**Something about AI in program**

**Just a demo for this game**

**(C++ implementation version)**

Chess programs use the "search" function to find the way. The search function gets the game information and then looks for the best move for the program side.

First, the minimum-maximum search Minimax Search

First: the minimum and maximum are relative, and only for one party, AI is beneficial to AI

The smallest and largest search in chess AI: In simple terms, the AI ​​is gone, and the best (maximum) move for AI in this process is the worst (smallest) move for me.

And this move is the best way to go for the AI ​​we are looking for.

This process is the same as guessing the opponent's move and playing chess when you play chess with others. However, the computer can think a few more steps. The number of steps here is the following search depth.

For example: suppose the search depth is 4. Then AI takes a step (he thinks that the best, count as step 1, search depth 4), will first consider if he walks this step 1, then I will definitely go relative to this One step

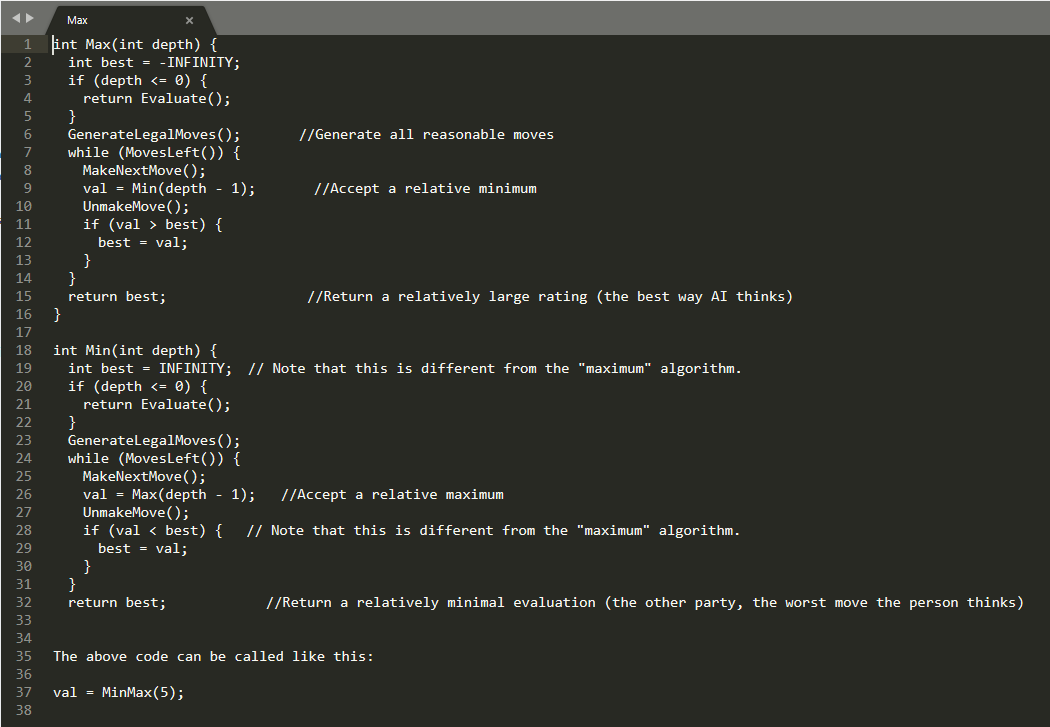
The worst step 2 (search depth 3), then AI again assumes the best number of steps 3 (search depth 2) according to step 2, continue to consider the worst step 4 I walk according to step 3 (search Depth 1)

Next, the search depth is 0, giving the situation evaluation function currently.

They are recursively calling each other, so this search idea is relative

Here is just a brief introduction to the principle of minimum and maximum search, and can not achieve specific AI functions.

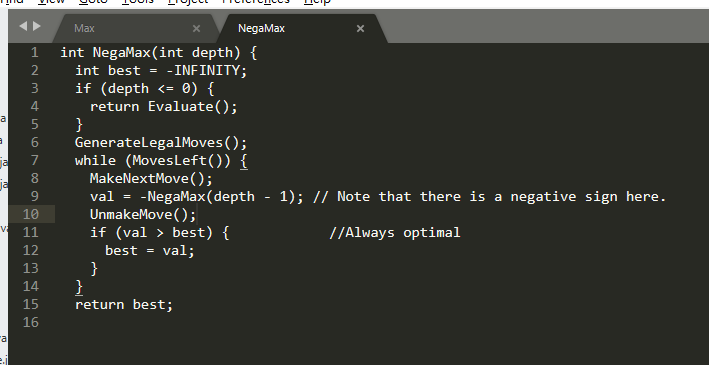
First a MAX algorithm,



This returns the evaluation of the current situation, which is the result of looking forward 5 steps. Related explanations, just read the comments.

The above algorithm code is long, and it is only for one party to speculate (that is, the best and worst way for one party), the following will introduce an optimized algorithm.

**Negamax Search**



It can be seen from this function that this function is always the optimal value of the current node (that is, always find the best way to move the current node), but only when transforming the node (that is, when transforming from AI to human), the result of the function Take a negative value and become the best way for people to evaluate AI. This eliminates the need to find the min function and reduces the amount of code.

AS FOR **Alpha**-Beta search

The minimum and maximum runtimes are to check the entire game tree and then choose the best route possible, but because the branching factor is too large, the efficiency is very low and deep search is not possible.

The advantage of Alpha-Beta search is that it cuts unnecessary branching factors.

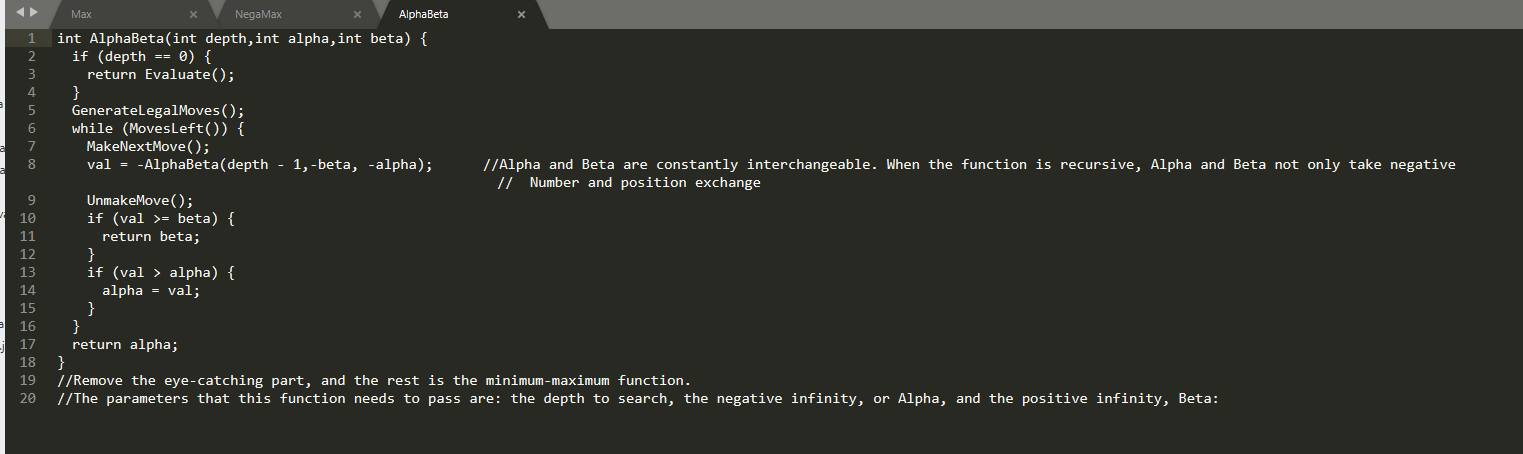
**Alpha pruning.**

For a MIN node (the second layer), if the upper bound Beta (170 and 50) of the back-calculation value can be estimated, and the Beta value is not greater than the estimated back-calculation value of the parent node (MAX node) of the MIN If Alpha (200) is defined, that is, Alpha ≥ Beta, then it is no longer necessary to extend the remaining sub-nodes of the MIN node (children of x), because the evaluation of these nodes has no reverse value for the MIN parent node. Influenced, this process is called Alpha pruning.

**Of course there are also beta pruning:**

For a MAX node, if the lower bound Alpha of the back-calculation value can be estimated, and the Alpha value is not less than the upper bound of the estimated back-calculation value of the parent node (MIN node) of MAX, that is, Alpha ≥ Beta, then It is no longer necessary to extend the remaining child nodes of the MAX node because the evaluation of these nodes has no effect on the backstepping value of the MAX parent node. This process is called Beta pruning.

The Alpha value of a MAX node is equal to the current maximum final reversal value of its successor node. The Beta value of a MIN node is equal to the current minimum final reversal value of its successor node.



Algorithm introduction

Passing two values ​​in the search, the first value is Alpha, which is the best value found, expressed in if (val > alpha) {alpha = val;}

The second value is beta, which is the worst value for the opponent. If the result of a move is greater than or equal to Beta, the entire node is invalidated.

Reflected in: if (val >= beta) {return beta;}

**Short of algorithm**

This algorithm relies heavily on the order in which the law is searched. If you always search for the worst move first, then the Beta truncation will not happen, so the algorithm is as low-maximum and very inefficient. The algorithm will eventually find the entire game tree, just like the min-max algorithm.

So, after generating all the moves, sorting is very important~~~

TOO late to start the AI bot,so only make sure the AI algorithm is correct.

Did not concretely implement