2021 International Conference on Communication information and Computing Technology (ICCICT) | 978-1-6654-0430-3/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICCICT50803.2021.9510148

Dev-Zero: A Chess Engine

Nilam Upasani
Dept. of Computer Science and Engineering
IIT Dhanbad
Jharkhand, India
nilamupasani@gmail.com

Ansh Gaikwad
Dept. of Computer Engineering
Vishwakarma Institute of Technology
Pune, India
ansh.gaikwad19@vit.edu

Arshad Patel
Dept. of Computer Engineering
Vishwakarma Institute of Technology
Pune, India
arshad.patel19@vit.edu

Nisha Modani
Dept. of Computer Engineering
Vishwakarma Institute of Technology
Pune, India
nisha.modani19@vit.edu

Prashanth Bijamwar
Dept. of Computer Engineering
Vishwakarma Institute of Technology
Pune, India
prashanth.bijamwar19@vit.edu

Sarvesh Patil
Dept. of Computer Engineering
Vishwakarma Institute of Technology
Pune, India
sarvesh.patil19@vit.edu

***Abstract*—Dev-Zero is a fully functional chess engine that is used to play through various standards such as with human, self-play and also with some Universal Chess Interface (UCI) compatible chess engines. The distinguishable characteristic is that it is designed to be universally accessible, and can be concurrently played by people with different abilities and intelligence.**

Minimax algorithm along with alpha-beta pruning is used simultaneously with chess books to select the next move. Based on parameters like King safety, mobility of each piece, structure and centre control, the board is evaluated using a heuristic function that checks for checkmate as well as stalemate conditions using values from initialised piece square tables and calculated material score.

Developers can create fascinating Graphical User Interface (GUI) and use the proposed algorithm in developing their chess games.

***Index Terms*—Alpha-Beta, Jupyter, Minimax, Portable Game Notation (PGN), Python, python-chess, Scalable Vector Graphics (SVG), Stockfish, Universal Chess Interface (UCI)**

I. Introduction

Nowadays, computer games have become one of the major sources of entertainment. For example, one out of two children play video games, and in US, the sales of games outnumber sales of books. Furthermore, the world market for video games continues to grow at a much faster rate than the cinema box office, DVD rentals and even music retail. Unfortunately, most of the existing computer games are quite stressful and uneducated. This fact forces guardians to make computer games inaccessible to a large percentage of children. In the light of the above, Dev-Zero is an engine for the development of a universally accessible multiplayer intelligent board game that is Chess.

Chess has been practised by a large number of smart minds from all over the world and has been proved as a game with enhancing problem-solving abilities. The players which play Chess at a good level have strong problem-solving and critical thinking skills which can make a person explore and use his brain more than that of others. In short, playing and analysing chess games can be a good exercise for the brain.

A model proposed here, also known as Chess Engines has been taught to play Chess. Nowadays, there are a lot of chess engines such as stockfish 13, alpha-zero and many more that are trained with more complex algorithms and can be hard for beginners to understand. Therefore, an effective and easy to understand model is proposed. This Chess engine is developed using the minimax algorithm along with alpha-beta[1]pruning with negamax.

The first thing is to start with the notations on the chessboard. Every square on a chessboard has its name and every move that a player makes can be noted down such as Qd4, e5, etc moves. The machine learning algorithm needs the moves in the form of Portable Game Notation (PGN)[2]as an input, but giving the moves as inputs from the chess books isn't a recommended approach, therefore this model is implemented using the minimax algorithm, which is well known for one vs one AI games such as Tic tac toe, Checkers and many more. And for optimization, alpha-beta pruning is used to turn down the time and space for evaluation functions.

After studying the working algorithm of Chess Engines, the implementation is done from scratch and is tested. Some best practices are taken under consideration to create this chess engine. In chess, to get a good position on the board, a player needs to dominate the centre. Therefore, to create a good model, opening moves are selected from the chess books. After analysing some chess games some rules are verified which leads to victory, some of them dominate the centre which helps the dominator with double attacks or forks and protects the king simultaneously. To phrase it properly the involvement of the king is also crucial in the end game, so as long as the king is safe from checks the king can be involved in the game.

Considering all these criteria, evaluation functions are designed. Evaluation functions are just functions which are calculated by Piece square value tables. For example, let's say that the Queen will be more effective in the centre as it covers most of the places on the board and also follows the rule ‘Dominate the Centre’. Some references are referred for the perfect evaluation function values and then implementation of the minimax algorithm is done. After evaluation, the negamax variant of the well-known minimax algorithm and alpha-beta pruning is implemented, as only one function is needed to maximise the score of both players. The only difference is that one player’s loss is equal to another player’s gain.

The last thing after the model was ready is the representation of the board with a GUI which is essential to visualize the moves for the players who cannot understand the Portable Gaming Notation (PGN). Therefore a flask server is hosted on the localhost using the Scalable Vector Graphics (SVG) format of the chess-board provided by python-chess.

II. LITERATURE REVIEW

‘Programming a Computer for Playing Chess’ paper showed chess developments for computers in 1950, and since then various chess applications and games have been developed till now. Some are made for children while some are developed for the physically challenged. Some are also developed for people diagnosed with intellectual disorders[3]. Many chess games are also developed using 3D graphics.

The proposed paper discusses a chess game engine, which is a conglomerate of most of the features from past papers but also enables players to play from different regions without being physically present together. It enables the player to practice without a strong opponent. The proposed chess engine is an approach towards intellectual gaming with no stressful and uneducated content.

Paper titled ‘A Genetic Algorithms for Evolving Computer Chess Program’ uses the move decided by computer, totally based on databases which contain moves of grandmaster level games. This engine takes into consideration all the moves played at that particular position, performs a 1-ply search and stores the selected move. Then it compares the stored move and the actual move played by the player. The fitness of the algorithm is checked by identifying the number of times both moves are the same. It shows that genetic algorithms are used to evolve the parameter values of chess programs [4].

Another paper titled ‘Phoenix: A Self-Optimizing Chess Engine’ presents another method to make chess engines better than humans. This presents an idea that the learning process can be reduced by defining position evaluation in terms of position value tables. And then multi-niche crowding can be used to optimize these position value tables. But this algorithm can not decide whether mutation will push the chess piece in the correct position or not. Using such genetic algorithms and position value tables, some models are implemented [5]. Here the algorithm is used to tune the weights of evaluation functions and to solve particular problems in chess, like finding check-mate in some moves. This model is designed and implemented in such a way, that it produces only the game tree branches using genetic algorithms. And the move generation is also done based on genetic algorithms[6]. Dev-Zero is implemented using minimax algorithm with alpha-beta pruning and negamax implementation, hence not only branches but also the complete tree is pruned, by which there is an efficient use of space and faster move generation as well. Thus, Dev-Zero evaluates better than these models.

There are some other pre-developed algorithms to design game trees and chess engines based on them. Some engines use these pre-developed algorithms which utilize incomplete information game trees, in which the next move of the opponent is sometimes not predicted accurately, which results in the generation of unorganized game trees. To avoid these types of predictions, an algorithm known as refresh probability table is needed which results in the expansion of the game tree[7]. ‘Particle Swarm Optimization Applied to the Chess Game’ paper describes machine learning optimization for weight initialisation using Particle Swarm Optimisation (PSO). The paper elaborates an idea to use piece values more efficiently for king’s safety and centre control. Moves that are usually selected based on chess books of opening moves and endgame positions or piece values are calculated based on position value tables or piece square tables that are dependent on weights (values in piece square table). The use of PSO has an advantage in some cases over static tuned piece square table values. However, it is highly sensitive to initial weights and is to be taken from static tuning. To overcome it, Dev-Zero uses moves from chess books for move generation and evaluation for initial moves [8].

Some models are implemented using dynamic programming. Where the game tree is evaluated, in which the algorithms like Depth-first search are used. In this type of evaluation, a depth-first search is applied by using memorization and transposition tables along with the implementation of hash tables. But in such types of implementations, the time taken for the game tree traversal and space for evaluation function is more, as there is no pruning of the tree. As Dev-Zero uses a backtracking algorithm that is minimax and alpha-beta pruning[9] with negamax variant, there is an efficient traversal of the game tree with a minimum amount of traversals by pruning the tree, which results in effective and faster move generation. This makes the Dev-Zero engine more efficient, faster and uses less space for evaluation functions.

III. METHODOLOGY

Dev-Zero model is implemented using backtracking algorithms which helps in faster move generation and uses less space for evaluation functions. Hence it is optimum in both time and space utilisation. The algorithmic implementation of the model is also represented on the board.

A. Algorithm

To make the chess engine, a minimax algorithm is used along with alpha-beta pruning for the game tree. The minimax algorithm is used in many game engines, such as chess, where there’s a 50% probability between two players of either winning or losing. Consider two players 1 and 2. A move is considered as the best move when the probability of the player winning increases tremendously. Suppose player 1 makes a move, threatening a piece on the board. After that move, player 2 has to make a move. Here, the best move considered is avoiding the threat by player 1 and making a counter play simultaneously. If in any case, both players played the best moves simultaneously, it leads to a draw.

The idea to build here is a Computer AI, known as a chess engine, which, calculates every move possible on a fixed specified depth and returns the best move possible. In this paper, the minimax algorithm is implemented, which works on the ideology of maximizing player 1's chances of winning and minimizing player 2's chances of winning. This is done using a game tree by working backwards. The concept here is the same as a Chess Grand masters brain works, i.e, thinking x moves ahead by playing a move and then analyzing the board whether it's winning for player 1 or player 2.

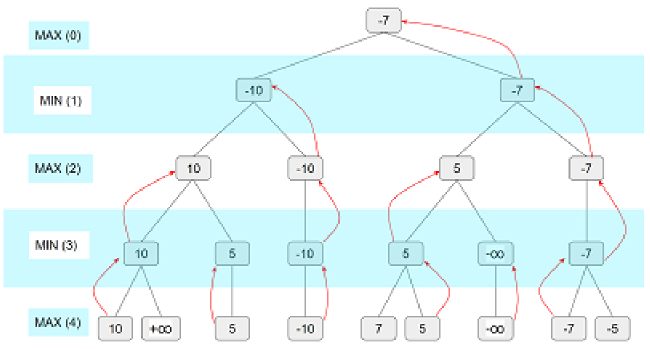


Fig. 3.1. Minimax Algorithm

Minimax Algorithm is often applied with a search algorithm and alpha beta pruning. It aims to reduce the number of nodes obtained by the minimax algorithm in the game tree. When a worse move is detected by the previously played move, it stops the evaluation and starts evaluating the next possibility. If the useless branches are pruned away to save some computation speed and power, it may not result in a loss of best move. The final result remains the same. This makes the algorithm fast and reliable.

The algorithm is designed using two entities, one is alpha (α) and another one is beta (β). Here, alpha (α) will represent least possible score winning player can have and beta (β) will represent best possible score losing player can have. Initially, alpha (α) and beta (β) both are set to their worst possible scores, that is alpha = -∞ and beta = +∞ . Whenever the best score of beta (β) is less than the least score of alpha (α), the maximizing player stops the search.

For illustration, consider an example between two chess players, player 1 and 2. Player 1 finds a good move to improve its positional plan, the end result, being to attack the king. The player 1 then starts finding a better move, if playable. After finalizing a move, player 1 starts thinking black's possible moves, i.e, what will be the best move for black and a forced checkmate is visible. And therefore, there is no sense in

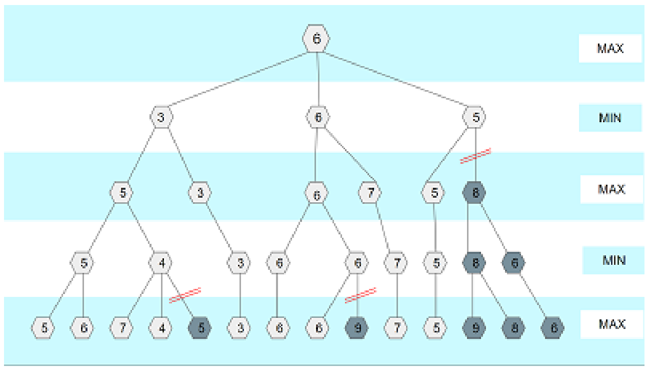


Fig. 3.2. Alpha Beta Pruning

looking for other possibilities for that move and therefore, that move is declined. In technical terms, further branches related to that move is simply pruned to save computational time and space.

Alpha–beta pruning is beneficial because branches of the game tree can be pruned. And hence the use of alpha-beta pruning reduces the search time to get an optimal pruned subtree, which helps to perform deeper search alongside. This optimization reduces almost half of the complete unpruned tree.

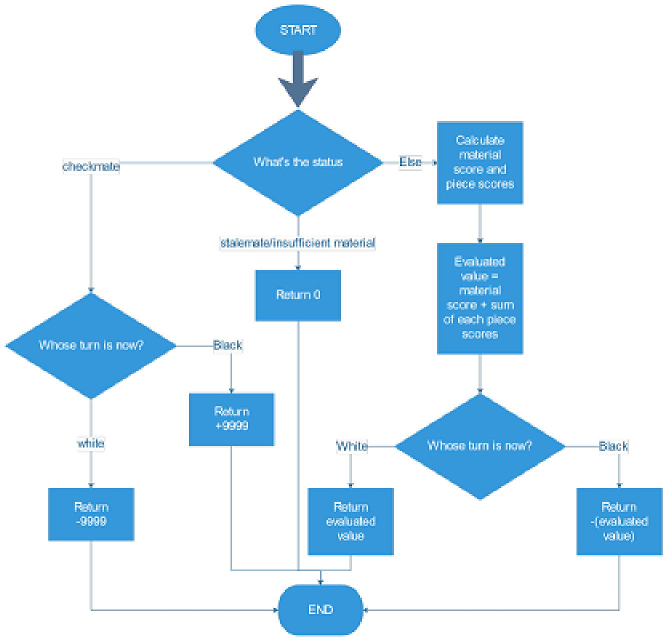

START
What's the status
checkmate
Else
Calculate material score and piece scores
stalemate/insufficient material
Return 0
Evaluated value = material score + sum of each piece scores
Whose turn is now?
Black
Return +9999
white
Return -9999
White
Whose turn is now?
Black
Return evaluated value
Return -(evaluated value)
END

Fig. 3.3. Evaluation Function Flowchart

B. Modules Implemented

The library used in building this engine is python-chess. Portable gaming notation (.pgn) format is used to represent the current representation of the game after it is over for future purposes. And one more important thing is to keep track of every instance of the game. It is achieved by keeping track

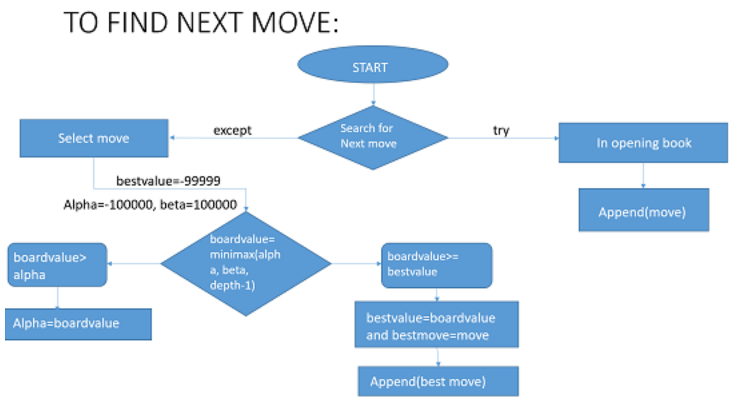


Fig. 3.4. Search Flowchart

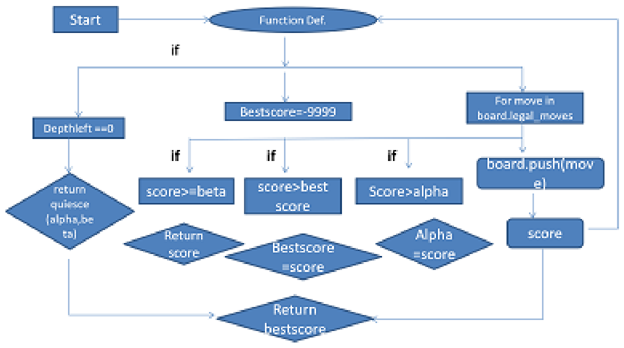


Fig. 3.5. Alpha-Beta Pruning Flowchart

of time and after each move, the current state of the game is stored in .pgn file format with respect to time.

For Jupyter notebook integration, the tiny SVG[10]images are rendered to display the board and pieces. An adapter (polyglot) is used to read top chess opening moves from the book provided. And also for communication with other UCI chess engines and XBoard.[11]It implements an abstraction for playing moves and analyzing a position. It is implemented to let the engine play with the famous stockfish and leelazero chess engines.

The three major objectives in the implementation of this engine are the board evaluation, board representation and move selection.

A position on the chessboard is evaluated if it's won by either white or black or it is a draw. If none of these conditions are satisfied, it is required to provide an estimation about what would be the probability that a player wins. Pieces on the board and positions of pieces are used for evaluation.

So the actual implementation of the model is, for each piece, different moves are calculated depending on its location. This is done with the help of piece square tables.

An evaluation function has been designed, which returns +∞, if there is a forced mate for white and -∞, if there is a forced mate for black. If the position on the board is exactly equal, neither white nor black has an advantage, then the evaluation function returns zero. The value returned by the evaluation function is the sum of the player's material pieces present on the board and the position evaluated with the piece-square tables defined. The evaluation returns negative for black's advantage and positive for white's advantage [12].

For every piece of chess, different matrices are defined known as the Piece Square Tables. The matrix contains positive and negative values where placing the piece on the positive value is preferable and placing it in a negative place is avoidable. The final score of the position is evaluated by adding values of all the pieces including those from the opponent side. These values are then calculated using minimax algorithm and alpha-beta pruning with its negamax variant for better results and to optimize the space and time for the algorithm.

The search function is implemented similar to any chess game plan. To get a better opening in a chess game, one should know different opening theories such as the Ruy Lopez, Slav defense, Evan's Gambit, etc. Similarly, choose the first move from the chess opening book's database, planning to get a good positional opening. This ideology is implemented such that the model must try to implement the best move to counter the human player's move.Then the best possible move to play in the opening will be to go with the theory followed by chess Grandmasters from the chess book database. In case the opening moves has no follow up inside the book, the game tree model is created based on the minimax algorithm and alpha-beta pruning with a variable depth. After evaluation of each move, minimax algorithm and alpha-beta pruning is applied and the best move is chosen. So, the evaluation engine was set with a depth of three and tried defeating the stockfish engine module but it could not. It's expected that after increasing the depth search, it will defeat it.

IV. Results and Discussions

Fig 4.1 shows the advantage metric which was used to analyse the position of the white pieces and black pieces respectively.

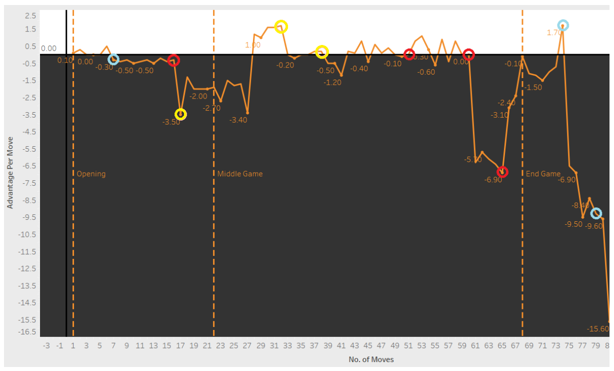


Fig. 4.1. Advantage versus no. of moves metric

The first game which was analysed in detail was between a Human rated 1500 Elo and Dev-Zero, where Dev-Zero is playing black and the Human is playing white. Despite white's start, Dev-Zero has the advantage throughout the game with minimum blunders. The inaccuracies, mistakes and blunders made inside the game have been plotted inside the graph. The negative advantage demonstrates that black has a winning position over white. Because of this game being one-sided, some famous bots of Mr. Beast (youtube streamer) and the Grandmaster Daniel Nariditsky were analysed.

To test how well the engine works, blundering the Queen was done to see if the engine takes it or not. Refer Fig 4.2.

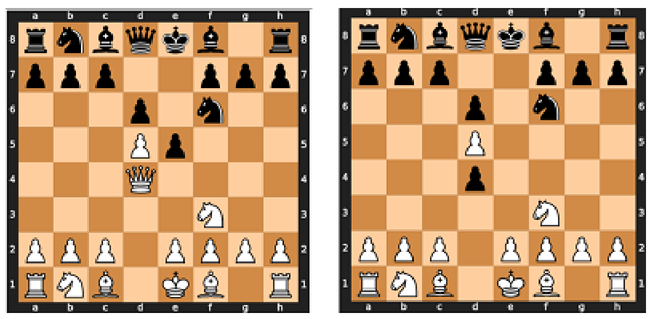
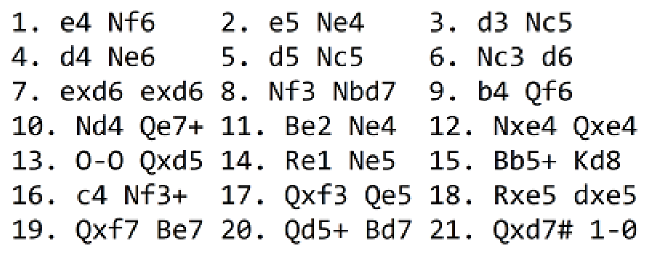


Fig. 4.2. Queen Blundering

In Fig 4.2, the human player blundered a move by playing Qd4, and the proposed engine spotted it and captured the queen as expected.

Then the engine was tested by competing with the famous stockfish 13. The results are displayed in Fig 4.3.



```
1. e4 Nf6     2. e5 Ne4     3. d3 Nc5
4. d4 Ne6     5. d5 Nc5     6. Nc3 d6
7. exd6 exd6  8. Nf3 Nbd7   9. b4 Qf6
10. Nd4 Qe7+  11. Be2 Ne4   12. Nxe4 Qxe4
13. O-O Qxd5  14. Re1 Ne5   15. Bb5+ Kd8
16. c4 Nf3+   17. Qxf3 Qe5  18. Rxe5 dxe5
19. Qxf7 Be7  20. Qd5+ Bd7  21. Qxd7# 1-0
```

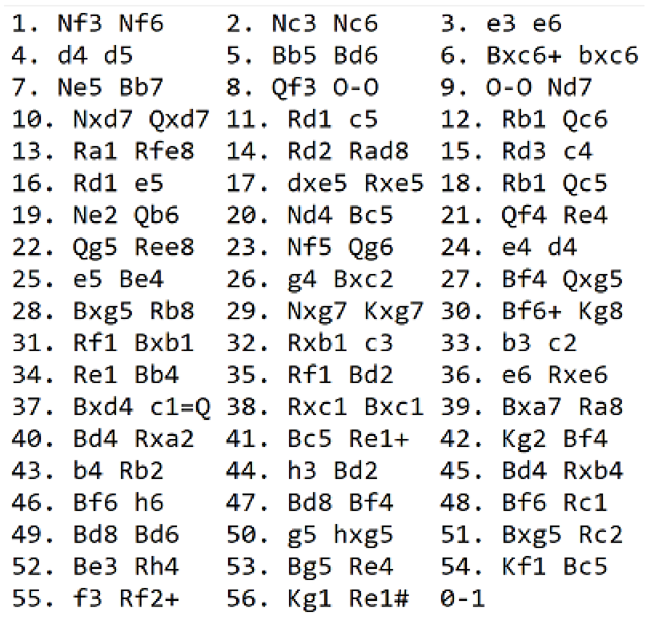
Fig. 4.3. Stockfish Game

The portable game notation for the game is shown in Fig 4.3, which was Dev-Zero Vs Stockfish. The winner was Stockfish in just 21 moves, due to Dev-Zero's depth being just three.

Let's analyse how the algorithm defeats itself. Just like one Grandmaster with the same brain is playing from both sides.

The self-play match was a bit long (check Fig 4.4) which is expected as both sides of the players have the same competence. The competence here is referred to as the AI model. But while analysing it, black wins the game which concludes that the model performs well, when played as black. After doing some critical thinking that why is it so, an assumption can be made that the first move played by white is countered by the best move played by black as the opening moves from the chess books are used.

Dev-Zero tried defeating the famous youtube streamer Mr Beast's Bot and was successful in doing so. Refer TABLE I for analysis. Mr Beast is rated as 1300 Elo and gave a one-way



```
1. Nf3 Nf6     2. Nc3 Nc6     3. e3 e6
4. d4 d5       5. Bb5 Bd6     6. Bxc6+ bxc6
7. Ne5 Bb7     8. Qf3 O-O     9. O-O Nd7
10. Nxd7 Qxd7  11. Rd1 c5     12. Rb1 Qc6
13. Ra1 Rfe8   14. Rd2 Rad8   15. Rd3 c4
16. Rd1 e5     17. dxe5 Rxe5  18. Rb1 Qc5
19. Ne2 Qb6    20. Nd4 Bc5    21. Qf4 Re4
22. Qg5 Ree8   23. Nf5 Qg6    24. e4 d4
25. e5 Be4     26. g4 Bxc2    27. Bf4 Qxg5
28. Bxg5 Rb8   29. Nxg7 Kxg7  30. Bf6+ Kg8
31. Rf1 Bxb1   32. Rxb1 c3    33. b3 c2
34. Re1 Bb4    35. Rf1 Bd2    36. e6 Rxe6
37. Bxd4 c1=Q  38. Rxc1 Bxc1  39. Bxa7 Ra8
40. Bd4 Rxa2   41. Bc5 Re1+   42. Kg2 Bf4
43. b4 Rb2     44. h3 Bd2     45. Bd4 Rxb4
46. Bf6 h6     47. Bd8 Bf4    48. Bf6 Rc1
49. Bd8 Bd6    50. g5 hxg5    51. Bxg5 Rc2
52. Be3 Rh4    53. Bg5 Re4    54. Kf1 Bc5
55. f3 Rf2+    56. Kg1 Re1#   0-1
```

Fig. 4.4. Dev-Zero Self Play Game

TABLE I
DEV-ZERO VERSUS MR BEAST ANALYSIS

Player	Inaccuracy	Mistake	Blunder
Dev-Zero	4	2	1
Mr Beast	4	3	3

fight with Dev-Zero. Analysing the game with stockfish 13, Dev-Zero had the advantage for the entire time and therefore Mr Beast wasn't a tough fight for it.

TABLE II
DANYA VERSUS DEV-ZERO ANALYSIS

Player	Inaccuracy	Mistake	Blunder
Danya	3	0	1
Dev-Zero	5	1	2

Dev-Zero also tried defeating some Chess Grand Masters, just at a depth of 3. Dev-Zero isn't at a high depth level of 52, which is currently used in the Stockfish 13 and is hosted on Cloud by Lichess. Just at the depth of 3, Dev-Zero gave a tough fight to the bot of Daniel Nariditsky. Refer TABLE II for analysis. Some engines, even at a depth of 50 struggle defeating Grand Master bots.

Some complex moves made by Dev-Zero were analyzed and gave some interesting results.

Let's take a good look at the board (Fig. 4.5.a). Here black has an opportunity to exchange Queens and get an advantage in the position as black has a better pawn structure. But the move which everyone skips is Re1 and black loses. So, Dev-Zero finds the best move here, which is to sacrifice the Queen and avoid Re1.

Here is one more puzzle (Fig. 4.5.b), where Dev-Zero found the best move just at the depth of 3, which is Nxd5. The pawn

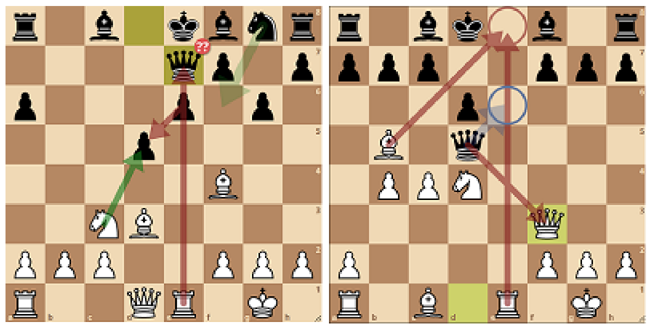


Fig. 4.5.a. and 4.5.b Complex Moves

can't be recaptured as the Queen will be taken by the Rook. This trap was planned beautifully three moves before by Dev-Zero by playing Re1.

To analyse the number of move generation with other engines, a bot of Elo 1500 was chosen to battle against Stockfish 13 and Dev-Zero (at depth=3). Stockfish was able to defeat the bot in 22 moves whereas 37 moves were needed by Dev-Zero to beat it. The last analysis conducted was to experiment the number of moves taken by the Dev-Zero Engine and Stockfish 13 in the number of moves taken to beat one. Dev-Zero took almost 56 moves with minimum blunder count at a depth of 3 and Stockfish excelled with 71 moves with a draw by repetition and a blunder count of zero at the depth of 54.

V. Conclusion

This engine is a simple classic chess engine with varying difficulties and modes by altering the depth of the game tree. It is compatible to play with any human, any UCI compatible chess engine and also it can play with itself. This engine is capable of optimizing the evaluations, which result in faster move generation. In the case of humans with this engine, the engine tries to maximize its score and make faster moves even at smaller depths of the game tree.

The model goes on optimizing the moves as the game goes on. It dynamically builds the game tree based on the move generated by the opponent and evaluates the score using evaluation functions.

In comparative speed analysis of Dev-Zero with other engines, Dev-Zero is trained up to the depth of three, but some strong engines like the stockfish 13 are running on highly cored GPUs with the game tree depth of 50. Despite the game tree depth being 3, Dev-Zero generates complex and advanced endgame moves with a better advantage in position.

Most engines are programmed in high-level languages such as C++ and Java by which they get benefits in terms of speed. Python takes comparatively more time to execute. Some moves such as the endgame fortresses are still hard for the computer chess engines to understand. So, playing with those moves against the Dev-Zero will trick it into losing its game.

Further tasks would be to use the socket and threads to make it universally available, so that people in different physical regions can play it. Also, voice-enabled features can be applied using speech recognition to make the project attractive and user friendly.

Acknowledgment

Thanks to Michael Saks and Avi Wigderson for explaining the optimal version of alpha-beta pruning. As alpha-beta pruning was a great boost for the game tree and it gave a tremendous boost in the execution of the engine. Special Thanks to Mark Lefler, who is a well-known chess programmer for founding the website 'chessprogramming.org' on October 26, 2007, which helped us a lot in deciding evaluation functions, move generation and developing the algorithm.

Also, thanks to Tord Romstad, Marco Costalba, Joona Kiiski and Gary Linscott for making the open-source UCI compatible chess engine, known as the Stockfish 13. At last, we would like to thank our mentor and guide Prof. Nilam Upasani for guiding us throughout.

References

[1] S. H. Fuller, J. G. Gaschnig, and J. J. Gillogly, "Analysis of the alphabeta pruning algorithm," 1973.

[2] D. A. Christie, T. M. Kusuma and P. Musa, "Chess piece movement detection and tracking, a vision system framework for autonomous chess playing robot," 2017 Second International Conference on Informatics and Computing (ICIC), Jayapura, Indonesia, 2017, pp. 1-6, doi: 10.1109/IAC.2017.8280621.

[3] Chess Game Application for people diagnosed with mental and Intellectual disorders, Mikhaylova Iv 1 , Alifirav Ai 1, Russian State Social University Number: 3 Year: 2017

[4] O. E. David, H. J. van den Herik, M. Koppel and N. S. Netanyahu, "Genetic Algorithms for Evolving Computer Chess Programs," in IEEE Transactions on Evolutionary Computation, vol. 18, no. 5, pp. 779-789, Oct. 2014, doi: 10.1109/TEVC.2013.2285111.

[5] A. R. Rahul and G. Srinivasaraghavan, "Phoenix: A Self-Optimizing Chess Engine," 2015 International Conference on Computational Intelligence and Communication Networks (CICN), Jabalpur, 2015, pp. 652-657, doi: 10.1109/CICN.2015.134.

[6] Hootan Dehghani and Seyed Morteza Babamir, "A GA based method for search-space reduction of chess game-tree", OCTl

[7] S. Pan, J. Wu, Y. Sun and Y. Qu, "A Military Chess Game Tree Algorithm Based on Refresh Probability Table," 2020 Chinese Control And Decision Conference (CCDC), Hefei, China, 2020, pp. 4818-4823, doi: 10.1109/CCDC49329.2020.9164878.

[8] J. A. Duro and J. V. de Oliveira, "Particle swarm optimization applied to the chess game," 2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence), Hong Kong, China, 2008, pp. 3702-3709, doi: 10.1109/CEC.2008.4631299.

[9] Game Engine with Continuation-Passing Style, Jeff Walden, Mujde Pamuk, Waseem Daher, May 16, 2007 2017, Springer, van Godewijckstraat 30, 3311 GZ DORDRECHT, NETHERLANDS,pp. 752–768, doi: 10.1007/s10489-017-0918-z.

[10] R. Ferraz, "Accessibility and search engine optimization on scalable vector graphics," 2017 IEEE 4th International Conference on Soft Computing Machine Intelligence (ISCMI), Mauritius, 2017, pp. 94-98, doi: 10.1109/ISCMI.2017.8279605.

[11] T. Shab, "XBoard: a framework for integrating and enhancing collaborative work practices," 2nd IEEE International Conference on Space Mission Challenges for Information Technology (SMC-IT'06), Pasadena, CA, USA, 2006, pp. 6 pp.-290, doi: 10.1109/SMC-IT.2006.88.

[12] A. A. Elnaggar, M. Gadallah, M. A. Aziem and H. El-Deeb, "Enhanced parallel NegaMax tree search algorithm on GPU," 2014 IEEE International Conference on Progress in Informatics and Computing, Shanghai, China, 2014, pp. 546-550, doi: 10.1109/PIC.2014.6972394.