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

Partner Names and ID Numbers: Arash Nabili – 37684183; Navninder Kaur Yadev – 33483708

Team Name: Team Eureka

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 the dot product of a vector of weights with a vector of features? What features? How did you set the weights? Did you simply write a block of code to make a good guess? What heuristic did you use? Please use a half a page of text or more for your answer to this question.

2. Describe how you implemented Alpha-Beta pruning. Since you put it on a switch so that you can turn it on and off, please evaluate how much it helped you, if any.

3. Describe how you implemented Iterative Deepening Search (IDS). Were there any surprises or difficulties?

4. 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? Describe the data structure you used. Did it help?

5. Describe your quiescence test, Quiescence(S). Did it help?

6. Any suggestions for improving this project? (One suggestion is to remove the first-player advantage: the first player initially places one single mark, and then the players alternate each placing two marks per turn. But, this would square your branching factor for each ply. I hope to compensate for this in the tournament by having each pair play both sides in alternation.)

Our heuristic evaluation function was based on the dot product of a vector of weights with a vector of features. The features used in the function were number of possible winning paths for each player, the proximity of each player's pieces to the center of the board, and the presence of a blocked threat row. For the possible winning paths, our evaluation function checked each possible row of k cells horizontally, vertically, and diagonally. If any row contained at least one of player 1's pieces and none of the player 2's pieces, that row was counted as a possible winning path for the player 1. The same applies for player 2 mutatis mutandis. For the proximity of a player's pieces to the center, we used a formula that would return a high value for a piece that was close to the center and a low value for a piece that was far from the center, both horizontally and vertically. For the blocked threat, we checked each row against a series of patterns representing blocked threats. For setting the weights, we decided that a blocked threat took priority over other features, such as possible win paths and proximity of pieces to the center. Therefore, we simply assigned a weight to the blocked threat feature that was much higher relative to the other features. The possible win paths and the proximity to the center features had equal weights.

For Alpha-Beta pruning, we created a separate method that was the same as the minimax method, but with the addition of int variables for keeping track of alpha and beta values for the game tree. At each internal node, after the heuristic value for each of the children was determined and the best heuristic value for the internal node was set, the method would check whether the value of alpha was greater than or equal to the value of beta. If so, the method would immediately exit and go back to the recursive call at the previous level. Comparing to minimax search, we found that Alpha-Beta pruning allowed us to search with an extra depth limit, with the last depth limit fully searched being 4 compared to 3 for the minimax search.

We implemented IDS in the same methods that performed minimax search and Alpha-Beta pruning. Implementing IDS itself was relatively straightforward. At each node, we generated the children of node, and recursively called the search method on each of the children, continuing this process down to the depth limit specified. The greatest difficulty we encountered was combining IDS with minimax and Alpha-Beta pruning. Our solution was to check if each node was a max node or a min node. If a node was a max node, the method would set the node's heuristic value to the highest heuristic value of its children, and if a node was a min node, the method would set its heuristic value to the lowest heuristic value of its children.

We did remember the values for each node found at the previous depth limit, which allowed us to order child nodes in order of best to worst heuristic value. We did so by enclosing each game state in a StateNode object, which also included the state's heuristic value, an array of child nodes, player that moved last, index of child with best value, and indicators for quiescence and timeout. The StateNode class also included a function to add child nodes, without exceeding the array's capacity. Using this data structure helped us keep track of nodes' heuristic values and children.

For the quiescence test, we created a method that would generate the children of a leaf node that wasn't quiescent, and recursively called itself on each of the children, until finding a node that was quiescent. A node's quiescence was determined by the evaluation function. In the minimax and Alpha-Beta pruning methods, the quiescence test was performed on child nodes that weren't quiescent. Quiescence testing helped our AI avoid losing to the opponent when it had the chance.

For future improvements, please provide help with available tools for developing the AI. For example, provide guidance on how to implement honoring of the deadline.