Fundamental of programming

Iran university of science and technology

Capstone: Checkers game – Project report

Introduction

This project is a two-player Checkers game developed in C++ that adheres to the classic rules of the game. The implementation includes a command-line interface to interact with the game, and it enforces all necessary rules such as valid piece movements, mandatory captures, and king promotions. The objective of this project was to gain deeper insights into algorithmic problem-solving, game logic design, and user interaction.

The project is designed for two human players and provides a seamless text-based interface. The program ensures rule compliance, making it suitable for beginners learning Checkers or as a foundation for more advanced implementations, such as AI-based gameplay.

Features and Highlights

1. **Board Representation**:
    - The game uses an 8x8 matrix to represent the board, where:
        - `0`: An empty cell.
        - `1`: Player 1's normal piece.
        - `2`: Player 2's normal piece.
        - `3`: Player 1's king piece.
        - `4`: Player 2's king piece.
2. **Gameplay Mechanics**:
    - **Player Turns**:
        - Alternating turns are maintained for fair play.
        - Prompts guide players to make valid moves or take required actions.
    - **Movement**:
        - Normal pieces move diagonally forward.
        - Kings, upon promotion, gain the ability to move diagonally in all directions.
    - **Capture Rules**:
        - Players can capture opponent pieces by jumping over them diagonally.
        - Multi-captures are supported and enforced when possible.
    - **Mandatory Capture Enforcement**:
        - If a piece can capture an opponent, the player is required to make that move.

3. **Advanced Features**:
    - **King Promotion**:
        - A normal piece becomes a king when it reaches the opponent's back row.
        - Kings are denoted differently and have enhanced movement capabilities.
    - **Winning Conditions**:
        - A player wins if the opponent has no valid moves or pieces remaining.
    - **Customizable Symbols**:
        - Players choose their own symbols to represent their pieces at the start of the game.
4. **Error Handling**:
    - The program includes robust checks for invalid moves, out-of-bounds actions, and incorrect inputs.
    - Players are guided with detailed error messages and are prompted to retry invalid actions.

Implementation Details

The project consists of several modular components, each handling a specific aspect of the game:

1. **Core Functions**:
    - **`startup()`**: Initializes the game, welcomes players, and sets up the board.
    - **`showboard()`**: Dynamically displays the current state of the board in the console, showing all pieces and their positions.
    - **`convert()`**: Maps numeric board values to the chosen player symbols or king symbols for display.
2. **Movement Validation**:
    - Movement functions like `moveupright`, `moveupleft`, `movedownright`, and `movedownleft` ensure pieces only move according to the rules.
    - Additional functions (`moveupright_king`, `moveupleft_king`, etc.) handle the extended movements of kings.
3. **Capture and Elimination**:
    - Functions like `elimination1` and `elimination2` validate potential captures for each player.
    - The `eliminate()` and `king_eliminate()` functions remove captured pieces and handle multi-capture sequences recursively.
4. **Game Logic**:
    - Functions such as `check1` and `check2` determine if a player must perform a capture during their turn.
    - The `winner1` and `winner2` functions check if a player has won by verifying the absence of opponent pieces or moves.
5. **King Mechanics**:
    - The `checkKing()` function identifies king pieces and enables their special movements and captures.

Challenges Encountered

1. **Rule Enforcement**:
    - Ensuring mandatory captures required recursive logic to account for multi-capture opportunities.
    - Managing different movement rules for normal and king pieces added complexity.
2. **User Input Validation**:
    - Handling invalid coordinates or inputs gracefully was essential to maintain a smooth user experience.
3. **Error Messaging**:
    - Creating clear and intuitive messages for players helped minimize confusion, especially for new users unfamiliar with Checkers.

Strengths and Areas for Improvement

1. **Strengths**:
    - The project adheres strictly to the rules of Checkers, ensuring fair play.
    - The modular design makes the codebase maintainable and extensible.
    - It provides detailed feedback to players for an engaging and error-free experience.
2. **Improvements**:
    - **User Interface**:
        - Adding a graphical user interface (GUI) would enhance visualization and make the game more appealing.
    - **AI Integration**:
        - Introducing AI opponents would allow single-player gameplay.
    - **Online Multiplayer**:
        - Extending the game to support remote multiplayer sessions.
    - **Custom Rules**:
        - Supporting variations in Checkers rules, such as flying kings or alternative board sizes.

Future Directions

- **AI Integration**:
    - Implementing algorithms like Minimax with alpha-beta pruning for a competitive AI opponent.
- **Graphics and Accessibility**:
    - Developing a GUI using frameworks like SFML or SDL.
    - Making the game accessible for mobile and web platforms.

- **Enhanced Features**:
    - Adding a move history, game replay options, and timed turns for a competitive edge.
    - Including a tutorial mode for beginners.

This Checkers project demonstrates the application of algorithmic thinking and modular programming to create an engaging and functional game. Its adherence to the rules of Checkers, coupled with thoughtful user experience design, provides a solid foundation for future enhancements. This project serves as both a fun game and a learning tool for understanding game mechanics and C++ programming.