RAM-Z: An Optimized Classical Chess Agent using Negamax and Alpha-Beta Pruning

Ramzi Al-Sharawi
Minnesota Robotics Institute
University of Minnesota, Twin Cities
Minneapolis, United States
alsha192@umn.edu

***Abstract*—While modern chess engines increasingly rely on resource-intensive deep reinforcement learning, classical search algorithms remain fundamental for developing lightweight and explainable artificial intelligence. This paper presents RAM-Z, an optimized chess agent designed to maximize tactical proficiency on standard hardware using a Negamax framework enhanced with alpha-beta pruning. To navigate around the exponential complexity of a chess game tree, the agent integrates a suite of pruning and ordering techniques, including iterative deepening, transposition tables, null-move pruning, and a four-stage move-reordering hierarchy. Furthermore, the horizon effect is addressed with quiescence search utilizing delta-pruning and capture ordering heuristics. Experimental performance benchmarks against Stockfish demonstrate that RAM-Z achieves an approximate playing strength of 1500 ELO, securing a 58% win rate against 1400-ELO Stockfish and maintaining competitive parity with 1500-ELO Stockfish. These results validate the effectiveness of combining classical heuristics to achieve competent play without the computational overhead of neural networks.**

***Index Terms*—Negamax Search, PeSTO Evaluation, Quiescence Search, Null-Move Pruning, Piece-Square Tables, Transposition Tables, Opening Books, Move-Reordering**

I. Introduction

The use of artificial intelligence (AI) in chess has been a subject of intense research since the 1950s, when Alan Turing was the first to develop a theoretical chess-playing program in 1951. Ever since then, progress in AI-based chess agents has advanced significantly, with modern data-driven chess engines including AlphaZero [1] and its open-source alternative, Leela Chess Zero. Such modern chess engines typically integrate deep reinforcement learning techniques to enhance decision-making within Monte-Carlo Tree Search (MCTS) [2]. By leveraging an abundance of chess data and top-tier computational resources, such modern chess engines have achieved superhuman chess abilities, transforming how the game is studied and analyzed today.

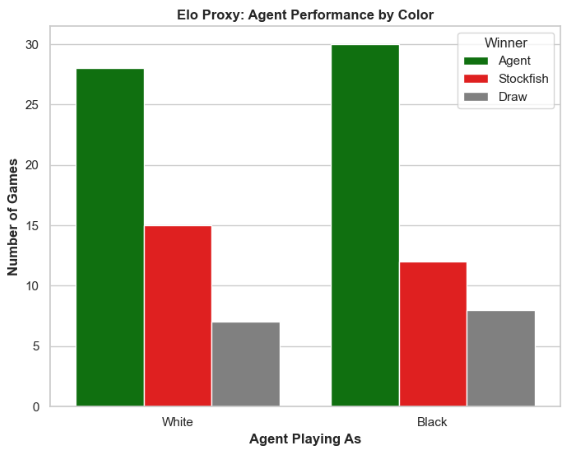
Despite the strength of modern chess engines like AlphaZero and Leela Chess Zero, they require intense computational resources and time for self-play training, often requiring TPUs/GPUs to play out millions of games. To address this computational bottleneck, this paper proposes RAM-Z: an optimized chess agent built upon classical AI methods. Through the proposed RAM-Z agent, this paper aims to benchmark the agent's tactical limits on standard hardware, offering a lightweight alternative to the resource-intensive data-driven pipelines of modern neural-network-based engines. The paper's contributions are listed below:

- Utilizes a **Negamax** search framework enhanced with **alpha-beta pruning** and **iterative deepening** to navigate the state space efficiently by framing the game as a minimax problem. **Quiescence search** is used to circumvent the horizon effect caused by RAM-Z's depth-limited search approach.
- Implements a **null-move heuristic** [3] and move-reordering using a "most valuable victim-least valuable attacker" **(MVV-LVA)** scheme to maximize the effectiveness of the alpha-beta pruning process.
- Integrates a **PeSTO-based evaluation function** that integrates piece value based on game phase and position for a comprehensive overview of the position.
- Integrate **opening books** and **transposition tables** to convert search problems into faster lookup problems whenever possible, thus increasing RAM-Z's speed of response.
- Develops a **Python-based chess GUI** using Pygame, allowing the user to play against the developed RAM-Z agent.
- Benchmarks RAM-Z's tactical limits by running a series of test games against **Stockfish at various ELO ratings**.

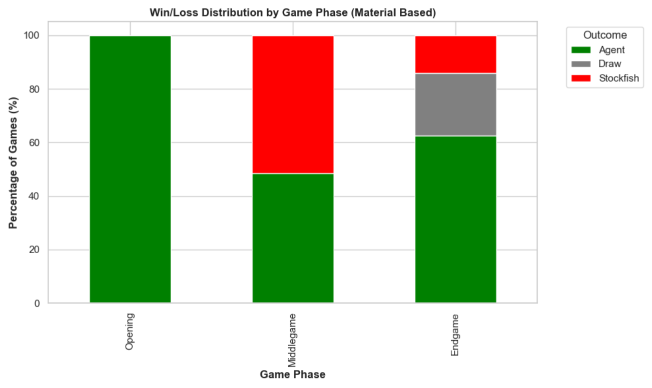
The rest of this paper is structured as follows: Section II highlights the different techniques integrated into the agent, the software used, and instructions on running the code. Background on the utilized techniques and websites used to research them can be found in Section V-A of the Appendix. The performances of the agent against a RandomPlayer agent and Stockfish at ELO ratings of 1400, 1500, and 1600 are presented and discussed in Section III. The paper concludes with final remarks in Section IV.

II. Methodology

The RAM-Z chess engine and its associated code-base are implemented in **Python 3.10.10**, chosen for its readability and extensive library support. The core game mechanics are handled by the `python-chess` library, which provides a framework for move generation, move validation, and board representation. To aid with user-interaction, the engine is coupled with a custom Graphical User Interface (GUI) developed using the **Pygame** library, displaying the chess board and allowing the user to play against all agents investigated in this paper. To ensure quick and robust play in the opening, RAM-Z employs the **gm2600.bin** opening book from [4]. All source



(a) Win Counts Based on Starting Color



(b) Percentage of Wins Based on Game Phase

Fig. 1: Analysis of agent against 1400-ELO Stockfish

code, dependencies, and execution instructions are publicly available on the GitHub repository at https://github.com/RamziSharawi/CSCI-5511-Project-Negamax-Chess-Agent. The project is fully reproducible using standard Python tooling without specialized hardware.

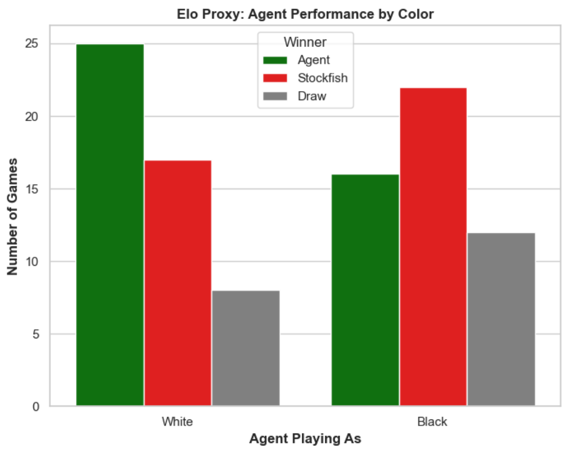
III. Results & Discussion

To evaluate RAM-Z's ELO performance, it underwent a series of games against different agents. Namely, it played against a RandomPlayer agent (plays moves at random) and Stockfish limited to ELOs of 1400, 1500, and 1600. For a fair comparison of RAM-Z's classical evaluation approach, and given computational limitations, both Stockfish and RAM-Z's search processes were limited to a depth of 4 and a maximum search time of 5 seconds per move. A series of 100 games, alternating between RAM-Z starting as White and starting as Black, was played against the aforementioned agents. The results of this experiment are reported in Table I below.

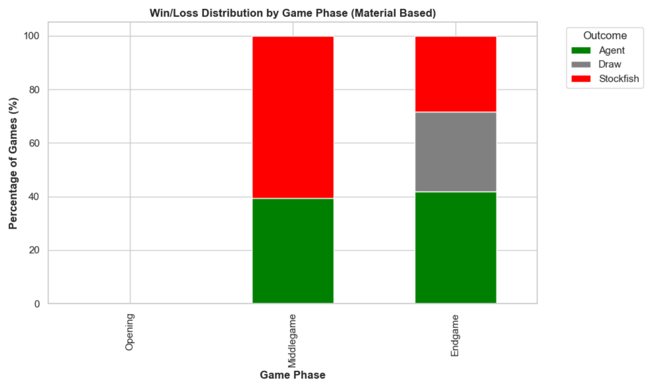
TABLE I: Performance of RAM-Z Agent against various opponents (100 games per match)

Opponent	Agent Win %	Opponent Win %	Draw %
RandomAgent	95%	0%	5%
Stockfish (Elo 1400)	58%	27%	15%
Stockfish (Elo 1500)	41%	39%	20%
Stockfish (Elo 1600)	32%	51%	17%

As highlighted in Table I, RAM-Z does exceptionally well against RandomAgent, indicating the inherent strategy within its search process. Furthermore, it bests ELO-1400 Stockfish across the 100 games, with a 58% win percentage, indicating its slight superiority. The games against 1500-ELO Stockfish were more balanced, with RAM-Z winning 41% of the time and losing 39% of the time. Finally, 1600-ELO Stockfish bests RAM-Z across the 100 games they played, with RAM-Z losing

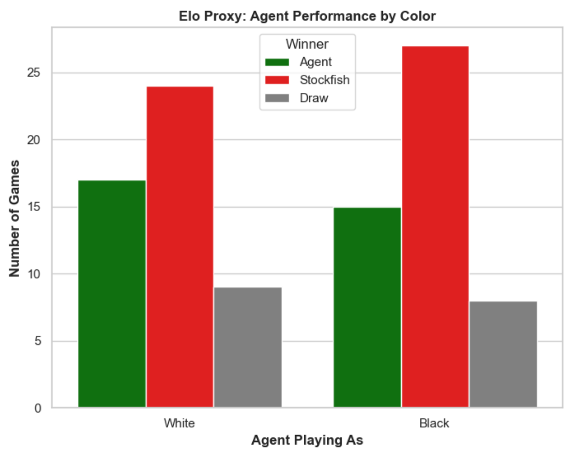


(a) Win Counts Based on Starting Color

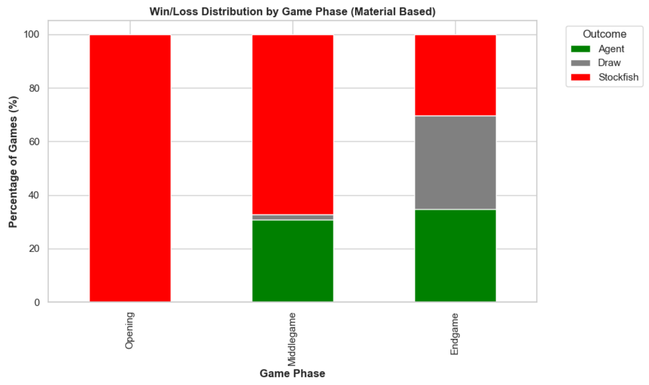


(b) Percentage of Wins Based on Game Phase

Fig. 2: Analysis of agent against 1500-ELO Stockfish



(a) Win Counts Based on Starting Color



(b) Percentage of Wins Based on Game Phase

Fig. 3: Analysis of agent against 1600-ELO Stockfish

51% of the games. Thus, based on these experiments, it seems that RAM-Z's ELO falls around the 1500 range.

To further analyze RAM-Z's performance against Stockfish, two different plots were generated to investigate win rates based on starting color and game phase. These plots are shown in Figures 1 - 3.

As shown in Figures 1a - 3a, RAM-Z dominates as both White and Black facing off against 1400-ELO Stockfish, with the opposite true in the case of 1600-ELO Stockfish. When facing off against 1500-ELO Stockfish, both players seem to edge out the other when playing as White, which aligns with the traditional interpretation that the player starting as White has a +0.7 score advantage. The results of Figure 2a seem to indicate a balanced showing, hinting that RAM-Z's ELO may fall around the 1500 range.

Figures 1b - 3b seem to highlight that Stockfish's NNUE evaluation function bests RAM-Z's hand-crafted evaluation function, as evidenced by Stockfish's higher win percentage in middlegames. On the other hand, RAM-Z seems to slightly edge out all versions of Stockfish in endgames. This seems to indicate that the longer the game goes on, the higher the chance of RAM-Z winning. No games were decided in the opening against 1500-ELO Stockfish. However, RAM-Z dominated opening-based wins against 1400-ELO Stockfish, whereas 1600-ELO Stockfish dominated opening-based wins against RAM-Z. Overall, these plots further highlight the balanced show-off between 1500-ELO Stockfish and RAM-Z, reinforcing RAM-Z's probable ELO-rating as being around 1500.

IV. Conclusion & Future Work

This paper presented RAM-Z, a classical chess agent optimized for performance on standard hardware without the reliance on resource-intensive deep learning pipelines. By integrating a robust Negamax framework with advanced search-optimization techniques like iterative deepening, null-move pruning, and quiescence search, the agent successfully navigates the complex search space of chess with high efficiency. Experimental results demonstrate that RAM-Z achieves an approximate playing strength of 1500 ELO, maintaining a positive win rate against 1400-ELO Stockfish and achieving parity with 1500-ELO Stockfish. These findings validate that classical search optimizations remain a viable and lightweight approach for developing competent tactical agents.

Future work could look towards further narrowing the gap between RAM-Z and superhuman engines. For example, the current PeSTO-based handcrafted evaluation function could be replaced or augmented with NNUE (Efficiently Updatable Neural Networks), allowing the agent to capture positional nuances that static tables miss, while maintaining high evaluation speeds. Finally, integrating Syzygy endgame tablebases would ensure perfect play in simplified endgames, eliminating calculation errors in theoretically solved positions. Overall, the current version of RAM-Z presents itself as a solid baseline to build upon, demonstrating that carefully engineered classical search techniques can still yield competitive performance in modern chess environments.

References

[1] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," 2017. [Online]. Available: https://arxiv.org/abs/1712.01815
[2] D. Klein, "Neural networks for chess," 2022. [Online]. Available: https://arxiv.org/abs/2209.01506
[3] O. David-Tabibi and N. S. Netanyahu, "Verified null-move pruning," 2008. [Online]. Available: https://arxiv.org/abs/0808.1125
[4] D. Stoll, "Chess opening prep," https://github.com/DannyStoll1/chess-opening-prep, 2022.
[5] "Main page," Chess Programming Wiki, Chessprogramming.org, online; accessed Dec. 13, 2025. [Online]. Available: https://www.chessprogramming.org/Main_Page

V. Appendix

A. *Background on RAM-Z Optimization Techniques*

To tackle the high-branching factor of searching throughout the chess state space, the agent problem was divided into three main parts. Firstly, given chess's nature as a minimax player game, a Negamax search algorithm with alpha-beta pruning was implemented to efficiently navigate the search space by pruning out unnecessary branches in the search tree. Next, opening databases and transposition tables were utilized whenever possible to look up the best move in a position as opposed to searching for it through the state space, further reducing computational time. Finally, techniques like iterative deepening, move-reordering, null-move pruning, and quiescence search were integrated to further optimize the pruning process while resisting the horizon effect caused by depth-limited searches. All research around these techniques and their implementations was performed through [5]. These techniques are discussed in more detail below.

1) Negamax Search & Alpha-Beta Search: To find the best move in a position, the RAM-Z agent employs a **Negamax search** framework. While standard Minimax typically considers two players (*Max* and *Min*) to distinguish between "maximizing" and "minimizing" layers of the search tree, Negamax simplifies this representation by utilizing the zero-sum property of chess to effectively treat both players as maximizers of their respective positions (i.e. $\min(a, b) = -\max(-b, -a)$). Through this, the Negamax search objective is to find the optimal move for the player who is playing at the root node for that turn.

While well-suited for zero-sum games, tree-based search methods like Negamax suffer from severe computational complexity in cases where the average branching factor is large. For chess, the average branching factor is around 35, with the size of the game tree in the order of 10^{123}. To tackle this issue, **alpha-beta pruning** can be used to prune irrelevant parts of the search tree through a branch-and-bound technique without the possibility of overlooking a better move. The alpha-beta algorithm keeps track of two parameters: α, which represents the minimum score that the *Max* player is assured of, and β, which represents the maximum score that the *Min* player is assured of. While the board is always evaluated from White's perspective in the traditional minimax approach, meaning the α and β values are from White's perspective, this is not true in Negamax; since Negamax involves evaluating from the current player's perspective, the window $[\alpha, \beta]$ must flip to $(-\beta, -\alpha)$ to match the new perspective across plies. By pruning off branches with scores that do not meet the alpha/beta cut-offs, the search space is extensively reduced, making the search algorithm run more efficiently.

2) Opening Books & Transposition Tables: To further speed up the search process, look-up procedures were utilized to avoid traversing the search tree whenever possible. One such method involves the use of **opening books**. The opening book contains various opening lines, continued to various depths, such that as long as the opponent plays moves within the book, the agent can look up the best response to that move. Once the game transposes to a position not found in the book, it reverts to the standard search algorithm. This technique significantly saves time, ensures high-quality play during the opening, and introduces variety by selecting different valid lines.

Similarly, **transposition tables** are used to avoid redundant searches. A transposition table is a cache that takes advantage of the fact that chess games often reach identical positions through different move orders (i.e. transpositions). By storing the results of previously performed searches, the agent can look up the solution for a recurring position as opposed to repeating a costly search. This drastically reduces the effective search space of the game tree without compromising search accuracy.

3) Iterative Deepening & Move Reordering: While Alpha-Beta pruning significantly reduces the search space, its efficiency is heavily dependent on the order in which moves are evaluated. In the worst-case scenario where the best moves are searched last, Alpha-Beta pruning provides no performance gain over standard Minimax, running in $O(b^d)$ time. However, if the optimal move is consistently examined first, the algorithm approaches its best-case time complexity of $O(b^{d/2})$, effectively doubling the searchable depth within the same time constraints. To approximate this optimal ordering, the RAM-Z agent implements **iterative deepening**. Rather than immediately searching to a fixed depth, the agent performs a depth-first search to depth 1, then depth 2, and so on, until the time limit is reached. While this results in re-searching the upper levels of the tree multiple times, the computational cost is negligible compared to the exponential growth of the tree at the final depth. Iterative deepening also facilitates effective **move reordering** by providing a "best guess" from the previous iteration. The RAM-Z agent prioritizes moves using a specific four-stage hierarchy to maintain high pruning efficiency:

1) **Hash Move:** The best move found during the previous iteration (or a prior search of the same position) is retrieved from the Transposition Table. Searching this move first offers the highest probability of triggering an immediate beta-cutoff.
2) **Most Valuable Victim - Least Valuable Attacker (MVV-LVA):** If the Hash Move does not cause a cutoff, the agent examines capturing moves. These are sorted using the MVV-LVA heuristic, prioritizing high-value trades over low-value ones.
3) **Killer Moves:** The agent maintains two "killer moves" per search ply. These are quiet moves that successfully caused a beta-cutoff at the same depth in sibling nodes. Since they refuted a similar position, they are prioritized here.
4) **History Heuristic:** Remaining quiet moves are sorted by their historical success. A 64×64 history table tracks how often a move from square A to square B causes a cutoff, weighted by $depth^2$. This dynamic heuristic biases the search toward moves that have proven effective throughout the game tree, such as moving a knight to a strong outpost.

This structured ordering ensures that strong moves are examined early, maximizing alpha-beta's pruning power.

4) Null-Move Pruning: **Null-Move Pruning**, also called **Null Move Heuristic (NMH)**, is a method based on the Null Move Observation. This observation states that, in almost all chess positions, making a null move is worse than the best legal move. With this in mind, if a reduced search on a null move fails high over beta, then we can be quite confident that the best legal move would also fail high over beta. This helps skip later move generation or any full-depth searches. However, this approach is prone to failing in Zugzwangs, which are positions where a player is forced to make a move, but any legal move they make will severely worsen their position, even though they might have been fine if they could pass their turn. To tackle this shortcoming, RAM-Z does not apply null-move pruning in cases where there are no major pieces on the board (which is where Zugzwangs tend to happen).

5) Quiescence Search: Given computational limitations, RAM-Z implements a depth-limited search approach. However, this leaves it prone to the horizon effect, which is when a significant change in material balance (such as a capture or promotion) occurs just beyond the fixed search depth, causing the agent to misjudge a dangerous position as safe. To tackle this, the agent employs **quiescence search**, a selective search extension applied at leaf nodes. Rather than accepting a static evaluation immediately, the algorithm continues to explore ”noisy” moves (specifically captures and promotions) until there are no more captures/promotions in the position.

To prevent this extension from becoming computationally expensive, RAM-Z implements two key optimizations. First, it uses a **stand-pat** lower bound: if the static evaluation of the current position is already sufficient to cause a beta-cutoff, the search stops immediately. Second, it employs **delta-pruning** with a safety margin of 1050 centipawns. This heuristic assumes that if the current score plus the value of a large piece (roughly a Queen) still fails to improve alpha, then no capture in the position is likely to alter the outcome, allowing the agent to safely prune the branch.