Final Project Report: AI Chess Engine

Jackie Trinh, Pranshu Jindal, Arnold Zhou, Tejvir Dureja, Evan Tan, Jennifer Tan, Sofia Cho

Project Overview

Motivated by the exploration of AI search and strategic decision-making, our chess engine project aimed to develop a competitive chess-playing AI capable of achieving an average proficiency level equivalent to a 1200 Elo rating. Our engine leverages deep learning and reinforcement learning to analyze positions, make strategic decisions, and improve its performance over time. By integrating algorithms with extensive training data, we designed an AI capable of competing with and learning from both human players and other AI engines. Our general objectives:

- **Develop a Competitive AI Chess Engine:** create an AI capable of playing chess at a high-level
- **Optimize Computational Efficiency:** implement techniques to reduce the computational load and increase the speed of decision-making
- **Incorporate Advanced Chess Strategies:** integrate evaluation metrics and game knowledge to enhance gameplay

Background

To understand our AI chess engine project, it's essential to grasp the fundamental concepts of chess and the challenges and opportunities it presents to artificial intelligence. Chess is a two-player strategy game that has been a benchmark for artificial intelligence research for decades. The complexity of chess arises from its vast search space, characterized by an average branching factor of 35 legal moves per position and a typical game extending to 40 moves deep. Since there are 2 players in a typical game, the search space would be around 2.3×100^{123} possible positions. This results in an astronomically large game tree, making exhaustive search impractical and necessitating AI techniques.

In chess, each player controls an army of 16 pieces: one king, one queen, two rooks, two knights, two bishops, and eight pawns. The game is played on an 8x8 grid of squares, with players alternating turns to move their pieces. Each type of piece has unique movement patterns:

- **King**: Moves one square in any direction. The game ends when a king is checkmated, meaning it is under threat of capture and cannot escape.
- **Queen**: Moves any number of squares in any direction.
- **Rook**: Moves any number of squares along a row or column.
- **Bishop**: Moves any number of squares diagonally.
- **Knight**: Moves in an L-shape: two squares in one direction and then one square perpendicular.
- **Pawn**: Moves forward one square, with an option to move two squares on its first move, and captures diagonally.

The objective is to checkmate the opponent's king while avoiding checkmate of one's own king. Checkmate occurs when the king is in a position to be captured ("in check") and there is no legal move to escape the threat.

These fundamental concepts set the stage for understanding the techniques we employ in our AI chess engine to navigate this complex game efficiently and effectively.

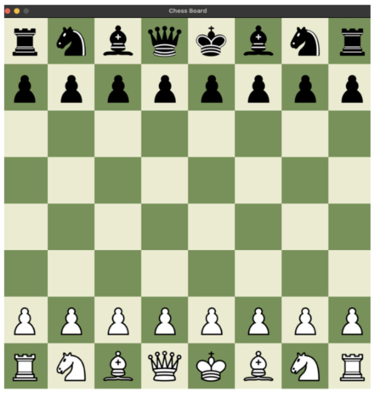
Methodology

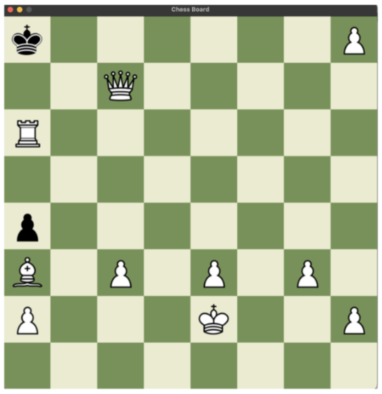
Our AI chess engine incorporated a variety of AI techniques, data sources, third-party tools, and non-AI software to achieve our objectives.

Development of Gameboard

Pygame: we chose Pygame, a popular python library for game development, to create the chessboard. Pygame provided the necessary tools to render a graphical chessboard and handle user inputs:

- Setting Up the Display: we initialized Pygame and set up an 8x8 grid representing the chessboard.
- Loading Chess Pieces: using a sprite sheet containing images of chess pieces, we utilized Pygame's blit function to draw the pieces on the board at their respective positions.
- Handling User Inputs: the game loop was designed to listen for mouse clicks to select and move pieces, updating the board and refreshing the display accordingly.





AI Performance Evaluation

Stockfish: to evaluate our AI's performance, we had it play against Stockfish, a renowned open-source chess engine known for its high level of play.

- Skill Levels: stockfish offers skill levels from 0 to 20. We set it to skill level 4 (approximately 1200 Elo rating) to provide a reasonable challenge for our AI.
- Performance Metrics: we tracked metrics such as the average time taken per move, wins, and draws to assess our AI's effectiveness and identify areas for improvement.

AI Development Timeline

Version 1: Alpha-Beta Pruning + Minimax, Simple Board Evaluation

Minimax Algorithm: a decision-making tool used in adversarial games like chess, simulating all possible moves and countermoves to determine the optimal move for the AI.

Alpha-Beta Pruning: enhanced the efficiency of the minimax algorithm by eliminating branches of the game tree that do not need to be explored, reducing computational load.
Simple Board Evaluation: evaluated position by calculating the weighted difference between total number of white pieces and black pieces.

Version 2: Enhanced Evaluation with Piece-Square Tables
Piece-Square Tables: evaluated the position of each piece on the board by assigning scores based on their location, aiding the AI in making more informed decisions.
Improved Minimax and Alpha-Beta Pruning: refined these implementations to work seamlessly with the piece-square tables, enhancing the AI's strategic depth.

- **Version 2.1: Iterative Deepening -** balanced depth and breadth in our search by incrementally increasing the search depth until a time limit was reached, ensuring optimal moves within constraints
- **Version 2.2: Move Ordering -** prioritized certain moves over others based on their likelihood of being optimal, further improving search efficiency
- **Version 2.3: Enhanced Evaluation with Mobility -** assessed the number of legal moves available to each piece, with greater mobility indicating a stronger position, aiding in better strategic choices

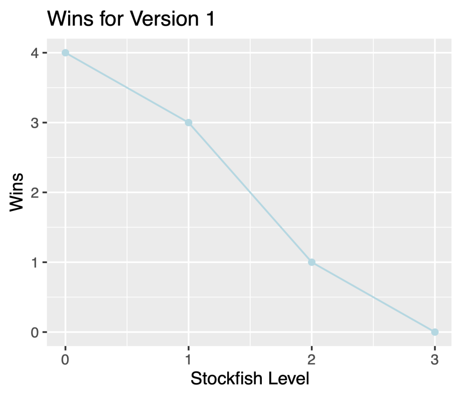
Version 3: Comprehensive Integration with Opening and Endgame Tablebases
Opening Tablebases: integrated Polyglot opening books into our AI, giving it access to a repository of expert-approved opening moves, allowing the AI to navigate the early game more effectively.

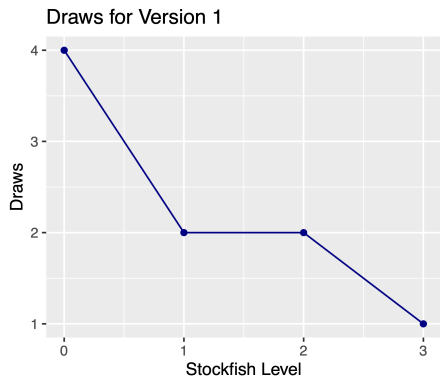
- **Piece Value**: the AI calculates the value of pieces it has compared to its opponent. Each piece is assigned a specific value (e.g., pawns = 1 point, knights/bishops = 3 points, rooks = 5 points, and queens = 9 points). The score reflects the material balance on the board, where a positive score indicates a material advantage and a negative score indicates a disadvantage.
- **Piece Position Score:** beyond the simple count of pieces, the AI evaluates the strategic positions of the pieces on the board. Certain squares are more valuable depending on the stage of the game. For example, controlling the center squares is crucial in the opening and middle game, while positioning pieces for optimal defense or attack throughout the game. Piece-square tables are used to assign these positional values.
- **Endgame Tablebases**: as the game progresses into the endgame, the AI utilizes endgame tablebases like Syzygy to assist with move decisions. These tablebases provide precomputed optimal moves for specific endgame scenarios. Syzygy tablebases return values of 1, -1, or 0, indicating a win, loss, or draw respectively. This helps the AI make precise decisions in endgame situations.
- **Pawn Advancement**: in the late game, advancing pawns becomes a priority. The AI strategically moves pawns towards promotion, aiming to convert them into more powerful pieces like queens. This is important in endgames, where pawns play a decisive role in achieving victory.

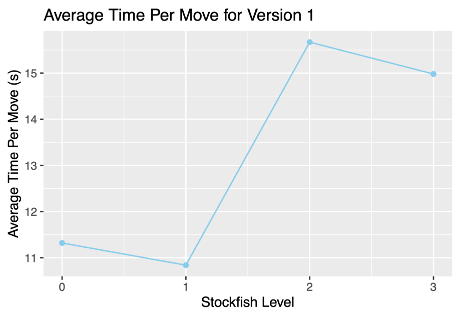
Evaluation Results

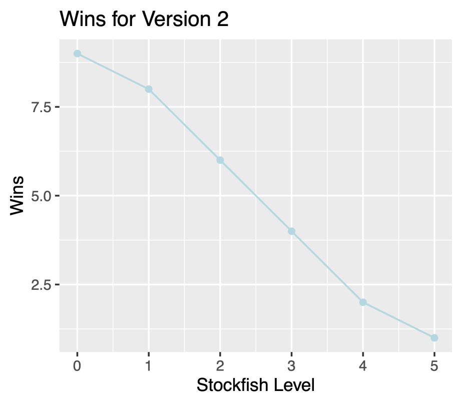
Github Link: https://github.com/JaekeeAI/ecs170project

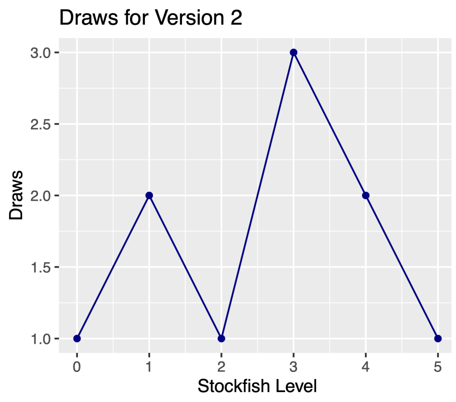
To evaluate the performance improvements of our AI chess engine across different versions, we plotted three key metrics: win rate (stockfish & human), average time per move, draw rate. These metrics provide insights into how each version performs against Stockfish at varying skill levels. The plots illustrate the effectiveness of enhancements such as piece-square tables, move ordering, and opening/endgame tablebases in improving the AI's strategic depth and efficiency (*data derived from a 10-game test). The new version of the Chess AI demonstrated a remarkable improvement by winning every game played against the older version. This unbroken winning streak provides strong evidence of the advancements made in the new version's strategic decision-making and move evaluation algorithms:

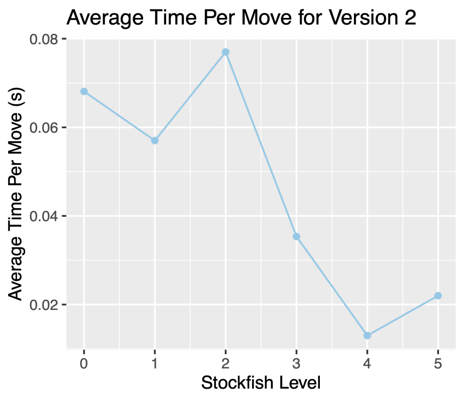


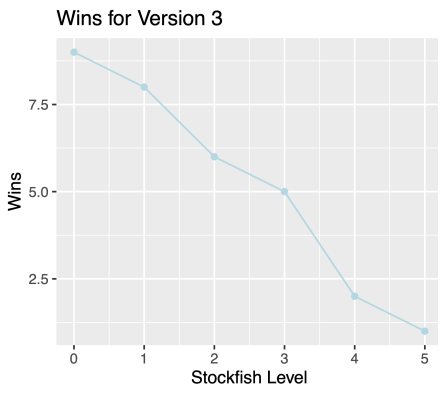


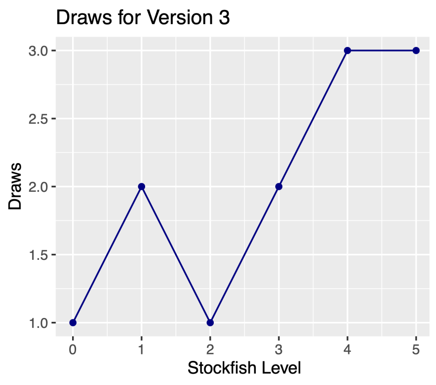


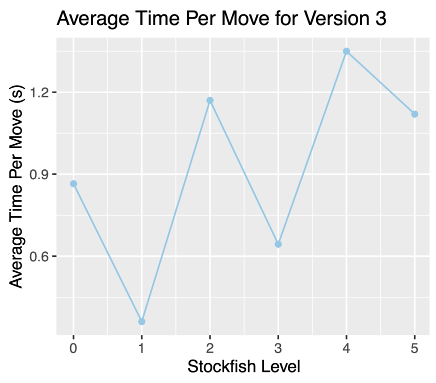


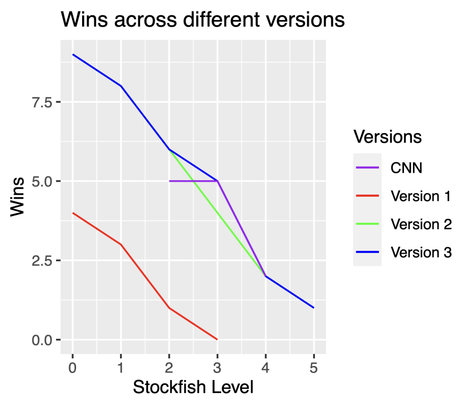


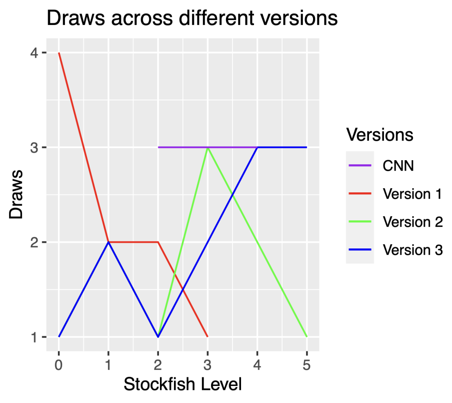


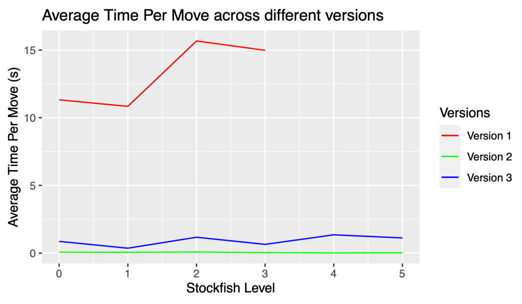












Wins Across Different Versions: throughout the development process, we rigorously tested each version of our AI against Stockfish. The AI managed to win the majority of games at skill level 3, demonstrating a performance level around 1200 Elo. However, it struggled to maintain a winning record at skill level 4, highlighting areas for further improvement. The plot for wins across different versions shows the following trends:

- Version **1**: the AI has a steep decline in wins as the Stockfish level increases, starting strong at level 0 but quickly losing effectiveness at higher levels.
- Version **2**: this version demonstrates a significant improvement over Version 1, maintaining a higher number of wins up to Stockfish level 3. However, the number of wins drops sharply at levels 4 and 5.
- Version **3**: this version shows further improvement, sustaining wins better at higher Stockfish levels compared to Versions 1 and 2. The AI remains competitive even at Stockfish level 5.

Draws Across Different Versions: the plot for draws across different versions highlights:

- Version **1**: the number of draws is relatively consistent across different Stockfish levels but generally low.
- Version **2**: this version shows an increase in draws, particularly at Stockfish levels 2, 3, and 4, indicating better performance in holding its own against stronger opponents.
- Version **3**: draws are more frequent compared to Versions 1 and 2, especially at higher Stockfish levels, suggesting a balanced improvement in defensive capabilities.

Average Time Per Move Across Different Versions: the plot for average time per move across different versions indicates:

- Version **1**: the AI takes a considerable amount of time per move, with a noticeable increase at higher Stockfish levels.
- Version **2**: this version shows a significant reduction in the average time per move, reflecting enhanced efficiency in decision-making.
- Version **3**: the AI maintains a low average time per move, similar to Version 2, indicating sustained improvements in computational efficiency.

Discussion

To reiterate how the changes we made were reflected in performance improvements:

- **Performance Improvement:** each successive version shows a clear improvement in performance metrics, particularly in the number of wins and draws. Version 3 demonstrates the most balanced performance, maintaining competitiveness even at higher Stockfish levels. This includes assigning higher values to better positions and recognizing the importance of maintaining a solid pawn structure.
- **Efficiency Gains:** the reduction in average time per move from Version 1 to Version 2, and sustained in Version 3, highlights significant efficiency gains. These improvements can be attributed to the enhancements in the evaluation function and the introduction of techniques such as move ordering and piece-square tables. Integrating opening and endgame tablebases gave the AI a solid foundation for attack in critical phases of the game, allowing it to transition smoothly from opening to mid-game to endgame.
- **Defensive Capabilities:** the increase in the number of draws, especially in Version 3, indicates that the AI has become more adept at holding its ground against stronger opponents, reflecting improved defensive strategies.

Conclusion

Initially, we developed a basic chess AI using Pygame, implementing a minimax algorithm with alpha-beta pruning. Our primary evaluation function focused on the number of pieces on the board, prioritizing moves that resulted in a higher score. However, the AI made poor decisions, prompting us to enhance the evaluation function with piece-square values for better strategic positioning.

To address the AI's slow decision-making, we introduced move ordering, which improved efficiency by prioritizing certain moves. In Version 2, we integrated an improved evaluation function with piece-square tables and move ordering, yet the AI's performance plateaued. We then experimented with various strategies, such as adding an opening book and tablebase, adjusting piece values, and incentivizing pawn promotions. Despite these efforts, the improvements yielded marginal performance gains.

Ultimately, our final version emphasized mobility and included only the opening and endgame tablebases. Other tactics, like forks and skewers, significantly increased evaluation time without substantial payoff. The AI's win rate against Stockfish level 4 remained unchanged.

Given these constraints, refining the heuristics used for move evaluation is one approach to enhance performance. By incorporating additional factors such as key square control, piece coordination, and potential threats, the AI's evaluations can become more sophisticated. However, these enhancements also increase computational complexity and evaluation times.

To further improve our AI, integrating deep learning techniques offers a promising solution. Deep learning allows the AI to learn from extensive datasets of expert games, capturing patterns and strategies beyond traditional heuristics. While deep learning doesn't eliminate the need for deep searches, it significantly enhances move selection and evaluation quality, leading to more efficient and effective decision-making. This approach could hold great potential for developing a stronger and more efficient chess AI.

Contributions

- **Jackie Trinh:** Collaborated with team members on different sections of the codebase. Ensured seamless integration and functionality of all parts, while keeping the project up to date.
- **Pranshu Jindal:** I helped in the development of evaluation methods with Tej and contributed to the project's overall strategy and planning and facilitated collaboration. This can be seen from my work in preparing certain final presentation slides and making code changes. I also attended meetings consistently and made sure to keep on track with the developments by running updated game code on my end.
- **Arnold Zhou:** Worked on front-end and developed basic working UI for chess games. Fixed some bugs with board evaluation and AI decision making, deployed a model to Lichess for other people to play with the AI.
- **Tejvir Dureja:** I implemented the pawn promotion logic within our chess engine and contributed to the development of evaluation methods. Additionally, I designed and integrated the introductory and endgame UI screens, handling user interactions such as player selection, starting new games, and displaying the winner upon checkmate.
- **Evan Tan:** I integrated the piece-square table into the evaluation function and reassigned piece values. Developed a deep learning approach by utilizing a convolutional neural network and game database to capture piece-board relationships.
- **Jennifer Tan:** my primary role was as project manager. I collaborated with Sofia to complete all deliverables, organizing logistics and streamlining our timeline. I worked alongside the team to translate complex technical requirements into actionable tasks, ensuring smooth communication and efficient progress throughout the project.
- **Sofia Cho:** My primary role was also as a project co-manager. I collaborated with Jennifer on deliverables and the presentation slides, and attended most team meetings to discuss and develop the project roadmap together.