CONNECT-K FINAL REPORT

Partner Names and ID Numbers: Vikram Sugumaran 67021409, Rushabh Shah 94725179

Team Name: La Flame

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. Our heuristic essentially functioned by first checking if the opponent was "close" to making k. If it was, we bypassed minimax search and just blocked that from the relevant end (so it functioned kind of like an "emergency button" in this case). Otherwise, we performed minimax alpha-beta IDS with the heuristic of rating positions where our AI is closer to k higher (our AI having 4 in a row being preferable to having just 2 in a row). In the former case ("emergency button"), we used the lastMove variable as our base; we checked k steps from lastMove in all 8 directions, and blocked from the direction where the opponent was making the most progress. If this progress didn't meet a certain threshold (i.e. there aren't 3 or more pieces k steps from lastMove), we defaulted to the IDS search. We tried at first to just do a raw IDS search without the defensive "emergency button," but we kept getting awkward results (probably mixing the weights of "close to winning" and "close to losing" in a bad way, so we instead decided to hardcode in the important defensive portion of our AI before getting involved in heuristic evaluation. At first, our "emergency button" only blocked moves where the opponent had greater than or equal to 3 *consecutive* pieces from lastMove, rather than 3 pieces from lastMove within a range of k. However, averageAI takes advantage of not placing its pieces too consecutively/obviously, so we had to modify the emergency button to not be "tricked" by the empty spaces that averageAI intends to fill in later. Our current AI usually ends in a draw against averageAI and sometimes loses.

2. Alpha-beta pruning was not very difficult to implement. It was only a minor layer above minimax search. For a max node, we evaluate the value of child nodes one-by-one. If a value (which takes the place of alpha here) is greater than or equal to beta, then we know a min node above will not pick this node because it has a better alternative, so we immediately break and return this value, pruning the rest of the node's children. An analagous pattern applies to a min node. Significantly, adding alpha-beta pruning allowed us to reach depths up to 30 in reasonable time while without it we were stuck at depths around only 3 and 4.

3. We used C++'s standard chrono library to keep track of time. At the beginning of makeMove, we record the start time. We repeatedly perform minimax search with an increasing depth limit. When time is up (current time – start time > 5000 ms), we immediately break and return the move found on the last fully completed minimax search.

4. We did not sort nodes.

6. We wrote a neural network for a fixed 8x8 size board and trained it on k=5 games. Although our implementation of the neural network itself was functioning, we had trouble figuring out

how exactly to train it. As the true cost for any board position is unknown, we did not have any good set of “y values” to perform supervised learning on it. One thing we tried was to use the values of nodes obtained in Monte Carlo Tree Search, as these values supposedly converge to the true values of the nodes over time, but we found that the net would “overfit” the MCTS search tree and fail to generalize to other board positions. Although we did not know much about reinforcement learning, we tried to implement that by making it play against itself. Although it could learn to beat itself, “itself” is not a very good opponent to begin with, and it did not really mean anything in practical play. Given the uncertainty about how long it would take to overcome a learning curve before exhibiting good play, we decided to ditch the idea. An idea that was not tried was training it to beat the average and good AI s provided in the samples. However, we decided not to spend effort on this route because we were running short on time at that point, openlab's installation of numpy did not appear to be compatible with Python3, and we had uncertainty over what the board size and k would be (the neural net's input layer is not dynamic, so the board size would have to be predetermined). In any case, the Python code for the neural net is included in the submission if you would like to inspect it (the majority of it being implemented in connectknn.py).