MEMORY MATCH MADNESS

A CONSOLE BASED MEMORY GAME IN C

GROUP MEMBERS:

SYEDA FAIZA ASLAM CT-25064
MAHREEN CT-25069

NED UNIVERSITY OF ENGINEERING AND TECHNOLOGY

OVERVIEW

- Introduction
- Objectives
- System Requirements
- System Design
- Implementation Details
- Testing and Output
- Results and Discussion
- Conclusion

INTRODUCTION:

- Games have long been an effective medium for developing programming skills due to their reliance on logic, data structures, and interaction design. The **Memory Match Madness** project implements a simple yet engaging memory puzzle game using the **C language**. The game operates in a console environment and allows players to flip hidden cards to find matching pairs.
- The primary motivation for this project is to create an entertaining and educational tool that strengthens logical thinking while illustrating key C programming concepts. Through this project, we explored how functions, loops, arrays, randomization, and conditionals can be integrated to form an interactive application.

OBJECTIVES:

- To design and implement a console-based memory game using the C language.
- To apply programming principles such as modular design, functions, arrays, and control flow.
- To integrate a scoring mechanism that rewards correct matches and penalizes incorrect guesses.
- To introduce multiple levels of difficulty (Easy, Medium, Hard) for progressive gameplay.
- To implement time tracking for player performance evaluation.
- To provide a clear and user-friendly interface suitable for beginners in programming

SYSTEM REQUIREMENTS:

Hardware Requirements

- **Processor:** Intel/AMD Dual Core or higher
- **RAM:** Minimum 2 GB
- **Storage:** 50 MB free disk space
- **Display:** Console or terminal capable of text display

Software Requirements

- **Operating System:** Windows
- **Compiler:** GCC
- **Language Used:** C
- **IDE (optional):** Dev-C++

FUNCTION DESCRIPTION:

- **LIBRARY HEADERS AND THEIR FUNCTIONS USED**
- **<stdio.h>**
- The stdio.h header file in C is responsible for handling input and output operations in a program. Its functions include:
- **FUNCTIONS:**
- **printf()** Used to print text, messages, and the game board on the console
- **scanf()** Takes user input such as difficulty level, row, and column choices.
- **getchar()** Reads a single character from the input buffer (used to pause or clear input).

<stdlib.h>

The stdlib.h header file in C stands for standard library and it is responsible and provide functions for general purpose tasks like

Memory management (malloc(),calloc(),free()).

Program control (exit(), abort()).

Random numbers (rand(), srand()).

FUNCTIONS:

system() is used to executes a command on the operating system here, used to clear the screen (cls on Windows).

- **srand()** is used to seeds the random number generator using the current time to ensure different shuffle results each run.
- **rand()** is used to generates random numbers used to shuffle the board values.

<time.h>
The time.h header file in C is used for date and time operations. It provides the function to get the current time intervals, and format

FUNCTIONS:

- **time()** Returns the current system time, used as a seed for randomization (srand(time(NULL))).

User-Defined Function

void clearScreen() is used in the program to clears the console screen to make the game visually clean and interactive after every turn. It uses system("cls") on Windows. It uses preprocessor directives (#ifdef _WIN32, #else and #endif) to determine the operating system at compile time and execute the appropriate system command (cls for Windows) to refresh the display for a cleaner user interface.

IMPLEMENTATION DETAILS:

Test ID	Test Description	Input	Expected Output	Pass Criteria
TC01	Launch game	Run the program memory match	Title screen and difficulty menu appear	Game starts without crash
TC02	Select Easy difficulty	Choose option 1	Game starts with a **2×2 grid**	Easy level board displayed
TC03	Select Medium difficulty	Choose option 2	Game starts with a **4×4 grid**	Medium board (4×4) displayed
TC04	Select Hard difficulty	Choose option 3	Game starts with a **6×6 grid**	Hard board (6×6) displayed
TC05	Invalid difficulty input	Enter a or 5	Error message shown: “Invalid input” or “Please enter 1, 2, or 3.”	Input validated, reprompt displayed
TC06	Select unrevealed valid card	Enter coordinates (e.g., Row: 1, Col: 1)	Selected card temporarily revealed	Correct tile shown

TC07	Select same card twice	Choose same coordinates twice	Message: “You selected the same card twice. Turn cancelled.”	Turn cancelled properly
TC08	Select already revealed card	Pick a tile that’s already open	Message: “That card is already revealed.”	Turn cancelled, no crash
TC09	Matching pair	Select two tiles with the same number	Message: “It’s a MATCH! +10 points.”	Both remain revealed, +10 score
TC10	Non-matching pair	Select two different tiles	Message: “Not a match. -2 points.”	Both hidden again, -2 score
TC11	Negative score check	Miss multiple pairs intentionally	Score decreases but not below 0	Score never < 0
TC12	Level completion	Match all pairs	“LEVEL COMPLETE” summary displayed	Moves to next level

TC13	Multi-level progression	Start from Easy and complete up to Hard	Three level summaries and final screen appear	Levels progress in correct order
TC14	Timing check	Let timer run for several seconds	Time counter increases accurately	Timer updates correctly
TC15	Clear screen check	Observe console after each turn	Screen clears automatically between moves	Clean transition between turns
TC16	Exit after game end	Finish Hard level	“CONGRATULATIONS” and “FINAL SCORE” displayed	Program exits gracefully

OUTPUT AND RESULT:

The program was tested manually by playing through it on three levels of difficulty: Easy, Medium, and Hard. Different test cases were carried out to ensure smooth gameplay and that the software responds well to any user input. During the testing, both normal and edge cases were test, like entering invalid inputs, selecting the same card in succession, and choosing already revealed tiles. The game responded correctly, showing appropriate error messages and not crashing. Also, the scoring system increased by 10 with every correct match and decreased by 2 upon a wrong pair but never below zero. The screen, after every turn, was properly cleared for better visibility. Level progression from Easy to Hard went successfully, and it showed the right messages upon completing each level. Overall, all test cases were passed, and everything worked fine: stable performance, accurate scoring, and smooth user interaction.

CONCLUSION:

Memory Match Madness demonstrates the use of C for interactive console applications, integrating arrays, loops, conditionals, and functions. The project taught management of two-dimensional arrays, randomization, function-based program structure, and user-friendly interface design. It provides a solid foundation for further development, including GUI-based enhancements, and serves as an effective learning exercise for beginner C programmer

THANK YOU