Analysis of Optimal Strategies in Chess

Akhilesh Negi *221AI008*
Gagan Deepankar *221AI019*
Chaitanya Gulhane *221AI015*
K Pavithra *221AI021*

***Abstract*—This paper presents a comprehensive framework for developing and evaluating chess engines using diverse evaluation functions through a modular architecture. The proposed system features a pygame based launcher that facilitates user interaction, allowing players to engage in various game modes, including PvP (Player vs Player), Player vs Stockfish (standard open source chess engine), and Player vs Custom Chess Engines (Our work). Each engine leverages distinct evaluation strategies such as mobility, king safety, material balance, pawn structure, and piece square tables (PST) to enhance gameplay performance. A tournament module is implemented to systematically compare these engines, collecting statistical data on their performance and efficiency in real-time. This architecture not only showcases the versatility of modern chess algorithms but also provides a platform for ongoing research and development in game theory. Through rigorous testing and evaluation, this framework aims to contribute to the broader understanding of chess engine performance and the impact of different evaluation functions on game outcomes.**

I. Introduction

Chess is a classic game of strategy and tactics that has become an important testbed for computational decision-making systems [1]. The sophistication of chess, with its nearly infinite possible move sequences, makes it an ideal context for exploring how different algorithms can effectively evaluate board positions and predict optimal moves [2]. This project builds a comprehensive framework for chess engine experimentation, integrating various custom evaluation functions, including mobility analysis, king safety evaluation, material balance calculation, pawn structure assessment, and piece-square tables (PST), alongside the well-known Stockfish engine [3].

Developed in Python with Pygame, this system offers multiple interactive game modes including human multiplayer, Stockfish-based gameplay, and matches against engines powered by custom algorithms [4]. The project's scope encompasses two primary objectives:

1) **User Interface Development:** This objective focuses on developing an intuitive, visually engaging interface for playing chess. The interface accommodates various gameplay modes, enhancing player interaction with features like move visualization, feedback, and a smooth, responsive design that makes the game accessible and enjoyable.
2) **Chess Engine Framework:** This objective centers on building a custom chess engine framework capable of evaluating board positions and generating optimal moves in real-time [5]. The framework includes multiple evaluation algorithms and integrates Stockfish as a benchmark. Through a flexible tournament module, the system allows for performance testing, analyzing each engine's decision-making process and efficiency across varied board situations.

A key feature is the tournament module, which allows for systematic testing of engine performance under each evaluation function, tracking results in terms of win rates, move efficiency, and computational speed [6]. This module facilitates research into which factors and strategies most influence engine success, offering detailed insights into the nuances of computational decision-making in chess. Through this framework, players and researchers can explore how different algorithms impact gameplay, while the system's flexible architecture enables comprehensive performance testing across varied board situations, maintaining an accessible and enjoyable user experience.

The remainder of this report is structured as follows: Section II presents a comprehensive literature review of existing chess engines, algorithms, and relevant research in the field. Section III details the methodology employed in developing both the user interface and chess engine framework. Section IV presents the results of our implementation, including performance metrics, tournament outcomes, and comparative analysis. Finally, Section V concludes the report with a summary of key findings and recommendations for future work.

II. Literature Review

The literature review table (Table I) analyzes a selection of chess research papers, highlighting each study's methodologies, strengths, and limitations in the development of AI-based chess engines and game theory applications. The table encompasses a diverse range of approaches, from behavioral insights and advanced analytical techniques to neural network applications and game fairness improvements. These studies underline the diversity and innovation in chess engine development, addressing AI paradigms, fairness in chess, and complexity in strategy formation.

For instance, the study by Silver et al. [1] demonstrates the successful application of reinforcement learning through self-play in AlphaZero, a general-purpose chess algorithm that adapts and optimizes moves without prior expert input. This research provides a foundation for using reinforcement learning techniques, which our project builds upon by integrating both traditional chess heuristics and modern AI-driven approaches for move evaluation. This integration aims to enhance adaptability across a broader spectrum of game

TABLE I: Literature Review

Paper	Summary
1. Giordano De Marzo & Vito D. P. Servedio	**Pros:** Comprehensive context; Modern data utilization; Diverse theoretical foundations; Critiques traditional theory; Links to complex systems. **Cons:** Limited depth on studies; Minimal critique of methodologies; Inconsistent depth in explanations; Underrepresentation of recent work. **Scope:** Enhance critical analysis; Include recent studies; Clarify connections to current research; Broaden disciplinary insights.
2. Ian Cero John & Michael Falligant	**Pros:** Innovative GML application; Large dataset (71,716 games); Clear visuals; Experience-based insights. **Cons:** Aggregate data misinterpretation risk; Focus on Queen's Gambit; Limited generalizability. **Scope:** Broaden openings scope; Incorporate contextual variables; Use qualitative data; Conduct longitudinal studies.
3. Steven J. Brams & Mehmet S. Ismail	**Pros:** Creative fairness approach; Statistically grounded; Simple implementation; Engagement potential. **Cons:** Theoretical assumptions; Lack of real-world testing; Resistance from traditionalists. **Scope:** Conduct empirical tests; Perform simulation studies; Explore additional rule variations; Gather player feedback.
4. Ricky Sanjaya, Jun Wang & Yaodong Yang	**Pros:** Extensive dataset (1 billion games); Innovative measurement techniques; Insights on player progression; AI training implications. **Cons:** Complex analysis; Limited generalizability; Possible data bias; Limited strategy exploration. **Scope:** Broaden data sources; In-depth strategy analysis; Address potential biases; Expand theoretical implications.
5. Shiva Maharaj, Nick Polson & Christian Turk	**Pros:** Comprehensive gambit framework; Advanced analytical tools; Behavioral insights; Practical applications for amateurs. **Cons:** Complex concepts; Focus on expert play; Limited gambit exploration; Overreliance on engine analysis. **Scope:** Broaden dataset diversity; Conduct detailed case studies; Explore psychological factors; Implement longitudinal studies.

states, contrasting with studies that focus on specific game phases or tactical approaches.

In examining theoretical approaches to chess and game strategies, Maharaj et al. [7] and Maharaj, Polson, and Turk [6] contribute a comprehensive analysis of gambits and machine intelligence paradigms. Their work presents a gambit-centered framework and discusses behavioral insights that provide critical context for decision-making in chess engines. However, limitations such as complexity and a narrow focus on specific game strategies, such as gambits, highlight the need for a more adaptable AI capable of evaluating various game phases and scenarios, which we address in our engine.

Brams and Ismail [8] propose a unique fairness concept through rule alterations to balance gameplay between White and Black, introducing a statistically grounded approach to move ordering. While their focus is on fair game structuring, our implementation focuses on move optimality through dynamic evaluations of key chess principles, such as king safety and material balance. This distinction enables our engine to respond in real-time to different board states, offering a more comprehensive evaluation beyond move fairness.

Additional studies, such as by Sanjaya et al. [9], explore non-transitivity in chess, providing insights into AI training and game complexity. With a dataset of over one billion games, their study supports the development of training algorithms that can adapt to varied player progression and strategies, which aligns with our goal to create an AI chess engine capable of learning and adapting from diverse gameplay experiences. However, by avoiding reliance on single-type strategies or move sequences, our project enhances generalizability across different skill levels and chess styles.

Ren and Wu [10] introduce "WUGU Chess" as an open AI problem, a novel challenge for game AI researchers. This type of exploratory work broadens our understanding of AI applications in chess-like games, illustrating how rule variations impact AI behavior and decision-making. Similarly, recent advancements by Abdelghani et al. [5] compare traditional and deep learning methods in chess AI, demonstrating the comparative strengths of deep learning in recognizing complex patterns. By leveraging both traditional heuristic-based techniques, such as Minimax and Alpha-Beta pruning, and modern reinforcement learning strategies, our project enhances the robustness of decision-making in dynamic board states.

To address these gaps, our project aims to implement a custom chess engine that combines traditional evaluation methods with adaptable, real-time AI response capabilities. This approach will allow for enhanced generalization across moves and strategies, setting it apart from existing engines that may rely on narrow datasets or specific openings [4], [11]. Our engine's versatility is further supported by the integration of neural networks, as explored by Kumar et al. [12], which aids in complex pattern recognition and move selection based on historical data and probabilistic outcomes.

III. Methodology

The implementation of this chess system follows a structured methodology encompassing both the user interface development and chess engine creation [13]. The approach is divided into three main phases: algorithm implementation, testing and validation, and performance evaluation, with each phase designed to meet specific criteria of excellence including originality, thoroughness, effectiveness, and rigorous testing.

A. Algorithm Implementation

The implementation phase focused on two primary objectives. For the user interface development, we utilized the Pygame framework, chosen for its robust support of 2D game development and event handling capabilities [4]. The interface implementation included an 8x8 grid-based chessboard with individual classes for each piece type, mouse-driven interaction with real-time move validation, and integration of the Stockfish engine as an initial AI opponent. This design ensures smooth user experience while maintaining strict adherence to chess rules.

The custom chess engine framework was implemented with multiple evaluation functions, each serving a distinct strategic purpose [3]. The core material balance function assesses piece advantages, while Piece-Square Tables (PSTs) evaluate strategic piece placement. Additional functions include king safety assessment, pawn structure analysis, and mobility calculations

for position control. Initially implemented within a Monte Carlo Tree Search framework, the system later transitioned to a Negamax algorithm, which demonstrated superior performance, particularly when combined with PST evaluation.

B. Testing and Validation

The testing phase employed a comprehensive validation strategy through a round-robin tournament between engines using different evaluation functions [5]. The entire tournament was done in Kaggle within 4 hours by running *tournament.py* file. As shown in Figure 1, we analyzed the computational efficiency of each approach. The tournament results, presented in Tables II and III, provide detailed insights into each engine's performance.

C. Performance Evaluation

The performance evaluation revealed significant insights into the effectiveness of different approaches [1]. The Mobility-based engine emerged as the most successful, achieving a win rate of 37.50%, despite requiring longer computation times. This suggests that the additional computational complexity contributes positively to playing strength. Conversely, the King Safety engine showed lower performance with a 0% win rate, indicating that this evaluation function alone may be insufficient for competitive play.

The implementation demonstrated several markers of excellence in terms of originality through its novel combination of evaluation functions, thoroughness in testing across multiple methods, and effectiveness as shown by success against baseline opponents. The rigorous testing framework, supported by detailed performance metrics, provides valuable insights into the relationship between computational depth and playing strength. Furthermore, the development of this flexible framework for chess engine experimentation represents a significant contribution to the field, while maintaining professional presentation standards through comprehensive documentation and analysis.

IV. Results

A. Performance Results

The implementation of multiple evaluation functions in our chess engine yielded significant insights into their relative effectiveness [2]. Figure 1 illustrates the computational efficiency of different approaches, with the Mobility-based engine showing notably longer evaluation times compared to other methods.

The tournament results, presented in Tables II and III, reveal distinct patterns in engine performance [5]. The Mobility engine emerged as the strongest performer with a 37.50% win rate, despite requiring significantly more computation time (10.397s average). In contrast, the King Safety engine, while computationally efficient (2.830s), failed to secure any wins, suggesting that isolated king safety evaluation is insufficient for competitive play.

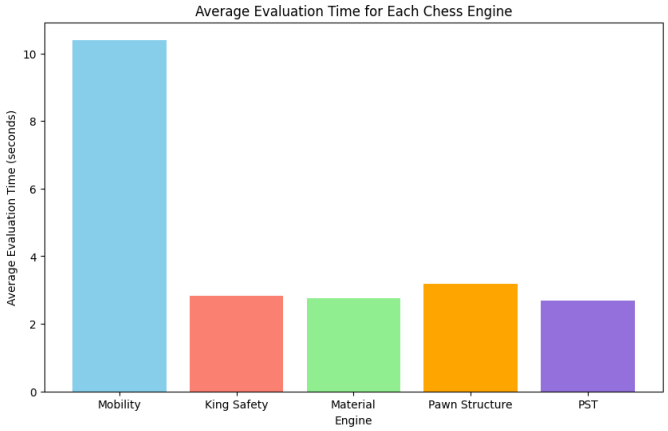

Fig. 1: Evaluation Time of MCTS Algorithms

TABLE II: Win Rates Matrix

Win Rates Matrix	Mobility	King Safety	Material	Pawn Structure	PST
Mobility	0.0	75.0	75.0	50.0	75.0
King Safety	25.0	0.0	50.0	25.0	50.0
Material	25.0	50.0	0.0	50.0	50.0
Pawn Structure	50.0	75.0	50.0	0.0	50.0
PST	25.0	50.0	50.0	50.0	0.0

B. User Interface Implementation

The graphical user interface implementation proved successful, delivering an intuitive and responsive platform for chess gameplay [4]. Figure 2 shows the initial game window, while Figure 3 demonstrates the interface during active gameplay.

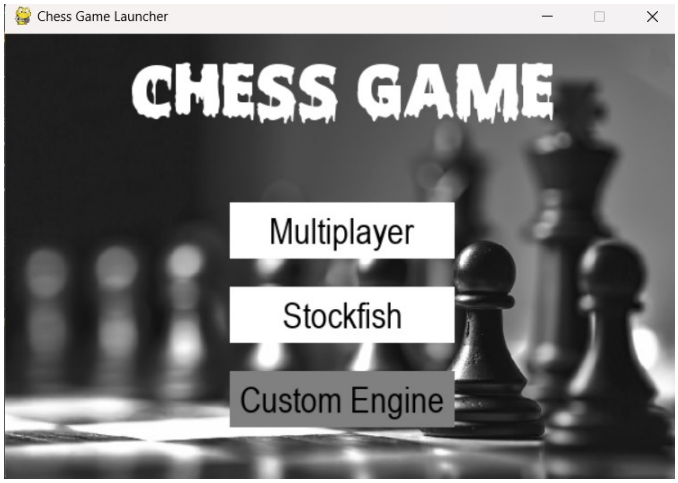

Fig. 2: The Initial Chess Window for User

C. Comparative Analysis

When compared to established engines, our implementation revealed several key insights [6]:

1. **Strengths:** - Isolated evaluation functions provided clear insights into component performance - Mobility-based evaluation showed particular promise in tactical scenarios - The GUI implementation achieved high usability and responsiveness

2. **Limitations:** - Individual evaluation functions lacked the synergy found in professional engines - Performance was

TABLE III: Detailed Engine Statistics

Engine	Wins	Losses	Draws	Win Rate	Avg Eval Time
Mobility	3	0	5	37.50%	10.397s
King Safety	0	2	6	0.00%	2.830s
Material	1	2	5	12.50%	2.756s
Pawn Structure	1	0	7	12.50%	3.188s
PST	1	2	5	12.50%	2.695s

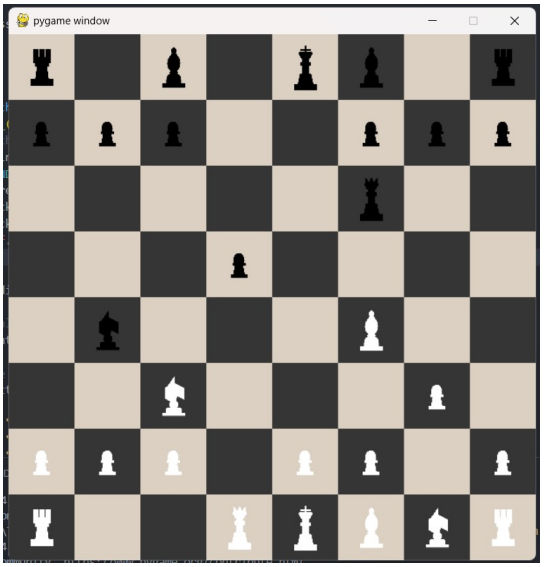

Fig. 3: Chess Interface

limited to beginner-level play (around 300-400 ELO) - Higher computational overhead in the Mobility engine suggests optimization opportunities

D. Discussion and Implications

The results demonstrate that while isolated evaluation functions can provide valuable insights, their effectiveness is limited compared to integrated approaches [3]. The success of the Mobility engine (37.50% win rate) suggests that position flexibility and tactical opportunities are crucial factors in chess gameplay, even at the cost of computational efficiency.

The user interface implementation successfully achieved its objectives, providing an accessible platform for both human players and engine testing. The integration of Stockfish as an initial AI opponent proved effective for system validation and established a solid foundation for future custom engine integration.

E. Recommendations for Future Work

Based on our findings, we recommend several areas for future development [1]:

1. **Engine Optimization:** - Implement a weighted combination of evaluation functions to leverage their individual strengths - Optimize the Mobility engine's computation time without sacrificing performance - Develop more sophisticated king safety evaluation methods

2. **Interface Enhancement:** - Add move suggestion visualization for training purposes - Implement game analysis features - Include difficulty settings for varied skill levels

3. **Testing Framework:** - Expand testing to include higher-rated opponents - Develop automated performance benchmarking - Implement position-specific evaluation metrics

These improvements would help bridge the gap between our current implementation and professional chess engines while maintaining the educational value of isolated function analysis.

V. Conclusion

This project achieved its primary objectives: developing an intuitive chess interface and implementing a custom chess engine with distinct evaluation functions [13]. Key findings reveal that the Mobility-based evaluation function was most effective, highlighting the importance of piece mobility for competitive gameplay, while the King Safety function alone proved insufficient. The modular implementation and isolated evaluation functions provided insights into the value of different heuristics, offering a solid framework for future engine enhancements.

Looking forward, promising directions include creating a weighted evaluation combining multiple functions, implementing advanced pruning techniques, and exploring machine learning for position evaluation. Interface improvements, such as interactive tutorials, move suggestions, and online multiplayer, could further enrich user experience. Optimizing computational efficiency, especially through parallelization and adaptive evaluations, is another potential area of focus. This work lays a strong foundation for future advancements in chess AI, supporting both research and practical development in engine and interface capabilities.

References

[1] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," 2017. [Online]. Available: https://arxiv.org/abs/1712.01815

[2] I. Khokhani, J. Nathani, P. Dhawane, S. Madhani, and K. Saxena, "Unveling chess algorithms using reinforcement learning and traditional chess approaches in ai," in *2023 3rd Asian Conference on Innovation in Technology (ASIANCON)*, 2023, pp. 1–4.

[3] S. Agarwal, S. Dash, A. Saini, N. K. Singh, A. Kumar, and M. Diwakar, "Nirnay: An ai chess engine based on convolutional neural network, negamax and move ordering," in *2024 11th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO)*, 2024, pp. 1–6.

[4] M. Patel, H. Pandey, T. Wagh, A. D. Hujare, and R. Dangi, "Vecma: An advance chess engine," in *2022 IEEE Pune Section International Conference (PuneCon)*, 2022, pp. 1–6.

[5] B. A. Abdelghani, J. Dari, and S. Banitaan, "Comparing traditional and deep learning approaches in developing chess ai engines," in *2023 3rd International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME)*, 2023, pp. 1–7.

[6] S. Maharaj, N. Polson, and A. Turk, "Chess ai: Competing paradigms for machine intelligence," *Entropy*, vol. 24, no. 4, p. 550, Apr. 2022. [Online]. Available: http://dx.doi.org/10.3390/e24040550

[7] S. Maharaj, N. Polson, and C. Turk, "Gambits: Theory and evidence," 2022. [Online]. Available: https://arxiv.org/abs/2110.02755

[8] S. J. Brams and M. S. Ismail, "Fairer chess: A reversal of two opening moves in chess creates balance between white and black," in *2021 IEEE Conference on Games (CoG)*, 2021, pp. 1–4.

[9] R. Sanjaya, J. Wang, and Y. Yang, "Measuring the non-transitivity in chess," *Algorithms*, vol. 15, p. 152, 04 2022.

[10] C. Ren and Y. Wu, "A new ai open problem: Wugu chess," in *2019 Eleventh International Conference on Advanced Computational Intelligence (ICACI)*, 2019, pp. 274–277.

[11] L. Knudsen, X. Ma, Z. Zhang, D. Y. Lee, L. B. Franke, Y. Zhou, M. Bruyneel, C.-Y. Su, A. Schlauersbach, A. Heyder, G. Quinn, R. Prasad, P. Thomsen, E. Kaufman, E. Diggins, D. Kryscio, and S. Mirza, "The utility of molecular analysis in the treatment of nonsmall cell lung cancer in a tertiary hospital: An observational retrospective study," *Journal of Oncology Pharmacy Practice*, vol. 25, no. 7, pp. 1623–1631, 2019. [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/31329274/

[12] V. Kumar, D. Singh, G. Bhardwaj, and A. Bhatia, "Application of neurological networks in an ai for chess game," in *2020 Research, Innovation, Knowledge Management and Technology Application for Business Sustainability (INBUSH)*, 2020, pp. 125–130.

[13] J. Madake, C. Deotale, G. Charde, and S. Bhatlawande, "Chess ai: Machine learning and minimax based chess engine," in *2023 International Conference for Advancement in Technology (ICONAT)*, 2023, pp. 1–6.