MEMORY GAME README

NAME: NANDANA KRISHNA

BANNER ID: 001197012

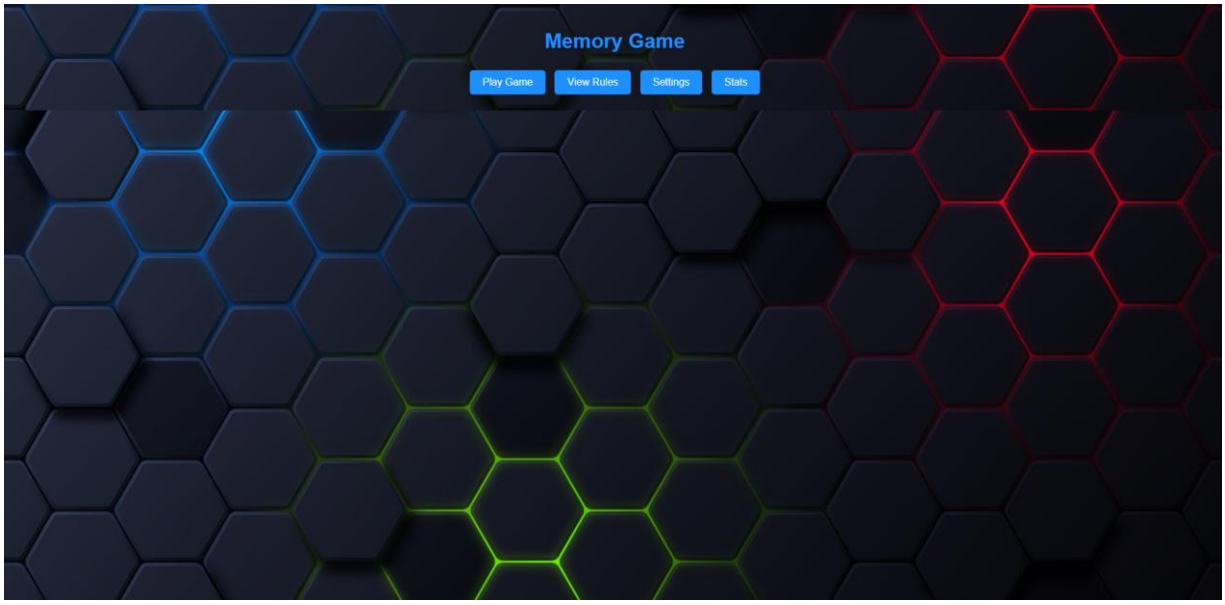
Module: ELEE-1159 Web Systems Engineering

Overview

The Memory Game is a fun and interactive game where players flip over two cards at a time to find matching pairs. The game continues until all pairs have been matched. Players can customize their experience, track high scores, and challenge themselves with different difficulty levels.

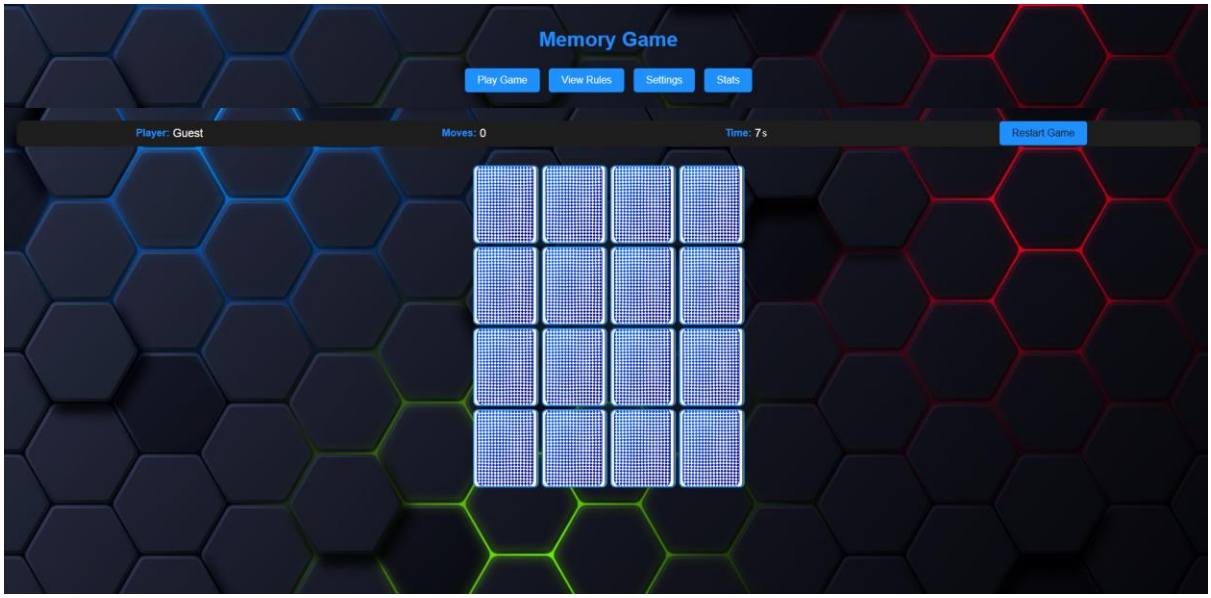
Game Features

The game has four main tabs in the user interface that allow the user to navigate through the app and interact with various features:



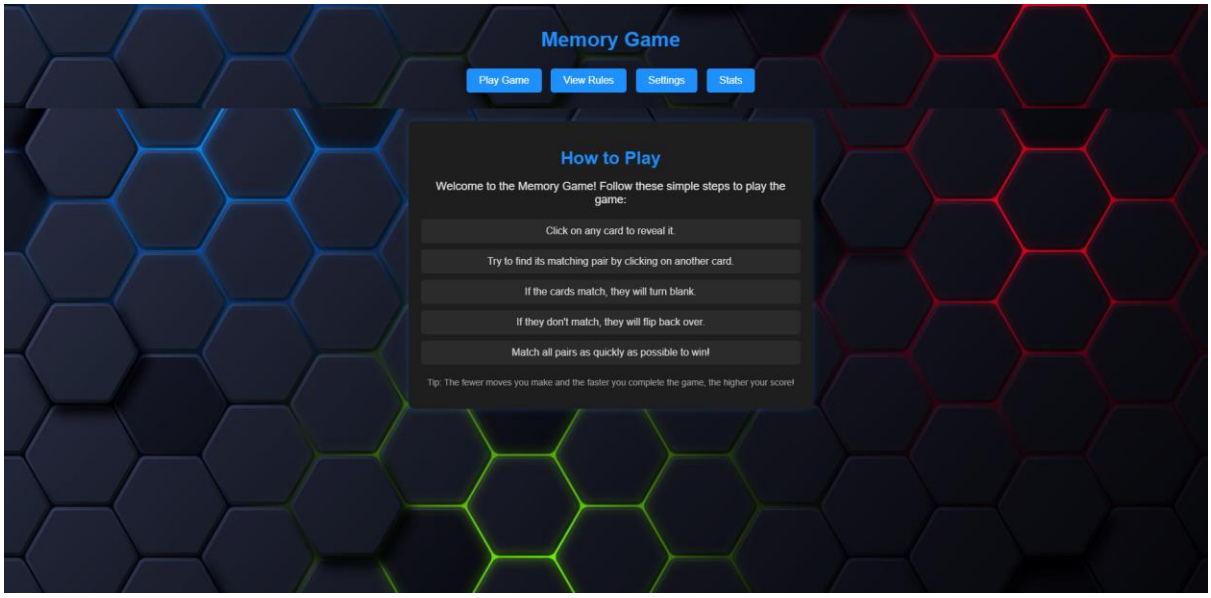
1. Play Game

- Objective: Start playing the Memory Game.
- When you click the Play Game tab, you'll enter the game screen where you can:
    - See your player name and the time taken to complete the game.
    - Click the Restart button to restart the game, which will reset the timer to 0 seconds and begin a new game.



2. View Rules

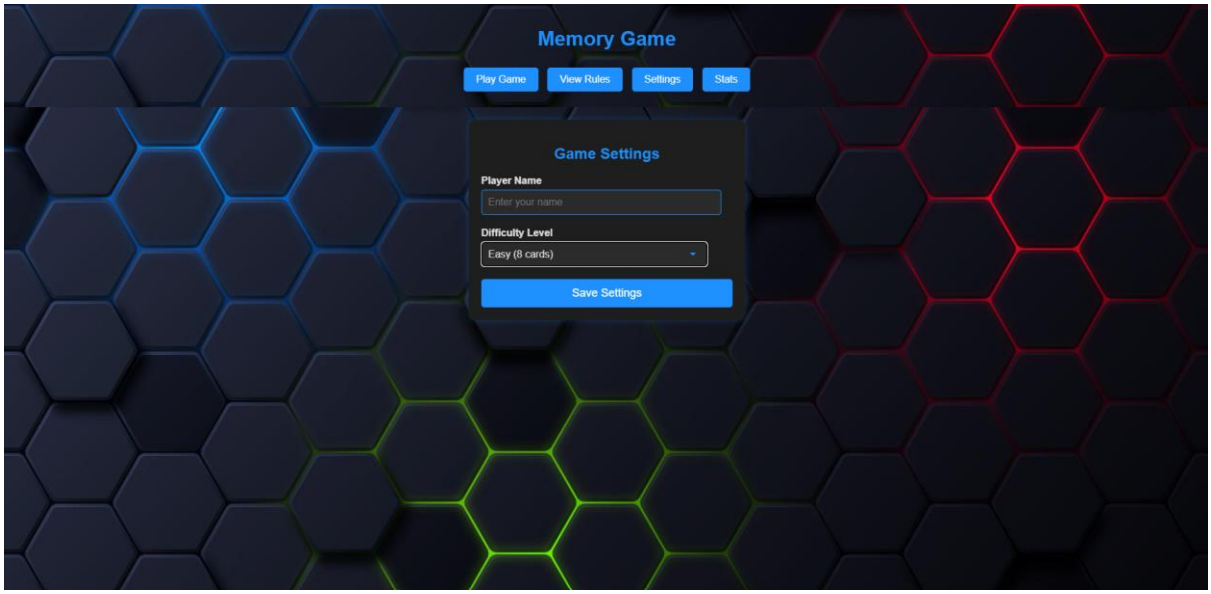
- Objective: Learn how to play the game.
- Clicking this tab will display the instructions and tips on how to play the Memory Game. This section provides guidance on how the cards are flipped, how to match them, and how the game ends.



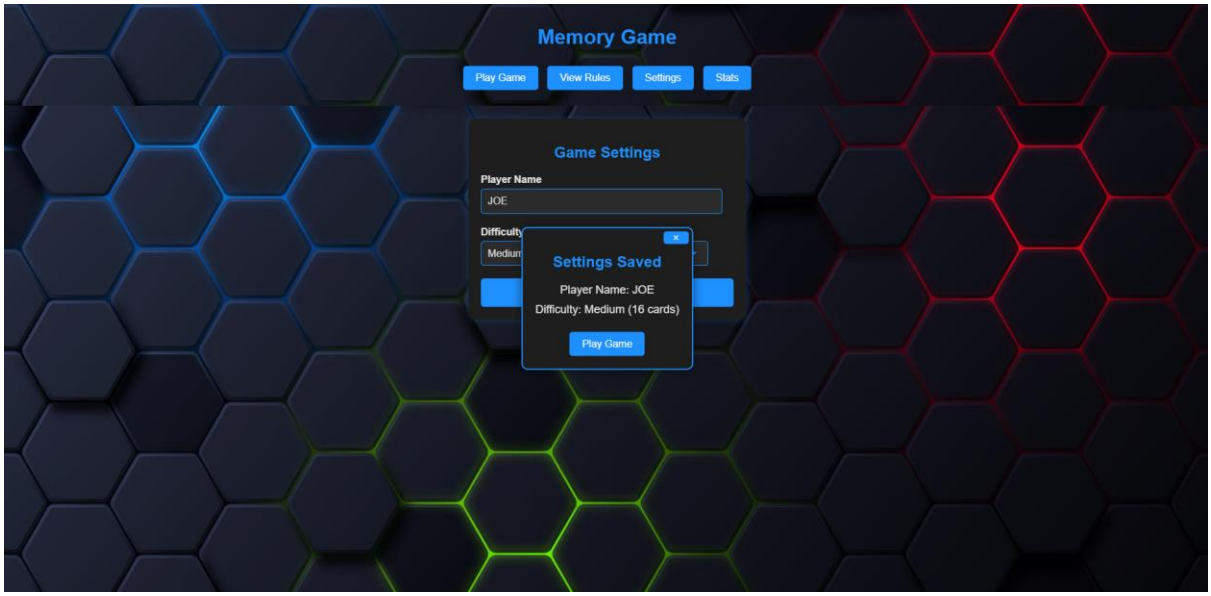
3. Settings

- Objective: Customize your game settings.
- In the Settings tab, you can:
    - Enter your player name.

- Choose the difficulty level for the game, which determines the number of cards:
    - Easy: 8 cards
    - Medium: 16 cards
    - Hard: 32 cards
    - Expert: 48 cards



- Once you enter your name and select the difficulty level, press the Save Settings button.
    - A popup will appear confirming your settings (player name and difficulty level).
    - The popup also contains a Play button that, when pressed, takes you to the game screen. There's also a Close button to dismiss the popup.
    - The default player name is set to "GUEST", and the default difficulty level is set to Expert.



4. Stats

- Objective: View the top scores and player rankings.
- In the Stats tab, you can:
    - See the top five scores saved in local storage, including:
        - Player name
        - Score (time taken)
        - Difficulty level
        - Rank secured

How to Play the Game

1. **Flipping Cards**:
    - Click on a card to flip it and reveal its face.
    - After the first card is flipped, click on a second card. Both cards will stay visible for 1 second.
    - If the cards match, they will be marked as matched and flipped to a blank image.
    - If the cards don't match, they will flip back over after a brief delay.
2. **End of Game**:The game ends when all card pairs are matched. Once finished:
    - A popup will display your score.
    - Score is calculate using the formula :

        score = maxScore - (weightForTime * timeTaken) - (weightForMoves * numberOfMoves)

        maxScore: Maximum achievable score.

        weightForTime: Penalty for time taken.

        weightForMoves Penalty for moves made.

        timeTaken: Time spent by the player.

        numberOfMoves: Moves made by the player.
    - Confetti will appear on the screen to celebrate your victory.
    - The score will be saved to the local storage.
    - The updated score will appear in the Stats tab.

3. **Restart the Game**: At any time, click the Restart button to reset the timer and start a new game.

Technologies Used

- HTML: Structure of the web pages.
- CSS: Styling and layout of the app.
- JavaScript: Game logic, interactions, and storage management.
- Local Storage: To save high scores and player statistics.

How to Run the Game

1. Download the ZIP File:
    - Click the "Download ZIP" button to download the project as a ZIP file.
2. Extract the ZIP File:
    - After downloading, extract the contents of the ZIP file to a folder on your computer.
3. Open the Game in a Browser:
    - Navigate to the extracted folder and open the index.html file in your web browser to start playing the game.
4. Play the Game:

- Use the Play Game, View Rules, Settings, and Stats tabs to interact with the game.

Code Implementation Overview for the Memory Game

This section outlines the core aspects of the code used to develop the memory game, focusing on key elements such as variables, data types, control flow, functions, DOM manipulation, local storage, and the game's overall look and feel.

1. Variables, Data Types, and Operations

The game is structured around a variety of variables and data types that manage its state, track user progress, and control gameplay elements. The code utilizes:

- Variables: Declared with let or const, these hold important data like the shuffled cards (cardArray), flipped cards (flippedCards), the score (score), and the move count (moves).
- Data Types: Arrays store the shuffled cards and track flipped cards, numbers are used to keep track of the score and moves, and strings represent card names or image paths.
- Operations: Arithmetic operations calculate scores and moves, while logical operations (e.g., ===) check if two flipped cards match.

2. Control Statements

Control statements, including if, else, and switch, govern the flow of the game logic:

- if Statements: These check for specific conditions, such as whether two flipped cards match or if the game is over.
- else Statements: Used when conditions are not met, such as flipping cards back when they don't match.

3. Arrays and Strings

Arrays and strings are central to managing game data:

- Arrays store the game elements, such as the shuffled cards (cardArray), and track flipped cards (flippedCards).
- Strings are used to store the names or paths of images representing the cards, which are dynamically altered as players flip the cards.

4. Functions

Functions structure the game logic into modular, reusable units, for example :

- initGame(): Initializes the game by setting up the cards and state.

- flipCard(): Flips the selected card and updates the state accordingly.
- checkMatch(): Compares two flipped cards and updates the game state if they match.
- updateScore(): Updates the score and displays it in the user interface.

5. DOM Manipulation

The game heavily relies on DOM manipulation to provide a dynamic and interactive user experience, for example:

- Card Flipping: The classList.add() method is used to visually flip cards, and setTimeout() handles delays when resetting unmatched cards.
- Updating Game State: Using document.getElementById(), the game dynamically updates the score, number of moves, and game status on the page.

6. Testing and Debugging

The following outlines some of the testing conducted during the development of the game.

Test/Debugging Method	Description
Console Logging	Used to log important variable values at different stages to track the flow and state of the game.
Breakpoints	Set breakpoints in the browser's developer tools to inspect variable states during execution.
Conditional Logging	Logs specific information when a condition is met to isolate issues.
Checking DOM Changes	Ensures the DOM updates correctly after actions like flipping cards or updating the score.
Simulated User Input	Simulates flipping cards to check game flow, ensuring the correct number of cards are flipped and checked for matches.
Game State Inspection	Inspects the state of the game, such as the flipped cards array, score, and moves, to ensure consistency.
Error Handling	Ensures that any unexpected behavior is caught and properly handled, such as handling undefined or null values.
Local Storage Checks	Verifies the proper saving and retrieval of data from local storage, ensuring progress persists.

7. Downloading to Local Storage

Local storage is leveraged to retain the player's progress across sessions. Using the localStorage API, the game saves the score and high scores, allowing players to pick up where they left off even after a page refresh.

8. Look and Feel of the Application and Code

The aesthetic design of the game was achieved using CSS to create an engaging and user-friendly interface:

- Styling: Cards and buttons are styled to be visually appealing, with hover effects enhancing interaction.
- Responsive Design: The layout adapts to different screen sizes, ensuring a consistent experience across devices.
- Color Scheme: A visually appealing color palette ensures the game is easy to navigate.

Further Improvements:

Some of the improvements that could be made in the future are:

- Timer: Add a countdown for urgency and competition.
- Sound: Integrate sound effects and background music.
- Mobile Responsiveness: Ensure the game is fully responsive across devices.
- User Customization: Allow theme, card, and background customization.
- Accessibility: Add text-to-speech and color scheme options for accessibility.

Conclusion

In conclusion, the memory game has been successfully developed using HTML, CSS, and JavaScript, incorporating essential programming concepts such as variables, control statements, functions, arrays, and DOM manipulation. The game provides an interactive and visually appealing user experience, with features like score tracking, card matching, and local storage for saving progress. Throughout the development process, testing and debugging ensured the game functions as intended, while further enhancements such as difficulty levels, sound effects, and multiplayer modes have been identified for future improvements. Overall, this project demonstrates an effective application of web development skills and provides a solid foundation for expanding the game’s functionality in future iterations.