Technical Specifications for Odyssey

Harry Zhou

My chess engine is better than me in chess. Should I be happy or sad?

Overview

Chess engines can be traced back to the 90s, when IBM's Deep Blue defeated Garry Kasparov, the reigning world champion; with the advancements of modern technology, more chess engines like Stockfish and Alphazero are developed using neural networks, and they can easily defeat any human player.

Naturally, as both a chess player and computer science enthusiast, I was curious of the underlying mechanisms behind those intelligent machines, and decided to create an engine of my own - Odyssey. This was an ambitious goal, given that most of my previous computer science background was focused on competitive programming rather than project development. But I was driven by passion and curiosity, and through relentlessly researching, coming up with new ideas, coding, and debugging, I'm super happy to announce that my chess engine, Odyssey, is estimated to be stronger than myself at chess!

Currently, Odyssey is estimated to have a ELO of 1870 (in comparison, my official rating is around 1400). It has undoubtedly fulfilled my original goal, but I've grown more ambitious, so I will address some potential improvements and plan moving forward in the document. Odyssey is mainly composed of four essential parts: board representation, move generation, evaluation function, and searching/pruning algorithms. The rest of the document will focus on those four parts, and a detailed credits section is at the end.

Board Representation

- Every chess engine needs a board representation to maintain chess positions for move generation, evaluation, and searching
- The most obvious way to maintain such a representation is to use an 8x8 2-D array. While this approach is intuitive, it's inefficient in generating moves later on
- Instead, Odyssey uses **bitboard representation**. Since a chess board has 64 squares, each square can be mapped to a single bit in a `long long` type in C++. For each piece in chess (pawn, knight, bishop, rook, queen, and king), Odyssey maintains such a `Bitboard` type, and for every square

that the piece is on, the corresponding bit would be `1`, and otherwise it'd be `0`. (For example, if one player's queen is on the square A4 and no other square, the queen bitboard would be $2^{(4-1)} = 8$)
- The benefit of bitboard representation is that we can take advantage of fast binary operation. Consider the following case: we want to find out what squares on the board are occupied. With bitboard representation, we don't have to go through each of the 64 squares; instead, we can simply call `occpuied = pawn | knight | bishop | rook | queen | king`, where `|` is the `or` operation in C++
- Odyssey can also read in a position using FEN (Forsyth–Edwards Notation, the standard way of expressing a position)

Move Generation

- Given a position, move generation is the process of generating all legal moves. This step has to be extremely robust and make sure there are no missed/extra moves being generated
- Just like board representation, binary operations are taken advantage of to make move generation faster. Odyssey uses a technique called magic bitboards to further speed up the process, especially when it's generating moves for bishops, rooks, or queens (which can be blocked by the placement of other pieces). Essentially, magic bitboards trade some space with a much better time performance
- There are a lot of border cases need to be taken care of as well. There are many "exceptions" in the rule of chess: en passsant (for pawns), castling (for rooks and kings), stalemate etc. Odyssey is tested using "perft", in order to ensure performance and robustness
- Odyssey uses libchess, an open source library to generate moves. See credits section for more information

Credits

- libchess
  - ◇ libchess is an open source "C++17 library that [provides] legal move generation"
  - ◇ Originally, I planned to write move generation myself too (see Ophelia, the predecessor of Odyssey). Well, that did not end up well... When I was done with writing code and started testing, I couldn't for the life of me figure out why Ophelia couldn't generate all moves (unit-testing and debugging are indeed two areas that I need to improve on!). Eventually, I realized that there's no point for reinventing the wheels, and I should spend more time on the evaluation function and searching algorithms
  - ◇ When I landed on libchess, I was very happy to see that its code is clean and comprehensible, which allows me to make changes and adjustments easily. Its fast and robust performance provides a solid

starting point for Odyssey

- BBC (Bit Board Chess)
  - ◇ BBC is an open source chess engine written by Code Monkey King
  - ◇ BBC was a particularly helpful reference when I was researching effective searching/pruning algorithm
- The Chess Programming Wiki (CPW)
  - ◇ Fabulous wiki on almost every topic on chess programming
  - ◇ I used CPW as a starting point for researching evaluation functions and searching algorithms
- Miscellaneous
  - ◇ CMake for generating Makefiles
  - ◇ Git for version control
  - ◇ Stackoverflow and various C++ forums (how can I not mention them?!) when I came across syntax problems