MEMORY MATCH MADNESS:

A CONSOLE-BASED MEMORY GAME IN C

SUBMITTED BY:

SYEDA FAIZA ASLAM
ROLL NO: (CT-25064)
MAHREEN
ROL NO: (CT-25069)

SUBMITTED TO:

Mr. MUHAMMAD ABDULLAH
COURSE CODE: CT-175
COURSE TITLE: PROGRAMMING FUNDAMENTALS

SEMESTER:

1ST SEMESTER FALL 2025

DATE OF SUBMISSSION:

12-11-2025

DEPARTMENT:

Department of Computer Science and Information Technology

INSTITUTE NAME:

NED University of Engineering and Technology

INDEX:

TABLE OF CONTENTS:

1. Introduction
2. Objectives
3. Literature Review / Background
4. System Requirements
5. System Design
    5.1 Architecture Overview
    5.2 Algorithm and Flowchart
6. Implementation Details
    6.1 Function Descriptions
    6.2 Data Structures Used
7. Testing and Output
8. Results and Discussion
9. Limitations and Future Scope
10. Conclusion

ABSTRACT:

This project report presents the design and implementation of a console-based game titled **Memory Match Madness**, developed in the **C programming language**. The game challenges the player’s memory and concentration by requiring them to find pairs of matching tiles within a grid. The project incorporates multiple difficulty levels, score calculation, time tracking, and progression through stages of increasing complexity.
The objective of the project is to demonstrate the practical application of programming concepts such as arrays, randomization, control structures, modular programming, and user input/output. The final implementation offers a user-friendly text-based interface and can serve as a foundation for more advanced versions with graphical enhancements.

---

1. INTRODUCTION:

Games have long served as an engaging and practical way to enhance programming skills because they naturally involve logic building, problem-solving, and interactive design. **The Memory Match Madness** project was developed as a console-based memory puzzle game using the **C** programming language. It challenges players to uncover and match pairs of hidden cards, testing both memory and focus while demonstrating how fundamental programming concepts can come together in a cohesive application.

The primary goal of this project is to create a simple as well as effective learning tool that blends entertainment with education. Through the design and implementation of this game, we gained hands-on experience in applying essential programming concepts such as functions, loops, arrays, conditionals, and randomization. These elements work together to manage game logic, track user progress, and generate dynamic gameplay

In addition to reinforcing technical concepts, this project also helped in understanding the importance of modular programming, error handling, and user interaction within a console environment. By developing Memory Match Madness, we not only improved my logical thinking and problem-solving abilities but also learned how to design a program that is both functional and enjoyable for users.

---

2. OBJECTIVES:

The key objectives of this project are as follows:

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

---

3. BACKGROUND:

The concept of *memory matching games* has existed for decades, often appearing in educational and recreational contexts to enhance cognitive performance. The most common version is known as *Concentration* or *Pairs*, where players flip over two hidden cards to find matching pairs.

In the context, of programming education, such games have become a common learning exercise because they require:

- Management of two-dimensional arrays to represent grids,
- Randomization to shuffle pairs,
- Input validation and user interaction handling, and
- Logical comparisons between selected tiles.

Previous versions of memory games have been implemented in various languages like Python, Java, and C++. However, implementing one in **C language** demonstrates deeper understanding of low-level memory management and procedural programming — foundational to modern software development.

---

4. 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++

---

5. CODE

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

// Function to clear the console screen (works on Windows &
Linux)
void clearScreen() {
#ifdef _WIN32
    system("cls");
#else
    system("clear");
#endif
}

int main() {
    int board[6][6], shown[6][6];
    int size, totalScore = 0;
    int i, j;

    // Title
    clearScreen();
    printf("========================================\n");
    printf("        MEMORY MATCH MADNESS\n");
    printf("========================================\n");
    printf("Match all pairs to win!\n");
printf("Scoring: +10 for a match, -2 for a wrong guess\n\n");

// Choose difficulty
printf("Choose difficulty:\n");
printf("1) Easy (2x2)\n2) Medium (4x4)\n3) Hard (6x6)\n> ");
int level;
scanf("%d", &level);
while (getchar() != '\n');

if (level == 1)
size = 2;
else if (level == 2)
size = 4;
else size = 6;

// Start from chosen level to Hard
for (int lvl = level; lvl <= 3; lvl++) {
    if (lvl == 1)
        size = 2;
    else if (lvl == 2)
        size = 4;
    else
        size = 6;

    // Initialize shown array
    for (i = 0; i < size; i++)
        for (j = 0; j < size; j++)
            shown[i][j] = 0;

    int total = size * size;
    int pairs = total / 2;
    int nums[36];
    int k = 0;
    // Fill pairs
    for (i = 1; i <= pairs; i++) {
        nums[k++] = i;
        nums[k++] = i;
    }

    // Shuffle numbers
    srand(time(NULL));
    for (i = total - 1; i > 0; i--) {
        int r = rand() % (i + 1);
        int temp = nums[i];
        nums[i] = nums[r];
        nums[r] = temp;
    }

    // Fill the board
    k = 0;
    for (i = 0; i < size; i++)
        for (j = 0; j < size; j++)
            board[i][j] = nums[k++];

    // Game loop
    int found = 0, score = 0, tries = 0;
    while (found < pairs) {
        clearScreen();
        printf("Level %dx%d | Total Score: %d | Attempts:
%d\n",size, size, totalScore + score, tries);
        // Print board
        printf("\n    ");
        for (j = 0; j < size; j++)
              printf("%3d", j + 1);
        printf("\n   +");
```

```
for (j = 0; j < size; j++)
      printf("---");
printf("\n");

for (i = 0; i < size; i++) {
   printf("%2d |", i + 1);
   for (j = 0; j < size; j++) {
      if (shown[i][j])
                  printf("%3d", board[i][j]);
      else
                  printf("%3s", "*");
   }
   printf("\n");
}

int r1, c1, r2, c2;

// First card
while (1) {
   printf("\nSelect first card:\n Row (1-%d): ", size);
   scanf("%d", &r1);
   printf(" Col (1-%d): ", size);
   scanf("%d", &c1);
   while (getchar() != '\n');
   r1--; c1--;
   if (r1 < 0 || r1 >= size || c1 < 0 || c1 >= size) {
      printf("Invalid position!\n");
      continue;
   }
   if (shown[r1][c1]) {
      printf("Already revealed!\n");
      continue;
   }
```

```
    break;
}

shown[r1][c1] = 1;
clearScreen();

// Show after first pick
printf("Level %dx%d | Total Score: %d | Attempts: %d\n",
       size, size, totalScore + score, tries);
printf("\n    ");
for (j = 0; j < size; j++)
       printf("%3d", j + 1);
printf("\n   +");
for (j = 0; j < size; j++)
       printf("---");
printf("\n");
for (i = 0; i < size; i++) {
    printf("%2d |", i + 1);
    for (j = 0; j < size; j++) {
        if (shown[i][j])
                    printf("%3d", board[i][j]);
        else
                    printf("%3s", "*");
    }
    printf("\n");
}

// Second card
while (1) {
    printf("\nSelect second card:\n Row (1-%d): ", size);
    scanf("%d", &r2);
    printf(" Col (1-%d): ", size);
    scanf("%d", &c2);
    while (getchar() != '\n');
    r2--; c2--;
    if (r2 < 0 || r2 >= size || c2 < 0 || c2 >= size) {
        printf("Invalid position!\n");
        continue;
    }
    if (r1 == r2 && c1 == c2) {
        printf("Same card! Try again.\n");
        continue;
    }
    if (shown[r2][c2]) {
        printf("Already revealed!\n");
        continue;
    }
    break;
}

shown[r2][c2] = 1;
tries++;
clearScreen();

// Display board again
printf("Level %dx%d | Total Score: %d | Attempts: %d\n",
       size, size, totalScore + score, tries);
printf("\n    ");
for (j = 0; j < size; j++)
      printf("%3d", j + 1);
printf("\n   +");
for (j = 0; j < size; j++)
      printf("---");
printf("\n");
for (i = 0; i < size; i++) {
    printf("%2d |", i + 1);
```

```
            for (j = 0; j < size; j++) {
                if (shown[i][j])
                            printf("%3d", board[i][j]);
                else
                            printf("%3s", "*");
            }
            printf("\n");
        }

        // Check match
        if (board[r1][c1] == board[r2][c2]) {
            printf("\nMATCH! +10 points.\n");
            score += 10;
            found++;
        } else {
            printf("\nNot a match! -2 points.\n");
            score -= 2;
            if (score < 0)
                    score = 0;
            shown[r1][c1] = 0;
            shown[r2][c2] = 0;
        }

        printf("\nPress Enter to continue...");
        getchar();
        clearScreen(); // <-- clears after every try
    }

    printf("\nLEVEL COMPLETE! Score gained: %d\n", score);
    totalScore += score;
    printf("Press Enter to continue...");
    getchar();
}
```

```
    clearScreen();
    printf("========================================\n");
    printf("     GAME COMPLETE! FINAL SCORE\n");
    printf("========================================\n");
    printf("Total Score: %d\n", totalScore);
    printf("Thanks for playing!\n");

    return 0;
}
```

<u>SYSTEM DESIGN:</u>

5.1 Architecture Overview

The architecture of the game follows a **procedural modular design**:

1. **Input Module** – handles user input and validation.
2. **Game Logic Module** – manages board setup, shuffling, and matching.
3. **Display Module** – outputs the grid and score information.
4. **Control Module** – coordinates levels, score, and time tracking.

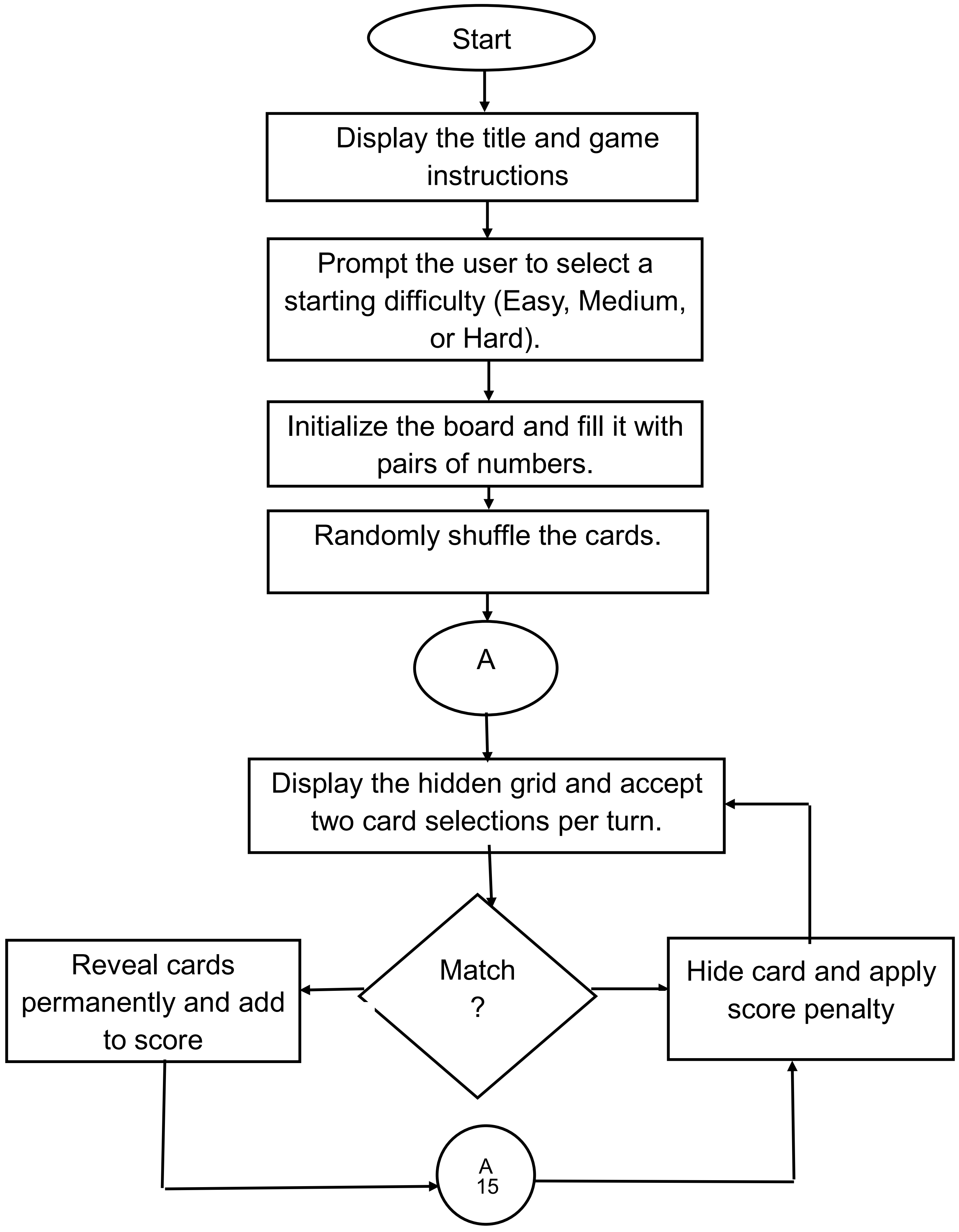
Each function is designed to perform a single logical task, promoting reusability and clarity.

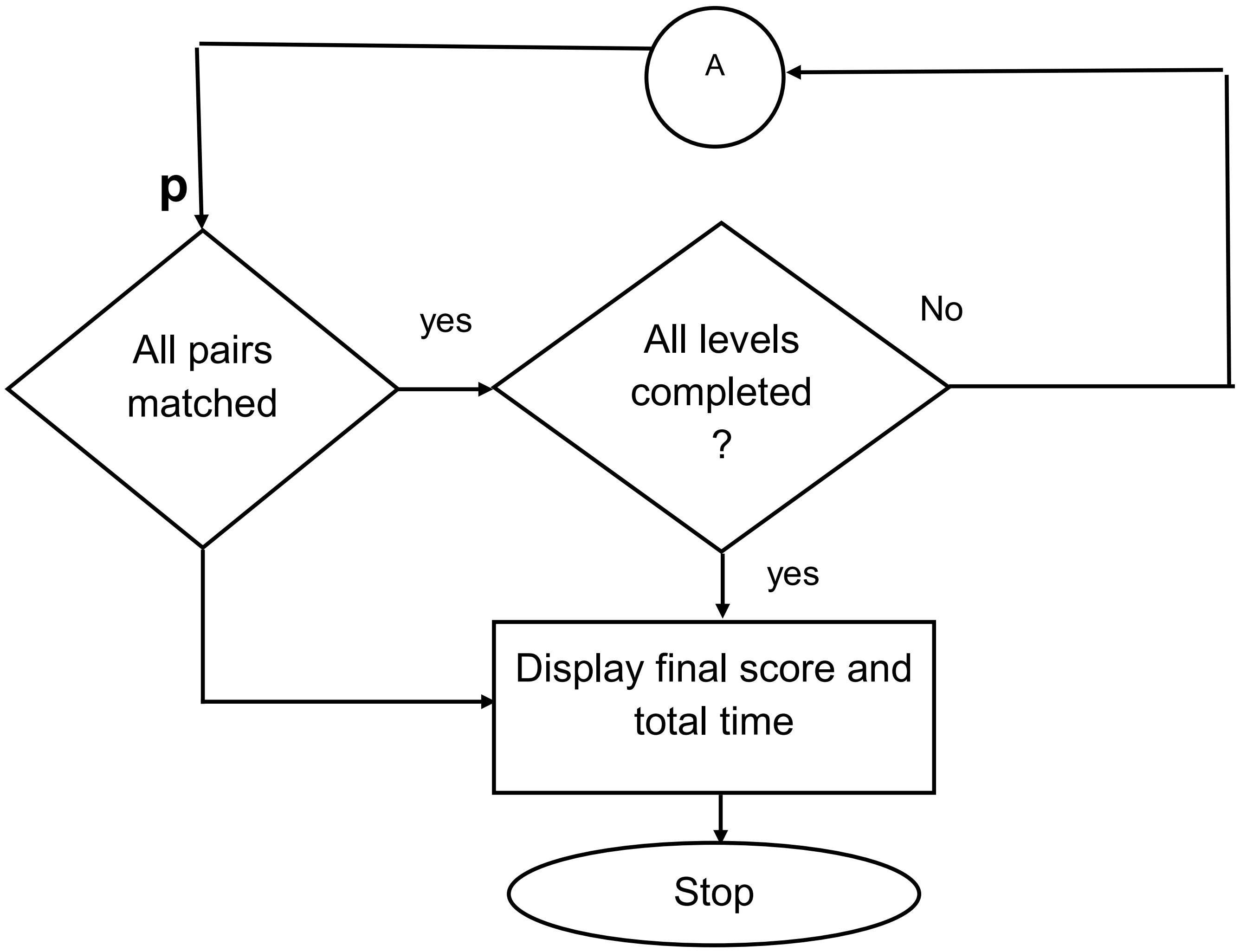
5.2 Algorithm and Flowchart

Algorithm Steps:

1. Start.
2. Display the title and game instructions
3. Prompt the user to select a starting difficulty (Easy, Medium, or Hard).
4. Initialize the board and fill it with pairs of numbers.
5. Randomly shuffle the cards.
6. Display the hidden grid and accept two card selections per turn.
7. If the two cards match, reveal them permanently and add to the score.
8. If they do not match, hide them and apply a score penalty.
9. Continue until all pairs are matched for that level.
10. Proceed to the next level until all levels are completed.
11. Display final score and total time.
12. Stop.

Flowchart:


Start
Display the title and game instructions
Prompt the user to select a starting difficulty (Easy, Medium, or Hard).
Initialize the board and fill it with pairs of numbers.
Randomly shuffle the cards.
A
Display the hidden grid and accept two card selections per turn.
Match ?
Reveal cards permanently and add to score
Hide card and apply score penalty
A
15



6.IMPLEMENTATION DETAILS:

FUNCTION DESCRIPTIONS

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.

---

7. TESTING AND OUTPUT:

Testing Methodology

Testing was conducted through manual gameplay at all difficulty levels. We tested edge cases by giving invalid inputs, repeated selections, and selecting the same tile twice.

TEST CASES:

Test ID	Test Description	Input / Action	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 2×2 grid	Easy level board displayed
TC03	Select Medium difficulty	Choose option 2	Game starts with 4×4 grid	Medium board (4×4) displayed
TC04	Select Hard difficulty	Choose option 3	Game starts with 6×6 grid	Hard board (6×6) displayed
TC05	Invalid difficulty input	Enter a or 5	Error message shown: “Invalid input” or “Please	Input validated, reprompt displayed

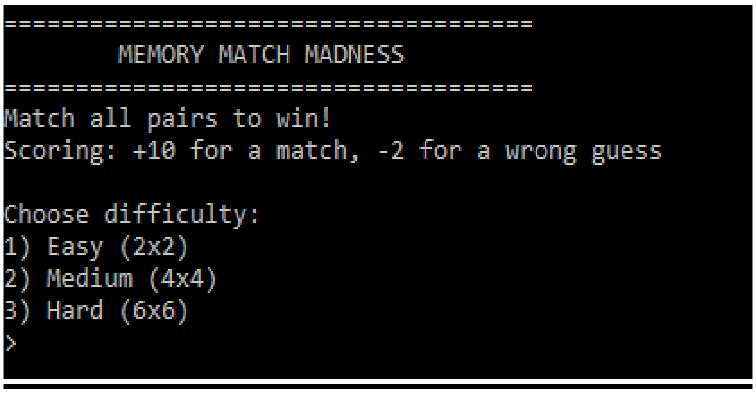
			enter 1, 2, or 3.”	
TC06	Select unrevealed valid card	Enter coordinates in range (e.g., Row:1, Col:1)	Card value revealed temporarily	Correct tile shown

TC07	Select same card twice	Choose same coordinate twice	Message “You selected the same card twice. Turn cancelled.”	Turn cancelled properly
TC08	Select already revealed card	Pick a tile that’s already revealed	Message “That card is already revealed.”	Turn cancelled, no crash
TC09	Matching pair	Select two tiles with 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	Score never goes below zero	Score >= 0 always
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, final screen shown	Correct progression order
TC14	Timing check	Let timer run for a while	Time in seconds increases properly	Timer updates

TC15	Clear screen check	Observe console during turns	Screen clears between moves	Clean transitions
TC 16	Exit after game end	Finish hard level	“CONGRATULATIONS” Message dispalys	Program exit

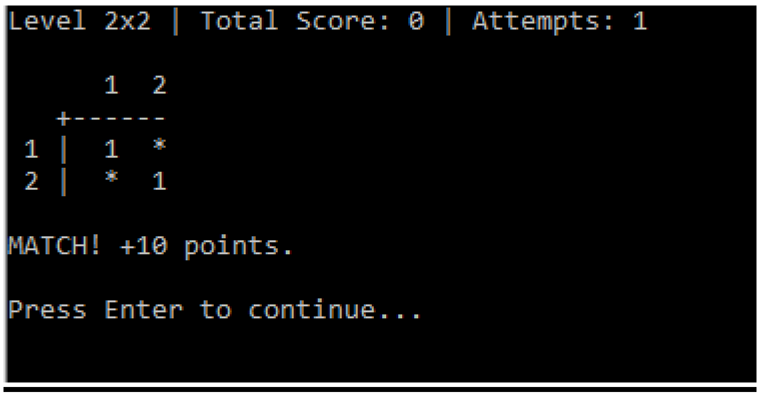
SAMPLE OUTPUT SCREENSHOTS:

Title screen



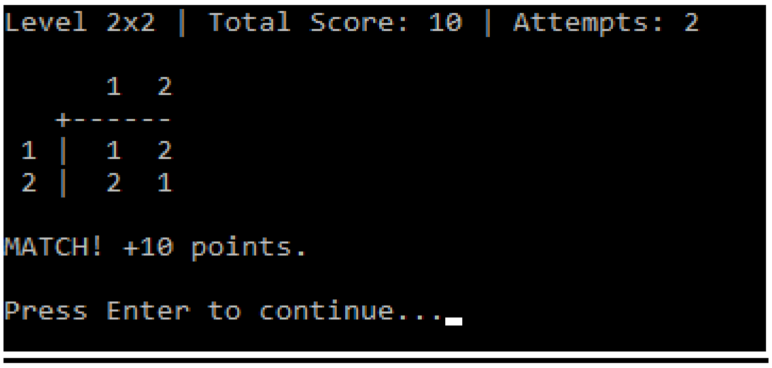
```
=====================================
        MEMORY MATCH MADNESS
=====================================
Match all pairs to win!
Scoring: +10 for a match, -2 for a wrong guess

Choose difficulty:
1) Easy (2x2)
2) Medium (4x4)
3) Hard (6x6)
>
```

Easy (level):



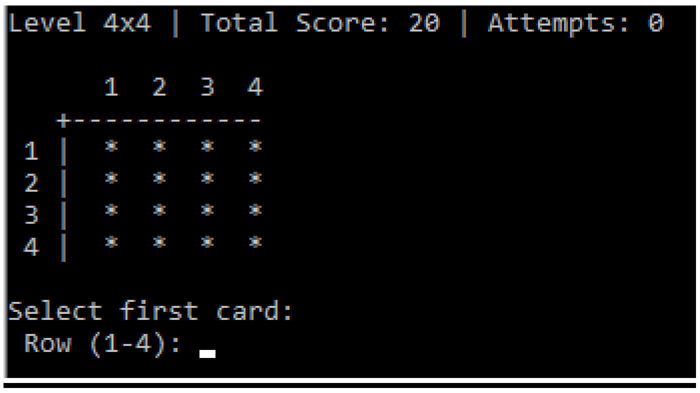
```
Level 2x2 | Total Score: 0 | Attempts: 1

      1  2
   +------
 1 |  1  *
 2 |  *  1

MATCH! +10 points.

Press Enter to continue...
```

Easy level completed:



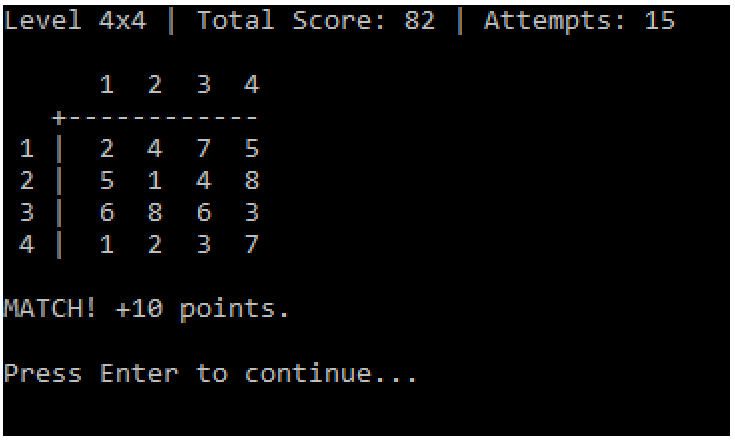
```
Level 2x2 | Total Score: 10 | Attempts: 2

      1  2
   +------
 1 |  1  2
 2 |  2  1

MATCH! +10 points.

Press Enter to continue...
```

Medium (level):



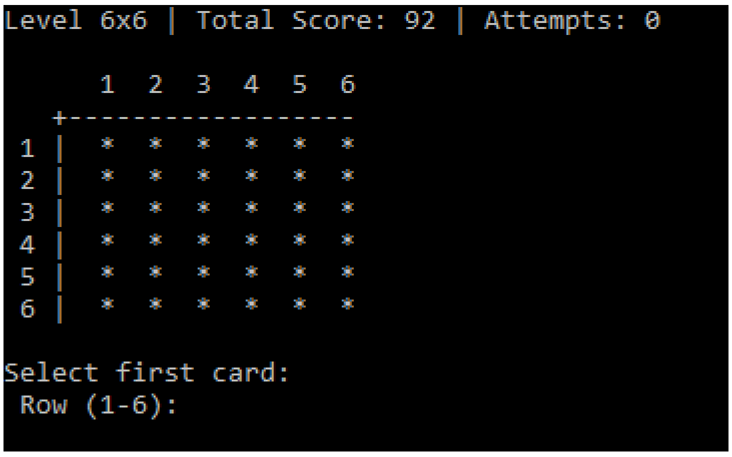
```
Level 4x4 | Total Score: 20 | Attempts: 0

      1  2  3  4
   +------------
 1 |  *  *  *  *
 2 |  *  *  *  *
 3 |  *  *  *  *
 4 |  *  *  *  *

Select first card:
 Row (1-4):
```

Medium level completed:



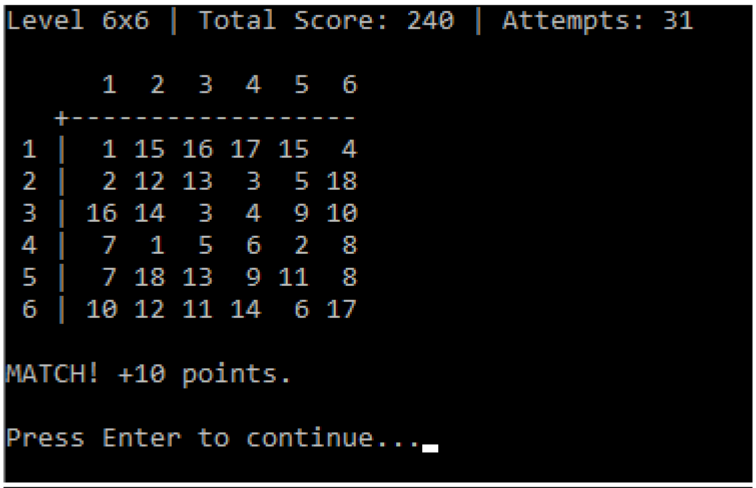
```
Level 4x4 | Total Score: 82 | Attempts: 15

      1  2  3  4
   +------------
 1 |  2  4  7  5
 2 |  5  1  4  8
 3 |  6  8  6  3
 4 |  1  2  3  7

MATCH! +10 points.

Press Enter to continue...
```

Hard (level):

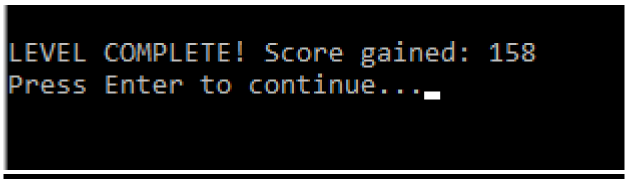


```
Level 6x6 | Total Score: 92 | Attempts: 0

      1  2  3  4  5  6
   +------------------
 1 |  *  *  *  *  *  *
 2 |  *  *  *  *  *  *
 3 |  *  *  *  *  *  *
 4 |  *  *  *  *  *  *
 5 |  *  *  *  *  *  *
 6 |  *  *  *  *  *  *

Select first card:
 Row (1-6):
```

Hard level completed:

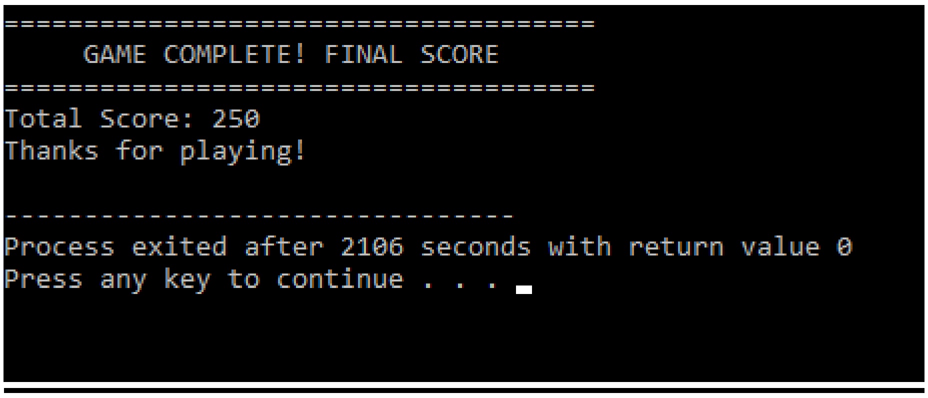


```
Level 6x6 | Total Score: 240 | Attempts: 31

      1  2  3  4  5  6
   +------------------
 1 |  1 15 16 17 15  4
 2 |  2 12 13  3  5 18
 3 | 16 14  3  4  9 10
 4 |  7  1  5  6  2  8
 5 |  7 18 13  9 11  8
 6 | 10 12 11 14  6 17

MATCH! +10 points.

Press Enter to continue...
```



```
LEVEL COMPLETE! Score gained: 158
Press Enter to continue...
```

End of game:



```
======================================
     GAME COMPLETE! FINAL SCORE
======================================
Total Score: 250
Thanks for playing!

--------------------------------
Process exited after 2106 seconds with return value 0
Press any key to continue . . .
```

7. RESULTS AND DISCUSSION:

The project successfully implements a multi-level memory match game. The player can start at any difficulty and progress through harder levels automatically. The scoring mechanism encourages accuracy, and the timer adds a sense of challenge.

The program handles invalid inputs gracefully and ensures the player experience is smooth within a console interface. The modular function-based approach makes the code easy to maintain and extend.

---

9. LIMITATIONS AND FUTURE SCOPE:

Limitations:

- The game operates in a console environment without graphical support.
- Sound effects and animations are not implemented.
- The grid size is limited to 6×6 due to static array declarations.

---

10. CONCLUSION:

The **Memory Match Madness** game is a functional and educational demonstration of how C can be used to develop interactive console-based applications. It successfully integrates essential programming constructs such as arrays, loops, conditional statements, and modular design.

Through this project, we learned about:

- Managing two-dimensional data structures,
- Implementing randomization in logic,

- Structuring programs with multiple interdependent functions, and
- Designing user-friendly command-line interfaces.

The project lays a strong foundation for further development into GUI-based games and provides an effective learning exercise for students beginning with the C language.
