**Memory Matching Game**

**- Final Project Report**

The Memory Matching Game is a terminal-based card matching game with cards on a 4\*4 grid. In each round, the player selects two cars to flip. If the cards match, they remain face-up. If not, they are hidden again. The game continues until all pairs are matched.

The game is entertaining and educational. It helps to build good memory in humans as the player must remember card positions while playing this game.

**ALGORITHMS:**

These are the three main algorithms that I created for my game. Each algorithm has an important role in this game. While I used CHATGPT to help with the initial logic structure of these algorithms and I then edited the code to ensure it fit in my project, added comments and made sure that it met all requirements.

1. ***Board Initialization and Shuffling:***

The initializeBoard() method is used in the Board class and is responsible for setting up the game board before the game starts. This algorithm ensures that the game is different each time a player plays. A player can also easily memorize positions from previous runs.

First, the total number of card pairs is calculated and then for each pair, two cards with the same value are added to a list. Once all pairs are calculated, the list is shuffled using Collections.shuffle().

**Time Complexity**:

- Generating values: O(n)

- Shuffling: O(n)

- Grid assignment: O(n²), for n x n board.

A computer code with text

AI-generated content may be incorrect.

1. ***Card Selection and Matching Logic:***

The playGame() method is used in Game class. This algorithm controls the main game loop. It allows the player to flip two cards and check if the flipped cards match. It determines if the move was valid and keeps track of player moves.

The game asks the player to select two card positions one by one and then selected cards are flipped to show their values. If the number matches, the cards are marked as matched and stay face up. If they don’t match, the cards are flipped back after a short pause. This process continues until all card pairs are matched.

**Time Complexity**:

-One round: O(1)

-Entire game: Up to O(n²) depending on player guesses.

A computer code with text

AI-generated content may be incorrect.

1. ***Game Completion Check:***

This algorithm is used in the Board class. The method allMatched() checks whether the game is over by verifying if all cards have been successfully matched. It makes sure that game only finishes once every pair has been correctly matched.

It loops through each position and determines a response based on whether cards are matched or not, as a match found or an invalid match.

**Time Complexity**:

-O(n²) to check each card once.

A computer code with text

AI-generated content may be incorrect.

**DATA STRUCTURE:**

1. ***2D ARRAY (Card[] [] grid):***

I used a 2D Array in the Board class to store in a matrix format. A 2D Array represents the grid layout of the board manually and allows me to access and update any card using row and column way.

Each position in the grid holds a Card object. This structure makes it easy to display, check matches and flip cards.

1. ***ArrayList (List<String> values):***

I used ArrayList in the initializeBoard() method of the Board class to temporarily store duplicate values for shuffling. It allows to resize and is easy to shuffle cards using Collection.shuffleI().

I filled the list with matching pairs of values, shuffled them to their card positions and then assigned the values to the grid.

1. ***Stack (Stack<int[]> moveStack):***

I used Stack in the Game class to keep track of the player’s previous moves. I chose it because a stack is used for tracking and can also be used as an undo feature.

Each time the player makes a move, the selected coordinates are pushed out to stack and help to store move history.

1. **String (Card Value):**

I used a string in the Card class to show the value of the card. Strings are easy to display, compare and store and make the board more flexible for different value types as letters, numbers, emojis, etc.

The value is stored in each Card object and is used to check for matching pairs.

***Opportunity Encountered:***

While designing the logic for tracking the player's moves in the game, I saw an opportunity to enhance the game experience by using a stack data structure. Originally, I just wanted to count the number of moves but then I realized that storing each pair of selected card positions would make more advanced features like undoing previous moves, tracking user patterns.

***Error Encountered:***

During the development, I found a bug where selecting the same card twice still counts as a match and then I realized that I didn’t add a check to prevent the player from choosing the coordinates twice. Then, I updated the logic inside pickCard() to ensure that a second card can't be the same as the first one and also ensured that flipCard() checks whether the card is already flipped to prevent invalid moves. This update made the game more fair and logical.

***Changes in the Next Version:***

In the next version of this game, I would like to add a score and timer system to track the performance and add challenges. Scores will build the interest of the player and make the game more interesting. I also create certain difficulty level as 4\*4, 6\*6, 8\*8 boards.