CONNECT-K FINAL REPORT

Partner Names and ID Numbers: **Yash Girishkumar Botadra (79921782), Sandeep Reddy Madugula (76321572)**

Team Name: **HVD**

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 evaluates the number of ways to win given the board configuration. Our Eval function returns the value [number of ways for our AI to win] minus [number of ways for the human to win]. To calculate the number of ways to win, we are calculating the number of ways to win from each cell in the 4 directions (South East, East, North East, North). To optimize this function, we are ignoring some directions for some cells which are near the North and East boundaries.*

*We tried to give more prominence to a winning way which has more pieces already than a winning way which doesn’t have any pieces or less pieces than the previous one. We wanted to define the weights in increasing order for 0 piece, 1 piece, 2 pieces, 3 pieces winning way (where 2 pieces means 🡪 there are already 2 pieces present in that winning way). But we didn’t come up with any concrete explanation on how we exactly need to define the weights. On top of that our initial evaluation function was performing well and we thought this would add unnecessary complications.*

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 implemented the Alpha-Beta pruning using the pseudo code presented in the lecture. Alpha-Beta pruning improved our AI significantly. We tested our AI with IDS-deadline-minimax algorithm and IDS-deadline-Alpha-Beta algorithm. We found that minimax always loses to Average AI and Alpha-Beta always wins against Average AI. We also ran a ConnectK game between Minimax Vs Alpha-Beta and Alpha-Beta AI had always won against Minimax AI.*

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

*We implemented IDS by iteratively getting the best move for a given max depth. We are setting our deadline as the given deadline minus 500 ms for keeping the buffer time. There were no surprises, it was a straight forward implementation.*

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?

*We implemented IDS sorting using the data stored from the previous IDS search. We are storing a map of gameState vs List of Ordered moves. GameState is a serialized string of 63 characters (for 9 \* 7 board) each character representing an empty piece, or an AI piece or a human piece. We are populating this map for all the game states the previous IDS search went through. In the next IDS search we either fetch the moves from the map (if present) or calculate the available moves (if not present in the map).*

*Despite implementing the sorting, we didn’t include this feature in our final submission. Our AI seems to be performing well without sorting. This is due to the decrease in the performance owing to the added computation required for sorting the moves and storing the map.*

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.