
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
Tiles

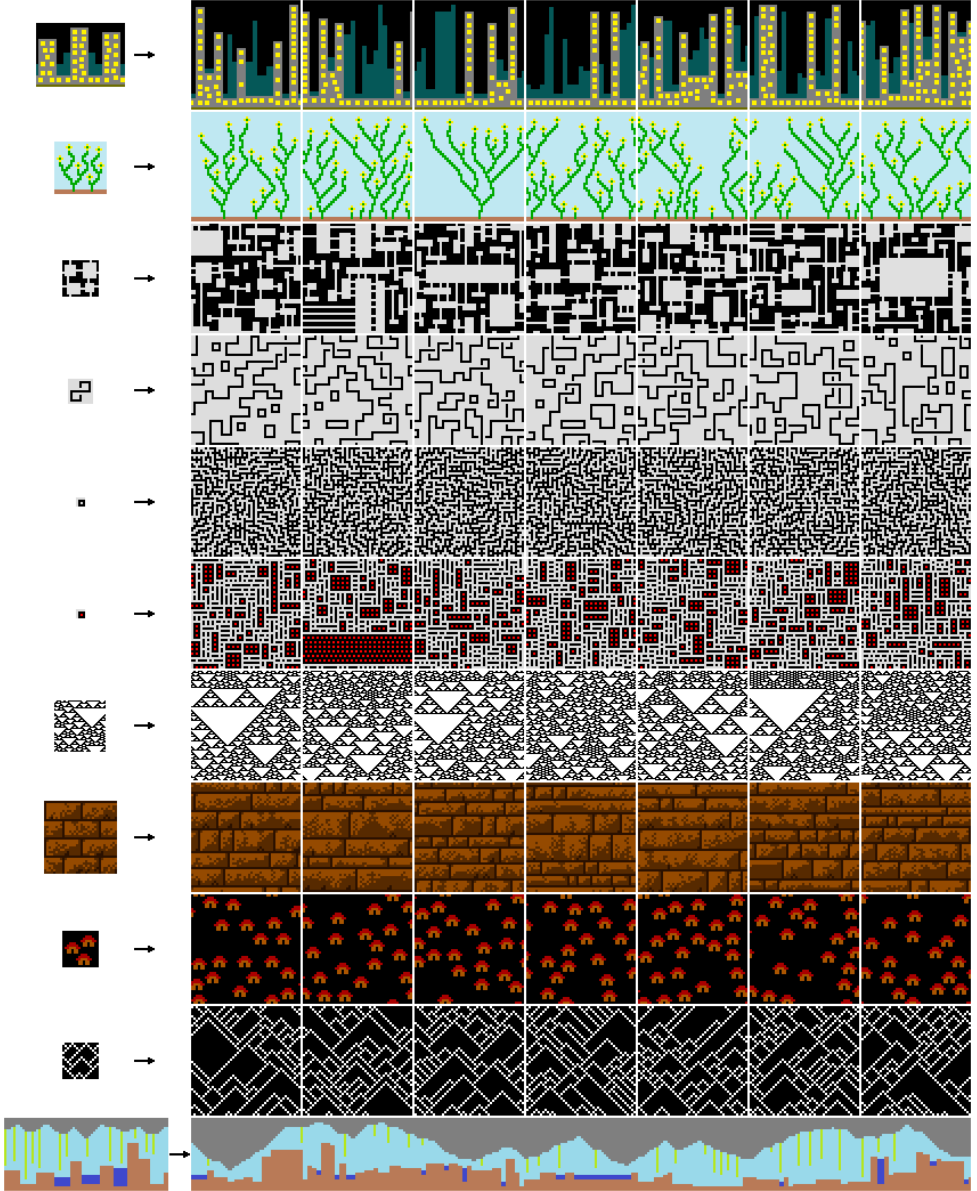
- Tiles
  - When you have
    - Equal sized blocks of content that can be placed anywhere
    - Not a lot of constraints on how tiles can be arranged
    - Variation in placement leads to interesting variation
  - Tiles can be large or small, and have more or less internal structure
  - e.g. tile maps,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

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

- 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

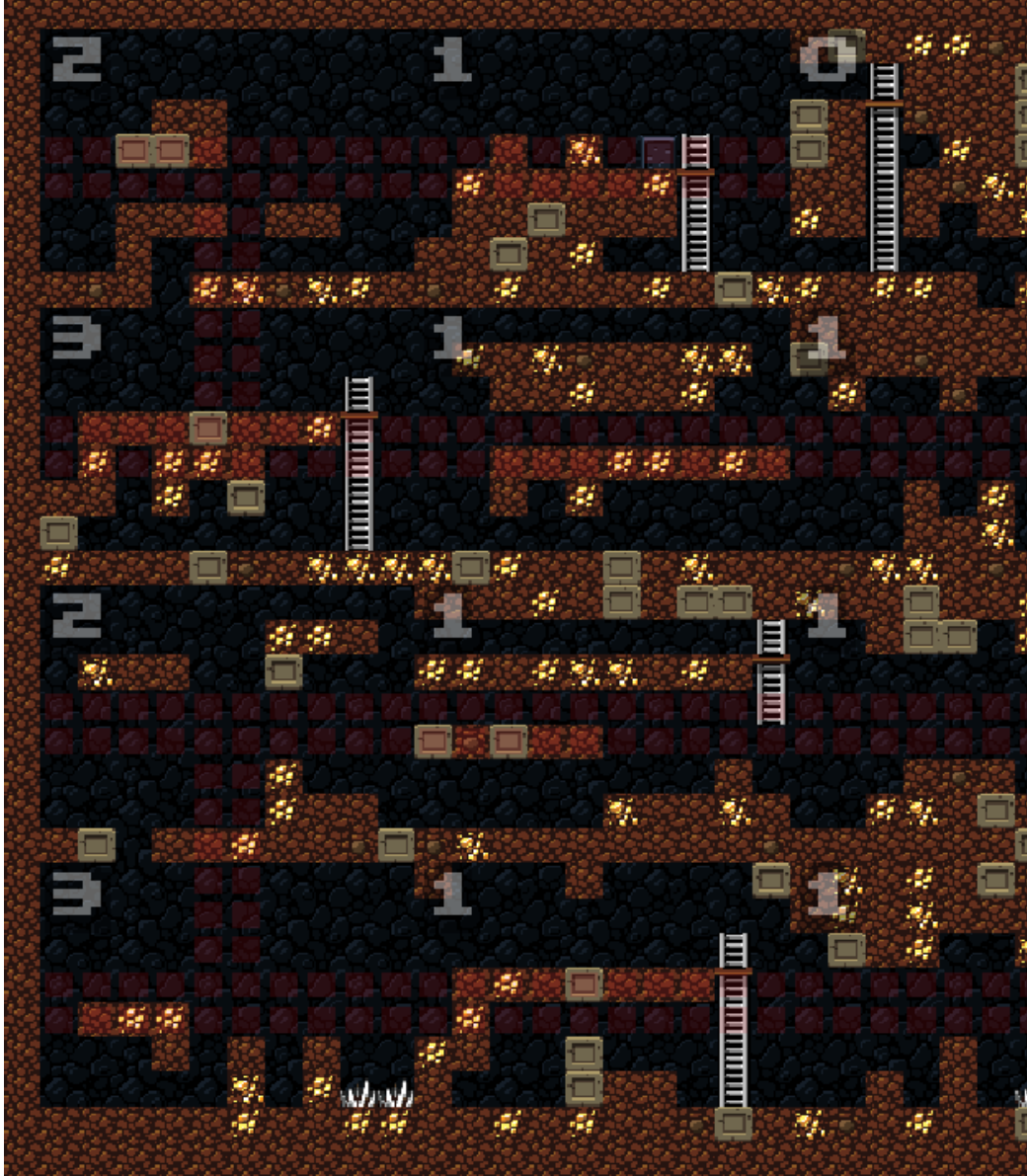
https://github.com/mxgmn/WaveFunctionCollapse

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

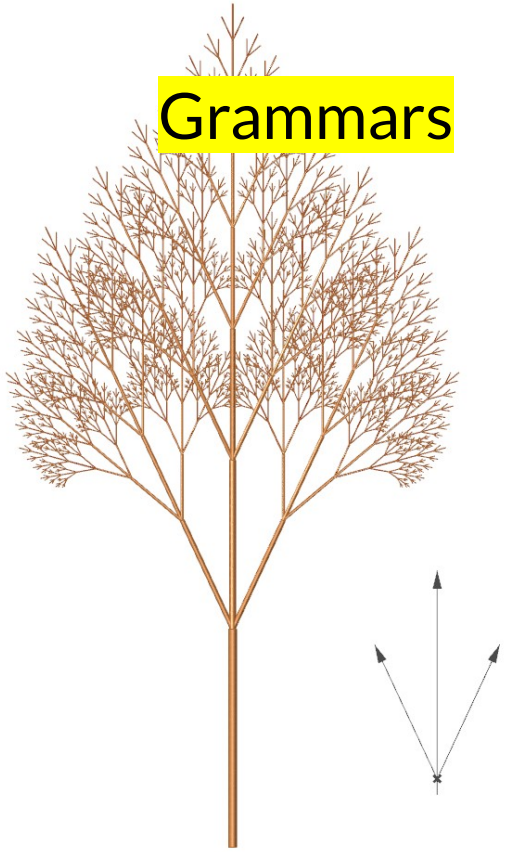


- Spelunky Level Generation
  - Template-based
  - Walks a path through a 4x4 grid, ensures passability
  - Templates include limited random elements

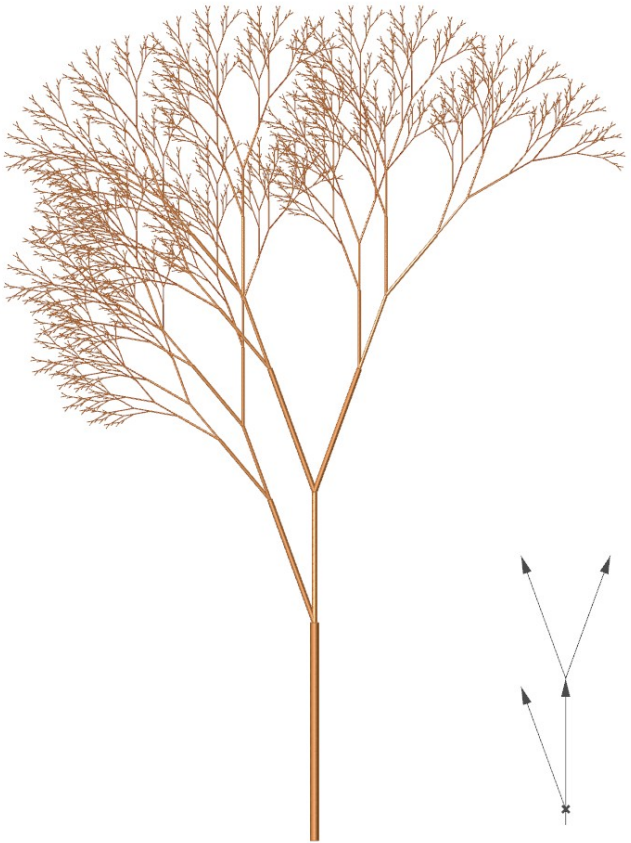
http://tinysubversions.com/spelunkyGen/



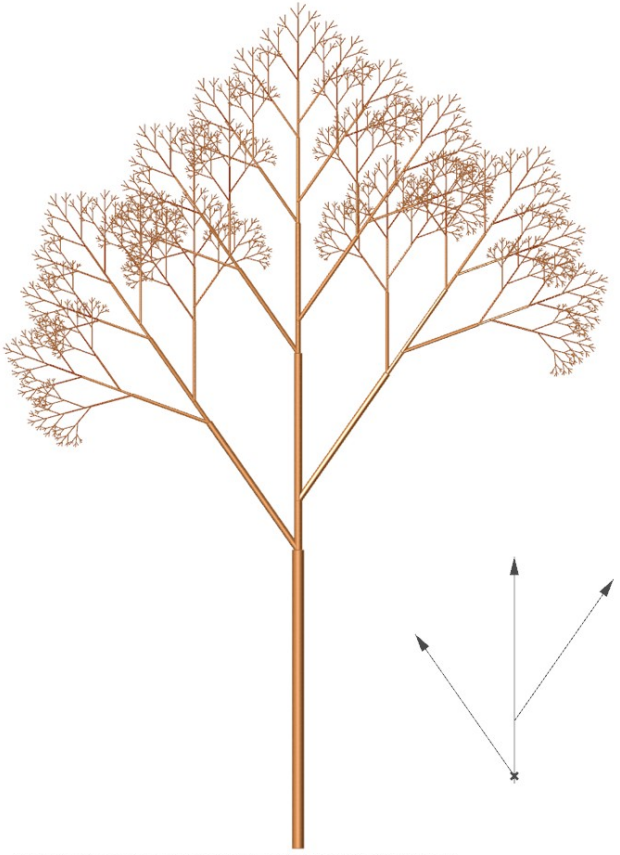
Grammars



Variant 1 (Two angled Branches and 1 Straight Branch)

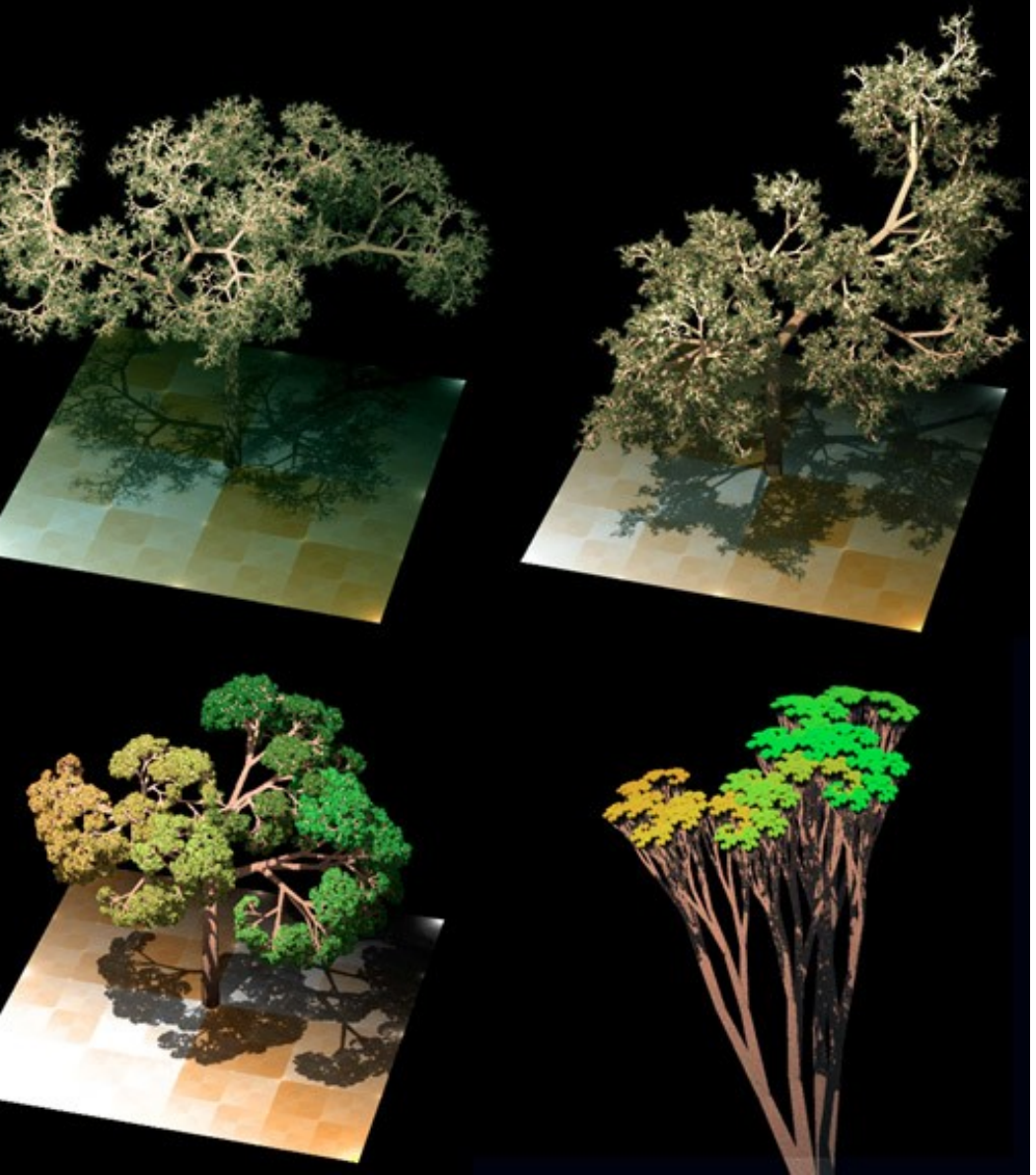


Variant 2 (One Angled and One Straight Branch, followed by two Angled Branches)



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

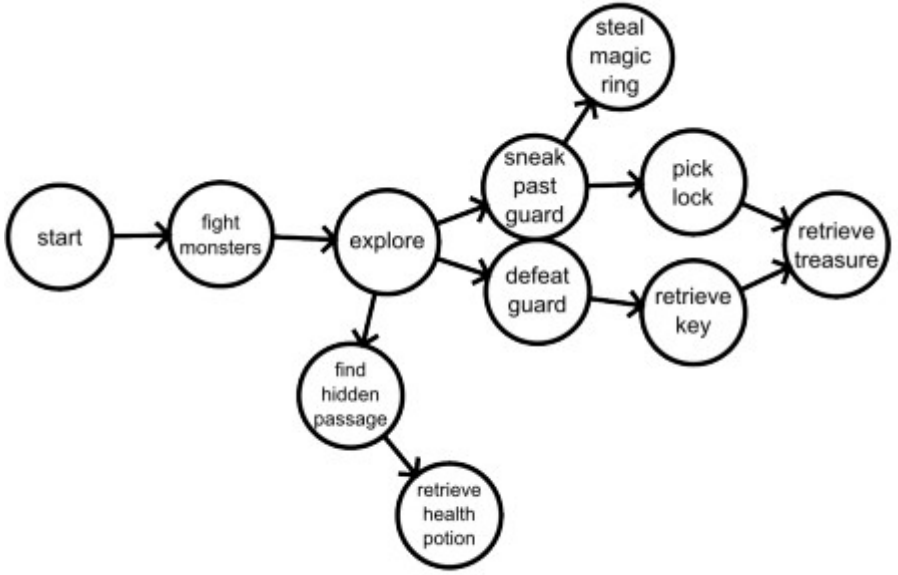


Fig. 5.6: A mission structure with two paths

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

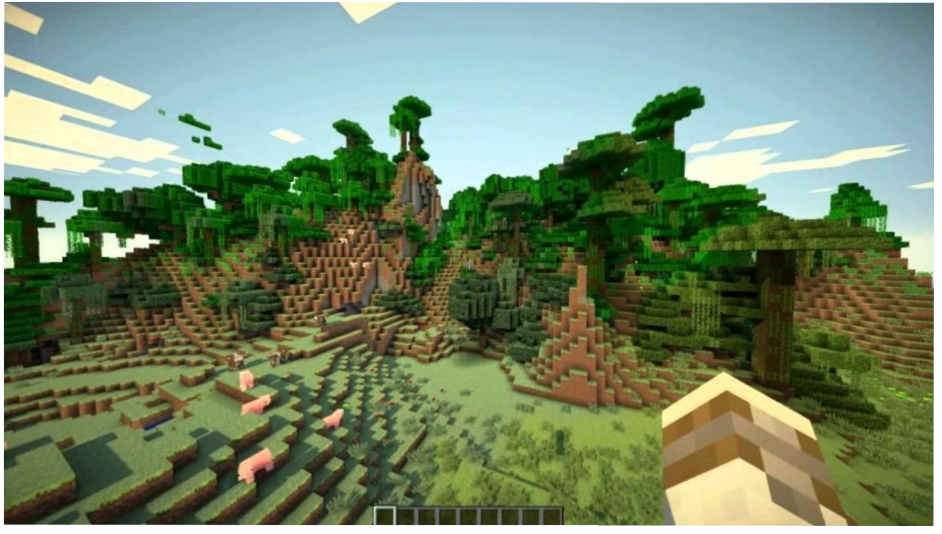
Image from http://pcgbook.com/

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

    http://tracery.io/

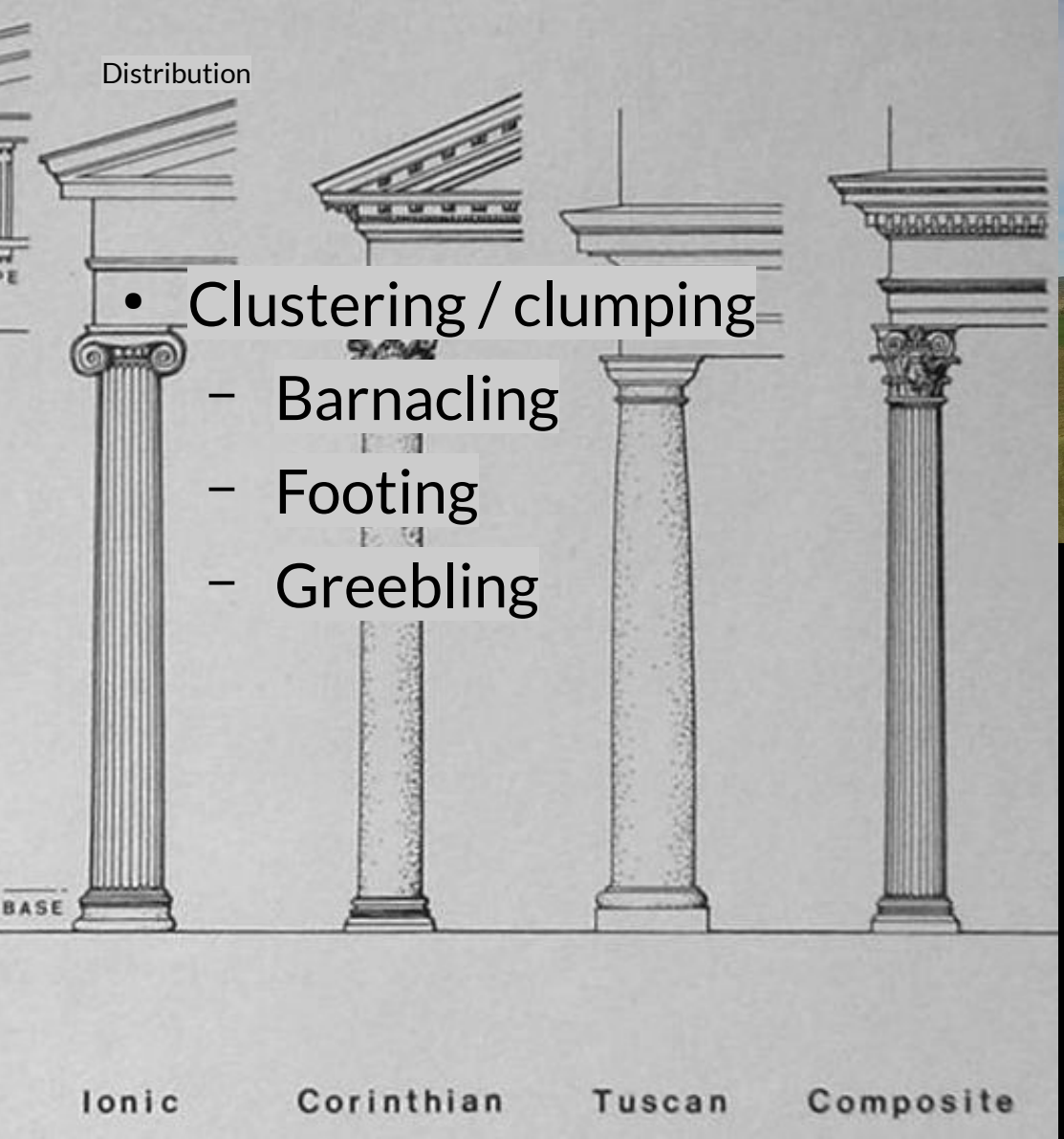

Distribution
Toinginomi Guas
-70.0°C

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

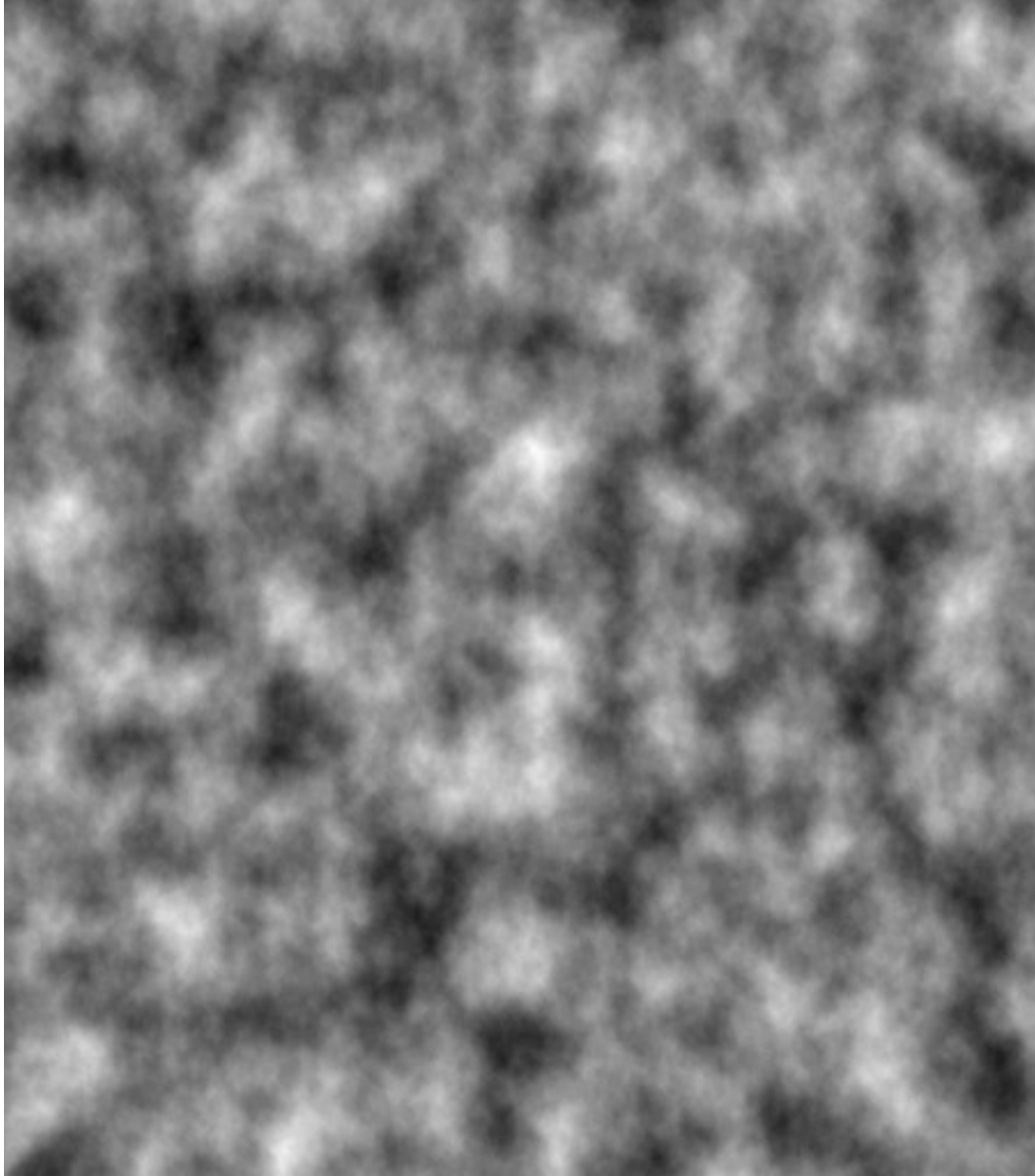


Distribution

- Clustering / clumping
  - Barnacling
  - Footing
  - Greebling



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

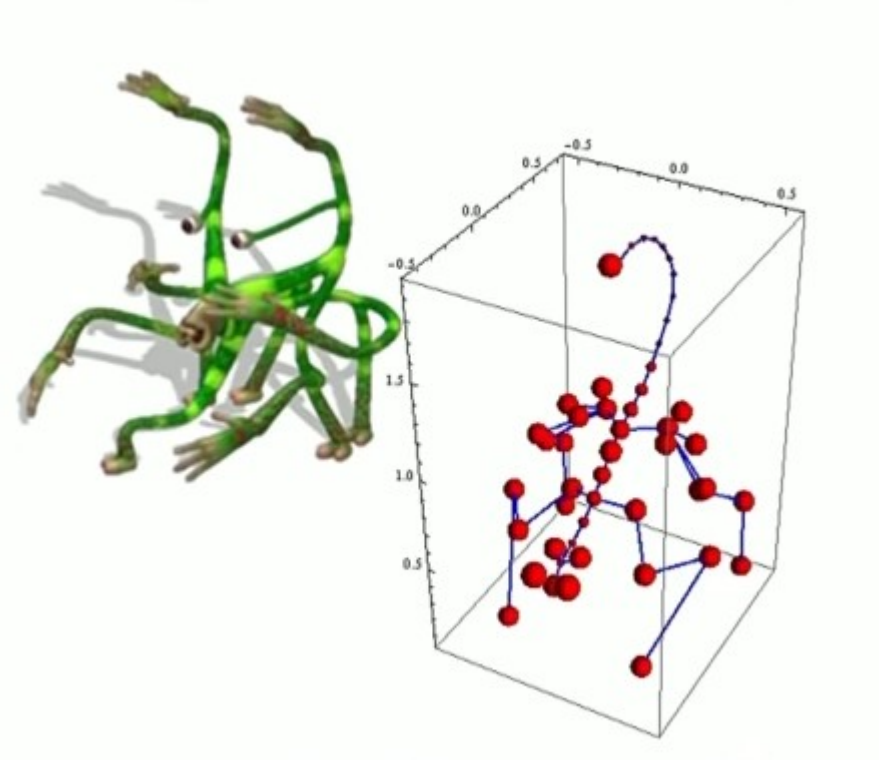



Parametric

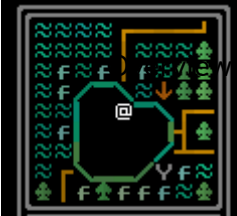
- Parametric
  - Define a multidimensional possibility space, defined by parameters
  - Structure of content is defined, but can vary on a number of dimensions


Interpretive
Base
1/3
Coat
Detail
Ictacular Grunflax

- 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 |



27 June

Simulations

Left Arm
Nothing

Right Arm
Nothing

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

Head:
[No Helmet]
[No Necklace]

Torso:
Religious Vestment
(Undamaged)
[No Cloak]

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

Right Arm:
[No Glove/Gauntlet]
[Nothing Held]
[No Ring]

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

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

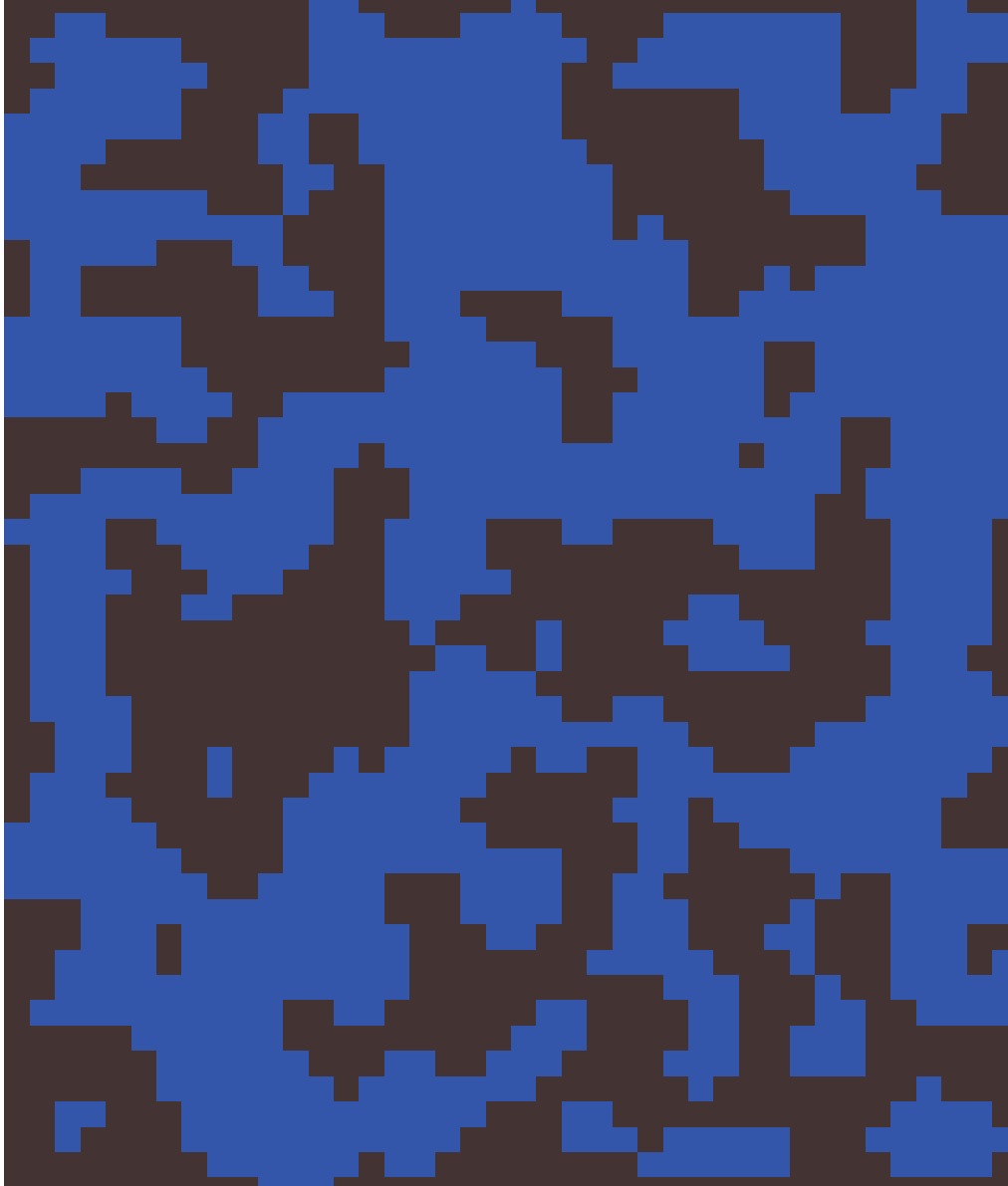
[TAB to change page, Enter to examine]

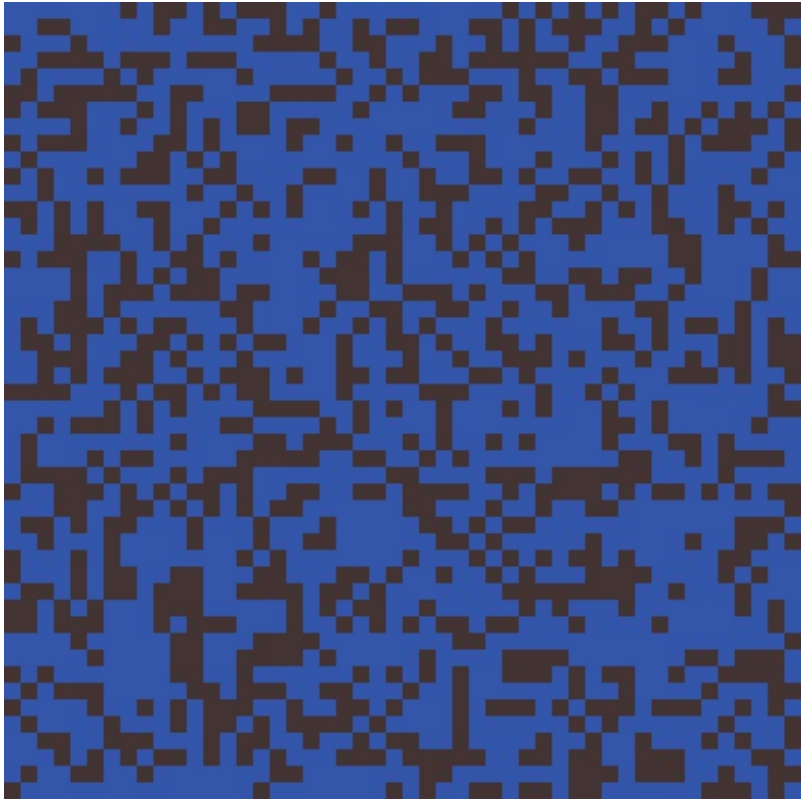
This is a religious vestment she is currently wearing.

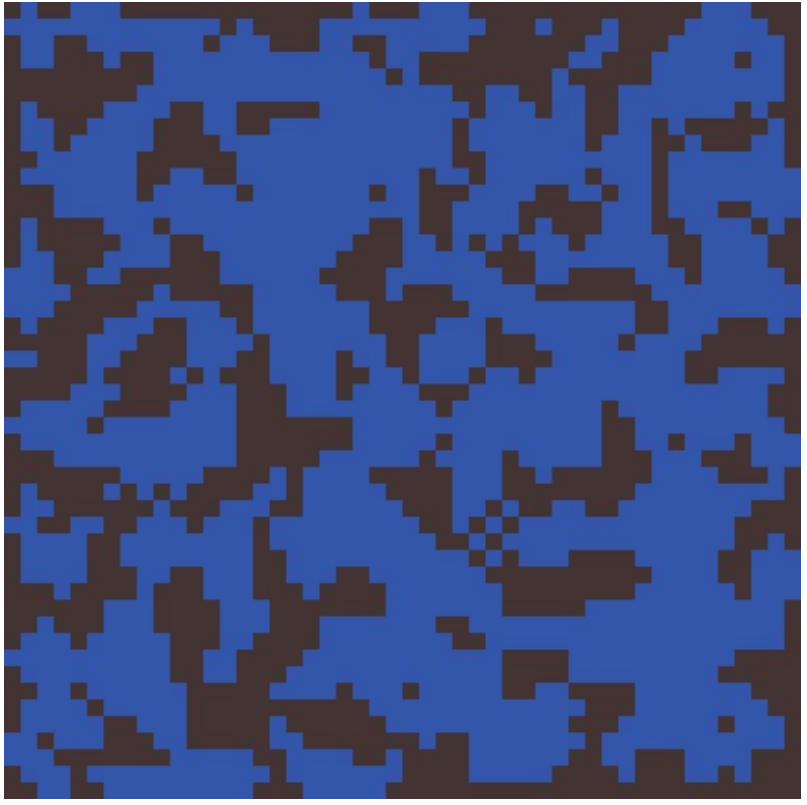
Stm: Pai: Air: Bld: Poi:

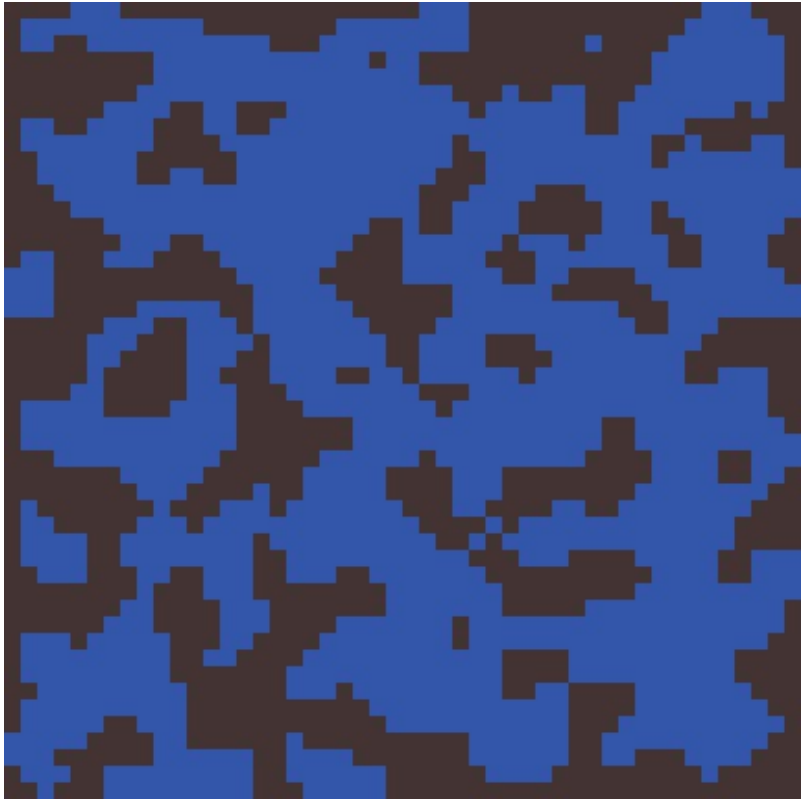
- Agent-based Simulations
  - Generate content based on actions of in-world agents
  - Agents can be vary complex...
    - e.g. in Dwarf Fortress
  - Or basic...
    - Particle trails
    - Flocking

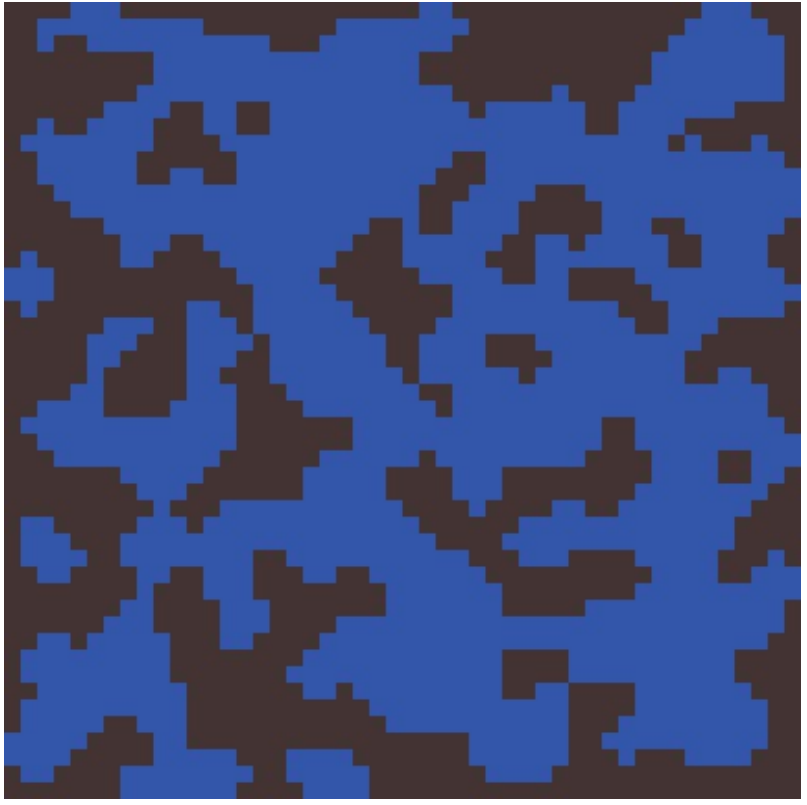
- Cellular Autometa
  - Simulate state of cells based on states of neighbours
  - For natural-looking forms such as caves, islands, etc.

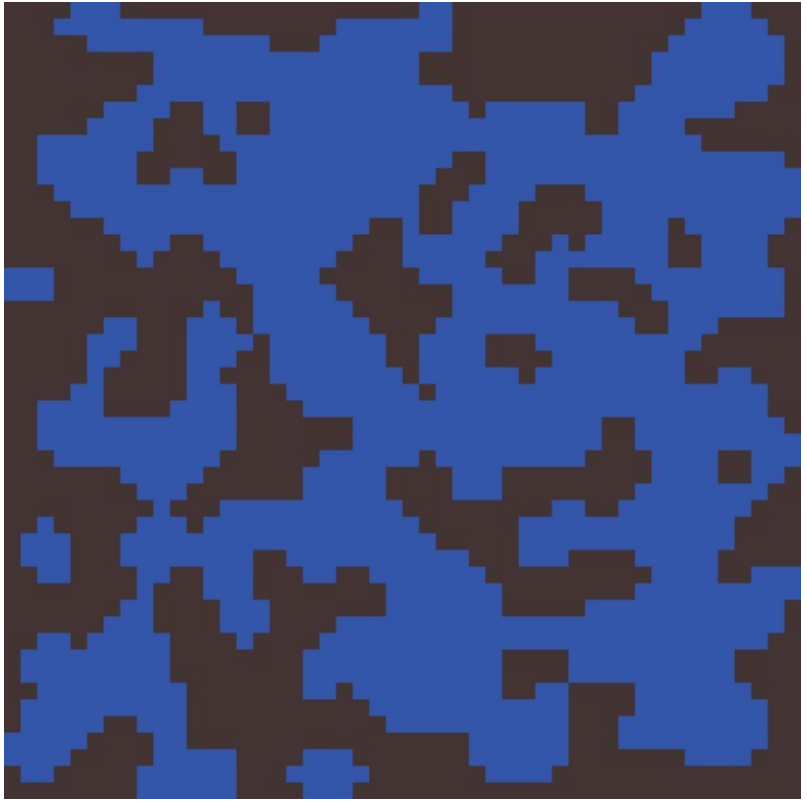


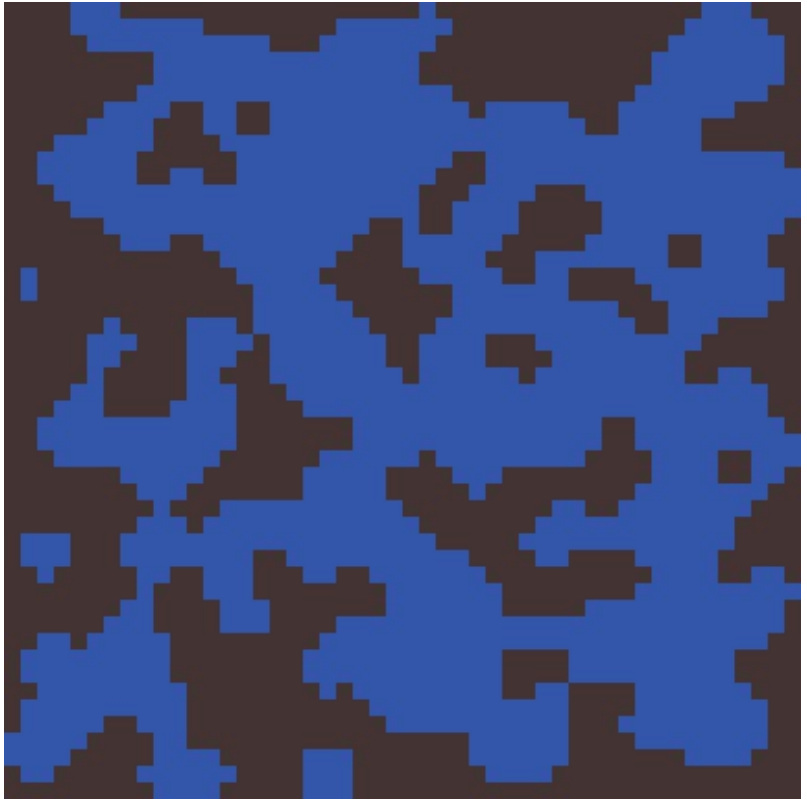


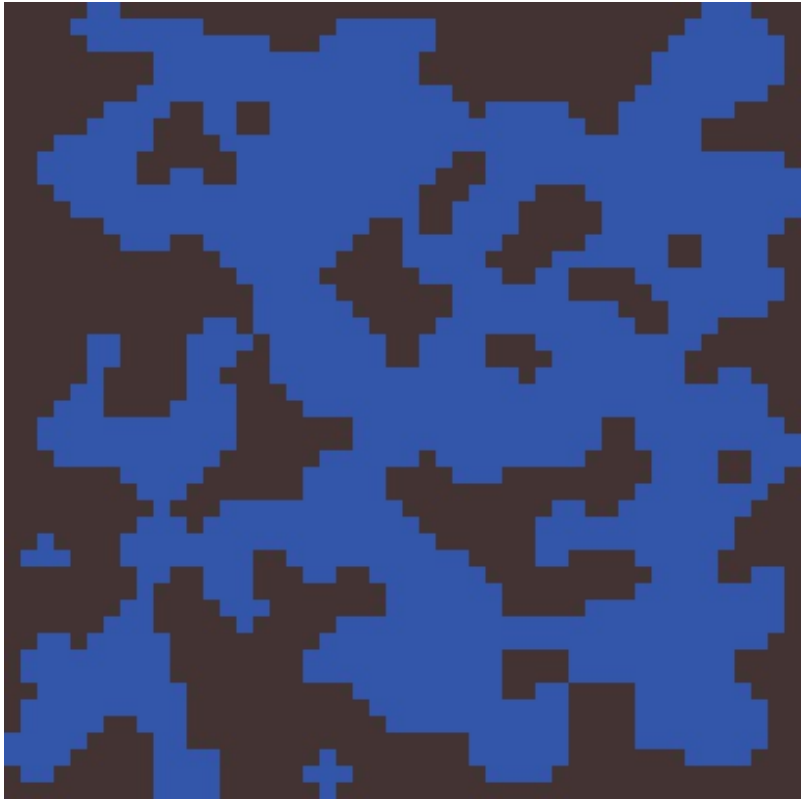


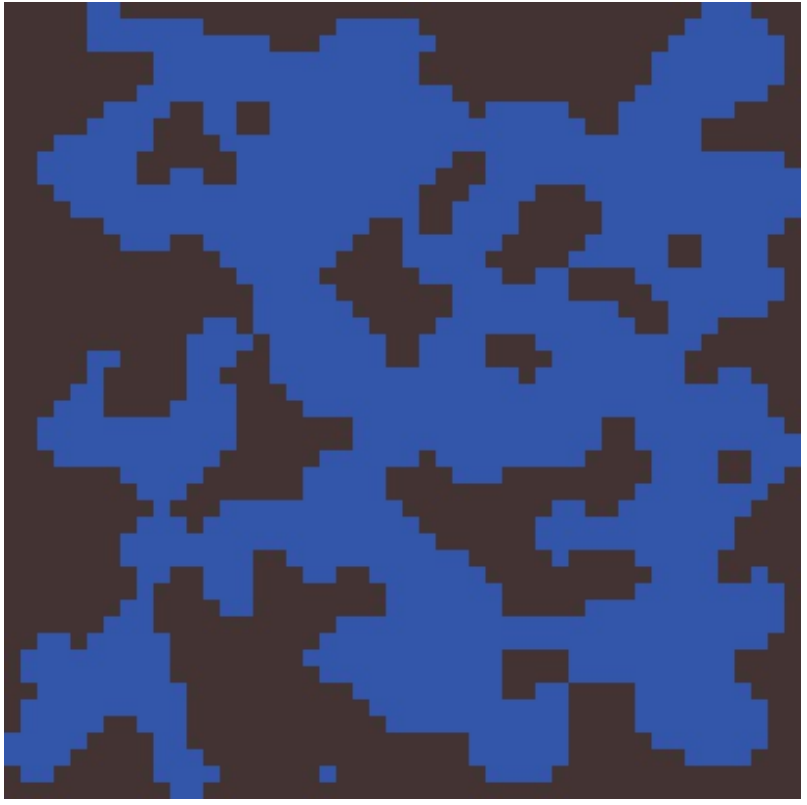


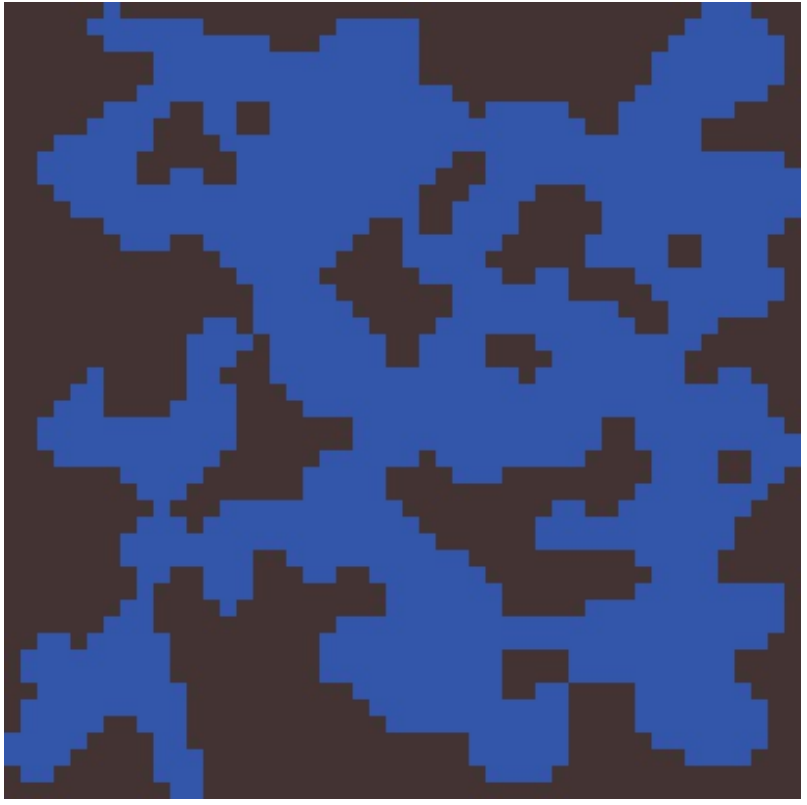




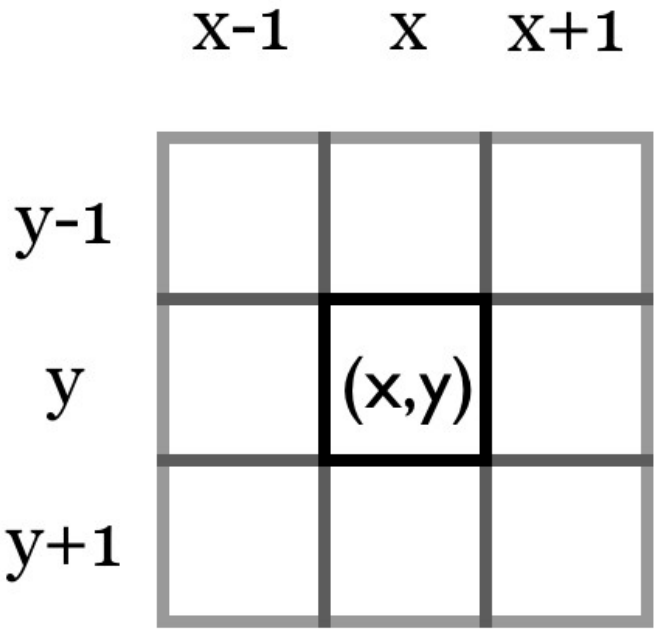








- The Game of Life (John Conway)
  1. If a living cell has less than two living neighbours, it dies.
  2. If a living cell has two or three living neighbours, it stays alive.
  3. If a living cell has more than three living neighbours, it dies.
  4. If a dead cell has exactly three living neighbours, it becomes alive.



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

- 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;
}
```

https://gamedevelopment.tutsplus.com/tutorials/generate-random-cave-levels-using-cellular-automata--gamedev-9664

- 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];
  //...
```

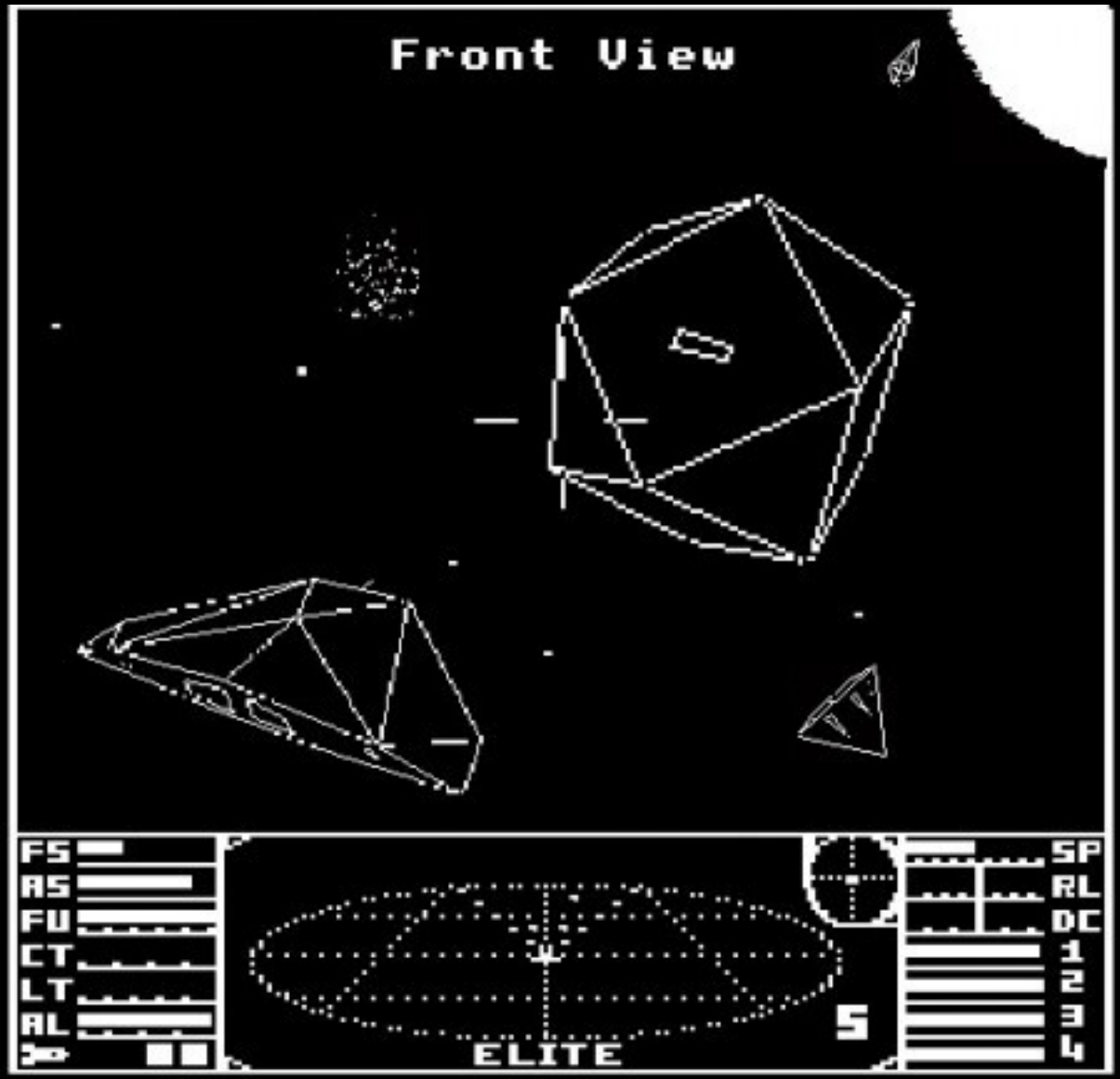
- Basic implementation:
  - Randomly initialise map
  - Write a simulation step function
    - Count alive neighbours
    - Apply rules simultaneously to all tiles

```
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[neighour_x][neighbour_y])
            count = count + 1;
      }
}
```

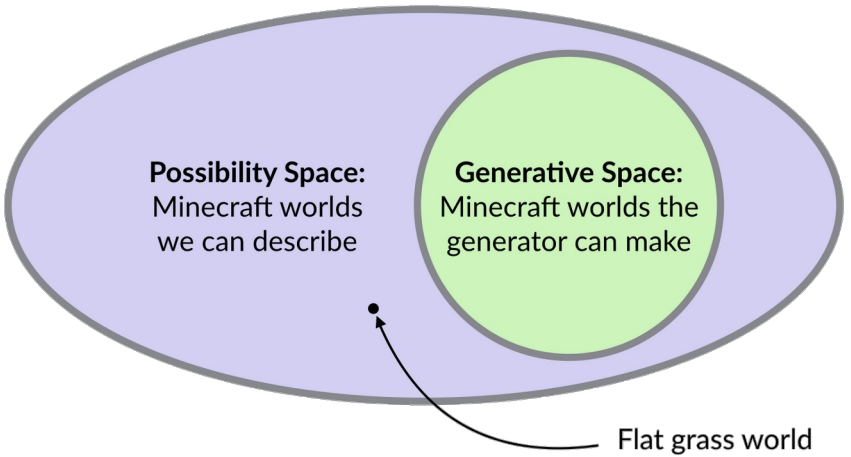
- Basic implementation:
  - Randomly initialise map
  - Write a simulation step function
    - Count alive neighbours
    - Apply rules simultaneously to all tiles

```
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 n ewMap[x][y] = false;
      }
    }
  return newMap;
}
```

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

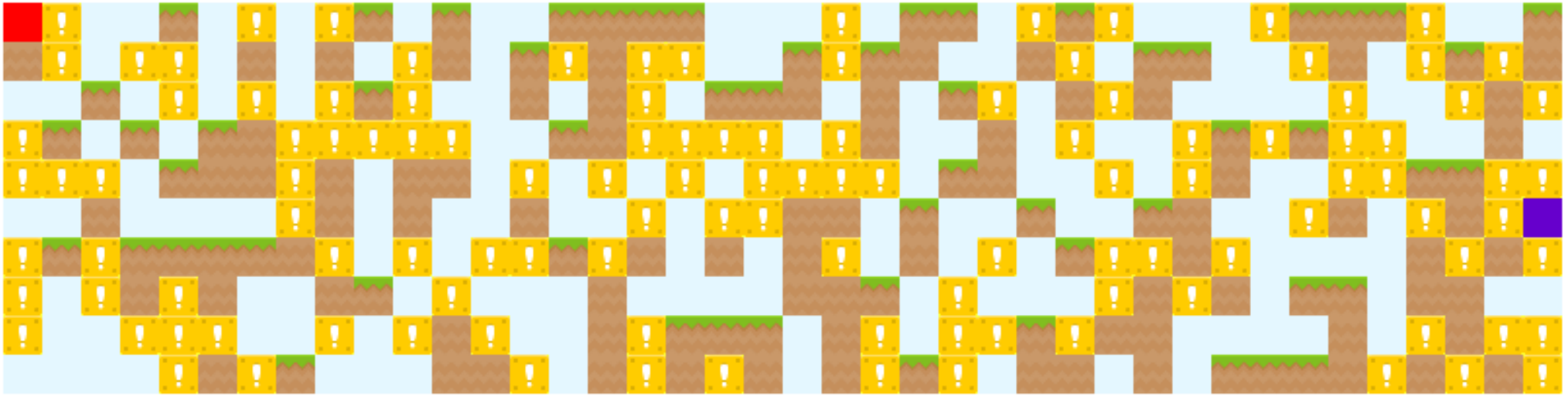
Empty Space

Solid Block

Powerup Block

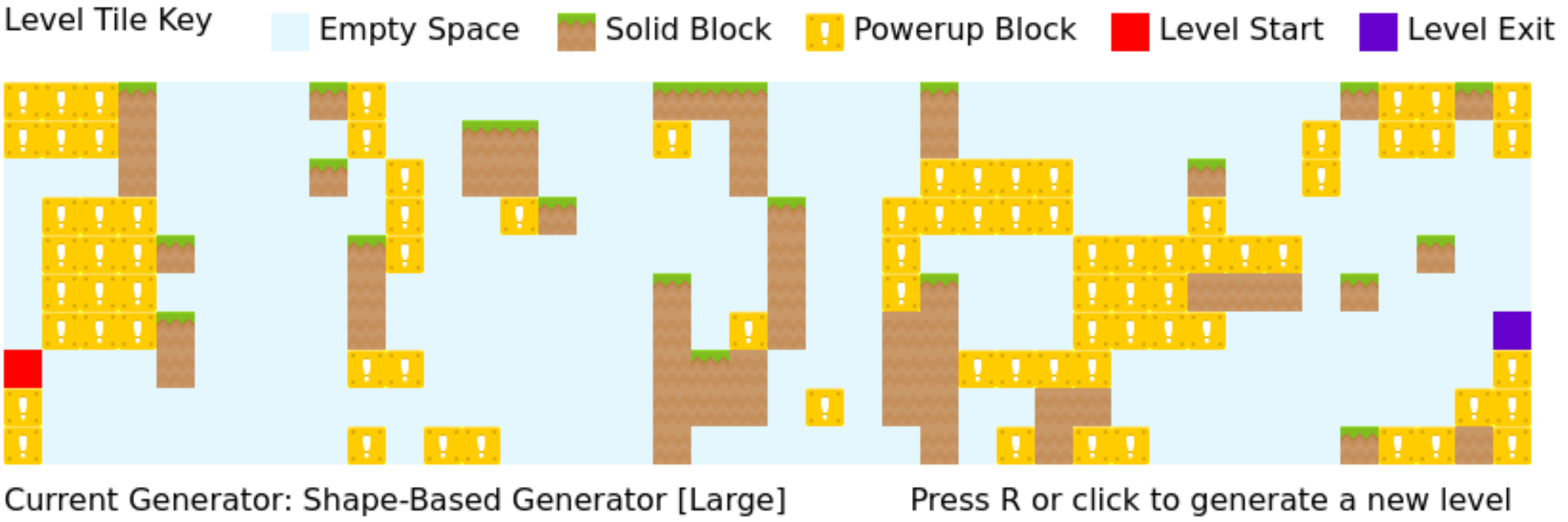
Level Start

Level Exit



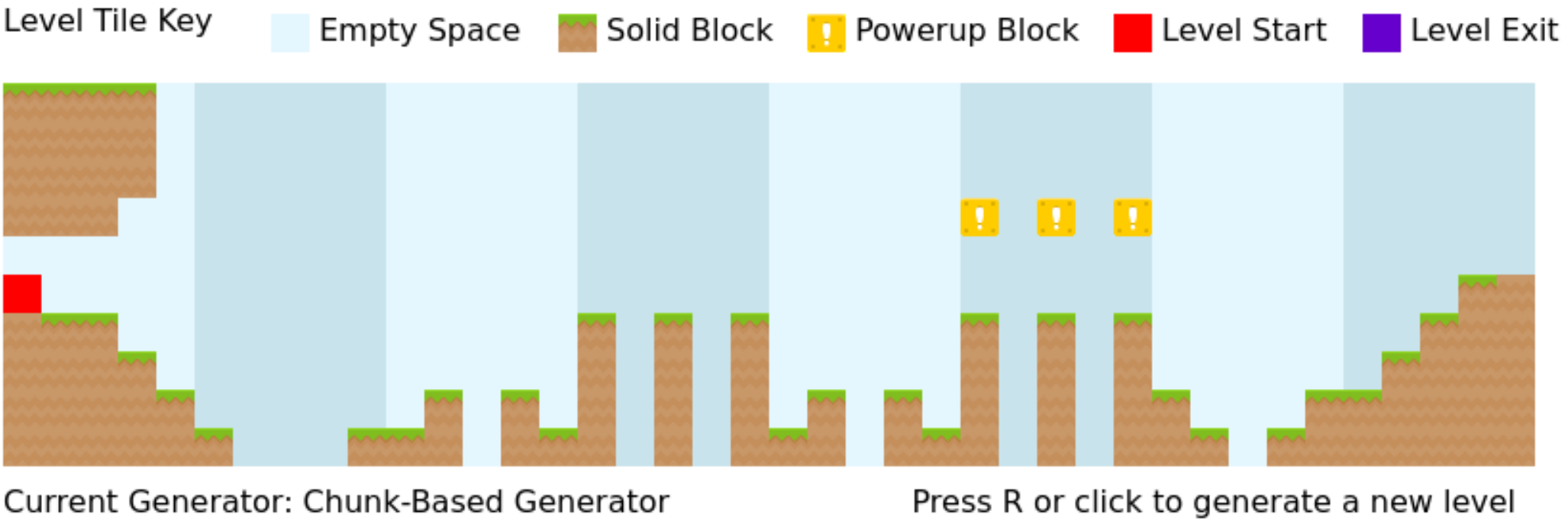
Current Generator: Random Generator

Press R or click to generate a new level



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 devopment? (**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 langauge
  - 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

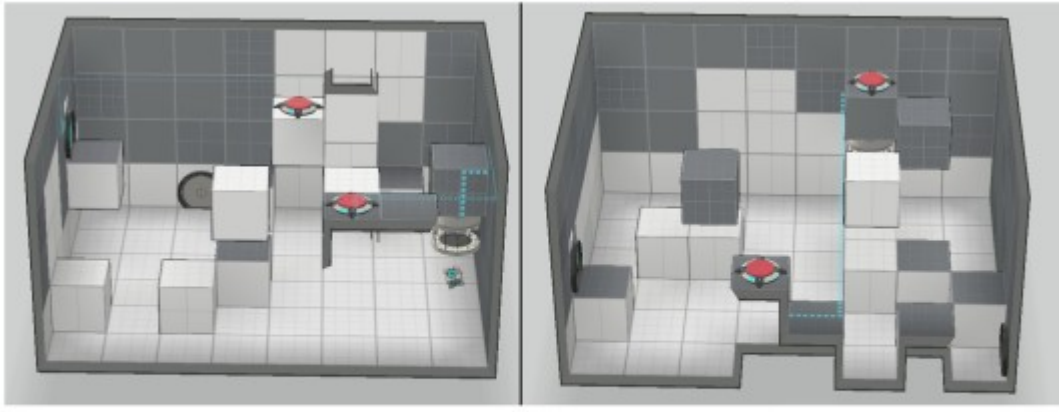
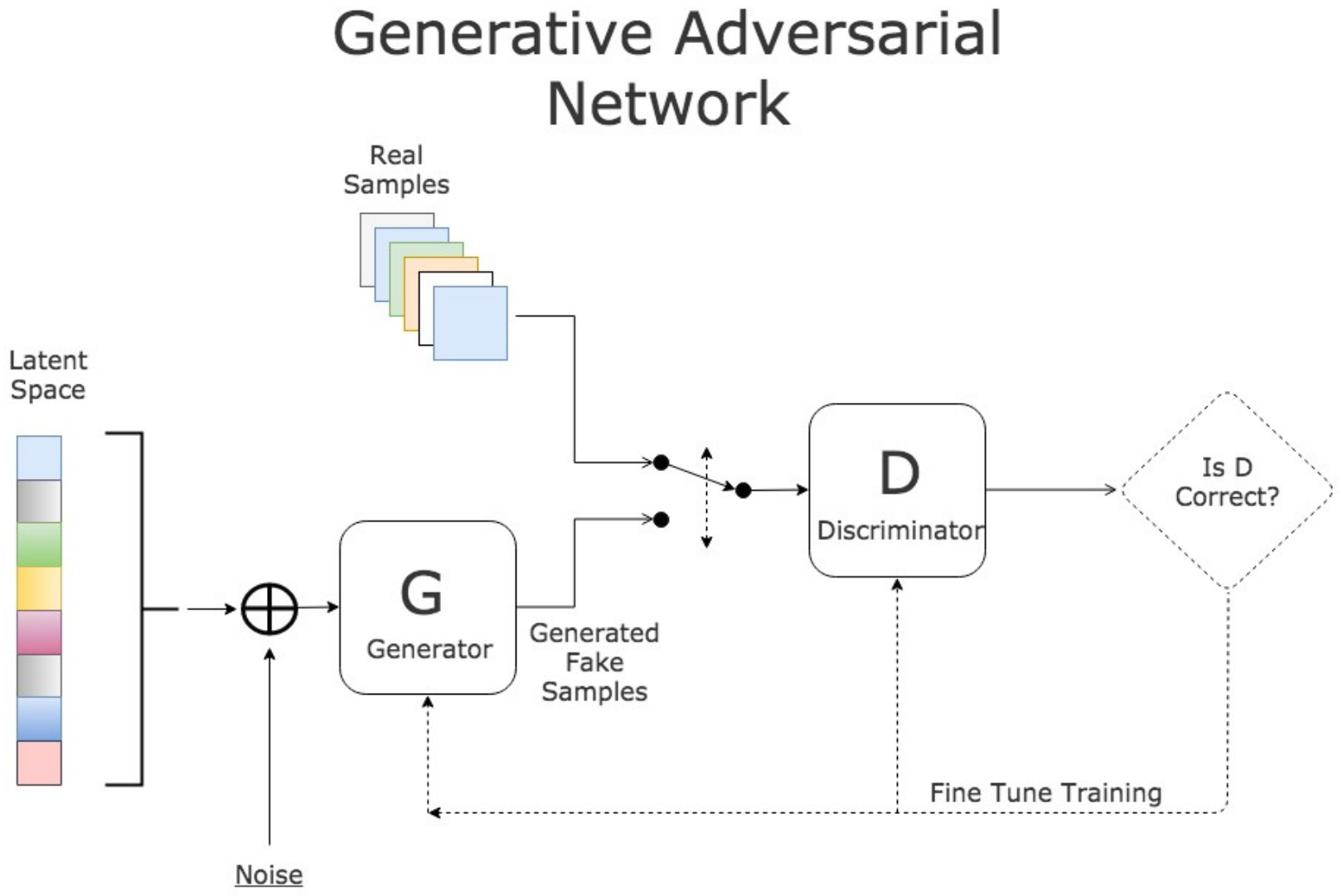
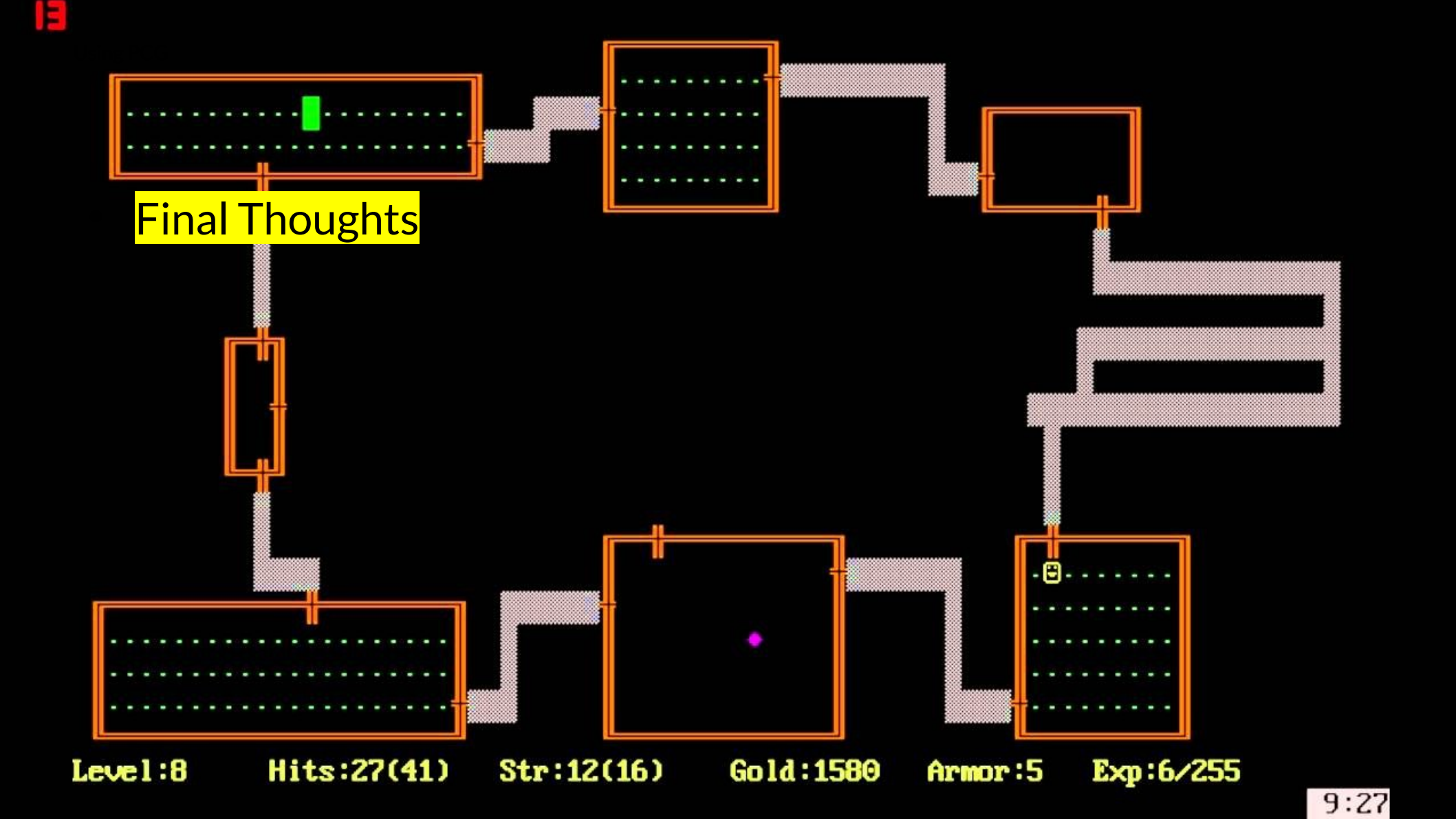


Figure 5.1: Two levels with reduced proportion of solid boxes generated by the simplified encoding.

Using PCG


Generative Adversarial Network
Real Samples
Latent Space
G
Generator
Generated Fake Samples
D
Discriminator
Is D Correct?
Noise
Fine Tune Training

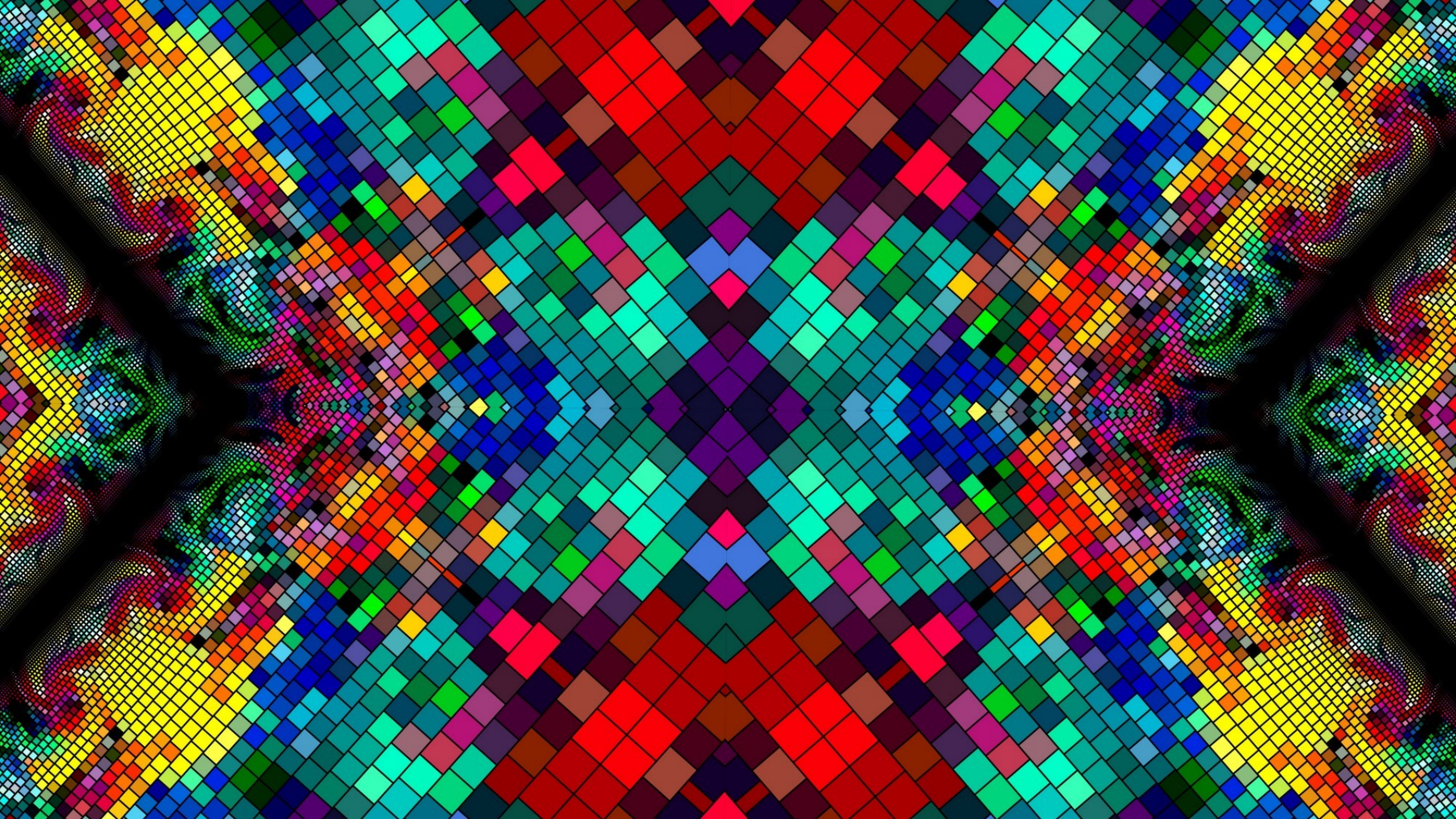
- PCG as search
  - **Content encoding** defines space
  - **Content function** renders the content encoding
  - **Content requirements** defines constraints to be satisifed/optimised
  - Search algorithm explores this space

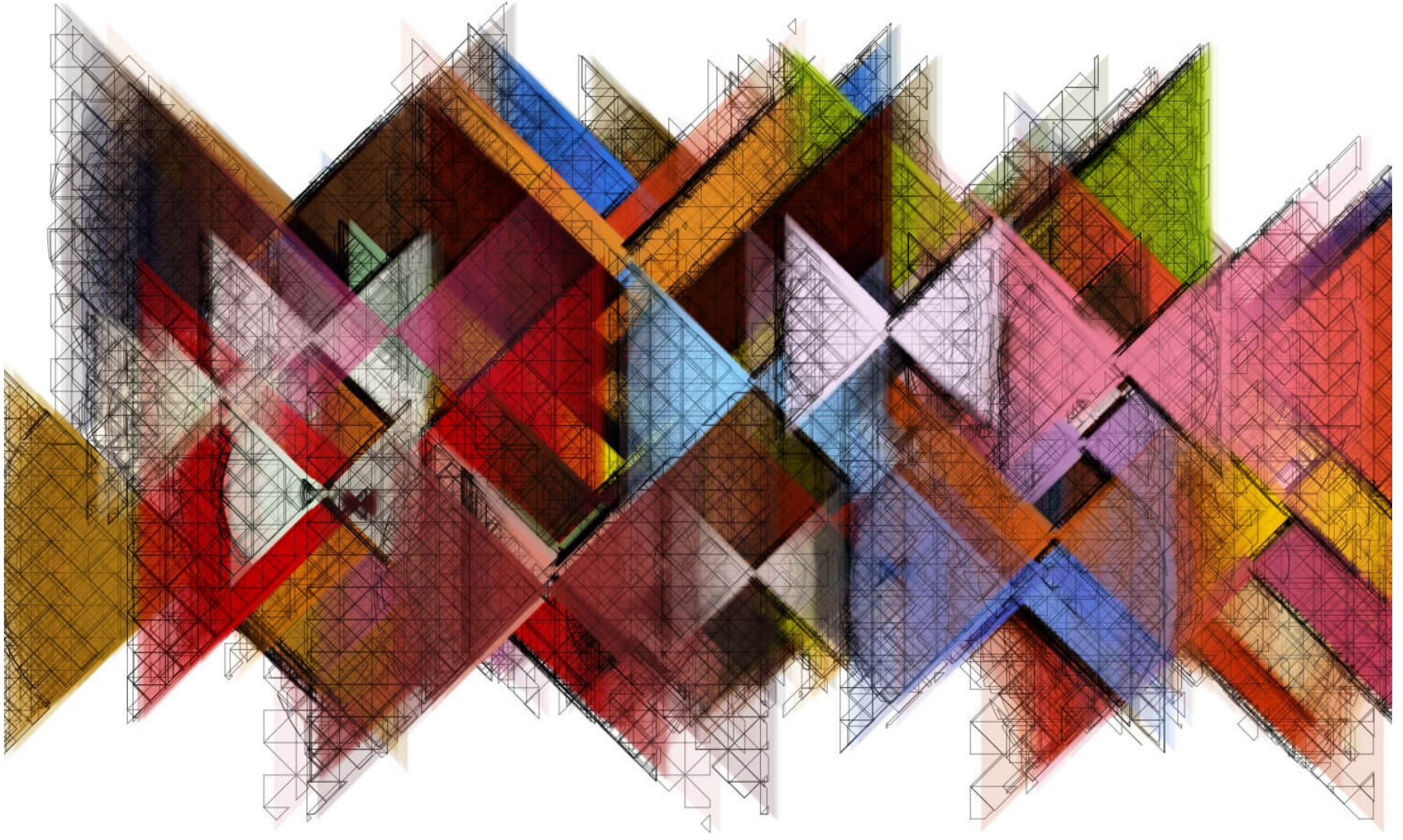

Final Thoughts
Level:8
Hits:27(41)
Str:12(16)
Gold:1580
Armor:5
Exp:6/255
9:27

- 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





- *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/