**Application Walkthrough: Creating a Simon Says Game**

**Step 1: Project Setup**

1.1. Create a new folder for your project, and give it a descriptive name like "SimonSaysGame."

1.2. Inside the project folder, create the following files: index.html, styles.css, game.js.

1.3. Download any necessary sound files (e.g., red.mp3, blue.mp3, etc.) and place them in a folder named "sounds" within your project directory.

**Step 2: HTML Structure**

2.1. Open index.html in your code editor.

2.2. Create the basic HTML structure by adding the following elements:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Simon Says</title>

<link rel="stylesheet" href="styles.css">

</head>

<body>

<!-- Add game elements here -->

<script src="https://code.jquery.com/jquery-3.7.1.min.js" integrity="sha256-/JqT3SQfawRcv/BIHPThkBvs0OEvtFFmqPF/lYI/Cxo=" crossorigin="anonymous" defer></script>

<script src="game.js" defer></script>

</body>

</html>

**Step 3: CSS Styling**

3.1. Open styles.css in your code editor.

3.2. Define the basic styling for the game elements, including the game board, buttons, and any customizations you desire. Use CSS classes to select and style the elements.

**Step 4: JavaScript Logic**

4.1. Open game.js in your code editor.

4.2. Define and initialize variables for the game, such as buttonColours, gamePattern, userClickedPattern, started, and level.

4.3. Create a function, nextSequence(), to generate a random color pattern for Simon and display it to the user. Use Math.random() and jQuery animations for this.

4.4. Implement event listeners to handle user input when buttons are clicked. When a button is clicked, play the corresponding sound and add the color to the userClickedPattern array.

4.5. Create a function, checkAnswer(currentLevel), to compare the user's pattern to Simon's pattern. Check if the user got the sequence right and proceed to the next level if they did. If the user makes a mistake, trigger the game over scenario.

4.6. Implement functions for playing sounds, animating button presses, and restarting the game.

4.7. Add a keypress event listener to start the game when a key is pressed.

**Step 5: Debugging and Testing**

5.1. Use console.log() statements strategically to debug your code. Print important variables and values to the console to identify issues.

5.2. Test the game at various stages of development. Verify that button clicks, animations, and sounds work as expected.

**Step 6: Customization (Optional)**

6.1. Allow customization of game elements such as colors, fonts, and sounds by modifying the CSS and adding more sound files to the "sounds" folder.

**Step 7: Final Touches and Optimization**

7.1. Optimize the game for performance, ensuring smooth animations and responsiveness on different devices.

7.2. Add any final touches, such as transitions between levels or feedback for the player.

**Step 8: Conclusion and Next Steps**

8.1. Recap what you've accomplished in the project.

8.2. Encourage further exploration of web development, JavaScript, and game development.

**Step 9: Homework Assignment (Optional)**

9.1. Assign additional features or challenges to students for further practice and creativity.

**Step 10: Assessment (Optional)**

10.1. Assess students' understanding of the project through a review of their customized games or a quiz.

**Step 11: Q&A and Assistance (Ongoing)**

11.1. Be available to answer students' questions and provide assistance as they work on their projects.

**Step 12: Follow-up Sessions (Optional)**

12.1. Consider scheduling follow-up sessions to review homework assignments, address challenges, and showcase students' projects.