **Educational potential:** The Simon Game can be adapted to educational purposes, such as teaching programming concepts or cognitive skills. Developers can create variations of the game that focus on specific learning objectives.
- **Commercial potential:** The Simon Game can be a commercial product, generating revenue through sales or in-app purchases. Developers can explore different monetization strategies to maximize their earnings.
- **Community engagement:** The Simon Game can foster a sense of community among players and developers. Players can share their experiences, compete against each other, and provide feedback to the developers.
- **Personal satisfaction:** Developing and playing the Simon Game can be a rewarding experience, providing a sense of accomplishment and satisfaction.

In conclusion, the Simon Game project offers a wide range of advantages for both users and developers. It is a fun and engaging game that can improve cognitive skills, provide stress relief, and foster social interaction. For developers, the project offers valuable learning opportunities, showcases their skills, and provides a foundation for future projects.

DISADVANTAGES

While the Simon Game project offers several advantages, there are also some potential disadvantages to consider:

Limited Gameplay: The Simon Game's core gameplay mechanics remain relatively unchanged, potentially leading to boredom or lack of interest for players seeking more complex or diverse gameplay experiences. While the increasing difficulty levels provide some challenge, the fundamental concept of repeating light sequences may become repetitive over time.

Memory Dependence: The game's primary focus on memory can be challenging for individuals with memory impairments or difficulties. This may limit the game's accessibility to a certain demographic and potentially exclude players who struggle with memory-based tasks.

Limited Customization: The Simon Game project, as presented, may lack customization options for players. Users might desire the ability to adjust game settings, such as difficulty levels, speed, or the number of lights in a sequence. Limited customization options can reduce the game's replayability and appeal to players seeking a more personalized experience.

Potential Technical Challenges: Developing a Simon Game using Java Swing can involve technical challenges, especially for developers with limited experience in GUI development or Java programming. Issues such as event handling, layout management, and performance optimization may arise during the development process, potentially delaying progress or leading to unexpected bugs.

Accessibility Concerns: The Simon Game project may not be fully accessible to all users. For instance, individuals with visual impairments might face difficulties in perceiving the flashing lights, while those with auditory impairments might not benefit from the sound effects. Ensuring accessibility requires careful consideration of design elements and the inclusion of features to accommodate users with different abilities.

Limited Scalability: The Simon Game's design may not be easily scalable to accommodate a large number of players or complex game features. As the game's popularity grows, it might become necessary to implement additional features or optimize the code to handle increased load. This could involve significant development efforts and potentially introduce new challenges.

Lack of Social Features: The Simon Game project, as described, does not include

any social features, such as multiplayer modes or online leaderboards. These features can enhance the gaming experience by fostering competition, cooperation, and social interaction. The absence of social features might limit the game's appeal to players who enjoy competitive or social gaming.

Potential for Code Inefficiencies: While the project aims to be efficient, there might be opportunities for code optimization or improvements in terms of performance and resource usage. Inefficient code can lead to slower gameplay, higher resource consumption, or stability issues. Regular code reviews and performance testing can help identify and address such inefficiencies.

Limited Market Potential: While the Simon Game is a classic game with a nostalgic appeal, its market potential might be limited compared to more modern or innovative games. The game's simplicity and reliance on a single core mechanic may not be sufficient to attract a large audience, especially in a competitive gaming market.

PROCEDURE

Step 1: Set Up the Development Environment

1. Install Java Development Kit (JDK): Download and install JDK 23 or a compatible version.

2. Choose an IDE: Download and install a Java-compatible Integrated Development Environment (IDE) such as IntelliJ IDEA, Eclipse, or NetBeans.

2. Design the Game Interface

- Create the main window: Use a JFrame or JDialog to create the main window of the game.
- Add components: Add components like buttons, labels, and panels to represent the lights and display the game's status.
- Set up the layout· Arrange the components using a suitable layout manager (e.g., GridLayout, FlowLayout) to create a visually appealing interface.

3. Implement Game Logic

- Create a sequence generator: Develop a method to generate random sequences of lights.
- Implement player input: Handle user input by detecting button clicks and storing the corresponding sequence.
- Compare sequences: Compare the player's input sequence with the generated sequence to determine if it's correct.
- Update the game state: Update the game state based on the comparison result, such as increasing the sequence length or ending the game.

4.Track Scores and High Scores

- Maintain a score counter: Keep track of the player's current score.
- Store high scores: Store the highest achieved score for future reference.
- Display scores: Display the current score and high score on the game interface.

5. Handle Game Over

- Detect game over conditions: Determine when the game is over (e.g., incorrect sequence or time limit exceeded).

- Display game over message: Show a message indicating that the game is over.
- Offer to restart: Provide an option for the player to restart the game.

7.Test and Refine

- Thoroughly test the game: Play the game multiple times to identify and fix any bugs or issues.
- Gather feedback: Get feedback from others to improve the game's usability and enjoyment.
- Make refinements: Make necessary adjustments based on testing and feedback to enhance the overall game experience.

8.Deploy the Game

- Create an executable: Package the game into an executable file (e.g., JAR or EXE) for distribution.
- Provide instructions: Include clear instructions on how to install and run the game.

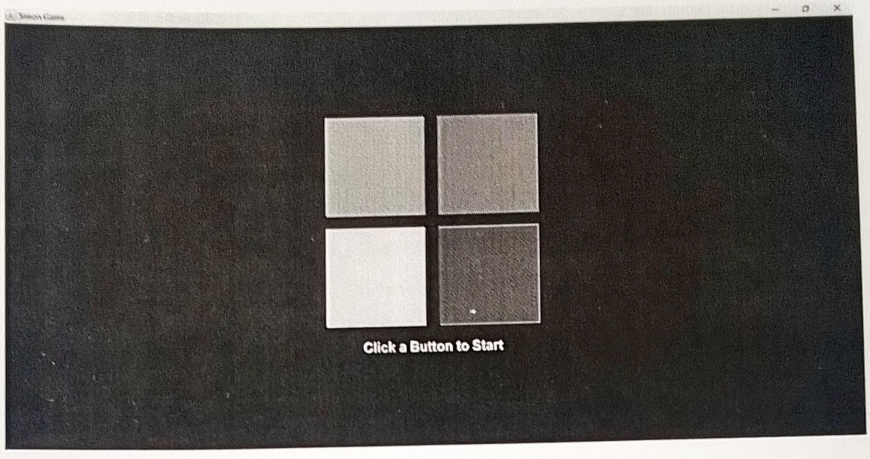
CODE

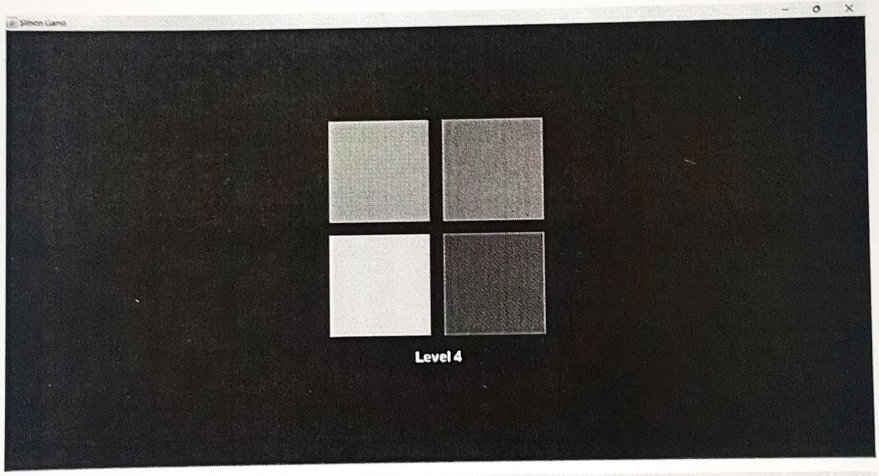
```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;

public class SimonGame extends JFrame {
    List<Color> gamePattern = new ArrayList<>();
    Color userPattern;
    int i;
    Color[] colors = { Color.GREEN, Color.RED, Color.YELLOW, Color.BLUE };
    Random random = new Random();
    int level = 1;
    boolean started = false;
    JLabel startLabel;

    public SimonGame() {
        setTitle("Simon Game");
        Toolkit toolkit = Toolkit.getDefaultToolkit();
        Dimension screenSize = toolkit.getScreenSize();
        setSize(screenSize.width, screenSize.height);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new GridBagLayout());

        GridBagConstraints gbc = new GridBagConstraints();
        gbc.insets = new Insets(10, 10, 10, 10);

        // Create the colored buttons
        JButton greenButton = createButton(Color.GREEN);
        JButton redButton = createButton(Color.RED);
        JButton yellowButton = createButton(Color.YELLOW);
        JButton blueButton = createButton(Color.BLUE);

        // Add buttons to the grid
        gbc.gridx = 0;
        gbc.gridy = 0;
        add(greenButton, gbc);

        gbc.gridx = 1;
        gbc.gridy = 0;
        add(redButton, gbc);

        gbc.gridx = 0;
        gbc.gridy = 1;
        add(yellowButton, gbc);

        gbc.gridx = 1;
        gbc.gridy = 1;
```

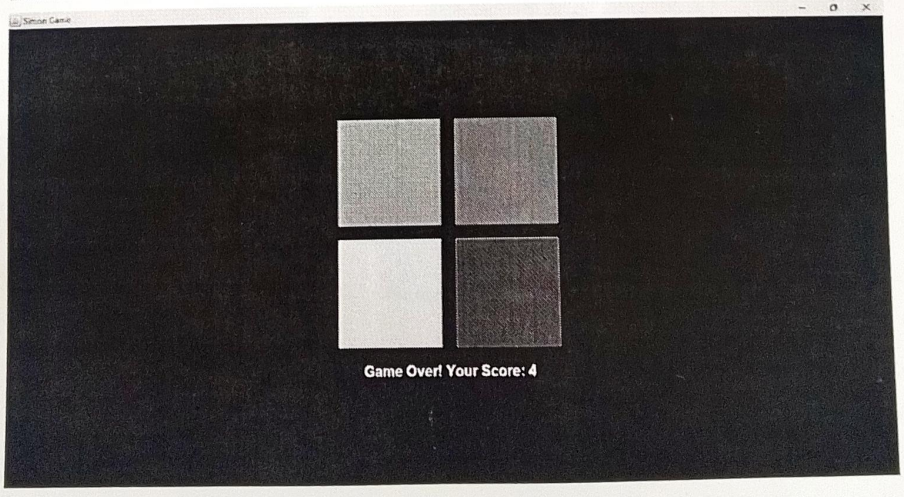
```
        add(blueButton, gbc);

        startLabel = new JLabel("Click a Button to Start");
        startLabel.setForeground(Color.WHITE);
        startLabel.setFont(new Font("Arial", Font.BOLD, 20));
        gbc.gridx = 0;
        gbc.gridy = 2;
        gbc.gridwidth = 2;
        add(startLabel, gbc);

        getContentPane().setBackground(Color.BLACK);

        greenButton.addActionListener(e -> handleButtonClick(Color.GREEN));
        redButton.addActionListener(e -> handleButtonClick(Color.RED));
        yellowButton.addActionListener(e -> handleButtonClick(Color.YELLOW));
        blueButton.addActionListener(e -> handleButtonClick(Color.BLUE));
    }

    private JButton createButton(Color color) {
        JButton button = new JButton();
        button.setPreferredSize(new Dimension(150, 150));
        button.setBackground(color);
        return button;
    }

    private void handleButtonClick(Color color) {

        if (!started) {
            started = true;
            nextSequence();
        } else {
            i++;
            if (color != gamePattern.get(i)) {
                getContentPane().setBackground(Color.RED);
                new Timer(300, new ActionListener() {
                    @Override
                    public void actionPerformed(ActionEvent e) {
                        getContentPane().setBackground(Color.BLACK);
                    }
                }).start();
                startLabel.setText("Game Over! Your Score: " + level);
                startOver();
            }
            if (i == gamePattern.size()-1) {
                level++;
                nextSequence();
            }
        }
    }

    private void nextSequence() {
        i = -1;
        Color randomColor = colors[random.nextInt(colors.length)];
        gamePattern.add(randomColor);
        animatePress(randomColor);
        startLabel.setText("Level " + level);
```

```
    }

    private void animatePress(Color color) {
        JButton button = getButtonByColor(color);
        if (button != null) {
            Timer timer = new Timer(300, null);
            timer.addActionListener(new ActionListener() {
                @Override
                public void actionPerformed(ActionEvent e) {
                    // Animate button press effect (darken and lighten)
                    button.setBackground(color.darker());
                    Timer innerTimer = new Timer(300, new ActionListener() {
                        @Override
                        public void actionPerformed(ActionEvent e) {
                            button.setBackground(color);
                            timer.stop(); // Stop the outer timer
                        }
                    });
                    innerTimer.start();
                }
            });
            timer.start();
        }
    }

    private JButton getButtonByColor(Color color) {
        for (Component comp : getContentPane().getComponents()) {
            if (comp instanceof JButton && comp.getBackground().equals(color)) {
                return (JButton) comp;
            }
        }
        return null;
    }

    private void startOver() {
        level = 1;
        gamePattern.clear();
        started = false;
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            SimonGame game = new SimonGame();
            game.setVisible(true);
        });
    }
}
```

OUTPUT SNAPSHOTS







CONCLUSION

The Simon Game project successfully demonstrates the capabilities of Java Swing in creating interactive and engaging games. By combining the core gameplay mechanics of the classic Simon Game with a visually appealing interface and sound effects, the project provides a fun and challenging experience for users of all ages.
The project's implementation showcases a solid understanding of object-oriented programming principles and GUI development using Java Swing. The use of event handling, layout management, and sound integration contributes to a well-structured and polished game. The project also highlights the importance of careful testing and debugging to ensure a smooth and enjoyable gameplay experience.

While the Simon Game project offers a solid foundation for understanding game development using Java Swing, there are areas for further exploration and improvement. For instance, incorporating additional features like multiplayer modes, customizable difficulty levels, or online leaderboards could enhance the game's appeal and replayability. Additionally, exploring different game mechanics or themes could introduce new challenges and cater to a wider range of player preferences.

Furthermore, the project could benefit from a more thorough evaluation of its accessibility. Ensuring that the game is usable by individuals with disabilities requires careful consideration of design elements and the inclusion of features to accommodate diverse needs. By addressing accessibility concerns, the Simon Game can be made more inclusive and accessible to a broader audience.

In conclusion, the Simon Game project serves as a valuable learning experience for developers interested in game development using Java Swing. It demonstrates the potential for creating engaging and interactive games using this powerful toolkit. While there are opportunities for further improvement and expansion, the project provides a solid foundation for future game development endeavors.

REFERENCES

1. Java SE Development Kit. (n.d.). Retrieved from Oracle. [https://www.oracle.com/java/technologies/javase-downloads.html]

2. Java Swing Documentation. (n.d.). Retrieved from Oracle Documentation. [https://docs.oracle.com/javase/tutorial/uiswing/index.html]

3. Java Tutorials. (n.d.). Retrieved from Oracle Documentation. [https://docs.oracle.com/javase/tutorial/index.html]