Daybook:

Use prediction machine learning to guess whether an opponent is likely to play a move, given the eval of difference of the best moves and the previous move for all previous moves. Then use this history of eval pre move, post move and eval potential triplets to determine whether the current move in search would be likely. Can use non conformal predictors.

This tests how likely an opponent is to play a given move based on their previous moves and opportunities missed

Move 1 of opponent: (+0.0, -0.1, +0.1)

Move 2 of opponent: (+x, +y, + z)

Current move: (+x, current move in search of opponents, best move in search of opponents)

Current pre move eval should be the eval of the best move?

SPRT EVERY functional change, to see what you changed that’s working and what isn’t.

1. Produce a report including SPRTs for each new feature and compare it to other engines to see if these ideas have promise. Can compare to other engines using cute chess. In the report, highlight example moves where the new approach has taken place. For example, the engine knows move x is best, but using the new algorithm, it decides on move y, and why it has decided to choose that move, for example it calculated that it is losing, and it has met the threshold to play a “bold” move.

Use doxygen to generate documentation

Would be cool if I could make a prediction search which guesses what move they may make

Maybe could add all outcome evals of moves and go for the highest sum. Means that maybe best move for black for highest sum would lead to worse than best move for other. But could be helpful for risks.

Some core concepts here are that:

* The opponent is less likely to catch complex lines
* They will see lines at a lower depth than the engine. (human may pick a line that looks good at depth 5, but engine realizes at depth 6 that line is bad, so they allow the human to take that line)
* Humans are great at concepts and long term planning, but will not see move by move their ideas through. Whereas an engine sees all combinations up to a given depth. This suggests that engines should prefer short and complex positions or long and simple positions. For example, the game of chess is already solved with 7 pieces meaning that we can make use of endgame tables. Tables may not be exactly helpful for various reasons, one being that theyre too slow: http://horizonchess.com/FAQ/Winboard/weaktablebase.html

Also very large, but with 5 pieces its small <https://kirill-kryukov.com/chess/nulp/results.html>

* Humans have memory and experience, meaning there are calculations it can simply skip, allowing more time to be spent thinking about specific moves in a higher depth. This is especially prevalent in the opening. Thankfully to counter this we provide the engine with this knowledge by using opening books.
* Humans are weaker than engines
* Humans are worse than engines at figuring out complex positions (to determine complexity, a baseline for this would be number of possible depth 1 nodes, and number of pieces on the board)
* Humans are better at seeing broadly and having a general better gameplan, which they leverage by playing unprovoking moves and aim to slowly build up a dominant position before the engine can figure out it is losing. (This should be countered if an engine is actively trying to pick up the pace of the game).
* <https://chessify.me/blog/what-is-chess-engine#:~:text=When%20it%20comes%20to%20opening,what%20the%20engine%20can%20do>.

The following diagram displays some core concepts we hope to implement in the decision-making process for Gambit.

A high-risk factor suggests “disrespect”, meaning it is more willing to accept a lower risk to reward (compared to the **safe move**) ratio. This means that it will accept higher risks for lower rewards on average than if it had

Safe Move: the objectively best move in a position deemed by alpha beta

Risky move: the objectively best move in a position deemed by Gambit, based on core concepts such as human fallibility and engine strength.

A screenshot of a phone

Description automatically generated

Figure 1:

Typically, with a move decision algorithm such as **minimax**, White (the engine) sees that if black plays perfectly, the line NC3, NF5 leads to a draw. Whereas E4, C4 leads to a 0.1 advantage.

However, perhaps given Black’s **Elo**, it can calculate a minimum threshold of risk, where if the risk is within this threshold, it would choose bolder moves as it would anticipate the opponent not finding the stifling move. In this example, this would look like the engine choosing NC3, expecting D4 to be played rather than NF5. It is willing to take this risk as at a very simplified look, the difference between the best and worst position of NC3 is higher than E4, therefore NC3 seems more promising. If the player’s **Elo** is very low, then NC3 would be picked, whereas if it were higher,

Mostly developing a new algorithm for choosing which move to play, based on already well constructed algorithms to search and evaluate.

With simple conditions within the evaluation function, we can easily toggle on and off the Gambit features, such that the player can choose whether to play versus the Gambit style engine or play it like any other chess engine would play.

[5] U. Schwalbe and P. Walker, “Zermelo and the Early History of Game Theory,” *Games and Economic Behavior*, vol. 34, no. 1, pp. 123–137, Jan. 2001, doi: https://doi.org/10.1006/game.2000.0794.

PERFT (Determine whether all expected move gen is accurate), and SPRT helps give a baseline of performance compared to future iterations of the project to see if additions help improve it), SPRT may have to be done after completion of search and eval as it needs to compare version 1 to version 2, although could compare it to other random movers. SPRT random mover vs various searches and evals

Such factors include: evaluation of the current position (whether the engine is winning or losing and by how much), strength of the opponent (which will limit the risk factor), complexity of the foreseen tactic / line. We assume that if the engine is in a winning position, it is able to take more risks. Zermelo’s theorem would suggest that “if a player is in a winning position, then he can always force a win no matter what strategy the other player may employ [5].” Whilst this is the case when assuming perfect information, chess is an unsolved game and therefore perfect information is never available. Furthermore, humans are fallible and often can be deceived by complex tactics, bluffs and gambits. Therefore, we suggest that an engine should not make modest moves that are objectively the best, but should instead make more bold and ambitious moves when playing versus a human.

Typical chess engine programming standards would suggest that if an engine is stronger than its opponent, it should make use of the large contempt factor. The aim of this would be to give weight to moves that avoid draws, as a draw would be seen as a negative outcome for the better player [6]. In other words, the engine will accept a worse position in the short term to avoid a draw if it believes in the long run it can get a win. However, contempt has shown limited impact on effecting the outcome of the game. It does however on average increase the number of short wins.

An engine using the contempt factor will accept a slightly worse position if the engine thinks it will win given a further drawn out game by avoiding the draw. But it would never make what effectively is a blunder, in hopes that if the opponent doesn’t find the line, which results in a position that would be better than if the engine made the technically best move originally. Therefore, we propose using a risk factor. If an engine is in a winning position, its confidence to take risks can increase as it has some advantage to fall back on. Furthermore, if an engine is in a losing position, it knows that it should take some risks to try and regain a winning position. Therefore, it could be argued that it is irrelevant the position, as there is always reason to go for risky moves. The question is, what level of risk is acceptable and what factors should impact this threshold?

Risk factor: the level of risk an engine is willing to accept when seeking a better position than the alpha beta sequence. Calculated in a formula, which takes into account the likelihood of the opponent finding the “counter move”, the eval of this position compared to the alpha beta sequence, and more. Risk would be determined by the likelihood of the opponent finding the counter move (based on an ongoing opponent model built off their skill and takes into account position complexity and tactical complexity patterns), the evaluation of the position after this move. Reward would be determined by the evaluation of the ideal scenario compared to the alpha beta

UCI:

A screenshot of a computer

Description automatically generated

<https://github.com/simpleguy747/MollyChessBot/blob/initial-commit-fen-parser/src/attacks.c>

<https://github.com/simpleguy747/MollyChessBot/blob/initial-commit-fen-parser/src/gen_knight_moves.c>

<https://www.rustic-chess.org/introduction/design.html>

Reading and Resources:

Chess Engine Programming Standards

* Standard Chess Programming Principles and Practices: <https://www.chessprogramming.org/Main_Page>
* The UCI Protocol: <https://backscattering.de/chess/uci/>

Forums

* Talk Chess A forum for chess talk, including engine programming: <https://talkchess.com/> <https://discord.gg/mJuKQbrEmN>
* Discord Server for Chess Engine Development: <https://discord.gg/uTDs7U5Vh7>

Engine Evaluation:

* Comparing Strength with Other Engines: <https://www.chessprogramming.org/Cutechess-cli>
* Engine used to determine strength: <https://www.chessprogramming.org/Stash>
* Engine Rating List: <https://computerchess.org.uk/ccrl/4040>
* Open Source Engine High Standard Engine: <https://github.com/AndyGrant/Ethereal>
* Open Bench for Evaluation: <https://github.com/AndyGrant/OpenBench>

Perft

* Move generation accuracy is verified using known numbers of legal moves from a given position.
* Information: <https://www.chessprogramming.org/Perft>
* Online Perft Interface: <https://analog-hors.github.io/webperft/>
* More Info on Test Results: <http://www.rocechess.ch/perft.html>

General Resources

* Pre-existing Open Source Engines: <https://github.com/EngineProgramming/engine-list>
* A Survey of Monte Carlo Tree Search Methods <https://www.researchgate.net/publication/235985858_A_Survey_of_Monte_Carlo_Tree_Search_Methods>
* Opening Books: <https://github.com/official-stockfish/books>

<https://www.reddit.com/r/chess/comments/18f44j0/opening_book_based_on_leela_alphazero_policy_data/>