

A screenshot from the game Minecraft, showing a detailed landscape. In the center, a waterfall cascades down a rocky cliff into a river. The river flows through a valley with various terrain types, including sand, dirt, and stone. A large, tall tree stands prominently on the left side of the waterfall. The sky is clear and blue.

# Games AI

## Lecture 5.1

Procedural Content Generation

- What can be generated?
  - Levels and maps
  - Visuals
  - Audio
  - Narrative
  - Rules and mechanics
  - Games

- Why generate content?
  - Replayability
  - Cost saving
  - New types of game
  - As a tool for artists/designers

- How do you generate it?
  - Tiles
  - Grammars
  - Distribution
  - Parametric
  - Interpretive
  - Simulations

1

4

4



\$7000

Tiles



- Tiles
  - When you have
    - Equal sized blocks of content that can be placed anywhere
    - Not a lot of constraints on how tiles can be arranged
    - Variation in placement leads to interesting variation
  - Tiles can be large or small, and have more or less internal structure
  - e.g. tile maps, solitaire boards, tarrot, ...

- Settlers of Catan
  - Shuffling cards gives random placement of terrain
  - Minimal placement constraints
  - Players adapt strategy to level

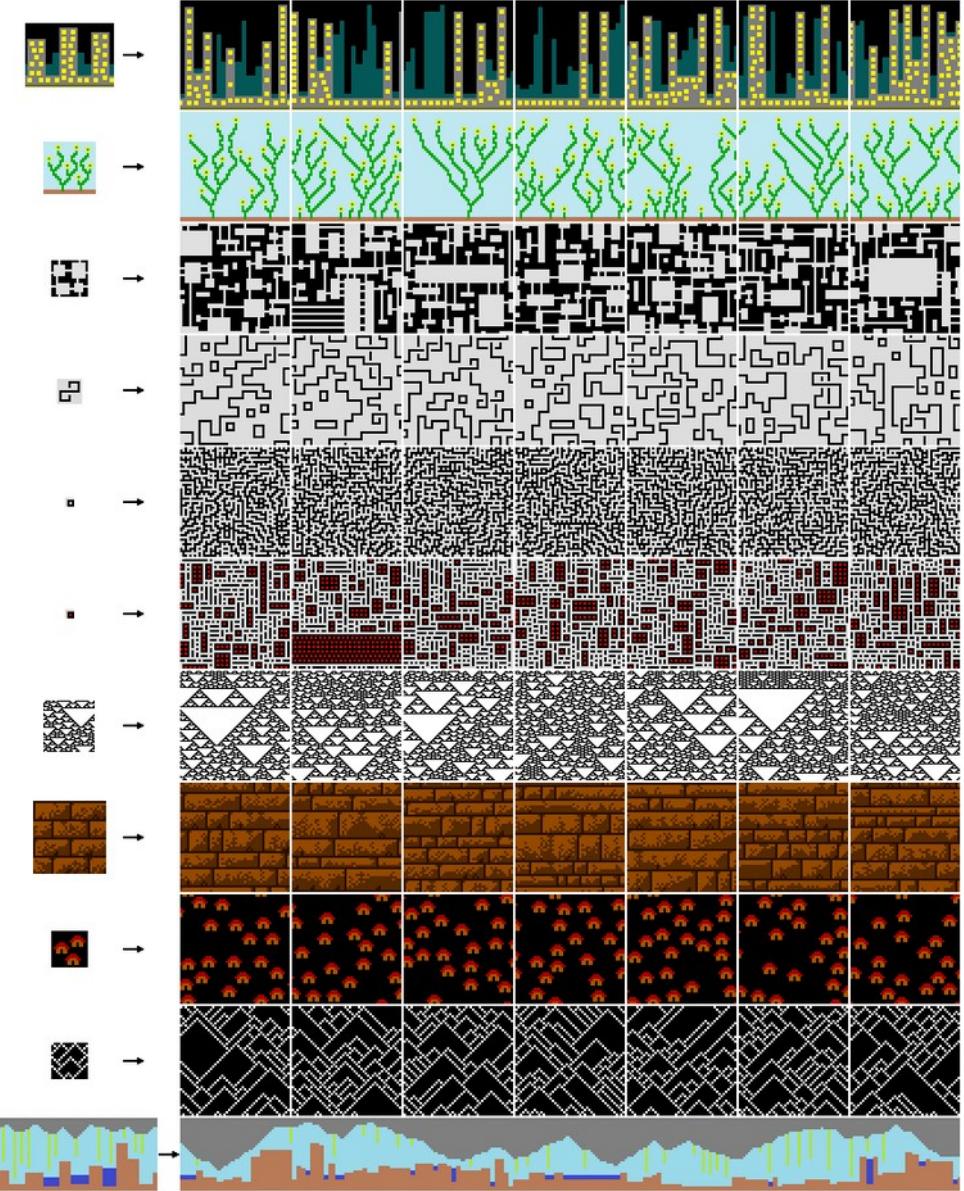


## Tiles

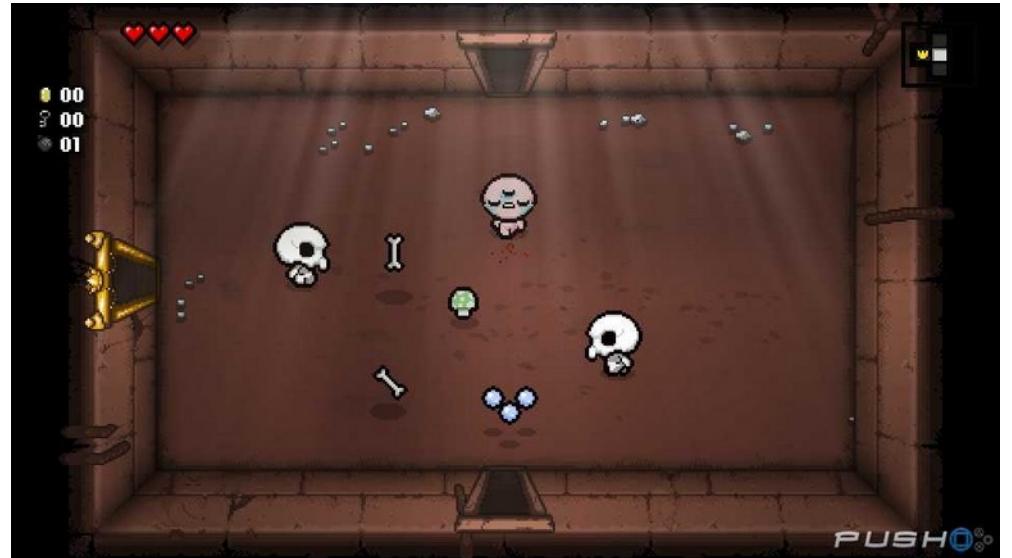
- Statistical Methods
  - Exploit statistical relationships in a sample input
  - e.g. Wave Function Collapse, inspired by quantum mechanics
  - Generates images that are similar to a small sample

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

<https://www.youtube.com/watch?v=1yvNv2Kaz4A>

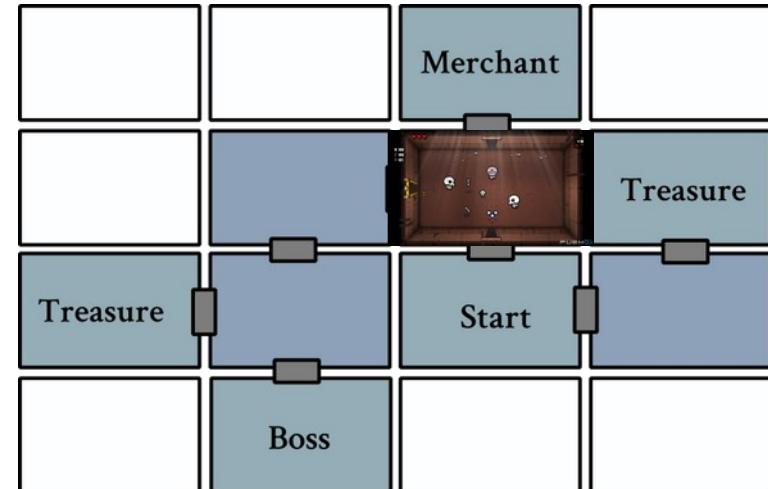
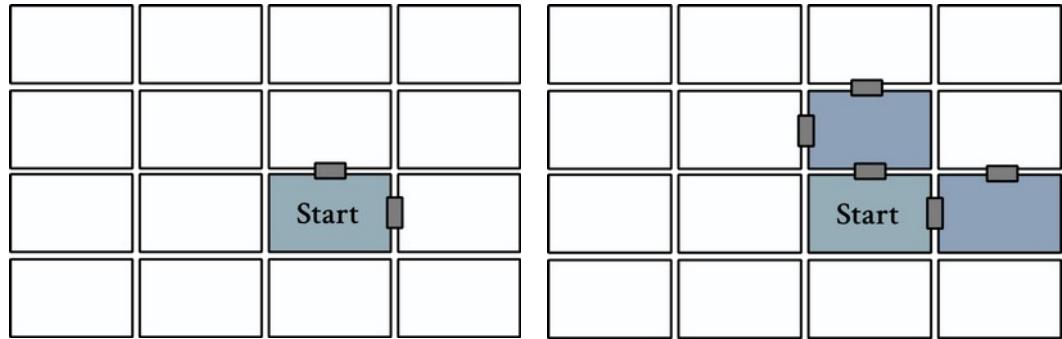


- Binding of Isaac
  - Top down roguelike shooter
  - Explore dungeon, find items, kill boss



[https://bindingofisaacrebirth.fandom.com/wiki/Level\\_Generation](https://bindingofisaacrebirth.fandom.com/wiki/Level_Generation)

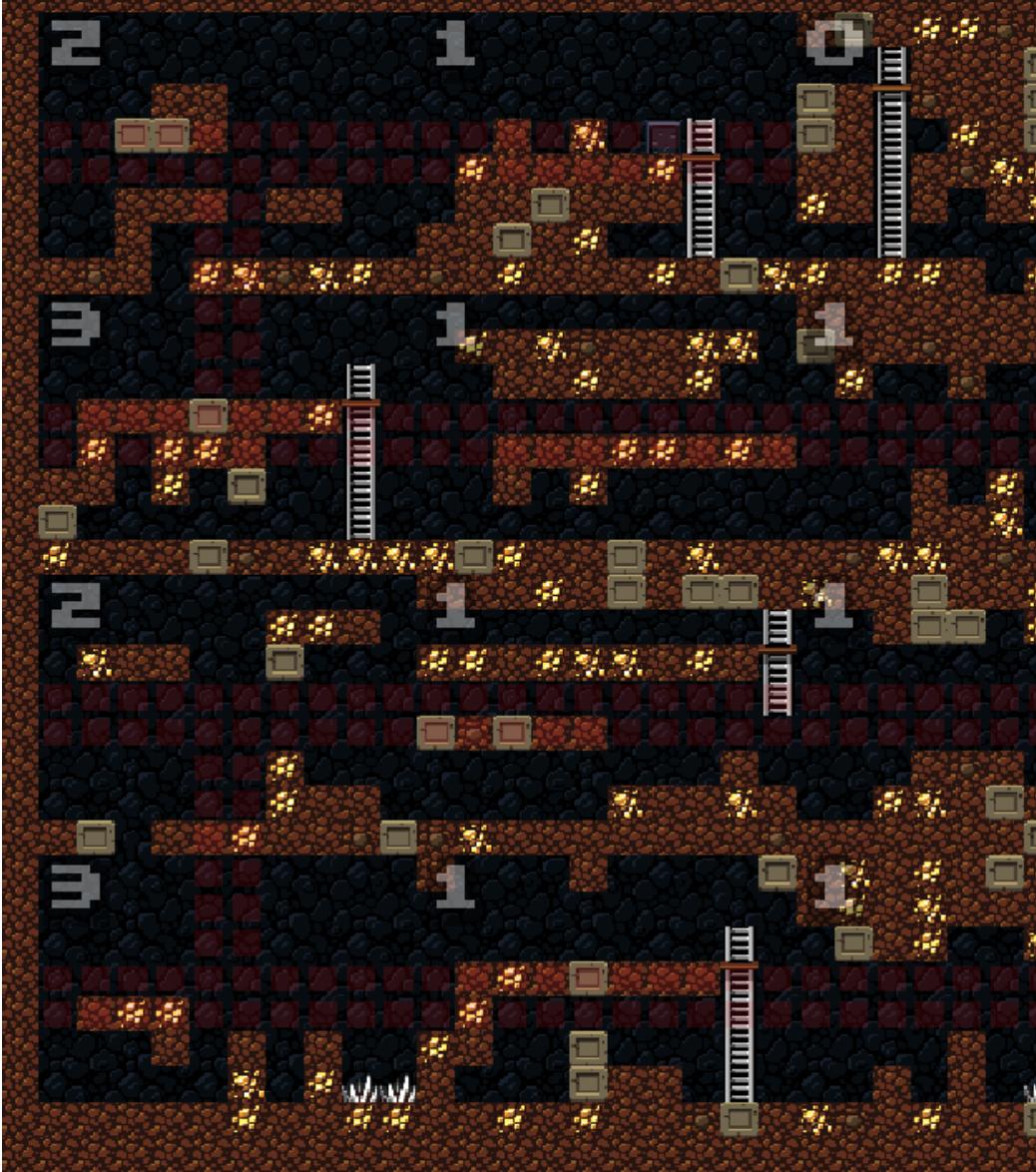
- Step 1:
  - Build layout
- Step 2:
  - Pick template rooms based on connectivity



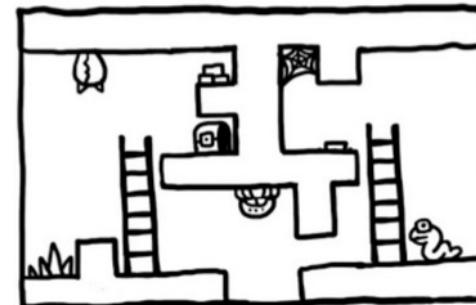
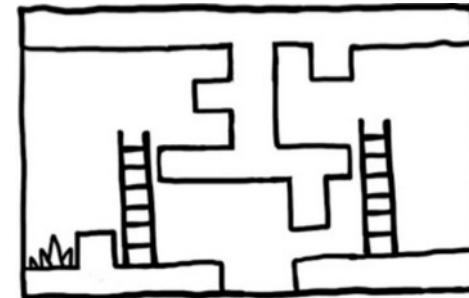
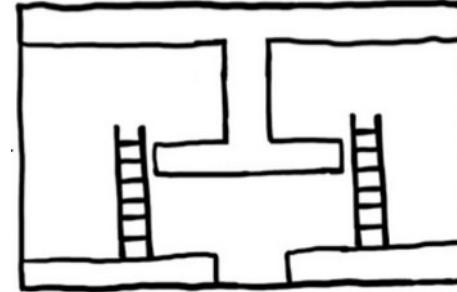
Tiles

- Spelunky Level Generation
  - Template-based
  - Walks a path through a 4x4 grid

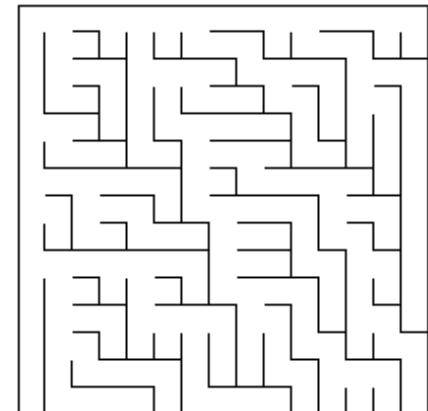
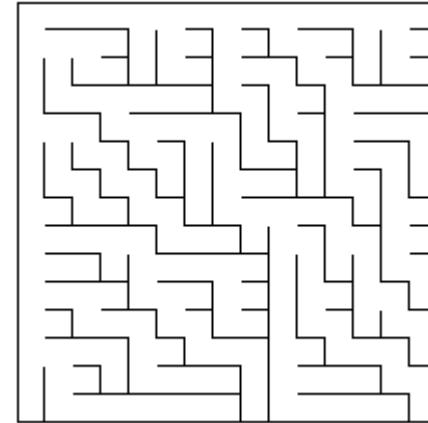
<http://tinysubversions.com/spelunkGen/>



- Spelunky
  - Room templates include limited random elements
  - Always passable



- Binary Tree Maze Algorithm
  - For each cell, randomly dig a path north or west\*. Simple!
  - Shaped like a binary branching tree
  - Problems
    - Long corridors at edge
    - Strong north-west bias

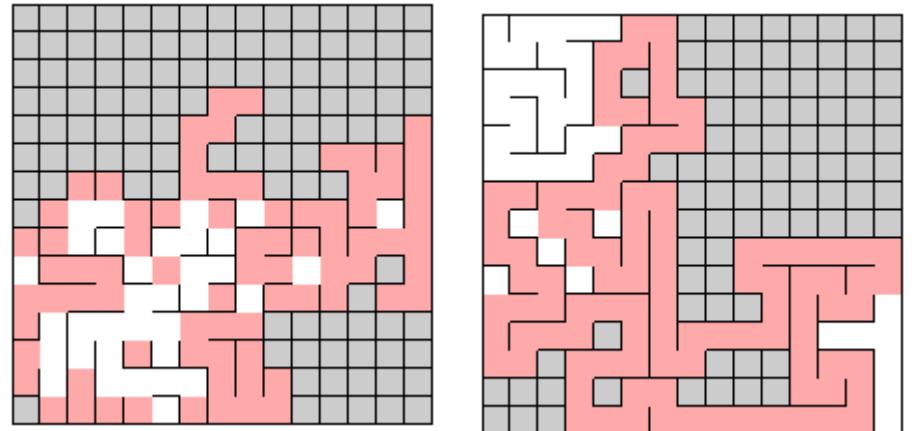
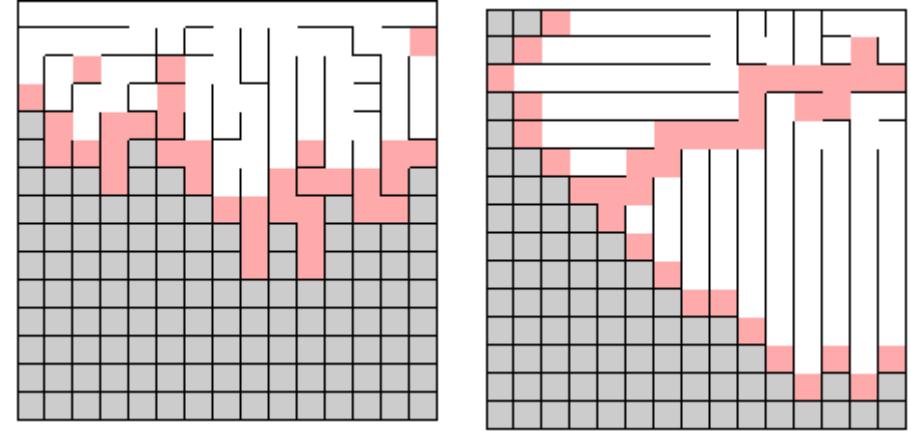


\*Or any other adjacent directions, so long as you're consistent

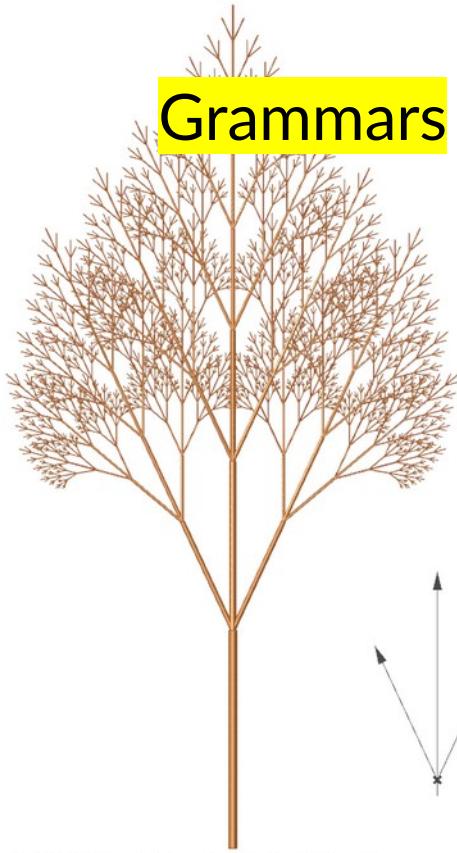
<http://weblog.jamisbuck.org/2011/2/1/maze-generation-binary-tree-algorithm>

Tiles

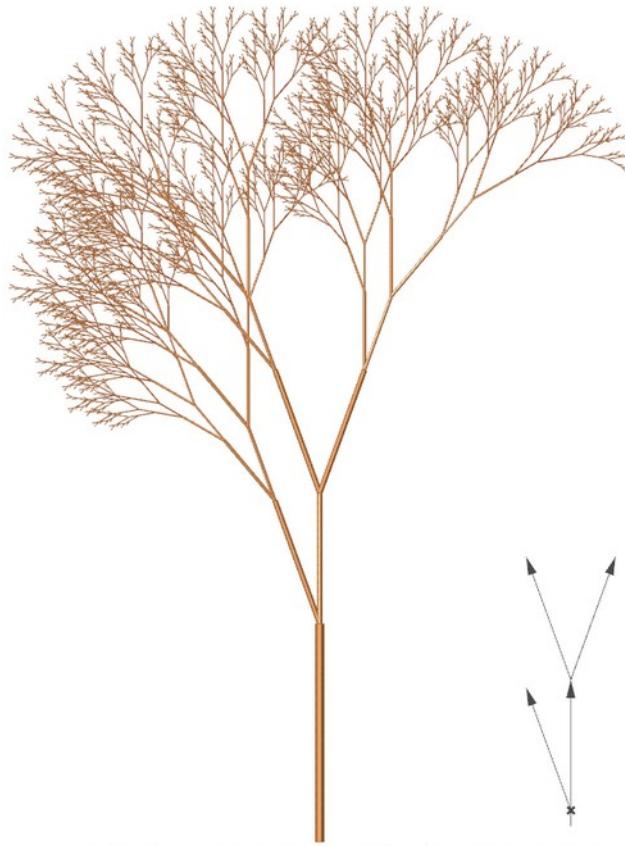
- Growing Tree Maze Algorithm
  - Create a list of cells, add starting cell
  - Choose a cell in list
    - Newest? Random?
    - How you choose affects style of maze
  - Carve path to any unvisited neighbour
  - Repeat until list is empty



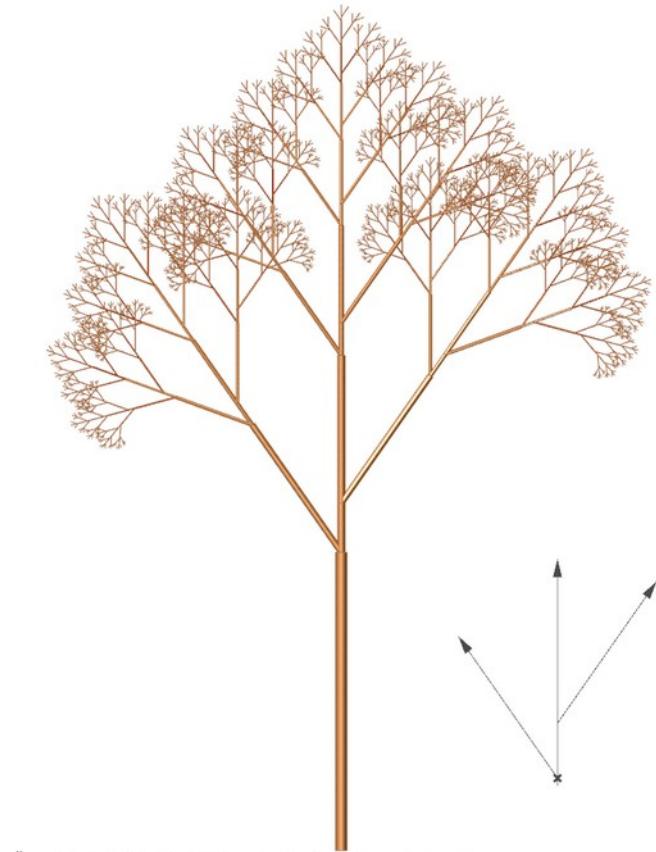
# Grammars



Variant 1 (Two angled Branches and 1 Straight Branch)

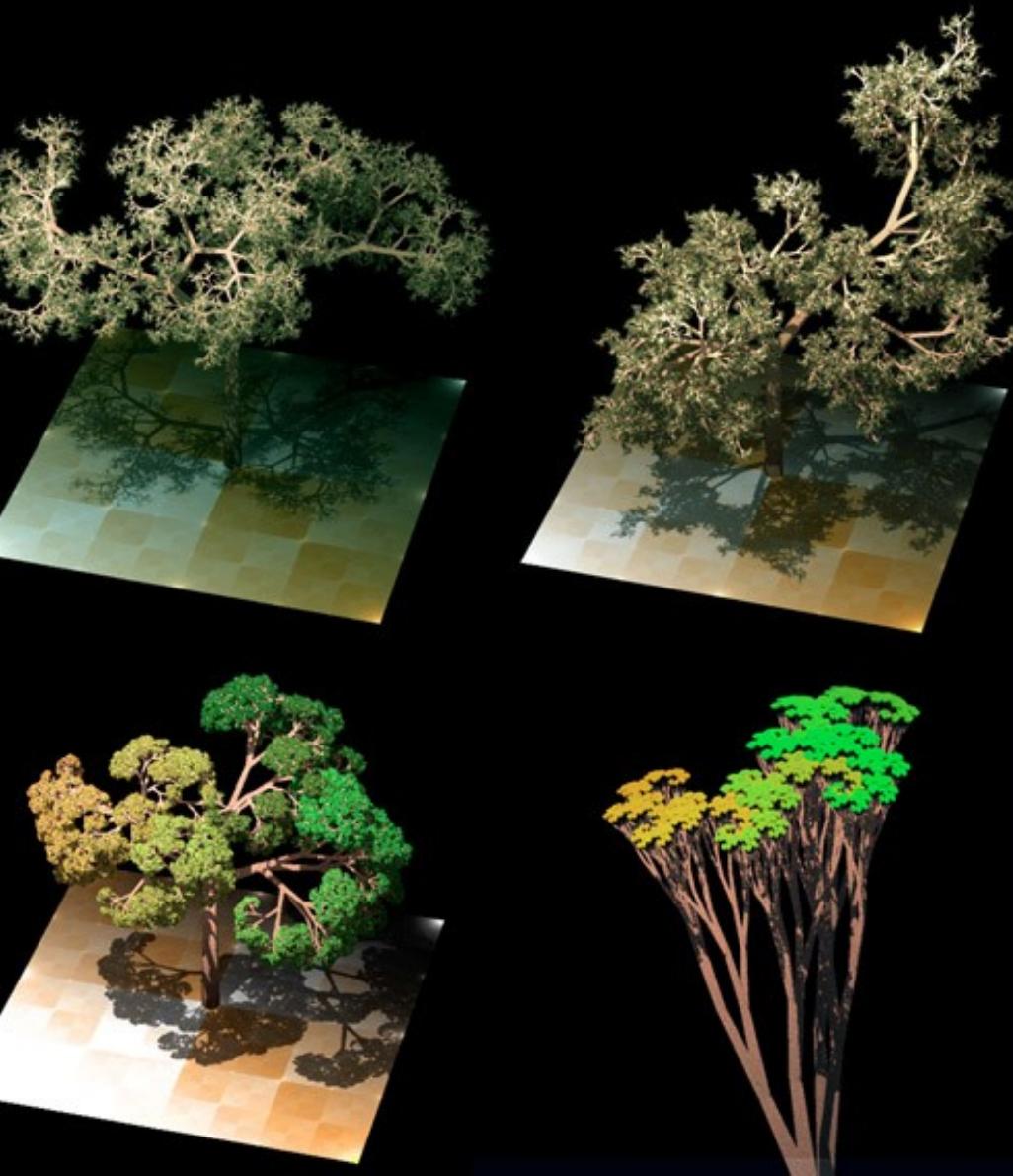


Variant 2 (One Angled and One Straight Branch, followed by two Angled Branches)



Variant 3 (One Straight Branch with alternating angled branches)

- Grammars
  - A grammar is a system of string-rewriting rules, such as
    - $A \rightarrow a$
    - $A \rightarrow A B$
    - $B \rightarrow b$
  - These generate recursive structures and are good at generating language
  - Grammars are formally defined by their complexity
  - Good at generating stuff that's defined as combinations of other stuff
  - Sub-generators can be reused



- L Systems
  - Parallel string rewriting grammar
  - Developed to model growth of algae and plants
  - Organic forms, plants, fractals

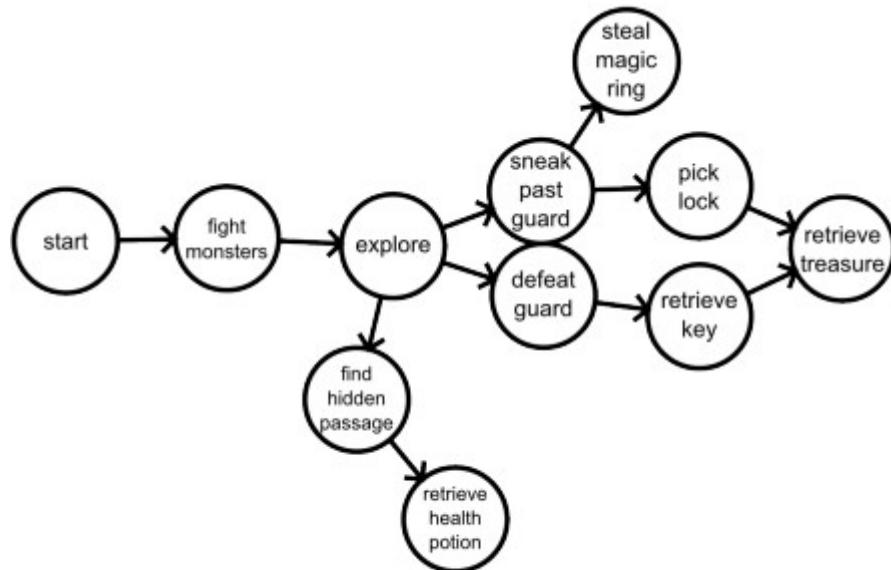


Fig. 5.6: A mission structure with two paths

- Graph Grammars
  - Rules define how to rewrite sub-parts of the graph
  - Start with trivial graph and grow complexity

Image from <http://pcgbook.com/>

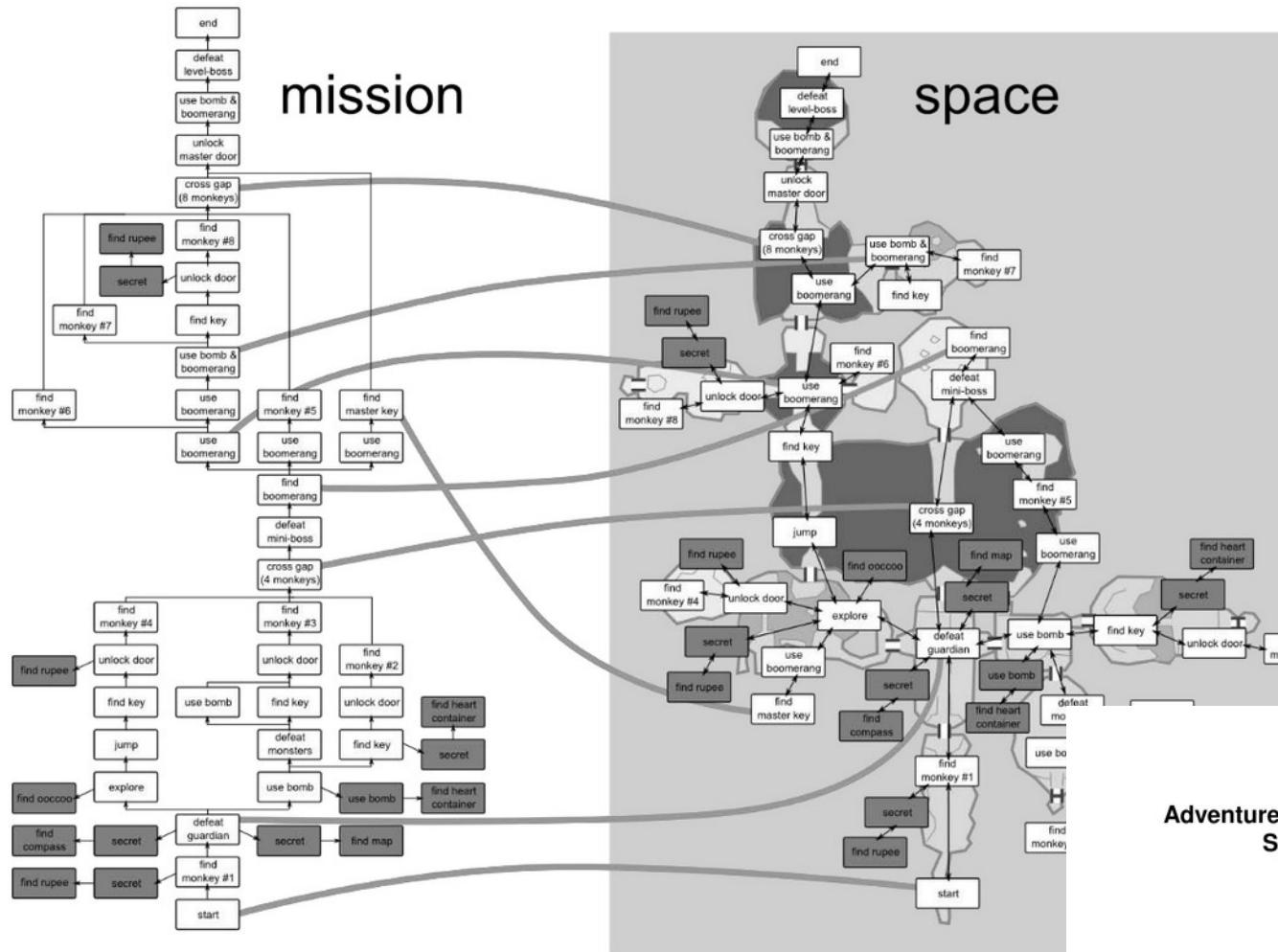


Figure 1. Mission and space in the Forest Temple level of *The Legend of Zelda: The Twilight Princess*

## Adventures in Level Design: Generating Missions and Spaces for Action Adventure Games

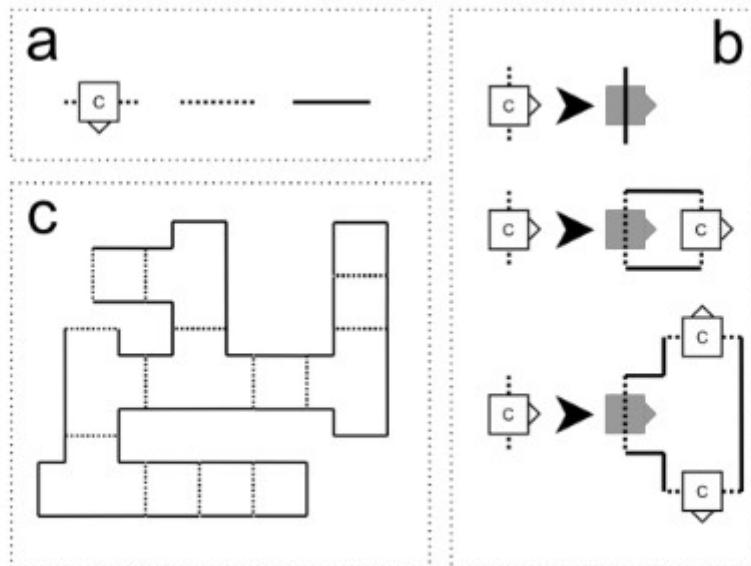
Joris Dormans  
Hogeschool van Amsterdam  
Weesperzijde 190  
1097DZ Amsterdam, The Netherlands  
+31 20 595 1686  
j.dormans@hva.nl

### ABSTRACT

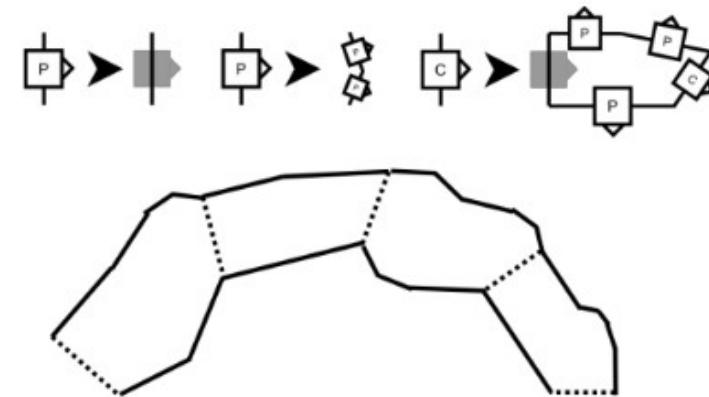
This paper investigates strategies to generate levels for action adventure games. This genre relies more strongly on well-designed levels than rule-driven genres such as strategy or roleplaying games for which procedural level generation has been successful in the past. The approach outlined by this paper distinguishes between missions and spaces as two separate structures that need to be generated in two individual steps. It

roguelike games is character building. This type of gameplay, which stems directly from a rather mechanistic interpretation of pen-and-paper roleplaying, resolves for a large part around gathering experience points and magical equipment to improve the main character. As game designer Ernest Adams points out in his satirical ‘letter from a dungeon’, there seems to be little purpose behind these mechanics, resulting in a shallow representation of character growth as a faint echo of the mythical

## • Shape Grammar



**Figure 6. Shape grammar a) alphabet, b) rules and c) output**



**Figure 7. Recursive shape rules and output**

## Adventures in Level Design: Generating Missions and Spaces for Action Adventure Games

Joris Dormans  
Hogeschool van Amsterdam  
Weesperzijde 190  
1097DZ Amsterdam, The Netherlands  
+31 20 595 1686  
[j.dormans@hva.nl](mailto:j.dormans@hva.nl)

### ABSTRACT

This paper investigates strategies to generate levels for action adventure games. This genre relies more strongly on well-designed levels than rule-driven genres such as strategy or roleplaying games for which procedural level generation has been successful in the past. The approach outlined by this paper distinguishes between missions and spaces as two separate structures that need to be converted in two individual steps. It

roguelike games is character building. This type of gameplay, which stems directly from a rather mechanistic interpretation of pen-and-paper roleplaying, resolves for a large part around gathering experience points and magical equipment to improve the main character. As game designer Ernest Adams points out in his satirical ‘letter from a dungeon’, there seems to be little purpose behind these mechanics, resulting in a shallow representation of character growth as a faint echo of the mythical

**LIVING STATUE  
GETTING ATTACKED  
BY A BUNCH OF  
CHICKENS. AND THEN**



- Tracery
  - Text/string generator
  - Context-free grammar-based with some extra bits
  - Adds ability to set variables

<http://tracery.io/>

# Distribution



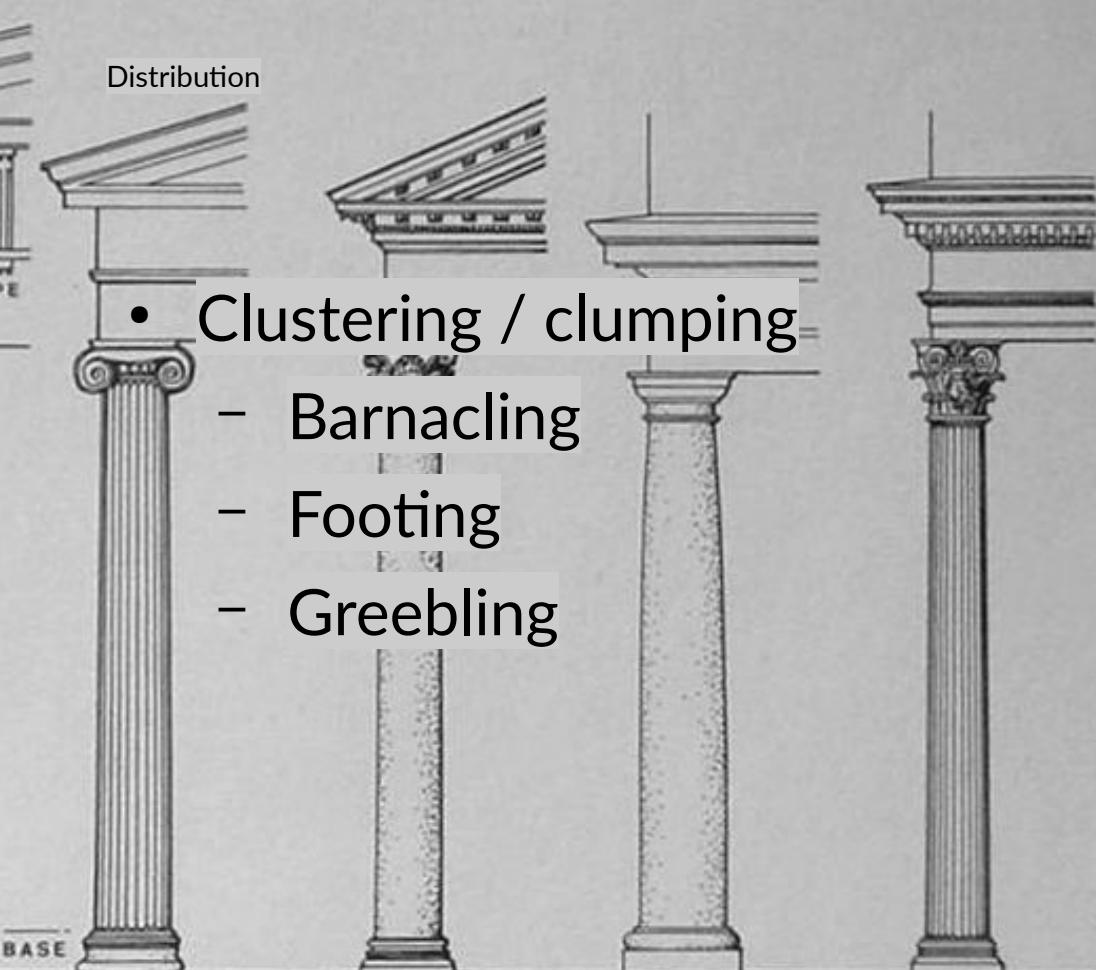
- Distribution
  - Putting stuff in the world
  - Pure randomness is unrealistic
  - Don't require much structure



Distribution

- Clustering / clumping

- Barnacling
- Footing
- Greebling



Ionic

Corinthian

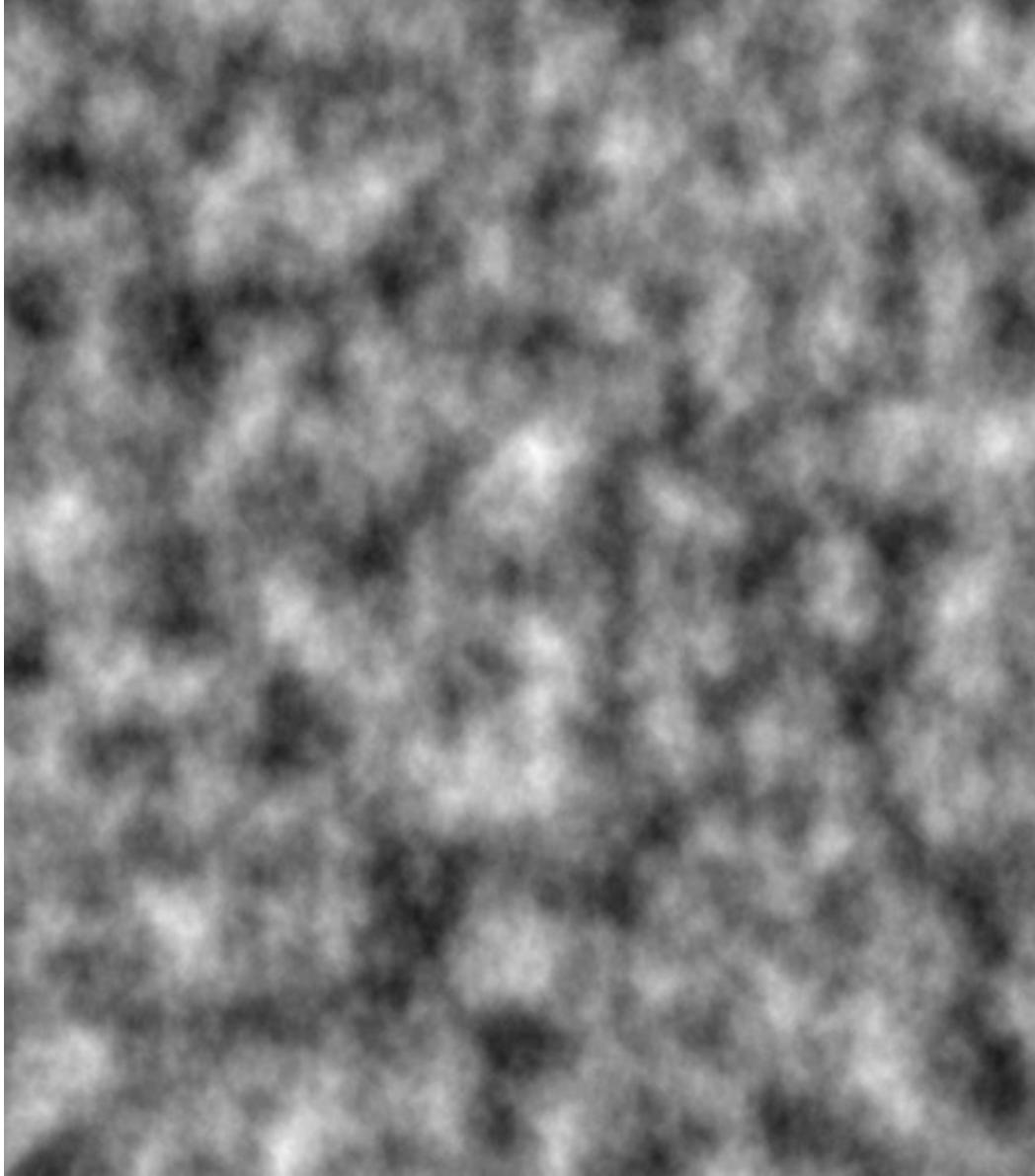
Tuscan

Composite



## Distribution

- Perlin noise
  - (Very!) Widely used
  - Other noises exist such as Simplex



Parametric



## Parametric

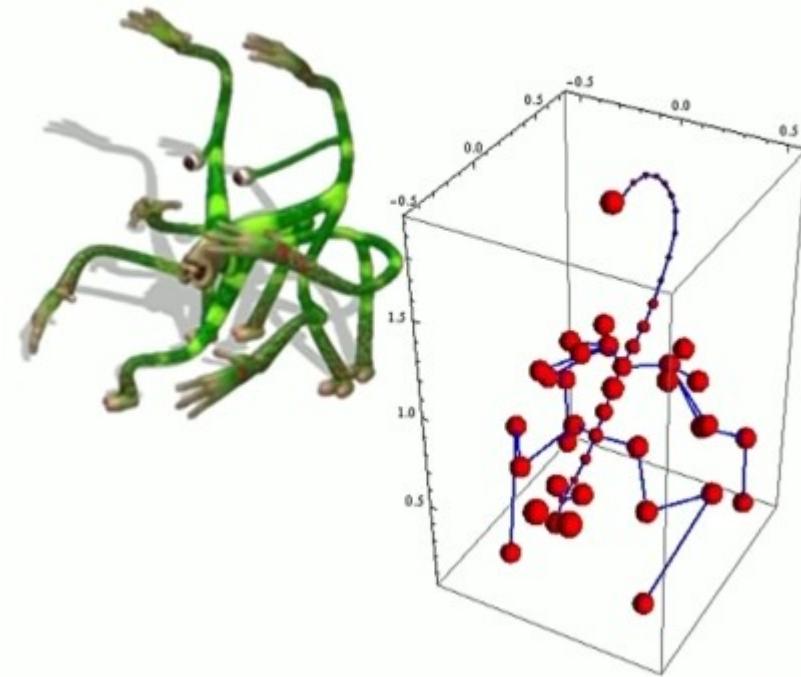
- Parametric
  - Define a multidimensional possibility space, defined by parameters
  - Structure of content is defined, but can vary on a number of dimensions





## Interpretive

- Interpretive
  - Take input data in one format and process it into another format
  - Exploit the underlying structure in data
  - Take a simple structure and make it more complex



Numpad Toggle: Off | Walking |

[Name Unknown]  
[No Known Nicknames]  
[No Known Titles]

27 June

## Simulations

**Head:**  
[No Helmet]  
[No Necklace]

**Torso:**  
**Religious Vestment**  
(Undamaged)  
[No Cloak]

**Left Arm:**  
[No Glove/Gauntlet] [No Glove/Gauntlet]  
[Nothing Held] [Nothing Held]  
[No Ring] [No Ring]

**Left Leg:**  
**Religious Vestment**  
(Undamaged)  
**Poor Boot**  
(Undamaged)

**Right Leg:**  
**Religious Vestment**  
(Undamaged)  
**Poor Boot**  
(Undamaged)

[TAB to change page, Enter to examine]

Stm: **███** Pai: **████** Air: **███**

This is a religious vestment she is currently wearing.



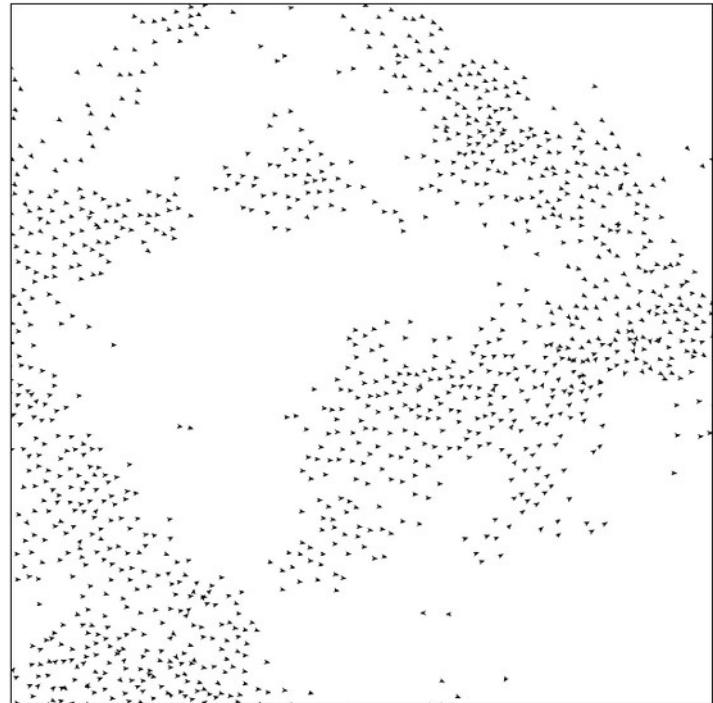
Left Arm  
Nothing  
Right Arm  
Nothing

- Agent-based Simulations
  - Generate content based on actions of in-world agents
  - Agents can be vary complex...
    - e.g. in Dwarf Fortress
  - Or basic...
    - Particle trails
    - Flocking

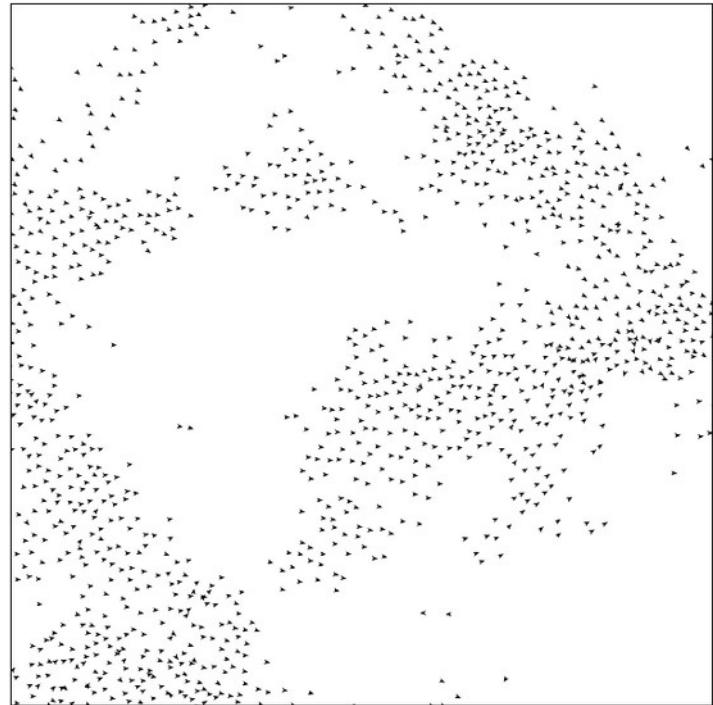




- Boids (Bird-like Objects)
  - Bird-like flocking behaviour
  - Result of each agent following three rules



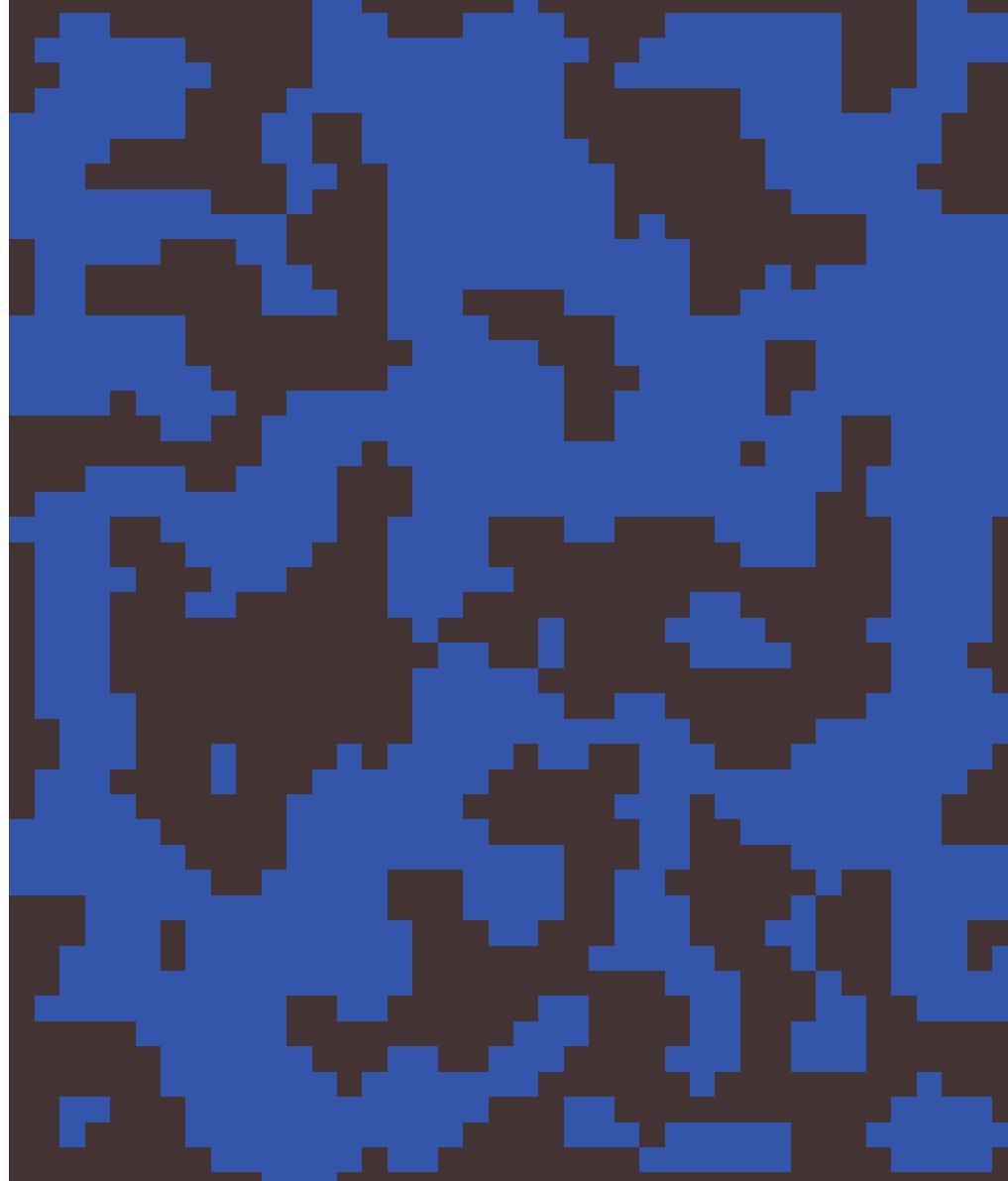
- Avoid
  - Steer away from nearby boids
- Align
  - Steer toward heading of nearby boids
- Approach
  - Steer toward centre of flock



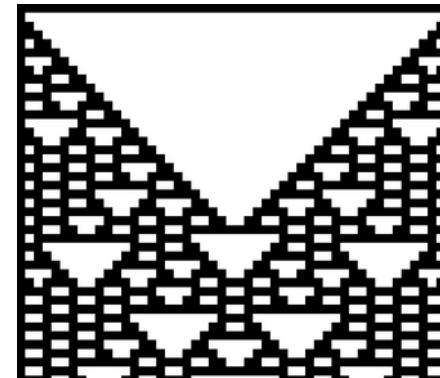
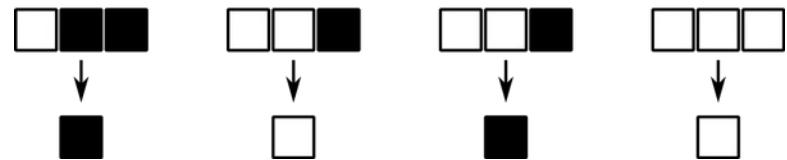
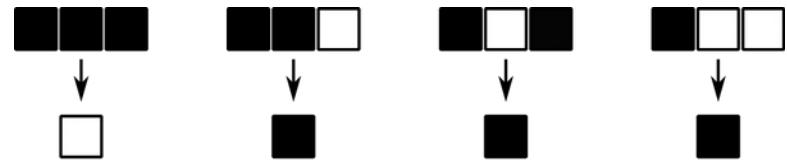
<https://www.harmendeweerd.nl/boids/>

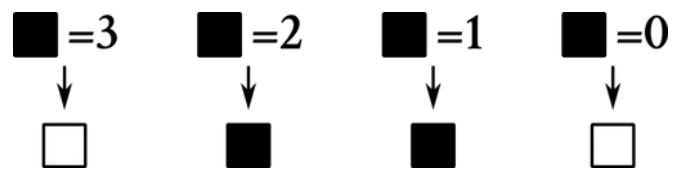
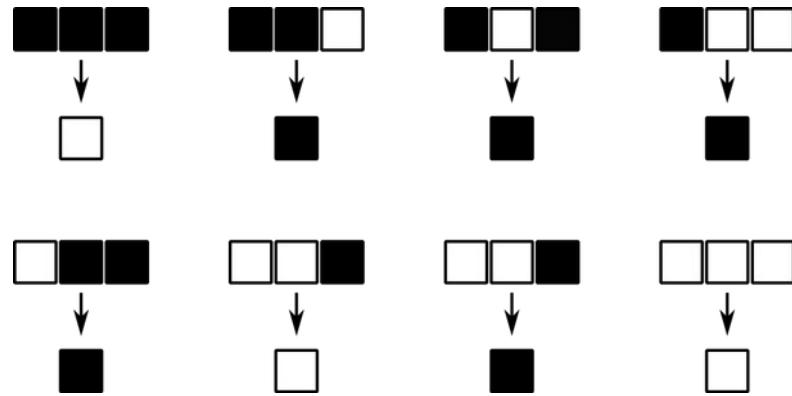
## Simulations

- Cellular Automata
  - Simulate state of cells based on states of neighbours
  - For natural-looking forms such as caves, islands, etc.



- 1D Cellular Automata
  - Each cell is determined by the cells above it





 = 3  
↓  


 = 2  
↓  


 = 1  
↓  


 = 0  
↓  



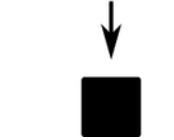

$\blacksquare = 3$



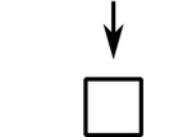
$\blacksquare = 2$



$\blacksquare = 1$



$\blacksquare = 0$





$\blacksquare = 3$



$\blacksquare = 2$



$\blacksquare = 1$



$\blacksquare = 0$





$\blacksquare = 3$



$\blacksquare = 2$

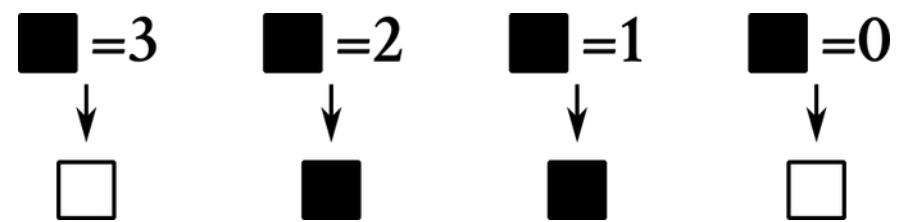


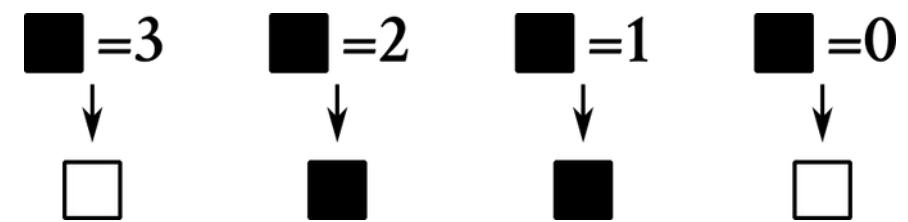
$\blacksquare = 1$

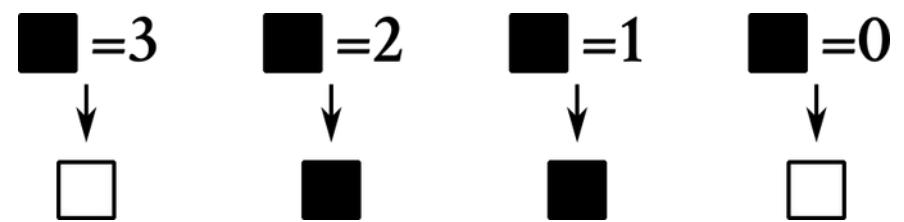


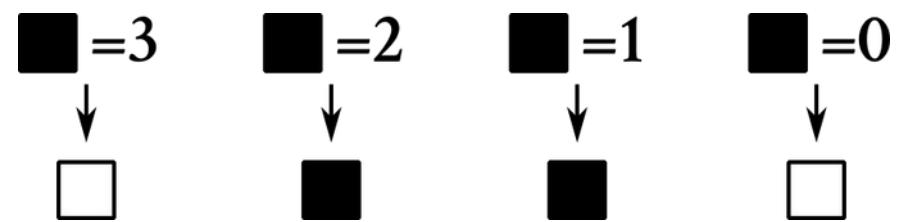
$\blacksquare = 0$

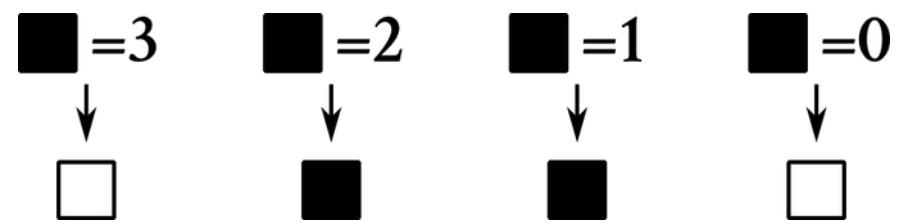


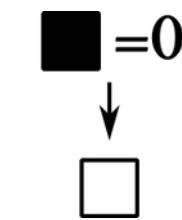
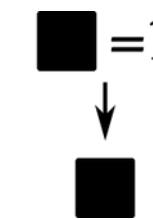
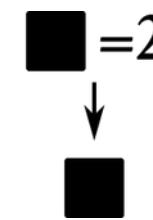
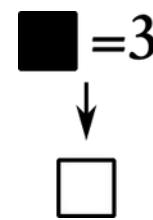
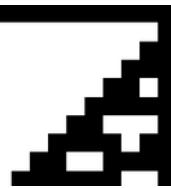
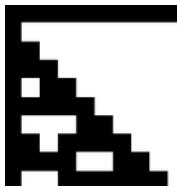


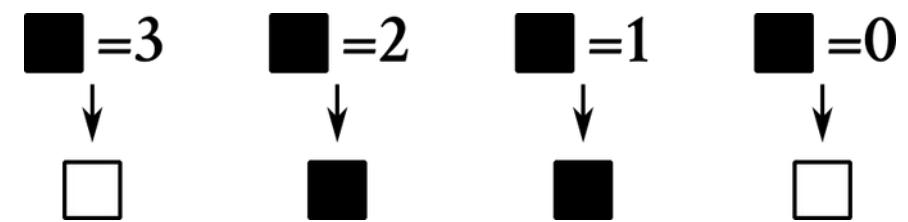


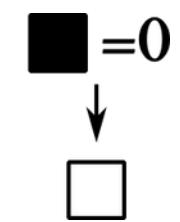
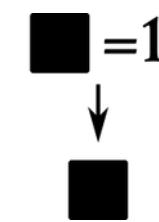
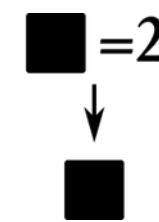
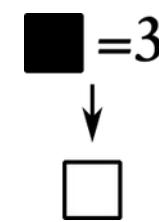


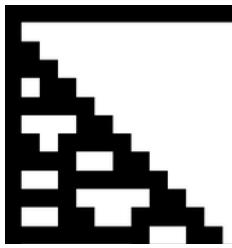












$\blacksquare = 3$   
↓  
 $\square$

$\blacksquare = 2$   
↓  
 $\blacksquare$

$\blacksquare = 1$   
↓  
 $\blacksquare$

$\blacksquare = 0$   
↓  
 $\square$



$\blacksquare = 3$



$\blacksquare = 2$

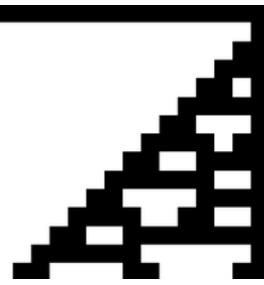
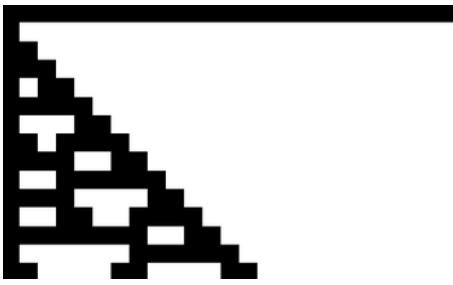


$\blacksquare = 1$



$\blacksquare = 0$





$\blacksquare = 3$



$\blacksquare = 2$

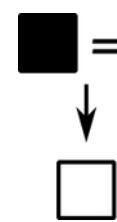
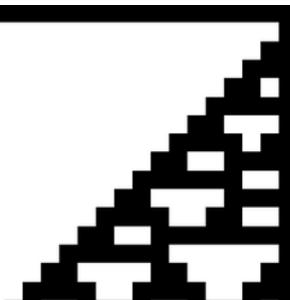
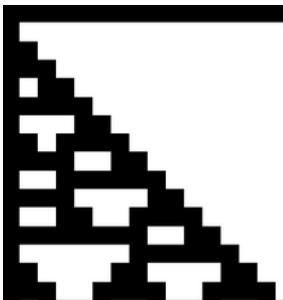


$\blacksquare = 1$



$\blacksquare = 0$





$\blacksquare = 3$



$\blacksquare = 2$



$\blacksquare = 1$



$\blacksquare = 0$





$\blacksquare = 3$



$\blacksquare = 2$

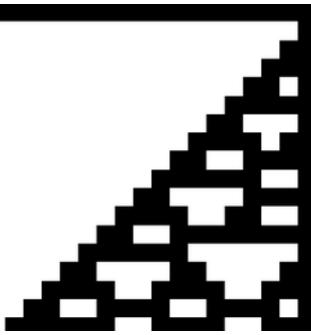
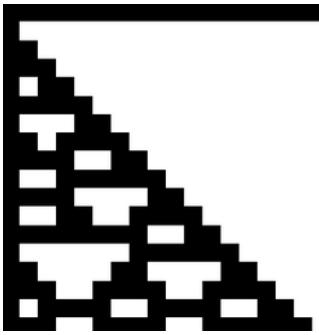


$\blacksquare = 1$



$\blacksquare = 0$





$\blacksquare = 3$



$\blacksquare = 2$

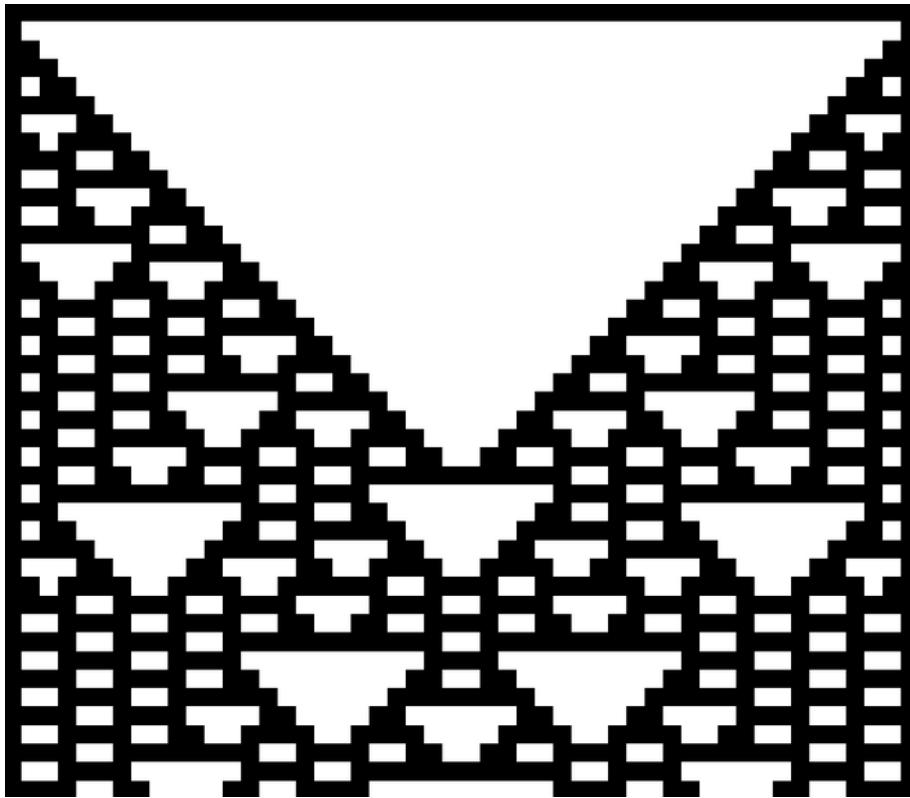


$\blacksquare = 1$



$\blacksquare = 0$





$\blacksquare = 3$



$\blacksquare = 2$



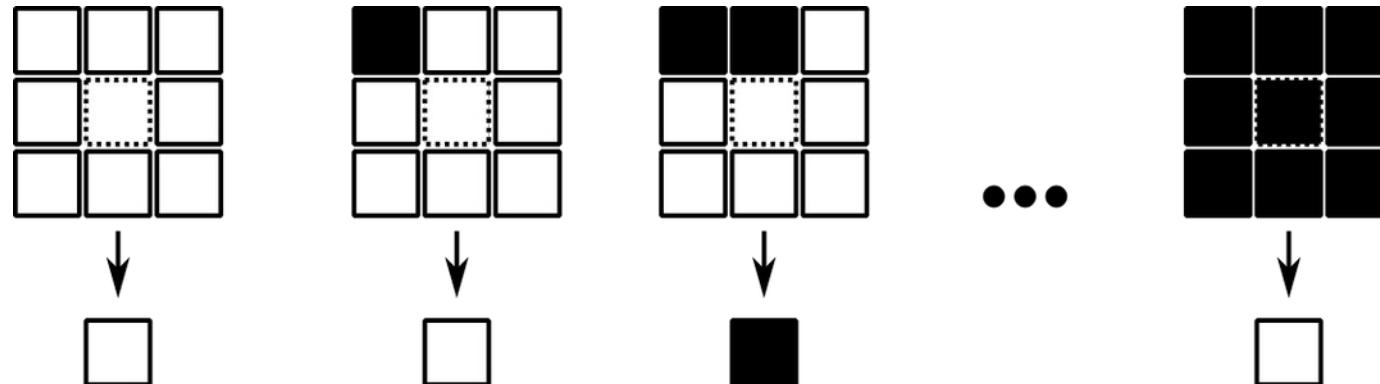
$\blacksquare = 1$



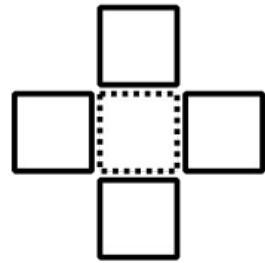
$\blacksquare = 0$



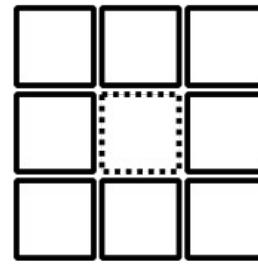
- 2D Cellular Automata
  - Each cell is determined by the cells around it (it's “neighbourhood”)



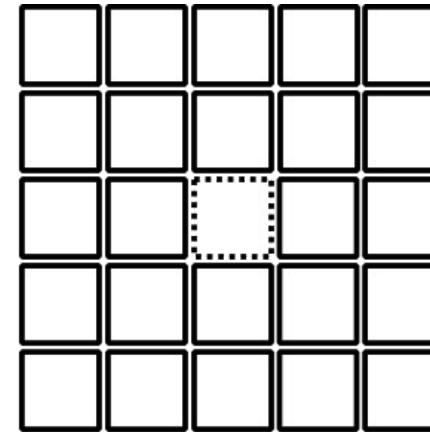
## Simulations



5 cell von Neumann  
neighbourhood

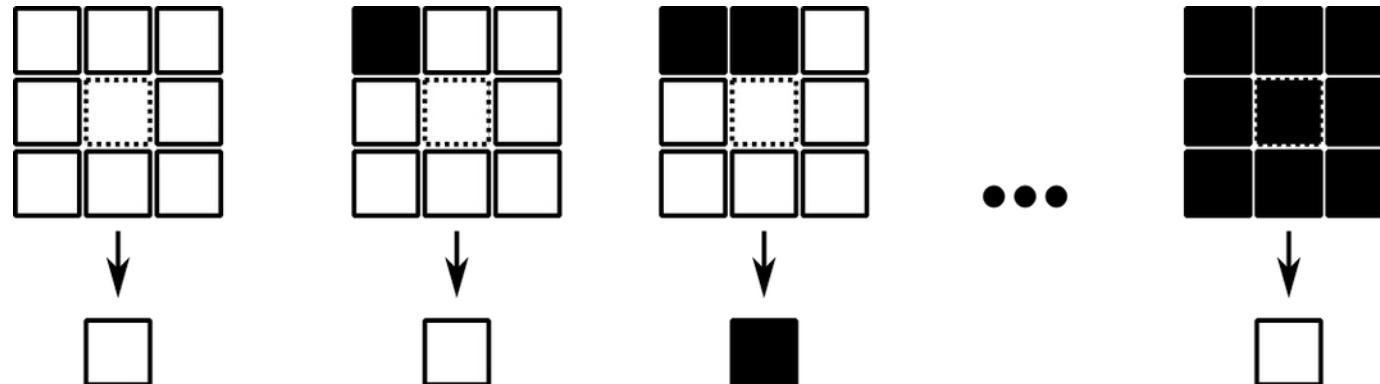


3x3 Moore  
neighbourhood



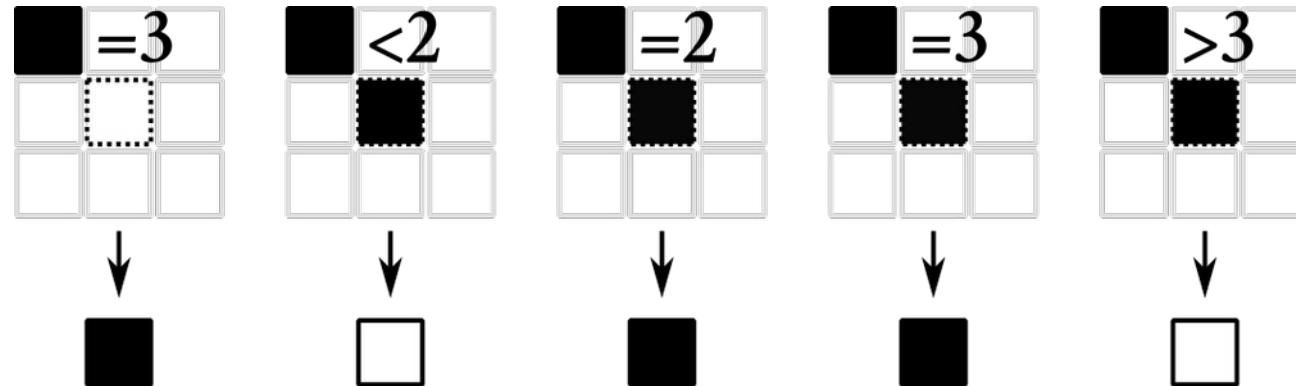
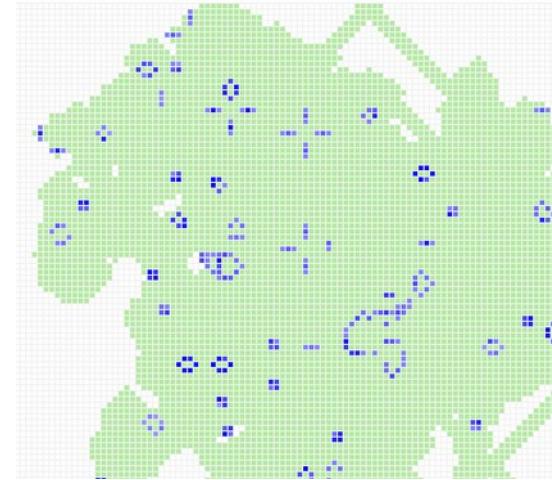
5x5 Moore  
neighbourhood

- 2D Cellular Automata
  - Each cell is determined by the cells around it (it's “neighbourhood”)



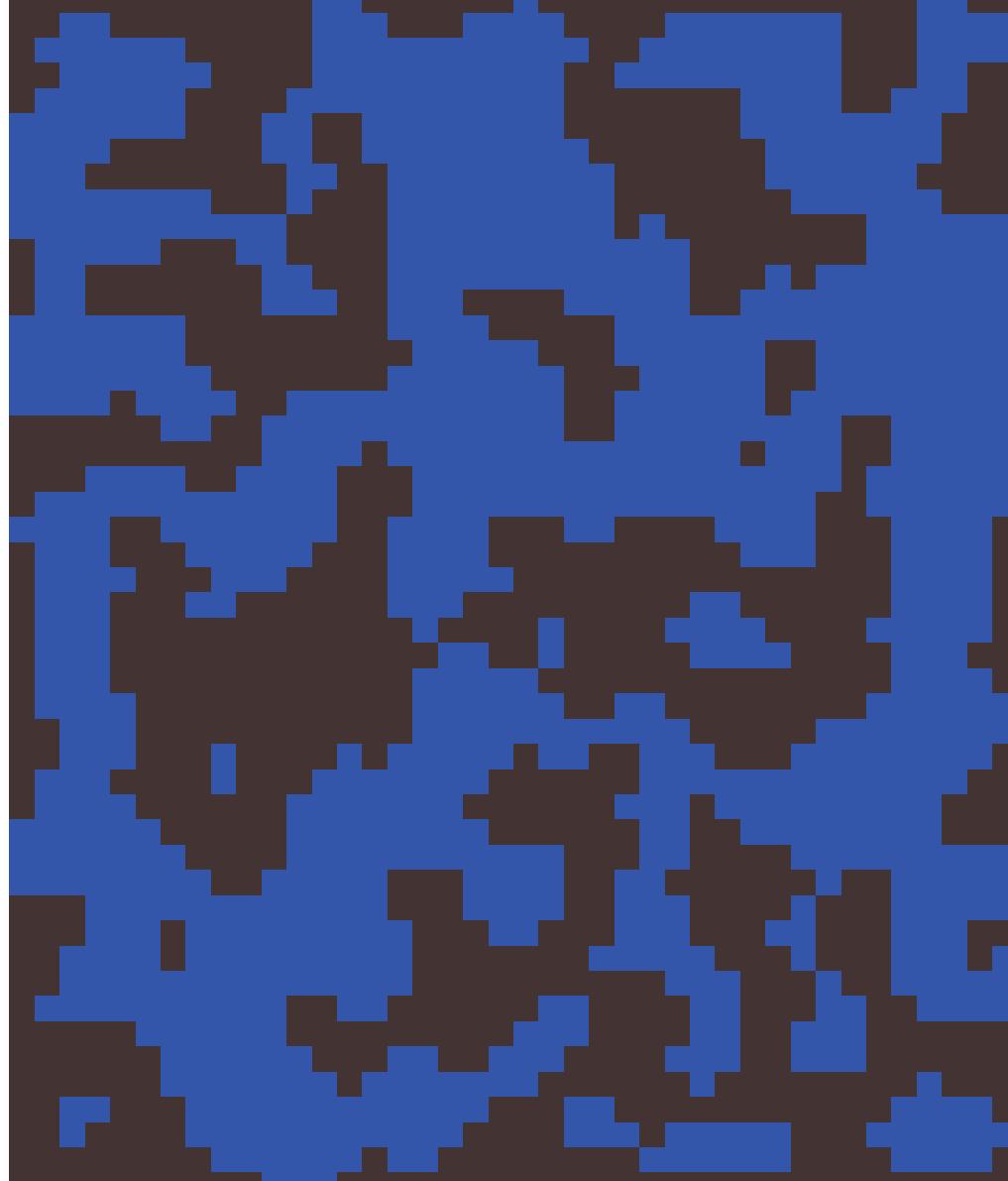
- The Game of Life (John Conway)

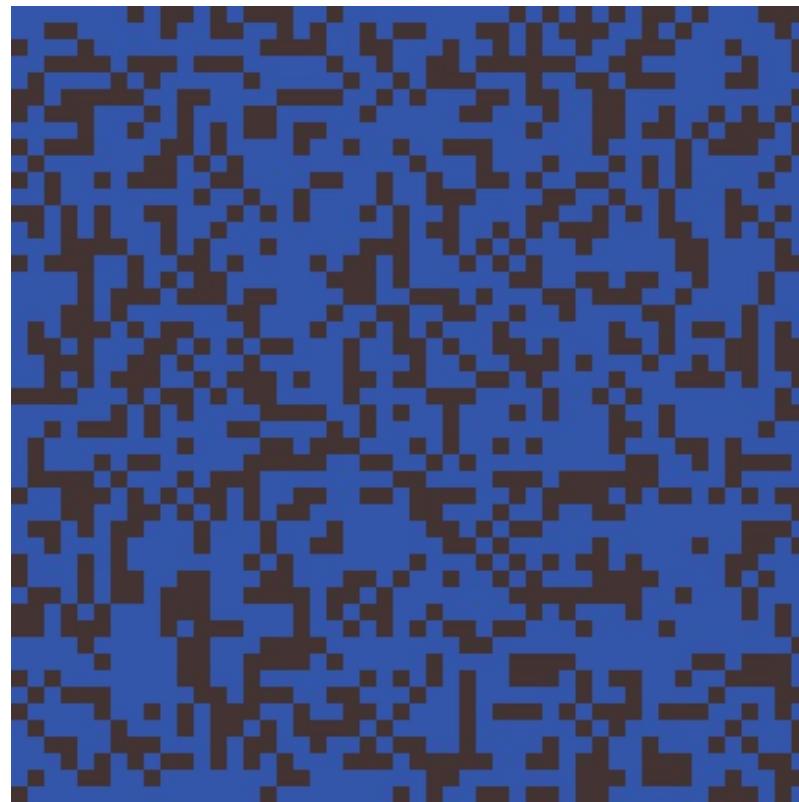
<http://pmav.eu/stuff/javascript-game-of-life-v3.1.1/>

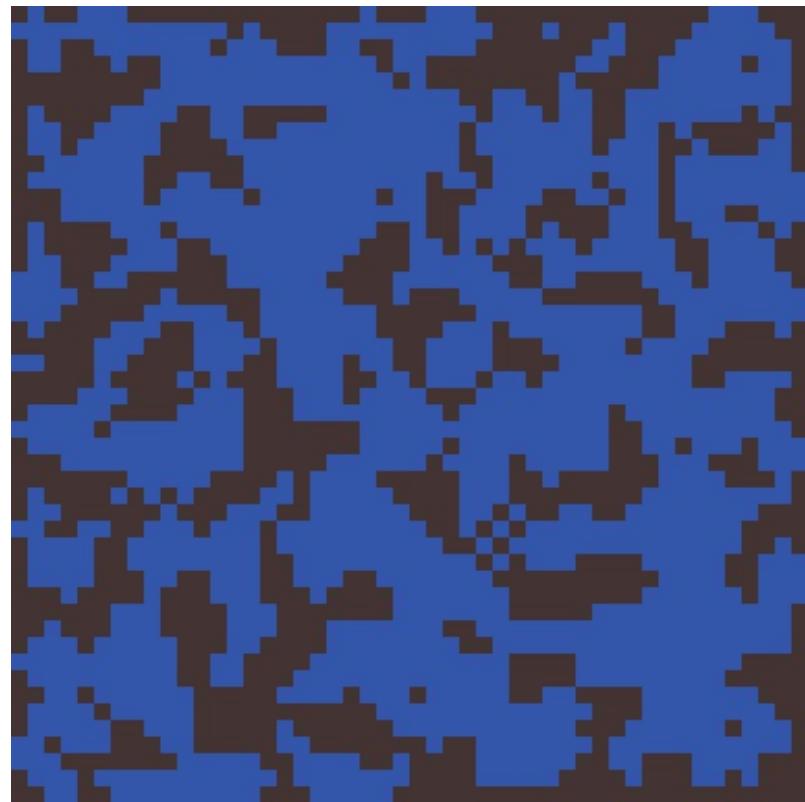


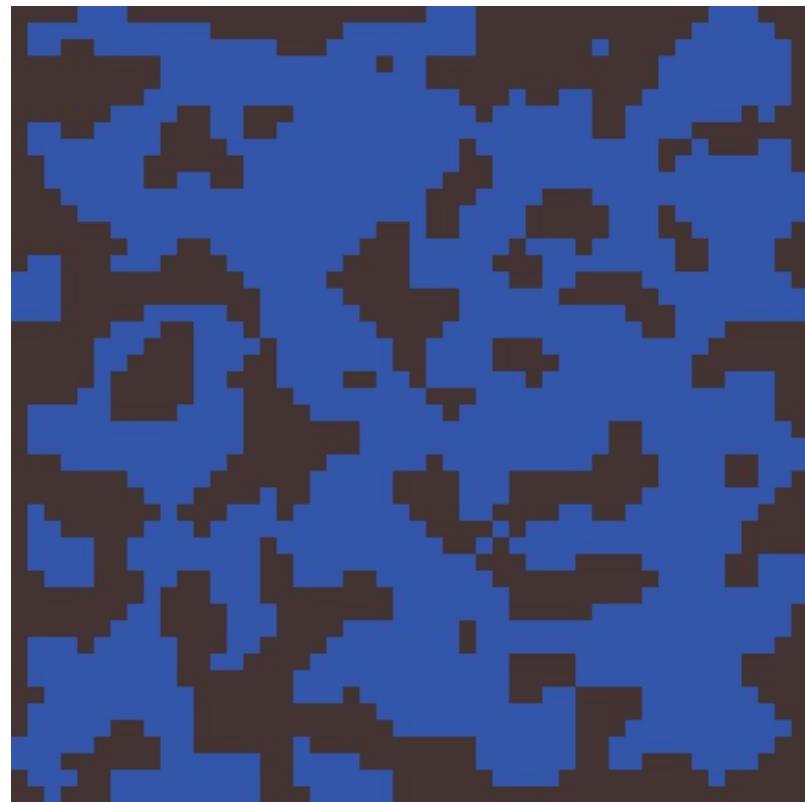
## Simulations

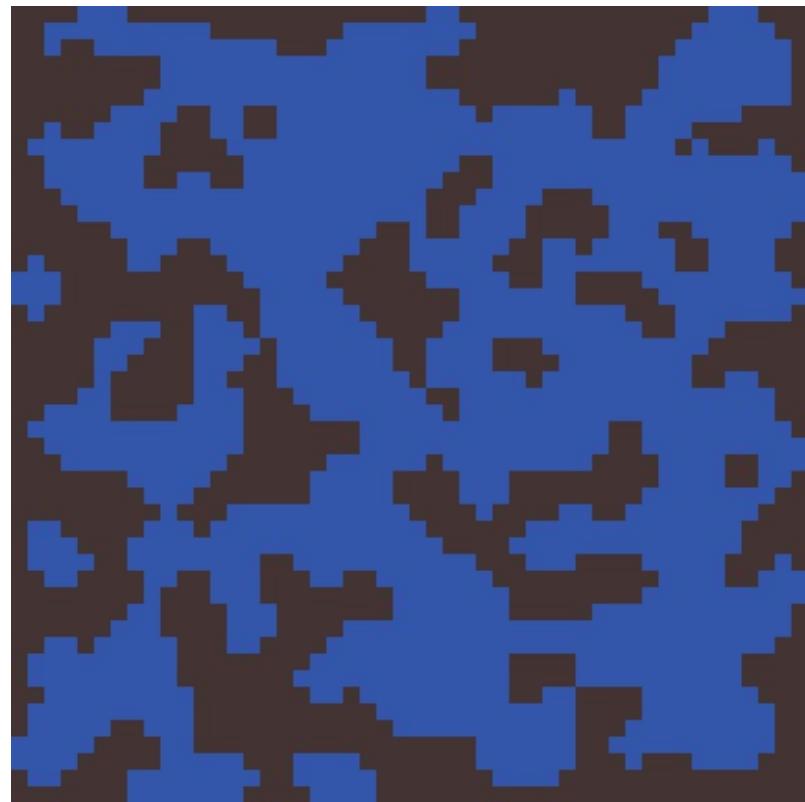
- Cellular automata can be used to make 2D and 3D maps

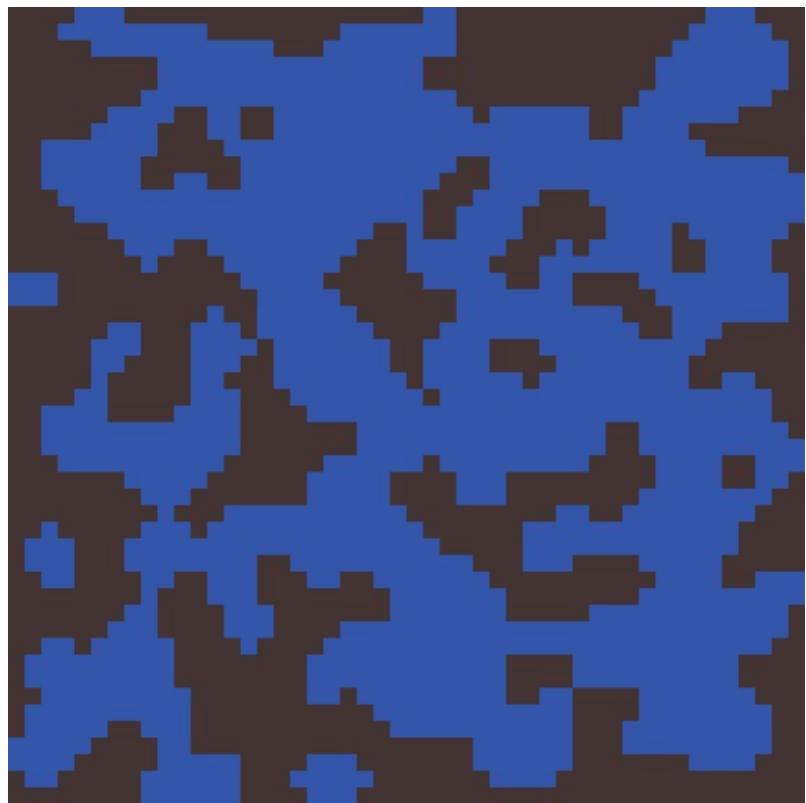


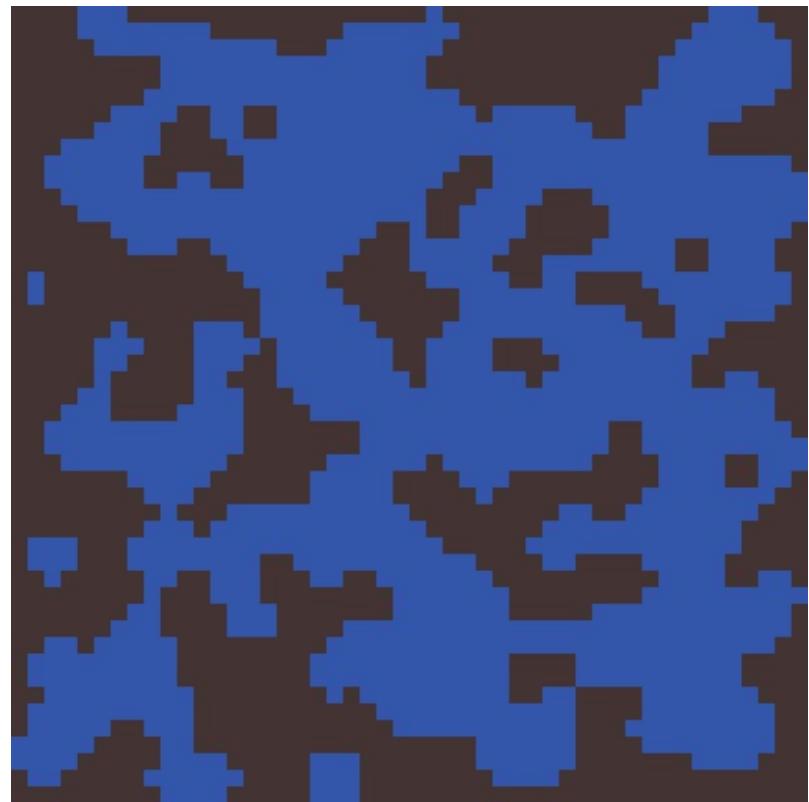


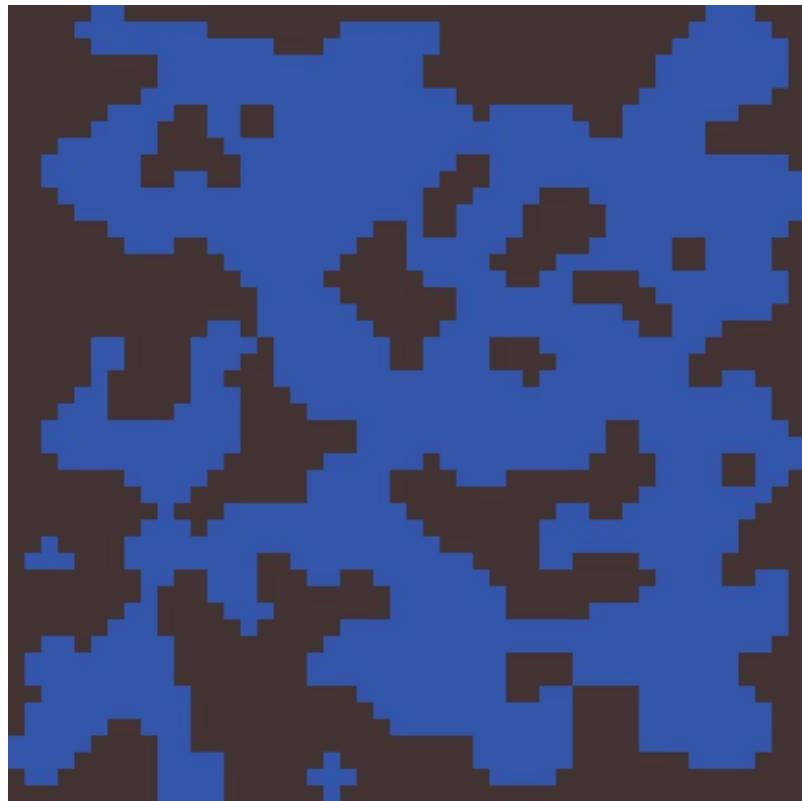


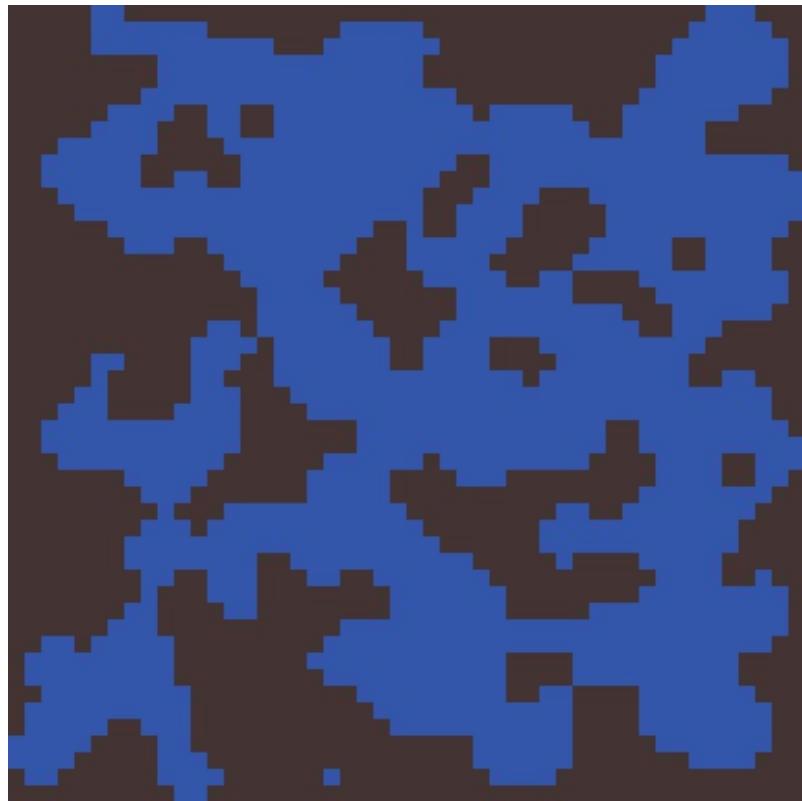


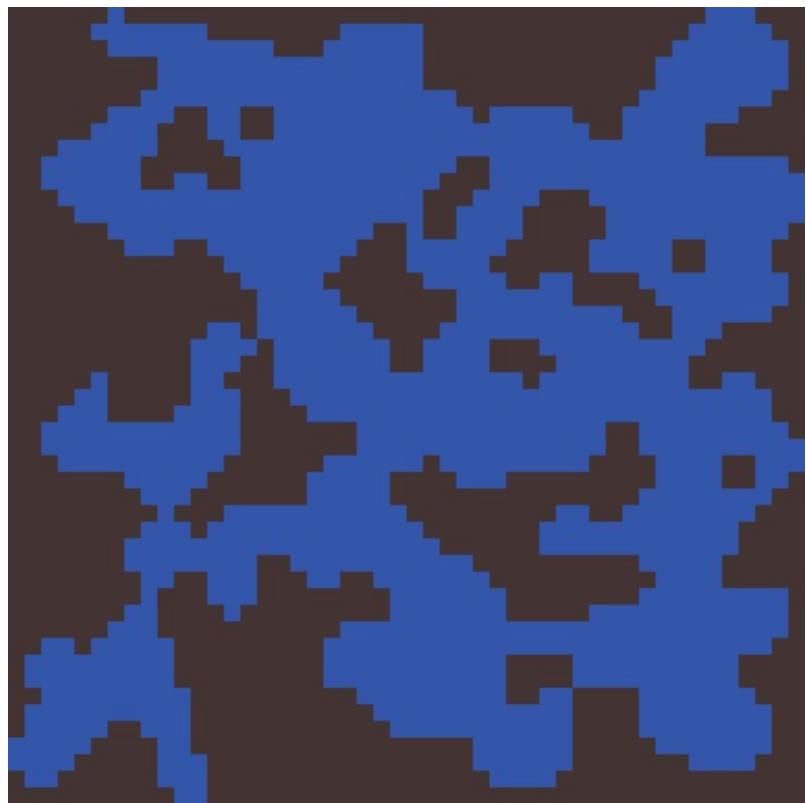






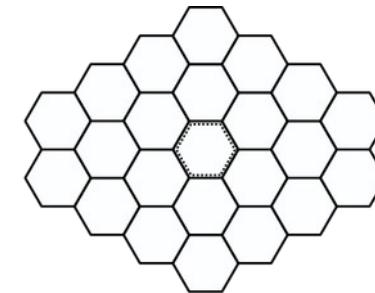
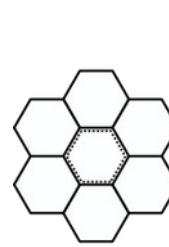
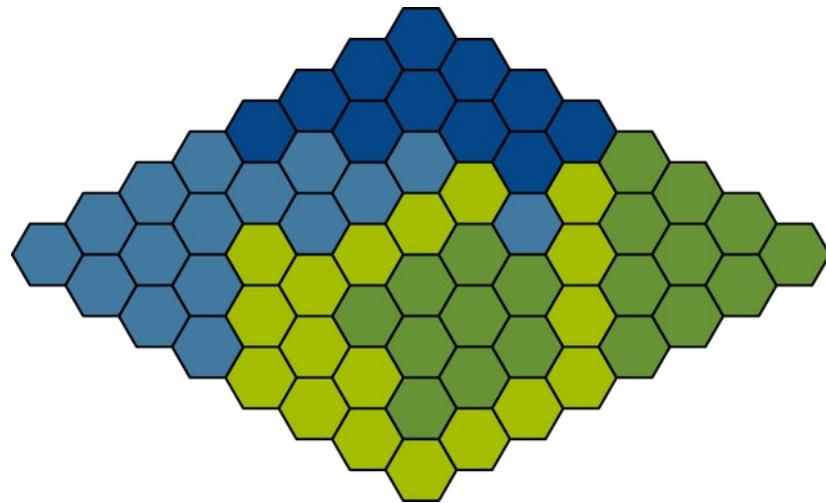
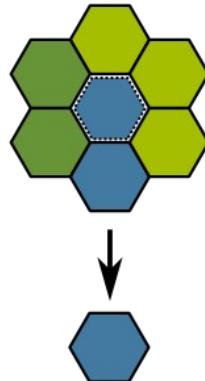
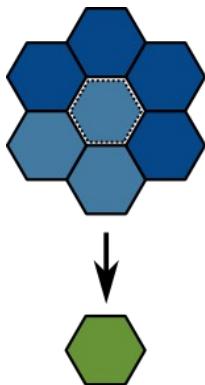






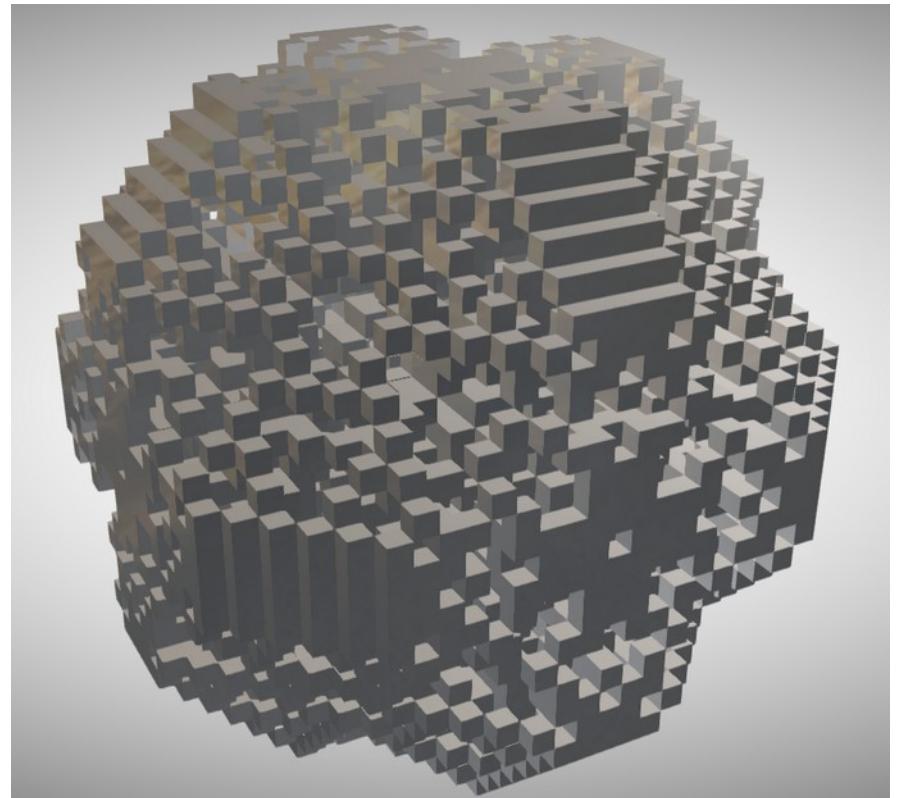
## Simulations

- Hexagonal grids
- More types of tile



## Simulations

- 3D?
- <http://www.cubes.io/>



- Basic implementation:
  - Randomly initialise map
  - Write a simulation step function
    - Count alive neighbours
    - Apply rules simultaneously to all tiles

```
float chanceToStartAlive = 0.45f;  
  
public boolean[][] initialiseMap(boolean[][] map){  
  
    for(int x=0; x<width; x++)  
  
        for(int y=0; y<height; y++){  
  
            if(random() < chanceToStartAlive)  
  
                map[x][y] = true;  
  
        }  
  
    return map;  
}
```

- Basic implementation:
  - Randomly initialise map
  - Write a simulation step function
    - Count alive neighbours
    - Apply rules simultaneously to all tiles

```
public doSimulationStep(boolean[][] oldMap){  
    boolean[][] newMap = new boolean[width]  
    [height];  
    //...
```

- Basic implementation:
  - Randomly initialise map
  - Write a simulation step function
    - Count alive neighbours
    - Apply rules simultaneously to all tiles

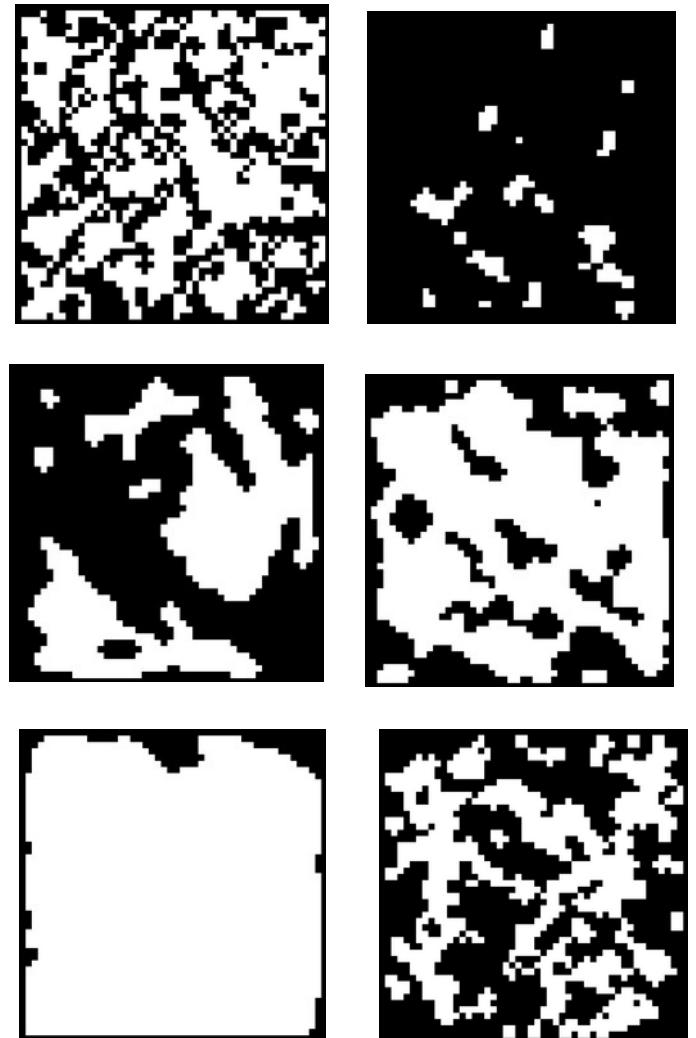
```
public countAliveNeighbours(boolean[][] map,  
int x, int y){  
  
    int count = 0;  
  
    for(int i=-1; i<2; i++){  
        for(int j=-1; j<2; j++){  
            int neighbour_x = x+i;  
            int neighbour_y = y+j;  
            if(i == 0 && j == 0) {}  
            else if(neighbour_x < 0 || neighbour_y  
< 0 || neighbour_x >= map.length ||  
neighbour_y >= map[0].length)  
                count = count + 1;  
            else if(map[neighbour_x][neighbour_y])  
                count = count + 1;  
        }  
    }  
}
```

- Basic implementation:
  - Randomly initialise map
  - Write a simulation step function
    - Count alive neighbours
    - **Apply rules simultaneously to all tiles**

```
public boolean[][] doSimulationStep(boolean[][] oldMap){  
    boolean[][] newMap = new boolean[width][height];  
  
    for(int x=0; x<oldMap.length; x++){  
        for(int y=0; y<oldMap[0].length; y++){  
            int nbs = countAliveNeighbours(oldMap, x, y);  
  
            if(oldMap[x][y]){  
                if(nbs < deathLimit) newMap[x][y] = false;  
                else newMap[x][y] = true;  
            }  
            else{  
                if(nbs > birthLimit) newMap[x][y] = true;  
                else newMap[x][y] = false;  
            }  
        }  
    }  
  
    return newMap;  
}
```

## Simulations

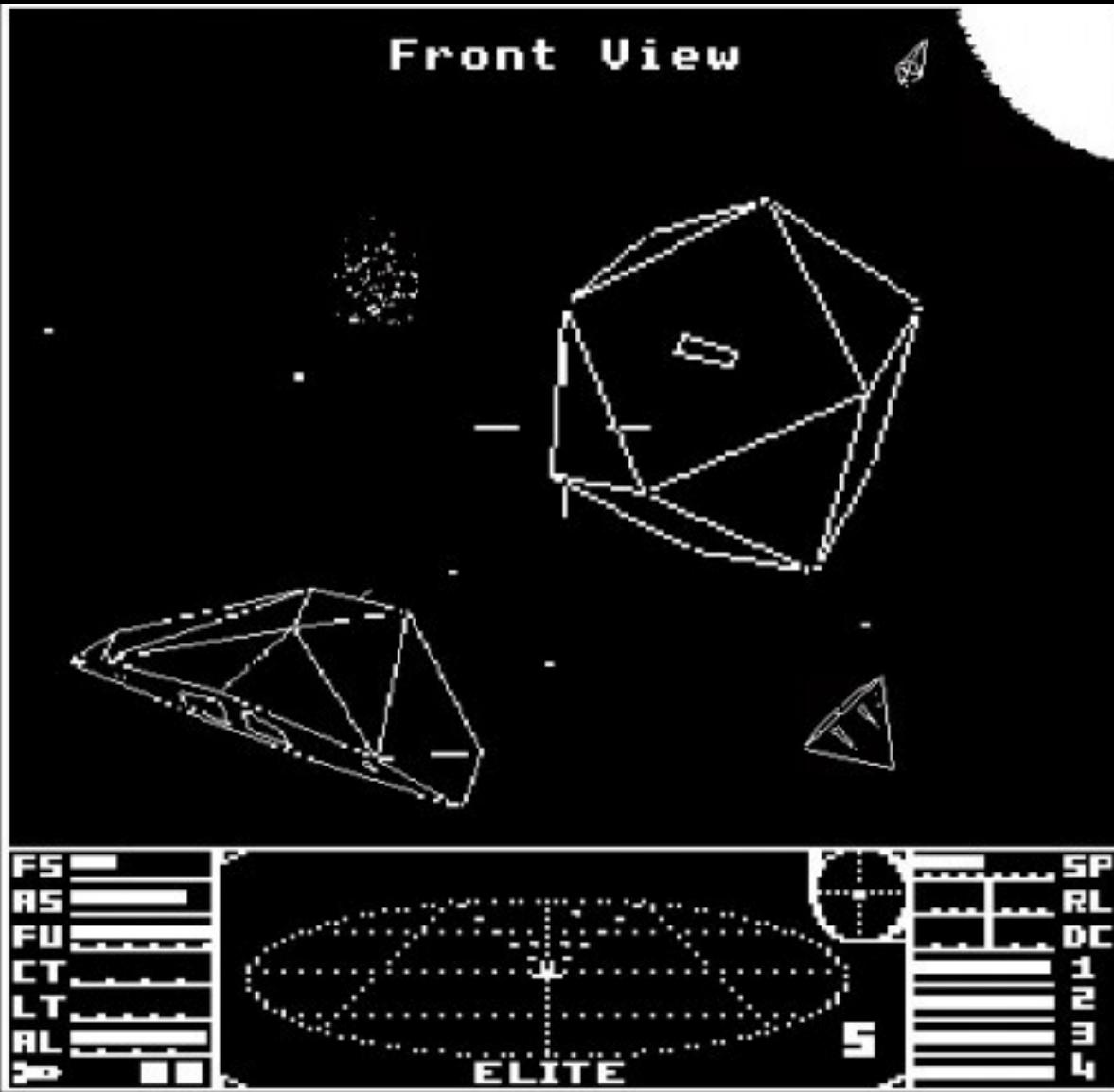
- Challenges
  - Neighbourhood type
  - Initialising map (what % of each tile?)
  - What rules to use?
  - How many iterations?
  - Hard to control



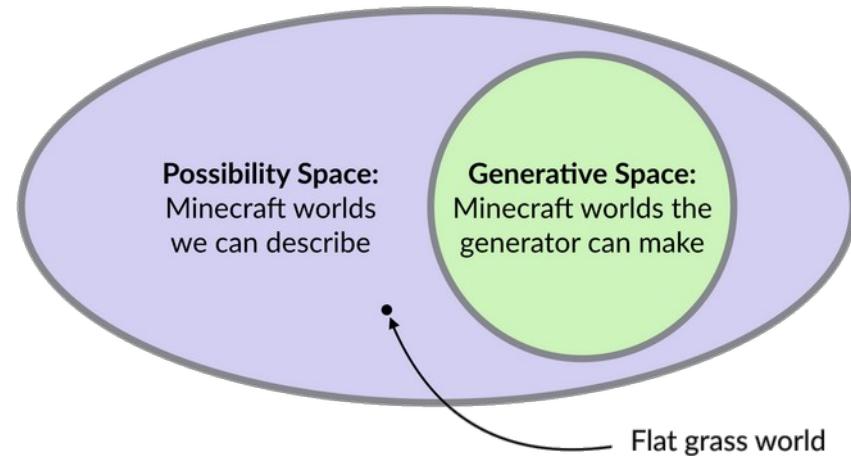
- Games
  - Tombs of Temeria
  - Luminous Corridor 0



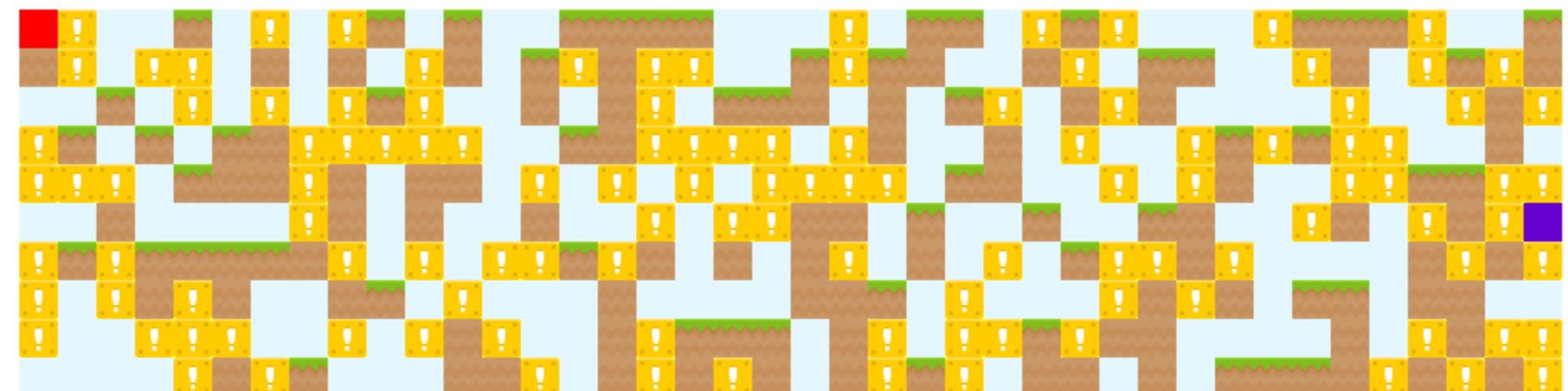
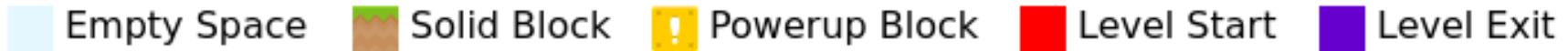
Using PCG



- What is PCG?
  - Content representation defines **possibility space**
    - i.e. the total set of things that can be defined
  - Algorithm defines **generative space**
    - i.e. the total set of things that can be generated by that algorithm



Level Tile Key



Current Generator: Random Generator

Press R or click to generate a new level

Level Tile Key

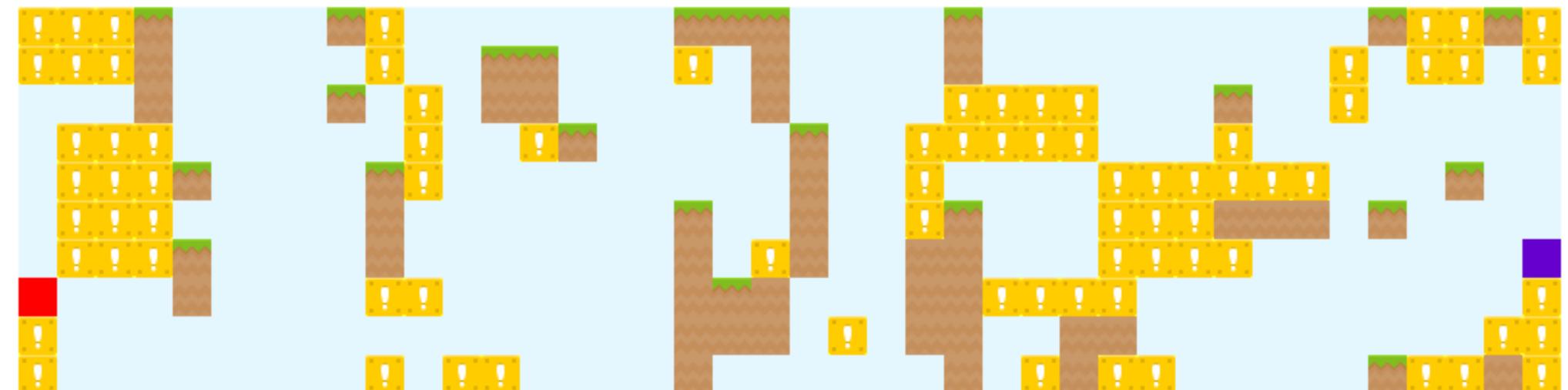
Empty Space

Solid Block

Powerup Block

Level Start

Level Exit



Current Generator: Shape-Based Generator [Large]

Press R or click to generate a new level

Level Tile Key



Empty Space



Solid Block



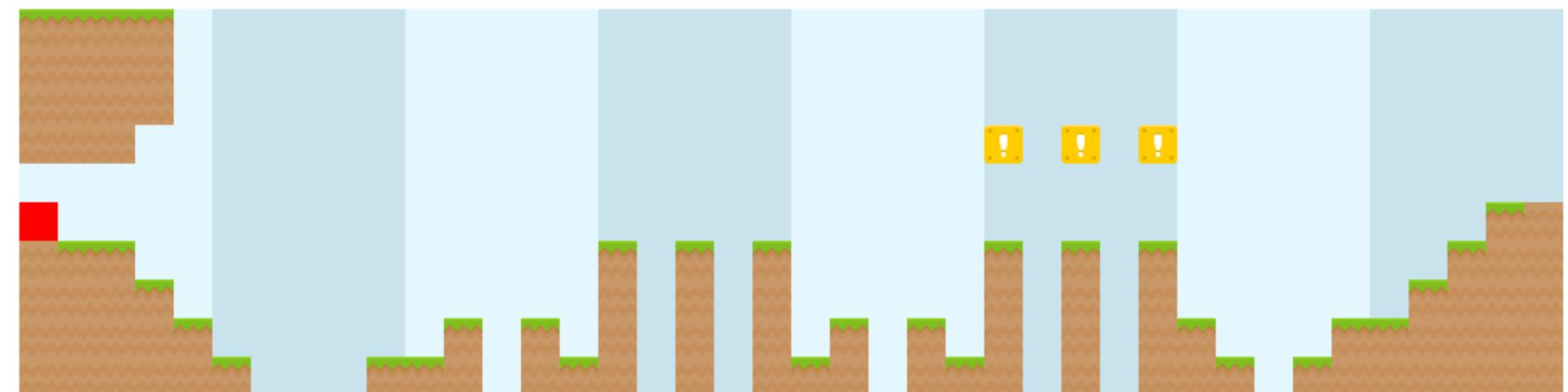
Powerup Block



Level Start



Level Exit



Current Generator: Chunk-Based Generator

Press R or click to generate a new level

- We don't necessarily want a large generative space!
  - We want to be able to generate *good* content
  - We might want to be able to guarantee certain properties
- How is our generator being used?
  - Within our game? (**Online**)
  - As a tool to support development? (**Offline**)

- Generator requirements
  - Functional
    - e.g. Playable
    - Traversable
  - Aesthetics
    - Looks good
    - Gives the desired player experience

- **Constructive**
  - Bake in requirements to algorithm
  - e.g. constraint satisfaction (e.g. ASP)
- **Generate and test**
  - Visualisation
  - AI (e.g. neural network)
  - Human Players

- Answer Set Programming (ASP)
  - Declarative programming language
  - Models the problem domain and expected result

```

1 :- 2 { place_object( _, X, Y, Z) , place_object( _, X, Y, Z) } ,
      floor(X, Y, Z) .
2 :- place_object( _, X, Y, Z) , not floor(X, Y, Z) .
3 :- button(b(X)) , not connected( b(X) ,
      c(N)) : potential_cube(c(N)) , not connected( b(X) , x) .
4
5 :- connected( O1,O) , connected( O2,O) , O1 != O2 ,
      object(O;O1;O2) .
6 :- connected( O,O1) , connected( O,O2) , O1 != O2 ,
      object(O1;O2;O) .

```

Listing 4.5: Encoding some of the hard constraints

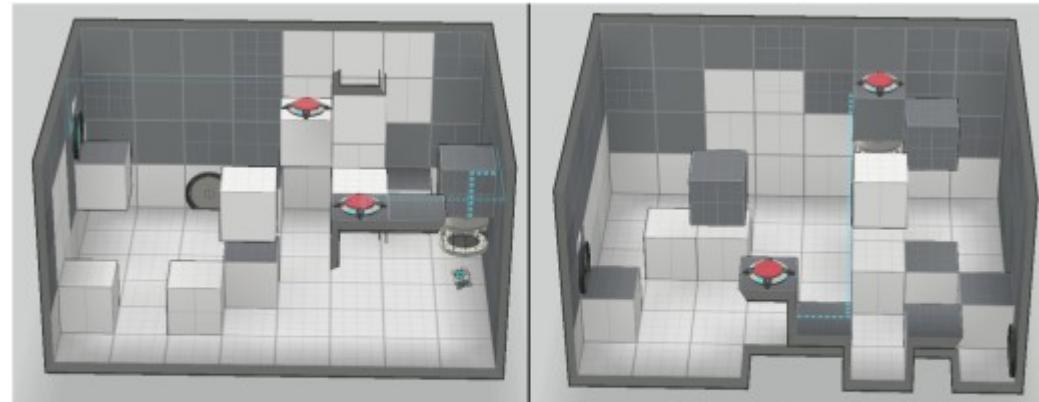
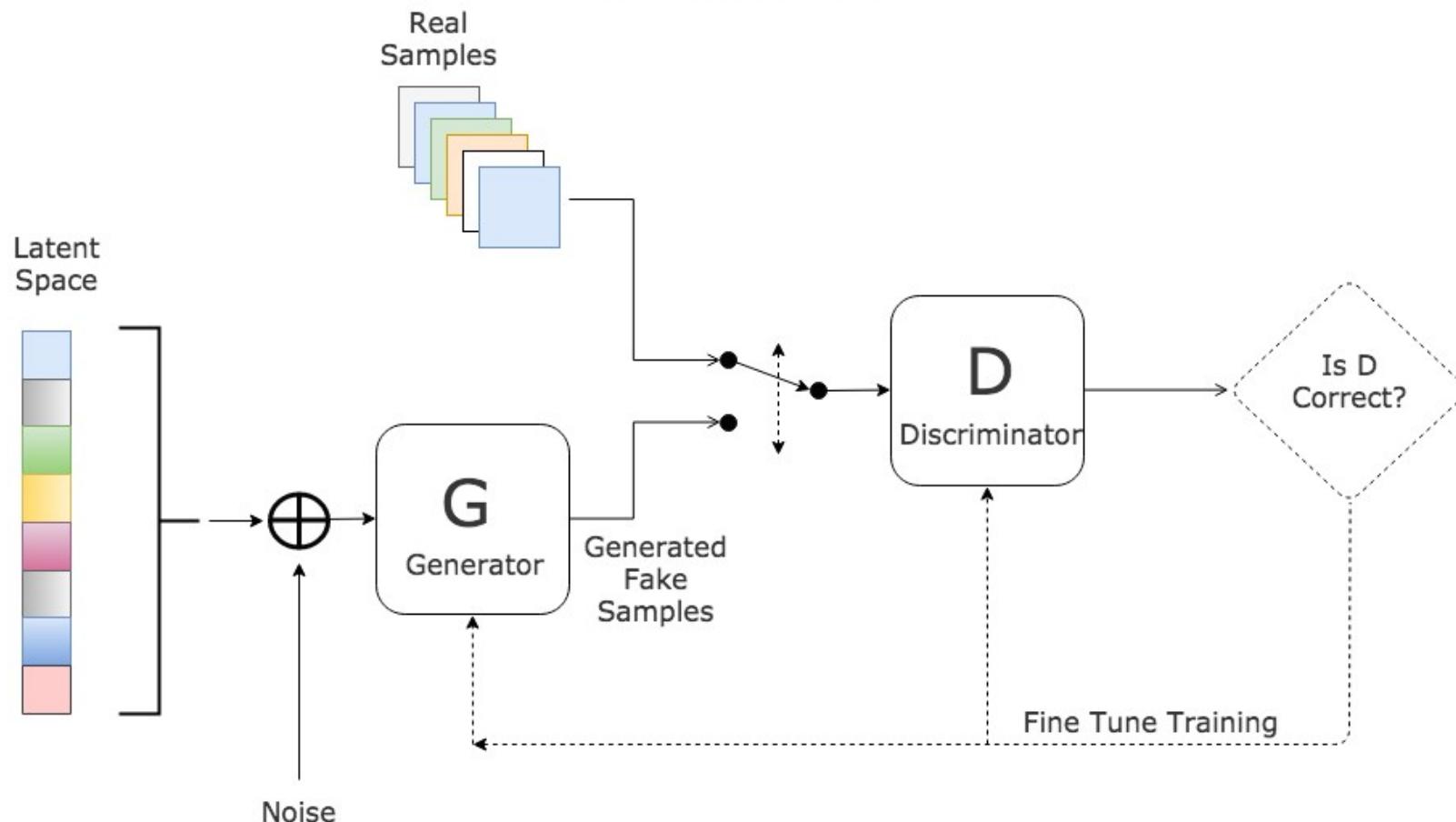


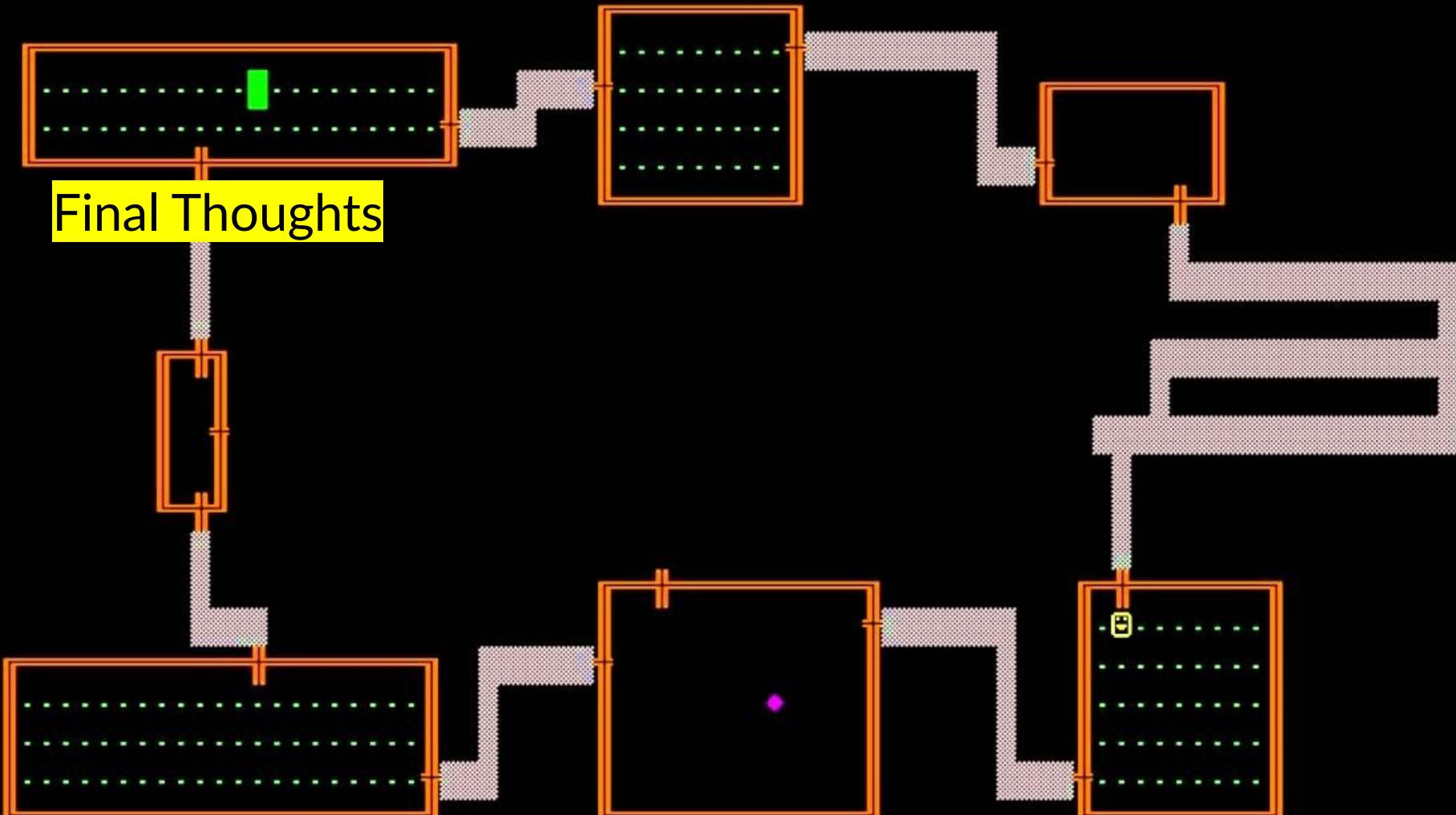
Figure 5.1: Two levels with reduced proportion of solid boxes generated by the simplified encoding.

# Generative Adversarial Network



- PCG as search
  - **Content encoding** defines space
  - **Content function** renders the content encoding
  - **Content requirements** defines constraints to be satisfied/optimised
  - Search algorithm explores this space

13

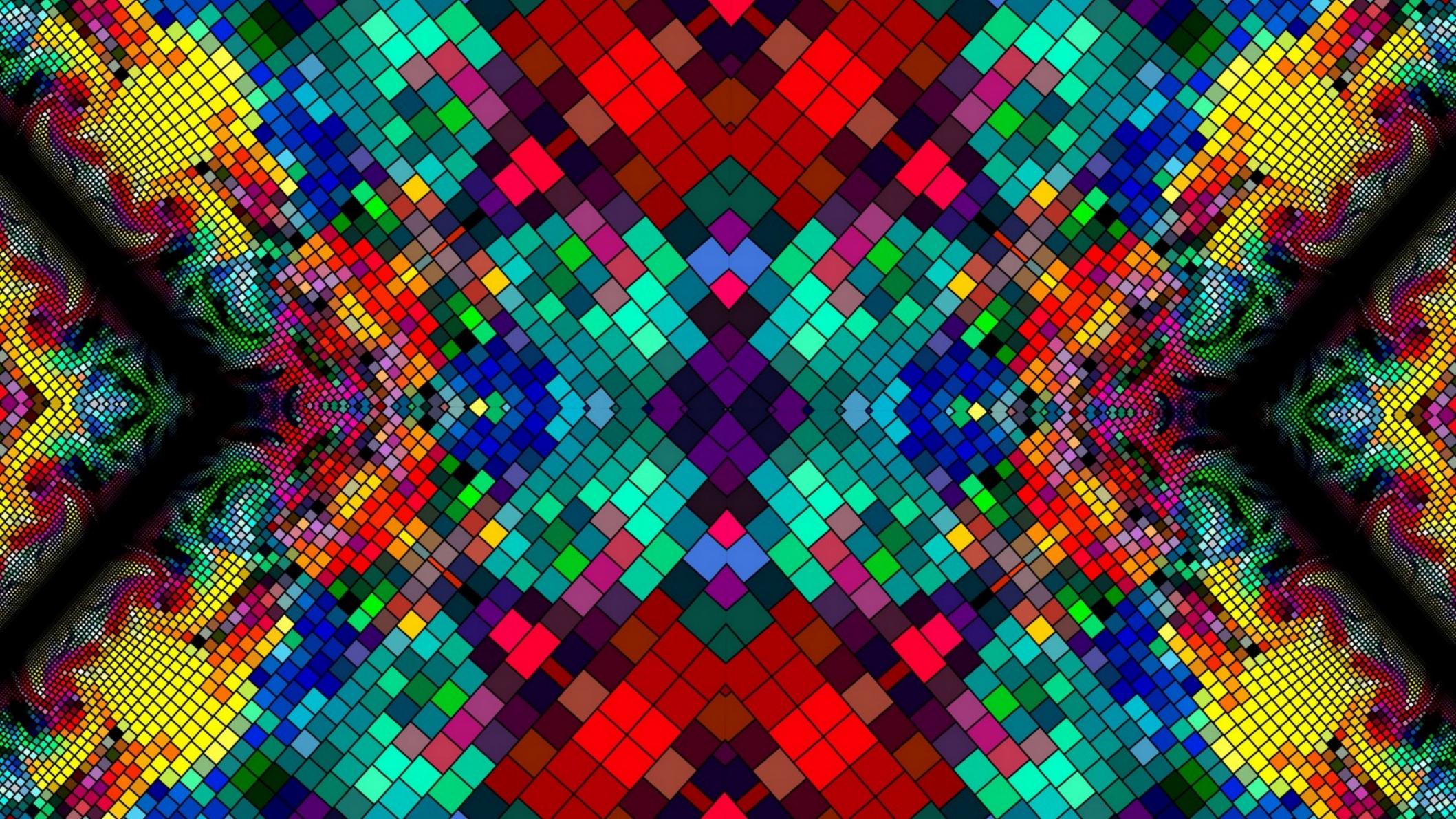


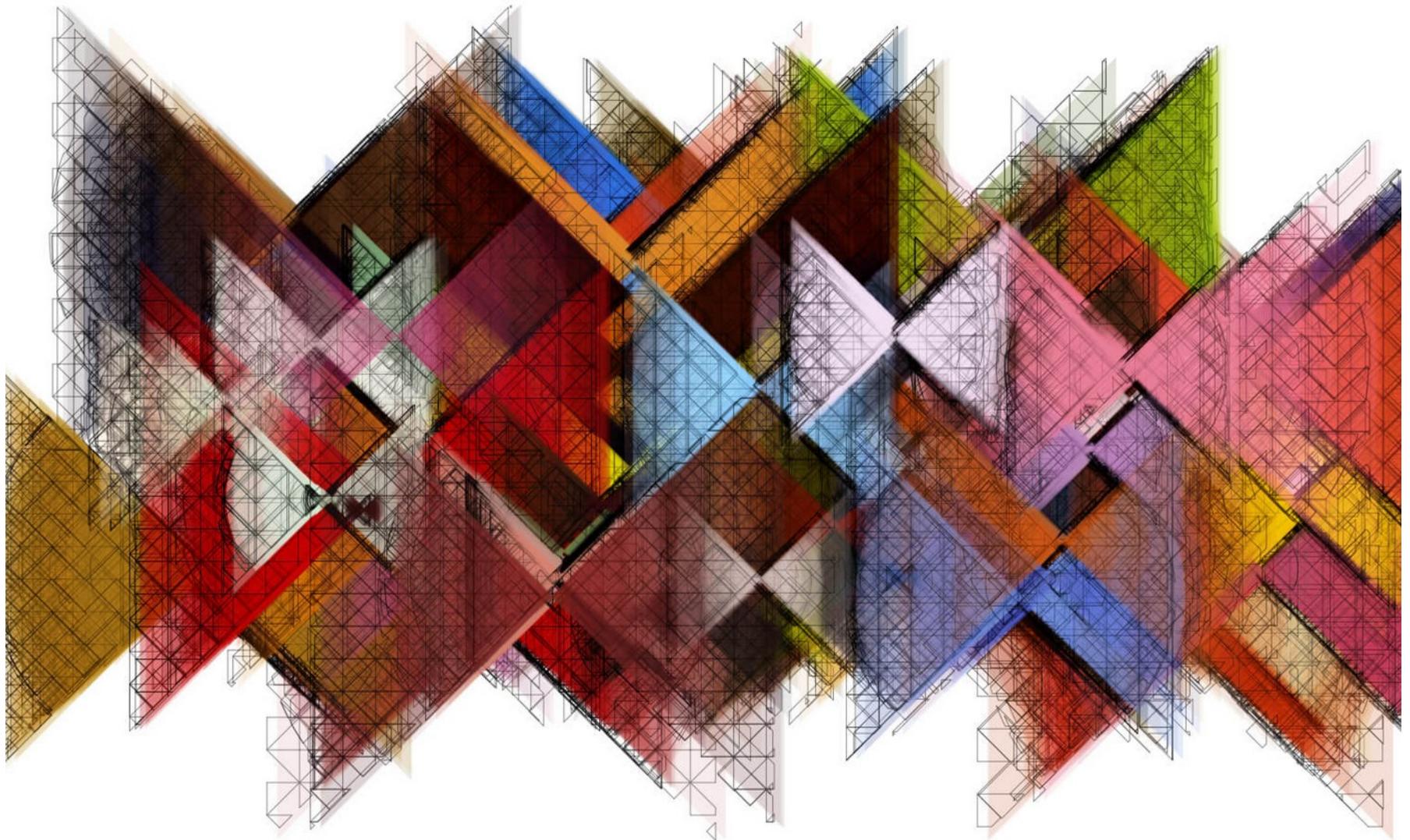
- Whitelisting good generated content by seed might be cheaper than making a robust generator
- Save seeds, make seed-based random generators
- Iterating is important
- Unexpected stuff happens, sometimes following what your generator is good at gives cool unexpected results

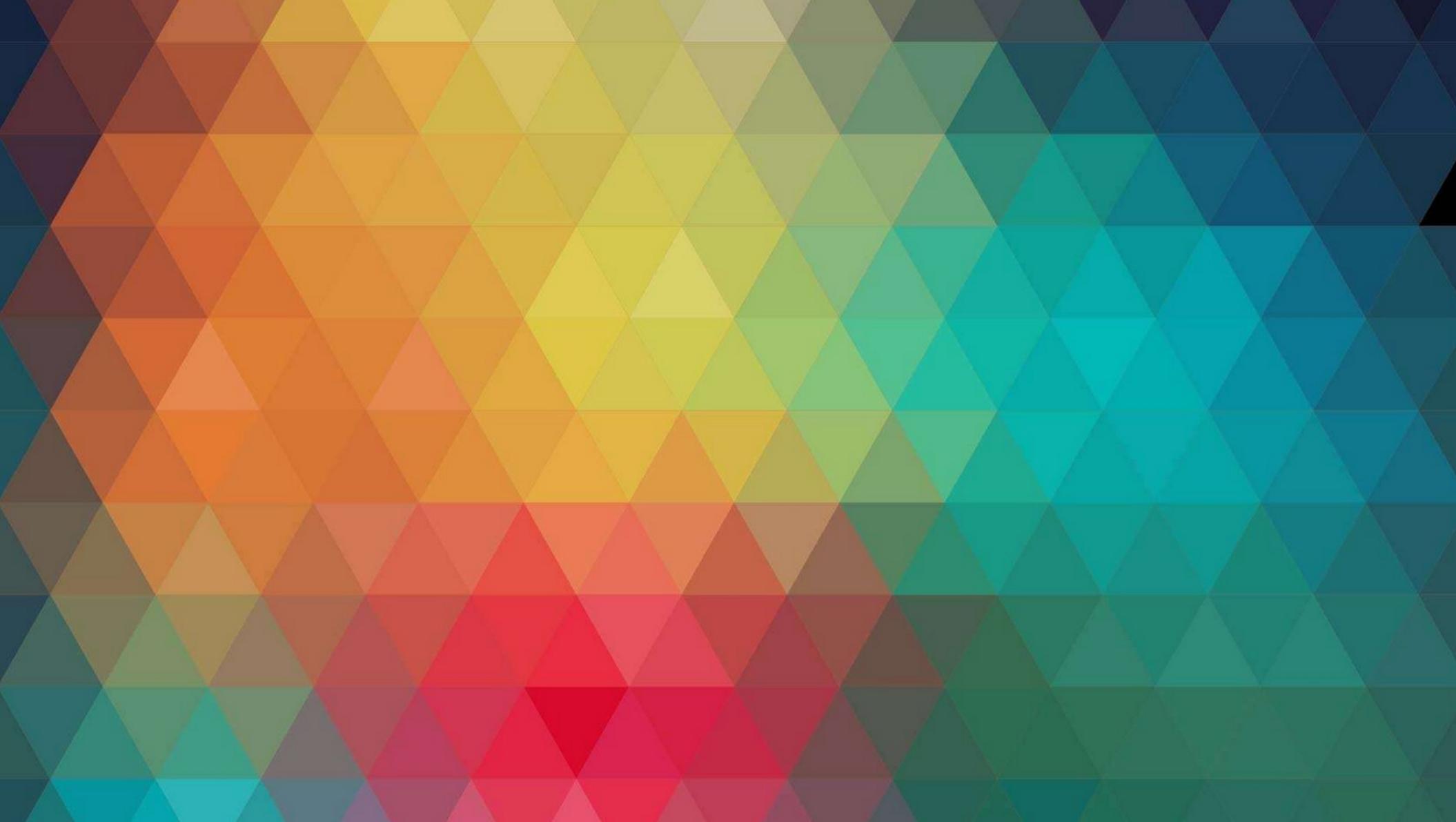
- What do we want from a PCG system?
  - Speed
  - Reliability
  - Controllability
  - Diversity
  - Creativity

- Diversity
  - How unique is your content?
    - Background
    - Perceptual differentiation
    - Perceptual uniqueness
    - Characterful









## Further Reading

- *Practical Procedural Generation for Everyone*, Kate Compton at GDC 2017  
<https://youtu.be/WumyfLEa6bU>
- ProcJam  
<http://www.procjam.com/>
- Procedural Content Generation in Games  
<http://pcgbook.com/>

