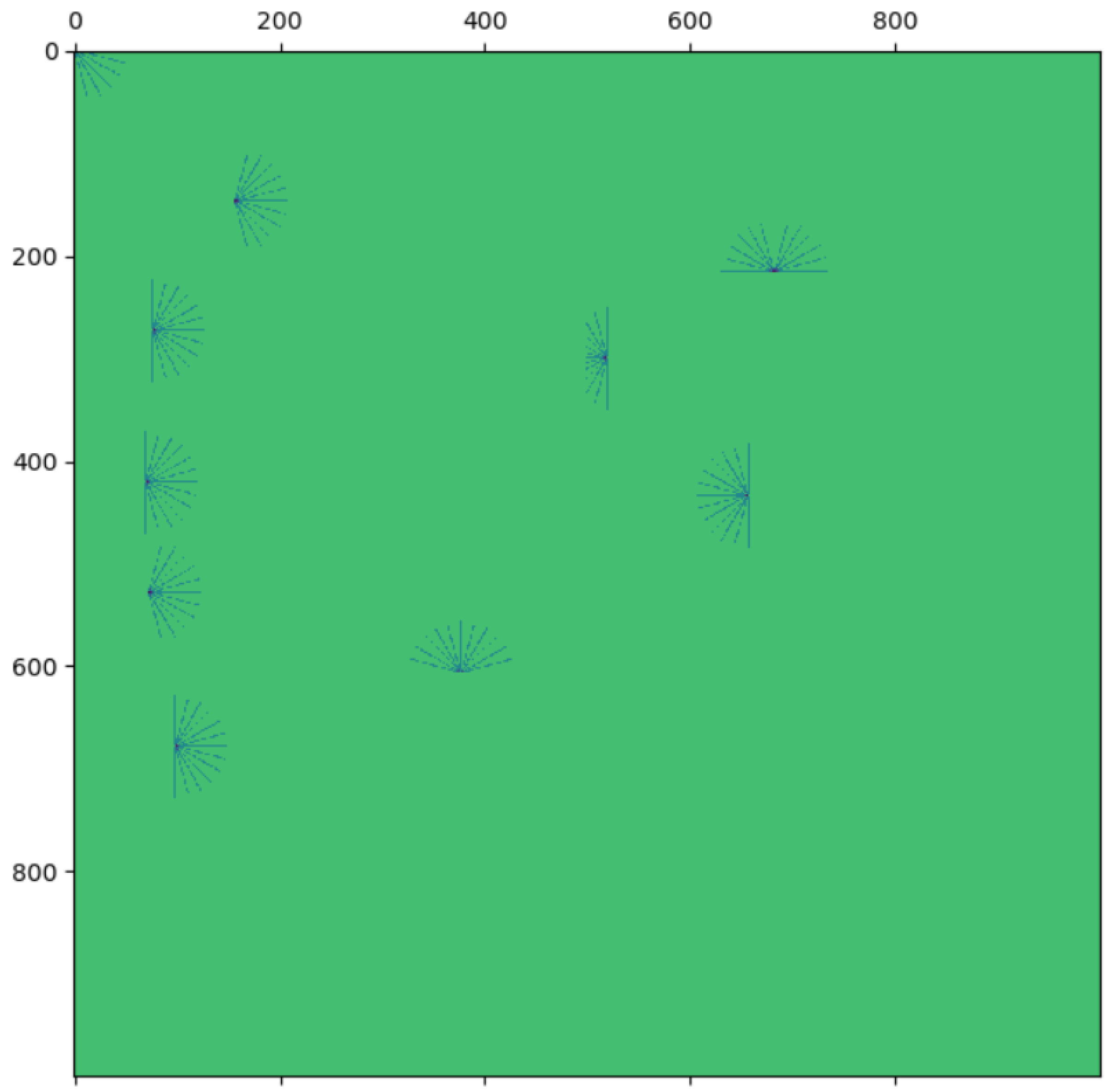
Program Output:

When run, the program will output a 1000 x 1000 matrix of probabilities into a file called "probabilities.txt". The raw data is hard to reason about and verify, so a python script called "visualize.py" has been included with this project to help visualize the occupancy grid map. In order to run this script, you must have python 3, numpy, and matplotlib installed on your machine. Otherwise, here is the data:



Each of the robot measurements can be seen here. A cell that is green marks a cell where the probability of being taken is 50%. Blue cells have a low probability of being taken by an obstacle, and as can be seen in the data the closer that a measurement is to the robot the darker blue it will be, as multiple rays will hit these and increase the confidence level of the measurements. Yellow cells have a high probability of being occupied (see task 3).

Task 1:

The data structure that we will be using for this project will be a graph. Our graph will be directed, with each edge being encoded with additional data of the direction that the edge is facing, and the internal structure of the graph will be an adjacency list. The graph will be sparsely connected, with each vertex having a maximum of eight edges associated with it.

Conceptually, we interpret this graph as being a two-dimensional, square grid. An edge stemming from one vertex will only be able to connect to one of its eight adjacent vertices. In comparison to a complete graph, our graph will have a much smaller space complexity. Since the sides and corners do not have eight surrounding vertices, they will have five and three edges respectively.

The space complexity of any graph is defined as *O(V + E)*, where *V* is the number of vertices, and *E* is the number of edges. We will use *N* to represent the number of vertices in a graph, and *n* to represent the width of the grid our graph represents ($N = n^2$). In a complete graph, the number of edges it has is given by $\frac{N(N-1)}{2}$. In our sparser graph, the number of edges is given as $8n^2 - 12n + 4$. Thus, the space complexity of a complete graph is:

$$O(N + \frac{N(N-1)}{2}) = O(\frac{N^2 + N}{2})$$

which is on the order of *O(N²)*. For our sparser graph, the space complexity is:

$$O(N + 8n^2 - 12n + 4) = O(N + 8N - 12\sqrt{N} + 4) = O(9N - 12\sqrt{N} + 4)$$

which is on the order of *O(N)*. Thus, our graph will be much more space efficient for larger values of N. This is necessary, as our plan is to use $N = 1{,}000{,}000$, since our graph represents a 1000x1000 grid.

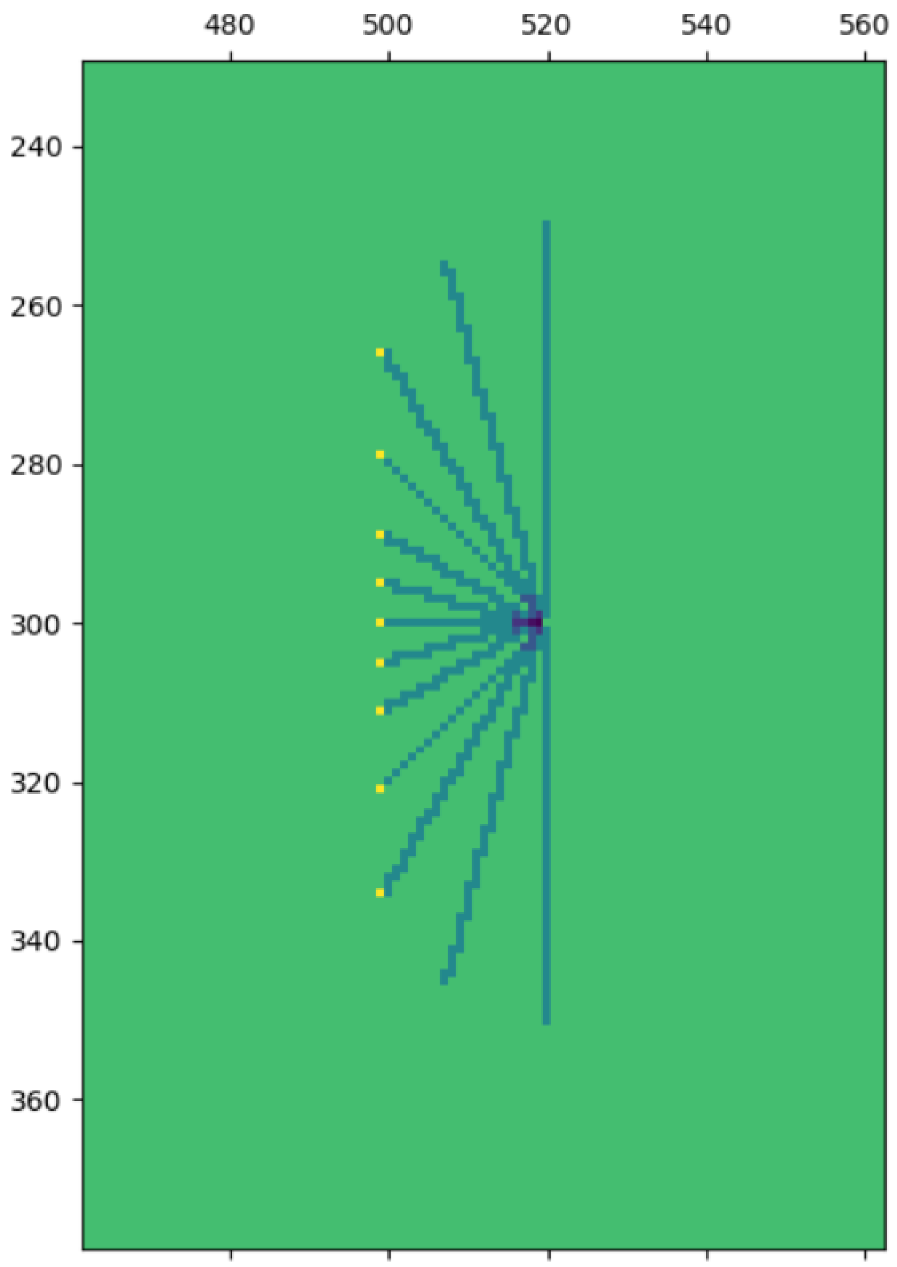
Initialization of our graph involves two *O(N)* operations. The first *O(N)* operation is necessary to initialize all vertices of the graph, and the second is to initialize all necessary edges. However, these operations will only occur once, as they are used to construct the map class that uses the graph. The other main operations are updating the map (graph) with a new set of probabilities, and the ray casting operation.

Task 2:

One of the main motivations that we had for using a graph was to simplify calculations. Ray casting became easier when each vertex had eight edges with a corresponding direction on the map (the cardinal and intermediate directions). Additionally, the fact that we can traverse through the graph allows us to easily update probabilities in a streamlined manner.

Task 3:

Because none of the provided robot measurements were close enough to hit the obstacle, we decided to put our 10th measurement on the right side of the obstacle, as seen here:



As can be seen here, a sensor ray will stop taking measurements when it hits an obstacle, and it will mark the cell that it hit with an occupied measurement. This type of cell is colored as yellow by the visualize script.

Task 4:

Updating the map will involve iterating through each vertex, which is an *O(N)* operation. Our method of ray casting will start at the robot's position and go to the end of the ray, it does not test each vertex in the graph, only the same range of vertices. However, recovering a vertex to test is an *O(N)* operation, so therefore the ray casting will be *O(N).* Updating a single probability in the probability map is an *O(N)* operation, and since this is only called in the ray casting method, updating all probabilities is an *O(N)* operation as well. We do not see a different data structure that would reduce the time complexity of the ray casting, and the extra information that we were able to encode into the graph made the implementation of ray casting easier to write. Therefore we feel that our choice of data structure is adequate.