

Final Project Chess



Department of Software Engineering-FIT-VNU-HCMUS

1

Basic features

Overview

Develop a chess game application in C++ that supports two modes:

1. Player vs Player (on the same device): Two players can play chess on the same device.
2. Player vs AI: A single player competes against an AI opponent with three difficulty levels: easy, medium, and hard.

Detailed Requirements

1. Game Modes:

- 2-Player Mode:
 - Two players can play against each other on the same device.
 - The game should manage turns automatically, alternating between Player 1 (white) and Player 2 (black).
- Player vs AI Mode: The user plays against an AI-controlled opponent with three selectable difficulty levels:
 - *Easy*: The AI makes random or less optimal moves with little to no strategy.
 - *Medium*: The AI uses moderate strategic calculations with some errors.
 - *Hard*: The AI uses advanced strategy, attempting to play with minimal mistakes.

2. User Interface (UI): create a user-friendly interface using a C++ GUI library that includes:

- A chessboard where pieces can be moved.
- Turn indicators showing whether it's white or black's turn.
- Notifications for check, checkmate, or stalemate.
- Buttons to:
 - Start a new game.
 - Choose game mode (2-player or Player vs AI).
 - Select AI difficulty level before starting a Player vs AI game.
 - Reset the current game.
 - Undo the last move.
 - Redo the last undone move.
 - Save the current game state.
 - Load a previously saved game.
 - Access game settings for customization.
 - Board Color: Options to select different colors or themes for the chessboard.
 - Piece Design: Options to change the design or style of the chess pieces (e.g., classic, modern, cartoon).
 - Sound Effects: Options to enable or disable sound effects for moves, check, checkmate, and background music. Provide a selection of different sound themes.

- Background Music: Options to choose background music or disable it. Include a volume control for sound effects and music.
- Exit the game.

3. Chess Board Representation:

- Implement a graphical chessboard using a C++ GUI library such as **SFML**, **SDL**, or **Qt**.
- Players can interact with pieces by selecting, dragging, and dropping them.
- Highlight legal moves when a player selects a piece.
- Visual feedback should be provided for check, checkmate, and stalemate situations.
- In **2-player mode**, alternate between Player 1 (white) and Player 2 (black) automatically after each move.
- In **Player vs AI mode**, alternate between the player's move and the AI's move. The AI should make a move immediately after the player completes theirs.

4. Game Logic:

- Implement the rules of chess, including:
 - Movement for all pieces (pawns, knights, bishops, rooks, queens, and kings).
 - Special moves like castling, en passant, and pawn promotion.
 - Detection of check, checkmate, stalemate, and draw.
- Ensure that only valid moves are allowed.
- Automatically detect and end the game with a notification in case of checkmate or stalemate.
- Game rules: <https://www.chess.com/learn-how-to-play-chess>

5. AI Opponent:

- Implement three levels of AI difficulty:
 - *Easy*: The AI makes quick, random moves without much evaluation.
 - *Medium*: The AI uses the Minimax algorithm with a shallow depth, making reasonable moves with occasional mistakes.
 - *Hard*: The AI uses Minimax with Alpha-Beta pruning, evaluating a deeper move tree to play strategically, reducing mistakes.
- AI should make moves efficiently and within a reasonable time for all difficulty levels.

6. Game State Management:

- Track the current state of the board and the game.
- Handle turn management, ensuring proper transitions between the player's turn and the AI's turn or between two human players.
- Implement an Undo feature that allows players to revert the last move:
 - Store a history stack of moves (using a suitable data structure) to facilitate undo operations.
- Implement a Redo feature to reapply the last undone move:
 - Use a second stack to manage redone moves.
- Provide the ability to reset the game to its initial state at any time.

7. Save and Load Functionality:

- Implement the ability to save the current game state to a file: save the positions of all pieces, the current player's turn, and any other relevant game state information.
- Implement the ability to load a previously saved game: read from the saved file and restore the game state to the last saved point, allowing players to continue from where they left off.

8. Optional Features:

- Display a move history in standard algebraic chess notation.
- Add piece animations for a smoother user experience.
- Implement sound effects for piece moves, check, and checkmate.
- Ensure the chessboard and game interface are responsive and adaptable to different screen resolutions.
- The game should handle user inputs smoothly via mouse or keyboard (depending on the platform).

9. Technology Stack:

- C++ for Core Logic: Write all game logic, AI algorithms, and game state management in C++.
- AI Algorithms: Implement AI using **Minimax** with **Alpha-Beta pruning** to handle decision-making for medium and hard difficulty levels.
- GUI Library: Use **SFML**, **SDL**, or **Qt** to build the user interface, render the chessboard, and manage input events.
- Save Game:
 - Serialize the current game state, including piece positions, turn information, and any other necessary data.
 - Write this data to a file in a format such as JSON, XML, or a simple text format.
- Load Game:
 - Open a file dialog to allow the user to select a saved game file.
 - Deserialize the data from the file and restore the game state accordingly, updating the chessboard and any other relevant UI elements.

2

Submission

When submitting your chess game project, it is essential to provide a well-organized zip file that includes all necessary components. Below are the detailed contents that should be included in the submission:

1. **Source Code:** create a structured folder hierarchy for your source code. A suggested structure might be:

```

ChessGame/
├── src/                                # Main source code files
│   ├── main.cpp                       # Entry point of the application
│   ├── Game.cpp                       # Game logic implementation
│   ├── Board.cpp                      # Chessboard representation and logic
│   ├── AI.cpp                         # AI opponent logic
│   ├── UI.cpp                         # User Interface implementation
│   └── ...                            # Other necessary files
├── include/                           # Header files
│   ├── Game.h                        # Header for game logic
│   ├── Board.h                       # Header for chessboard representation
│   ├── AI.h                          # Header for AI logic
│   ├── UI.h                          # Header for UI components
│   └── ...                            # Other header files
├── assets/                            # Asset files (images, sounds)
│   ├── pieces/                       # Chess piece images
│   └── sounds/                       # Sound effects and music files
└── README.md                         # Brief description and instructions

```

2. **Report Document**

- **Title Page:** Title of the project, team members, and date of submission.
- **Project Overview:** A brief summary of the project, including objectives and goals.
- **Requirements Specification:** Detailed description of the game requirements, including all features implemented (as outlined in previous messages).
- **Design Document:** High-level architecture and design decisions. Include diagrams if necessary (e.g., UML diagrams for classes).
- **Implementation Details:** Brief description of how the code is organized, any specific libraries used, and major components of the codebase.
- **Testing:** Description of testing strategies and results. Include any known issues or limitations.
- **Future Improvements:** Suggestions for potential future work or enhancements to the game.

3. Work Division: create a document that outlines how work was divided among team members. This should include:
 - **Team Members:** List all members involved in the project.
 - **Responsibilities:** Describe the specific tasks each member was responsible for. For example:
 - Member A: Game Logic Implementation (Game.cpp, Board.cpp)
 - Member B: AI Development (AI.cpp)
 - Member C: User Interface (UI.cpp, assets)
 - Member D: Testing and Documentation
 - **Collaboration Tools Used:** Mention any tools or platforms used for collaboration (e.g., GitHub, Trello, Slack).
4. Submission Instructions
 1. **Create a Zip File:** Compress the entire project folder (e.g., `StudentId1_StudentId2.zip`).
 2. **Double-Check Contents:** Ensure that all required files are included in the zip file.
 3. **Upload:** Submit the zip file according to the specified submission guidelines provided by your instructor or project supervisor.