Search Algorithms:

Allows the program to look ahead at possible future positions before deciding what moves it wants to make in the current position.

|  |  |
| --- | --- |
| Minimax | Static evaluation: Trying to estimate how good the position is for one side without making any more moves.  E.g. Add up the values of the remaining white pieces and subtract of that the values of all the remaining black pieces.   * Negative evaluation = Black advantage. * Positive evaluation = White advantage.   White is trying to maximize the evaluation. Black is trying to minimize it. |
| Alpha-beta pruning | White / Black already has a better options available to him.   * Don’t go down that branch. * Don’t process unnecessary nodes. Prune positions.   The result of the search is exactly the same as before. We just saved some time by not considering positions when they can’t affect the outcome.  Pruning isn’t guaranteed to occur. It depends on what order the moves are in. |
| Null Move Heuristic | Enhance the speed of the Alpha-beta pruning algorithm. Guess cut-offs with less effort than would otherwise be required, whilst retaining a reasonable level of accuracy. It lets the opposing player play two moves in sequence, and computes the score after that. Any move made by the current player should beat a score obtainable by the opponent getting two chances to move. |
| Quiescence Searching | Search algorithms that stop at a fixed depth sometimes cause problems. The evaluation function can give unreliable answers. What might seem like a great move in a current state, ends up being a tactical decision by the opponent.   * White queen takes black knight. *White thinks he’s winning.* * Black pawn takes white queen. *Black is winning.* |
|  | One name for this problem is the horizon effect. Search algorithm goes for moves that seem good within the horizon but soon turn bad.  Possible solution: Smart evaluation function that understands tactical actions and predict their outcome. Tactics are hard and difficult to implement.  A quiescence search is an additional search, starting at the leaf nodes of the main search (e.g. Minimax), which tries to solve this problem.  E.g. a quiescence search for all capture moves, so that tactical exchanges don’t mess up the evaluation.  Basic principle: quiescence searches should include any move which may destabilize the evaluation function. |
| Static evaluation function | The function looks only at the current position and does not explore possible moves. Possible things to evaluate:   * Remaining chess pieces. * Position of the pieces: Increase of decrease the evaluation based on the location of the piece. E.g. a knight on the centre of the board is better than a knight on the edge of the board. * Piece mobility: How many different spaces can the given piece move to? * Piece protects: How many of your own pieces are protecting the given piece to prevent it from being captured without repercussion?   E.g. Large integers to represent piece possession, and small ones to represent the other features. |
| Chess Database | * Pre-processed position evaluation. * Hundreds of famous games from top players.   Databases are mainly used for openings and endings.  Choose positions that will provoke a type of game that it often wins.  Endgame database: Tree will become smaller when there are less pieces. E.g. database that computes every single position with 5 pieces or less on board. |
| Other options | * Monte Carlo Tree Search: Often used for games with a large number of potential actions to be taken at each state.   A game like Go will use Monte Carlo: branching factor of Est 300.  A game like Chess will use Minimax: branching factor of Est 30.   * Negamax is a simplified version of Minimax. It will get rid of the two subroutines maximizingPlayer and minimizingPlayer. It passes on the negated score due to the following mathematical relation:   Max(a, b) == -Min(-a, -b). |
| Links | Minimax & Alpha-beta pruning: <https://www.youtube.com/watch?v=l-hh51ncgDI&t=319s>  Chess algorithms: <https://www.cs.cornell.edu/boom/2004sp/ProjectArch/Chess/algorithms.html>  Quiescence Searching: <http://satirist.org/learn-game/methods/search/quiesce.html> |