Procedural Environment and Architecture Generation in Windforge

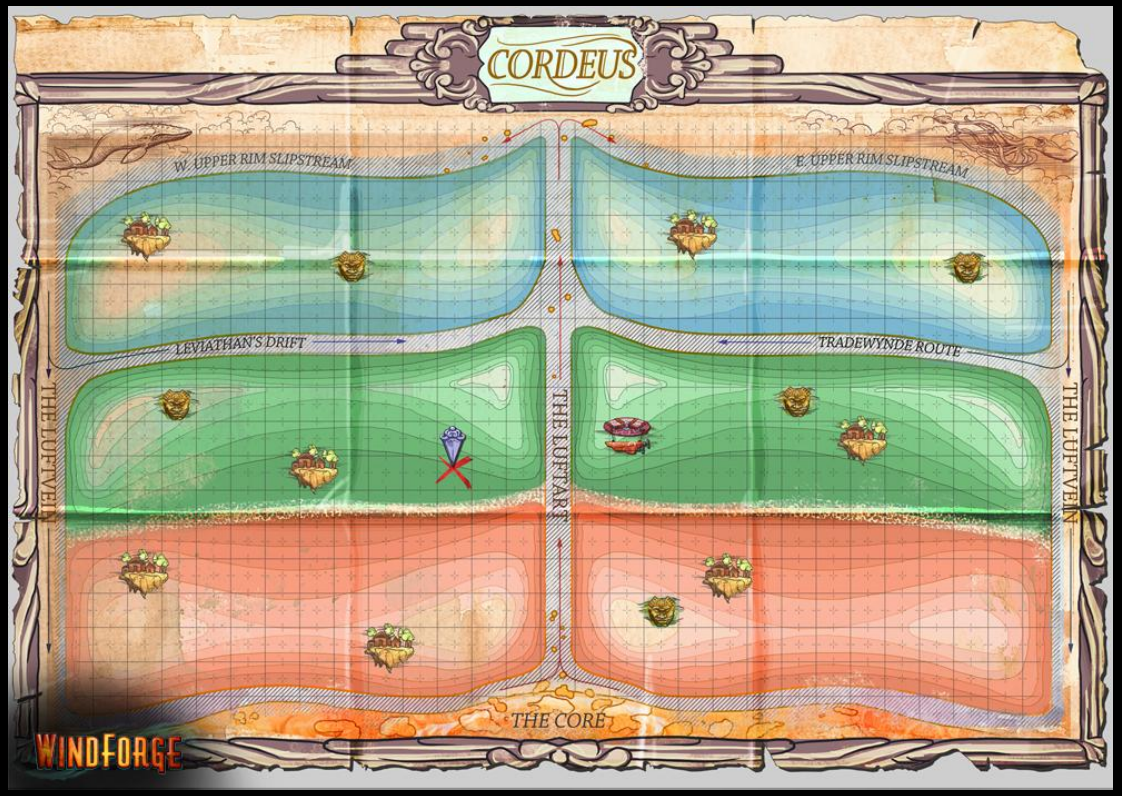
What is Windforge

Screenshot from Windforge

What is Windforge

- A 2D action building block RPG
- Embraces freedom and creativity
- Almost everything in the world can be created and destroyed
- Creating this world by hand would be very tedious and time consuming.

Windforge: Procedural Generation



Almost the entire world is procedurally generated

Windforge: Procedural Generation

Some benefits of procedural generation:

- Can create large sandbox worlds with ease
- Easy to balance and tweak the world
- Great replay value
  - (even for the developers!)
- Emergent results are possible

Windforge: Procedural Generation

Some drawbacks / challenges:

- Can be unpredictable
- Testing / debugging is challenging
- Hard to compete to compete with real people.

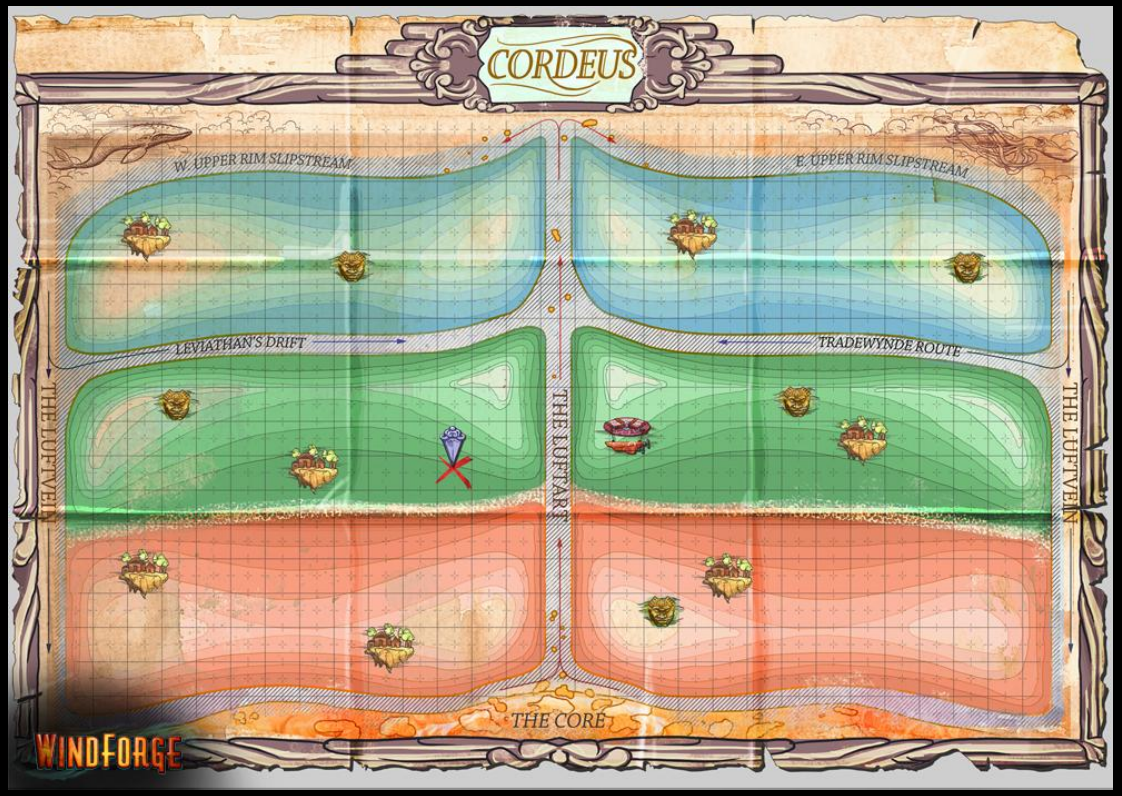
Windforge: Procedural Generation

Things that are procedural in Windforge:

- Area definition and landmark placement
- World areas
- Dungeons
- Enemy placement and spawning

World Area Generation

Windforge: World Generation



The World Map

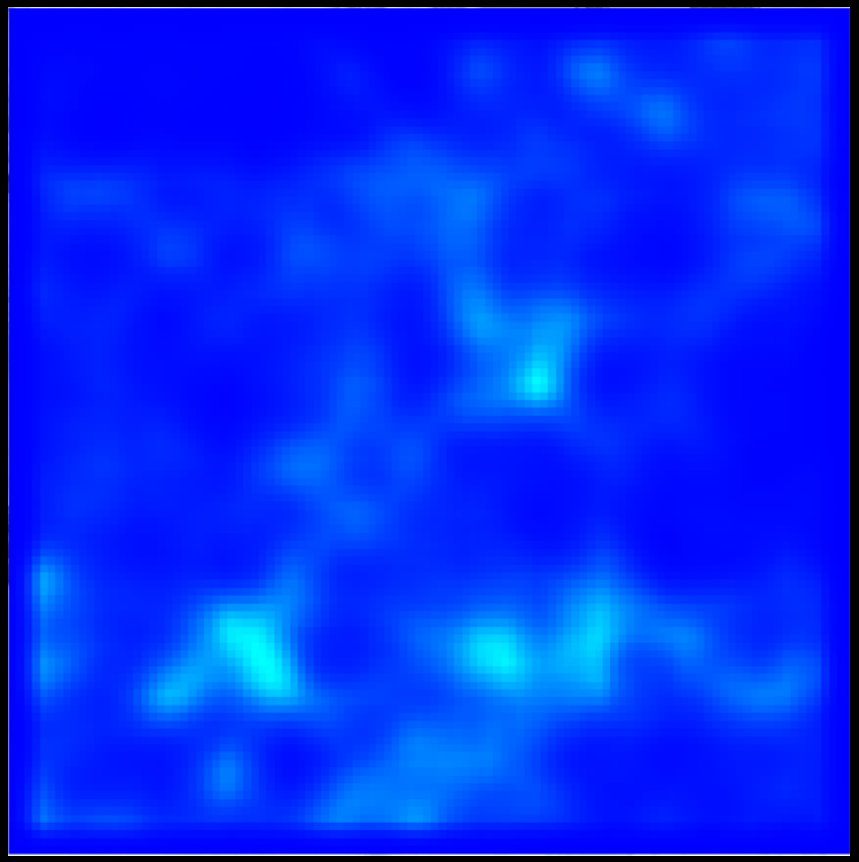
World Area Generation

- Each grid cell of the world map corresponds with a world area.
- The world area seed is generated from the coordinates of the world area.
- Makes heavy use of noise maps.

World Area Generation

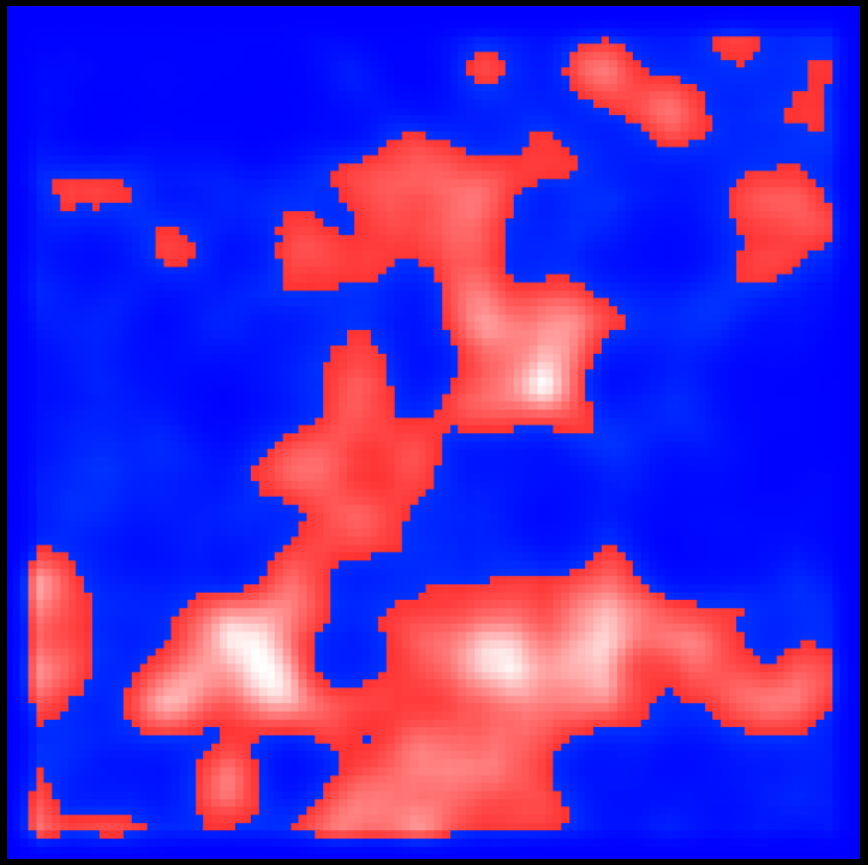
1. Generate coarse grid noise map
2. Determine island cells
3. Create island stamps
4. Create caves
5. Place different block types
6. Place dungeons
7. Place objects

Step 1: Generate Noise Map



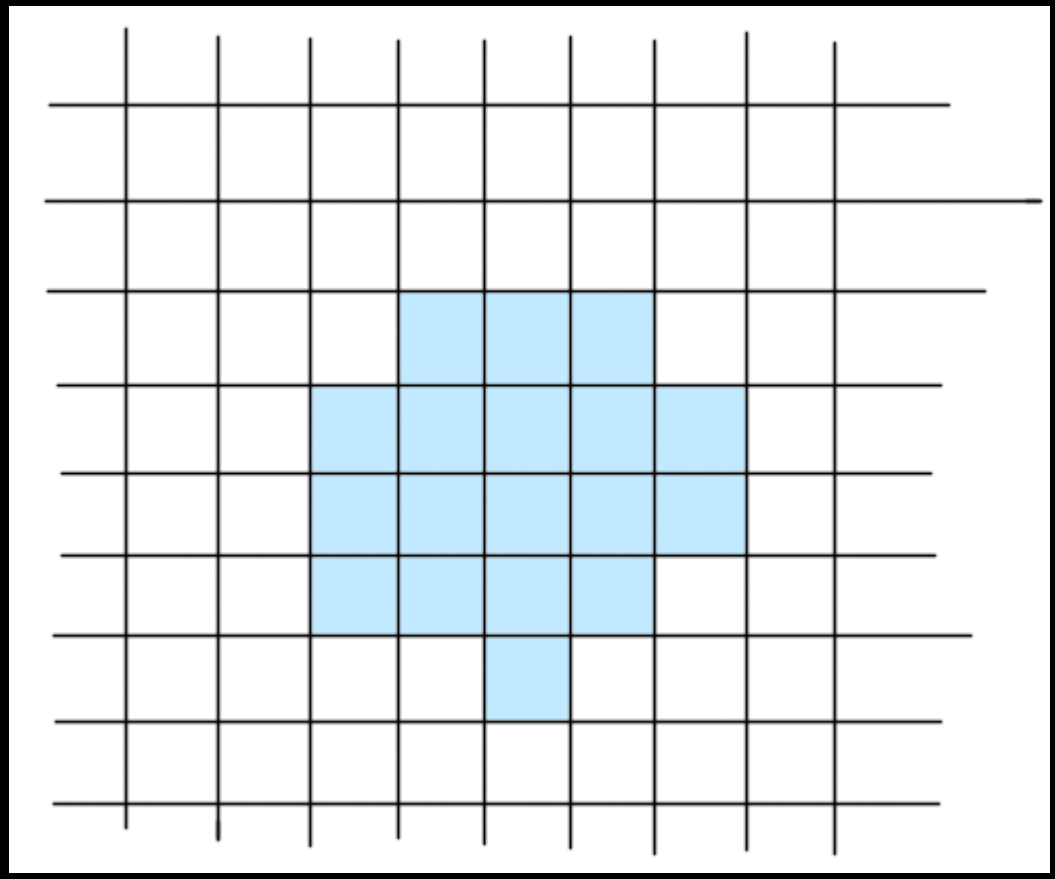
Processed Perlin Noise

Step 2: Determine Islands



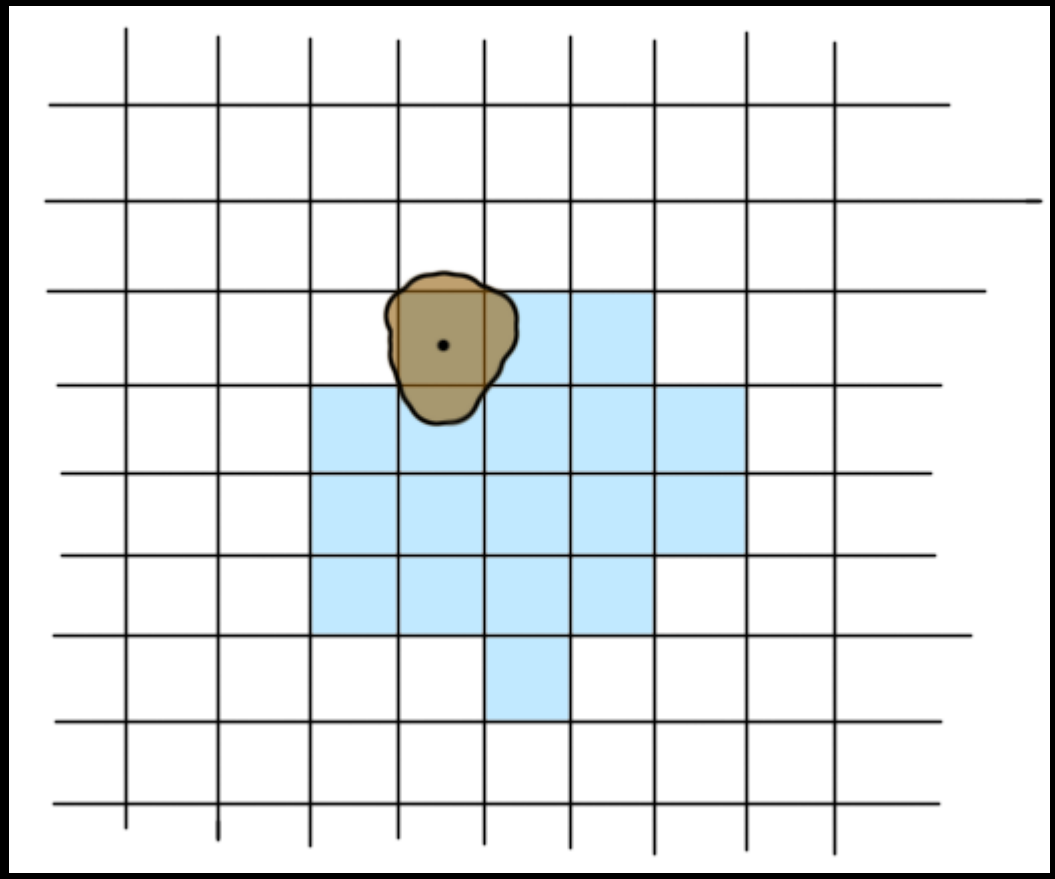
Threshold height + depth first search

Step 3: Place Island Stamps



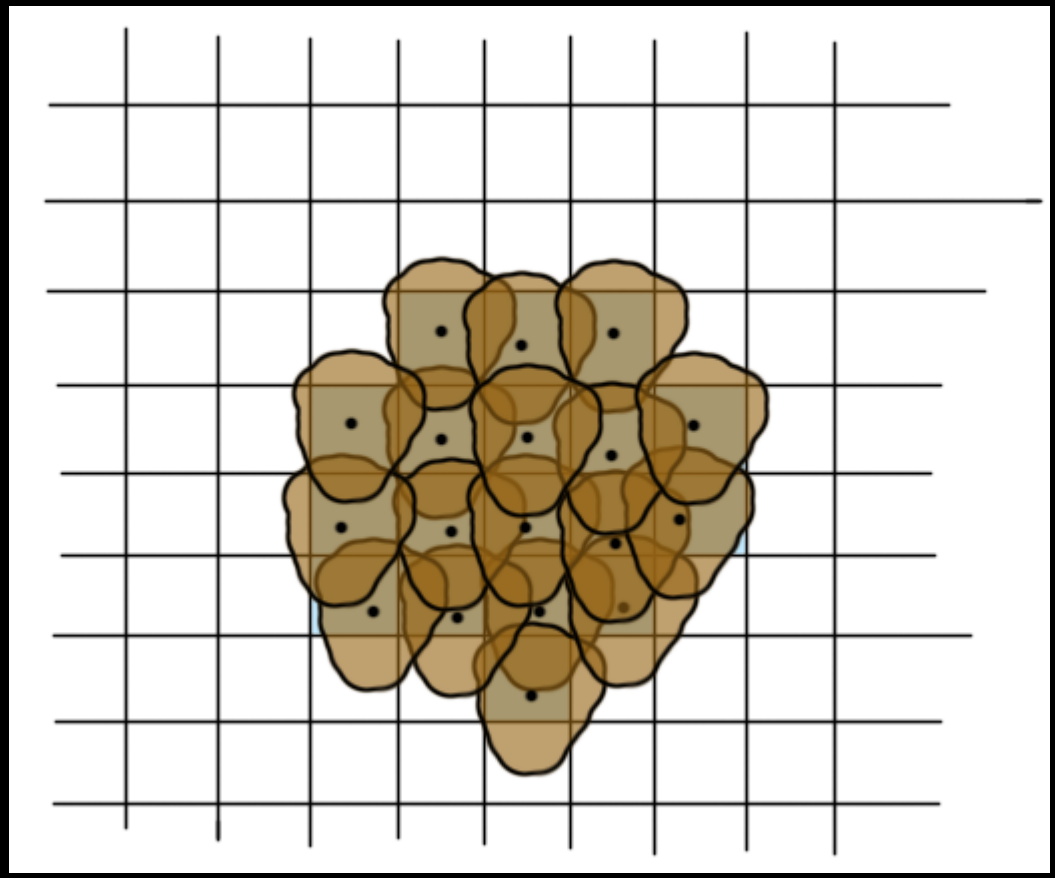
The noise map gives a coarse grid

Step 3: Place Island Stamps



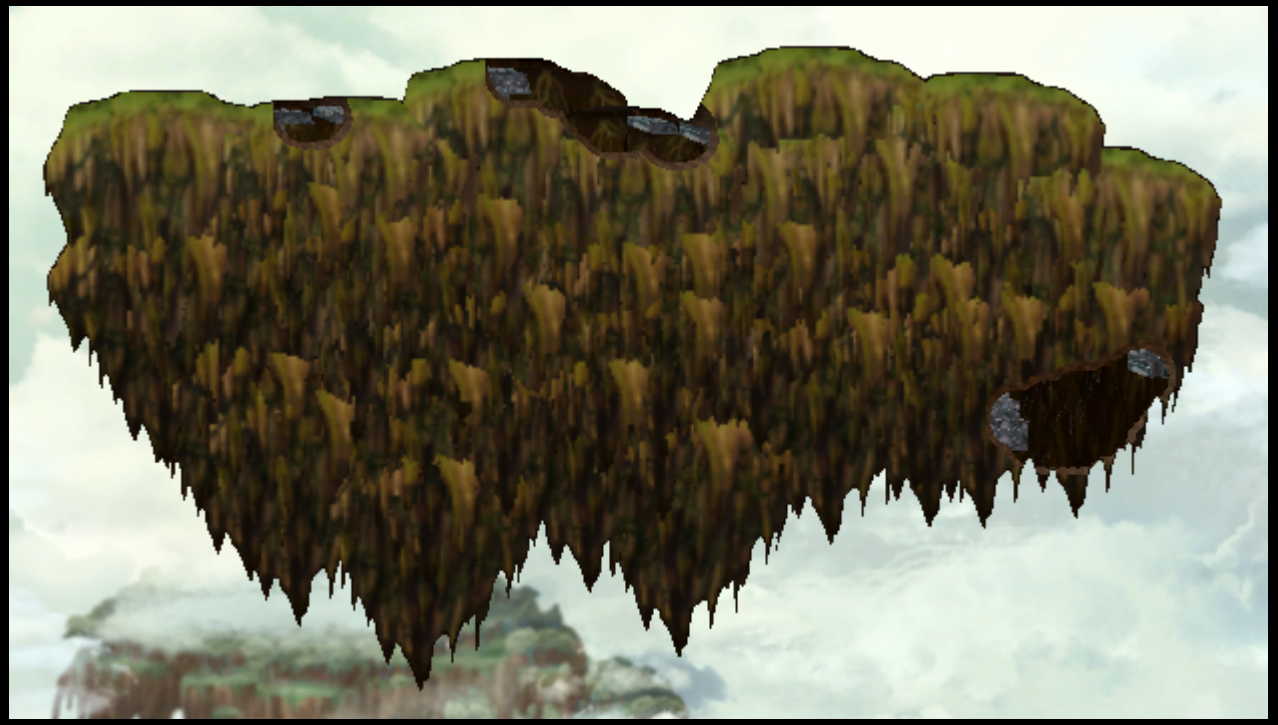
Place island stamps with random offsets in each cell

Step 3: Place Island Stamps



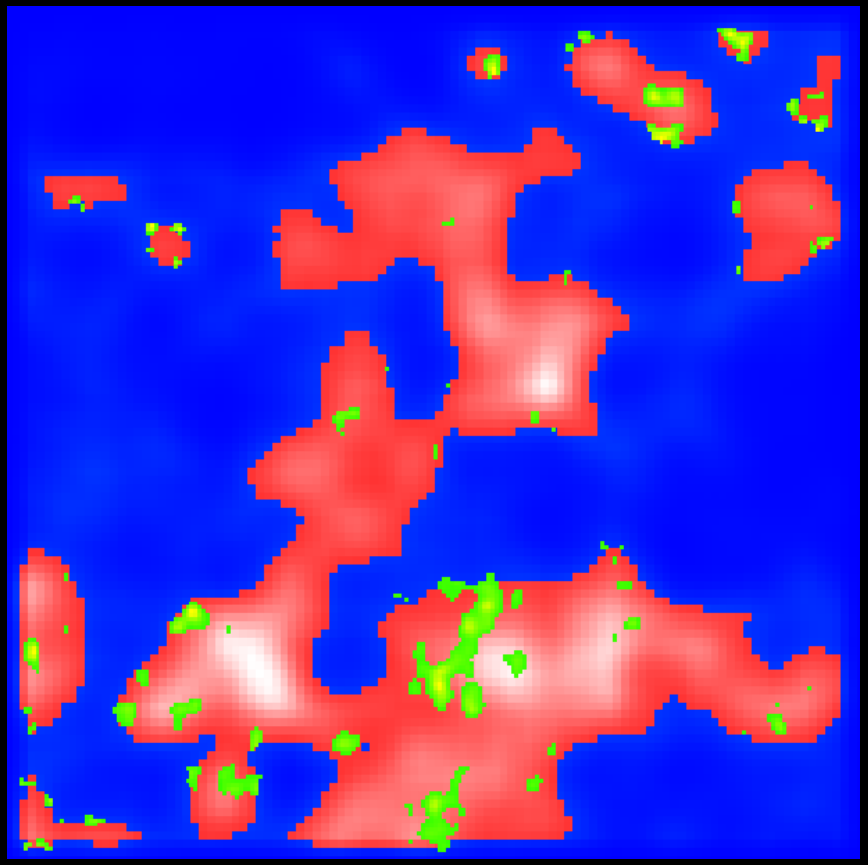
Repeat for all island cells

Step 3: Place Island Stamps



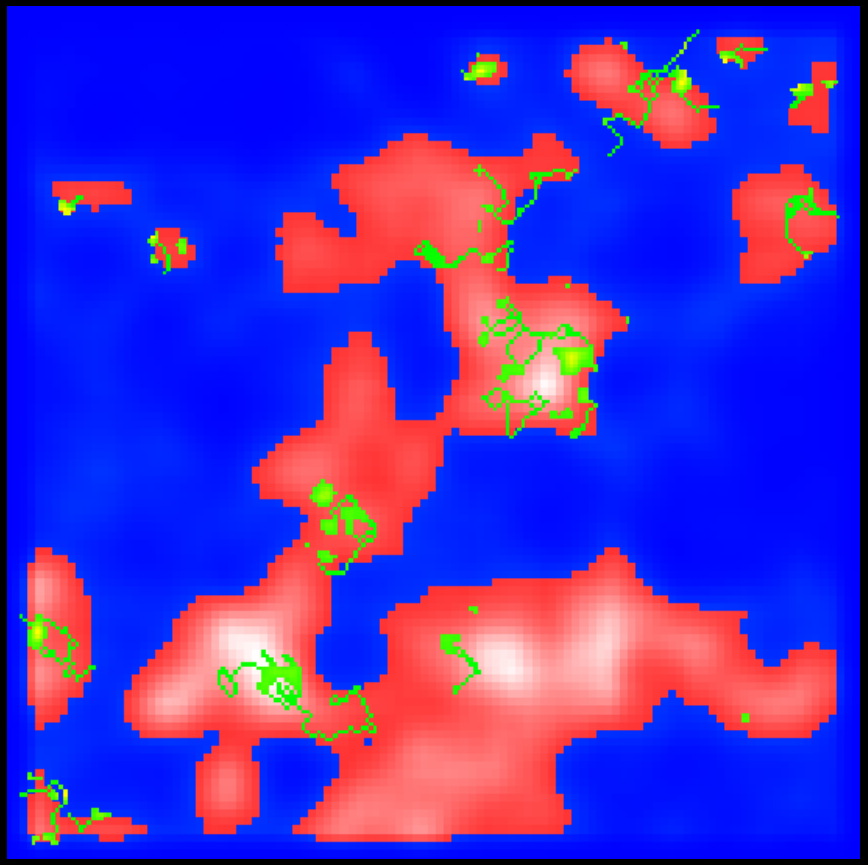
Island stamps provide the island visuals and specify where to place the blocks

Step 4: Add Caves



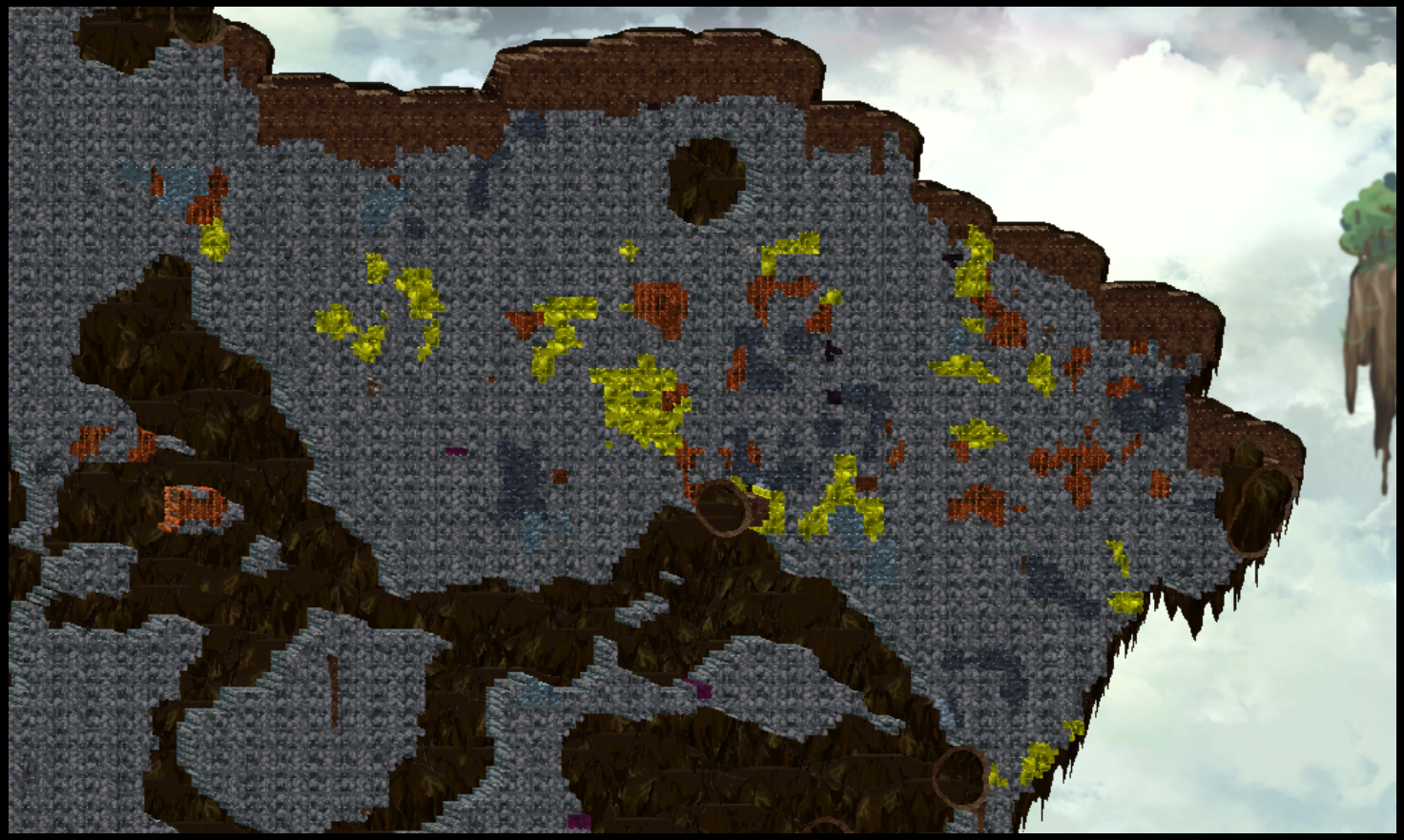
Use noise maps to create cave chambers

Step 4: Add Caves



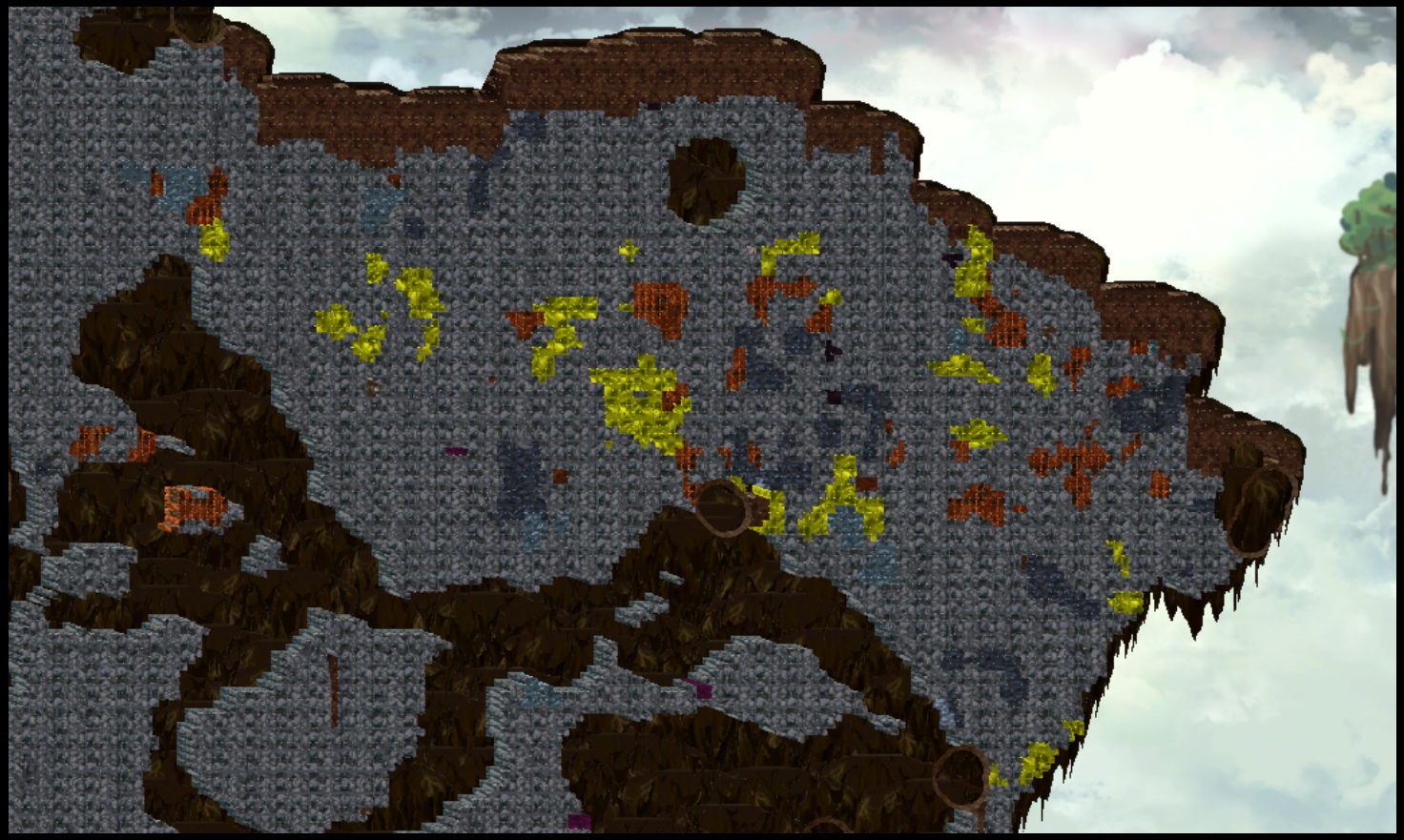
Place cave tunnels using noise map gradients

Step 5: Place Blocks



Place dirt using 1D noise

Step 5: Place Blocks



Place ore cluster lumps using mini noise maps

Step 6: Place Dungeons

This step will be described in more detail later.

Step 7: Place objects

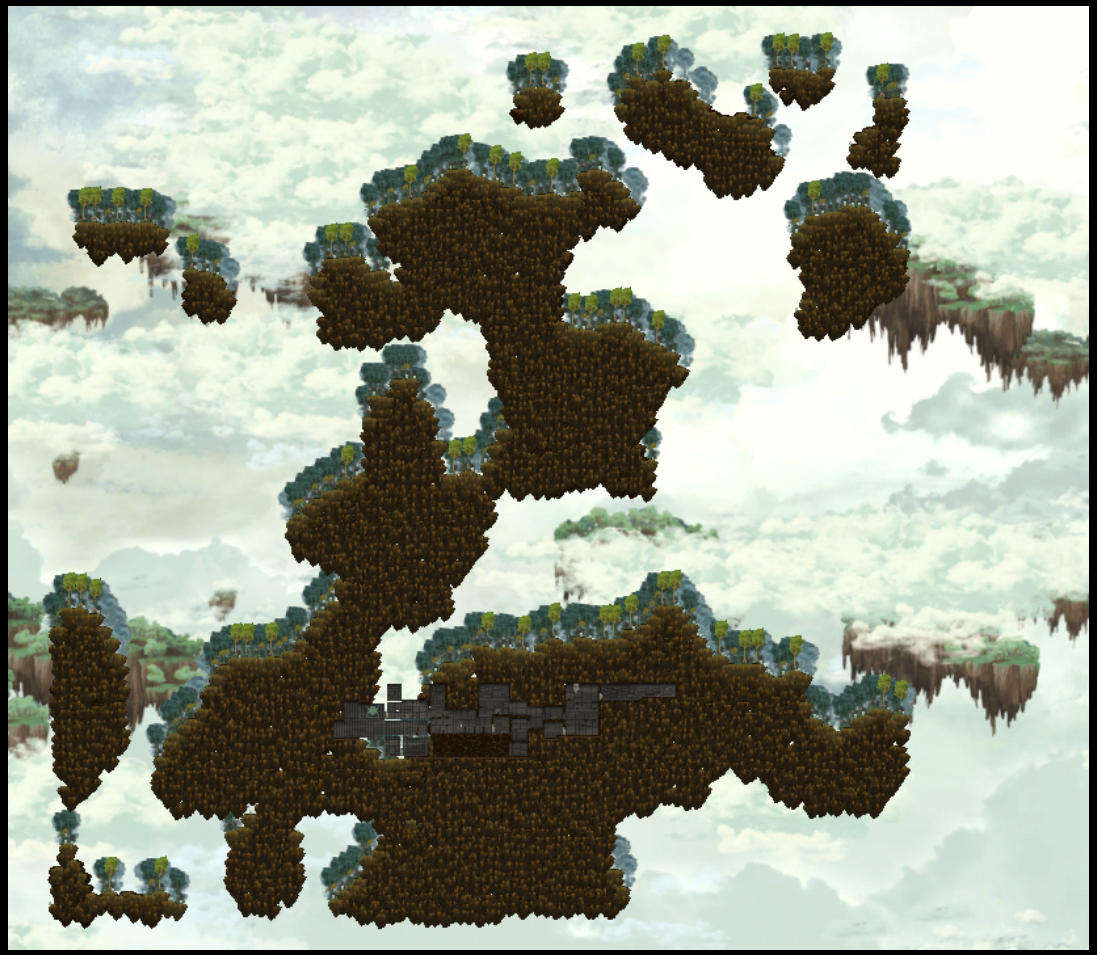
1. Determine all horizontal surfaces

2. Repeat until you place enough objects.
   a. Choose a random object to place
   b. Find a random horizontal surface to place it on.
   c. Subtract from this surface.

Step 7: Place objects

(Example object placement)

Final Result



Dungeon Generation

Dungeon Generation

- Divide and conquer approach
- Lots of steps, but they are fairly simple in isolation
- Boundaries are known before the dungeon is generated.
- Flexible and easy to extend

Dungeon Generation

1. Calculate Bounding Box
2. Split into regions
3. Skim off some perimeter regions
4. Place connections
5. Assign room types
6. Make adjustments
7. Generate

Step 1: Calculate Bounding Box

Choose an island

Step 1: Calculate Bounding Box

Create a random box, within size and aspect ratio constraints.

Step 2: Split Into Regions

- Recursively divide up the space.
- Currently using random axis aligned binary splits.
- Stop splitting regions when they will become too small. (or meet other criteria)

Splitting Regions Example

Recursive Splitting: Depth 1

Splitting Regions Example

Recursive Splitting: Depth 2

Splitting Regions Example

Recursive Splitting: Depth 3

Splitting Regions Example

Recursive Splitting: Depth 4...

Splitting Regions Example

...and so on. Stop splitting when regions are too small, etc.

Splitting Approaches

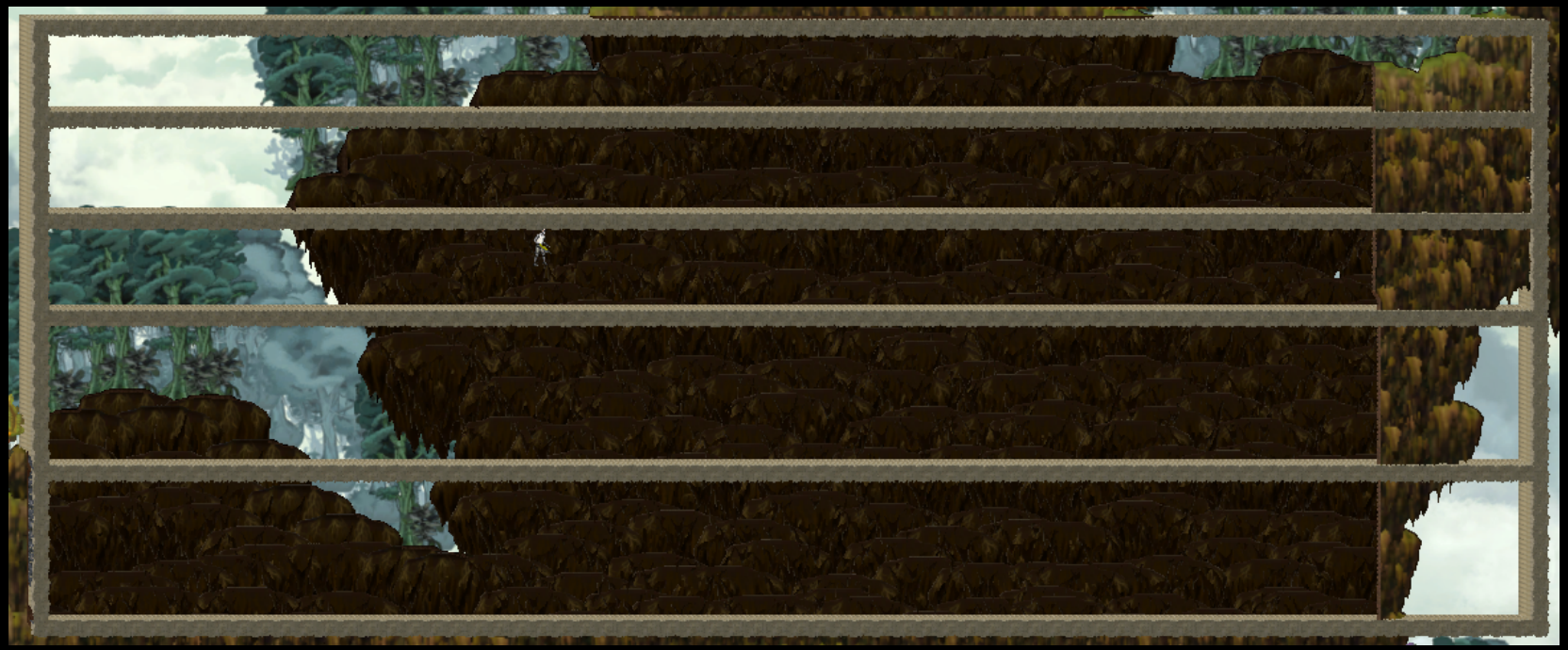
There are different ways to approach splitting:

- Split along a random axis.
- Always split the shorter axis.
- Always split the longer axis.
- Hybrid approaches.

Splitting Approaches

Split along a random axis

Splitting Approaches



Split along the shortest axis

Splitting Approaches

Always split the longest axis

Splitting Approaches

Weighted selection of the last three

Splitting Approaches

Weighted selection and early termination

Step 3: Skim Perimeter Regions

- Don't want every dungeon to be a rectangle.
- Remove a random percentage of perimeter regions.
- Deal with floating regions later.

Step 3: Skim Perimeter Regions

(Perimeter Skimming Example)

Step 4: Place Connections

- Track possible connections with each split
- This helps to make a graph that a computer can easily traverse.
- Currently randomly traversing the graph and inserting travel connections along the way.

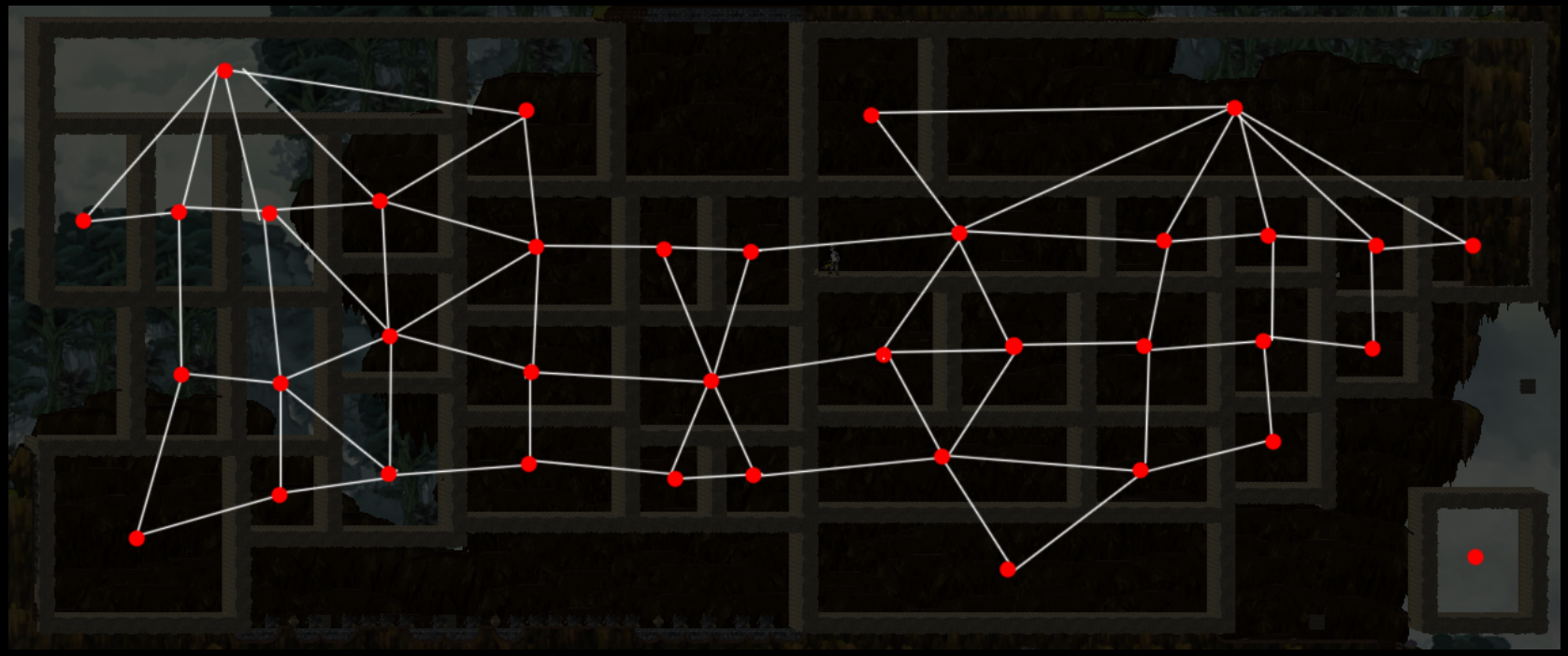
Example Connection Graph

Each region is a node

Example Connection Graph

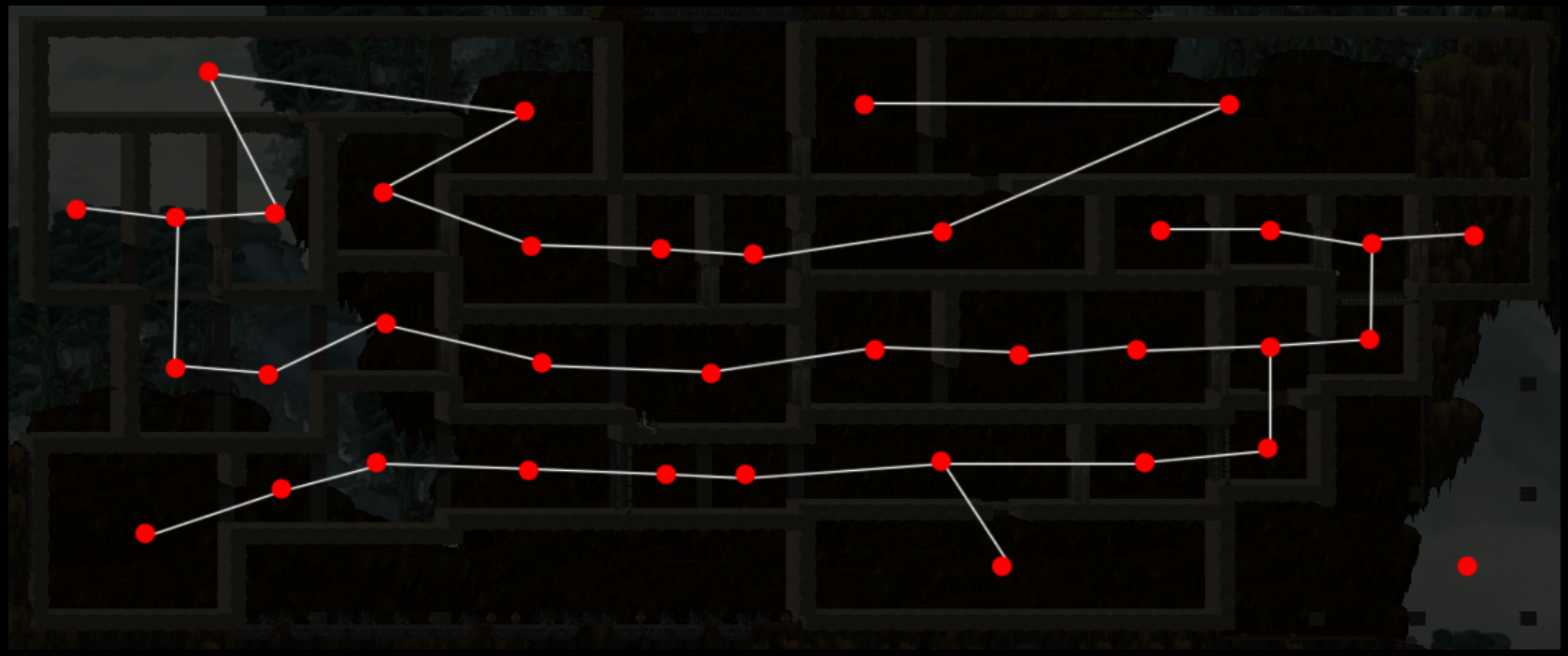
The Connection Information

Example Connection Graph



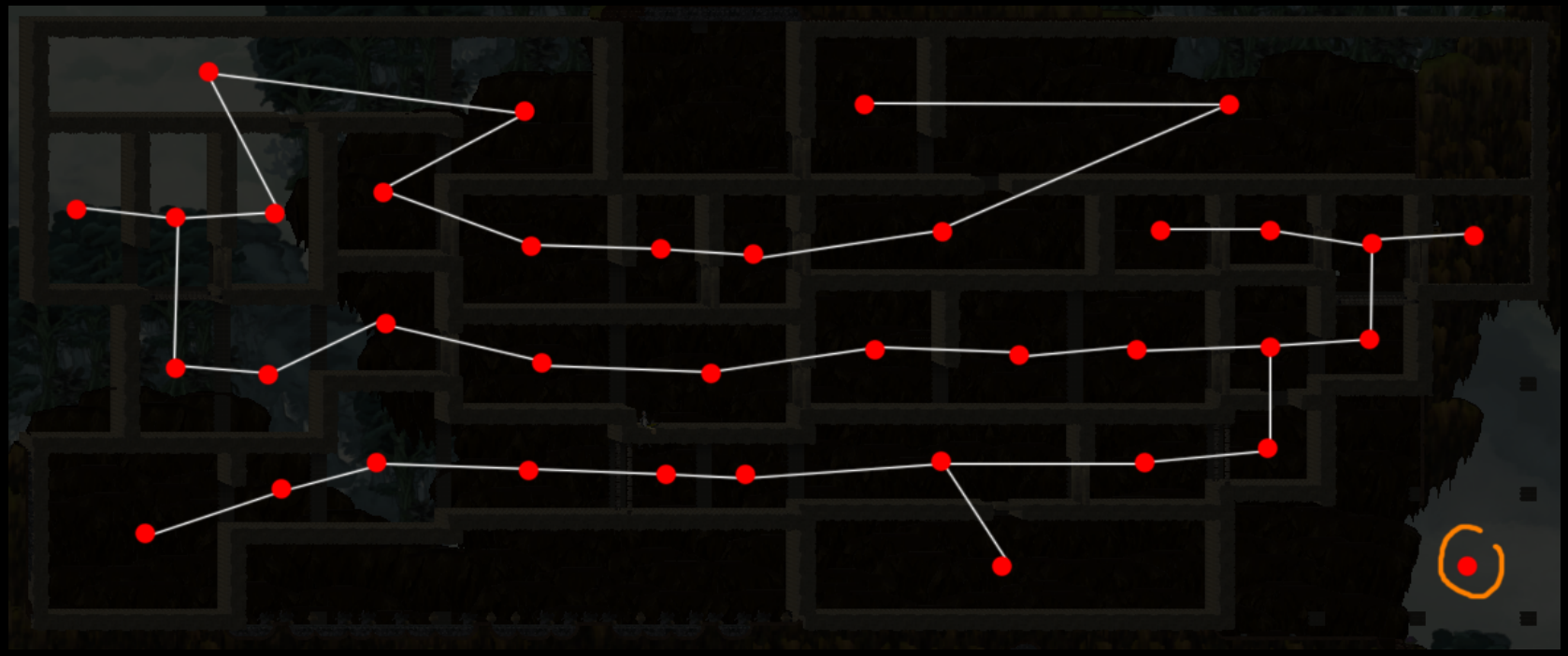
The Connection Graph

Example Connection Graph



Random traversal to place connections

Example Connection Graph



Delete unvisited regions

Example Connection Graph

Place random connection types

Step 5: Assign Region Types

- The region type specifies how the region will be generated.

- Example region types:
  - Treasure room
  - Boss room
  - Entrance room
  - Hub region
  - Trap region
  - etc.

Step 5: Assign Region Types

- We use a variety of methods to assign region types
- We also assign them in their order of importance.

Example Region Assignment Rules

Place the boss treasure room in a terminal region, a far distance away from the entrance.

Example Region Assignment Rules

Place the boss room near the treasure room so that you must travel through the region

Example Region Assignment Rules

Assign trap rooms to terminal regions with ceiling connections.

Example Region Assignment Rules

(More example region assignments)

Step 6: Make Adjustments

- Adjust some of the connections to better match their region types.

- For example:
  - Place empty connections between adjacent boss rooms.
  - Place trap doors with trap regions.
  - Place doors on boss treasure regions.

Step 6: Make Adjustments

- Adjust region types if necessary.
- Make any other corrections that might be necessary.

Step 7: Generate

1. Generate the walls / connections
2. Generate the regions
3. Create the entrance tunnel if necessary
4. Apply random damage to blocks

Region Generation

- We do the following here:
  - Background pattern generation
  - Object placement
  - Platform placement
- Manage the complexity of this by dividing into sub-regions.
  - This is very similar to the region splitting.
  - Also similar to shape grammars

Sub-Region Example

Each box on the wall is a sub-region

Sub-Region Example

Object placement was based on sub-regions

Sub-Region Example

Platform placement as well

More Sub-Region Examples

Random Dungeon Damage

(Before Random Damage)

Random Dungeon Damage

(After Random Damage)

Adding Variety

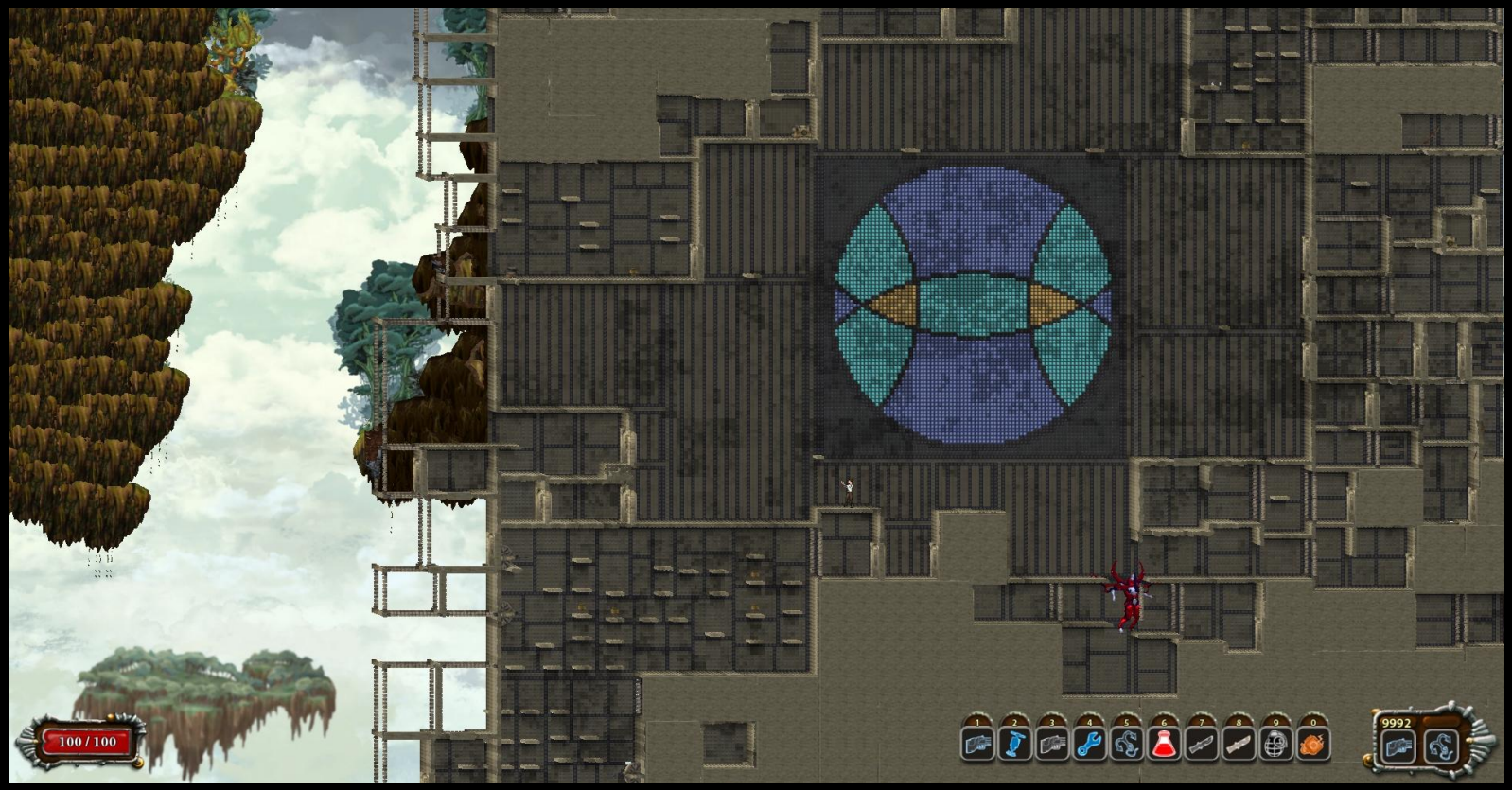
A lot of variety can be achieved by:

- Adjusting parameters
- New types of region removal
- New types of region connection traversal
- Hierarchies of generation parameters
- and more!

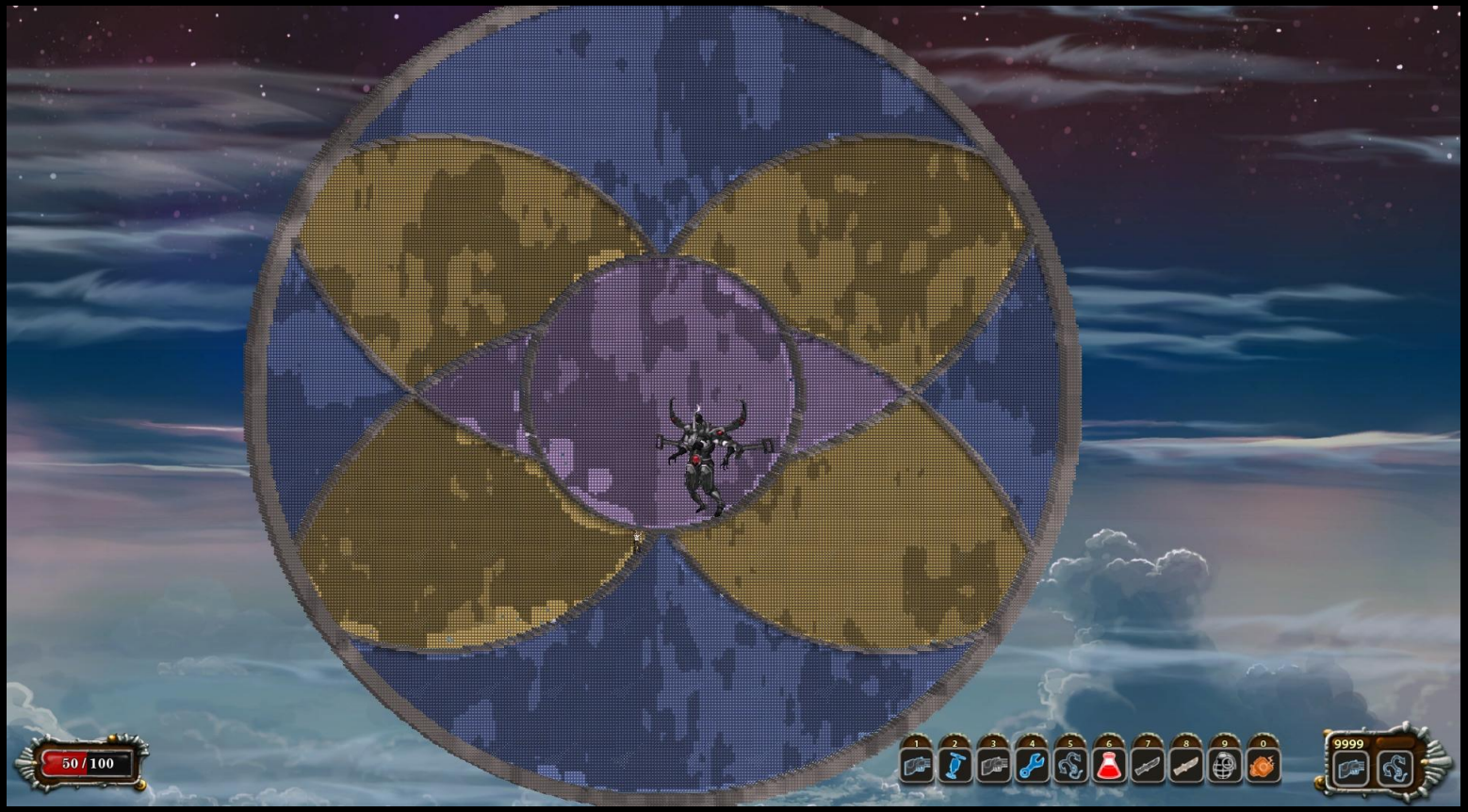
Architecture Examples

Architecture Examples

Architecture Examples



Architecture Examples



Architecture Examples

Architecture Examples

Architecture Examples

Enemy Placement

Enemy Placement: Requirements

The placement algorithm needs to:

- Support procedural environments, that can be changed by the player.
- Support a wide range of enemy sizes.
- As fast as possible

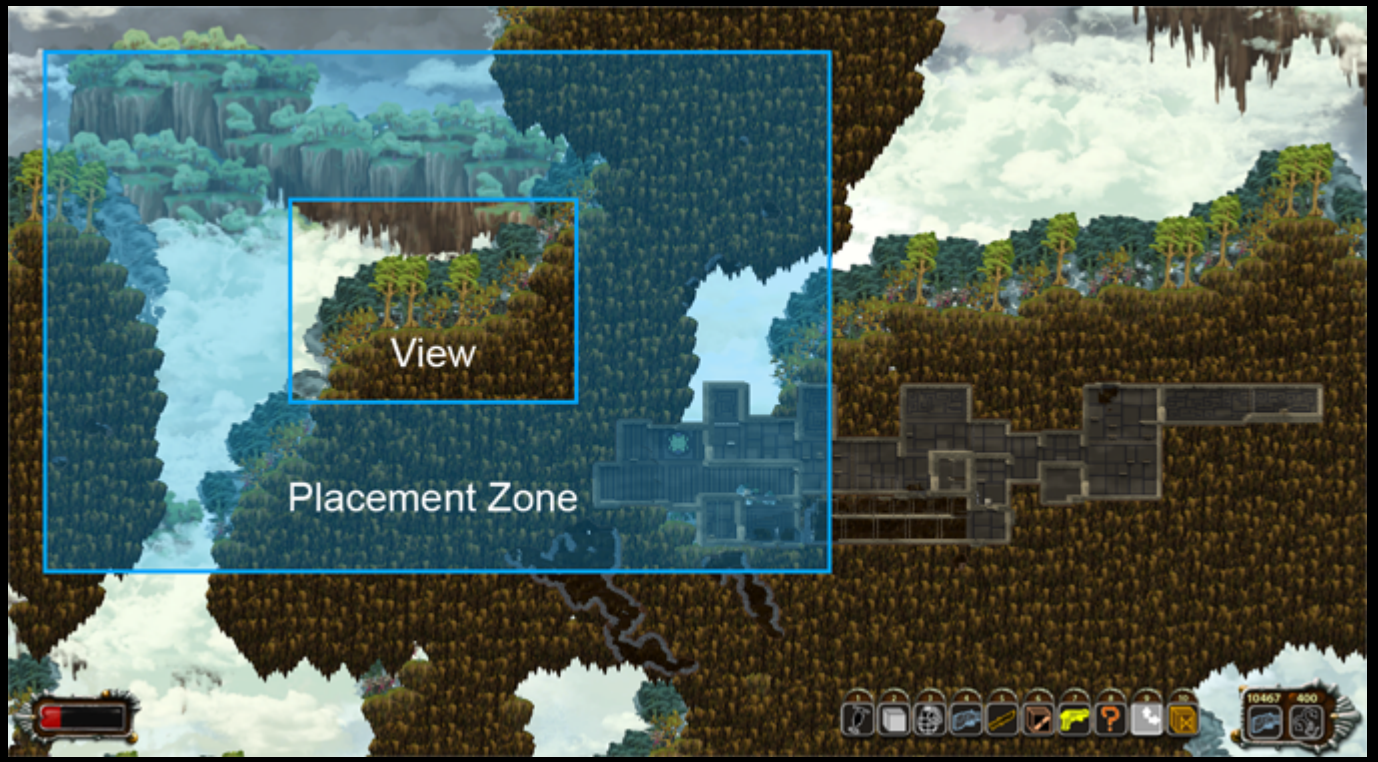
Enemy Placement: Solution

- It was the easiest to control the experience by dynamically spawning most of the enemies.
- The current system has a dynamically changing goal danger level
- It will add enemies until the goal danger level is reached

Enemy Placement: Solution

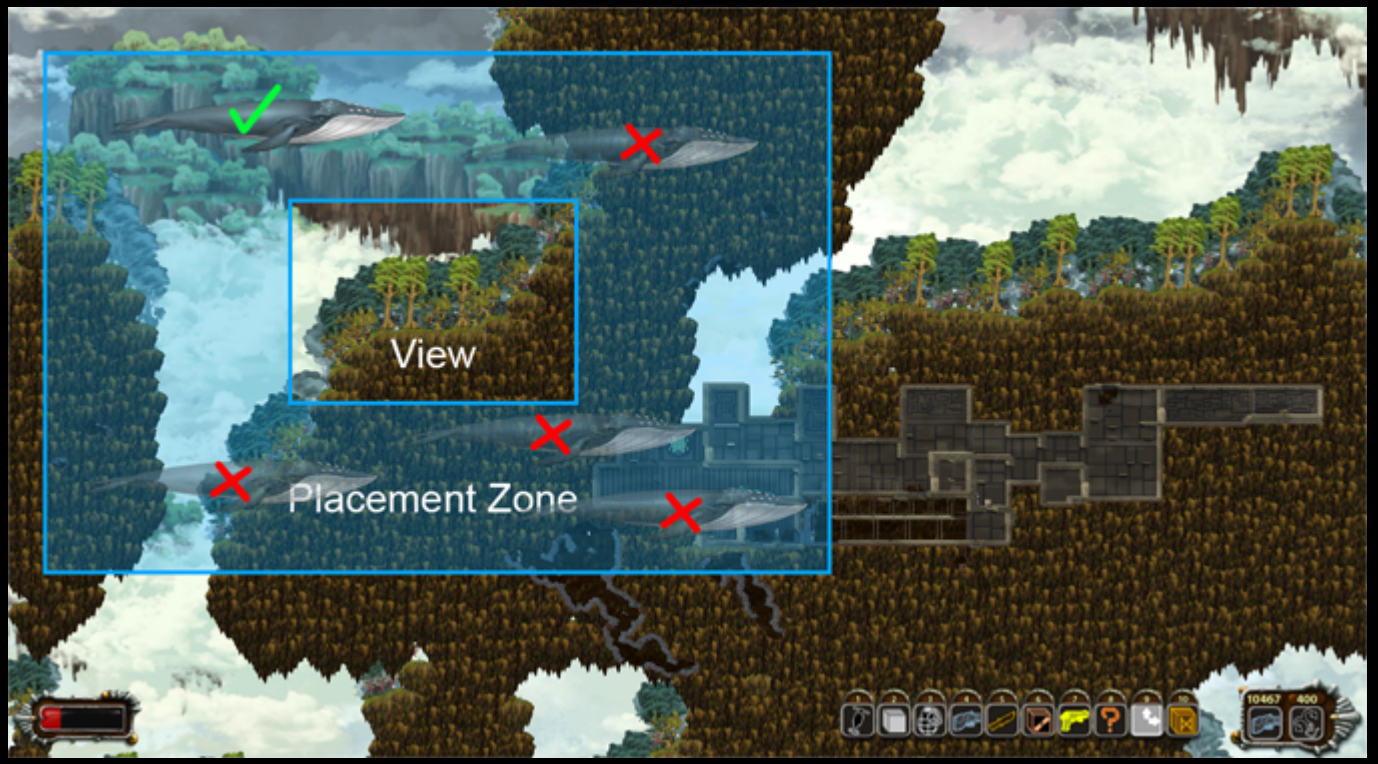
- The types of enemies placed depends on the situation the player is in. (in ruins, flying, etc.)
- Enemies are also dynamically removed if they aren't needed.

Enemy Placement: Solution



Randomly place in a zone surrounding the player's view

Enemy Placement: Solution



Try once a frame until the enemy is placed.

Enemy Placement: Solution

Similar for ground enemies:

- Choose a random point in the air
- Ray cast down to the ground
- If the ray hits the ground, check if the collision area is suitable

Lessons Learned

Automated Testing = Important

- It’s quite possible that players will see bugs that you never will.
- Try to implement this as soon as it makes sense.
- Make sure you log your generation seed and other parameters.
- Some errors are hard to detect.

Random Notes

- Author control vs emergent results
- Can be hard to guarantee that a level will be perfect.
  - Gameplay can be designed to make the experience more robust.
- Procedural / designer hybrids can work really well.

Things to Remember

- Noise maps are really handy for organic things
- Recursive subdivision is nice for architecture
- Try to keep things simple and organized.
  - The complexity grows really fast!

Contact Info

Evan Hahn

Email: evan.hahn@snowedin.ca

Twitter: @ehahnda

Web: snowedin.ca


WINDFORGE


Questions ?
?
?
768