BABEŞ-BOLYAI UNIVERSITY CLUJ-NAPOCA
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
SPECIALIZATION COMPUTER SCIENCE

DIPLOMA THESIS

Using artificial intelligence to assist chess players

Supervisor
Asist. univ. dr. Florentin Bota

Author
Cadar Eduard

2023

BABEŞ-BOLYAI UNIVERSITY CLUJ-NAPOCA
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
SPECIALIZATION INFORMATICĂ

DIPLOMA THESIS

Utilizarea inteligenței artificiale în asistarea jucătorilor de șah

Supervisor
Asist. univ. dr. Florentin Bota

Author
Cadar Eduard

2023

ABSTRACT

The usual way of searching for a move from a chess position is through tactics or strategy. Tactics are series of moves that would bring an immediate advantage, while strategy refers to a general 'sense' of advantage on the board, without too much calculation (having better developed pieces, better pawn structure etc.).[5]

Chess engines are usually built using a minimax algorithm, along with an evaluation function that takes into consideration the positions of the pieces on the board. This approach is able to find the moves that would bring material gain (capturing pieces) in a given depth, but cannot find moves that would slowly improve the position. Another approach would be training a neural network on professional chess games, which would be able to find good positional moves, but may be weak at finding tactical moves.

An algorithm combining these two approaches should yield better results than each of them on their own.

In the second chapter I will go over some of the techniques and algorithms used in programming chess engines, as well as some of the programming methods that modern chess engines utilize. In the third chapter I will describe the methods and algorithms I used in building my chess engine, and in the fourth chapter I will describe the tools and technologies with which I built the game and the engine.

Contents

1 Introduction **1**

2 Background **2**
2.1 Techniques/Algorithms used 2
2.2 State of the art chess engines 2

3 Methodology **3**
3.1 Move generation 3
3.2 Search 3
3.2.1 Minimax algorithm 3
3.2.2 Alpha-beta pruning 4
3.3 Evaluation 4
3.3.1 Board pieces 4
3.3.2 Neural network 5

4 Training **6**
4.1 Dataset 6
4.2 Layers 8

5 Technologies **9**
5.1 Chess game 9
5.2 Chess engine 9

6 Results and evaluation **10**

7 Conclusions **11**

Bibliography **12**

Chapter 1

Introduction

Chapter 2

Background

2.1 Techniques/Algorithms used

2.2 State of the art chess engines

Chapter 3

Methodology

In this chapter I will describe the three main steps taken by the engine to find the best move: generating legal moves for a position, searching for legally reachable positions from the current position, and then evaluating each one and determining which one is the best.[6]

3.1 Move generation

Regarding legality of the moves, there are two types of move generation: pseudo-legal and legal. In pseudo-legal move generation, the moves generated follow the normal rules of each piece's movement, but they are not checked to see if the king is left or moved into check. In legal move generation only legal moves are generated, checking beforehand if the king would be left in an illegal position. Pins, along with en passant moves are particularly difficult to check.

My approach is to first generate the pseudo-legal moves, then execute each generated move, check to see if it puts the king of the side that moved into check, and finally reverse the move. If the move doesn't leave the king in check, then I add it to the list of legal moves.

3.2 Search

3.2.1 Minimax algorithm

The minimax algorithm represents the search as a tree, where the root node is the current position, edges to its children are the possible moves, the children are the positions reached by making each respective move, and so on. Generating the nodes stops when an end position is reached (checkmate or draw), or at a given depth, since the tree grows very rapidly. The position of each leaf node is evaluated by an evaluation function, and then the results are propagated back to the root node. For each node that has all of its children evaluated, if it is white's turn we take the maximum value of the child nodes, and if it is black's turn we take the minimum value of the child nodes.[5]

The evaluations would be flawless if each branch of the tree was generated all the way down to the positions where the game had ended, but it would not be feasible from most positions, because it would take up more time and space than one could possibly allocate. Instead, the evaluations would become more accurate the deeper the tree is generated.

3.2.2 Alpha-beta pruning

Alpha-beta pruning is an optimization technique used in the minimax algorithm that can reduce the number of nodes that need to be evaluated in the game tree, making the search more efficient. Minimax searches through all leaf nodes to find the minimax value, while alpha-beta prunes leaves that have no influence on the outcome.

The algorithm works by maintaining two values: alpha and beta. Alpha represents the best score that the maximizing player has found so far, while beta represents the best score that the minimizing player has found so far.

As the algorithm searches through the game tree, it compares the scores of each possible move to alpha and beta. If a score is found that is worse than alpha (for the maximizing player) or better than beta (for the minimizing player), then that branch of the game tree can be pruned, since it will not lead to a better outcome.[1]

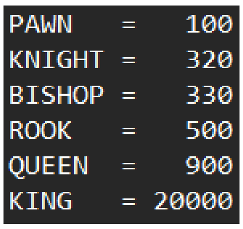
3.3 Evaluation

I used two methods of evaluating a chess board position. The first one is a simple evaluation function based on the position of the pieces on the board, and the second one is a neural network model, trained on master matches.

3.3.1 Board pieces

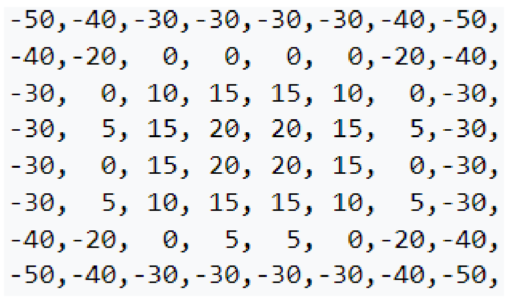
This evaluation function assigns a value to the board configuration based on the position of the pieces and the state of the game (opening/middlegame/endgame). I considered the state to be the opening if there were less than 13 moves made, middlegame if there are at least a queen or 3 minor pieces (rook/bishop/knight) on the board, and the endgame otherwise.

Figure 3.1



```
PAWN   =   100
KNIGHT =   320
BISHOP =   330
ROOK   =   500
QUEEN  =   900
KING   = 20000
```

(a) Base values for the pieces

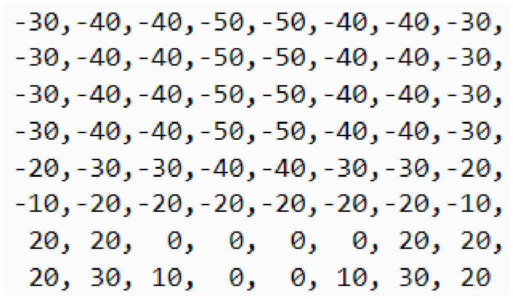


```
-50,-40,-30,-30,-30,-30,-40,-50,
-40,-20,  0,  0,  0,  0,-20,-40,
-30,  0, 10, 15, 15, 10,  0,-30,
-30,  5, 15, 20, 20, 15,  5,-30,
-30,  0, 15, 20, 20, 15,  0,-30,
-30,  5, 10, 15, 15, 10,  5,-30,
-40,-20,  0,  5,  5,  0,-20,-40,
-50,-40,-30,-30,-30,-30,-40,-50,
```

(b) Knight table

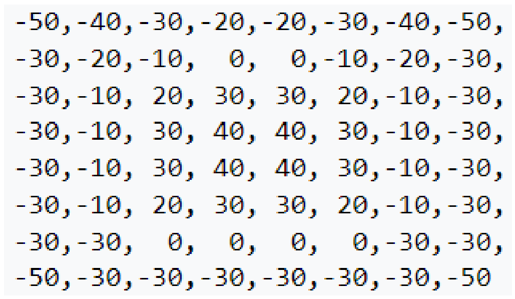
Each piece has a base value (fig. 3.1a), to which another value is added based on the square the piece occupies. Each piece has an 8x8 table, with a value for each square. The tables contain values as to encourage development of the pieces (example - fig. 3.1b shows the table for the white knights). The tables for black pieces are horizontally simetric to the ones for the white pieces. The king has an additional table for the endgame, because in the opening and middlegame it is encouraged to castle and discouraged to move far from the first rank (fig. 3.2a), while in the endgame it is encouraged to go towards the center and play an active part in the game (fig. 3.2b).[4]

Figure 3.2: King tables



```
-30,-40,-40,-50,-50,-40,-40,-30,
-30,-40,-40,-50,-50,-40,-40,-30,
-30,-40,-40,-50,-50,-40,-40,-30,
-30,-40,-40,-50,-50,-40,-40,-30,
-20,-30,-30,-40,-40,-30,-30,-20,
-10,-20,-20,-20,-20,-20,-20,-10,
 20, 20,  0,  0,  0,  0, 20, 20,
 20, 30, 10,  0,  0, 10, 30, 20
```

(a) King middlegame value table



```
-50,-40,-30,-20,-20,-30,-40,-50,
-30,-20,-10,  0,  0,-10,-20,-30,
-30,-10, 20, 30, 30, 20,-10,-30,
-30,-10, 30, 40, 40, 30,-10,-30,
-30,-10, 30, 40, 40, 30,-10,-30,
-30,-10, 20, 30, 30, 20,-10,-30,
-30,-30,  0,  0,  0,  0,-30,-30,
-50,-30,-30,-30,-30,-30,-30,-50
```

(b) King endgame value table

3.3.2 Neural network

For the second evaluation function, I trained a neural network on master chess matches, which I will expand in a separate chapter.

Chapter 4

Training

4.1 Dataset

The dataset consists of chess matches played at grandmaster level downloaded in PGN format from the website chessabc.com[2]

A PGN (portable game notation) file contains multiple chess matches. Each match has several headers (name of player with white pieces, name of player with black pieces, event it was played at, date it was played on, result, elo of players etc.), an empty line, and then the 'movetext section'. The matches in a PGN file are separated by two empty lines. Following is a sample PGN match:

```
[Event "F/S Return Match"]
[Site "Belgrade, Serbia JUG"]
[Date "1992.11.04"]
[Round "29"]
[White "Fischer, Robert J."]
[Black "Spassky, Boris V."]
[Result "1/2-1/2"]

1. e4 e5 2. Nf3 Nc6 3. Bb5 a6 4. Ba4 Nf6 5. O-O Be7 6. Re1 b5 7. Bb3 d6 8. c3
O-O 9. h3 Nb8 10. d4 Nbd7 11. c4 c6 12. cxb5 axb5 13. Nc3 Bb7 14. Bg5 b4 15.
Nb1 h6 16. Bh4 c5 17. dxe5 Nxe4 18. Bxe7 Qxe7 19. exd6 Qf6 20. Nbd2 Nxd6 21.
Nc4 Nxc4 22. Bxc4 Nb6 23. Ne5 Rae8 24. Bxf7+ Rxf7 25. Nxf7 Rxe1+ 26. Qxe1 Kxf7
27. Qe3 Qg5 28. Qxg5 hxg5 29. b3 Ke6 30. a3 Kd6 31. axb4 cxb4 32. Ra5 Nd5 33.
f3 Bc8 34. Kf2 Bf5 35. Ra7 g6 36. Ra6+ Kc5 37. Ke1 Nf4 38. g3 Nxh3 39. Kd2 Kb5
40. Rd6 Kc5 41. Ra6 Nf2 42. g4 Bd3 43. Re6 1/2-1/2
```

The movetext section contains the chess moves, move numbers, optional annotations and a concluding game termination marker (the result of the match). The result can be '1-0' (white won), '0-1' (black won) or '1/2-1/2' (draw). The moves are written in SAN (standard algebraic notation). SAN identifies each square on the board as a combination of file (a-h) and rank (1-8) (see fig. 4.1).

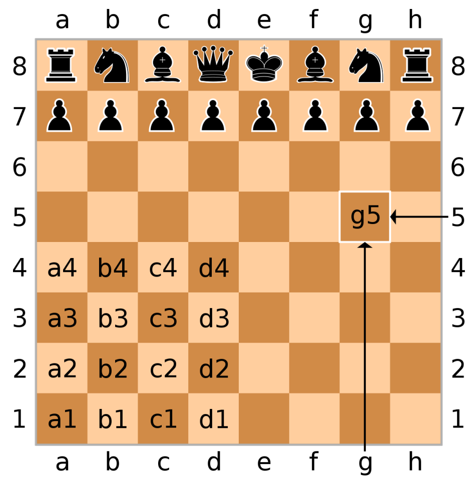


Figure 4.1: Board squares

Each piece can be identified by a letter (P - pawn, N - knight, B - bishop, R - rook, Q - queen, K - king). Simple SAN moves contain only the destination square, and the letter of piece that moves, if the piece is not a pawn (for example, pawn to c6 is written simply as c6, while knight to c6 is written as Nc6). There are additional rules for moves:

- Moves that capture a piece contain the 'x' character right before the destination square (ex: Bxb4)
- Castling kingside is written as 'O-O', while queenside castling is written as 'O-O-O'
- Moves of pawns to the last rank (promotion moves) contain the equal sign ('=') right after the destination square, followed by the letter of the piece that the pawn promotes to (ex: e8=Q)
- If there are more pieces of the same kind that can move to the destination square, the file of the originating square is added right after the piece (ex: Nbc6); if this does not solve the ambiguation, the rank of the originating square is added instead (ex: N8c6), and if neither of these works, both the file and the rank are added (ex: Nb8c6)
- If the move checks the opponent's king, a plus sign ('+') is added at the end of the move (ex: Qe7+); if the move checkmates the opponent's king, the octothorpe sign ('#') is added instead (ex: Qe7#)

[3]

4.2 Layers

[3]

Chapter 5

Technologies

5.1 Chess game

5.2 Chess engine

Chapter 6

Results and evaluation

Chapter 7

Conclusions

Bibliography

[1] Jeroen WT Carolus. Alpha-beta with sibling prediction pruning in chess. *Amsterdam: University of Amsterdam*, 2006.

[2] Chess ABC, 2023. URL: `https://www.chessabc.com/en`.

[3] Steven J Edwards et al. Standard portable game notation specification and implementation guide, 1994. URL: `https://ia902908.us.archive.org/26/items/pgn-standard-1994-03-12/PGN_standard_1994-03-12.txt`.

[4] Simplified evaluation function, 2018. URL: `https://www.chessprogramming.org/index.php?title=Simplified_Evaluation_Function`.

[5] Dominik Klein. Neural networks for chess. *arXiv preprint arXiv:2209.01506*, 2022.

[6] T Anthony Marsland. A review of game-tree pruning. *ICGA journal*, 9(1):3–19, 1986.