CS 440 MP 1 Report

Adarsh Ramchandran (adarshr2)
Section R3

Jessica Gomes (jsgomes2)
Section Q3

02.03.2019

Section I

DFS:

In the DFS search algorithm, nodes/states are represented by each block in the maze which is not a wall. We use the Python list data structure as a stack to represent the frontier, which consists of the states we wish to visit. Our explored states list is represented by the 'visited' list. In addition, we maintain the path using a hashmap 'prev' in which the key represents the node and the value represents it's parent node. While exploring nodes, we only add the node's neighbors if it is not already in the visited list. This detects repeated states and ensures the same node is not explored more than once. After a goal state was reached which was the state with a dot, no more states in the frontier are visited. We then construct the path using the 'prev' hashmap and the number of states explored using the length of the visited list. The algorithm returns these two values, the path and the number of states explored.

BFS:

In the BFS search algorithm, nodes/states have the same representation as the DFS algorithm. A major difference in this algorithm is that the frontier is represented by a queue data structure. The reason for this difference is because in BFS, you want to explore states layer by layer and as a result want to pop out nodes in the order you push them in. Thus, a queue functions properly for that purpose. Explored states are represented in the same way as DFS using the 'visited' list. Similarly, exploration of states terminates once the goal state is reached. Finally, the path is reconstructed using the 'prev' hashmap and the number of states is calculated through the length of the visited array. The algorithm returns these two values.

Greedy:

Greedy Best-First Search differs from BFS and DFS in that it takes into account each coordinates' relative position to the goal. The traversal of the maze is determined by which node has the lowest computed distance from the goal (the used heuristic is manhattan distance, which simply computes distance as if there are no walls in the maze). Since we always need access to the lowest distance, we use a priority queue and decide which node to visit next by popping off the queue. The visited nodes are kept track of in a dictionary that has visited nodes as keys, and their parent nodes as values. This allows us to reconstruct the path by simply following back the parents dictionary from the goal node back to the start node.

A*:

Just as in Greedy BFS, A* takes into account relative distance to the goal. It is an improved version of greedy because it actually computes the current cost of the path to decide which

node upon which to expand. The heuristic used is manhattan distance used in a cumulative form, where the total cost of the path equals the manhattan distance of the current node added to the cost of the path so far. We therefore again use a priority queue called ‘openSet’ to pop off the minimum distance node and a parents dictionary to retrace our steps after we have found the goal node. The visited nodes are represented by the list called ‘closedSet’.

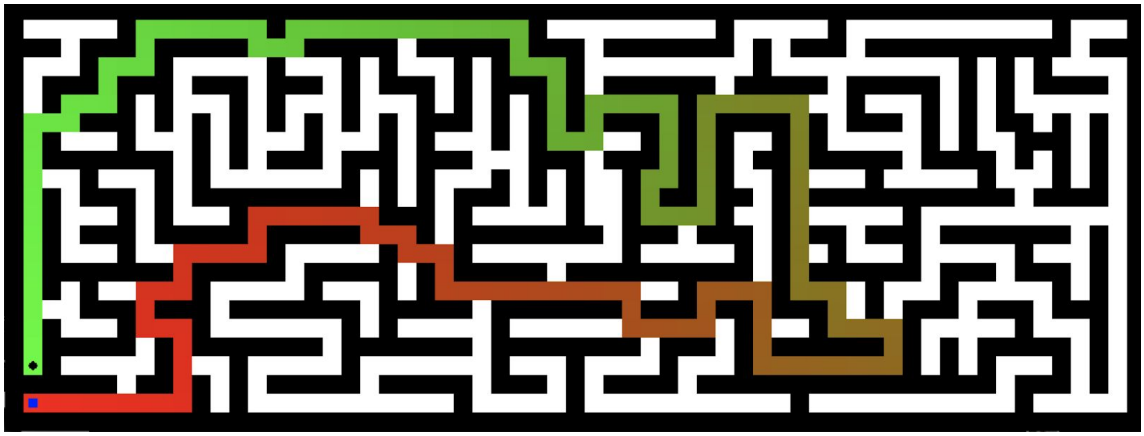
Section Ⅱ

To modify A* to deal with a maze with multiple dots, we first wanted to find the order of dots we should traverse to get the optimal solution. This required getting all the permutations of possible traversals, and then using our original A* algorithm with an additional parameter: a single possible permutation of dots. For each permutation, we perform A* to go from dot to dot while keeping track of the path and path distance, so that we can determine the optimal path by simply taking the minimum of these values. The heuristic used is clearly admissible, as essentially we use the already admissible aforementioned A* heuristic on each permutation to decide what traversal of dots is optimal. However, the efficiency of the heuristic made it unusable, as it took far too long to produce a solution. Instead, we pivoted to a different implementation in which we reduced the problem of multiple goals to several iterations of single goal A*. We started by finding the closest dot to the start node (using the already admissible manhattan distance as a heuristic), and then calculated the next dot with the minimum manhattan distance from the current dot. We extended our final path list with each of these intermediate paths and returned this final path along with the number of the nodes explored (which was the length of the parents dictionary we maintained). While this method actually resulted in a solution image, it did not return an optimal path, and we suspect an optimized version of our original idea of going through permutations would have given us an optimal path.

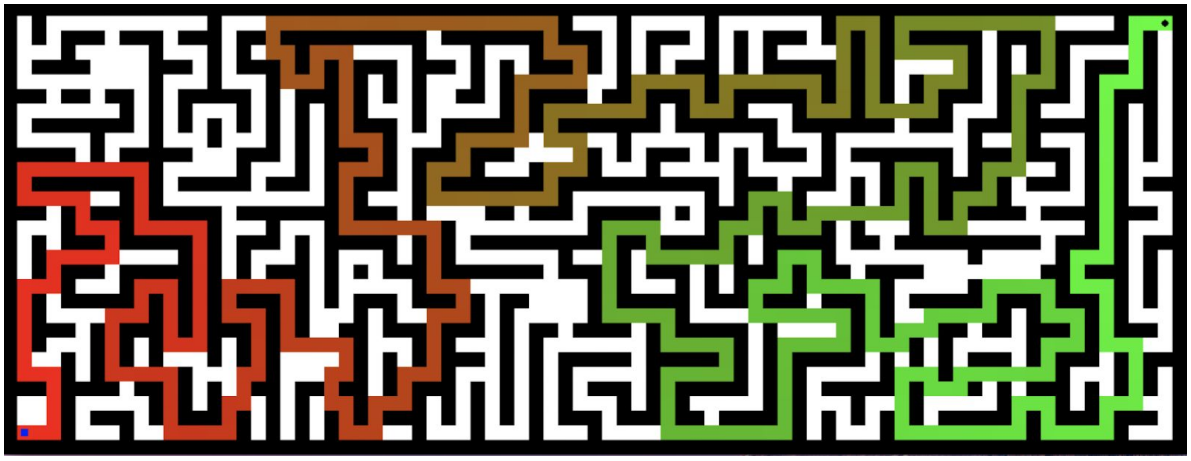
Section Ⅲ: Results

DFS

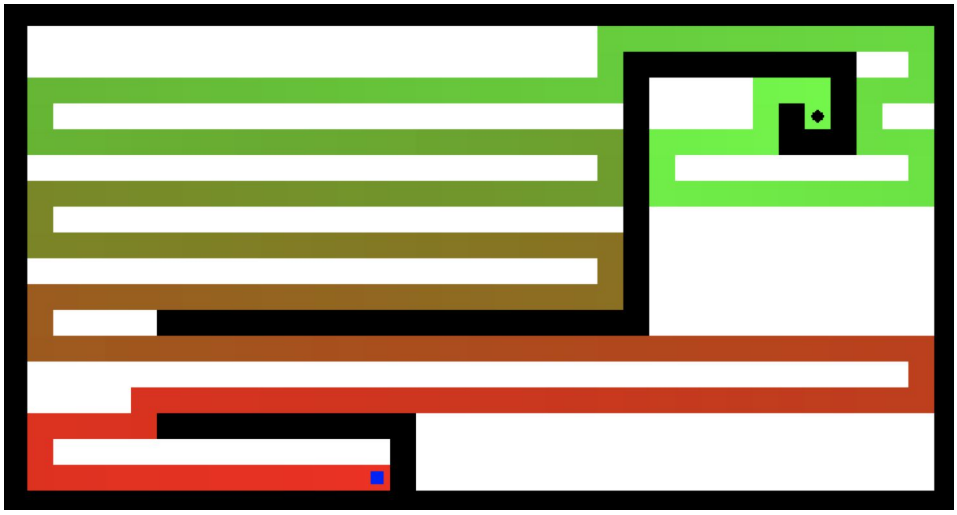
- mediumMaze.txt → (Path Length: 175, States Explored: 392)



- bigMaze.txt → (Path Length: 537, States Explored: 1088)



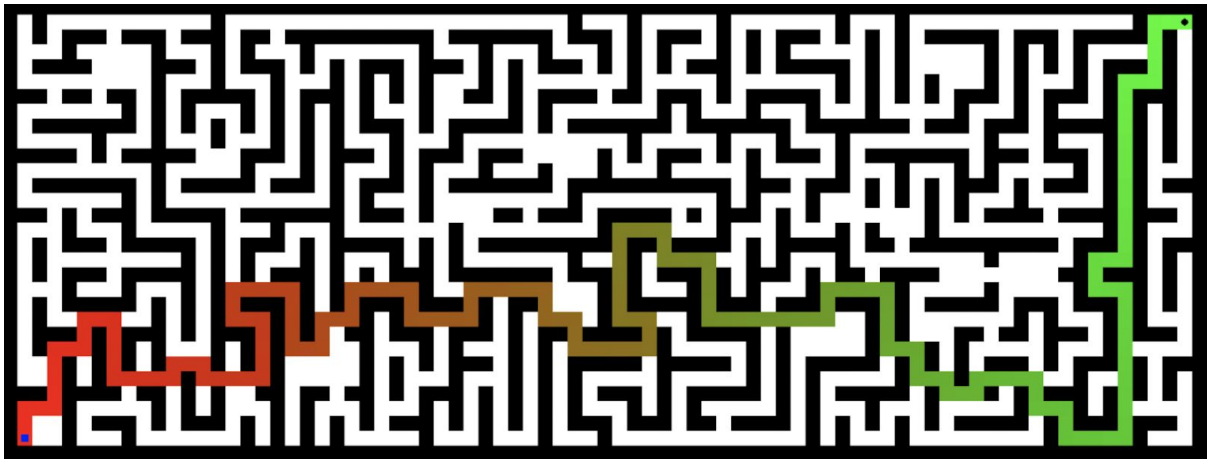
- openMaze.txt → (Path Length: 252, States Explored: 502)



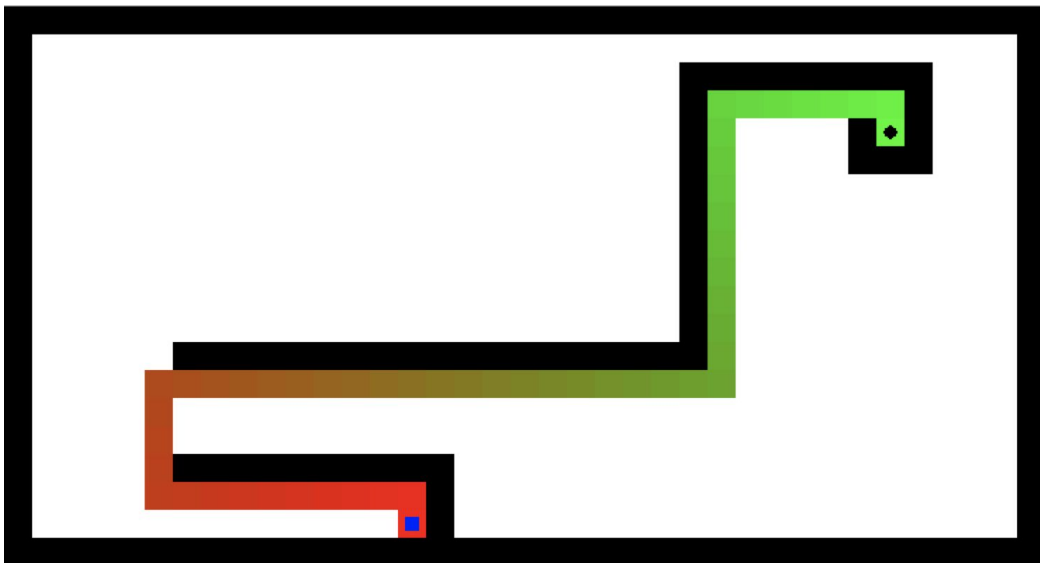
BFS

- mediumMaze.txt → (Path Length: 111, States Explored: 647)

- bigMaze.txt → (Path Length: 183, States Explored: 1287)



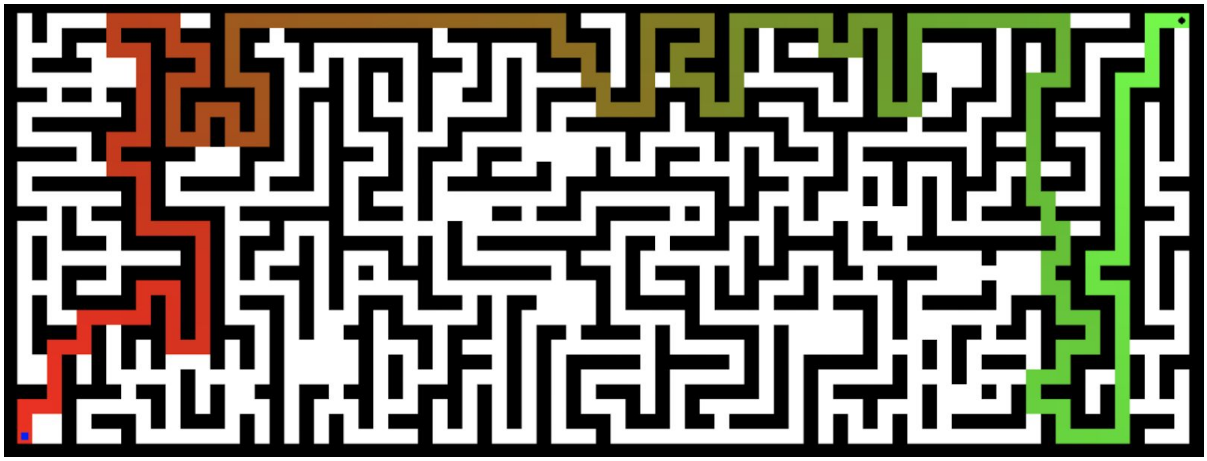
- openMaze.txt → (Path Length: 52, States Explored: 562)



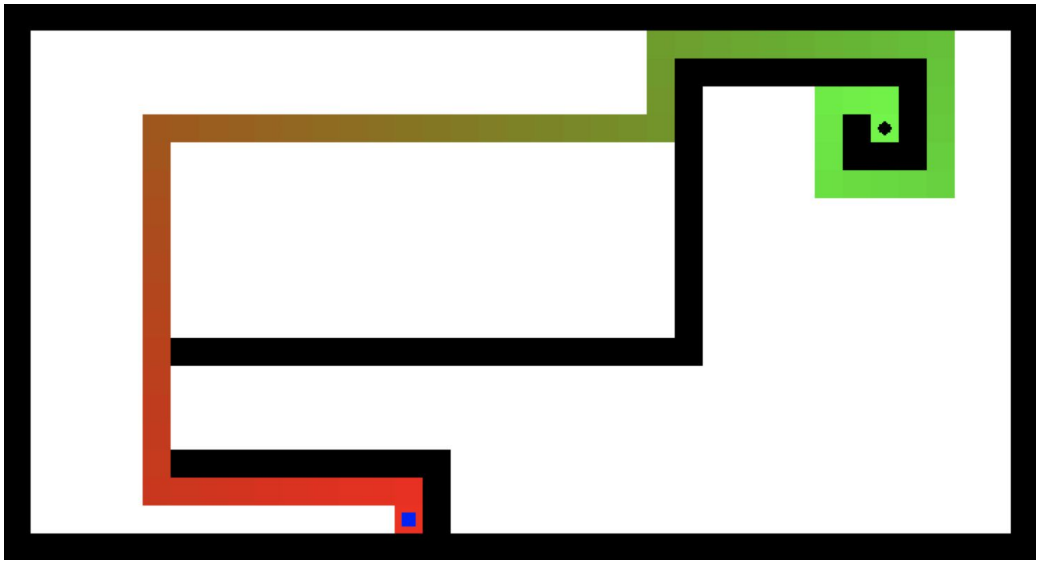
Greedy

- mediumMaze.txt → (Path Length: 147, States Explored: 351)

- bigMaze.txt → (Path Length: 277, States Explored: 492)



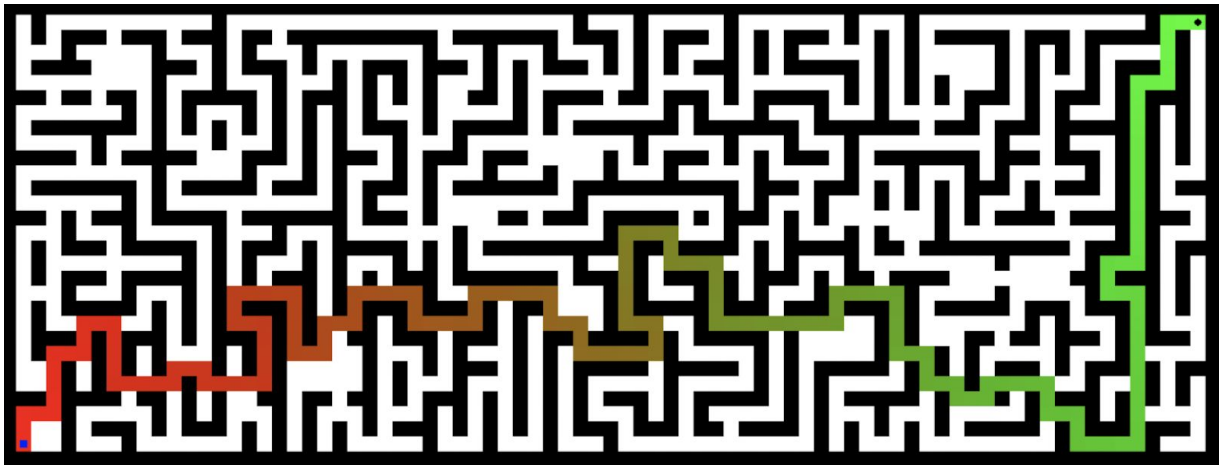
- openMaze.txt → (Path Length: 70, States Explored: 162)



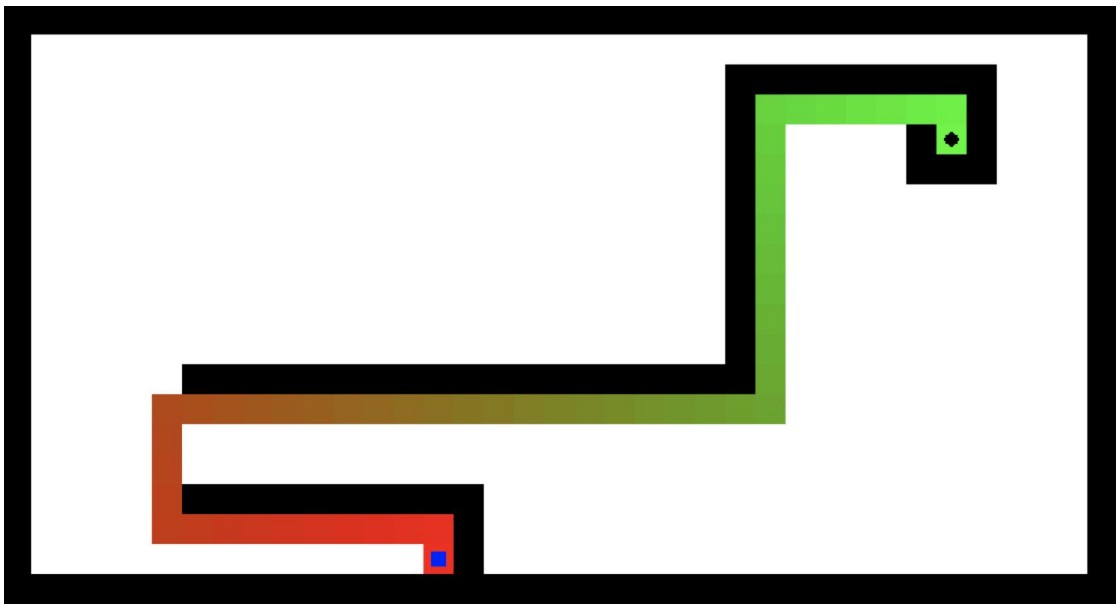
A*

- mediumMaze.txt → (Path Length: 111, States Explored: 376)

- bigMaze.txt → (Path Length: 183, States Explored: 1252)

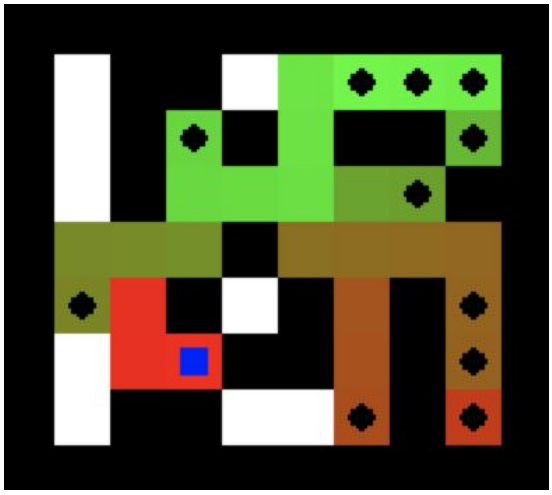


- openMaze.txt → (Path Length: 52, States Explored: 219)

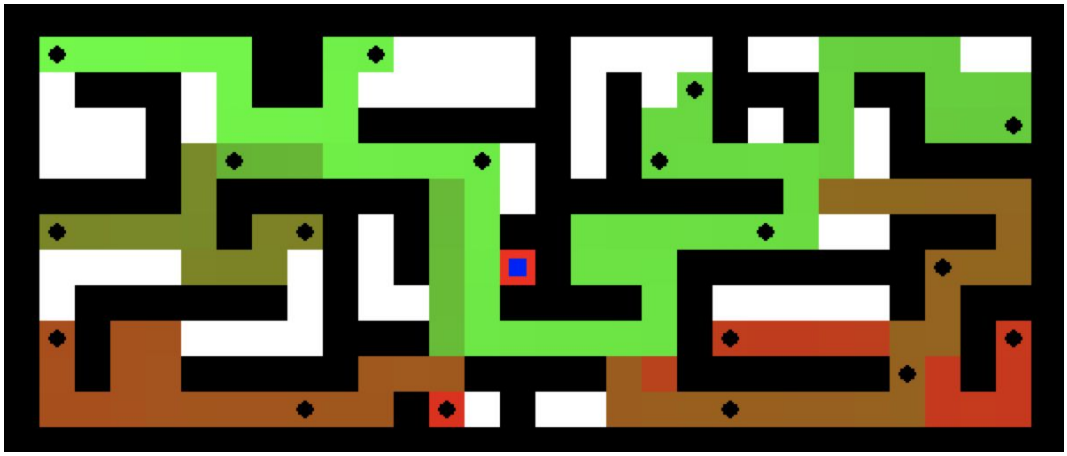


Section Ⅳ

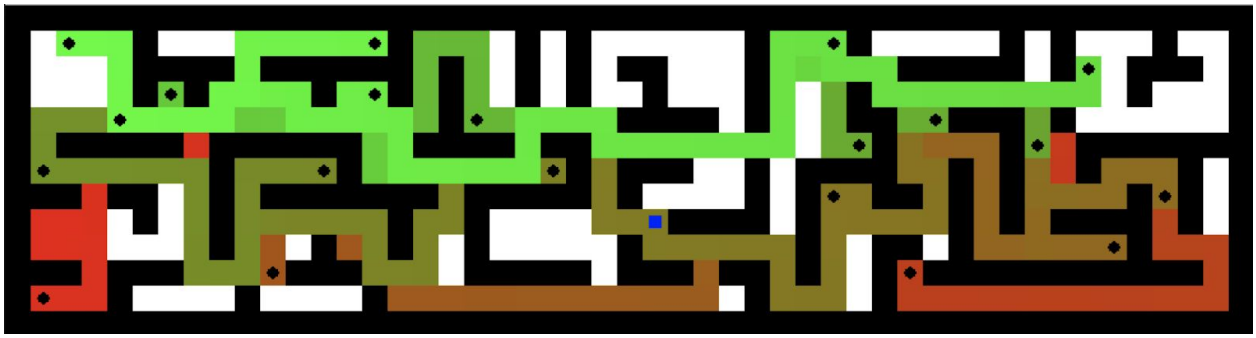
- tinySearch.txt ⟶ (Path Length: 77, States Explored: 147)



- smallSearch.txt ⟶ (Path Length: 384, States Explored: 1040)



- mediumSearch.txt ⟶ (Path Length: 631, States Explored: 2132)



Statement of Contribution

Jessica and Adarsh pair programmed to implement algorithms for BFS and DFS. They tested it together and made sure it was working as expected. Adarsh completed the implementation of the Greedy algorithm and Jessica finished up the A* algorithm for a single dot. Both of them worked on the A* implementation for multiple dots.

Jessica and Adarsh split up Section 1 of the report. Adarsh worked on Section 2. Jessica worked on Section 3 and Section 4.