Path planning

In my path planning program I first created a cell class to make it easier to work with each cell.
Every cell has a position as well as a h,g,f value. The g value represents how many steps away from the start that the current tile is. The h value is the heuristic value of the cell and the f value is the sum of g and f. I then created the function get_neighbors to return a list of each available neighbor of the current cell. I have a list for tracking all the cells that has been accessed to avoid redundancy. I also created the function get_prio to return what value the cell should correspond with in the search tree. This will be different based on what search type is given as input when starting the search. For example, BFS will sort the cells based on how far away they are from the start. Cells further away will have a lower priority in the search tree.
I implemented the get_prio function for random search, BFS, DFS, A* and greedy search. When creating the custom heuristic I first check whether the starting position is located closer to the bottom of the obstacle. If such is the case then cells closer to the bottom of the obstacles will be prioritized. Once the bottom is reached the rest of the heuristic is based on manhattan_distance. If the starting cell was closer to the top we do the same thing only upwards. This heuristic utilizes less memory since it opens fewer cells, however, the path found is often longer then the path found using A*. The same goes for greedy search. This opens fewer cells than A* but the final path is not the optimal. It shall be noted that the custom heuristic opens less amount of cells than greedy search.

Poker

I implemented the search algorithm for the poker assignment in a very similar way to the path planning. Instead of using the cell class I used the states and therefore the get_next_states instead of get_neighbors. As a goal state I chose to look at the difference between how much money the two players have. If this difference is 200 or greater, we have reached our goal. I used the same get_prio function as in the previous task to retrieve a priority based on the type of search conducted. I also implemented a max depth in my search algorithm to avoid getting stuck on a bad path.
I implemented two different heuristics. One based on how many bets have been made (as the suggestion in the lab description) and one based on how much money we have and how much money is in the pot. The latter one proved to be more efficient in terms of both speed and memory.
When I laborated with the different search types I noticed that the random search was surprisingly good. It found a solution in 4 hands or less most of the times and often quite fast.
Using exhaustive search the solutions were often the most effective but the computation time was the longest. It often needs more than 40k steps to find a solution. A* almost always finds a solution in less than 3000 steps using my own heuristic. Using the suggested heuristic the steps are often >30k using either greedy or A*.