NATIONAL UNIVERSITY
of Computer & Emerging Sciences

PROJECT REPORT

CL1002
PROGRAMMING FUNDAMENTALS

Prepared BY:

MUHAMMED AHMED ASHFAQ
BCS-1E / 25K-0781

Contents

Abstract

This report displays the development and implementation of a **Connect 4** game in the **C programming language** using the **Raylib** graphics library. The game supports both Player vs Player (PVP) and Player vs CPU modes, with three AI difficulties, Easy, Medium and Hard. The scoring system tracks the player's scores against the CPU in permanent storage. This project demonstrates **event-driven programming, I/O operations in filing, and basic AI algorihms**.

1. Introduction

Connect 4 is a two-player strategy board game, played on a 6x7 grid. Players take turns dropping their respective colored discs into columns, with the aim to align four discs consecutively in any direction. The purpose of this project is to:

1. Apply the Connect 4 game in C.
2. Integrate AI opponents of varying difficulty.
3. Provide a graphical user interface with interactive menus.
4. Track and store player scores using file handling.

2. System Design

2.1. Software Architecture

The program follows a modular design:

- **Main Module (main.c):** Manages the game loop, graphics, animations, and menus
- **Logic Module (game-logic.c and game-logic.h):** Implements board management, winner checking, and AI algorithms
- **Data Handling:** Scores are read from and written to **scores.txt**.

2.2. Data Structures:

- **Board Representation:** 6x7 2D character array with elements used: **' ', 'X', 'O'**.
- **GameState:** Enum representing the current state of the UI and game.

```
typedef enum { MAIN_MENU, CPU_MENU, PVP_GAME, CPU_GAME,
SHOW_RULES, SHOW_SCORES, EXIT_GAME, END_SCREEN } GameState;
```

- **Difficulty**: Enum representing the AI difficulty. It also tells if the game is PVP or not by adding PVP difficulty

```
typedef enum { EASY, MEDIUM, HARD, PVP } Difficulty;
```

- **ScoreEntry:** Struct containing difficulty, moves, and scores for CPU games.

```
typedef struct {
    Difficulty diff;
    int moves;
    int score;
} ScoreEntry;
```

2.3. Functional Design:

2.3.1. Game Initialization

Function: startGame()

- It resets the board, moves count, and current player.
- Initializes **fallingPieces** animation structure.

2.3.2. Adding Pieces

Function: addPieceAnimated(int col, char type)

- Determines the target row and triggers drop animation.
- Updates board and check for a winner.

2.3.3. Winner Detection

Function: checkWinner(char player)

- Returns **WinnerInfo** containing winner ID and winning line coordination
- Supports horizontal, vertical, and diagonal detection.

2.3.4. AI Logic

Functions: cpuMoveEasy(), cpuMoveMedium(), cpuMoveHard()

- **Easy:** Random and valid move.
- **Medium:** Block opponent if the player is on verge of adding a winning piece.
- **Hard:** Firstly, check for a winning piece, then check for a blocking piece to block the player from winning and lastly used weightage system to focus more on center columns.

2.4. User Interface Design (UI):

- **Menu(s):** Main Menu, CPU Difficulty Menu, End Screen, Rules, Scores.
- **Buttons:** Highlight on hover and detect clicks using mouse events.
- **Board Display:** Drawn with rounded rectangles and circles for discs.
- **Turn Display:** Shows current player's turn and winning message.
- **Score Display**: Updates dynamically when the player wins against CPU.

3. Implementation

3.1. File management:

- Scores are stored in **scores.txt** in this format:
  difficulty moves score (where difficulty is integer: 0-Easy / 1-Medium / 2-Hard)
- Read at program start to display high scores.
- Updated only when a human player wins against CPU

3.2. Key Algorithms:

- **Winner Detection:** Scans all rows, columns, and diagonals for 4 consecutive discs.
- **AI Moves Selection:** It's dependent on difficulty and then select AI's move.
- **Score Calculation:** Score = Base[Difficulty] – (Moves x Penalty[Difficulty]), capped at 0 in start. The higher the difficulty, the higher the Base and Penalties will be

Difficulty	Base	Penalty
Easy	1000	10
Medium	1200	12
Hard	1500	15

3.3. Animation:

- Smooth falling piece animation done by **fallingPiece** struct.
- The movement is Frame rate independent and is calculated by **GetFrameTime().**

4. Testing and Results

- **Functional Testing:**
  - Correct placement of each piece in all columns.

    - Winner detection for horizontal, vertical, and diagonal connections.
    - AI behavior changes according to each difficulty.
    - Scores correctly updated only for human wins against CPU.
- **UI Testing:**
    - Button highlighting and click detection confirmed.
    - End Screen navigation and menu transitions verified.
- **Performance Testing:**
    - Smooth animations at 90 FPS.
    - No significant memory or CPU spikes.

Main game loop:

```c
int main(void){
    InitWindow(800,600,"Connect 4");
    SetTargetFPS(90);
    startGame();
    while(!WindowShouldClose() && current_state!=EXIT_GAME){
        BeginDrawing();
        ClearBackground((Color){173,216,230});
        switch(current_state){
            case MAIN_MENU: menu(); break;
            case CPU_MENU: cpuMenu(); break;
            case CPU_GAME: cpuDiff(diff); break;
            case PVP_GAME: diff=PVP; cpuDiff(diff); break;
            case END_SCREEN: endScreen(); break;
            case SHOW_RULES: rules(); break;
            case SHOW_SCORES: showScores(); break;
            default: break;
        }
        EndDrawing();
    }
    CloseWindow();
    return 0;
}
```

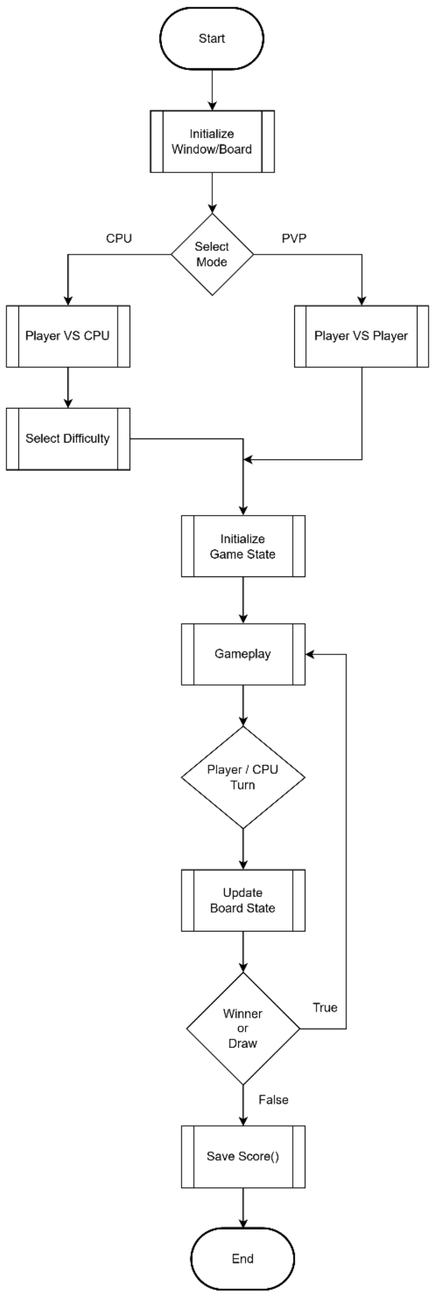
5. Discussion

The Project demonstrates:

- Combination of **graphics and game logic** in a modular design.
- Effective **AI with variable difficulty** without complex algorithms.
- **File I/O** for score tracking ensures permanent across all sessions.
- The code is structured for **readability, maintainability, and extendibility**.

Limitations:

- ✓ No network multiplayer support
- ✓ Hard AI still uses very simple shortcuts rather than advanced mechanisms.
- ✓ Graphics and layout fixed for 800x600 resolution.
- ✓ No Keyboard Support. Only mouse inputs are acceptable.



6. Conclusion

This project successfully demonstrates a fully functional **Connect 4 game** with PVP and CPU modes, a scoring system, and an animated GUI. The modular design of this project segregates logic from UI, ensuring that clarity and maintenance are present. Future improvements may include network multiplayer, more advanced AI strategies, adaptive graphical scaling to support higher resolution, and Keyboard support.

7. References

1. Wikipedia. *Connect Four*. https://en.wikipedia.org/wiki/Connect_Four
2. Raylib Library Documentation. https://www.raylib.com
3. Kernighan, B., & Ritchie, D. (1988). *The C Programming Language*. Prentice Hall.