Chess Playing Agent

Dhruv Khanna: 2022BCS-023
Ravjot Singh: 2022BCS-057
Rudra Suthar: 2022BCS-066

Department of Computer Science, Indian Institute of
Information Technology and Management, Gwalior, Madhya Pradesh, 474015, India

April 15th 2024

Developing an AI chess opponent is a challenging pursuit in computer science, leveraging heuristics-based AI to analyze board positions and prioritize moves. Alpha-beta pruning optimizes the search process, enhancing decision-making. To manage computational complexity, users can specify a search depth. This project balances thoroughness with efficiency, ensuring the program considers a reasonable number of moves while maintaining a high level of play. The focus on legal moves underscores the program's adherence to the game's rules, facilitating strategic decision-making for optimal game play.
Keywords: Heuristics, Alpha-beta pruning, Search depth, computational complexity

1 Introduction

In recent years, there has been a significant surge in interest within the field of artificial intelligence (AI) research towards developing AI agents specifically tailored for playing classic board games, with chess standing out prominently among them. Chess, renowned for its deep strategic complexities and rich history, presents an ideal testing ground for various AI algorithms. Its well-defined rules and finite decision tree make it an attractive domain for studying and refining AI techniques. The game's appeal lies not only in its inherent complexity but also in its ability to challenge AI systems to think critically, strategically, and adaptively. As a result, chess has become a quintessential benchmark for measuring the progress and capabilities of AI systems, showcasing the evolving landscape of AI research and its applications in solving complex, real-world problems.

2 Background Information

Historically, board games have been a focal point in AI research. IBM's Deep Blue's victory over chess grandmaster Garry Kasparov in 1997 marked a significant milestone. Since then, AI agents for board games have rapidly progressed. The appeal of these games lies in their complex strategy and decision-making processes, offering a rich testing ground for AI techniques.
Despite the computational challenges posed by classic board games, the Minimax algorithm with Alpha-Beta Pruning remains a prevalent choice due to its effectiveness in determining optimal moves while mitigating the exponential growth of the search space. In conclusion, AI agents for classic board games like chess continue to advance with the utilization of Minimax with Alpha-Beta Pruning, contributing to the ongoing exploration of intelligent decision-making processes in AI research.

3 Literature Review

The literature on AI agents developed for classic board games focuses on various AI aspects, including search algorithms, heuristic evaluation, and machine learning. Adversarial search algorithms like Minimax, coupled with Alpha-Beta Pruning, have been foundational. This combination provides a systematic way to explore possible moves and counter-moves while efficiently pruning branches of the search tree, significantly reducing computational complexity.

3.1 Minimax Algorithm

In the Minimax algorithm, let's denote the current game state as s, the player whose turn it is as P(s), and the

opponent as O(s). The goal is to find the optimal move for the player to maximize their score and minimize the opponent's score.
Let V(s) be the value of state s, which is determined by a heuristic evaluation function or the actual game rules. At each depth of the game tree, the algorithm computes the value of each state based on the following:

1. If the current state is a terminal state (e.g., win, loss, or draw), the algorithm returns the value of the state directly: V(s).

2. If the current state is not a terminal state, the algorithm recursively computes the value of each possible future state up to a certain depth. For each future state s' resulting from a possible move:

- If it's the player's turn (P(s)), the algorithm chooses the maximum value among the future states: V(s')=max(V(s')).

- If it's the opponent's turn (O(s)), the algorithm chooses the minimum value among the future states: V(s')=min(V(s')).

Formally, the minimax value of the current state s, denoted as MM(s), is computed as follows:

$$\text{MM(s)} = \begin{cases} V(s) & s \text{ is a terminal state} \\ \max_{s'}(MM(s')) & P(s) \text{ is player's turn} \\ \min_{s'}(MM(s')) & O(s) \text{ is opponent's turn} \end{cases}$$

In this expression, $\max_{s'}$ denotes the maximum value over all possible future states s', $\min_{s'}$ denotes the minimum value over all possible future states s'. The Minimax algorithm continues this process until it reaches the specified depth in the game tree, at which point it returns the score of the best move found at the root node, along with the move itself.

3.2 Alpha-Beta pruning

Alpha-Beta pruning is an optimization technique used in the Minimax algorithm to reduce the number of nodes evaluated in the game tree. It eliminates branches of the tree that are guaranteed to be worse than the current best move, without affecting the final result. This helps to improve the efficiency of the algorithm by reducing the number of nodes that need to be evaluated.
Let's denote the current state of the game as s, the player whose turn it is as P(s), and the opponent as O(s). The goal is to find the optimal move for the player to maximize their score and minimize the opponent's score. Alpha-Beta pruning maintains two values, alpha (α) and beta (β), which represent the minimum score that the maximizing player is guaranteed and the maximum score that the minimizing player is guaranteed, respectively. At each depth of the game tree, the algorithm computes the value of each state based on the following:

1. If the current state is a terminal state (e.g., win, loss, or draw), the algorithm returns the value of the state directly: V(s).

2. If the current state is not a terminal state, the algorithm recursively computes the value of each possible future state up to a certain depth. For each future state s' resulting from a possible move:

- If it's the player's turn (P(s)), the algorithm updates the alpha value as the maximum of the current alpha value and the value of the future state: α=max(α,V(s')). If α becomes greater than or equal to β, the algorithm prunes the remaining branches and returns the current alpha value.

- If it's the opponent's turn (O(s)), the algorithm updates the beta value as the minimum of the current beta value and the value of the future state: β=min(β,V(s')). If β becomes less than or equal to α, the algorithm prunes the remaining branches and returns the current beta value.

Formally, the alpha-beta value of the current state s, denoted as AB(s,α,β), is computed as follows:

$$\text{AB(s},\alpha,\beta) = \begin{cases} V(s) & s \text{ is a terminal state} \\ \max_{s'}(AB(s',\alpha,\beta)) & P(s) \text{ is player's turn} \\ \min_{s'}(AB(s',\alpha,\beta)) & O(s) \text{ is opponent's turn} \end{cases}$$

where s' represents the possible future states resulting from a move. The algorithm continues this process until it reaches the specified depth in the game tree, at which point it returns the score of the best move found at the root node, along with the move itself.

4 Problem statement

Developing an AI agent capable of playing classic board games, with a primary focus on strategic decision-making in Chess, presents a significant challenge in the field of artificial intelligence. The goal of this project is to implement the Minimax algorithm with Alpha-Beta Pruning to enhance the AI player's decision-making process, aiming to create a competitive and adaptable game playing agent. The complexity and strategic depth of Chess provide an ideal environment for testing and evaluating the effectiveness of these algorithms.

5 Objective

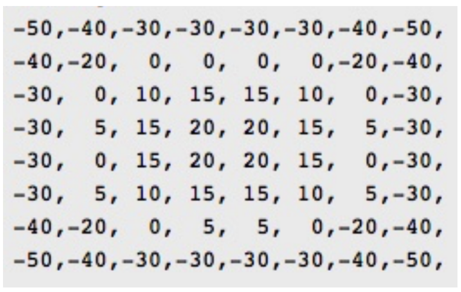
The objective of this project is to develop an AI agent capable of playing classic board games, with a specific focus on strategic decision-making in Chess. The primary goal is to implement the Minimax algorithm with Alpha-Beta Pruning to enhance the AI player's decision-making process, aiming to create a competitive and adaptable game playing agent. Chess's complexity and strategic depth make it an ideal game for testing and showcasing the capabilities of these algorithms. This project seeks to explore the effectiveness of Minimax with Alpha-Beta Pruning in the context of Chess, contributing to the advancement of AI in board games.

6 Deliverables

- Implementation of the Minimax algorithm with Alpha-Beta Pruning in Python.
- A functioning AI agent capable of playing Chess against a human player.
- Presentation or report summarizing the project's methodology, results, and insights gained from implementing Minimax with Alpha-Beta Pruning in Chess.

7 Solution Approach

1. **Algorithmic Framework:** Our chess engine is constructed upon the minimax algorithm, a commonly utilized method in game-playing AI. Minimax explores the game tree recursively, assessing positions at different depths to identify the optimal move. To enhance efficiency, we employ alpha-beta pruning, which minimizes node evaluations by discarding branches that offer no improvement over previously examined ones.
2. **Evaluation Function:** At the heart of our engine lies the evaluation function, which assigns a numerical value to each board configuration. This function takes into consideration various elements such as piece values, positional advantages, pawn structures, king safety, mobility, and control over the center. For example, a knight is preferably moved to the center of the board and this is reflected by the value table:
3. **Endgame Handling:** We pay special attention to addressing endgame scenarios, where fewer pieces are on the board and strategic dynamics shift. Our strategy involves implementing specialized evaluation and search techniques tailored specifically for the endgame phase, ensuring consistent performance and accuracy across different game stages.



```
-50,-40,-30,-30,-30,-30,-40,-50,
-40,-20,  0,  0,  0,  0,-20,-40,
-30,  0, 10, 15, 15, 10,  0,-30,
-30,  5, 15, 20, 20, 15,  5,-30,
-30,  0, 15, 20, 20, 15,  0,-30,
-30,  5, 10, 15, 15, 10,  5,-30,
-40,-20,  0,  5,  5,  0,-20,-40,
-50,-40,-30,-30,-30,-30,-40,-50,
```

Figure 1: Value table of knight

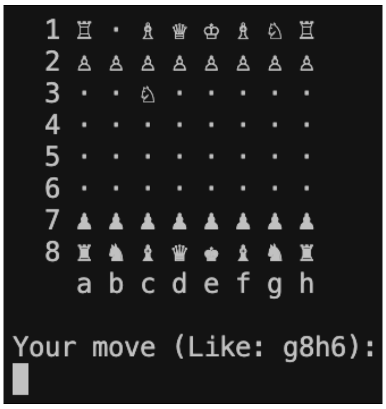
4. **Optimization Techniques:** To optimize the efficiency and effectiveness of our engine, we incorporate a range of optimization techniques:

- **Move Ordering:** Prioritizing moves based on their potential to maximize the effectiveness of alpha-beta pruning.
- **Transposition Table:** Utilizing a table to store previously evaluated positions, reducing redundant evaluations and improving search efficiency.
- **Iterative Deepening:** Incrementally increasing search depth until reaching a time limit, enabling more efficient time management and superior move selection.
- **Parallel Processing:** Leveraging parallel processing to explore multiple branches of the game tree simultaneously, speeding up the search process.

Larger tables and figures could be placed in a single column layout.

8 Results and Analysis

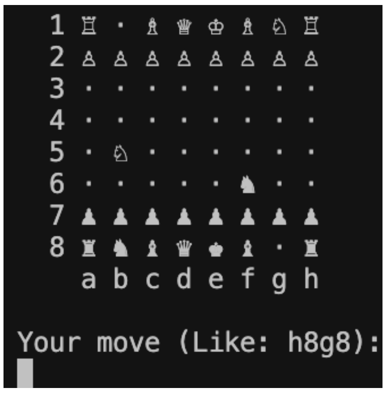
While some of the values used within these functions are a culmination of general guidelines used by many others when building chess engines, many of the values have the ability to be tweaked to an infinite number of combinations to possibly allow for a better board evaluation score.
The chess game itself will repeatedly ask the user to input their choice of move from a list of all possible generated moves. After the player has entered their move, the AI will automatically respond. During this process, the AI will be going through its alpha beta minimax algorithm trying to find the optimal move in response. A total amount of configurations tested is kept track of while the AI performs this search to help show some of the computing power exhausted behind each move generation. With even a small increase in search depth, the AI can take much longer to produce a move and will have many more board configurations tested. The game terminates based on two cases. Firstly, the game may terminate if when producing the players move set, the



```
1 ♖ · ♗ ♕ ♔ ♗ ♘ ♖
2 ♙ ♙ ♙ ♙ ♙ ♙ ♙ ♙
3 · · ♘ · · · · ·
4 · · · · · · · ·
5 · · · · · · · ·
6 · · · · · · · ·
7 ♟ ♟ ♟ ♟ ♟ ♟ ♟ ♟
8 ♜ ♞ ♝ ♛ ♚ ♝ ♞ ♜
  a b c d e f g h

Your move (Like: g8h6):
█
```

Figure 2: Board layout before



```
1 ♖ · ♗ ♕ ♔ ♗ ♘ ♖
2 ♙ ♙ ♙ ♙ ♙ ♙ ♙ ♙
3 · · · · · · · ·
4 · · · · · · · ·
5 · ♘ · · · · · ·
6 · · · · · ♞ · ·
7 ♟ ♟ ♟ ♟ ♟ ♟ ♟ ♟
8 ♜ ♞ ♝ ♛ ♚ ♝ · ♜
  a b c d e f g h

Your move (Like: h8g8):
█
```

Figure 3: Board layout showcasing AI's move selection after all evaluation techniques

total possible moves is empty, the player is in stalemate and will lose. On the other hand, after the players have successfully made their moves, if the board is absent of either player's king piece a checkmate has occurred.

9 Future Work and Conclusion

- **Advanced Evaluation Function:** Enhancing the AI's evaluation function is vital for better performance. This involves incorporating advanced features like piece mobility, pawn structure, king safety, and control of the center.

- **Machine Learning Integration:** By integrating machine learning methods, such as reinforcement learning (e.g., Q-learning) or deep learning (e.g., neural networks), the AI can learn from its gameplay and adapt strategies over time.

- **Parallelization Techniques:** Implementing parallelization methods, such as utilizing multiple CPU cores or GPU acceleration, can significantly improve efficiency by enabling deeper searches within the same time frame.

- **Endgame Database:** Implementing an endgame database can help the AI make optimal decisions in endgame scenarios with a small number of pieces remaining on the board. By precomputing optimal moves for all possible positions, the AI can ensure optimal play and avoid costly mistakes in critical endgame situations.

- **Online Learning:** Implementing online learning techniques can enable the AI to adapt and improve in real-time based on feedback from human players or online gameplay data. By continuously updating its knowledge and strategies, the AI can stay competitive and evolve over time.

In conclusion, our chess playing AI agent has demonstrated promising performance through the implementation of minmax algorithms with alpha-beta pruning. By efficiently searching through the game tree and evaluating potential moves, the AI is capable of making strategic decisions to compete against human players and other AI opponents. However, there is still ample room for improvement and future work to enhance the AI's capabilities further. By focusing on above mentioned areas we can continue to advance the AI's performance and make it a formidable competitor in the world of chess. Overall, our journey in developing this chess AI has been rewarding, and we look forward to continuing our efforts to push the boundaries of AI in the realm of strategic board games.

10 Declarations

10.1 Acknowledgements

We extend our heartfelt gratitude to Dr. Wilfred Godfrey, our AI professor, for his invaluable guidance and support. His expertise and encouragement have been instrumental in shaping our understanding and achieving success in the realm of artificial intelligence.

10.2 Use of AI

This report was crafted with the assistance of ChatGPT, an AI language model developed by OpenAI, to ensure clarity, coherence, and conciseness in conveying the project's findings and acknowledgments.

References

[1] Chess Programming Wiki. *Simplified Evaluation Function*. Accessed: April 2024. URL: `https://www.chessprogramming.org/Simplified_Evaluation_Function`.

[2] Free CodeCamp. *MinMax Chess Piece Weights*. Published: March 2017. URL: `https://www.freecodecamp.org/news/simple-chess-ai-step-by-step-1d55a9266977`.

[3] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Accessed: April 2024. Prentice Hall, 2010. URL: https://people.engr.tamu.edu/guni/csce421/files/AI_Russell_Norvig.pdf.

[1] [2] [3]