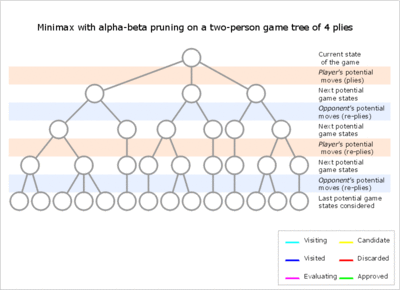
What is Alpha Beta Pruning? Explain with example.

**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#Minimax_algorithm_with_alternate_moves)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 ([Tic-tac-toe](https://en.wikipedia.org/wiki/Tic-tac-toe), [Chess](https://en.wikipedia.org/wiki/Chess" \o "Chess),[Go](https://en.wikipedia.org/wiki/Go_(board_game)" \o "Go (board game)), etc.). 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

The benefit of alpha–beta pruning lies in the fact that branches of the search tree can be eliminated. This way, the search time can be limited to the 'more promising' subtree, and a deeper search can be performed in the same time. Like its predecessor, it belongs to the [branch and bound](https://en.wikipedia.org/wiki/Branch_and_bound) class of algorithms. The optimization reduces the effective depth to slightly more than half that of simple minimax if the nodes are evaluated in an optimal or near optimal order (best choice for side on move ordered first at each node).

With an (average or constant) [branching factor](https://en.wikipedia.org/wiki/Branching_factor) of *b*, and a search depth of *d* [plies](https://en.wikipedia.org/wiki/Ply_(game_theory)), the maximum number of leaf node positions evaluated (when the move ordering is [pessimal](https://en.wiktionary.org/wiki/pessimal" \o "wiktionary:pessimal)) is [*O*](https://en.wikipedia.org/wiki/Big_O_notation)(*b*\**b*\*...\**b*) = *O*(*bd*) – the same as a simple minimax search. If the move ordering for the search is optimal (meaning the best moves are always searched first), the number of leaf node positions evaluated is about *O*(*b*\*1\**b*\*1\*...\**b*) for odd depth and*O*(*b*\*1\**b*\*1\*...\*1) for even depth, or {\displaystyle O(b^{d/2})=O({\sqrt {b^{d}}})}. In the latter case, where the ply of a search is even, the effective branching factor is reduced to its [square root](https://en.wikipedia.org/wiki/Square_root), or, equivalently, the search can go twice as deep with the same amount of computation.[[10]](https://en.wikipedia.org/wiki/Alpha%E2%80%93beta_pruning#cite_note-RN03-10) The explanation of *b*\*1\**b*\*1\*... is that all the first player's moves must be studied to find the best one, but for each, only the best second player's move is needed to refute all but the first (and best) first player move—alpha–beta ensures no other second player moves need be considered. When nodes are ordered at random, the average number of nodes evaluated is roughly {\displaystyle O(b^{3d/4})}

[](https://en.wikipedia.org/wiki/File:Minmaxab.gif)

An animated pedagogical example that attempts to be human-friendly by substituting initial infinite (or arbitrarily large) values for emptiness and by avoiding using the [negamax](https://en.wikipedia.org/wiki/Negamax" \o "Negamax) coding simplifications.

Normally during alpha–beta, the subtrees are temporarily dominated by either a first player advantage (when many first player moves are good, and at each search depth the first move checked by the first player is adequate, but all second player responses are required to try to find a refutation), or vice versa. This advantage can switch sides many times during the search if the move ordering is incorrect, each time leading to inefficiency. As the number of positions searched decreases exponentially each move nearer the current position, it is worth spending considerable effort on sorting early moves. An improved sort at any depth will exponentially reduce the total number of positions searched, but sorting all positions at depths near the root node is relatively cheap as there are so few of them. In practice, the move ordering is often determined by the results of earlier, smaller searches, such as through [iterative deepening](https://en.wikipedia.org/wiki/Iterative_deepening_depth-first_search).

The algorithm maintains two values, alpha and beta, which represent the maximum score that the maximizing player is assured of and the minimum score that the minimizing player is assured of respectively. Initially alpha is negative infinity and beta is positive infinity, i.e. both players start with their lowest possible score. It can happen that when choosing a certain branch of a certain node the minimum score that the minimizing player is assured of becomes less than the maximum score that the maximizing player is assured of (beta <= alpha). If this is the case, the parent node should not choose this node, because it will make the score for the parent node worse. Therefore, the other branches of the node do not have to be explored.

Additionally, this algorithm can be trivially modified to return an entire [principal variation](https://en.wikipedia.org/wiki/Principal_variation) in addition to the score. Some more aggressive algorithms such as [MTD(f)](https://en.wikipedia.org/wiki/MTD(f)) do not easily permit such a modification.