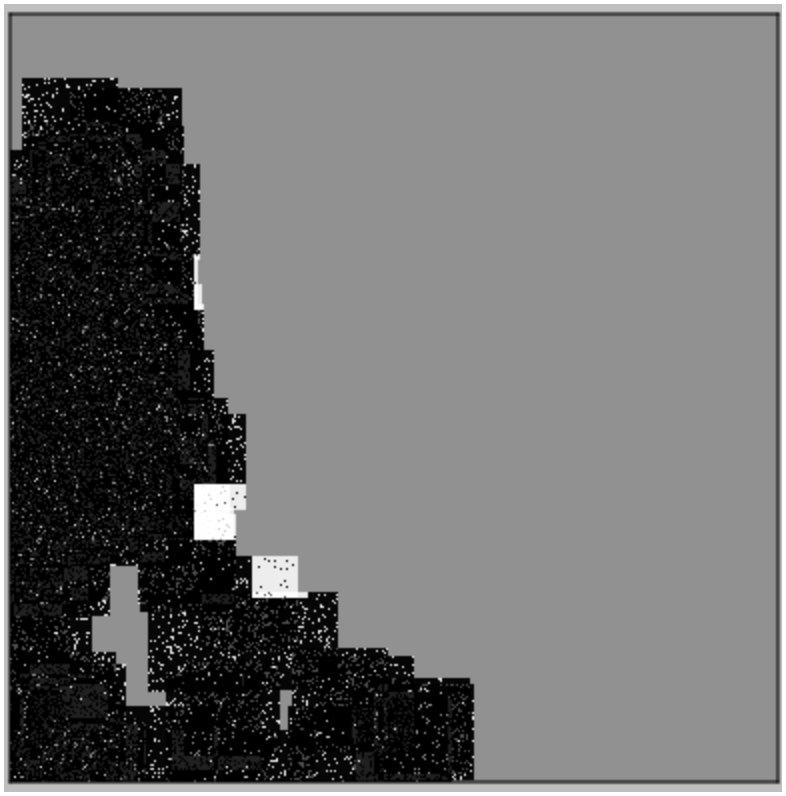
Garrett Thornburg
Chris Fortuna
CS 470
6/3/2015

Grid Project

Using 3 late days for this project.

1.

Four L's Map:



Point at: (-362, 284)
Current prior given (1): 0.500000
New value estimation: 0.906542

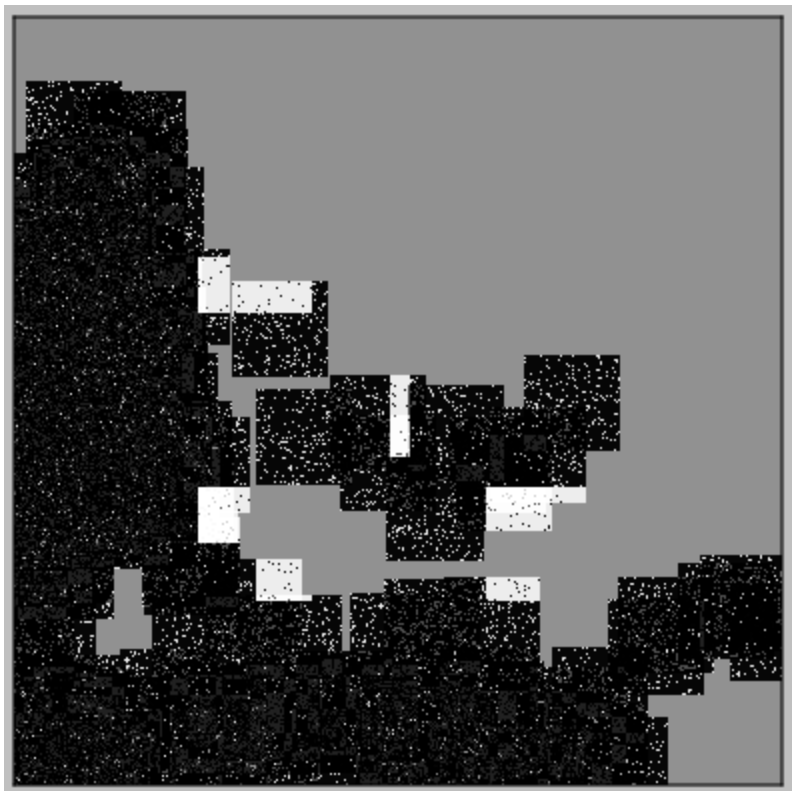
The previously unseen point (-362, 284) starts with a prior value of 0.5. Given the that a 1 was observed and the true positive rate is 0.97 the new estimate for the point in the model's map is 0.906542.

Point at: (-361, 267)
Current prior given (0): 0.244332
New value estimation: 0.010663

The point (-361, 267) was observed previously and thus has a existing prior of 0.244332. It uses the fact that a 0 was observed along with the true-negative rate of 0.9 to determine the new posterior value of 0.010663.

Point at: (-361, 283)
Current prior given (0): 0.500000
New value estimation: 0.032258

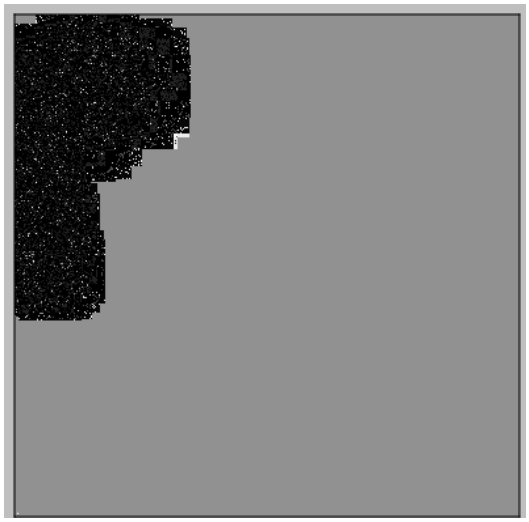
This process is the same as the last point exept this point was previously unseen so it's initial prior was 0.5. After the posterior of 0.032258 is computed for the point this value is set as the new prior for this point which will be used in all future updates.



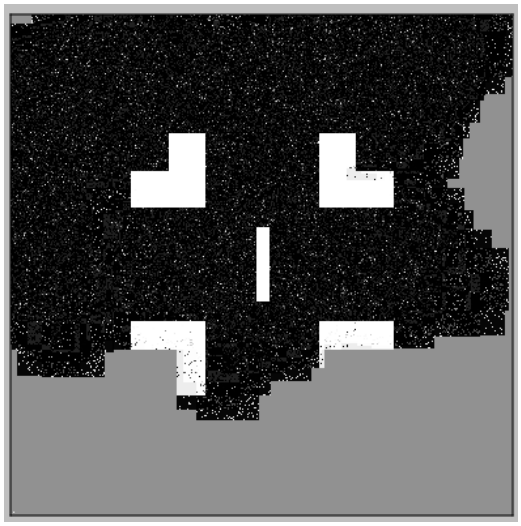
Point at: (-29, -336)
Current prior given (1): 0.100000
New value estimation: 0.518717

Updating this point in the map follows the same process as the 3 listed above.

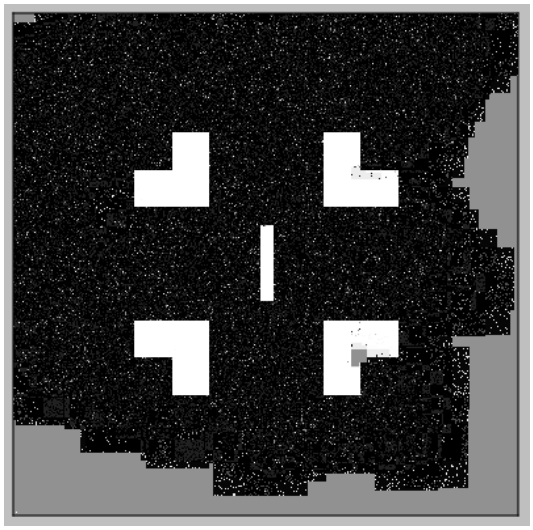
And now for a walk through of Four L's:



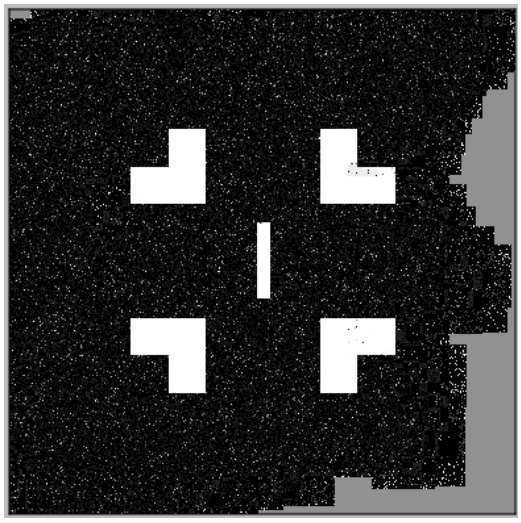
Here the red team starts at their home base, and quickly navigates to the topmost and leftmost point on the graph, and then begin to sweep to the right. It is also visible that one of the tanks randomly discovered a tip of one of the L's in the map.



A few minutes later, the tanks have scanned the majority of the map, and the target point is skipping ahead due to the randomized optimization we gain from the scanning approach. The top two L's have now been completely discovered, and the middle bar has also been discovered.

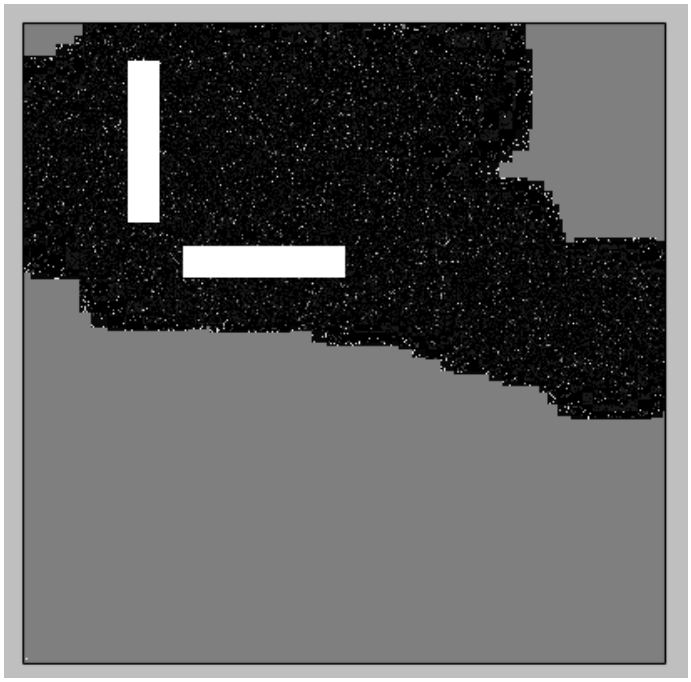


Shortly after, the bottom two L's are nearly completely discovered. Even though the filter grid scan has passed the bottom-right L, the randomness that comes with the move_to_position function, will make sure those missing points are filled in before completion.

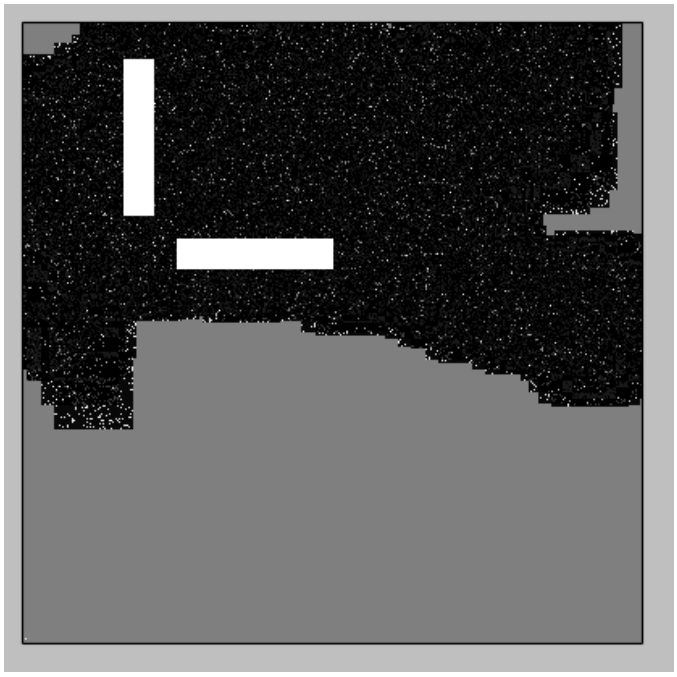


And there you have it! The map has been completely discovered, and the tanks have been able to randomly discover those missing points in the bottom-left L. Ther robots will actually search a bit more, but this is where we arrive at another problem. We were unable to successfully use the OpenGL GUI because we wanted this to run on a Mac and Linux, so we opted for matplotlib, which starts drawing very slowly, meaning the program slows down, and we start to lose control of the tanks. We know there are ways to optimize this, but we feel confident that these parameters illustrate that our approach was successful in completing the lab.

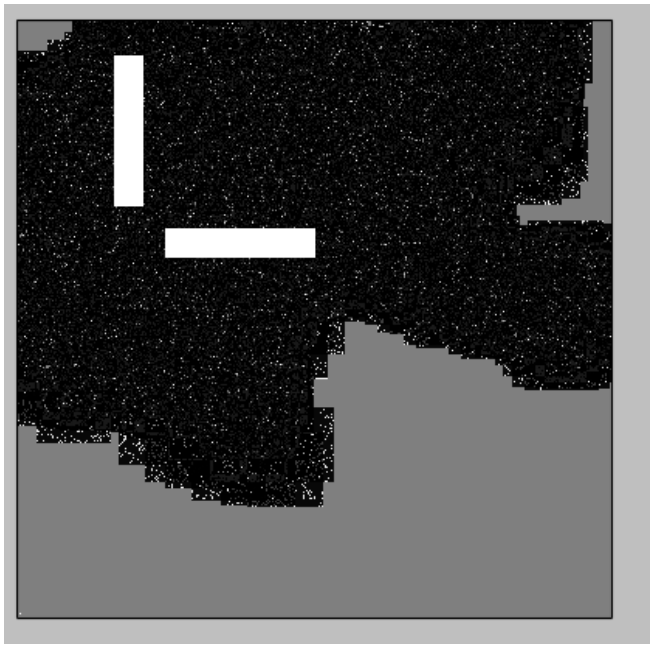
Final1:



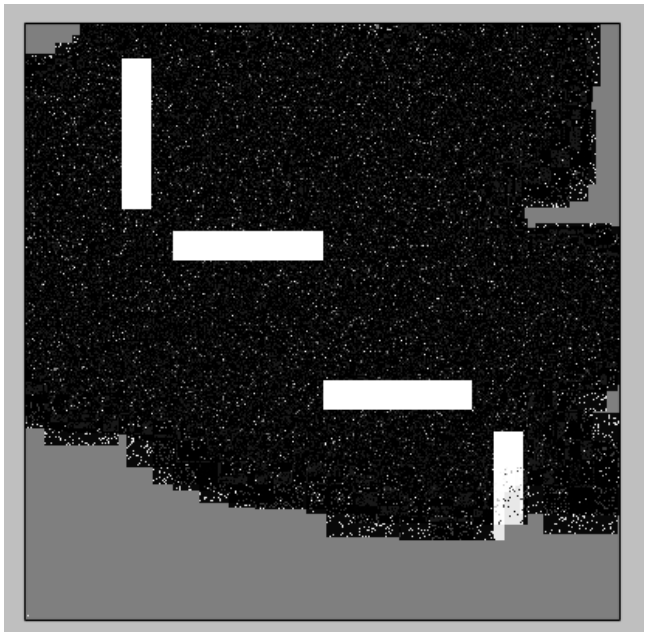
1 minute in. The tanks in red base move to (-400,300) since it was undiscovered at that point. The search then continues from the upper-left hand corner to the bottom right looking for the next undiscovered point in the same was you'd read a book (left to right, up to down).



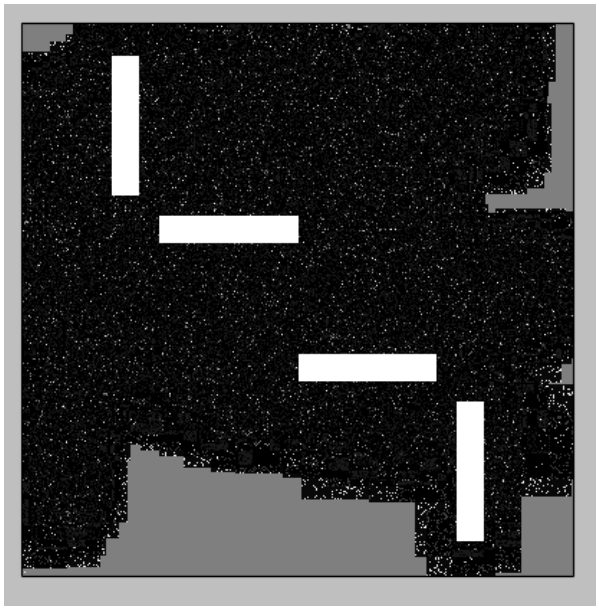
2 minutes in. The tanks explore back and forth moving down the map as points are discovered. The tanks use a simplified grid of the entire map broken down into 100 map unit squares.



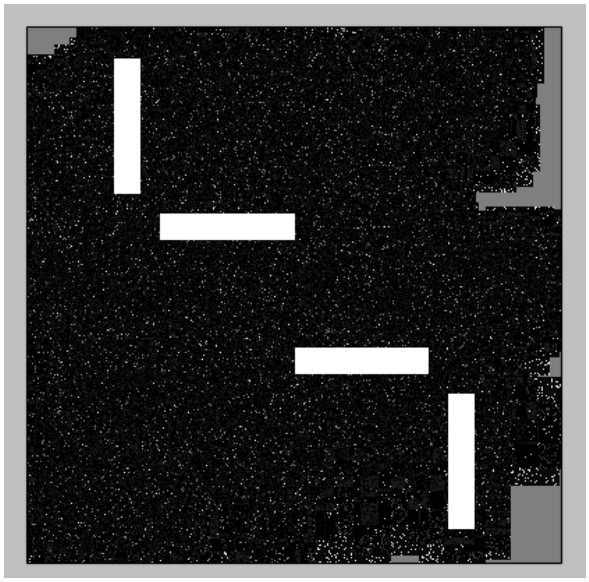
3 minutes in. The tanks are moving towards (100, -100) at this point.



4 minutes in. The tanks are moving towards (300, -300) now.



5 minutes in. The tanks are trying to reach (-400, -400) now.



6 minutes in. The map is basically discovered. The tanks are moving towards (400, -400) at this point.

2.

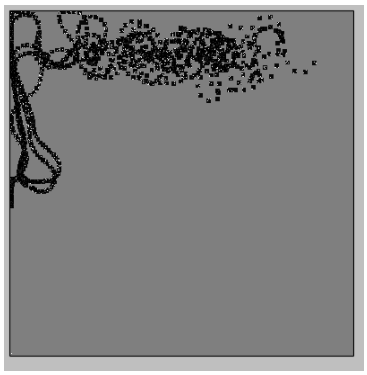
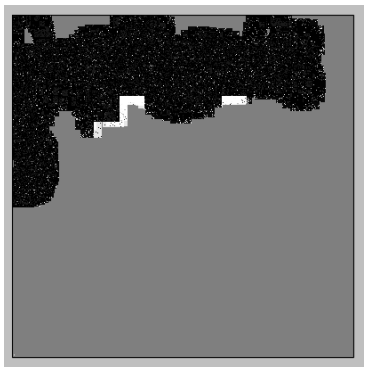
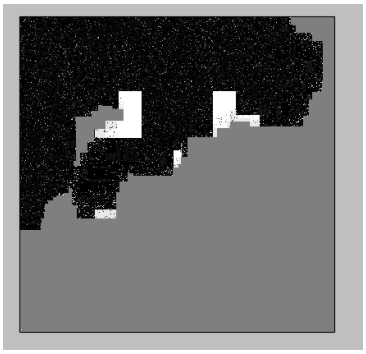
We found out program running too slow so we lowered our tank count, but when that wasn't enough, we switched from potential fields to move_to_position, which dramatically improved our results, and came in with a built in random movement to avoid getting stuck. We then attempted to raise our tank count from 5 to 30, and then back down until we hit our performance limit. We found promising results with a tank count from 8 - 10, but ultimately, we settled with 5 tanks.

Now, let's look at our strategy for discovering the space. We started by averaging the four quadrants and determining which one was closest in value to our default value (in this case, 0.5); however, we ran into issues where this approach would actually get stuck after discovering a space, and our best guess as to why is that the average just happened to be a little bit closer to 0.5 that some of the other quadrants. To avoid this, we switched our strategy to search from top to bottom, left to right. We are essentially doing a full table scan every 1 second, but to stay efficient, we're only examining every 100th value. This is just fine, because the occgrid is large enough, and the tanks are random enough, to make up for any underexamined pixels.

When it came to distributing the tanks across the map, we found that approach less efficient, and instead sent all 5 tanks to the same location. The reason this is more efficient, is because the move_to_position function actually spread the tanks out near a point, and assisted our algorithm to find a new target point, because when it scanned again, there's a good chance the next region was already discovered. In other words, sending a bunch of tanks to the same point, with our scanning approach, helped us scan ahead and optimize our scanning for untouched pixels.

3.

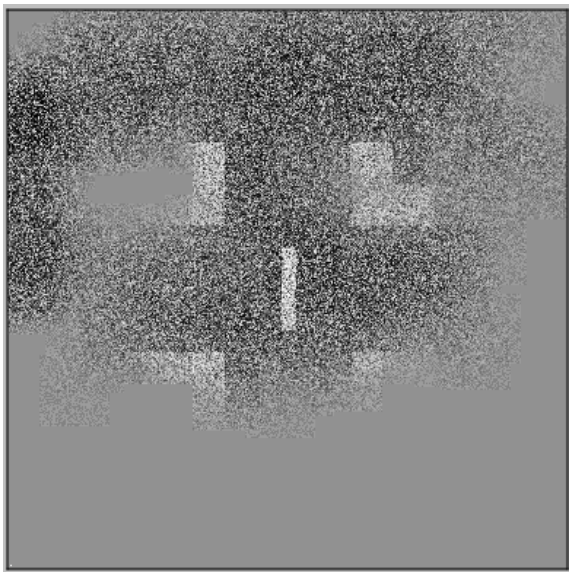
<u>Varying range:</u>
For all tests we stop the tanks after 2 minutes of searching.


The first test has a range of 10, the second 50, the third 100, and the fourth 150. It's important to note that the 4th test was only run for 1 minute as the computer ran out of resources and froze. As can clearly be seen increasing the range of each tank's vision greatly increases the amount of the map that can be discovered in a set amount of time. This also uses many more resources as well as can be seen from the fact that we couldn't run the 150 range for the full 2 minutes. There is a clear tradeoff between map discovery time and resources needed when you're only varying the tank's vision.

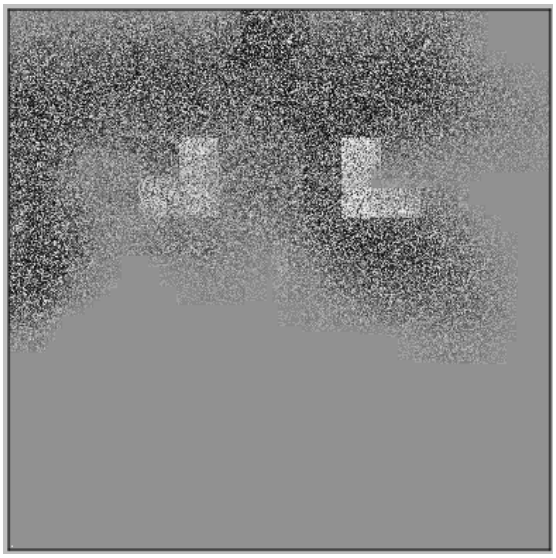
Noisier elements:

True Positive: 65% - True Negative 45%



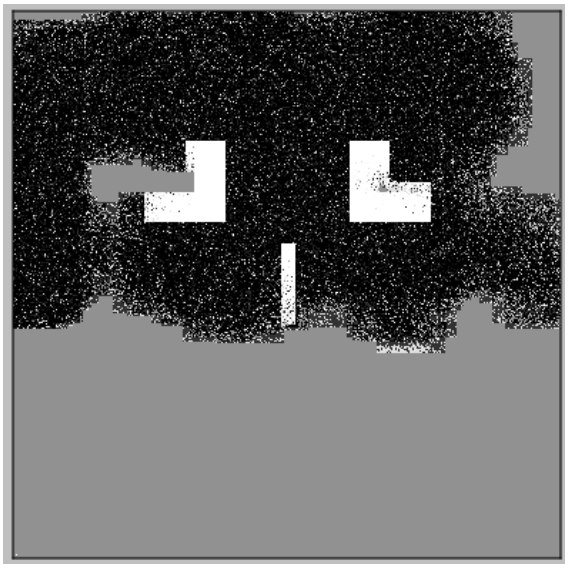
This image is very light, and seems to be the result when true_positive is significantly higher than true_negative. Because the true negative values are slightly mistrusted the open space has more random instances of obstacles then free space in its noise. Thus the free space is more light grey and the obstacles are a little whiter since the model tends to trust positive values.

True Positive: 45% - True Negative: 65%



The opposite is seen here. Here the model tends to trust negative values and distrust positive values. Thus the empty space is much darker than the example seen previous while the obstacles themselves are darker too as the model tends to mistrust positive values.

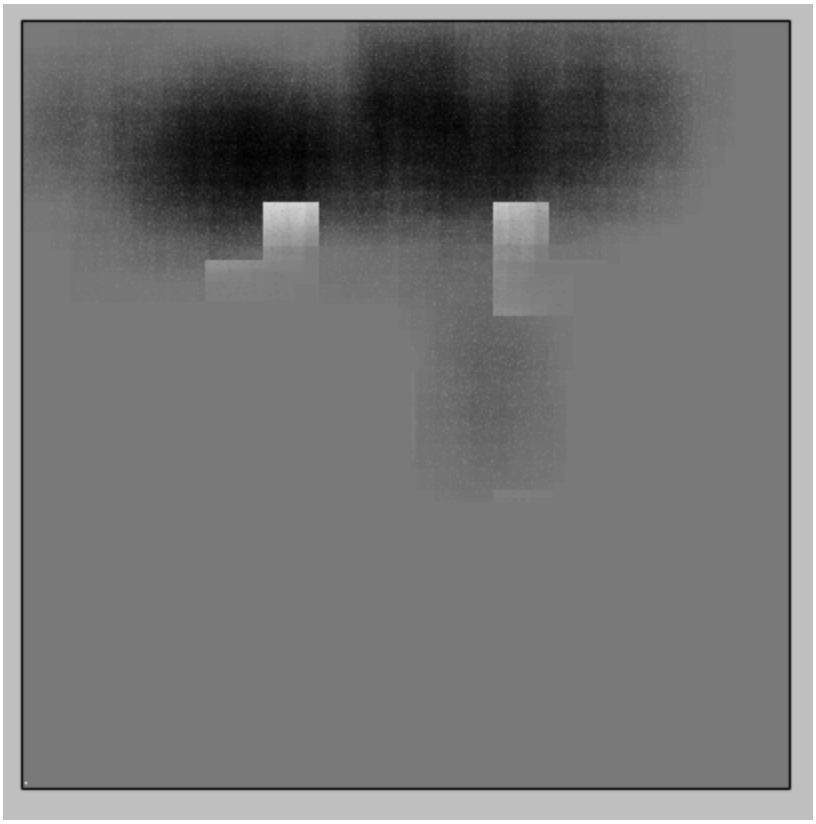
True Positive: 25% - True Negative: 25%



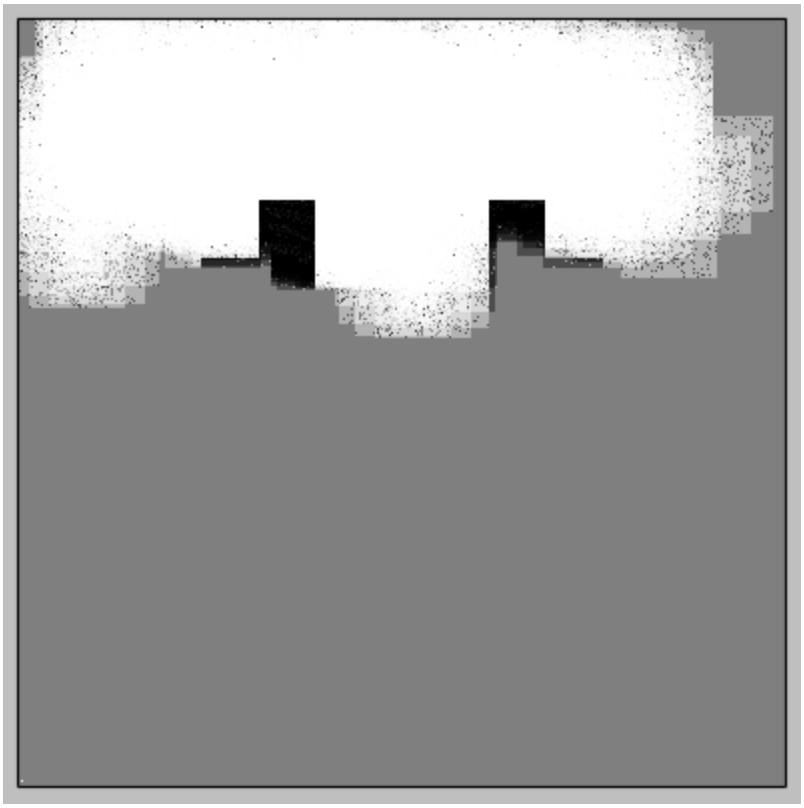
Because the values returned are have a 0.75 probability of being wrong, and because the model knows this, the model is able to assume the opposite of all information it receives and is able to build a correct map of the world. Random values much closer to 0.5 are much harder to build maps with as is seen by the two examples above.

Model incorrect:

The real values are always 0.97 and 0.9 as specified in the lab spec.



In this example the model believed that the values it was had a 0.51 true positive and true negative rate. Because of this the model was much less confident of the values it was receiving and only increased or decreased the values slowly and is seen by all the shades of grey everywhere on the map. The image shows the model's map after 2 minutes of searching. It's obvious that the model has some idea of where the first few obstacles are but is not 100% sure it's receiving correct data so it's probabilities are still near 0.5.



In this example true positive and true negative rates of 0.3 were used. This means that the model assumed that the data it was receiving was usually false, thus the model decided to believe the opposite of any data it received. This is seen in the image where all the free space is labeled as a white obstacle even though it was correctly sent to the model. The obstacles in turn are labeled as free space in the model's map of the world.

4. A discussion of how you would apply the grid filter when competing with a live opponent. Comment on the fact that you might get shot if you spend too much time wandering around the opponent's territory without having a good map, and that asking for the occupancy grid takes time.

If we were scanning for obstacles while simultaneously attempting to capture flags, we would probably double the tanks and only use half for obstacle discovery, and then use those to

create our repulsive and tangential fields for the flag-runner-tanks. We discussed this for quite some time, and we both think that this approach would be our first attempt at being competitive opponent. We know that calculating the occupancy grid takes *forever*, so that's why we thought dividing the problem into two "known" solutions seemed like a promising concept.

We further speculated and thought that we might be able to gain additional advantage by first acting like a dumb agent, and rely only on move_to_position (or add some kind of random potential field, or random turns) to make sure tanks don't get stuck before we have a certain percentage of the occupancy grid completed.

5. A summary of what you learned.

We learned a lot about bayes rule, that's for sure. We also really enjoyed taking a noisy sensor, and using a bayesian filter to discover when it was correct and when it was an error. Before, we didn't really have any kind of intuition on how we could find the stability that a bayesian filter can offer, but now, we can both reason about other problems that use sensor estimations, but may not be accurate (ex: predicting climate with temperature sensors, etc).

We learned that it's ok if you receive false information as long as you know it's incorrect. If you correctly assume everything is wrong it's the same as knowing the truth all the time! We learned that bayesian filters can overcome basically any level of noise (as long as it's above random). The filter just needs significantly more time to gather evidence to discover a map with near-random information noise. We learned that more tanks can discover a map much more quickly and that giving these tanks a greater field a vision can both greatly speed up the search process. The problem with this strategy is that these things require many more resources so there is a clear balance between discovering the map quickly with these methods and having your search run super slowly.

We also learned that it's sometimes nicer to have a server and client that don't require a TCP connection to communicate (this is supposed to be a joke). But really though, we learned a lot about profiling our code and figuring out where the major bottlenecks were, so we could speed things up--adding a tank or two can make a big difference!

6. The amount of time you and your partner spent on the lab.

Garrett: 16 hours
Chris: 13 hours