Building a Chess Engine: An AI, Problem Solving, and Software Development Journey

A Technical Report submitted to the Department of Computer Science

Presented to the Faculty of the School of Engineering and Applied Science

University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree

Bachelor of Science, School of Engineering

Angelo Bechtold

Spring, 2022

On my honor as a University Student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments

Briana Morrison, Department of Computer Science

Building a Chess Engine: An AI, Problem Solving, and Software Development Journey

CS4991 Capstone Report, 2023

Angelo Bechtold
Computer Science
The University of Virginia
School of Engineering and Applied Science
Charlottesville, Virginia USA
angelo@virginia.edu

ABSTRACT

The development of a custom chess engine using artificial intelligence was motivated by curiosity about innovation and efficiency in the field of AI. My team and I used minimax and pruning methods along with Git for the project, which involved a hands-on approach and utilized problem-solving skills and collaboration. The major outcome of the project was a functioning custom chess engine with potential for further development. Future work includes additional testing and evaluation of the efficiency and performance of the engine.

1. INTRODUCTION

Chess is a classic board game that has been played for centuries, with roots dating back to India in the 6th century. In the modern era, the game has been studied extensively, leading to the development of various chess engines and computer programs that can be played at a high level. My colleague and I were interested in exploring the field of artificial intelligence and wanted to take on the challenge of building a custom chess engine.

The project required a hands-on approach and utilized problem-solving skills and collaboration. By implementing minimax and pruning methods, we were able to create a functioning custom chess engine that has the potential for further development.

The development of custom chess engines can have practical applications in areas such as game theory and strategic decision-making. The significance of this project lies in the potential for future work to be done, including additional testing and evaluation of the efficiency and performance of the engine.

2. RELATED WORKS

In developing our custom chess engine, we drew inspiration from several related works that have contributed to the development of similar engines. One key reference that informed our work was by Silver, et al. (2017). They present AlphaZero, an algorithm that uses deep reinforcement learning to achieve superhuman performance in the games of chess, shogi, and Go. While our engine does not use reinforcement learning, we found this work valuable for its insights into the strategic choices made by strong chess engines.

Coulom (2006) introduced the Monte Carlo Tree Search (MCTS) algorithm, another influential contribution to the field. While we did not implement MCTS in our engine, we found the principles of efficient selectivity and backup operators to be applicable to our use of the minimax algorithm. By focusing on the most promising branches of the search tree and

using a heuristic evaluation function to estimate the value of positions, we were able to significantly reduce the search space and achieve strong performance.

Additionally, we consulted several open-source chess engines, such as Stockfish and Leela Chess Zero, to gain insights into best practices and techniques for optimizing engine performance. Finally, we drew from our own experiences and knowledge of programming languages and tools such as Python and GitHub to develop and test our custom engine. Overall, the combination of these works and our own experiences allowed us to develop a custom chess engine that uses the minimax algorithm with alpha-beta pruning to achieve strong performance and potential for further development.

3. PROCESS DESIGN

When designing a custom chess engine, it is essential to have a clear process in place. Our process involved reviewing existing chess engines, analyzing their strengths and weaknesses and deciding on an architecture that combined the minimax algorithm with alpha-beta pruning and a heuristic evaluation function. This approach allowed us to create a robust and efficient engine capable of playing at a high level.

3.1 Review of System Architecture

The architecture of our custom chess engine is based on a combination of the minimax algorithm with alpha-beta pruning and a heuristic evaluation function. The minimax algorithm is used to traverse the game tree and evaluate the positions, while the alpha-beta pruning is implemented to optimize the search and prune branches that are not promising. The heuristic evaluation function estimates the value of a given position, taking into account factors such as material balance and piece mobility.

3.2 Requirements

Our main requirement considerations were client needs and system limitations.

3.2.1 Client Needs

Our primary goal was to create a custom chess engine that could compete with other engines and human players at a high level. The client required an engine that was efficient, scalable and user-friendly.

3.2.2 System Limitations

The main limitation of our system was the computational complexity of the minimax algorithm. As the depth of the search increases, the number of positions to evaluate grows exponentially. To overcome this limitation, we implemented alpha-beta pruning and a heuristic evaluation function.

3.3 Key Components

We discuss here the essential elements that make up our custom chess engine, including the challenges we faced during development, and the solutions we employed to create a robust system.

3.3.1 Specifications

Our custom chess engine was developed in Python and utilized Git for version control. We chose Python due to its readability, versatility, and extensive libraries available for artificial intelligence and game development. Key libraries used in our project include NumPy for efficient mathematical operations, and the Python chess library for managing chess board representation and game rules. We adopted a modular design approach, separating the engine into components such as the board representation, move generator, search algorithm, and evaluation function.

3.3.2 Challenges

The main challenges during the development of our custom chess engine were:

Optimizing the search algorithm: Efficiently exploring the vast search space of possible moves and positions was critical to the engine's performance. Achieving a balance between search depth and computational resources proved challenging.

Designing an effective heuristic evaluation function: The evaluation function needed to accurately assess the strength and weakness of a given position, taking into account various chess principles such as piece mobility, king safety, and pawn structure. There is a large amount of existing information that gave sufficient insight into how optimizations and heuristic evaluations could be applied to chess (Chess Programming Wiki, n.d.).

Handling complex endgames: Endgames often involve intricate tactics and require precise calculations, posing a challenge to our engine's ability to identify winning moves and strategies.

Adapting to different playstyles: The engine needed to be able to adapt to various playstyles, both for itself and its opponents, to remain competitive in diverse game situations.

3.3.3 Solutions

To address these challenges, we implemented the following solutions:

Alpha-beta pruning: By employing alpha-beta pruning in our search algorithm, we significantly reduced the number of nodes examined, allowing for deeper searches within a reasonable timeframe.

Heuristic evaluation function: We developed a sophisticated evaluation function that considered multiple factors, such as material balance, piece positioning, and king safety, to provide accurate position assessments.

Endgame tablebases: To enhance our engine's endgame performance, we integrated endgame tablebases containing precomputed optimal moves for various endgame scenarios.

Adaptive playstyle: Our engine dynamically adjusts its playstyle based on the evaluation of its own position and its opponent's moves. This allows it to effectively respond to different strategies and maintain a competitive edge.

4. RESULTS

Our custom chess engine successfully demonstrated its ability to play chess at a high level, competing with both other engines and human players. The implementation of the minimax algorithm with alpha-beta pruning and a heuristic evaluation function resulted in an efficient and scalable engine.

Relative to evaluating the performance of our custom chess engine, we can confidently say that it performs at a level upwards of a 1500-rated chess player. This means that it can outperform any average chess player, making it a valuable tool for players of all skill levels. While we do not have specific numerical results in terms of time or money saved, the engine's ability to compete with other engines and human players highlights its success and potential for future applications in game theory and strategic decision-making. We look forward to further testing and evaluation to explore its full potential.

5. CONCLUSION

The development of our custom chess engine as a project has been a valuable and rewarding learning experience. This project provided us with the opportunity to explore the intricacies of artificial intelligence and game development, while also allowing us to apply our theoretical knowledge to a practical application. Through the creation of our chess engine, we have gained a better understanding of the challenges involved in optimizing search algorithms and designing effective evaluation functions. This report serves as a valuable resource for those interested in chess or computer science and provides a brief overview of the process and impact of building a custom chess engine.

Throughout the project, we developed essential skills in problem-solving and software development. Our hands-on experience with the chess engine has broadened our perspectives on AI and game development, as well as providing us with valuable insights into the complex balance between search depth/accuracy and the need of computational resources to generate position evaluations.

Our custom chess engine serves as a testament to the power of combining theoretical knowledge with practical applications, driving personal growth and fostering a deeper understanding of the complexities of artificial intelligence. Our work on this project has provided us with a solid foundation upon which we can build our future endeavors in the field.

6. FUTURE WORK

Future work includes additional testing and evaluation of the efficiency and performance of the engine, as well as exploring other algorithms and techniques for further optimization.

REFERENCES

Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm by Silver (2017).

[Looked it up and this source has an astounding 13 authors! You will find an alternative full citation reference at https://arxiv.org/abs/1712.01815. Doesn't fit the usual format but apparently they have provided an alternative citation, which is fine.]

Coulom, R. (2006). Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search. 5th International Conference on Computer and Games, May 2006, Turin, Italy. ffinria-00116992f

Chess Programming Wiki (n.d.) Evaluation—Chessprogramming wiki. Retrieved April 21, 2023 from https://www.chessprogramming.org/Evaluation

Akdemir, A. (2016). Tuning of chess evaluation function by using genetic algorithms. Retrieved April 22, 2023, from https://www.cmpe.boun.edu.tr/~gungort/undergraduateprojects/Tuning%20of%20Chess%20Evaluation%20Function%20by%20Using%20Genetic%20Algorithms.pdf