CSCI-5511 Project Proposal: Developing an Alpha-Beta Based Chess Agent

Ramzi Al-Sharawi
UMN ID: 5974003
E-mail: alsha192@umn.edu

Motivation

The use of artificial intelligence (AI) in chess has been a hot topic 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 grown leaps and bounds, 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. As such, using the topics taught throughout this course, I intend to implement a traditional chess engine by framing the game as a minimax problem and using alpha-beta pruning for searching the state space. However, given the high average branching factor per turn, it is crucial to modify the traditional alpha-beta pruning algorithm to enhance its efficiency and effectively handle the complexity of chess positions. Specifically, I intend to use methods such as move-reordering, iterative deepening, aspiration windows, transposition tables, and null-move pruning [3] to avoid redundant searches and optimize the pruning process. Additionally, since I will need to adopt a depth-limited alpha-beta search due to my limited computational resources, I intend to experiment with quiescence search to avoid the horizon effect and investigate its impact on performance. Such a classical approach circumvents the need for large training sets and computational resources to train a deep-learning model beforehand, allowing for a practical implementation that highlights the effectiveness of classical search-based AI methods in complex decision-making environments like chess.

Methodology

Given that the final project is due on December 18th, 2025, the following steps will be completed within their corresponding time frames to achieve the outlined goals of the project. All aspects of the programming will be done in **Python**.

1. **Modeling the Chess Environment and State Space**: This task will involve programming all aspects of the problem formulation. Specifically, I will focus on modeling the chess state representation, the valid actions per turn, the evaluation function, and the terminal state check. The ***python-chess*** library will be used to achieve this. These aspects will be important in guiding the search problem. I intend to finish this step by **November 9th, 2025**.

2. **Opening/Endgame Databases & Transposition Table**: Given the large average branching factor of chess, it is important to use lookup instead of search wherever convenient to reduce the size of the search space. Thus, I intend to use polyglot books to look up the best moves in the opening and endgame phases of the chess game. Furthermore, I aim to develop a transposition table using Zobrist hashing to retrieve previously analyzed positions to avoid redundant calculations. I intend to finish this step by **November 16th, 2025**.

3. **Null-Move Pruning**: This is used to reduce the search space by trying a "null" move (i.e. passing a move), then seeing if the score of the sub-tree search is still high enough to cause a beta cutoff. It is based on the null move observation, whereby making a null move is worse than the best legal move in almost all chess positions. However, zugzwangs are an exception to this rule, whereby every possible move from the player will worsen their position. Thus, I might need to disable null-move pruning in stages of the game that are prone to zugzwangs, such as pawn endgames. I intend to finish this step by **November 23rd, 2025**.

4. **Quiescence Search**: Since I will need to adopt a depth-limited alpha-beta search due to my limited computational resources, I intend to experiment with quiescence search to avoid the horizon effect and investigate its impact on performance. Given I intend to finish this step by **December 7th, 2025**.

5. **Iterative Deepening & Aspiration Windows**: Once I have a baseline alpha-beta agent achieved by completing the aforementioned steps, I intend to investigate using iterative deepening with aspiration windows [4] to evaluate its impact on performance. The technique is to use a guess of the expected value (usually from the last iteration in iterative deepening), and use a window around this as the alpha-beta bounds. Because the window is narrower, more beta cutoffs are achieved, and the search takes a shorter time. The drawback is that if the true score is outside this window, then a costly re-search must be made. I intend to finish this step by **November 30th, 2025**.

6. **Solution Evaluation & Report Write-Out**: Once I finalized the chess agent, I plan to play it against ELO-limited Stockfish (1400, 1600, 1800+) to evaluate its performance and get an idea of its ELO rating. This stage will also include making minor adjustments to the agent if needed. Then, I will write a report summarizing my findings and results to submit alongside my code for the project. I intend to finish this step by **December 18th, 2025**.

References

[1] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm, 2017.

[2] Dominik Klein. Neural networks for chess, 2022.

[3] Omid David-Tabibi and Nathan S. Netanyahu. Verified null-move pruning, 2008.

[4] Reza Shams, Hermann Kaindl, and Helmut Horacek. Using aspiration windows for minimax algorithms. pages 192–197, 01 1991.