ANNA UNIVERSITY: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report **“CHESS USING ARTIFICAIL INTELLIGENT” is** the Bonafide work of **“Aadharsh R, Bhuvaneshwar S, Dheepak Raaj, Parusu Sunny “**who carried out the project work as a part of Creative and Innovative Project Laboratory.

DATE:	**SIGNATURE**
PLACE:	**Dr. K. THANGARAMYA**
	COURSE IN-CHARGE
	TEACHING FELLOW
	DEPATMENT OF CSE
	ANNA UNIVERSITY
	CHENNAI

ABSTRACT

Computers have reached the grandmaster level, and are beginning to vie for the World Championship. Chess has proved to be too challenging for many of the AI techniques that have been thrown at it. The aim is to promote chess as the fundamental test bed recognized by our founding researchers and increase awareness of its contribution to date. Nowadays, artificial intelligence has grown to great heights and it had contributed a lot in the field of computer chess. But it is computationally expensive and requires a lot of time to train. It is not feasible to implement such algorithms in the day-to-day machines we use. This brings in the need to develop a system that can work in any device without requiring high computation machines. This project implements chess games with computationally less expensive and more responsive in our day-to-day device without requiring high graphics, resolutions, processors. Using two such algorithms and finding out the better algorithm amongst the two and also combining both the algorithms. The features of the game are included game against computer (with different algorithm), game against another player (Multiplayer), set performance evaluation and along with a user-friendly GUI and Implementing the User vs Artificial Intelligence by using CNN Algorithm and Alpha-Beta pruning Algorithm and comparing the metrics (ELO rating) and conclude which algorithm works better. It also displays the score at each step which is used to analyze the corresponding players position in the game. The scope of this project is very user-friendly GUI with fast and responsive game design in small usage of CPU. Algorithms implementable in any device which rules out the requirement of high computation machines. Provides medium level competitive game to users. This project gives us an idea of how computers understand chess and helps to understand how different algorithms work and how to improve its performance. It also educates us about how computer chess game algorithms are evaluated based on ELO Rating.

LIST OF CONTENTS

CHAPTER 1

INTRODUCTION

Chess is a board game played between two players. It is sometimes called Western chess or international chess to distinguish it from related games such as xiangqi and shogi. The current form of the game emerged in Southern Europe during the second half of the 15th century after evolving from chaturanga, a similar but much older game of Indian origin. Today, chess is one of the world's most popular games, played by millions of people worldwide.

Chess is an abstract strategy game and involves no hidden information. It is played on a square chessboard with 64 squares arranged in an eight-by-eight grid. At the start, each player (one controlling the white pieces, the other controlling the black pieces) controls sixteen pieces: one king, one queen, two rooks, two bishops, two knights, and eight pawns. The object of the game is to checkmate the opponent's king, whereby the king is under immediate attack (in "check") and there is no way for it to escape. There are also several ways a game can end in a draw. One of the goals of early computer scientists was to create a chess-playing machine. In 1997, Deep Blue became the first computer to beat the reigning World Champion in a match when it defeated Garry Kasparov. Today's chess engines are significantly stronger than the best human players, and have deeply influenced the development of chess theory. One of the goals of early computer scientists was to create a chess-playing machine. In 1997, Deep Blue became the first computer to beat the reigning World Champion in a match when it defeated Garry Kasparov. Today's chess engines are significantly stronger than the best human players, and have deeply influenced the development of chess theory.

Computer chess includes both hardware (dedicated computers) and software capable of playing chess. Computer chess provides opportunities for players to practice even in the absence of human opponents, and provides opportunities for analysis, entertainment, and training. Computer chess applications, whether implemented in hardware or software, utilize different strategies than humans to choose their moves: they use heuristic methods to build, search and evaluate trees representing sequences of moves from the current position and attempt to execute the best such sequence during play. Such trees are typically quite large, thousands to millions of nodes. The computational speed of modern computers, capable of processing tens of thousands to hundreds of thousands of nodes or more per second, along with extension and reduction heuristics that narrow the tree to mostly relevant nodes, make such an approach effective. Hardware computers is not so useful now-a-days since it cannot be used for other purposes. So, chess software is mostly used and it is more convenient.

Chess machines/programs are available in several different forms: stand-alone chess machines (usually a microprocessor running a software chess program, but sometimes as a specialized hardware machine), software programs running on standard PCs, web sites, and apps for mobile devices. Programs run on everything from super-computers to smartphones. Hardware requirements for programs are minimal; the apps are no larger than a few megabytes on disk, use a few megabytes of memory (but can use much more, if it is available), and any processor 300Mhz or faster is sufficient. Performance will vary modestly with processor speed, but sufficient memory to hold a large transposition table (up to several gigabytes or more) is more important to playing strength than processor speed.

Most available commercial chess programs and machines can play at super-grandmaster strength (Elo 2700 or more), and take advantage of multi-core and hyperthreaded computer CPU architectures. Top programs such as Stockfish have surpassed even world champion caliber players. Most chess programs comprise a chess engine connected to a GUI, such as Winboard or Chessbase. Playing strength, time controls, and other performance-related settings are adjustable from the GUI. Most GUIs also allow the player to set up and to edit positions, to reverse moves, to offer and to accept draws (and resign), to request and to receive move recommendations, and to show the engine's analysis as the game progresses.

The chessboard is represented in the simplest possible manner - as an 8 by 8 matrix, each containing a Piece (with a "blank" piece representing empty board spaces). Furthermore, flag variables keep track of whether queen/king side castling is allowed for each player, and whether an en-passant capture move is allowed at a given point in time. In order to save space and time during the min-max search, its optimal not to have separate board instance at each branch. After all, they differ only by the position of one piece. Hence each move contains information not only about which piece was moved from where to where, but also whether it affected castling, en passant, and whether it captured an enemy piece in the process. Thus, reversing a move on a board is very simple. The algorithm thus only needs one board object, on which it makes and reverses all the moves it considers during its search. Advanced chess playing programs have far more clever board representations, which operate on bits. Separate instances are kept to keep track of individual pieces, and often bit-wise operations can reveal a lot of information about board positions very quickly (particularly with respect to pawns). Years of research have been spent trying to optimize these representations for speed.

The core of the Minmax chess playing algorithm is a local min-max search of the gamespace. The algorithm attempts to MINimize the opponent's score, and MAXimize its own. At each depth (or "ply" as it's as its referred to in computer chess terminology), all possible moves are examined, and the static board evaluation function is used to determine the score at the leafs of the search tree. These scores propagate up the tree and are used to select the optimal move at each depth. The bigger the ply, the better the chosen move will be (as the algorithm is able to look ahead more moves). The branching factor is typically between 25 and 40 moves per ply (the average is around 35).

This AlphaBeta pruning algorithm is common pruning fuction is used to considerably decrease the min-max search space. It essentially keeps track of the worst and best moves for each player so far, and using those cancompletely avoid searching branches which are guaranteed to yield worse results. Using this pruning will return the same exact moves as using min-max (i.e. there is no loss of accuracy). Ideally, it can double the depth of the search tree without increasing search time. To get close to this optimum, the available moves at each branch should be appropriately sorted.The sorting is done by the looking at the scores of each possible move, looking only 1 ply ahead. The intuitive sort would be to arrange them from best to worst, but that's not always best. Most moves in chess end up being ones with small gains and losses (ones that improve position, not necessarily capturing pieces), so it makes sense to order the "stand pat" moves first. So the sorting is based on the absolute value of the move scores (with the smaller ones coming first). The average branching factor for min-max in chess is about 35, but with the alpha-beta pruning and sorting, the program acheives a branching factor of around 25.

The Null Move Heuristic is a simple heuristic improves the beginning of the alpha-beta search. Initially, there are no values for what the worst and best possible moves are, so they default to negative and positive infinity respectively. But using this heuristic the algorithm can find an initial lower bound on the best moves. It lets the opposing player play two moves in sequence (choosing them based on a small-ply min-max search), and computes the score after that. Certainly, any move made by the current player should beat a score obtainable by the opponent getting two chances to move.

The Genetic Algorithm, while certain aspects of evaluating a board are obvious (such as piece values - a queen is clearly worth more than a pawn), other factors are not as easily determined purely by intuition. How much is a bishop's mobility worth? How important is it to check the opponent? Is threatening an enemy's piece better than protecting your own? One can make relatively good educated guesses to such questions, and thus develop a decent static board evaluation function, but I was hoping for a more analytical method. My goal was to attempt to optimize the board evaluation function by utilizing genetic algorithms to determine it. One module of the program is capable of running chess tournaments, where the computer plays against itself with different evaluation functions. It generates random evaluation functions, which then get mutated or preserved based on how well they perform in the tournaments. The core of the tournament algorithm does the following. It has a set of 10 evaluation functions, and pits them all against each other. Each side gets to play both black and white for fairness. Subsequently, it selects the best five, and generates 5 new ones to replace the worst 5. This continues for any desirable number of iterations (the default was set to 10). There are two version of the algorithm that were run. One was a "preservation" one, which kept the best 5 "as is" in between iterations. The other algorithm was a "mutation" one, which kept 1 of the 5, and mutated the other 4. Each mutation was between a pairing of some 2 of the best 5 functions. Determining the winner of a given game is not always trivial. For time constraints, each game in the tournament is limited to 50 moves, which won't necessarily yield an outright check-mate. Also, draws are possible. Furthermore, for low plys (a ply of 2 was used), it is unlikely for the computer to ever reach check-mate when playing deterministically against itself (since there is not end-game database). But the genetic algorithm requires that there be a “winner" for each game played. The way this done is by scoring the board position from the perspective of each of the functions. Most likely they will both has a consensus as to which side has more points (and hence is winning); however, since obviously each side has a different evaluation function, there is a small probability in a close game that each side will think it's winning.

This game works well on any machine and rules out the requirement of high computation machines, it uses a simple and easily available dataset or uses no dataset to train which reduces the training time. But since our game is designed to be used in day- to-day machines the level of play of the computer would be quite less and it would be much easier for grandmasters to defeat. But still, it will play much better than 90% of the chess players. This game is really a good contribution to computer chess and it also helps us to evaluate and compare the algorithms used behind the scenes. For the players it helps to improve the game and their understanding of the position using the score displayed after every move. There is no big threat but if the depth of alpha beta algorithm or the number of layers is increased in the neural network it may slow the game and consume a lot of CPUS. This project gives us an idea of how computers understand chess and helps to understand how different algorithms work and how to improve its performance. It also educates us about how computer chess game algorithms are evaluated based on ELO Rating.

CHAPTER 2

LITERATURE SURVERY

First Computer Chess has a long and interesting history. The very first machines, such as the Turk, that could play chess were simple hoaxes and trickery. This was the beginning of modern chess computers, which evolved and could beat the absolute best in the world. However, a time span of 60 years of work was needed to reach that stage.

Chess being a well-known game, a game with long-term strategies was a fundamentally attractive prospect for academics. In the middle of the 20th century, there was an early breakthrough in computer chess, with the work of eminent scientists such as Alan Turing and Claude Shannon. As days and years pass by and computers evolved from simple different engines into Turing Machines, Chess continues to be an attractive problem for Computer Scientists to tackle.

Deep Blue was the first chess-playing computer developed by IBM to win a chess game and to smartly overplay a chess world champion within the regular time controls. The contribution made by Murray Campbell and others in the chess-playing computer was tested twice because it was defeated by the chess grandmaster Garry Kasparov in 1996 but later in the year 1997 the chess-playing computer went on to defeat the grandmaster. The first time ever to defeat a grandmaster in tournament play. The two advancements in chess-playing computers, Deep Thought 1 and Deep Thought 2 were the stepping stones to Deep Blue.

Deep Thought 2 was played in many events from 1991-1995. Deep Though 1, which was defeated by the grandmaster made it clear that it had certain shortcomings which had to be fulfilled. A new chip was designed which had 8000 features instead of 6400 features. A combination of search in C language, as well as hardware search on the chip, generated the best move possible. The capability of analyzing 200 million moves per second was achieved by Deep Blue. In comparison with Kasparov, could analyze approximately three. Using machine learning there have been many advancements in various AI challenges, one of them is using in various AI challenges Convolutional Neural Networks have shown successes that can be formulated to classification problems. A model developed to predict professional moves in a strategy board game called Go reported an accuracy of 44.4% which was built by Clark and Storkey.

CNN when trained with appropriate architecture and validation data, can learn to function based on the reasoning in complex logical tasks. Chess is played by evaluating the current game board state and then based on the evaluation play the most feasible piece. The modern chess engines usually have a three steps approach, which can generate and analyze many moves faster than humans. The chess engines will use the approach as –Use the board representation to get the positions, Search through a "tree" that represents the possible game states, evaluate the position using the positional evaluator.

Later in the year 2017, everyone was awestruck when Stockfish, one of the best engines out there, was beaten in a chess game. It was beaten by an unknown computer program but not by a human called Alpha Zero. It was developed by Deep Mind, an AI and research company, which was later conquered by Google. Alpha Zero used reinforcement learning and played against itself for the training of its neural networks. Only the governing laws of the game were given and then it self-played many millions of times (44-million chess-games in the initial 9 hours, stated by DeepMind).

Alpha Zero makes use of its neural networks to make extraordinarily advanced evaluations of board states, which completely cuts out the necessity to analyze at over 70-million game states per second (like Stockfish). DeepMind stated that Alpha Zero achieved the benchmarks required to beat Stockfish in just 4 hours. Alpha Zero needs custom-made systems to run on that many people have referred to as a "Google Supercomputer"—So DeepMind has also shed light and illuminated that Alpha Zero ran on 4 TPUs in its games against Stockfish. Alpha Zero is not obtainable at hand to the world. The game outcomes of Stockfish and Alpha Zero's incredible matches have inspired many open-source NN chess projects to being developed. David Silver and Thomas Hubert ,2018 [2]

The strongest programs are based on a combination of sophisticated search techniques, domain-specific adaptations, and handcrafted evaluation functions that have been refined by human experts over several decades. In contrast, the AlphaGo Zero program recently achieved superhuman performance in the game of Go, by tabula rasa reinforcement learning from games of self-play. In this paper, we generalise this approach into a single AlphaZero algorithm that can achieve, tabula rasa, superhuman performance in many challenging domains. Starting from random play, and given no domain knowledge except the game rules, AlphaZero achieved within 24 hours a superhuman level of play in the games of chess and shogi (Japanese chess) as well as Go convincingly defeated a world-champion program . David Silver and Thomas Hubert ,2017 [3]

The 'Adaptive Genetic Algorithm's Implement on Evaluation Function in Computer Chinese Chess 'Paper helps us avoid local optimum of traditional Hill-Climbing and slow convergence of Simulated-Annealing. AGA increases the power of our program. But GA is computationally expensive. It is time consuming which makes us choose a more efficient algorithm. Wang Jiaol Luo and Yan-hong,2005 [10].

Neurological networks in an AI is the term used for mapped networks generated by an artificial intelligence, they are inspired by the biological neural networks. These systems learn and grow by seeing or processing various examples for doing a task without actually doing specific programming for the same the network will learn and grow according to the set of instructions provided to it by the programmer and modify itself to recognize and create patterns based on the requirements. The aim is to create an ultimate chess game that can learn using image modulation techniques and play against a real human. The initial data set for learning was provided using the ultimate chess guide as well as number of games against human and itself. Vinay Kumar et al,2020 [8].

The 'Genetic Algorithms for Evolving Computer Chess Programs' Paper adapts Improvement upon by means of coevolution. b) The search mechanism is evolved by learning from tactical test suites whereas Computation and training is expensive and Training is time are some of its drawbacks Omid E.David , H.Jaap van den Herik , 2014 [6].

The 'Chess Moves Prediction using Deep Learning Neural Networks' evaluating function to make an evaluation on a given situation and Useful for training quick and moderately successful chess algorithm but main disadvantages where heavy computing machines are needed and taking lot of time. Siddhant Mishra Hitanshu Panchal, 2021 [7]

The 'Candidate Moves Method Implementation in Minimax Search Procedure of the Achilles Chess Engine' which helps us learn about the working of min-max algorithm in the chosen paper. They used Candidate moves algorithm's idea exploits the independences in game tree calculating (as shown in second Section) to accelerate the game tree calculation in distributed environment. Heavy computing machines are required and training is Time consuming. Vladan Vuckovic, 2015 [9]

The 'Deep Chess: End-to-End Deep Neural Network for Automatic Learning in Chess' training relies entirely on datasets of several million chess games, and no further domain specific knowledge is incorporated. It takes a lot of time to train and Deep Chess demonstrates the very opposite by exhibiting an adventurous playing style with frequent positional sacrifices. Eli (Omid) Davidet al, 2016 [4]

The 'Playing Chess with Limited Look Ahead' The strength of our chess engine is assessed by comparing its proposed moves against those proposed by Stockfish. This paper will not affect the daily lives of most people, it can certainly affect those who play chess competitively. Arman Maesumi, 2010 [1]

The ‘Giraffe: Using Deep Reinforcement Learning to Play Chess’ approach works on generality and This approach can easily be ported to other zero-sum turn-based board games, and achieve state-of-art performance quickly but the search speed is a major drawback to this algorithm. Matthew Lai, 2015 [5]

CHAPTER 03

PROPOSED SYSTEM

3.1. SYSTEM ARCHITECTURE

Initially we start the game and it launches the user-friendly GUI and we go into the game. The game has a main menu which asks if the user wants to play a game with another player or with AI. On choosing the option to play with another player, the game begins and each player’s score will be displayed simultaneously. On choosing the Vs AI mode, the player gets to choose which Algorithm he/she wants to play with. And simultaneously while playing, the user score will be displayed alongside. The overall system architecture is depicted in the figure 3.1.

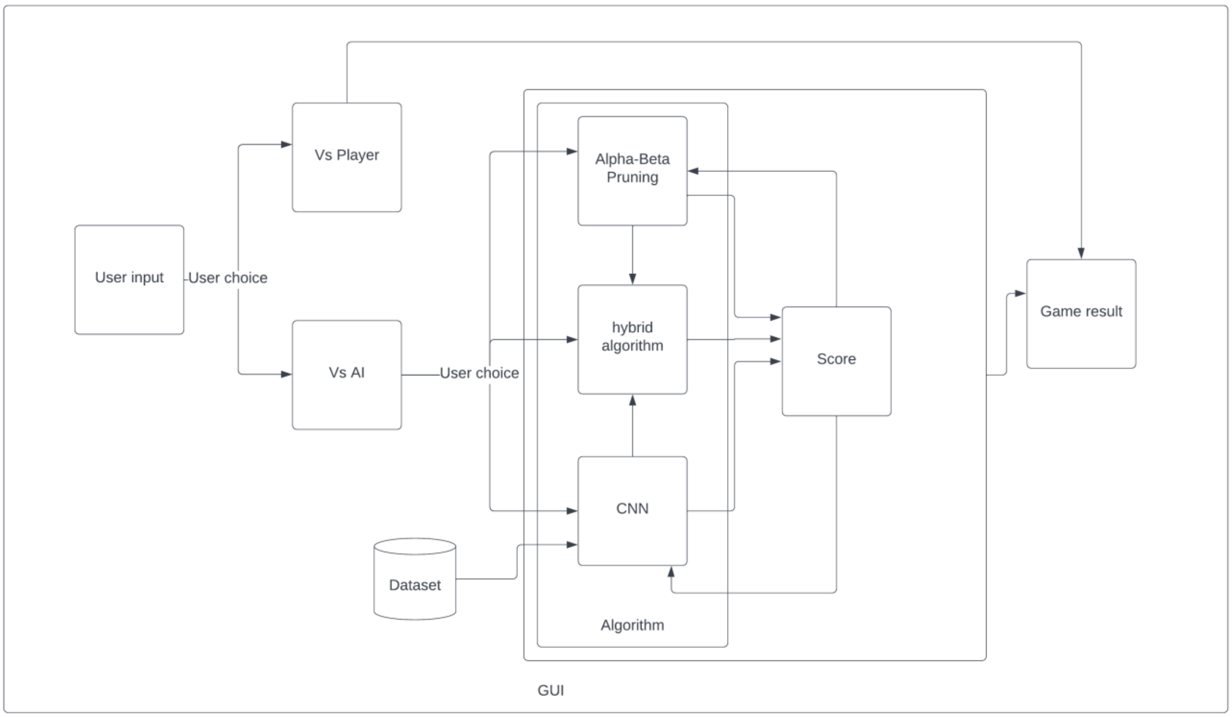


Figure 3.1: OVERAL SYSTEM ARCHITECTURE

3.2 PROPOSED METHODOLOGY

The game can be played against other users by choosing the Vs Human mode. Here each player score on will be calculated and displayed simultaneously while playing the game. The player score will be calculated based on the coin and its position in the game. By choosing the Vs AI mode, the player gets to choose between three algorithms to play against. By choosing the Vs Alpha-Beta pruning algorithm which is an extension of the min-max algorithm. By choosing the Vs CNN mode, the player gets to play against the CNN algorithm. By choosing the Vs Hybrid mode the user gets to play with an algorithm which is a combination of CNN backed-up by Alpha Beta Pruning algorithm. This mode offers quite a challenging game.

3.2.1 HUMAN VS HUMAN

The Functions Used to build this chess game are, the Valid_moves – T o list the valid moves of each coin , the Is_check - To check if king is under check , the Search_moves - To list out the valid moves of each coin when under check , the Pawn_promotion - Pawn promoting into other higher rank coins , the Castling - Castling , the Active_player - Player turn , the Undo_moves - Player can undo a move and go back to the previous state and finally the En_passant - It allows a pawn that has just advanced two squares to be captured by a horizontally adjacent enemy pawn.

3.2.2 VS ALPHA BETA PRUNING

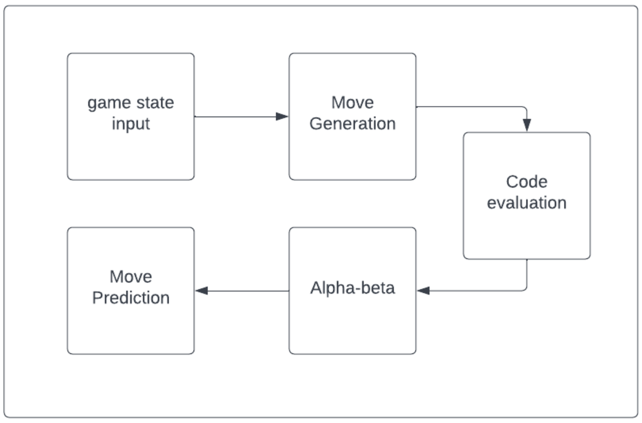


Figure 3.2.: Alpha-Beta Pruning Algorithm

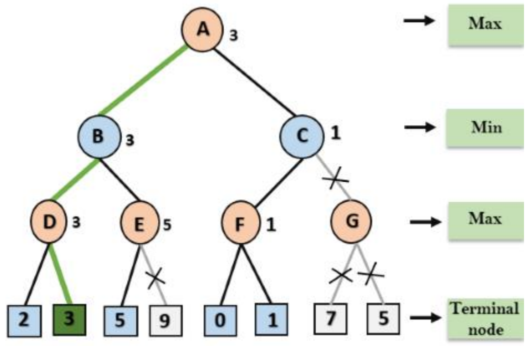


Figure 3.3: Alpha-Beta Pruning Min-max [Javatpoint]

Alpha-beta pruning is a modified version of the minimax algorithm. It is an optimization technique for the minimax algorithm.

As we have seen in the minimax search algorithm, the number of games it must examine are exponential in depth of the tree. Since we cannot eliminate the exponent, we can cut it in half. Hence there is a technique by which without checking each node of the game tree we can compute the correct minimax decision, and this technique is called pruning. This involves two threshold parameters Alpha and beta for future expansion, so it is called alpha-beta pruning. It is also called Alpha-Beta Algorithm.

Alpha-beta pruning can be applied at any depth of a tree, and sometimes it not only prunes the tree leaves but also entire sub-tree.

The two-parameter can be defined as: Alpha: The best (highest-value) choice we have found so far at any point along the path of Maximizer. The initial value of alpha is -∞. And, Beta: The best (lowest-value) choice we have found so far at any point along the path of Minimizer. The initial value of beta is +∞.

The Alpha-beta pruning to a standard minimax algorithm returns the same move as the standard algorithm does, but it removes all the nodes which are not really affecting the final decision but making algorithm slow. Hence by pruning these nodes, it makes the algorithm fast.

3.2.3 VS NEURAL NETWORK

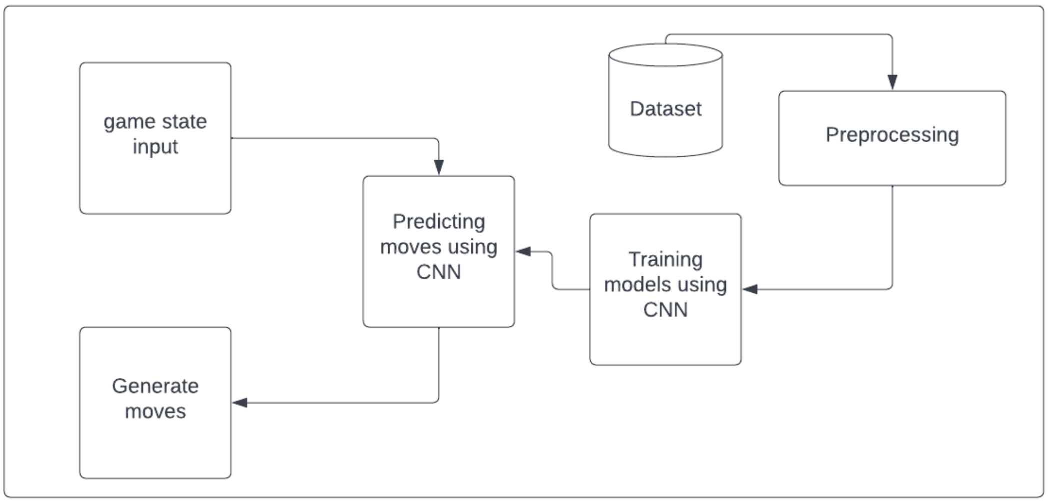


Figure 3.4: Convolution Neural Network

The neural network is made up of elements which are called neurons these elements receive the input information and change their state as per the instructions this changing in the state can result in the creation of activation function this function is a node defined for a generation of the output. The network is formed by connecting the output of a few neurons to the input of the other set of neurons which form a directed or a weighted graph. The entire network is weighed on directed edges. The weight and the activation function are continuously modified by the process of learning or teaching AI.

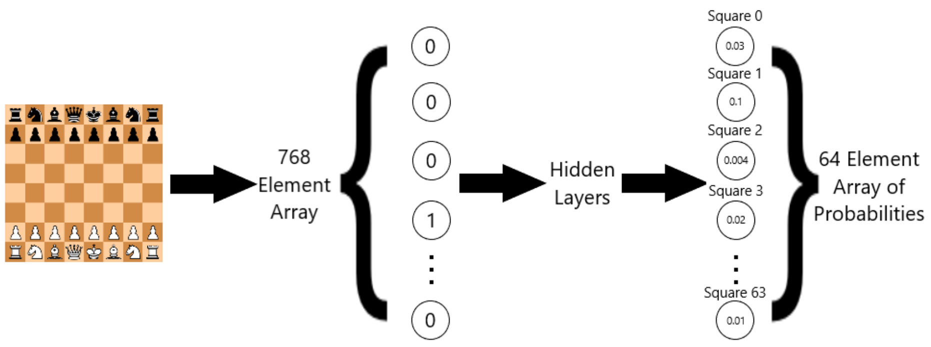


Figure 3.5: CNN

3.2.4.VS HYBRID ALGORITHM

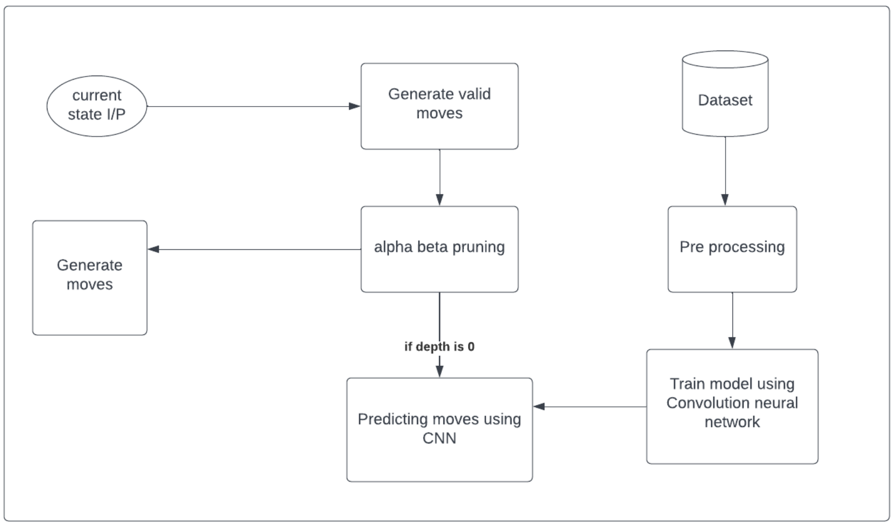


Figure 3.6: Hybrid Algorithm

It is a combination of the two algorithms, the Alpha-Beta Pruning algorithm and Neural network Here it is implementing a neural network with three hidden layers and train with the chess game dataset which predicts moves using the dataset. In cases which are not explored already, the moves are predicted with the help of Alpha beta pruning algorithm of depth three.

CHAPTER 04

RESULT AND DISCUSSION

On entering the game, the user will need to choose the type of game that he/she wants to play(Fig:4.1). It includes Human Vs Human or Human vs Computer.

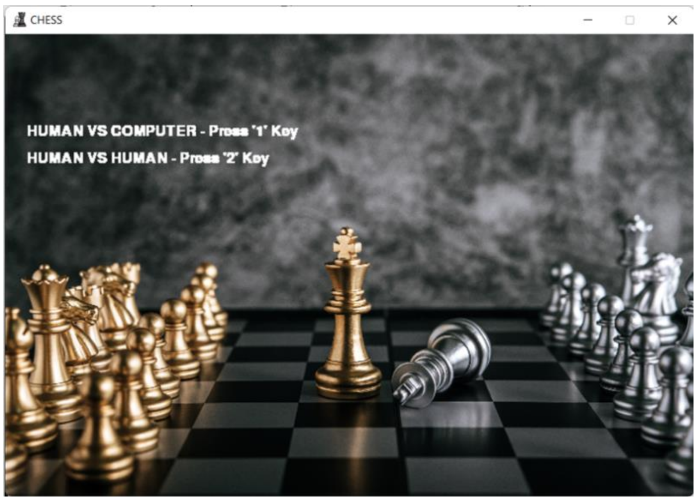

CHESS
HUMAN VS COMPUTER - Press '1' Key
HUMAN VS HUMAN - Press '2' Key

Figure 4.1: Main Menu

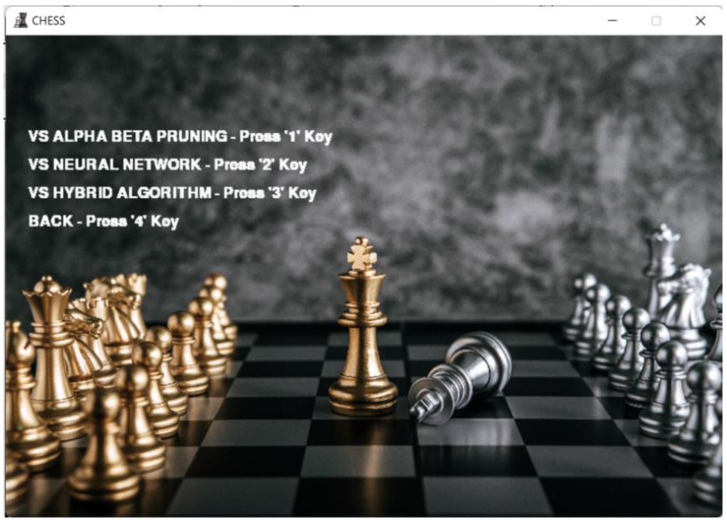

CHESS
VS ALPHA BETA PRUNING - Press '1' Key
VS NEURAL NETWORK - Press '2' Key
VS HYBRID ALGORITHM - Press '3' Key
BACK - Press '4' Key

Figure 4.2: VS AI Human vs computer (if user press key '1')

On choosing the mode Vs AI, the user gets to choose to play against three different algorithms (Fig 4.2).

If user press ‘1’ key – alpha beta pruning.

Initial state

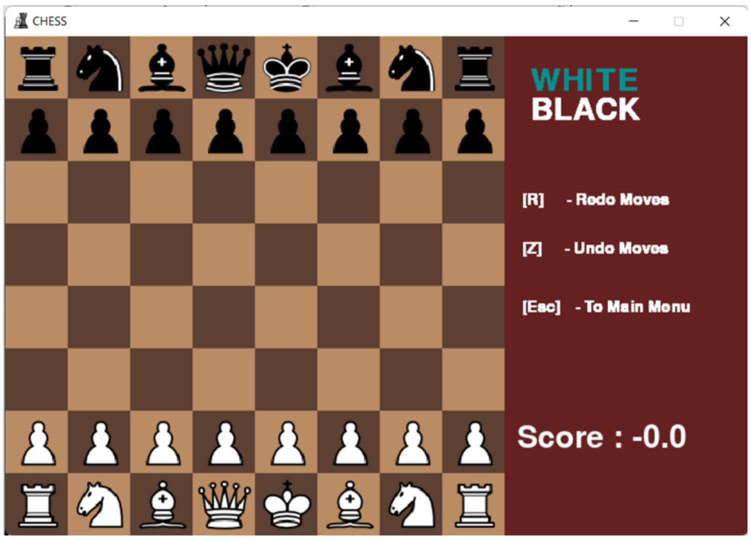


Figure 4.3: Alpha-Beta

User going to move pawn

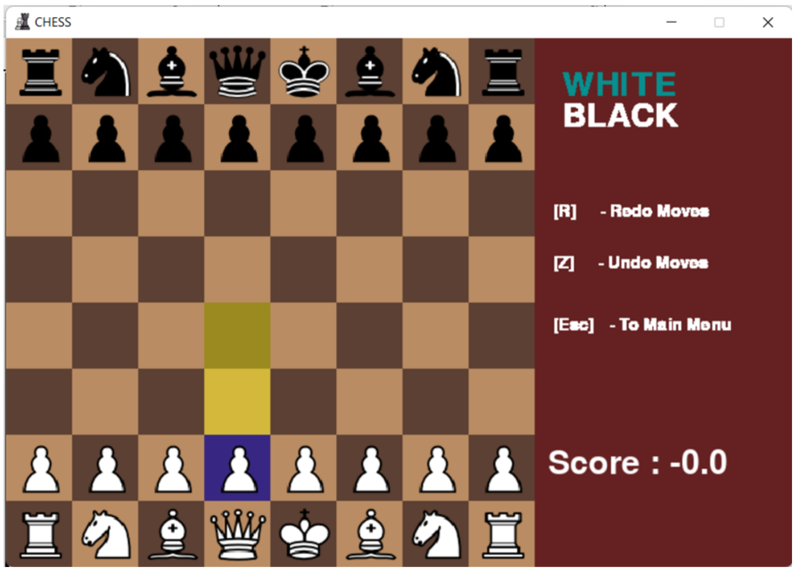


Figure 4.4: Moving a pawn

After moving the pawn, the opponent picks a move using alpha-beta pruning. The following figures (Fig 4.4) depict playing against the Alpha Beta Algorithm driven system .

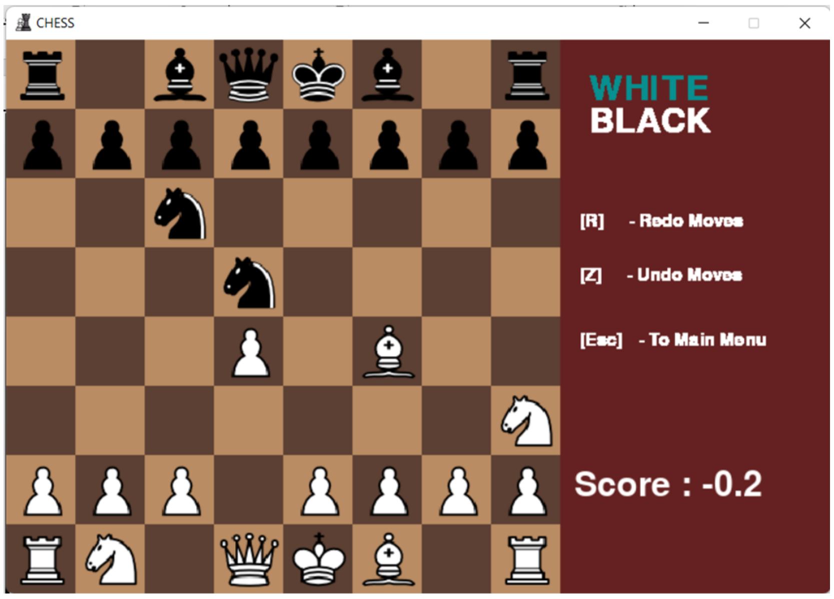


Figure 4.5

Vs Neural network

The game played against Convolution Neural Network is depicted below (Fig 4.6).

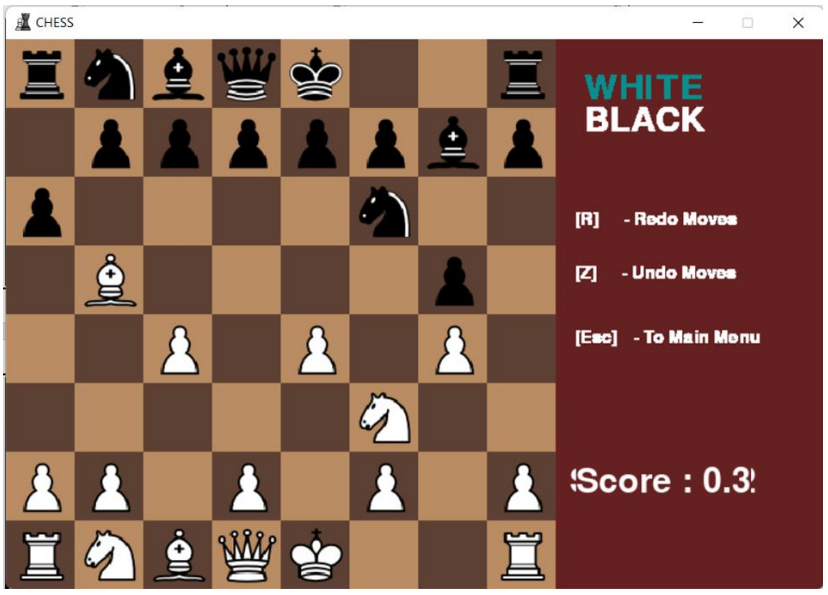


Figure 4.6

Vs Hybrid Algorithm

The game vs Hybrid Algorithm is depicted in the below figure(Fig 4.7)

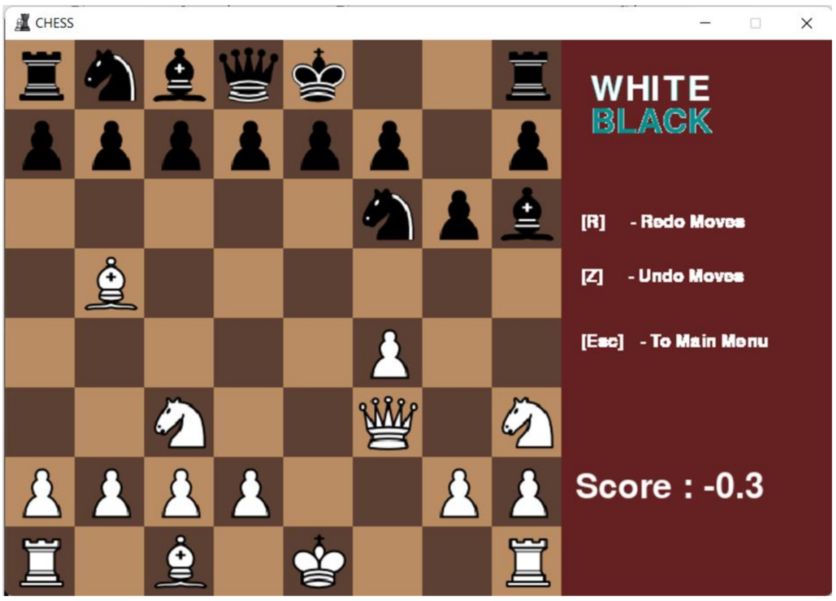


Figure 4.7

Human vs Human

The normal player versus player game by choosing 2 in the main menu is depicted int the following figures (Fig 4.8, Fig 4.9)

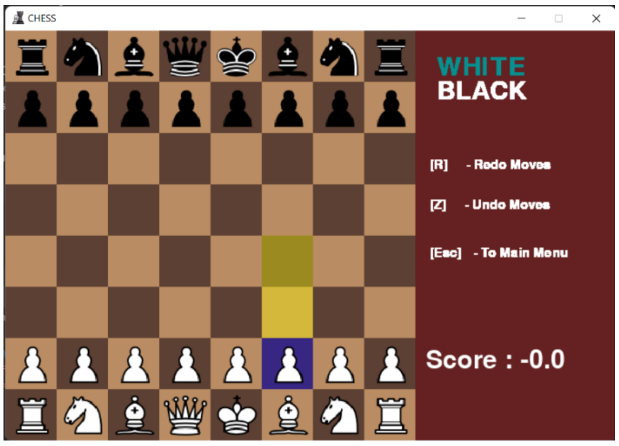


Figure 4.8

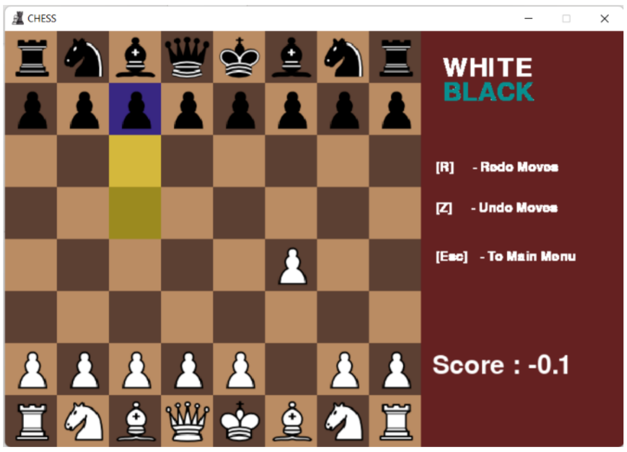


Figure 4.9

Castling

Special functions like castling are depicted in the folowing figures (Fig 4.10, Fig 4.11)

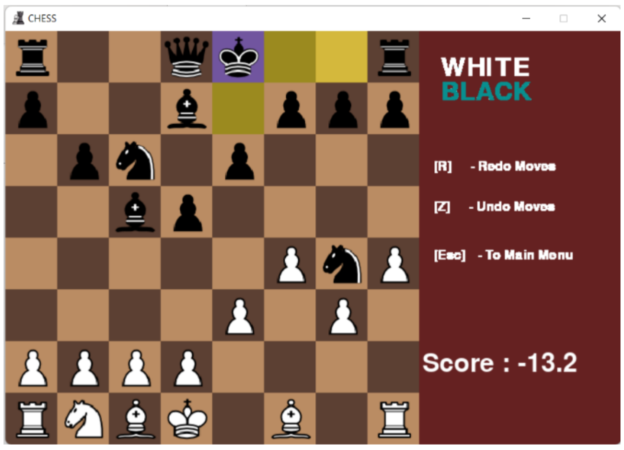


Figure 4.10: BEFORE CASTLING

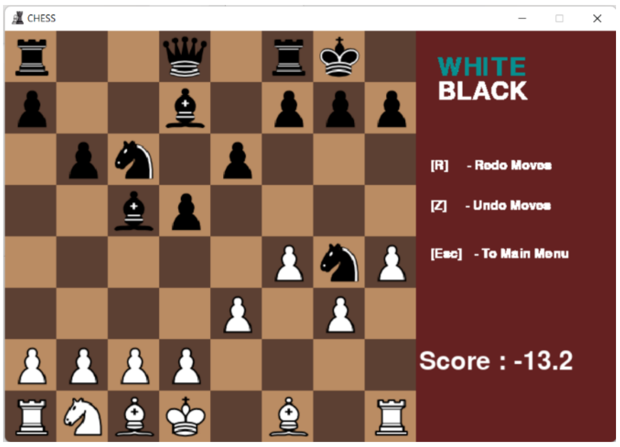


Figure 4.11: AFTER CASTLING

GAME OVER

On winning a game, the respective score is depicted in the following figure (Fig 4.13)

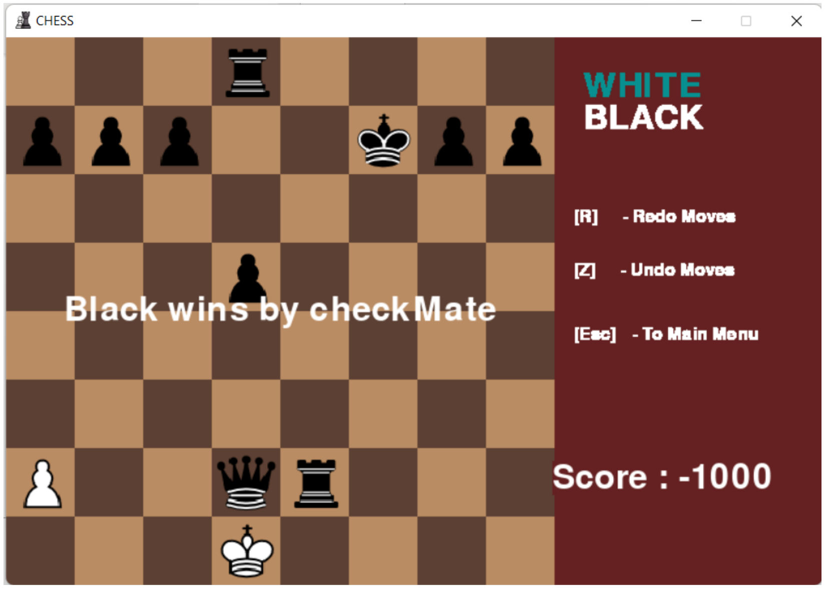


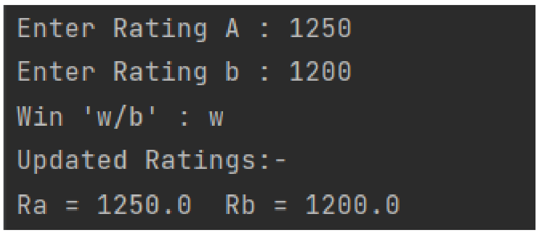
Figure 4.13: GAME OVER

ELO RATING:

Matches are conducted between the algorithms and initially has the ELO rating value of all as 1250.After every match the ELO score is updated. And finally, we concluded the ELO ratings of the respective algorithms.

Table 4.1: GAME OVER

Algorithms	Before		After updating using ELO Score		Result
	Rating 1	Rating 2	Rating 1	Rating 2	
Alpha-beta vs CNN	1250 (Alpha-Beta)	1250 (CNN)	1250	1200	Alpha-Beta
Hybrid vs CNN	1250 (Hybrid)	1200 (CNN)	1250	1180	Hybrid
Alpha-Beta vs Hybrid	1250	1250	1250	1250	Stalemate



```
Enter Rating A : 1250
Enter Rating b : 1200
Win 'w/b' : w
Updated Ratings:-
Ra = 1250.0  Rb = 1200.0
```

Figure 4.14: ELO RATING CALCULATION (Hybrid vs CNN)

CHAPTER 5
CONCLUSION

Alpha-beta pruning acts as a great optimization over the minimax algorithm and how these algorithms are the foundation of state-space searching techniques, paving the way for much more advanced approaches to solving such problems. The neural network, although trained on many different games, plays an opening that barely appears in any of the games. This could be due to the fact that the opening that the algorithm plays could be some sort of average of all the openings it has seen. Hybrid algorithm is combining with alpha-beta and neural network, it play efficiently as long as depth has long values but increasing depth values, time consuming will take long. Alpha-beta vs hybrid will become stalemate, if both depth values is equal. Neural Network vs hybrid, hybrid play better than neural network algorithm.

5.1 FUTURE WORKS

The game algorithm is easy to implement in all devices but the move generation takes some time, which can be worked on and reduced. Increasing the depth which makes the algorithm play better and implement multithreading. AI is a vast group of algorithms and approaches which have all developed over the basic idea of neurological networks when these machines are fed information, they develop a sense of learning. Even though these networks have their ups and downs. The new approaches like deep learning and cognitive computing have raised the bar. We are still far from a self-conscious machine and what it brings for us, but we are close for some mysterious breakthroughs, and who knows what a self-aware machine would bring on the table. The current Systems are good enough for actually helping and improving people’s lives and giving them a sense of terror for what the future holds.

APPENDIX

ALGORITHMS CODE SNIPPETS

1. ALPHA – BETA PRUNING

```
def findAlphaBeta(gs, valid_moves, depth, alpha, beta, turn_multiplayer):
    global bestMove
    if depth == 0:
        return turn_multiplayer * scoreMaterial(gs)
    max_score = -CHECKMATE
    for move in valid_moves:
        gs.makeMove(move)
        bestMoves = gs.getValidMoves()
        score = -findAlphaBeta(gs, bestMoves, depth - 1, -beta, -alpha, -turn_multiplayer)
        if score > max_score:
            max_score = score
            if depth == DEPTH:
                bestMove = move
        gs.undoMove()
        if max_score > alpha:
            alpha = max_score
        if alpha >= beta:
            break
    return max_score
```

2. VS NEURAL NETWORK

```
class NeuralNetwork():
    def __init__(self):
        self.optimizer = 'Adam'
        self.loss = 'categorical_crossentropy'

    def define(self):
        input_layer = Input(shape=(8, 8, 12))
        x = Conv2D(filters=64, kernel_size=2, strides=(2, 2))(input_layer)
        x = Conv2D(filters=128, kernel_size=2, strides=(2, 2))(x)
        x = Conv2D(filters=256, kernel_size=2, strides=(2, 2))(x)
        x = Flatten()(x)

        x = Dense(4096, activation='softmax')(x)
        output = Reshape((1, 64, 64))(x)

        model = Model(inputs=input_layer, outputs=output)
        model.compile(optimizer=self.optimzier, loss=self.loss)
        self.model = model

    def train(self, X, y, epochs, EarlyStop=True):
        if EarlyStop:
            es = EarlyStopping(monitor='loss')

        self.model.fit(X, y, epochs=epochs, callbacks=[es])
        self.model.save('chess_model')

    def predict(self, board, side):
        model = load_model("chess_model")
        translated = translate_board(board)
        move_matrix = model(translated.reshape(1, 8, 8, 12))[0][0]

        move_matrix = filter_legal_moves(board, move_matrix)
        move = np.unravel_index(np.argmax(move_matrix, axis=None), move_matrix.shape)
        move = chess.Move(move [0], move [1])
        return move, 1
```

3. VS HYBRID ALGORITHM

```
def hybrid(gs, valid_moves, depth, alpha, beta, turn_multiplayer):
    global bestMove
    side = 'White' if gs.whiteToMove else 'Black'
    if depth == 0:
        board = chess.Board(boardToFen(gs, gs.board))
        engine = NeuralNetwork()
        flag = False
        move =  engine.predict(board,side)
        for i in range(len(valid_moves)):
            if str(valid_moves[i].getChessNotation()) == str(move):
                gs.makeMove(valid_moves[i])
                flag = True
                break

        t = turn_multiplayer * scoreMaterial(gs)
        if flag:
            gs.undoMove()
        return t

    max_score = -CHECKMATE
    for move in valid_moves:
        gs.makeMove(move)
        bestMoves = gs.getValidMoves()
        score = -findAlphaBeta(gs, bestMoves, depth - 1, -beta, -alpha, -turn_multiplayer)
        if score > max_score:
            max_score = score
            if depth == DEPTH:
                bestMove = move
        gs.undoMove()
        if max_score > alpha:
            alpha = max_score
        if alpha >= beta:
            break
    return max_score
```

4. ELO RATING

```
def Probability(rating1,rating2):
    return 1.0 * 1.0 / (1 + 1.0 * math.pow(10,1.0*(rating1-rating2)))

def EloRating(Ra,Rb,K,d):
    Pb = Probability(Ra,Rb)
    Pa = Probability(Rb,Ra)

    if d == 1:
        Ra = Ra + K * (1-Pa)
        Rb = Rb + K * (0-Pb)
    else:
        Ra = Ra + K * (0 - Pa)
        Rb = Rb + K * (1 - Pb)

    print("Updated Ratings:-")
    print("Ra =", round(Ra, 6), " Rb =", round(Rb, 6))
```

REFERENCES

[1] Arman Maesumi, Playing Chess with Limited Look Ahead, Technical report, Stanford University,2010

[2] David Silver and Thomas Hubert, A general reinforcement learning algorithm that master chess, shogi and Fo through self-play, Mathematical Biophysics, 5:115–137, 2018

[3] David Silver and Thomas Hubert, Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm, Journal of Evolution and Technology, 1:1–14, 2017

[4] Eli (Omid) David et all, DeepChess End to End Deep Neural Network for Automatic Learning in Chess, Foundations of Genetic Algorithms, pages 316–337. Morgan Kaufmann, San Mateo,2016

[5] Matthew Lai, Giraffe:Using Deep Reinforcement Learning to Play Chess, Frontiers of Evolutionary Computation, pages 1–36. Kluwer Academic Publishers, Boston,2015

[6] Omid E. David and H. Jaap van den Herik, Genetic Algorithm for Evolving Computer Chess Programs, Evolutionary Computation, 13(1):1–27,2014

[7] Siddhant Mishra Hitanshu Panchal, Chess Moves Prediction using Deep Neural Network, Connec-tionism in Perspectives, pages 173–198. North-Holland, 2021

[8] Vinay Kumar et al, Application of Neural Network in an AI for Chess Game, Amity University, Morgan Kaufman, San Mateo, Connection Science, 13:1–30,2020

[9] Vladan Vuckovic,Candidate Moves Method Implementation in MiniMax Search Procedure of the Achilles Chess Engine, Mind, 59:433–460, 1950, 2015

[10] Wiang Jiaol Luo Yan- hong, Adaptive Genetic Algorithm's Implement on Evaluation in Computer Chinese Chess, Machine Intelligence 6, pages 3–23. Oxford University Press, Oxford, 2005

[11] https://github.com/victorsimrbt/chess_4096 .