CS 520
Assignment 1
Path Planning and Search Algorithms

October 7, 2018

Adhish Shrivastava, Hiral Nagda, Yash Nisar

Part 1:

Answer 1:

We have implemented DFS/BFS/A* search algorithms on the maze that we generated for different values of p (probability of a cell being occupied) and size (dim X dim). The time representation in below is in seconds.

Algorithm \ dim	10	50	100	250	500	1000
DFS (p = 0)	0	0.03	0.17	1.26	4.36	21.21
DFS (p = 0.1)	0	0.06	0.25	0.81	3.03	16.25
DFS (p = 0.2)	0	0.02	0.12	0.64	2.35	12.98
DFS (p = 0.3)	0	0.03	0.06	0.62	2.5	11.91
BFS (p = 0)	0	0.06	0.23	1.77	7.18	29.76
BFS (p = 0.1)	0	0.05	0.45	2.16	8.16	40.89
BFS (p = 0.2)	0	0.09	0.29	2.12	9.68	43.19
BFS (p = 0.3)	0	0.05	0.25	2.49	10.04	42.64
A* EU (p = 0)	0.02	0.13	0.4	3.42	15.38	68.34
A* EU (p = 0.1)	0.02	0.05	0.41	2.44	14.8	62.01
A* EU (p = 0.2)	0	0.07	0.26	2.26	11.59	54.52
A* EU (p = 0.3)	0	0.08	0.22	1.89	9.34	42.71
A* MH (p = 0)	0	0.11	0.33	2.63	11.79	50.45
A* MH (p = 0.1)	0	0.11	0.35	1.92	10.74	52.89
A* MH (p = 0.2)	0	0.05	0.3	1.27	6.83	24.84
A* MH (p = 0.3)	0	0.03	0.11	0.38	2.01	4.25

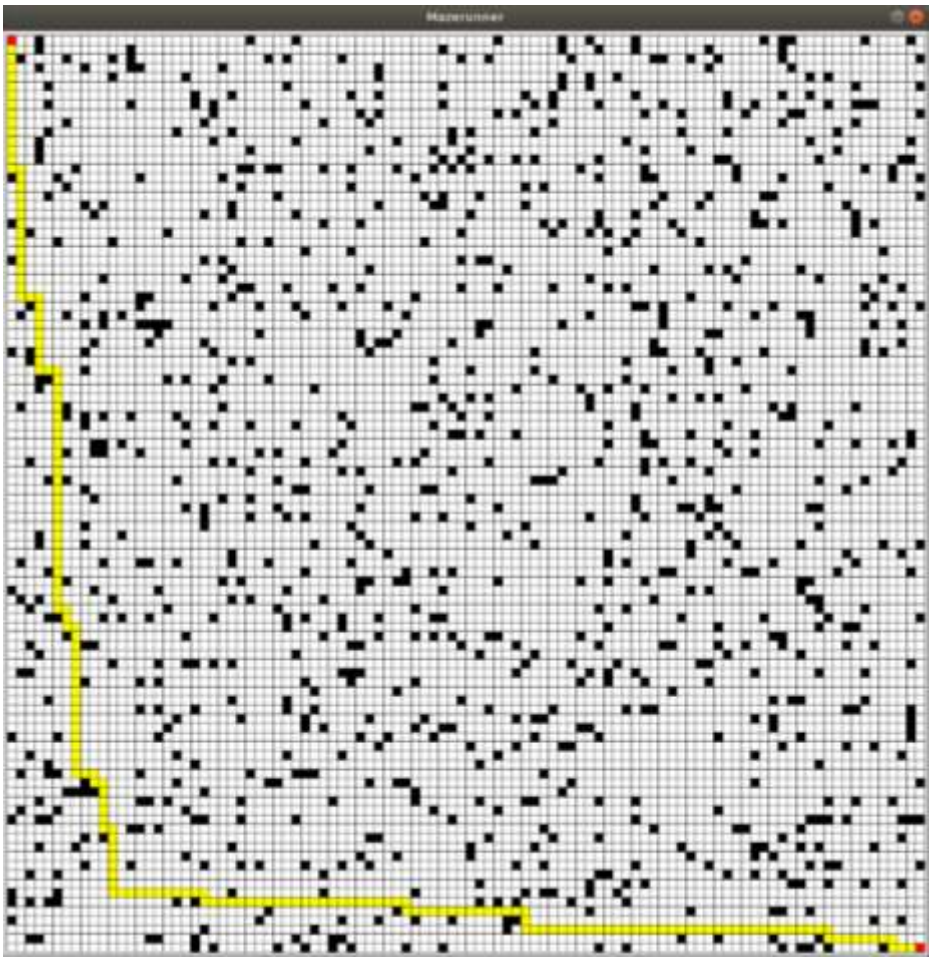
As to measure various parameters of the maze and the algorithms, we needed to generate the maze multiple times and hence found working with a maze of size 100*100 to be reasonable.

Answer 2:

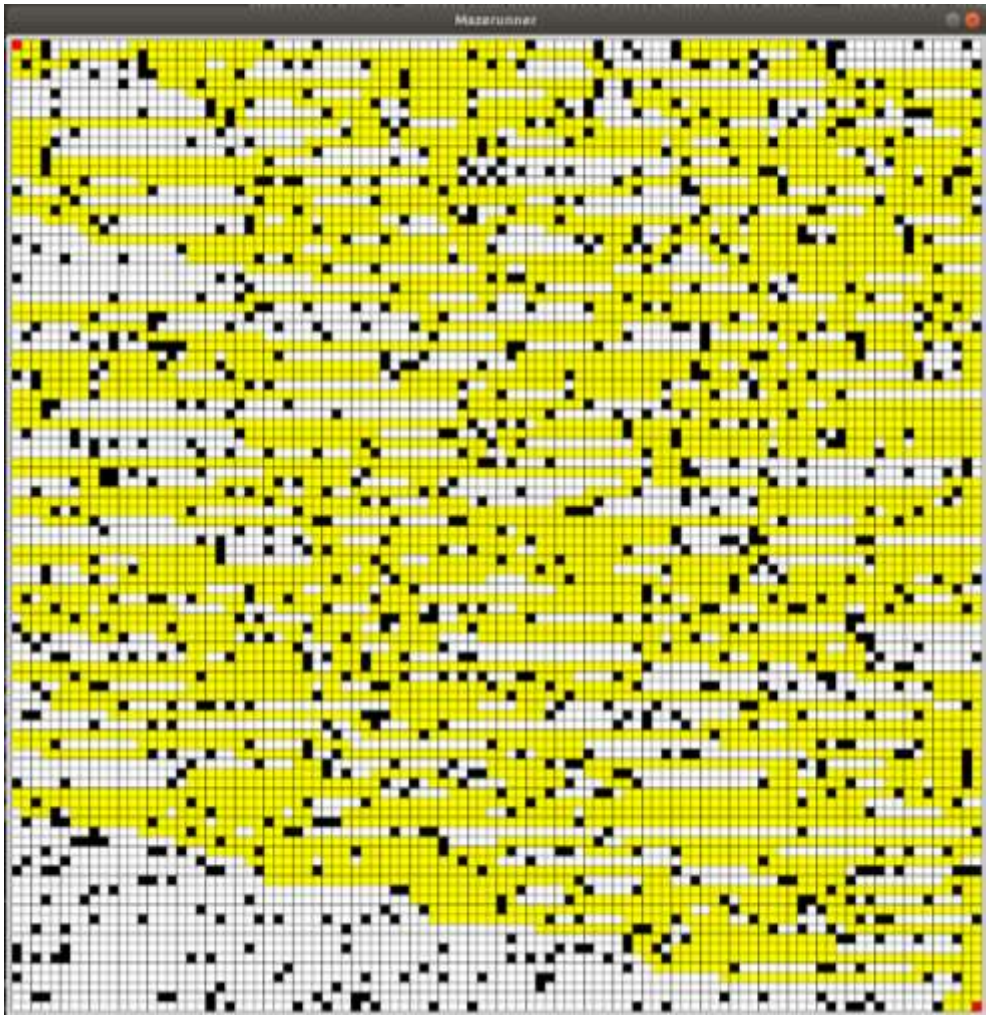
dim: 100

p=0.2

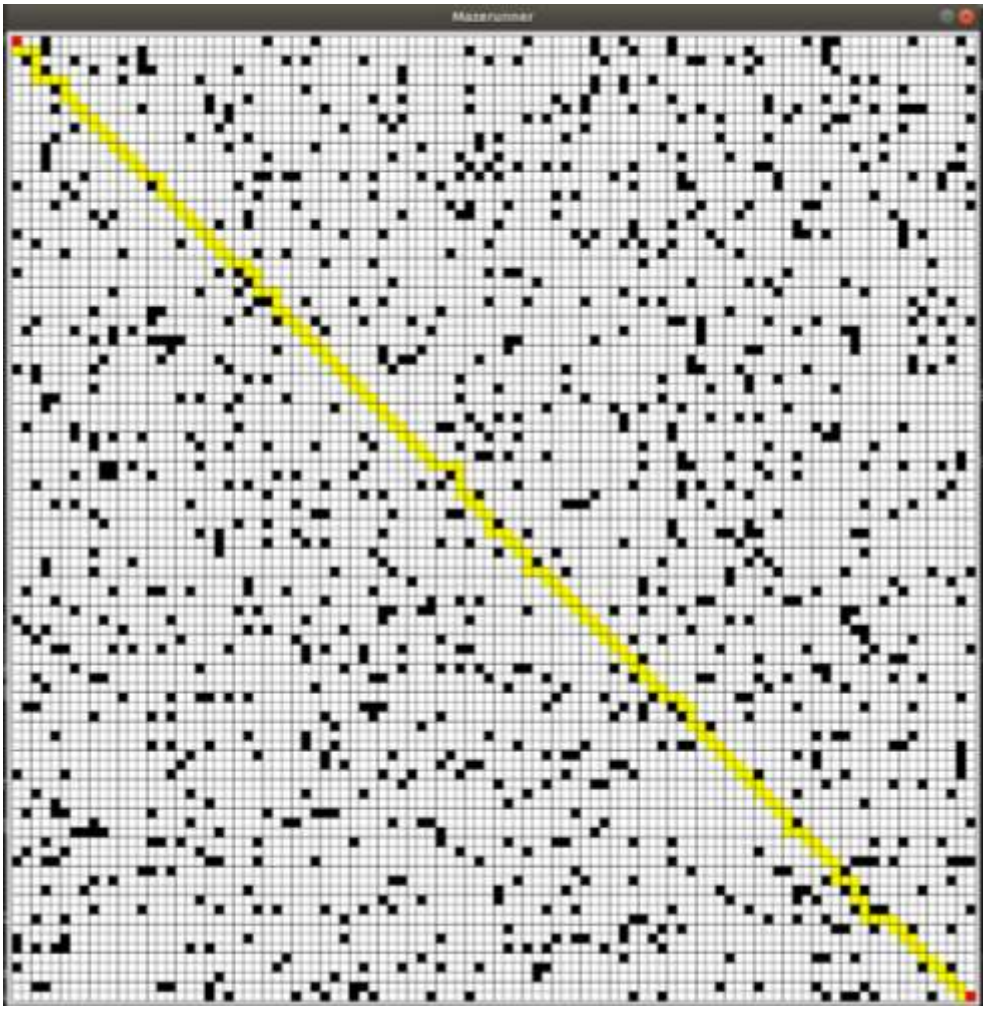
1. Breadth First Search
   Path Length: 197
   No. of Nodes Expanded: 9031



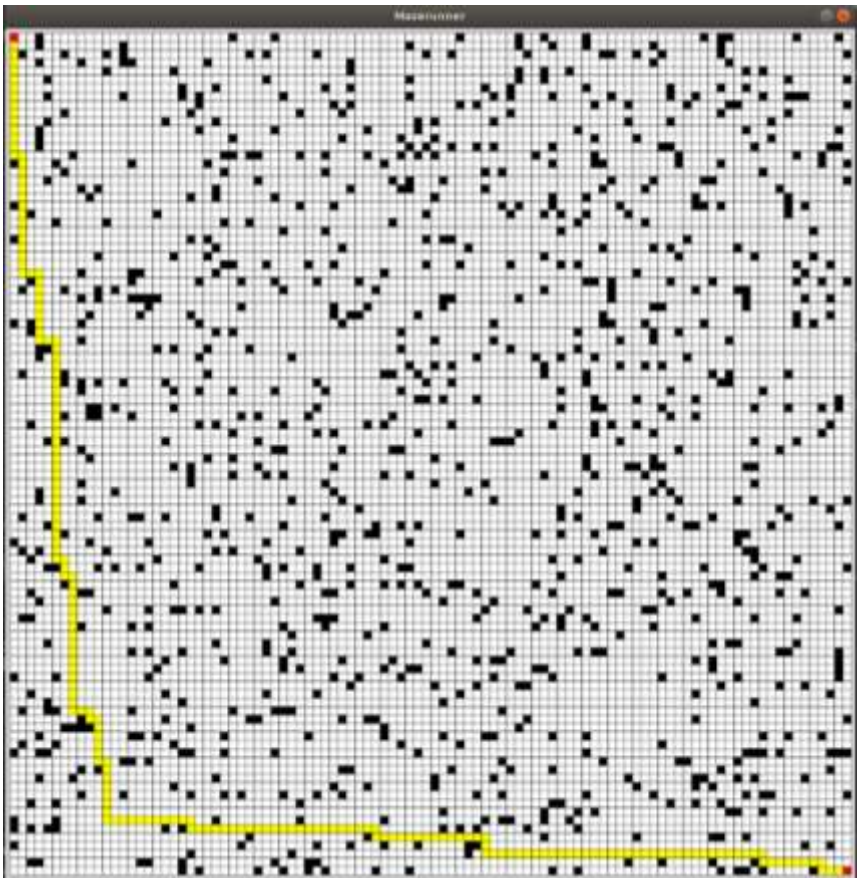
2. Depth First Search
   Path length: 5225
   No. of Nodes Expanded: 7841



3. A* Algorithm (Euclidean Distance)
   Path Length:199
   No. of Nodes Expanded: 8874

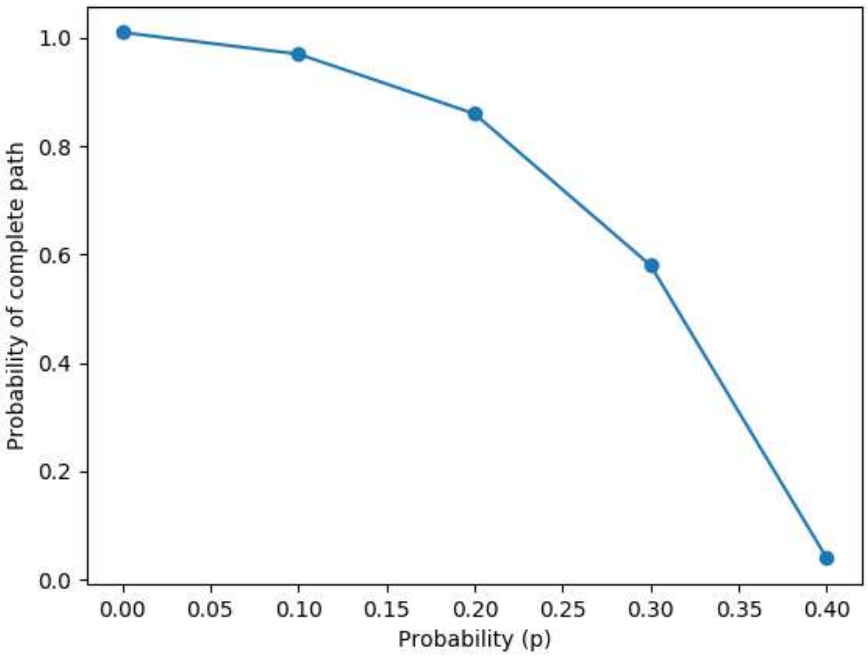


4. A* Algorithm (Manhattan Distance)
   Path Length:199
   No. of Nodes Expanded: 7825



Answer 3:

We generated 100 mazes of dim = 100 and calculated the number of times the path existed. We estimated the probability of path existence for p = 0, 0.1, 0.2, 0.3, 0.4. We generated multiple mazes to get a better average the number of times the path is existing and reduce the chances of inferring result on the basis of particular maze setup. Thus, helping in getting more average data. The estimated probability p* such that a random map generated with p = k where k = {0, 0.1, 0.2, ..., 0.9} are:



At p = 0, 0.1, 0.2, 0.3, 0.4, the number of times that a path from start to the end existed on an average are 100, 97, 86, 58, 4 respectively. Thus, their probability will be 1, 0.97, 0.86, 0.58, 0.04 respectively. We also inferred that for p >= 0.4 the number of obstacles increases, thus filling up the maze more. This usually results that there is no path from start to end for a sample of 100 random selected mazes.

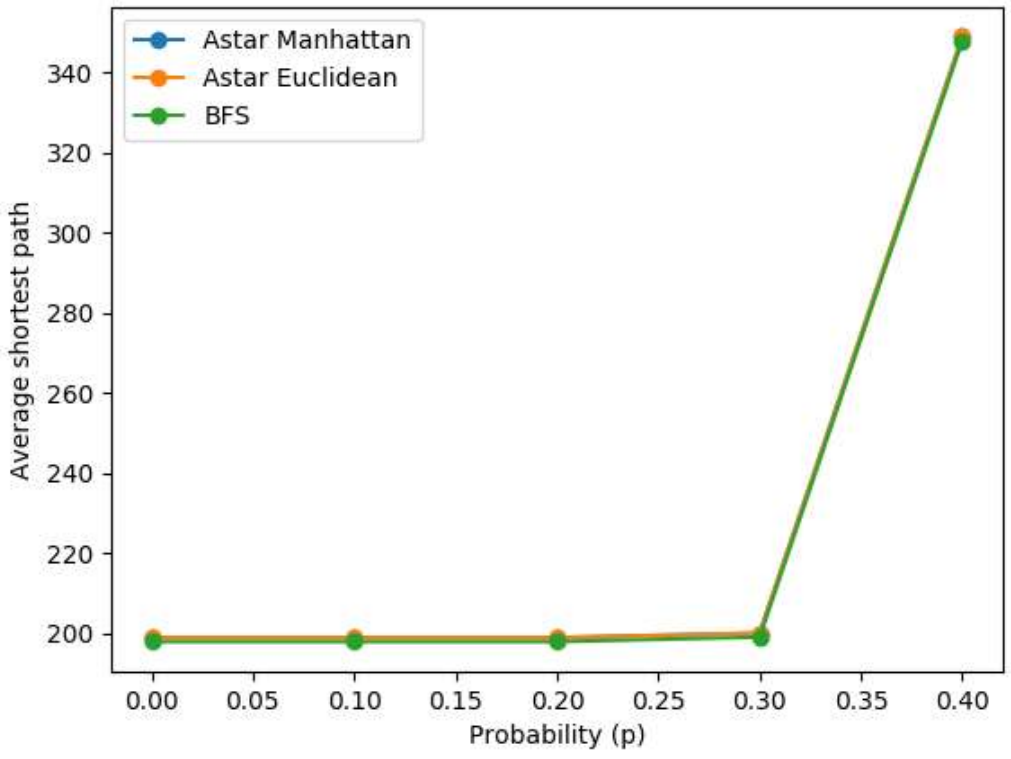
From the graph, we can see that there is a significant change between p = 0.3 and p = 0.4, where it drastically drops down to just 0.04 from 0.58. Thus, we select threshold probability, po, i.e. probability for which the path generally exists to be 0.3. For every $p < p_o$, almost half of the sample cases had a path from start to end.

Answer 4:

For estimating the average length of the shortest path, we repeatedly generated 100 random mazes of dim = 100 and p <= 0.3 and then using these values calculated the average for every p.

	A* Euclidean	A* Manhattan	BFS
p=0	199	199	198
p=0.1	199	199	198
p=0.2	199	199	198
p=0.3	200.14	200.14	199.14
p=0.4	349	349	348

We selected A* as the algorithm for the experiment, firstly as it is optimal and secondly on comparing the average shortest path length for solving the maze using DFS/BFS/A*, we got the shortest path using A*.

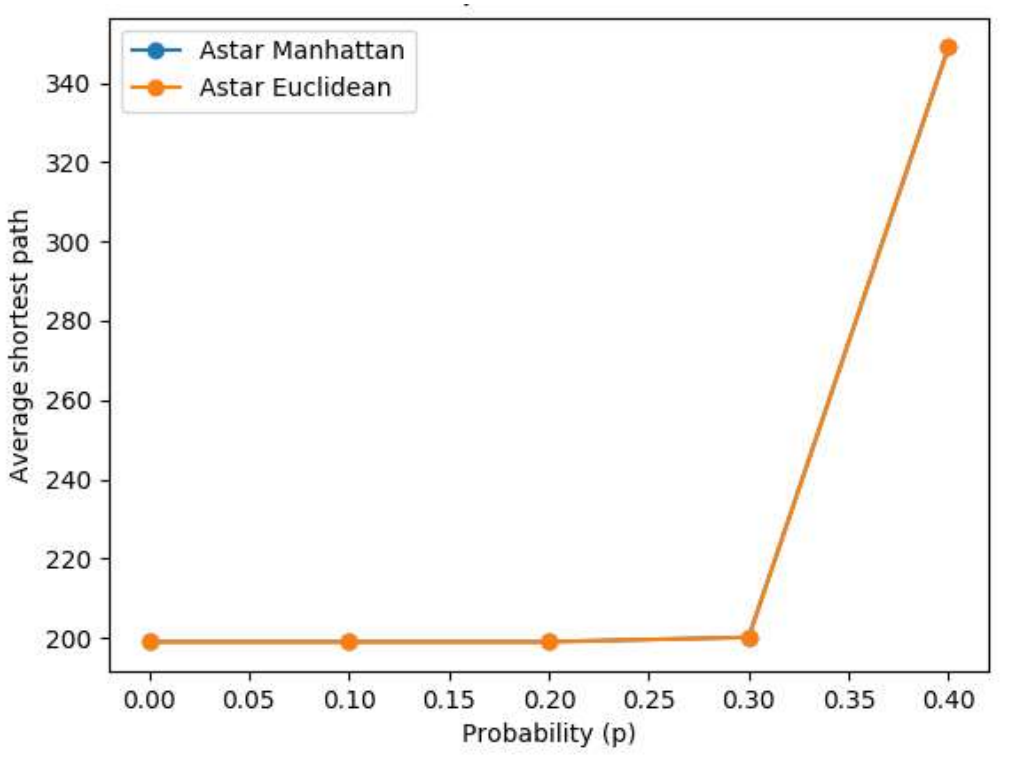


Answer 5:

To better estimate the average path length for A* (Euclidean and Manhattan Heuristic), we generated 100 random sample mazes of dim = 100 and for each probability p, such that p <= 0.3 (p_o). The average data computed for both the heuristics using Euclidean distance and Manhattan distance are as follows:

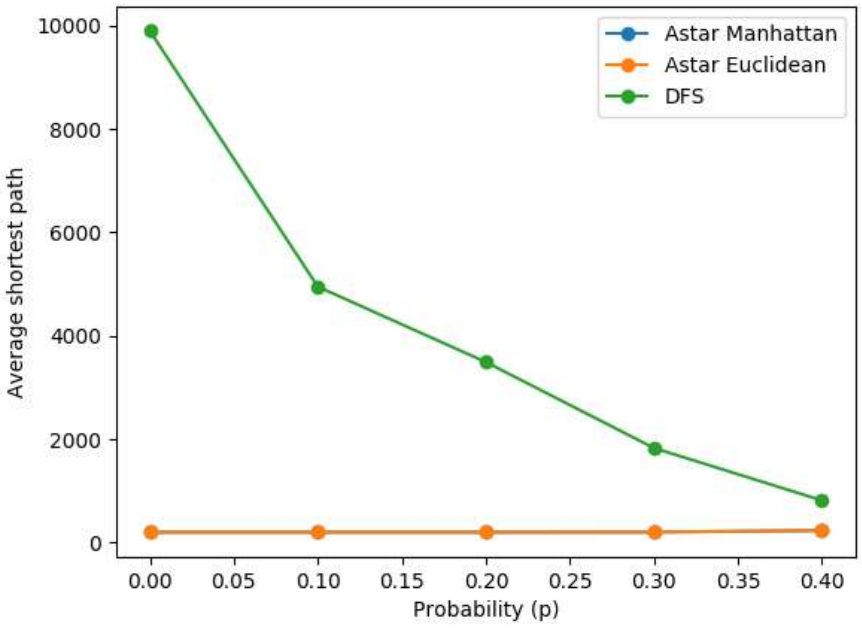
	A* Euclidean	**A* Manhattan**
p = 0	199	199
p = 0.1	199	199
p = 0.2	199.01	199
p = 0.3	200.14	201.14
p = 0.4	349.2	349

From the above data we can infer that for both the A* heuristics, the average path length is almost equal. This inference is due to the fact that both the heuristics find optimal paths in any given environment.



	A* Euclidean	**A* Manhattan**	**DFS**
p=0	199	199	9900
p=0.1	199	199	4949.61
p=0.2	199.072	199.07	3490.70
p=0.3	201.29	201.28	1828.32
p=0.4	231	231	813

In the case of DFS, the average path length is more than that of both the A* heuristics as DFS first finds an available path of out of all the possible paths to solve the maze. The path depends on the arrangement of the obstacles. If the end of the maze is not reached from the first selected path, it expands other nodes and tries again to reach the goal node or stops if the all the nodes have been expanded but the goal node is not possible to reach. DFS comparison with both the A* heuristic is given below:



Answer 6:

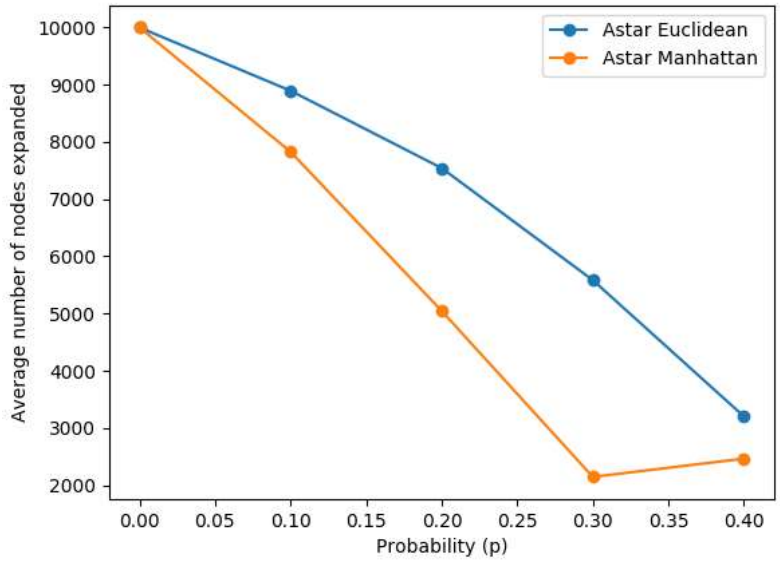
For estimation of a total number of nodes expanded in A* algorithm, we ran the algorithm multiple times (100 iterations) using both heuristics (Manhattan and Euclidean) multiple times for different probabilities. We discarded mazes for which no paths existed.

	A* Euclidean	**A* Manhattan**
p=0	10000	10000
p=0.1	8894.52	7826.75
p=0.2	7542.20	5044.47
p=0.3	5583.27	2144.29
p=0.4	3200.67	2464.17

We observe here that the average number of nodes expanded by A* Algorithm with uses Manhattan distance as heuristic expands less number of nodes compared to the algorithm which uses Euclidean distance as the heuristic.

Both the algorithms give optimal path. However, cost values using Manhattan distance are closer to original cost than Euclidean distance. Euclidian distance under-estimates the cost compared to Manhattan distance according to Pythagorean theorem. Hence, fewer nodes are expanded using Manhattan Distance as a heuristic.

For values of p_o beyond 0.4, the number of nodes explored decreased drastically.

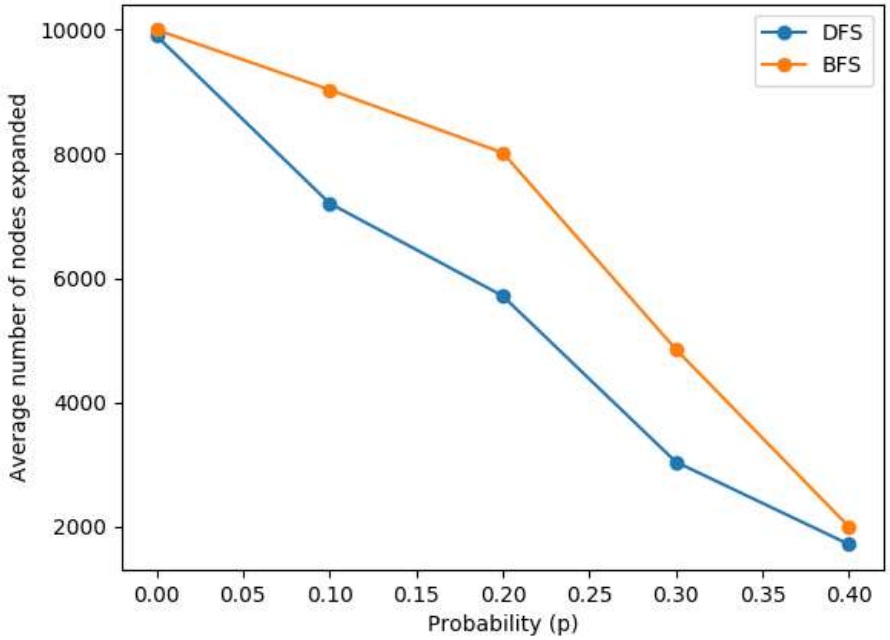


Answer 7:

For getting the number of nodes expanded in DFS and BFS we generated mazes of dim = 100 with different probabilities. We sampled the data for 100 random mazes with discarding the ones which did not have a possible path from start to the end. The findings of an average number of nodes expanded for DFS and BFS are as follows:

	DFS	**BFS**
p = 0	9900	9999
p = 0.1	7204.4	9032.9
p = 0.2	5712.8	8018.1
p = 0.3	3040.9	4853.8
p = 0.4	1713.2	1995.8

It is evident that DFS expands less nodes when compared to BFS for solving the same maze. DFS goes down in the search tree and returns the first path it finds. In this process, it visits all the nodes closer to the goal before it backtracks to start expanding nodes to try for a different possible path to reach the goal node. In BFS all the nodes of the same level are first expanded and only after that we move to the next level. As the probability increases or in other words as the obstacles increases, BFS encounters more blocked path, where it skips that fringe and therefore it expands lesser nodes when p increases. A graph showing the comparison:



We know that A* is optimal, i.e. it visits nodes that have more chances of being on the shortest path by using heuristic functions. In an average case, A* expands more nodes as compared to DFS which searches by going to lower levels of the tree. After completion of one path, DFS has explored more nodes near to the goal node.

In BFS all the nodes of the same level are expanded first before it can go to the next level, whereas in A* there is no constraint that it needs to explore all the nodes in the level before it can go to the next level. In A* it follows the heuristic functions to move to nodes that are more likely to be on the shortest path to the goal node. Hence, BFS expands more number of nodes when compared to A* for different probabilities on a dim 100 maze.

Bonus question: Why were you not asked to implement UFCS?
Uniform cost search algorithm does not use any heuristic (hence does not use information about goal). In the current scenario for each movement from one tile to another adds a cost of 10. It explores possibilities in all directions. Hence it becomes same as BFS.

Part 2:

Answer 8:

We have used Genetic Algorithm to build hard mazes. We chose the Genetic Algorithm in particular because we're given an optimization problem of maximising the given properties like: Length of solution path returned, Total number of nodes expanded, Maximum size of fringe during runtime. They do not require any derivative information and are faster as compared to other local search algorithms.

We have represented the maze as an Ordered Dictionary of the co-ordinates (X & Y) denoting the keys and a Boolean value (True or False) as the value in the key-value pair. The Boolean value indicates whether the cell is obstructed or not.
E.g. ((5, 5), True)

Our initial population consisted of 100 mazes that definitely have a path from the source to the destination, sorted according to the fitness parameter (Length of solution path returned, Total number of nodes expanded, Maximum size of fringe during runtime). Selection was performed based on the maximum values of the fitness parameter and allotted those mazes as parents (mother and father) respectively. Then crossover was performed by choosing the first half of the maze from mother and second half of the maze from father resulting in the formation of the child. The fitness function for the child was computed and the maze with the least fitness function was removed from the population. This was performed for 50 iterations and the values seemed to converge.

Answer 9:

Yes, it was kind of difficult to know when we had constructed the hardest maze. After performing the experiment multiple times, we concluded that 50 iterations were a good fit because it started to converge. The 'hardest' maze would include of exploring all possible cells in the grid and the destination would be the last cell searched. Since our algorithm randomly generated mazes, it is very difficult to generate the hardest possible maze where each and every node would be explored. The termination condition can be convergence, i.e. we can stop this process when the new generated children have fitness functions that are almost comparable to the population. One such drawback of using genetic algorithm can be premature convergence where we hit a local optimum.

Answer 10:

We initially thought of setting priorities for expansion (right, down, left, top), but that would work specifically for the case where we have some information about the source and the destination, i.e. the source is at the top left corner and the destination is at the bottom right corner. To keep the algorithm generic, we've not set priorities for expansion of neighbours. The mazes generated were hard but not the hardest because the probability of obstruction was relatively low, giving rise to more free cells rather than blocked cells.

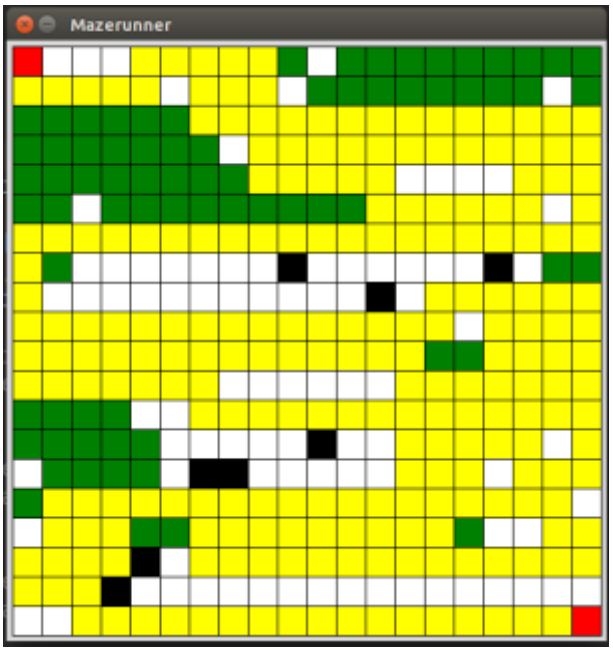
Depth First Search: The DFS solutions did not guarantee a shortest path from the source to the destination. The hard solutions consisted of exploration and traversal of a lot of cells. Any obstruction would enforce the algorithm to backtrack and thus take a longer route. The hardest solution would consist of going down, one cell to the right, going up, one cell to the right, going down and so on.
Breadth First Search: The BFS solutions guaranteed a shortest path from the source to the destination. Since BFS is a level by level traversal, the entire maze is searched and therefore almost all the cells are explored. We noticed that almost all the solutions had the same length of the shortest path, since there are multiple possibilities available.

A* Algorithm: We observed that when number of nodes explored was taken as fitness function we got the hardest maze as number of nodes explored and path length was highest for both the heuristics (Euclidean Distance and Manhattan Distance).

Depth First Search (DFS)

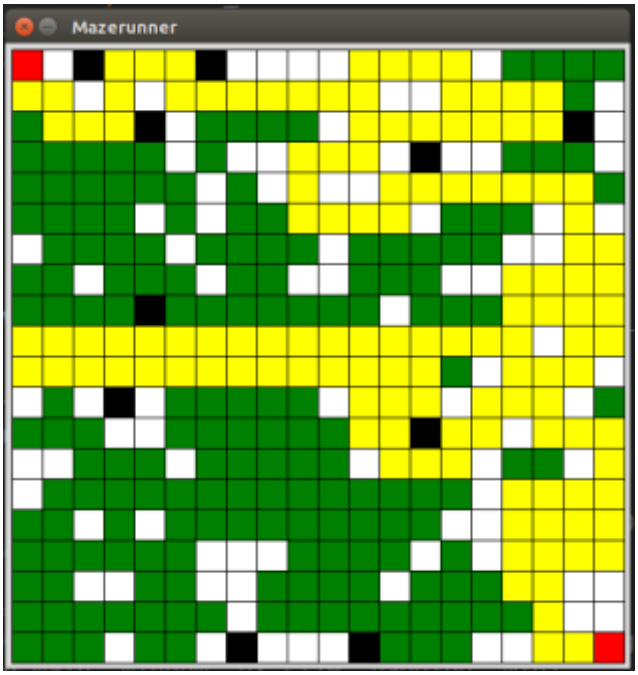
Fitness function: Length of solution path returned

Length of solution path returned	230
Total number of nodes expanded	303
Maximum size of fringe during runtime	238



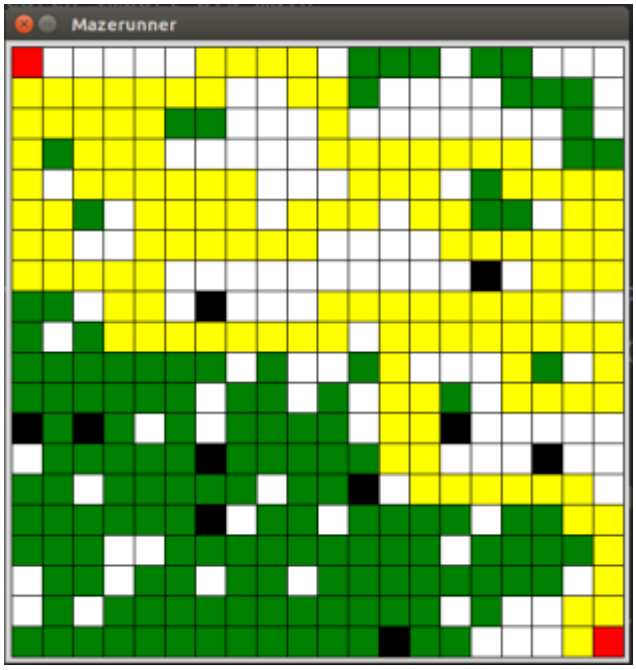
Fitness function: Total number of nodes expanded

Length of solution path returned	128
Total number of nodes expanded	301
Maximum size of fringe during runtime	146



Fitness function: Maximum size of fringe during runtime

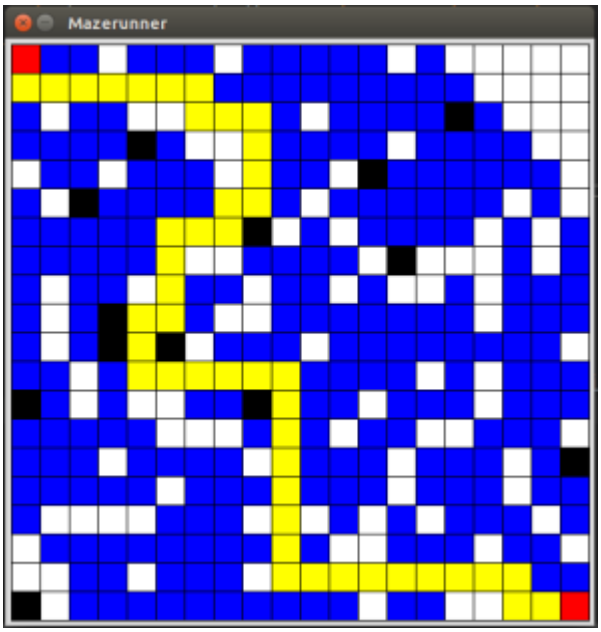
Length of solution path returned	132
Total number of nodes expanded	272
Maximum size of fringe during runtime	136



Breadth First Search (BFS)

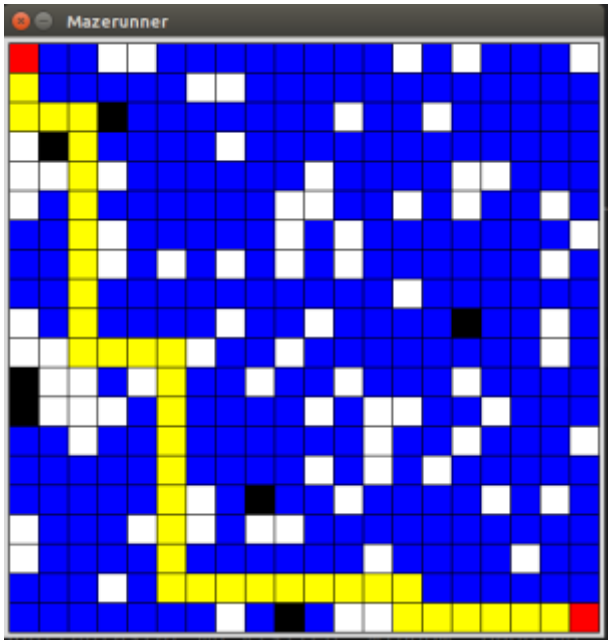
Fitness function: Length of solution path returned

Length of solution path returned	46
Total number of nodes expanded	283
Maximum size of fringe during runtime	15



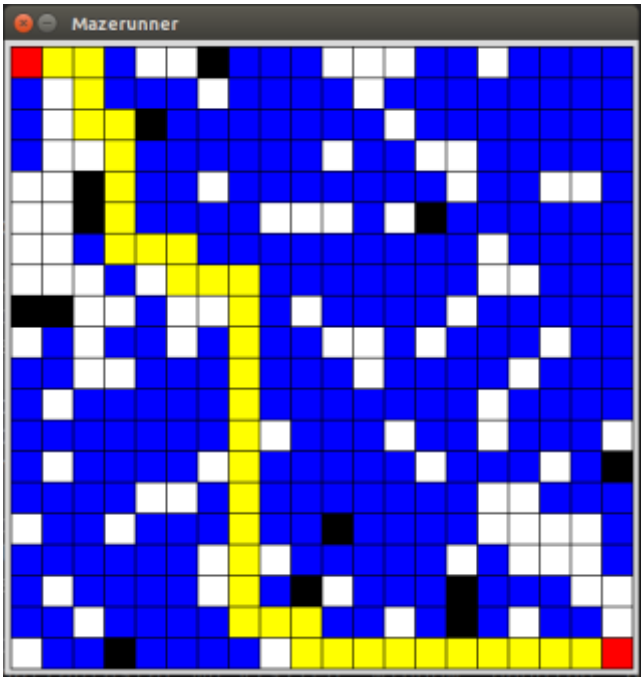
Fitness function: Total number of nodes expanded

Length of solution path returned	38
Total number of nodes expanded	313
Maximum size of fringe during runtime	23



Fitness function: Maximum size of fringe during runtime

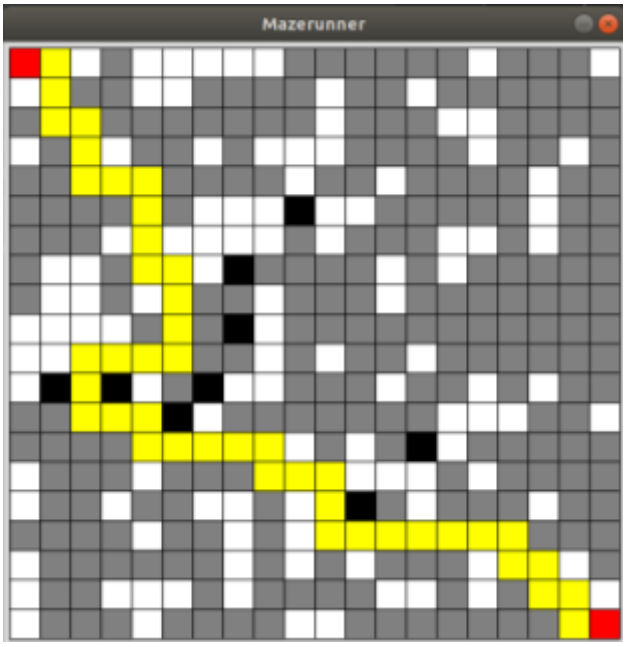
Length of solution path returned	38
Total number of nodes expanded	295
Maximum size of fringe during runtime	26



A * with Euclidean Distance Heuristic

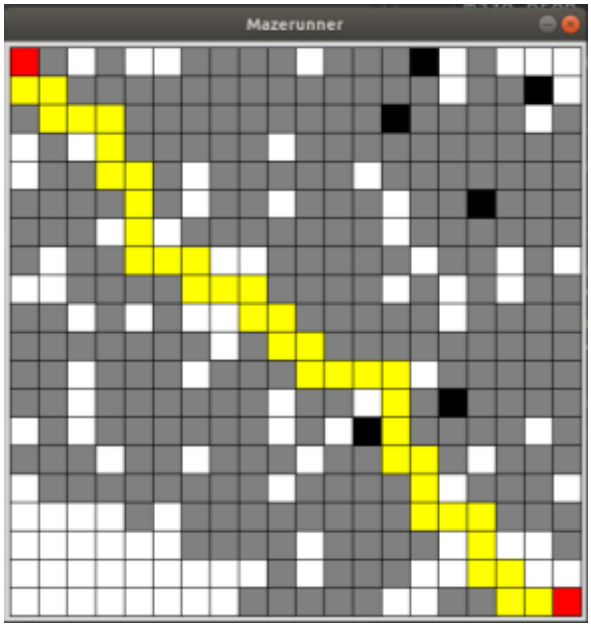
Fitness function: Length of solution path returned

Length of solution path returned	45
Total number of nodes expanded	268
Maximum size of fringe during runtime	43



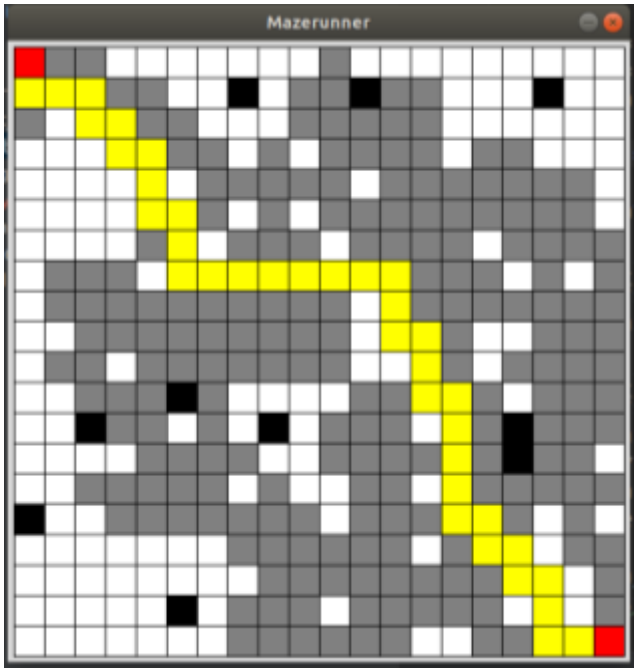
Fitness function: Total number of nodes expanded

Length of solution path returned	39
Total number of nodes expanded	285
Maximum size of fringe during runtime	29



Fitness function: Maximum size of fringe during runtime

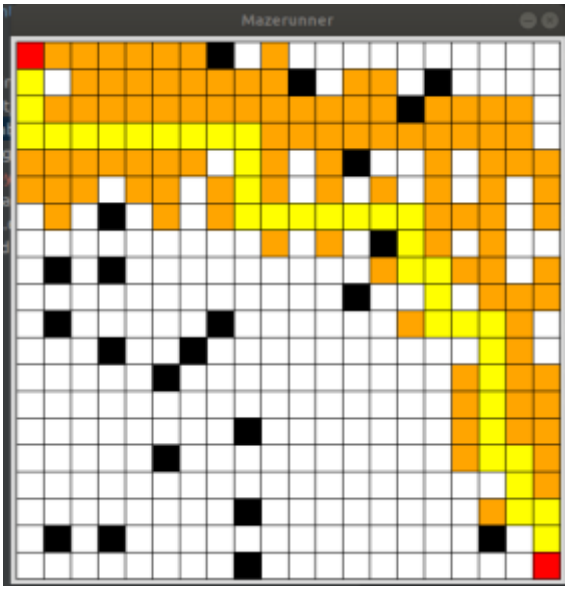
Length of solution path returned	39
Total number of nodes expanded	225
Maximum size of fringe during runtime	41



A* with Manhattan Distance Heuristic

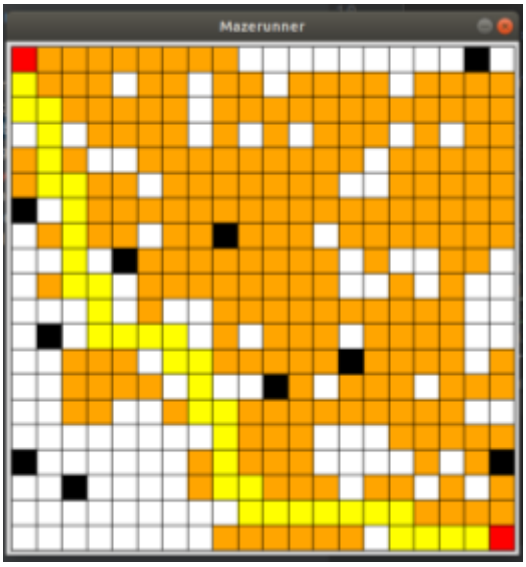
Fitness function: Length of solution path returned

Length of solution path returned	39
Total number of nodes expanded	124
Maximum size of fringe during runtime	25



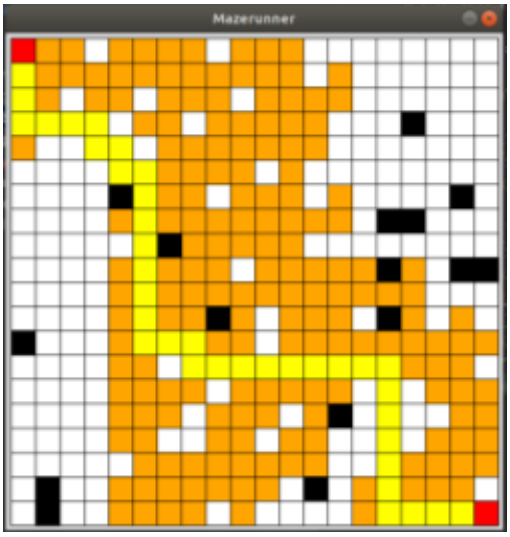
Fitness function: Total number of nodes expanded

Length of solution path returned	39
Total number of nodes expanded	232
Maximum size of fringe during runtime	40



Fitness function: Maximum size of fringe during runtime

Length of solution path returned	39
Total number of nodes expanded	188
Maximum size of fringe during runtime	36



Work Division

Adhish Shrivastava: Breadth First Search, Maze Generation, Report

Hiral Nagda: A* Algorithm using Manhattan and Euclidean Distance heuristic, Analysis, Report

Yash Nisar: Depth First Search, Genetic Algorithm, Report