AI for Game Programming 2: Maze Generation (A3.6)

Jonas Nockert

January 8, 2020

Basic steps: Maze generation on a Voronoi diagram

See figures 1, 2, 3, 4, 5 and 6.

Growing Tree using the gamma distribution for branching

The Recursive Backtracker algorithm consists of the following steps

1. Add a starting cell to an empty stack, marking the cell as visited,
2. pop a cell of the top of the stack,
3. find a non-visited neighbor to that cell (or step 7 in case none exists,)
4. remove the edge between this neighbor and cell,
5. re-add the cell to the top of the stack in case there are more neighbors,
6. add the neighbor to the top of the stack, marking it as visited,
7. return to step 2 until the queue is empty.

The growing tree algorithm is a variation of the above regarding how the next cell is picked in step 2. If the growing tree algorithm is set to always choose the most recently added cell, it becomes equivalent to Recursive Backtracking or depth-first search (see figure 7). If cells are chosen at random, it becomes equivalent to Prim's algorithm (a minimum spanning tree of a graph can be viewed as a maze). Another option is to pick the

oldest cells in the queue, essentially breadth-first search, which tends to create long straight corridors and not good mazes (see figure 8).

It seems that often a ratio is picked between two or more of the above cases, e.g. randomly picking most recent or by random in a 50/50 split.

Here, for variation, choices are sampled according to the gamma distribution which can be seen as the picks are done with some regularity in terms of rate of occurence. I've chosen to pick the oldest cells with a mean of every sixth time with most recent cells picked in between.

The idea (hope) is that this will lead to longer winding dead-end branches off the "main" path but with a guarantee that there's always some winding segments between branches as the gamma distribution is positive valued. The branches should not be allowed to compete too much for space in order for both branches and main path to achieve a degree of windingness.

The benefit is longer dead-end corridors but the downside is that the longest path generated is shorter than what Recursive Backtracker algorithm would produce. This is inherent in the trade-off between DFS and BFS.

Examples of mazes generated by this algorithm is shown in figures 9 and 10.

References

Yonghe, L. et al. (2013). "A Simple Sweep-line Delaunay Triangulation Algorithm". In: *Journal of Algorithms and Optimization (JAO)* 1.1, pp. 30–38.

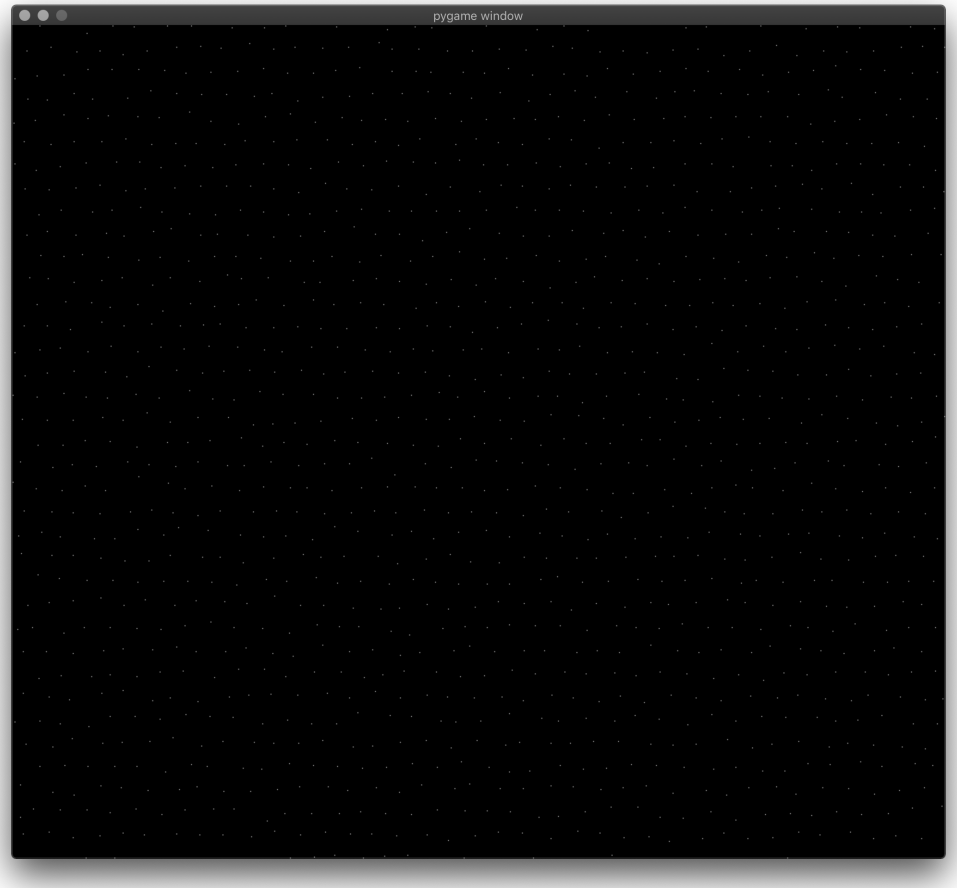

Figure 1: *From a number of points arranged (semi-randomly) on a grid. . .*

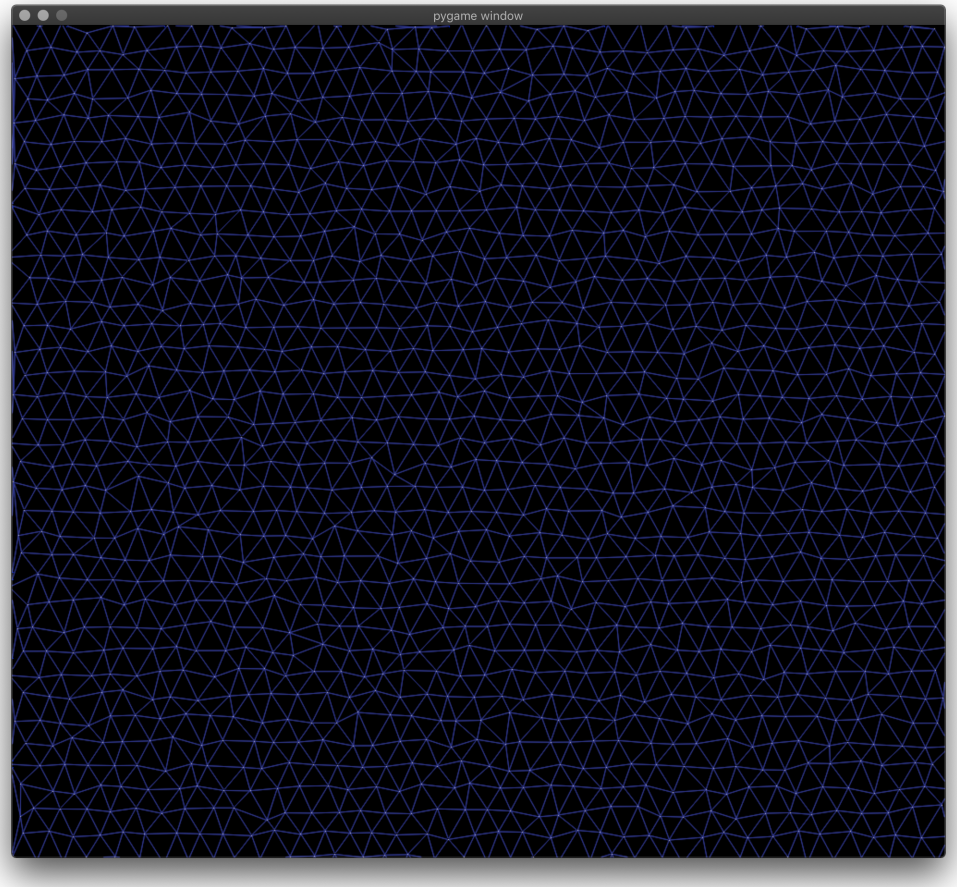

Figure 2: *. . . a Delaunay triangulation is created using the procedure described in (Yonghe et al., 2013). Turns out that simple is relative. In order to become Delaunay, the triangulation needs to be Lawson legalized, which I struggled with for a long time.*

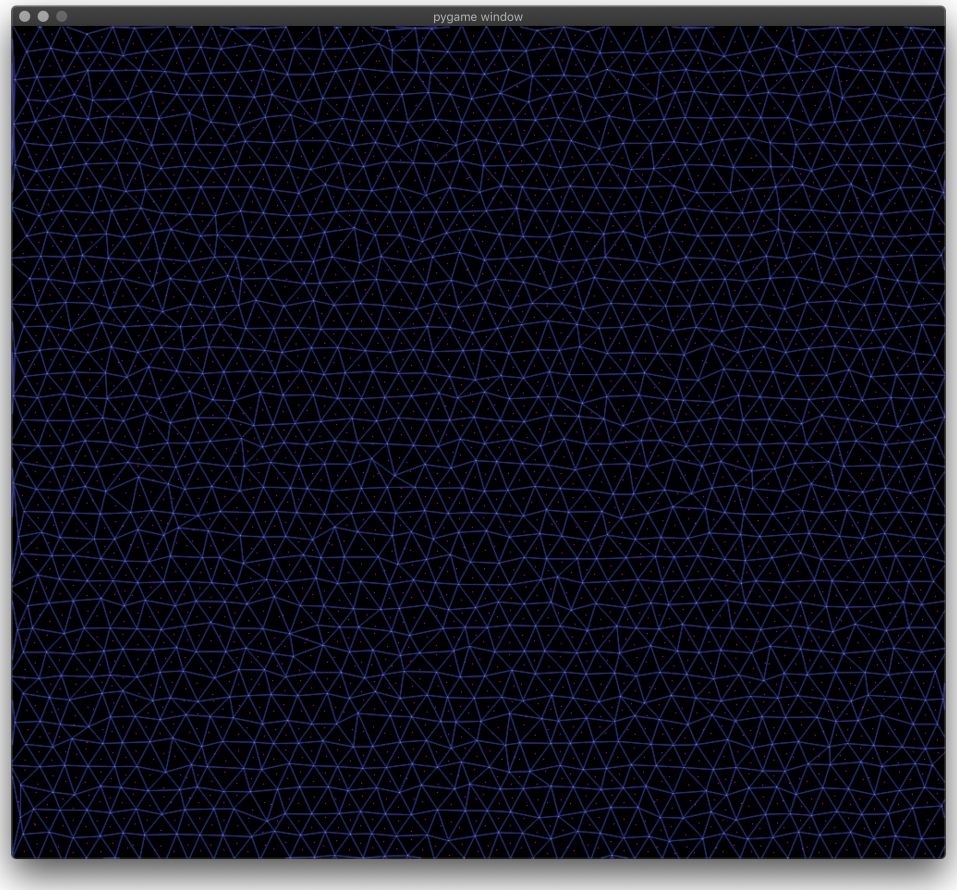

Figure 3: *The Delaunay triangulation is dual with its Voronoi diagram so the latter can be created from the former. Each triangle has a circumcenter (pink).*

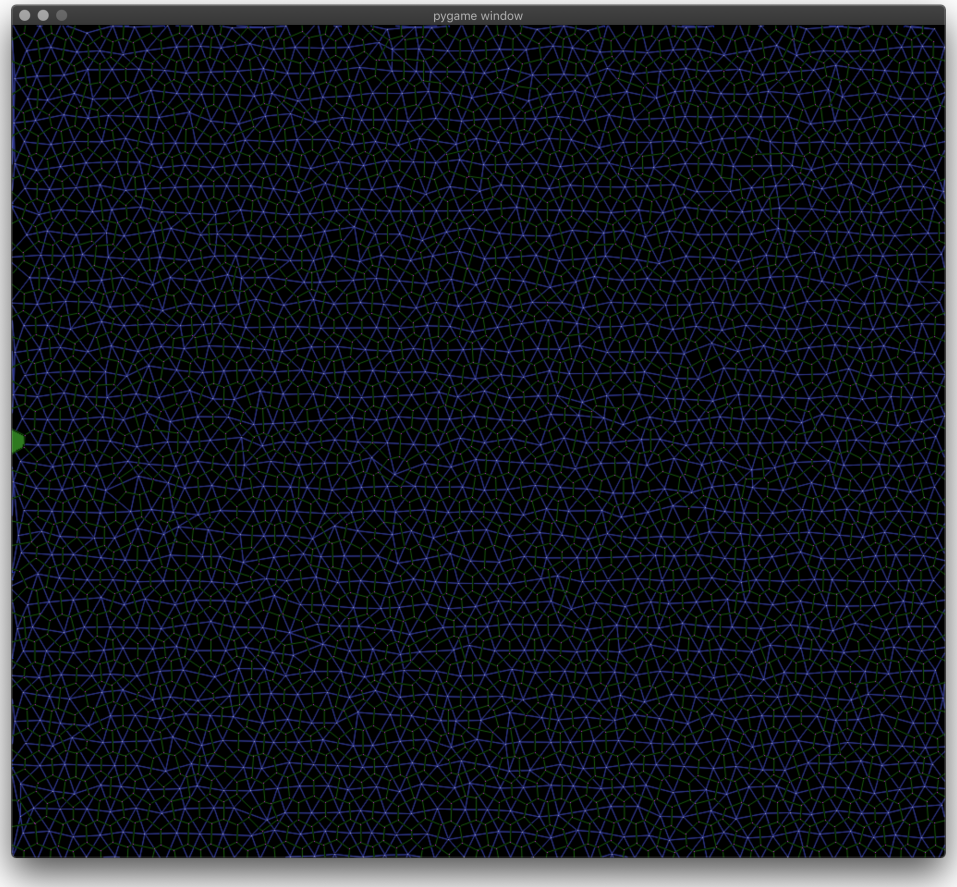

Figure 4: *Edges between the circumcenters of adjacent triangles can be connected into cells. Starting with an edge and following edges counter-clockwise until the edge one started with is reached again we have acquired the polygon of a single cell. Each cell can have up to six neighbor cells.*

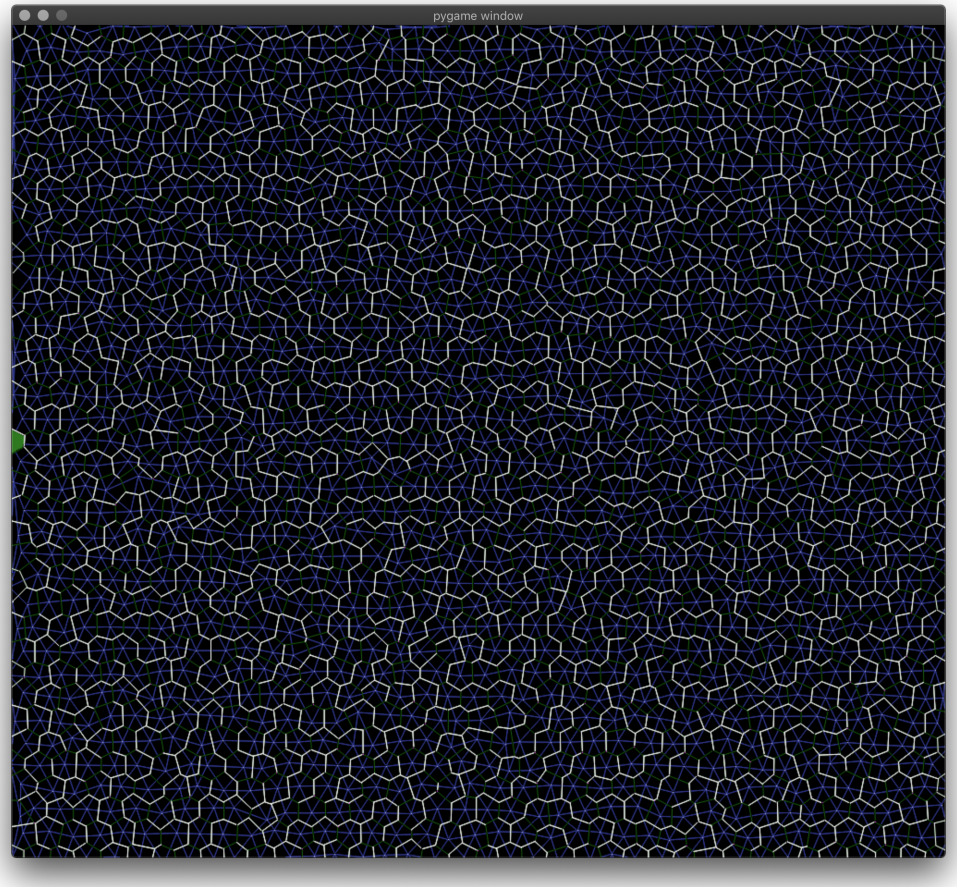

Figure 5: *Although the Recursive Backtracker and Growing Tree algorithms are typically shown with square grids, they work on >4-sided polygons as well. Here we see the resulting maze after cell edges has been removed using a variation of the Growing Tree algorithm.*

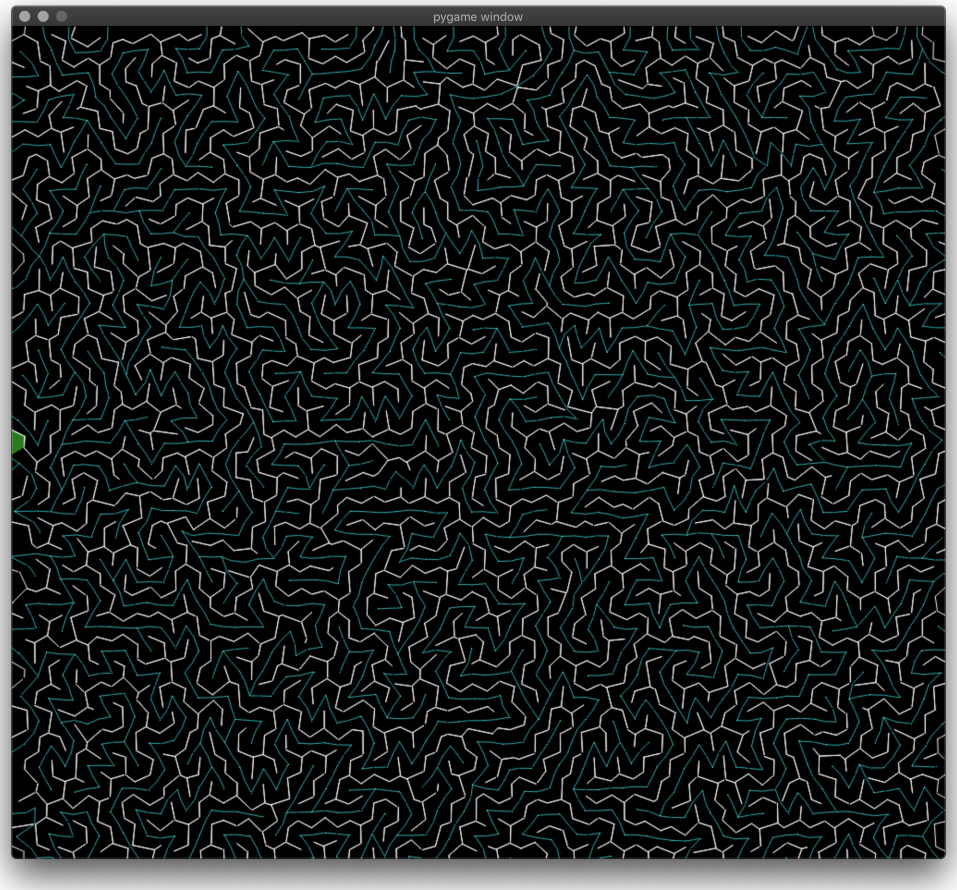

Figure 6: *Since openings between cells can be narrow due to randomness in initial point placement, it can be difficult to see how they are connected. Here the maze paths are shown in cyan.*

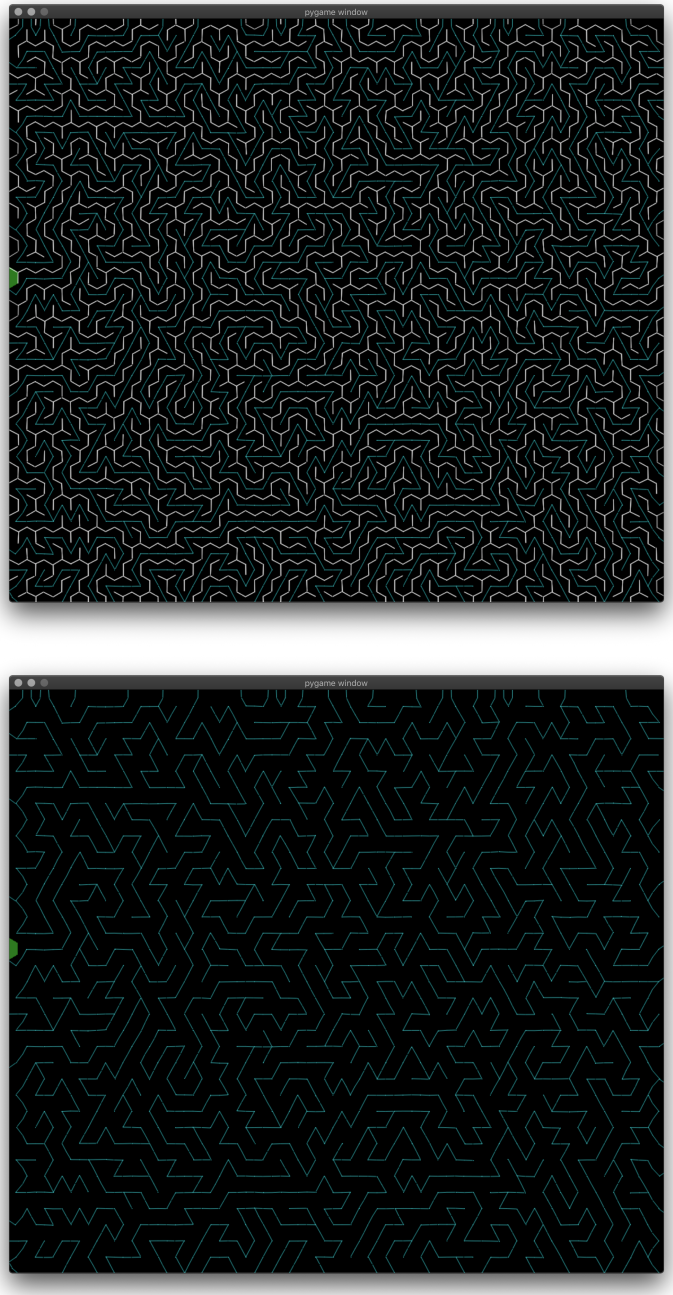

Figure 7: *Using the Recursive Backtracker algorithm (essentially depth-first search) to generate mazes tends to lead to one long, winding corridor with short dead end corridors branching off. This is equivalent to using the Growing Tree algorithm while always picking the cell on top of the stack (i.e. most recently added cells).*

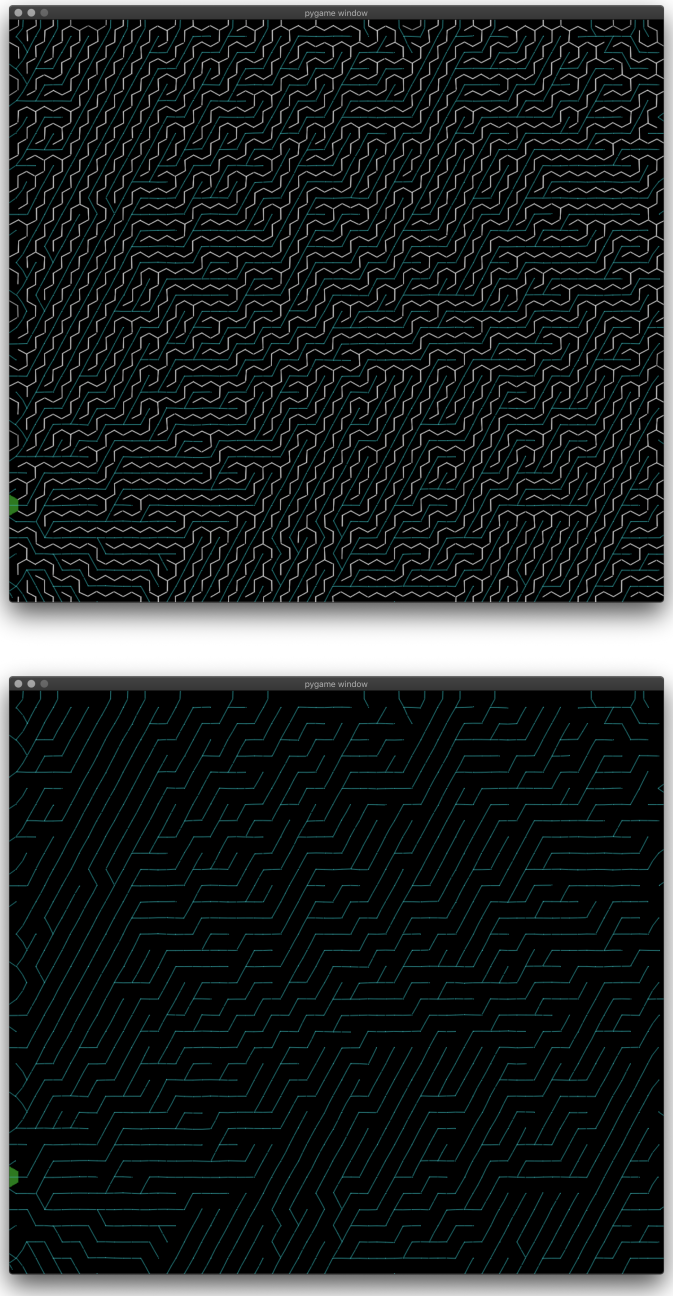

Figure 8: *Using breadth-first search to generate mazes, not surprisingly, tends to lead to the opposite. Here we get many, mostly straight, corridors competing for the space.*

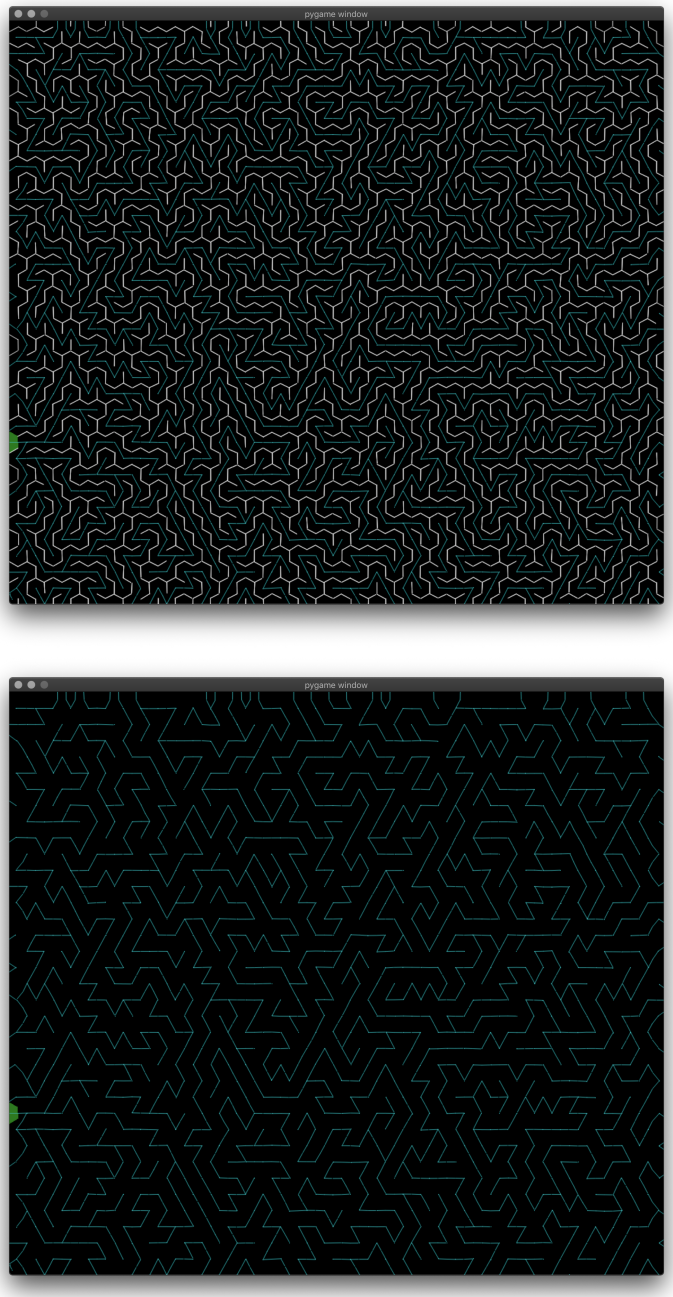

Figure 9: *Combining DFS and BFS seems to be a good idea. However, instead of just mixing the two randomly, I wanted to do it with some regularity. My idea was to use the gamma distribution to model the rate of branching, which guarantees spacing in between branches and should allow for winding and length in both main path as well as branches. The gamma distribution also means branches will occur regularly (but not too regularly).*

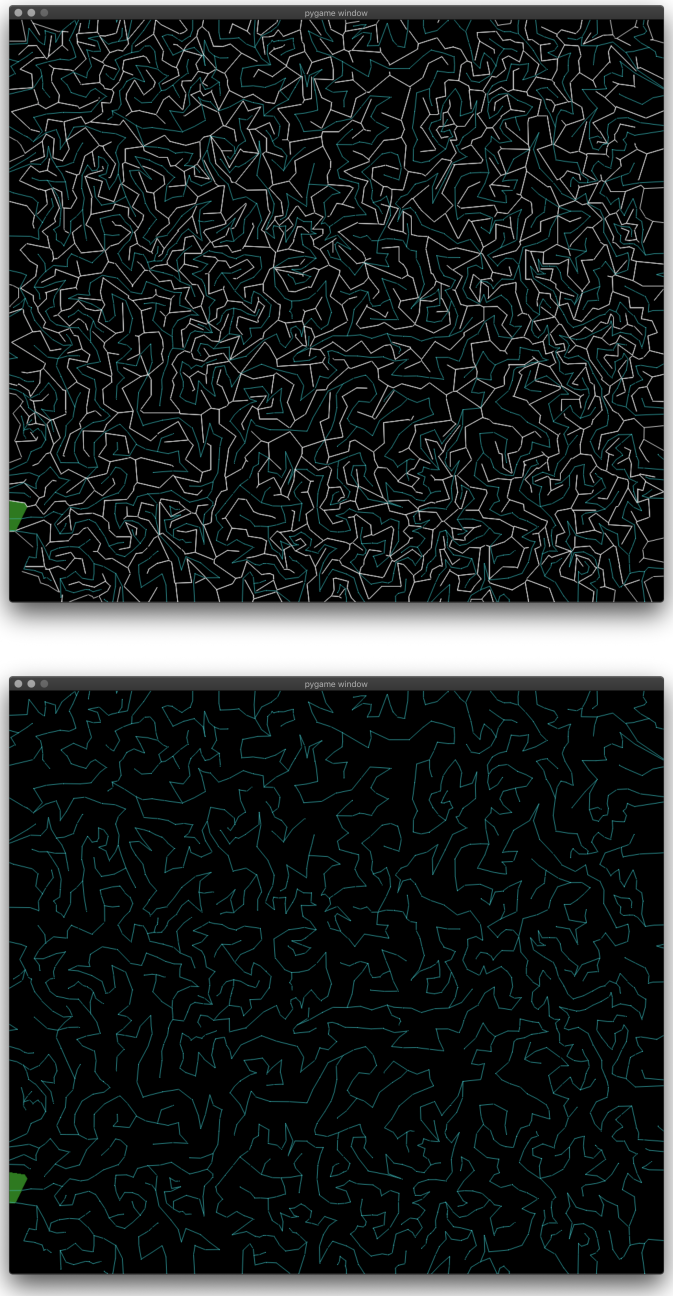

Figure 10: *An example of a maze generated by the algorithm with initial points placed by sampling coordinates uniformly at random rather than on a semi-regular grid.*