EAST WEST UNIVERSITY

ASSIGNMENT REPORT

Submitted To

Dr. Md. Nawab Yousuf Ali

Professor

Department of Computer Science and Engineering

Submitted By

Name : Mohua Akter

Id No : 2022-3-60-112

Course Title : Computer Architecture

Course Code : CSE-360

Section : 01

Semester : Spring 2025

Table of Contents:

Introduction

This report provides an overview of the Multiplication Grid Game, detailing its functionality, architectural principles, challenges faced, insights gained, and potential improvements. The game is implemented in C using the **`ncurses`** library for terminal-based graphics and interaction.

Game Overview

The Multiplication Grid Game is a two-player game where a human competes against the computer. The game is played on a 6x6 grid filled with numbers, and the objective is to be the first to align four of your selected numbers in a row, column, or diagonal. Players take turns selecting two numbers from a selection array, multiply them, and place the product on the grid if it is a valid move. The game ends when a player aligns four numbers or when all possible moves are exhausted, resulting in a draw.

```
int grid[ROWS][COLS] = {
    {1, 2, 3, 4, 5, 6},
    {7, 8, 9, 10, 12, 14},
    {15, 16, 18, 20, 21, 24},
    {25, 27, 28, 30, 32, 35},
    {36, 40, 42, 45, 48, 49},
    {54, 56, 63, 64, 72, 81}
};

```

Architectural Principles

1. Arithmetic and Logic Simulation

- **Bitwise Operators and Modular Arithmetic:** While the game does not explicitly use bitwise operators, it employs arithmetic operations to calculate products and determine valid moves.

```
int product = selection_array[ptr1] * selection_array[ptr2];

```

- **Control Flow:** The game uses loops and conditional statements to simulate decision-making processes, akin to instruction execution paths in a CPU.

```
while (move_count < MAX_MOVES) {
    // Player's turn
    player_move();

    // Check for player win
    if (check_winner(1, win_cells)) {
        // Handle player win
    }

     // Computer's turn
     computer_move();

     // Check for computer win
     if (check_winner(2, win_cells)) {
         // Handle computer win
     }
}
```

2. Memory Organization

- **Pointers and Arrays:** The game uses arrays to represent the grid and selection array. Pointers are used to manage the current selection indices (`**ptr1**` and `**ptr2**`).

```
1  int selection_array[9] = {1, 2, 3, 4, 5, 6, 7, 8, 9};
2  int ptr1, ptr2;
```

- **Data Structures:** The `used` array tracks the ownership of grid cells, indicating whether a cell is empty, occupied by the player, or occupied by the computer.

```
1  int used[ROWS][COLS]; // 0=empty, 1=player, 2=computer
```

3. Instruction-Level Logic

- **Control Flow Constructs:** The game logic is implemented using loops (`while`, `for`) and conditional statements (`if-else`) to handle player moves, computer moves, and win condition checks.

```c
int check_winner(int player, int win_cells[4][2]) {
    // Iterate over each cell in the grid
    for (int i = 0; i < ROWS; i++) {
        for (int j = 0; j < COLS; j++) {
            // Check if the current cell belongs to the player
            if (used[i][j] == player) {
                // Check horizontal alignment
                if (j <= COLS - 4 && used[i][j+1] == player && used[i][j+2] == player && used[i][j+3] == player) {
                    for (int k = 0; k < 4; k++) { win_cells[k][0] = i; win_cells[k][1] = j + k; }
                    return 1;
                }

                // Check vertical alignment
                if (i <= ROWS - 4 && used[i+1][j] == player && used[i+2][j] == player && used[i+3][j] == player) {
                    for (int k = 0; k < 4; k++) { win_cells[k][0] = i + k; win_cells[k][1] = j; }
                    return 1;
                }

                // Check diagonal down-right alignment
                if (i <= ROWS - 4 && j <= COLS - 4 && used[i+1][j+1] == player && used[i+2][j+2] == player && used[i+3][j+3] == player) {
                    for (int k = 0; k < 4; k++) { win_cells[k][0] = i + k; win_cells[k][1] = j + k; }
                    return 1;
                }

                // Check anti-diagonal down-left alignment
                if (i <= ROWS - 4 && j >= 3 && used[i+1][j-1] == player && used[i+2][j-2] == player && used[i+3][j-3] == player) {
                    for (int k = 0; k < 4; k++) { win_cells[k][0] = i + k; win_cells[k][1] = j - k; }
                    return 1;
                }
            }
        }
    }
    return 0; // No winner found
}
```

4. I/O Simulation

Standard Input/Output: The `ncurses` library is used to handle user input and display the game state, simulating how a system interfaces with external input devices.

```
1  initscr();
2  noecho();
3  keypad(stdscr, TRUE);
4
```

5. Performance Awareness

Optimized Logic: The game employs efficient algorithms to check for valid moves and determine the winner, minimizing unnecessary computations and memory overhead.

```
1  int is_move_valid(int val) {
2      int r, c;
3      if (!find_in_grid(val, &r, &c)) return 0;
4      return !used[r][c];
5  }
6
```

Challenges and Solutions

Challenges

- **Handling User Input:** Managing user input in a terminal-based environment can be challenging, especially when ensuring responsiveness and accuracy.
- **Win Condition Checks:** Implementing efficient checks for win conditions (four in a row) required careful consideration of all possible alignments.

Solutions

- **`ncurses` Library:** Using `ncurses` provided a robust framework for handling user input and rendering the game interface.

```
1  void init_colors() {
2      start_color();
3      init_pair(1, COLOR_WHITE, COLOR_BLACK);
4      // Additional color pairs
5  }
```

- **Efficient Algorithms:** The win condition checks were optimized by iterating over the grid and checking only necessary alignments, reducing computational overhead.

```
if (check_winner(1, win_cells)) {
    // Clear the screen and redraw the game state
    clear();
    draw_title();
    draw_grid();

    // Highlight the winning cells for the player
    highlight_cells(win_cells, 1);

     // Redraw the selection array
     draw_selection_array();

     // Refresh the screen to show updates
     refresh();

     // Display a victory animation for the player
     victory_animation(1, win_cells);

     // Display a message indicating the player has won
      mvprintw(15, 5, "PLAYER WINS! Press any key to exit.");

     // Refresh the screen to show the victory message
     refresh();

     // Wait for the player to press a key before exiting
     getch();
}
```

Insights Gained

- **Memory Management:** The use of arrays and pointers highlighted the importance of efficient memory management in low-level programming.
- **Control Flow:** Implementing game logic reinforced the understanding of control flow constructs and their role in simulating decision-making processes.
- **I/O Handling:** The project provided insights into how systems handle input/output operations, particularly in a terminal-based environment.

Low-Level Simulation and Optimizations

- **Grid Representation:** The grid is represented as a 2D array, allowing for efficient access and manipulation of cell values.

```
1  int grid[ROWS][COLS];
```

- **Selection Array:** The use of a selection array with pointers (`ptr1` and `ptr2`) simulates low-level pointer arithmetic and memory access patterns.

```
1  int selection_array[9];
2  int ptr1, ptr2;
```

Suggestions for Future Improvements

- **AI Enhancements:** Improve the computer's decision-making algorithm to make it more challenging for the player.
- **Dynamic Grid Size:** Allow for dynamic grid sizes to increase replay ability and challenge.
- **Graphical Interface:** Develop a graphical user interface to enhance the visual appeal and user experience.

Conclusion

The Multiplication Grid Game project provided valuable insights into computer architecture principles, particularly in memory management, control flow, and I/O handling. The challenges faced and solutions implemented contributed to a deeper understanding of low-level programming concepts. Future improvements could further enhance the game's complexity and user engagement.