

Procedural Generation

How can shaders be used in 2D games?

It's often faster to render certain effects on the GPU, even in 2D. This is, for example, how Processing achieves better performance in P2D mode.

Can artists export the shaders that they used to design textures?

Not really.

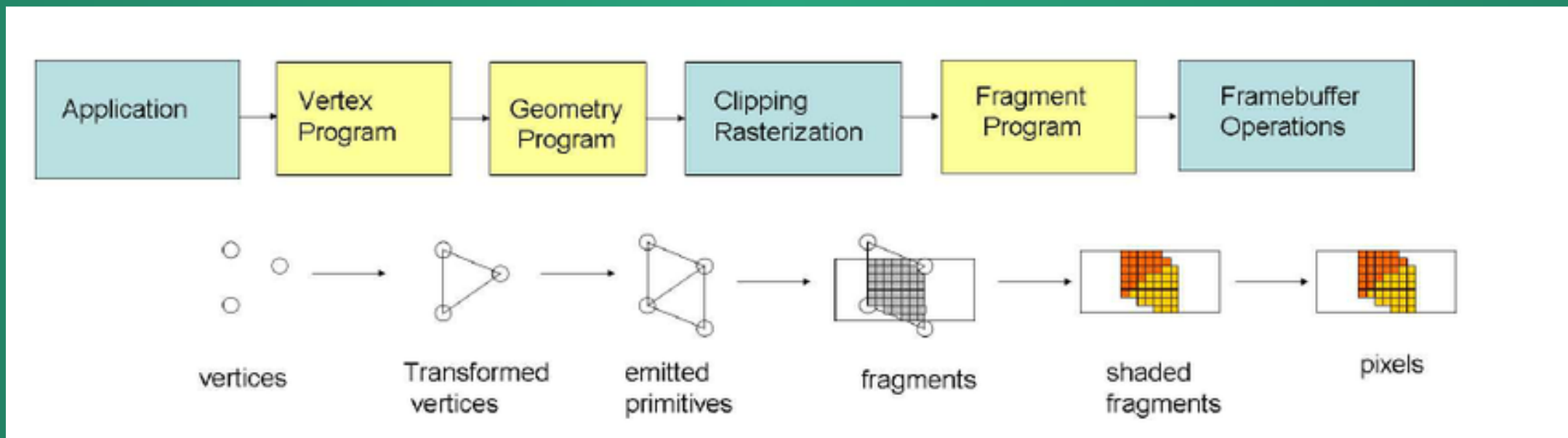
Do shaders actually draw anything, or do they just process data?

Shaders take part in a procedure that tells the GPU how to draw things.

How can Vulkan have 35% better performance than OpenGL?

Partially because AMD's OpenGL support at the time was kinda trash.

OpenGL does extra work as part of its attempt to be nice and easy to work with. However, this work is not always necessary.



What if our code isn't complete by the time we have to present?

If you don't have a minimal working example by the time of your presentation, I'd be a little concerned. It doesn't have to be pretty or complete, but the basic outline should be in place.

You don't need a complete product to present the idea of what you're working on and challenges you're facing. Most tech companies present with products that are utterly broken.

Disclaimer

I am going to get way too excited and talk too much. It is not critical that you remember everything (or really, anything) about what's discussed here.

If you want more details on any of these ideas, there are slides at the end with links where you can dig into the subjects discussed.

In the early days of electronic media, all assets, levels, and programs were human-generated.



and then someone thought

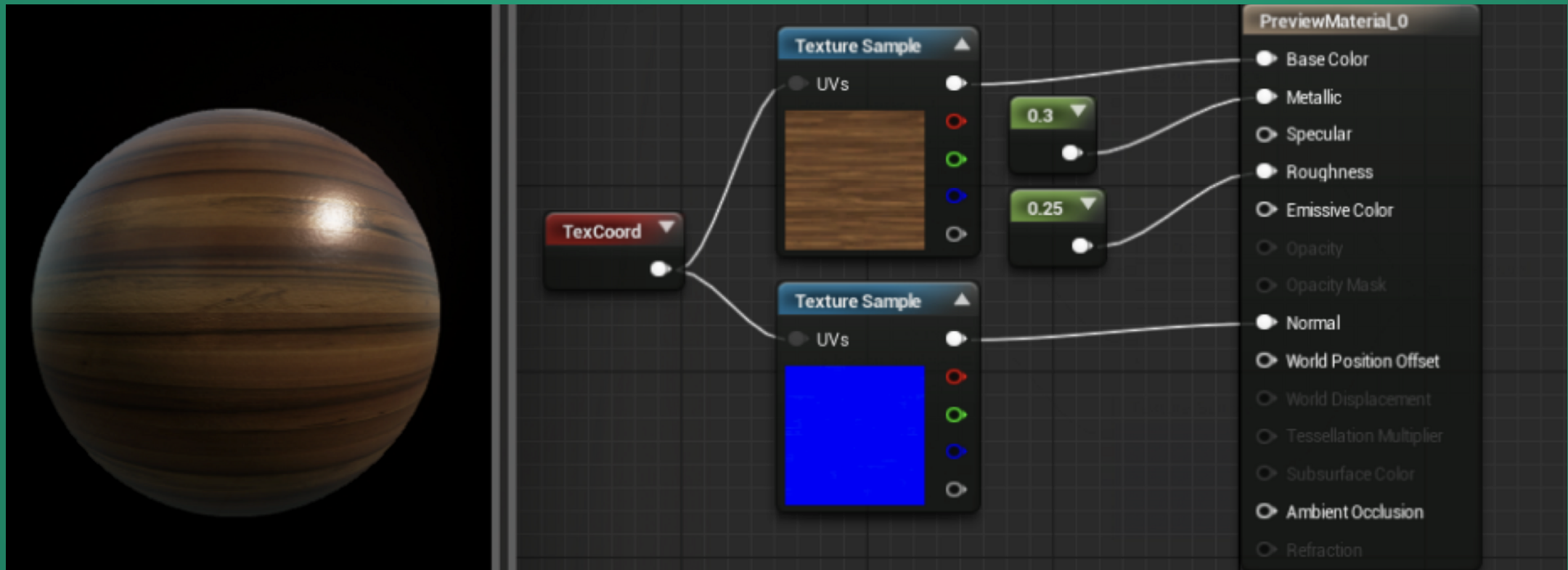
Wait, why can't I make the
computer do this work for me?

Procedural Generation

“ a method of creating data algorithmically as opposed to manually, typically through a combination of human-generated assets and algorithms coupled with computer-generated randomness and processing power.

Use the computer and randomness to generate data for your graphics program instead of requiring a human to generate it.

Today, most interactive media (and lots of noninteractive media!) uses computer-generated components.



Tree_01 x Asset Type: FoliageType_InstancedStaticMesh

Save Browse

Platforms

Settings

Details x

Search

Collision	
Collision Radius	100.0
Shade Radius	50.0
Clustering	
Num Steps	4
Initial Seed Density	0.125
Average Spread Distance	100.0
Spread Variance	150.0
Seeds Per Step	3
Distribution Seed	0
Max Initial Seed Offset	0.0
Growth	
Can Grow in Shade	<input checked="" type="checkbox"/>
Spawns in Shade	<input checked="" type="checkbox"/>
Max Initial Age	0.0
Max Age	15.0
Overlap Priority	1.0
Procedural Scale	Min 1.0 Max 5.0



Even when the level and assets are handmade, the *experience* may be due to a complex combination of systems interacting---that is, the *behavior* is generated.



Before we get into this though, there's a fundamental topic we need to talk about.



Random Numbers



```
float x = random( )
```

How in the world can your computer possibly get a random number?

Without `rand()`, every program we write is totally predictable. But how can we implement `rand()` if all code is completely predictable??

Solution: Cheat!

Okay not really.

Random Number Generators (RNGs) on your computer do not actually try to make random numbers. Instead, they try to make a sequence that is *unpredictable*: even if you've seen a lot of output, it's still hard to predict the next number.

But if you have the right key....

```
>>> random.seed(10)
>>> a = [ random.random() for _ in range(5) ]
>>> random.seed(10)
>>> b = [ random.random() for _ in range(5) ]
>>> print(a)
[0.5714025946899135, 0.4288890546751146, 0.5780913011344704, 0.20609823213950174, 0.81332125135732]
>>> print(b)
[0.5714025946899135, 0.4288890546751146, 0.5780913011344704, 0.20609823213950174, 0.81332125135732]
>>> a == b
True
>>> █
```

Example: LCG

The "Linear Congruential Generator"

```
1 int seed; // Initialized elsewhere
2 int m = 2^16 + 1;
3 int a = 75;
4 int c = 74;
5
6 int random(){
7     int next = a * seed + c;
8     next = next % m;
9     seed = next;
10    return next;
11 }
```

With seed = 100

[4, 8, 4, 7, 2, 6, 1, 6, 4, 1, 0, 1, 4, 4, 8, 3, 1, 0,
9, 8, 4, 1, 2, 8, 0, 5, 1, 1, 4, 7, 3, 7, 9, 6, 0, 4,
7, 9, 8, 0, 5, 1, 2, 7, 3, 7, 4, 9, 1, 1, 2, 0, 6, 8,
4, 0, 9, 3, 6, 3, 8, 7, 8, 4, 5, 3, 8, 8, 4, 2, 6, 0,
3, 1, 3, 2, 6, 8, 3, 1, 2, 5, 1, 2, 1, 0, 3, 2, 6, 4,
8, 7, 8, 7, 2, 1, 8, 0, 2, 5]

This RNG is fast, but tends to generate low-quality randomness.
Avoid it where possible.

Implications

If we know the seed used for an RNG, we can perfectly replicate the sequence of "random" numbers that the RNG generates.

We can specify lots and lots and lots of "random"-looking data with a single number: the random number seed!

[10, 1, 7, 8, 10, 1, 4, 8, 8, 5, 3, 1, 9, 8, 6, 2, 4, 6, 1, 7, 3, 10, 6, 7, 7, 5, 5, 8, 3, 5, 6, 3, 8, 4, 8, 10, 7, 1, 10, 1, 4, 3, 4, 5, 9, 6, 4, 6, 9, 8, 7, 8, 2, 10, 6, 9, 3, 4, 7, 4, 1, 1, 8, 5, 10, 2, 9, 2, 3, 7, 10, 6, 10, 3, 2, 2, 8, 3, 4, 6, 7, 7, 8, 4, 5, 3, 10, 9, 3, 2, 5, 8, 5, 3, 3, 3, 8, 6, 6, 7, 4, 1, 9, 1, 6, 6, 4, 2, 5, 8, 7, 10, 3, 7, 8, 4, 9, 5, 9, 8, 10, 8, 2, 3, 8, 8, 7, 3, 7, 9, 10, 6, 9, 7, 8, 3, 9, 8, 2, 7, 1, 1, 8, 9, 2, 1, 6, 2, 3, 2, 10, 8, 8, 3, 8, 7, 9, 8, 6, 5, 8, 7, 6, 9, 9, 3, 4, 6, 10, 1, 4, 10, 10, 8, 1, 2, 5, 7, 4, 5, 2, 1, 10, 9, 2, 5, 3, 8, 3, 6, 10, 7, 4, 6, 3, 2, 10, 4, 7, 4, 2, 5, 8, 9, 7, 8, 9, 4, 5, 10, 6, 2, 1, 4, 7, 10, 7, 7, 4, 2, 7, 2, 1, 6, 5, 6, 8, 7, 4, 5, 8, 5, 9, 8, 9, 5, 7, 8, 6, 6, 5, 8, 8, 8, 10, 8, 2, 9, 6, 7, 3, 10, 7, 10, 2, 10, 5, 1, 10, 6, 6, 2, 7, 10, 6, 5, 10, 3, 4, 3, 5, 2, 9, 1, 4, 10, 2, 7, 4, 10, 10, 8, 10, 1, 7, 4, 6, 10, 4, 9, 10, 6, 1, 10, 1, 1, 10, 6, 7, 6, 10, 3, 3, 2, 10, 6, 3, 1, 9, 9, 6, 2, 7, 8, 10, 6, 1, 3, 8, 8, 10, 8, 8, 10, 3, 6, 7, 6, 9, 9, 8, 7, 4, 2, 7, 3, 1, 3, 4, 10, 7, 6, 9, 7, 1, 10, 1, 3, 5, 4, 1, 10, 3, 2, 10, 6, 3, 5, 1, 7, 10, 7, 2, 6, 4, 8, 3, 3, 7, 8, 6, 1, 3, 3, 10, 7, 3, 9, 7, 2, 7, 2, 4, 2, 7, 5, 5, 9, 2, 6, 9, 10, 8, 10, 4, 9, 5, 1, 3, 7, 4, 2, 9, 8, 2, 10, 7, 10, 4, 5, 6, 2, 4, 10, 9, 9, 1, 2, 7, 1, 1, 7, 6, 3, 1, 6, 6, 8, 3, 1, 4, 3, 5, 4, 4, 4, 7, 10, 9, 10, 10, 7, 9, 1, 3, 1, 2, 1, 1, 3, 2, 7, 2, 1, 7, 3, 10, 7, 1, 9, 3, 1, 7, 10, 8, 8, 6, 3, 1, 7, 8, 8, 6, 2, 3, 5, 5, 3, 2, 8, 1, 3, 5, 2, 9, 5, 3, 2, 9, 2, 5, 10, 6, 1, 10, 5, 9, 7, 9, 3, 1, 1, 10, 1, 4, 9, 1, 4, 3, 5, 7, 3, 1, 6, 9, 2, 8, 5, 8, 8, 3, 3, 1, 7, 10, 3, 8, 3, 6, 9, 8, 3, 1, 8, 10, 7, 3, 8, 5, 4, 8, 9, 6, 10, 9, 7, 3, 6, 6, 3, 1, 8, 1, 6, 9, 9, 1, 7, 10, 2, 1, 3, 1, 1, 6, 7, 4, 8, 10, 2, 2, 2, 6, 8, 7, 5, 3, 3, 7, 9, 9, 4, 4, 3, 7, 4, 9, 10, 8, 7, 4, 2, 9, 3, 10, 10, 9, 3, 2, 7, 8, 2, 4, 1, 8, 3, 3, 6, 7, 4, 4, 6, 5, 10, 5, 3, 1, 8, 7, 6, 6, 1, 6, 5, 9, 10, 1, 7, 5, 8, 3, 2, 7, 7, 2, 2, 8, 3, 2, 7, 9, 2, 1, 9, 9, 5, 10, 8, 2, 7, 2, 7, 8, 4, 7, 1, 8, 8, 10, 8, 7, 5, 9, 3, 8, 3, 4, 8, 8, 10, 8, 7, 5, 9, 8, 10, 1, 9, 9, 6, 8, 5, 3, 4, 4, 6, 10, 3, 9, 1, 9, 10, 1, 1, 3, 7, 5, 3, 4, 1, 3, 8, 6, 1, 9, 4, 1, 1, 3, 8, 8, 6, 1, 7, 6, 5, 1, 3, 5, 1, 3, 4, 2, 9, 8, 5, 6, 6, 9, 7, 6, 10, 5, 7, 1, 2, 8, 6, 8, 7, 8, 6, 2, 3, 10, 4, 9, 6, 5, 4, 1, 10, 7, 1, 3, 3, 8, 6, 7, 2, 7, 4, 4, 10, 2, 8, 1, 9, 5, 8, 2, 10, 2, 4, 3, 3, 8, 7, 9, 4, 7, 4, 7, 7, 4, 4, 6, 8, 3, 4, 10, 1, 9, 3, 4, 2, 7, 9, 1, 10, 9, 4, 10, 6, 6, 8, 10, 3, 8, 4, 7, 7, 8, 9, 2, 7, 10, 9, 2, 4, 1, 2, 2, 5, 4, 5, 6, 5, 9, 6, 3, 6, 10, 3, 2, 1, 8, 2, 7, 10, 10, 1, 3, 10, 3, 7, 4, 8, 8, 2, 3, 2, 6, 5, 3, 1, 8, 9, 1, 1, 1, 8, 9, 7, 9, 9, 5, 10, 10, 6, 5, 10, 3, 8, 6, 6, 2, 5, 5, 6, 9, 2, 5, 7, 6, 8, 7, 7, 3, 5, 2, 3, 9, 10, 3, 1, 8, 6, 7, 5, 3, 6, 4, 3, 1, 1, 10, 3, 10, 9, 9, 3, 1, 10, 5, 6, 8, 6, 5, 5, 2, 4, 6, 2, 2, 10, 4, 8, 9, 4, 10, 10, 5, 9, 5, 1, 9, 7, 6, 8, 10, 1, 6, 6, 8, 8, 5, 9, 8, 1, 6, 5, 10, 10, 8, 3, 4, 7, 7, 6, 5, 2, 6, 2, 2, 2, 5, 7, 4, 3, 9, 3, 1, 9, 9, 5, 3, 6, 10, 4, 10, 2, 3, 6, 1, 6, 8, 4, 8, 1, 10, 8, 10, 3, 1, 9, 5, 9, 3, 9]

```
1 random.seed(10)
2 b = [ random.randint(1, 10) for _ in range(1000) ]
```

Early Procedural Generation



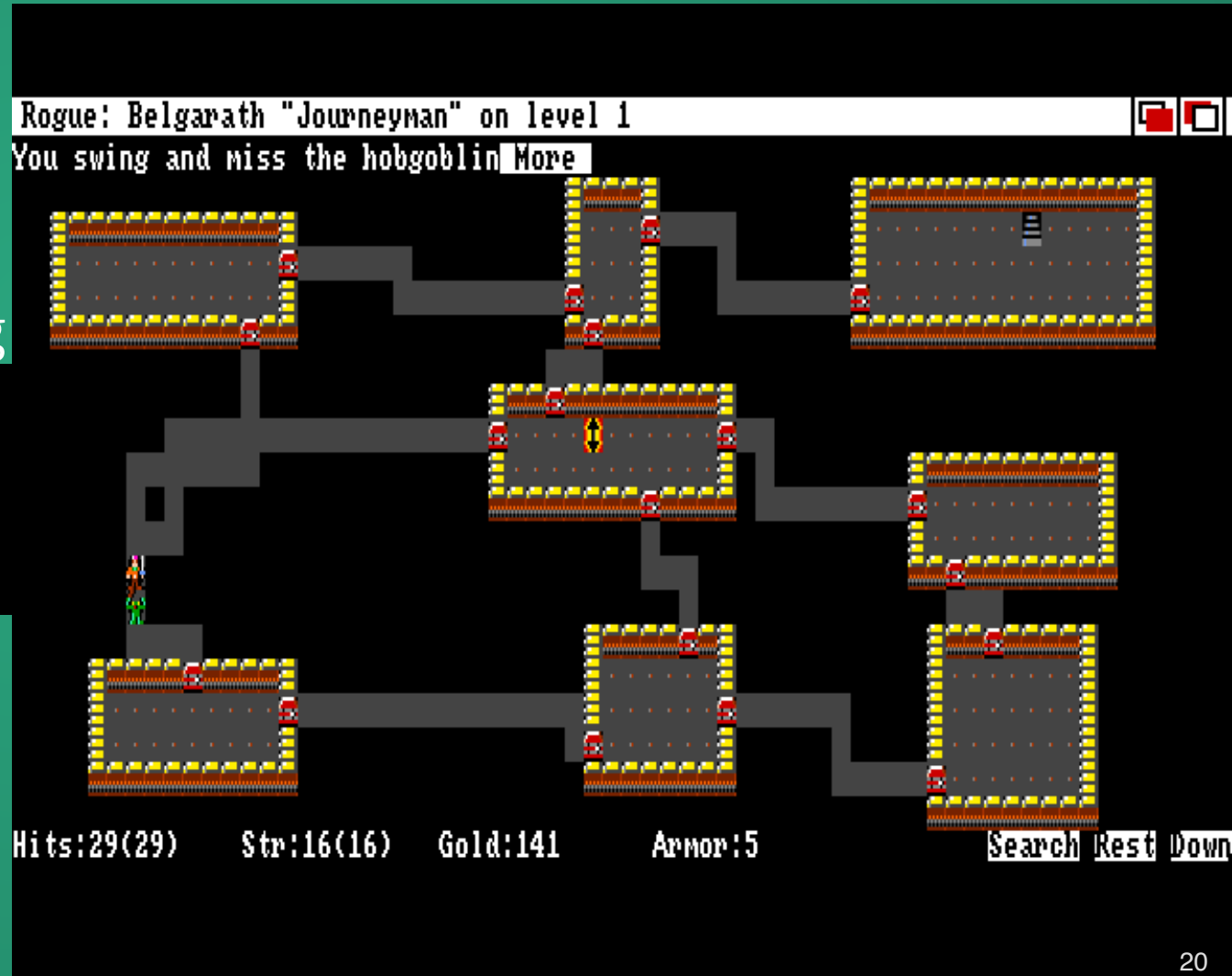
Rogue

A video game written in 1980 at UC Santa Cruz.

Upon death, the player character is completely lost, along with all progress, and a new player character is created.

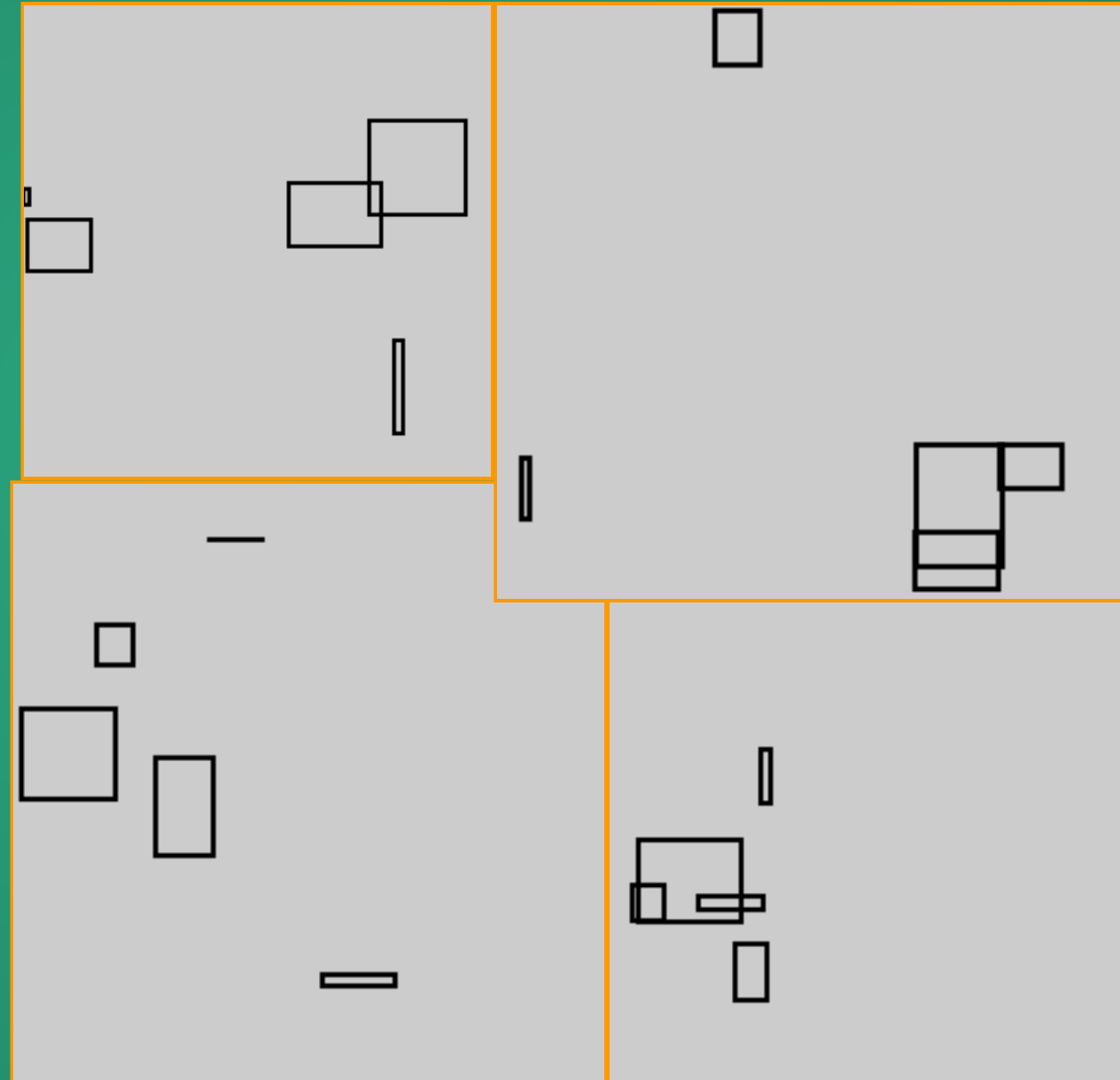
Gave rise to the descriptor of a game being

For various reasons, the dungeons are randomly generated every time the character dies.



How do you generate dungeons in a way that looks and plays nice?

```
1 size(500, 500);
2 noFill();
3 stroke(0);
4 strokeWeight(4);
5
6 for(int i = 0; i < 5; i++){
7   int x = (int)random(0, 400);
8   int y = (int)random(0, 400);
9   int w = (int)random(0, 100);
10  int h = (int)random(0, 100);
11  rect(x,y,w,h);
12 }
```



Worst 4 out of 8 attempts!

Slinging random numbers around in a naive way very rarely results in quality!

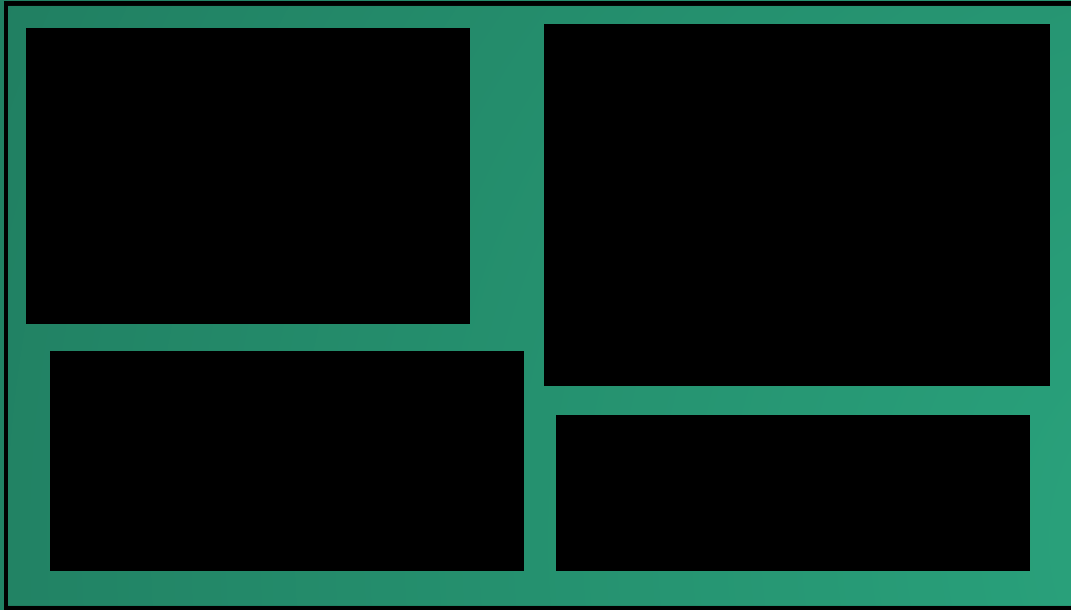
It is *hard* to get a procedural generation system that feels good. Proccgen is a terrible way to be lazy---you'll probably spend 3x as much effort tuning the generation system as creating content to get similar quality.

Attempt 2: Reject rooms that don't meet certain criteria.

```
1 size(500, 500);
2 noFill();
3 stroke(0);
4 strokeWeight(4);
5
6 int numRooms = 0;
7
8 while(numRooms < 10){
9     params = genRandomNumbers();
10    if( intersectsExistingRoom(params) ){
11        continue;
12    }
13    if( areaTooLow(params) ){
14        continue;
15    }
16    if( isOffscreen(params) ){
17        continue;
18    }
19    rect(x,y,w,h);
20 }
```

Problem: how many tries will it take before we can generate all the rooms we need?

We might loop forever!



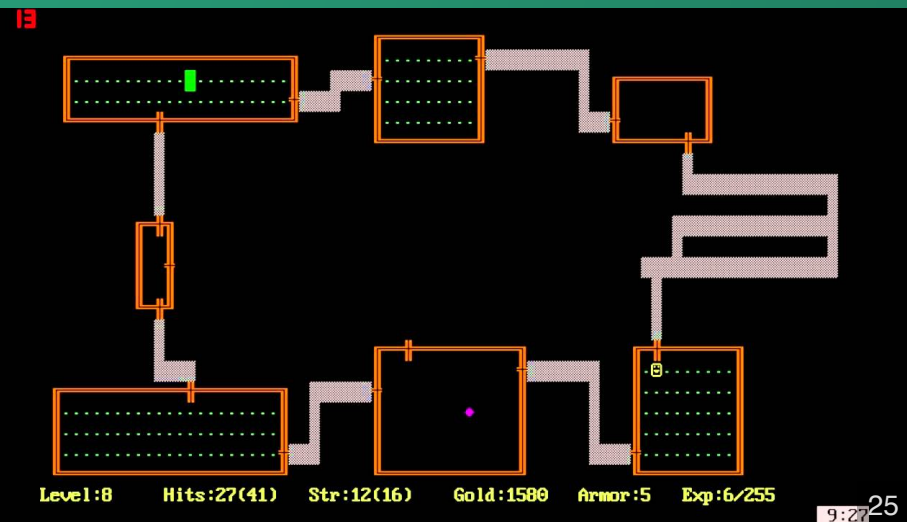
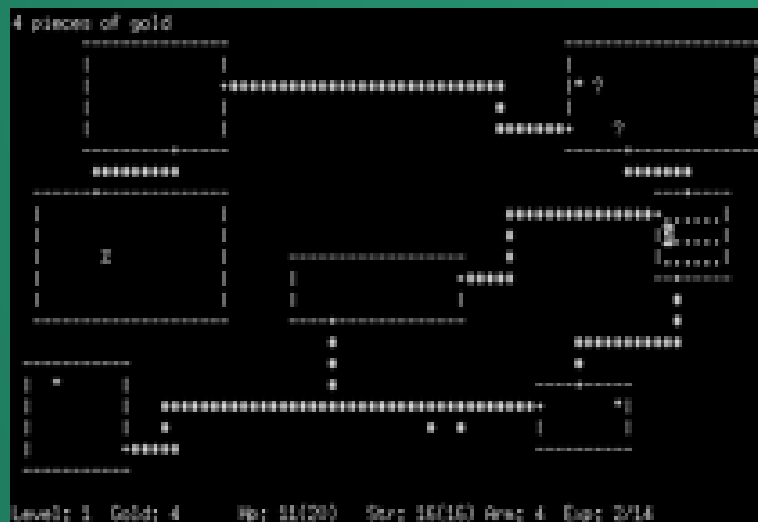
Target: 5 rooms, with
minimum area constraint.

*“ I was using the biggest, fastest computer I
could get my hands on, but it only had 128K
of memory*

Rogue's Solution

Create a 3x3 grid on the screen, and only allow one room per grid cell. Reject that room if it's too small or falls too close to another one (but do not try again).

Then link up rooms to rooms that are close to each other until every room has at least one connection.



Rogue generated a lot of content



Characters, loot, enemies, even traps!

Sadly, the design of many of these systems is not preserved in plain text (you have to read C source code to understand what the game does).

Fortunately, Rogue has generated a huge number of spiritual successors, the so-called roguelike and roguelite genres.

L-Systems

A system to generate strings created by a biologist (Lindenmayer) who was working with fungi and yeast

Consist of a series of rules and a starting symbol.

Input	Output
1	11
0	1[0]0

At each iteration, apply all possible rules to symbols from the previous iteration.

Start: 0

Iter 1: 1[0]0

Iter 2: 11[1[0]0]1[0]0

Iter 3: 1111[11[1[0]0]1[0]0]11[1[0]0]1[0]0

And so on and so forth.

...strings??

1[0]0

Symbol	Meaning
1	Draw a straight line
0	Draw a line terminated by a leaf
[Push current transform and turn left 45°
]	Pop last transform and turn right 45°

1 [0] 0



11[1[0] 0] 1[0] 0

1111[11[1[0] 0] 1[0] 0] 11[1[0]



L-Systems let us draw fractal shapes!

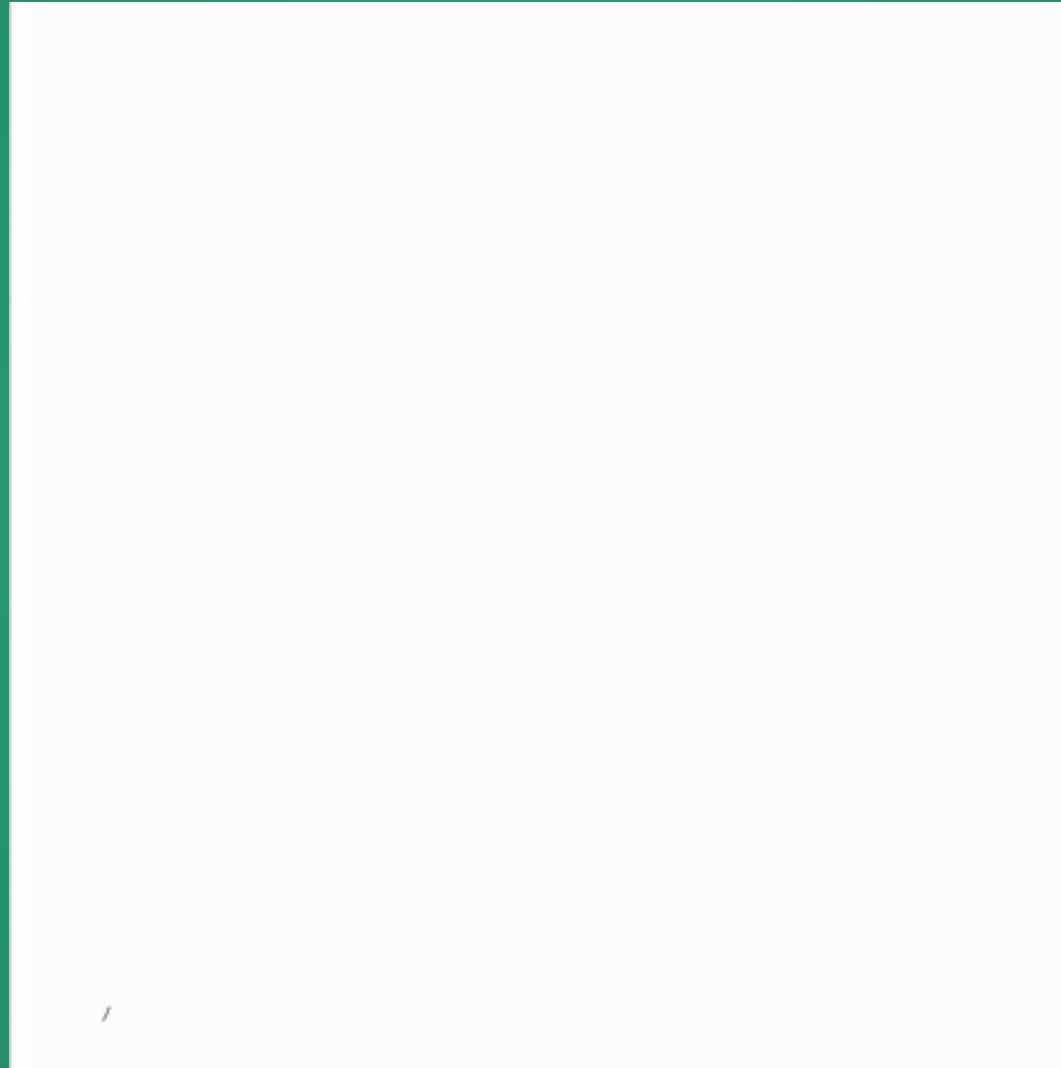
In fact, with very little extra work, we can create some very complex shapes with L-Systems.

Input	Output
X	F+[[X]-X]-F[-FX]+X
F	FF

Symbol	Meaning
F	Draw a straight line
-	Turn right 25°
+	Turn left 25°
[Push current transform
]	Pop last transform

X does not have a meaning for drawing
(only used for the generation phase)

Barnsley Fern



Stochastic L-Systems

As pretty as L-Systems are, they do have the drawback of being deterministic. You can change the number of iterations you want to run, but at each iteration, the shape is the same.

Solution: add *stochastic* production rules.

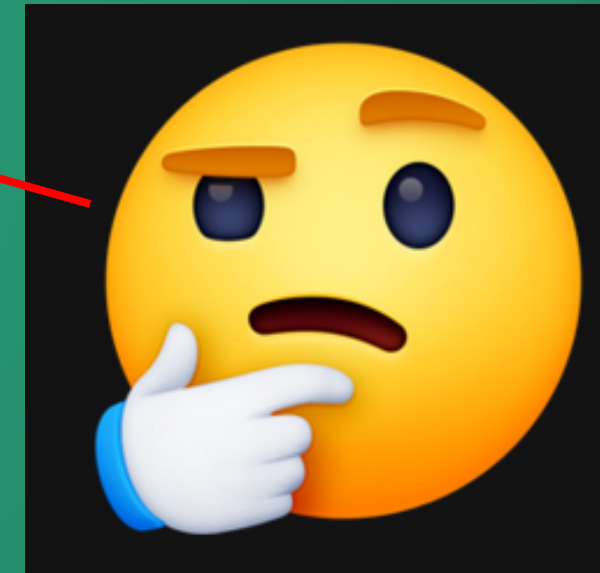
Input	Output	Probability
1	11	100%
0	1[0]0	75%
0	0	15%
0	1	10%

When we see a "0", apply one of the rules w/ appropriate probability.

Input	Output	Probability
1	11	100%
0	1[0]0	75%
0	0	15%
0	1	10%

How can we pick one of the three "0" rules?

```
1 float r = random();
2
3 if(r < 0.75){
4     applyFirstRule();
5 } else if (r < 0.9){
6     applySecondRule();
7 }else{
8     applyThirdRule();
9 }
```



We can save an RNG seed!

So all we need to specify is:

Input	Output	Probability
1	11	100%
0	1[0]0	75%
0	0	15%
0	1	10%

L-System Generation Rules



RNG Seed

Symbol	Meaning
1	Draw a straight line
0	Draw a line terminated by a leaf
[Push current transform and turn left 45°
]	Pop last transform and turn right 45°

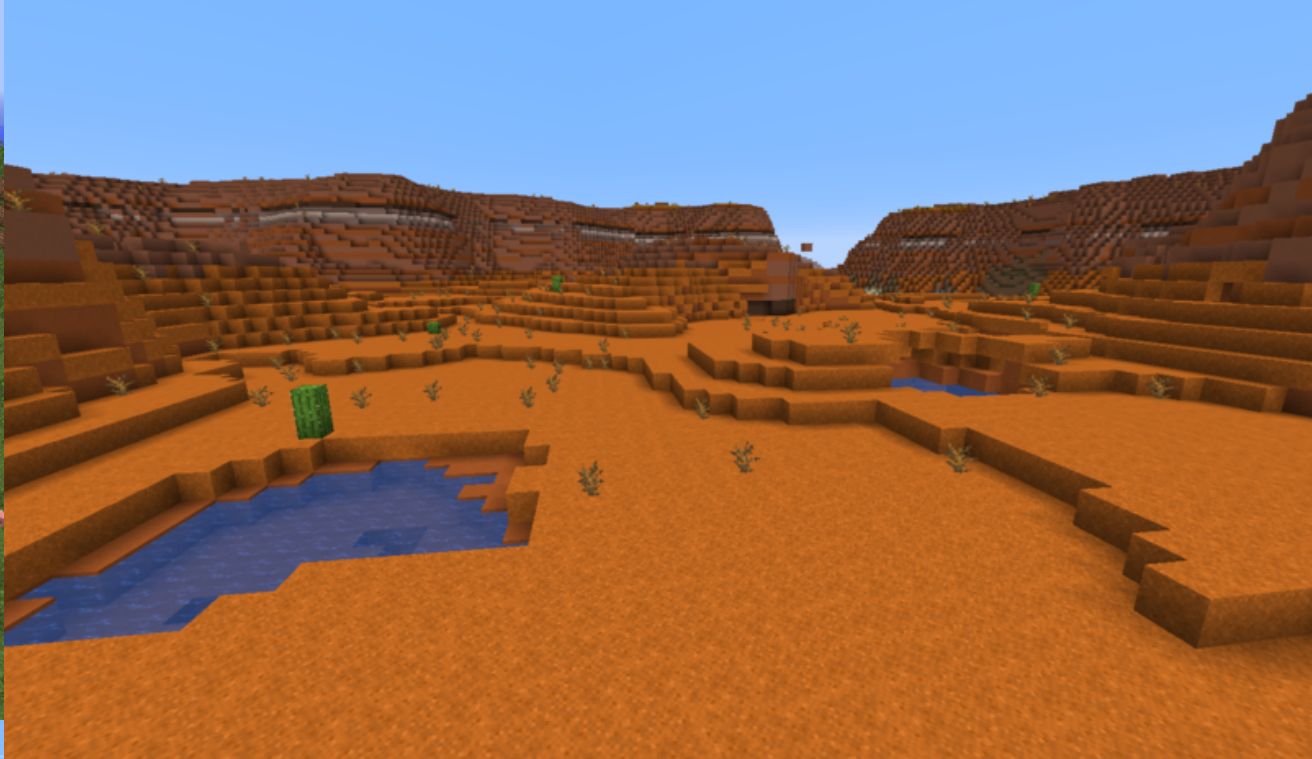
Drawing Rules

...and we can specify *thousands* of geometries like this, each slightly different from the others.



Terrain Generation





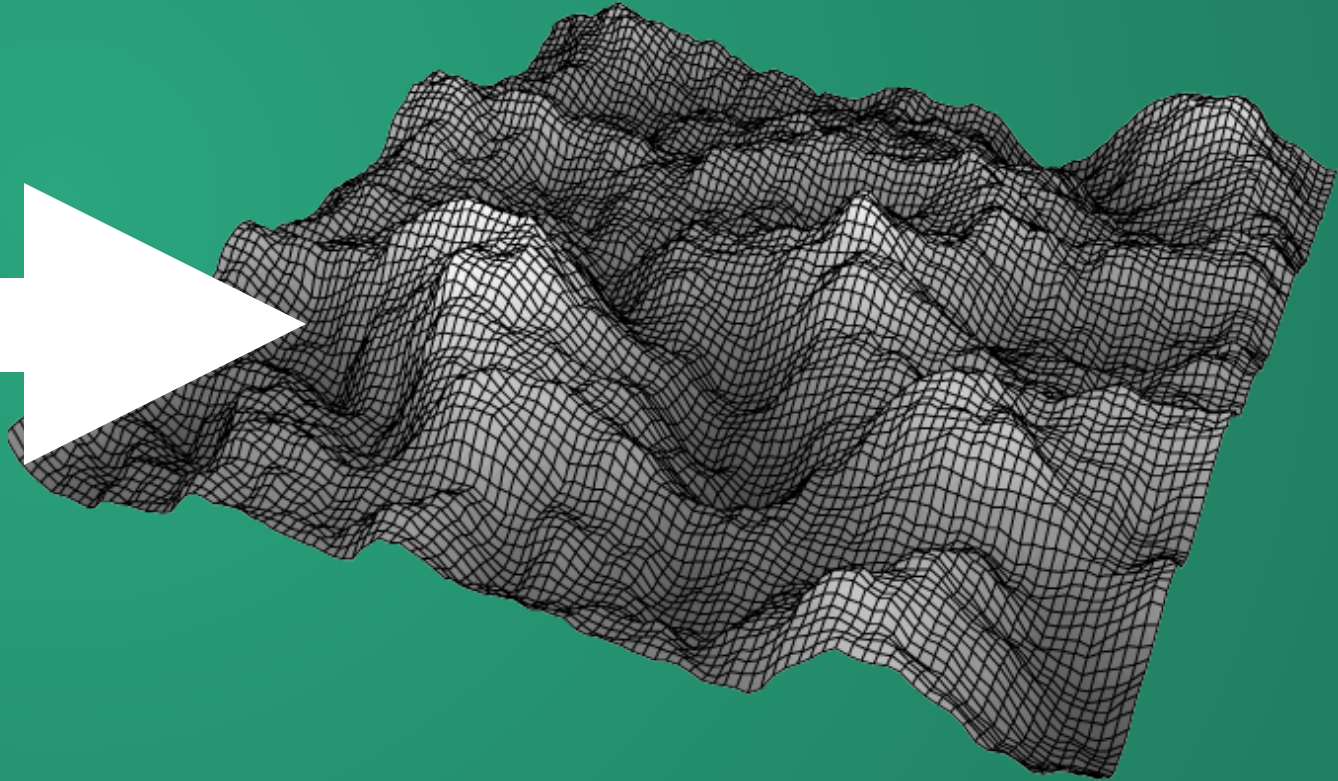
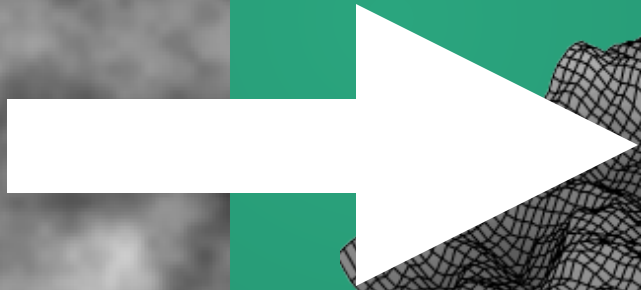
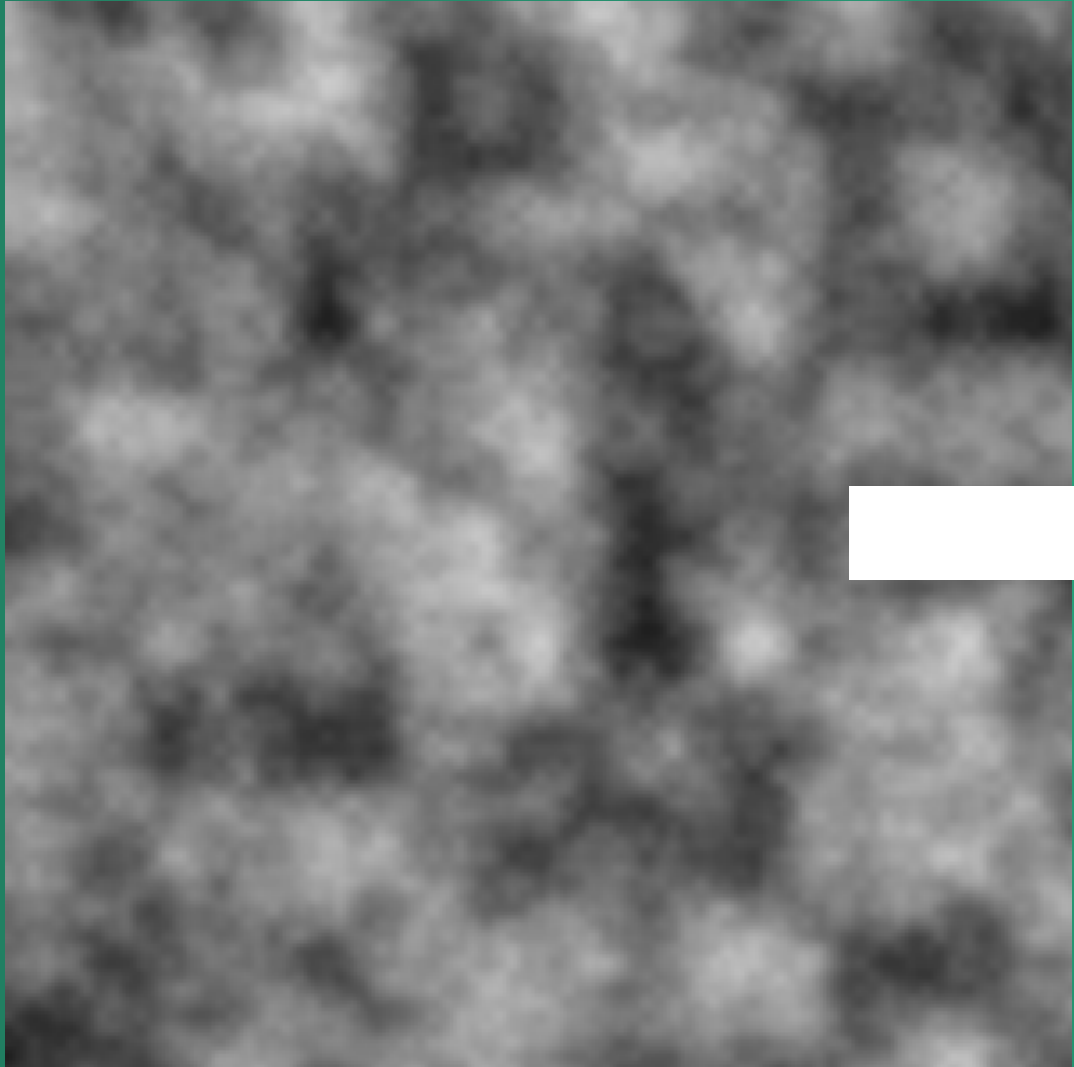
How big is the Minecraft World?

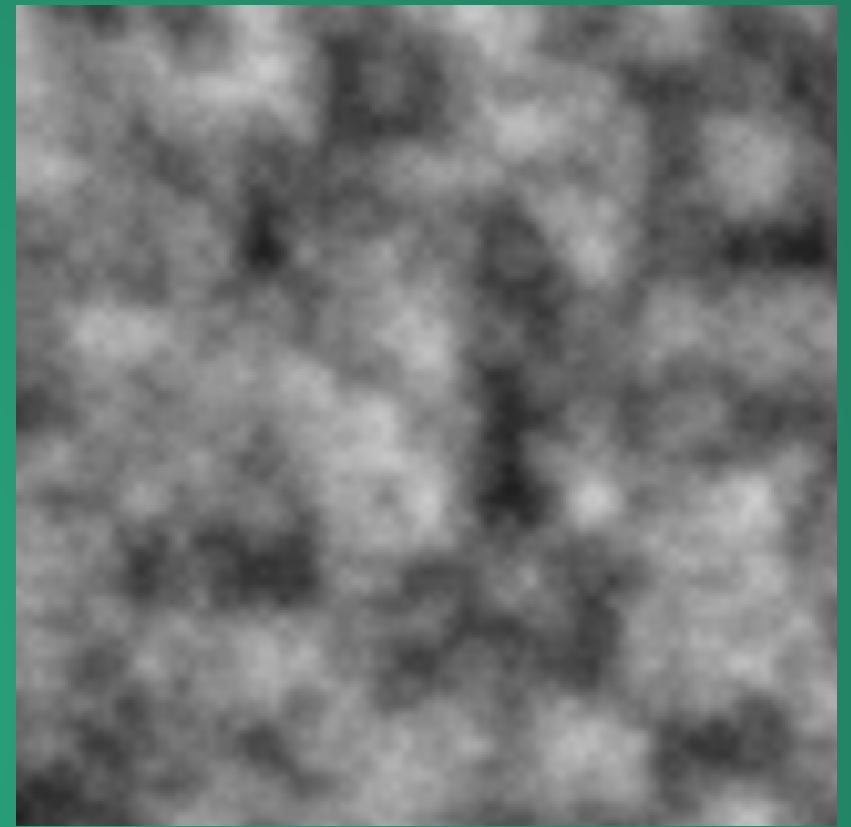
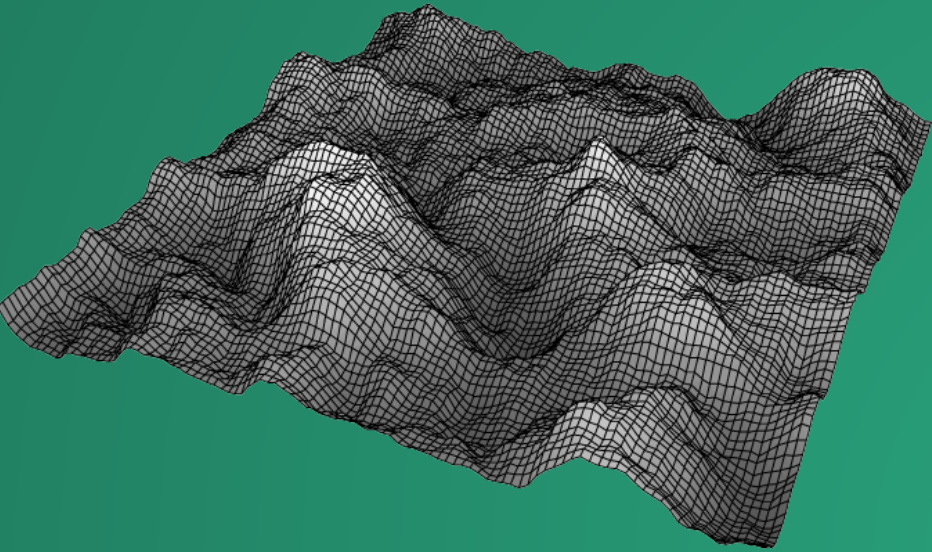
The next layer is the [world border](#), which lies at X/Z $\pm 29,999,984$ by default, and establishes an arbitrary (but capped at this default value) blockade to prevent the [player](#) from advancing. There are [several methods](#) of bypassing this border.

If we can go 29,999,984 blocks in the X or Z direction, then there must be $2 \times 29,999,984 = 59,999,968$ blocks along each side of the Minecraft world. This means there are $59,999,968^2 \times \text{height}$ blocks in the world. The wiki tells us that we can go down to $y=-64$ or up to $y=320$ in the world, so we get 382 blocks in the y-direction. This adds up to 1,382,398,525,440,393,216 blocks in the overworld (plus whatever is in the Nether or the End). The Minecraft wiki lists 876 different block types, which means we would need at least 10 bits per block to store the block type, which gives us just over 13,823,985 terabytes of data in order to store a single Minecraft world in its entirety. At **current hard drive prices**, it would cost approximately \$230 million dollars for this much storage.

Conclusion: Minecraft can only be played by billionaires.

In order to generate terrain like in the Minecraft world, we need some way to generate *noise*.

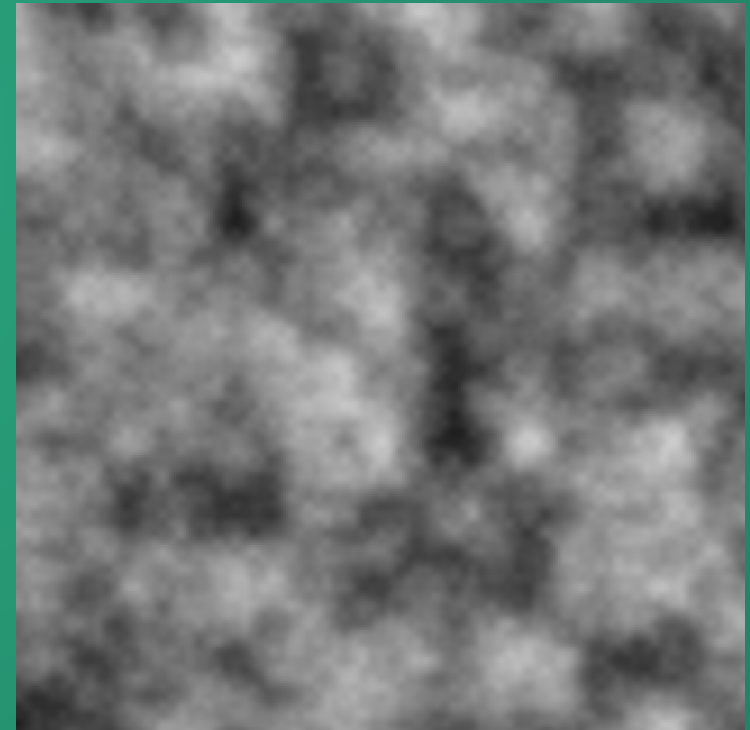
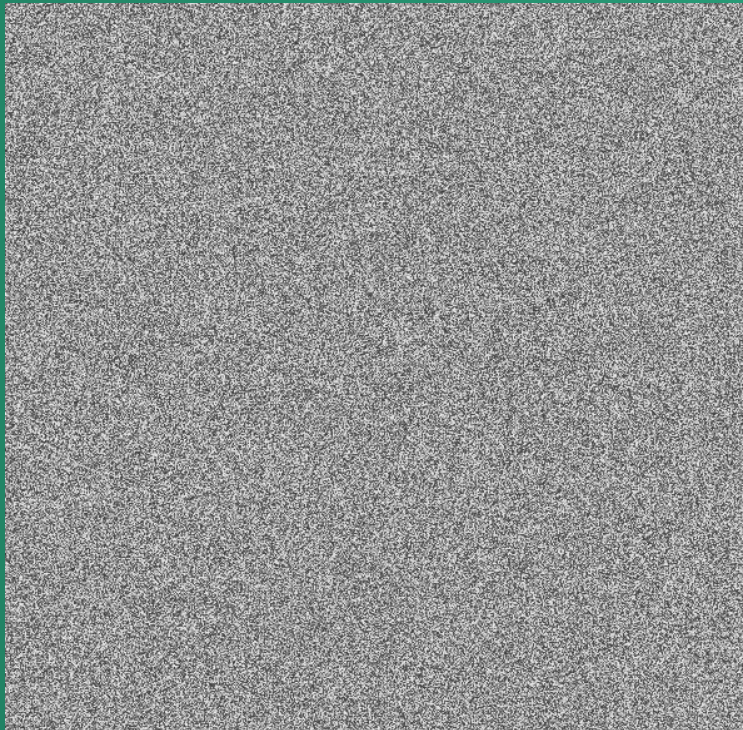


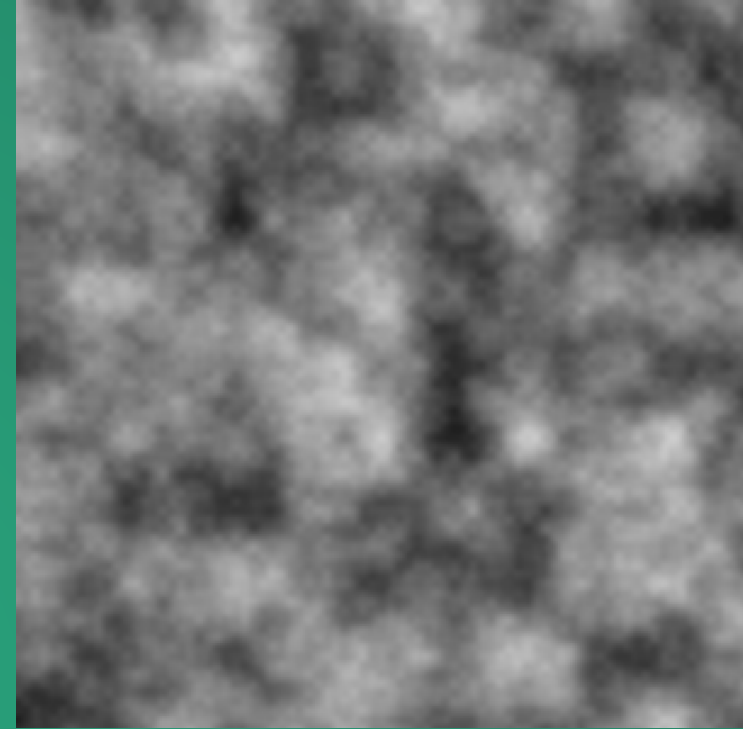
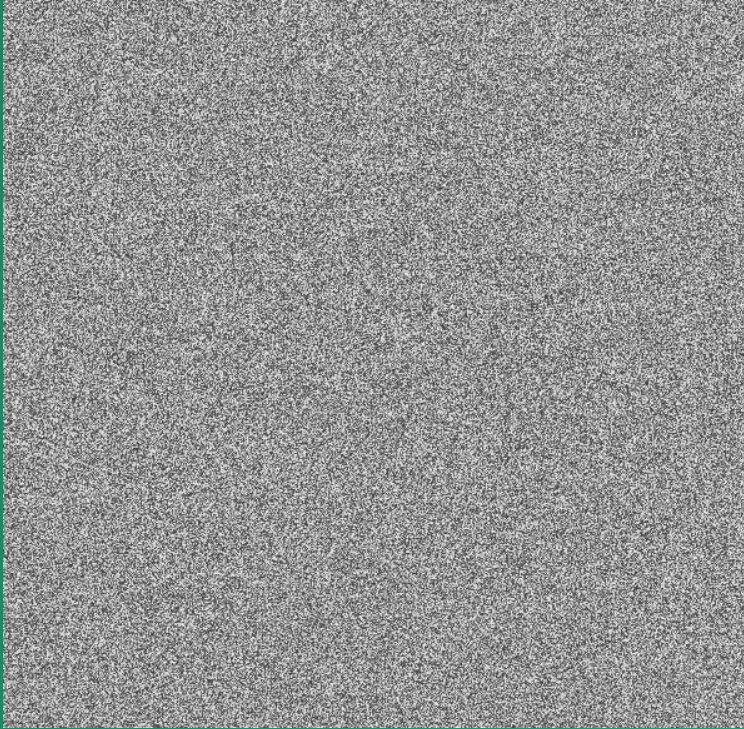


Create another layer of noise for temperature, then use elevation + temperature to decide biome (e.g. low + cold = cold ocean, high + cold = frozen peaks)

So how do we make noise?

Attempt 1: Give each pixel a random color between 0-255.

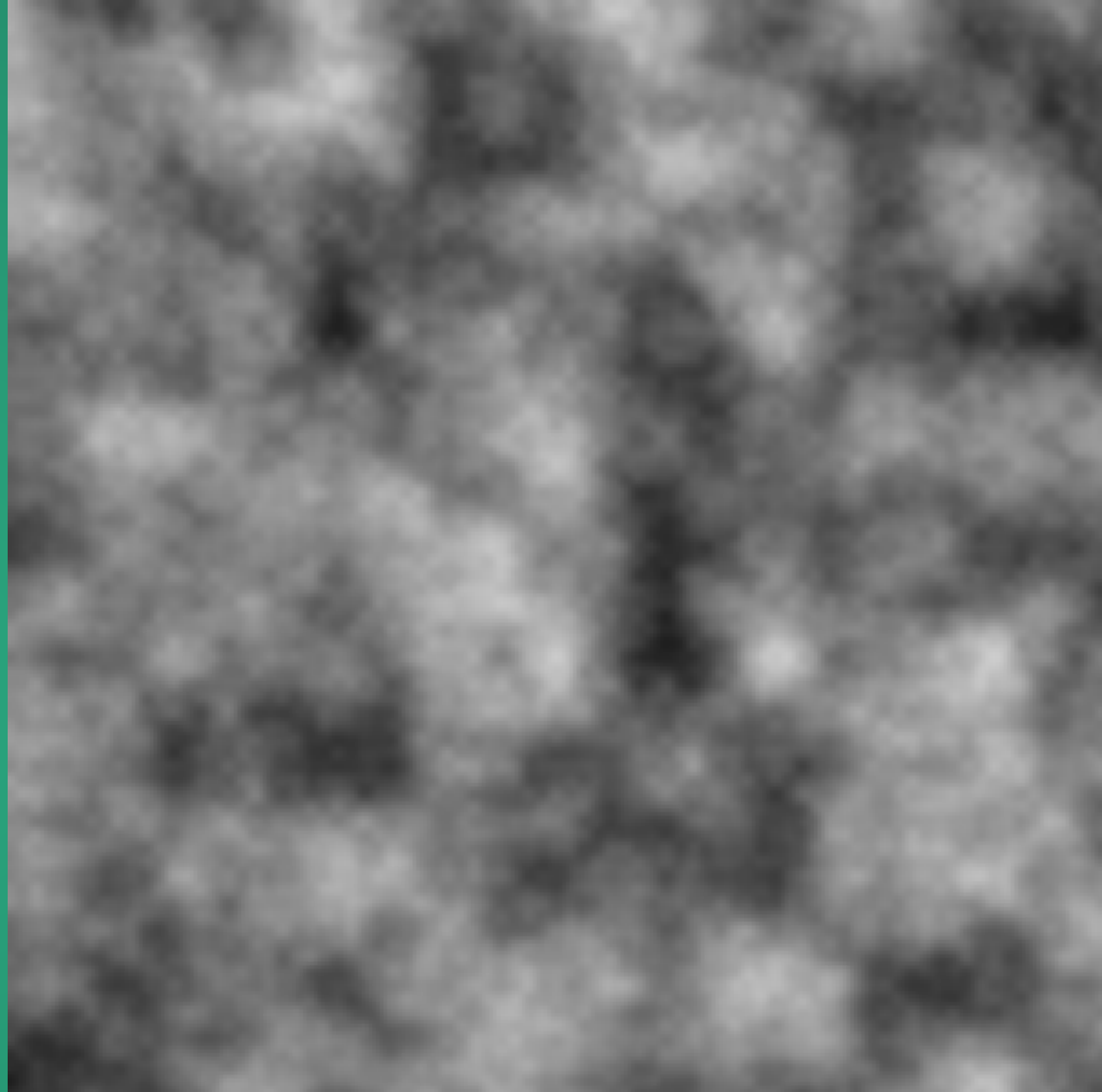
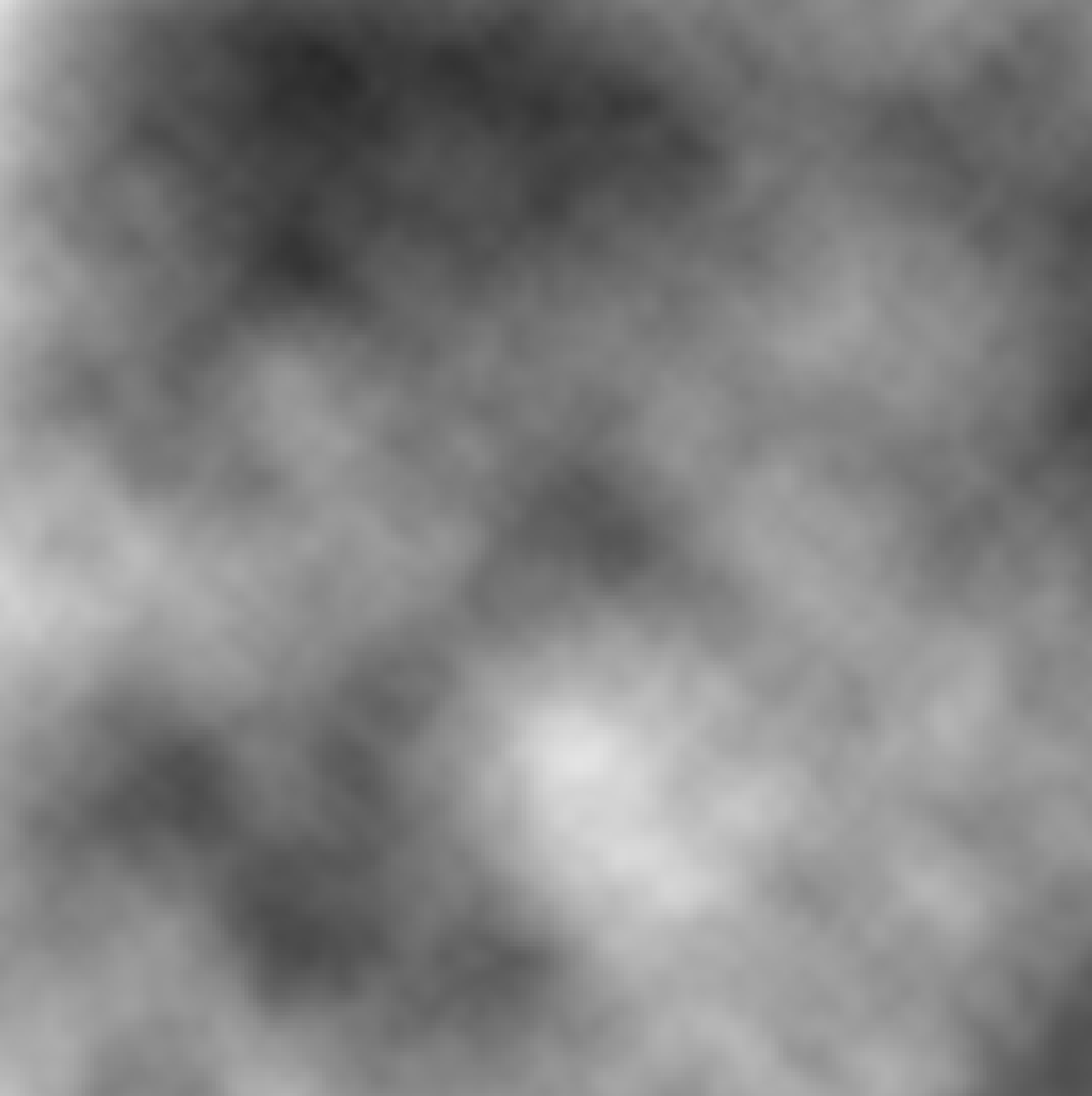




We want noise that's smoothly changing!

Terrain where heights are randomly chosen every 5 feet are not going to be fun for the player to explore, or look very realistic.

We can try to be smarter about how we generate the noise values, but ultimately....

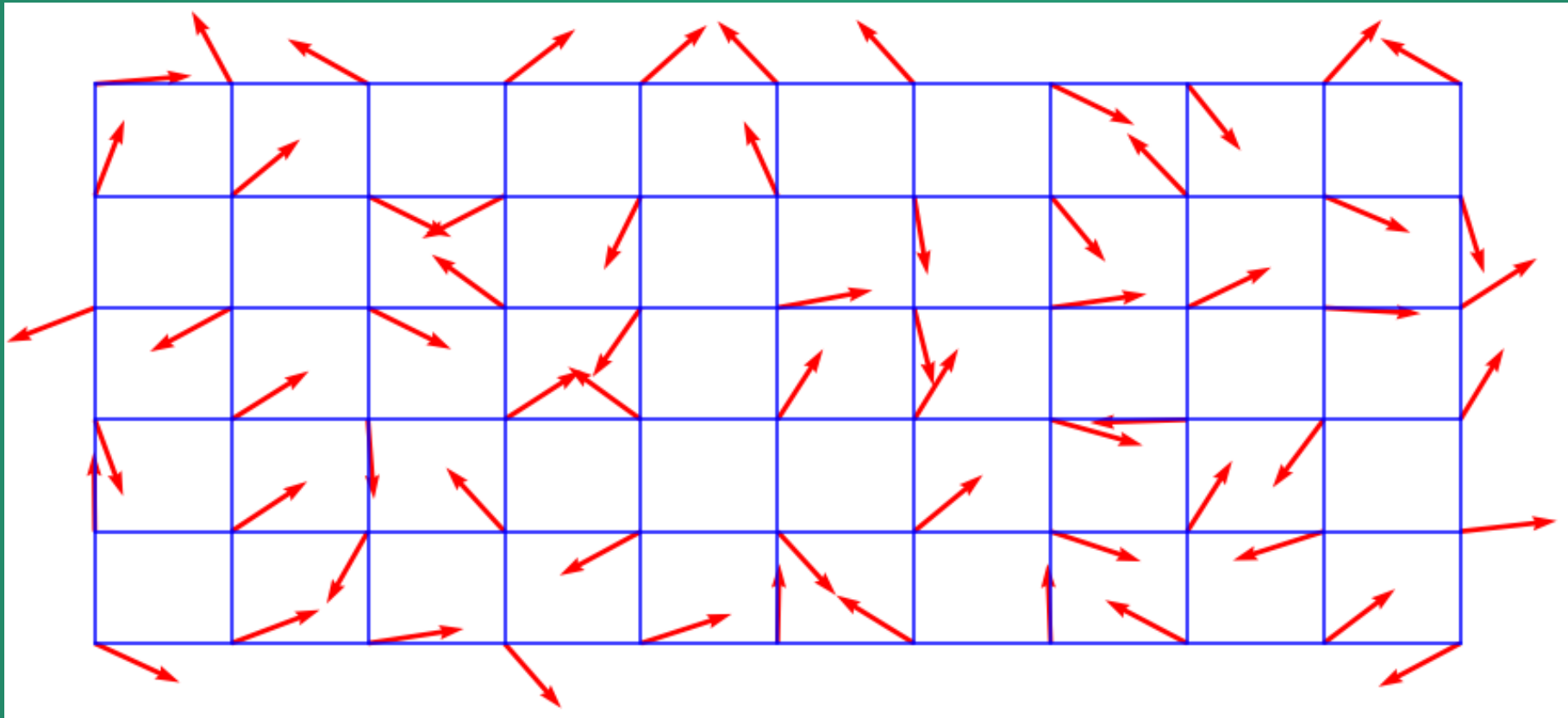


Value noise: notice how there are clear diagonal streaks running through the image, almost like a quilt.

No observable visual patterns.

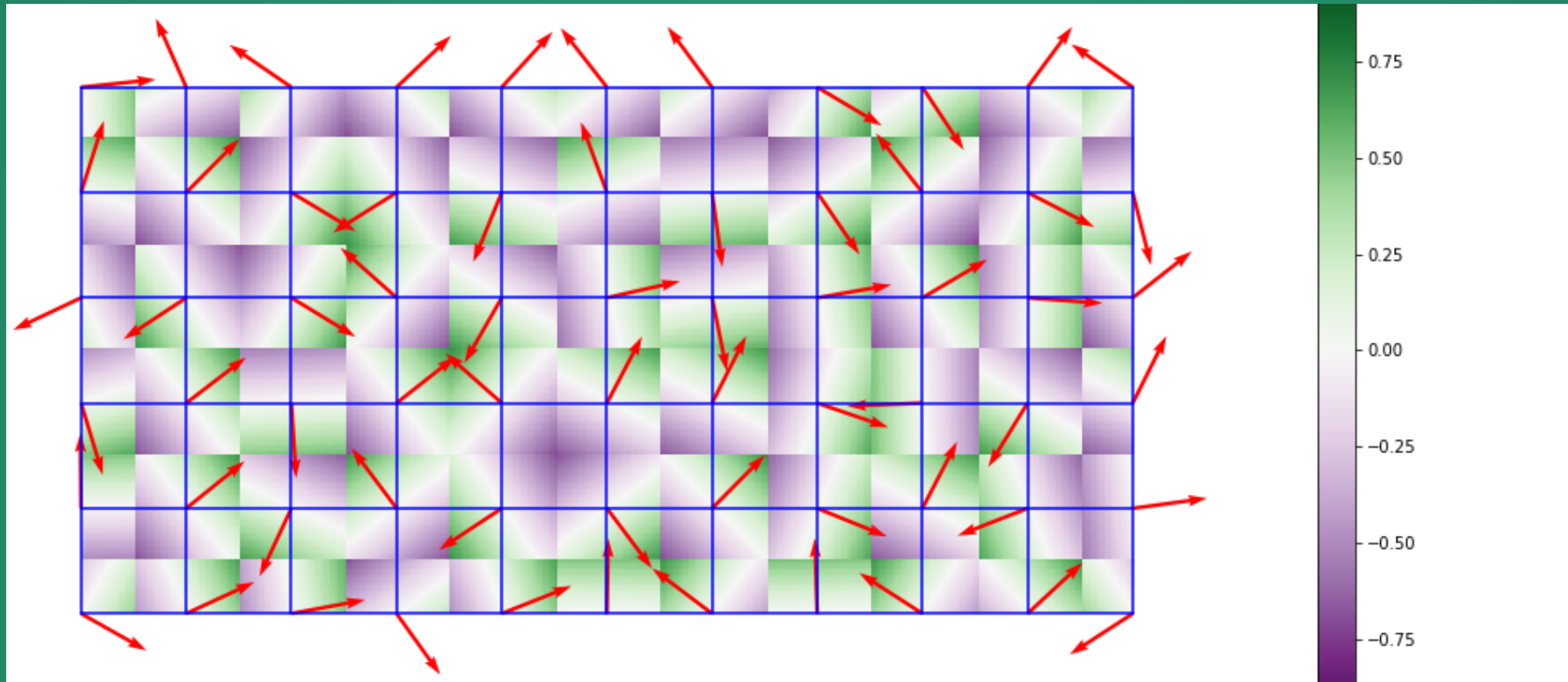
Gradient Noise

Instead of randomizing the *value* of the noise, we randomize its *direction of increase*.



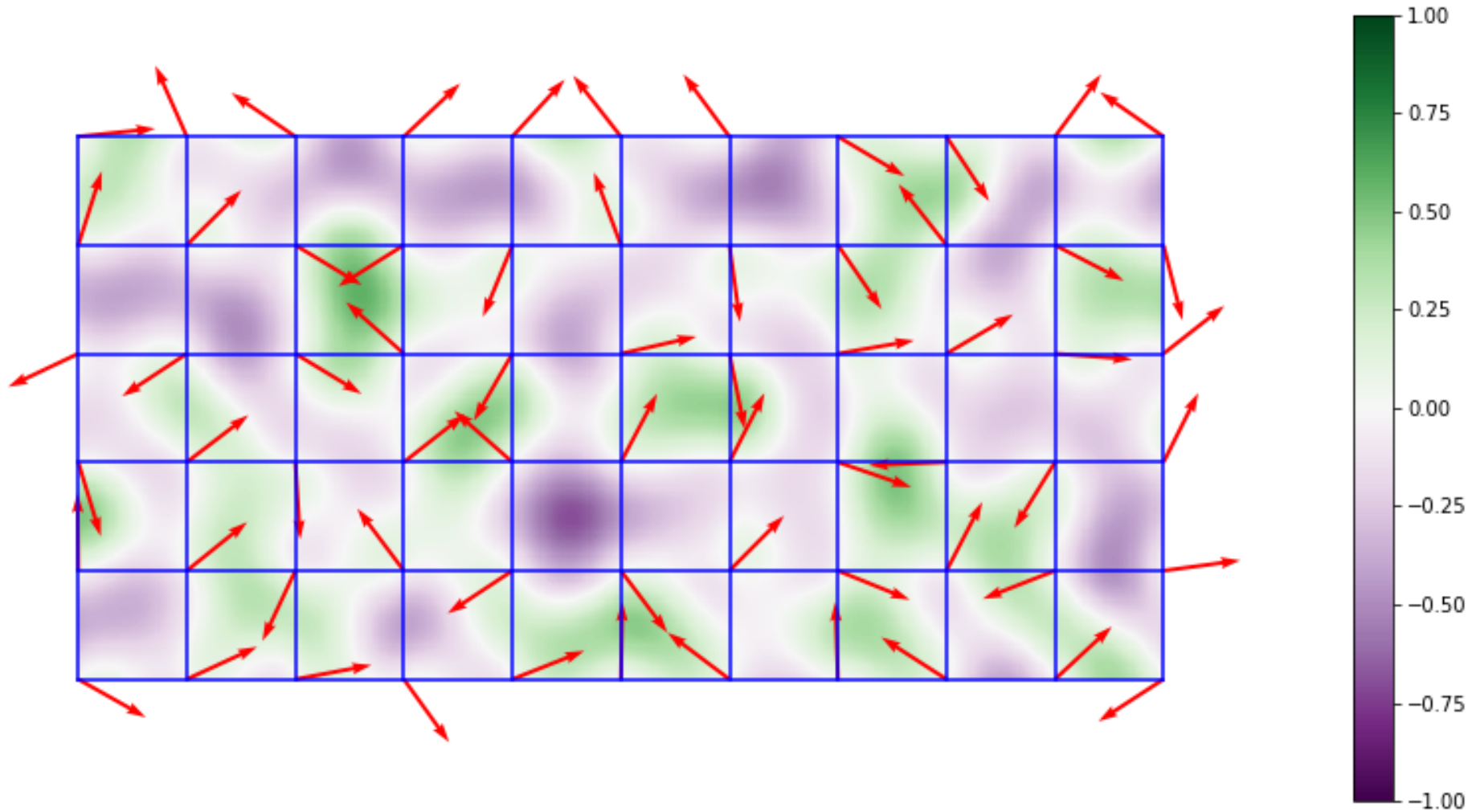
Step 1: Pick random unit vectors on a grid.

Step 2: Determine coarse noise values from gradients.



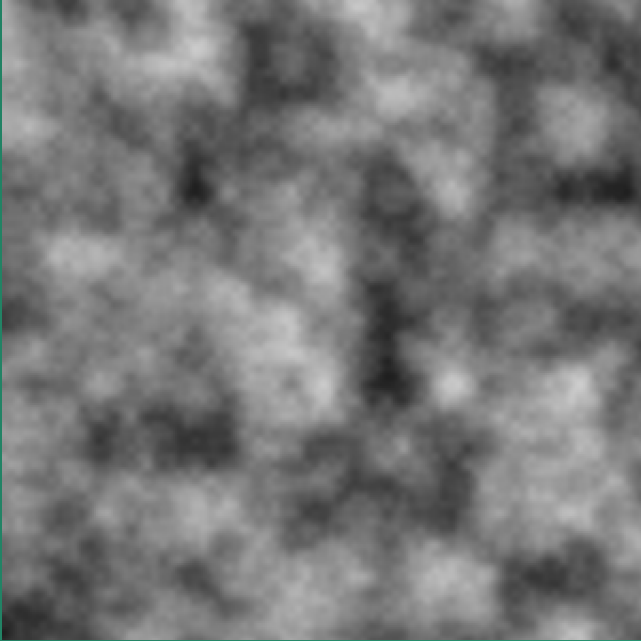
Note that noise value always increases when traveling along a red arrow and decreases when traveling directly opposite it.

Step 3: Determine in-between values from coarse values



How?

Perlin Noise



One of the classic gradient noise functions.

Can be used to generate values over an area:

- Height map
- Temperature map
- Distribution of resources

But still need to enforce logical constraints (e.g. rivers cannot flow over peaks), noise alone cannot account for this.

World Generation: Phase 1

Build a biome map. This uses value noise in pixel-randomness and then applies a series of rules to try to smooth out biome weirdness, e.g. joining landmasses together and avoiding oceanic puddles.

What system does this look like?

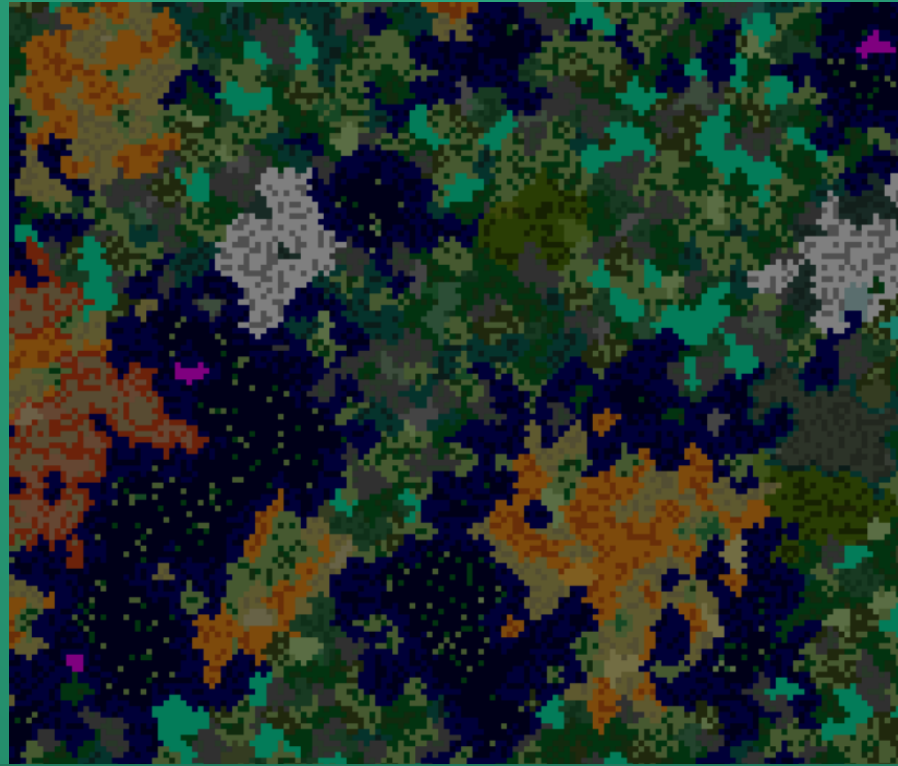
So perhaps a better way to understand how most of them work, is to see them as stochastic cellular automata. Which is just a fancy way of saying that they're simple rules that change a pixel based on the colour of the surrounding ones. The "stochastic" part comes into play because some of those rules might be driven by randomness.

Cellular automata look deceptively simple, but they hide an endless universe of complexity. And are one of the most used tools when it comes to procedural content generation in video games. If you want to learn more about them, there's an entire documentary on my YouTube channel.

World Generation: Phase 2

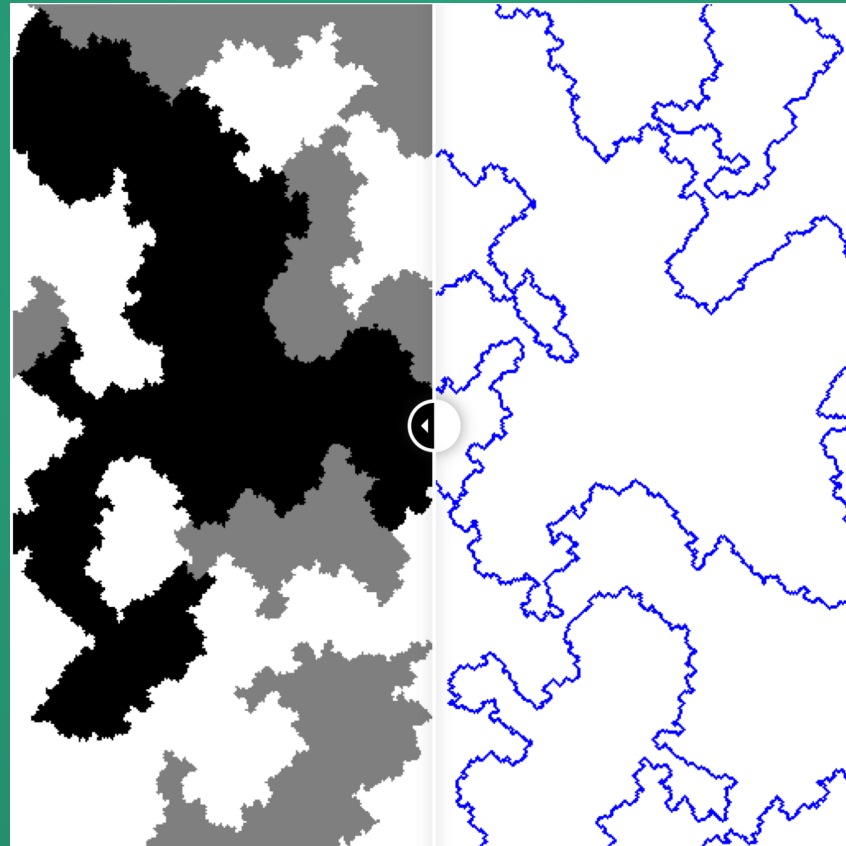
Use Perlin noise to decide on temperatures. Apply another blending layer (another CA) to smooth out temperature differences so that we don't have a snow field next to a hot desert.

Also make rare biomes, hilly biomes, etc. variants here.



World Generation: Phase 3

Now that our map has several large regions, there's a patchy look, where large biomes meet each other. Run edge detection over the biome map (again, Zucconi's words, not mine!) to find "seams" that we can place rivers along.



World Generation: Phase 4

Terrain! Terrain! Terrain!

Use Perlin noise (sampled at several scales) to set the heights of terrain in the world, along with a modified noise algorithm (Perlin worms) to carve out caves and overhangs.



If you want to learn more about Minecraft's terrain generation process

Cannot recommend this article highly enough.

[https://www.alanzucconi.com/2022/06/05/
minecraft-world-generation/](https://www.alanzucconi.com/2022/06/05/minecraft-world-generation/)

Very detailed, relatively accessible explanation (low math!)
and you should recognize many of the concepts in there.

So in Minecraft

Terrain Generation
Algorithm



Random Seed

```
19  unsigned int levenshtein(const vector<unsigned int> &col, const vector<unsigned int> &prevCol, int len1, int len2) {
20  const size_t len1 = col.size(), len2 = prevCol.size();
21  vector<unsigned int> col(len2+1), prevCol(len2+1);
22  for (unsigned int i = 0; i < prevCol.size(); i++)
23  prevCol[i] = i;
24  for (unsigned int i = 0; i < len1; i++) {
25  col[0] = i+1;
26  for (unsigned int j = 0; j < len2; j++)
27  col[j+1] = std::min( std::min( prevCol[i+1] + 1, col[j] +
28  prevCol[j] + (s1[i]==s2[j] ? 0 : 1) ));
29  col.swap(prevCol);
30  }
31  return prevCol[len2];
32  }
33  static void
```

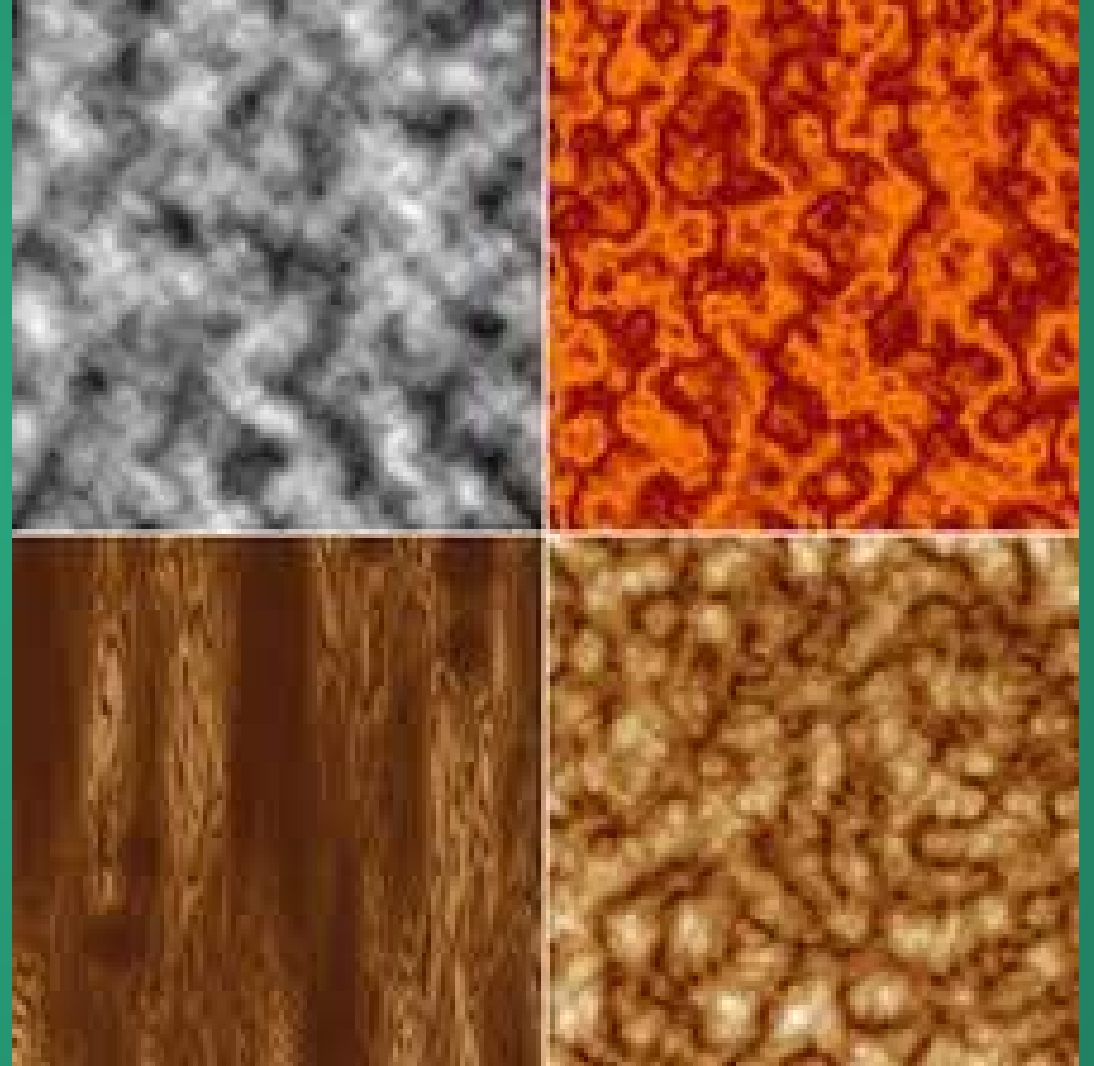


Is enough to store 13 million terabytes worth of info!

In fact, it probably takes more space to record the modifications you've made to the world.

Side Note

Perlin Noise is also really good at generating certain textures!



AI Behavioral Systems

Not traditionally considered part of procgen, but I will argue that there's a case for certain AI systems being part of a procedurally generated system.

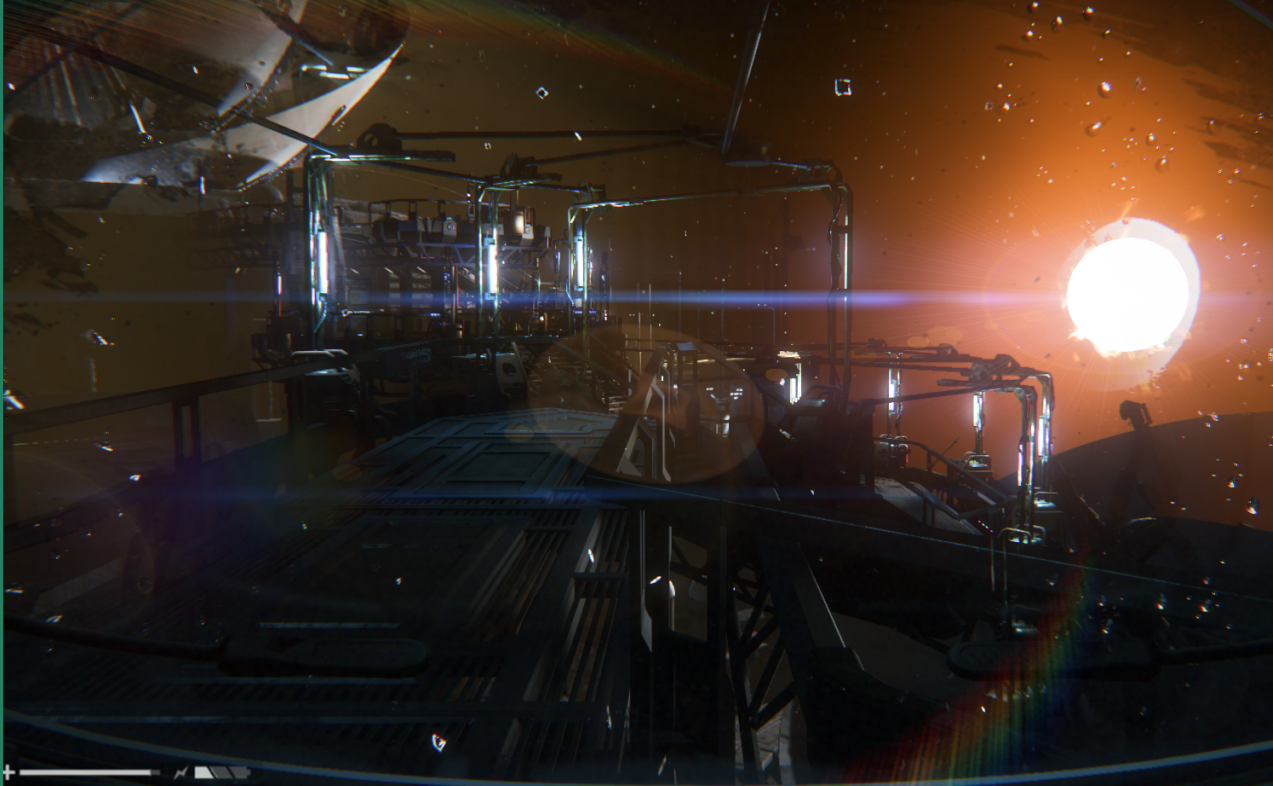


Classic game AI: interaction on timescale of minutes, player learns and adapts to game AI.

AI for procedurally generated experiences: repeated interaction over hours, player adapts to AI, AI adapts to player (or appears to!)



Alien: Isolation



You're an engineer who shows up to retrieve some data from a space station.

Unfortunately, the space station is completely wrecked due to an attack by xenomorphs, and you're trapped on the station.

Now you have to survive while the alien tries to turn you into an hors d'oeuvres.

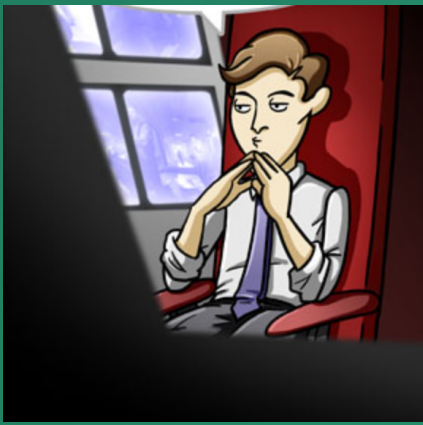
You cannot kill the alien.

What are the challenges when making a monster behave scarily for 20 hours?

- Has to be a threat to the player
- Has to move aside to let player advance every once in a while
- Can't be too scary all the time, or player quits
- Can't be too effective, or horror is replaced by frustration



Alien Behavior



Two-level system



"Steve"

Director AI

- Knows where you are and what you're doing at all times.
- Goal: create a tense experience without overdoing it (leading to frustration/ragequit)
- Can give the alien AI high-level directions.

Alien AI

- Is lonely :(
- Does not know where you are and must search for you with sensors.
- Has a complex *behavior tree* which determines how it behaves.
- Just wants to give your face a big old hug. With its teeth.

The Director

A concept which first appears in Left 4 Dead as a way to manage tension of the player.

General cycle:

- Send the alien to a location near the player and tell the alien to begin hunting.
- Monitor the situation and keep track of how tense the situation is. Things that add to tension:
 - Proximity to player
 - Visibility of alien to player
 - Presence of attack path from the alien to the player
- Once things are too tense, send the alien away from the player so that the player can relax a little bit and the game can progress.

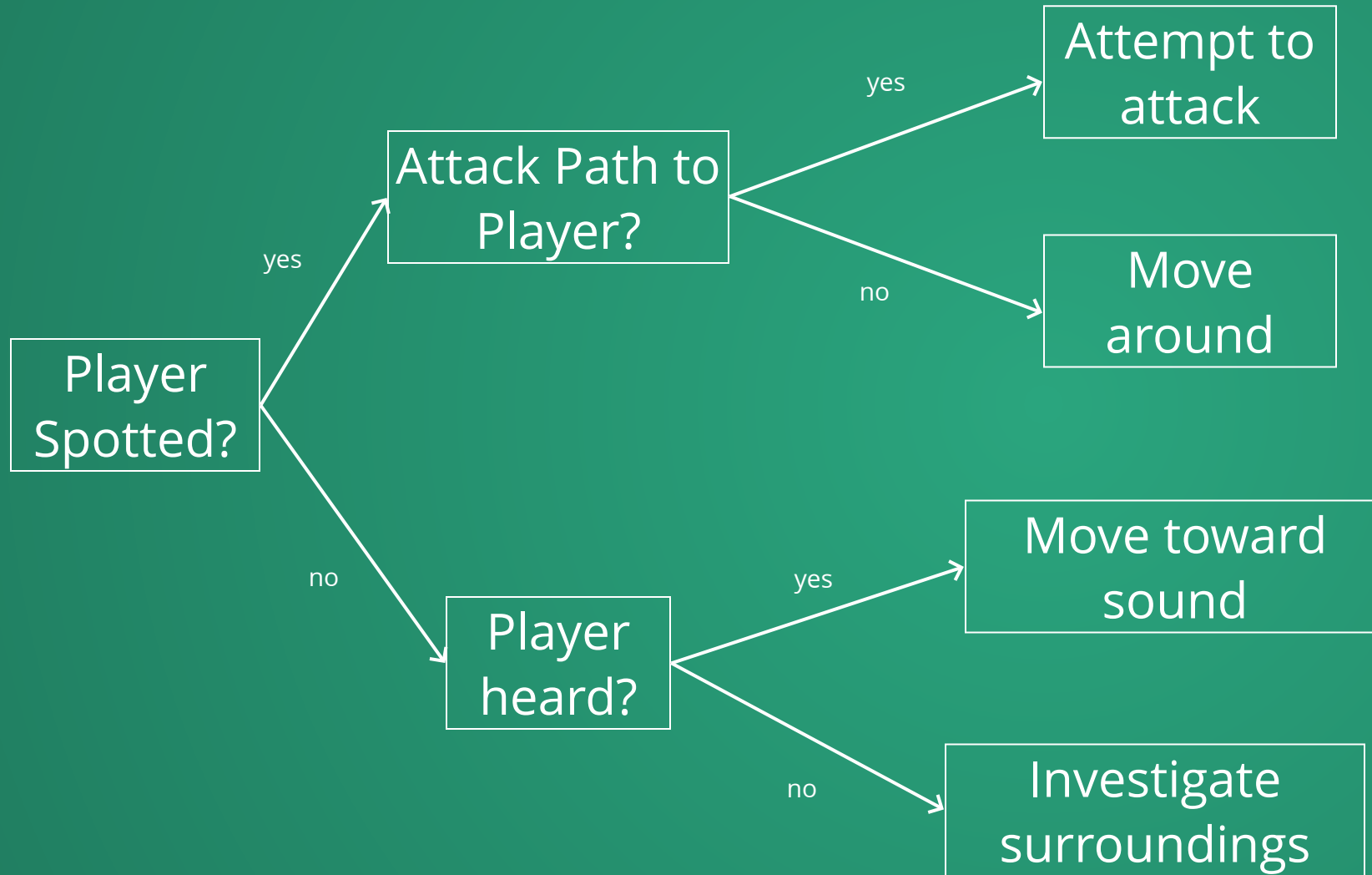
The Alien

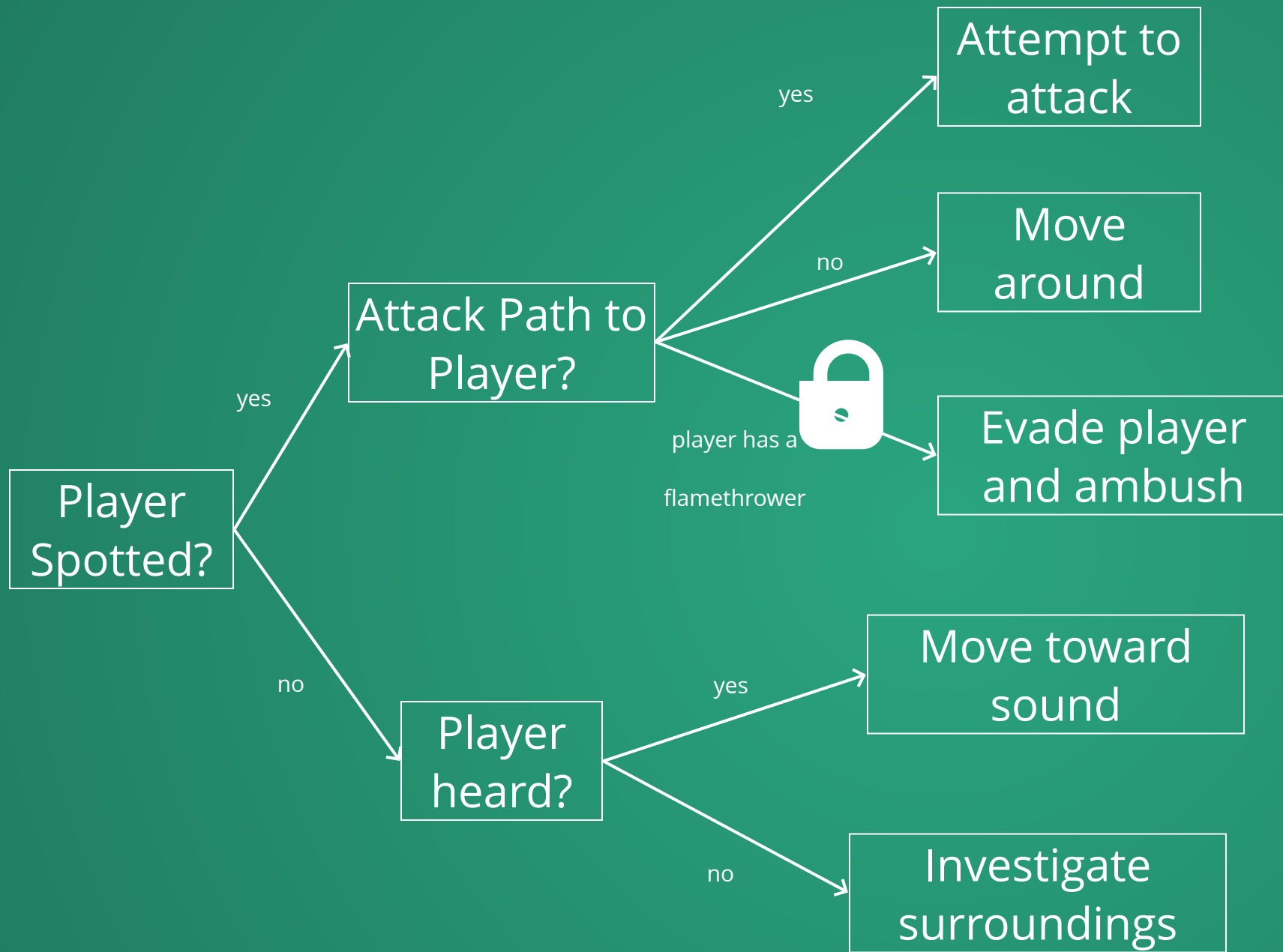


Fun fact, in early builds of the game, playtesters would run up behind the alien and stay right behind it to avoid detection. To avoid this, the developers placed an "eye" on the alien's tail.

- Alien has several visual/auditory sensors it can use to detect the player.
- Different stimuli are detected at different rates
 - Player is detected slowly (even more slowly if smoky/foggy in the room)
 - Flares, flashlights, and gunfire instantly trigger detection
- Alien uses a *behavior tree* to decide how to respond to what its sensors are telling it.

The Alien: Behavior Trees





Branches of the behavior tree are locked off initially and unlocked as the game progresses, either due to player action or game progress.

Example: Once the alien has chased enough flares, it unlocks a branch of its behavior tree that ignores flares and keeps hunting for the player.

Combined with a sufficiently complex behavior tree, gives the impression of a thinking, learning enemy.

Many, *many* additional details (e.g. how the alien AI is allowed to detect you, how quickly it picks up on clues, etc.) meticulously hand-tuned by the developers while making the game.

Result



An experience which often *feels* handcrafted for the player, even though it is utterly unique.

Procedural Generation lets us create assets, worlds,
and even behaviors for each player individually.

Things that we've seen generated

- Levels
- Plants + Geometry
- Terrain + Minecraft World
- Player Experience

But there are lots more ideas in each category that we haven't covered, and a few categories that we've still skipped!



Borderlands procedurally generated many of its items and storyline quests.

(arguably with mixed results)

No Man's Sky attempted to procedurally generate wildlife, including their movement.



What about Generative AI?

A crucial component of procedurally generated content is that it is *cohesive*: that it fits well with other content in the game.

Generative AI has only begun to be able to do this, and it's not clear that it can be consistently consistent without lots of engineering effort.

Exciting work on this front, but no major games using stable diffusion or LLMs for *procedural* generation yet.

FIN

More Info

Rogue and RNGs

Random Number Generation Introduction

<https://www.freecodecamp.org/news/random-number-generator/>

If you want a deeper dive on RNGs

<https://www.pcg-random.org/>

Note: The author of the technical pieces is the creator of the PCG (a particular RNG algorithm), and is naturally biased in favor of it, but the explanations on this site and its blog are still pretty good.

Rogue Overview

<https://www.alanzucconi.com/2022/06/05/minecraft-world-generation/>

Story of Rogue's Creation

Dungeon Hacks by David L. Craddock

UT has infinite free access through the library,

https://search.lib.utexas.edu/permalink/01UTAU_INST/be14ds/alma991058320981006011

Perlin Noise

Perlin Noise Overview

For a quick overview of Perlin noise, the Wikipedia article is pretty good:

https://en.wikipedia.org/wiki/Perlin_noise

Perlin Noise Implementation

For a more in-depth view of implementing Perlin noise, try this article:

<https://adrianb.io/2014/08/09/perlinnoise.html>. Unfortunately, there are few resources on Perlin noise generation that don't involve some level of mathematics.

Simplex Noise

Ken Perlin introduced an improved variant of Perlin noise with fewer directional artifacts and faster generation in higher dimensions. It is known as simplex noise. https://en.wikipedia.org/wiki/Simplex_noise

Minecraft World Generation

World Generation

Alan Zucconi's article is a fantastic dive into the world generation algorithm, including its phases, how Perlin noise is used, and even some modifications that happened in 1.18. <https://www.alanzucconi.com/2022/06/05/minecraft-world-generation/>

World Generation API

If you would like to see what it might look like to modify world generation yourself, take a look at the Sponge docs. Sponge is a Minecraft server which lets you write custom plugins in Java, so the code should be somewhat familiar.

<https://docs.spongepowered.org/5.1.0/en/plugin/wgen/customwgen.html>

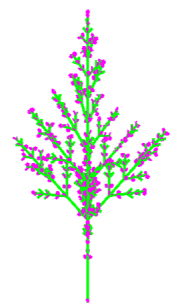
L-Systems

Play with L-Systems

<https://lssystem.club/> has a short tutorial on deterministic and stochastic L-systems, along with visualizations of various L-systems that you can modify and play around with.

lilac inflorescence

```
/(90) -(180) f(120) +(180)
#(120) !(2) A~K
A: [-/~K][+/~K]I(0)
/(rnd(0,90))A
I(t) {t!=2}: FI(t+1)
I(t) {t==2}: I(t+1)[-FFA]
[+FFA]
K: [#(300) !(0.9) F
E(10,300)]
```



+
-
up
dw
lf
rt
lr+
lr-
e+
e-

edit Iterations: 14

Usage in Houdini

Houdini, a 3D modeling/animation application, has an L-System module for rapidly generating geometry. You can see its documentation at <https://www.sidefx.com/docs/houdini/nodes/sop/lssystem.html>

Alien: Isolation

Articles:

- <https://www.gamedeveloper.com/design/the-perfect-organism-the-ai-of-alien-isolation>
- <https://www.gamedeveloper.com/design/revisiting-the-ai-of-alien-isolation>

Videos:

- <https://www.youtube.com/watch?v=Nt1XmiDwxhY>
- <https://www.youtube.com/watch?v=P7d5IF6U0eQ>

Other

<http://pcg.wikidot.com/> has algorithmic details, papers, and blogs on procedural content generation.