CONNECT-K FINAL REPORT

Partner Names and ID Numbers: Pablo Kang (58842064)

Team Name: Draken

1. My evaluation function goes through all the possible k-length lines on the board (horizontals, verticals, etc.) and counts the number of player and enemy pieces present in each line. It then scores the line based on these counts before moving on to the next line and adds the score to either a player or enemy score variable, depending on whose pieces were in the line. Once all lines of every variety have been scored, eval(s) returns the difference between the player’s score and the enemy’s score for the entire board state. It may not be a very elegant or complex means of evaluating the board, but it does a good enough job to beat the AverageAI now and then.
2. Alpha-Beta pruning is implemented in a manner similar to that shown in our lecture slides. Pruning occurs by essentially interrupting minMove(s) or maxMove(s) as they try to evaluate the child board states formed by trying all possible moves in a board state if the current board state if alpha is greater-than or equal to beta. Beta, in this case, is the best score returned from calling either minMove(s) or maxMove(s) on a new state formed by trying out a move from the ArrayList or PriorityQueue of available moves. If this beta is lower than alpha, then all other possible moves in the current state are not worth attempting, and they are pruned. Oddly enough, my DrakenAI still managed to beat AverageAI with a score of 2-0 in the tournament with pruning turned off, so it’s possible that the performance difference is minor or requires much more testing to observe.
3. The implemented Iterative Deepening Search is housed in a for-loop located in getMove(s) and increases its depthLimit variable, starting at 0, until the move deadline approaches. As this for-loop increments the depthLimit variable with each pass, the minMove(s) function it calls is allowed to go to an incrementally deeper depth as it searches down through possible player-enemy turns. There was a bit of difficulty restructuring my older minimax(s) algorithm into two, separate algorithms (minMove and maxMove) in order to work out the depth-searching, but with enough time and coding, I managed to get it done.
4. My minMove(s) and maxMove(s) recursive functions (though they technically call their partner, not themselves) both store the current best move’s score that they have found so far, replacing it as they go along with better moves. As soon as the time limit draws near, they quickly evaluate the board they were looking at and return this score. If it surpasses the best move score of the previous depth, it becomes the new best move score. As each depth returns the best move score they found (which are compared with the best move scores of the depths above), the true best move score surfaces in the minMove(s) call within getMove(s).
5. My quiescence function determines if the current line being analyzed has enemy pieces counting k-1 in number. If so, the function returns true, determining the line to be of interest, as the enemy may be setting a trap. The function has no effect on my DrakenAI, as I have not yet implemented a means of acting upon a quiescent position.
6. I have no suggestions for this project. I thoroughly enjoyed it, despite the difficulty brought on by working alone. I look forward to CS 175 “Projects in AI”.