Xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Algorithm for chess Engine

Step1: Start

Step2: Represent Chessboard using array of integers.

Step3: Define variable for each piece; eg wP for white Pawn.

Step4: Declare array to store Rank and File of each Square on Chessboard.

Step5: Define counter to store number of moves played.

Step6: Declare an Integer to store 64bit Position key for each piece.

Step7: Declare integer to store piece number and score.

Step8: Generate BitBoards to store information of position of all pieces on chessboard.

Step9: Declare array to store position-history of each piece.

Step10: Declare a piece-list array to store which pieces are present on board.

Step11: Declare a function to check whether castling is possible.

Step12: Check which squares are under attack by opponent.

Step13: Declare Move Format to store the information of “ from, to, captured ,Promoted ” square on chessboard.

Step14: Define function to print “move and information” about squares from step

Step15: Segregate Slider and non-Slider moves.

Step16: Generate all the possible moves.

Step17: Get score for each move using depth analysis, Min-max Algorithm and Alpha-Beta Pruning.

Step18: Select best move according to the best score obtained from step

Step19: Get the “from , to ,captured , promoted” square values from step according to the best move selected from step

Step20: Store the current position in position-history array.

Step21: Move the piece from current position(from square) to new position(to square).

Step22: Check if capture was made , then remove that piece from piece-List array.

Step23: Check for promotions.

Step24: Check for castling permission.

Step25: For all pieces added, moved, removed update all position counters and piece-Lists.

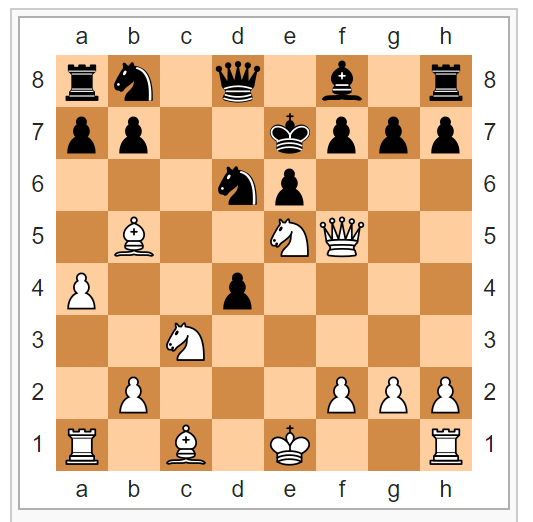
Step26: Stop

Xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

ADD IN APPENDIX

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Chess Diagram:REPRESENTATION OF CHESSBOARD USING RANK AND FILE.



A **chess diagram** is graphic representation and stylized of a specific moment in a [chess](https://en.wikipedia.org/wiki/Chess) game, showing the different positions occupied by chess pieces in given time during game development.

This graphical representation is done through symbols previously agreed for this purpose in order to facilitate reading the diagram.

The chess diagrams are a resource widely used not only in teaching game in the courses offered to people just getting into it; but they are also used in subsequent analysis, that amateurs and professionals players at this discipline can make of games played, especially in championship games.

Xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Bitboards

A **bitboard** is a [data structure](https://en.wikipedia.org/wiki/Data_structure) commonly used in [computer systems that play](https://en.wikipedia.org/wiki/Game_AI) [board games](https://en.wikipedia.org/wiki/Board_game).

They help the programs analyze chess positions with few CPU instructions and hold a massive number of positions in memory efficiently.

Bitboards allow the computer to answer some questions about game state with one logical operation. For example, if a chess program wants to know if the white player has any pawns in the center of the board (center four squares) it can just compare a bitboard for the player's pawns with one for the center of the board using a logical AND operation. If there are no center pawns then the result will be zero.

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

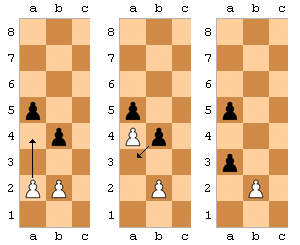
Castling :

**Castling** is a move in the game of [chess](https://en.wikipedia.org/wiki/Chess) involving a player's [king](https://en.wikipedia.org/wiki/King_(chess)) and either of the player's original [rooks](https://en.wikipedia.org/wiki/Rook_(chess)). It is the only move in chess in which a player moves two pieces in the same move, and it is the only move aside from the [knight](https://en.wikipedia.org/wiki/Knight_(chess))'s move where a piece can be said to "jump over" another.

Castling consists of moving the king two squares towards a rook on the player's first [rank](https://en.wikipedia.org/wiki/Rank_(chess)), then moving the rook to the square over which the king crossed. Castling may only be done if the king has never moved, the rook involved has never moved, the squares between the king and the rook involved are unoccupied, the king is not in [check](https://en.wikipedia.org/wiki/Check_(chess)), and the king does not cross over or end on a square in which it would be in check.

Xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

En Passant

[](https://en.wikipedia.org/wiki/File:Ajedrez_captura_al_paso_del_peon.png)

***En passant*** (from [French](https://en.wikipedia.org/wiki/French_language): *in passing*) is a move in [chess](https://en.wikipedia.org/wiki/Chess). It is a special [pawn](https://en.wikipedia.org/wiki/Pawn_(chess)) [capture](https://en.wikipedia.org/wiki/Glossary_of_chess#Capture), that can only occur immediately after a pawn moves two ranks forward from its starting position and an enemy pawn could have captured it had the pawn moved only one square forward. Note that the capturing pawn must be on its fifth rank prior to executing this maneuver. The opponent captures the just-moved pawn "as it passes" through the first square. The resulting position is the same as if the pawn had moved only one square forward and the enemy pawn had captured it normally.

The *en passant* capture must be made at the very next turn or the right to do so is lost.[[2]](https://en.wikipedia.org/wiki/En_passant#cite_note-2) It is the only occasion in chess in which a piece is captured but is not replaced on its square by the capturing piece. Like any other move, if an *en passant* capture is the only legal move available, it must be made. *En passant* capture is a common theme in [chess compositions](https://en.wikipedia.org/wiki/Chess_composition).

Xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

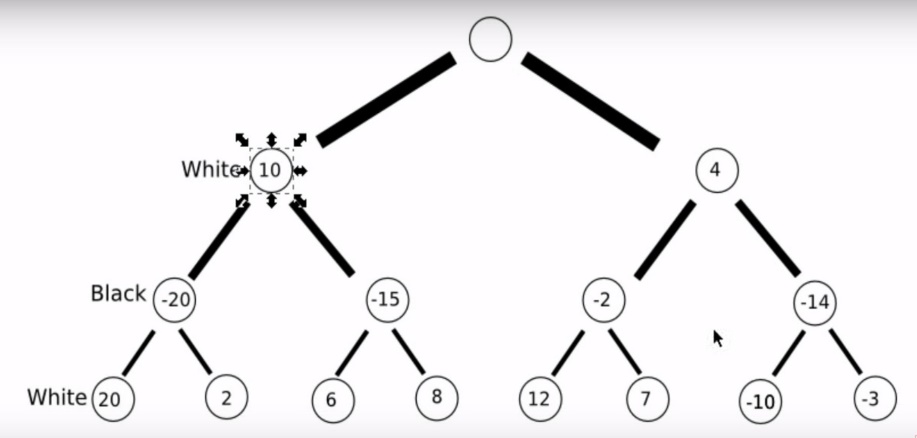
Depth Analysis:

The most important piece of information in a game of chess is the “Depth” value. Quite simply, this tells you how far ahead the engine is looking. It’s crucial to understand that the depth value is given in **plies**. A “ply” is a half move – in other words, a move for one player. Eg: Black is to move in the current position (and it would be Black’s 19th move); while the engine is evaluating Black’s moves in this position, it is doing a **one-ply search**. When it finishes (and records the best move for Black) it starts analyzing and evaluating all of White’s replies to each of Black’s possible 19thmoves. So as it is analyzing White’s possibilities at move 20, it’s doing a **two-ply search** (and vice would display a “Depth” value of “2”). When vice finishes that ply and begins evaluating all of Black’s move 20 replies to White’s moves, it’s performing a **three-ply search** (and the “Depth” value would be “3”).

Reference link: https://uscfsales.wordpress.com/2011/07/08/understanding-chess-engine-evaluations/

Xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Minimax algorithm



A **minimax algorithm**[[4]](https://en.wikipedia.org/wiki/Minimax" \l "cite_note-4) is a recursive [algorithm](https://en.wikipedia.org/wiki/Algorithm) for choosing the next move in an n-player [game](https://en.wikipedia.org/wiki/Game_theory), usually a two-player game. A value is associated with each position or state of the game. This value is computed by means of a [position evaluation function](https://en.wikipedia.org/wiki/Evaluation_function) and it indicates how good it would be for a player to reach that position. The player then makes the move that maximizes the minimum value of the position resulting from the opponent's possible following moves. If it is **A**'s turn to move, **A** gives a value to each of his legal moves.

A possible allocation method consists in assigning a certain win for **A** as +1 and for **B** as −1. This leads to [combinatorial game theory](https://en.wikipedia.org/wiki/Combinatorial_game_theory) as developed by [John Horton Conway](https://en.wikipedia.org/wiki/John_Horton_Conway). An alternative is using a rule that if the result of a move is an immediate win for **A** it is assigned positive infinity and, if it is an immediate win for **B**, negative infinity. The value to **A** of any other move is the minimum of the values resulting from each of **B**'s possible replies. For this reason, **A** is called the*maximizing player* and **B** is called the *minimizing player*, hence the name *minimax algorithm*. The above algorithm will assign a value of positive or negative infinity to any position since the value of every position will be the value of some final winning or losing position. Often this is generally only possible at the very end of complicated games such as [chess](https://en.wikipedia.org/wiki/Chess) or [go](https://en.wikipedia.org/wiki/Go_(board_game)), since it is not computationally feasible to look ahead as far as the completion of the game, except towards the end, and instead positions are given finite values as estimates of the degree of belief that they will lead to a win for one player or another.

Xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Alpha-Beta Pruning:

**Alpha–beta pruning** is a [search algorithm](https://en.wikipedia.org/wiki/Search_algorithm) that seeks to decrease the number of nodes that are evaluated by the [minimax algorithm](https://en.wikipedia.org/wiki/Minimax" \l "Minimax_algorithm_with_alternate_moves" \o "Minimax) in its [search tree](https://en.wikipedia.org/wiki/Game_tree). It is an adversarial search algorithm used commonly for machine playing of two-player games like chess. It stops completely evaluating a move when at least one possibility has been found that proves the move to be worse than a previously examined move. Such moves need not be evaluated further. When applied to a standard minimax tree, it returns the same move as minimax would, but prunes away branches that cannot possibly influence the final decision.