Grader Description:

Same task as Othello 4, with an additional stipulation: If the number of open positions is LESS THAN some N (where N is 11 for now), then the script is to run Negamax (or Minimax) to determine the actual best move, as opposed to a rule-of-thumb best move.

Details: you should keep what you have done for Othello 4. However, in the case where the number of open positions is less than N, then you will run an additional piece of code. This code will run Negamax (or Minimax) and determine a guaranteed best score (assuming optimal play by both sides), and a move sequence which will guarantee it. You should output this minimal guaranteed minimum score first (possibly negative), followed by a sequence of moves IN REVERSE which will lead to the score. The score is from the point of view of the token given (rather than the number of x tokens minus the number of o tokens). The reason for the reverse sequence is that this way, the final move is what will be picked up by the moderator as the script's best move. The sequence of moves includes both your script's moves and the moves of the opponent. Negative numbers will be ignored, so passes may be indicated by a -1.

Scoring is .5 for a (there may be more than one) correct Next move (ie. the final integer output), .25 for a correct Minimum score, and .25 for a correct Move sequence. Note that just because a move sequence leads to the correct score does not mean that the move sequence is correct (see below for details).

The grader will first test the submitted script on 10 fixed boards, where the number of free positions increases from 1 to 10. Then, the grader will take 10 random board positions with one free spot, then 10 with 2 free spots, and so on until 100 tests have been run. The 10th test is the only one where there are 10 free spots left on the Othello board. The grader will deliver a board and a token to your script, where there is at least one move by that token. The grader will take the most recent line of the script's output which has the word "score" and at least one number (if it does not find such a line, it will assume the script has timed out - see below for what the grader will then do). The first number on this line is the minimum score that the given token is guaranteed to achieve under perfect play. The remaining numbers, in reverse order, constitute a sequence of moves that will achieve this score. If there is a script or syntax or script error, or the final number the grader sees is not a valid move, then the grading script will summarily terminate.

If the script being tested does put the word score on one line, along with at least one number, then the grader script will find the last number in the output and take that as its move. In prior testing, the likelihood of optimal rule-of-thumb moves near the end of the game is around 50% (ie. the midgame rules of thumb are not so good in the endgame). On the plus side, the testing is faster :)

Eric’s Notes:

This is probably the hardest lab so far (luckily Gabor has made his own pdf’s for these ones)

Conceptually, Negamax is not too difficult an algorithm to understand. There are winning positions and losing positions (and sometimes draw), and you want the opponent to always have a losing position (so you win). Negamax plays a move and then recurs on each new board possible with the enemy player. Obviously, you want the opponent to always have the most losing position, so you can win!

But how do you know that it’s winning or losing? You start at the base case. For othello, it’s when no players have moves left. This happens when there are no legal moves, or the board is full. The final score is your tokens minus the enemy tokens.

Now, we take a step back to the move before the final state. There’s only one possible move, so that possible move decides the score, and that position is assigned that score.

Take a few more steps back, and then you have several moves to choose from, and all of those new positions are figured out to be losing or winning. You want to choose the move that is the most losing for the opponent.

Eventually, you take enough steps back that you reach the initial state, which for this lab is 11 holes left in the board.

This was probably confusing, so you should take a look at Gabor’s notes!

The main speedups for this lab doesn’t come with the actual algorithm, but with the workhorse functions, specifically possibleMoves() and makeMove(). (see my notes for Othello 1)