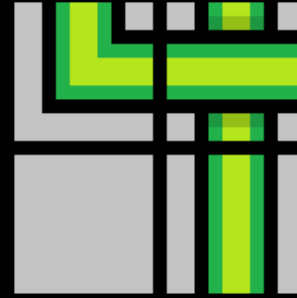


# Procedural Content Generation: Planners and Constraints

2019-11-11



# Model Free & PCG Rules (CAs, Agents, Grammars)

1. What are 3 high-level forms of PCG we discussed so far?
2. We described cellular automaton and agents as \_\_\_\_\_
3. Discuss the relationship of evaluation/fitness functions and (a) player models, and (b) designer preferences
4. L-systems are a form of \_\_\_\_\_, and are particularly useful for \_\_\_\_\_
5. What is the rewriting order of L-systems, and why does this matter?
6. What class of techniques have we seen before that seem particularly suited to quest and story generation?

# Model free Approaches

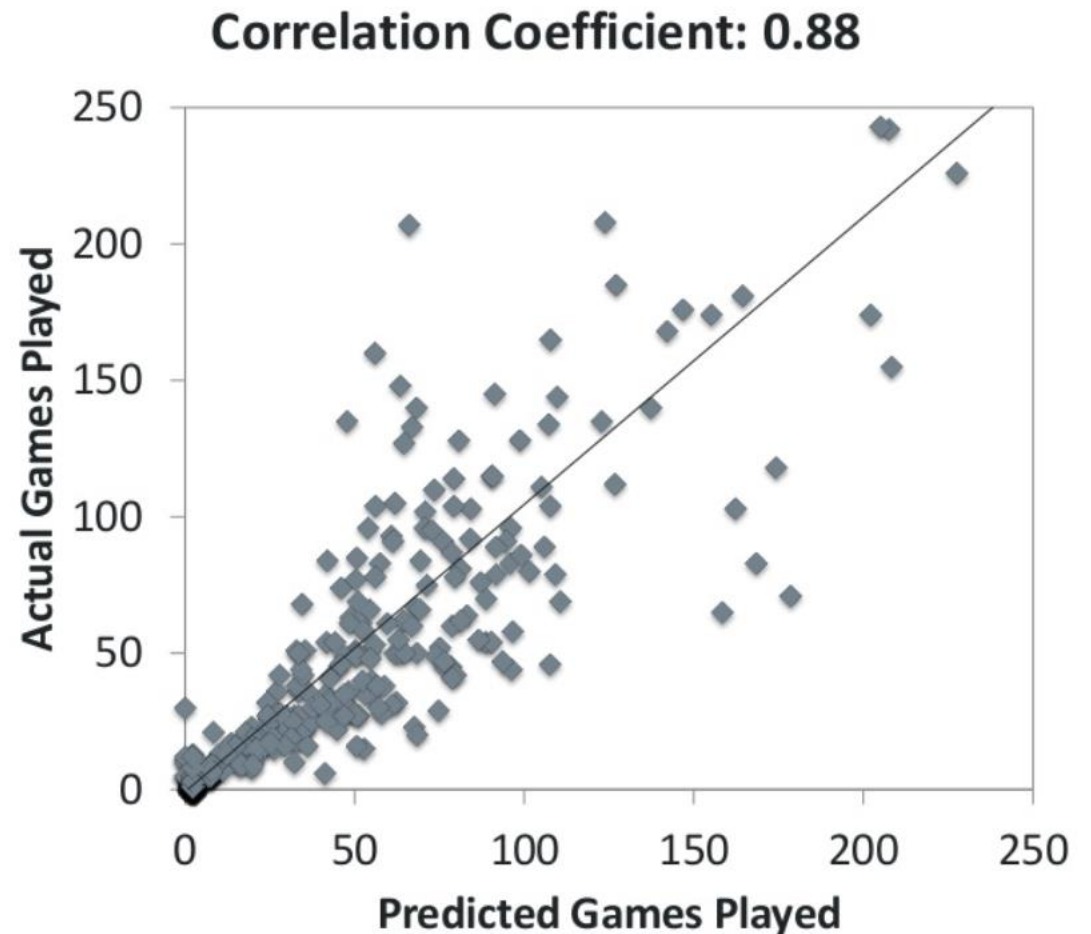
- Start with some data about a playerbase and game world
- **Categorize** learn what types exist
- **Regression analysis** how does  $x$  relate to  $y$ ?
  - $X$ : Input about the playerbase/game world
  - $Y$ : Something we want to predict/understand
- **Classify** which  $y$  is  $x$  a member of?

# Quick Tangent: Churn

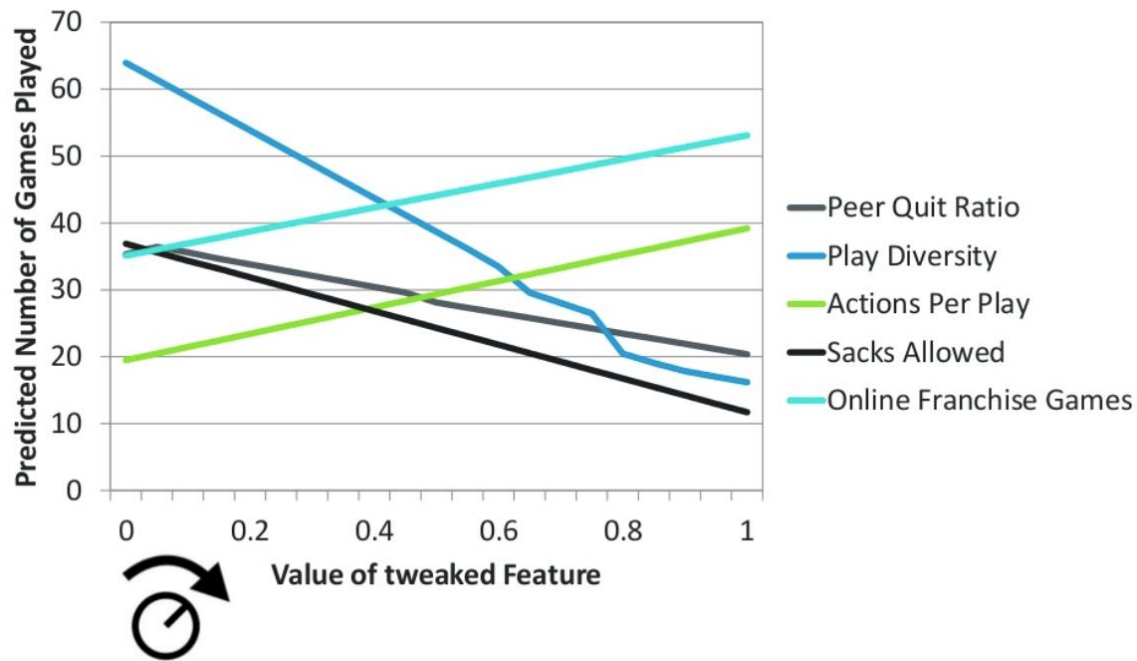
- Churn rate: The rate at which customers cut ties with a company
- In games this is how quickly a game loses players. After how long do they stop playing?
- Churn is one of the most common problems player analytics teams are tasked with

# Mining The Madden ('11) Experience

- What gameplay features impact player retention? What are optimal win rates for retention?
  - Use ML to build models of player behavior
  - Analyze generated models to ID influential gameplay elements
- Used 350 GB of play-by-play summaries of ~25k Xbox 360 players.
- Converted player behavior to (46) feature vector rep
  - Game modes (usage, win rates)
  - Performance metrics (turnovers, gain)
  - End conditions (completions, peer quits)
  - Feature usage (gameflow, scouting, audibles, special moves)
  - Play preference (running, play diversity)



# Most influential features in predicting retention



Correlation Strength ↑

Feature	Impact
Play Diversity	Negative
Online Franchise Wins	Positive
Running Plays	Positive
Sacks Made	Positive
Actions per Play	Positive
Interceptions Caught	Positive
Sacks Allowed	Negative
Peer Quit Ratio	Negative

# Cellular Automaton

## Pros

- Easy to implement
- Complex behavior from small set of rules
- Intuitive mapping to game space generation
- Local coherence

## Cons

- Reliant on guess and check for modifying
- Hard to spot edge cases
- No way to forbid undesired output
- Upper bound is slow
- Global incoherence



# Comparisons: CA vs Agents

- Similarities (rule system commonalities):
  - Rely on “emergent” output of simple rules
  - Hard to make strong designer restrictions
  - Requires a lot of guess and check
  - Can take a long time to converge
- Differences
  - Agents allows for some global assurances
  - CA more emergent in its output



# Major Drawback of CA

- What cellular automaton could you construct to build a house?  
A sword? An NPC?
  - Local rules don't adapt well to generating bits that require strong global coherence
  - Said another way, no one wants an NPC that only looks locally like a human

# Generative Grammars

## Pros:

- General usage
- High-quality, “feels designed” quality
- Accessible
- Fast

## Cons:

- Large burden on design
- Hard to “debug”
  - How do you know if a fix actually fixed anything unless you generate infinite output?

# Comparing the Approaches

- **Rule Systems** allow for emergent output from simple rules. But hard to control output.
- **Generative grammars** can create high-quality output, but depend upon expert authoring of components and production rules
- **Search** cuts back on authoring burden to just a heuristic and searchable representation, but these are unintuitive for many designers

# On Maps

- Making maps with noise:
  - <https://www.redblobgames.com/maps/terrain-from-noise/>
  - <https://www.redblobgames.com/maps/mapgen2/>
  - “Also take a look at mapgen4, my newer map generator that allows painting your own mountains, valleys, and oceans. It then simulates evaporation, wind, and rainfall, generating biomes and rivers that fit your map.”
    - <https://www.redblobgames.com/maps/mapgen4/>
- More maps (Jason Grinblat):  
<https://twitter.com/ptychomancer/status/980968298002006016>

See <http://pcgbook.com/wp-content/uploads/chapter07.pdf>

# **PCG: PLANNING & CONSTRAINT SOLVING**

# Story/Quests in Games

**Backstories** set up action/mood/theme/motive, but may or may not relate to level progression and game mechanics

**Story:** Series of events players experience

- Linear: players experience the same story
- Multilinear/Non-linear: players experience variations on the same story based on choices

**Quests:** Give the player something to do for experience points, challenge, or the story

# Why would we want to generate these?

- Add more content to a game
- Create new types of game experiences
  - More on this in later lecture



## **EA wants to use machine learning to create real-time game narratives**

[https://www.gamasutra.com/view/news/299840/EA\\_wants\\_to\\_use\\_machine\\_learning\\_to\\_create\\_realtime\\_game\\_narratives.php](https://www.gamasutra.com/view/news/299840/EA_wants_to_use_machine_learning_to_create_realtime_game_narratives.php)

# Another motivation

- PCG'd game worlds can lack meaning / motivation to the player
  - “without context and goals, the generated behaviours, graphics, and game spaces run the danger of becoming insubstantial and tedious.” (Ashmore & Nitsche)
  - Fix: tie PCG of game world into the PCG of the game story
- Computer RPGs often have a particularly degenerate form of quest, “generally structured as a list of tasks or milestones,” rather than open-ended goals the player can creatively satisfy
- How can quests be dynamically generated and adapted during gameplay?



# Quest Generation: Templates

- We can create a quest template based on standard quest types or to fulfill a specific in-game purpose
- Example:  
Assassination: Go to <LOCATION> and kill <Non-Essential NPC>

# Quest Templates

- Variables: like <LOCATION>, slots that can be filled by specific values
- Values: The elements that can fill variables
- Rules: Constraints between variables that limit the types of values we can place in them.
- Generation is then trivial random selection

# Quest Templates Example: Bethesda

- Collect values of types (LOCATION, NPC, etc) once the player has visited or been made aware of them
- On generation select a template and fill in with values from these types



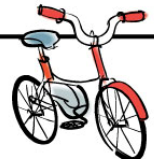
# Quest Templates

## Pros

- Fast (basically a generative grammar)
- Can be customized to player experience


## Cons

- Basically madlibs where you do the same madlib over and over again
- Gets boring quickly



**BIKE RIDING!**

Most doctors agree that bicycle \_\_\_\_\_ is a/an \_\_\_\_\_ form of exercise. \_\_\_\_\_ a bicycle enables you to develop your \_\_\_\_\_ muscles as well as \_\_\_\_\_ increase the rate of your \_\_\_\_\_ beat. More \_\_\_\_\_ around the world \_\_\_\_\_ bicycles than drive \_\_\_\_\_. No matter what kind of \_\_\_\_\_ you \_\_\_\_\_, always be sure to wear a/an \_\_\_\_\_ helmet. Make sure to have \_\_\_\_\_ reflectors too!



© ClassroomJr.com. All Rights Reserved.

# Story Generation

- Templates
- Planning (forward, backward, POP, HTN)
- Talespin (1992)

One day Joe Bear was hungry. He asked his friend Irving Bird where some honey was. Irving told him there was a beehive in the oak tree. Joe walked to the oak tree. He ate the beehive.

# Applying Planning to Story Generation

- What are the **actions** for a story, and what are the **goals**?
  - Option 1: a story is a sequence/chain of events in a story/game world; in world action sequence eventually leads to the story's ending
  - Option 2: a story is a sequence/chain of narrative events in support of achieving the author/narrator goals

# Planning a Story Domain

- The model of possible actions, locations, characters that can occur from a specific domain
- This must be authored by a designer instead of a typical story to use a planning approach
- What are some considerations in modeling the story domain?

# On the Perspective of the Planner

- Character/Agent-based: Each agent is a planner, comes up with own plans. Replan if they break
  - State: Current local state to an agent
  - Actions: Actions that the agent (and those local to it) is capable of
  - Goals: that of the agent (and those local to it)
- Author/Narrator-based: High-level story plans, with a much (?) larger search space
  - State: The entire current story state (characters, locations, items, etc), and state of the narrative
  - Actions: Everything any character/world and author/narrator could do
  - Goals: that of the author, and of the agents
- Hybrid: plan author-level goals on top of story-world events
  - Intent-driven planning to balance author-centric and character-centric approaches to generation



# Option 1: Agent's Goals

- Generation via story world simulation and recounting:
  - generate stories by simulating/projecting what happens as characters move around and take actions in the story world,
  - the story consists of simply recounting the events that happened
- Drawback?
  - Stories are carefully crafted by authors: pace, dramatic tension, foreshadowing, a narrative arc, etc.,
  - A simulation of a day in the life of a virtual character does not necessarily have these features, except by accident
  - An optimal plan (minimum steps) is rarely the most desirable story

# Story Planning

## Pros:

- Many possible stories
- Can adapt to the player

## Cons:

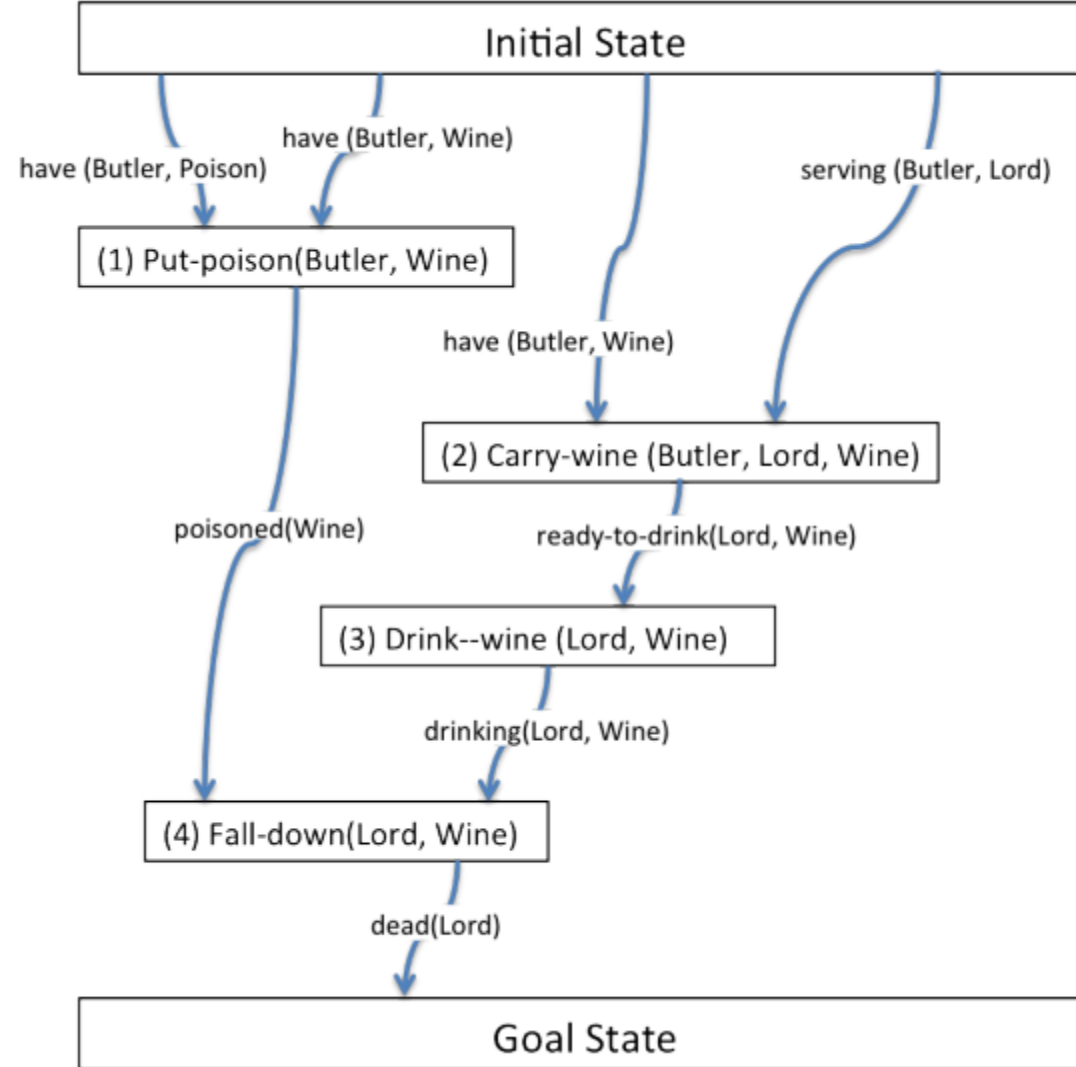
- Slow
- Hard to get heuristics right (optimal stories rarely interesting)
- Have to ensure tie-in to the game world

# Plan-space vs State-space

- Both state-space search and plan-space search algorithms have advantages. Which do you think has been favored?
- Plan space planners have been favored in creating stories because
  - their representations are similar to the mental structure that humans construct when reading a story (Trabasso & Sperry)
  - their search processes resemble the way humans reason to find a solution (Rattermann et al.).
- Furthermore, causal relationships encoded in the plan structure allow further investigation of computational models of narrative such as summarization and affect creation

# Recall that

- State-space planners
  - Make total-ordered plans
  - Search a tree where nodes are states, and arcs are state transitions caused by an operator
  - The solution is a sequence of arcs from initial to goal states
- Plan-space planners
  - Make partially-ordered plans
  - Search a tree where nodes are partial plans, and arcs are refinements to those partial plans
  - The solution is a leaf node, a plan without flaws
- Hierarchical task network planners
  - Can involve both partial and totally ordered tasks
  - Search a tree of methods that accomplish the specified tasks and subtasks by recursively splitting composite non-primitive tasks into primitive tasks
  - The solution is a full decomposition from task to primitives obeying the method and operator constraints



# On Planning Languages

- Most STRIPS-style representations make a closed-world assumption
  - Any conditions not explicitly specified are considered false.
  - Only positive literals are used for the description of initial states, goal states, and preconditions.
  - Effects of actions may include negative literals to negate particular conditions
- Action Description Language (ADL), adds a number of additional features
  - Assumes open-world semantics: any unspecified conditions are considered unknown, not false
    - Both positive/negative literals allowed
  - Allows quantified variables, conjunction and disjunction, conditional effects, equality and non-equality predicates, typed variables

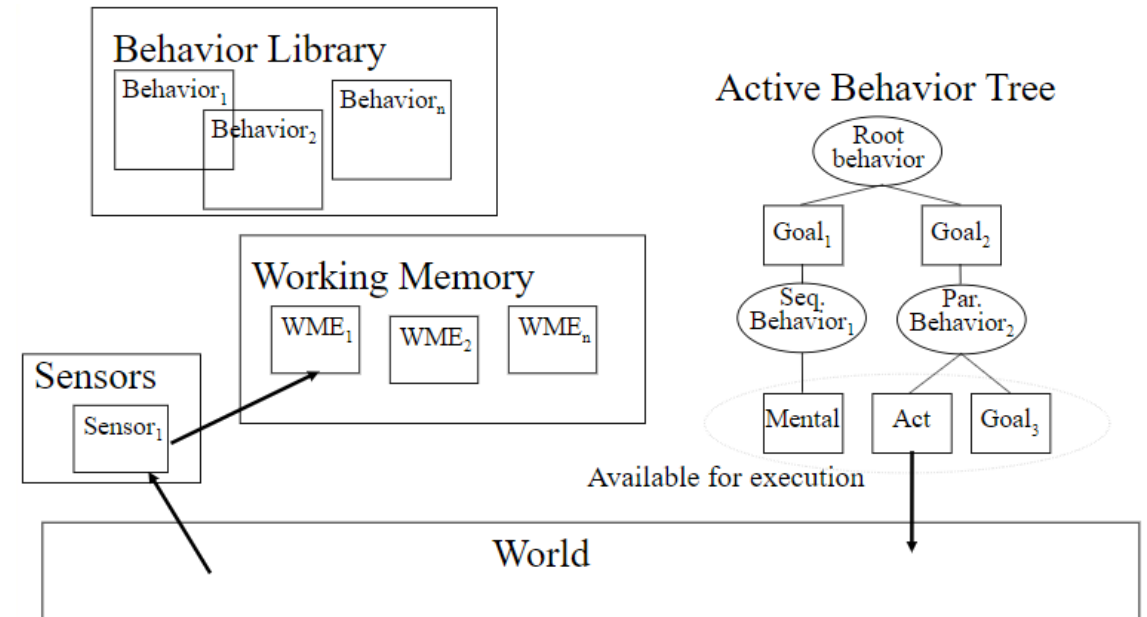
[PDDL](#) is an attempt to standardize planning languages and used at the IPC

# On HTNs in Story Planning

- May ease some challenges:
  - More naturally support interleaving author and character goals
  - Allow replanning just the chunks of the story the player messes up
  - Can delay fully planning out tasks/subtasks
  - Are often much faster (highly constrains the search)
- HTN Story Planning Examples by Fred Charles, Marc Cavazza and Steven Mead
  - <https://www.youtube.com/watch?v=0erFey9hQTY>
  - <https://www.youtube.com/watch?v=3wzSb8fzDA4>

# Behavior Trees + HTN Planning: ABL

- ABL: A behavior Language
  - Reactive planning for real-time, complex agents
- Features
  - Sequential & Parallel behaviors
  - Demons & continuously monitored conditions
  - Multiple simultaneous goal pursuit with reactivity
  - Joint goals and behaviors (joint intentions)
  - Reflection (meta-behaviors)
  - Sensory-motor abstraction





# ABL On Executing a Plan

- All steps (subgoals, primitive acts, mental acts, wait) succeed or fail
  - Pass/fail propagates up the ABT
  - Waits are used with conditions to accomplish demons
- Continuously monitored conditions: make behavior immediately reactive to world changes
  - Success test: spontaneously pass if test is satisfied
  - Context conditions: spontaneously fail a behavior if test is satisfied
- Execution:
  - ABL: action begins executing as soon as it is selected by the decision cycle
  - BT: action selection and execution decoupled
- ABL runs asynchronously from the main game update, while BTs are updated during an AI tic
- In ABL, the behavior tree is expanded as needed
  - Instantiated as a single root behavior, which can subgoal additional behaviors
  - When a behavior is selected for expansion, its steps (child nodes) are added to the active behavior tree (ABT)
  - Leaf nodes in the ABT are nodes that are available for execution, unless currently executing
- Key benefit: Architecture coordinates author-specified joint action

# **CONSTRAINT SATISFACTION**

# Constraint Satisfaction

- Also called constraint solving/constraint propagation
- Similar to search *but* instead of a heuristic with a single value, designers give constraints
  - Constraints are facts that must be true in the final “answer” output
  - Constraints can be authored or learned from an example
- Shout out to Dr. Adam Smith at UCSC

# Constraint Satisfaction Problem (CSP)

## Input

- Set of variables/slots
- Set of all values that can go in the slots
- Set of constraints that relate slots to each other

## Output

- All variables are filled with a specific value

## Trivial Example: Name generator

- Variables: <First Name> and <Last Name>
- Values: List of names
- Constraints: Names cannot start with the same letter

# CSP (General) Process

- Initially, all variables have a list of all values they could *potentially* have
- Each update, pick one variable and choose arbitrarily from its remaining values (collapse)
- Based on the constraints, remove values from the surrounding variables to represent that they are no longer possible (propagation)
- End when all variables have only one value







# CSP related to Search/Planning

- We have seen the idea of iteratively solving constraints before in partial order planning
  - Use similar heuristics: most constrained variable, minimum remaining values
- CSP is essentially search, but is generally faster as we cut out vast parts of the search space due to constraint propagation

# CSP: Dungeon Room Example

1-4	E	1-4	1-4
1-4	1-4	1-4	1-4
1-4	1-4	1-4	1-4
1-4	1-4	1-4	1-4

Values:

1. Blank 
2. Wall 
3. Enemy 
4. Treasure 

Constraints:

- There must be a path from the entrance to all treasure and all enemies
- There can only be at most 1 treasure
- There can only be at most 2 enemies
- Enemies cannot be next to each other

# CSP Step 1

Collapse:

1-4	E	1-4	1-4
1-4	1-4	1-4	1-4
1-4	1-4	1-4	1-4
1-4	1-4		1-4

Propagate:

1-4	E	1-4	1-4
1-4	1-4	1-4	1-4
1-4	1-4	1,2, 4	1-4
1-4	1,2,4		1,2, 4

Constraints:

- There must be a path from the entrance to all treasure and all enemies
- There can only be at most 1 treasure
- There can only be at most 2 enemies
- **Enemies cannot be next to each other**



# CSP Step 2

Collapse:

1-4	E	1-4	1-4
1-4	1-4	1-4	1-4
1-4	1-4	1,2, 4	1-4
1-4	1,2, 4		

Propagate:

1-3	E	1-3	1-3
1-3	1-3	1-3	1-3
1-3	1-3	1,2	1-3
1-3	1,2		

Constraints:

- There must be a path from the entrance to all treasure and all enemies
- **There can only be at most 1 treasure**
- There can only be at most 2 enemies
- Enemies cannot be next to each other

# CSP Step 3

Collapse:

1-3	E	1-3	1-3
1-3	1-3	1-3	1-3
1-3	1-3		1-3
1-3	1,2		

Propagate:

1-3	E	1-3	1-3
1-3	1-3	1-3	1,3
1-3	1-3		1,3
1-3			

Constraints:

- **There must be a path from the entrance to all treasure and all enemies**
- There can only be at most 1 treasure
- There can only be at most 2 enemies
- Enemies cannot be next to each other

# CSP Output

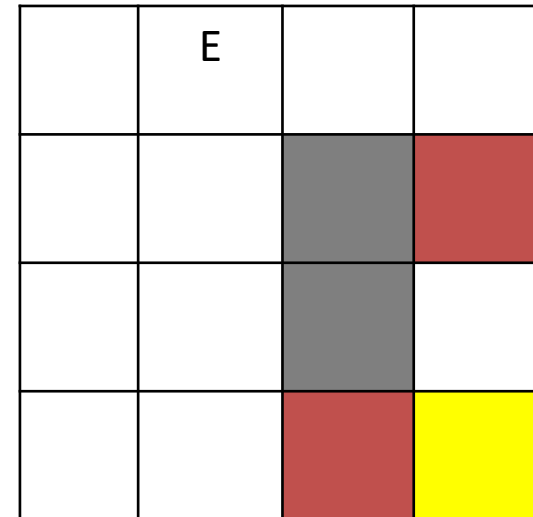
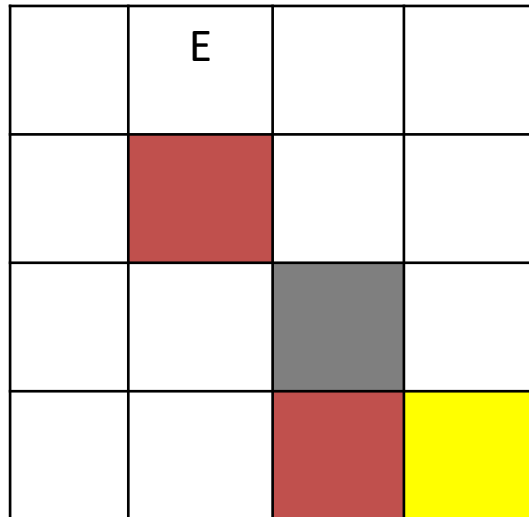
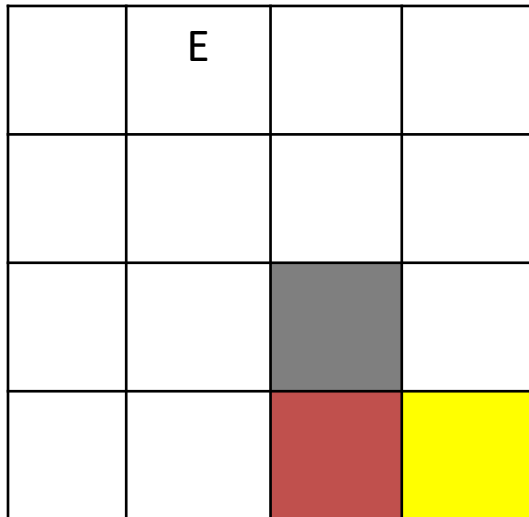
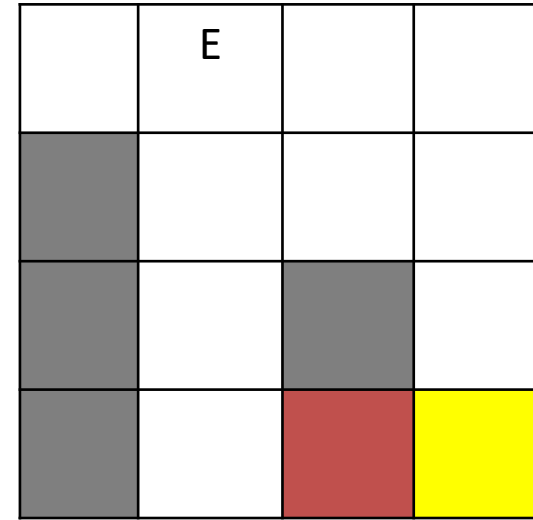
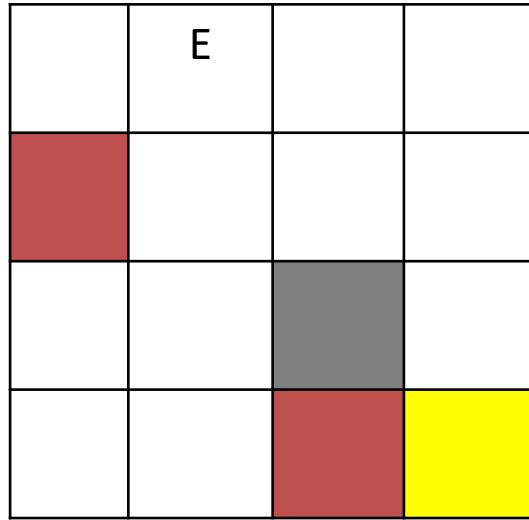
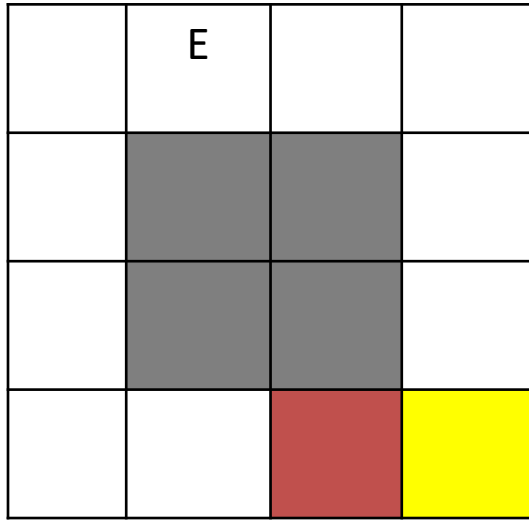
Values:

1. Blank 

2. Wall 

3. Enemy 

4. Treasure 



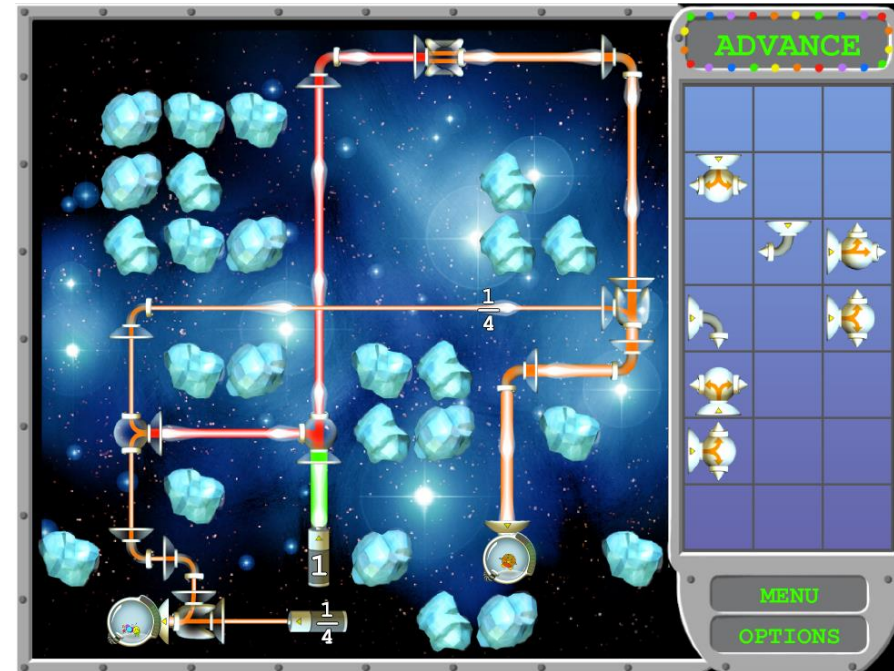
# CSP Usage

If output doesn't look good, just add more constraints!

- Don't have to consider balancing heuristic

Need to strike balance of overly constrained/avoiding bad output

- In general, designers prefer this over heuristic tuning



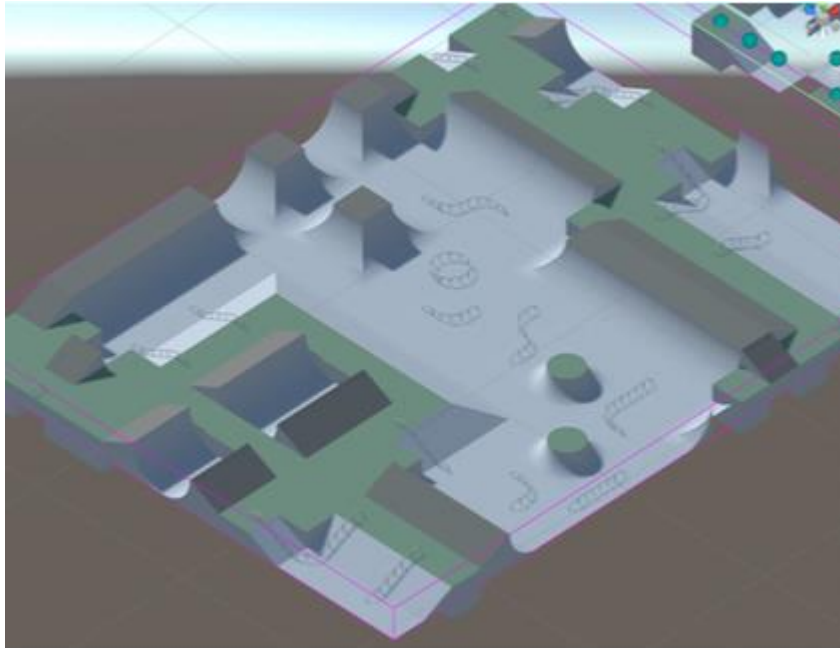
# Question 1

- Come up with a very simple generation task to which you could apply constraint satisfaction
  - Give the variables for the problem
  - Give the values for those variables
  - Give the constraints

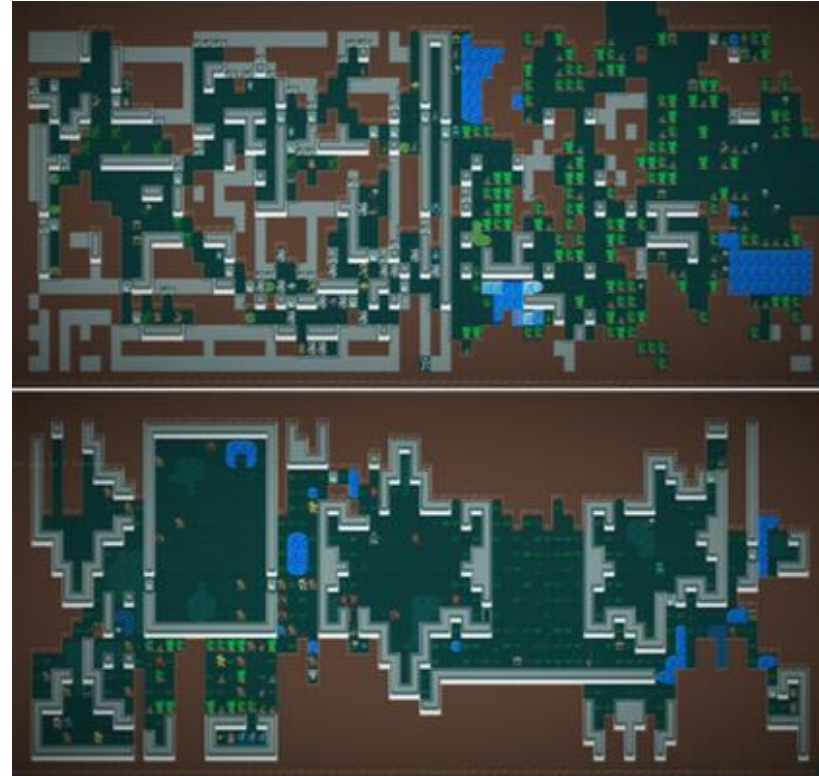
Writing constraints seems hard

Didn't you say constraints could be learned?

# Wave Function Collapse: The New Hotness



Proc Skater, Joseph Parker (2016)



Caves of Qud, Freehold Games (2016)

<https://github.com/mxgmn/WaveFunctionCollapse>

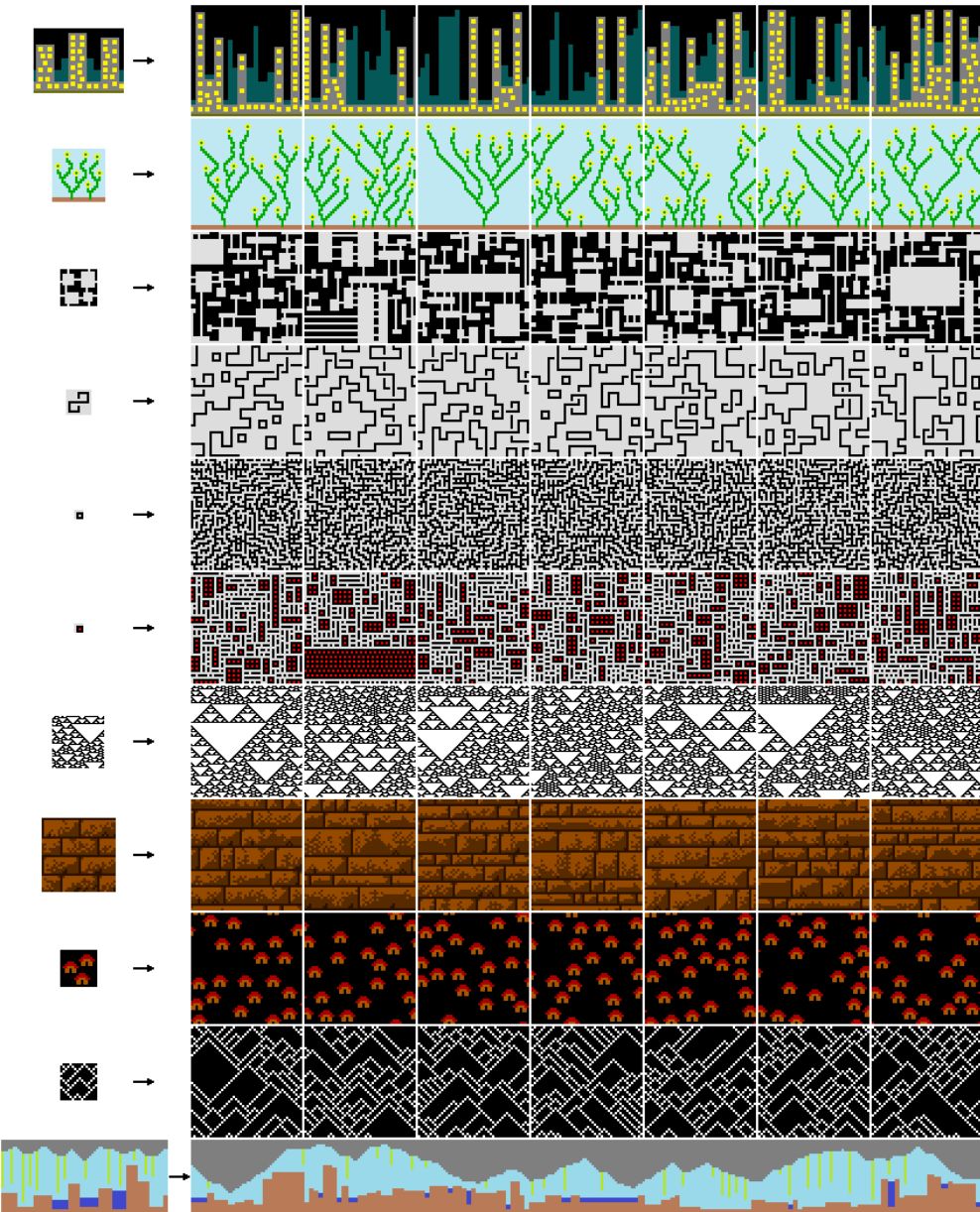
# Wave Function Collapse

- <https://github.com/mxgmn/WaveFunctionCollapse>
  - <https://twitter.com/OskSta/status/865200072685912064>
- Developed by Maxim Gumin and released as open source 2016
  - Caves of Qud was first commercial use, many others quickly followed
- Two primary execution modes: tile maps and textures
  - Tilemap generation creates tile sets via propagation of adjacency constraints
  - Texture mode uses small (~16x16) training images to make arbitrarily large output textures locally similar to input

See also: [http://bfnightly.bracketproductions.com/rustbook/chapter\\_33.html](http://bfnightly.bracketproductions.com/rustbook/chapter_33.html)



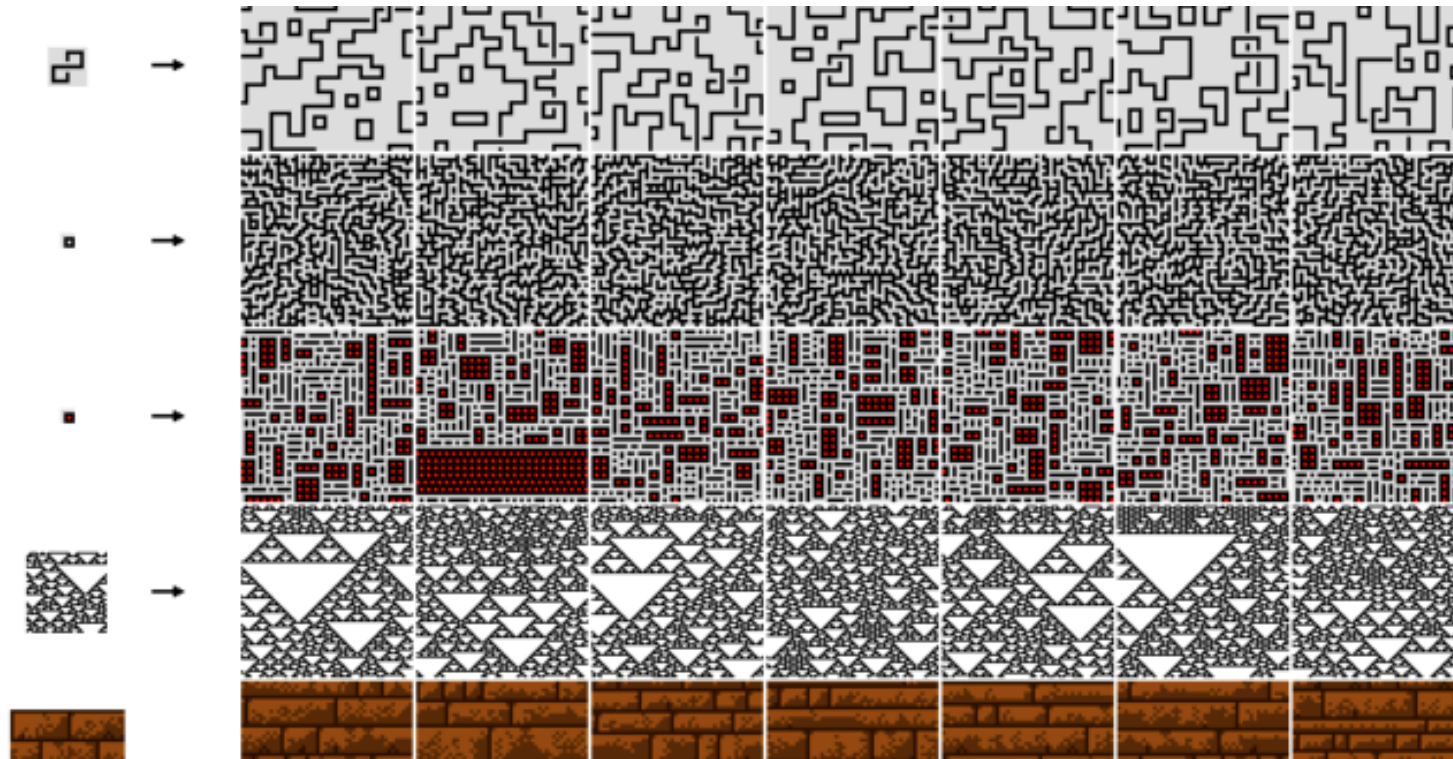
# Wave Function Collapse



- Procedural generation from a single example with WaveFunctionCollapse
  - <https://www.youtube.com/watch?v=DOQTr2Xmlz0&feature=youtu.be>
- Play with it!
  - <http://oskarstalberg.com/game/wave/wave.html>

# Wave Function Collapse Big Ideas

1. Derive legal patterns from a sample input
2. Determine legal intersections of patterns to get constraints



# Wave Function Collapse Algorithm

PatternsFromSample()

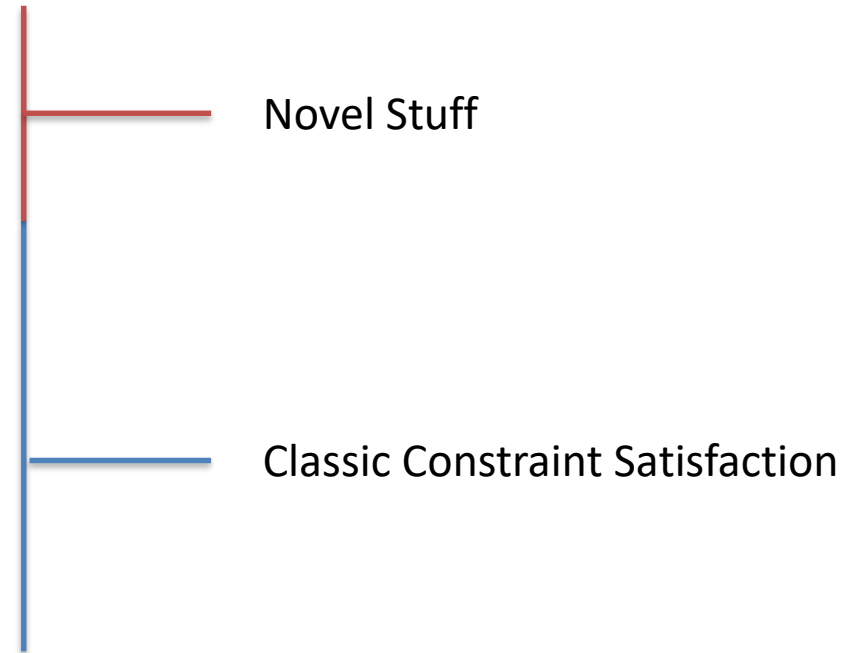
BuildPropagator()

While notFinished:

    Observe()

    Propagate()

OutputObservations()



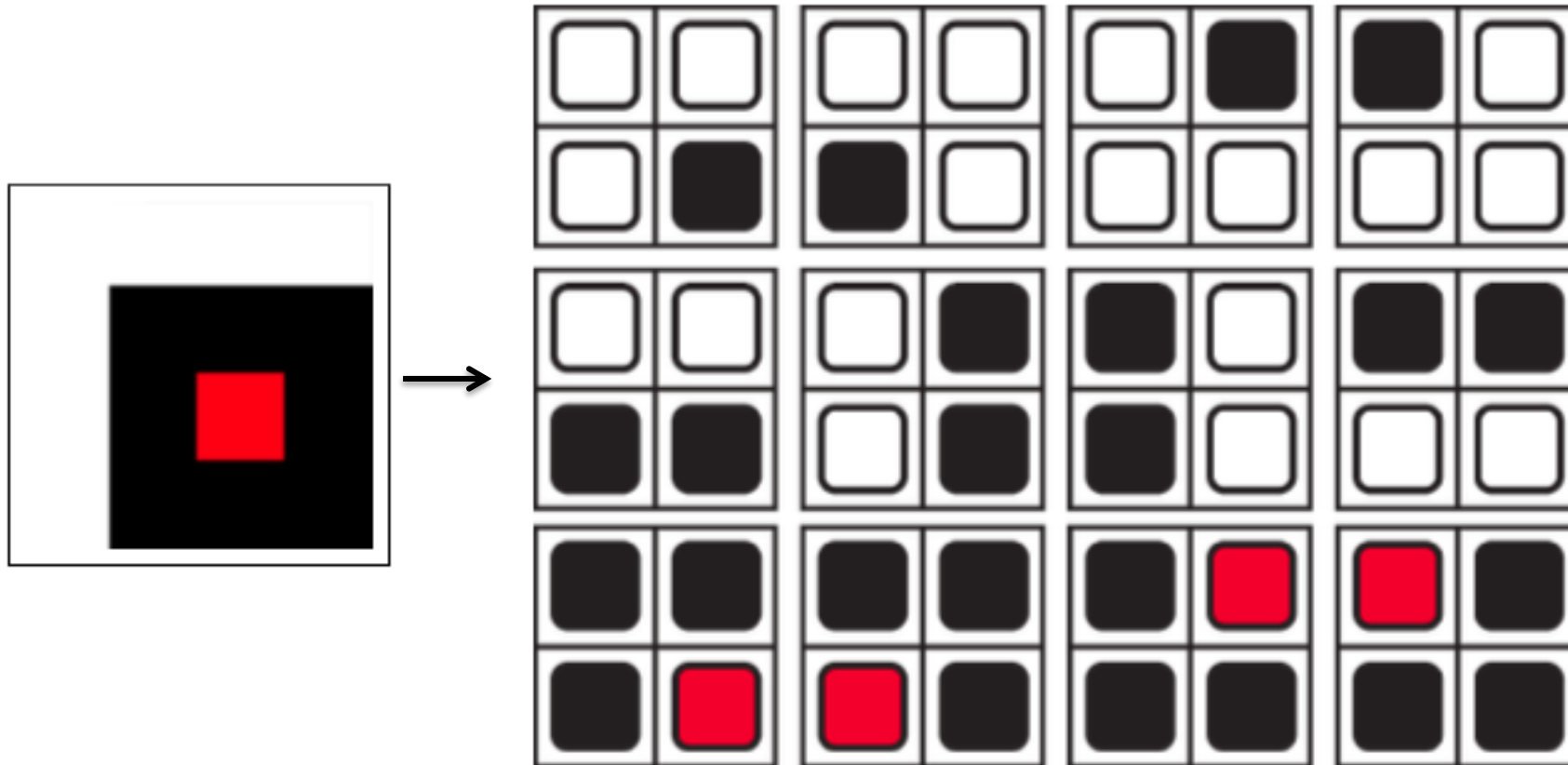
# Approach Sketch

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

<https://www.gdcvault.com/play/1026263/Math-for-Game-Developers-Tile>

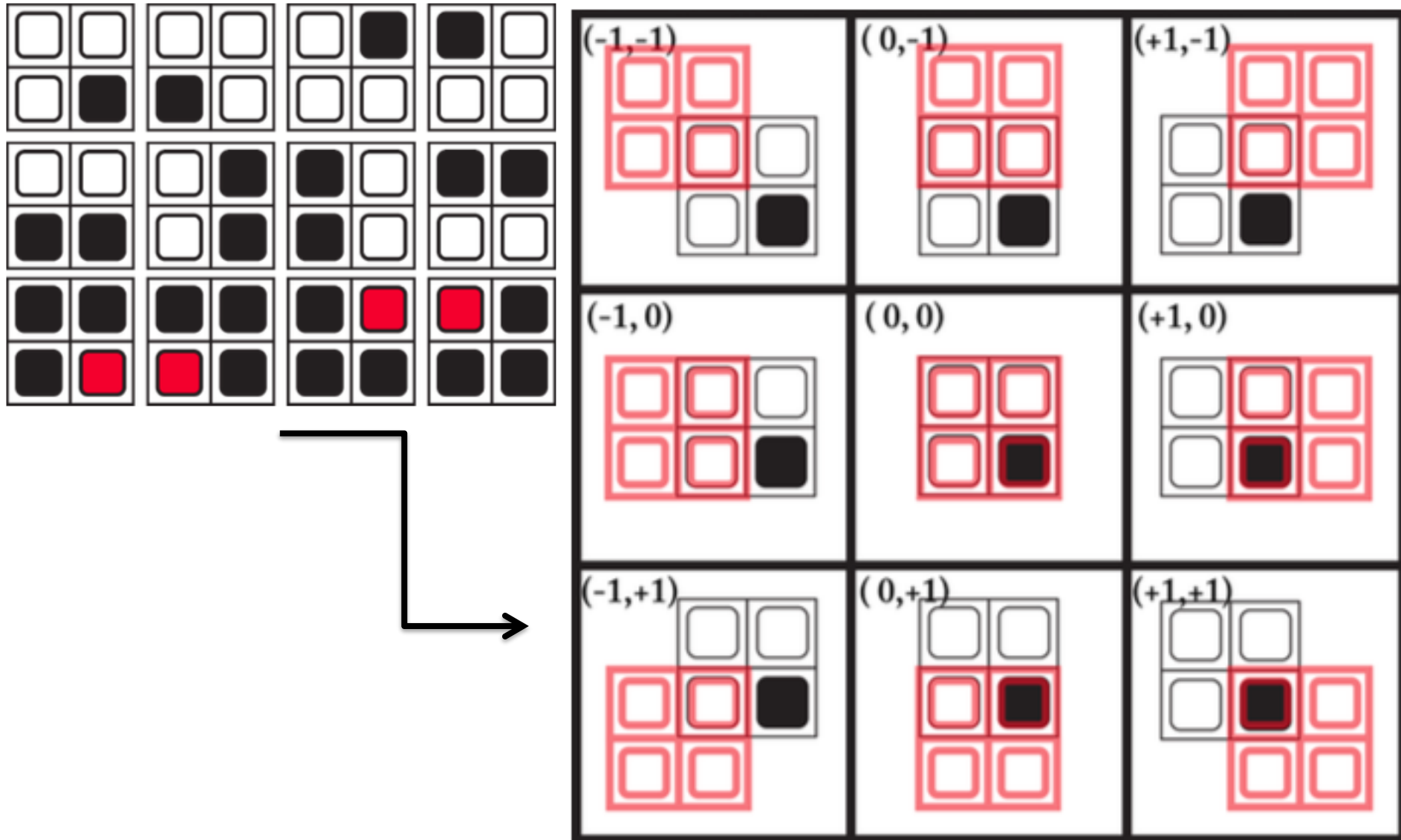
Video explanation by Brian Bucklew: <https://www.youtube.com/watch?v=fnFj3dOKclQ>

# Patterns from Sample (given neighbor size $N=2$ )

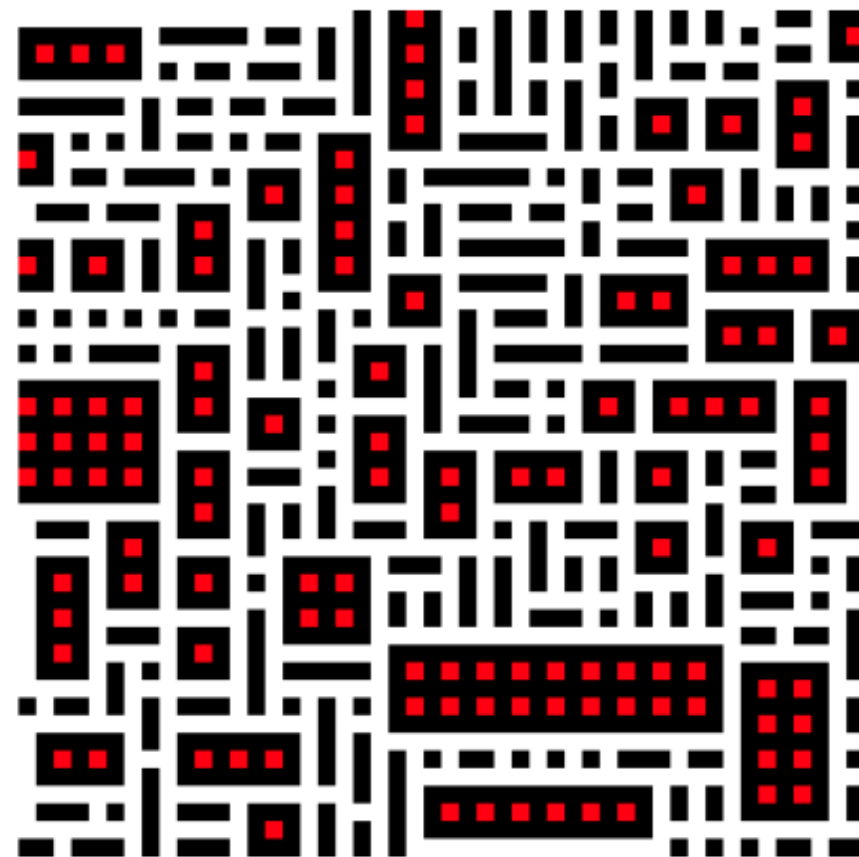
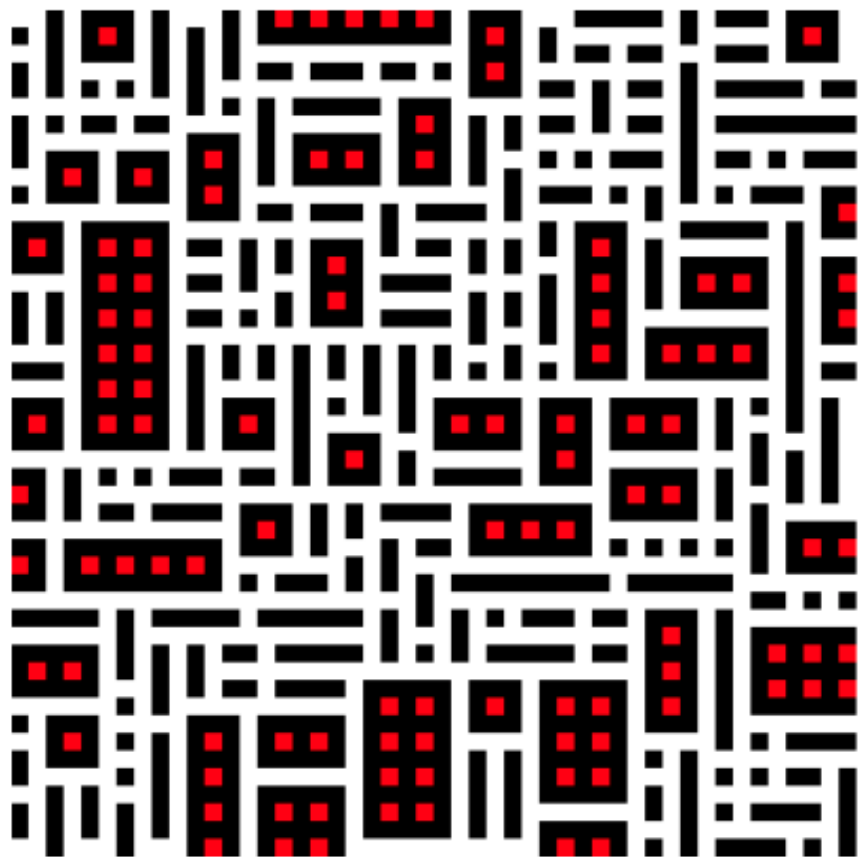


The larger the neighbor size, the more like the input sample

# Build Propagator



# Output



# CSP Pros/Cons

## Pros

- Constraints are more intuitive than heuristics
- Faster in many cases than search
  - Though slower than grammars due to propagation
- Can learn constraints from exemplar

## Cons

- Must be able to represent problem in terms of formal logic