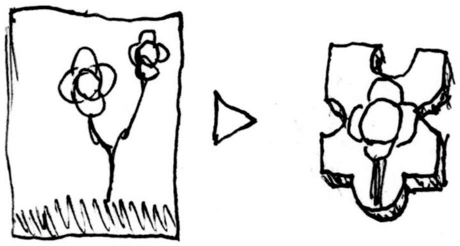
WaveFunctionCollapse *and You*

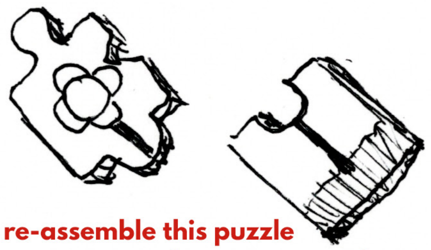
By Isaac Karth
@proc_gen

Today there is a visual interface for teaching the computer how to imitate your pictures. It was invented in 2016 by Maxim Gumin. It's called WaveFunctionCollapse.

Reminds me of *Life, A User's Manual*



We take an image and chop it up into puzzle pieces.



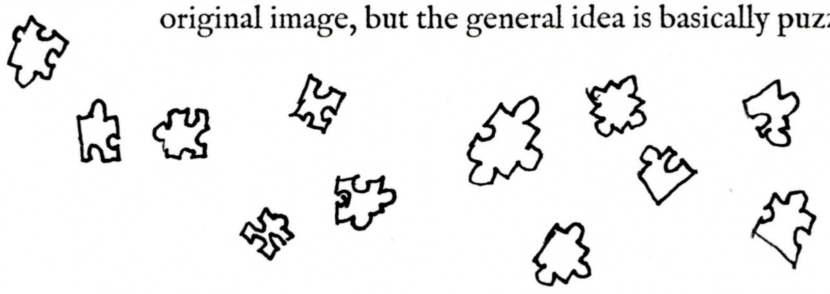
We're going to re-assemble this puzzle to make a new image using the same pieces!

Some puzzles piece make sense to connect to each other, so we give them the same edge shape. (In the computer, we make a list of which ones fit together, which I call an *adjacency*, or we make a big chart that we can use to see which pieces fit, as a matrix.)

WaveFunctionCollapse has been used for image generation, map generation, making 3D levels in video games, making looping animations, and composing poetry. People have implemented it in many different programming languages.

That's All?

Pretty much. I glossed over the implementation details for making it run faster and with probabilities that match the original image, but the general idea is basically puzzle pieces.



I also glossed over the step of deciding which puzzle pieces fit together. You can do it by hand, but the fun ways involve giving it an image and having the algorithm figure how they fit together. **The "overlapping" algorithm uses patterns instead of tiles, but it's just a clever way to use machine learning to specify adjacencies.**

Watch Out!

Sometimes you'll remove too many pieces and the remaining pieces don't fit. This doesn't happen very often unless you have an impossible puzzle, so usually we just start over from the beginning.

Some people have implemented backtracking to help with this. There are many different variations on WFC, with different features!

WFC is based on ideas from texture synthesis and 3D model synthesis, reapplied to image synthesis. I think there's a lot more uses of it still waiting to be discovered!

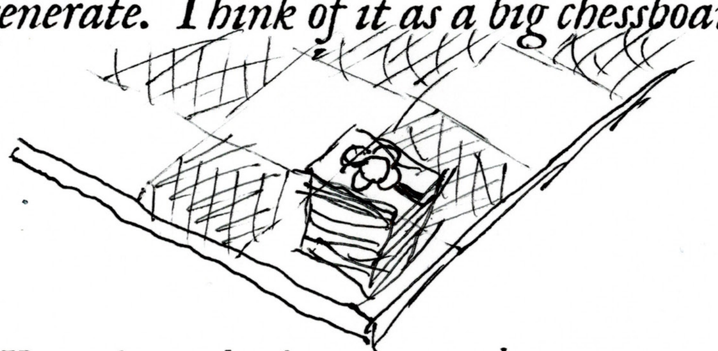
Want to know more?

Read the Code! And @ExUtumno's design notes!
https://github.com/mxgmn/WaveFunctionCollapse

Read my paper about WFC!
https://adamsmith.as/papers/wfc_is_constraint_solving_in_the_wild.pdf

The Fell Types are digitally reproduced by Igino Marini. www.iginomarini.com

We make a grid for the image we want to generate. Think of it as a big chessboard.



This is called *The Wave Table*
(on the computer it can be a matrix of bits)

We stack puzzle pieces onto each square—one copy of each piece we have. We end up with some big stacks of pieces.

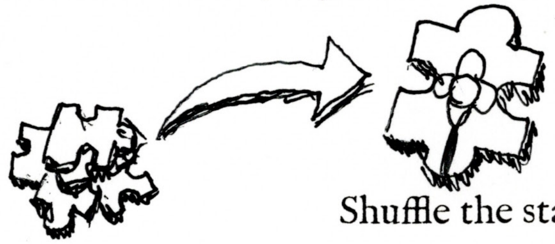
Now We Loop

The goal of this game is to remove pieces from the board until only pieces that fit together are left on the board.

Constraint Solving Terminology!
Nodes = the spaces where we put puzzle pieces
Domains = the stack of puzzle pieces allowed in the node
Edges = the relationship between neighboring spaces

Observation

Pick a stack of pieces. It should be short and next to a stack that has the fewest possible pieces. There will be ties, so just tiebreak at random.



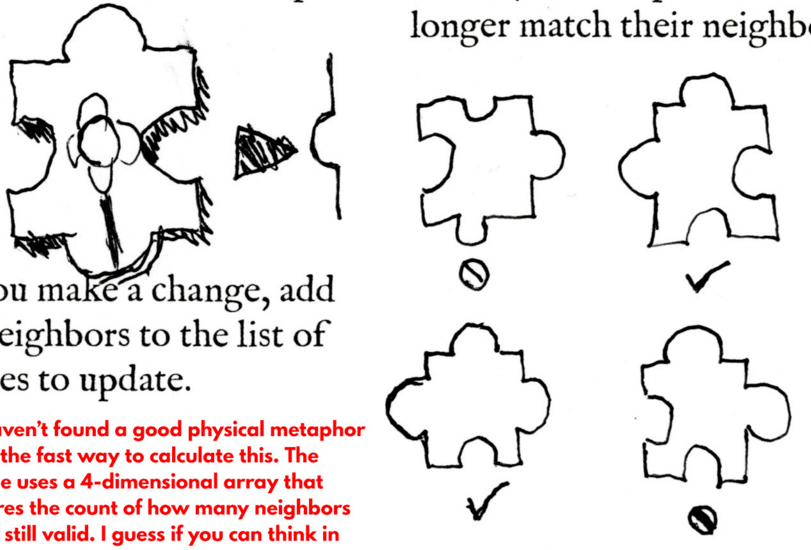
Technically, weighted random sampling is even better here, weighted by the number of pieces in the original image. But that makes the physical metaphor way more complicated.

Shuffle the stack and pick one piece.
Toss away the rest.
The piece we picked is now the only one in this space.
Write down which spaces are next to this one.

This is tedious to do by hand!
We can automate it with computers.
There are dozens of implementations here:
https://github.com/mxgmn/WaveFunctionCollapse

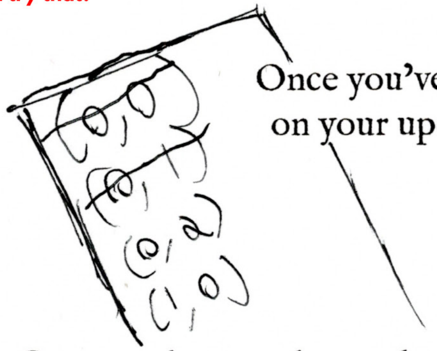
Propagation

Got your list of spaces? This is the update list.
For each space in the list, remove pieces that no longer match their neighbors.



If you make a change, add its neighbors to the list of spaces to update.

I haven't found a good physical metaphor for the fast way to calculate this. The code uses a 4-dimensional array that stores the count of how many neighbors are still valid. I guess if you can think in 4D you can try that?



Once you've crossed off everything on your update list, go back to the Observation step.

Once each space has only one puzzle piece left, you're done!
Snap together your finished puzzle.
You've made a new picture!

This isn't the only way to solve the puzzle. You can use other constraint solvers if you want. We've used ASP + Clingo and various MiniZinc solvers.