**Best-move search algorithm: Minimax**

Minimax is a recursive search algorithm, which uses a function that evaluates the advantage of the players, for a board position (evaluate function explained in more detail further down).

The basic algorithm checks all the possible board states up to a given depth. Starting from the last depth, the algorithm selects either the maximum or the minimum evaluation, depending on which player's turn it is, as one of the players is looking to maximize the evaluation and the other to minimize it. After selecting either the minimum or maximum evaluation, the evaluation is passed up to the lesser depth of recursion and so on.

After the end of recursion, the algorithm returns the board state which leads to the minimum/maximum evaluation (depending which player's turn it is).

In other words, the algorithm selects the best move based upon the best possible next moves that can be made by the opponent and the player himself.

**Evaluate function and Negamax**

The evaluate function equals to 0 if no player has an advantage.

It evaluates to a positive value if the first player has an advantage, and the higher the advantage, the higher is the value. The board game is considered as a zero sum game - where one's player advantage is another's disadvantage, and so if the second player has an advantage, function's value is negative.

For player A and B:

Advantage A = - Advantage B

This identity is the basis for the algorithm called **Negamax**,which is identical in its functionality to minimax, but instead of either selecting the maximum or the minimum score, it negates the score for the minimizing player and always finds the maximum score (negative maximum score = minimum score).

**Alpha-beta pruning**

Alpha-beta pruning is an improvement that can be made to the minimax algorithm. Alpha is the lower bound for evaluations of board states that will be considered further, beta is the upper bound.

If the evaluation of the board is too low (another move was recursively found to give better evaluation) then it is known that the first player won't consider this and further moves. Same goes for the second player, if another move was found to give lower evaluation (better for the second player), then he won't consider moves that lead to higher evaluation, and the next moves to each one of the discarded moves.

Using alpha-beta pruning effectively can reduce the complexity of minimax by a square root, doubling the possible search depth.

**Pseudo-code for negamax with alpha-beta pruning**

**MinimaxResult negamax(state, depth, alpha, beta, player)**

**if** depth = 0 or state.getNextStates() = NULL:

evaluateValue = evaluate(state)

**if** player = BLACK:

evaluateValue \*= -1

return evaluateValue

bestValue = −INFINITY

bestState = NULL

**foreach** nextState of state:

result = negamax(nextState, depth − 1, -1 \* beta, -1 \* alpha, getNextPlayer(player))

value = -1 \* result.getScore()

**if** value > bestValue:

bestValue = value

bestState = nextState

alpha = max(alpha, score)

**if** alpha >= beta: // Stops checking next states

**break**

**return** new MinimaxResult(bestValue, bestState)

The initial call to the function will be:

negamax(state, depth, -INFINITY, +INFINITY, player)