Mosquito Writeup

Group 4: Hari Rajaraman, Sam Raper, Patrick Wingo

CIS 700

Table of Contents

1 Introduction 3

2 Initial Insights and Observations 4

3 Strategies and Concepts 6

4 Implementation 11

4.1 Voronoi Tessellation 11

4.2 Sweeping 13

4.3 Non-sweeping movement 16

5 Results 17

6 Future Directions & Limitations 31

7 Contributions 33

8 Acknowledgements 34

9 Conclusion 35

# 

# 1 Introduction

In this problem, we were given a 100x100 board filled with mosquitoes that move randomly. Our goal was to place and move lights that “capture” these mosquitos and bring them to a “collector” as quickly as possible. The board may also have any number of “walls” that will restrict the movement of both lights and mosquitos.

We started out with three different implementations on a blank board to get the feel for the problem. We describe them here and will discuss what we learned in the next section:

**Approach 1:** *Sweep from top to bottom.* We docked the collector near the point (50,98), right above the bottom center, and placed the lights at the top. They were spread out such that they collectively ‘cover’ the board’s width. From here we moved the lights in a vertical fashion until they reached the bottom of the board. We then moved them horizontally towards the collector.

**Approach 2:** *Sweep from top to bottom with oscillation.* This approach was effective when the lights didn’t cover the board’s width. If we were to just move it straight down, it would leave out all mosquitos in the uncovered areas of the board. To combat this, we would bring the lights down 40 units and then move it towards the edge of the board and then down 40, then towards the initial x coordinate and form a zigzag motion.

**Approach 3:** *Divide board into section and collect mosquitoes from each section.* We split the board into 16 sections of 25x25 units, small enough for one light to cover the entire section, and placed the lights in the most ‘dense’ sections. Each light then returned to the collector before proceeding to the next most dense section that hadn’t yet been visited. We left one light docked at the collector so that any mosquitos near the collector wouldn’t get lost. When a light left from the collector to go to the next section, it turned itself off in order to prevent dragging any of the mosquitos away from the collector.

# 2 Initial Insights and Observations

We made some important observations while watching our initial players in motion. Some of them were trivial while others were more subtle, and ended up being critical to our future implementations. We’ll list them all for the sake of completeness:

*A light point doesn’t need to cover the entire board.* To capture a mosquito, we need only get near it, having a light touch all 10,000 points on the board is unnecessary.

*The closer a mosquito is to the center of the light, the less likely it is to get lost.* This is true when a light moves around obstacles and makes a “tight” turn around a wall. If any mosquitos are near the edge of the circle, they will no longer be in the “line of sight” of the light and as a result, will no longer follow it. Even one lost mosquito means another light needs to collect it in the future.

Taking these two observations into account, we realized that “sweeping” could potentially be a powerful strategy. We call an area “swept” if the radius of a light has touched every point.

*Creating simple quadrants made the sweeping algorithm simple.* In our third approach, the algorithm was simple: find the next quadrant and go to the center. The sections were so small that the light’s radius easily encompassed the 25x25 grid. As boards got more complex, it was necessary to find a better solution for sweeping.

*Collecting denser sections first resulted in a slightly quicker runtime.* Because mosquitoes have a given probability of moving out of a section, if there are more mosquitoes, then more mosquitoes will drift from that section. Because we swept by section, a higher probability of mosquitoes drifting from a particular section into an already-swept section means that we would have to re-sweep.

*Our final score depends on the last light that reaches the collector.* In fact, our score is the number of rounds that the last light takes to finish, then it took a few more rounds for collection to occur. One of our classmates put it more eloquently: “minimize the longest trip”. This hinted at some sort of ‘collapsing’ algorithm where each light would travel a lesser distance. Of course, even if a few lights take a longer route, we will improve our score as long as the longest light takes the shorter route. A prime example of “minimizing the longest trip” involves the strategy on the blank board: place the collector in the center and spread out each light and then bring each of them towards the center, creating a collapsing effect.

*Only go to the collector when needed.* Our third implementation would sweep a quadrant, drop lights off at the collector, then repeat until all quadrants are swept. This was particularly inefficient as the light makes an unnecessary trip to ‘drop off’ lights whereas sweeping the nearest section would save two trips – one to drop off and another to reach the next section.

*If above statement is true, then it is useless to dock a light at the collector.* The solepurpose of that light is to ensure dropped off mosquitos don’t wander off and it’s no longer needed if no mosquitos are dropped off.

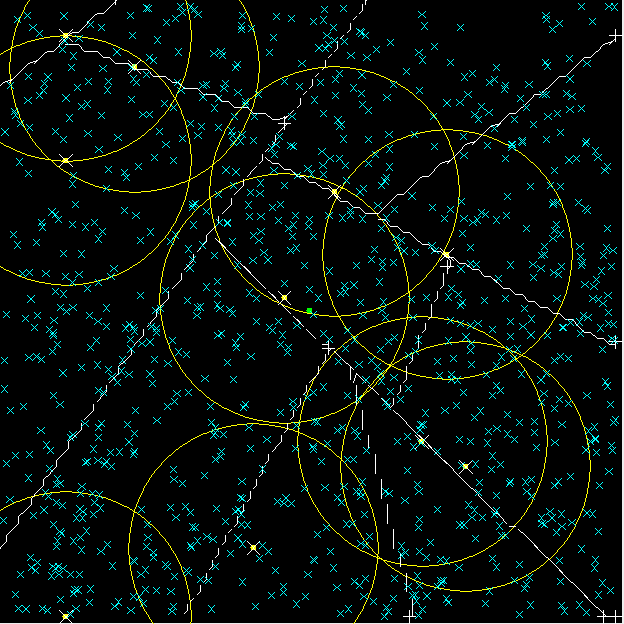
# 3 Strategies and Concepts

Our overall strategy revolved around sweeping any section as quickly as possible and efficiently moving onto the next unswept partition. To do so, we have to come up with three main components:

* Create a sectioning algorithm.
* Create a sweeper algorithm.
* Figure out a best way to move the lights between sections.

We quickly realized that the complexity of the sectioning and sweeper are inverses of each other. The simpler the sectioning algorithm is, the more complicated a section might become. Because of this, the sweeper has to handle all of the various shapes and forms and can make no assumptions. It is safer to statically analyze the board to partition versus creating a location- and obstacle-aware sweeper. Moreover, the sweeper is only efficient if the shape allows it to be. If the same forces the sweeper to have any sort of ‘back-and-forth’ motion, then the whole purpose of sweeping is defeated.

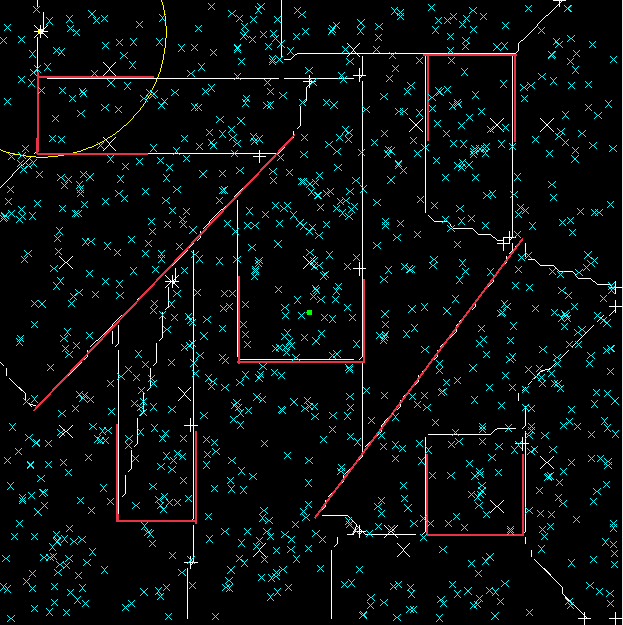
We found that it was better to shift complexity to the section algorithm to keep the sweeper as simple as possible. We thus arrived at our strategy for sectioning, Voronoi Tessellations. We’ll go into more details on the process in the implementation section, but ultimately we end up with sections like those shown below in Figure 1.



*Figure 1: Example of Voronoi Tesselation*

*The Voronoi algorithm divided the map into sweepable partitions.* We hypothesized that static coverage of the map would be an effective proxy to chasing mosquitoes, as we assumed that rate of mosquito movement (without being affected by the light) would be relatively small compared to the rate of collection. In order to do this effectively, we wrote an algorithm that would partition the board into easily sweepable sections. These sections could be split further if extra lights were provided. With this strategy we were able to divide the map up into monotonically smaller, contiguous regions. We also eliminated the need to account for walls when we swept through the region.

The maximum number of sections that a board can create with this algorithm is (# of intersections of walls \* 4 + # of walls \* 2). However, we need at least one section for each light, and maps don’t always produce enough sections using this algorithm. For instance, when we use the blank board, there are no walls or intersections. With no sections, we cannot efficiently place the lights and thus we artificially split sections. Basically, the take the largest section, split it, and continue doing so until # sections = # lights. This process is illustrated in the below two figures where the number of sections is determined by the number of lights.



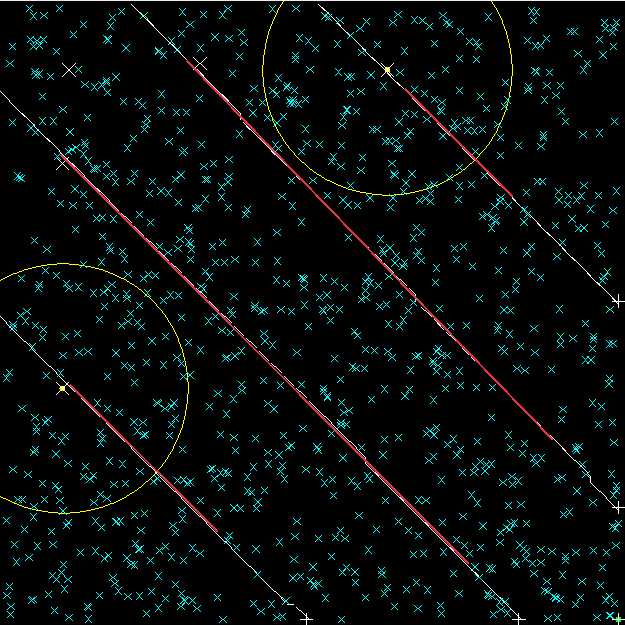
*Figure 2: Boxes and Lines Voronoi Tessellation*

Another facet of the Voronoi strategy was pruning. Because there were so many walls, and sections emerge as a function of the walls, there were initially a lot of sections. In order to reduce the number of times we had to sweep sections and increase the size of sections (there is travel cost to each section, and reduced effectiveness to smaller sections due to relatively large light radius), we checked to see which Voronoi regions had line of sight with other regions, and merged the two. The strategy behind line of sight pruning is that light will travel only across line of sight, but that the radius of lights is fairly large compared to the average size of the Voronoi regions across maps we had dealt with; this means that one light can more or less do the job of two lights if it has full line of sight with the region of the other light. Therefore, we can use combine those regions and put that light to work sweeping another region.

*Lights should converge towards the collector.* To determine our initial light placement we first had to determine the most logical position for the collector. While it was obvious that the lights should start in the most distant sections and converge towards the center the question of what metrics to use remained. In the end we decided on the following:

* For each section, calculate the aggregate path length from every other section to this section
* Place the collector at the end point of the minimum calculated aggregate section
* Place the lights at the sections with the longest path to the collector

By doing the above we were able to put the collector in a mathematically calculated most accessible position. Additionally, the sections that would cause the longest return trip to the collector were swept first, saving time. It is important to note that we placed the collector at the end point of the section. By doing so, we were able to keep the flow of our lights on certain boards such as Parallel, shown below in Figure 2, towards the collector.

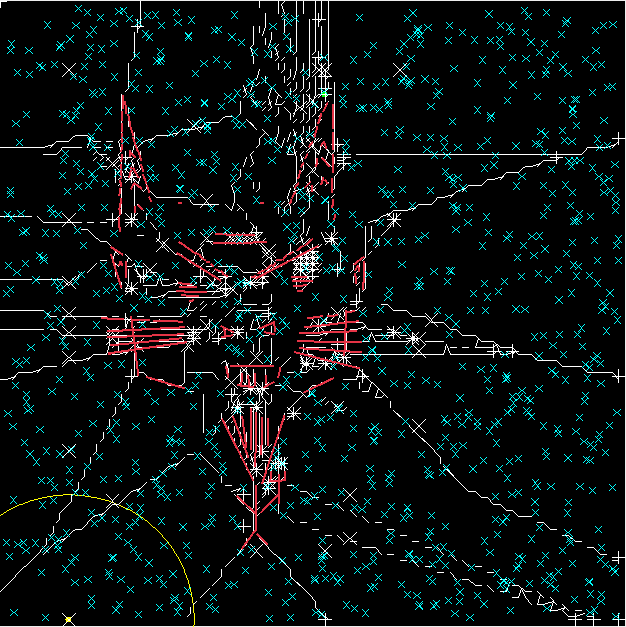


*Figure 2: Parallel board with 2 lights, collector (green) in bottom right corner of board*

*Sweeping should collect all mosquitos in a subsection.* If a section has not yet been swept, then there are no guarantees on whether or not it has been covered in some way by the lights. After sweeping, it is guaranteed that every point within the section has been covered by the light. By dividing the map into subsections and guaranteeing that every subsection is swept, we can achieve full static coverage of the board.

In order to perform sweeping, we move to the leftmost, uppermost point in the section, and move down until the radius of the light reaches the boundary of the section. The light then moves right a constant number of units or until it hits another boundary, in which case it moves along the edge. When our light is within a certain distance from the rightmost point in the section it will stop moving rightwards and do a final updown sweep, insuring that all parts of the section are covered.

While sweeping seems like a relatively simple strategy, there were complexities that had to be taken into account. For instance, we found that there were times when lights, moving along the edges of their assigned sections, would capture a large amount of the mosquitos in regions not assigned to them. In fact, small sections would often be completely covered by other lights before a light had even been assigned to them. On very complex boards such as Satan we created a large number of very small sections as shown below in Figure 3 . These sections were often superfluous and could be skipped altogether. This problem was solved by a simple method that aborted the sweeping process early by constantly checking the number of mosquitos left in the section. If the section was devoid of unclaimed mosquitos, the section would be marked as swept and the light would move on. By applying this method constantly we were able to save large amounts of time.



*Figure 3: We were able to skip many sections in the beard and whiskers by aborting early.*

Another problem we encountered when sweeping was the issue of insurance. There was an obvious tradeoff in making the sweeper move to the perimeter of the section versus having an amount of padding. On the one hand, moving to the boundaries ensured that mosquitos from other sections would be caught in the light. Consequently, stray mosquitos were less likely to fly into the section. However, in the cases where this was not necessary, time was wasted when we could simply have gone within a distance of the edges. In our tests we found that the former strategy was heavily favourable. By making our lights work a little extra at each pass of a section, we were nearly able to cut the need for resweeps of sections altogether.

Once we had determined how to divide the map into regions and search through those regions, we needed to determine how to move lights effectively between those regions and to the collector. For this, we chose the A\* algorithm in order to find the shortest path between two points on the map. The algorithm was fairly trivial, but we realized that the minimum distance caused mosquitoes to be lost when turning corners and navigating around walls, so we had to create a variety of sub-strategies in order to make movement as fast as possible, but keep the mosquitoes on track with the light.

We realized that it was worth sacrificing some speed in the collector movement to allow the mosquitoes to keep up. Because mosquitoes moved at the same rate as the light, but in a randomized fashion around the light, they tended to lag behind while it was moving. In order to keep the mosquitos in a tight formation, we periodically paused the light. We also weighted nodes close to walls much higher than those far from walls, and created a gradient of node desirability depending on proximity to walls to encourage lights to remain reasonably far from the walls. Finally, we noticed that maps were largely man-made, and had lines that were irregular: seemingly straight lines that looked like singular lines had imperceptible gaps and slight slopes. Because A\* looked for the “shortest” path, collectors often tried to go through these miniscule holes. While this made the trip shorter, the small size of the hole caused the collector to lose mosquitoes. At the end of the simulation when we should have caught ~99% of the mosquitoes, we had only caught 80%.. By extending the walls slightly (~.1 squares), we tricked the algorithm into thinking that the shortest path was the one which did not include passing through these “mosquito traps.”

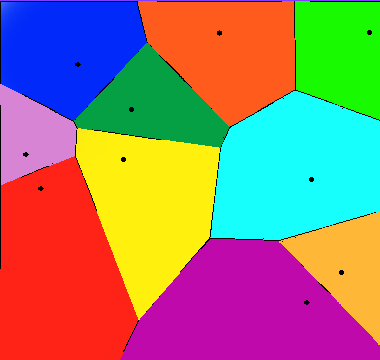
After sweeping a section, a light must decide- where does it go? It has to figure out if it should move to the collector, or if should move to another section, and which. We decided that the most logical approach to this would be to reduce distance travelled between sections as much as possible, so when a light finishes sweeping, it searches through each of the sections, and finds the closest unclaimed section. It then claims that section, moves to it, and begins sweeping. If a light searches through all of the sections and each is claimed, then it begins its final trip to the collector.

# 4 Implementation

In this section, we will discuss how we actually built the three main components discussed above. We’ll take a look at how a modification of Voronoi Tessellations helped us create sections, how our sweeping/placement algorithm was created and finish off with how A\* gave paths directed around walls.

## 4.1 Voronoi Tessellation

Voronoi works in the following ways. Given a grid and set of sectioning points (or Voronoi points or VPs), it will compute the set of points that has the lowest euclidean distance to each VP. While our algorithm was a bit more modified (more on this later), the greatest benefit was that it produced set of points that are contiguous; this is largely because we are using distance as our metric to find the closest set of points.



*Figure 4: Voronoi Map (Credit: Wikipedia page on Voronoi)*

While this was a good concept to build on there were some blanks that we had to fill in. Voronoi takes in two parameters: a grid (or just a set of points in the board) and a set of VPs. What exactly defines these VPs? How do we handle the walls across the boards? With these in mind, we came up with following approach:

To create the set of VPs:

init empty VP set

for every wall:

create an imaginary line that’s perpendicular and cuts the walls midpoint. Now find two points that’s .5 units away either side of the wall on that imaginary line. Add these two points to VP set

for every intersection point x, y (10th grade math):

create 4 points:

x+.5, y+.5;

x-.5, y-.5;

x+.5, y-.5;

x-.5, y+.5;

//this ensures each partition created by the intersection will have its own VP

//and it will guarantee that no section will have an obstacle running through it

add them to VP set

We can run the VP algorithm now, but it will create as many sections as size(VP list). If we have 50 walls and 50 intersections, we’d end up with 300 sections! We noticed with even 80 sections our player started to become slow. To combat this, we did the following to remove certain points:

sort VPs by their X coordinate (because VP was in a set, the iteration was non-deterministic and there’s no telling which points will be removed and which ones determine the sections. Sorting fixed that)

for vp1 = every VP:

for vp2 = every VP:

if vp1 == vp2: do nothing

if vp1 has line of sight with vp2: remove vp1

Removing points will reduce the number of sections and our condition that there’s at least one VP in the same vicinity gave us a vote of confidence that sections won’t be ‘lost’. In other words, if there exists a set of points that are reachable by only one VP, then that VP won’t be removed. While there’s no format proof for this, our observations on various boards sustained our thinking so we went with it.

Now onto Voronoi.

Each point on the grid has a score associated with it. Set to Double.MAX\_VALUE

for vp = each VP:

for gp = each point on the grid:

if vp has line of sight with gp and vp’s distance to gp < gp.score:

update gp’s score with distance

add this point to some section object associated with this vp

remove this point from previous section object

And that’s it! we’ll now have each point associated with some VP and in turn part of some section.

Now we add our final wrinkle: ensuring we have at least as many sections as lights.

We maintain our sections objects in a PriorityQueue sorted by size (descending). So,

while #sections < #lights:

pop section from queue

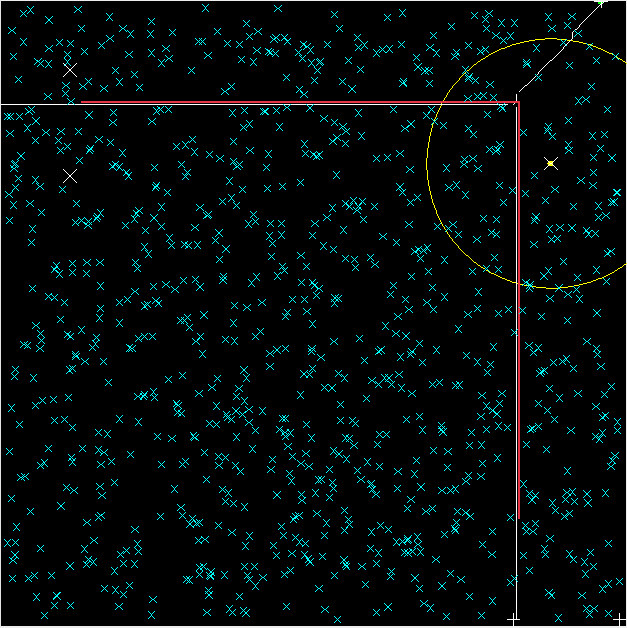
get two random points from within the section

run Voronoi with VP = {the two random points} and GP = {all points within that section}

## 4.2 Sweeping

The sweeping algorithm assumed several important concepts illustrated in Figure 4 below.

* The section has a start point, in the figure denoted by an X. This start point was determined by taking the leftmost, uppermost point of the section and making a padding of at most 12 units around it. This value of 12 units will be used extensively through the sweeper. We found that we could be confident any mosquitos within this distance would be captured without fail.
* The section has an endpoint, in the figure denoted by a +. This point was determined by finding the rightmost point in the section. The light did not actually have to see this endpoint to finish sweeping, we merely wanted to know the x value and be within our confidence, 12, of that. The placement of the collector is on the endpoint of one of the sections.
* The sections will have no walls in them. Thanks to Voronoi, we were able to assume that all sections would be unobstructed, vastly simplifying the problem.



*Figure 5: Right Angle board with three sections*

When a light reaches the start point of the section that it is assigned to the algorithm can begin. Since we know that our start point is both leftmost and uppermost, we always begin by moving downwards. When the light has reached a boundary or is within our confidence from the edge of the board, we try to move right for, again, 12 steps. If we reach a boundary of the section, we move along it until we’ve made our required 12 steps to the right. At this time, we begin sweeping upward. In this way we progress across the section.

To determine when the light is finished sweeping the following logic is applied:

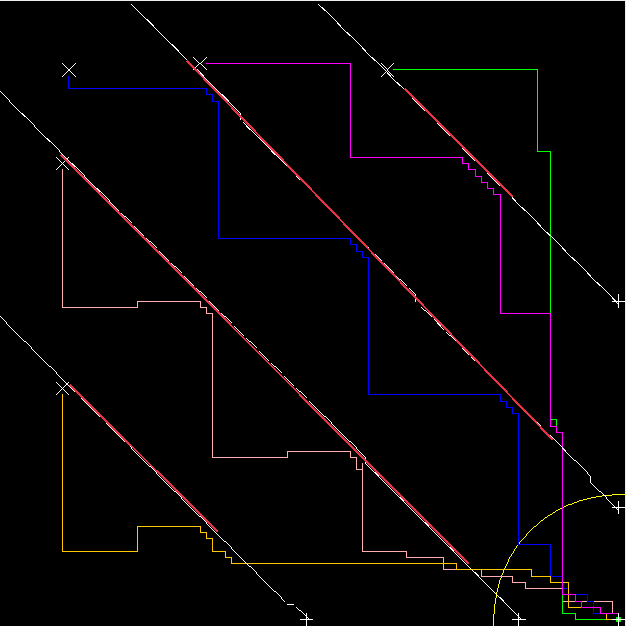
if light position x + 12 > rightmost point for this section

do one final sweep up or down sweep in this section

signal that section is done being swept

As this condition will only be met when the light is moving rightwards it is important to do one final sweep. The reason for this is that you may have moved up to 11 moves to the right before you see that you are within the proper distance to the rightmost point. To not sweep one last time up and down could leave mosquitos in the section.

A toy example of the sweeper logic can be seen in Figure 5 below. The five lights each start at the beginning of their section and work together to sweep the entire board before moving to the collector to deliver the mosquitos.



*Figure 6: Parallel board with 5 lights after being swept*

An important aspect of the sweeper that was touched on early was the ability to abort early. The logic for this was quite simple.

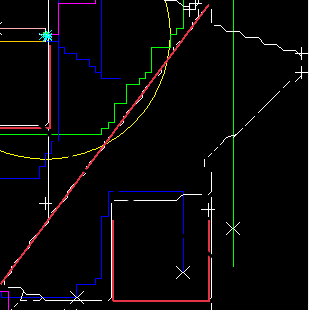
For each point in the section:

if there is a mosquito on that point

continue sweeping

abort sweeping

This was an important part of our strategy as it allowed us to react to the position of the mosquitos at runtime. While we still felt that a static approach could be effective, we realized that not taking the particular configuration and status of the board was foolish. Especially on boards with many walls and thus many Voronoi sections, we were faced with sections that would accidentally be collected by other lights. The large radius of the lights heavily influenced this behavior. The abort early logic can be seen in the below figure from the Boxes and Lines board:



*Figure 7: Skipped section on Boxes and Lines board*

Observe in the above section how the dark blue and green lights at the bottom of the figure incidentally sweep the large section in the middle. The dark blue, then dispatched to that section, skips it entirely as it sees that it is devoid of mosquitos to collect.

This logic for marking sections as done early was checked constantly throughout the simulation, not only when lights were sweeping.

## 4.3 Non-sweeping movement

When the lights were not sweeping there was still a substantial amount of decision making going on. The first problem that we had to tackle was dispatching to new, unclaimed sections. This was done by selecting the numerically next unclaimed section. So if a light had just swept section 10 and 3 was open, they would then move towards section 3. However, if, on their way to section 3, they crossed into an unclaimed section, they would relinquish their claim to section 3 and immediately begin sweeping whatever section they had stumbled upon. The reason for doing this was that moving over sections that would remain unswept ultimately led to a lot of wasted time. If a light were on an unclaimed section, it mine as well sweep it. Note that all inter-section movement was determined using A\*.

Once all sections had been claimed (either swept or in the process of being swept), the lights would simply move to the collector. Lights never dispatched once they were at the collector. SInce the collection target was always 100%, there was no reason to come to the collector before everything had been swept.

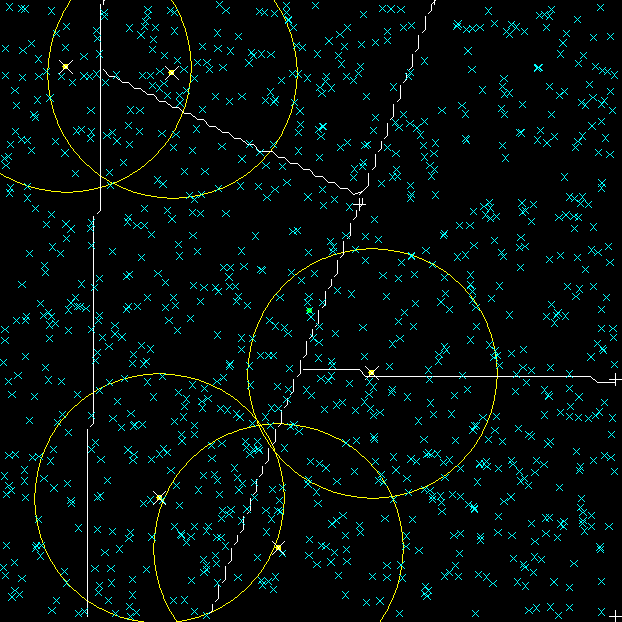
# 5 Results

When we looked at the raw summary statistics for the tournament, we were admittedly discouraged by the behavior of Voronoi Player. We decided to graph each of the results for visual analysis and to add standard deviation, and we were often surprised by the extremely high standard deviation in the graphs- we often had the highest standard deviation of any player for a given gameplay configuration. So we went through and analyzed each of the 5 lights configurations against the other players to have a generalized idea of gameplay performance.

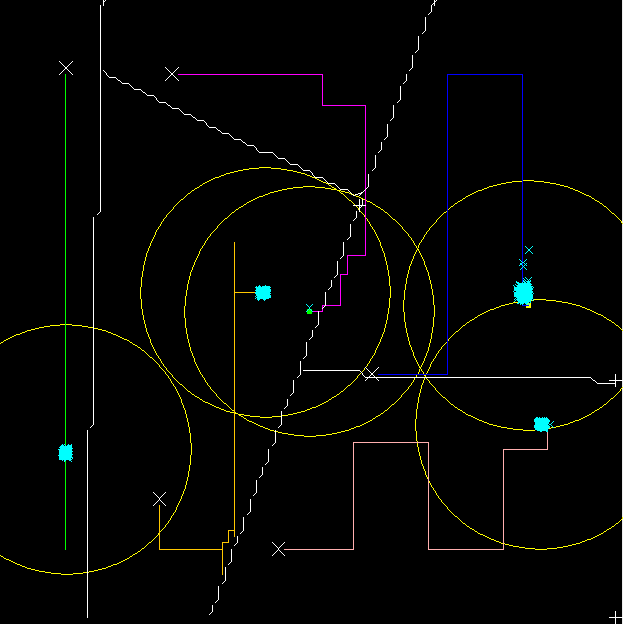
# 

*Figure 8: Performance Graph on Blank*

Blank 5 lights looked pretty bad with an average above 2000, but we didn’t remember results being that bad! However, the standard deviation is huge, implying a large variance in the results. Perhaps we had good performances and really poor performances- something to be analyzed later. However, by looking at these results, we discovered a serious flaw in our approach that severely limited our performance, especially on a board like Blank. While other strategies either swept statically using an MST or went after the mosquitoes directly, implying a more or less straight path to the center of the map, along with relatively even path distances for each of the lights, our strategy divided the map into even sections and swept through those sections before returning to the center. Both sweeping distances and distances to the collector were non-uniform. We will further demonstrate this using the screenshots below.



*Figure 9: Blank, 5 Lights, Initial Sections*

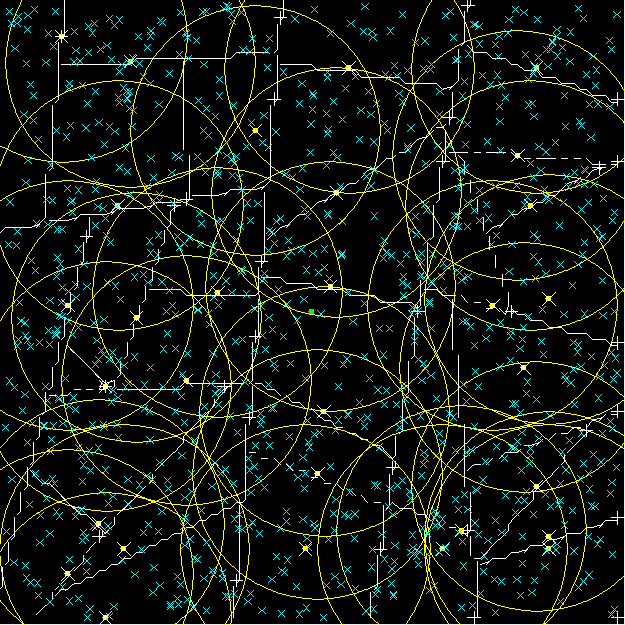


*Figure 10: Blank, 5 Lights, After Sweeping*

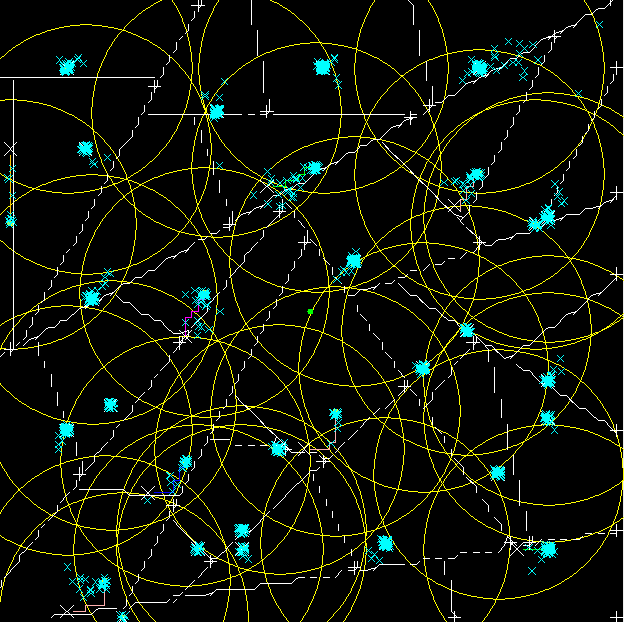
Notice how the sections are somewhat uneven, so that the sweeping times are not going to be constant. This is the first inefficiency we noticed in our analysis of blank performance, as the actual performance time is dependent on the longest trip of the last-returning light. Not only did we need to keep sweeping distances constant, which our strategy did not do, we needed to minimize the total distance sweeped.

Additionally, the Voronoi Tessellations were based on the position of the walls. When no walls were present, the algorithm would create only one section. Creating new sections by breaking down larger ones was done randomly and thus we often ended up with awkwardly sized and shaped sections.

Our strategy did not minimize the distance sweeped because of a fundamental flaw in our initial approach to the problem: our voronoi sectioning was based off of the idea of identifying closest points to a singular starting point. Imagine that fire stations have been placed throughout a city at fairly uniform intervals, but not completely uniform. Voronoi sectioning would be ideal in determining which portions of the city were served by each fire station, as the goal is to minimize distance and travel time to the singular point where the fire is, then get back as quickly as possible to be ready for the next fire; you don’t know where the next fire is going to be, or how long until the next fire. By definition, Voronoi assigns sections to central points by minimizing the distance function from central points to individual points in each section. Now, imagine that every house in the city catches fire (this is the apt analogy for the Mosquito problem); the Voronoi analysis breaks down because the fire truck is not going back to the station every time.The flaw with our logic is that we are not trying to move to an arbitrary point and back, therefore we should try to optimize for such a trip.



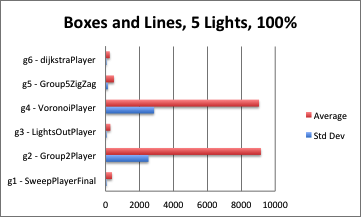
*Figure 11: Blank, 30 Lights, Initial Sections*



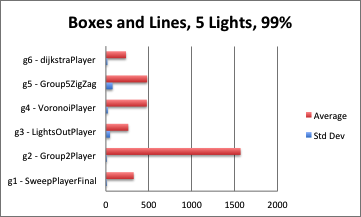
*Figure 12: Blank, 30 Lights, After Sweeping*

The second inefficiency we noticed based on performance analysis of the Blank map is more easily illustrated when we up the number of lights to 30. Notice how each of the sections is relatively even, but for sections on the end, the distance to the collector will account for about half of the travel the light does. For a high number of lights on a board like Blank, the ideal strategy would be to place the collector in the middle and each of the lights around the outside in a circle, then to sweep inward, cutting the board like a pizza. We realized in our analysis that we should try to take distance to the collector into account; the strategies of other teams inherently do so when they automatically check if mosquitoes within the assigned section are captured by the radius and line of sight, then move to the collector automatically, or when they dynamically search for mosquitoes.

We, on the other hand, completely sweep each section we’ve already started to sweep, then if no more sections are assigned to the light, move to the collector. In calculating sections, we do not taken into account the radius of other lights, nor do we optimize the total distance traveled including distance to collector. In fact, we have very few optimizations on total distance. While we abort sections that have no more mosquitoes to collect, sections can only be aborted before they are started. This strategy sweeps the board in a way that minimizes both individual and total light travel distances. Simply by analyzing our performance against that of other teams on a blank board, we have already figured out some major problems with our assumptions and approach.

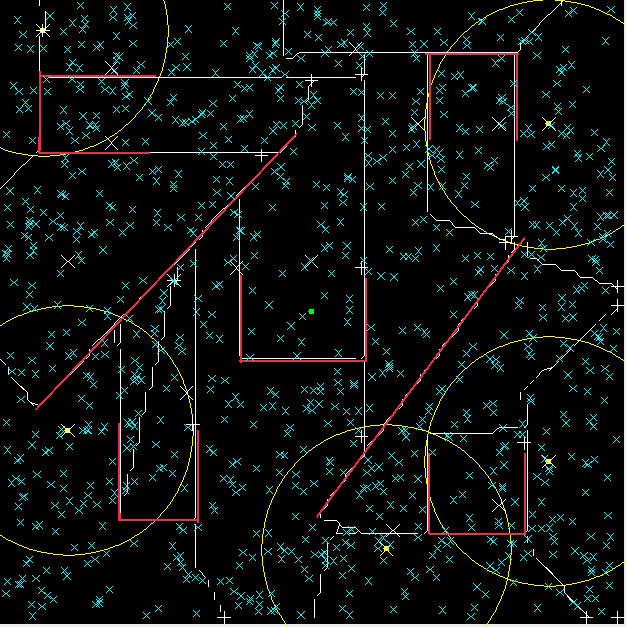
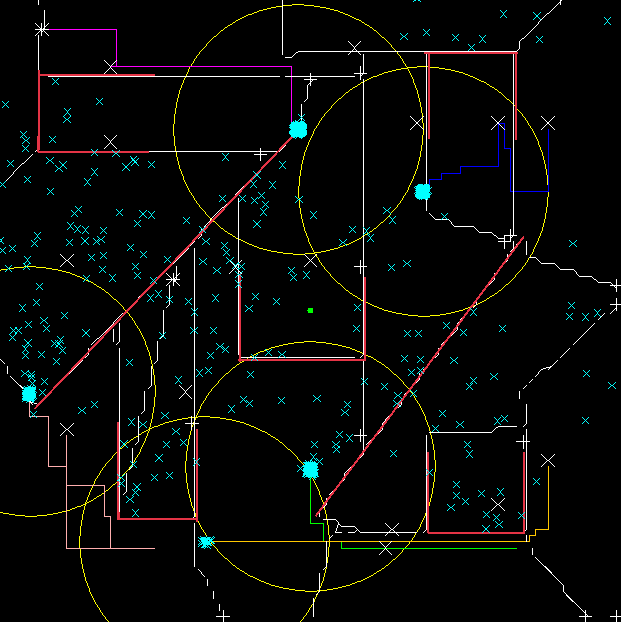


*Figure 13: Performance Graph of Boxes and Lines, 5 Lights, 100%*

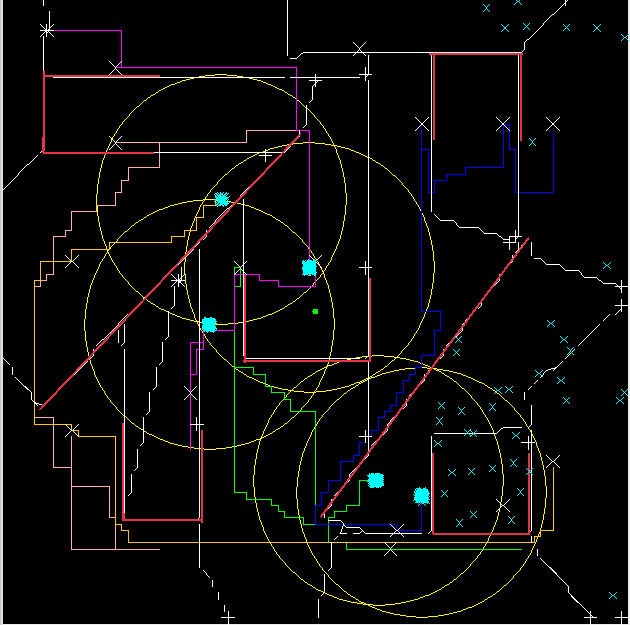
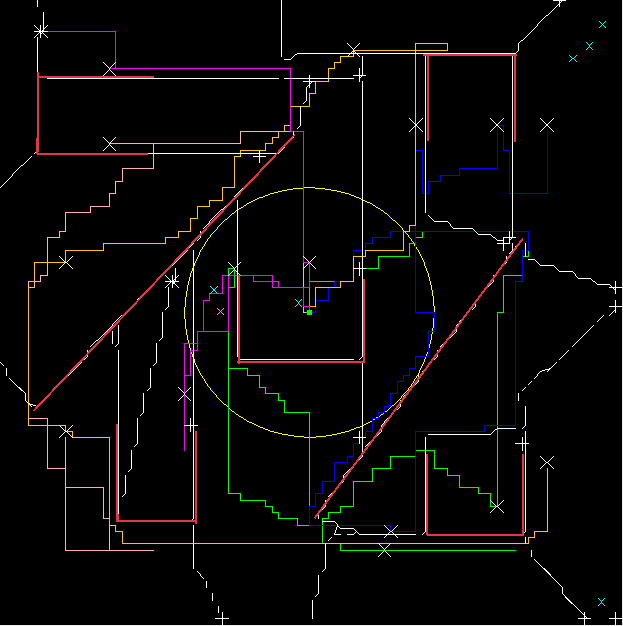


*Figure 14: Performance Graph of Boxes and Lines, 5 Lights, 100%*

We performed pretty poorly (last but not least?) on Boxes and Lines, 5 lights, 100%. Our average was around 9000, but we noticed a huge standard deviation of 3000, implying that we had both poor and not-so-poor results. While we pointed out some serious flaws in our strategy based on the Blank map, those flaws are somewhat masked by the addition of obstacles, which make sectioning almost implicitly existent. The addition of walls creates sections which naturally go together regardless of the mosquito pursuit logic, and traveling to an arbitrary point becomes somewhat equivalent to traveling to a section of arbitrary points because of small section size with the addition of obstacles. Because of this, we thought Voronoi sectioning was actually not that bad compared to other strategies with the addition of walls, but this didn’t show in the 100% results. We were a little confused because performance in our own tests seemed to be much better, but we hypothesized that the vast majority of mosquitoes were being captured, but some mosquitoes were getting lost on the trip back. This was reinforced by the same map, same lights, but 99% capture time, which showed an average that puts us in the middle of the pack, with a very low standard deviation. This leads us to believe that we have an inherently pretty good performance with obstacles, but our implementation is flawed in that we lose mosquitoes either while sweeping or while traveling in between sections and to the collector. In summary, we have a serious issue where we miss a few mosquitoes in most of the trials, but our strategy consistently performs very well excluding those few mosquitoes. We wanted to better understand the source of high variance at the 100% level and low variance at the 99% level, so we ran a few trials of Boxes & Lines with 5 lights. We analyze one of those trials below.



*Figure 15: Boxes & Lines, 5 Lights, Initial Figure 16: Boxes & Lines, 5 Lights, 80 turns*

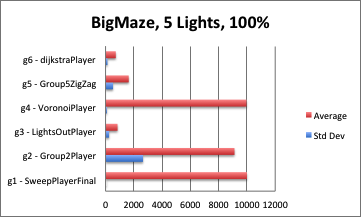


*Figure 17: Boxes & Lines, 5 Lights, 200 Turns Figure 18: Boxes & lines, 5 Lights, Finish*

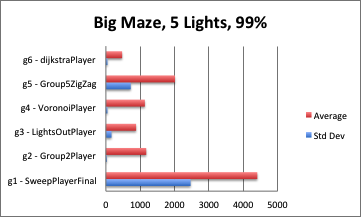
While sectioning is fairly effective in distributing the work among the lights for maps that have a moderate number of walls, we notice a few issues, especially when there are more sections than there are mosquitoes. Notice how in round 80 above, the right-hand section at the top-right has been swept, but the left-hand section at the top-right has not been swept. Some of the mosquitoes that were previously in the left-hand section actually traveled from that section to the right section after that section had been swept. Even though it contains mosquitoes, it is no longer a section we need to sweep according to our algorithm. By the time all the lights go to the collector, even though the left-hand section has been swept, not all of the mosquitoes which drifted from section to section had been collected. As a result, we lose a small percentage of mosquitoes to drift.

Another issue we found in analyzing this Boxes & Lines trial was that mosquitoes still got lost when rounding tight corners. Notice how at 200 rounds, all of the lights are headed back to the collector, and the majority of mosquitoes have been collected (there are a few left over because of the issue described above.) However, when one of the lights rounds the final corner to the collector in the middle, it leaves a few mosquitoes behind (see Boxes & Lines, 5 Lights, All Lights In) because they are no longer in line of sight. This is the other major way that we miss a small number of mosquitoes, accounting for the significant difference in our performance between 100% and 99% collection.

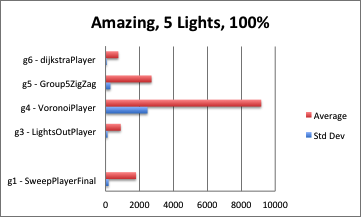
We saw a very similar picture with results from the other maps: we performed fairly well up until the 99% of mosquitoes, but had issues with bringing back the last few mosquitoes. Our performance is graphed below.



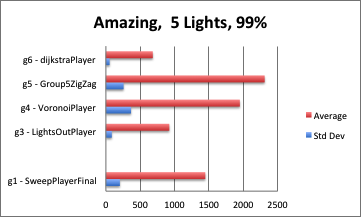
*Figure 19: Performance Graph of BigMaze, 5 Lights, 100%*



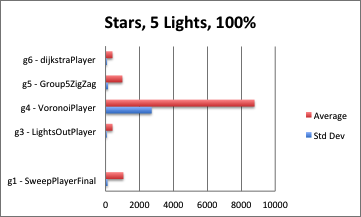
*Figure 20: Performance Graph of BigMaze, 5 Lights, 99%*



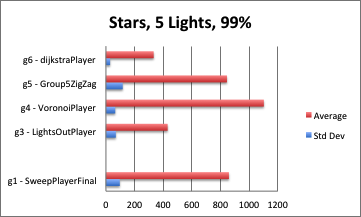
*Figure 21: Performance Graph of Amazing, 5 Lights, 100%*



*Figure 22: Performance Graph of Amazing, 5 Lights, 99%*

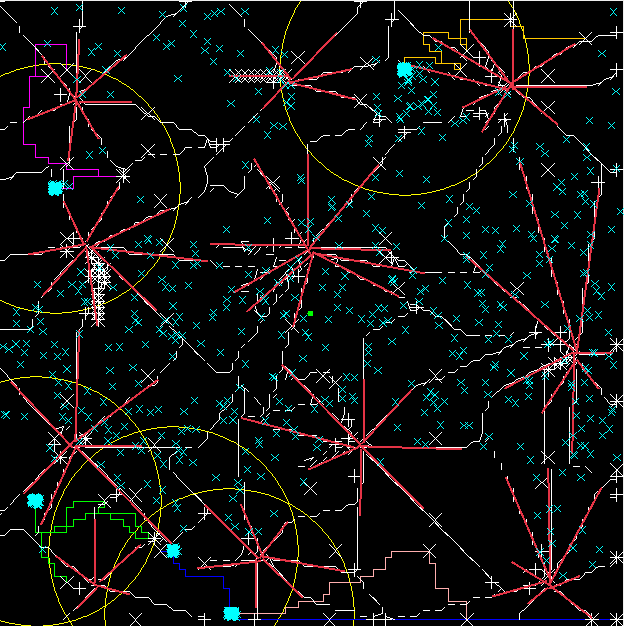


*Figure 23: Performance Graph of Stars, 5 Lights, 100%*



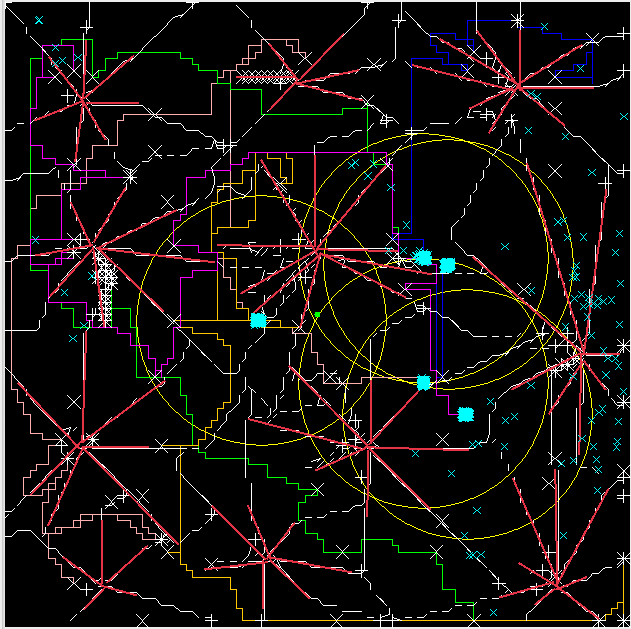
*Figure 24: Performance Graph of Stars, 5 Lights, 99%*

While analysis of BigMaze yielded more or less similar results, we found our performance on Stars at 99% to be lacking compared to the that of the other boards. While our average performance improved from around 9000 rounds to over 1000 rounds (significant) when switching from 100% to 99%, we are still in last place for 99%, and the low standard deviation shows that this is not a “mosquito loss” problem. Instead, it’s a fundamental issue with the way that we collect mosquitoes for stars. An ideal strategy would have lights circle each of the stars, but not go into any of the sections, because the radius of the light would cause all the mosquitoes in each of the section to be attracted to the lights and to move to the lights. However, our lights are “unaware” of this, because are sections are constructed partially from walls. Once a section has started being swept, it must be fully swept, even if all the mosquitoes are already attracted to the light. Not only do lights go too far into the stars because of the way we section, but we also have issues with 5 and 10 lights as a result of the way we decide what section to go to.

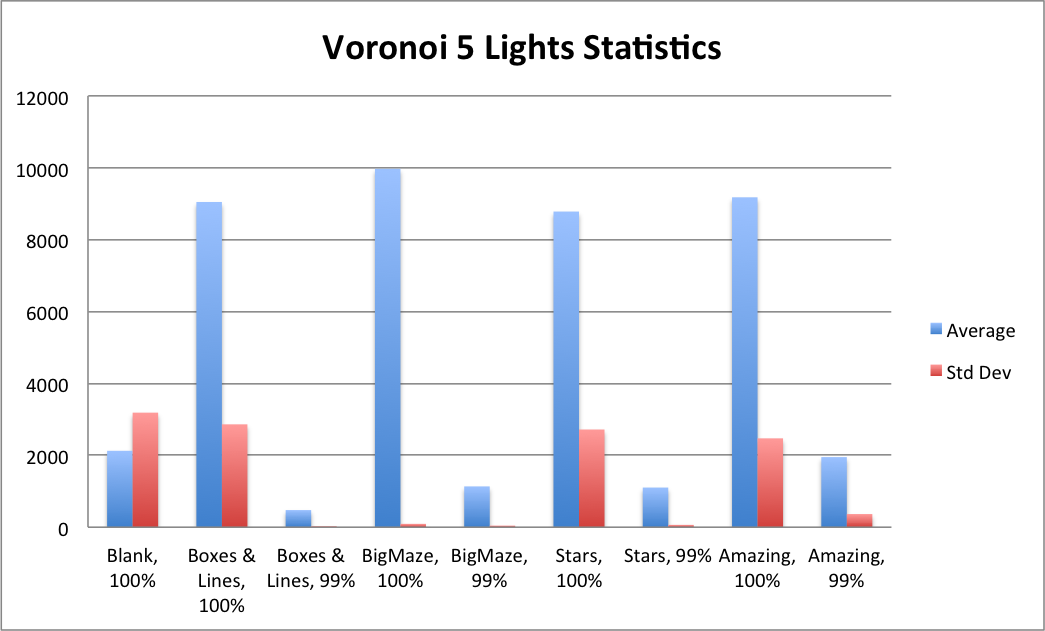


*Figure 25: Stars: Lights going into Arms*

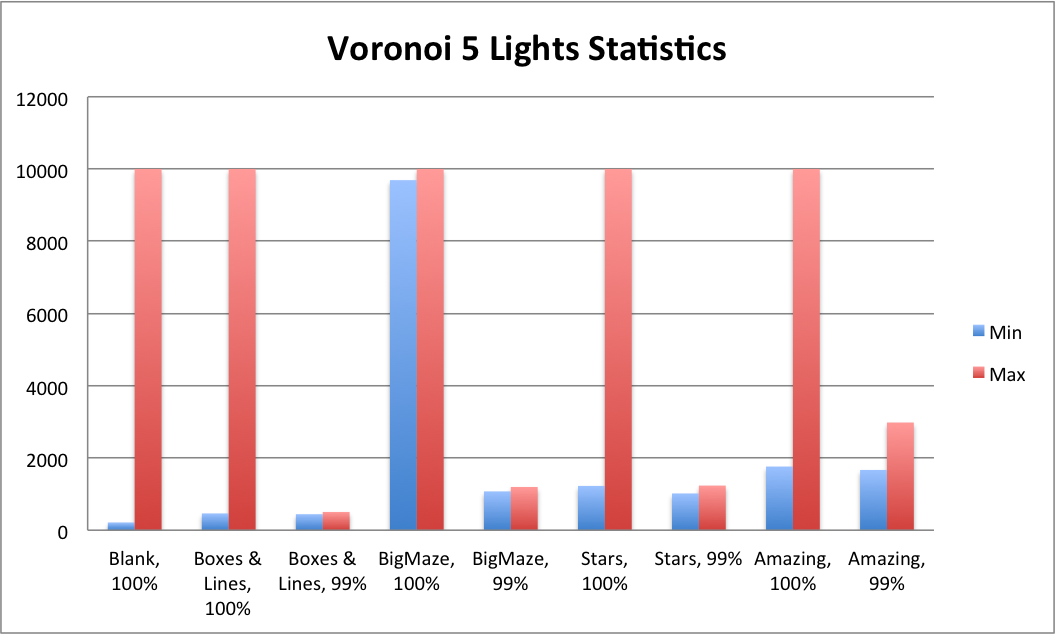
We did some analysis into what we thought were problems with scaling the number of lights, as we noticed that whenever we ran trials with 5 lights, they would begin to stack on one another and lose effectiveness. 5 Lights tend to act sometimes like 4 lights and sometimes like 3 lights, as they tend to pile up and sweep sections together, making the sweeping process take much longer. The same thing happens with 10 lights, but our numbers tend to decrease as the number of rounds increases. This leads us to believe that there is an issue with the code that assigns sections to lights, leading sections to be double-assigned. We looked into the code, but could not find anything yet that would be the cause of this error, but our strategy explicitly forbids doubling up of lights on sections, implying that this is a problem with our implementation- perhaps a race condition. Notice in the figure below how light paths are very similar, and how close lights are in proximity. This should never occur in our strategy- instead, lights should fan out in a fairly random fashion because they start distributed fairly evenly over the board.



*Figure 26: Stars: Lights “piling” on top of each other*



*Figure 27: Performance Graph of VoronoiPlayer Average and Standard Deviation*



*Figure 27: Performance Graph of VoronoiPlayer Minimum and Maximum*

Everyone knows that in life, you need to compete against yourself. So we wanted to analyze our own results next to each other. In the opinion of the team, a score less than 1000 on any map is a decent score, a score of less than 500 is a great score, and a score of less than 300 is fantastic. Using those very simple heuristics, we can see where do we do well and where we do poorly- we will of course have to adjust somewhat depending on map difficulty. On each of the 99% maps, we’re actually pretty good! We get less than 1000 on each, except for Amazing. We’ll get to that in a bit. We wanted to compare our best performances with our worst performances to try to understand a bit further about each map, and to tease out any more potential issues that we could work on. However, we only saw huge differences between min and max on the 100% mosquitoes trials; we know that this is a result of a few mosquitoes being lost a portion of the time. However, we did see a fairly high difference between min and max on Amazing.xml, and found that it was probably the result of a race condition. Let’s explore that further.

We looked into the performance for Amazing.xml step-by-step and noticed a couple issues with the map. All of the tight corners caused us to lose mosquitoes as we traveled around; our mosquitoes got stuck because of an unknown race-condition type error; our sectioning was not as efficient as it should have been: lights explored part of a maze, then moved on to do another part of the maze without continuing down and finishing up the part of the maze it was in, therefore, another light had to come in and finish it up. This helped to expose a further issue with our choice of section movement: it was done using raw board distance rather than the A\* distance. As a result, sections that were physically close in the maze but hidden by walls were assigned to lights, and this caused much more movement than was necessary. Because sections are assigned when they finish, a race condition occurs, and part of the time, the “right” light may be assigned to the section because the nearest by is not yet done, but the light assigned is often the “wrong” one due to our use of physical proximity. This caused us to perform poorly compared to other teams, who likely used A\* or Dijkstra’s distance to find where the light should go next.

# 6 Future Directions & Limitations

Our analysis of the results led us to believe that we would need significant strategic overhaul in order to optimize the player. For each of these limitations, we have attempted to think of a solution in order to fix the problem. Specifically, we found strategic 5 points that we believed could lead to a much more effective player.

1. Sweeping should actively check to see if all of the mosquitoes within a section are already within radius and line of sight, minus some margin of safety. If so, abort the section and continue to the next section. This could save significant time on larger sections that may have been partially covered by other lights already.
2. When moving around walls, lights should wait till all of the mosquitoes within line of sight and light radius are within a certain smaller radius in order to prevent mosquito loss in moving around the wall.
3. Our method of choosing the next section not only has the physical distance bug described in the section above, but it is faulty in its strategy. We did not taken into account final travel to the collector in choosing sections to travel, and ended up often traveling to the collector, then back out away from the collector. A future strategy take into account the A\* distance of a section to the collector (attempting to maximize) while minimizing on A\* distance of a section to the light.
4. Our strategy does not address mosquitoes “lost,” even though there are many ways for mosquitoes to not be collected by the lights. Therefore, we should implement a “hunting” strategy that checks for nearby sections that have already been swept, but where there are mosquitoes. Assign the nearest light to go fetch those mosquitoes, then let it return and continue sweeping or traveling to its destination.
5. Our strategy for assigning sections is wrong because it is based on a faulty assumption of distance minimization. We should use a different way to section, rather than Voronoi, because it optimizes on the wrong metric. Our sectioning should minimize the sweeping distance while keeping in mind travel distance to the middle, as all lights should arrive in the middle at the same time under ideal conditions.

**Implementational Future Directions & Limitations**

Based on our above analysis, there are more issues with our strategy than with our implementation. However, there are some race conditions and other bugs which we need to fix in order to tune our performance. Furthermore, there are some optimizations we can make on our existing strategy in order to increase the speed of mosquito collection and decrease the potential for mosquito loss. We’ve listed our findings below.

1. Optimize on A\* to ensure no mosquito loss from traveling around corners; modify the speed of travel to find the fastest rate at which lights do not lose mosquitoes
2. Fix the race condition involving light “pileup,” which manifested itself in “Stars.” Lights tended to move together in relatively similar or same paths, which reduced the parallelism we could normally exploit from having multiple lights to gather mosquitoes.
3. Optimize on our sweeping algorithm in order to find the best tradeoff between safety and speed: certain margins of safety were faster than others but less safe, and certain methods of sweeping also presented these issues. We need to further explore and experiment in order to find ways to sweep that are fast, but that do not lose mosquitoes.
4. In computing which sections to move to, we currently use physical distance to find the nearest un-sweeped section. We should instead use A\* distance because walls could stand in the way and cause significant travel time to the next section.
5. We found another race condition which caused lights to become “trapped,” and to have a cyclic movement pattern, often up several steps, then down several steps, causing them to waste many turns. There was even a probability that stuck lights would not return to the collector, causing the score to “max out.” because many mosquitoes never reach the collector.

# 7 Contributions

Our solution was largely unique in its use of sections and a sweeping algorithm. The Voronoi Tessellations were unique to our group and while other groups tried to break the board down into sections, ours was the only one to do so with satisfactory stability and consistency, despite the flaws.

The sweeping algorithm, a large part of our solution, was important as it made the size of our sections unbounded. Just as the sectioning algorithm helped the sweeper by providing predictable, simple sections, the sweeper helped the sectioner by being able to handle a variety of sections with ease.

Additionally, our combining of static and dynamic solutions was largely novel. As we watched other groups develop, we realized that the more reactive groups were tending to do better than those who were strictly trying to see every point on the map. Taking this into account, we decided to add functionality for viewing the board without abandoning our overall strategy.

Lastly our implementation of A\* with weighting nodes close to the wall was novel. Many other groups got around this problem by putting waypoints several points away from the end of walls. Our solution was able to weigh the consequences of going through a small hole and determine whether it was worth the moves to go around a large obstacle.

# 

# 8 Acknowledgements

* Extending the walls artificially to compute distance
* Pausing the lights briefly in order to centralize mosquitoes
* A\* as a potential algorithm to get around the walls in a distance-minimizing fashion
* Wikipedia for Vorornoi Algorithm
* Wikipedia for A\* algorithm pseudocode

# 

# 9 Conclusion

We thought that project was a really interesting project, and what started out as a trivial solution, requiring only the number of lights as an input, turned into a highly involved, dynamically calculating, graph theory-involving, mosquito-munching machine. We had some issues, as we demonstrated (and quite honestly, found while working on) in the results and analysis section. In fact, our strategy was based on some incorrect assumptions and faulty logic, but they turned out to be close enough to the truth to perform decently in many of the simulations. We found this project to be challenging and fun, but to also be highly time-consuming- much more so than the previous project. In addition, there was no backing down from a strategy.

Because of the highly intensive nature of implementing a strategy and getting rid of the bugs, it was extremely different to change strategies because the entire process would have to be repeated. As a result, we backed ourselves into a corner basing our entire strategy on static analysis of the map, and it would have been extremely difficult to get ourselves out of that corner by starting over with a new strategy. Perhaps a method for making strategic experimentation more viable would be to provide students with a better API to control the movement of mosquitoes. We found that their movement and control was not very abstracted, and it took a lot of code just for us to be able to move them effectively. However, a lot of this code was tailor-made to work with our initial strategy, making it much harder for us to switch in between strategies.

We did enjoy trying to find inefficiencies and optimizing on our code, as well as the examination and debugging process for this particular project. The results were much more deterministic in this project, which make the isolation of issues and the visualization of strategies and gameplay much easier to deal with in comparison with the previous project. Overall, we enjoyed the project and wish the teams in future years good luck; our advice: “go after the mosquitoes.”