CONNECT-K FINAL REPORT ***[TEMPLATE --- do not exceed two pages total]***

Partner Names and ID Numbers: Hongkun Chen (62214504), Misook Sohn (77472789)

Team Name: ChenSohn

Note: this assumes you used minimax search; if your submission uses something else (MCTS, etc.), please still answer these questions for the earlier versions of your code that did do minimax, and additionally see Q6.

1. Describe your heuristic evaluation function, Eval(S). This is where the most "smarts" comes into your AI, so describe this function in more detail than other sections. Did you use a weighted sum of board features? If so, what features? How did you set the weights? Did you simply write a block of code to make a good guess? If so, what did it do? Did you try other heuristics, and how did you decide which to use? Please use a half a page of text or more for your answer to this question.

Our heuristic evaluation function examines all possible combinations of k fields in a row horizontally, vertically, and diagonally. First, we break it down to lines horizontally, vertically and diagonally and evaluate pieces in each line. For example, if the board is 8 x 9 and k is 5, then it would lead to the following possible winning combinations in a horizontal row.
01234567
(0,0)(1,0)(2,0)(3,0)(4,0) (1,0)(2,0)(3,0)(4,0)(5,0) (2,0)(3,0)(4,0)(5,0)(6,0) (3,0)(4,0)(5,0)(6,0)(7,0)

Since our evaluation function examines the each of combination taken by each player, it adds each player's score depending on the number of pieces taken in a row to evaluate how close it is to be k in a row. We gave different weights depending on number of occupied pieces in each block of k. We set AI as MAX player and human as MIN player. So, if AI occupies the fields, then we gave +5 points to AI. If human occupies the fields, then we gave -5 points to human. If the field is empty (nobody occupies the field), then we give points to both player, +3 to AI and -3 to human. And then it counts how many pieces are taken by AI or human, and multiply the number of taken pieces to each player's score (If AI occupied 2 pieces, and 2 pieces are empty => 16*2). Finally, if the state is resulted in k in a row, then it would get a big bonus point +1000 for the AI and -3000 for the opponent. The reason why the opponent gest more point is that we can not search very deep, sometimes the AI will not notice that the opponent already going to win in several steps, so we give opponent bigger bonus if it has k in row, -3000, to make sure the AI will prevent opponent from wining the game.

2. Describe how you implemented Alpha-Beta pruning. Please evaluate & discuss how much it helped you, if any; you should be able to turn it off easily (e.g., by commenting out the shortcut returns when alpha >= beta in your recursion functions).

We write a new wrap function called alpha_beta_pruning which will call min_search and max_search functions. In both min_search and max_search functions, we added two more parameters called alpha and beta. These two parameters will be passed to their childen to do the pruning and the children will return their node value to help them update the value of alpha and beta. Furthermore, we compare the value of alpha and beta after each updated best value in min_search and max_search. If beta is less than alpha, the function will just return current node value and will not explore any of its remaining children.

3. Describe how you implemented Iterative Deepening Search (IDS) and time management. Were there any surprises, difficulties, or innovative ideas?

Our IDS runs alpha beta pruning repeatedly with increasing depth limits until it reaches to time limitation. We used 95% of time(deadline) to make sure that it does not over the time limitation. To keep the track of time after execution, we used chrono for time measurement. First, we set the start time before it runs IDS search and get the end time after IDS running to keep track of clock cycle. While the elapsed time(end time- star time) does not reache to time limitation(deadline * 0.95), it runs alpha beta pruning search and increase depth and updated end time again. During alpha beta search, if it overs time limitation, then it terminates the search. If the current depth is same with depth of IDS, it returns best move from search. I think time management and IDS is innovative and necessary because the state board is too large to explore in reasonable time, it's necessary to limit the time to search. Alpha

beta search without IDS, it took too much time to find the optimal path, and it was able to go to a depth of 4. While IDS and time limitation, it was able to go to deeper and deeper and found relatively fast.

4. Describe how you selected the order of children during IDS. Did you remember the values associated with each node in the game tree at the previous IDS depth limit, then sort the children at each node of the current iteration so that the best values for each player are (usually) found first? Did you only remember the best move from a given board? Describe the adat structure you used. Did it help?

In our AI, we just simply remember the best move from previous iteration. We use another global variable called final_best_move to remember the best move from previous iteration. After the 2 iteration, when our AI expands root nodes, it will expand the final_best_move first. We use a queue to implement frontier. So we just push last best move into the queue first to make sure it will be expanded first. IDS do increase the depth of search, but it does not increase much, especially in the beginning of the game.

5. [Optional] Did you try variable depth searches? If so, describe your quiescence test, Quiescence(S). Did it help?

6. [Optional] If you implemented an alternative strategy search method, such as MCTS, please describe what you did, how you implemented it, and how you decided whether to use it or your minimax implementation in the final submission.