Lesmond Chow 17412911
Colin Ma 71323642
BboysAnonymous

1.

We took a lot of the ideas from the hints and suggestions write up. We did not do some of the more advanced techniques such as using dot product. Merely, we used the idea of lining up of the pieces and checking the number of consecutive pieces given a position. We applied weights to the various outcomes similarly to the write up.

We measure 3 different ways moves are checked: diagonal, horizontal, vertical in order of decreasing weight (and therefore importance). For example a k-1 for vertical may be factored in at a weight of 1 *x but a k -1 of horizontal may hold a weight of 1.1 * x. When we do the heuristic check we take a combination of the best move and add .5 of both of the other 2. The best move is defined by the score returned from the direction checked.

A goal state would return roughly 1000 for example while a k-1 might return 90. The difference between the two numbers is not extremely important so long as the goal state is large enough so that a combination of other states not containing a goal state would be larger. Therefore, if a move contains a goal state it will never be outscored by a move that does not have a goal state. Since the max it could be if all 3 directions had k-1 is 90 then the max score that that does not hold a goal state would be 90 + .5*90 + .5 * 90 = 180.

We also took into account dead ends. If, for example, there were k-1 pieces of a given player that was blocked on both ends (by either the opponent and/or the end of the board) then that would direction would return the score of nothing. This is because we did not care about that particular direction at that point since it had no potential. Additionally, while a number of consecutive pieces had neither side blocked then it would return 1 * y where y is the score. However, if just one side of the pieces were blocked then it would return, for example .8 * y. This is because if pieces had nothing blocking it one either end it would hold more potential.

If gravity is on then any vertical piece combination will automatically have at least 1 dead end. To improve we would add more weight to moves closer to the middle and would take into account situations in which there are k-2 pieces in a row which contains free spots on either side. If that situation is not addressed by the opponent then it can be an automatic win. This should be given extremely high priority and heuristic score.

2. We did not implement Alpha-Beta pruning. While we were finishing our Min-Max algorithm along with the heuristic function, it already was good enough to find the best moves to beat or tie Poor AI and Average AI and didn’t see the need to implement Alpha-Beta pruning.

3. Iterative Deepening Search (IDS) wasn't really complicated to implement. It was something that was implemented later throughout our programming process (we wanted to complete Min-Max first). Essentially, we pass through a current depth and the depth limit each time we call a generateMAX or generateMIN function. At the beginning of these functions we'd increment the current depth and check if it's greater than the depth limit. If it's greater than the depth limit, we'd calculate the heuristic of the current node and return the result.

First getMove is called and depth limit starts at 1. Normally it starts at 0, but to avoid the obvious redudnant loop, we decided to start with depth limit of one. We would go into our while loop that

would exit if our turn time is over or if our depth limit is greater than the K length because it wouldn't be as necessary to check moves more than K depth as our goal is to achieve a length of K in order to win in comparison to check all the moves available. The most efficient move would just to make K moves to make K pieces in a row to win. Starting at depth 0 we increment our depth limit each time we finish gathering the best move at getMove.

Figuring out to make our cutoff for our depth limit at K was the most challenging part. Otherwise it would keep checking for moves and consume more time. In addition incorporating IDS with how Connect K works took some times in order to grasp the ideology and logic behind making it work, as sometimes we'd have to move our code around the loop or if statements or making sure we're checking the right boundaries for our values.

4. We used an arraylist that would store the potential moves that one can make at a certain point. We'd call a function that would generate this arraylist whenever we call generateMIN or generateMAX and iterate these moves to call the corresponding generateMIN or generateMAX. There wasn't any sorting that was done. Instead, we'd take into account from the list of moves that we checked and took the greatest heuristic score and keep track of the greatest score along with the move. It would update it if it found a better score. The data structure did help with our algorithm and keep it from running into errors.

5. We didn’t do a quiescence test. Why? Because according to the report it did not definitively improve the AI. We knew there were more important features we could have worked on that would give us an improvement so we didn’t see the quiescence test as a priority.

6. Initially, the instructions to test the project on openlab were not very clear and my partner and I were quite worried that it would not work correctly. This is a fun assignment, but perhaps the instructions could make it clear that this project is mostly about creating a great heuristic after creating the min-max algorithm and alpha beta pruning.