Chess Engine using Negamax Algorithm

Atharva Bansod
Department of Computer Science and Engineering- AIML G.H. Raisoni College of Engineering and Management Pune, India

Devesh Mhaske
Department of Computer Science and Engineering- AIML G.H. Raisoni College of Engineering and Management Pune, India

Devashish Murai
Department of Computer Science and Engineering- AIML G.H. Raisoni College of Engineering and Management Pune, India

Asst. Prof. Amruta Kulkarni
Department of Computer Science and Engineering- AIML G.H. Raisoni College of Engineering and Management Pune, India

Abstract : **This research presents the design and development of a chess engine utilizing the Negamax algorithm with key functionalities such as multiplayer gameplay, an AI opponent, chess puzzles, and a hint system. The focus lies on leveraging the Negamax algorithm to optimize decision-making in real-time chess games. Additionally, features like AI-driven puzzles and hints enhance both learning and competitive play.The chess engine integrates alpha-beta pruning to improve efficiency, enabling faster and deeper move calculations. It offers adjustable difficulty settings, making it suitable for players of all skill levels.**

I. INTRODUCTION

Chess, often referred to as the "game of kings," has captivated strategists and enthusiasts for centuries. Its complexity and requirement for both tactical and strategic thinking make it a perfect platform for the development and application of artificial intelligence (AI). Over the years, the creation of chess engines has been a significant area of study in the field of AI, offering challenges that involve both search algorithms and heuristic evaluations. In recent years, the advent of powerful engines such as Stockfish and AlphaZero has showcased the incredible potential of AI to play chess at a superhuman level. This project aims to develop a competitive chess engine using a simplified, yet efficient, approach: the Negamax algorithm.

The Negamax algorithm is a variation of the well-known Minimax algorithm, specifically optimized for two-player, zero-sum games like chess. In such games, the gain of one player is equal to the loss of the other, making the game zero-sum. The Negamax approach simplifies the decision-making process by assuming that both players aim to maximize their advantage and minimize their opponent's, but through a uniform mechanism. This allows for a more compact and efficient implementation compared to the traditional Minimax approach.

This research project focuses on creating a chess engine that uses Negamax combined with alpha-beta pruning to optimize the search process. The goal is to balance computational efficiency with high-quality decision-making, providing a strong AI opponent for human players. In addition to the AI, the project also incorporates features such as real-time multiplayer, a puzzle mode, and a hint system, creating an interactive platform for both competitive and educational purposes.

The project's significance extends beyond gameplay. By using a well-structured AI approach, it provides developers and researchers with insights into the integration of game theory, search algorithms, and heuristic evaluations. It also offers an educational tool for beginners to learn chess through puzzles and hints, further enhancing the learning experience by giving players real-time feedback on their moves.

1.1 Objectives

- Develop a Chess Engine: Utilize the Negamax algorithm for generating optimal moves in a chess game, incorporating depth-limited search and alpha-beta pruning.
- AI Chess Bot: Design an AI opponent with adjustable difficulty levels, providing a challenging experience for players of all skill levels.
- Puzzle Mode and Hints: Include a puzzle mode and hint system that helps players solve chess problems and offers strategic advice, enhancing learning opportunities.

1.2 Justification

The rapid development of chess engines has made it possible for computers to play at a superhuman level, but many existing engines, such as Stockfish and AlphaZero, are designed for advanced users and professional players. While these engines are incredibly powerful, they are often too complex for casual players to engage with meaningfully, especially those who wish to learn and improve their chess skills.

This project fills that gap by providing a chess engine that offers competitive gameplay through the Negamax algorithm while incorporating features aimed at education and user interaction. The real-time multiplayer function ensures that players can engage with each other, while the puzzle mode and hint system offer a tailored learning experience. By making the chess engine both competitive and educational, the project provides a well-rounded solution for a broad audience, from casual players to developers looking to explore AI game logic.

II. LITERATURE REVIEW

The development of chess engines has seen a steady evolution, from early brute-force methods to sophisticated AI-driven solutions. As chess AI advanced, the need for more efficient algorithms became apparent, leading to the development of various search strategies and optimizations. The Negamax algorithm, an extension of the classic Minimax algorithm, plays a key role in this progress, particularly in two-player, zero-sum games like chess.

2.1 Evolution of Chess Engines

The first chess engines relied heavily on brute-force search techniques, where all possible moves and subsequent positions were explored to determine the best course of action. One of the earliest algorithms applied in chess engines was Minimax, which evaluates the best move by assuming that the opponent will always make the move that is most detrimental to the player. The algorithm recursively explores all potential future board configurations, assigning a value to each position based on an evaluation function that considers factors like material balance and positional strength.

While Minimax is theoretically sound, it suffers from the "combinatorial explosion" problem—the number of possible board states grows exponentially as the game progresses. To make Minimax more practical, Negamax was developed as a simplified version tailored for two-player zero-sum games. In Negamax, the game tree is evaluated from the perspective of the current player at every node, simplifying the implementation by reducing the complexity of managing both maximizing and minimizing operations.

2.2 Negamax Algorithm

The Negamax algorithm is a natural extension of Minimax. Instead of evaluating the best move for both players separately, Negamax assumes that the gain of one player is the loss of the other, allowing the same evaluation function to be used for both players. This is possible because chess, like other zero-sum games, is symmetrical in terms of objectives. In Negamax, the goal is to maximize the value of the current player's move and minimize the opponent’s value by flipping the sign of the evaluation.

The Negamax process follows these steps:

- Generate Legal Moves: The engine generates all possible legal moves for the current player.
- Recursive Search: The algorithm evaluates the board recursively by simulating future moves using a depth-limited search.
- Sign Inversion: Each time a move is made, the evaluation function is called with the sign of the result inverted, reflecting the zero-sum nature of the game.
- Backtrack and Prune: As the recursion unfolds, alpha-beta pruning is applied to discard branches of the search tree that are not likely to affect the final decision.

This allows the algorithm to efficiently explore the decision tree, searching for optimal moves while avoiding unnecessary evaluations. The use of alpha-beta pruning significantly reduces the number of nodes evaluated, enabling deeper searches without sacrificing performance.

2.3 Alpha-Beta Pruning

Alpha-beta pruning is an optimization technique used in conjunction with Negamax (and Minimax) to reduce the number of nodes evaluated in a game tree. The basic idea is to prune branches that cannot affect the final decision, thus avoiding unnecessary calculations. The algorithm keeps track of two values:

- Alpha: The best score that the maximizer (the current player) can guarantee.
- Beta: The best score that the minimizer (the opponent) can guarantee.

When the value of a position is worse than a previously examined position (for the maximizer or minimizer), the search can be cut off, or "pruned," as the move would not be chosen in any optimal strategy. This pruning drastically reduces the size of the game tree, allowing for deeper searches without exponential growth in computation.

2.4 Existing Chess Engines

Several chess engines have employed Negamax and alpha-beta pruning with significant success:

- Stockfish: One of the most powerful chess engines in the world, Stockfish uses Negamax and alpha-beta pruning to evaluate millions of board positions per second. Its evaluation function considers a wide array of factors, from material balance to piece activity and king safety. Despite its complexity, Stockfish serves as a benchmark for efficiency in AI-driven chess engines.
- AlphaZero: Developed by DeepMind, AlphaZero revolutionized chess AI by using deep neural networks combined with Monte Carlo Tree Search (MCTS) to evaluate positions through self-play. Although AlphaZero does not use Negamax, its success demonstrates the potential of combining search algorithms with machine learning techniques.
- Fritz: Another commercially successful chess engine, Fritz also employs Negamax in combination with a powerful evaluation function. It is known for providing an engaging user experience, especially with its graphical interface and multiplayer options.

2.5 Educational Tools in Chess Engines

Many existing chess engines, particularly open-source ones like Stockfish, focus on high-level play and optimization. However, there has been a lack of engines that provide educational tools for beginners. Features like hint systems, puzzle modes, and interactive tutorials can make chess more accessible to new players. While some engines like Chess.com offer puzzle and hint systems, these features are often standalone and not integrated into the core of the chess engine itself. This project aims to fill this gap by integrating such tools directly into the engine, making it both competitive and educational.

III. METHODOLOGY

The development of a chess engine using the Negamax algorithm involves integrating multiple components such as algorithm design, game logic, user interface, multiplayer synchronization, and AI strategy. Each part plays a crucial role in the overall functioning and user experience of the chess engine. The following sections provide a comprehensive breakdown of the methodologies employed to build the engine, including the AI decision-making process, system architecture, multiplayer functionality, puzzle mode, and hint generation.

3.1 Negamax Algorithm Overview

The Negamax algorithm is a streamlined version of Minimax, used specifically in two-player zero-sum games like chess. In Negamax, the goal is to evaluate moves recursively, assuming that both players aim to maximize their advantage and minimize the opponent's advantage. The key to its simplification is that the evaluation function remains the same for both players, with scores being inverted between turns.

Steps in the Negamax process include:

1. Board Evaluation: The current board state is evaluated based on a function that assigns values to pieces and their positions. The evaluation includes:
    - Material count: The relative value of pieces (e.g., pawns = 1, bishops and knights = 3, rooks = 5, queen = 9).
    - Positional evaluation: Factors like control of the center, development of pieces, king safety, and open files are considered.
2. Recursive Search: The algorithm performs a recursive search of possible future board states up to a pre-determined depth limit. For each move, the engine alternates between maximizing its own score and minimizing the opponent’s score by inverting the evaluation score.
3. Alpha-Beta Pruning: To optimize the search process, alpha-beta pruning is used to eliminate branches of the decision tree that are guaranteed to result in worse outcomes. This reduces the number of nodes that need to be evaluated and allows the engine to search deeper without a significant performance hit.
4. Depth-Limited Search: A depth limit is applied to the recursion, ensuring that the algorithm does not search too deeply and cause delays. The depth can be adjusted based on the desired difficulty level of the AI.
5. Move Selection: Once the search is complete, the best move is selected based on the highest evaluation score generated by the recursive search process.

3.2 System Architecture

The chess engine is built using a modular architecture, ensuring that each component (frontend, backend, and AI) operates independently but communicates efficiently. This architecture allows for scalability and easy integration of additional features like multiplayer functionality and the puzzle mode.

- Frontend: The frontend, built using React.js, provides an interactive chessboard where players can move pieces using a drag-and-drop interface. The frontend communicates with the backend using Socket.IO for real-time multiplayer functionality and HTTP requests for other actions such as AI move generation and puzzle solving.
- Backend: The backend is implemented using Python with the Flask web framework. It manages the game logic, handles move validation, processes AI decisions, and facilitates multiplayer synchronization. The Negamax algorithm is integrated into the backend to evaluate and generate optimal moves for the AI opponent.
- AI Engine: The AI engine is the core decision-making component, running the Negamax algorithm with alpha-beta pruning to evaluate board states and determine the best moves. The engine is designed to handle various difficulty levels by adjusting the depth of the search tree.

3.3 Move Validation and Game Logic

The game logic is managed by the backend, which ensures that all moves are legal and conform to the rules of chess. The engine validates moves by:

- Piece Movement: Ensuring that each piece moves according to the rules (e.g., pawns move forward but capture diagonally, knights move in an L-shape).
- Game State: Keeping track of the current state of the game, including whose turn it is, whether a player is in check, and whether checkmate or stalemate conditions have been met.
- Check and Checkmate Detection: The engine checks whether a move leaves the opponent's king in check and whether the current player is in checkmate, ending the game if necessary.

3.4 Multiplayer Functionality

The multiplayer functionality allows two users to play against each other in real-time. Socket.IO is used for real-time communication between the clients and the server, ensuring smooth synchronization of game states across both players' interfaces.
Steps in the multiplayer process:

- Player Connection: Two players connect to the server, and a game session is initialized.
- Move Transmission: When one player makes a move, it is transmitted to the server via Socket.IO. The server validates the move and updates the game state.
- Real-Time Update: The updated game state is then broadcast to both players, ensuring that the board reflects the same positions on both ends.
- Handling Disconnections: If a player disconnects during a game, the server stores the current game state so that the player can resume once they reconnect.

The multiplayer system is designed to handle multiple simultaneous games, ensuring that each player's game session is independent of others.

3.5 Puzzle Mode and Hint System

The puzzle mode and hint system are designed to help players improve their chess skills by solving tactical problems and receiving strategic advice.

- Puzzle Mode: In puzzle mode, players are presented with pre-configured or dynamically generated board positions that require them to find the best move. The puzzles are based on common tactical motifs like forks, pins, and checkmates in two or three moves. The puzzle generation system uses historical game data and tactics to ensure that each puzzle provides a meaningful challenge.
- Hint System: The hint system is powered by the Negamax algorithm. When a player requests a hint, the engine evaluates the current board state and suggests the most optimal move based on the same recursive search process used by the AI opponent. The hint system ranks potential moves and displays the top suggestion without directly revealing the full solution, allowing players to make their own decisions.

The hint system provides educational value by offering feedback on why a particular move is considered optimal, helping players understand the underlying strategy.

3.6 Difficulty Settings

The difficulty of the AI opponent is controlled by adjusting the depth of the Negamax search tree. At lower difficulty levels, the AI performs shallow searches, focusing on immediate tactical opportunities rather than long-term strategy. At higher difficulty levels, the AI conducts deeper searches, simulating multiple future moves and employing more sophisticated strategies.

IV. IMPLEMENTATION

The implementation phase involves the integration of the Negamax algorithm, user interface elements, and real-time multiplayer systems. The following sections outline the key steps in implementing each component of the chess engine.

4.1 Game Logic and Move Generation

The core of the chess engine is the game logic, which is responsible for managing the rules of chess and generating valid moves. The Self Build Chess Logic Package library is used to validate moves, ensuring that they conform to the rules of chess.
Key components of the game logic include:

- Move Generation: The game logic generates all possible legal moves for the current player and validates them according to the rules.
- Checkmate and Stalemate Detection: The engine checks if the current move leads to checkmate or stalemate, and ends the game if necessary.
- Turn Management: The system tracks whose turn it is and updates the game state accordingly.

4.2 Negamax Algorithm Implementation

The Negamax algorithm is implemented in the backend as the decision-making engine for the AI opponent. The algorithm uses the following process to determine the optimal move:

1. Depth-Limited Search: The algorithm performs a recursive search to evaluate potential future board states up to a specified depth limit. The depth limit increases with the difficulty level.
2. Alpha-Beta Pruning: As the algorithm explores the game tree, it applies alpha-beta pruning to eliminate branches that cannot lead to better outcomes, reducing computational complexity and allowing for deeper searches.
3. Move Evaluation: Each move is evaluated based on a scoring system that considers material count, positional factors (such as control of the center), and tactical opportunities (such as forks or pins).

4.3 Multiplayer Implementation

The multiplayer system is implemented using Socket.IO for real-time communication between clients and the server. The server is responsible for:

- Synchronizing Game States: Ensuring that both players' boards reflect the same moves.
- Validating Moves: The server validates each move before broadcasting it to the opponent.

- Session Management: The server tracks each player's session, ensuring that players can reconnect and resume their game if they are disconnected.

4.4 User Interface

The user interface is built using React.js and provides an interactive 8x8 chessboard. The UI allows players to:

- Drag and Drop Pieces: Players can select and move pieces by dragging them across the board.
- Real-Time Updates: The board updates in real-time to reflect moves made by both players or the AI opponent.
- Puzzle Mode: In puzzle mode, the UI displays a static board with a tactical problem, and players must find the best move to solve the puzzle.

4.5 Testing and Debugging

Extensive testing was conducted to ensure the engine functions correctly under different scenarios. Unit tests were created for:

- Move Validation: Ensuring that illegal moves are blocked.
- AI Decision-Making: Verifying that the AI makes optimal decisions based on the board evaluation.
- Multiplayer Synchronization: Testing real-time synchronization between two players.

V. RESULTS

The chess engine, utilizing the Negamax algorithm with alpha-beta pruning, was thoroughly tested in various environments and under different conditions to assess its performance, functionality, and user experience. Key areas of focus for evaluation included the AI's ability to generate optimal moves, the responsiveness of the multiplayer mode, and the effectiveness of the puzzle and hint systems. Below is a detailed analysis of the results obtained:

5.1 AI Performance

The Negamax algorithm enabled the chess engine to make intelligent decisions at various difficulty levels. The AI's performance was evaluated on three key parameters: response time, move quality, and scalability.

- Response Time: Thanks to alpha-beta pruning, the AI was able to evaluate positions and generate moves in less than a second for low to medium difficulty settings. Even at higher depths, the AI responded within 1-2 seconds, maintaining a smooth user experience without noticeable lag.
- Move Quality: At higher difficulty levels, the AI consistently played optimal moves, often exploiting tactical errors made by human players. It successfully identified and executed tactics such as forks, pins, and checkmates in a few moves, demonstrating the engine's strength in understanding both positional and tactical play.
- Scalability: The AI adapted well to different difficulty settings by adjusting the depth of its search tree. For beginners, the engine played at a reduced depth, focusing on immediate gains rather than long-term strategy. At higher levels, the AI was more strategic, considering multiple future moves and employing a longer-term plan.

5.2 Multiplayer Functionality

The multiplayer functionality was tested to ensure smooth, real-time gameplay between two human players. The following results were observed:

- Latency and Synchronization: Using Socket.IO, the chess engine successfully maintained real-time synchronization between players, with no significant delay between moves. The system handled multiple concurrent games efficiently, providing a seamless experience for all users.
- Stability: The engine maintained stability during gameplay, with session
- management allowing players to reconnect if disconnected and continue their game without losing progress.
- Scalability: The server successfully managed multiple game sessions at once without any noticeable performance issues. The engine was able to handle the influx of moves from various users while ensuring real-time updates and synchronization for each game.

5.3 Puzzle Mode and Hint System

The puzzle mode and hint system were implemented to help users improve their chess skills, and they performed effectively based on user feedback:

- Puzzle Variety: The engine generated a wide range of puzzles, from basic tactical problems like forks and pins to more complex endgame scenarios. The puzzles were dynamically generated, ensuring that users encountered a variety of challenges with each session.
- Hint Accuracy: The hint system, driven by the Negamax algorithm, provided players with optimal suggestions based on the current board state. The hints were not only accurate but also strategic, often guiding players toward better positions without directly giving away the solution.
- Educational Value: User feedback indicated that the hint system helped beginners understand chess strategies better. The system was designed to offer suggestions without revealing the entire move sequence, fostering learning by encouraging players to think through the suggestions.

5.4 User Feedback

User testing was conducted to gather feedback on the chess engine's performance, user interface, and overall experience:

- User Experience: Players found the interface to be intuitive, particularly the drag-and-drop functionality for moving pieces. The real-time updates in multiplayer mode were also praised for their responsiveness.
- AI Competitiveness: Users appreciated the varying difficulty levels of the AI, with beginners finding the lower levels approachable while advanced players found the higher difficulty levels challenging.
- Puzzle and Hint System: Users enjoyed the puzzle mode and found the hints to be valuable for improving their skills. Many reported that the hint system added educational value, helping them think critically about their moves.

VI. CONCLUSION

The chess engine developed using the Negamax algorithm has proven to be a robust, efficient, and engaging solution for both competitive and educational purposes. By incorporating multiplayer functionality, a competitive AI opponent, and an interactive puzzle mode with hints, the engine offers users a comprehensive chess experience.

6.1 Summary of Achievements

- AI Competitiveness: The Negamax algorithm with alpha-beta pruning allowed the AI to generate strong, competitive moves at all difficulty levels. The AI adapted well to different skill levels by adjusting the depth of the search tree, providing an optimal experience for both beginners and advanced players.
- Multiplayer Functionality: The engine successfully implemented real-time multiplayer games with smooth synchronization and minimal latency. Players were able to engage in seamless, competitive matches without interruptions.

- Puzzle Mode and Hint System: The puzzle mode and hint system enhanced the engine's educational value, allowing users to improve their tactical skills while solving chess problems. The AI-powered hints guided players without directly solving the problem for them, encouraging learning and strategic thinking.
- User Interface: The responsive and intuitive user interface, powered by React.js, provided a smooth gameplay experience, whether players were competing against the AI or other users.

6.2 Limitations

Despite the success of the chess engine, there are a few limitations that could be addressed in future iterations:

- Search Depth: While the engine performed well at all difficulty levels, the depth of the Negamax search was limited by computational constraints. Expanding the depth of search for higher levels could make the AI even more formidable for advanced players.
- Puzzle Complexity: The dynamically generated puzzles covered a wide range of tactical themes, but more advanced multi-move puzzles could be added to challenge higher-level players and provide greater diversity.

6.3 Future Work

Future developments of the chess engine could include the following:

- Deeper AI Strategy: Implementing more advanced algorithms or optimizations for the Negamax engine could allow deeper searches and better performance at higher difficulty levels. Techniques like iterative deepening or parallel processing could help improve the AI's strength.
- Integration of Machine Learning: Introducing machine learning techniques, such as reinforcement learning, could make the AI more adaptive and capable of learning from previous games or user behavior.
- Expanded Puzzle Library: More complex puzzles and endgame challenges could be added to provide higher-level players with greater variety and more difficult tactical scenarios.
- Mobile and Cross-Platform Support: Expanding the chess engine to mobile platforms (iOS and Android) would allow a wider range of players to engage with the engine on-the-go.
- Enhanced User Customization: Offering more options for customizing the AI's play style, difficulty settings, and puzzle complexity could make the engine more personalized and tailored to individual users' needs.

6.4 Conclusion

In conclusion, the chess engine developed in this project has successfully demonstrated the power of the Negamax algorithm in building a competitive, educational, and interactive chess platform. The integration of real-time multiplayer functionality, dynamic puzzles, and an AI-driven hint system makes this chess engine an excellent tool for both casual play and skill development. By addressing the limitations and incorporating advanced features, the engine has the potential to evolve into a highly sophisticated and versatile chess-playing platform suitable for users of all skill levels.

ACKNOWLEDGMENT

Special thanks to the department of CSE-AIML of G.H Raisoni College of Engineering and Management, Pune. For the structure provided and the support in carrying out this research.

REFERENCES

1.Schaeffer, T., & Buro, M. (2001). Building a World-Class Checkers Player. AI Magazine.

- This paper discusses the development of a world-class checkers AI using search algorithms and optimization techniques, which influenced the creation of similar approaches in chess engines like Negamax.

2.Knuth, D. E., & Moore, R. W. (1975). An Analysis of Alpha-Beta Pruning. Artificial Intelligence Journal, 6(4), 293-326.

- This landmark paper explains alpha-beta pruning, the technique used to reduce the number of nodes evaluated in the Negamax algorithm. It provides theoretical underpinnings for efficient game tree search in chess engines.

3.Silver, D., Huang, A., & Maddison, C. J. (2016). Mastering the Game of Go with Deep Neural Networks and Tree Search. Nature, 529, 484-489.

- While focusing on Go, this paper demonstrates the application of Monte Carlo Tree Search (MCTS) and neural networks in game AI, offering a parallel to how search algorithms like Negamax are used in chess engines.

4.Russell, S. J., & Norvig, P. (2020). Artificial Intelligence: A Modern Approach (4th Edition). Pearson.

- This textbook provides a comprehensive overview of AI techniques, including Minimax and Negamax, along with alpha-beta pruning and their applications in game theory and decision-making.

5.Stockfish Developers. (2023). Stockfish Chess Engine. Retrieved from https://stockfishchess.org/

- Stockfish is an open-source chess engine that utilizes Negamax and alpha-beta pruning for efficient move generation. It serves as a benchmark for performance in chess AI.

6.Chess.com. (2024). How to Play Chess - The Complete Rules of Chess. Retrieved from https://www.chess.com/learn-how-to-play-chess

- This source provides foundational rules for chess, which were critical in building the chess engine's game logic and move validation system.

7.Python Software Foundation. (2024). Flask Web Framework Documentation. Retrieved from https://flask.palletsprojects.com/en/2.0.x/

- The Flask web framework was used for the backend implementation of the chess engine. This source provides detailed documentation on how to manage web applications using Flask.

8.Palacios, H., & Tavares, L. (2020). Minimax and Negamax Algorithms for Chess AI. Journal of Computational Intelligence and Game Theory, 13(2), 109-123.

- This paper provides an in-depth analysis of the Minimax and Negamax algorithms in chess engines, discussing the efficiency of the algorithms and the benefits of pruning techniques.

9.Coulom, R. (2007). Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search. In Proceedings of the International Conference on Computers and Games, 72-83.

    - This work focuses on Monte Carlo Tree Search and selectivity in game search algorithms, which inspired the selective pruning used in Negamax to optimize game tree evaluation.