**From:**

[**https://liamappelbe.medium.com/how-to-generate-slither-link-puzzles-6c65510b2ba1**](https://liamappelbe.medium.com/how-to-generate-slither-link-puzzles-6c65510b2ba1)

**A better approach**

I was stuck on this deformation approach for a while, but I recently figured out a way of verifying that a particular deformation of the line will increase wiggliness, and won’t cause line crossings, in O(1) time.

The basic idea is simple. Instead of trying to generate a line from one side to the other, we think of it as a cell coloring problem. The goal is to divide the grid into two orthogonally contiguous regions of color, and the final line is the boundary of these regions. Our initial straight line state is translated into having the top half of the grid in one color, and the bottom half in the other color:

A blue and black flag

AI-generated content may be incorrect.

Once all the deformations are done, we should end up with something like this:

A blue and black square with black rectangles

AI-generated content may be incorrect.

The final line is the boundary of these regions:

A blue and yellow rectangular shape with black lines

AI-generated content may be incorrect.

One important thing to note about this reformulation is that the constraint that the final line is a single continuous line with no branches or crossings is preserved. This constraint is exactly the same as the constraint that the two regions of color are orthogonally contiguous. As long as we don’t have patches of color floating by themselves, the boundary between the colors will form a single unbroken line from one side to the other.

Each deformation of the line is just a matter of flipping the color of one of the cells adjacent to the boundary. So the question is how do we make sure that flipping the color of a particular cell will increase the wiggliness of the boundary, without separating either of the colored regions?

The wiggliness check involves counting how many of the orthogonally adjacent cells have the same color as the cell we are considering flipping.

A blue and black square with red squares

AI-generated content may be incorrect.

Let’s look at these cases starting from the left. If all of the surrounding cells are the same color as the cell, flipping its color would leave a disconnected patch of color. If 3 are the same color and one is different, flipping it increases wiggliness, so this would be a valid flip. If 2 adjacent cells are the same color, flipping it doesn’t change anything. If only 1 adjacent cell is the same color, flipping it would reduce wiggliness. And on the right, we should never see a case where all 4 adjacent cells are a different color to our candidate cell, because that would mean we already have a disconnected patch of color.

Actually the 2 adjacent cells case is a little more complicated. It may or may not change the wiggliness, depending on the color of the 4 diagonally adjacent cells.

A blue and black square with red squares

AI-generated content may be incorrect.

The first case shouldn’t happen, and the last case would cause disconnected regions. Of the remaining cases, the only one that increases wiggliness is the case where 3 of the diagonal cells are a different color than the candidate cell. There are also cases where it’s not enough to just count the number of same and differently colored cells, but they’re all covered by the region connectivity check below.

These rules don’t quite work at the edges of the grid, so we have to modify the check a bit: the wiggliness increases if the number of same colored orthogonally adjacent cells is greater than the number of differently colored ones, or if they’re the same but the number of same colored diagonally adjacent cells is greater than the number of differently colored ones. That’s a mouthful, so to summarize in pseudocode:

*same(O) > diff(O) or (same(O) == diff(O) and same(D) > diff(D))*

Where O is the set of orthogonally adjacent cells, and D is the diagonally adjacent cells.

The connectivity check turns out to be pretty simple. Let’s look at some examples where flipping the candidate cell color would cause disconnected regions.

A blue and black striped background

AI-generated content may be incorrect.

All these cases have one thing in common. As we walk around the 8 adjacent cells, we count the number of times the color flips. If the total is 4 or more, then flipping the candidate cell color will cause disconnected regions.

Again, the edges of the grid are a little more complicated. Let *num(O)* be the number of orthogonally adjacent cells, then the connectivity check is:

*flips > 2 \* num(O)* − *4*

So putting all this together, we start with a grid with the top half one color and the bottom half the other color. We repeatedly choose a random cell on the border between the colors, by maintaining an array of candidate coordinates and using pop-swap to remove a random element. If that cell meets the aforementioned conditions we flip its color and add its neighbors to the array. Repeat until the array is empty. Overall this algorithm is (roughly) O(n), where n is the number of cells in the grid.

A blue and black maze

AI-generated content may be incorrect.

**Refining the results**

This algorithm generates pretty good results, but there’s still a bit of a remnant of the initial conditions. It can be hard to see from just one example, but after looking at a lot of them, I noticed that there’s a lot of long vertical lines in these patterns. It’s a bit more obvious when you rotate it and compare it to the original.

A blue and black maze

AI-generated content may be incorrect.

My guess as to why this happens is that the initial state being divided in half horizontally means that all the valid deformations early in the process are vertical, and this biases the results.

The most straightforward fix is to randomize the initial conditions somehow. I tried initializing it to a sine wave pattern, and this eliminated a lot of the visible bias. But I wanted to truly randomize it, because the bias was still there, even if I couldn’t easily see it. I considered coming up with a variation on [simplex noise](https://www.shadertoy.com/view/Xs2fRd), but making that obey the connectedness rule would be a pain.