Ceng461 HW1 Report

Ali Görkem Yalçın

November 17, 2019

Brief Explanation

The algorithm I created evaluates the board with an evaluation function and depending on the result, plays the best move for the white side, which is the ai. It can "think ahead" for any depth but after the 5th depth the wait times become problematic.
Evaluation function uses different sub functions such as, piece values, piece values depending on their location, rook on file. In this document I will explain the functions I have created and the minimax algorithm I used.

Functions

0.1 calculate piece value(piece type):

This function takes an integer and returns the piece's piece value. In order to define the piece values I used many different sources where they calculated each piece value. At the end the piece values I used were:
pawnValue = 100
knightValue = 330
bishopValue = 340
rookValue = 530
queenValue = 970
kingValue = 10000000000

I made the king's value 10000000000 because in every stage of the game, king is the most valuable piece and if it gets taken or gets threatened, we must see a drastic change in the evaluation function to either checkmate the opponent or to save itself from getting checkmated.

0.2 calculate piece value score(board):

This function calculates the board's total piece value. It iterates over the board's pieces and depending on their color and piece type, either adds or subtracts the piece value from the board's score. I defined that white pieces are the positive values, black pieces are the negative values such that depending on the final calculation we can see which player has the advantage.

0.3 calculate piece value score for game stage(board):

This function is similar to the calculate piece value score, but instead of subtracting the black pieces points from the total sum, it adds it to the total sum. I needed this function to determine if the game is in a "late game" stage. I will talk about the late game in "calculate late game stage" function.

0.4 calculate late game stage(board):

This function calls the "calculate piece value score for game stage(board):" and depending on the result determines if the game is in late game or not. I checked online sources on game stages and most of the result said that when all the pieces' values on the board sums up to less than 26 pawns (2600 for my definition), game is in the late game stage.

0.5 generic piece square value calculator

I found for each piece, a piece square value table where depending on their place on the board, certain pieces have bonus or less points for them. I created 7 global variables namely "pawn piece square table, knight piece square table..." and those variables show us whether they are advantageous at certain places or not. For example a knight is disadvantageous in corners because it can only move to 2 places from the corners, but in the center of the board, it gains more points because now it can move up to 8 different places. The piece square value calculations are calculated in similar manner. But in the python-chess module, when we iterate over the board's pieces we see that indexing is done like this:
63 62 61 60 59 58 57 56
55 54 53 52 51 50 49 48
47 46 45 44 43 42 41 40
39 38 37 36 35 34 33 32
31 30 29 28 27 26 25 24
23 22 21 20 19 18 17 16
15 14 13 12 11 10 9 8
7 6 5 4 3 2 1 0

But in python indexing starts from 0. So I needed to make an arrangement when calculating the piece square values. This function gets the index on the board then finds the corresponding index for the lists and returns the piece square value.

0.6 calculate piece value score(board):

In a for loop, calculates each pieces location points. For black pieces again it subtracts and for white pieces it adds up.

0.7 find black - white pawns(board):

Finds the indexes of black and white pawns, returns a list.

0.8 isolated white - black pawns(board):

Isolated pawns are the pawns where they do not have a pawn in an adjacent file(column). This function finds the number of isolated black - white pawns in the board.

0.9 calculate isolated pawn value(board):

In the sources I read online, isolated pawns are a vulnerability to the player and 3 sources weighed the isolated pawn value as -5 -5 and -6. So I defined the isolated pawn value as -5 and calculated the board's isolated pawn value.

0.10 check attacking king(board):

This function checks if in the board there are pieces that attack the king piece. It checks for both sides and depending on their color either adds or subtracts from the result value.

0.11 check checkmate(board):

This function first checks whose turn it is. If its the ai's turn, it checks if ai is in a checkmate situation and if so it subtracts a very large number from the board's value, vice versa otherwise. Since this number is bigger than any other number, decision will be made on this functions result.

0.12 calculate mobility value(board):

I read that the amount of moves a player can make is an advantageous thing for the player, I gave 5 centipawns to the players for each legal move they can make.

0.13 find black - white rooks(board):

Finds the indexes of black and white rooks. Returns a list.

0.14 calculate white - black rook on open file value(board):

Rooks on open files(columns) are a great advantage for the players, since they can invade enemy base from there. I weighed rook on open field value as 15 centipawns. These functions calculate the rook on open field values.

0.15 calculate move first(board):

This is the main function that starts the chain process of calculating each legal move for their points and returning the best possible move from them. It first finds all the legal moves. Depending on whose turn it is, for each move it pushes the move to the board, calls the evaluation function, returns the value, if that value is the maximum(or minimum if ai has the black pieces) of all the values that it looped over already, it assigns the currently iterated move as the best move. After it is done looping over moves it pops the move since board should be left as it is found in the first place, returns the best move.

0.16 calculate deep move(board):

This function is a recursive function that depends on the depth value it has been called with. If the depth is 0, it calculates the boards value and returns it. This evaluation function includes:
Piece value score, pieces points for their location score, checking the checkmate, checking if there are any pieces attacking the king, isolated pawns, rooks on open fields and mobility score.
If the depth is not 0, for each legal move in the leaves(since in the main function, we already pushed one move) it checks whose turn is it with the "is maximizing" parameter, I send board.turn since it will calculate depending on the turn. Depending on the turn value program will either try to maximize or minimize the score. In the calculation for ai, if there are no available legal moves(meaning that it is a checkmate) I assigned the best move value to the checkmate functions return value so that it will be a very small number(-999999999) if its not a checkmate, I assign the best move value to the -50000 since it is small enough to be converted into a higher value in any evaluation function call. Then I pop this move to the board, call the recursive function with the "depth - 1" and "not is maximizing" because we lower the depth value by 1 by pushing and now we want to calculate for the other player who wants to minimize the value. Then depending on the result and the best move value, I assign the best move value to the maximum of that two values since ai will want to maximize the value(In players calculation, I assign the best move value for the minimum of those two values since its player's best move value) then pop the move the then return the best move value. In the outer function - calculate move first - we assign the "move value" to the this "best move value" and since we were already in a loop of legal moves in the outer function, we do not need to keep track of the moves.

1 Additional Notes

Changing the second parameter in the "ai play" functions return statement will define the depth of the program.