CHESS ENGINE

[1] Gautam Mayya, [2] Chandan M, [3]Gagan Y, [4]Degala Gagan, [5]Assoc.Prof.Dinesh Singh.
[1]gaytammayya@gmail.com, [2]chandanmallesh0@gmail.com, [3]gaganak.yalamuri@gmail.com,
[4]degalagagan@gmail.com@gmail.com, [5]dineshs@pes.edu .
[1, 2, 3 ,4 ,5] Department of Computer Science and Engineering,
PES University (560085).

***Abstract*— This abstract discusses a chess engine that utilizes Minimax search with alpha-beta pruning and considers only the top 50 percent of possible moves. The selection of these moves is based on a trained dataset where moves played by the winner are labelled as good, and possible legal moves that could have been played. but weren't labelled as bad. The top 50 percent of moves, ordered by their good percentage, are selected. This approach reduces the number of positions evaluated. by the algorithm, resulting in faster and more efficient computation. The use of a trained dataset to identify the best moves has shown to be effective in improving. the performance of the chess engine. This technique has been used by top computer chess engines, and the continued development of artificial intelligence and machine. Learning will likely lead to even more advanced chess engines in the future.**

[I] INTRODUCTION

Chess is a game of strategy, intelligence, and skill, and has been played for centuries. In recent years, there has been an explosion in the development of computer chess engines, which can play the game at a level that is beyond even the best human players. These engines use advanced algorithms and artificial intelligence techniques to analyze millions of possible moves and find the best one. One such chess engine that has gained popularity is the one which uses minimax search with alpha beta pruning and only the top 50 percent of the possible moves are taken into consideration.

The Minimax algorithm is a commonly used method in computer chess engines, which works by considering all possible moves that can be made from the current position and evaluating the best possible move based on a scoring function. Alpha-beta pruning is a technique used to reduce the number of positions that need to be evaluated by discarding positions that are guaranteed to be worse than a previously evaluated position. This algorithm is particularly useful in chess, where the number of possible moves at any given point can be very large.

The approach of considering only the top 50 percent of the possible moves is based on a trained dataset, where thousands of games have moves played by the winner labeled as good and the possible legal moves that he could have played but didn't are labeled as bad. The top 50 percent of the moves in order of their good percentage are taken. This dataset is trained using machine learning techniques to identify the moves that are most likely to lead to a win.
By using this approach, the chess engine can focus on a smaller subset of the possible moves, which reduces the computation time required to evaluate the best move. This approach has been shown to be effective in improving the performance of the chess engine and has been used by many top computer chess engines.

In conclusion, the chess engine that uses Minimax search with alpha-beta pruning and only the top 50 percent of the possible moves has proven to be a successful approach in the development of chess engines. By using a trained dataset to identify the best moves, this engine can quickly evaluate the best move and improve its performance. With the continued advancement in artificial intelligence and machine learning, it is likely that we will see even more advanced chess engines in the future.

[II] LITERATURE SURVEY

After referring to multiple models of chess engine which uses a variety of techniques like Reinforcement learning, Monte Carlo Search, Machine learning, Minimax algorithm and the likes, we concluded that utilizing just one technique in a chess engine is not worth the time or the processing power required and the engine can be made way more efficient with the technique of combining two different methods to save money and time. One of the most famous and efficient engines in the world currently is Stockfish which uses the minimax tree alone with the help of multiple evaluation functions but with the help of a huge processor it searches ahead till the endgame is reached. This obviously is not going to be made possible while using the regular engines and hence modifications must be made. So, in building our model, we have combined two methods, that is the minimax algorithm and the machine learning part.

```
[Event "Interpolis International Tournament"]
[Site "Tilburg NED"]
[Date "1994.09.10"]
[Round "1.1"]
[White "Seirawan, Yasser"]
[Black "Smyslov, Vassily"]
[Result "0-1"]

1. d4 Nf6 2. c4 e6 3. Nf3 Bb4+ 4. Nbd2 c5 5. a3 Bxd2+ 6. Bxd2 d6 7. dxc5 dxc5
8. Qc2 Nbd7 9. O-O-O Qc7 10. g3 Ng4 11. Bf4 e5 12. Bh3 h5 13. Bxg4 hxg4 14.
Nxe5 Nxe5 15. Rd5 Rh5 16. Rxe5+ Rxe5 17. Qh7 f6 18. Rd1 Kf7 19. Qh8 Qc6 20. Rd8
Re8 21. Qh5+ Ke7 22. Rd6 Qe4 23. Rd3 Be6 24. Bd6+ Kd8 25. Be5+ Bd7 26. Qf7 Qf5
27. Bf4 g5 28. Be3 Qe6 29. Qg7 b6 30. Rd5 Qe7 31. Rxg5 Qxg7 32. Rxg7 Kc7 33. b3
Rxe3 34. fxe3 Rh8 35. e4 Rxh2 36. Kd2 Rg2 37. Rf7 Rxg3 38. e3 Rf3 39. Ke2
{On time} 0-1
```

Fig.1 DATASET: *PORTABLE GAME NOTATION*

[III] ALGORITHMS USED

Minimax Algorithm:

The minimax algorithm is a commonly used technique in game theory and artificial intelligence for determining the best move in a two-player game, such as chess.

In a game like chess, where each player takes turns making moves, the minimax algorithm involves evaluating all possible moves and predicting the outcome of each move for both players.

The algorithm works by assuming that the opponent will always try to make the best move possible,
and therefore, it minimizes the maximum possible loss (hence the name "minimax").

The algorithm recursively generates a game tree, with each node representing a possible move and each leaf node representing a possible game outcome. It then evaluates the leaf nodes by assigning a score based on how favourable the outcome is for the player who made the move.

This score is then propagated back up the tree to the root node, where it is used to determine the best move for the player.

In chess engines, the minimax algorithm is typically combined with alpha-beta pruning, which is a technique that reduces the number of nodes that need to be evaluated by pruning.
branches of the tree that are unlikely to lead to a good outcome. This greatly reduces the search space,
allowing the algorithm to search deeper into the game tree and make more informed decisions about the best move.

In addition to the basic minimax algorithm, there are several extensions and variations that have been developed over the years to improve the performance of chess engines. For example, the use of transposition tables can greatly reduce the amount of redundant evaluation by storing previously evaluated positions and their associated scores. Other techniques, such as iterative deepening, move ordering, and quiescence search, have also been developed to improve the efficiency and accuracy of the algorithm.

Overall, the minimax algorithm is an essential tool for any serious chess engine, as it allows the computer to evaluate and analyse the vast number of possible moves and positions in a game of chess, and ultimately make the best possible decision for each move.

Alpha Beta Pruning:

Alpha-beta pruning is a technique used in game tree search algorithms to reduce the number of nodes that need to be evaluated. It is particularly effective in two-player games like chess, where the search space can be very large, and it is important to search as deeply as possible to find the best move.

In a game tree search, the algorithm evaluates each possible move and its resulting game state, creating a tree of possible moves and outcomes. Alpha-beta pruning is a method of cutting off branches of the tree that are unlikely to lead to a good outcome, without evaluating all the leaf nodes in those branches.

The algorithm works by maintaining two values: alpha, which represents the maximum value found so far for the maximizing player, and beta, which represents the minimum value found so far for the minimizing player. As the algorithm searches deeper into the tree, it updates these values and prunes branches that are guaranteed to be worse than the current best move.

For example, consider a position where the maximizing player has two possible moves, A and B. The algorithm evaluates move A and finds that it leads to a position with
a score of 10. It then evaluates move B and finds that it leads to a position with a score of 8. Since the maximizing player will always choose the move with the highest score,
move A is better than move B. Therefore, the algorithm can safely prune the subtree of move B, since it is guaranteed to be worse than move Alpha-beta pruning can greatly reduce the number of nodes that need to be evaluated, especially when combined with other techniques like iterative deepening and move ordering. In a chess engine, alpha-beta pruning is an essential tool for evaluating possible moves and finding the best move in a reasonable amount of time.

Overall, alpha-beta pruning is a powerful and widely used technique in game tree search algorithms, and it is particularly effective in two-player games like chess, where the search space is large, and the goal is to find the best move as quickly and accurately as possible.

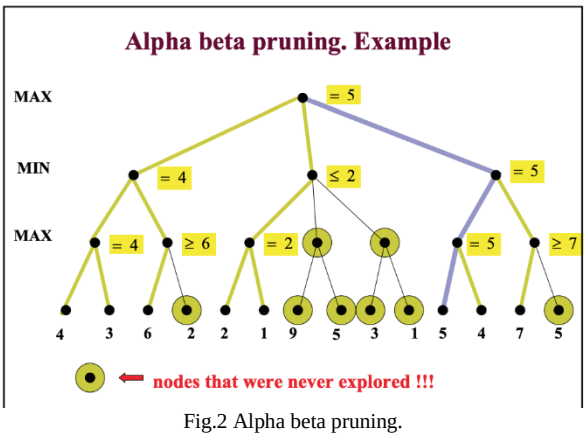

Fig.2 Alpha beta pruning.

[IV] PROPOSED WORK

A chess engine that uses Minimax search with alpha-beta pruning and selective consideration of only the top 50 percent of possible moves has several working components that enable it to play chess at a high level. In this system, the evaluation function is calculated by assigning specific positional scores and the actual value of each piece. The move which gives the highest evaluation after a certain depth will be the move played.

The evaluation function is a crucial component of a chess engine. It assigns a score to each possible position on the chessboard. The score is based on various factors such as piece values, pawn structure, control of the centre, king safety, and more. By assigning a score to each position, the evaluation function provides a means to compare different positions and determine which move is the best.

To calculate the positional scores, the engine considers several aspects of the chessboard, such as the presence of pawns, knights, bishops, rooks, and the queen, and the control of the centre. Each of these aspects is assigned a value, and the overall score is calculated by adding up the values for each aspect.
For example, a bishop on a central square may be worth more than a bishop on the edge of the board. Similarly, a pawn structure that allows for the easy development
of pieces may be worth more than a pawn structure that is more cramped.

The actual value of each piece is also considered. In general, a queen is worth nine points, a rook is worth five points, a bishop and knight are worth three points each, and a pawn is worth one point. However, the actual value of each piece can vary depending on the position of the pieces and the overall situation on the board. For example, a bishop may be more valuable than a knight in an open position with many diagonals, while a knight may be more valuable in a closed position with many pawns.

Once the evaluation function has assigned a score to each possible position, the search algorithm is used to determine the best move. The Minimax algorithm works by recursively evaluating the score of each possible move and selecting the move that leads to the best outcome. In other words, the engine considers all possible moves at each level and evaluates the best move based on the score generated by the evaluation function.

Alpha-beta pruning is then used to discard positions that are guaranteed to be worse than a previously evaluated position, which greatly reduces the number of positions that need to be evaluated. Alpha-beta pruning works by tracking two values, alpha and beta, that represent the best possible score for the maximizing player and the worst possible score for the minimizing player, respectively. As the engine evaluates each move, it updates the alpha and beta values accordingly.
If a move leads to a score worse than the current alpha or beta value, it is discarded, as it is guaranteed to be worse than a previously evaluated position.
Finally, the selection of the top 50 percent of possible moves based on a trained dataset is used to further improve the engine's performance.

This dataset is created by analysing thousands of games played by the winners and labelling their moves as good. The possible legal moves that could have been played but weren't labelled as bad. The engine then takes the top 50 percent of moves in order of their good percentage and considers only those moves during its search. This technique greatly reduces the number of positions that need to be evaluated, further improving the engine's performance.

So, a chess engine that uses Minimax search with alpha-beta pruning and selective consideration of only the top 50 percent of possible moves, with the evaluation function calculated by assigning specific positional scores and the actual value of each piece, is a complex but highly effective system.
The evaluation function provides a means to compare different positions, while the search algorithm and selection of top moves enable the engine.

Fig.3 INTERFACE

[V] CONCLUSIONS

In conclusion, the use of Minimax search with alpha-beta pruning and selective consideration of only the top 50 percent of possible moves, based on a trained dataset, has resulted in a successful chess engine. The approach of labelling moves played by the winner as good and legal moves that could have been played but weren't as bad has enabled the engine to quickly evaluate the best move and improve its performance. This technique has been used by top computer chess engines and has shown to be effective in reducing computation time while still achieving excellent results. As artificial intelligence and machine learning continue to advance, we can expect to see even more advanced chess engines in the future that will further push the limits of what is possible in the game of chess.

[VI] REFERENCES

[1] Nilma Upasani, Ansh Gaikwad, Arshad Patel, Nisha Modani, Prashanth Bijamwar, Sarvesh Patil, “Dev Zero : A Chess Engine”, 2021 International Conference on Communication information and Computing Technology (ICCICT)2021.

[2] Varsha Shrivastava, Siddhant Mishra, Himanshu Panchal, “Chess Moves Prediction using Deep Learning Neural Networks”.

[3] “Deep Learning Neural Networks” (2021), doi:10.1109/ICACC-202152719.2021.9708405.

[4] Paul Grunke, “Chess AI and epistemic opacity”, Informacios Tarsadalom, May 2020 doi:http://dx.doi.org/10.22503/inftars.XIX.2019.4.1

[5] Victor Sim, “Implementing a Deep Learning Chess Engine”

[6]. Steven James, George Konidaris, Benjamin Rosman, “An Analysis of Monte Carlo Search”, Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17) At:SanFrancisco,USA,2017.

[7]. Eli David, Nathan S. Netanyahu, Lior Wolf, “DeepChess: End to End Deep Neural Network for Automatic Learning in Chess”, International Conference on Artificial Neural Networks (ICANN), Springer LNCS, Vol. 9887, pp. 88-96, Barcelona,Spain,2016, doi:https://doi.org/10.48550/arXiv.1711.09667 .