Erick Suarez
4292702
erick_suarez@ucsb.edu

• Architecture: Explanation of your code architecture (briefly describe the classes and basic functionality):

The Architecture is very similar to that of the RandomHex.py file. I made the functions into a class titled HexAgent instead. Keeping track of the hexboard, size, color, playermoves, and whether it's the first move or not. I replaced the random_search with a function minimax which also returns the best move. All other functions assist the minimax function in returning the best value.

• Search: How you search for the best move (which algorithm you use, what heuristics, optimizations, etc)

I used the minimax Algorithm with alpha beta pruning in order to determine what the best move is. In the minimax algorithm, for all adjacent moves for all my currently marked cells, I call the maxValue function which will return the maximum heuristic score given a current depth for that move I passed in. The internals of the function include keeping track of a depth, alpha, and beta. For every adjacent move again I simulate a move for the opposite player and call minVal for that state. The minVal function takes in the same parameters and will simulate a move from the original player and call maxVal with that state. So on and so forth unless the depth = 0, or the board is full, which if it is, the heuristic function generates a score for the current state and returns it.

The heuristic function goes through the current state, keeping track of the lowerlimit and upperlimit of the cluster of cells. For every move that belongs to the player it keeps track of the number of nodes its connect to. Afterwards I calculate a value, giving a higher importance to states that are closest to the upperlimit and lowerlimit while still maintaining a chain of cells. To keep track of the lower limit and upper limit as well as the connections I recursively check every nodes neighbors that have the same color, keeping track of the total as well as updating the limits as needed.

It was important to limit the algorithm with a depth as it would take too long to search the entire game tree. Another optimization was using moves that were adjacent to current cells instead of all available moves. When calculating the limits and number of cells in a chain I also use a map of moves to keep track of which ones i've visited in order to not recount any cells.

• Challenges: The challenges you faced and how you solved them

Some challenges I faced that I solved was a long compute time which I solved by implementing the optimizations I talked about. Another was trying to figure out how to ignore dead cells but was not able to in time. Another challenge was making sure the minimax algorithm and heuristic function was working correctly, which I had to go step by step to make sure each step had the expected results or values.

• Weaknesses: Weaknesses in your method (you cannot say the method is without flaws) and suggest ways to overcome them.

My method is definitely a greedy algorithm that tends to get trapped by, especially by baseline2. Another weakness is that it wastes moves picking useless cells.