

GDC

Math for Game Developers:
**Tile-Based Map Generation Using
WaveFunctionCollapse in *Caves of Qud***

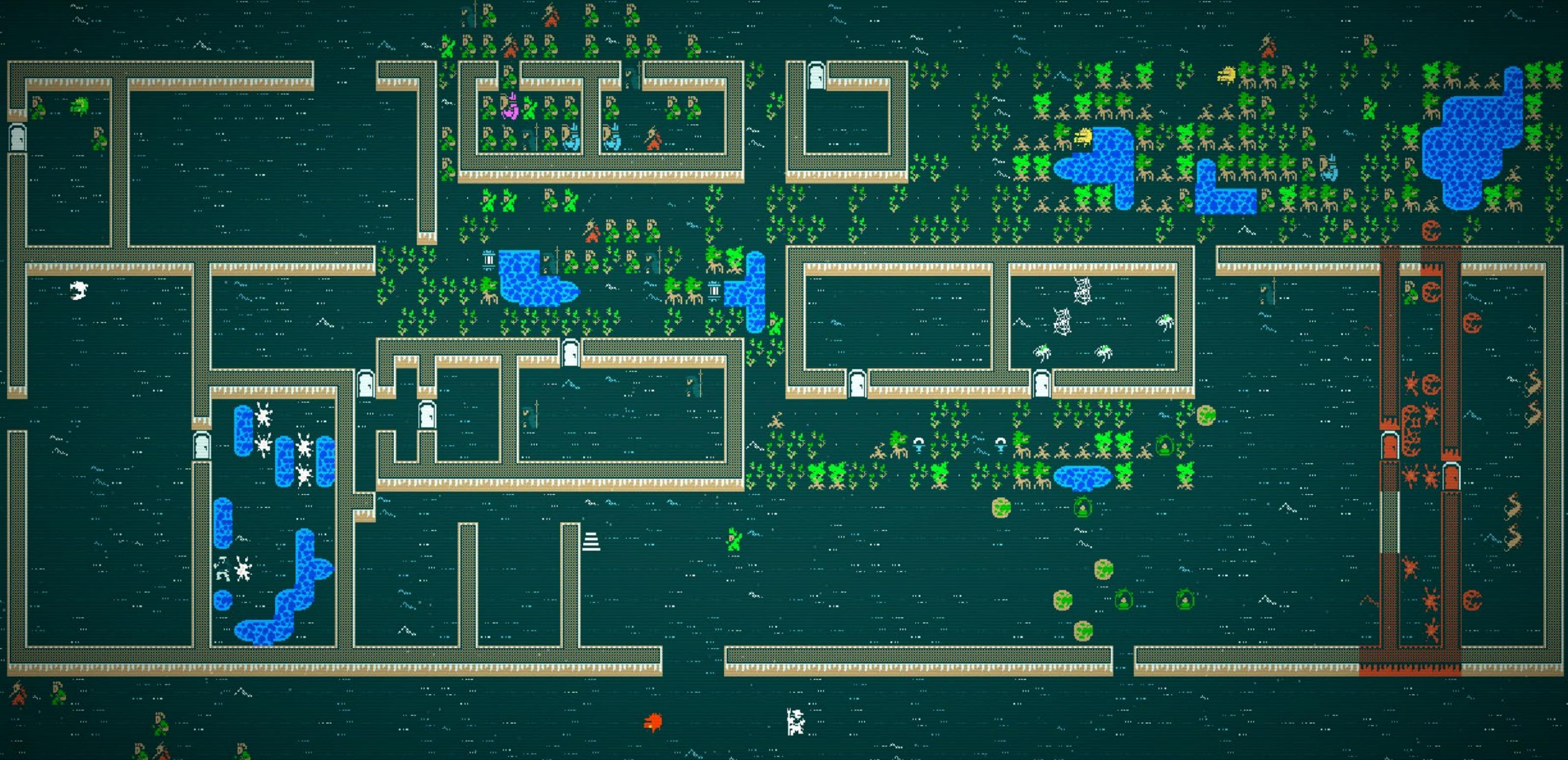
Brian Bucklew
Co-founder at Freehold Games, LLC

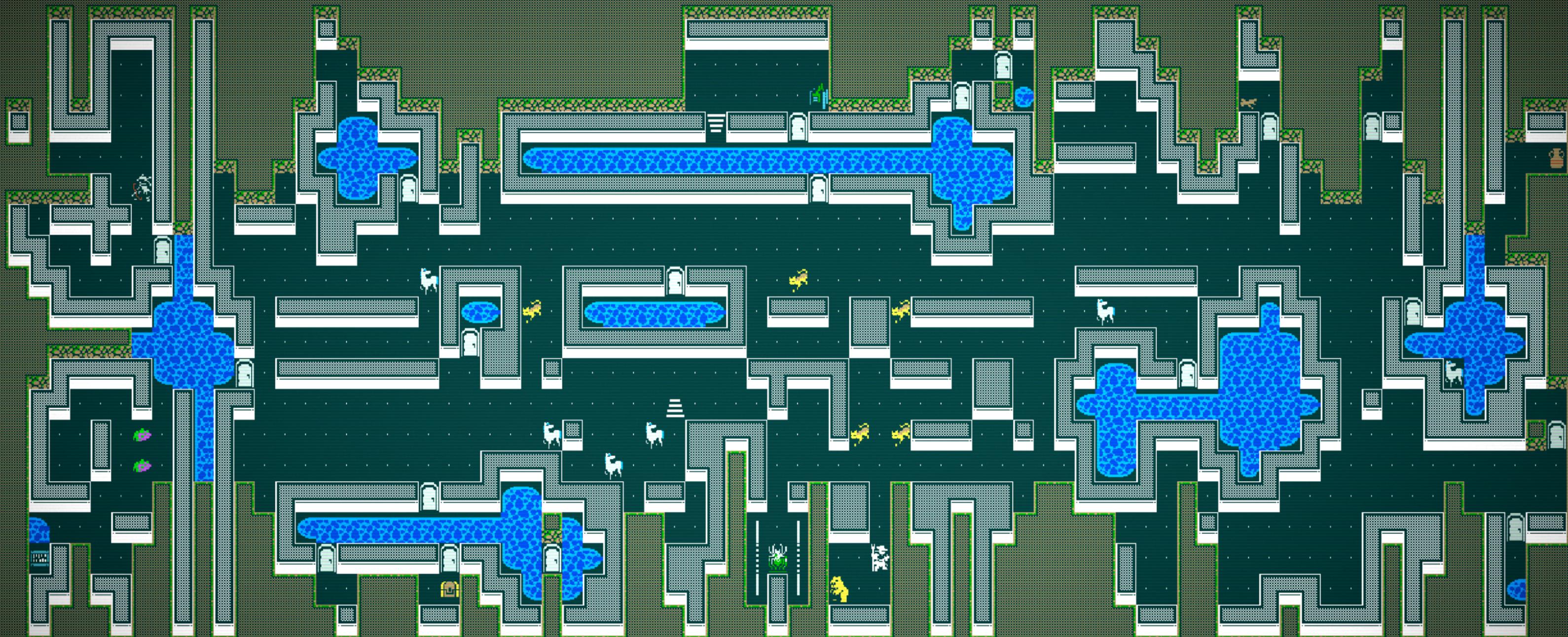
GAME DEVELOPERS CONFERENCE

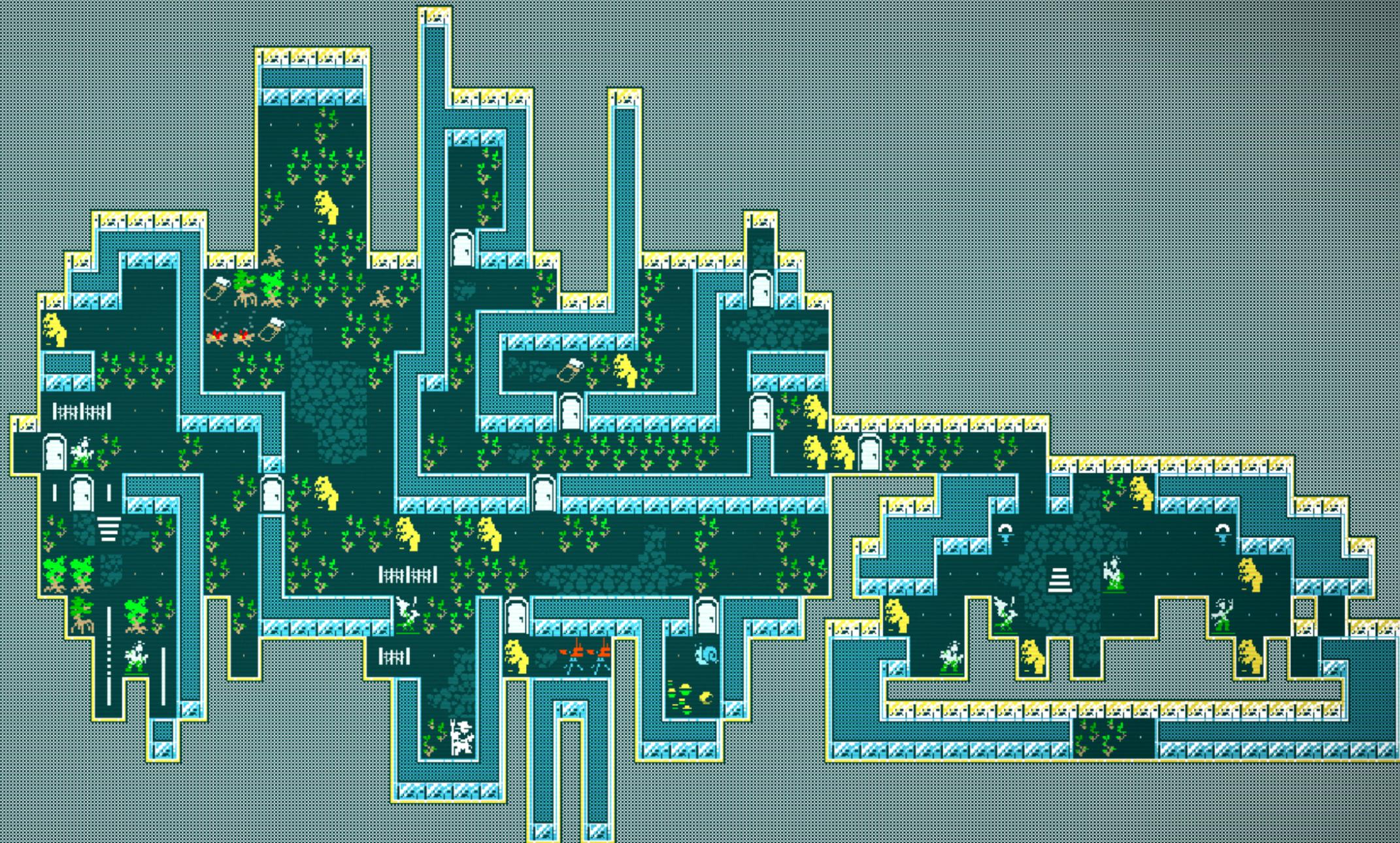
MARCH 18–22, 2019 | #GDC19

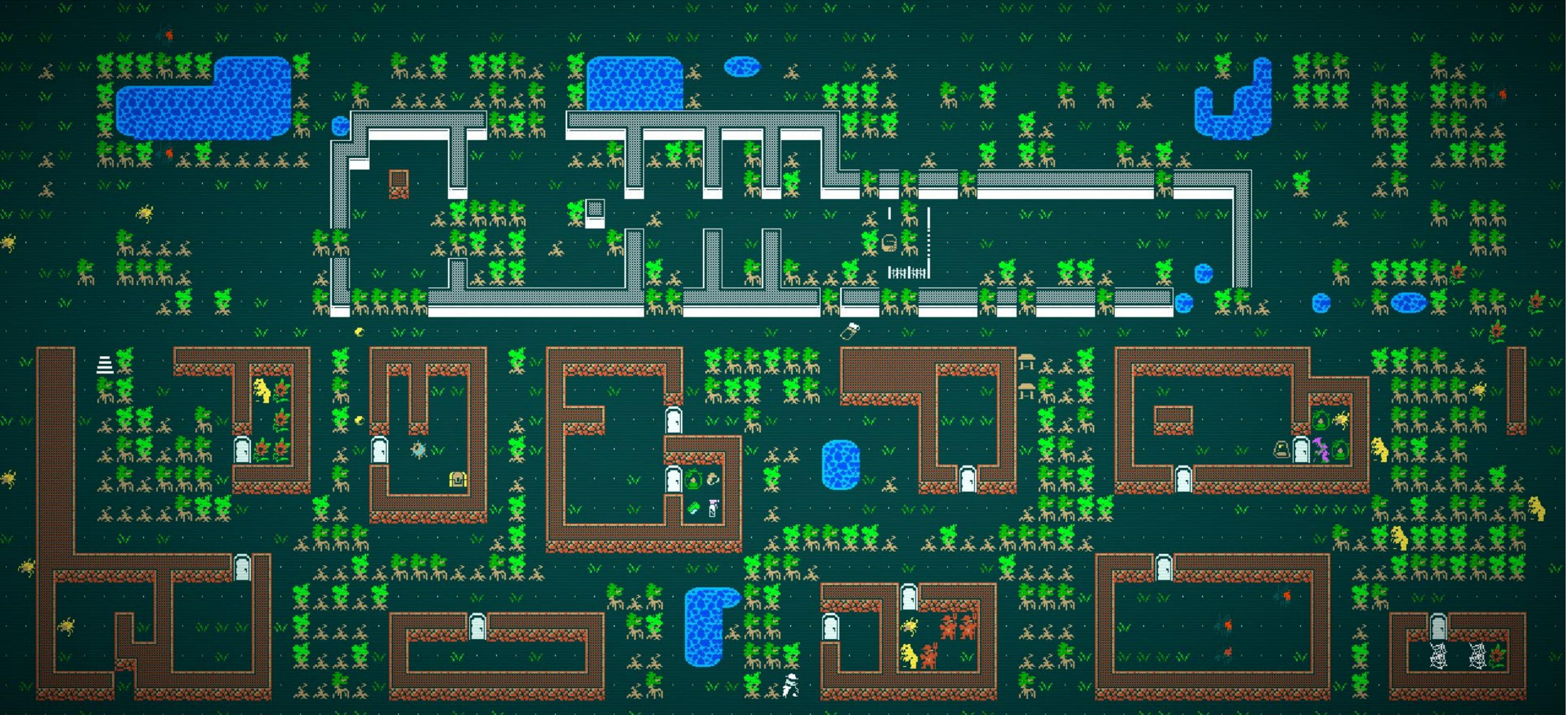
Why are we here

- *Caves of Qud* is a game with lots of procedural content
- We utilize a few novel techniques, we're going to talk about one, Wave Function Collapse (WFC)
- Let's look at the final result then we'll talk about how we got there









Wide variety of results, single algorithm

- Each of these had a very different character though the basic algorithm was identical between them
- We use an extremely powerful new texture synthesis technique called “WaveFunctionCollapse” alongside a set of other procedural tools engineered to supplement it’s unique weakness when used for map making.

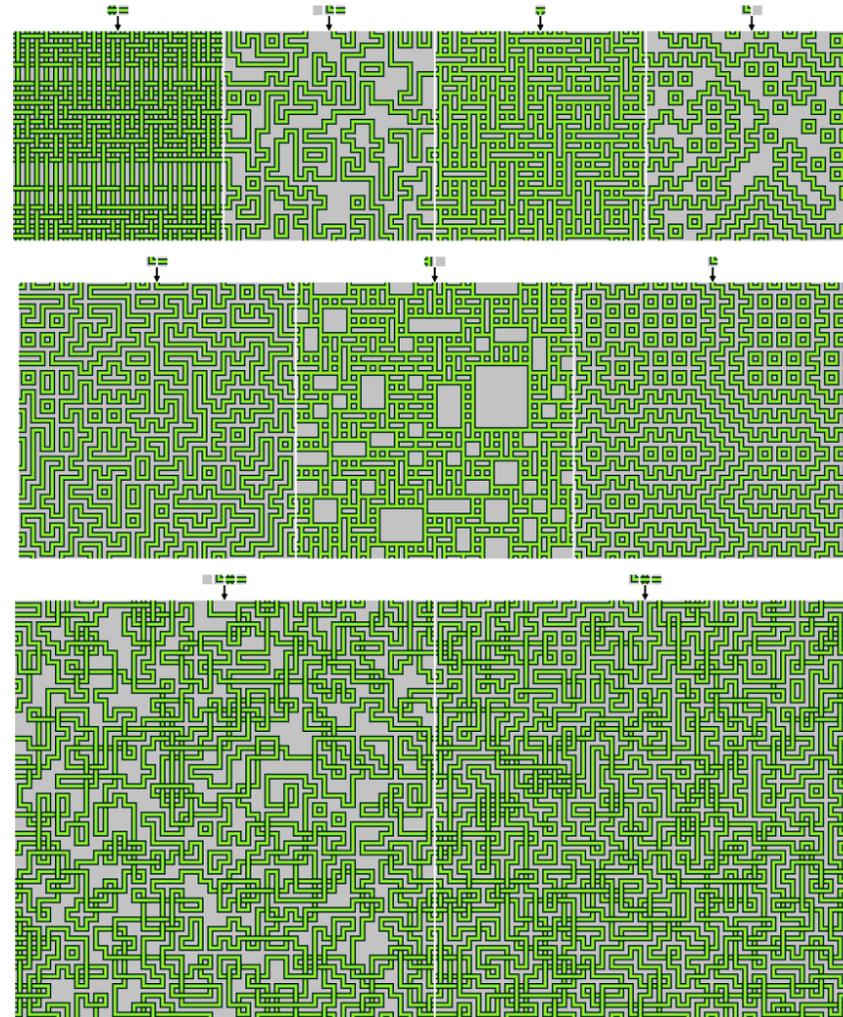
WaveFunctionCollapse

- Developed by Maxim Gumin and released as open source in 2016
- <https://github.com/mxgmn/WaveFunctionCollapse>
- *Caves of Qud* was the first commercial use, many others quickly followed

WaveFunctionCollapse Texture and Tiles

- WFC has two primary modes of function, tile maps and textures

The tilemap generation mode creates tile set solutions via propagation of defined tile adjacency constraints.



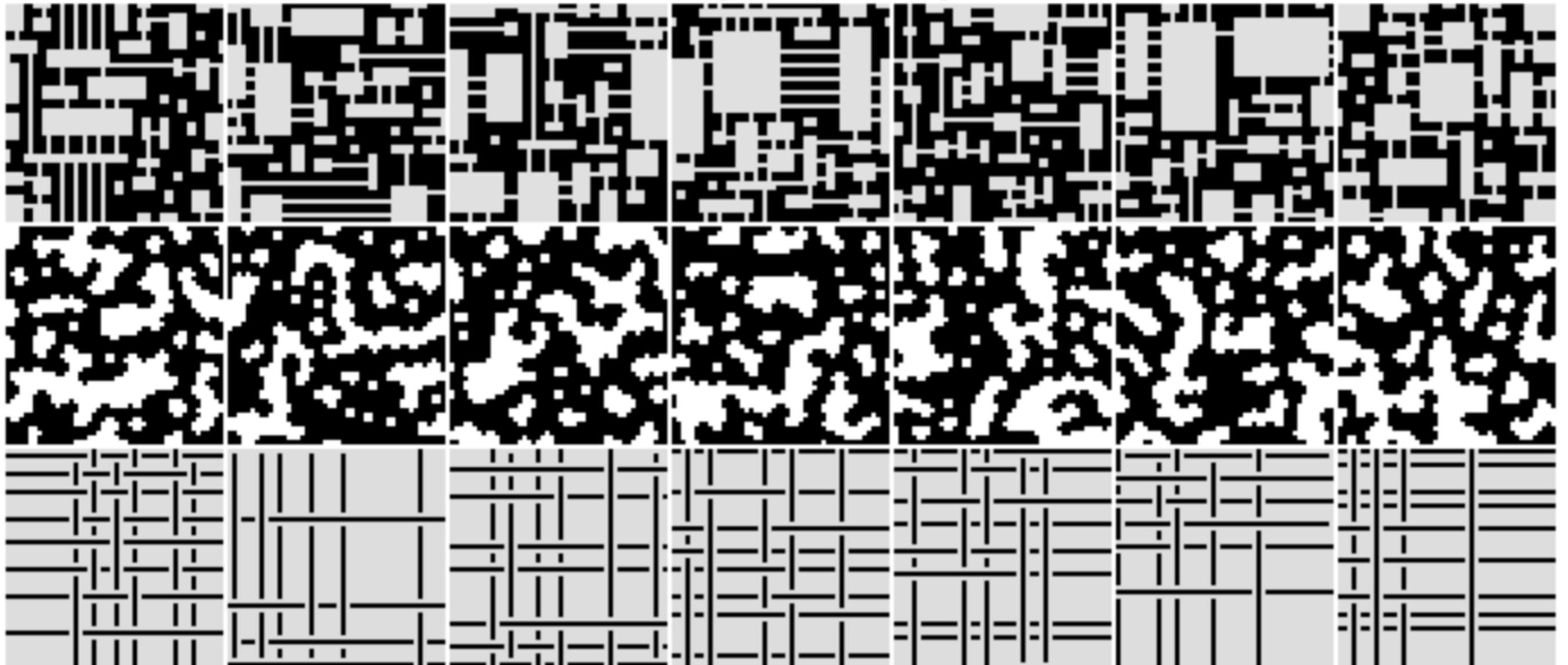
Texture Mode

- In *Caves of Qud* we use the other mode, Texture Mode
- Easy training inputs (small ~16x16px training images that can be easily created in tools like mspaint)
- Powerful outputs (arbitrarily large output textures that are locally similar to the input)

Texture mode, wow!

7 sample outputs

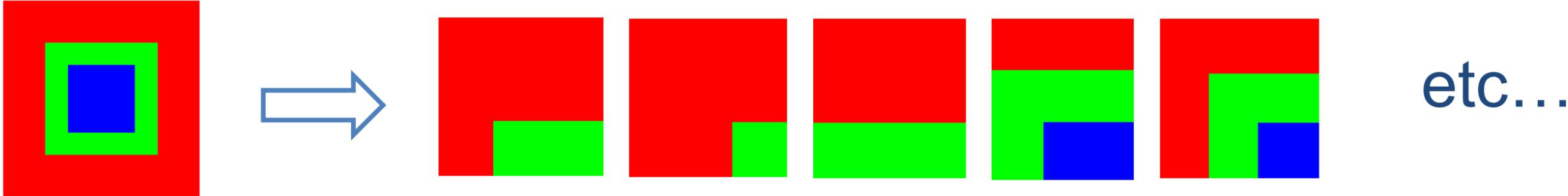
Input



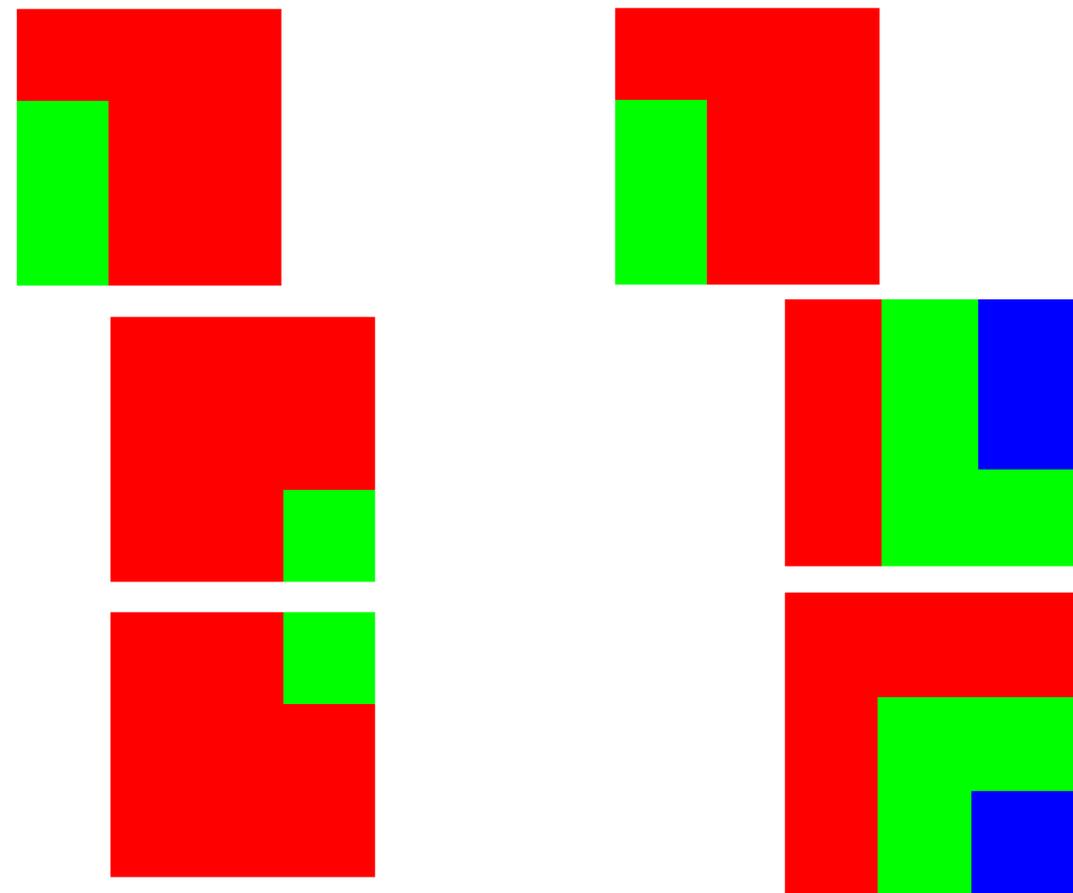
What's going on here

1. The input is divided into NxN tiles and their overlap with other tiles is calculated
2. The output is initialized with each pixel being a full superposition of possible output tiles.
3. The lowest entropy NxN area is selected from the output and one option is selected at random from the remaining possibilities.
4. New information based on that selection are propagated to adjacent areas, removing possibilities that won't properly overlap.
5. If any elements are still uncollapsed Goto 2!

Identify the NxN patterns

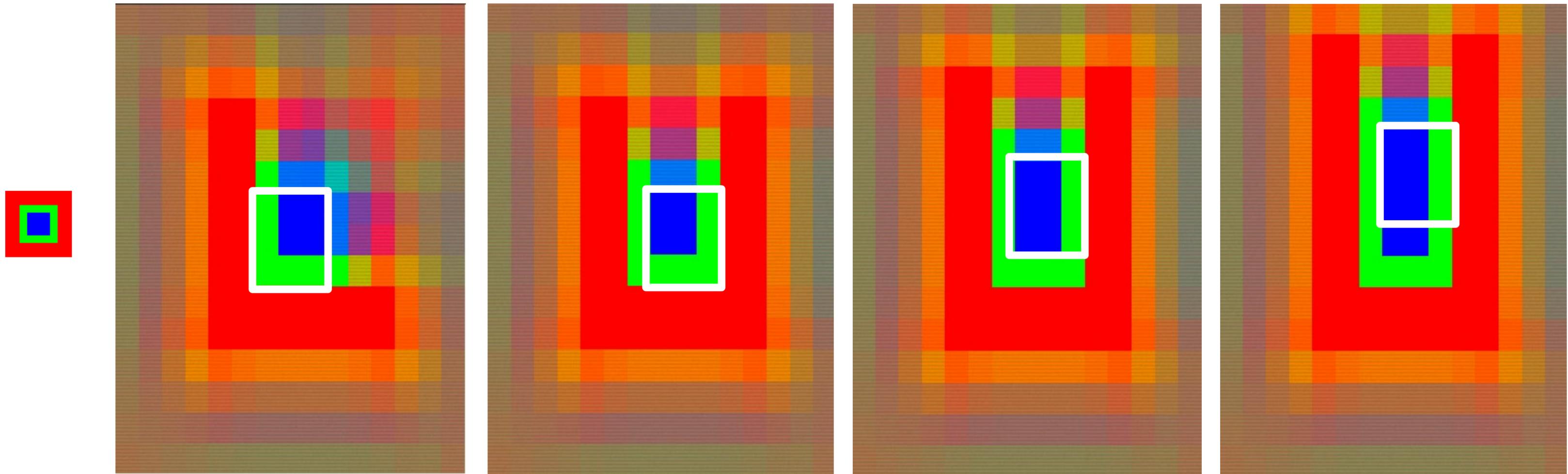


Identify tiles that overlap at offsets of 1..n in each dimension

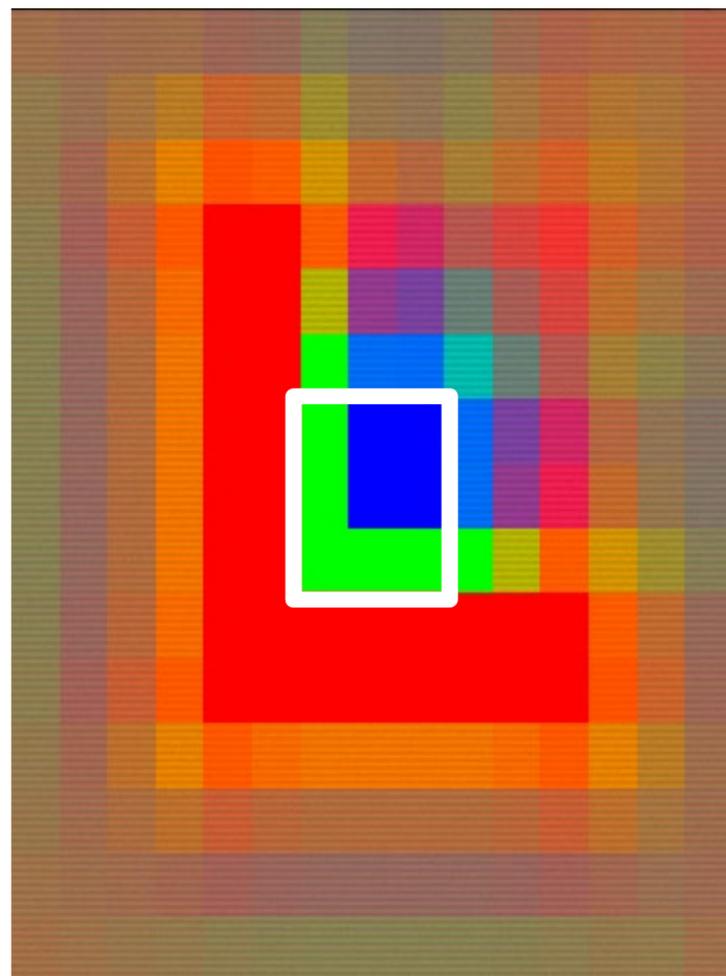


etc...

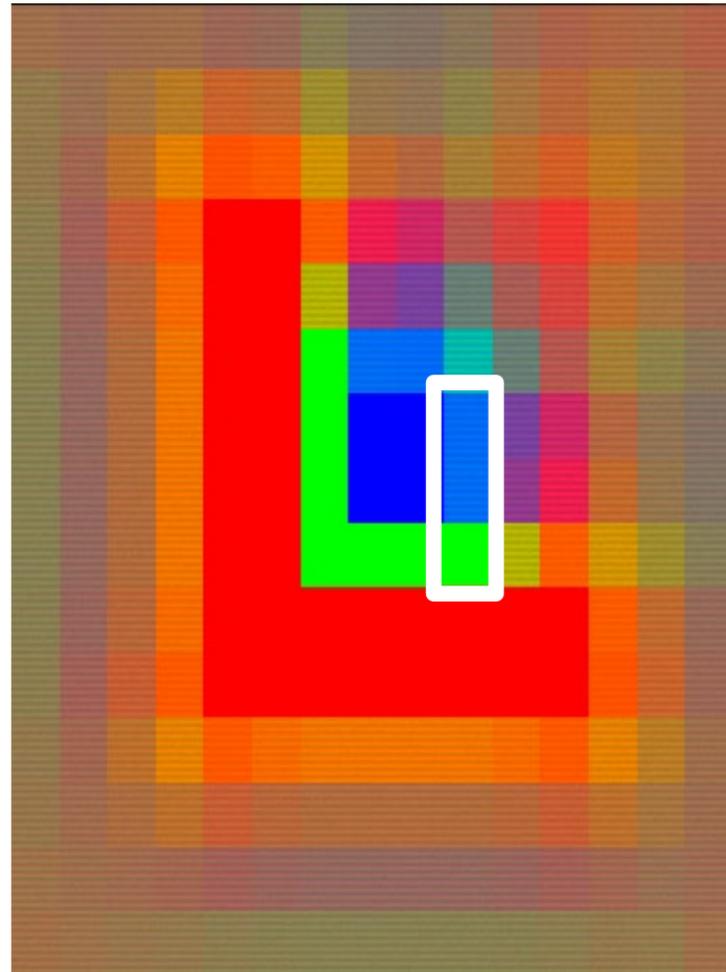
Collapse one segment, propagate information, repeat...



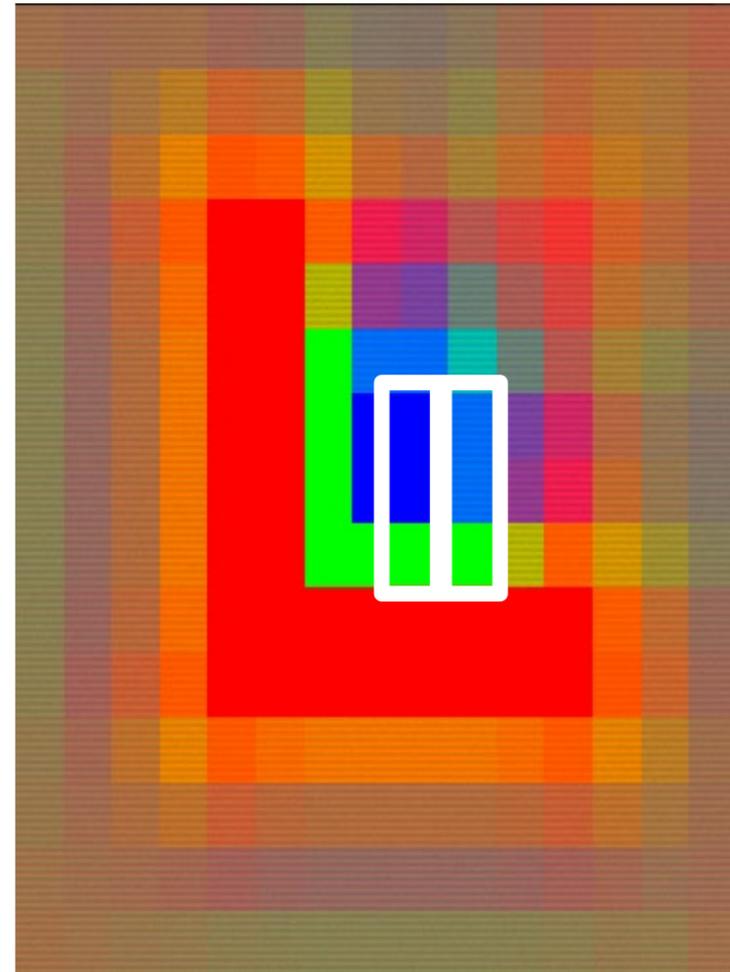
Take a look at the first step...



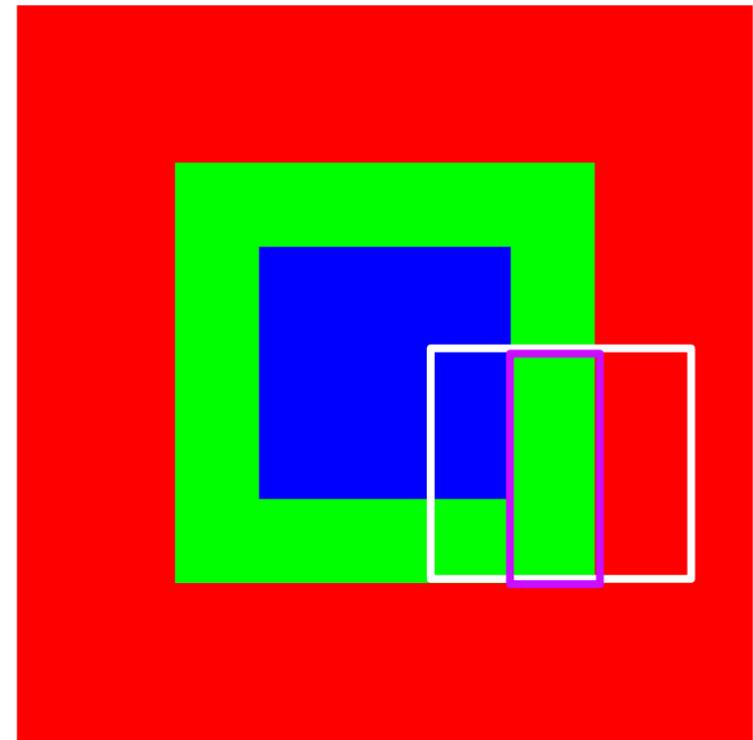
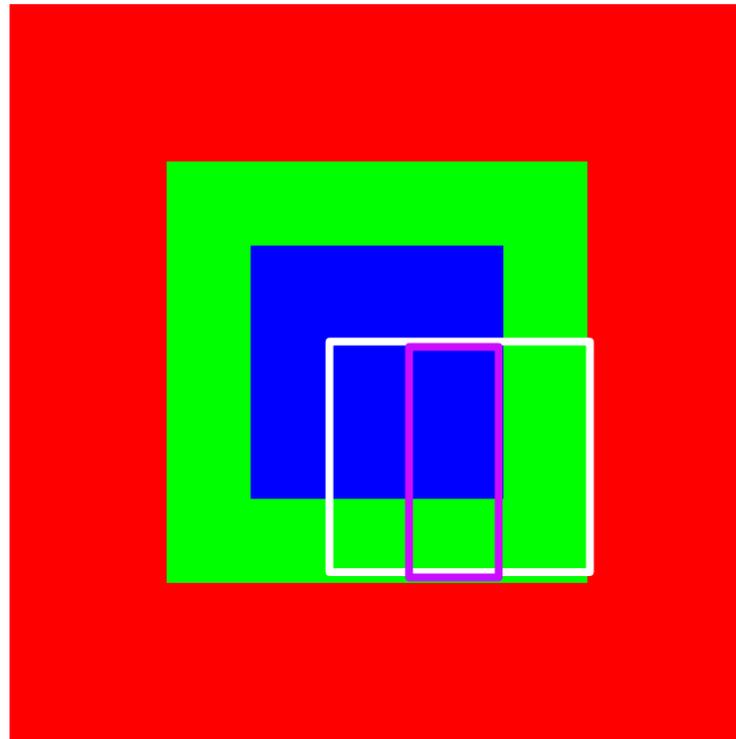
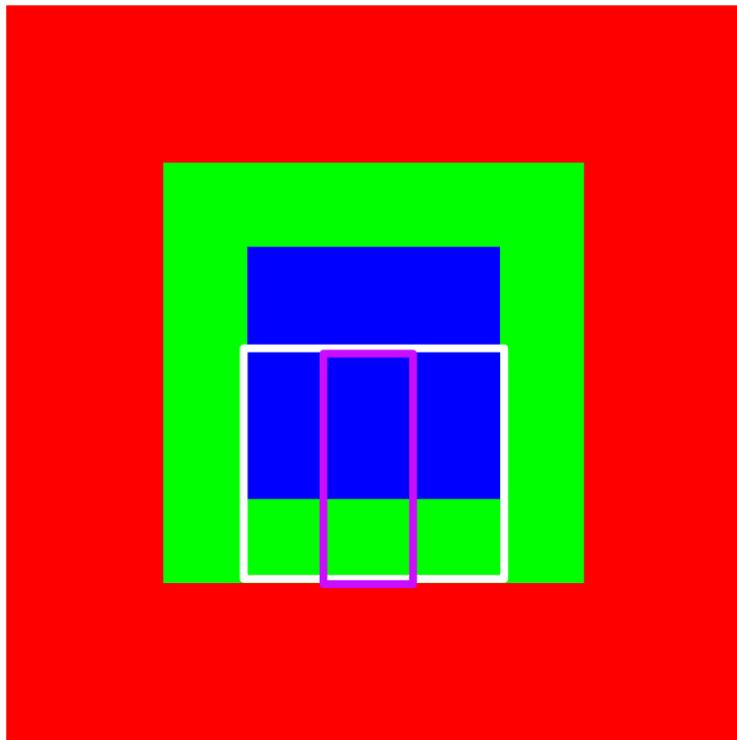
Focus in on the three pixels here...



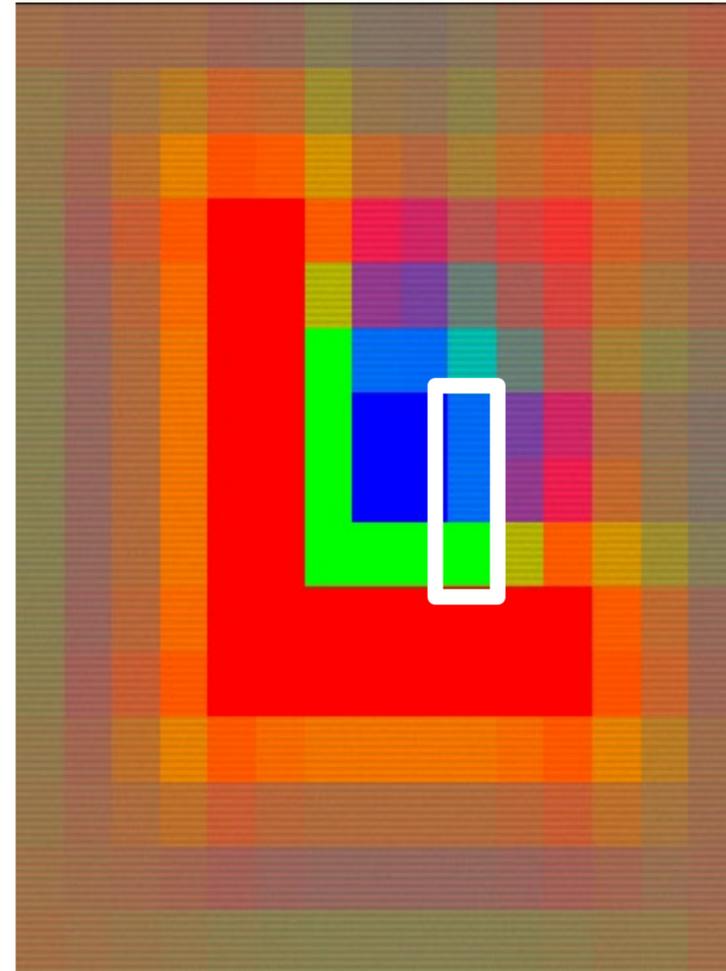
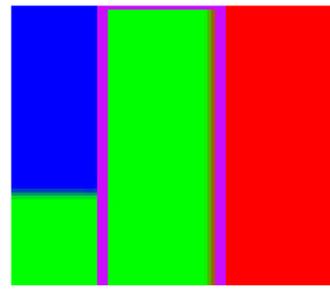
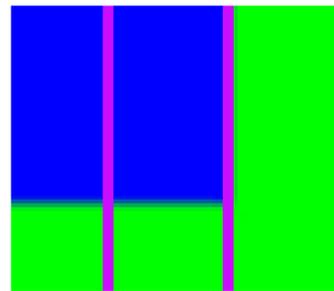
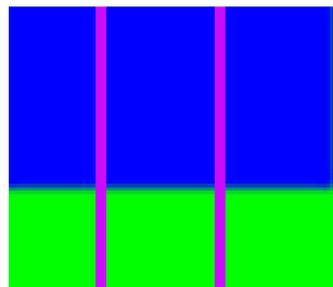
The pattern to the left is blue, blue,
green for certain.



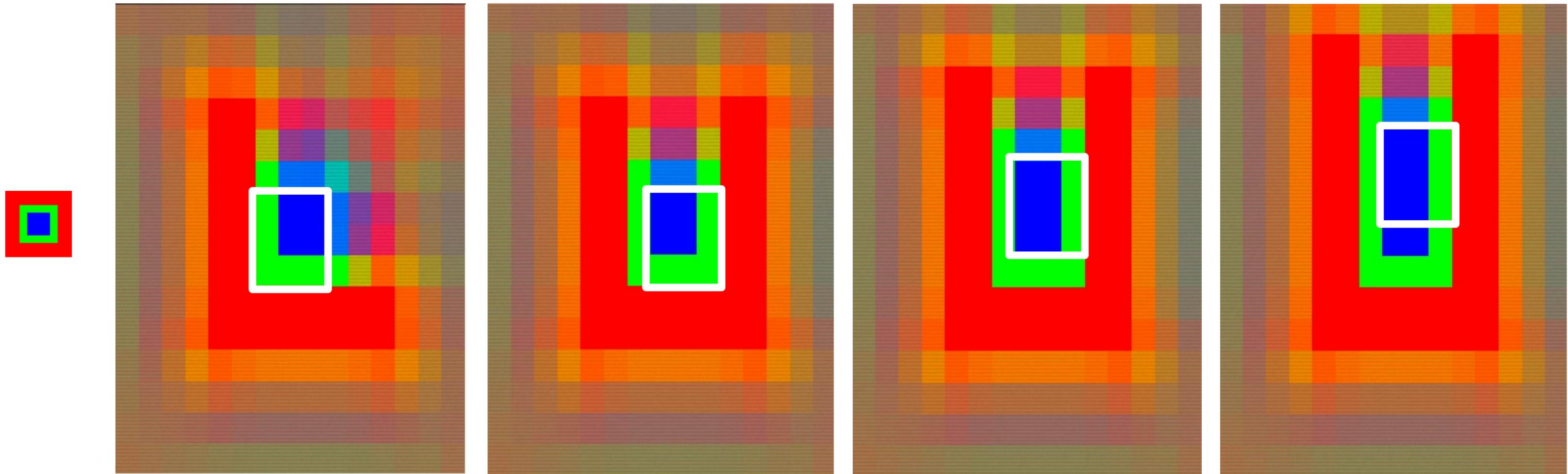
Those three pixels know blue, blue, green is to their left and thus could end up as any of the three pixels highlighted in magenta because those three 3x3 areas share the blue blue green pattern in the overlapping area



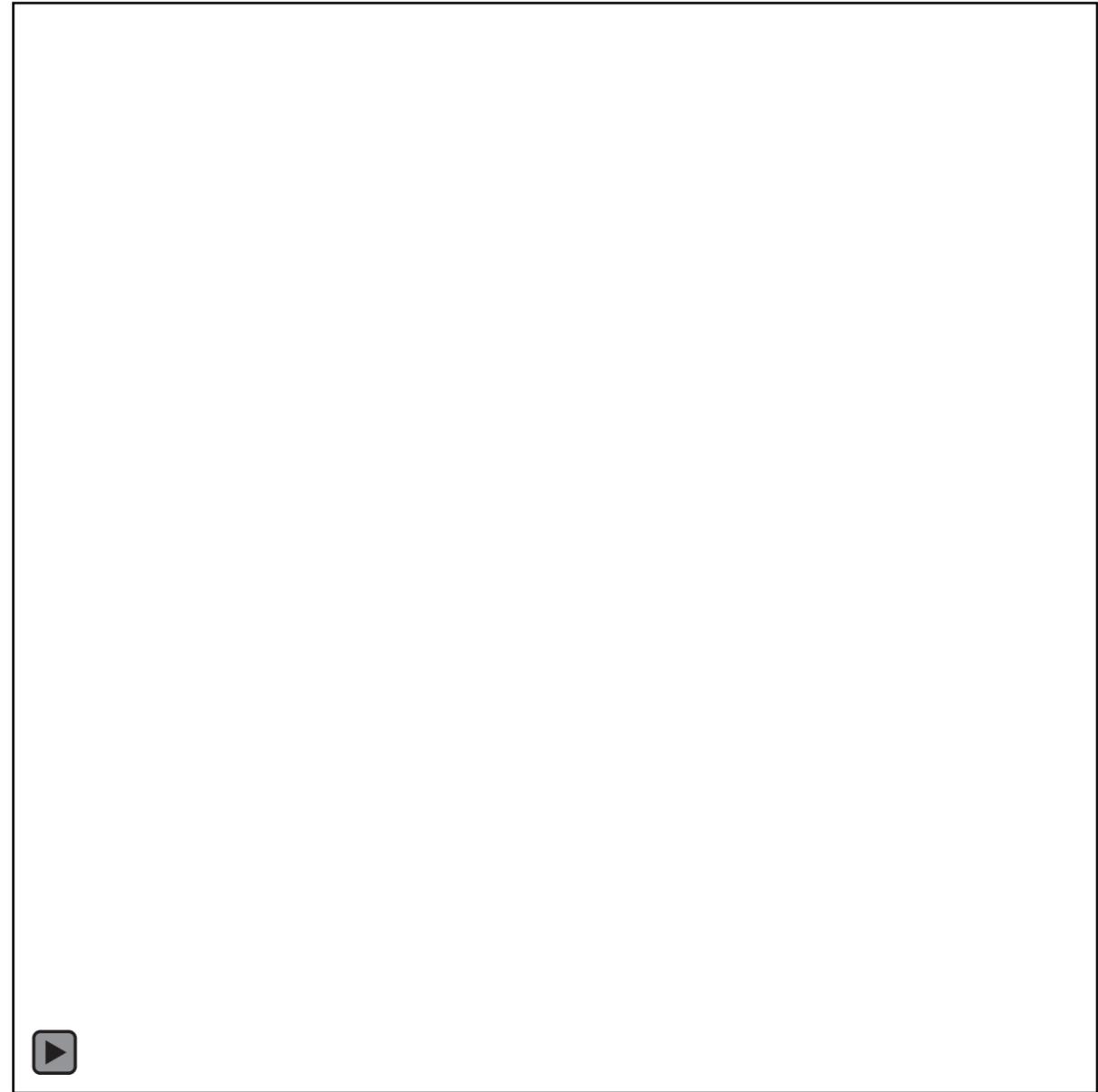
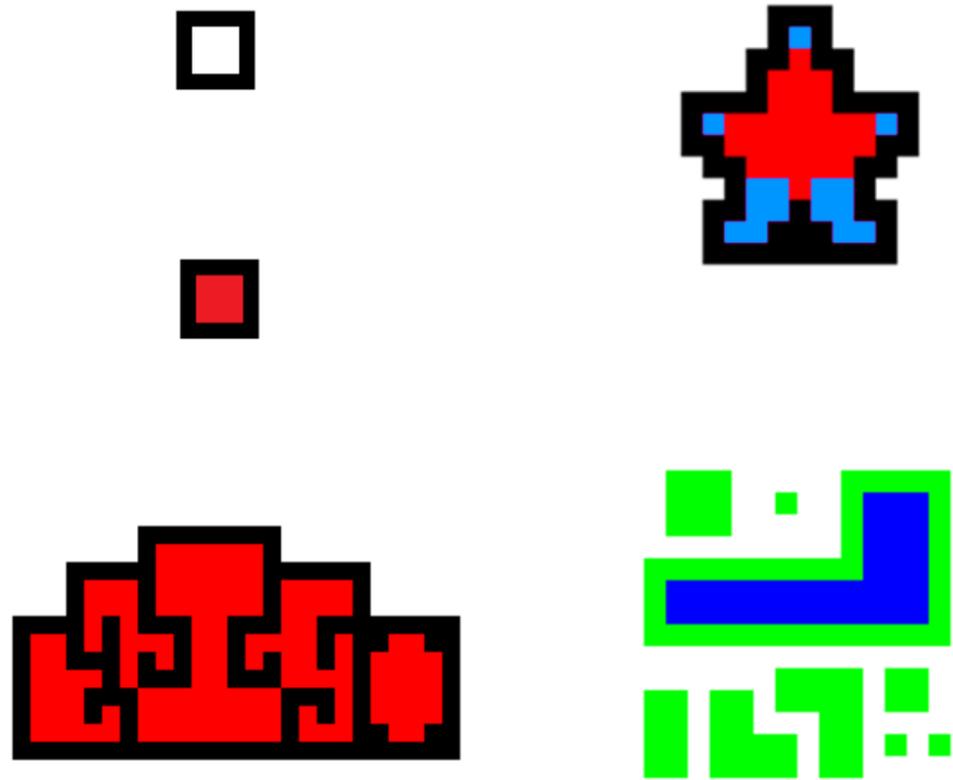
2 blue options, 1 green option for the top two pixels.
Only green for the bottom.



Collapse one segment, propagate information, repeat...

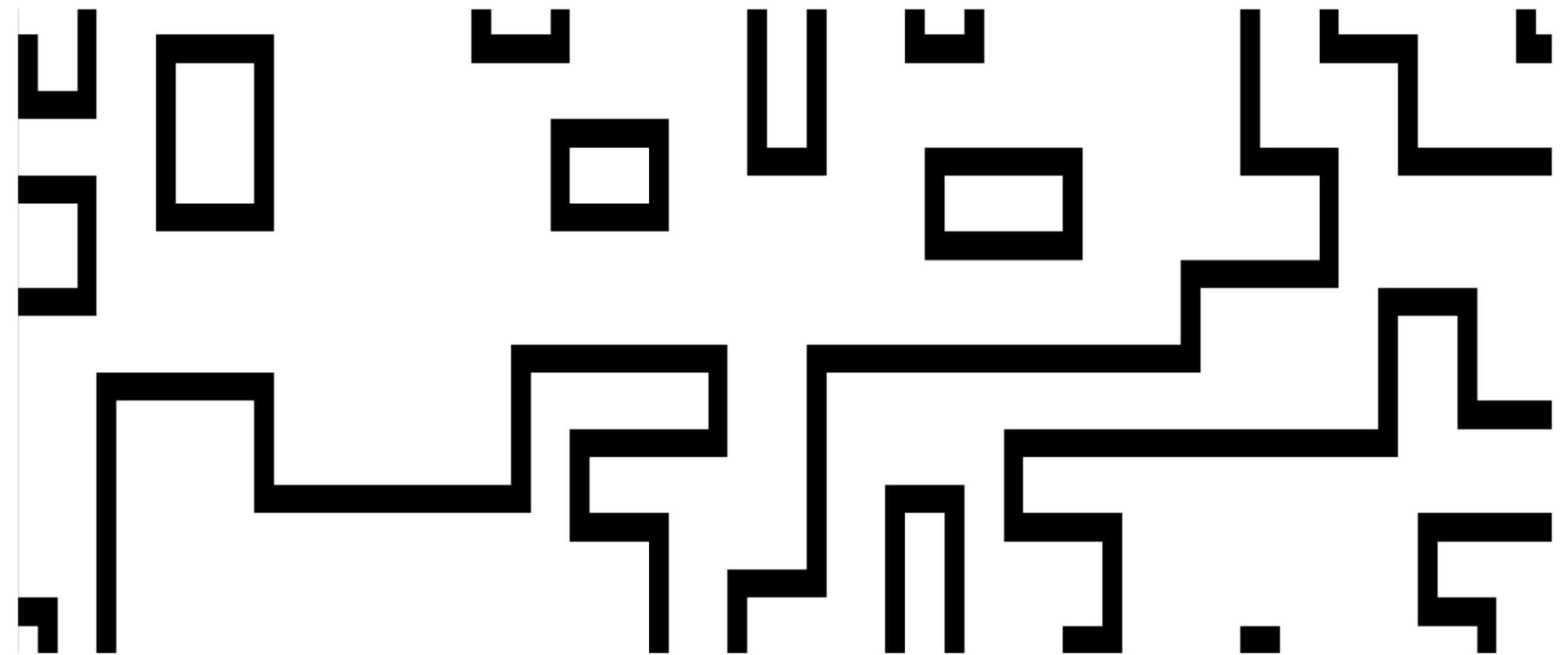
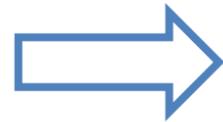
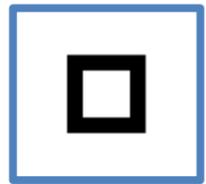


Process Demo



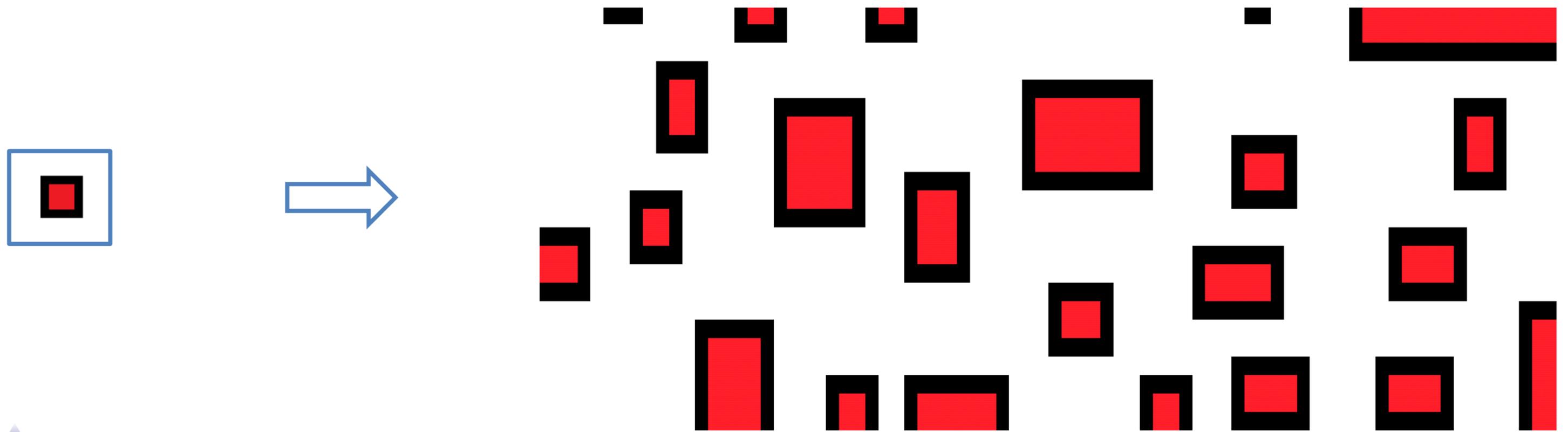
Experiments in texture mode

Let's look at a couple examples. Here was something that represents my first attempts. Neat but lacks controllability.



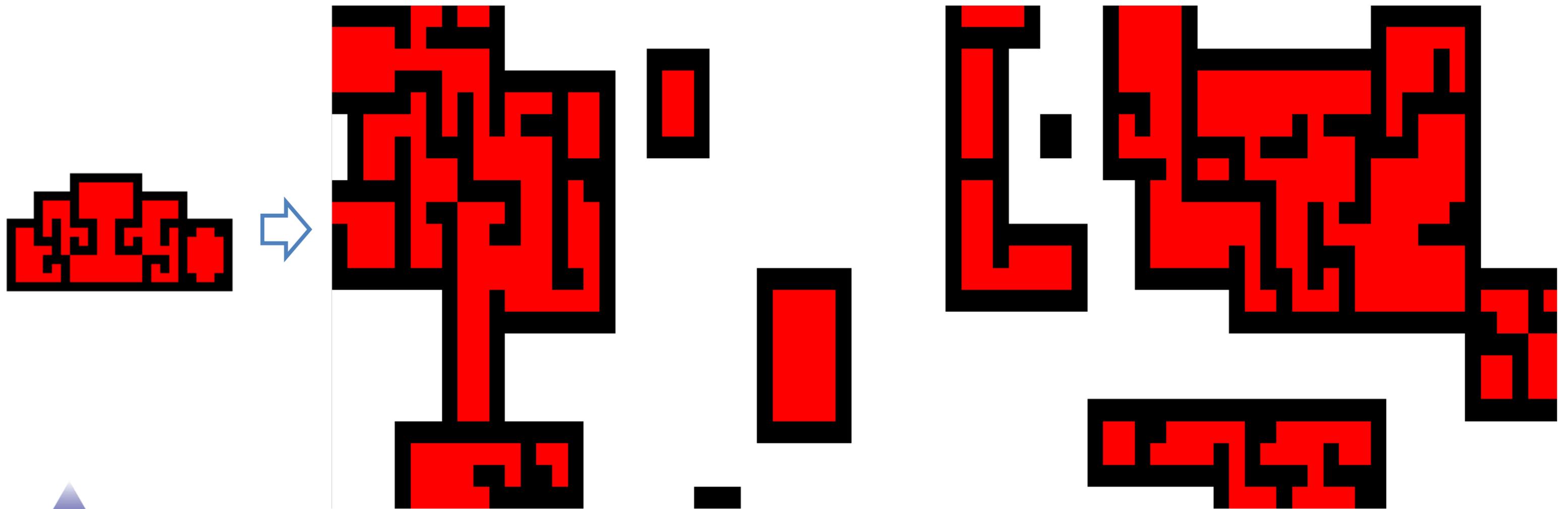
Texture mode, an example (2)

Using an additional color to hint the “inside” pixels was an effective solution to improve control.



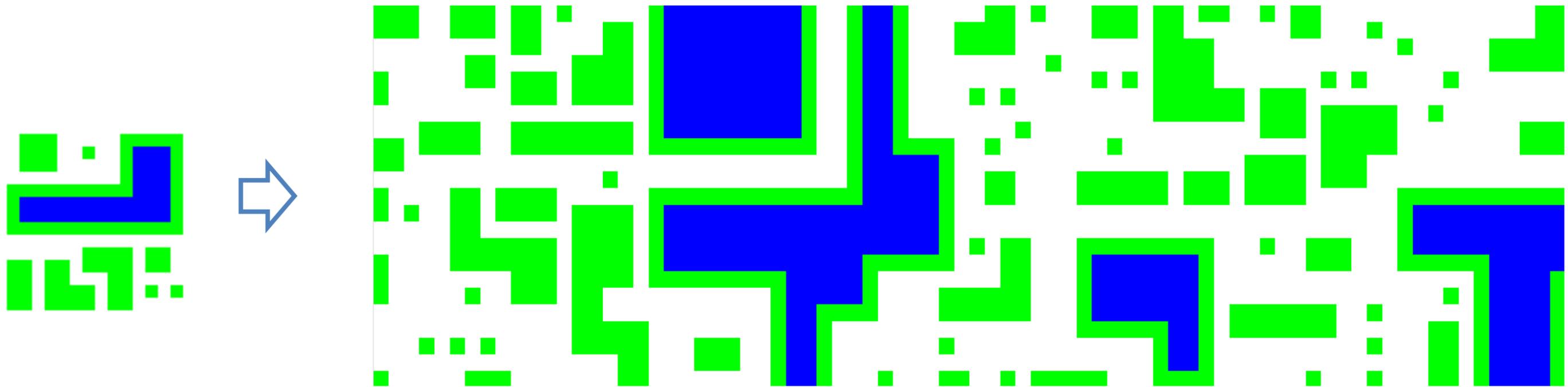
Texture mode, an example (3a)

We can create some nice results!



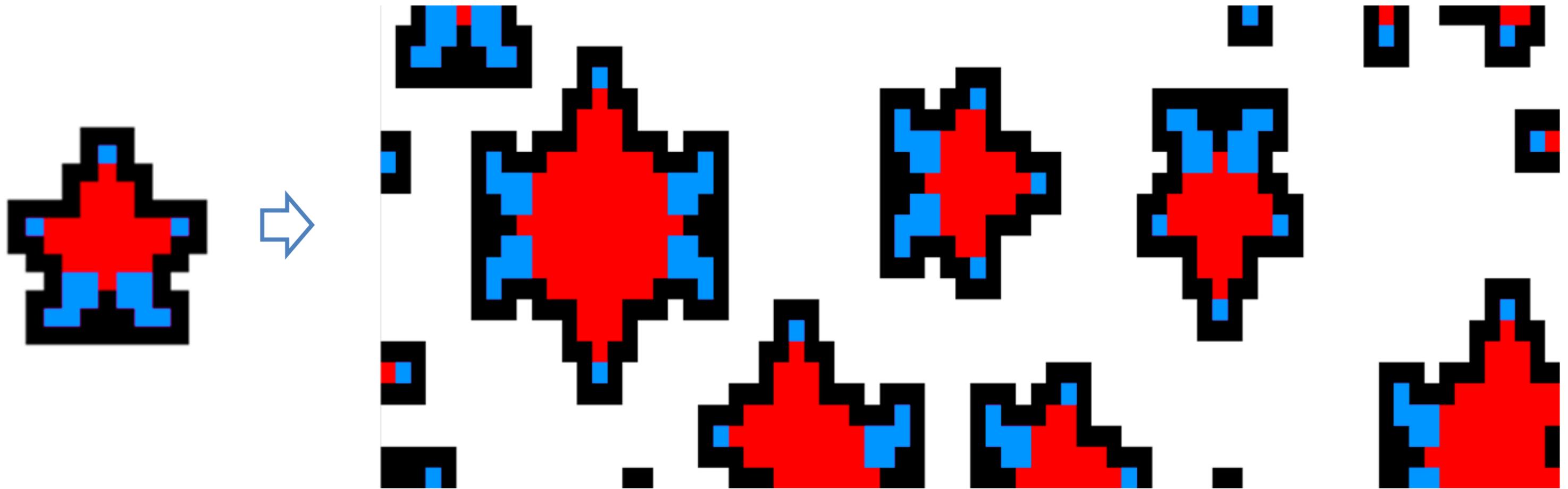
Texture mode, an example (3b)

We can create some nice results!



Texture mode, an example (3c)

We can create some nice results!



Quick Code Example

```
1. var model = new OverlappingModel(input, N:3, width:48, height:48, periodicInput:true, periodic:false, symmetry:8, ground:0 );  
2. model.Run(random.Next, limit:0);  
3. model.Graphics().Save($"output.png");
```

Input - the training image

N - How large of blocks (NxN) to sample from the input as input patterns. (higher N leads to rising CPU and memory cost)

Width - The output width

Height - The output height

periodicInput - Whether to sample the input across edges

periodic - Whether the output should be sampled across edges to create edge-wrapping output

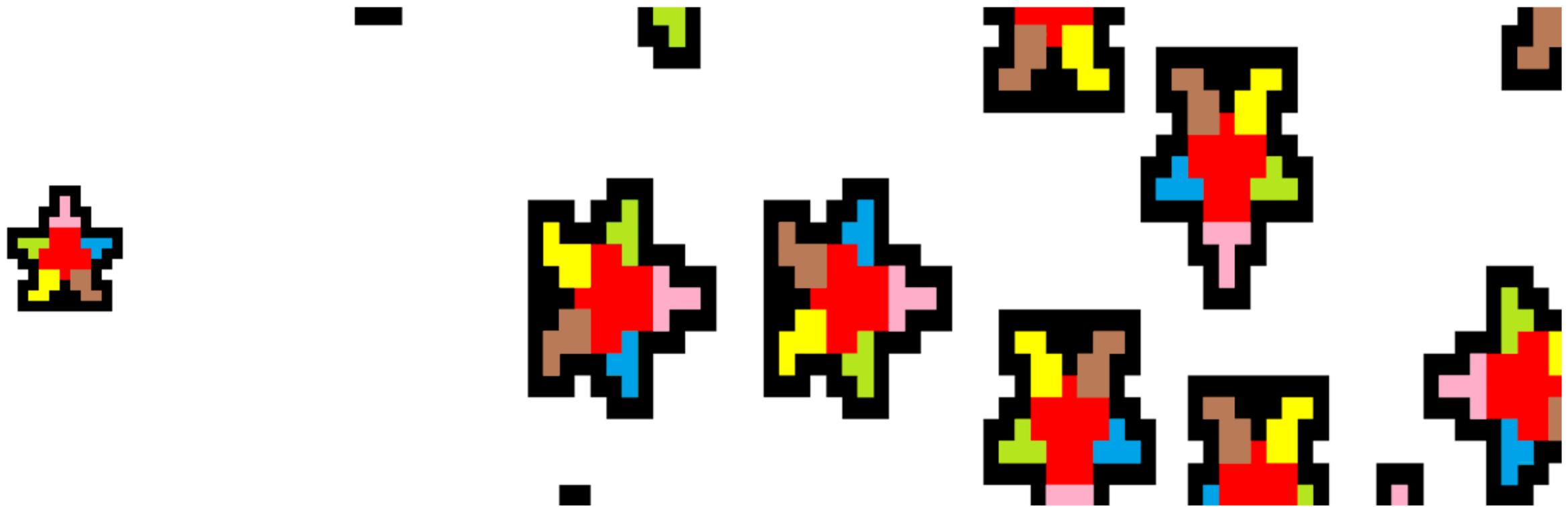
symmetry - a value between 1..8, indicating how many reflection and rotation symmetries should be sampled from the input

But ... Problem 1 - Homogeny

It just goes on forever in every direction, there is no inherent large scale structure

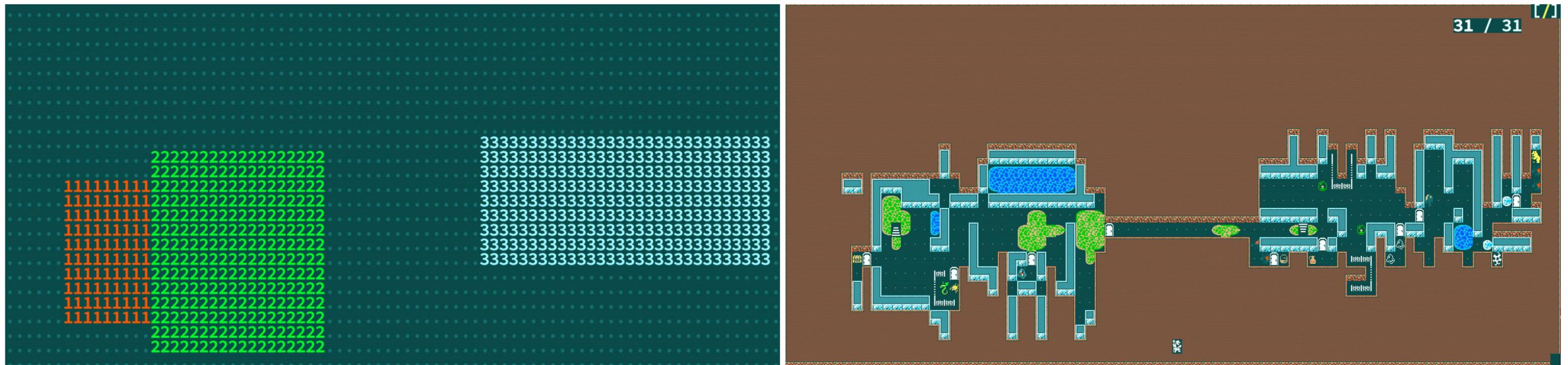
Problem 2 - Overfitting

1. Adding more detail often results in overfitting small details, reducing variability of the output.



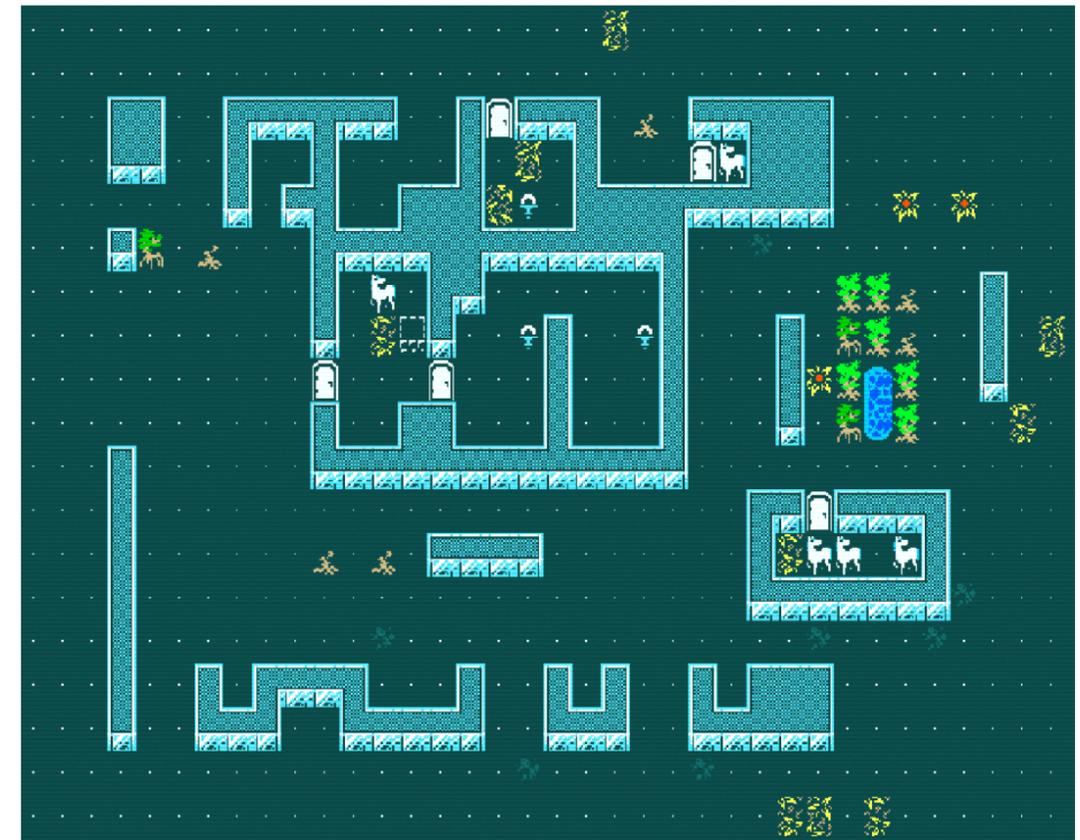
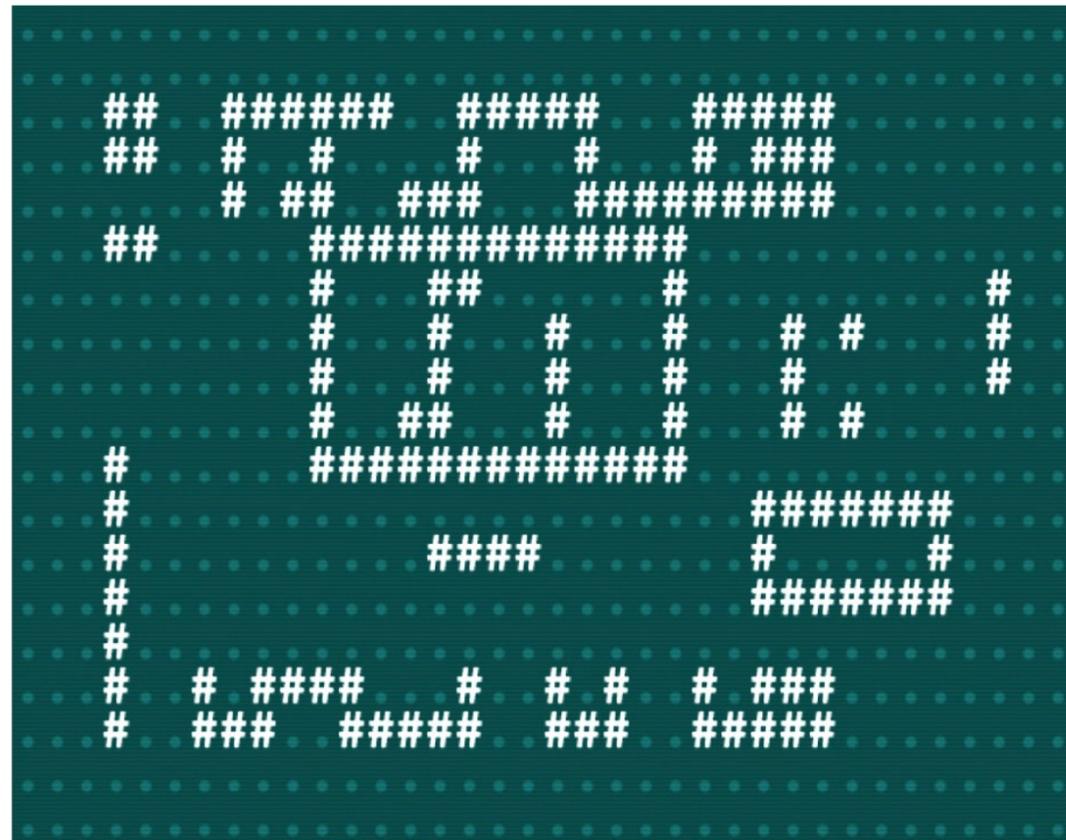
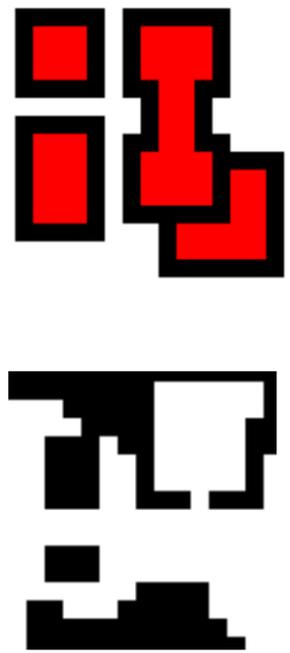
Solution to Homogeneny

Partition large scale chunks whose interior walls are generated by WFC. Additional details are added in subsequent passes.

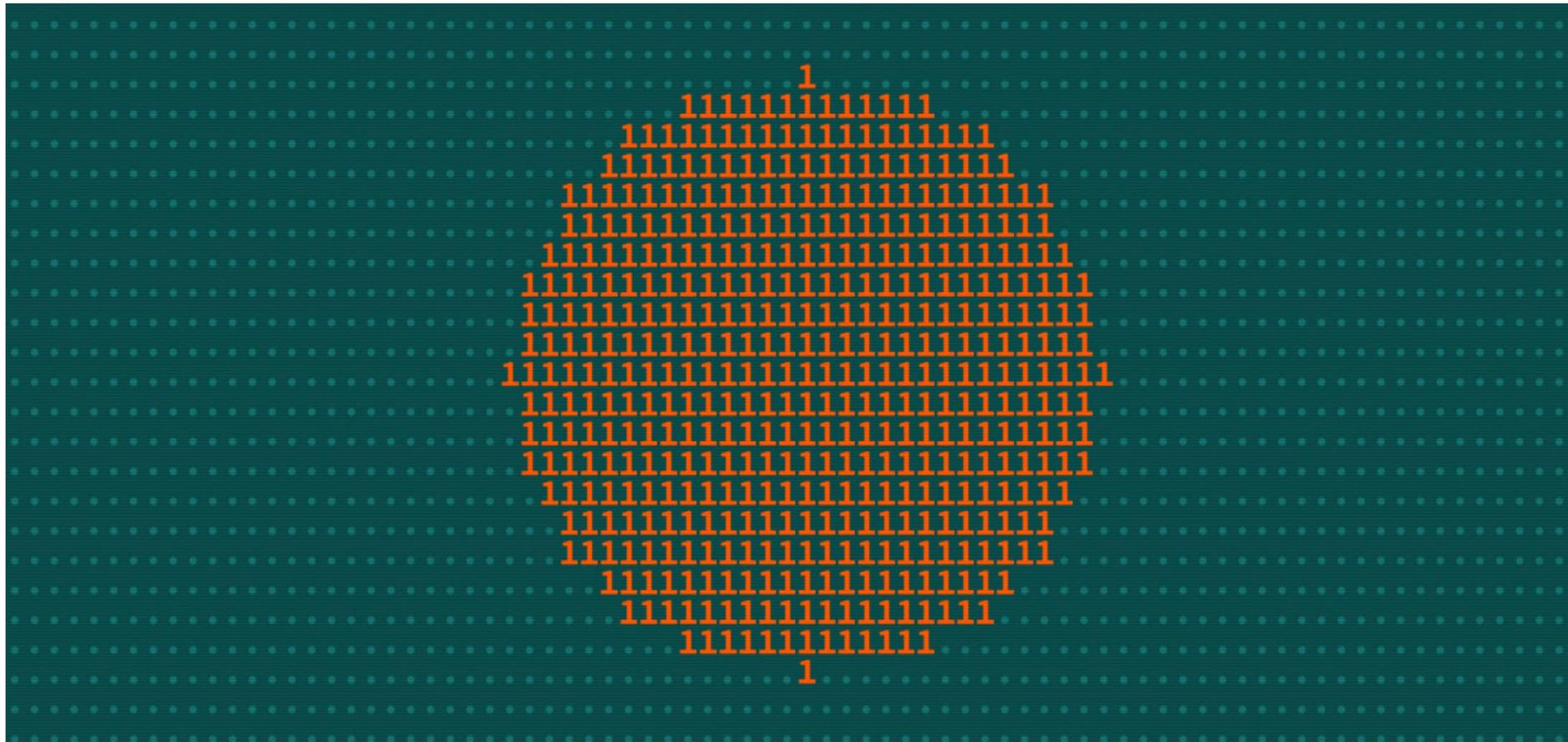


Solution to Overfitting

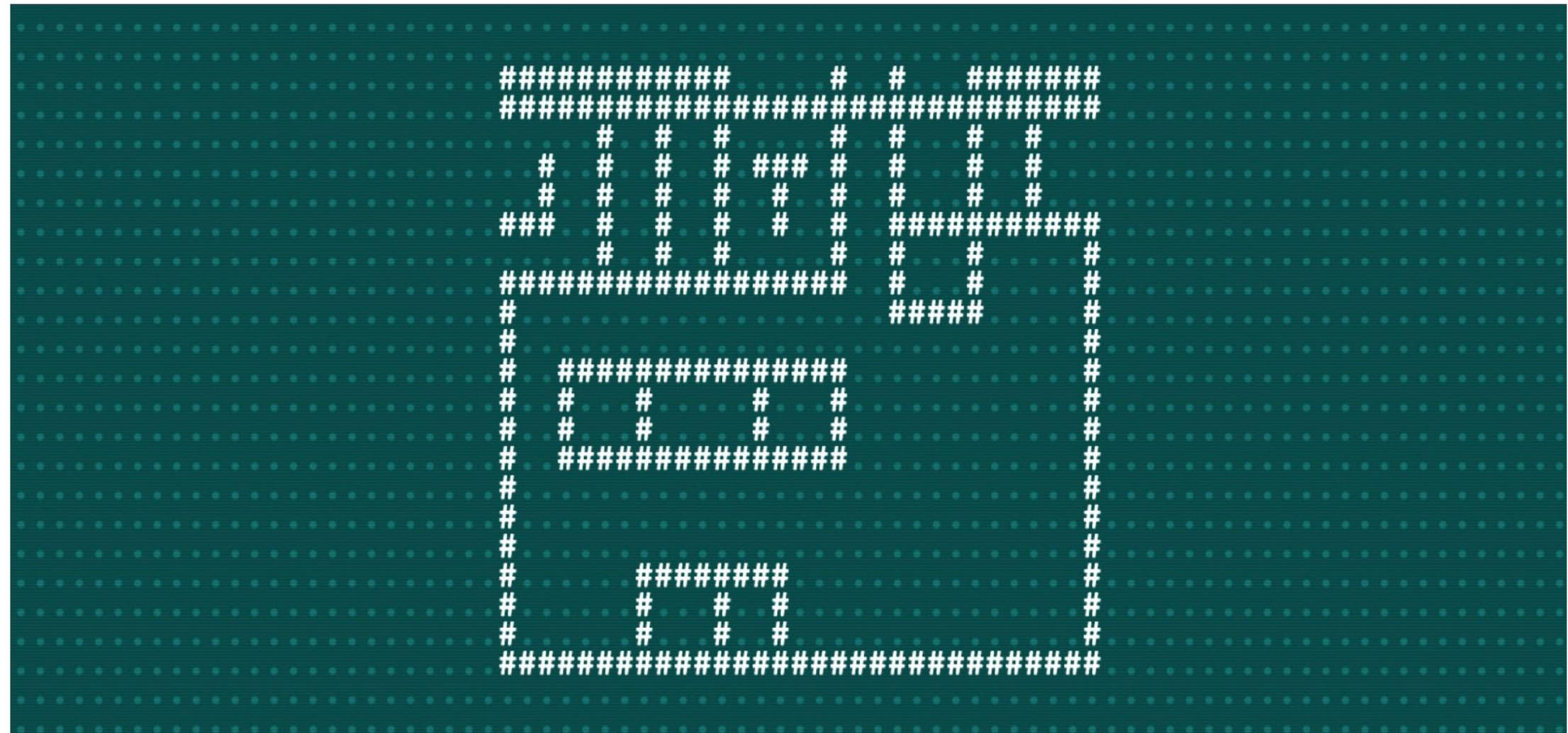
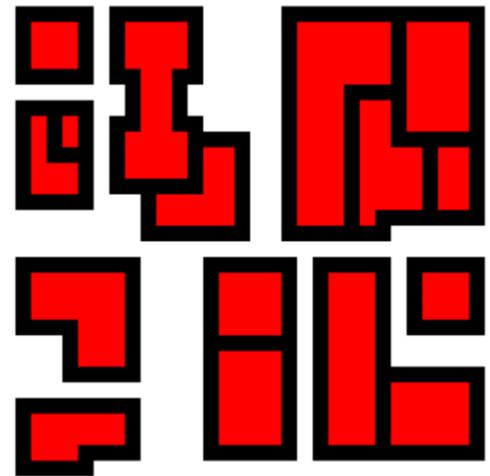
Use WFC to create overall architecture then create additional detailing and connectivity (doors, etc) via additional generative passes.



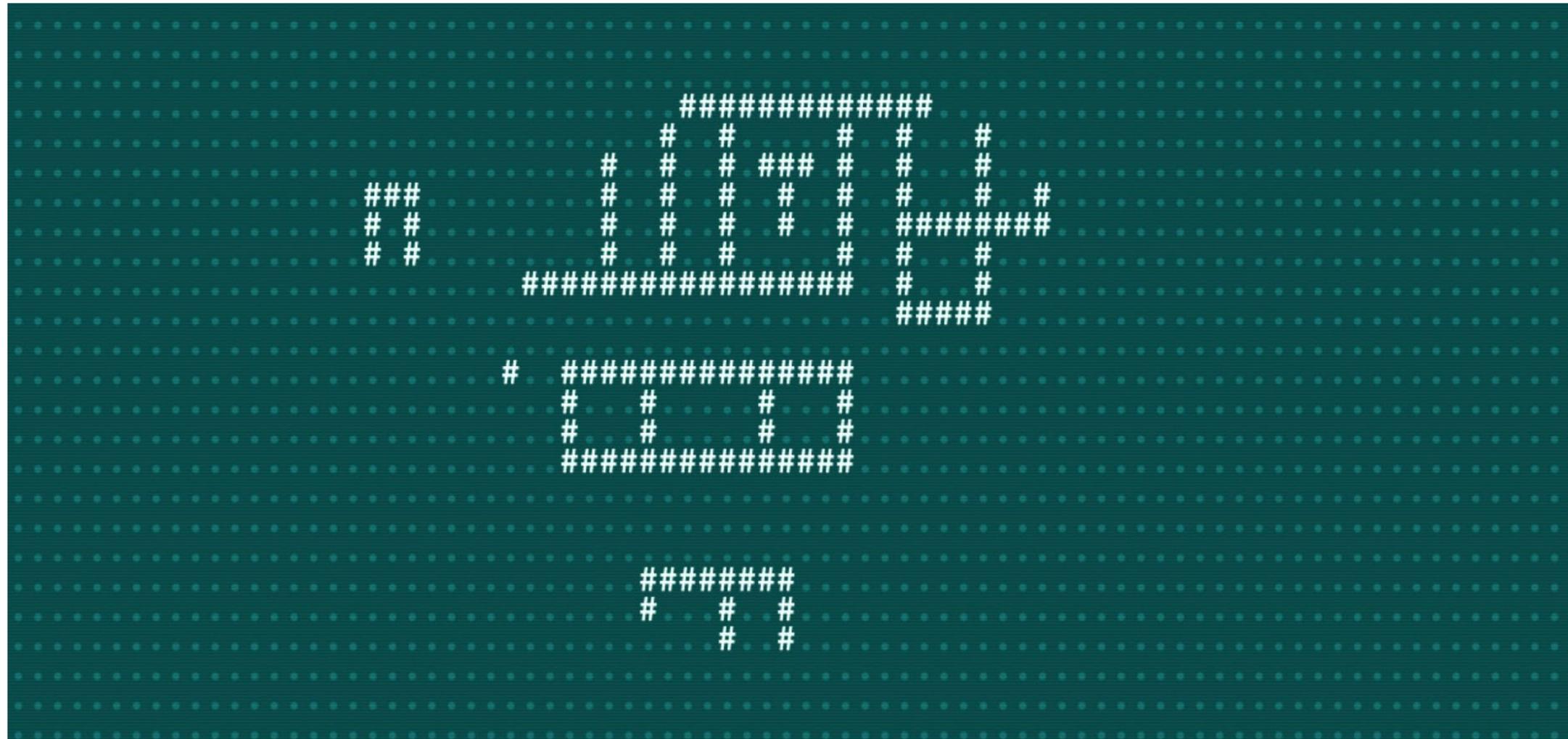
Full Example - Segmentation



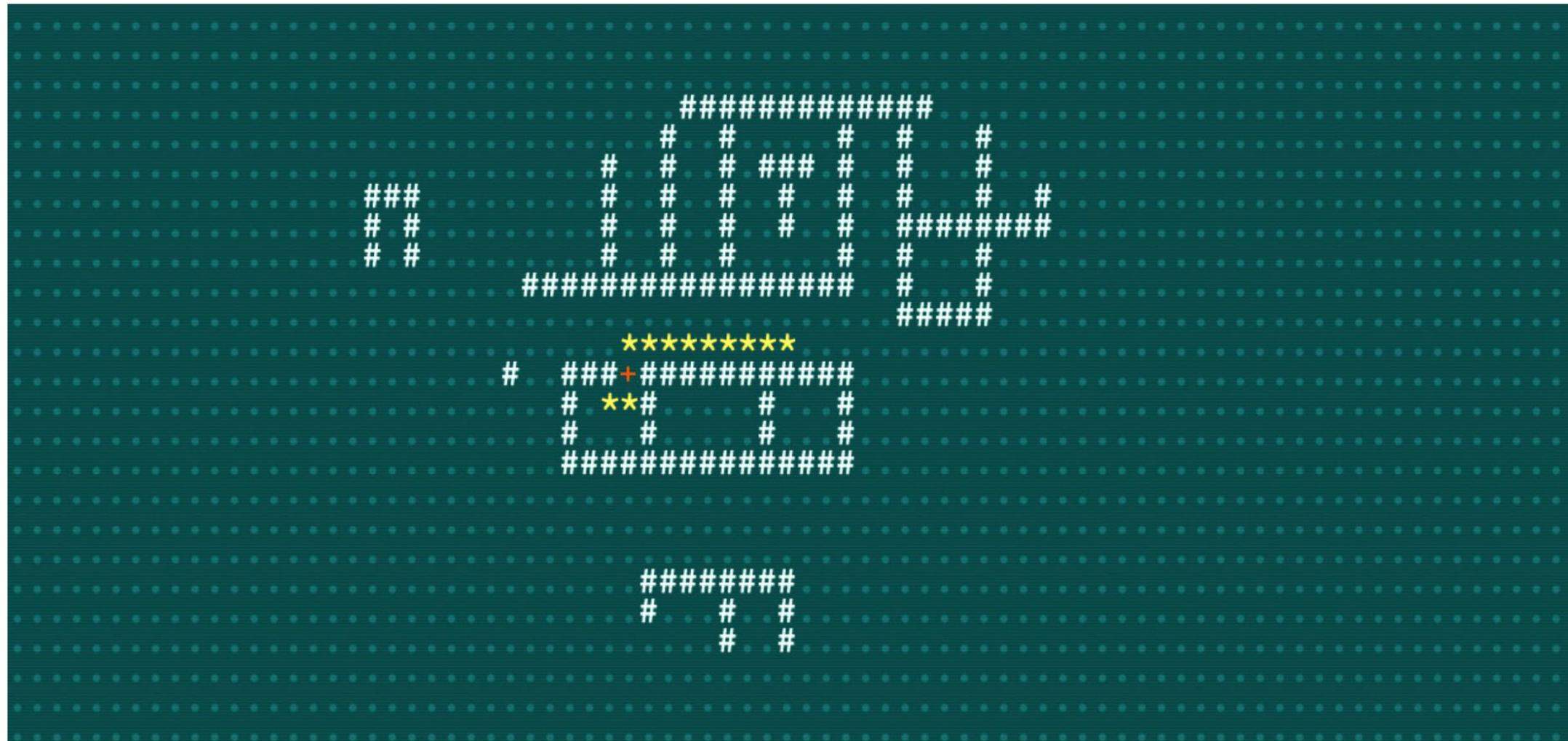
Full Example – WFC Output



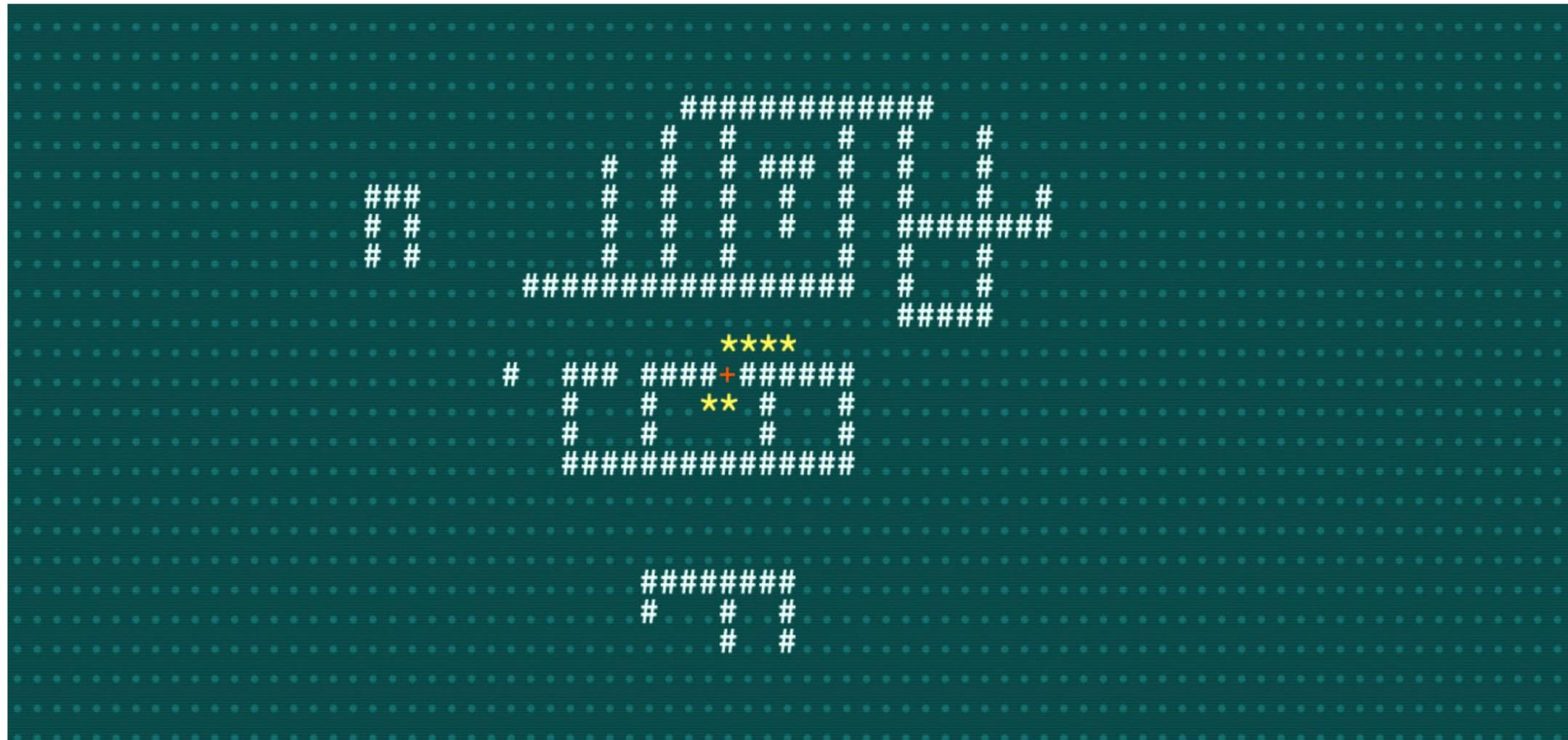
Full Example – Fill Segments



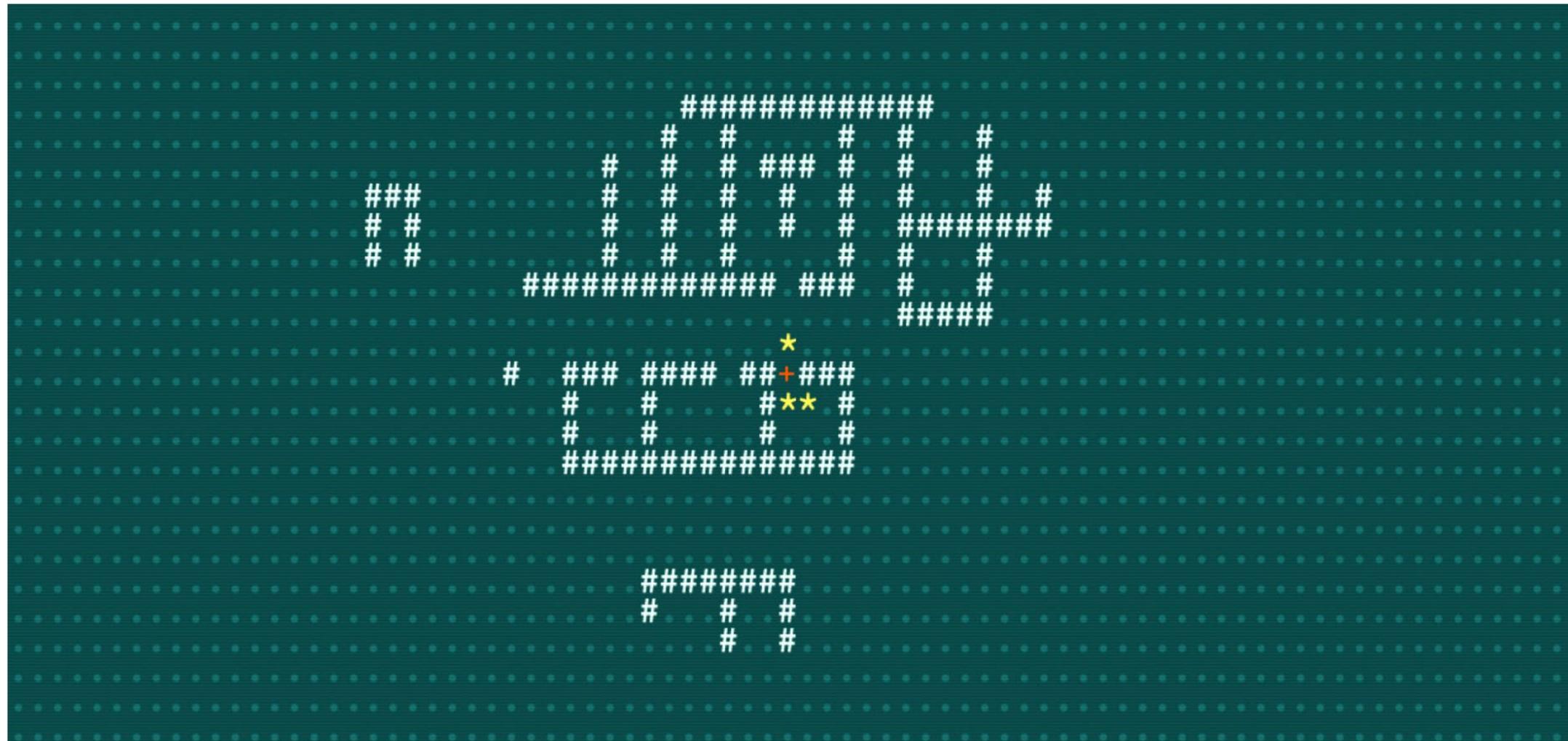
Full Example - Connectivity



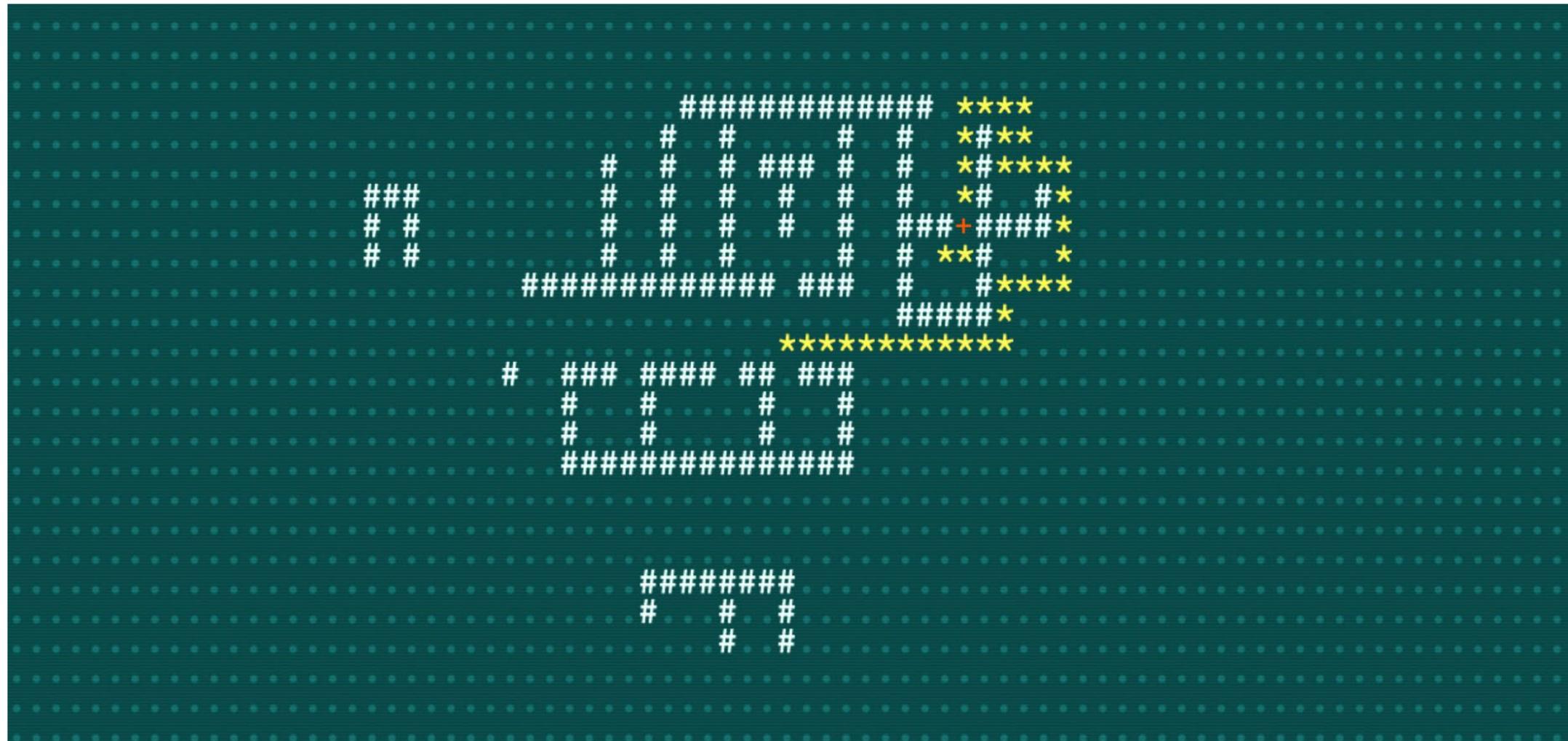
Full Example - Connectivity



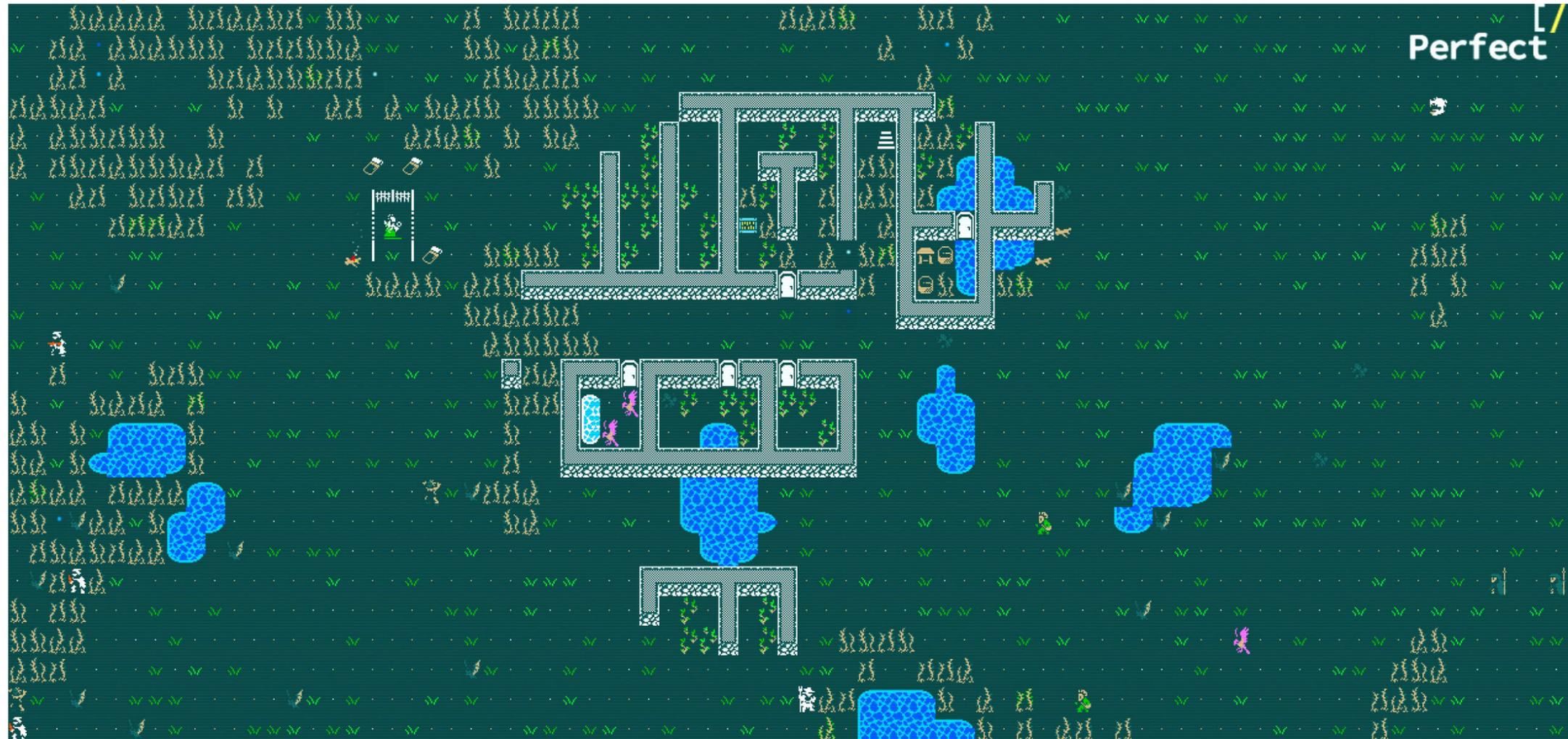
Full Example - Connectivity

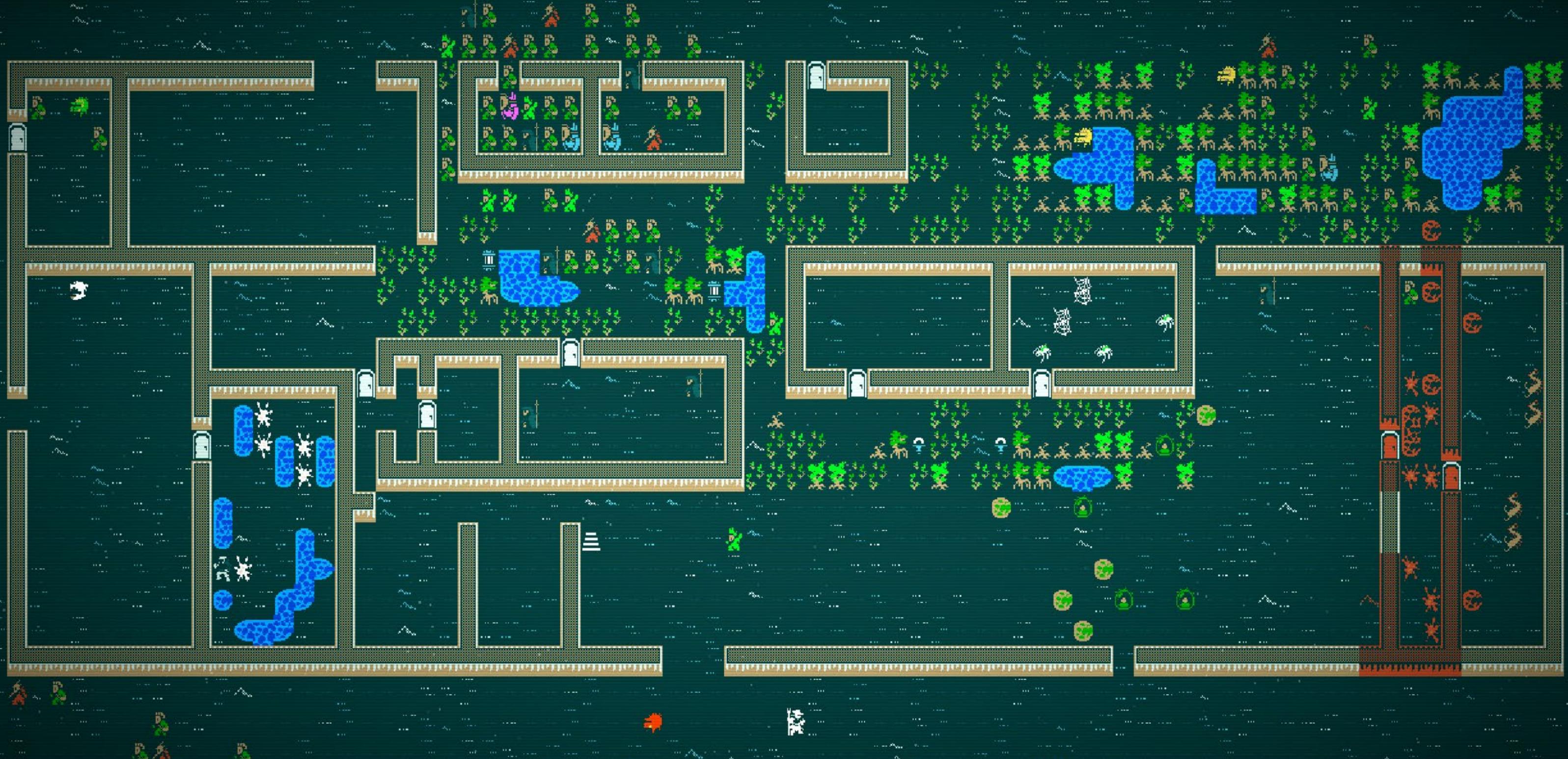


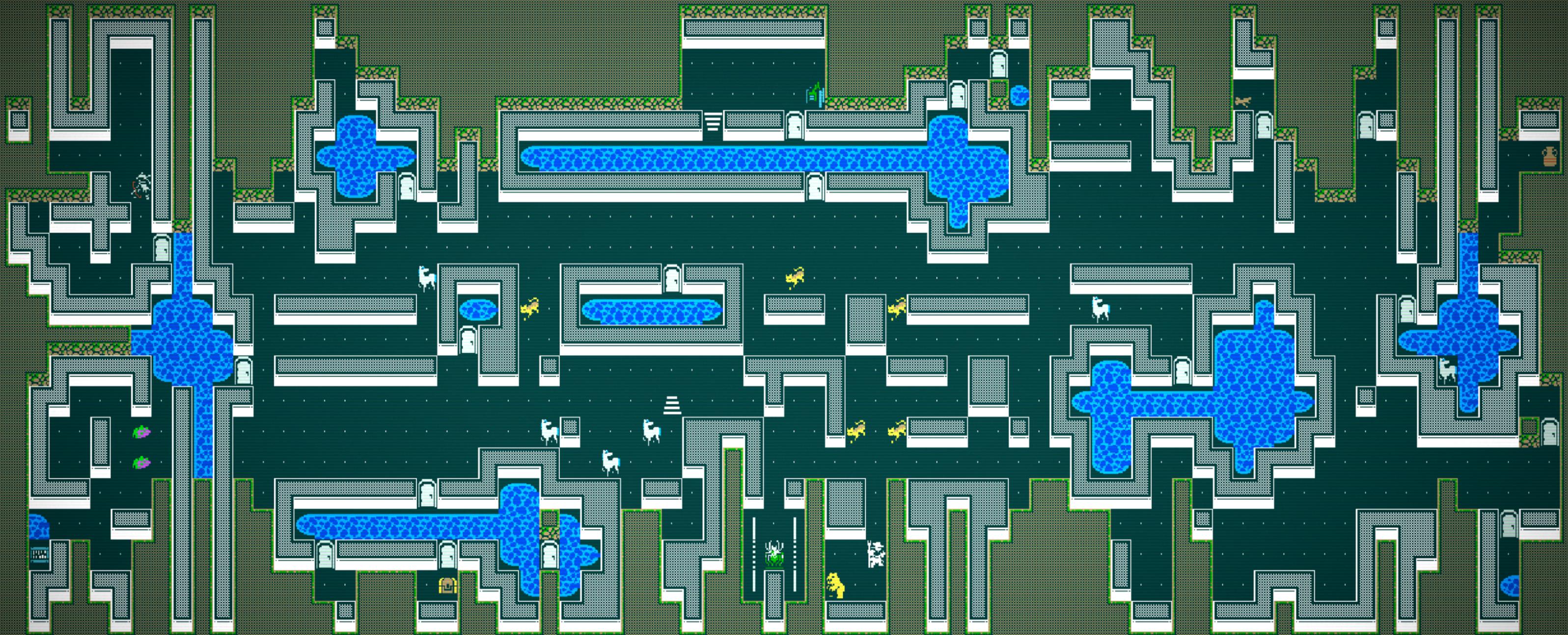
Full Example - Connectivity

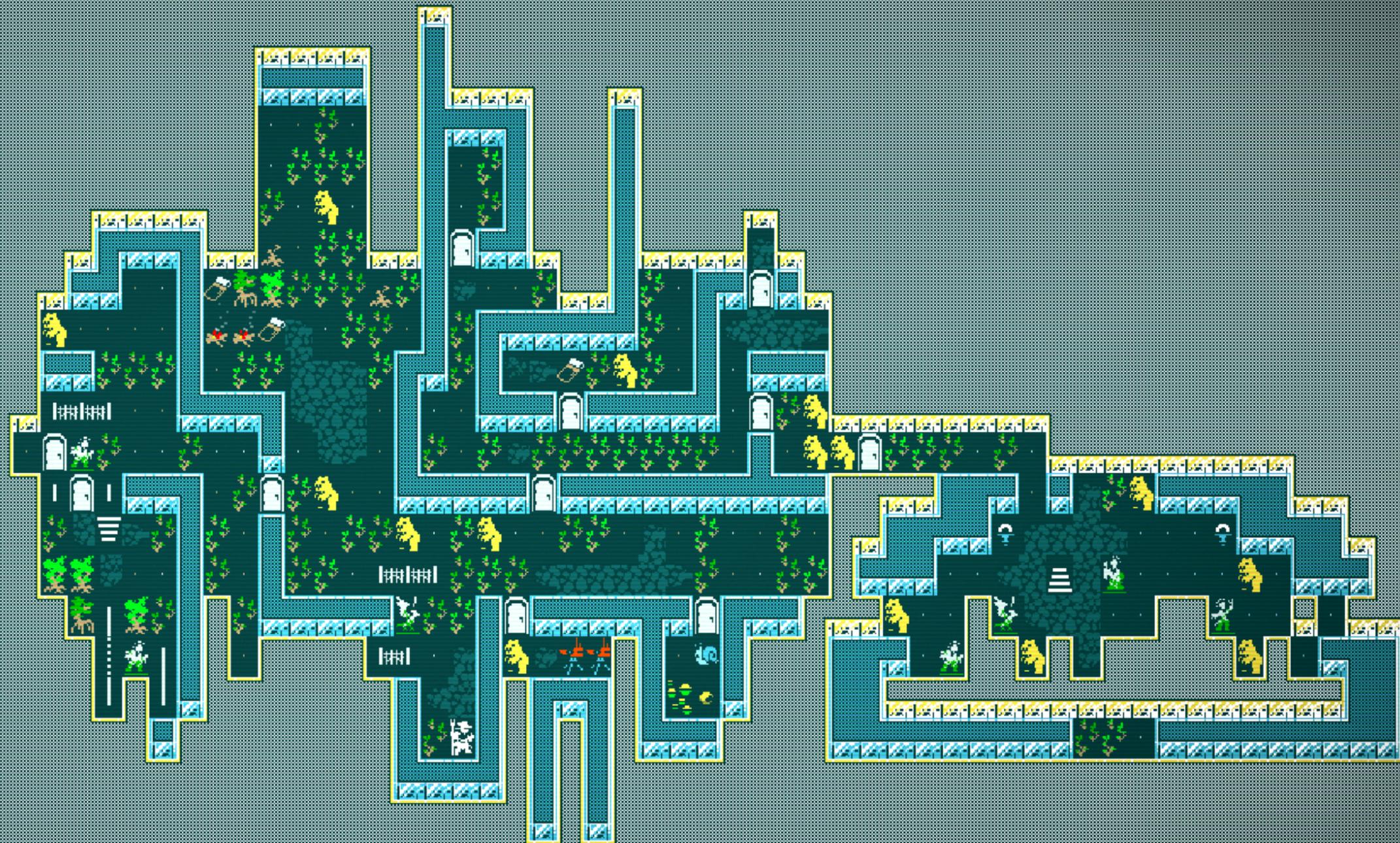


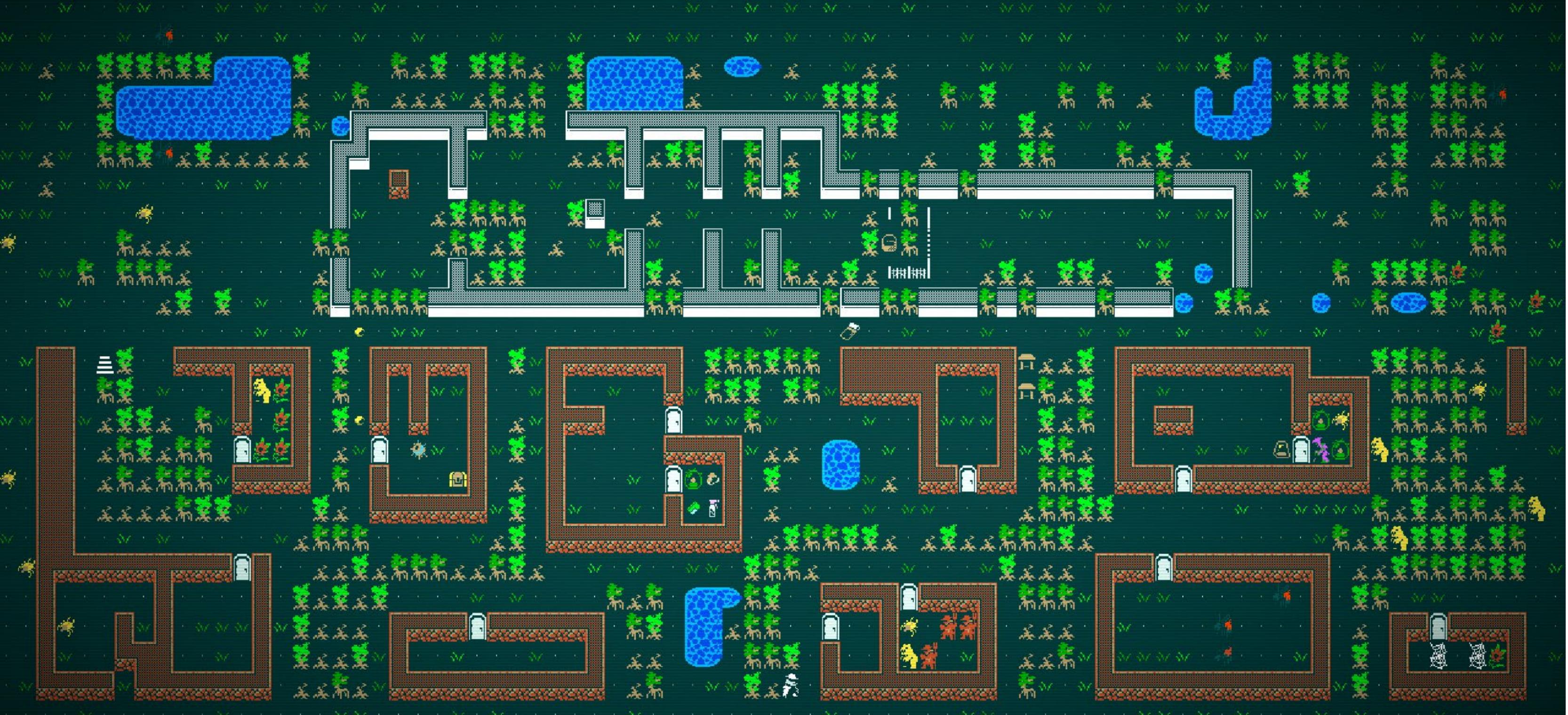
Full Example – Finished Map









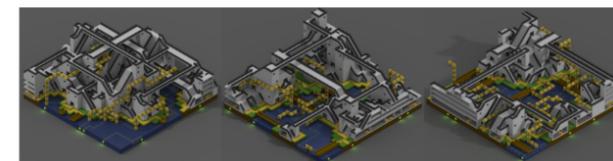


Additional Reading & Questions

- “A Brief Introduction to Wave Function Collapse”
https://youtu.be/pcZQILKxo_M?t=445
- “WFC is constraint Solving in the Wild”
https://isaackarth.com/papers/wfc_is_constraint_solving_in_the_wild/
- WFC Repository
<https://github.com/mxgmn/WaveFunctionCollapse>

Higher dimensions

WFC algorithm in higher dimensions works completely the same way as in dimension 2, though performance becomes an issue. These voxel models were generated with $N=2$ overlapping tiled model using $5 \times 5 \times 5$ and $5 \times 5 \times 2$ blocks and additional heuristics (height, density, curvature, ...).



Higher resolution screenshots: 1, 2, 3.

Voxel models generated with WFC and other algorithms will be in a separate repo.

Constrained synthesis

WFC algorithm supports constraints. Therefore, it can be easily combined with other generative algorithms or with manual creation.

Here is WFC autocompleting a level started by a human:

[GIF](#) | [GIFV](#)

ConvChain algorithm satisfies the strong version of the condition (C2): the limit distribution of $N \times N$ patterns in the outputs it is producing is exactly the same as the distributions of patterns in the input. However, ConvChain doesn't satisfy (C1): it often produces patterns that are not possible in the input. ConvChain first uses a well-crafted heuristic and then a WFC to



Local similarity means that

- (C1) Each $N \times N$ pattern of pixels in the output should occur at least once in the input.
- (Weak C2) Distribution of $N \times N$ patterns in the input should be similar to the distribution of $N \times N$ patterns over a sufficiently large number of outputs. In other words, probability to meet a particular pattern in the output should be close to the density of such patterns in the input.

In the examples typical value of N is 3.



WFC initializes output bitmap in a completely unobserved state, where each pixel value is in superposition of colors of the input bitmap (so if the input was black & white then the unobserved states are shown in different shades of grey). The coefficients in these superpositions are real numbers, not complex numbers, so it doesn't do the actual quantum mechanics, but it was inspired by QM. Then the program goes into the observation-propagation cycle:

- On each observation step an $N \times N$ region is chosen among the unobserved which has the lowest Shannon entropy. This region's state then collapses into a definite state according to its coefficients and the distribution of $N \times N$ patterns in the input.
- On each propagation step new information gained from the collapse on the previous step propagates through the output.

On each step the overall entropy decreases and in the end we have a completely observed state, the wave function has collapsed.

It may happen that during propagation all the coefficients for a certain pixel become zero. That means that the algorithm has run into a contradiction and can not continue. The problem of determining whether a certain bitmap allows other nontrivial