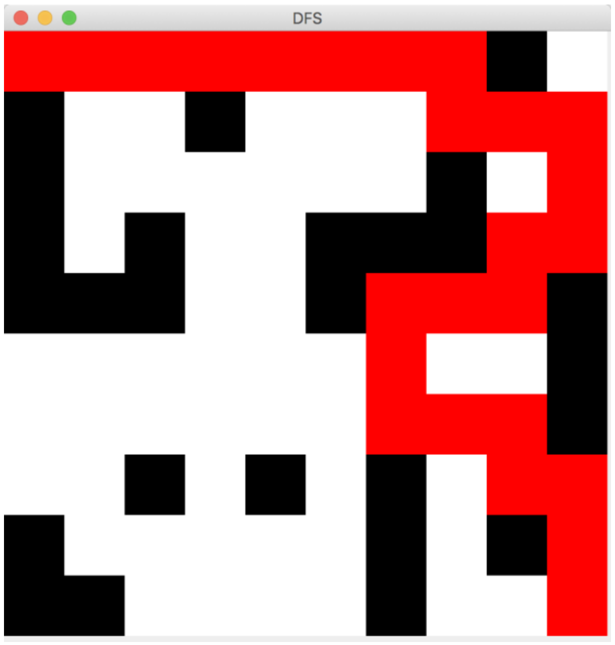
CS 520: Assignment 1 - Path Planning and Search Algorithms

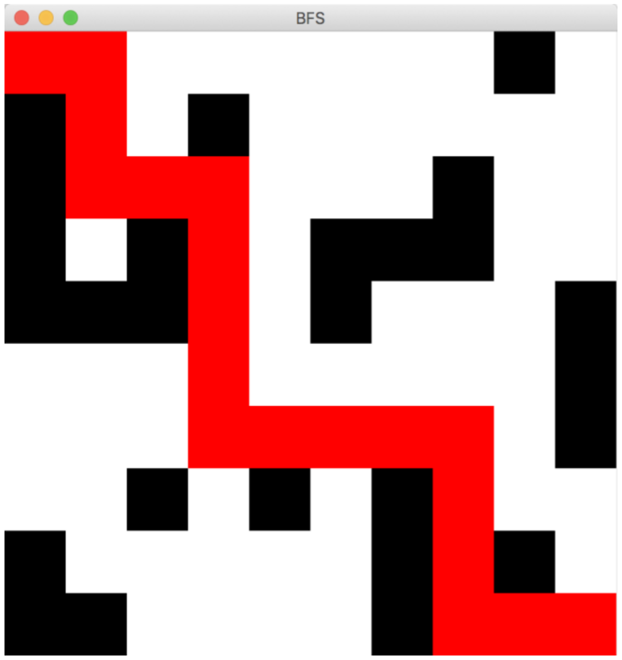
Group Member:
Xin Yang(xy213)
Zhuohang Li(zl299)

Part1:

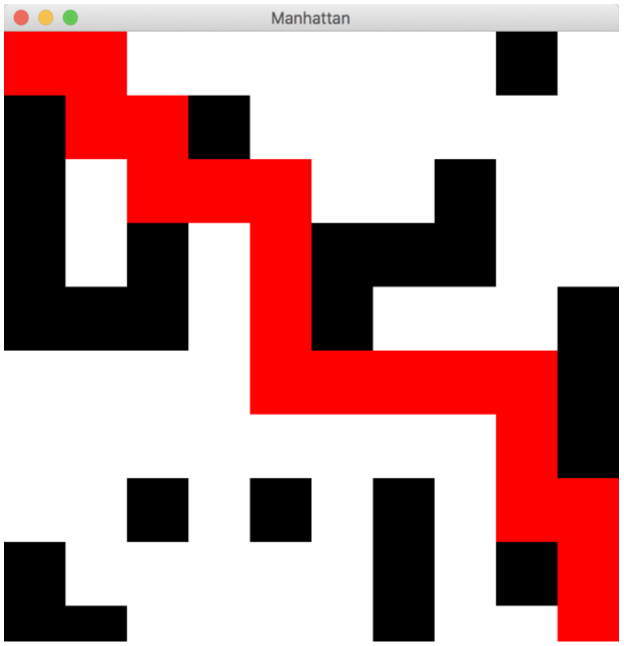
1) As p goes higher, the size of the maze that these search algorithms can handle gets larger. This is because mazes with high p values are often not solvable, and therefore results in an early exit of these algorithms. To get a general conclusion, we run our experiment with p = 0.1, 0.2 and 0.3. The results are as follow:
   For DFS, the dim of the maze can be as large as 2000.
   For BFS: 1000
   For A* using Manhattan as heuristic: 1500
   For A* using Euclidean as heuristic: 2500
2) DFS:



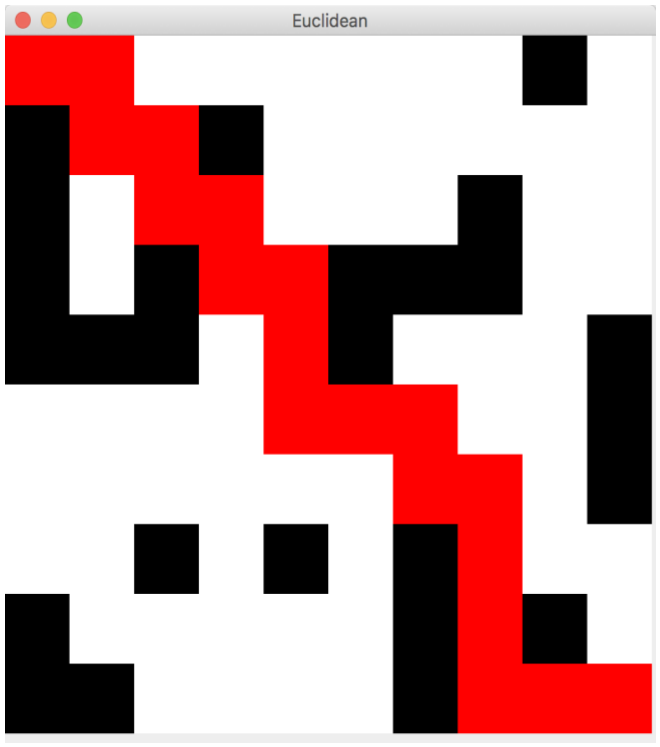
BFS:



A* with Manhattan as heuristic:

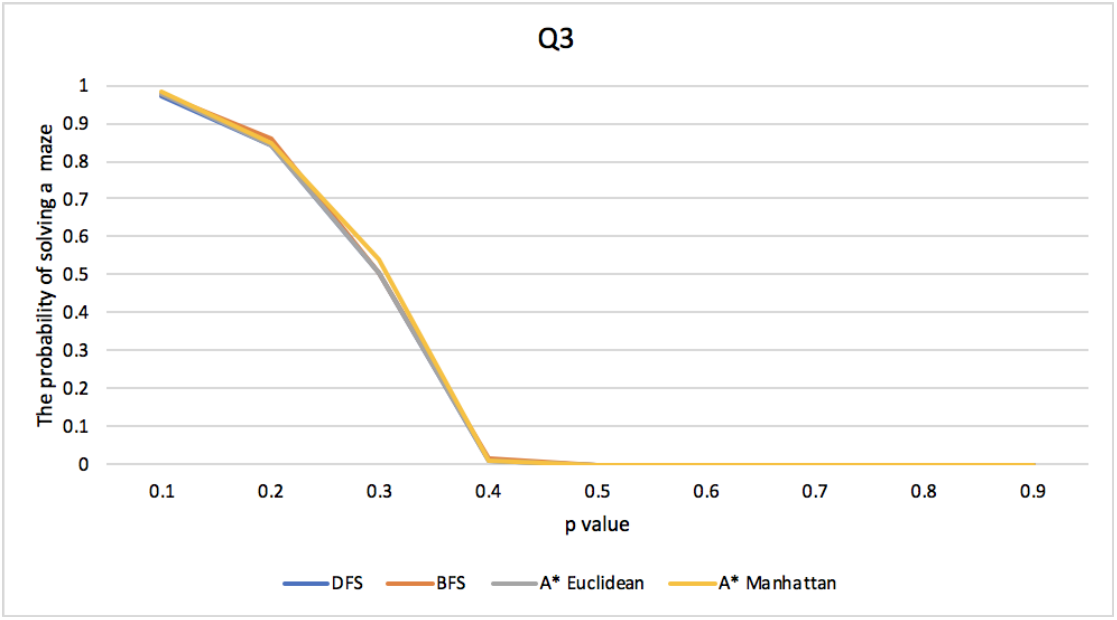


A* with Euclidean as heuristic:



3) For question 3-7, we choose the dim of the maze to be 200, and run 1000 iterations for each p:

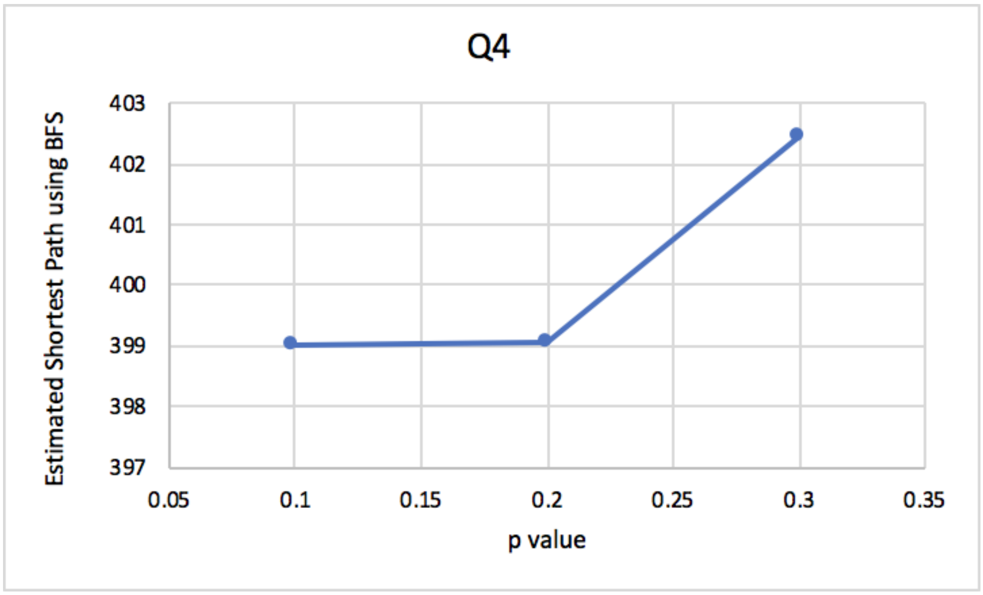
P =	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
DFS	0.973	0.843	0.508	0.01	0	0	0	0	0
BFS	0.977	0.859	0.506	0.015	0	0	0	0	0
A* Euclidean	0.975	0.84	0.507	0.012	0	0	0	0	0
A* Manhattan	0.982	0.845	0.542	0.008	0	0	0	0	0



From the chart we can see for $p < 0.3$, there is a clear path, while for $p > 0.3$, there is no path. We think A* algorithm with Manhattan as heuristic is most useful here. In terms of finding a path, the performance of all 4 algorithms is similar. But considering the efficiency of the algorithm, A* algorithm can usually find a shorter path (comparing to DFS) within less time (comparing to BFS).

4)

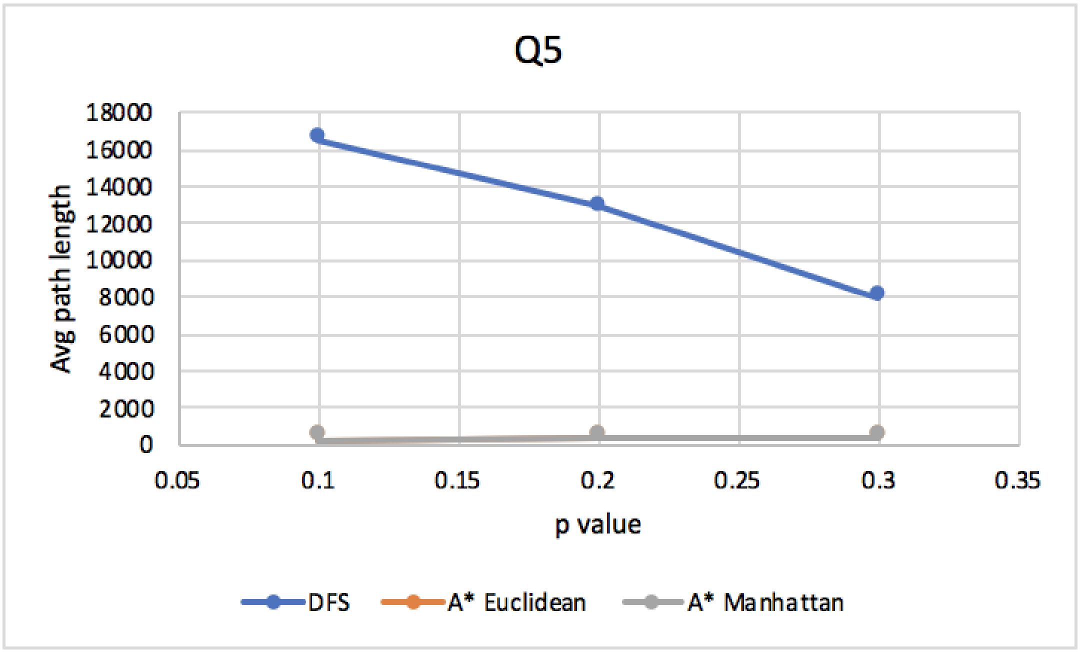
p	**0.1**	**0.2**	**0.3**
BFS	399.00	399.04	402.45



BFS is most useful in finding the shortest path. From the chart, we can see that the shortest path provided by BFS agrees with the theoretical shortest path which is 400 for a square maze with dim = 200.

5)

p	**0.1**	**0.2**	**0.3**
DFS	16566.98	12938.07	7991.66
A* Euclidean	399.00	399.04	402.83
A* Manhattan	400.95	401.89	410.02

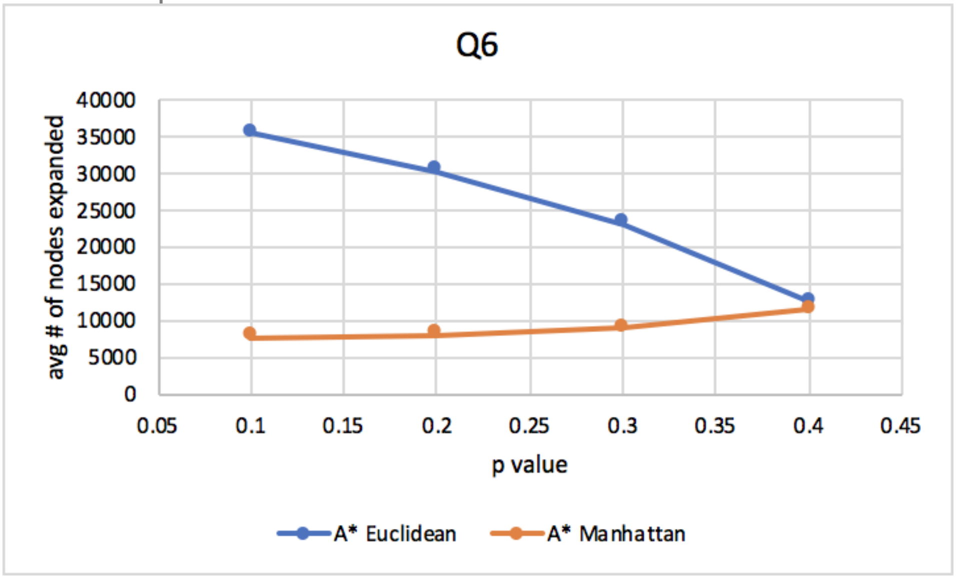


From the chart, we can see that in terms of path length, DFS's performance is really bad compared to the other three algorithms. Therefore, without regard for memory complexity, BFS, A* using Euclidean as heuristic and A* using Manhattan as heuristic all perform well. If we take memory complexity into consideration, A* using Manhattan as heuristic will still be the most useful algorithm. Because we can see from part 2 that BFS requires a much larger piece of memory to store fringe, while Euclidean distance is not a good heuristic in this problem. Also, an interesting point we can see is that as p goes from 0.1 to 0.3, the path length returned by DFS is going down while other 3 algorithms remain the same, or even gets a little bit longer. The reason being that unlike BFS which is bound to find the shortest path, what DFS returns is an arbitrary path it first found from start to goal. So a lower p value makes it much easier for DFS to make 'mistakes'. However, there are few choices in a maze with high p, so DFS is more likely to find a shorter path.

6)

p	0.1	0.2	0.3	0.4
A* Euclidean	35400.2	30325.2	23061.2	12614.7

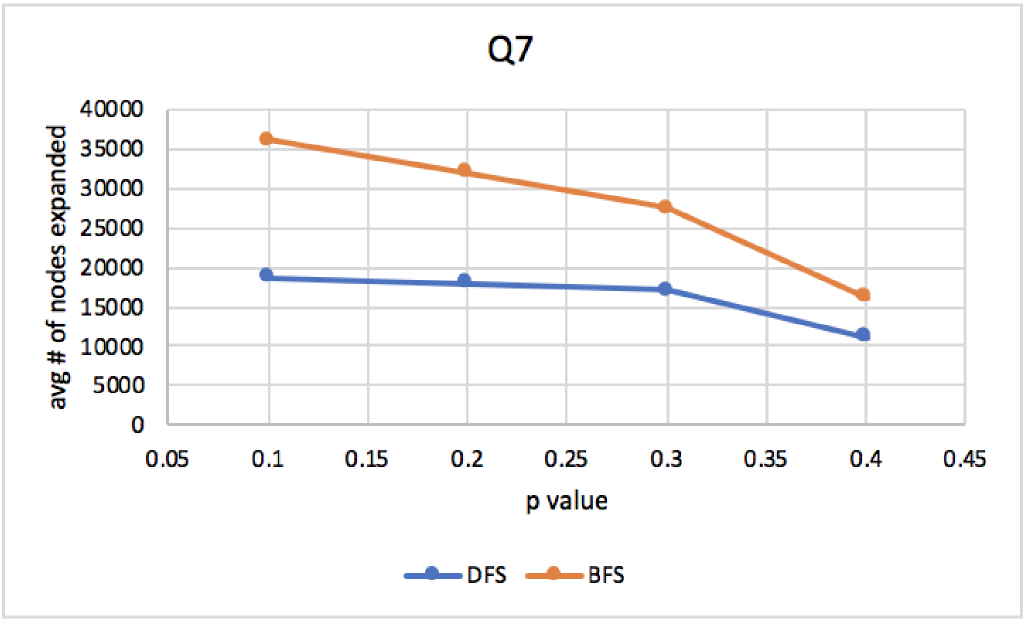
A* Manhattan	7734.3	8069.6	8911.4	11457



From the chart, we can clearly see that A* using Manhattan distance as heuristic tends to expand fewer nodes than the one using Euclidean distance. This is because that in this case, using Manhattan distance as a heuristic is more accuracy. Using Euclidean distance as heuristic may lead the algorithm into a wrong way or dead ends when the algorithm then needs to backtrack and expand a lot more nodes. For p = 0.4, the average number of nodes expanded by these two algorithms becomes close. Similar to question 5. This is because there are few choices in a maze with a high p value.

7)

p	**0.1**	**0.2**	**0.3**	**0.4**
DFS	18620	18020.6	17007.2	11168.8
BFS	35994.8	31918.5	27439.6	16250



For the chart above we can see that the numbers of nodes expanded by DFS are usually fewer than BFS. This is because BFS aims to find the shortest path while DFS just return an arbitrary path. For the maze we are using with dim = 200, the total number of nodes is 200*200=40000. For p = 0.1, 0.2, 0.3, the number of unoccupied nodes is 36000, 32000, 28000. We know that both algorithms expanded nearly every unoccupied node to find a path. For p= 0.4, the number of nodes expanded is around 15000-16000 which is less than 24000, the total number of unoccupied nodes. We think this is because some of the unoccupied nodes are surrounded by occupied nodes hence are still unreachable. In all, BFS still need to expand all reachable nodes to find a path. In terms of the number of nodes expanded, DFS performs better than A* using Euclidean distance as heuristic but worse than A* using Manhattan distance. BFS has similar performance as A* using Euclidean distance.

Bonus 1)

Uniform Cost Search is aimed to find the optimal solution for edge-weighted graphs. Although Uniform Cost Search is always optimal, its time and space complexity even worse than Breadth First Search. In this case, every step is of the same weight so Breadth First Search is sufficient and there will be no need of using Uniform Cost Search.

Part 2:

8) We selected the local beam search for that it can remember multiple previous states and have O(n) space complexity. Local beam search can better find the harder maze in a relatively shorter time, also the complexity of implementation is acceptable.

   We represent the maze as a Java class, which contains properties like the dimension, a probability of nodes being occupied, etc. The maze is defined as a two-dimensional array of Nodes, which is another Java class we created to keep useful information, such as the previous Node pointer, the value of H, G, and F in the heuristic, and if the node is occupied or has been visited. A priority queue is maintained to keep the successors for next round based on the properties we selected.

   We set the number of successors to be dynamically changed according to the convergence. For a 10 x 10, occupation probability 0.3 maze, we set the initial number of successors as 10, which is fast and efficient for the iteration.

9) Our algorithm tries to maximize the value of the property we choose. But the local beam search could fall into a local optimal since each time we only change one node. To overcome this problem, we record how many times the algorithm returns the same hard maze, if the number exceeds half of the successors, which is an effective value based on our experiments, then we enlarge the number of successors twice as before, trying to include more successors and get a harder maze. If the number of successors equals the total number of nodes in the maze and more iterations would only return a same hard level, then it is not likely to have a harder maze and we terminate the algorithm.

   This approach can effectively avoid falling into local optimal and can achieve a relatively decent result based on our test. But the problem is that with the growth of successors, more iterations are required to see if it comes to a local optimal. The convergence speed decreases as the algorithm runs to a late period. However, this feature ensures that we took more chances when the maze gets harder, and can make it always get a harder maze as possible.

10) For a better visualization result, we set the dimension of the maze as 10x10, and the probability of a node being occupied = 0.2, the number of successors = 20.

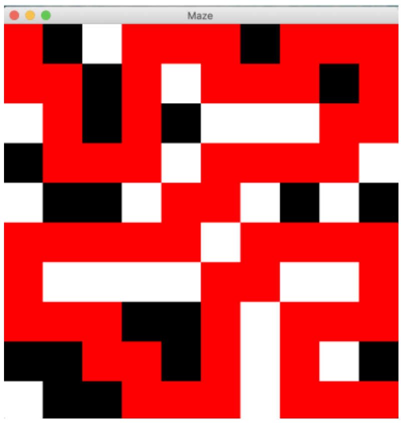
Red Square: Route from start to the goal

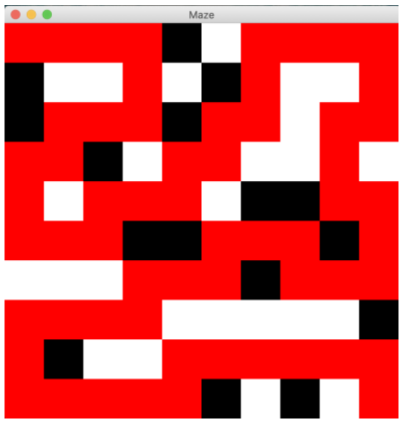
Black Square: Occupied Node(Wall)

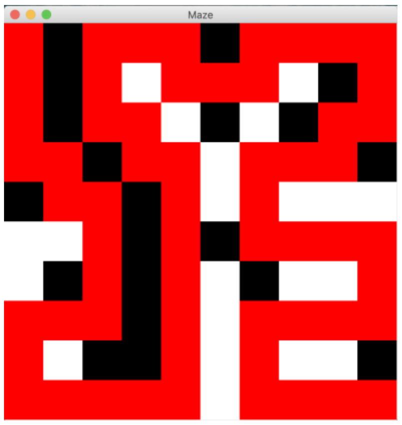
White Square: Empty Node

a) DFS

a.i) Length of solution path returned

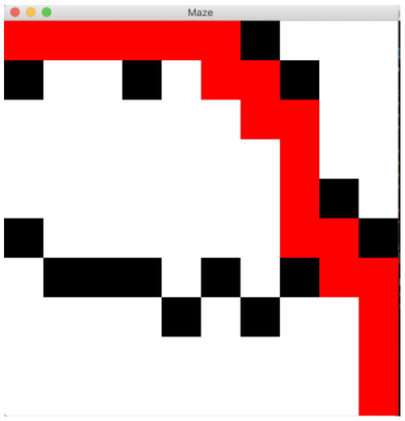


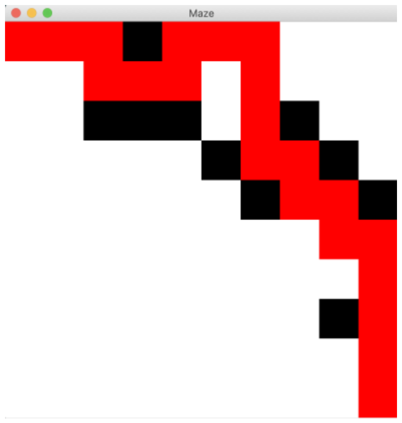


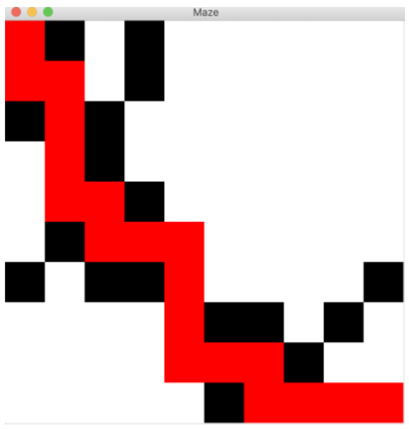


Round 1: Length 57 Round 2: Length 59 Round 3: Length 59

a.ii) Total number of nodes expanded

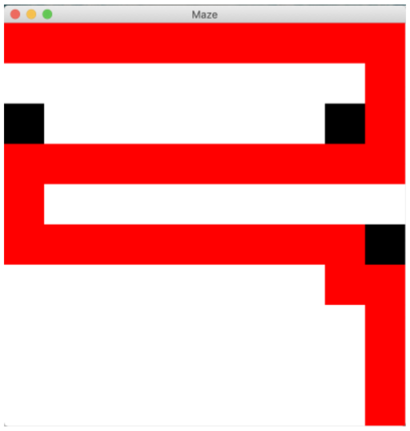




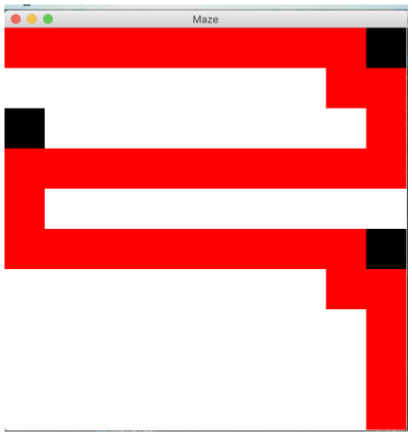


Round 1: Expand 80 Round 2: Expand 85 Round 3: Expand 83

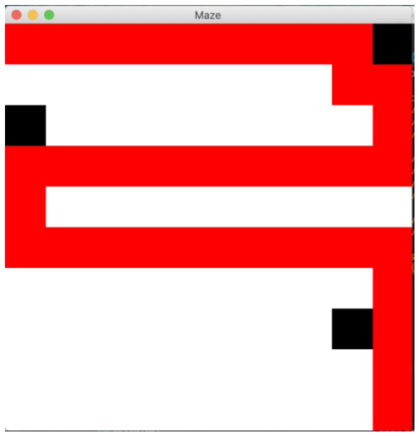
a.iii) Maximum size of fringe during runtime



Round 1: Fringe 51



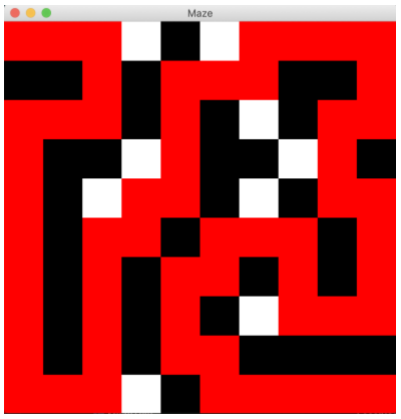
Round 2: Fringe 51



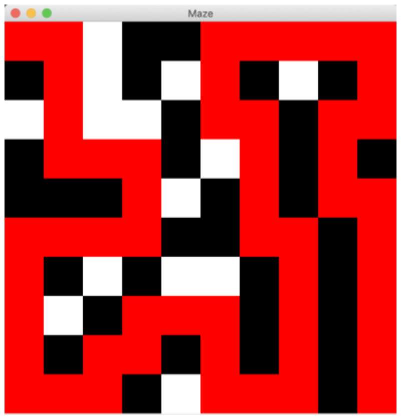
Round 3: Fringe 51

b) BFS

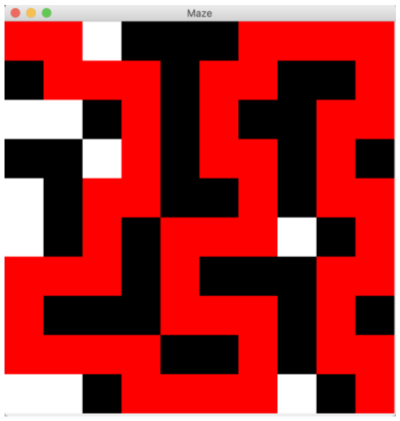
b.i) Length of solution path returned



Round 1: Length 57

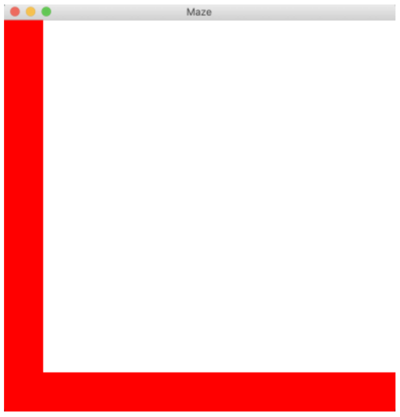


Round 2: Length 53

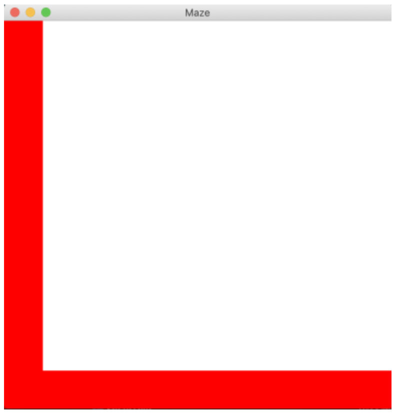


Round 3: Length 53

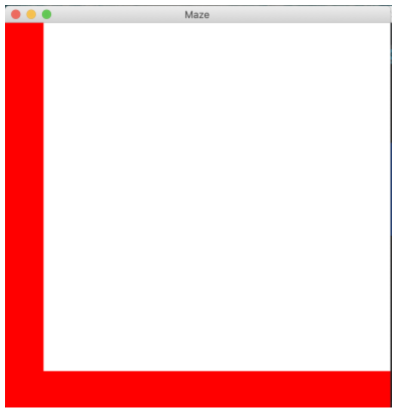
b.ii) Total number of nodes expanded



Round 1: Expand 100

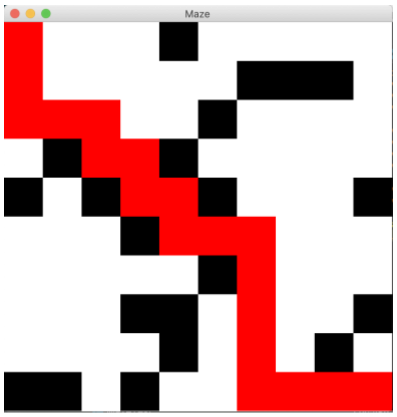


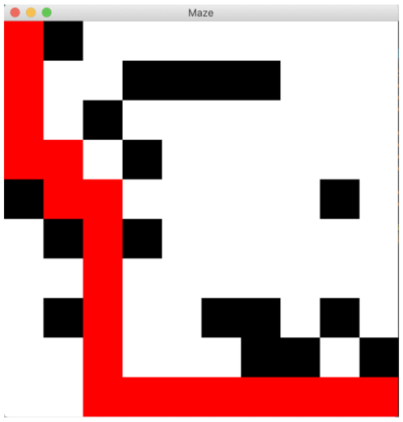
Round 2: Expand 100

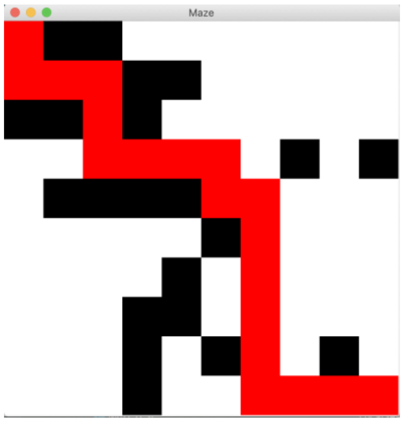


Round 3: Expand 100

b.iii) Maximum size of fringe during runtime



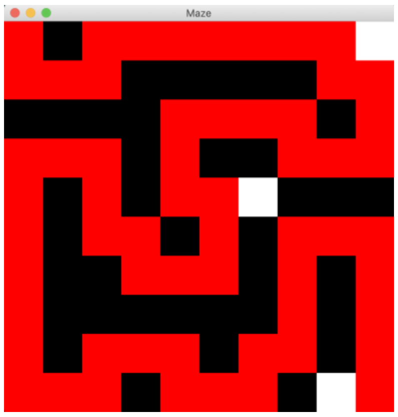


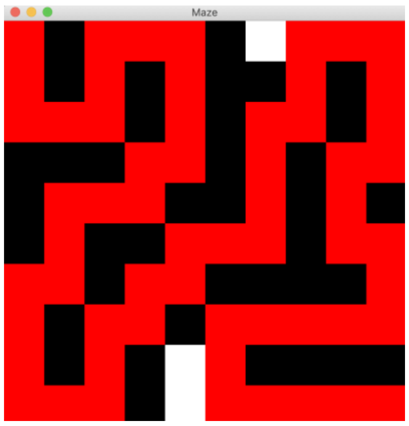


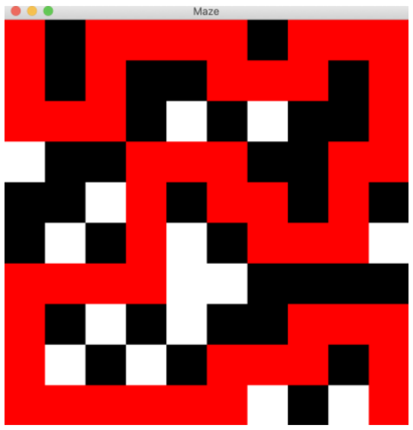
Round 1: Fringe 18 Round 2: Fringe 15 Round 3: Fringe 16

c) A* with Euclidean Distance Heuristic

c.i) Length of solution path returned

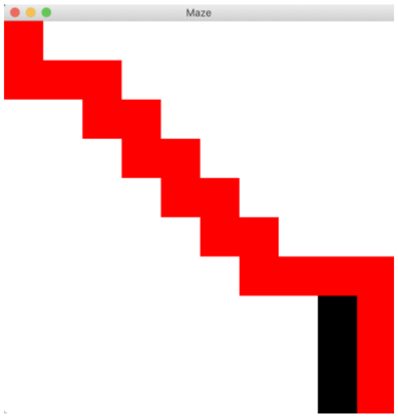


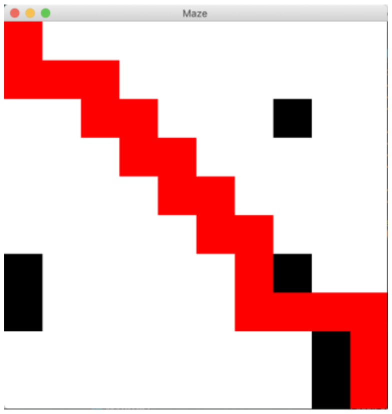


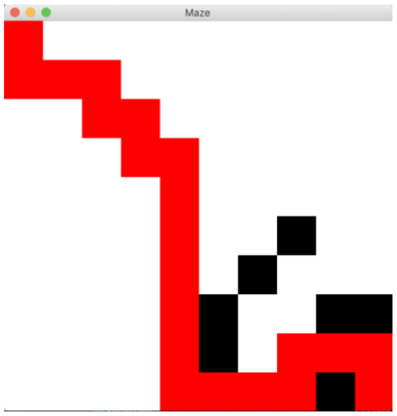


Round 1: Length 59 Round 2: Length 59 Round 3: Length 51

c.ii) Total number of nodes expanded

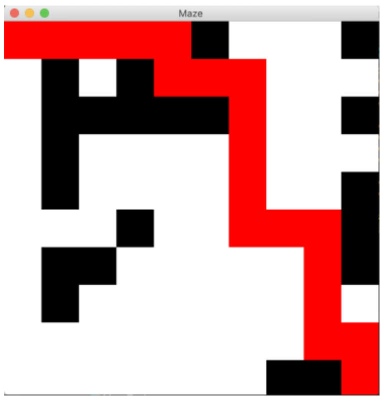




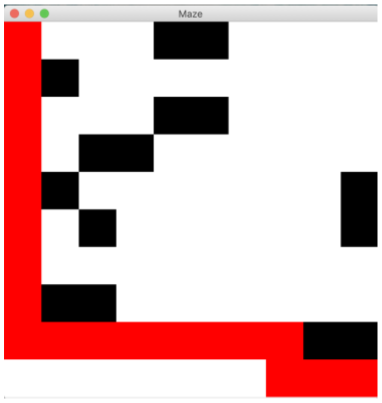


Round 1: Expand 96 Round 2: Expand 92 Round 3: Expand 92

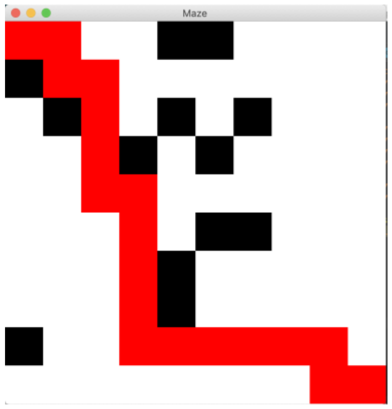
c.iii) Maximum size of fringe during runtime



Round 1: Fringe 34



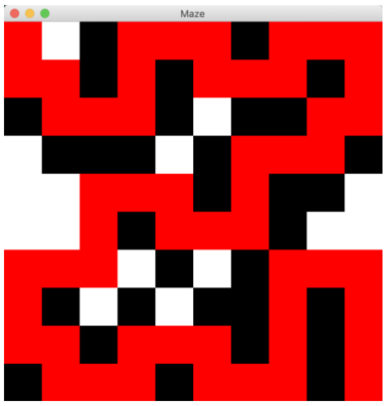
Round 2: Fringe 31



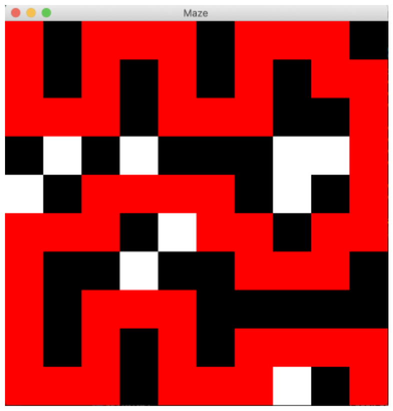
Round 3: Fringe 30

d) A* with Manhattan Distance Heuristic

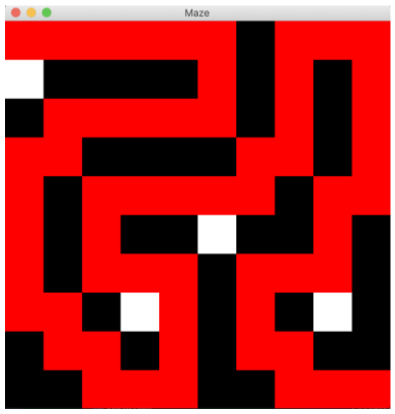
d.i) Length of solution path returned



Round 1: Length 53

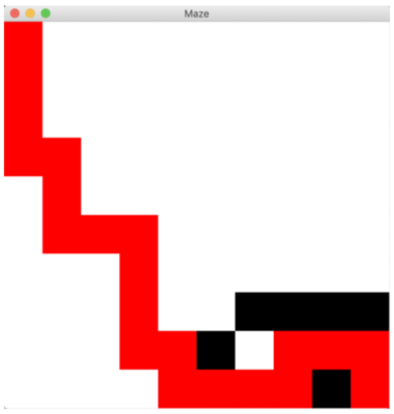


Round 2: Length 55

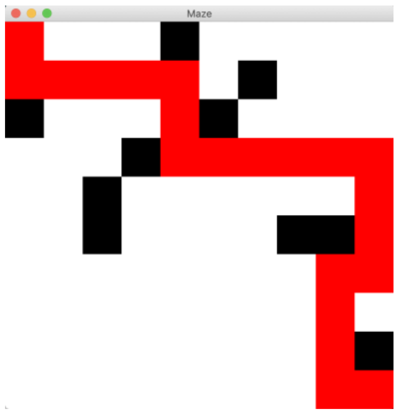


Round 3: Length 57

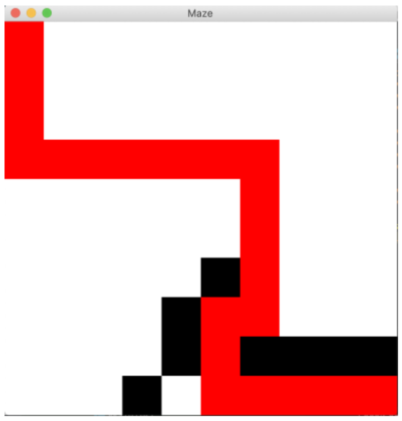
d.ii) Total number of nodes expanded



Round 1: Expand 93

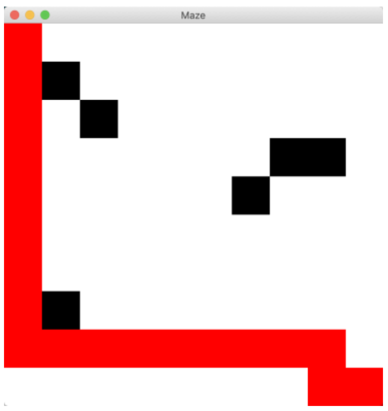


Round 2: Expand 89

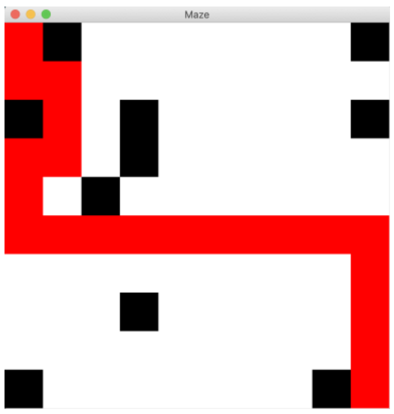


Round 3: Expand 91

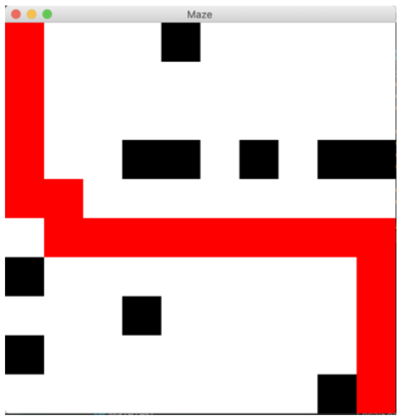
d.iii) Maximum size of fringe during runtime



Round 1: Fringe 45



Round 2: Fringe 43



Round 3: Fringe 44

The longest path option makes all algorithms took the most tortuous way with the number of occupied nodes biggest. The maximum path length is slightly greater than half of the total nodes number.

To maximize the expanded nodes, all algorithms took the diagonal path with fewer occupied blocks. The BFS is an extreme example as there is zero wall node. Thanks to the properties of BFS, under this condition the expanded nodes equal to all the nodes in the maze. A small quantity of wall nodes means more nodes could be expanded, and the diagonal path with some constraints can let the algorithm go through as many neighbors as possible and walk towards the goal. The maximum number of expanded nodes plus the wall nodes are all close to the total number of nodes.

As to the maximum fringe, the DFS goes tortuously because each time it only expands the neighbors along the path. BFS has the least number of fringes because each time the available nodes are the ones close to the visited range having a shape close to a diagonal curve. A* is based on BFS but utilizes heuristic, in this case, they can build a harder maze than BFS, but still not as good as the DFS.

According to the above results, the hardest maze should be better defined as the maze which has the longest path. The max number of expanded nodes would force BFS based algorithms into a tricky way to increase the index rather than build a more complex maze. Since the A* is based on the BFS, we can see they have very similar performances to the BFS.

Work Distribution

Xin Yang: Developed the BFS, A* using Manhattan as heuristic, maze visualization and Local Beam Search.

Zhuohang Li: Developed the DFS, A* using Euclidean as heuristic, naive version of visualization and Hill Climbing, helped debug and optimize the algorithms.