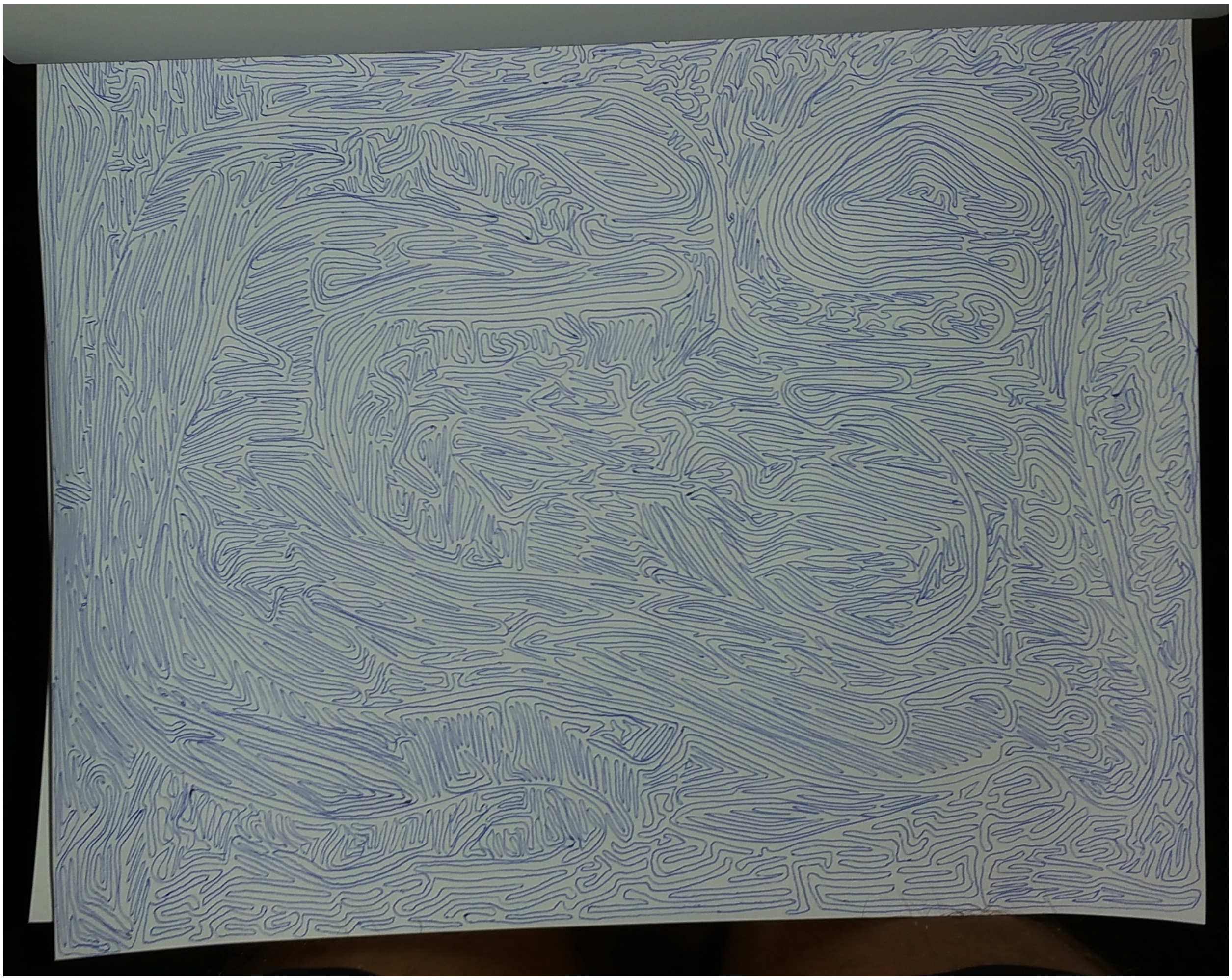
Wavy Line Search Problem
CS 221 Final Project — Final Report
Fall 2017

Mateo Garcia



Task Definition

My system generates a "wavy line." Shown above is a wavy line drawn with a pen. It covers the entire page with a single line. To approximate this, my system draws a single line through a grid of points. The task of my system is to fill as much of the grid as possible with this line. As the merit of this work is artistic, resulting outputs are evaluated by humans.

Infrastructure

I chose to use Processing in Python for my project. Processing is a programming language and integrated development environment that allows you to program drawings and animations.

Approach
As my project title suggests, I chose to model this task as a search problem. I defined the search problem as follows:

- *Start state:* 2-D array of grid locations, starting point
- *Actions:* Move forward, left, or right
- *Cost:* Based on the distance of the current point from the starting point, the distance to the nearest page edge, whether the successor state has three or more visited points surrounding it, and some randomness
- *Successor state:* A new 2-D array of grid locations, with the last current point updated to point to the new current point
- *End state:* True if all surrounding points have been visited

The primary element of experimentation in this project was composing the cost function for the search problem. How I arrived at this cost function is further discussed in the Error Analysis section.

At one point, per the suggestion of CA Steven Mussman, I attempted to formulate the problem as having two actions: Move clockwise or move counterclockwise around the starting point. At each step taken, with some probability the line would begin moving in the opposite direction. The idea was that this would cause the line to emanate outward from the starting point by wrapping back and forth around itself in a circular fashion. To encourage the line to fill the rectangular page, costs could be assigned based on how square the resulting line pattern would be after taking an action. I unfortunately did not get around to fully implementing the search problem in this way.

After several discussions with the CAs I found the most appropriate search algorithm for this search problem to be depth-first search with iterative deepening (DFS-ID). My implementation composes a wavy line of segments found using DFS-ID with a fixed depth. The search runs to a depth d, selects the lowest-cost path of those with the greatest length (at most d), and then repeats, beginning at the endpoint of the selected path.

I should also mention that the size of the state for the wavy line search problem is prohibitive to the use of certain algorithms. Because the state includes the entire grid of points, this search problem does not lend itself well to, for example, uniform cost search, which would need to compare entire grids in order to check whether a state has already been explored.

The oracle for this task is a hand-drawn line that fills an entire page and achieves aesthetic beauty. The baseline is a basic line that is aesthetically interesting, but does not fill the entire page.

Literature Review
From what I could find, this is the first attempt to build such a system. There is work on maze generation, but the mazes generated are not composed of a single, non-branching line, which is what distinguishes my system's task.

Procedural generation is an umbrella term used in game development and other related spheres to randomly generate graphics that appear as if they were created by a human. I could not find any work within procedural generation that related specifically to my system's task of working with a single line.

Error Analysis
In this section I will walk through the progression of my approach to solving the wavy line search problem, as documented in the Appendix section. As I mentioned in the Approach section, determining an appropriate cost function turned out to be my primary target of optimization in solving this problem. My secondary target was finding what fixed depth made DFS-ID most effective. Due to the nature of creative work like this, I did not cleanly optimize for each one at a time, but rather found myself optimizing for both simultaneously.

My baseline approaches are shown in Figures 1, 2 and 3. Figure 1 shows the first wavy line generated by my system. The costs are completely random, and as a result, the line has a wandering behavior. Figure 2 demonstrates the effect of setting the cost to equal the distance of the current point from the starting point. Figure 3 shows this effect when starting from the center of the grid. While Figure 2 may appear more stochastic than Figure 3, it similarly uniform in the way the line traverses back and forth from the top of the grid to the left side.

Figures 4 and through 8 demonstrate the combination of these two basic factors in the cost of each action: distance of the resulting current point from the starting point and uniform randomness. Some wavy lines, like that in Figure 5, did in fact reach to the edges of the page (in a similar fashion to what I imagine would have resulted from a clockwise-counterclockwise formulation of the search problem's actions). However, I find wavy lines like these to be less aesthetically pleasing than those where the randomness factor is higher—like in Figure 7. This highlights a key tension in this search problem: A balance between stochasticity and uniformity in forming a wavy line. The former leads to more beautiful results, while the latter brings us closer to solving the task, which is to cover as much of the grid as possible.

In a first attempt to relax this tension, beginning with Figure 9 the cost additionally depends on the distance of a point from the nearest edge of the grid. This component of the cost is smaller for points that are closer to the edge.

Figures 10 and onward make a second attempt to relax this tension by including the last component I added to the cost: the number of visited points surrounding a successor state. My hope was that this would incentivize the wavy line to veer outwardly into unexplored territory. As Figure 10 shows, increasing this cost for each visited point caused the line to no longer wrap itself efficiently. So beginning in Figure 11 I only imposed a cost for 3 or more visited points. The result of this additional component to the cost overall increased the length the wavy line.

Unfortunately I was not able to fill an entire grid. I believe this was because my cost function was still very elementary at the end of my experimentation. It is likely that there are more complex features that could be explored to this end, such as the considering the geometry of the outline of the wavy line's pattern.

Appendix

Changes in conditions between figures are indicated by *italicized text.*

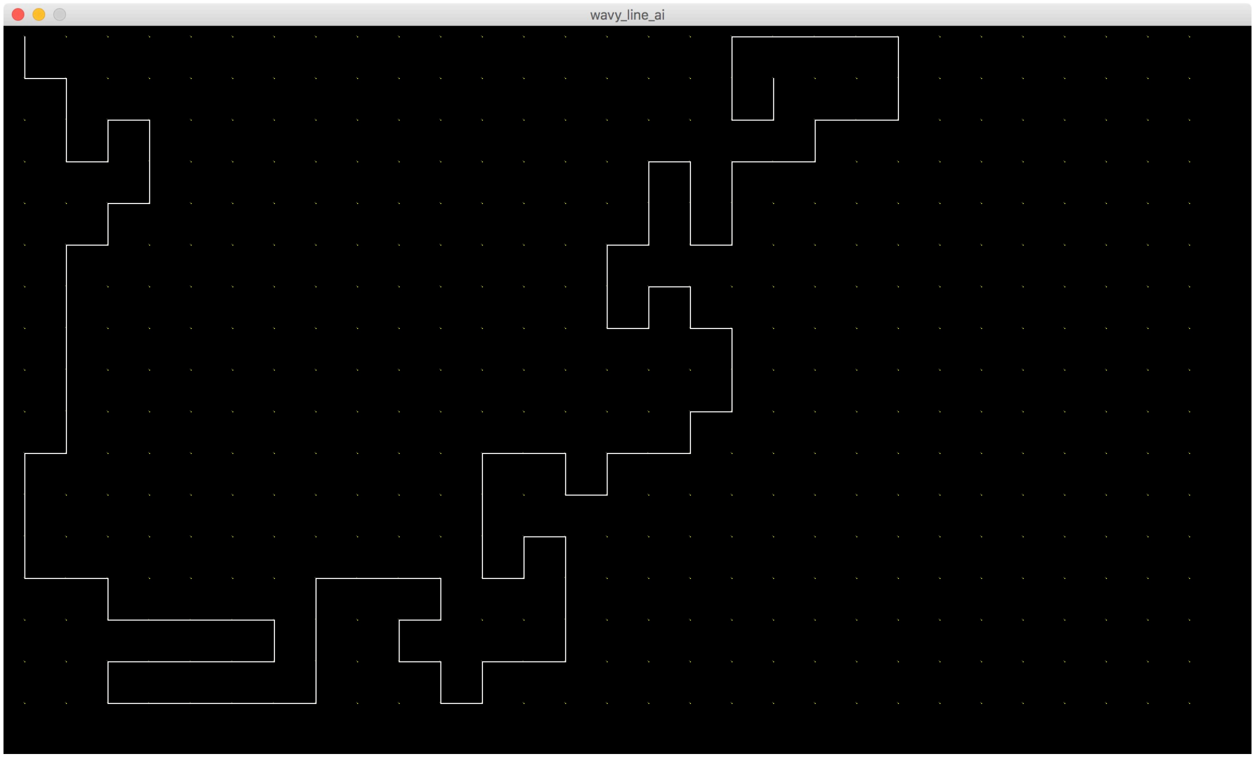


Figure 1: Random costs; search depth of 5.

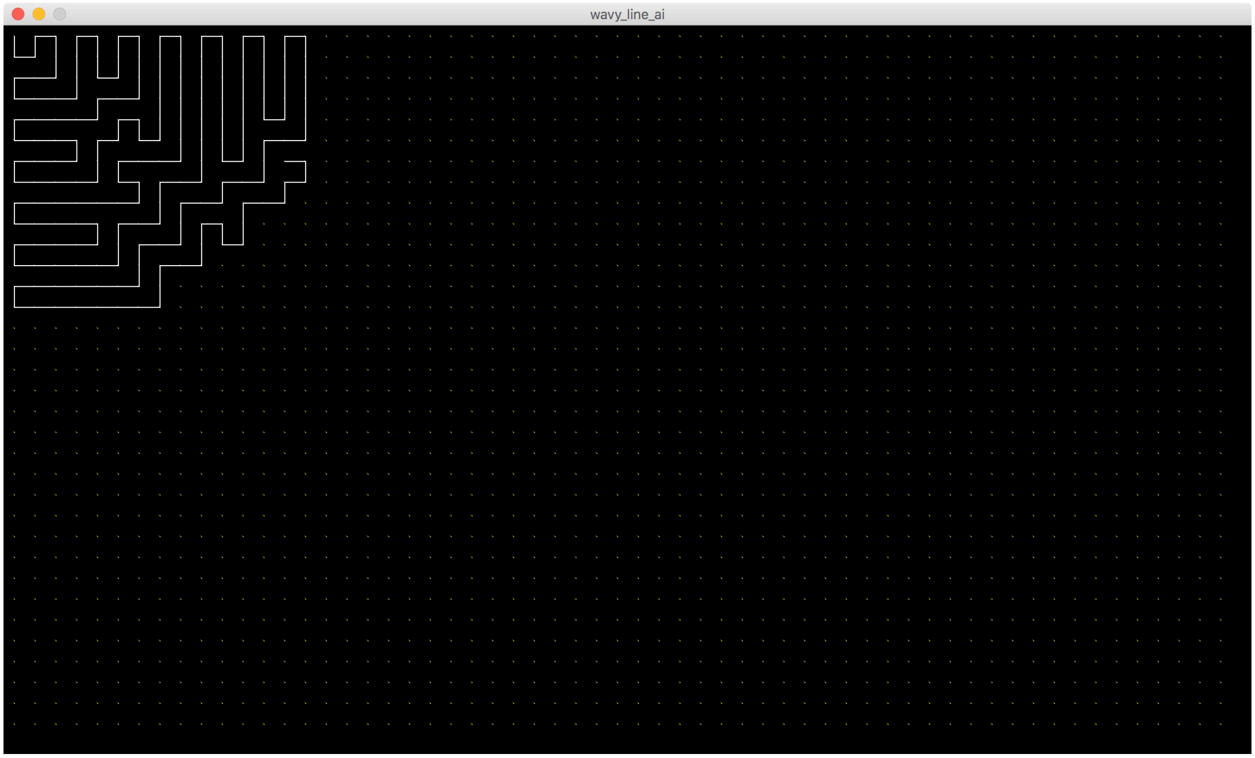


Figure 2: Cost equal to the distance from the starting point (top left); search depth of 3.

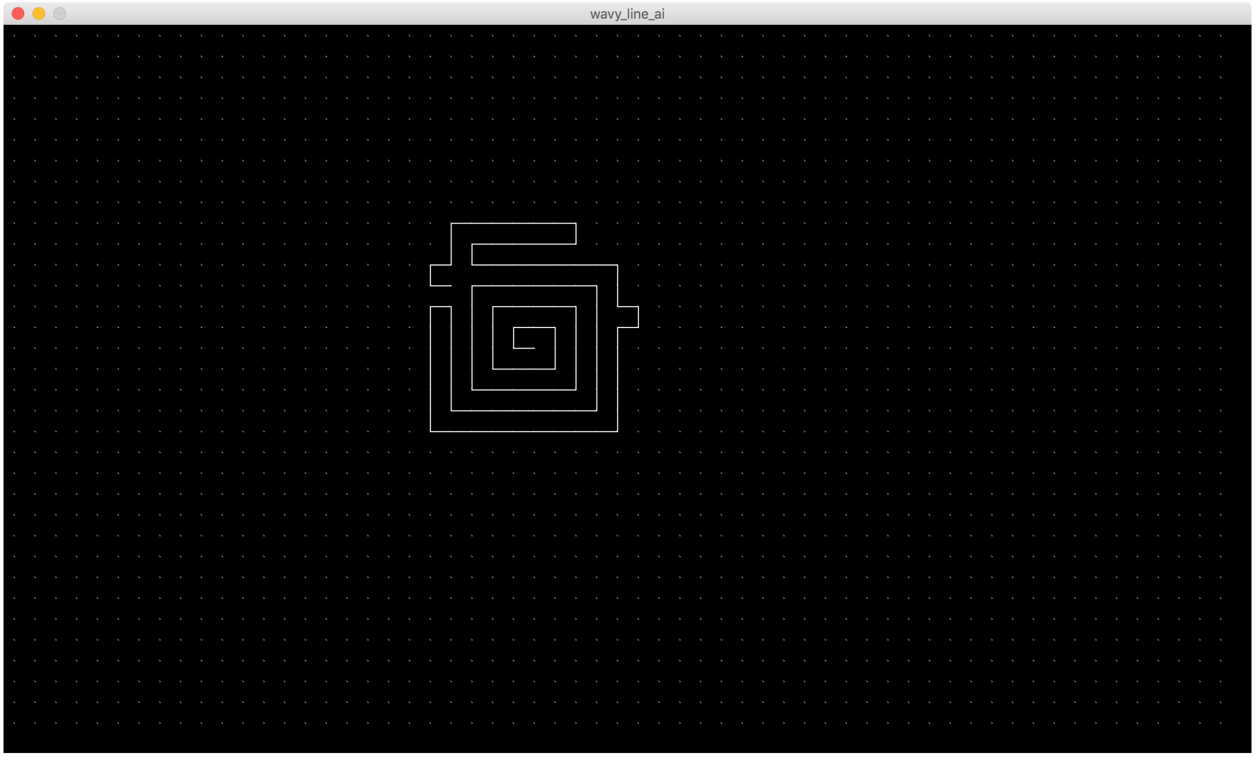


Figure 3: Cost equal to the distance from the starting point *(near center)*; search depth of 3.

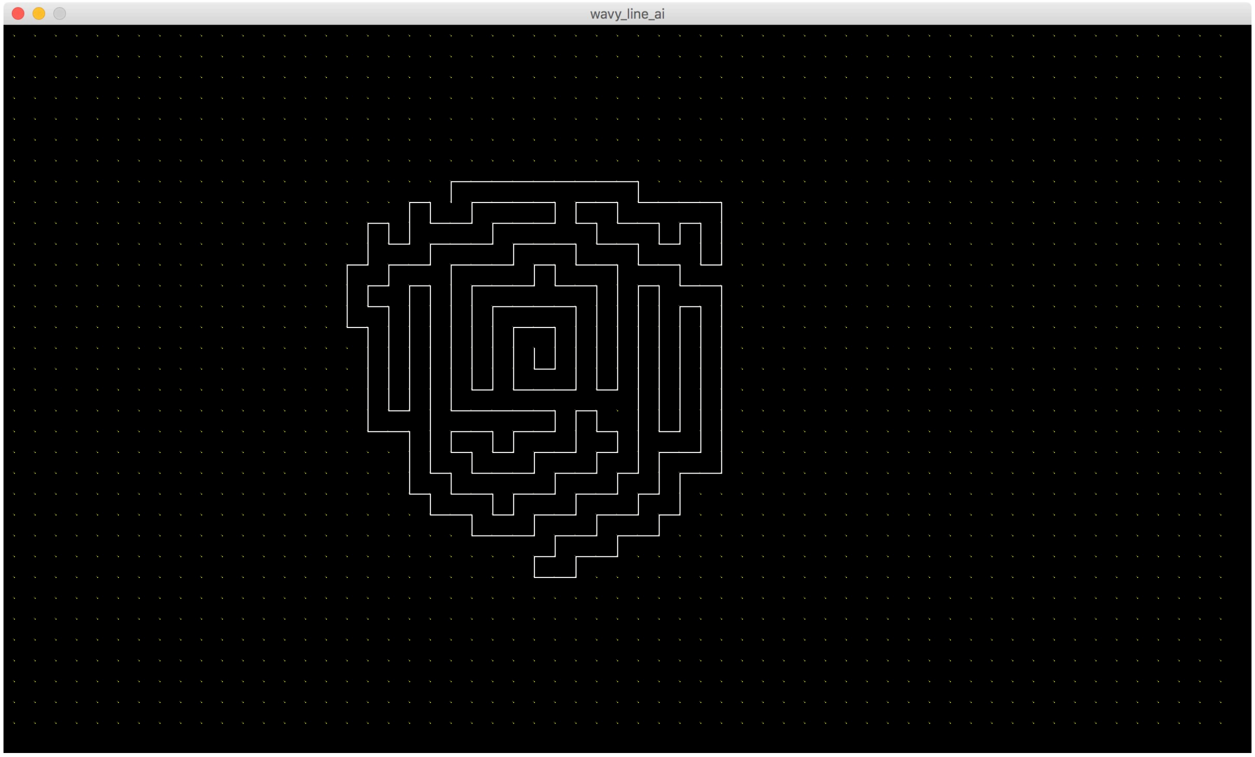


Figure 4: Cost equal to the distance from the starting point (near center), *plus a random number in the range of the average of the width and height of the grid divided by 50;* search depth of 3.

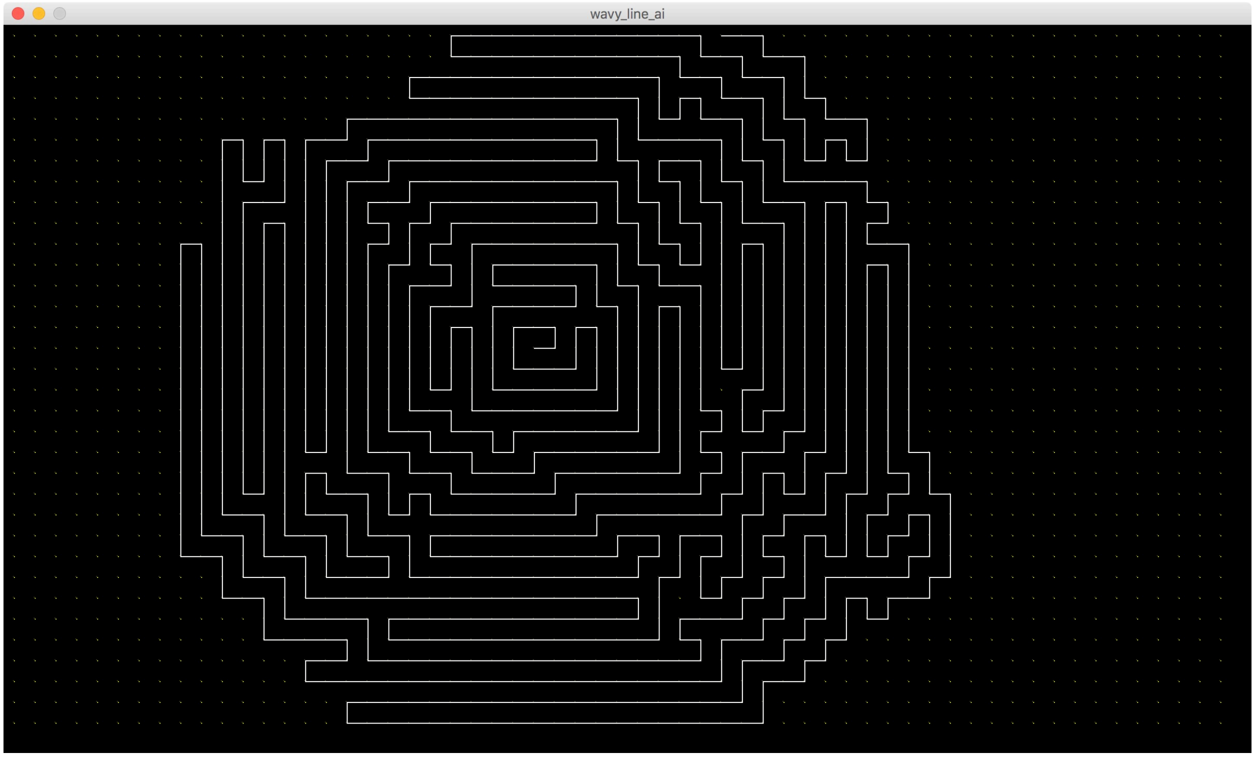


Figure 5: Cost equal to the distance from the starting point (near center) *multiplied by 1.5,* plus a random number in the range of the average of the width and height of the grid *divided by 30;* search depth of 2.

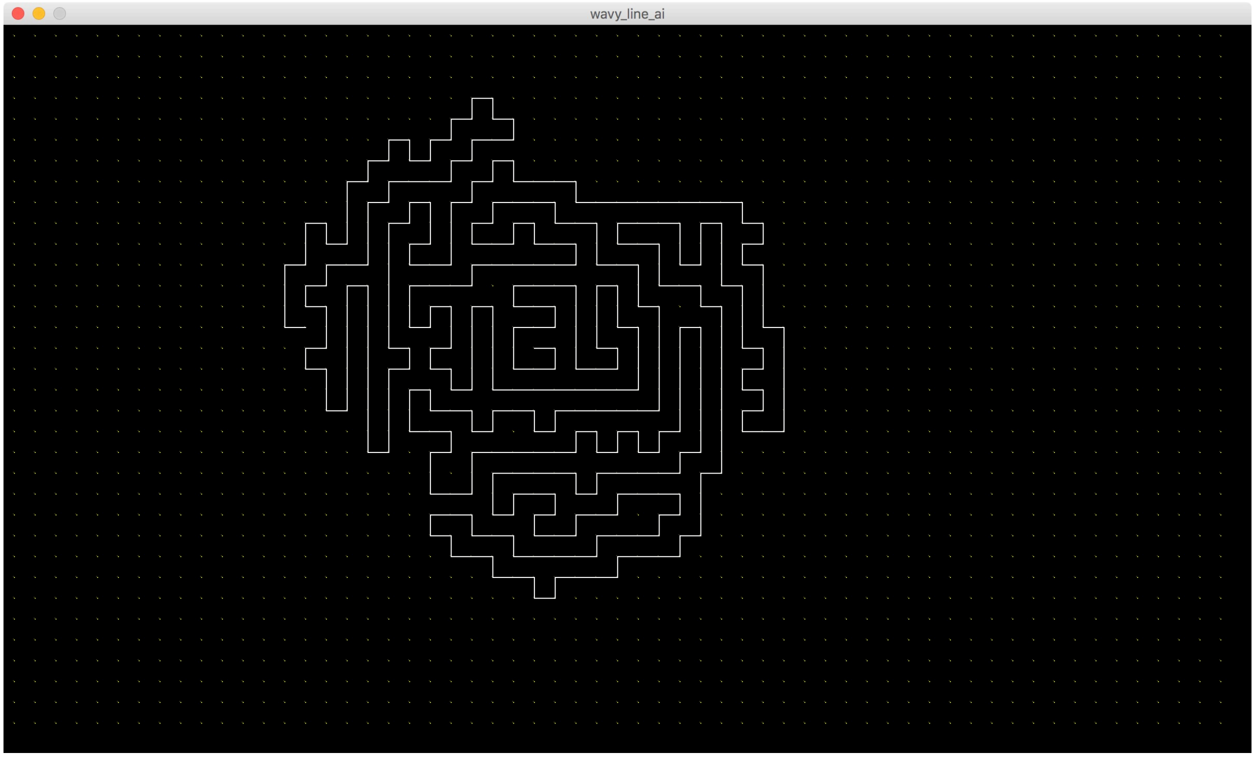


Figure 6: Cost equal to the distance from the starting point (near center) *multiplied by 0.75,* plus a random number in the range of the average of the width and height of the grid divided by 30; search depth of 2.

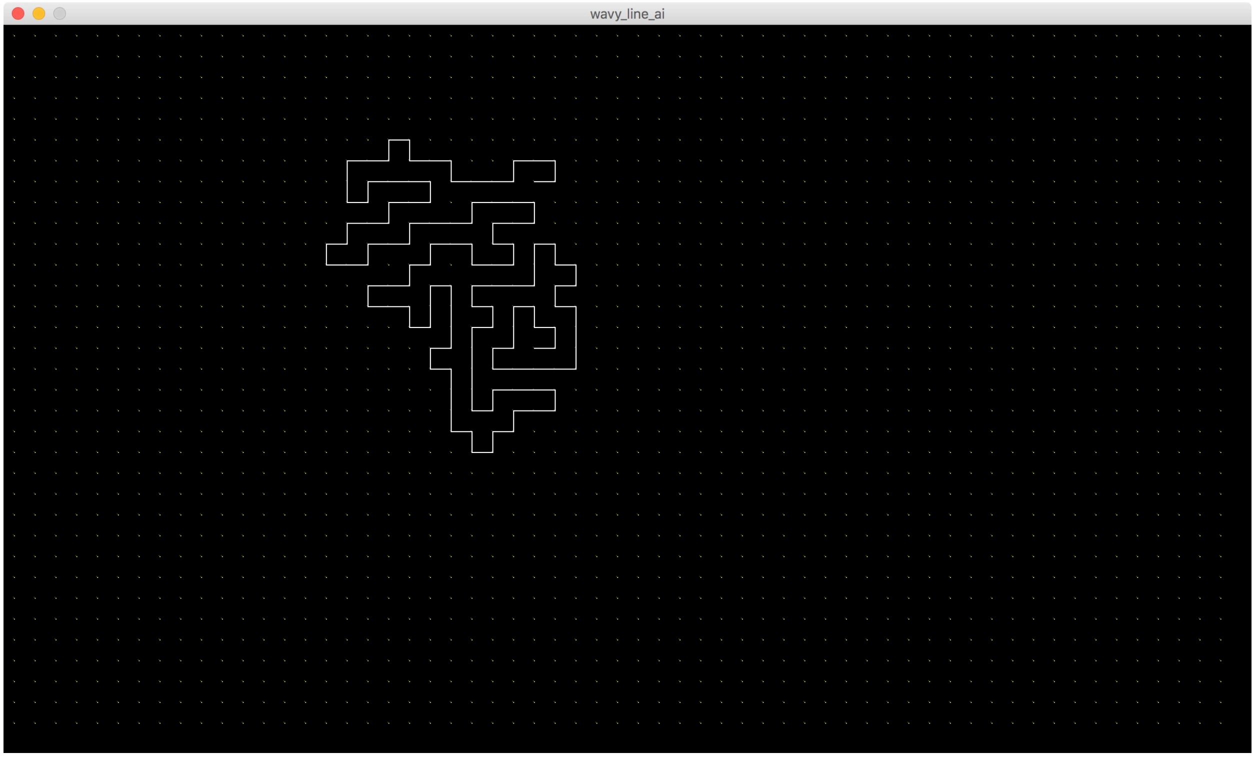
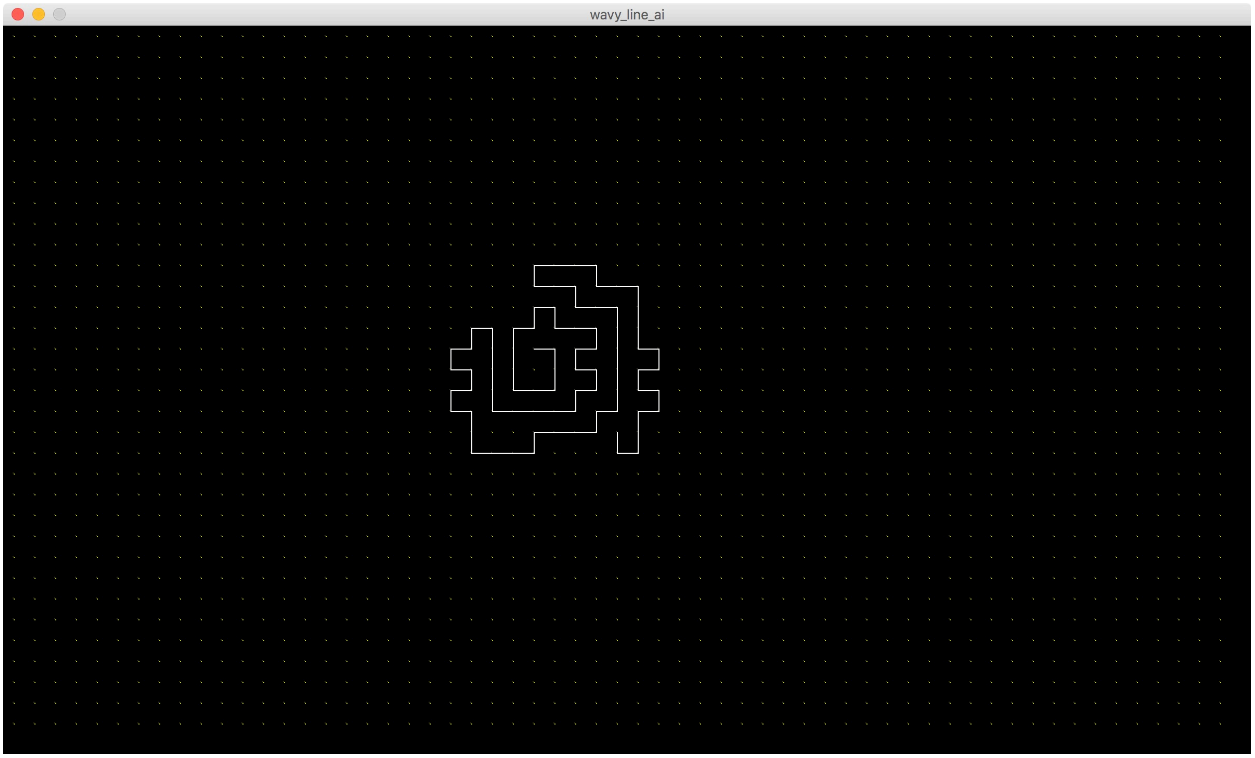


Figure 7: Cost equal to the distance from the starting point (near center) *multiplied by 0.5,* plus a random number in the range of the average of the width and height of the grid divided by 30; search depth of 2.



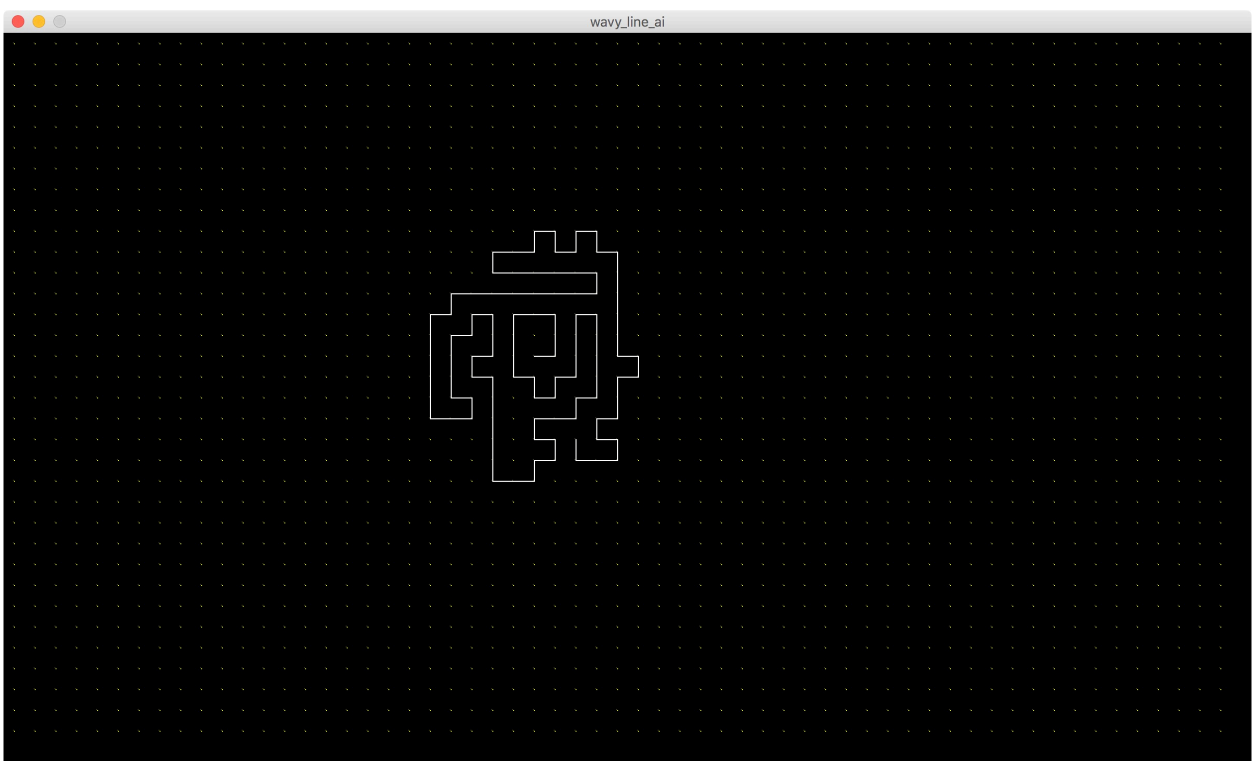


Figure 8: Same conditions as Figure 7.

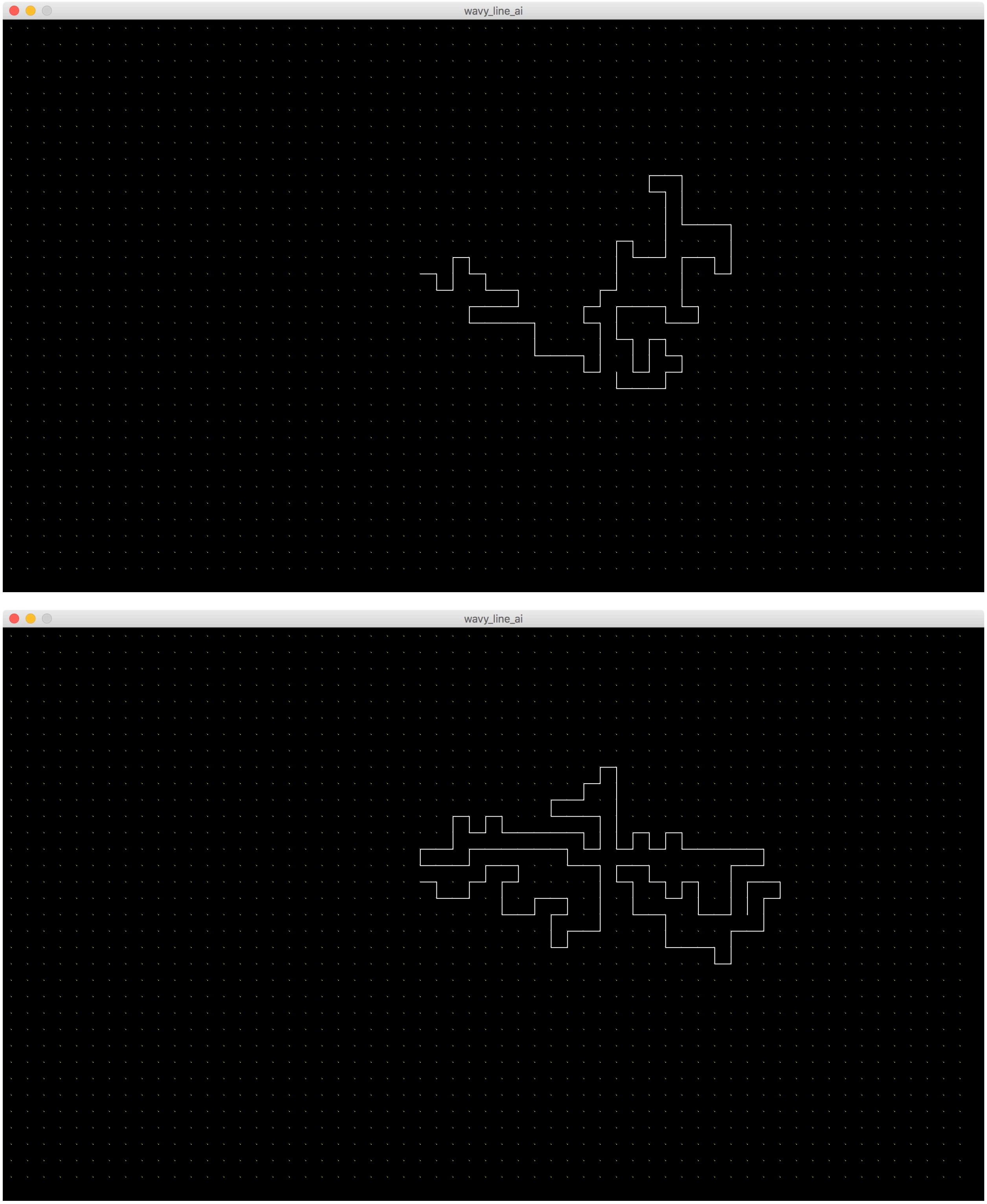


Figure 9: Cost equal to the distance from the starting point (near center) multiplied by 0.5, *plus the distance to the nearest canvas edge multiplied by 0.5,* plus a random number in the range of the average of the width and height of the grid divided by 30; search depth of 2.

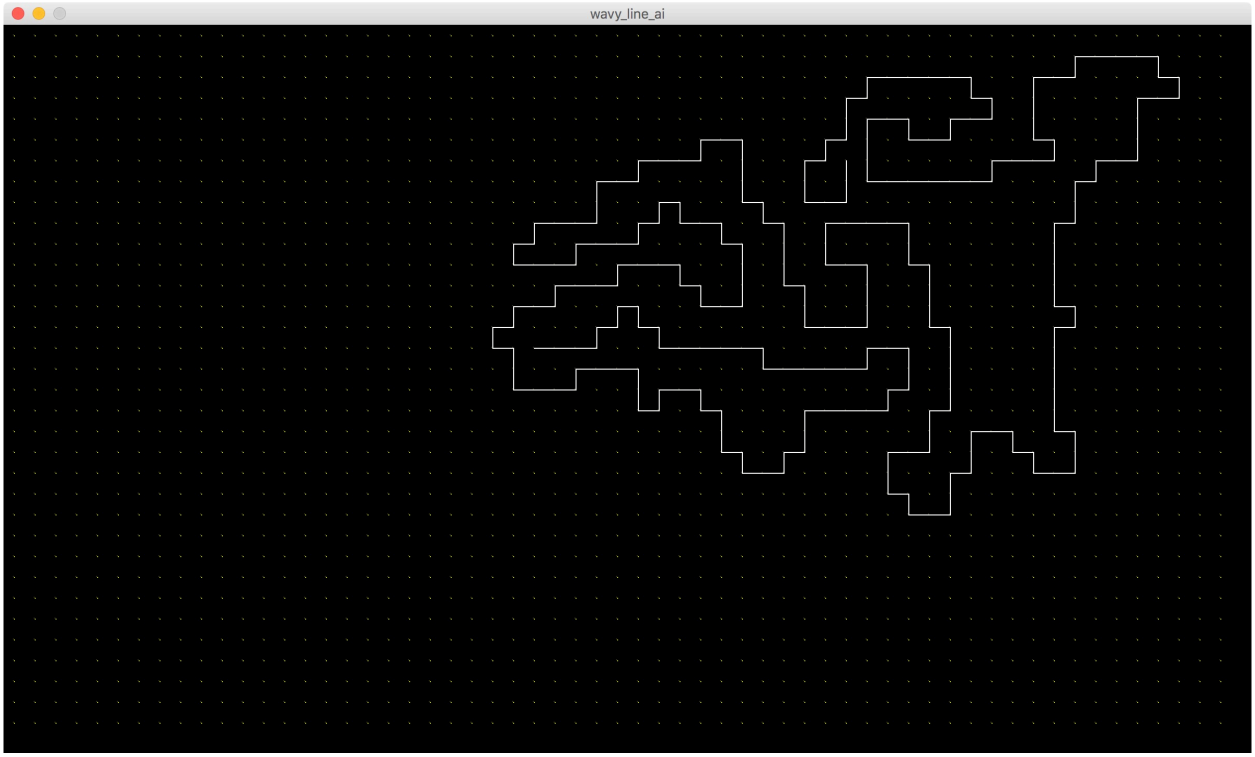


Figure 10: Cost equal to the distance from the starting point (near center) multiplied by 0.5, plus the distance to the nearest canvas edge *multiplied by 0.4, plus a cost for each visited point surrounding the successor state,* plus a random number in the range of the average of the width and height of the grid divided by 30; search depth of 2.

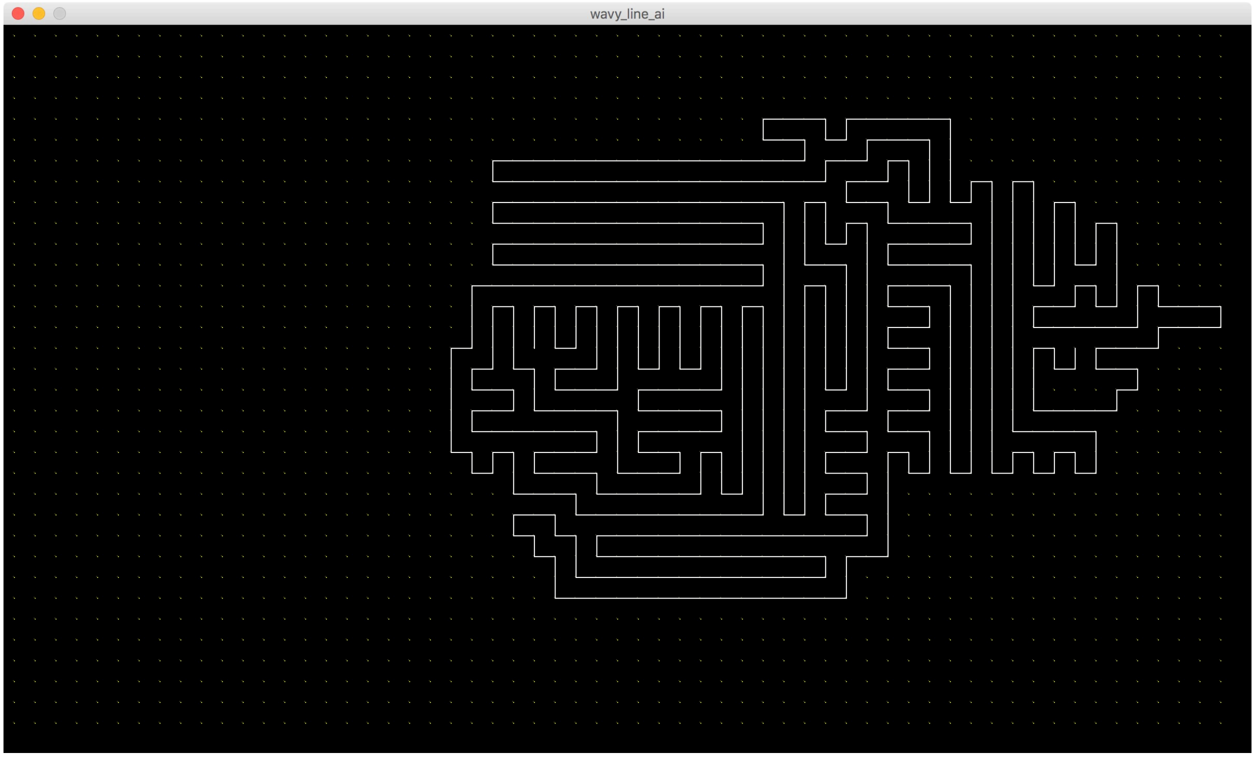


Figure 11: Cost equal to the distance from the starting point (near center) multiplied by 0.5, plus the distance to the nearest canvas edge multiplied by 0.4, *plus the average of the width and height of the grid multiplied by 2 if the successor state has three or more visited points surrounding it,* plus a random number in the range of the average of the width and height of the grid divided by 30; search depth of 2.

Figure 12: Cost equal to the distance from the starting point (near center) multiplied by 0.5, plus the distance to the nearest canvas edge multiplied by 0.4, plus the average of the width and height of the grid *multiplied by 1.75* if the successor state has three or more visited points surrounding it, plus a random number in the range of the average of the width and height of the grid divided by 30; search depth of 2.

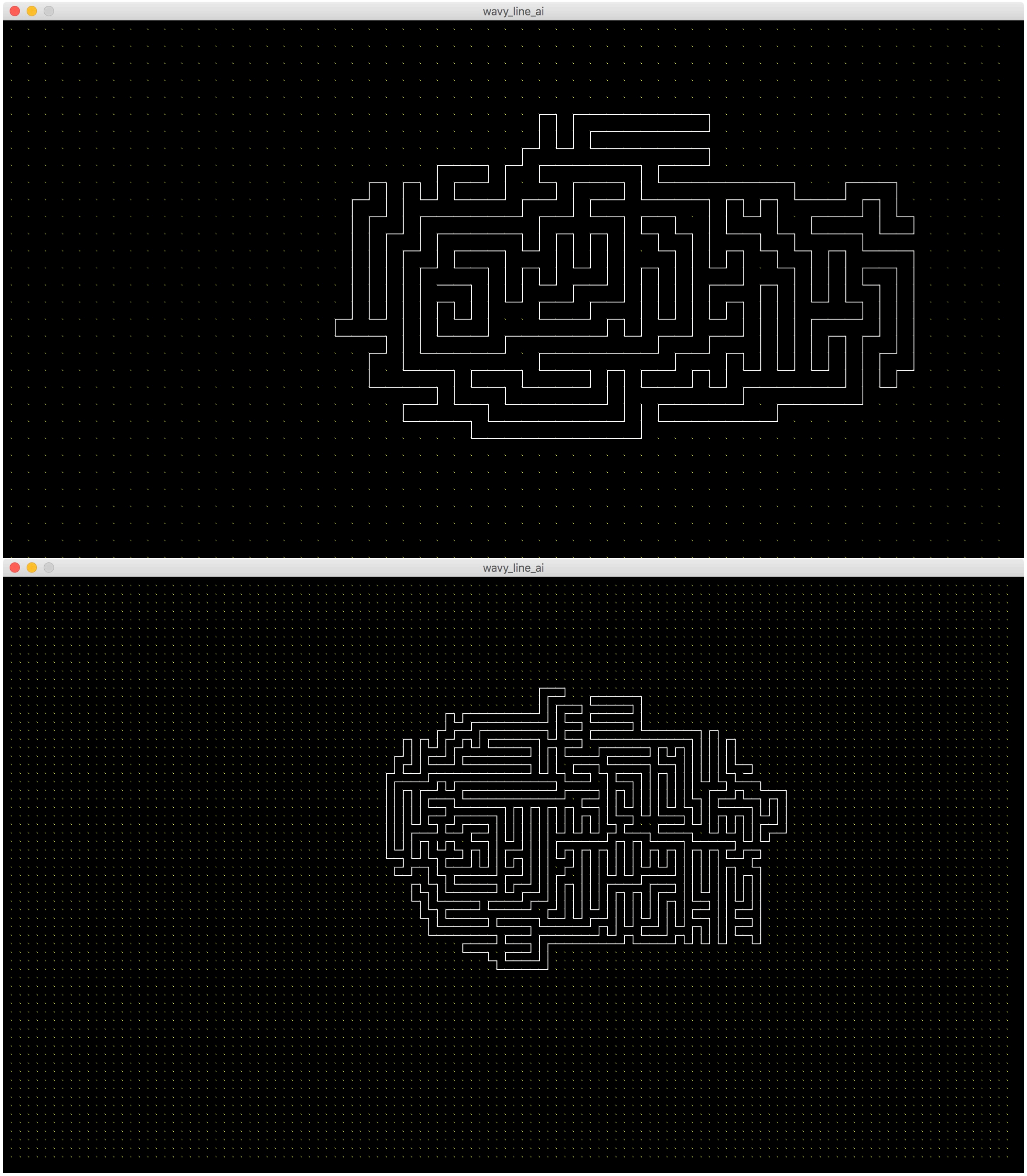


Figure 13: Cost equal to the distance from the starting point (near center) multiplied by 0.5, plus the distance to the nearest canvas edge multiplied by 0.4, plus the average of the width and height of the grid multiplied by 1.75 if the successor state has three or more visited points surrounding it, plus a random number in the range of the average of the width and height of the grid divided by 30; *search depth of 3*.

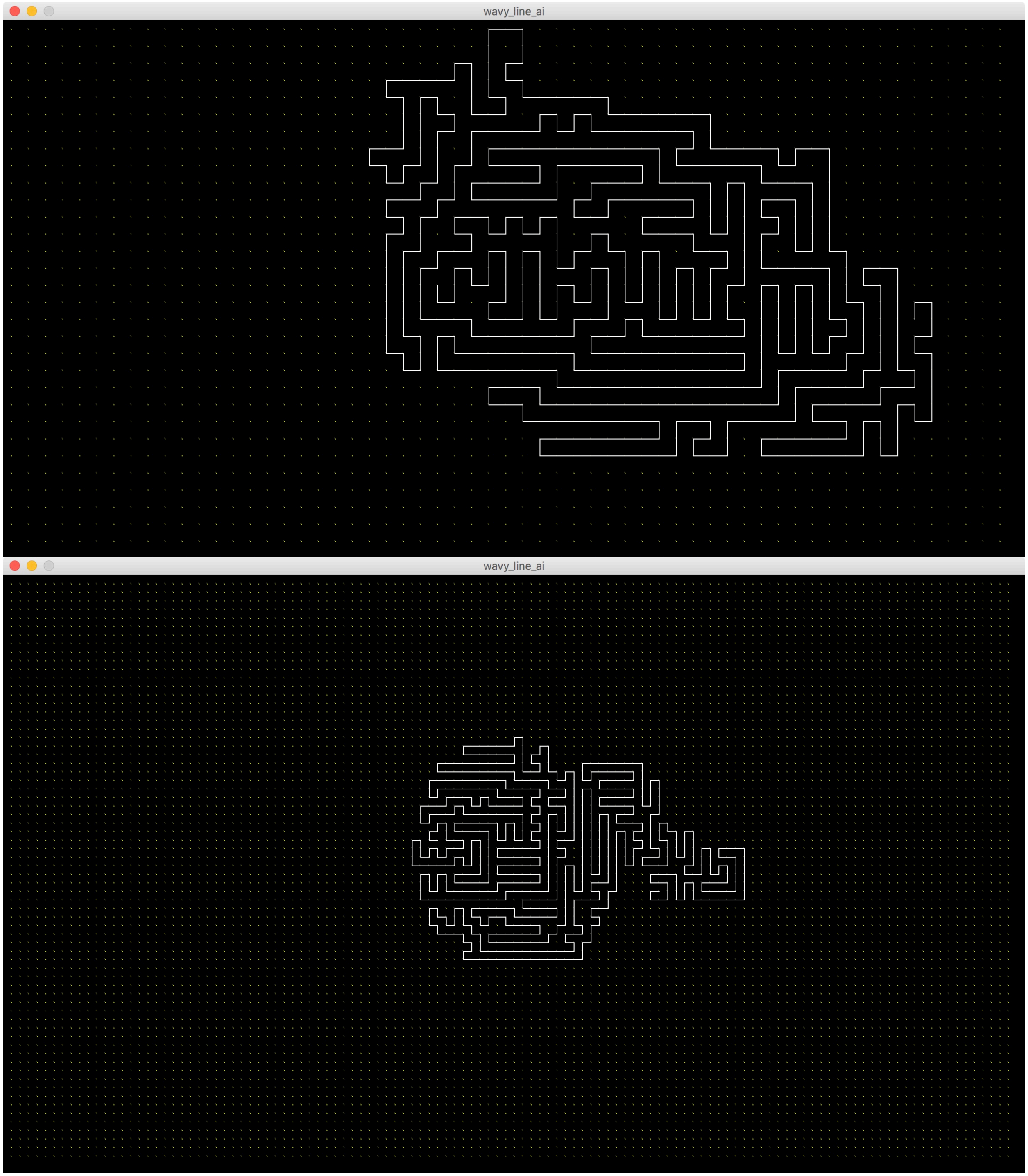


Figure 14: Cost equal to the distance from the starting point (near center) multiplied by 0.5, plus the distance to the nearest canvas edge multiplied by 0.4, plus the average of the width and height of the grid multiplied by 1.75 if the successor state has three or more visited points surrounding it, plus a random number in the range of the average of the width and height of the grid divided by 30; *search depth of 4*.

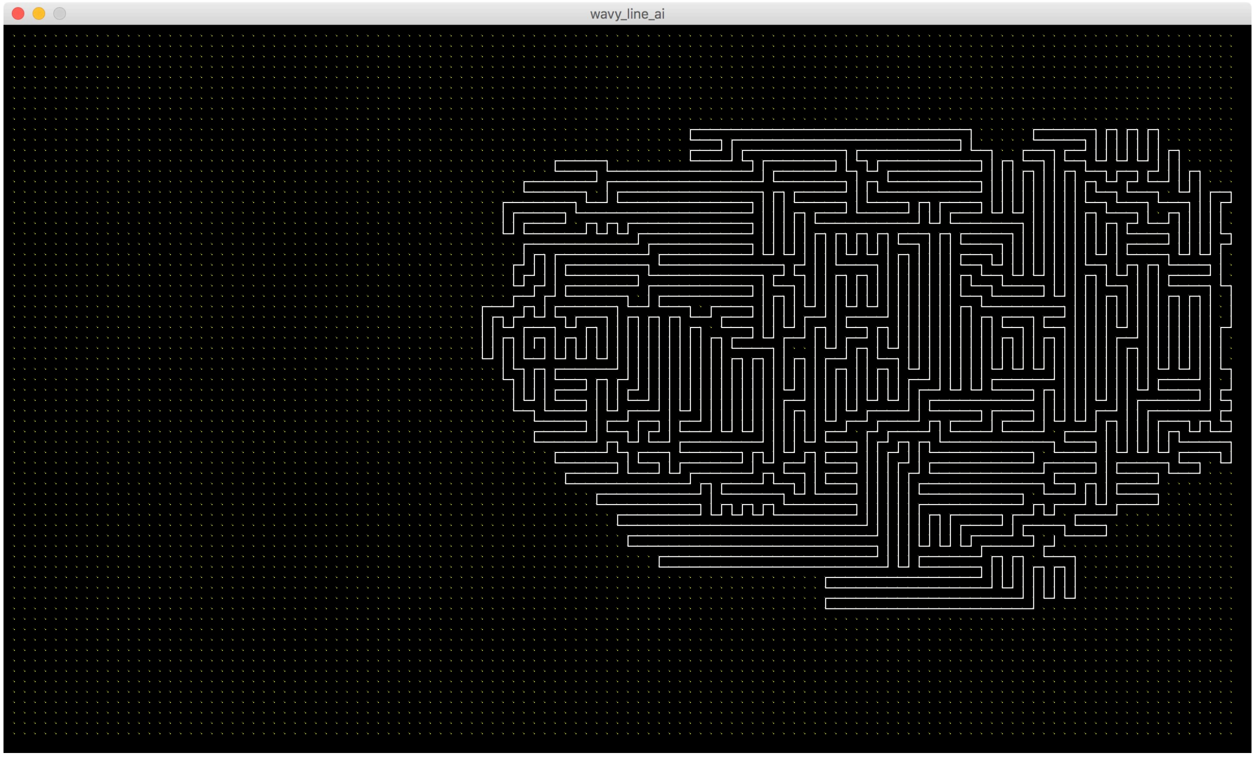


Figure 15: Cost equal to the distance from the starting point (near center) multiplied by 0.5, plus the distance to the nearest canvas edge multiplied by 0.4, plus the average of the width and height of the grid *multiplied by 2.25* if the successor state has three or more visited points surrounding it, plus a random number in the range of the average of the width and height of the grid divided by 30; search depth of 4.

Figure 16: Cost equal to the distance from the starting point (near center) multiplied by 0.5, plus the distance to the nearest canvas edge multiplied by 0.4, plus the average of the width and height of the grid *multiplied by 4* if the successor state has three or more visited points surrounding it, plus a random number in the range of the average of the width and height of the grid divided by 30; search depth of 4.

Figure 17: Same conditions as figure 16, but on a larger grid.

Figure 18: Cost equal to the distance from the starting point (near center) multiplied by 0.5, plus the distance to the nearest canvas edge multiplied by 0.4, plus the average of the width and height of the grid *multiplied by 6* if the successor state has three or more visited points surrounding it, plus a random number in the range of the average of the width and height of the grid divided by 30; search depth of 4.