CS246 - Fall 2022 - Final Project - Abdullah, Ariq and Soham

Question: Chess programs usually come with a book of standard opening move sequences, which list accepted opening moves and responses to opponents' moves, for the first dozen or so moves of the game. Although you are not required to support this, discuss how you would implement a book of standard openings if required.

Standard openings and chess games are often encoded in Portable Game Notation (PGN). There are several opening books available online that go through all the potential lines (different variations) of a particular opening. The LiChess database in fact provides access to millions of user games as well as a collection of grandmaster games; this can be used to build a list of openings and their variations.

We can add a PGN parser that compares the current moves played with that of existing openings and lines. This would allow us to add features such as showing what openings have been played (intentionally or not), and providing computer players with what moves to play next by seeing which opening and line they are playing.

We can even have a repository of opening moves that users can read through outside of the game to learn openings and improve their skills.

Question: How would you implement a feature that would allow a player to undo their last move? What about an unlimited number of undos?

Our implementation of an undo function relies on board states and a stack. Every time a move is made, we create a FEN string which is a standardized format to represent the board state and push it to stack. The undo function simply pops the items on the stack and parses the FEN string to rebuild the board to its previous state.

We can even implement a redo function that stores the game states that are popped off the undo stack and provided that no new moves are made, we can use it to easily redo the moves that were undone. If the user makes new moves after undoing then we simply clear the redo stack to prevent any conflicts.

Question: Variations on chess abound. For example, four-handed chess is a variant that is played by four players (search for it!). Outline the changes that would be necessary to make your program into a four-handed chess game.

The biggest change would be to the board class and the logic when it came to checking legal moves and attacked squares. We would have to redefine the arrays used to generate the board to accommodate a non-rectangular shape in the board class and create new player classes to accommodate four players.

The types of pieces are unchanged, so the methods for calculating legal moves, moving the pieces, and attacked squares will need to be updated for the four player format.

The text and graphics renderers will have to be updated to properly render the 4-player board, and incremental changes will have to be made for the main interface to show statistics for the four players.

CS246 - Fall 2022 - Final Project - Abdullah, Ariq and Soham

Plan of Attack:

Writing declarations for all classes and their interactions - Soham, Nov. 26th

- This will help us ensure our UML is correct and that all appropriate methods have been made.

Implementation of the Board Class - Soham, Nov. 26th

- A working model of the board will be needed to implement basic testing for the pieces and renderer.

Implementation of the Text Renderer - Abdullah, Nov. 26th

- A basic text renderer will be needed for the core functionality of the program and further testing for methods and functions for other classes

Implementation of Peace Classes - Ariq, Nov 29th

- This will include creating classes representing each of the six pieces and implementing the rules for piece movement in the legal moves methods including edge cases as en passant and castling.
- The constructors will be implemented for pawn promotions and destructors for captures

Implementation of the Game Class - Soham, Nov 29th

- Completion of this Game wrapper class will allow us to test basic chess games and ensure functionality of the piece and board classes.

Implementation of Human Players - Ariq, Nov 30th

- This will include implementing a basic two-player functionality to test implementations of the board and piece classes.

Implementation of Computer Players and Basic AI - Abdullah, Soham, Ariq, Dec 3rd

- This will include setting up the first 3 levels of computer player difficulty. Computer vs Computer matches will be used to test functionality

Implementation of the Graphical Renderer - Soham, Dec 5th

- Creating the graphical renderer in XWindows and having a GUI for the chess board.

Implementation of Level 4 AI - Abdullah, Dec 5th

- Creating a more sophisticated AI using methods such as a MinMaxing algorithm and Quiescence function

CS246 - Fall 2022 - Final Project - Abdullah, Ariq and Soham

Notes

AI:

- A **MinMax search** algorithm that evaluates the board material state to a certain depth
    - **Quiscence function** improves decision making by reducing possible moves so that no captures are available after that depth
- Adding **Alpha-Beta pruning** to the MinMax function optimizes the search by disregarding certain branches
- Implement a **heuristic move-ordering** function that increases the weightings of moves that use low-value pieces to capture high-value pieces, moves that promote pawns, and decreases the weightings of moves that get killed by enemy pawns
- Adding a function which forces the king to corner towards the end
    - Adding weights to situations where the king is further from the center of the board
    - Increase in weighing as the opponent has fewer pieces remaining
- **Transposition table**
    - Store game states using **Zobrist Hashing** to avoid reevaluating previously explore positions
- Implementing a **heat-map** (matrix) which incentivizes pieces to play opening moves that will benefit them later on

Results:

- Implementing Alpha-Beta pruning reduces evaluated moves by **87%** and the time taken to evaluate the moves by **85%**
- Implementing the move-ordering function on top of the Alpha-Beta pruning reduces the total evaluated moves by **99.86%** and the time taken to evaluate the moves by **97.84%**