Playing Chess Using Minimax

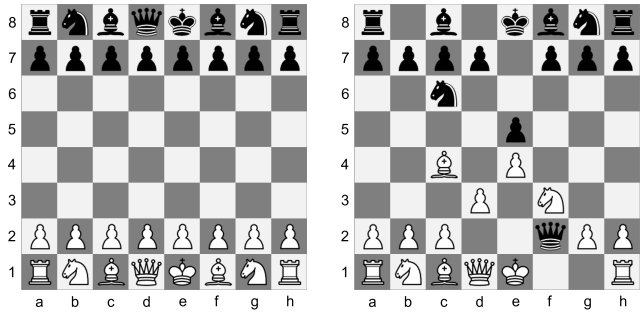
Luke Althoff
AlthoffLJ19@gcc.edu

Katherine Bennett
BennettKE20@gcc.edu

Keegan Woodburn
WoodburnKB20@gcc.edu

I. Introduction

The game chess is a widely popular competitive strategy board game. Given Chess' complexity and the strategy involved in a successful win, the field of game theory quickly gained interest in the challenge of developing an Artificial Intelligence capable of competently competing, both against other such AI systems, as well as humans themselves. From the first paper by Alan Turing proposing a Chess AI in 1951, Chess has continued to present itself as a popular area of AI development for many years.



(a) (b)

Fig. 1: An example of a starting state Chess board (a) and a "check" situation (b).

II. Terminology

A. Chess Board Setup & Goal State

The game of Chess is played on an 8 by 8 board of squares, with 16 pieces for each player, split between 6 types. Pieces are placed initially in the two rows on opposite sides of the board. Conventionally, one set of pieces is colored white, while the other set is colored black. Each row of the board is termed a "rank" and is given a number iterating upward from 1-8, with rank number 1 being the first row on the side of the white pieces and rank number 8 being the row closest to the black pieces. Each column of the board is termed a "file," with an alphabetic assignment of a-h, starting at the furthest left column. The player with the white pieces moves first. The goal state of the game for each player is reached when the opposing player's king is placed into "checkmate". Check occurs when a player's king piece is within the capture path of an opponent's piece, and checkmate occurs when no move by the player can place the king into a safe square [Fig. 1].

In order to reach this goal state, players move their pieces according to each piece's unique moveset.

Pawn pieces can move one square forward on each turn, with the exception being their initial movement from their starting location, from which they can move two squares forward. Bishop pieces can move any number of empty spaces diagonally away. Rook pieces can move any number of empty spaces vertically and horizontally. Knight pieces can move in an "L" shape over any number of occupied or unoccupied spaces, two squares in one direction and one square perpendicular to that direction. Queen pieces essentially combine the movement capabilities of the bishop piece and the rook piece. Finally, king pieces can move one empty space away in any direction, including diagonally, as long as that move does not place the king in check. [Fig 2.].

Players can also "take", or "capture", an opponent's piece is in their piece's allowed path of movement, removing it from the board as their piece replaces it. Pawns form the sole exception to this rule, only allowed to capture pieces which are located one square diagonally forward from the player's pawn, rather than within the pawn's general movement path.

B. Game Implementation

Generally, in more abstract AI research, the implementation of AI algorithms and other necessary tools are less discussed than the actual theory behind the AI being researched. However, since the implementation of our chess board is critical to the performance of the algorithms we discuss, we find it important that the implementation is noted.

We represent any given board with a bit-focused strategy called Zobrist Hashing. Boards are assigned their own 64-bit integers with bits representing the

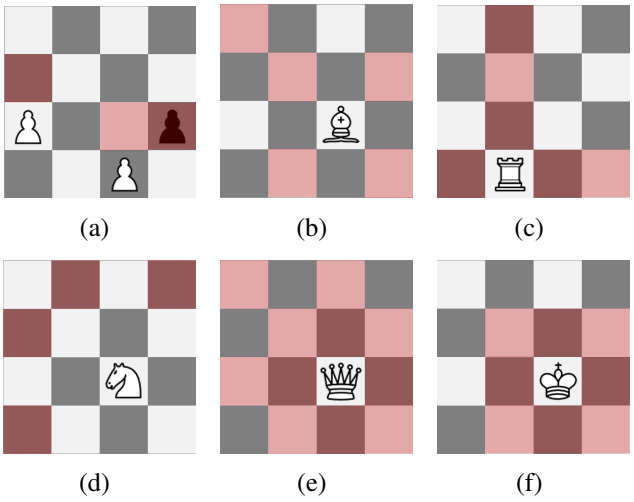


Fig. 2: A diagram depicting each unique moveset for each piece type (a) Pawn, (b) Bishop, (c) Knight, (d) Rook, (e) Queen, (f) King

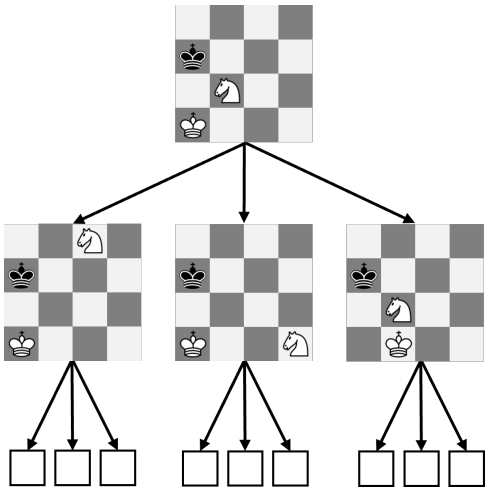


Fig. 3: A subsection of a game tree for Chess, showing three possible moves for some initial board state.

locations of pieces (1's indicate a piece at that location, while 0's indicate no pieces at that location). Each board stores integers for representing the locations of the full board piece set, and for each color set, and for each individual chess piece type (black pawns, white rooks, etc.). Integers for the individual colors and pieces are known as a sub-boards since they are subsets of all piece locations.

These integer boards can be used to implement other facets of the chess board as well. For example, if working with an individual piece, a bit board can be used to represent all the possible spaces it can move to. Thus, bit boards can be used to optimize implementations of other chess rule algorithms as well.

III. Planning Chess Moves

A. *MiniMax*

The most common approach for chess AI implementations is the MiniMax algorithm. Since chess is a zero-sum game, in which maximizing the player's chance of winning involves minimizing the opponent's chance of winning, each turn can be thought of as maximizing some evaluation function while the opponent *tries* to minimize it [4]. The MiniMax algorithm recursively implements a search tree that branches downward from some initial game board state. Different move choices create the different branches, and default MiniMax branches to all reachable states until all branches have reached termination states (checkmates in this case). In the MiniMax algorithm, two opposing players, a max player and a min player, alternate making moves. When the termination states are reached, a score is assigned to each board according to some evaluation function. If the termination board occurs on the max player's turn, scores are assigned based on the $max(board, player)$ function, where the score is a positive number whose magnitude is equal to how good of a board the state is for the max player. If the termination board occurs on the min player's turn, score are assigned based on the $min(board, player)$ function, where the score is a negative number whose magnitude is equal to how good of a board the state is for the min player. Moving up the search tree level by level, max player then selects moves and assigns scores to tree nodes equal to the the highest possible board score of its children, while the min player selects moves and assigns scores equal to the the lowest board score of its children. Once the top of the search tree is reached, the process of max and min player moves concludes with a final score corresponding to the best possible next move for the current player. Additionally, since each player in chess essentially follows the same play strategy, the min player and the max player scoring functions can follow this simple mathematical relation: $max(board, player) == -min(board, player)$. This modified algorithm is commonly called NegaMax, and it means a single scoring function can be used for both min scores and max scores. For the remainder of this report, however, the algorithm this chess AI implementation uses will continue to be called MiniMax.

To call MiniMax initially, maximizing must be set to true and initial depth must be set to zero. When the algorithm is at depth zero and an optimal move is found, the best moved is saved externally in a class

Algorithm 1 Basic MiniMax for Chess

Input: Whether we are $maximizing$, the current $board$, the current $player$ (black or white), an integer search tree $depth$

Output: A score value for the optimal move

```
if board is terminal then
    if maximizing then
        return max(board,player)
    end if
    return min(board,player)
end if
possible_moves = get all possible moves
board_copy = copy of board
if maximizing then
    max_score = -∞;
    for each move in possible_moves do
        board_copy.make_move(move)
        score = MiniMax(not maximizing, board_copy, opponent of player, depth+1)
        if score > max_score then
            max_score = score
            if depth == 0 then
                overwrite current saved best move
            end if
        end if
        board_copy.undo_last_move()
    end for
    return max_score
end if
min = ∞;
for each move in possible_moves do
    board_copy.make_move(move)
    score = MiniMax(not maximizing, board_copy, opponent of player, depth+1)
    if score < min then
        min = score
    end if
    board_copy.undo_last_move()
end for
return score
```

variable. New best moves overwrite previously saved best moves.

B. Improving Minimax Efficiency

The primary concern when implementing game tree-based algorithms is the size of the tree itself. MiniMax runs in $O(b^d)$ time [1], and in the case of chess, the game tree size approaches 10^{40} nodes, necessitating enhancements to efficiently program the AI. This was done in two ways: by setting a max tree depth and by incorporating alpha-beta pruning.

Firstly, in order to reduce computational time, a maximum depth was set for the search tree, preventing the algorithm from having to search the full game tree size. The min and max game scoring functions in this scheme must thus be capable evaluating non-checkmated boards. Secondly, in order to speed up the MiniMax algorithm, Alpha-Beta pruning was implemented, adding 'Alpha' and 'Beta' values to each node in the game search tree. The 'Alpha' value denotes the highest board score choice found so far along the path of the Max player, while the "Beta" value denotes the lowest board score choice found so far along the path of the Min player. If at any point the current branch score is worse than the saved 'Alpha' or 'Beta' values, that branch can be pruned. Alpha-beta pruning improves runtime by a constant value [1], and it speeds up the algorithm through avoiding searching full subsections of the game tree [2].

In order to consistently generate best possible moves through the MiniMax algorithm, effective evaluation function is required for generating scores. For our Chess AI, we constructed a multifaceted evaluation function, considering both individual piece strengths and overall board states.

IV. Evaluation Functions

Most implementations of Chess AI use adopt a two-fold board evaluation strategy, where boards have a base score derived from a piece-strength system and where additional points are scored based on parameters like piece mobility, attack potential, etc. The board evaluation methods used in this AI implementation are derived from work by Lyudmil Tsvetkov [3].

In this report, points will be abbreviated as ps, centipoints as cps, and millipoints as mps.

A. Piece Strength Based Scoring

Piece strength based scoring allows the AI to consider captures and piece losses throughout the game, a key metric in player advantage.

1) Piece Strength Assignments: The piece-strength value set used for our AI implementation is shown in [Fig. 4]. Such a system provides a point value for each piece type according to a metric called piece strength (ps).

A total piece-strength based score for a player is calculated by summing the number of each piece type for the player on the board multiplied by the strength of those pieces, then subtracting the piece strengths of the opponent in the same way.

Fig. 4: The piece strength (ps) point values assigned to each piece type for our implementation.

2) Grading Based on Game Stage: We also further refine these piece scores through grading, which adds or subtracts some percentage of a piece's score based on the stage of the game. The grading system implemented grades pawns based on the number of other pieces on the board and grades pieces based on the number of pawns on the board.

Pieces are graded on based on four types of pawn positions: closed positions, where 13-16 pawns remain; semi-closed positions, where 9-12 pawns remain; semi-open positions, where 5-8 pawns remain; and open positions, where 0-4 pawns remained. The grading of non-pawn piece types according to each of these positions is detailed in Figure 5.

Position Type	Queen	Bishop	Rook	Knight
Closed	-10%	0%	-15%	10%
Semi-Closed	-5%	-10%	5%	10%
Semi-Open	20%	15%	10%	-10%
Open	30%	20%	10%	-15%

Fig. 5: A table showing grading of non-pawn pieces based on position type.

Pawn pieces are graded based on the game stage, determined by the overall sum of ungraded piece strengths (including both black and white pieces). Four game stages are defined based on this sum, and the pawn grading according to these stages are shown in Figure 6. This scheme weights pawns to be more powerful as the number of pieces on the board decreases.

Overall Piece Strength	45-60ps	30-45ps	15-30ps	0-15ps
Pawn Grading	no change	5%	10%	15%

Fig. 6: A table showing grading of pawn pieces based on total piece strength of the board. Positive percentages correlate to an addition of that percentage of the ps for that piece, while negative percentages correlate to a subtraction of that percentage of the ps for that piece.

B. Additional Parameters for Board Scoring

The total Board Score at each state is calculated using the base score derived from the piece strengths as noted above. This score is further adjusted based on additional parameters such as position, mobility, attack potential, defense potential, and king security. Specific strategies are also considered for opening and endgame stages.

According to the game evaluation methods discussed in the work by Lyudmil Tsvetkov [3], the game is considered in three distinct sections: opening, where the total board ps value is 45-60, middlegame, where the total board ps value is 30-60, and endgame, where the total board ps value is 0-30. Some evaluation strategies only apply in specific stages. If a specific stage is not noted, however, the evaluation function applies in any stage.

1) Opening Control: During the opening section of the game, the following adjustments are made to the total board score based on central-board position control.

The focal center of a Chess board includes the squares e4, d4, e5, and d5. For each Pawn occupying one of these squares, a . For non-Pawn, non-Queen pieces, a board score value of 0.20 was added, and for Queen pieces a value of 0.30 is added.

If a Pawn is in squares c3-f3 or c4-f4, it is considered as "in control" of a center square and thus a value of 0.10 is added to the board score. For each square controlled by a non-pawn piece, the player gets 0.10 added to their current board score.

Any pieces in the square bounded by c3-f3-f6-c6 excluding the focal square also give an increase in board score of 0.10.

2) Opening Order of Development: The board score is also altered based on which pieces are developed, or moved, first. If a Knight is moved before the Bishop, a score of 0.2 is added. If the Queen is moved prior to moving two minor pieces, a score of 0.3 is subtracted. If a Rook is moved before two minor pieces, a score of 0.5 is subtracted. Finally, if a piece is played twice within the opening, a score of 0.35 is subtracted.

3) Position: The position score of a player's pieces is an important factor for purposes of encouraging pieces to move towards the opposing side of the board from their starting locations. As such, a bonus or deduction is appended to the score based on each piece's rank location according to Figure 6. These bonus and deduction values apply in the opposite direction for pieces whose start locations are in ranks 7 and 8.

Rank	1	2	3	4
Bonus (mps)	-15	15	30	45
Rank	5	6	7	8
Bonus (mps)	60	75	60	30

Fig. 7: A table showing the bonus (in mps) applied to the board score for each piece at a given rank 1-8.

4) Mobility: Mobility of a player's pieces is another factor to consider, since pieces can be more fully utilized if they more free to move. For example, incorporating this parameter would encourage the AI to move pieces so they are not trapped behind the AI's own pieces, advancing them on the board.

In this implementation, for each free space the AI's own piece has access to, 10cps is added to the board's mobility score.

5) Attack Potential: A piece is considered to be attacking another piece if it can capture that piece in the next move. Attacking pieces can force a player's opponent into retreat and provide opportunities for captures, so this is another valuable parameter to consider. For each non-king piece attacked by the AI, 1/10 of the attacked piece's strength is added to the attack potential score.

6) Defense Potential: A player's piece is considered defended if that piece is in the path of any of the player's other pieces. If a defended piece is captured by an enemy piece, the the player can use a defending piece to capture that enemy piece. Considering this factor allows the AI to develop a secure positions and defend against potential enemy attacks.

7) King Security: How secure the AI's king is can be numerically analyzed based on how many pieces occupy the king's shelter. The kings shelter is divided into four parts: (1) the immediate shelter, which consists of squares directly vertically, horizontally, or diagonally adjacent to the king; (2) the cross shelter, which consists of squares vertically and horizontally two spaces away from the king; (3) the diagonal shelter, which consists of squares diagonally two spaces from the king, and (4) the sinuous shelter, which consists of the squares a knight's move away from the king. The cross, diagonal, and sinuous shelter are all considered the wider shelter region.

For each pawn and non-pawn piece in any of these regions, points were added to the king security score based on the table in Figure 8.

8) King Attack: Attacking the enemy king and the enemy king's shelter is another parameter to consider, since to win the game the AI must be able to reach checkmate. For each piece attacking the king, 1/10 that piece's value is added to the king attack score. For each piece attacking an enemy piece in the king's immediate shelter, 1/20 the attacking piece's value is added to the king attack score. For each piece attacking an enemy piece in the enemy king's wide shelter, 1/30 that piece's value is added to the king attack score.

Piece Type	Location	Point Bonus
Pawn	Immediate Shelter	0.5
Pawn	Cross Shelter	0.25
Pawn	Diagonal Shelter	0.25
Pawn	Sinuous Shelter	0.17
Bishop	Any Shelter Location	0.3
Knight	Any Shelter Location	0.25
Rook	Any Shelter Location	0.15
Queen	Any Shelter Location	0.1

Fig. 8: A table showing point additions for each piece type within specific king's shelter regions.

9) Endgame Additional Strategies: In the endgame, additional points are scored for the AI's king's mobility and position. For each free space the king can move to, an additional 15cps are awarded. If the white king is in c3, d3, e3, or f3, then 0.25 points are awarded. If the white king is in c4, d4, e4, or f4, then 0.35 points are awarded. For a white king on the fifth rank or sixth rank, 0.50 and 0.75 points are awarded, respectively. The black king's position is scored based where it's placement mirroring these positions.

C. Calculating Total Board Score

The total board score is calculated using both piece strengths and the additions noted above. However, summing all the additional parameter scores outweighs the base piece strength values, meaning captures are not prioritized over the parameters dictated. To compensate for this, all parameter scores were divided by a factor of four before summing with the piece-strength's board. The total board score is thus calculated as the base piece score plus one-fourth the sum of all additional parameter scores. The MiniMax algorithm chooses moves based on this score.

V. Results

We can evaluate the AI's performance based on the efficiency of our implementation and the strategic intelligence of its moves. For the former, runtime and ply-depth good indicators of performance.

A. Runtime Results

Prior to implementing evaluation functions (using all 0 board scores, so no alpha-beta pruning), the AI consistently reached a 5-ply search depth in about 3 seconds, and 6-ply search depths had average runtimes of 18 seconds.

After implementing the evaluation functions, the AI made notably strategic decisions, but was unable to meet the same runtimes. While MiniMax was likely pruning more boards with the additional evaluation functions, the code runtime was such that any time benefit of more pruning was negated. The AI could reach 3-ply on average in 15 seconds, with occasional 90 second outliers, and it could reach 4-ply depth on average in 3 minutes, with outliers being in the 10 minute range. The AI unable to reach 5-ply search depth. The strategic boost, while it was inconsistent, did compensate for loss in runtime efficiency.

B. AI Behavior

In general, it is clear that the AI is making decisions based on the evaluation functions. It has made moves to defend pieces, develop pieces, capture pieces, protect the king, and more, which indicate that the evaluation functions were successful in influencing the AI's decision-making. The evaluation functions, however, do not consistently support good moves. In general, when a poor move were performed, the move successfully supported one of the evaluation parameters, but neglected another in ways that were directly detrimental to strategic play. For example, the AI frequently moved the queen piece early in the game rather than pieces more important to develop, likely due to the high mobility score granted by this piece.

The AI was also generally more defensive than offensive, likely because we are quantitatively considering more defensive parameters than attacking parameters. There were several occasions in which the AI could have taken pieces but opted for a more defensive move instead.

Since ply depth was limited, and due to the defensive preferences of the AI parameters, the algorithm also performed poorly in pursuing endgame situations. In general, the AI did not prioritize attacking in such a way as to make endgame situations attainable.

However, while the AI did not consistently make strategically beneficial moves, the effects of parameters were still evident in positive ways when good moves were performed. More optimized parameter weights, including more weights based on particular game stages, could improve strategic behavior significantly.

C. Chess Expert Analysis

In initial tests, the AI was able to successfully defeat a novice challenger in multiple games.

In order to gather empirical data from more experienced players regarding the intellect of the AI, the Grove City College chess club president, John Treusch, was asked to challenge our AI algorithm.

Treusch played multiple games against the AI at 3-ply and 4-ply depths. Based on these games, the president rated the AI at an Elo score of 400-500, with "Elo" being a metric denoting relative strength of a player. The 4-ply depth AI also performed better than the 3-ply.

The chess club president did note a few particular issues in its performance. These issues included a tendency for the AI to move the same piece multiple times in a row early in the game rather than developing other pieces, and a tendency to move the queen piece early in the game rather than pieces of lesser value. The latter issue is likely due to poor weighting in the evaluation function leading it to focus on mobility rather than piece value.

VI. Conclusion

Overall, the AI was semi-competent in its move choices, and had reasonable runtimes up to 4-ply. However, the evaluation functions adopted from the paper written by Tsvetkov resulted in relatively unpredictable behavior. The relative weights of the evaluation functions Tsvetkov illustrated did not work well when only incorporating some of them. Re-weighting of the board state evaluation function additions we chose to include was necessary for the AI to prioritize basic captures, and the evaluation functions semi-frequently resulted strategically detrimental moves. Additional research would necessitate experimenting further with weighting of our parameters in order to better develop the AI move-set in consistently beneficial ways. Most chess engines also incorporate opening and ending databases, which could also be a source of improvement for our AI.

References

[1] S. J. Russell and P. Norvig, 'Artificial Intelligence,' Pearson Education, 3rd ed., 2009.

[2] Rodriguez, Roberto Marrero, 'Developing a Chess Engine', 2021.

[3] Tsvetkov, Lyudmil, 'Little Chess Evaluation Compendium', 2012.

[4] Walker, April, 'The Anatomy of a Chess AI', 08 2020.