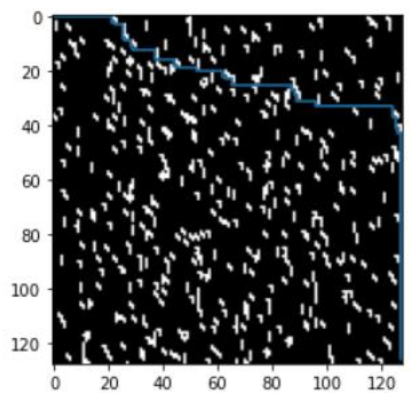
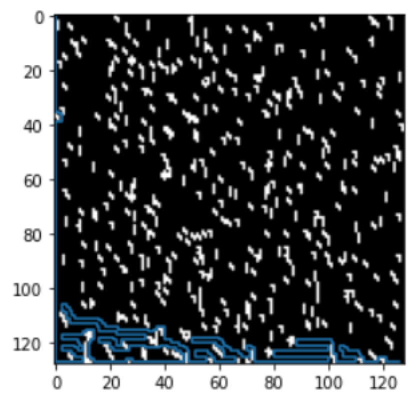
Report

Source Code: The code has been divided into four main functions namely:

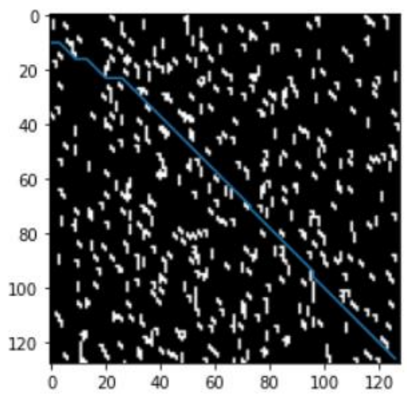
1. The environment setup was taken from assignment 2 of the course.
2. A function for **Breadth First Search** was implemented. We know that it follows the queue algorithm. Queue follows the FIFO (First in First out) methodology.
    - This further has another function known as isValid where we check whether the neighboring cells have an obstacle. This will return to us the validity of visiting that cell in our path.
    - In the main function, we have a queue where we pop out the first element and assign that to x,y, and the third element as our path.
    - If this x, and y are the goal coordinates then we return to the path. However, if not, then we run a for loop to iterate 4 times for each neighboring cell and execute the isValid function. These values are then appended and a path is calculated for the point robot to follow in order to reach the goal location.



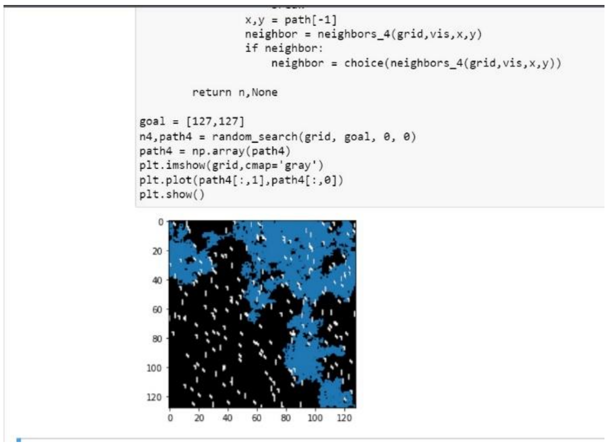
3. Another function was created for **Depth First Search**.
    - We know that in DFS it follows the logic of stack which is FIFO(First in First out). Therefore, the only difference between the codes of BFS and DFS would be while popping a value out of the queue.



4. Next in line, we have the **Dijkstra search**. Here, we calculate the shortest distance between the starting position and the goal position.



5. Finally, a **Random search** is also implemented.
    - Here, we decide which random neighbor cell to visit with the help of ‘choice’. This eventually generates a random search path.
    - However, that is not the case every time. Most of the time the number of iterations would go out of range since it is random.
    - There were times when I did get an output. Although it was not the desired or even an efficient path. It can be observed in the image below.



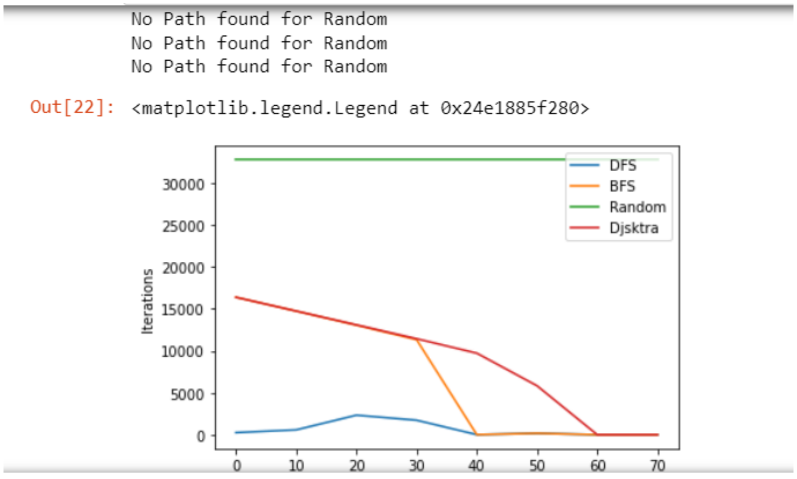
```
                x,y = path[-1]
                neighbor = neighbors_4(grid,vis,x,y)
                if neighbor:
                    neighbor = choice(neighbors_4(grid,vis,x,y))

        return n,None

goal = [127,127]
n4,path4 = random_search(grid, goal, 0, 0)
path4 = np.array(path4)
plt.imshow(grid,cmap='gray')
plt.plot(path4[:,1],path4[:,0])
plt.show()
```

PLOTS

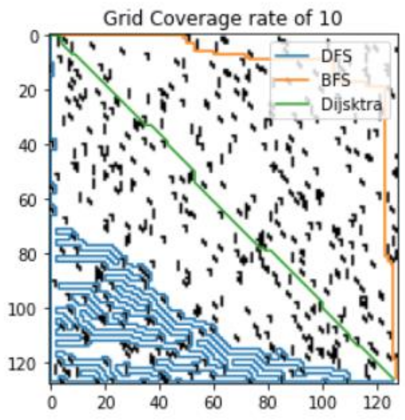
1. In the graph shown below, we can observe the number of iterations it took for each Search algorithm its respective coverage of obstacles to reach the final goal position.

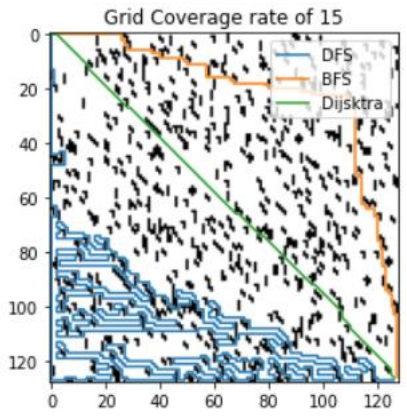


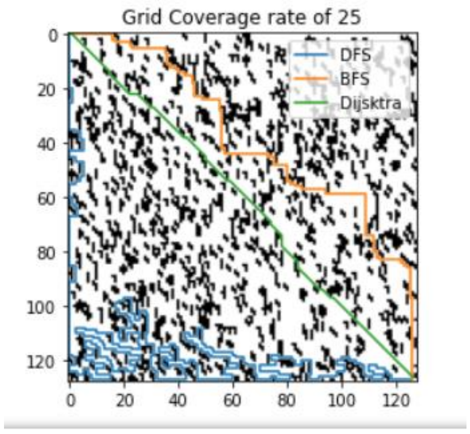
```
No Path found for Random
No Path found for Random
No Path found for Random
```

```
Out[22]: <matplotlib.legend.Legend at 0x24e1885f280>
```

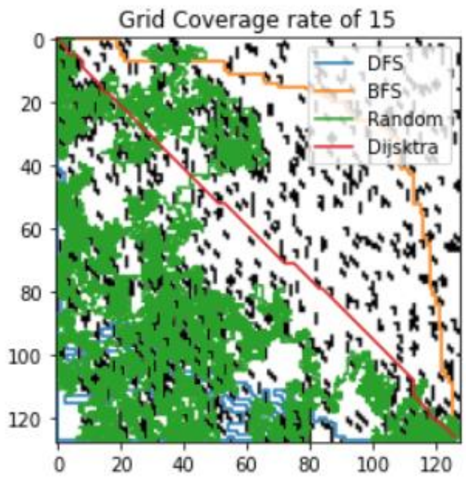
2. The graphs below show the implementation of Searches at 10,15 and 25 percent of coverages.







- It is important to note that there were times when the random search program would run and would give us this graph where it almost overlapping all the other searches.


Grid Coverage rate of 15
DFS
BFS
Random
Dijsktra