The Design and Use of a Chess System

Brandon Cheshire
Department of Neuroscience
Brock University
St. Catharines, Canada
Email: bc14tm@brocku.ca

I. Introduction To Chess

A. Overview and History

Chess is an abstract strategy board game for 2 players. The board has 64 squares laid out in an 8x8 grid, which is mapped to a rank and file scheme. There are 32 pieces, 16 for each player (black and white). The goal of the game is to have the opponent's king piece in a position where it cannot escape the threat of attack from your pieces (checkmate). A player can also resign, in the case of losing most of their pieces (referred to as material) and/or in a position unlikely to win. Draws can occur in several ways. Since both players can see all the pieces on the board at any time, the game is said to have perfect information.

Chess has a long history stretching back to the first centuries of the common era in India (though some evidence suggests China). The pieces and their allowed moves have evolved over the centuries until being standardized in the 19th century into the game that we know today (Figure 1).

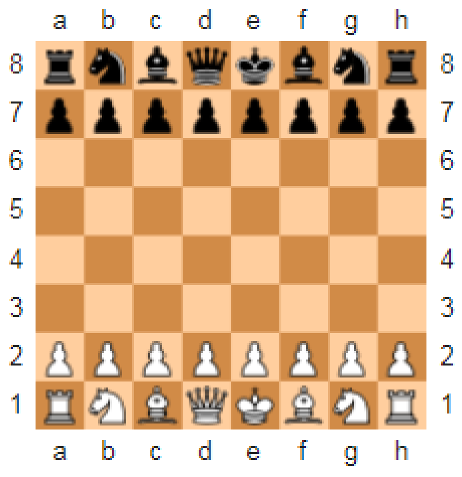


Fig. 1. The initial configuration of a chess board

Programming a computer to play chess began shortly after modern computers were invented. A chess engine will evaluate a given board for the best move to make. It wasn't until the late 1970s that computer chess engines began to defeat top human players. It is generally agreed that a computer player needs to see at least 5 moves ahead ("5 ply") to play well against a seasoned human player. This is therefore the recommended search depth when running the chess system. A computer match plays with a noted decrease in skill at lower depths.

B. Rules and Pieces

Each player begins with the same number and types of pieces: 8 pawns, 2 bishops, 2 knights, 2 rooks, a queen, and a king. Each piece type has a unique set of positions it can move to from its current location (Figures 2-8). Pieces that can move indefinitely along a vector in specific directions (until a square is already occupied) are referred to as sliding pieces, these are the rooks, bishops, and queens. Knights, pawns, and kings have unique non-sliding moves. A move is said to be a capture when a piece replaces another on a square in the board. This is the case in all captures except en passant (Figure 9). With the exception of the knight, pieces cannot jump over one another.

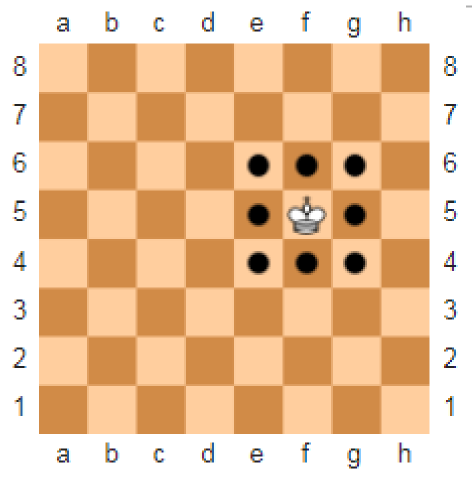


Fig. 2. Moves of a King

The **king** can move to any square immediate adjacent to it in any direction (Figure 2). A king can also perform a special move called castling with a rook, if neither piece has moved yet, there are no pieces between the king and the rook, and the king and squares for the move are not under attack. Depending on which rook the move is made with, this is referred to as queenside or kingside castling (Figure 4). The king cannot be captured, only put in check. Check occurs when a player's king is under threat of attack from an opponent piece, but the king can move to a legal position under no threat of attack. If the king is in check and has no escape moves, then checkmate occurs. Stalemate occurs when the king is not in check but doesn't have any legal moves available.

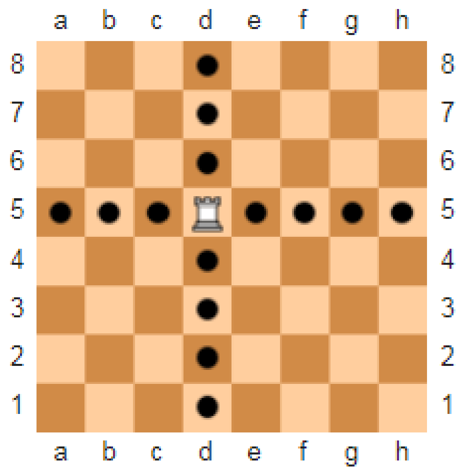


Fig. 3. Moves of a Rook

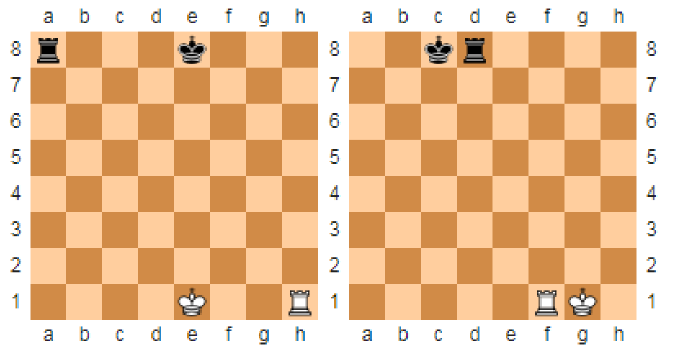


Fig. 4. Castling. The left board shows the positions before castling, the right board shows the final positions of queenside (black) and kingside (white) castling.

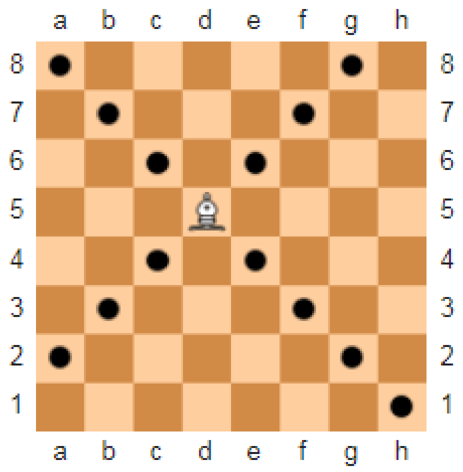


Fig. 5. Moves of a Bishop

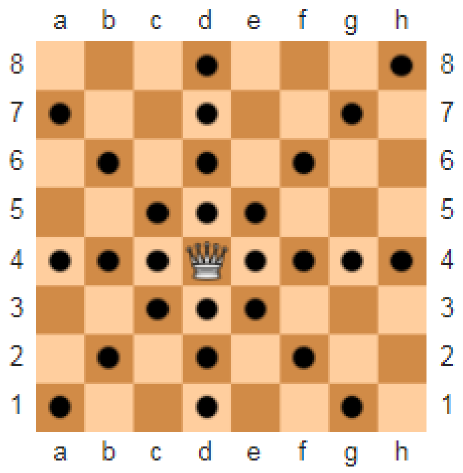


Fig. 6. Moves of a Queen

The **rook** is a sliding piece that can move in vectors horizontally and vertically (Figure 3). In castling, the king moves two squares kingside or queenside, and the rook jumps over the king (Figure 4). Note that the rook is allowed to jump over the king for a castling move, but otherwise this is not allowed.

The **bishop** is a sliding piece similar to the rook. Instead of moving along vertical and horizontal vectors, the bishop can move in vectors of either diagonal (Figure 5). Since a bishop is limited to diagonal movement, each player has a bishop limited to either light or dark tiles on the board, and can be referred to as the "light-squared" or "dark-squared" bishop.

The **queen** is considered the strongest piece in the game. The queen is a sliding piece that is allowed to move in any vector direction: horizontal, vertical, and diagonal, essentially combining the moves of the rook and bishop (Figure 6). The initial configuration of the chess board ensures that each queen is "on her colour," referring to the dark and light tiles of the board (for black and white, respectively) (see Figure 1). Queens and rooks are considered "major pieces."

The **knight** has a unique move set, essentially it moves in what can be described as an L. The knight can either move two squares horizontally and one square vertically, or vice versa, alternating between light and dark tiles (Figure 7). As mentioned, the knight is the only piece that can jump over pieces, making it particularly effective in closed positions (board configurations lacking open lines (ranks/files)). This also allows it to move ahead into the centre of the board without moving pawns. Knights and bishops are referred to as "minor pieces."

The **pawn** can only move forwards towards the opponent's side. If it's the pawn's first move it may move forwards two tiles, otherwise it may advance one square forward per move. Pawns cannot capture a piece immediately in front of it, instead it becomes blocked by this piece. Pawns can only capture a piece diagonally on the rank immediately ahead of it, except in the special case of en passant capture (Figure 9). Pawns that advance to their respective final rank of the board can be promoted, meaning the player can exchange the pawn for another piece. This opens up the possibility of a player

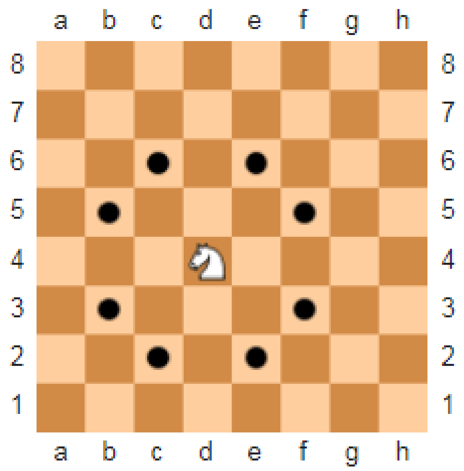


Fig. 7. Moves of a Knight

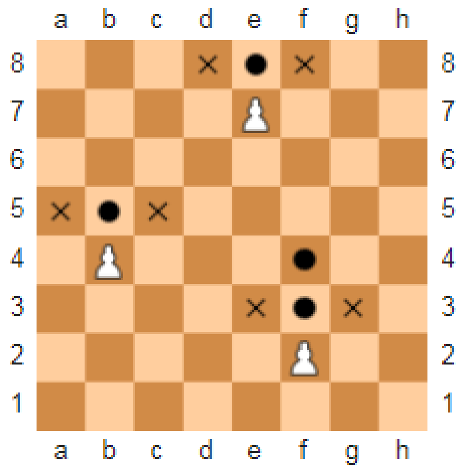


Fig. 8. Moves of a Pawn. Captures are denoted with an x, moves with a black circle. From left to right: a pawn advancing one step forward; a pawn reaching its final rank will be promoted and exchanged for another piece; a pawn may advance two steps forward on its first move.

having more of any one piece type than they started out with (for example, 2 queens).

II. Representation

The chess system acts on its representation of the chess board to search and evaluate which moves to make. Therefore the choice of representation is one of the first considerations when designing a chess system. There are **piece centric** and **board centric** representations, the former keeps track of pieces and associates that with a position on the board, the latter keeps track of whether or not a square is occupied, and associates a piece with that square.

A. Bitboards

Bitboards (also known as bitmaps or bitsets) are a piece centric representation–simply sets containing 64 elements which map perfectly to the squares on a chess board, with each bit representing one square on the board. In Java, the 64-bit long data type is used for this system. The advantage of using bitboards with 64-bit processors is that logical bitwise operations (AND, OR, NOR, and XOR) are available and complete in one cycle on virtually all modern CPUs.

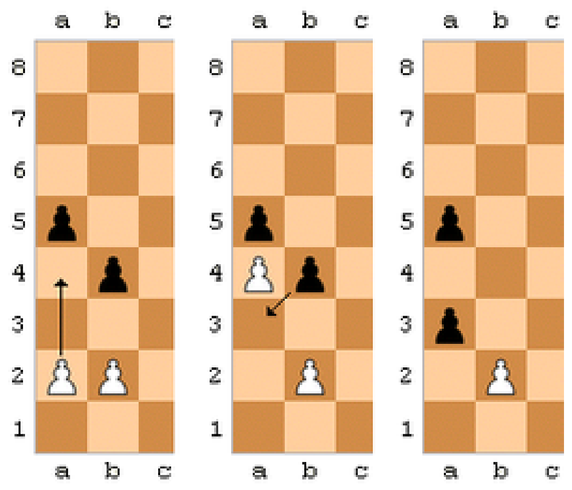


Fig. 9. En passant pawn capture. If a pawn makes a double-step move and passes an opponent pawn that would have captured it had it made a one-step move, the opponent pawn can cross diagonally behind and capture the player's pawn.

a	b	a ∧ b
0	0	0
0	1	0
1	0	0
1	1	1

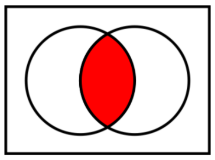
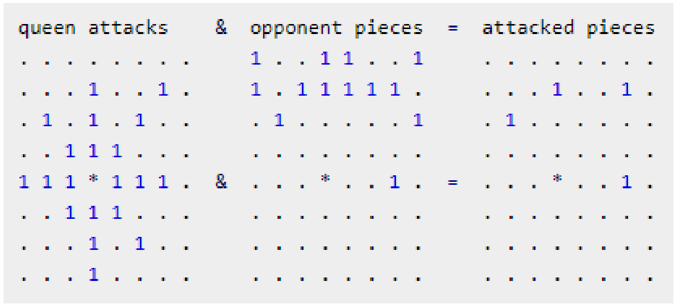


Fig. 10. Binary truth table for boolean conjunction and venn diagram for set intersection. Both concepts are captured in the bitwise AND operator.

To represent all the information captured on a chess board generally each piece type and colour are given their own bitboard. By combining the information from all bitboards you capture the chess board. Any one-bit in a bitboard indicates the presence of that colour's piece type on the square that maps to the position of the bit. Another consideration in this representation is how to map the positions to the binary digits of the bitset. For this system, the least significant bit (LSB) is mapped to h1 and the most significant bit (MSB) is mapped to a8. For example, the initial position bitboard for white bishops at c1 and f1 would be as follows:

$$\begin{array}{cccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 & 0
\end{array}$$

For this representation to be useful, one makes use of bitwise boolean operations, which covers both set and logic operations. For instance, bitwise AND captures set theory's intersection and boolean algebra's conjunction (Figure 10). Operations are performed in a bitwise manner, meaning the same positions are evaluated in turn among all sets undergoing the operation (Figure 11). Since all bitboards in the system contain 64 elements, the results are valid.

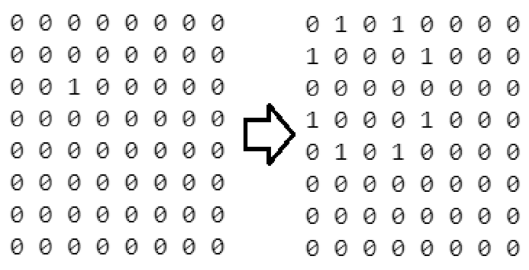


```
queen attacks     &  opponent pieces  =  attacked pieces
. . . . . . . .      1 . . 1 1 . . 1     . . . . . . . .
. . . 1 . . 1 .      1 . 1 1 1 1 1 .     . . . 1 . . 1 .
. 1 . 1 . 1 . .      . 1 . . . . . 1     . 1 . . . . . .
. . 1 1 1 . . .      . . . . . . . .     . . . . . . . .
1 1 1 * 1 1 1 .   &  . . . * . . 1 .  =  . . . * . . 1 .
. . 1 1 1 . . .      . . . . . . . .     . . . . . . . .
. . . 1 . 1 . .      . . . . . . . .     . . . . . . . .
. . . 1 . . . .      . . . . . . . .     . . . . . . . .
```

Fig. 11. Example of bitwise AND operation on two bitboards. In this case the queen attacks are ANDed with opponent pieces and the resulting bitboard represents the attacked pieces.

B. Move Generation

Move generation refers to generating the moves a piece can make on the board given its current position. In a bitboards paradigm, move generation begins with a single occupancy bitboard representing the location of the piece in question, and results in a bitboard representing all the positions on the board it can move to (Figure 12). **Pseudo-legal** moves are generated without regard to whether they leave the king in check, whereas **legal** moves takes this into account. In this system, pseudo-legal moves are allowed, and checking the king's status happens before the next move. The pseudo-legal moves bitboards are considered with other bitboards (i.e. occupancy, opponents, etc.) to generate attacks sets which are moves that can actually be made without attacking one's own pieces or passing through enemy occupied squares in the case of sliding pieces.



```
0 0 0 0 0 0 0 0      0 1 0 1 0 0 0 0
0 0 0 0 0 0 0 0      1 0 0 0 1 0 0 0
0 0 1 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0  ⇨   1 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0      0 1 0 1 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
```

Fig. 12. Pseudo-legal move generation for a knight. Given a bitboard for a knight on c6, the resulting bitboard of moves is returned. The resulting bitboard can be further operated on with other bitboards representing enemy and friendly pieces to produce a valid move set.

III. Search

Many chess systems utilize a game tree-based search to find and evaluate the best moves to make on any given board. The game tree is a subset of the search space of the game, where boards represent nodes and moves represent directed edges between them.

A. Minimax Algorithm

This system utilizes the minimax algorithm with alpha-beta pruning to improve performance (Algorithm 1). For this system, minimax uses a Type A strategy, performing a brute-force search of all possible moves to a given depth for evaluation.

Algorithm 1 Minimax Algorithm

```
1:  function MINIMAX (node, depth, maximizingPlayer)
2:      if depth = 0 or node is a terminal node then
3:          return the heuristic value of node
4:      if maximizingPlayer then
5:          bestValue := $-\infty$
6:          for each child of node do
7:              v := minimax (child, depth $-$ 1, FALSE)
8:              bestValue := max(bestValue, v)
9:          return bestValue
10:     else (* minimizing player *)
11:         bestValue := $+\infty$
12:         for each child of node do
13:             v := minimax (child, depth $-$ 1, TRUE)
14:             bestValue := min(bestValue, v)
15:         return bestValue
```

Minimax is used in two player zero-sum games, operating on the assumption that each player is equally intelligent and each will try to foil the brilliant moves of the other. A depth first search to the specified depth is performed and a heuristic value is assigned to each leaf node. At each level coming back up the search tree, the score that optimizes the score of the side to move at that level is assigned to the parent node at that level. The player making the call will have to optimize their choice by either maximizing the minimum or minimizing the maximum (Figure 13).

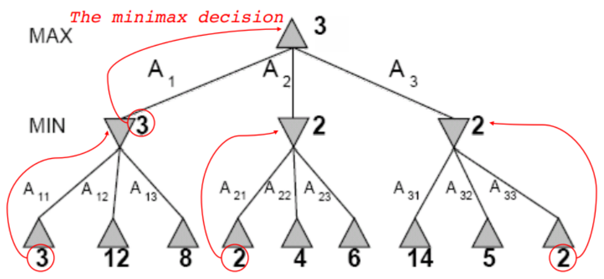


Fig. 13. Minimax example of 2-ply game. Min player will choose the lowest scores from the leaf nodes, and the maximizing player has to minimize their loss by maximizing the minimum player's choices.

B. Alpha-Beta Pruning

The search depth is limited by time considerations. The average number of moves available to a player at any turn is approximately 35. Therefore, the average branching factor in a chess decision tree is 35, meaning that the nodes that are considered in a naive minimax call is approximately equal to 35^d, where d is the search depth ("ply"). For instance, a search depth of 5 would require approximately 52 million node considerations. Clearly, examining the entire game tree in a timely manner would be infeasible, considering that the average game of chess runs for 80 ply. To improve performance of minimax, the alpha-beta pruning algorithm is used to remove branches from consideration that do not influence the final decision (Algorithm 2). This pruning strategy can optimally reduce node consideration by half while returning the same result, effectively increasing the depth of search that can be accomplished in a given time.

Algorithm 2 Alpha-Beta Pruning

```
function ALPHABETA (node, depth, $\alpha$, $\beta$, maximizingPlayer)
    if depth = 0 or node is a terminal node then
        return the heuristic value of node
    if maximizingPlayer then
        v := $-\infty$
        for each child of node do
            v := max (v, alphabeta(child, depth − 1, $\alpha$, $\beta$, FALSE)
            $\alpha$ := max($\alpha$, v)
            if $\beta \leq \alpha$ then
                break (* $\beta$ cut-off *)
        return v
    else
        v := $+\infty$
        for each child of node do
            v := min (v, alphabeta(child, depth − 1, $\alpha$, $\beta$, TRUE)
            $\beta$ := min($\beta$, v)
            if $\beta \leq \alpha$ then
                break (* $\alpha$ cut-off *)
        return v
```

Alpha-beta keeps track of the best score that the maximizing player (α) and minimizing player (β) can hope for. These are initialized to the worst score possible for each player, and whenever the value of β becomes less than α, the maximizing player can ignore further descendents of the current branch as they won't ever be used. Similar logic applies to the minimizing player (Figure 14).

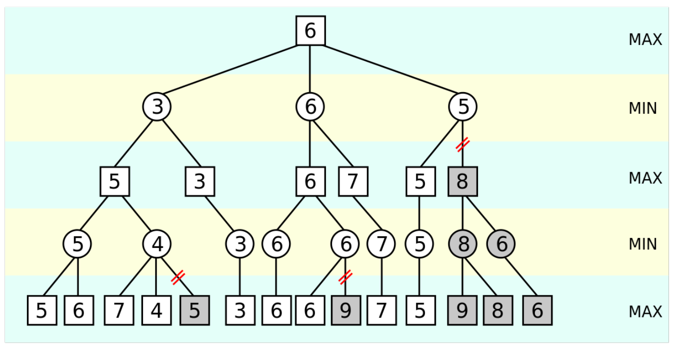


Fig. 14. An example of alpha-beta pruning. Subtrees that need not be explored are greyed out since it is known their value will be worse or equal to the value of other subtrees.

IV. Evaluation

The heuristic value associated with the leaf nodes of the decision tree are assigned using an evaluation function. This system considers material, piece square tables, and mobility, summing these to assign the heuristic value to a given chess board configuration.

A. Material

Material refers to the sum of piece values that each side fields on the board. It is the most influential aspect of evaluation. Generally it is scaled to the "centipawn," which sets the value of a pawn at 100 and allows other aspects of board evaluation to be considered while preserving the score as an integer value, avoiding fractions. The score is returned from the difference of the player's material values. Table I gives the piece values for the system.

TABLE I
Piece values (as suggested by Shannon, 1949)

Piece	Value
Pawn	100
Knight	300
Bishop	300
Rook	500
Queen	900
King	10000

B. Piece-Square Tables

Using piece-square tables a value can be assigned to a piece based on its location on the board. Tables are generated for each piece type: positive values for favourable positions, negative values for unfavourable positions, and 0 for neutral positions. This can encourage the engine to play better, for example the table for white pawns encourages advancement, and center pawns are discouraged from remaining unmoved. Note that mirroring the following table vertically produces the same values for black pawns:

$$\begin{array}{cccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
50 & 50 & 50 & 50 & 50 & 50 & 50 & 50 \\
10 & 10 & 20 & 30 & 30 & 20 & 10 & 10 \\
5 & 5 & 10 & 25 & 25 & 10 & 5 & 5 \\
0 & 0 & 0 & 20 & 20 & 0 & 0 & 0 \\
5 & -5 & -10 & 0 & 0 & -10 & -5 & 5 \\
5 & 10 & 10 & -20 & -20 & 10 & 10 & 5 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}$$

A simple evaluative function utilizing material and piece-square tables is considered sufficient for a decent playing chess engine.

C. Mobility

The number of moves open to a player in a given position is a measure of mobility (Figure 15). The premise of calculating mobility for evaluation is that a stronger player position entails having more options to choose from. It has been shown empirically that all other evaluation considerations being equal, higher mobility is positively correlated with games won. A player's mobility score is calculated by intersecting the summed attack set bitboard of each players pieces with the complement of their pieces' bitboard and taking the difference of these values.

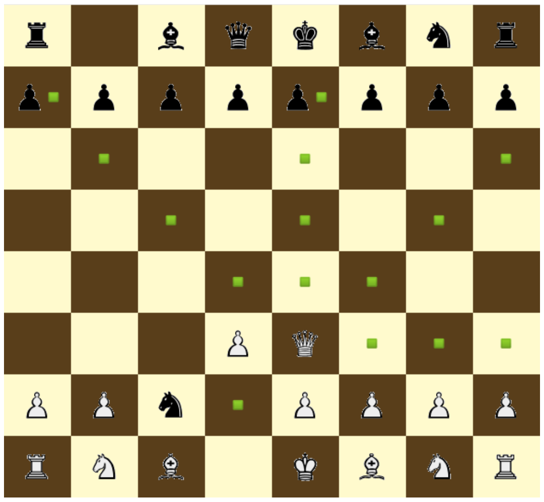


Fig. 15. An example of piece mobility. This queen on e3 has many moves open to it (denoted by green squares), while in its starting position behind the pawn shield it has none.

V. Class Structure

- **Board:** Stores the bitboards that make up the board, castling rights, and methods for evaluation, game status, and making moves.
- **Move:** Representation for moves. Stores from and to positions and a code for marking special moves.
- **MoveGenerator:** This class returns all the legal moves for a given player to the engine, or a given piece by GUI.
- **Engine:** Contains the minimax and alpha-beta algorithms that return the best move to make for a given player on a given board.
- **GUI:** Plugs the rest of the system into a graphical user interface.
- **Chess:** This is the main class for system execution.

VI. User Interaction

Upon executing the main class, the user is asked to select the game setup. This determines whether the user plays against the computer or watches the computer play against itself.

Following game setup, the user is asked to select the search depth. The recommended setting is 5 ply—shallower depths lead to less challenging computer player, deeper depths will require significantly more time waiting for computer moves. After the game is configured, a window with a menu bar and a 2-dimensional graphical representation of the board will appear. The preferences menu allows for flipping the board vertically.

In a human vs. computer match, the user can use the mouse to interact with their pieces on the board. Left-clicking the mouse on a square with one of their pieces will select that piece and highlight its legal moves as a set of green squares overlaid on the board tiles, including king castling moves and en passant pawn captures (Figure 16). Right-clicking deselects the piece, as does left-clicking a non-legal square. Left-clicking the mouse on one of the legal squares of a selected piece moves the piece (Figure 17). To undo a move, select Undo Last Move from the Options menu. To reset the game to the initial board configuration, select Reset Board from the Options menu.

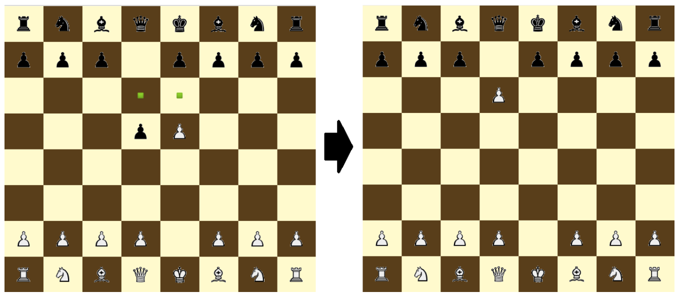


Fig. 16. An example of in-game en passant pawn capture. The black pawn has made a double push and is now beside the white pawn (left). The en passant square is highlighted as a legal move for the white pawn, and clicking it will perform the capture (right).

Text output to the console notifies the user when their king is in check, an illegal move that keeps or puts their king in check is attempted, or any terminal position of a game has been attained, such as stalemate and checkmate. If the user gets a pawn to Rank 8 the console will ask the user to select a piece type for promotion. To terminate the program, the user can select Exit from the File menu.

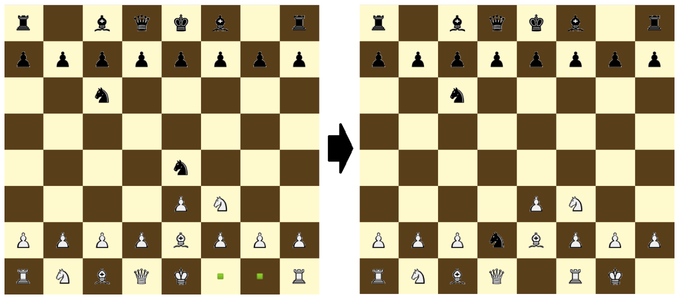


Fig. 17. Example of a castling move using the GUI. After the user selects their king its legal moves become highlighted. In this case, kingside castling rights are available (left). Clicking the castling square moves the king and rook into the castle position (right).

Acknowledgment

Alex Van de Kleut, for general inspiration and high-level discussion of bitboard and programming concepts.

Amir Afghani, for tutorial on Java Swing used for GUI and piece icons.

chessprogramming.wikispaces.com, for great general chess programming resource/reference and figures.

Wikipedia, for chess resource/reference, figures, and pseudocode.