Finding Pairs

Vishnu Vulli - C17241088
Harshini Gaddam - C11074076
Punugupati Sai Kumar - C13042143

Github repository link: https://github.com/Vishnu2819/Finding_Pairs/tree/main
Link to play the game: https://vishnu2819.github.io/Finding_Pairs/

Finding Pairs:

Finding pairs is a type of cognitive exercise designed to stimulate and challenge various cognitive functions, including perception, memory, problem-solving, decision-making, reasoning, attention, and language. It tests and enhances a player's memory and concentration skills. In a typical memory game, players use a set of face-down cards, taking turns to flip them over and find matching pairs.

This classic Finding pairs game features cards with Fortnite characters. The inclusion of Fortnite characters in the game is primarily for aesthetic and engagement purposes.

Game Objective:

The main objective of the memory game is to challenge and improve a player's memory and concentration skills by matching pairs of cards. The game typically consists of a set of face-down cards, and the primary goal is to find all the matching pairs within the set.

Here are the key objectives of a memory game:
Memory Enhancement: The game aims to improve a player's memory by requiring them to remember the locations of various images or symbols on the cards and matching them.
Concentration: Players need to maintain focus and concentration to track the positions of the cards they've flipped and identify matching pairs.
Cognitive Skills: Memory games help exercise various cognitive functions, including short-term memory, attention, and pattern recognition.
Entertainment and Engagement: Memory games are often designed with engaging themes and images to make the exercise more enjoyable and fun.
Challenge: Players are challenged to complete the game with the fewest attempts and in the shortest time possible, adding an element of competition and motivation.

Game rules:

The rules of the Finding pairs game are relatively simple and consistent across most versions of the game. Here are the typical game rules for a memory game:
1.Setup:
Lay out of 4*4 grid of face-down cards on a playing surface. There are 16 cards in total, with each card having a matching pair. The number of cards can vary, but they are usually arranged in rows and columns.
Each card has a matching pair. This means that there are two cards with the same image, symbol, or pattern on the playing surface.

2.Objective:
The main objective of the game is to find and match all the pairs of cards in the 4x4 grid. Each pair consists of two identical cards.
3.Gameplay:
Revealing Cards: On a player's turn, they can choose any two face-down cards to flip over, revealing the images on those cards. The player must specify which two cards they want to flip.
Matching Pairs: If the two revealed cards match (i.e., they have the same image or symbol), the player keeps the pair and gets another turn. Matching pairs are kept open to keep track.
Not Matching Pairs: If the two revealed cards do not match, then the cards face down again, and it becomes the next turn.
Memory and Concentration: The challenge of the game is to remember the positions of the cards that have been revealed previously. Players must use their memory and concentration to find matching pairs based on their observations.
4.Winning:
If number of turns <=12, it displays Incredible! You've got a great memory. You took (number of turns) tries to finish.
If number of turns >=12 and turns<16, it displays Great job! Keep it up!. You took (number of turns) tries to finish.
If number of turns >=16 and turns<20, it displays Good job!Keep trying!. You took (number of turns) tries to finish.
If number of turns >20, it displays I'd check with a doctor if I were you. You took (number of turns) tries to finish.
5.Variations and Additional rules:
To make this memory game more challenging, we thought of introducing variations and additional rules that test and enhance players' memory and concentration skills.
Here are some ways to increase the challenge in a memory game:
Increase the Number of Cards: Introducing additional cards to expand the grid's dimensions enhances the cognitive demands on players, necessitating the retention of an increased number of card positions, thereby elevating the level of challenge. To intensify the intricacy of the game, we've opted for a 4*4 grid configuration, accommodating a total of 16 cards.
Decrease the Time Limit: Implement a time limit for each turn. Players must find and match pairs within a set time frame, which adds pressure and requires quicker thinking.
Limit the Number of Turns: Restrict each player to a specific number of turns. When players have a limited number of opportunities to flip cards, they need to make each turn count.

Technologies:

To enhance the visual presentation and interaction of web page elements, the code is primarily a React component written in JavaScript, Cascading Style Sheets (CSS), a style sheet language used for describing the look and formatting of a document written in HTML. Technologies and libraries used outline the dependencies and devDependencies used in the "memory-game" project, including React and various tools and libraries for development, linting, and optimization. CSS is used in conjunction with various web development technologies and libraries to streamline development and create more complex, interactive web applications.

1. HTML: HTML is used to structure the content of a web page. It defines the elements (e.g., headings, paragraphs, buttons, App, Card grid, Confetti) and their organization on the page.
CSS (Cascading Style Sheets): CSS is used to control the presentation and styling of the HTML elements. We used CSS to style elements like buttons, images, and grids.

2. React: React is a JavaScript library for building user interfaces, and managing components and states in the application. The entire component is built using React, making use of React features like functional components, hooks (useState, useEffect), and JSX for rendering

3. CSS: The component includes a reference to an external CSS file named "Score.css" to style its elements. The CSS styles are applied to control the visual presentation of the component.

4. JavaScript: JavaScript is a programming language used to add interactivity and dynamic behavior to web pages. JavaScript is used to create interactive elements, handle user input, and respond to events like button clicks or form submissions.

5. Node.js: Node.js is the runtime environment used for executing JavaScript code on the server or in this project's build process.

6. GitHub Pages and GitHub repositories: We've leveraged GitHub repositories for collaborative coding and chosen GitHub Pages as our hosting solution. GitHub Pages excels in serving our static game directly from our repositories, streamlining development and deployment.

Libraries and Frameworks:

React-DOM: React-DOM is a library used in React applications to render components in the browser.

React-Confetti: This library is used for rendering confetti animations in the application when certain conditions are met.

Vite: Vite is a build tool and development server for web projects. It is used for development and building the project.

Math.random(): JavaScript's built-in Math.random() function is used for shuffling the cards by generating random values.

Bootstrap: A popular CSS framework that provides pre-designed CSS components and responsive layouts.

jQuery: A JavaScript library that simplifies tasks like DOM manipulation and event handling.
React, Angular, Vue.js: Front-end JavaScript frameworks for building complex, interactive user interfaces.

Custom Components: The application utilizes custom components such as SingleCard, ConfettiComponent, and Score to provide specific functionalities. These custom components are essentially libraries of reusable code.

setTimeout: JavaScript's setTimeout function is used for managing timed events and actions within the application.

useState and useEffect: These are built-in React hooks. useState is used to manage local component state variables (text and text2), and useEffect is employed to handle side effects and lifecycle events, including updates to the component's state and the execution of certain functions based on changes in props or state.

Document Object Model (DOM): The code interacts with the DOM by assigning the document.body object to the body variable. Although not a library, it's part of standard JavaScript and is used to reference the body element of the HTML document.

Setup and deployment instructions

Step 1: Prerequisites

These are installed prior before starting the execution.
Node.js and npm (Node Package Manager)
A code editor (e.g., Visual Studio Code)
A command-line interface (e.g., Terminal or Command Prompt)
npm install is used to install the prerequisites.

Step 2: Creating a New React Project

Using the command-line interface, navigated to the directory to create your project using the cd command.
Used npm create vite@latest to create a new React application.

Step 3: Navigate to the Project Directory

Once the project is created, navigated to the project directory

Step 4: Start the Development Server

npm install is used to install the package.json file
npm run dev command is used to run the React application locally during development

Step 5: Executing Components and Code

Started writing React components and application code. Necessary components are created.

Step 6: Added Styling

For any CSS or styling, Own styles is written in CSS files, used CSS-in-JS libraries like styled-components or CSS preprocessors like SASS. Imported CSS files in the components.

Step 7: Testing and Development
Developed and tested the React application as needed. Used the local development server to view the changes in real-time.

Step 8: Built the Production Version
npm run build command is used when the application is ready for deployment, created a production build. This command is used to create an optimized and minified version of the application in the build directory.

Step 9: Deployment
GitHub serves as the deployment platform where individual developers push their code and diligently commit changes.

Steps to execute the code

- Step1: Install the node.js
- Step2: Install the package, confetti library
  command used: **npm install**, **npm install react confetti**
- Step3: Launch the game
  command used: **npm run dev**

Credits for third-party assets and code:
https://github.com/iamshaunjp/React-Firebase/tree/lesson-58
https://www.w3schools.com/
https://chat.openai.com/
https://youtu.be/LxkXT9WD7yk?si=ONZHvJheeqBeRRp1

Reflection on Design and development process:

Challenges Faced During the Project:
Complex Logic: The game involved intricate game logic, including turn management and determining the end of the game. Ensuring that the logic worked flawlessly presented a significant challenge.
Game State Management: Managing the game state, including tracking which cards are flipped, matched, or not matched, proved to be tricky.
UI/UX Design: Designing an intuitive and visually appealing user interface was a challenging task.
Responsiveness: Making the game responsive to different screen sizes and orientations posed a significant challenge.
Testing: The memory game required thorough testing to identify and fix issues such as card mismatches, game freezes, or any unexpected behaviors.
Scalability: When planning to expand the game with additional features or levels, managing the codebase's scalability became a challenge.
Debugging: Identifying and fixing bugs in the game logic or user interface turned out to be a time-consuming task.

Specific Technical Challenges: We encountered difficulties in achieving a smooth card flip animation and in setting up pop-up timing.

What Worked in the Project:
The success of the project was driven by several factors:
Effective Teamwork: Our collaborative efforts and adept time management played a vital role in the project's success.
React Integration: Using React, a popular JavaScript library for building user interfaces, simplified the development process. React's component-based architecture allowed for the creation of modular and reusable application components.
State Management with React Hooks: Leveraging React's useState and useEffect hooks effectively managed the application's state and side effects.
Third-Party Libraries: Integration of third-party libraries, such as react-confetti, added fun and interactive elements to the game, enhancing the user experience.
Code Organization: Separating components into individual files, each with a clear and specific purpose, improved code readability and maintainability.
Best Practices: Adhering to best practices, such as keeping components stateless when possible and providing clear and concise comments, enhanced code quality.

What Didn't Work in the Project:
To mitigate challenges and improve project efficiency, the following lessons were learned:
Effective Task Breakdown: We learned to break the project into smaller tasks, tackle one challenge at a time, and continuously test and develop the game.
Seeking Help: We didn't hesitate to seek help from online communities, forums, and tutorials. Learning from others' experiences smoothed the development process.
Project Management Practices: We emphasized proper planning, risk assessment, clear communication, and adaptability. These project management practices were instrumental in addressing challenges effectively.

In conclusion, the project's success was attributed to the effective use of React, collaborative teamwork, and best practices in code organization and state management. Overcoming challenges involved careful planning, testing, and seeking help when needed. The project's outcome was a rewarding memory card game with an engaging user experience. Our project's success is a testament to our development team's unwavering dedication and adaptability. Testing and error handling, ensured a seamless user experience. Navigating through evolving web development challenges underscored the importance of continuous learning. Meticulous planning and code organization were foundational to our triumph. We overcame challenges in scope management, time allocation, quality assurance, and effective communication. The complex game logic, user interactions, and CSS integration demanded meticulous attention, while achieving end-game score display mastery unveiled potential opportunities for advanced animations.