

Game Specification (Chess Engine)

Individually

This chess engine is implemented by one person. I have experience in programming in C, C++, Java and C#. I started my journey with chess programming 2 years ago, when I wrote my first engine using the “mailbox” implementation.

Description

A chess bot, which plays chess against the player. There are black and white pieces. Each player has 8 pawns, 2 knights, 2 bishops, 2 rooks, 1 queen and a king. The main goal of the game is to attack the opponent's king and trap it so it has no possible moves. The game might also end in a draw when the player has no moves and their king is not in check or the same position will appear 3 times during one game.

OS requirements

There is no requirement regarding the OS. However, the terminal used for launching the game should support ANSI escape codes. Linux terminal works fine.

Outside libraries

No outside libraries. Only C functions executed in assembly.

Interface

Chessboard is displayed on the terminal and is updated every move. It uses ANSI escape codes to prettify and overwrite the terminal printing the new state of the board again. The chess pieces are displayed as unicode characters. We highly recommend zooming in since the board might appear very small :).

User input

The engine will use modified Universal Chess Interface (UCI) for more pleasant interaction between the engine and the player. Here are the commands supported by the chess engine:

- *position startpos* - sets the current position to the starting position of the game of chess
- *position startpos moves e2e4 e7e5 ...* - also could be used to set the board after specifying moves from the beginning
- *move [source square target square]* - moves a piece from source square to target square ex. *move e2e4*.
- *flip* - flips the board. The default board is set for the player to play with white pieces and the chess engine to play with black pieces
- *quit* - stops the program
- *go [depth x]* - used to evaluate the position and give the best move according to the chess engine. If no depth is provided (the command is just “go”) then the default depth is 6. ex. *go depth 4*

Data

There are multiple ways to store the position, however the chess engine uses bitboards (64-bit variables), since it is both very efficient and easy to operate on. Moves are encoded in a single integer using bitwise operations. Such a move consists of a source square (a number from 0 to 63 - 6 bits), a target square (a number from 0 to 63 - 6 bits), a piece, which is moved (a number from 0 to 11 - 4 bits), a promoted piece (a number from 0 to 12 [if 12 than no promotion] - 4 bits), a capture flag (1 bit), a double pawn push flag (1 bit), a enpassant capture flag (1 bit) and a castling flag (1 bit). Encoded move consists of 24 bits in summary.

Move generation

The engine is using attack masks (for leaping pieces) and magic bitboards (for sliding pieces). Masks are a collection of bitboards for a given piece with possible attacks on a given square. Magic bitboards is a technique used to efficiently generate and store information about moves for sliding pieces (like rooks, bishops, and queens).

Evaluation

The chess engine not only evaluates the position by the difference in material (pieces), but also scores the position of each piece on the board. This will ensure every piece is active and uses its potential.

Search

For searching and picking the best moves the chess bot uses negamax alpha beta search algorithm and quiescence search. The negamax uses two values, alpha and beta, representing the minimum score that the maximizing player is assured of and the maximum score that the minimizing player is assured of respectively. After the negamax search, the chessengine performs a more limited quiescence search, containing fewer moves. The purpose of this search is to only evaluate "quiet" positions, or positions where there are no winning tactical moves.