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

Partner Names and ID Numbers:Devon Kenneally 78552559

Team Name: DKK

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.

My heuristic function is called "heuristic" . For each piece on the board, the potential for each piece is checked in the directions: right, up, diagonally up right, and diagonally down right. For each of these, if a move state has an inevitable outcome, IE connect 5, and there are 4 pieces with an open space on each end, that move state has a much higher priority than any others besides a winning state. If any move is deemed to be worthless it's given a value of zero, though in retrospect that may have hurt my scoring of the overall board. My AI plays to win.. A winning state has the highest priority.

Worthlessness is determined by having no potential to reach K length, that is there are not enough empty spaces or friendly spaces in front or behind the piece. A tell for this is if the space has what I labeled the "tail" variable, which means the end of the direction the function is testing is either the edge of the board or an enemy player.

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).

With my alpha beta pruning, if ever the alpha is greater than the beta in the max function or vice versa for the min function, the tree terminates there and the function returns. I actually wound up commenting some of the alpha beta code and my code runs much slower without it.

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

As I used Python, the issue with IDS as was with much of the AI was implementing the data structures storing used moves without copying (costly) or needlessly mutating the lists. For a while, my moves strangely would not persist through more than two depths, upon which I discovered that in the second call, gradually my available moves list would shrink. Python's inability to pass by value was a great source of challenge and I would not choose python again if I were to do the project again.

As for time management, I simply have my function continue it's iterative searching until the time is about to run out, at which time I yank whatever the saved best move is. Returning an alpha of negative infinity or a beta of infinity in minV forces alpha beta pruning to trim the trees at the exact point that the return is called regardless how deep I have searched. This would cause a problem for the parent minimax function, returning incorrect moves potentially, etc. however I accounted for this by having my minimax function return before it can get the new moves whenever the time is out and my escape from the min or max has been called.

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 data structure you used. Did it help?
I implemented the sorting for my IDS as I believe was suggested on Piazza, storing the strongest path and trying to play against it. So I stored the best move and it's successors in a python list

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