Report -Assignment 1-Maze Runner

Sharvani Pratinidhi (NetID: spp133)
Amit Patil (NetID: amp508)
Pranita Eugena Burani (NetID: peb63)

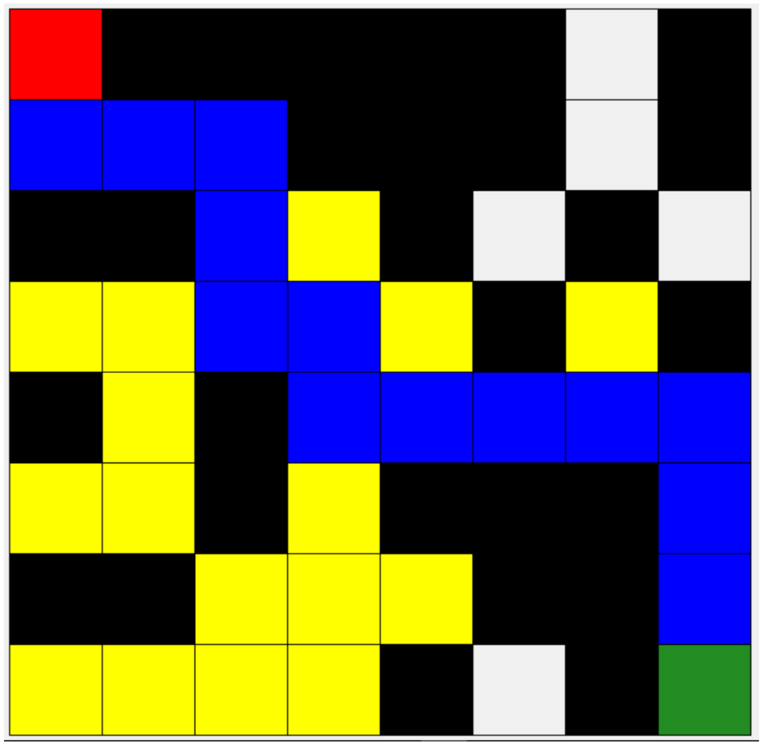
1) Environments and Algorithms

We have used the following conventions in the assignment:

- (*dim x dim*) is the size of the maze
- *p* is probability of each cell to be an obstacle

A sample Maze is provided below (dim=8, prob =0.4, BFS)
The same representation has been used throughout the assignment

- The start cell is represented with red color
- The empty cells are white in color
- Obstacles are black in color
- Nodes visited by algorithm while solving the path are yellow in color
- The final path is represented with blue
- The goal cell is in green

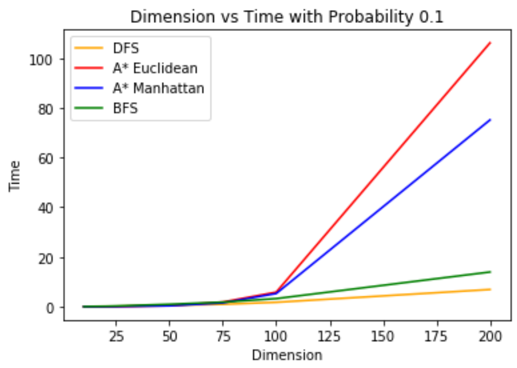


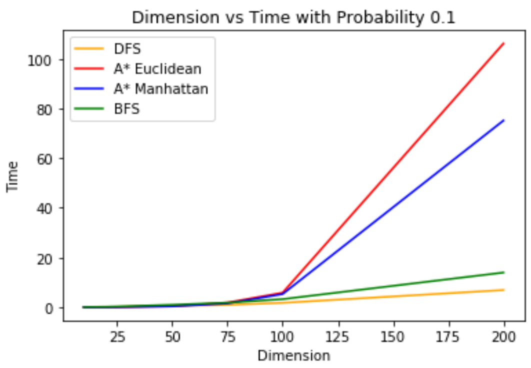
2) Analysis and Comparison

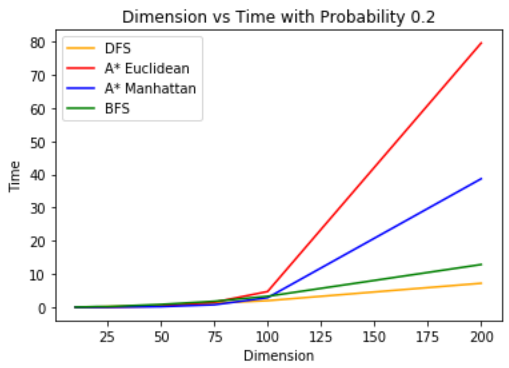
• *Find a map size (dim) that is large enough to produce maps that require some work to solve, but small enough that you can run each algorithm multiple times for a range of possible p values. How did you pick a dim?*

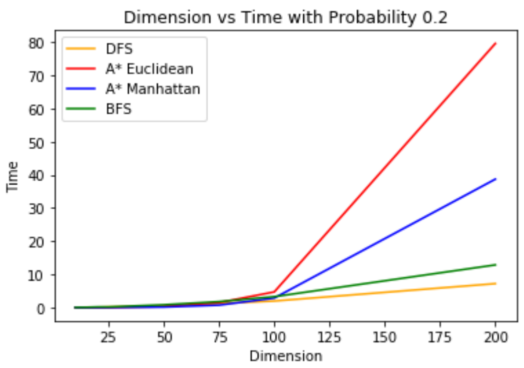
To pick a *dim*, large enough to produce solvable mazes, we generated random mazes for different probabilities ranging in (0.1-0.5), and found out that beyond 0.4, the mazes were unsolvable.

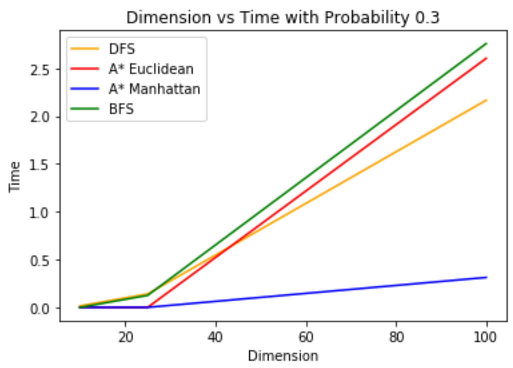
On plotting *Dim vs Time(mean)* graphs for each algorithm (BFS, DFS, A* with Manhattan distance and Euclidean distance as heuristic), we found that for probability up to 0.2, mazes of *dim* 200 can be solved in a maximum of 100 seconds, and for probability 0.3 solvable mazes were generated only up to *dim* 100. (Graphs plotted can be seen below). Considering these, max *dim* can be set as 200 produce solvable mazes. However, we consider *dim* as 50 in our experiments as they are both challenging and solvable in a small amount of time.

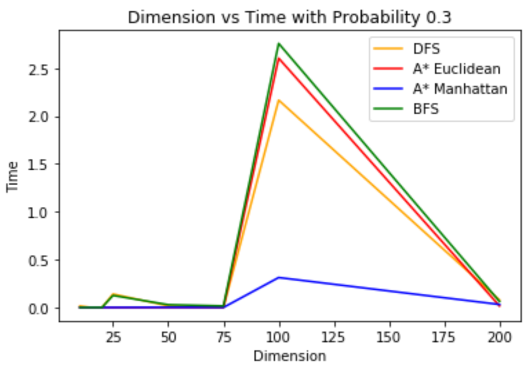










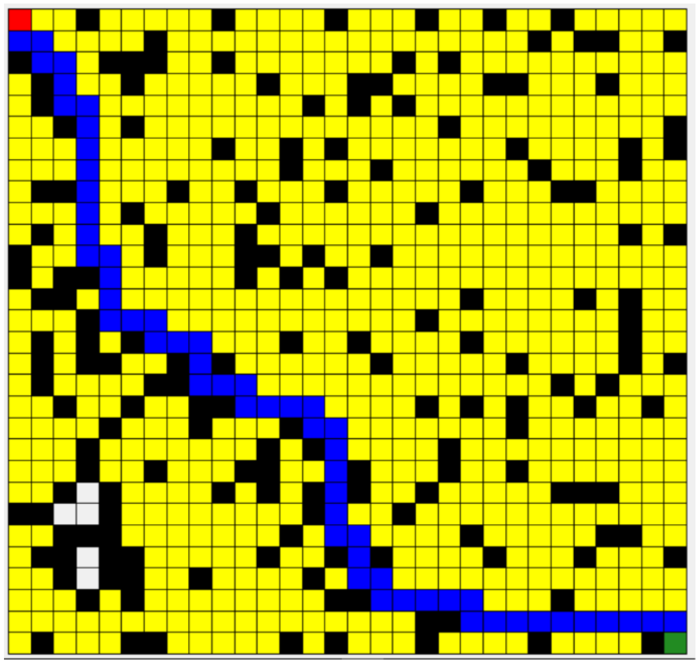


The graphs in the left are generated only for solvable mazes, whereas the ones in the right are for both solvable and unsolvable, the time taken on the y- axis is the mean time of 100 observations.
The time for 0.3 prob, is decreasing in the right graph because of these unsolvable mazes

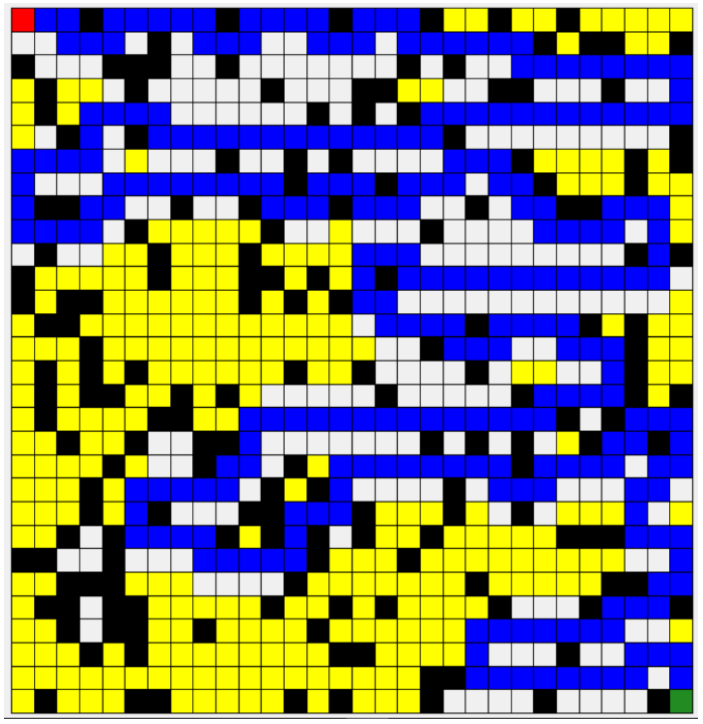
• *For p≈0.2, generate a solvable map, and show the paths returned for each algorithm. Do the results make sense? ASCII printouts are fine, but good visualizations are a bonus.*

Dim=30, prob = 0.2

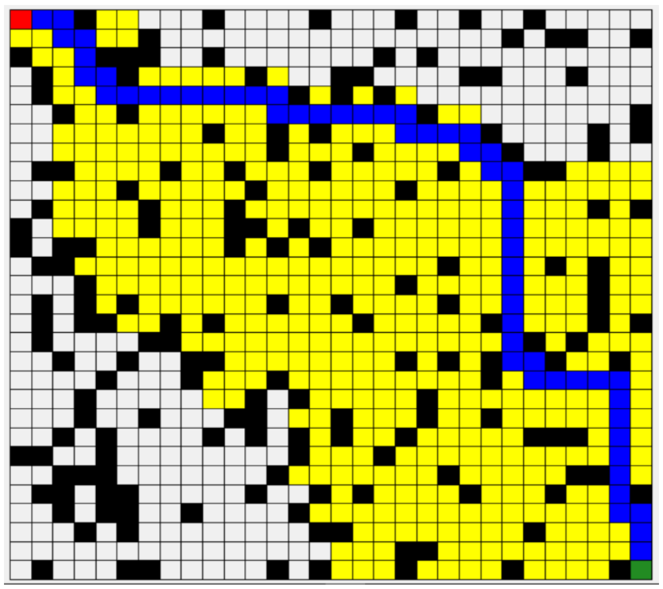
BFS



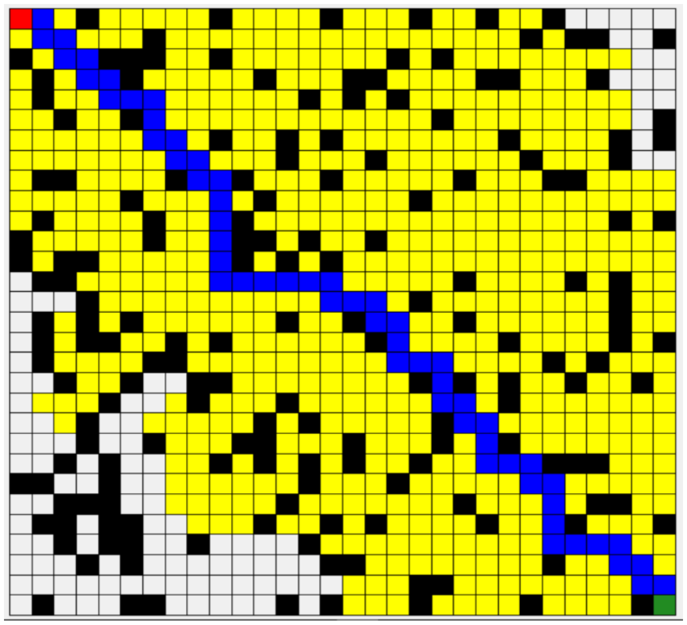
DFS



A*- Manhattan



A* - Euclidean



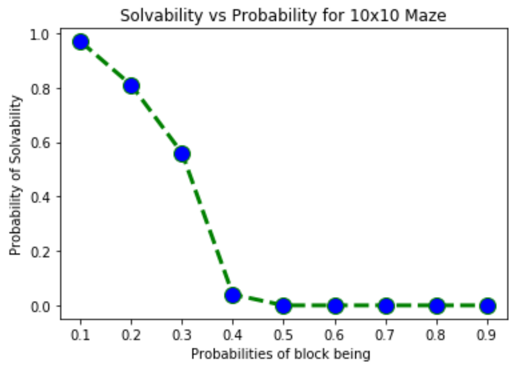
From the results of the paths returned from each algorithm the following things can be observed

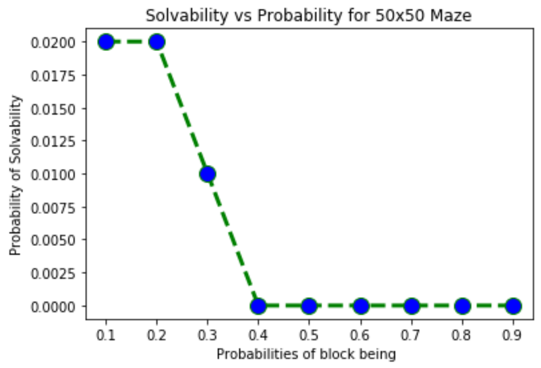
1. BFS returned the optimal shortest path
2. BFS visited every reachable node to find the optimal solution
3. DFS found a solution before finding an optimal solution and returned the non-optimal path
4. Both A* also explored nodes layer by layer but they used heuristics to reduce the number of nodes visited.

   The above results can be easily observed from the mazes generated and they make absolute sense because they are working in line with the theory and confirm the fact that our algorithms are working correctly.

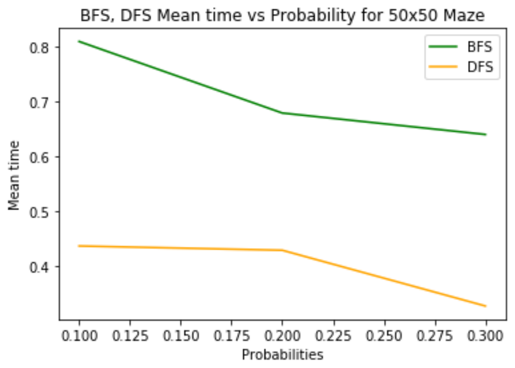
• *Given dim, how does maze-solvability depend on p? For a range of p values, estimate the probability that a maze will be solvable by generating multiple mazes and checking them for solvability. What is the best algorithm to use here? Plot density vs solvability and try to identify as accurately as you can the threshold p_0 where for $p < p_0$, most mazes are solvable, but $p > p_0$, most mazes are not solvable.*

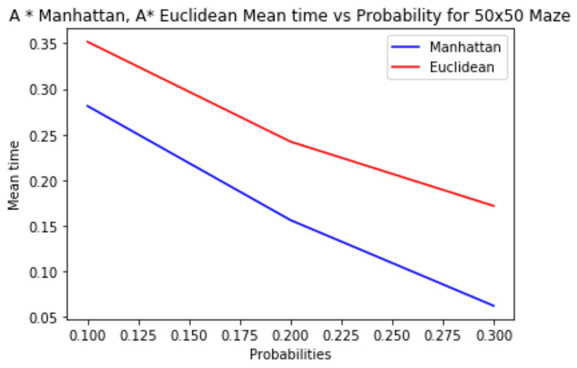
For this, we chose *dim* to be 50 and generated random mazes for different values of *p* (0.1 – 0.9) and applied the algorithms of BFS, DFS, A* with Manhattan distance and Euclidean distance heuristics. The number of solved mazes pitted against probability values gives the dependence of solvability on probability. These results were plotted as a graph and threshold probability(p_0) is found to be **0.4** . Two plots one for maze *dim* 10x10 and 50x50 are shown below, and though there is the difference in the solvability, the threshold probability remains the same.





To find the best algorithm, we plotted a graph, *Runtime against Probability* for each algorithm and found -A*-Manhattan to be the best algorithm in run-time, because all we need to find is whether there is a path or not, optimal path doesn't matter. The A * Algorithm being a complete algorithm, guarantees that if the maze is solvable, it will solve it and thus serves the purpose.



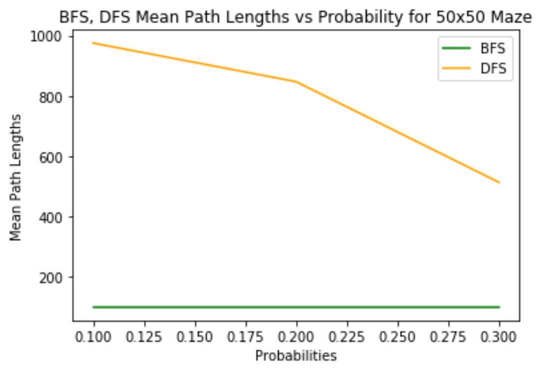


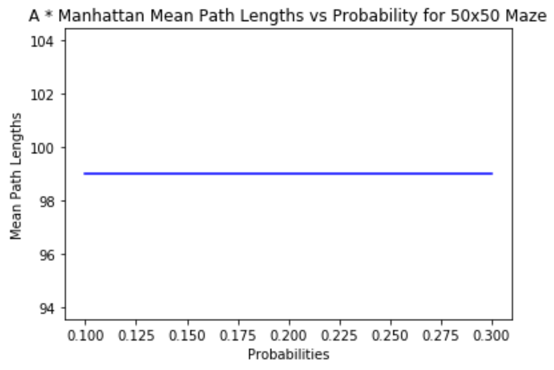
• *For p in [0, p_0] as above, estimate the average or expected length of the shortest path from start to goal. You may discard unsolvable maps. Plot density vs expected shortest path length. What algorithm is most useful here?*

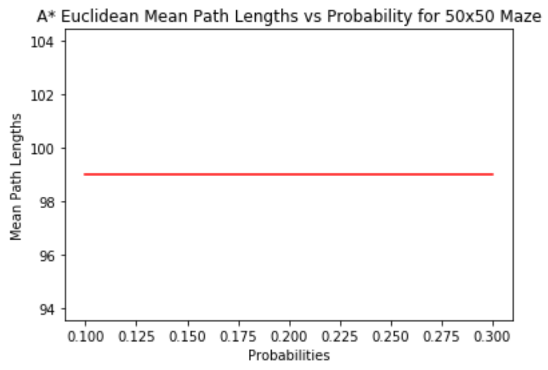
i) Graph is plot between Density and average shortest path length for all four algorithms and from the results it is found that **BFS, A*(**both Manhattan distance and Euclidean distance heuristic) are useful algorithms because they return optimal path.

ii) Also, to further choose the best, Runtime vs Density graph is plot for each algorithm and from the results of these two plots **A* with Manhattan distance heuristic** is the **most useful algorithm** because it returns both least short path to solve the maze and takes the least amount of time.

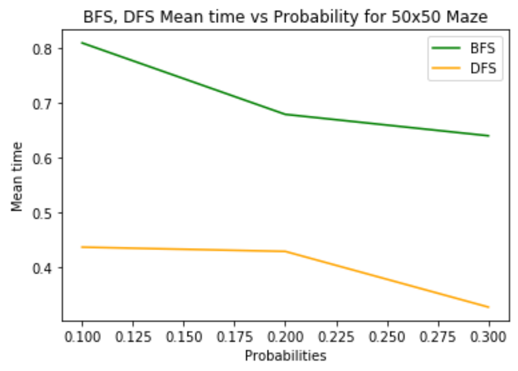
Average Shortest Length vs Probabilty

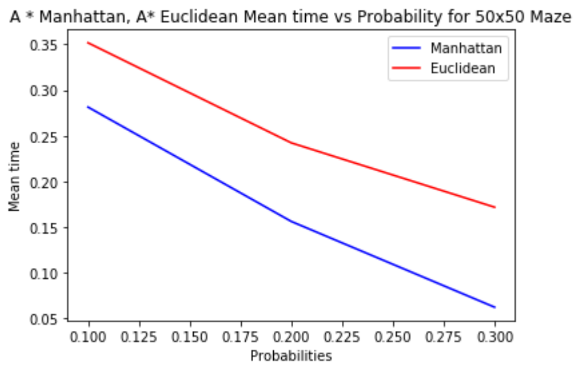






Time vs Probability



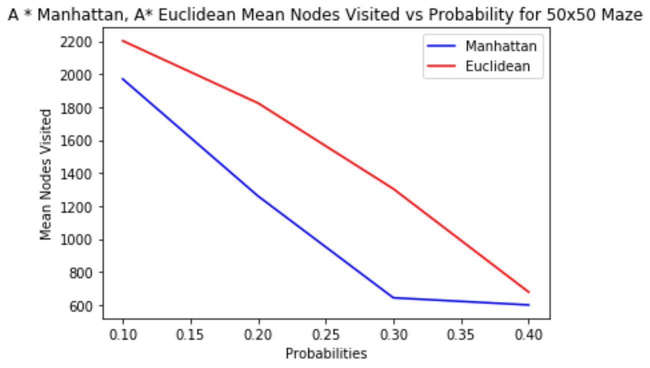


• *Is one heuristic uniformly better than the other for running A^* ? How can they be compared? Plot the relevant data and justify your conclusions.*

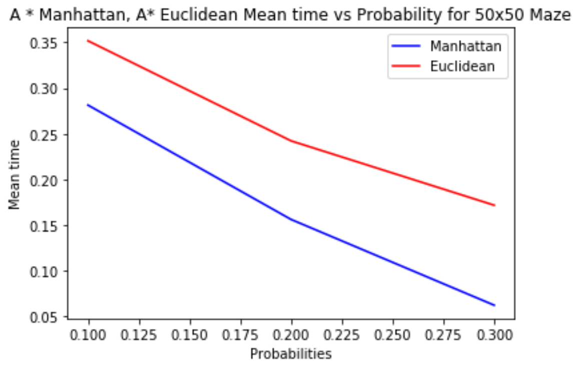
We know that any Heuristic which gives an estimate close to actual path usually has better performance compared to other heuristics.
In this assignment, for solving mazes, we found that A*- with Manhattan distance heuristic is much better than A*-with Euclidean distance heuristic. Because Manhattan distance is more aligned with the natural property of the maze whereas Euclidean distance is more suitable for continuous space. In these maze solving problems Euclidean underestimates, the cost of the path, compared to Manhattan.
To confirm this, analysis is made using the numbers of nodes visited by each algorithm from these results it proves that **A* with Manhattan distance heuristic is better than A* with Euclidean distance heuristic** to

solve mazes. This is because Euclidean distance prioritized the path based on the heuristic around the middle diagonal line. Most of the nearby nodes on top and bottom of the diagonal are explored which leads to the extra number of nodes when compared to Manhattan distance. This can be clearly seen from visualization of solved mazes using Euclidean distance and Manhattan distance as heuristics. Plots drawn between number of nodes visited and probabilities also prove this point.
Also, A* with Manhattan distance heuristic takes less time compared to A* with Euclidean distance heuristic, this is obvious because Euclidean distance heuristic takes time exploring extra nodes.
Therefore, Manhattan distance is a better heuristic for solving mazes.

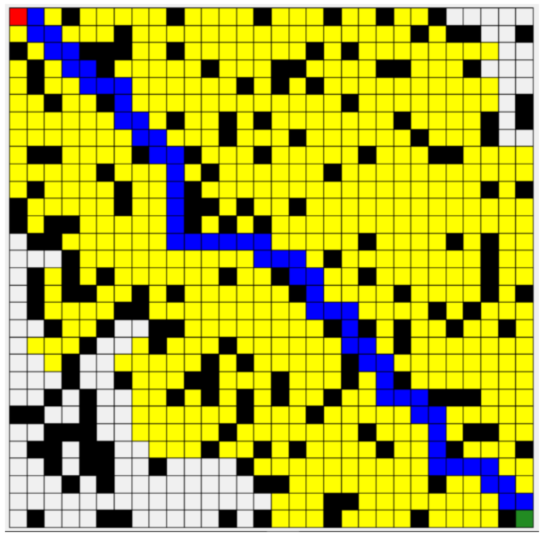
Nodes vs Probability



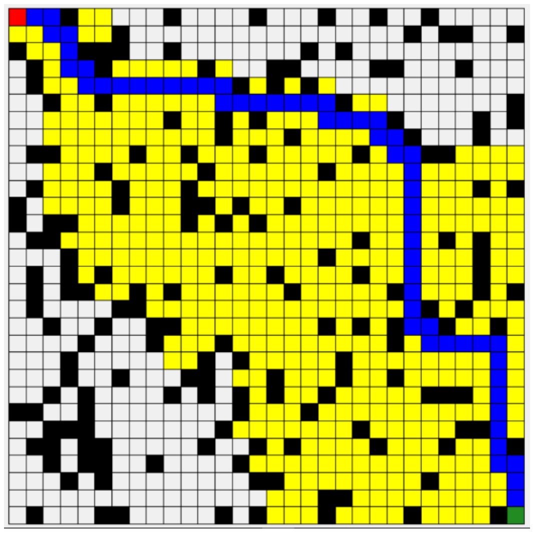
Time vs Probability



A*- Euclidean



A*-Manhattan

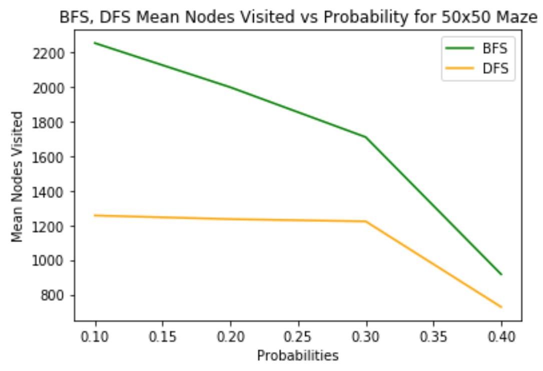
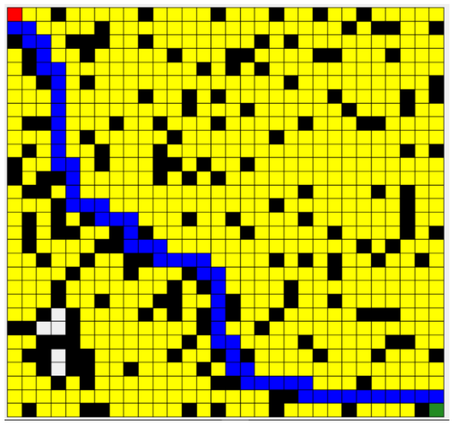
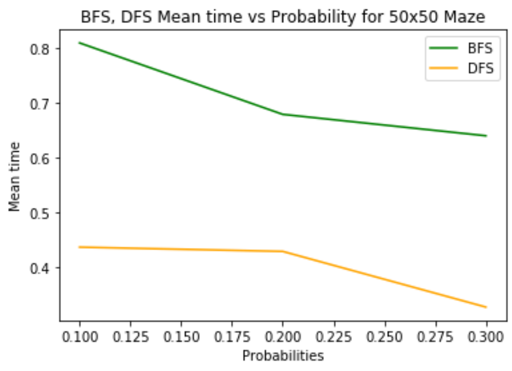
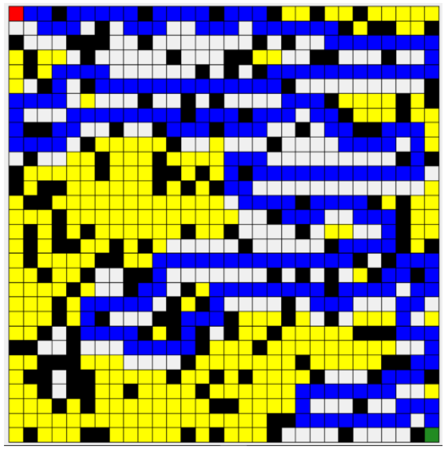


• *Is BFS will generate an optimal shortest path in this case - is it always better than DFS? How can they be compared? Plot the relevant data and justify your conclusions.*

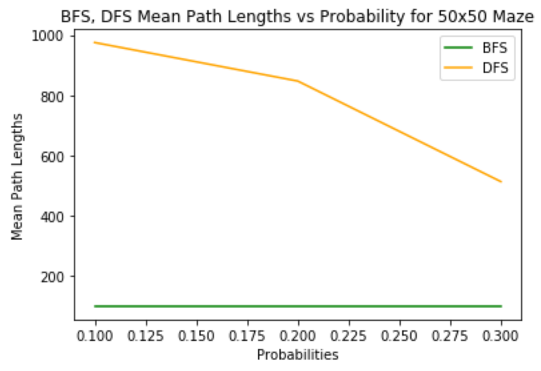
BFS expands all nodes reachable for each layer. Even though this property ensures to find the optimal path it tends to visit every reachable cell in the map, increasing the cost of finding the path.
DFS on the other hand may find a non-optimal solution before it finds the optimal. This algorithm also ensures to have less cost than BFS.
Depending on the Requirement

- To find the shortest path?
- To find path in lesser time and with less cost?

One algorithm will prove to be better useful than the other. To compare BFS and DFS following plots can be useful:

Nodes visited vs probability 	BFS 
Time taken vs probability 	DFS 

Average length vs probability

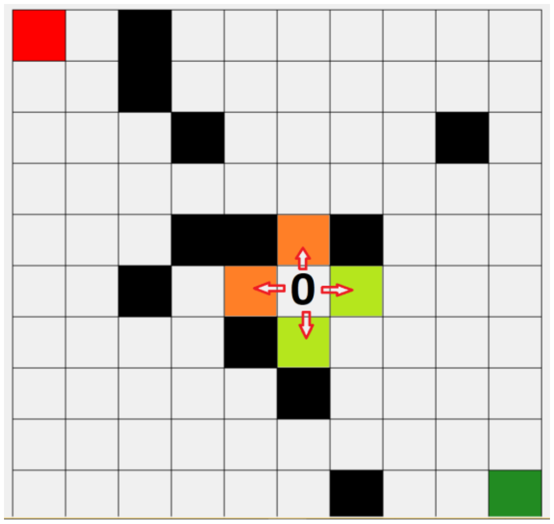


- *Do these algorithms behave as they should?*

From the graphs all these points can be concluded about the algorithms and the results show that they behave exactly how they should.
BFS – visits all nodes, return the optimal shortest path, takes more time
DFS – visits less nodes compared to bfs, Greater shorter path, takes less time
A*(M) – visits lesser nodes than Euclidean, returns optimal short path, takes the least time
A*(E) – visits more nodes, returns optimal shortest path, takes more time

- *For DFS, can you improve the performance of the algorithm by choosing what order to load the neighboring rooms into the fringe? What neighbors are 'worth' looking at before others? Be thorough and justify yourself.*

The performance of DFS can be improved by choosing what order to load the neighbors into the fringe. This can be explained simply using a maze. DFS uses stack data structure as a fringe. Stack is Last In First Out (LIFO) data structure. So, if you first push the actions that takes you away from the goal and then push the actions that takes you closer to the goal the total cost of algorithm will decrease. In the maze below, if you push the orange nodes first and then push the green ones, then due to stack data structure the green ones are popped out and get visited first. Conclusively, the order of neighbors visited in DFS matters.



Bonus: *How does the threshold probability p0 depend on n? Be as precise as you can*
We have plotted solvability vs density graph to find the threshold probability, solvability is nothing but the ratio of (solvable mazes)/ (total mazes generated).
In our case p_0 is found to be 0.4 for n=100, suppose we generate only 3 mazes with p=0.4 and if all the 3 are solvable then it will change the p_0, this is the relation between n and p0.
Keeping this in mind we generated about 100 mazes to get a better estimate of p_0

3) Generating Hard Mazes

- *What local search algorithm did you pick, and why? How are you representing the maze/environment to be able to utilize this search algorithm? What design choices did you have to make to make to apply this search algorithm to this problem?*

For this question, we have used simulated annealing as our local search algorithm. This algorithm is better than Hill climbing algorithm. Hill climbing algorithm has disadvantage of getting trapped into local optima. Simulated annealing tries to improve this problem by introducing some randomness.
Here we take the dimension and the probability as the input from user for generating the random maps. We are also taking as input the method and the metric to use. For this question we are only considering the solvable mazes.
In simulated annealing function we have taken initial temperature as T =10. For temperature drop we have used a (T / 1.0008 - 0.0005) function which gradually reaches to 0. For finding out the probability of a random cell we have used $P(x, x', T) = \exp(-k (F(x) - F(x')) / T)$ this function.
With this given value we are getting almost random iterations 6,500.

- *Unlike the problem of solving the maze, for which the 'goal' is well-defined, it is difficult to know if you have constructed the 'hardest' maze. What kind of termination conditions can you apply here to generate hard if not the hardest maze? What kind of shortcomings or advantages do you anticipate from your approach?*

Our program terminates in 2 conditions:

- When the temperature falls below the threshold value
- If we are not able to generate solvable mazes even after 5000 iterations

Advantages of this method:

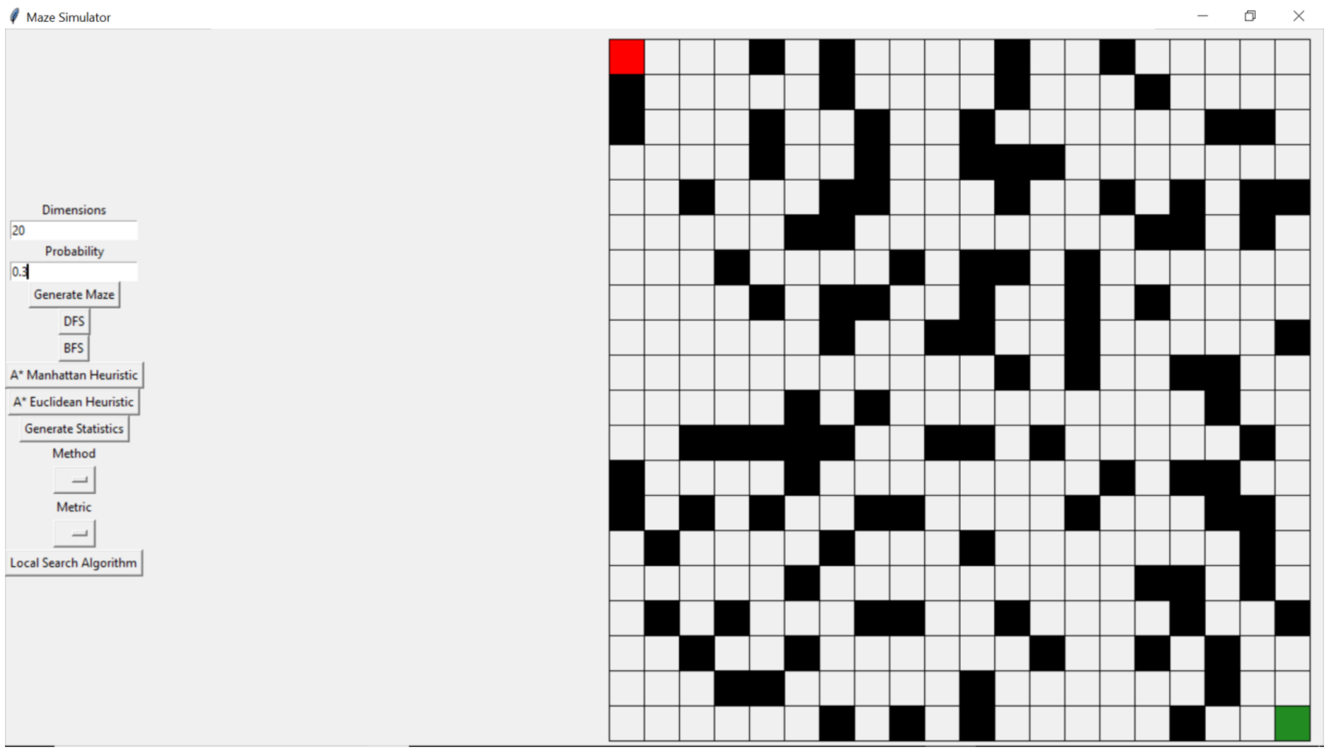
- When temperature is higher, the maze has higher possibility to accept easier maze, otherwise the algorithm is approaching a "local optimum". The probability of bad moves decreases with the temperature decrease.
- In Simulated annealing, if we decrease our temperature slowly enough we will reach the optimal state. The more iterations you perform, closer you will reach of the global optimum. In this "Generating hard maze" part the optimum state will be the hardest maze.

Disadvantage:
You must run this algorithm for very long time for generating the global optimum.

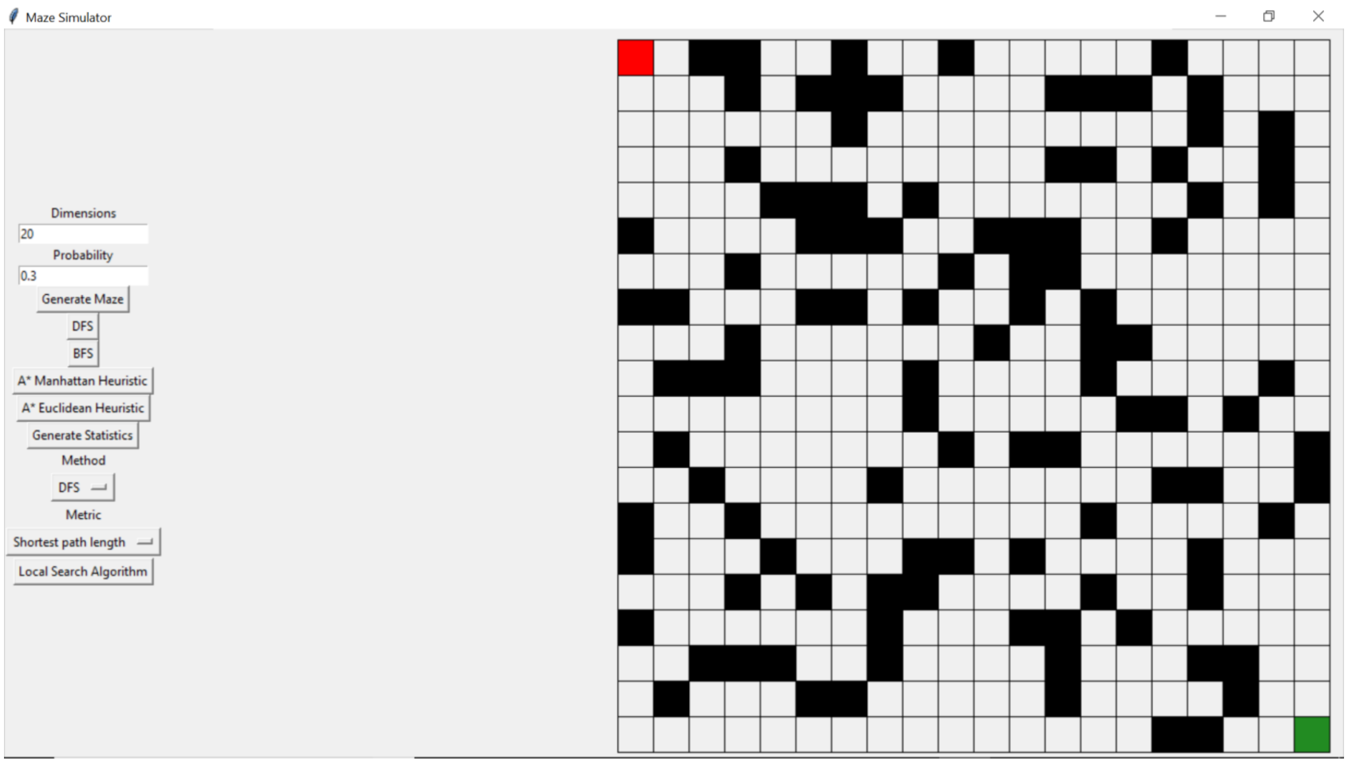
• *Try to find the hardest mazes for the following algorithms using the paired metric:*

For this question, we implemented the paired metric for all 4 algorithms considering 3 metrics, started with initial random maze, chose the algorithm and metric and took screen shots of each hard maze generated, we also noted the initial and final values of the respective metric used.

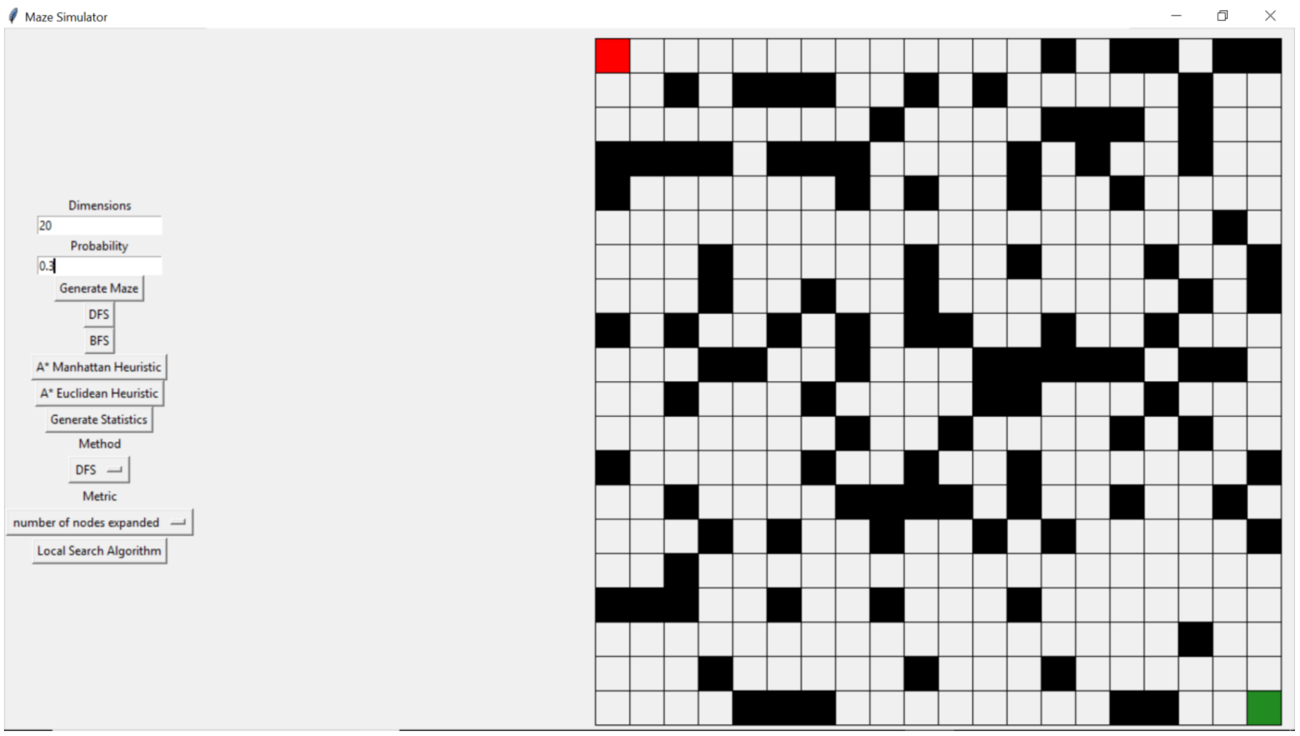
INITIAL MAZE: Dim 20; Prob 0.3



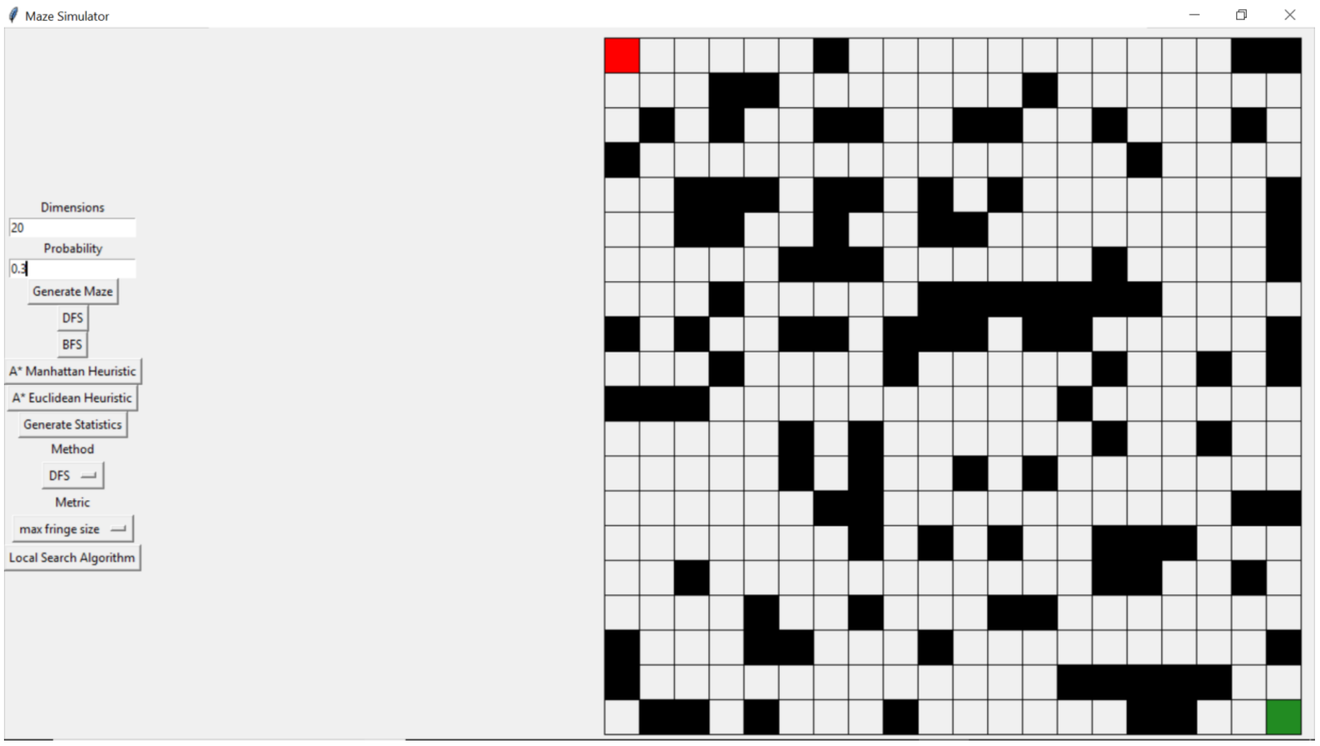
DFS: SHORTEST PATH LENGTH [Before: **143** After: **161]**



DFS: NUMBER OF NODES EXPANDED [Before: **247** After: **274]**



DFS: MAXIMUM FRINGE SIZE [Before: **70** After: **92]**



Maze Simulator

Dimensions

20

Probability

0.3

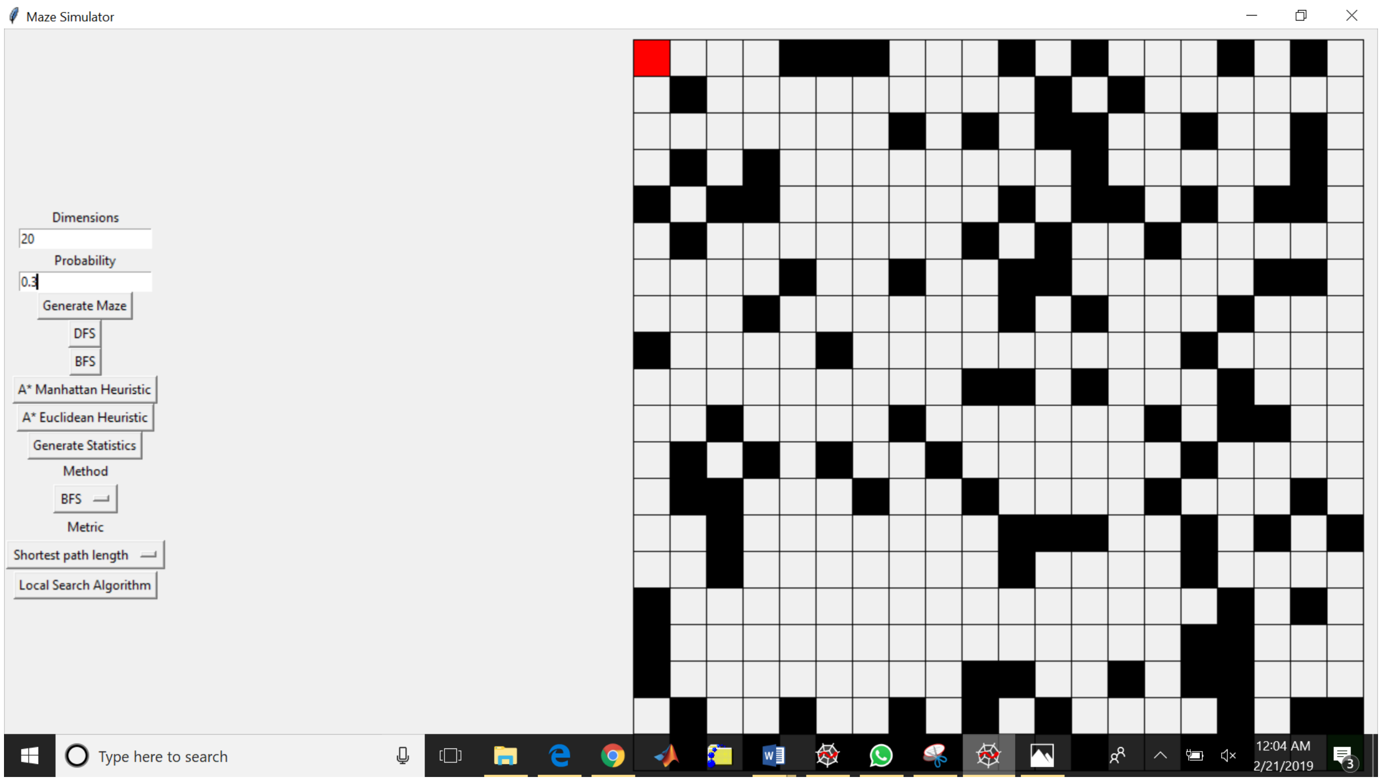
Method

DFS

Metric

max fringe size

BFS: SHORTEST PATH LENGTH [Before: **39** After: **51]**



Maze Simulator

Dimensions

20

Probability

0.3

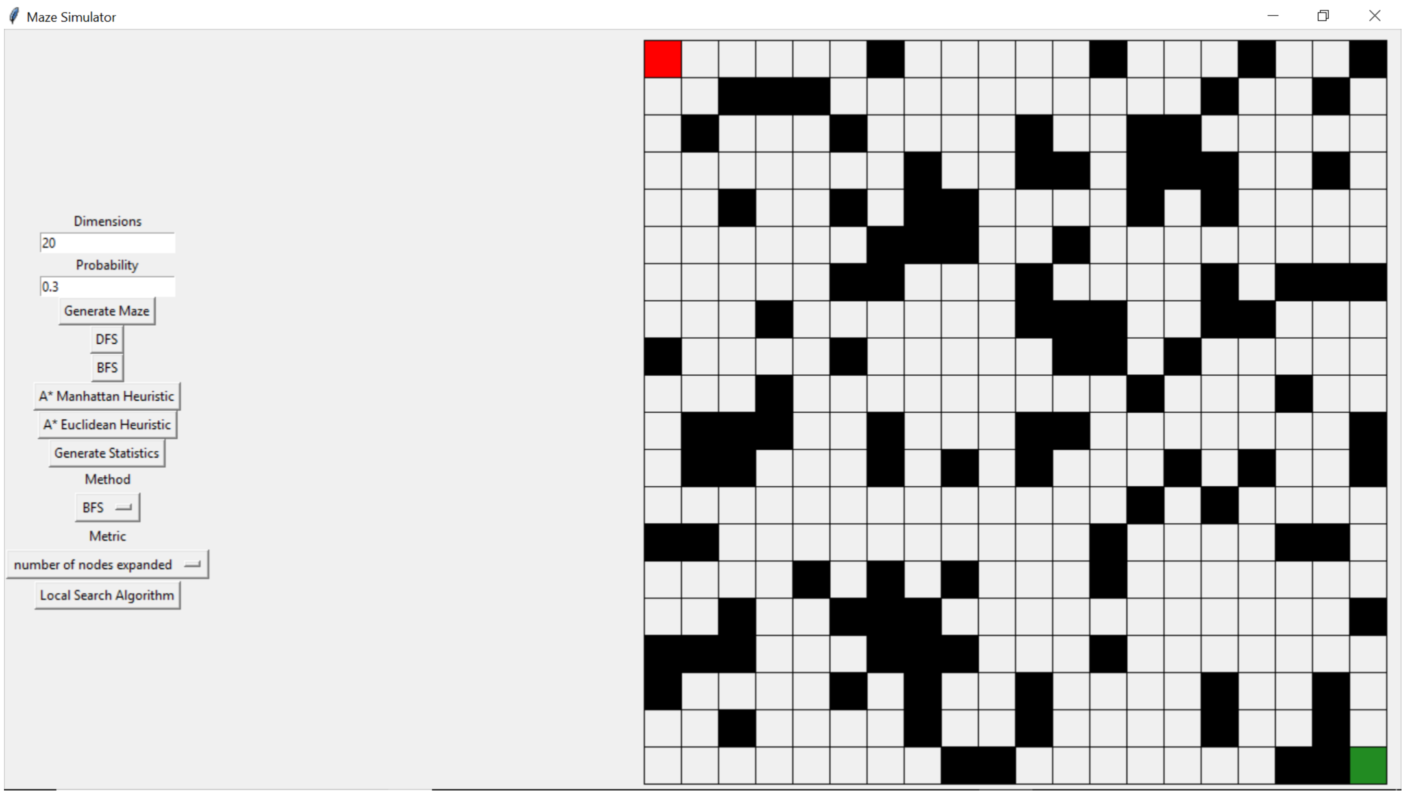
Method

BFS

Metric

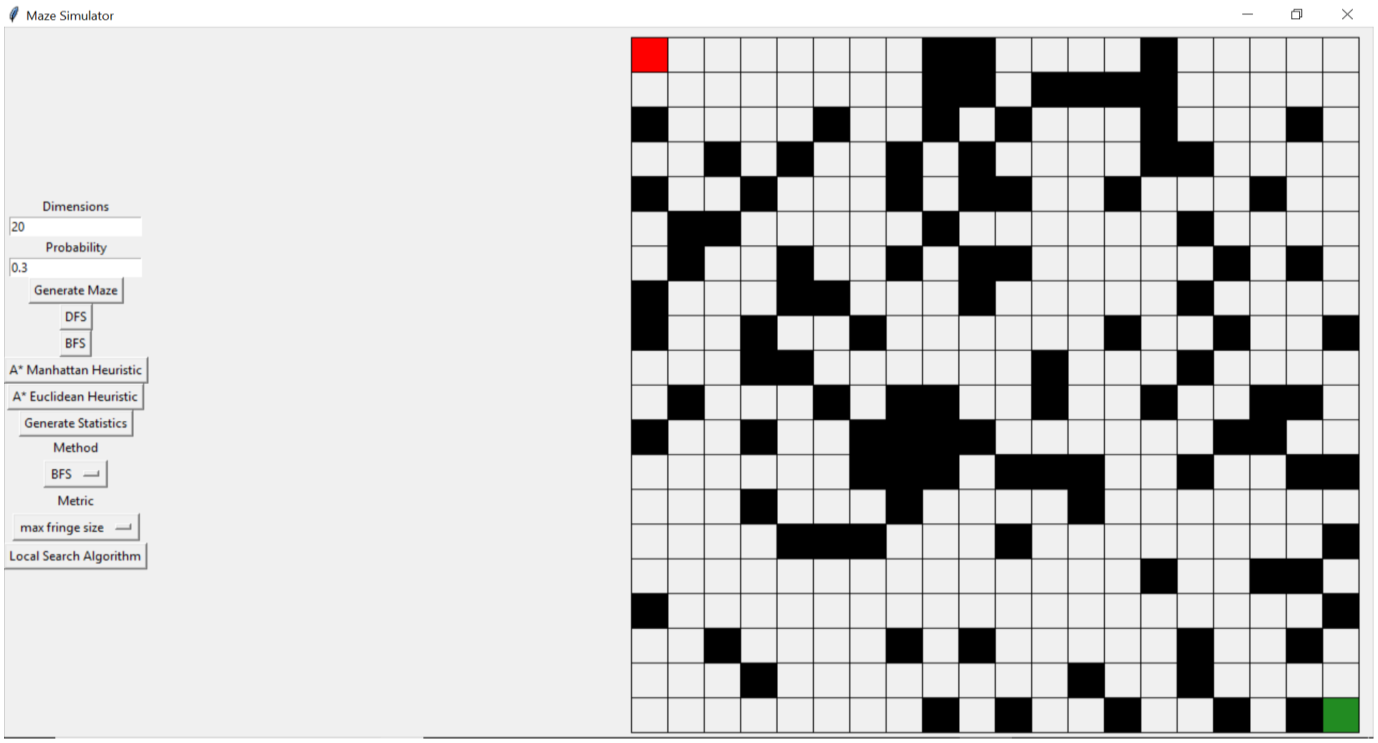
Shortest path length

BFS: NUMBER OF NODES EXPANDED [Before: **291** After: **295]**



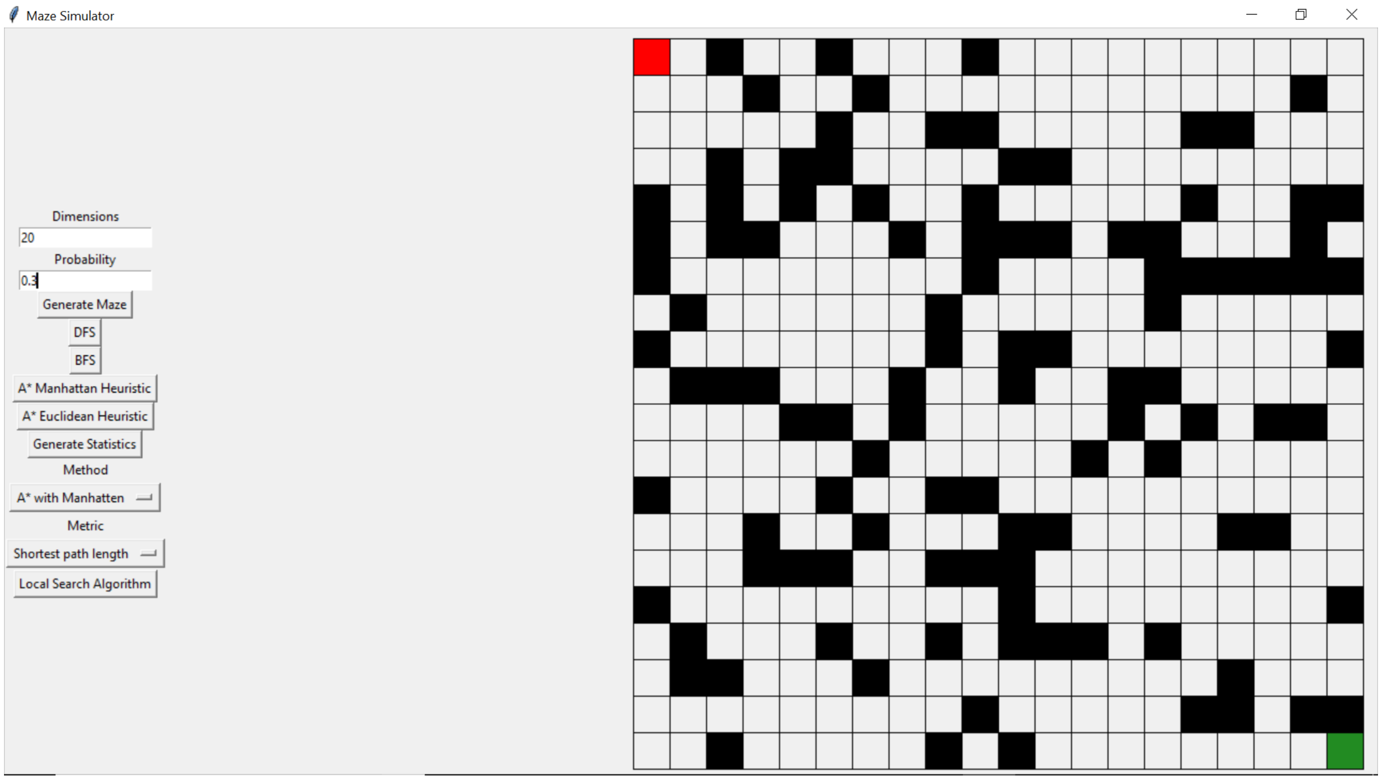
Dimensions
20
Probability
0.3
Method
BFS
Metric
number of nodes expanded

BFS: MAXIMUM FRINGE SIZE [Before: **20** After: **28]**

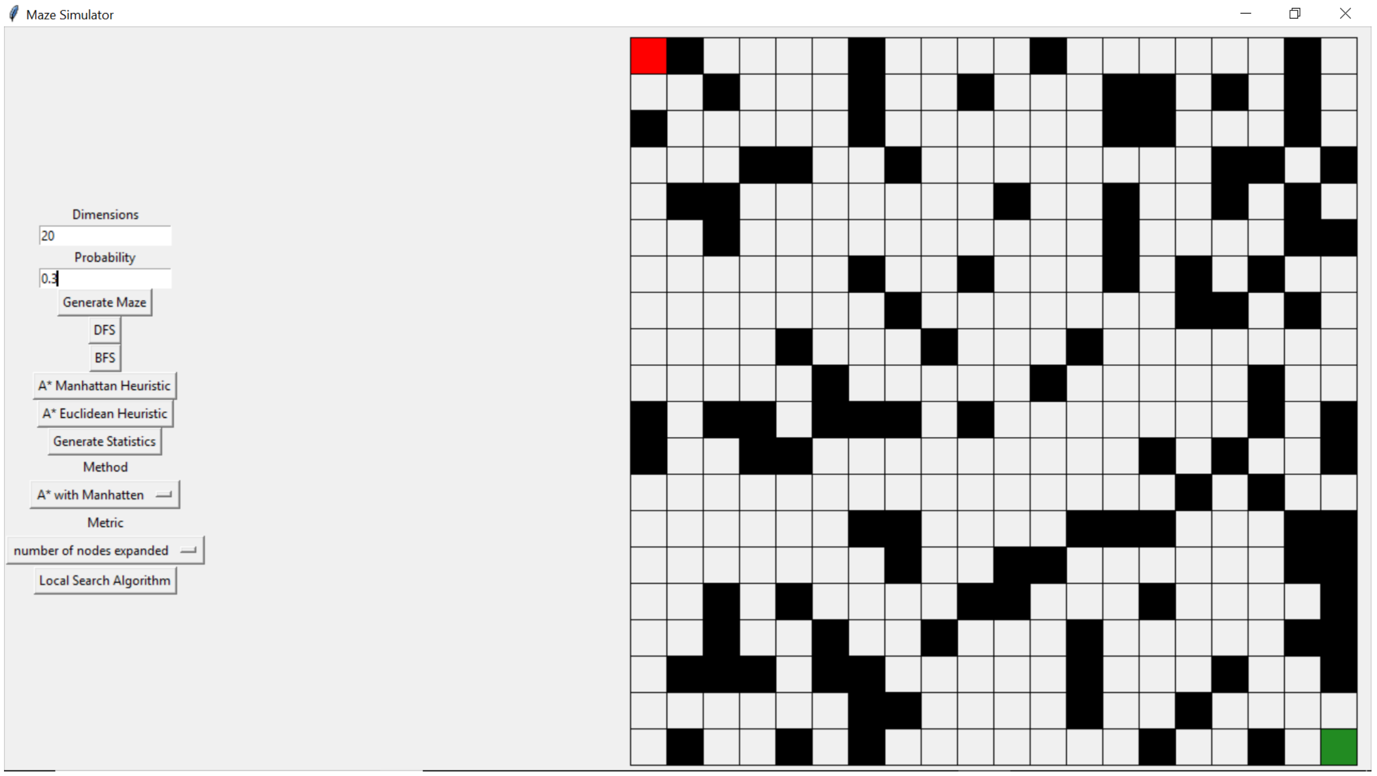


Dimensions
20
Probability
0.3
Method
BFS
Metric
max fringe size

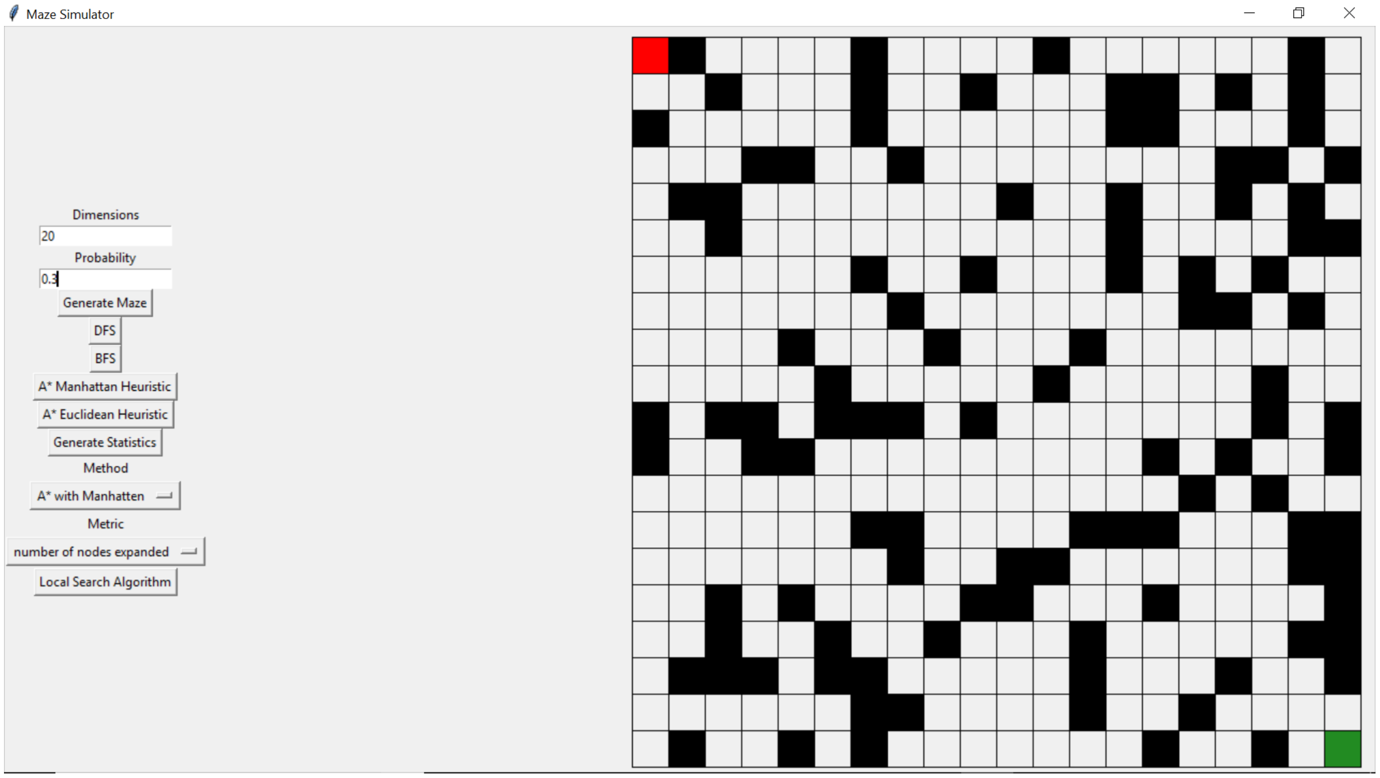
A*-MANHATTAN: SHORTEST PATH LENGTH [Before: **39** After: **53]**



A*-MANHATTAN: NUMBER OF NODES EXPANDED [Before: **153** After: **273]**



A*-MANHATTAN: MAXIMUM FRINGE SIZE [Before: **41** After: **53]**


Maze Simulator

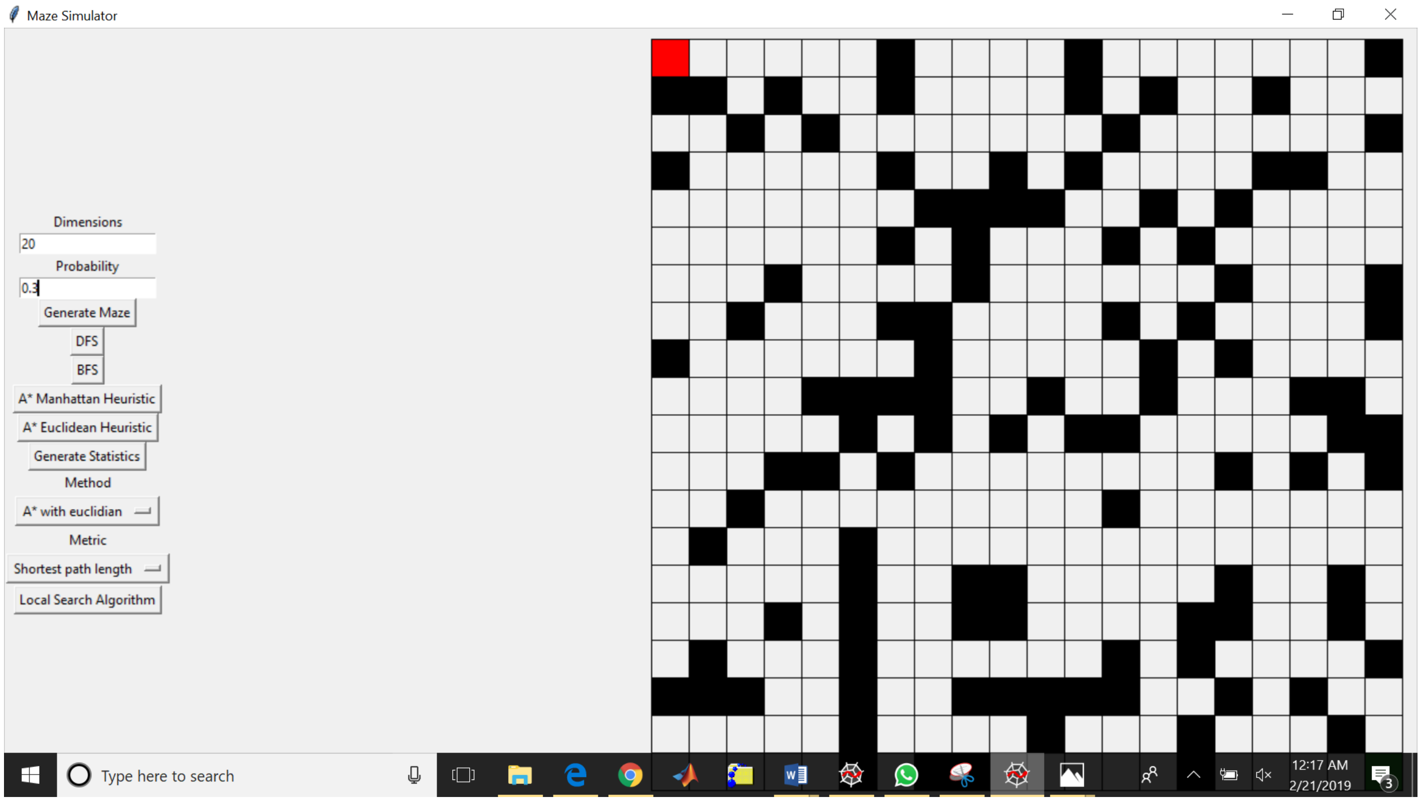
Dimensions: 20

Probability: 0.3

Method: A* with Manhatten

Metric: number of nodes expanded

A*-EUCLIDEAN: SHORTEST PATH LENGTH [Before: **39** After: **53]**


Maze Simulator

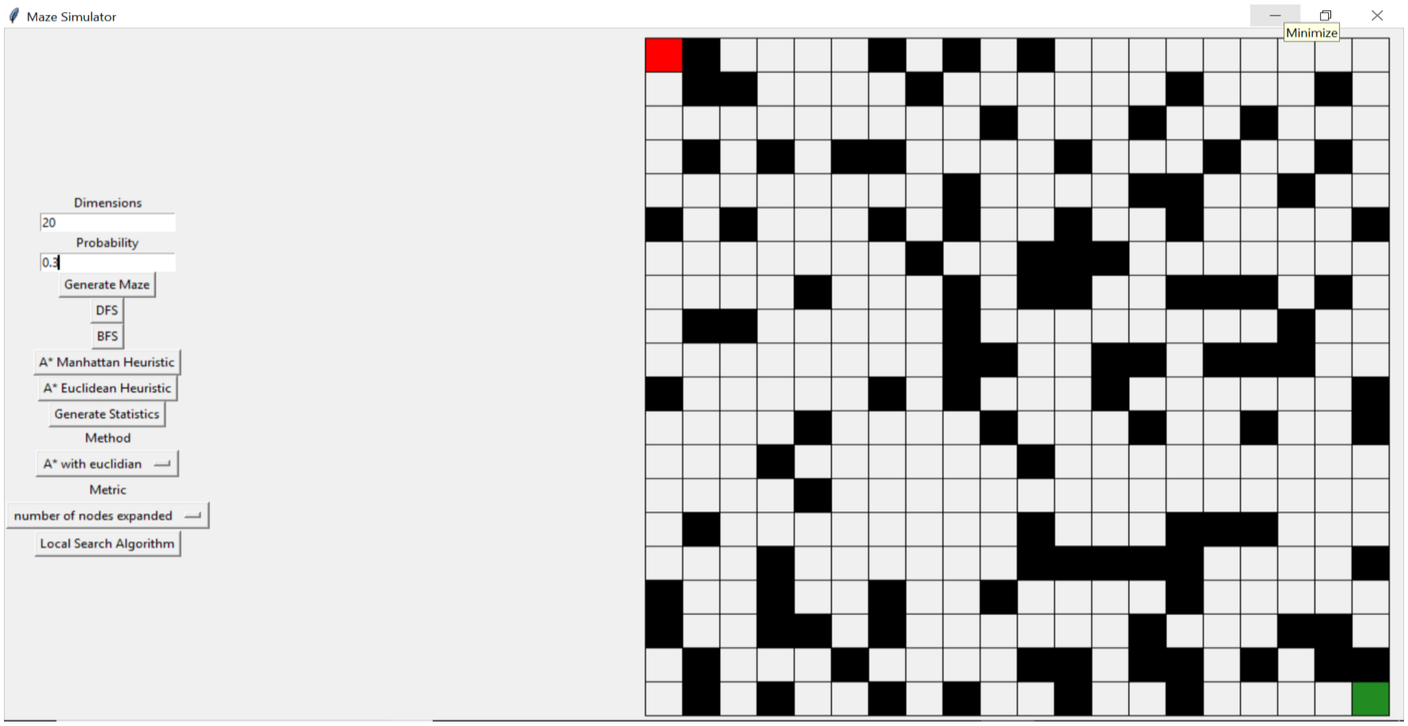
Dimensions: 20

Probability: 0.3

Method: A* with euclidian

Metric: Shortest path length

A*-EUCLIDEAN: NUMBER OF NODES EXPANDED [Before: **153** After: **291]**



Maze Simulator

Minimize

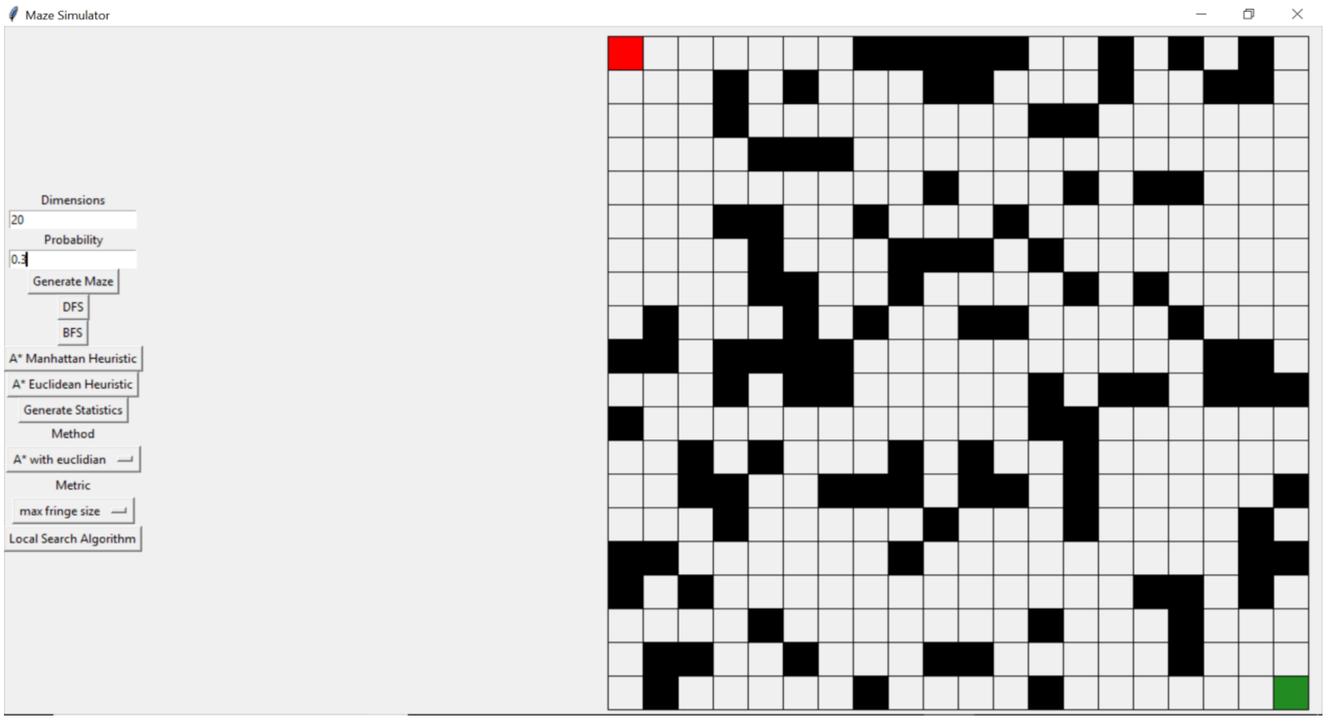
Dimensions: 20

Probability: 0.3

Method: A* with euclidian

Metric: number of nodes expanded

A*-EUCLIDEAN: MAXIMUM FRINGE SIZE [Before: **21** After: **46]**



Maze Simulator

Dimensions: 20

Probability: 0.3

Method: A* with euclidian

Metric: max fringe size

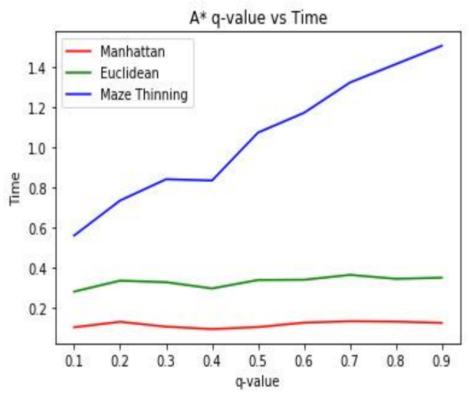
- *Do your results agree with your intuition?*

Yes, the algorithm is generating the harder mazes. If we would have executed it for more time then it would have generated more harder mazes and eventually the hardest maze over the time.

4) A*- Thinning

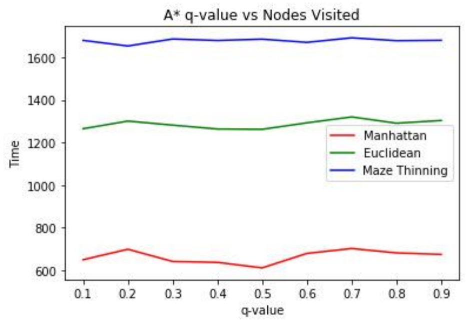
To implement this, we first thinned the maze and gave this thinned maze as input to A*- with Manhattan distance as a heuristic and used this as heuristic to solve the actual maze using A*-Thinning.
This is done for q= 0.1 to 0.9
To compare the results of A*-Thinning, with other Heuristics, the following two graphs are plot

- q vs Time



Here we can see that as q value increases, the time difference is increasing. This is because the thinned maze is becoming simpler and simpler, underestimating the cost by larger values, moving away from the actual nature of the maze.
Even for the lower values of q, it can be seen from the graph that A* thinning takes more time than A* Manhattan and Euclidean distances. This tells that A*-thinning is not a good strategy for solving mazes.

- q vs Nodes visited



Plot between number of nodes visited and q also proves that A*-thinning is not a good strategy, it visits more nodes than the others

Contributions to the assignment.

Sharvani Prathinidhi - A*- Euclidean, A*- Manhattan, A*-Thinning
Amit Patil – BFS, DFS, Simulated Annealing
Pranita Eugena – Report, Graphs

Logic behind all the ideas is a combined contribution.