Shaurye Mahajan

MP2 Report

Architecture: I have 4 classes that I used to complete this assignment:

Main: Takes in command line arguments and makes a board object to pass to a game object. The start function is called on the game function.

Board: Makes a 2D array of ints. 1 -> light, 2 -> black. It also has a function to generate the next moves.

Game: This is where the action happens. The player gets input from command line and the AI calls its best move function from the AI class continuously until the game is over.

AI: Contains the minimax function with AB pruning as well a getscore function and its subfunctions to calculate a board heuristic.

Search: To get the best move at any given board situation, I called a method to get the best move. First, it would check if there was a winning move by seeing if there were 4 consecutive stones with open ends and return that. If not, it would return the result of a function that implemented the Minimax Algorithm with Alpha-Beta pruning. Moving on now to the minimax function, whenever the depth hit 0 (base case). It would evaluate the board. This evaluation process calculated an informed heuristic based on the sum of three calculated scores: vertical, horizontal, and diagonal. These individual scores were calculated by checking how many consecutive stones (w/ open ends) there were and assigning varying amount of points for them. I had to fiddle with how much each was worth of course. I also optimized my minimax search by making it only search through empty spaces that had adjacent stones, so it wouldn't have to search for all empty spaces.

Challenges: One of the biggest challenges I was having was sue to my program taking too much time. Even though the move was under 30 seconds, it would usually go over the 2 minutes. After timing each step out, I realized that I could significantly reduce the time spent if my NextMoves array only contained spaces that were adjacent to already placed stones. This change allowed me to stay under the time limit.

Weaknesses: When I was running the referee.py script to see how my AI matched to the baseline, I realized that the only time my AI lost was near the start of the game. It seemed like the evaluation function would not find that the opponent had a winning move early on. This was probably because there was a slight problem with my heuristic,

or because my NextMove function only analyzed spaces with adjacent stones. To overcome this, I could have implemented another function to search for a winning move for the opponent and not just for myself and make a move there.