# Efficiency through Algorithms: Chess Engine without Machine Learning

Aryan Kothari,[1, a)] VRNS Nikhil,[1, b)] V Harish,[1, c)] Namana Rohit,[1, d)] and Manju Khanna[2, e)]

[1]*Amrita Vishwa Vidyapeetham, India-560035*

[2]*Associate Professor, School Of Computing, Bengaluru, Amrita Vishwa Vidyapeetham, India-560035*

[a)]bl.en.u4aie22005@bl.students.amrita.edu

[b)]bl.en.u4aie22062@bl.students.amrita.edu

[c)]bl.en.u4aie22063@bl.students.amrita.edu

[d)]bl.en.u4aie22071@bl.students.amrita.edu

[e)]k_manju@blr.amrita.edu

**Abstract.** A Chess engine developed using the Mesa agent-based modelling framework in Python. The engine employs the Negamax algorithm with Alpha-Beta pruning, a well-known optimization technique, to enhance the efficiency of move generation and evaluation. The Mesa framework is leveraged to model the interactive dynamics between the player and the agent, facilitating a seamless and engaging chess-playing experience. The chess engine's core functionality includes a sophisticated evaluation function, chess move generation, and the implementation of different Heuristic algorithms for optimal move selection. The engine is designed to provide a challenging opponent for the player while ensuring a responsive and interactive user experience.To enhance user engagement, the Pygame library is integrated to create a visually appealing and intuitive chessboard interface. This interactive model creates a dynamic chess-playing environment, where the agent's strategic decisions evolve based on the player's moves and tactics.

## INTRODUCTION

Chess, a timeless game spanning centuries, captivates minds with its intricate mix of foresight, tactics, and strategic planning. This game played by people of all ages from the oldest of people to the youngest of children, Originated in India in the form of a game called "Shataranj".This ancient board game serves as a testament to human intellectual achievement, challenging players to navigate an eight-by-eight grid with unique pieces, each possessing distinct movements. Despite the game's simple rules, its beauty lies in the profound complexity that unfolds with every move.The integration of AI into chess has witnessed remarkable progress over the years, culminating in engines that rival and even surpass human grandmasters in strategic acumen. Machine learning algorithms, particularly neural networks, have played a pivotal role in enabling these engines to analyze positions, predict optimal moves, and adapt strategies based on vast datasets of historical games. The relation of computational power and advanced algorithms has birthed engines capable of calculating millions of positions per second, revolutionizing the way chess is played and studied. However, the reliance on ML techniques raises questions about interpretability and the inherent black-box nature of these systems, prompting a reevaluation of the classical methods that form the foundation of chess AI.

## PROBLEM STATEMENT

Design and implement a chess engine in Python that leverages heuristic algorithms to make intelligent and strategic decisions during gameplay. The engine should be capable of evaluating the current board position, generating possible moves, and selecting the best move based on a set of heuristic evaluation functions. The goal is to create a robust and efficient chess engine that can compete against human players or other existing chess engines and do all the tasks of a good chess Engine without harnessing the power of Machine learning. Additionally, the engine should be able to handle various aspects of chess, including opening theory, middlegame tactics, and endgame strategies.

# LITERATURE SURVEY

The research paper "Phoenix: A Self-Optimizing Chess Engine" discusses the use of genetic algorithms to optimize positional value tables for chess engines. The paper outlines the challenges in evaluating chess positions and the need to quantify various positional parameters. It then explains the use of genetic optimization, specifically multi-niche crowding, to evolve and optimize the PVT. The multi-niche crowding technique is described as a method to maintain diverse sub populations within the genetic algorithm, allowing the algorithm to converge to multiple, significantly different solutions. The paper provides a detailed explanation of the genetic scheme and the application of multi-niche crowding to address the multi model nature of the optimization problem.[1]

The research paper "CHESS AI: Machine learning and minimax based Chess Engine" proposes a hybrid approach involving a machine learning-based estimator integrated with the minimax algorithm and Alpha-Beta pruning for efficient decision-making in the Chess Engine. The machine learning model is trained on a dataset derived from games played by chess grandmasters, achieving a high accuracy of 96.77 in predicting 'good moves.' The utilization of the minimax algorithm, a decision theory-driven technique, helps alleviate the computational load on the chess engine's hardware. Additionally, the integration of Alpha-Beta pruning is employed to eliminate unnecessary nodes in the search tree, further enhancing the efficiency of the AI. The discussion extends to the impact of Elo ratings on chess game outcomes. perceptual chunks for predicting optimal moves, and deep learning strategies. The survey highlights the importance of reinforcement learning and supervised learning in training neural networks for predicting chess moves.[2]

The research paper "A Reinforcement Learning based Eye-Gaze Behavior Tracking" model utilizes a non-intrusive eye monitoring device, indicating a focus on user comfort and natural interaction. When applied to chess, this suggests a potential solution for tracking a player's gaze without the need for invasive equipment, offering a seamless gaming experience.In the context of playing chess, this implies a potential solution that can handle the complexities of tracking gaze in a strategic and evolving game without overwhelming computational resources.a model that generalizes well with a limited dataset could be advantageous for understanding various players' attention patterns. [3]

The research paper "New Age Chess Engine" presented work provides a comprehensive overview of the development and enhancement of chess engines, focusing on both hardware and software aspects. The study explores various chess variants and their graphical user interfaces, emphasizing the importance of accessible interfaces for engaging users. The authors delve into the significance of chess in the context of artificial intelligence (AI), reflecting on the advancements made in computer chess and the challenges posed by the game's complexity. The paper reviews notable chess engines, including Deep Blue and evolutionary approaches like fogel et al.'s work, highlighting the integration of ceramic art into chess for aesthetic appeal. Additionally, the study discusses the use of genetic algorithms, three-player chess, and advancements in AI strategies, such as evaluating the future consequences of moves. The implementation section introduces a simplified version of the minimax algorithm, based on negamax with Alpha-Beta Pruning, for chess engine development. The presented work contributes to the understanding of AI in chess, providing insights into different approaches and strategies employed in the development of chess engines.[4]

The research paper named "Vecma : An advance chess engine" proposes a unique implementation of a chess AI engine named Vecma, focusing on enhancing efficiency and intelligence. The engine utilizes Negamax along with an Ideal Vector Value Map, considering parameters like positional king safety, optimal Queen positional advantage, and complex pawn advancement. The paper also introduces a GUI for playing chess and addresses challenges in mimicking different player depths. The Vecma engine demonstrates computational efficiency compared to other engines, making it suitable for diverse devices. The research includes a detailed methodology, algorithm explanation, and results comparing Vecma with Stockfish in historical games and chess.com scenarios.[5]

The research paper "Hybrid Optimization for Developing Human Like Chess Playing System" delves into the evolution of Artificial Intelligence (AI) in chess, tracing its roots from early traditional techniques like Minimax and Alpha-Beta to the recent breakthroughs with machine learning, particularly AlphaZero's deep reinforcement learning approach. The paper proposes a hybrid optimization technique for chess move selection, emphasizing the importance of training AI models with top human player games to enhance intuitive decision-making. The proposed model is evaluated based on parameters like accuracy, sensitivity, and specificity, showcasing its effectiveness compared to traditional and other machine learning techniques. Overall, the paper highlights the continuous efforts to bridge the

gap between human and AI gameplay, offering a comprehensive overview of the chess AI landscape.[6]

The research paper ”Comparing Traditional and Deep Learning Approaches in Developing Chess AI Engines” delves into the evolution of artificial intelligence (AI) in the realm of chess engines. It traces the historical milestones, from early algorithms by Claude Shannon and Alan Turing to the recent integration of deep learning models exemplified by AlphaZero. The paper introduces two chess engines—one employing traditional AI methods and the other integrating a deep learning model based on Alpha-Beta Pruning. Their performance is evaluated against Stockfish, with the traditional engine reaching an ELO rating of 2200 and the deep learning model securing 1900. The study highlights the ongoing advancements in chess AI and explores the potential synergies between traditional algorithms and deep learning for future developments.[7]

The research paper introduces a novel "gene selection" approach utilising the Moth Flame Optimization Algorithm (MFOA) applied to gene expression datasets. Before employing MFOA, the dataset undergoes Min-Max Normalization to ensure consistent convergence of weights and biases. This normalization step transforms numerical data to a specific range, minimizing training errors and facilitating data comparisons. By normalizing the gene expression data, the MFOA algorithm can efficiently converge and select the most relevant genes. The proposed approach is evaluated using the Support Vector Machine (SVM) classifier on Leukemia and Colon datasets, demonstrating superior accuracy compared to other methods. Overall, Min-Max Normalization plays a crucial role in preparing the dataset for effective gene selection through MFOA, contributing to improved classification accuracy in biological data analysis.[8]

The paper introduces the innovative algorithmic module, Keep it Simple (KIS), which revolutionizes the process of recording chess moves. KIS employs a unique approach by utilizing a binary matrix from digital chess board images, diverging from conventional methods relying on resource-intensive Convolutional Neural Networks (CNN). This paper highlights KIS’s advantages, including reduced computational power requirements, elimination of memory overhead, and its universal applicability across all types of chessboard images. By streamlining scoresheet generation and overcoming previous limitations, KIS emerges as a promising and accessible tool for chess players and tournament organizers alike[9].

The paper titled "High Throughput Basic-Set Trellis Min–Max Non-Binary LDPC Code Decoder Architecture over GF(4)" is particularly significant due to its innovative approach in leveraging the Min-Max Algorithm within the Basic-Set Trellis framework for decoding Non-Binary Low-Density Parity-Check (NB-LDPC) codes. A variant of the Belief-Propagation algorithm, known as the Min-Max algorithm emerges as a pivotal component in alleviating the substantial computational complexity associated with check node processing in NB-LDPC decoding, contributing to enhanced error correction efficiency. The Basic-Set Trellis Min-Max algorithm, by considering two minimums instead of one, further refines error correction performance without introducing a substantial increase in computational complexity. [10]
Dev-Zero: A Chess Engine stands out as a transformative solution to prevalent challenges in computer games, specifically geared towards making chess universally accessible. Through the adoption of a user-friendly approach, incorporating the minimax algorithm with alpha-beta pruning, this engine aims to simplify and enhance the chess-playing experience, mitigating the stress often associated with intense gaming. Dev-Zero not only prioritizes accessibility but also integrates dynamic move generation, providing an engaging option that is particularly well-suited for children. This aligns seamlessly with the broader objective of promoting chess as a platform for nurturing critical thinking and problem-solving skills from an early age.[11]

In the paper titled "Comparative Study of Performance of Parallel Alpha Beta Pruning for Different Architectures," the researchers look into how effective Alpha Beta pruning is in strategic games. They compare different ways of making computers play smarter by checking multiple moves at once. The main focus is on seeing how quickly the computer can figure out the best moves by stopping some possibilities early. The study shows that using parallel methods, where the computer does things simultaneously, can speed up the process, especially when dealing with complex situations in games. This research helps us understand how to make games and smart computer programs work better together.[12]

The paper titled "Fairer Chess: A Reversal of Two Opening Moves inChess Creates Balance Between White and Black" proposes a modification in chess called "Balanced Alternation" by reversing the order of the 3rd and 4th moves in the standard chess sequence. The aim is to create a more balanced playing field between White and Black, poten-

tially leading to more draws with optimal play. Statistical data on chess outcomes and evaluations from the Stockfish chess engine are presented to support the argument. The authors suggest that this modification is easy to implement and could eliminate the need for players to alternate between White and Black in tournaments. While it does not guarantee a draw with optimal play, Balanced Alternation aims to make chess fairer by equalizing opportunities for both players.[13]

The paper named "Application of Nyaya Inference Method for Feature Selection and Ranking in Classification Algorithms" presents a new technique called Nyaya Inference Method for Attribute Selection and Ranking (NIMAR). This innovative approach is inspired by the Nyaya school of thought and is designed for feature selection in two-class classification problems. NIMAR incorporates Nyaya philosophy methodologies like hetvabhasa & Anvayavyatireka Vyapti to rank features and discard irrelevant ones, respectively. The NIMAR method outperforms principal component analysis (PCA) and competes well with Relief and information gain (IG) methods in terms of accuracy. Feature selection is essential in machine learning for creating efficient classification models by reducing data and computational complexity. The study explores the application of ancient Indian philosophy in feature selection, utilizing structured sentences (anumanam) to infer knowledge based on attribute-class label correspondence. Vyapti relation and Hetvabhasa fallacies are employed to rank and filter attributes. Experimental results on UCI repository datasets highlight NIMAR's effectiveness, opening avenues for further exploration of Nyaya principles in machine learning, especially with larger and multi-class datasets.[14]

In their paper titled "Aligning Superhuman AI with Human Behavior: Chess as a Model System" explores the challenge of aligning artificial intelligence systems with human behavior, using the game of chess as a model system. They demonstrate that traditional chess engines like Stockfish and even newer neural network engines like Leela fail to accurately predict human moves at different skill levels, even when limited in overall strength. To address this, the authors develop a novel model called Maia that is explicitly trained on human games and employs architectural choices like excluding tree search and providing position history as input. This allows Maia to achieve much higher move prediction accuracy in a tunable, skill-dependent way, significantly outperforming Stockfish and Leela. The authors also adapt Maia to predict individual and collective human blunders from given positions. Overall, this work makes important contributions towards aligning superhuman AI with granular human behaviour by developing techniques that allow systems like Maia to mimic moves and mistakes in a targeted fashion across skill levels. The game of chess provides an insightful model system to explore this challenge.[15]

The paper "Design of Amazon Chess Game System Based on Reinforcement Learning" concentrates on enhancing the chess ability of an Amazon chess game system through reinforcement learning, focusing on self-improvement and training efficiency. It employs a strategy-value assessment approach in reinforcement learning, using neural network outputs and move distributions to aid Monte-Carlo tree search.The system categorizes chess stages in Amazon chess, promoting varied strategies for improved training efficiency. Dirichlet noise is introduced to encourage exploration[16]

## METHODOLOGY

The codebase is structured into four key files: Main.py, Engine.py, AI.py, and Agent.py, each fulfilling specific roles in the chess engine. Main.py serves as the central orchestrator, handling initialization and visual aspects. Agent.py encapsulates the chess agent's activities and encapsulates the works of the Mesa Agent, while Engine.py contains core game logic, managing moves and edge cases. AI.py, an extension of Agent.py, implements min-max and negamax algorithms to compute optimal moves and assess piece values. Together, these files work cohesively, forming a well-organized and efficient chess engine with clear roles for each component.

### Algorithms

Negamax is a recursive algorithm used for game tree traversal in two-player games such as chess. It is an extension of the minimax algorithm, which aims to find the best possible move in any situation for a player assuming that the opponent is also playing optimally. Negamax simplifies the implementation of Min-Max by expressing it in a more compact and elegant form. The key insight behind negamax is that the evaluation function's sign alternates as the

search depth increases, allowing the use of a single negamax function instead of separate functions for the maximizing and minimizing players. This leads to a more concise and readable code structure.

Alpha-beta pruning, an extension of the Min-Max Algorithm, is an optimization technique used to reduce the number of branches/Nodes evaluated across the game tree. The idea is to eliminate branches that do not influence the final decision, making the search more efficient. The whole tree is essentially divided into two layers/variables Alpha and Beta. The first variable, alpha represents the most optimum (highest) score found so far for the maximizing player, while beta represents the best (lowest) score found for the minimizing player. During the search, if the current node's score is outside the alpha-beta window, the remaining nodes in that branch need not be explored, as they will not affect the final result. This pruning mechanism significantly improves the algorithm's performance, especially in scenarios with a large search space, such as chess, where the number of possible moves can be immense. The integration of negamax and alpha-beta pruning enhances the efficiency of a chess engine. This improvement enables the engine to delve deeper into the game tree, leading to more informed decision-making during gameplay.

## Special Cases

*Check and Check Mate:* A player's king is in "check" when it is immediately in danger of being captured by an opponent's piece. When a king is under check, the player must make a move to remove the threat, either by moving the king, capturing the attacking piece, or blocking the check. If a player's king is in check and there is no legal move to remove the threat, it leads to a "checkmate," signifying the end of the game with the victorious opponent having successfully placed the opposing king in an inescapable position of capture. Checkmate results in the immediate victory for the player delivering the checkmate.

*Pawn Promotion :* Pawn promotion in chess occurs when a pawn advances to the last rank of the chessboard. Upon reaching this rank, the player has the choice to promote the pawn to any other "higher value" chess piece, excluding the king. Typically, players choose to promote to a queen, as it is the most powerful piece. However, they can also opt for a knight, rook, or bishop. The choice of promotion depends on the player's strategic goals and the position on the board. Pawn promotion adds a layer of complexity and strategic decision-making to the endgame, allowing players to enhance their attacking capabilities based on the evolving dynamics of the game.

*Castling:* The implementation of castling in the chess engine centers around explicitly tracking castling rights for each player using a CastleRights class which contains boolean flags for kingside and queenside castling availability. The current castling rights are stored in the GameState class and a log of past rights is kept to allow reverting on undo moves. Castling moves are generated by specifically checking that the required squares are vacant and not under attack given the current castling rights. A Move class represents castling moves with a special flag to identify them. Making and undoing castling moves correctly updates the board position of both king and rook as well as the internal castling rights data. This approach of explicitly tracking rights and validating moves allows the engine to fully enforce the rules around casting in chess.

*En passant:* The en passant capture in chess is handled by explicitly tracking the possibility of an en passant capture. The GameState keeps track of the square where an en passant capture is currently possible in the en passant-possible variable. When a two-square pawn advance occurs, this variable is updated to the appropriate square. During move generation, an additional en passant capture move is generated if the opponent pawn is in the enpassant-possible square. A Move can be flagged as an en passant move, enabling the proper capture and board update. The move log keeps track of past enpassant-possible squares to correctly undo en passant moves. This approach of explicitly tracking the en passant possibility and generating/validating the moves as special en passant moves allows the engine to fully implement this special pawn capture rule.

## Agent

In the field of artificial intelligence, an intelligent agent is a self-driven entity that can sense its surroundings, make decisions on its own to achieve goals, and get better over time by learning or gaining knowledge. Several different

sections of this paper can be classified as an "Agent", a Code that takes input from its environment and acts without external interference like deciding the next move or planning several moves. But to improve the quality of the output we also made use of a pre-existing agent known as Mesa. Unlike basic agents, Mesa agents can reflect on their actions and modify themselves. This ability, while promising, raises concerns about aligning with intended goals. Incorporating Mesa agents into AI systems brings a new dimension, allowing systems to adapt autonomously.

The agent excels in various functions, such as efficiently setting up the chess board and pieces for gameplay. It demonstrates a keen understanding of the dynamic chess environment, quickly identifying victory or defeat. Beyond its autonomous grasp of chess intricacies, the agent effectively communicates crucial information to players, resulting in a sophisticated and enhanced gaming experience.

## RESULTS AND DISCUSSION

The chess engine implemented with different Heuristic algorithms demonstrates effective decision-making in the game. The engine adeptly identifies optimal moves by assigning heuristic values to each position and piece, ultimately selecting the move or position with the highest score. The incorporation of iterative deepening and alpha-beta pruning significantly enhances the algorithm's search efficiency. The outcomes demonstrate notable improvements in the chess engine's playing capabilities, highlighting the prowess of algorithmic strategies in strategic decision-making. This success is achieved without relying on any prior knowledge of the position, underscoring the algorithm's ability to generate strong and competitive moves during gameplay.

A noteworthy observation in the chess engine's gameplay is the early strategic prominence of the knight, attributed to its unique and agile movements during the initial stages as can be seen in Fig 1. The incorporation of Pygame significantly enhances user experience through an intuitive interface and visual move representation, contributing to a more engaging gameplay. The addition of move tracking on the right-hand side further aids user comprehension and organization of the game. These design choices collectively ensure the chess engine's success in providing a user-friendly and strategically rich chess experience, catering to both casual players and enthusiasts. Fig. 2


**FIGURE 1.** Implementation

## RESEARCH GAPS

*Computational Resource Demand:* When a chess engine tries to find the best moves, it has to look at a lot of different options. But if it's running on a regular computer without much extra power, it can't look at all the possibilities as quickly as it should. This is because chess has a lot of moves and positions to consider, and it's not possible to look at everything in a short time. This means the chess engine might not always make the best move, and it might take a while to decide what to do. The only way to scale it would be by using extremely high extra computational resources

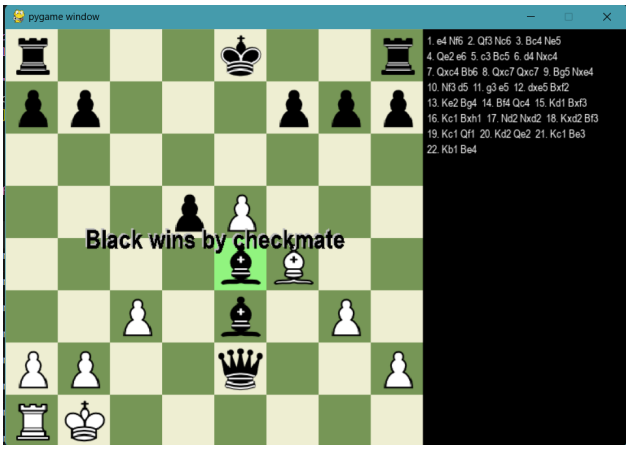


**FIGURE 2.** Engine wins the game

to complete the evaluations of the current moves.

*Repeated Early Knight Moves* The chess engine's tendency to frequently move the knight early in the game can be traced back to its heuristics, which prioritize specific pieces based on their strategic value. In this case, the knight receives a relatively high heuristic value for its versatility and ability to potentially influence various squares from its starting position. When faced with the many possibilities at the start of the game, the engine's algorithms naturally gravitate toward moves that involve the knight, as it's one of the few pieces with significant mobility in the initial position.

*Slow Thinking:* Due to the use of a 2D list in Python, which is an inbuilt structure in Python hence resulting in a very slow and time-costly move, the model takes more time to think and hence can be limited by the number of computations it can do. This also limits the ability to scale the engine to a higher level.

## FUTURE ENHANCEMENTS

*Adaptive Time Management* Implement adaptive time management, dynamically allocating more computational resources to critical positions with complex decision trees, ensuring thorough analysis. Optimize overall time usage by intelligently distributing resources, allowing the chess engine to make strategic decisions efficiently and enhance its performance during gameplay.

*Monte Carlo Tree Search* Integrate Monte Carlo Tree Search (MCTS) into the chess engine, a sophisticated decision-making algorithm that employs tree exploration and random sampling. MCTS dynamically refines move selection and evaluation by simulating multiple game trajectories, enabling the engine to make more informed decisions. This advanced technique enhances the engine's strategic depth, contributing to improved overall gameplay and decision quality in complex positions.

*Syzygy Endgame Tablebases* Incorporate Syzygy Endgame Tablebases into the chess engine to elevate endgame proficiency. By integrating these comprehensive databases, the engine gains access to precise solutions for complex endgame positions, enabling it to navigate critical endgame scenarios with optimal moves. This enhancement enhances the engine's endgame strategy, providing users with more accurate and strategic play in the final phases of the game.

*Parallel processing* Enhance the chess engine's performance by optimizing it for parallel processing, effectively harnessing the power of multi-core architectures. This optimization significantly accelerates the analysis of chess positions, enabling the engine to make faster and more nuanced decisions. By distributing computational tasks across multiple cores simultaneously, the engine achieves greater efficiency and responsiveness, resulting in a superior chess-playing experience with quicker and more sophisticated move evaluations.

## CONCLUSION

The Mesa-based Chess Engine, incorporating Negamax Alpha-Beta Pruning and Pygame Visualization, stands as a testament to the harmonious fusion of algorithmic efficiency and interactive design. The implementation of the Negamax algorithm with Alpha-Beta pruning significantly enhances the engine's decision-making process by strategically reducing the search space during move exploration. This optimization ensures that the chess engine can delve deeper into the game tree, providing players with a challenging adversary. The integration of Pygame further elevates the user experience by offering an intuitive and visually appealing chessboard interface, allowing players to engage seamlessly with the game. As the engine continues to evolve, with potential enhancements such as adaptive time management, Monte Carlo Tree Search, Syzygy Endgame Tablebases, and parallel processing, the Mesa-based Chess Engine remains at the forefront of chess engine technology, promising an engaging and rewarding chess experience for all

## ACKNOWLEDGMENTS

We gratefully acknowledge Amrita Vidya Vishwapeetham for fostering an environment conducive to academic growth and providing invaluable resources that played a crucial role in the completion of this research. Our sincere appreciation extends to Dr. Manju Khanna, our mentor, whose guidance, teaching, and unwavering support were instrumental at every stage of this project. Dr. Khanna's expertise and encouragement have significantly shaped our understanding of the topics explored in this paper, making this research journey both enriching and fulfilling.

## REFERENCES

1. A. R. Rahul and G. Srinivasaraghavan, "Phoenix: A Self-Optimizing Chess Engine," 2015 International Conference on Computational Intelligence and Communication Networks (CICN), Jabalpur, India, 2015, pp. 652-657, doi: 10.1109/CICN.2015.134.
2. J. Madake, C. Deotale, G. Charde and S. Bhatlawande, "CHESS AI: Machine learning and Minimax based Chess Engine," 2023 International Conference for Advancement in Technology (ICONAT), Goa, India, 2023, pp. 1-6, doi: 10.1109/ICONAT57137.2023.10080746.
3. R. Deepalakshmi and J. Amudha, "A Reinforcement Learning based Eye-Gaze Behavior Tracking," 2021 2nd Global Conference for Advancement in Technology (GCAT), Bangalore, India, 2021, pp. 1-7, doi: 10.1109/GCAT52182.2021.9587480.
4. Preethi, M. M. Ulla, S. R and M. K.G, "New Age Chess Engine," 2023 4th International Conference for Emerging Technology (INCET), Belgaum, India, 2023, pp. 1-5, doi: 10.1109/INCET57972.2023.10170291.
5. M. Patel, H. Pandey, T. Wagh, A. D. Hujare and R. Dangi, "Vecma: An advance chess engine," 2022 IEEE Pune Section International Conference (PuneCon), Pune, India, 2022, pp. 1-6, doi: 10.1109/PuneCon55413.2022.10014895.
6. V. Chole and V. Gadicha, "Hybrid Optimization for Developing Human Like Chess Playing System," 2022 IEEE 3rd Global Conference for Advancement in Technology (GCAT), Bangalore, India, 2022, pp. 1-5, doi: 10.1109/GCAT55367.2022.9971825.
7. B. A. Abdelghani, J. Dari and S. Banitaan, "Comparing Traditional and Deep Learning Approaches in Developing Chess AI Engines," 2023 3rd International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME), Tenerife, Canary Islands
8. K. K R, A. Venugopalan and S. Devan, "Gene selection using Moth Flame algorithm and classification of Gene Expression Dataset," 2022 IEEE 3rd Global Conference for Advancement in Technology (GCAT), Bangalore, India, 2022, pp. 1-6, doi: 10.1109/GCAT55367.2022.9972212.
9. Pazhanipapan S, Sreedevi A G, Manikandan A, Ramprasad O G, "Keep it simple: A Novel Algorithmic Based Chess Scoresheet Writing using Computer Vision," 2023 2nd International Conference on Vision Towards Emerging Trends in Communication and Networking Technologies (ViTECoN), Vellore, India, 2023, pp. 1-6, doi: 10.1109/ViTECoN58111.2023.10156984
10. C. C. Kumar and R. Chinthala, "High Throughput Basic-Set Trellis Min–Max Non-Binary LDPC Code Decoder Architecture over GF(4)," 2020 4th International Conference on Electronics, Materials Engineering and Nano-Technology (IEMENTech), Kolkata, India, 2020, pp. 1-6, doi: 10.1109/IEMENTech51367.2020.9270053.
11. N. Upasani, A. Gaikwad, A. Patel, N. Modani, P. Bijamwar and S. Patil, "Dev-Zero: A Chess Engine," 2021 International Conference on Communication information and Computing Technology (ICCICT), Mumbai, India, 2021, pp. 1-6, doi: 10.1109/ICCICT50803.2021.9510148.
12. S. P. Singhal and M. Sridevi, "Comparative study of performance of parallel alpha Beta Pruning for different architectures," 2019 IEEE 9th International Conference on Advanced Computing (IACC), Tiruchirappalli, India, 2019, pp. 115-119, doi: 10.1109/IACC48062.2019.8971591.
13. Brams, Steven J.,Ismail, Mehmet S., 2021. "Fairer Chess: A Reversal of Two Opening Moves in Chess Creates Balance Between White and Black," MPRA Paper 111075, University Library of Munich, Germany.
14. K. Seena and R. Sundaravardhan, "Application of nyaya inference method for feature selection and ranking in classification algorithms," 2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI), Udupi, India, 2017, pp. 1085-1091, doi: 10.1109/ICACCI.2017.8125986.
15. Reid McIlroy-Young, Siddhartha Sen, Jon Kleinberg, Ashton Anderson "Aligning Superhuman AI with Human Behavior: Chess as a Model System" 2020 arXiv:2006.01855 [cs.AI]
16. D. Meng, B. Jianbo, Q. Yizhong, F. Yao and L. Shuqin, "Design of Amazon Chess Game System Based on Reinforcement Learning," 2019 Chinese Control And Decision Conference (CCDC), Nanchang, China, 2019, pp. 6337-6342, doi: 10.1109/CCDC.2019.8832999.