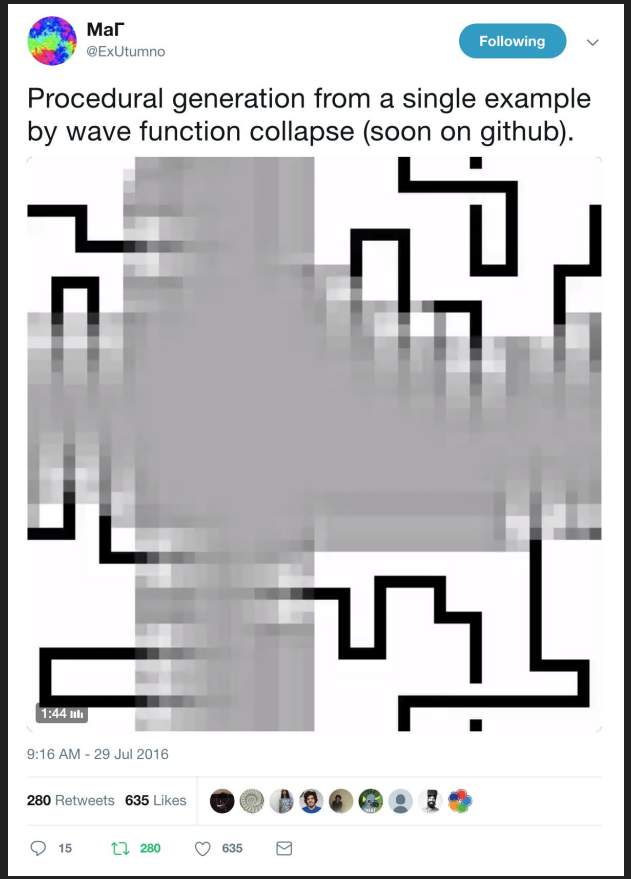
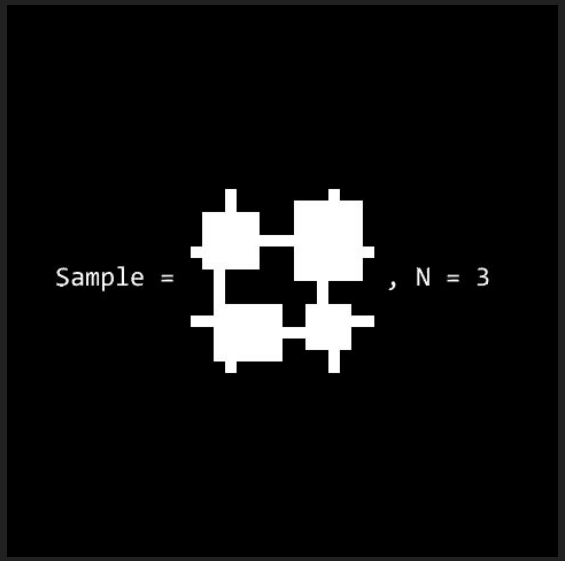
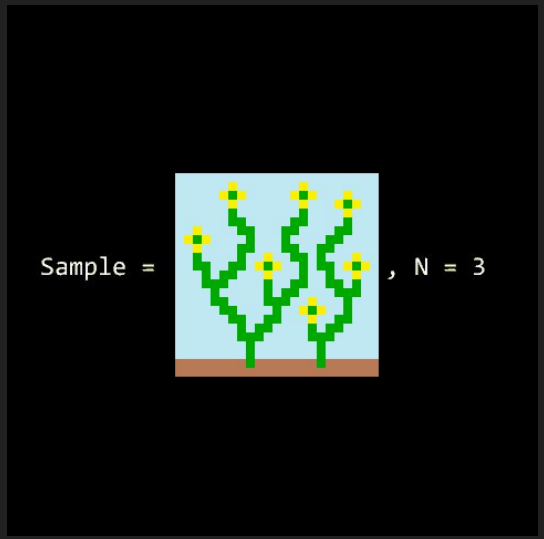
An Intro To WaveFunctionCollapse

Isaac Karth* and Adam M. Smith
Design Reasoning Laboratory
Dept. of Computational Media
UC Santa Cruz

* ikarth@ucsc.edu







https://www.youtube.com/watch?v=DOQTr2Xmlz0

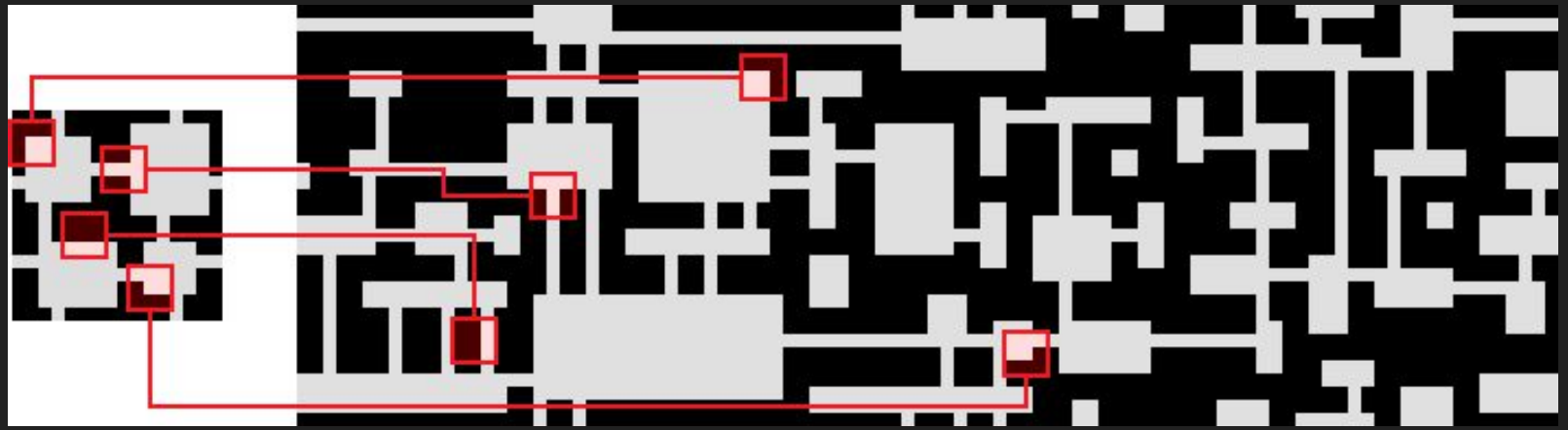
WFC and the Problem it Solves

Input:

- Low-resolution image ***or*** set of tiles and allowed connectivity
- Target output size

Output:

- New image of target size where every local pattern taken from somewhere in the input (hard constraint)



Typical usage stories

- I downloaded the original code from github. I needed it in some language other than C#, so I rewrote it line-for-line in my language without understanding it. It works in my game now!

- The C# code looked scary (I didn’t really read it), but the animation was awesome. I decided to write my own algorithm that does what I think the original code does. It works in my game now!

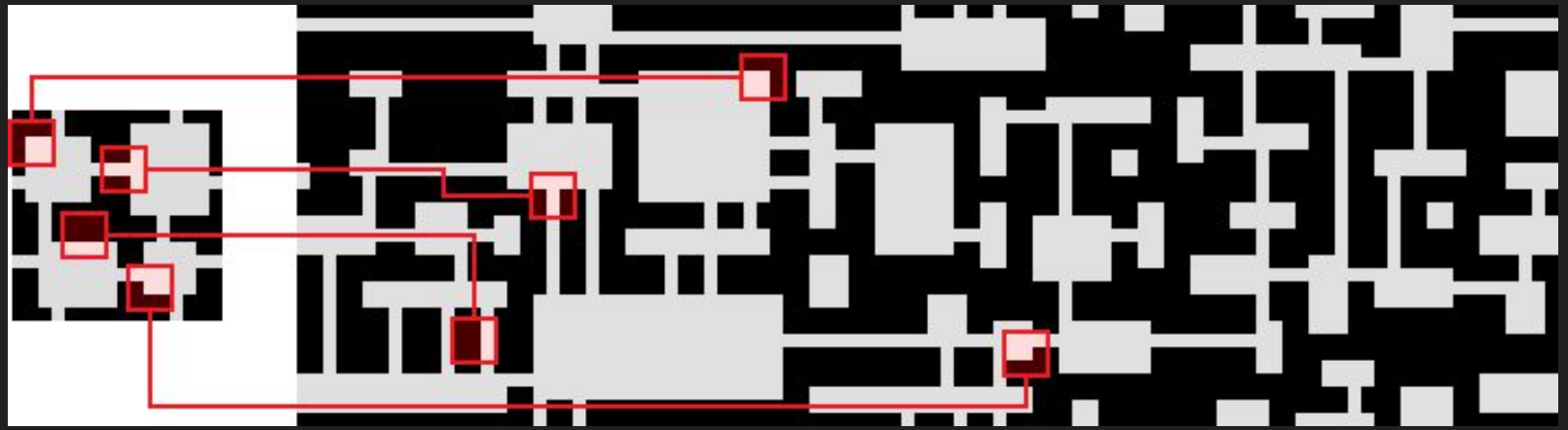
WFC and the Problem it Solves

Input:

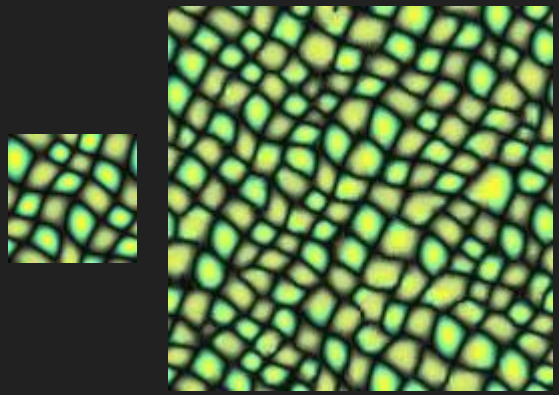
- Low-resolution image ***or*** set of tiles and allowed connectivity
- Target output size

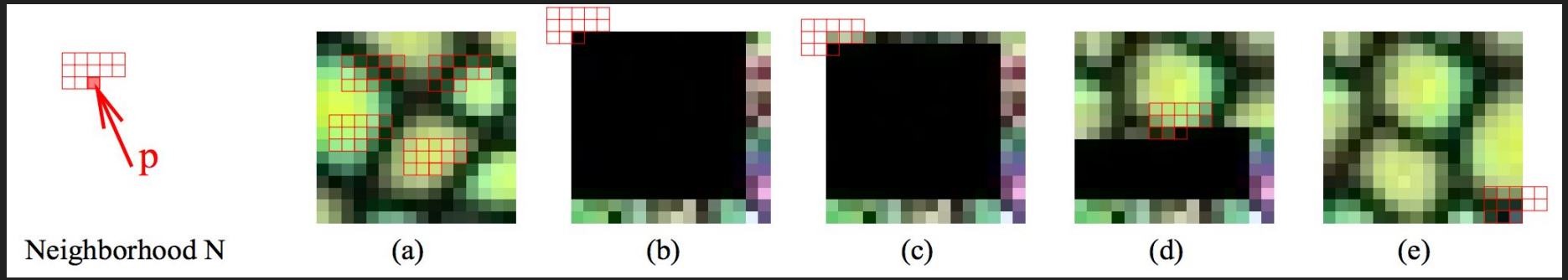
Output:

- New image of target size where every local pattern taken from somewhere in the input (hard constraint)



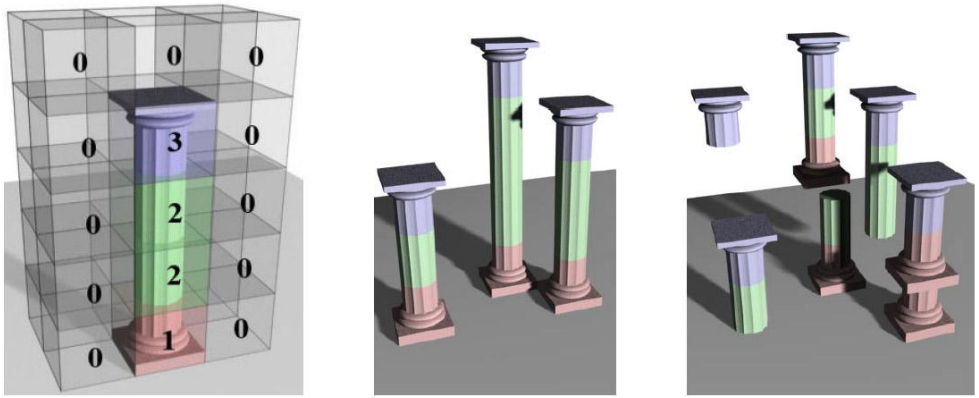
Graphical Texture Synthesis





http://graphics.stanford.edu/papers/liyiwei_thesis/ (2001)

3D Model Synthesis



(a) Model divided into *model pieces*

(b) A Seamless Model

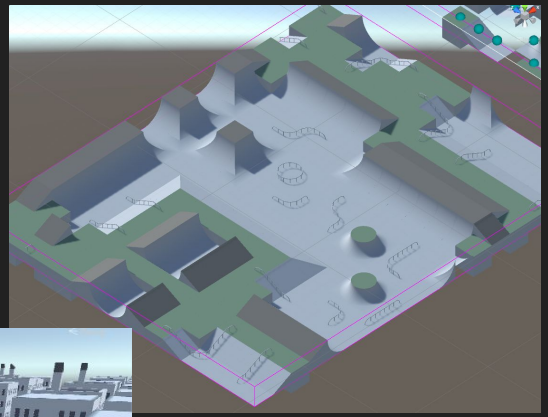
(c) A Model with Many Conflicts

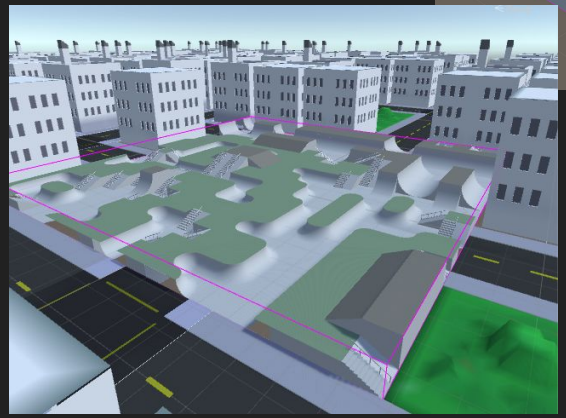
Figure 1.4: In discrete model synthesis, the user divides the model into model pieces (a). The goal is to generate a new model whose pieces fit seamlessly (b). If the pieces do not fit together, the model (c) does not resemble the input.

Paul C. Merrell, Dissertation, 2009
http://graphics.stanford.edu/~pmerrell/thesis.pdf

Interesting non-image domains

ProcSkater
at
ProcJam 2016



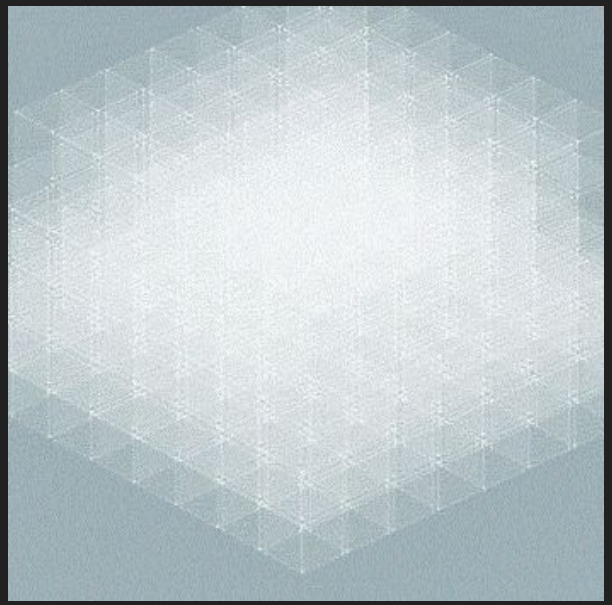


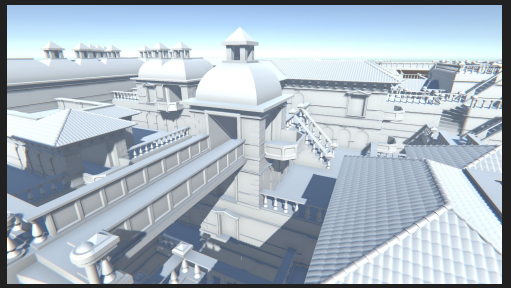
by Joseph Parker,
Ryan Jones,
and Oscar Morante

```
*** *** *** *** *** *** *** *** *** ***
*** *** *** *** *** *** *** *** *** ***
*** *** *** *** *** *** *** *** *** ***
*** *** *** *** *** *** *** *** *** ***
*** *** *** *** *** *** *** *** *** ***
*** *** *** *** *** *** *** *** *** ***
*** *** *** *** *** *** *** *** *** ***
*** *** *** *** *** *** *** *** *** ***
*** *** *** *** *** *** *** *** *** ***
*** *** *** *** *** *** *** *** *** ***
*** *** *** *** *** *** *** *** *** ***
*** *** *** *** *** *** *** They sweep down
The city. Then we build seats on the town.
```

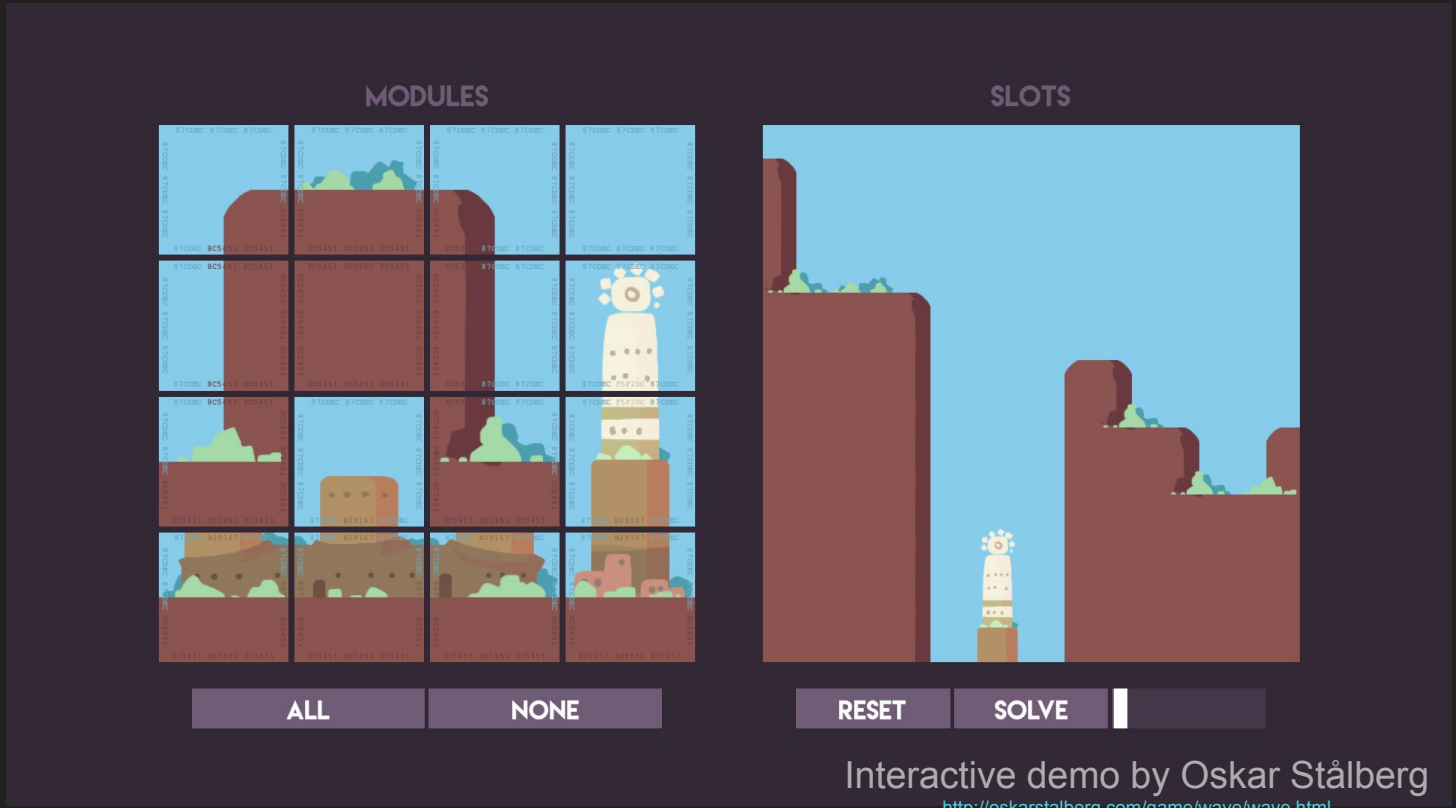
```
*** *** *** *** *** *** *** *** *** ***
*** *** *** *** *** *** *** *** *** ***
*** *** *** *** *** *** *** *** *** ***
*** *** *** *** *** *** *** *** *** ***
*** *** *** *** *** *** *** *** *** ***
*** *** *** *** *** *** *** *** *** ***
*** *** *** *** *** *** *** *** *** ***
*** *** *** *** *** *** *** *** *** ***
*** *** *** *** *** *** *** *** *** ***
*** *** *** *** *** *** *** *** *** ***
*** *** *** *** *** *** *** *** *** ***
*** *** *** *** I must be kind to them
Thought Alice soon began to cry again.
```

Oisín poetry generator by Martin O'Leary,
a glaciologist





Technical description of observe and propagate cycle



Interactive demo by Oskar Stålberg

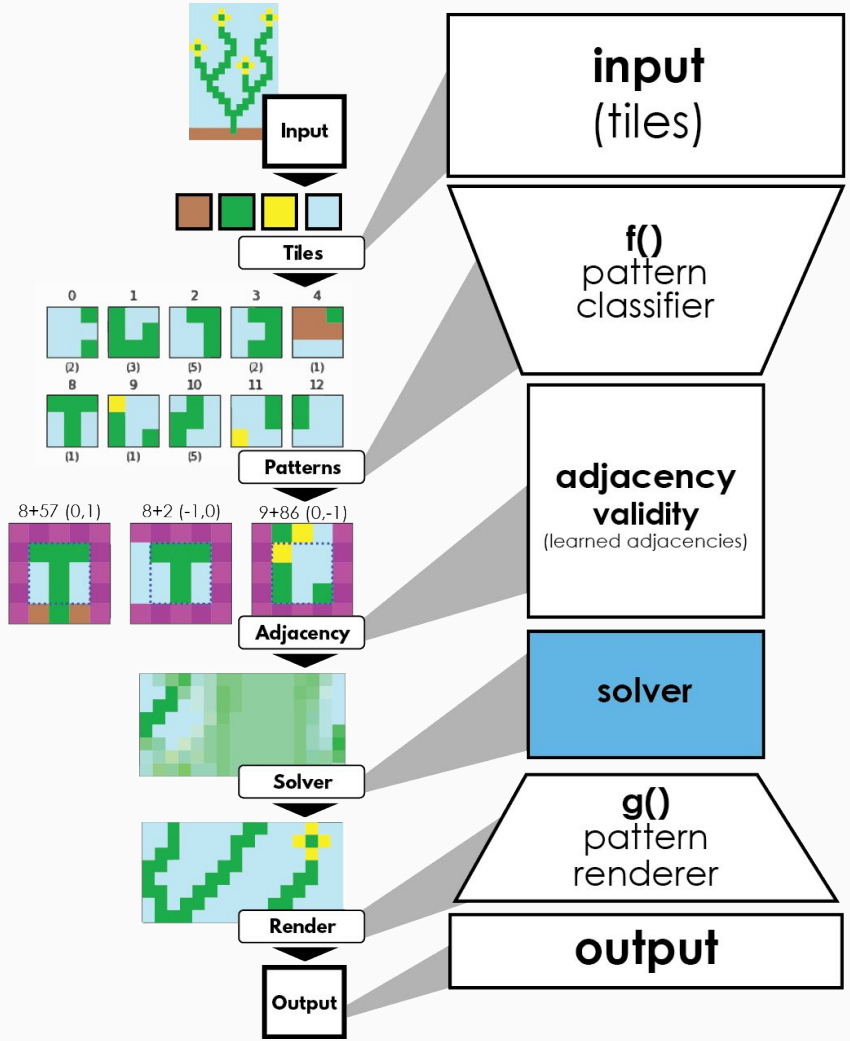
http://oskarstalberg.com/game/wave/wave.html

WaveFunctionCollapse has two major subcomponents:

Learning and Solving

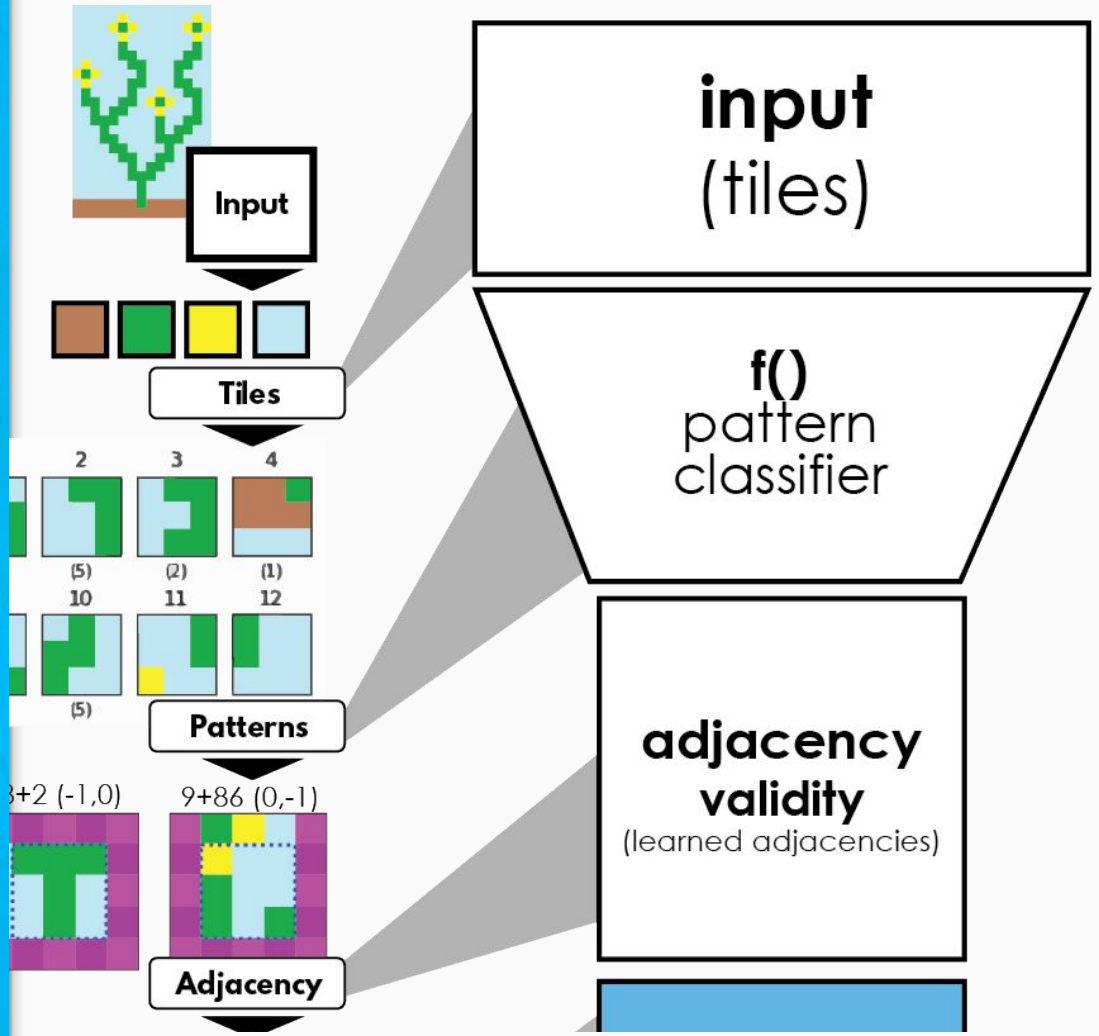
Generative Pipeline

When we visualize the architecture pipeline the role of the classifier becomes apparent.



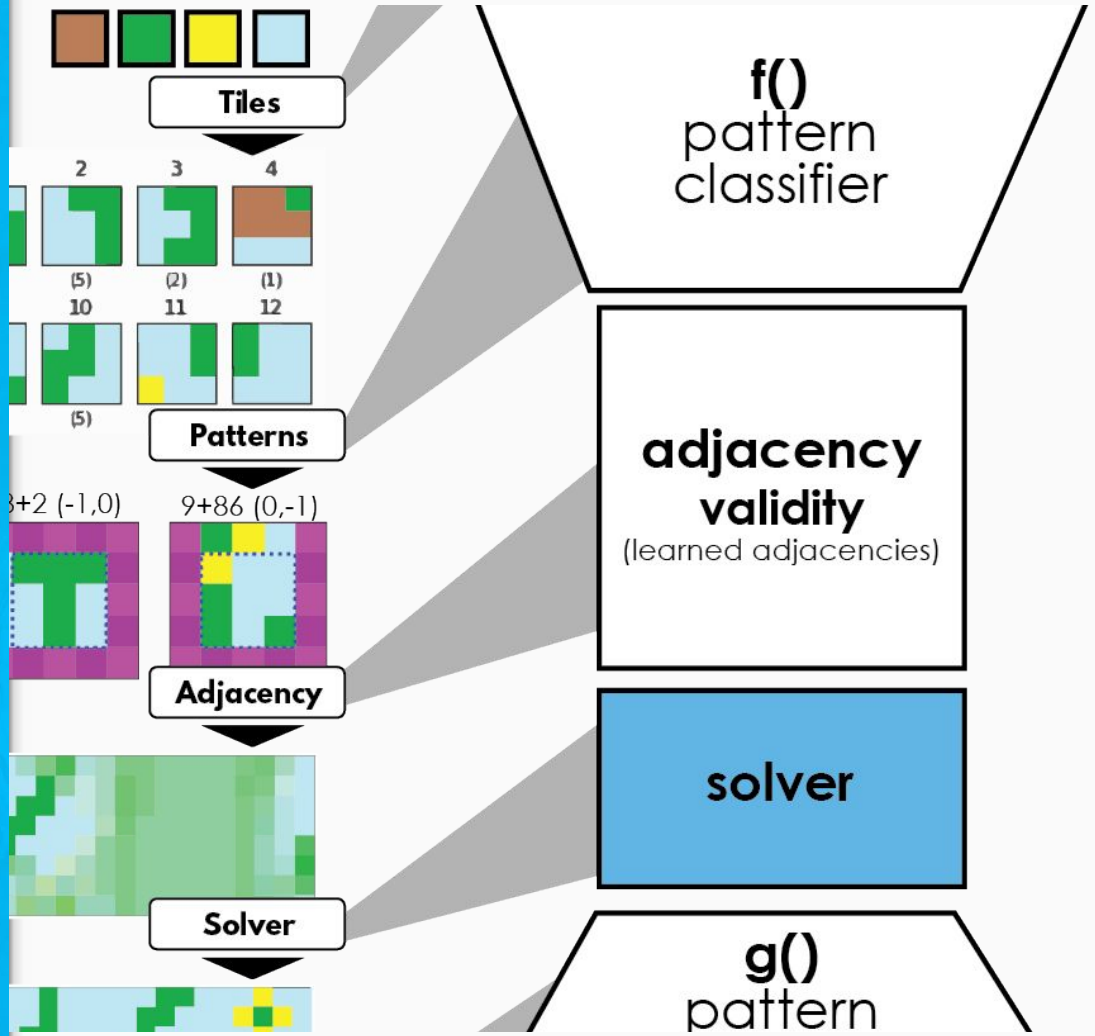
Pattern Classifier

A powerful part of the WFC overlap model's learning is that it operates on **patterns** rather than directly on pixels.



Adjacency Validity

Rather than specifying the possible adjacencies by hand, WFC's overlap model learns them from the example image.



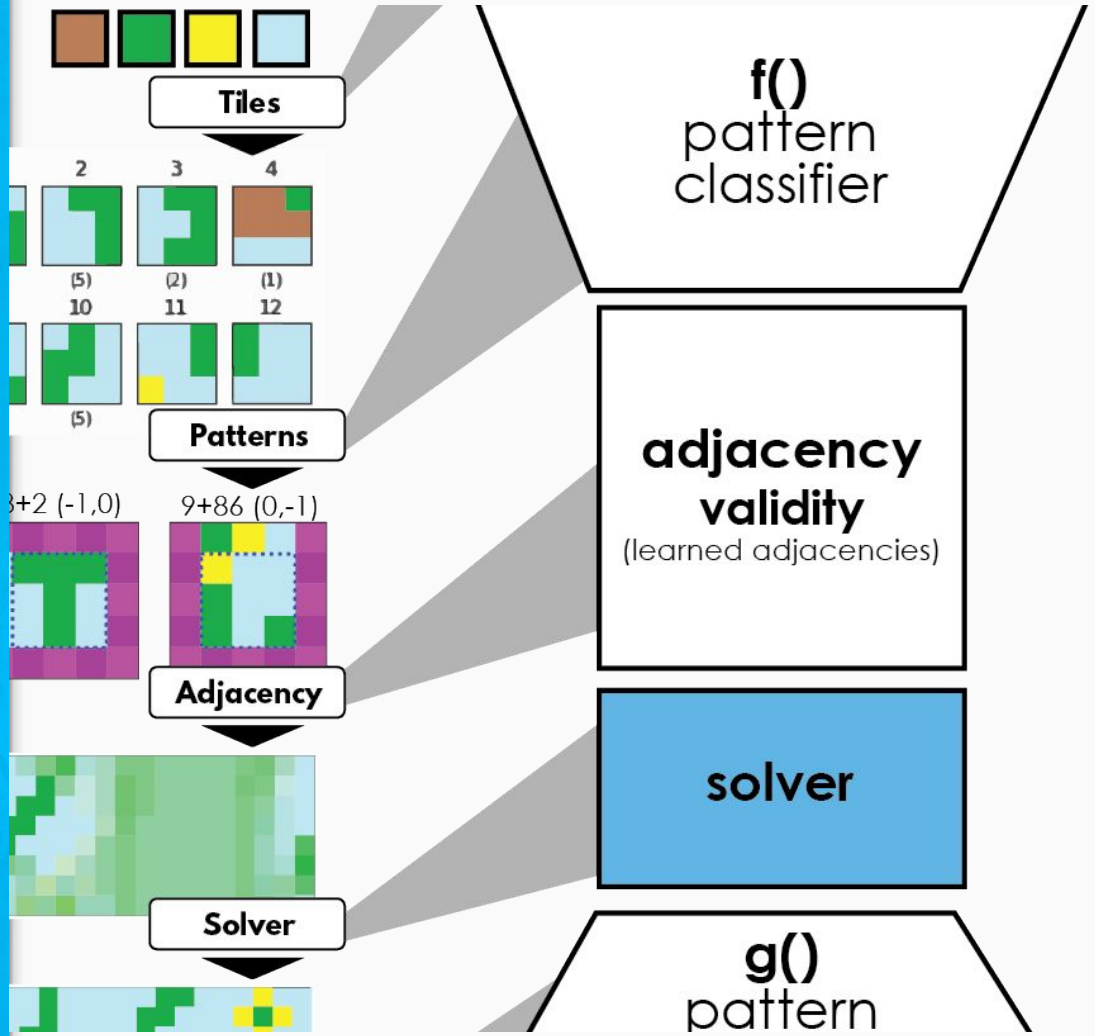
Training the classifier

Patterns that match when placed in partial overlap can be adjacent in the generated solution.

Solver

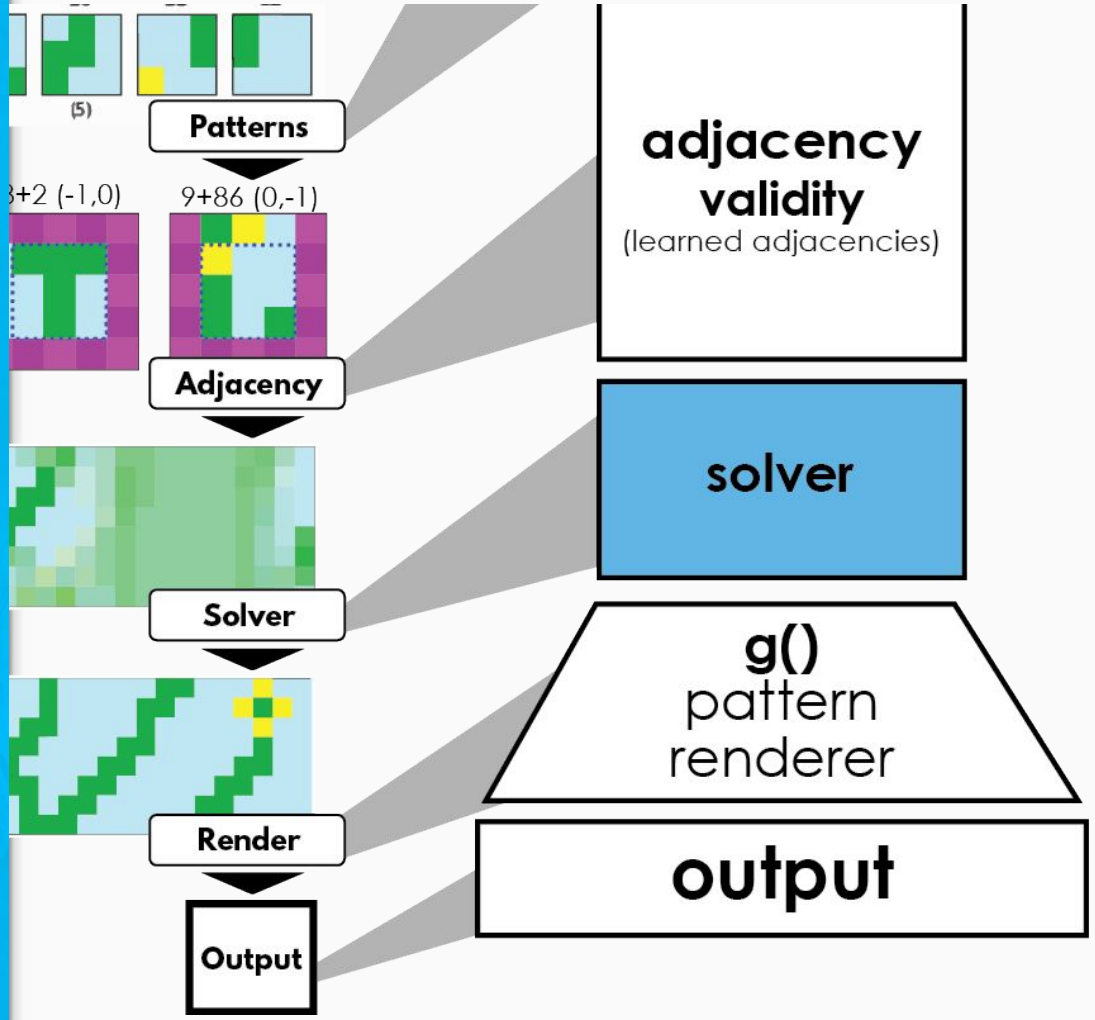
WFC's use of a custom constraint solver was one of the first uses of constraint-based procedural generation to see widespread adoption in the wild.

It's design is streamlined compared to solvers intended for general use cases: the original implementation doesn't use backtracking or complicated heuristics.



Pattern Renderer

Once a solution has been generated, it needs to be translated from a grid of patterns to a grid of tiles.



Pseudocode

```
defn Run():
    GetPatternsFromSample()
    BuildIndex()
    Loop until finished:
        Observe()
        Propagate()
    OutputObservations()
```

```
defn FindLowestEntropy(coefficient_matrix):
    Return the cell that has the lowest greater-than-zero
           entropy, defined as:
        A cell with one valid pattern has 0 entropy
        A cell with no valid patterns is a contradiction
        Else: the entropy is based on the sum of the frequency
              that the patterns appear in the source data, plus
              use some random noise to break ties and
              near-ties.


defn Propagate(coefficient_matrix):
    Loop over the cells to be updated:
        For each neighboring cell:
            For each pattern that is still potentially valid:
                Compare this location in the pattern with the cell's values
                    If this point in the pattern no longer matches:
                        Set the array in the wave to false for this pattern
                        Flag this cell as needing to be updated in
                            the next iteration


defn OutputObservations(coefficient_matrix):
    For each cell:
        Set observed value to the average of the color value
            of this cell in the pattern for the remaining
            valid patterns
        Return the observed values as an output image
```

Actual Code

```
def propagate(wave, adj, periodic=False, onPropagate=None):
 last_count = wave.sum()
 while True:
   supports = {}
   if periodic:
     padded = numpy.pad(wave,((0,0),(1,1),(1,1)), mode='wrap')
   else:
     padded = numpy.pad(wave,((0,0),(1,1),(1,1)), mode='constant',constant_values=True)
   for d in adj:
     dx,dy = d
     shifted = padded[:,1+dx:1+wave.shape[1]+dx,1+dy:1+wave.shape[2]+dy]
     supports[d] = (adj[d] @ shifted.reshape(shifted.shape[0], -1)).reshape(shifted.shape) > 0
   for d in adj:
     wave *= supports[d]
   if wave.sum() == last_count:
     break
   else:
     last_count = wave.sum()
 if onPropagate:
   onPropagate(wave)
 if wave.sum() == 0:
   raise Contradiction
```

Actual Code

```
def observe(wave, locationHeuristic, patternHeuristic):
 i,j = locationHeuristic(wave)
 pattern = patternHeuristic(wave[:,i,j])
 return pattern, i, j

def makeEntropyLocationHeuristic(preferences):
 def entropyLocationHeuristic(wave):
   unresolved_cell_mask = (numpy.count_nonzero(wave, axis=0) > 1)
   cell_weights = numpy.where(unresolved_cell_mask, preferences + numpy.count_nonzero(wave, axis=0), numpy.inf)
   row, col = numpy.unravel_index(numpy.argmin(cell_weights), cell_weights.shape)
   return [row, col]
 return entropyLocationHeuristic

def makeWeightedPatternHeuristic(weights):
 num_of_patterns = len(weights)
 def weightedPatternHeuristic(wave):
   # TODO: there's maybe a faster, more controlled way to do this sampling...
   weighted_wave = (weights * wave)
   weighted_wave /= weighted_wave.sum()
   result = numpy.random.choice(num_of_patterns, p=weighted_wave)
   return result
 return weightedPatternHeuristic
```

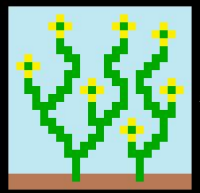
My implementations of WFC

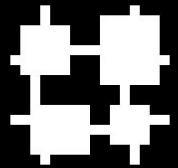
https://github.com/ikarth/wfc_python

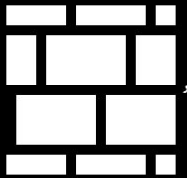
https://github.com/ikarth/wfc_2019f

https://github.com/ikarth/wfc_2019f/blob/master/wfc/wfc_solver.py

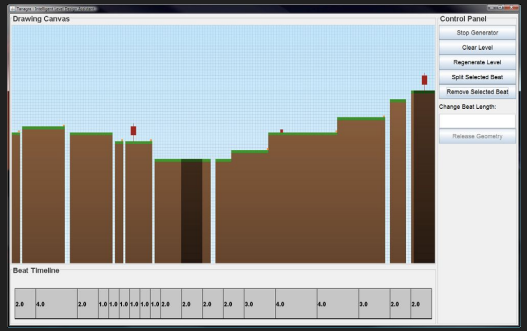
Revisiting the original animations

Sample =  , N = 3

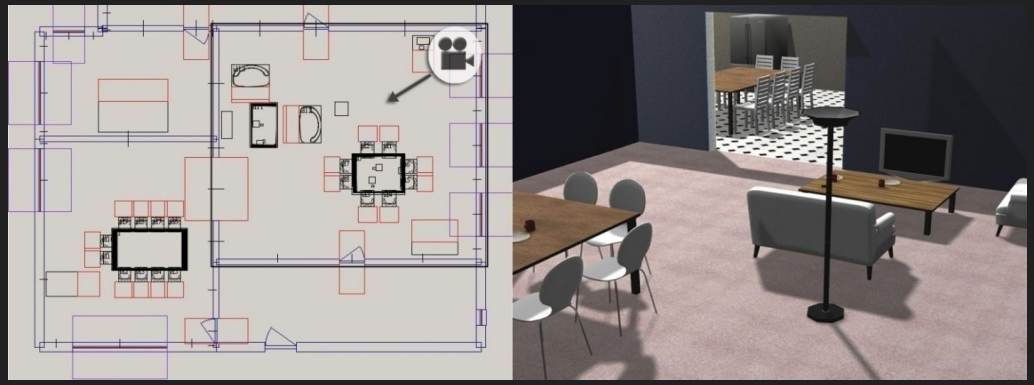
Sample =  , N = 3

Sample =  , N = 3

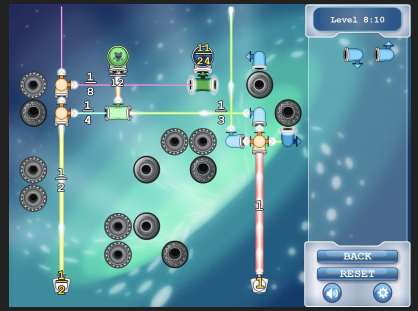
PCG Research stuff that also uses constraint solving



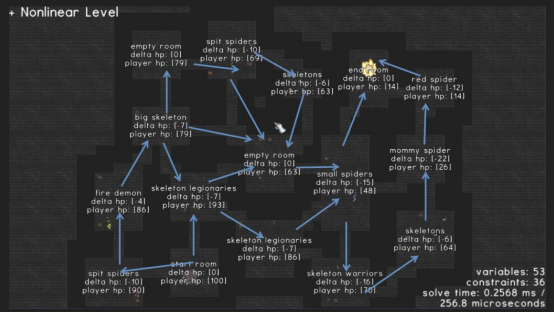
Tanagra using Choco solver library
(G. Smith et al. 2010)



Layout solving for interior design
(Tutenel et al. 2009)

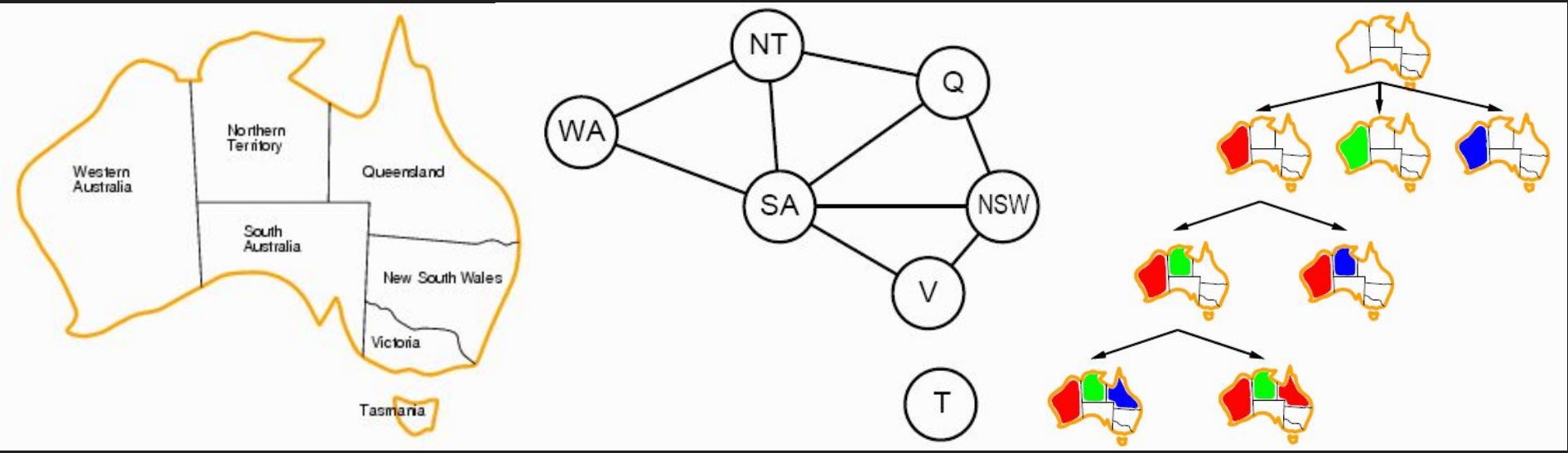


Refraction using answer set programming
(A. Smith et al. 2010)



Fast Procedural Level Population with Playability Constraints
(Horswill and Foged 2012)

Constraint Solving by analogy with Graph Coloring



- **Key terms:** Variables (Nodes), Values (Colors), Domains (Legal colors for a node), Constraints (Edges)
- **Algorithm state:** Partial assignment (Partial coloring), Current domains for unassigned variables
- **Heuristics:** variable selection (where to paint)
  & value decision heuristics (what to paint there)

Analyzing WFC as a CSP algorithm

Variable selection heuristic: “entropy” ~ Minimum-Remaining-Values
+ random tiebreaking

Value decision heuristic: sample from distribution in input (no general CSP equiv.)

Propagation: Arc-Consistency (reduce domains considering each edge independently until no more changes)

Backtracking: No

Restarts: Global on first conflict (when a domain becomes empty).

ASP Surrogate Implementation

Question:

- How might WFC behave if we added backtracking, constraint learning, dynamic heuristics, restart schedules, or other ideas?

Strategy:

- Replace the entire generation phase of WFC with a call to a modern constraint solver. We used Clingo (an answer set solver).

https://potassco.org/clingo/

AnsProlog Formulation

```
1 { assign(X,Y,P):pattern(P) } 1 :- cell(X,Y).

:- adj(X1,Y1,X2,Y2,DX,DY),
   assign(X1,Y1,P1),
   not 1 { assign(X2,Y2,P2):legal(DX,DY,P1,P2) }.
```

Nondeterministically choose exactly one pattern to *assign* for every cell.

Reject a (partial) solution if one cell is *assigned* a pattern and *adjacent* cell (in some direction) is not assigned one of the *legal* patterns for that directional adjacency.

Experiments

Understanding Heuristics

Variable selection heuristics:

- VSIDS* [no conflicts]
- Reading-order [no conflicts]
- Random [timed out]

Implication: The “entropy” heuristic isn’t critical. So long as you make choices near where you’ve already made choice, you’re likely to finish generation without hitting a conflict.

*Variable State Independent Decaying Sum

Understanding Backtracking

Let’s make the problem harder: add a global constraint that each input pattern is used at least once.

- Local backtracking

  [solution after only a few conflicts]

- Global restart at first conflict

  [repeatedly timed out]

Implication: We are going to need local backtracking to repair solutions in the face of global constraints. It’s hard to get lucky with only global restarts.

Variations in the wild:

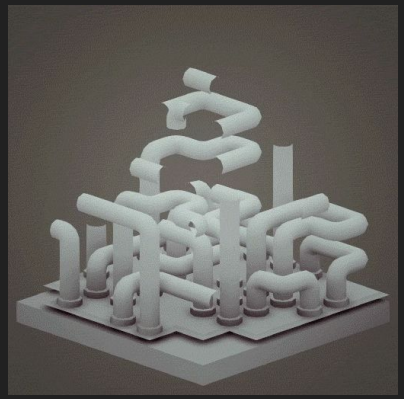
Other axes don’t have to be spatial:

Can use **time** (also makes it easy to find animated loops!)

(https://twitter.com/MattRix/status/872674537799913472)
(https://twitter.com/MattRix/status/872884946918150145)

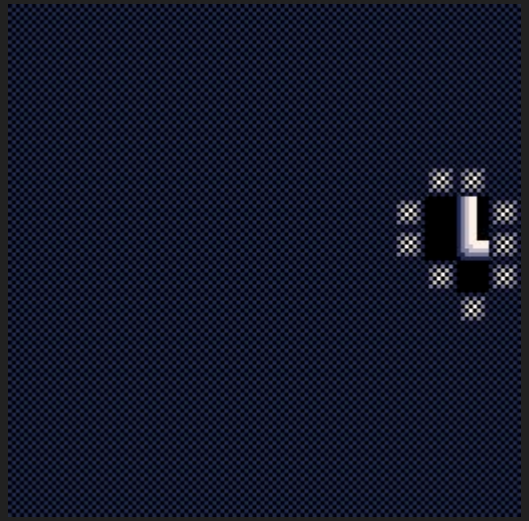
Local backtracking is useful

(https://twitter.com/OskSta/status/793806535898136576)



Variations in the wild:

Propagation not required: Backtracking but no constraint propagation (& on the PICO-8!)

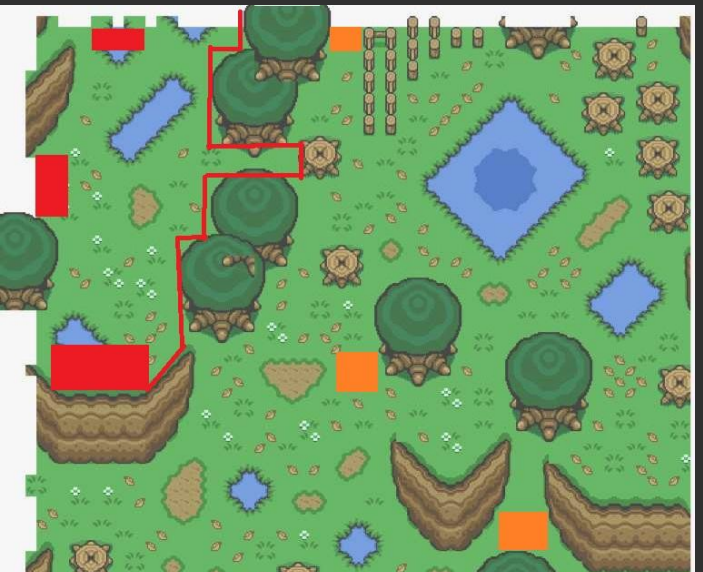


WFC on the PICO-8
By Rémy “TRASEVOL_DOG” Devaux
https://twitter.com/TRASEVOL_DOG/status/876524701027315712
https://trasevol.dog/2017/06/19/week60/

Often discussed but not implemented:
Global constraints, such as connectivity and guaranteed traversal

Follow-up work at UC Santa Cruz

Requiring/forbidding reachability between zones in generated maps: (Jo Mazeika)



Parallelization strategies: (Ruben Fitch)

- Try multiple generation attempts in parallel, then keep the first that finishes.
- Parallelize propagation at the level of columns of the image.

(orthogonal to speedups from better data structures)

Takeaway Messages

- Study **algorithms in the wild** to inspire new applications and ideas!
  - We can use computer science literature and ideas to explain **why** these algorithms do what they do
- Paying attention to **data-driven generators** and what they do for **accessibility**
- Constraint solving is a way of **using search in PCG** that doesn't generate-and-test whole designs at a time

- **WaveFunctionCollapse is constraint solving in the wild.**

WaveFunctionCollapse is Constraint Solving in the Wild

Isaac Karth* and Adam M. Smith
Department of Computational Media
UC Santa Cruz

* ikarth@ucsc.edu

Selected References and Further Reading

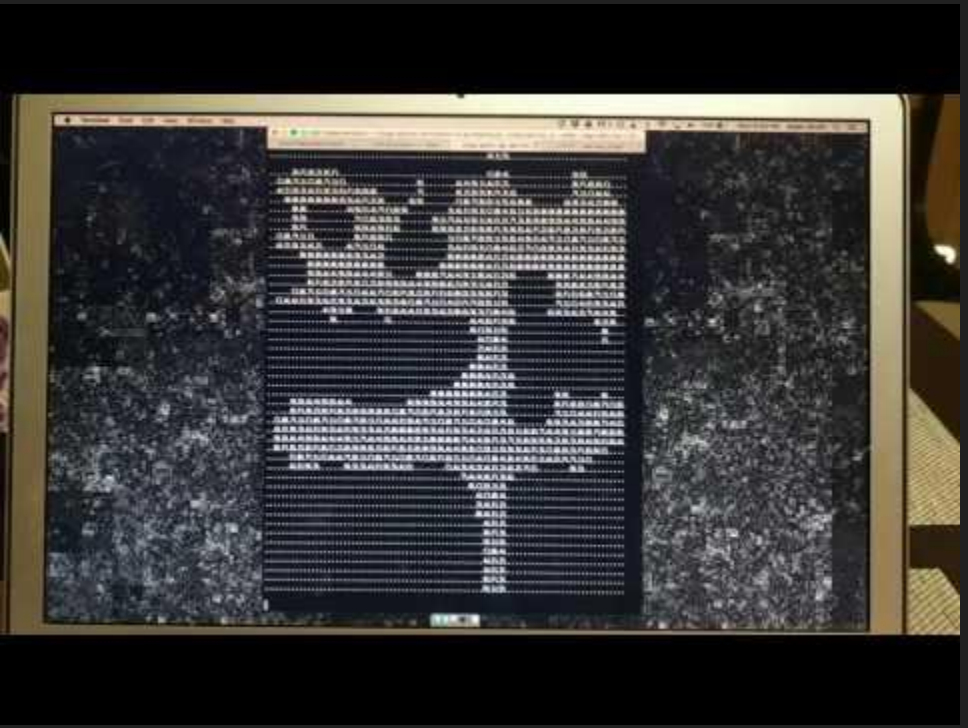
Selected References

- Maxim Gumin. 2016. **WaveFunctionCollapse**. https://github.com/mxgmn/ GitHub repository (2016).
- Adam M. Smith and Michael Mateas. 2011. **Answer Set Programming for Procedural Content Generation: A Design Space Approach.** *IEEE Transactions on Computational Intelligence and AI in Games 3*, 3 (Sept 2011), 187–200. https://doi.org/10.1109/TCIAIG.2011.2158545 https://games.soe.ucsc.edu/sites/default/files/tciaig-asp4pcg.pdf

Solvers

- Clingo: https://potassco.org/
- Choco: http://www.choco-solver.org/
- Minizinc: http://www.minizinc.org/

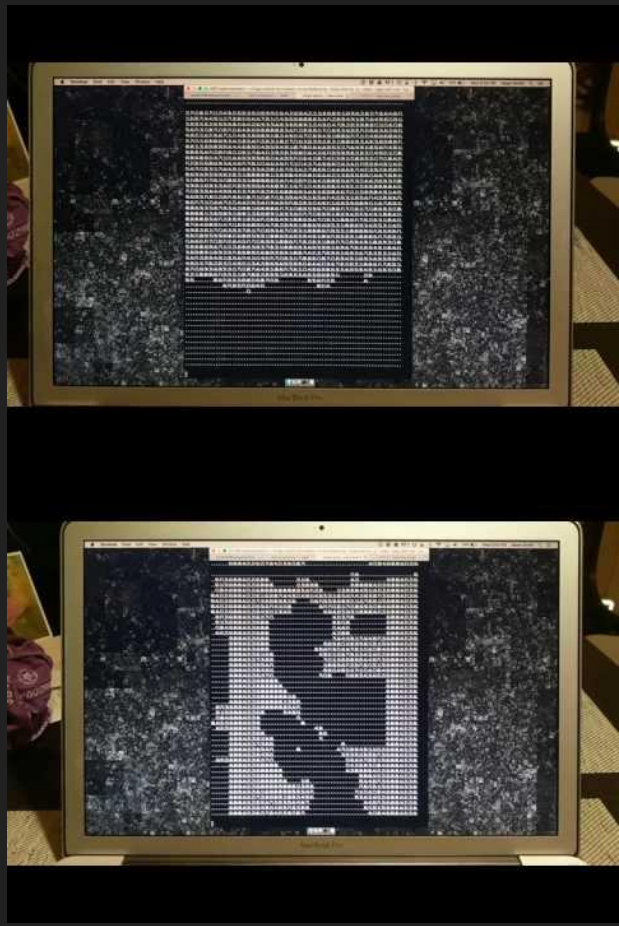
Visualizing how Clingo searches*



Reading-order →

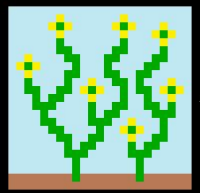
VSIDS ←

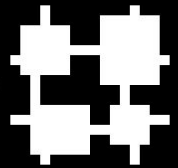
Random →

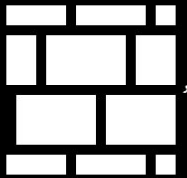


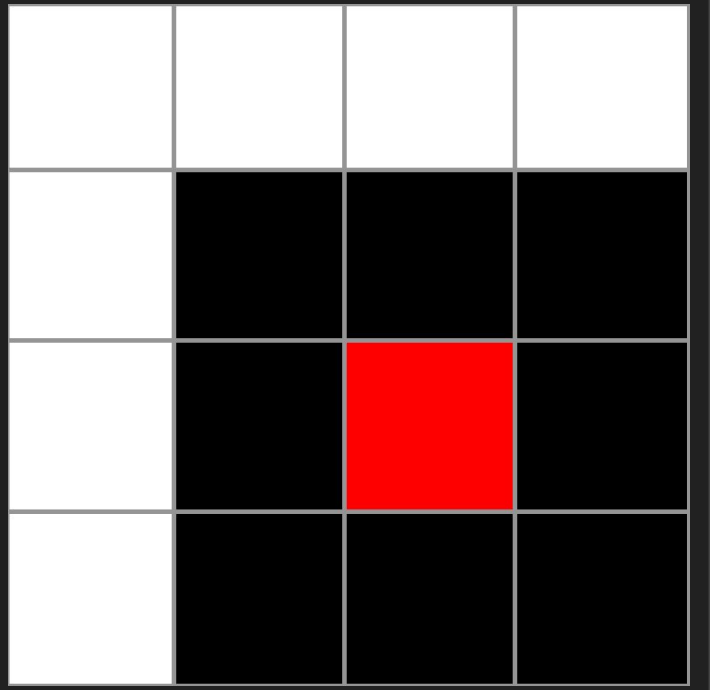
*Active global constraint: every pattern used at least once.

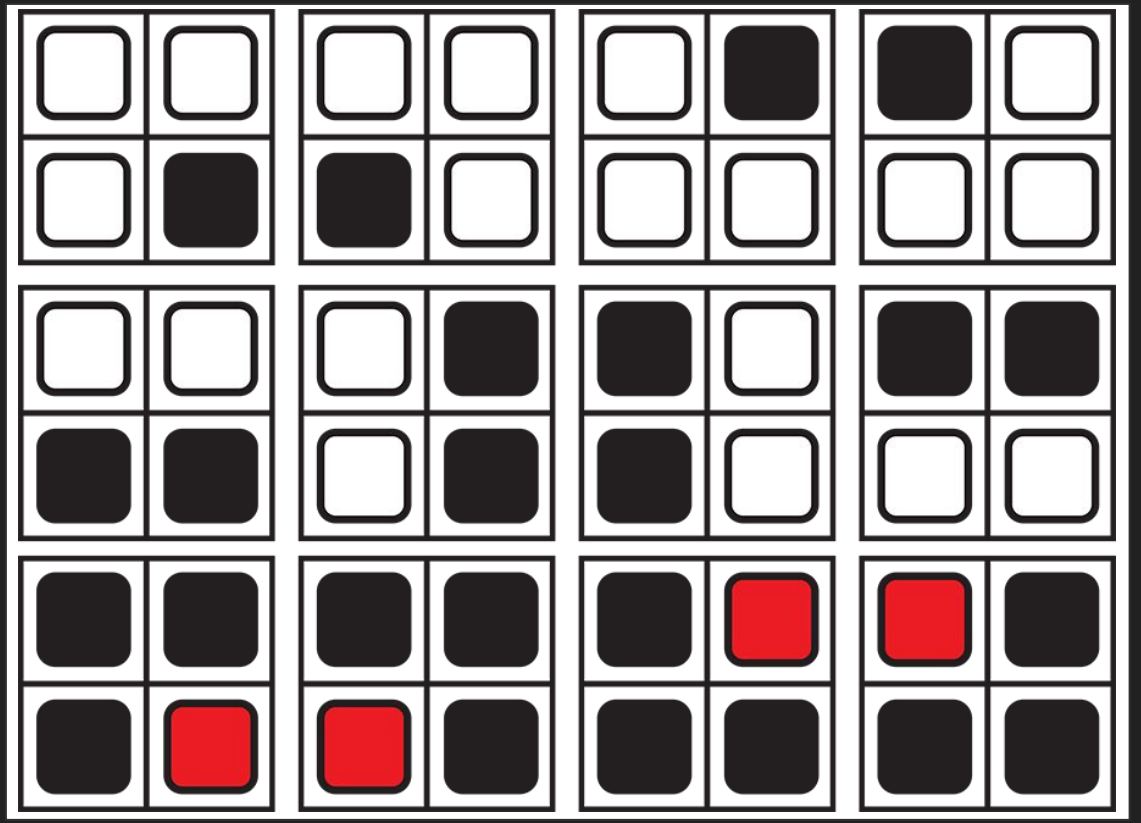
Revisiting the original animations

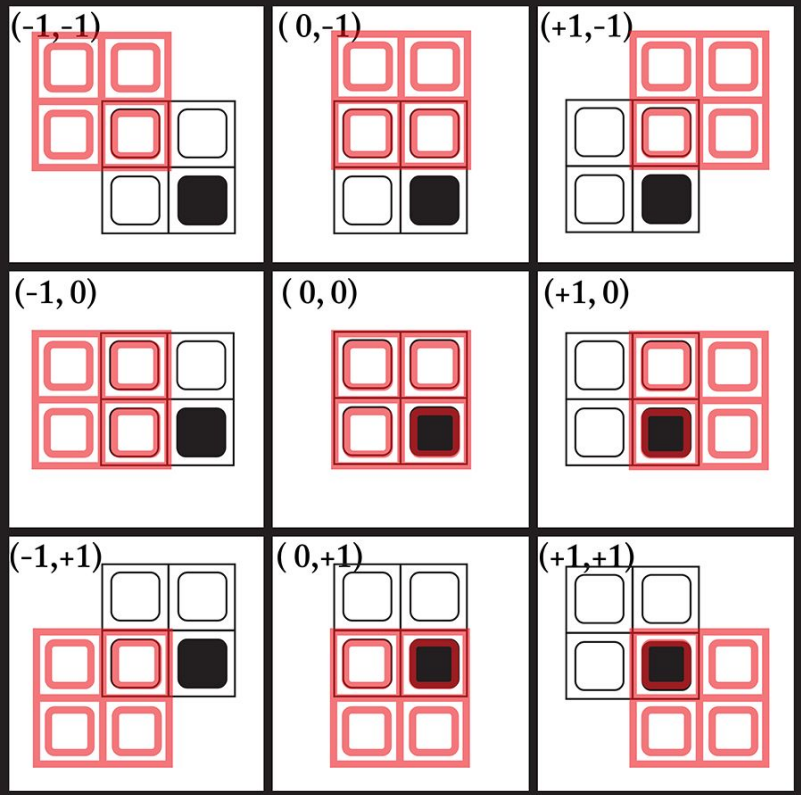
Sample =  , N = 3

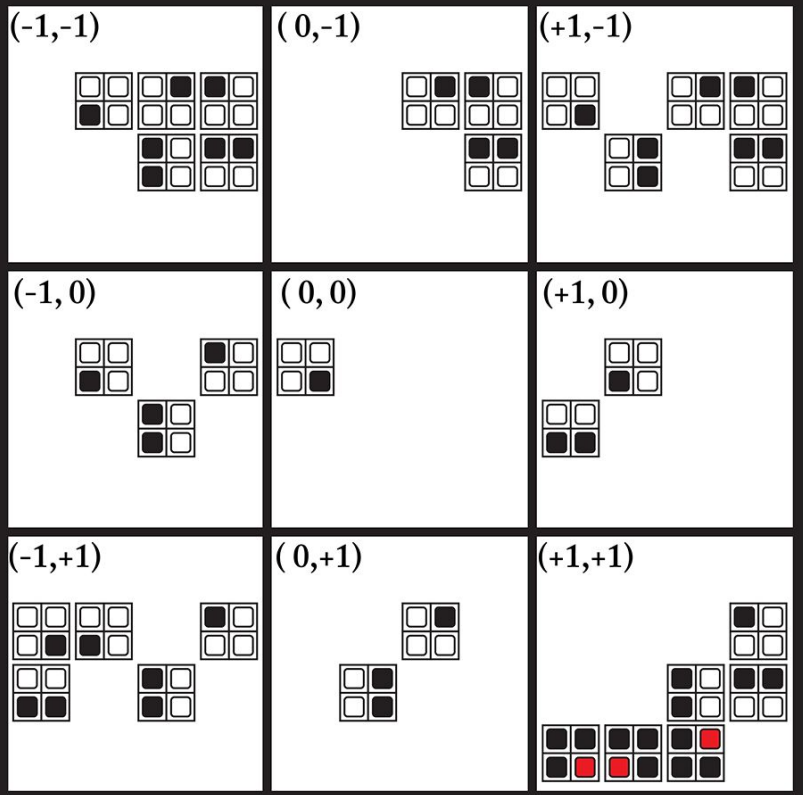
Sample =  , N = 3

Sample =  , N = 3






(-1,-1)
(0,-1)
(+1,-1)
(-1, 0)
(0, 0)
(+1, 0)
(-1,+1)
(0,+1)
(+1,+1)


(-1,-1)
(0,-1)
(+1,-1)
(-1, 0)
(0, 0)
(+1, 0)
(-1,+1)
(0,+1)
(+1,+1)