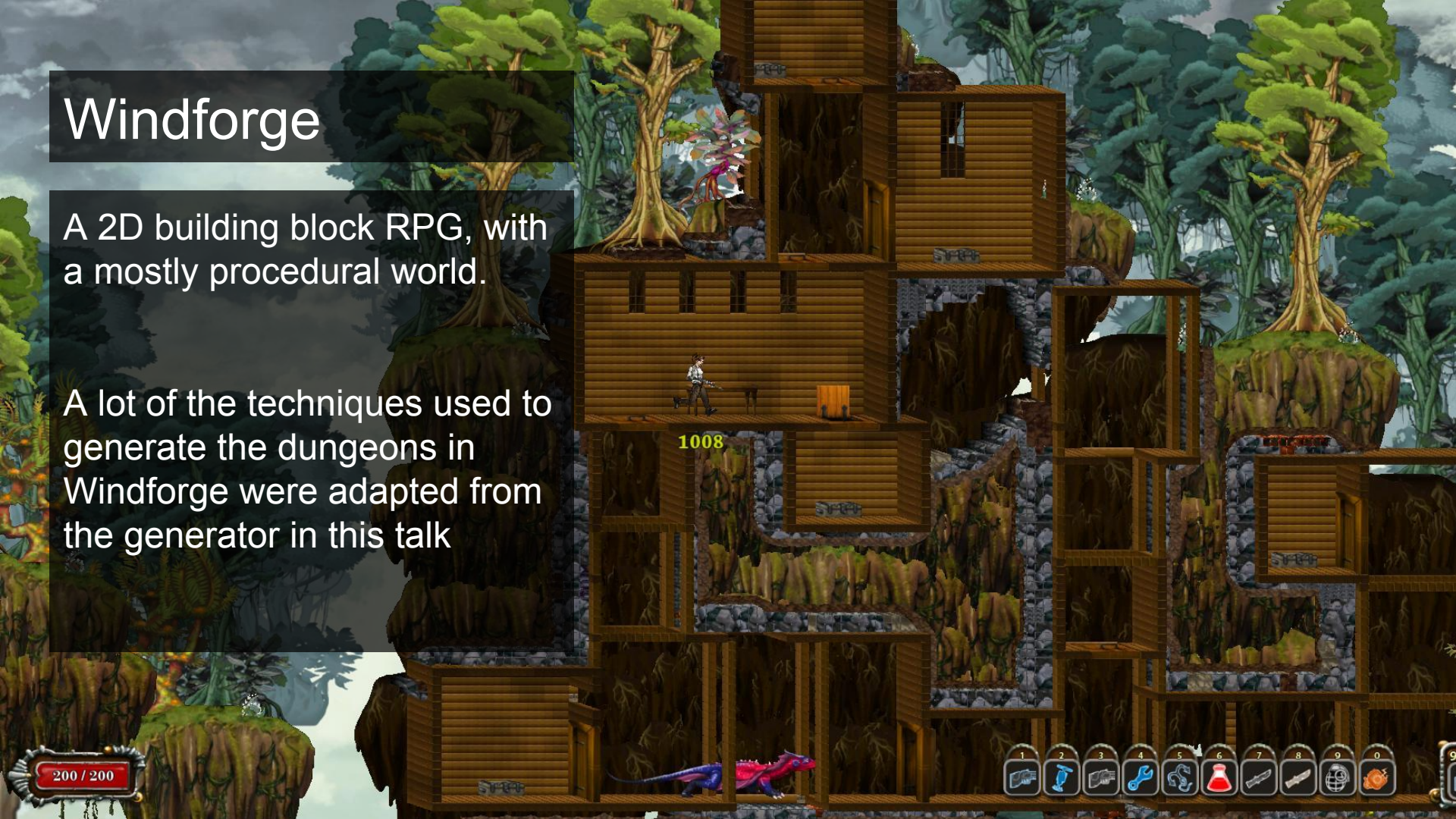
Grammar Techniques for Procedural Architecture

Evan Hahn

Windforge

A 2D building block RPG, with a mostly procedural world.

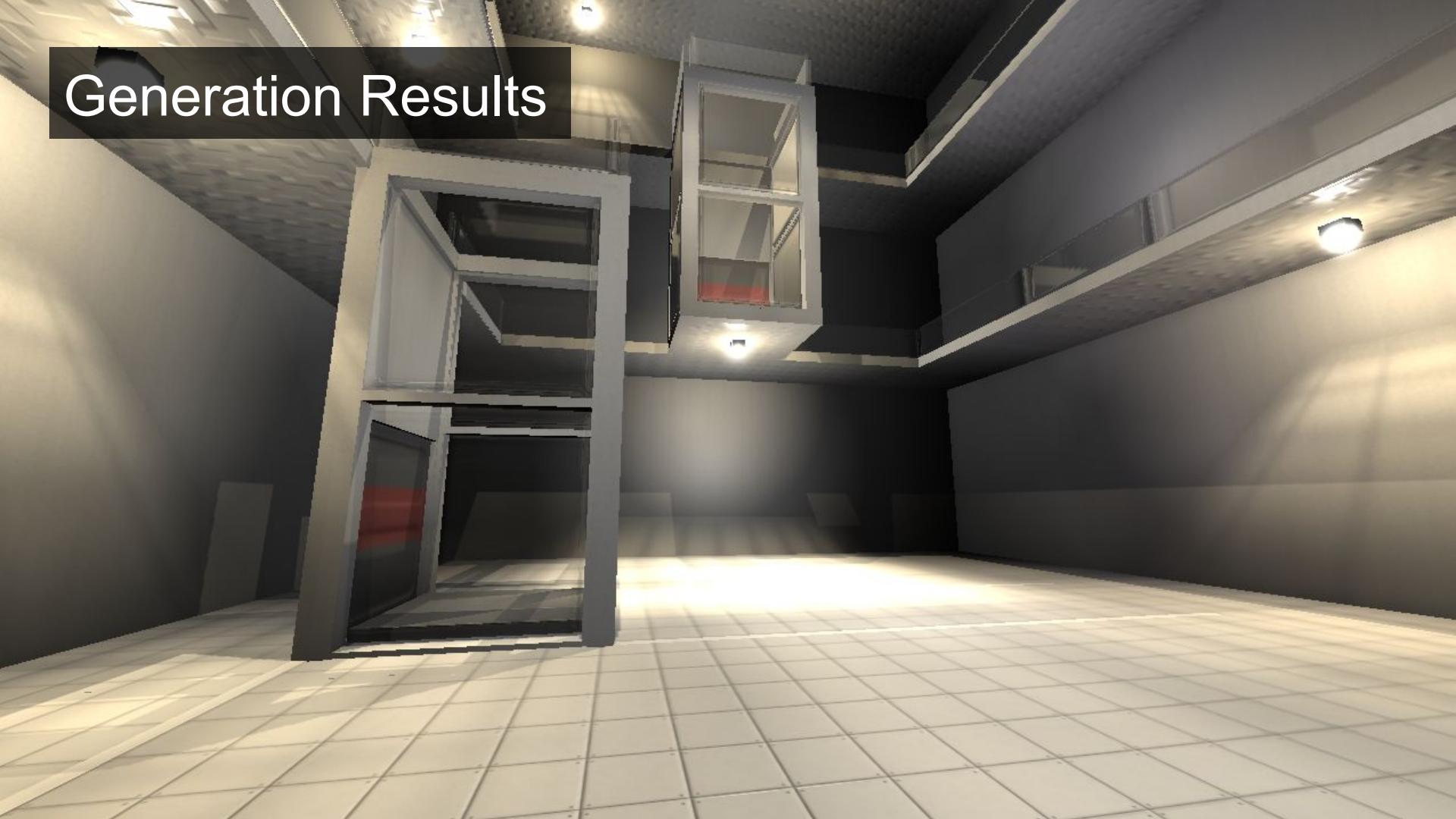
A lot of the techniques used to generate the dungeons in Windforge were adapted from the generator in this talk



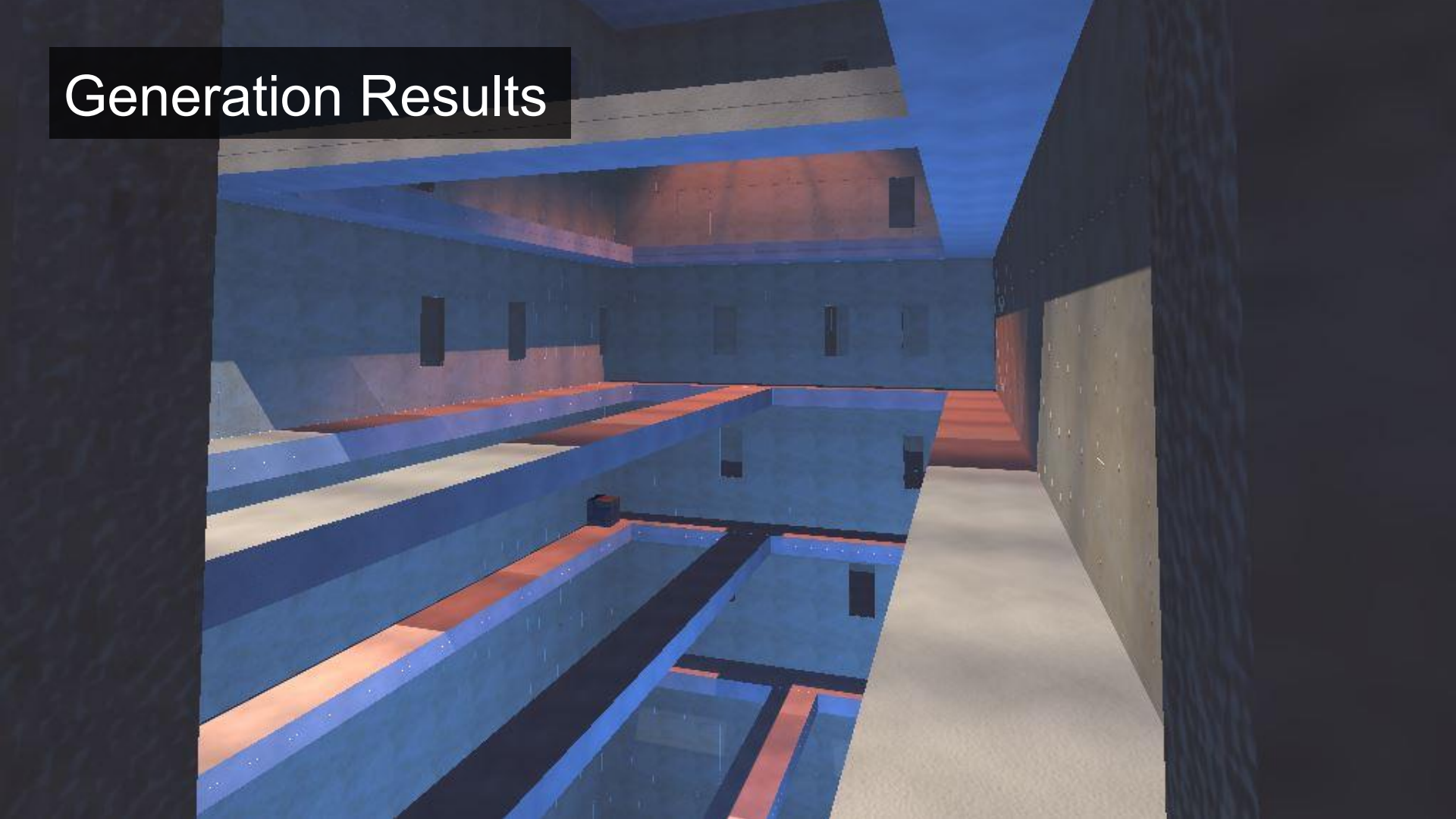
Generator Features

- Generates full 3D architecture and levels
- Only a small set of simple meshes are needed
- Provides good control over the generated results
- Wide range of results are possible
- Simple object placement
- Large levels can be generated quickly

Generation Results

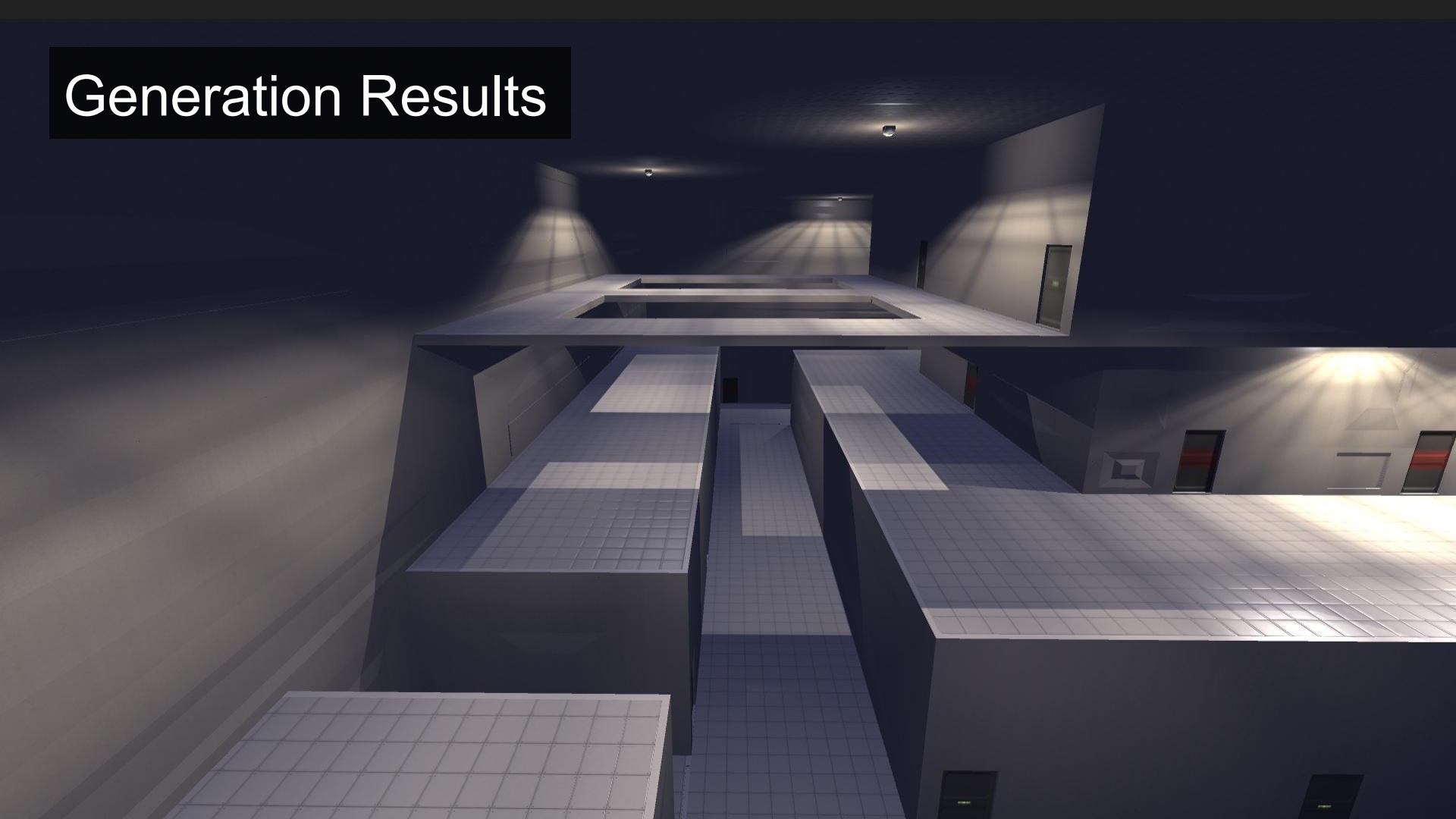


Generation Results

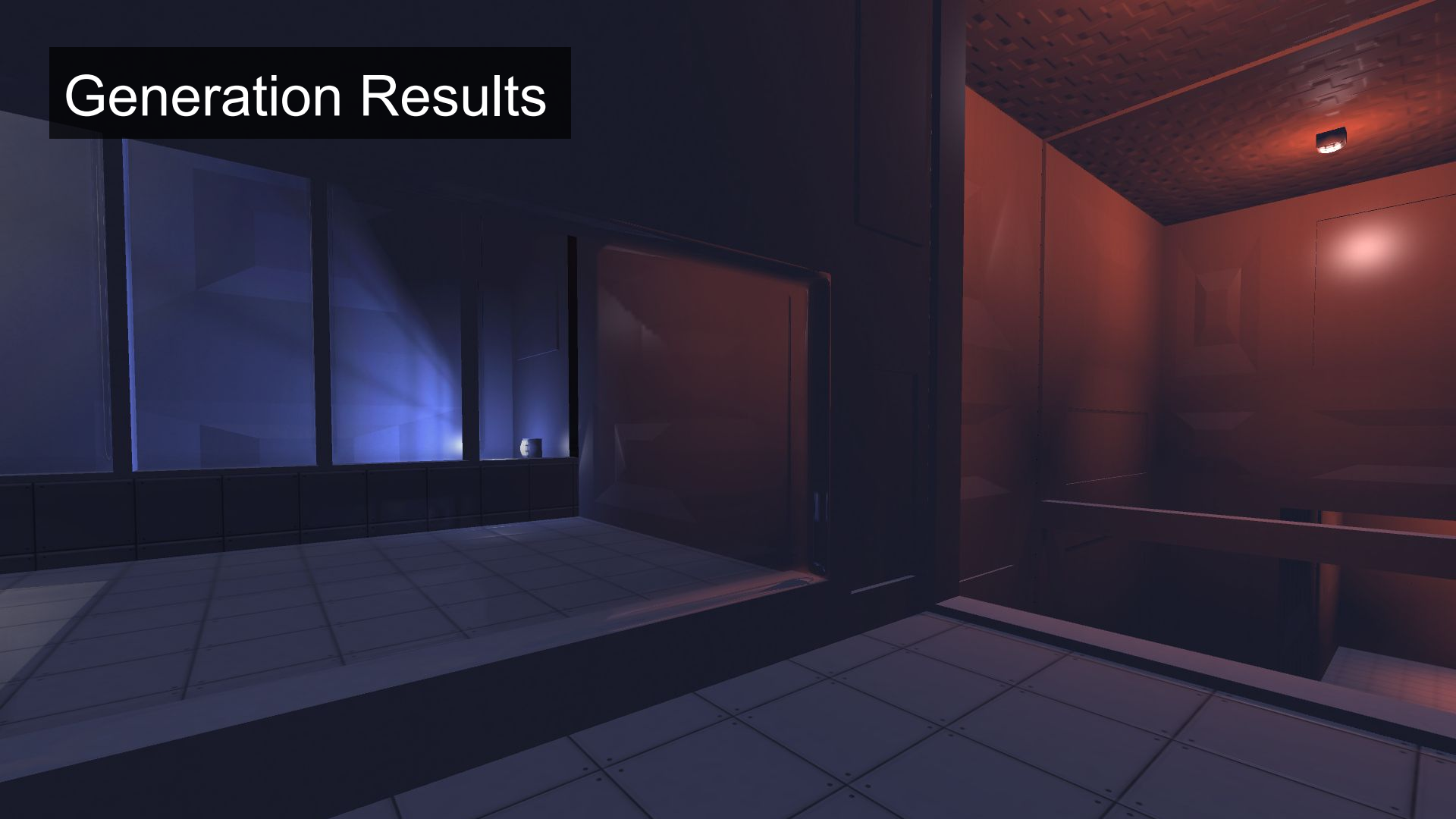


Generation Results

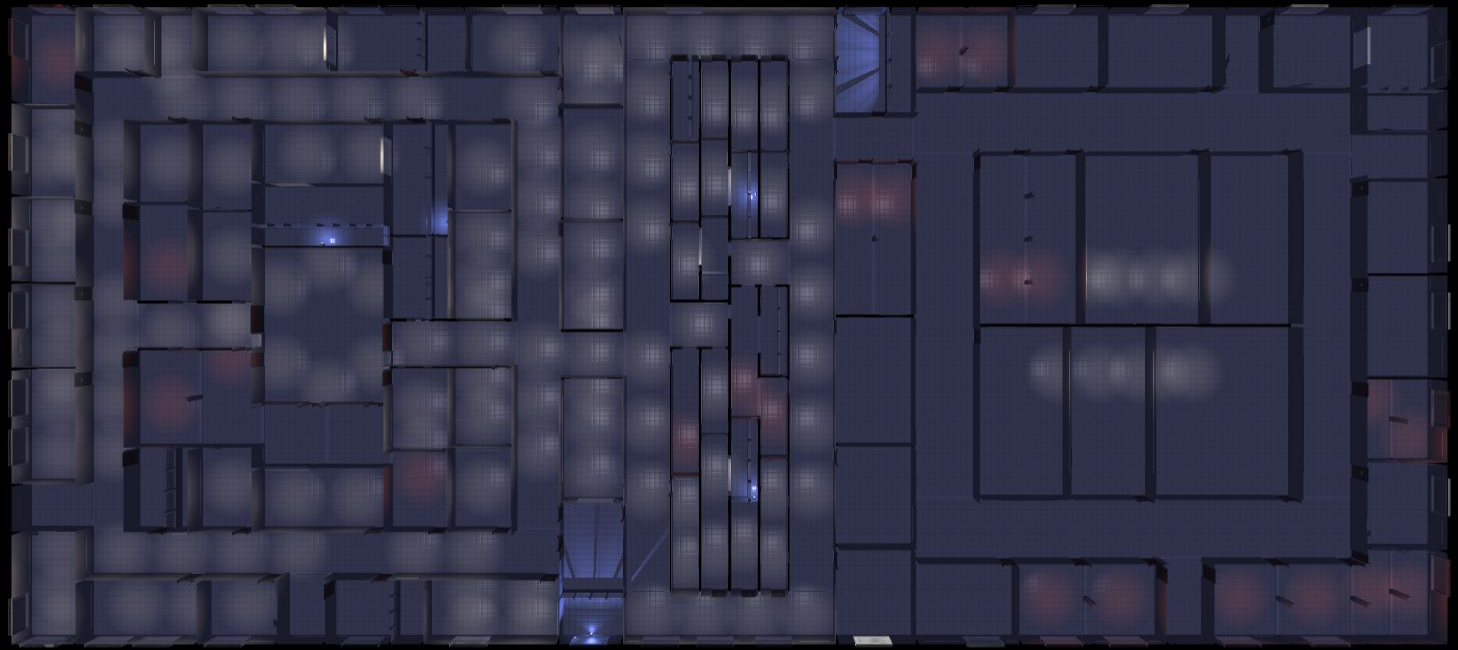
Generation Results



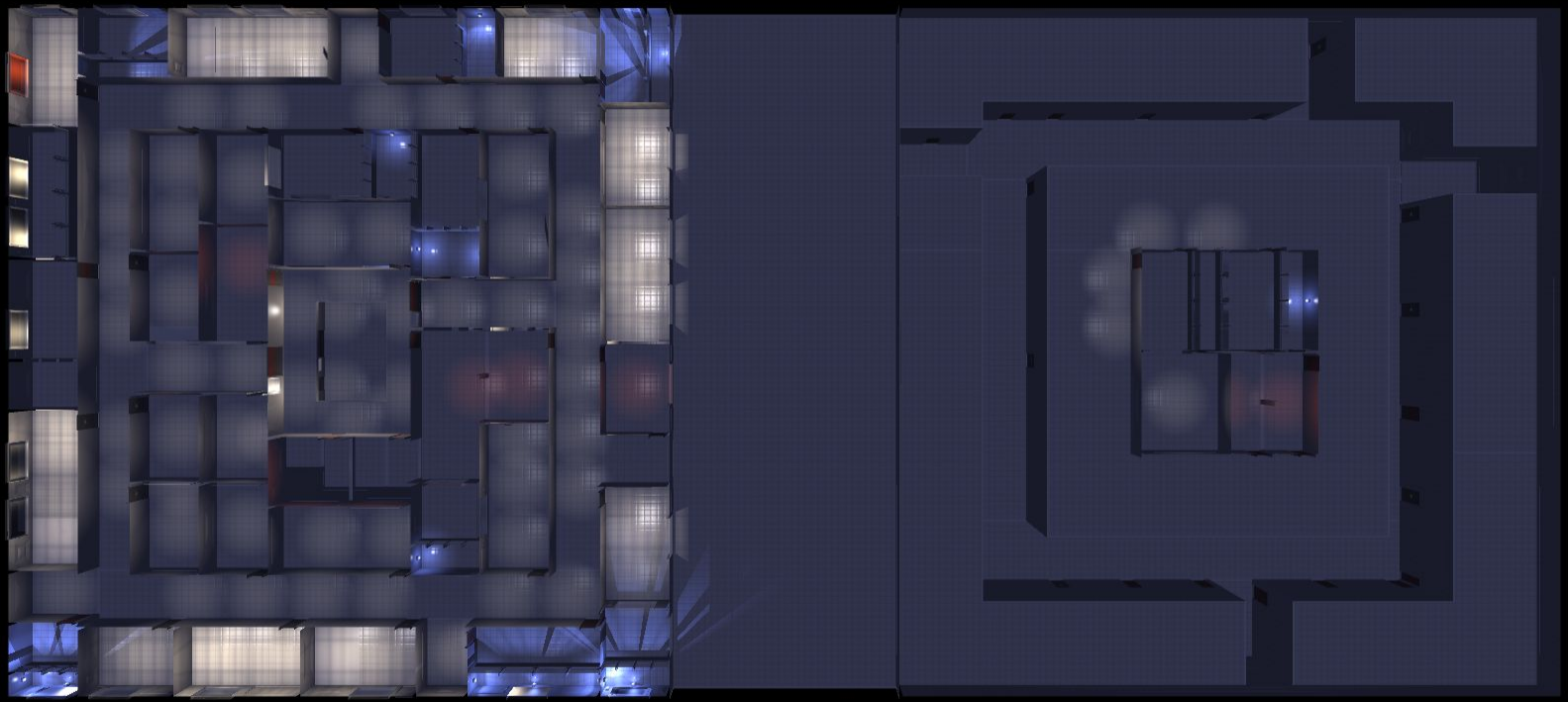
Generation Results



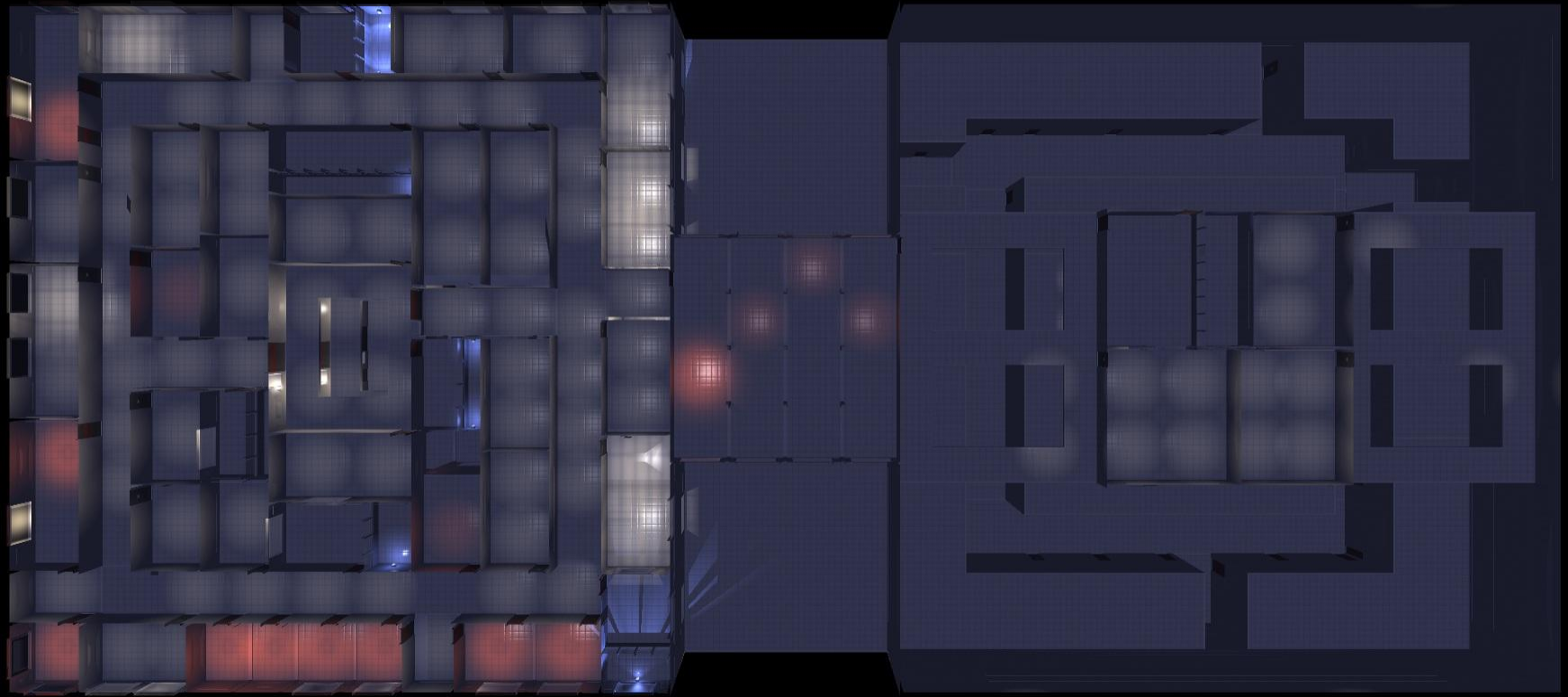
Generation Results



Generation Results

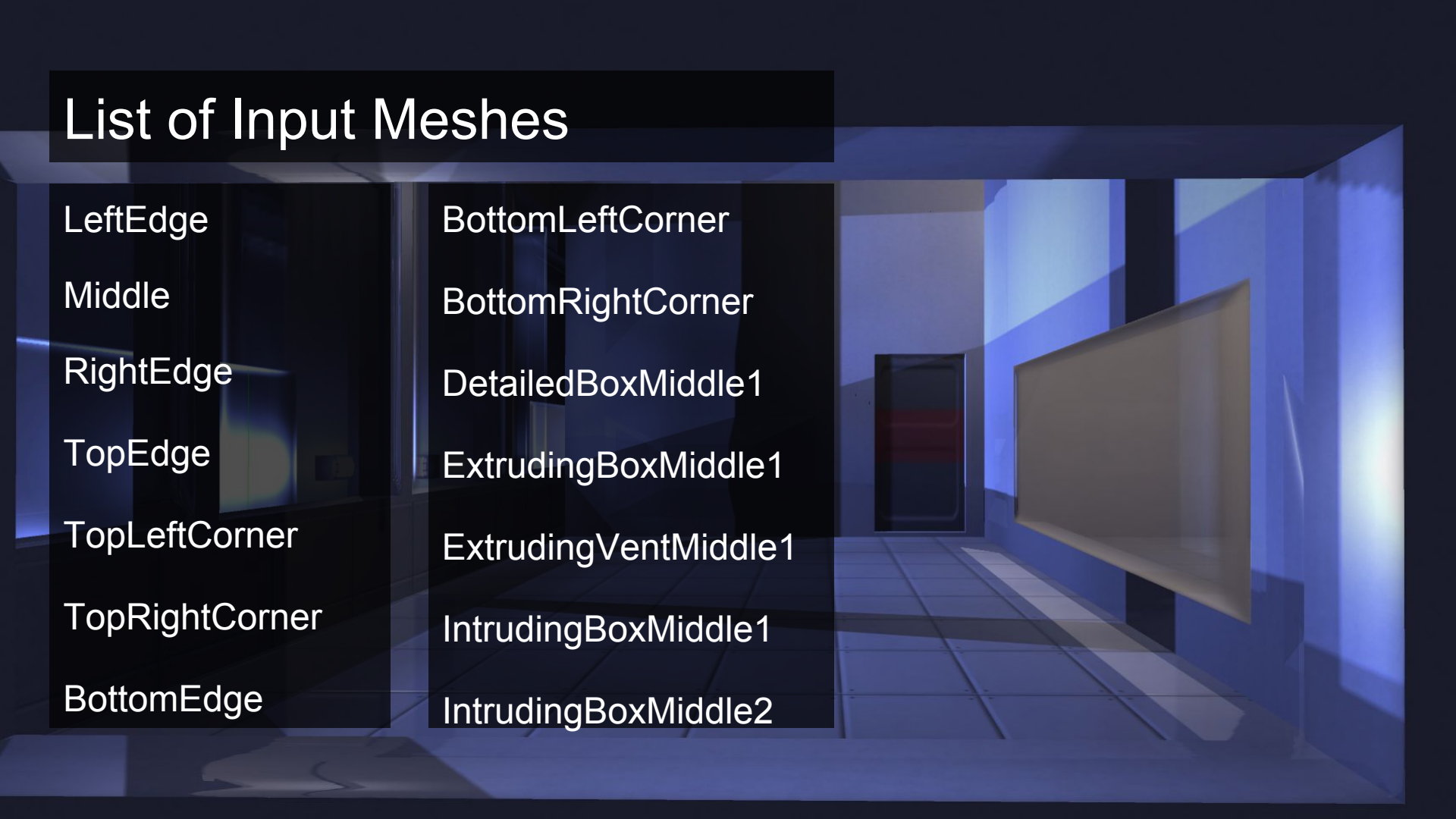


Generation Results



List of Input Meshes

- LeftEdge
- Middle
- RightEdge
- TopEdge
- TopLeftCorner
- TopRightCorner
- BottomEdge
- BottomLeftCorner
- BottomRightCorner
- DetailedBoxMiddle1
- ExtrudingBoxMiddle1
- ExtrudingVentMiddle1
- IntrudingBoxMiddle1
- IntrudingBoxMiddle2



Why Use Grammars?

- They represent self-similarity well
- Architecture can often be decomposed in to abstract parts
- A lot of variety can be achieved by substituting different parts
- Easy to control random outcomes

Using Spacial Grammars

Spacial grammars work in a similar way to other grammars

[English Sentence] = [Simple Sentence] | [Compound Sentence]

[Simple Sentence] = [Declarative Sentence] | [Interrogative Sentence] | [Imperative Sentence] | [Conditional Sentence]

[Declarative Sentence] = [subject] [predicate]

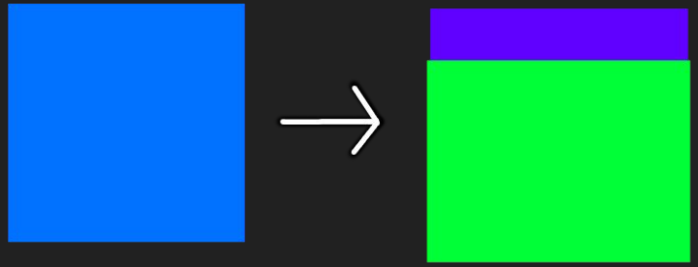
[subject] = [simple subject] | [compound subject]

[simple subject] = [noun phrase] | [nominative personal pronoun]

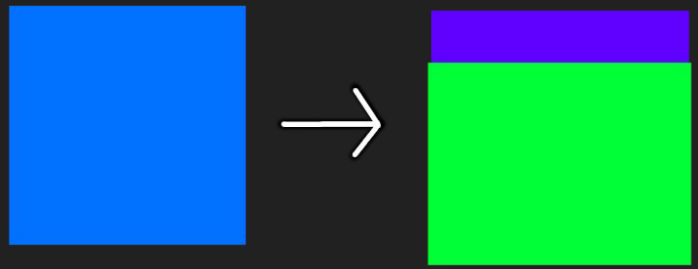
etc.

Using Spacial Grammars

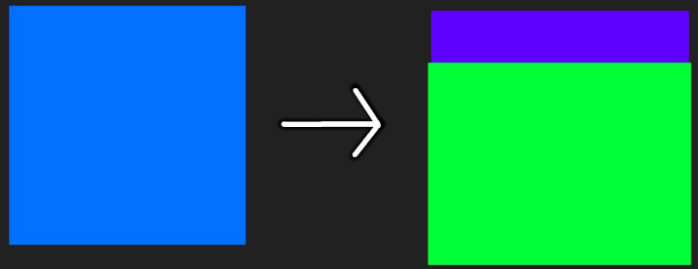
They use symbol substitution as well, but the symbols are associated with geometry.

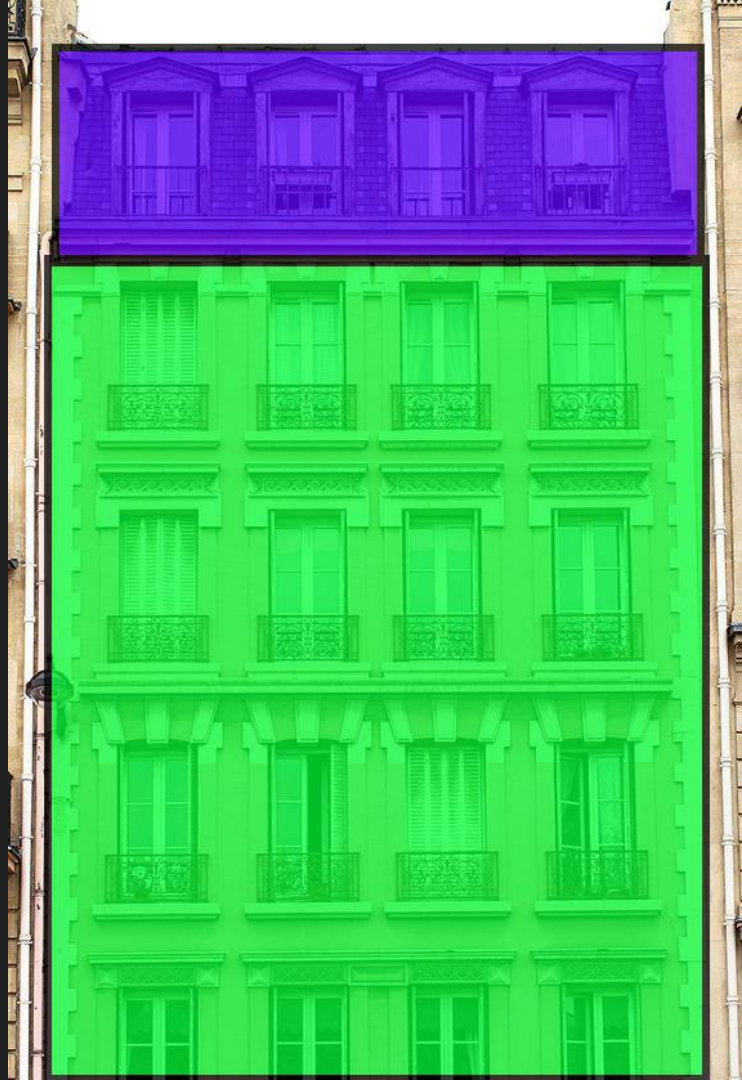


Real Life Example

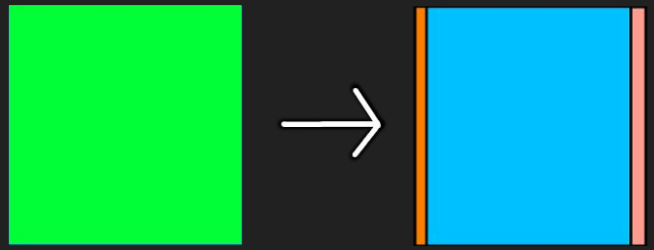


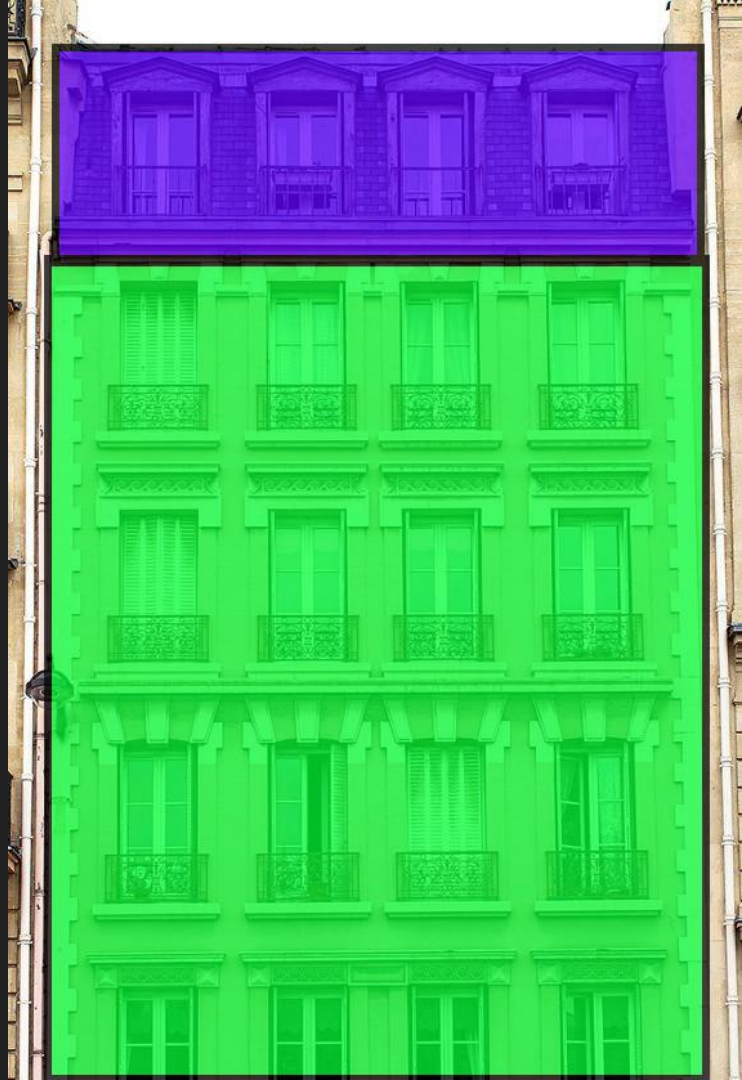
Real Life Example



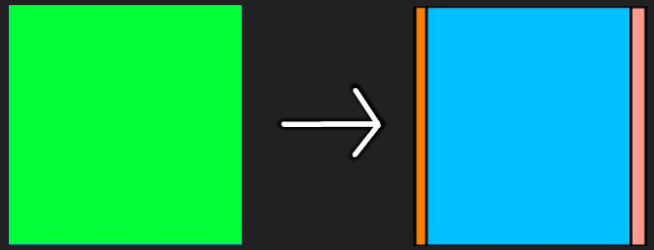


Real Life Example

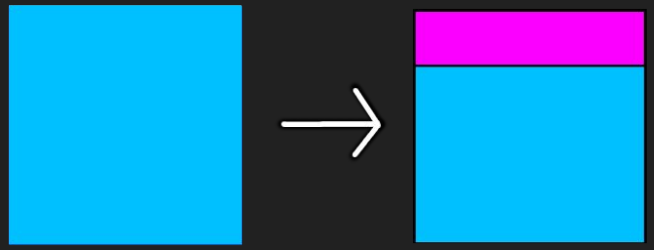




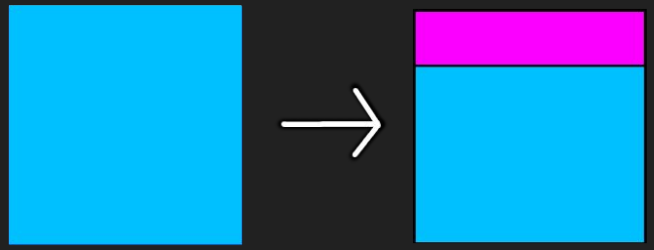
Real Life Example



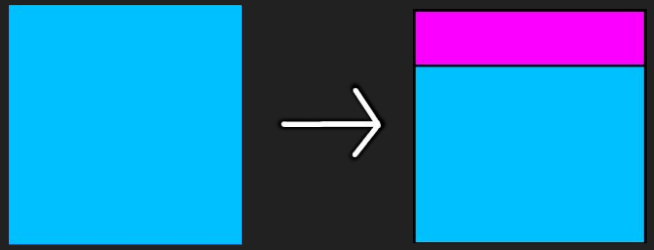
Real Life Example



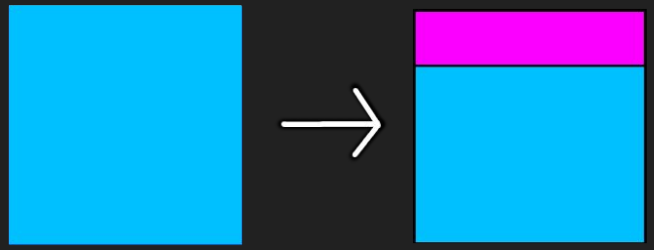
Real Life Example

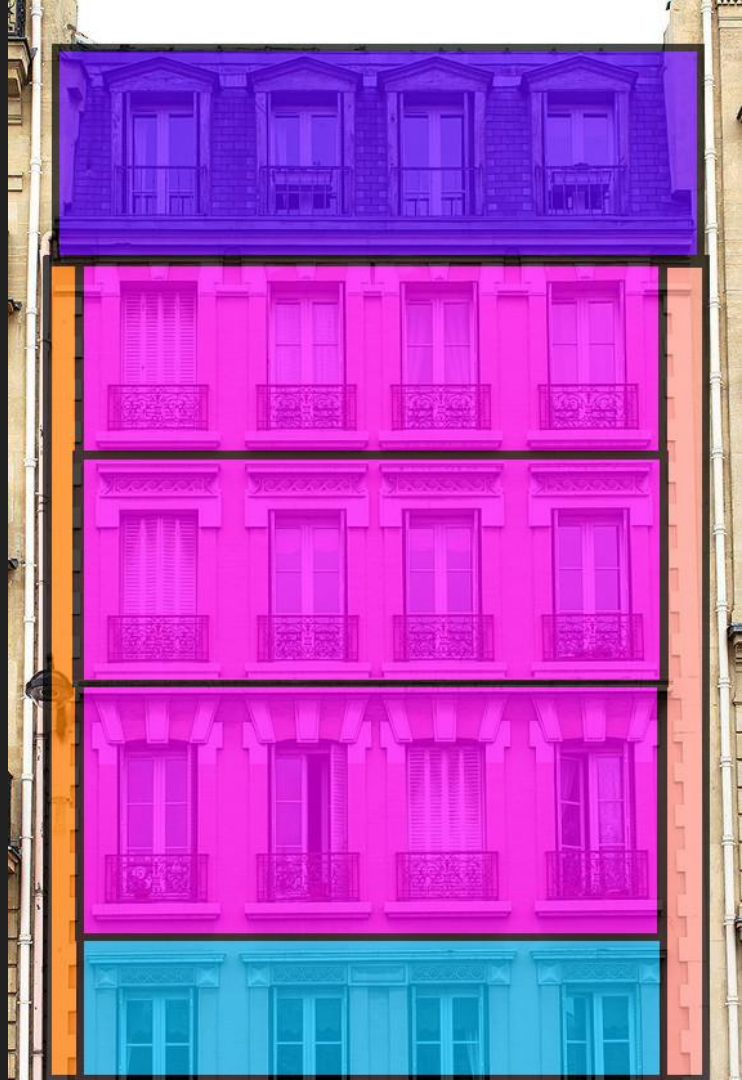


Real Life Example

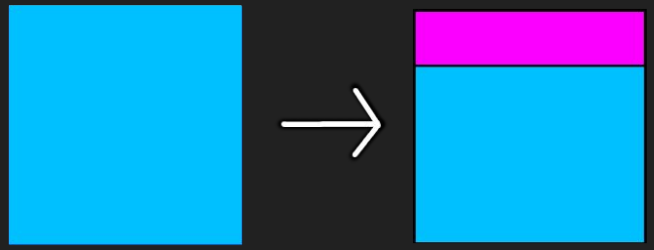


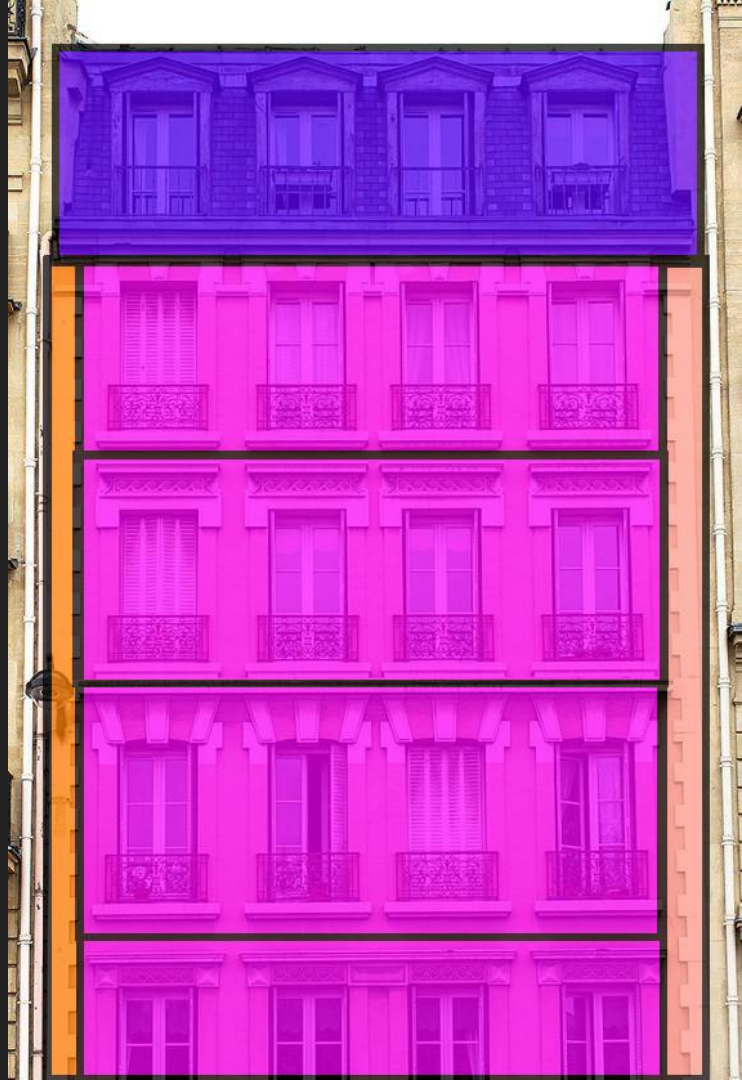
Real Life Example



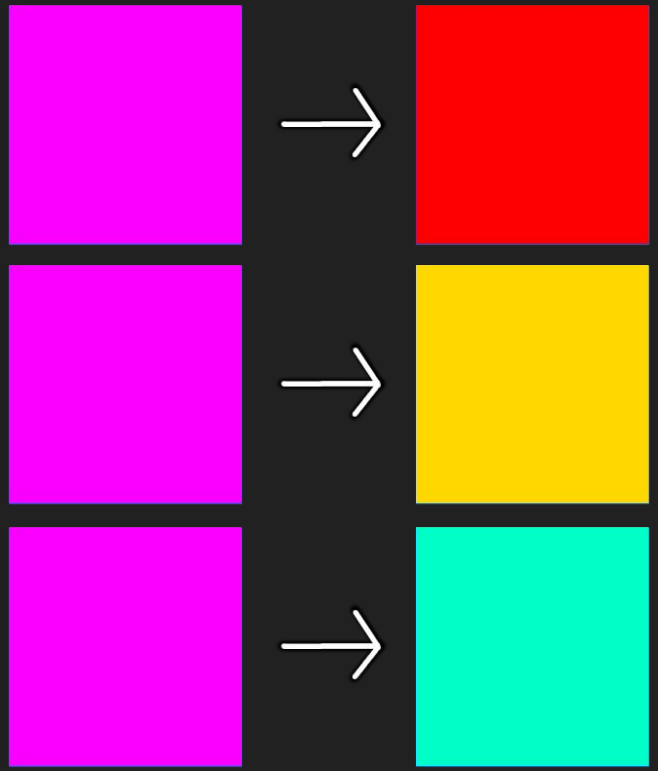


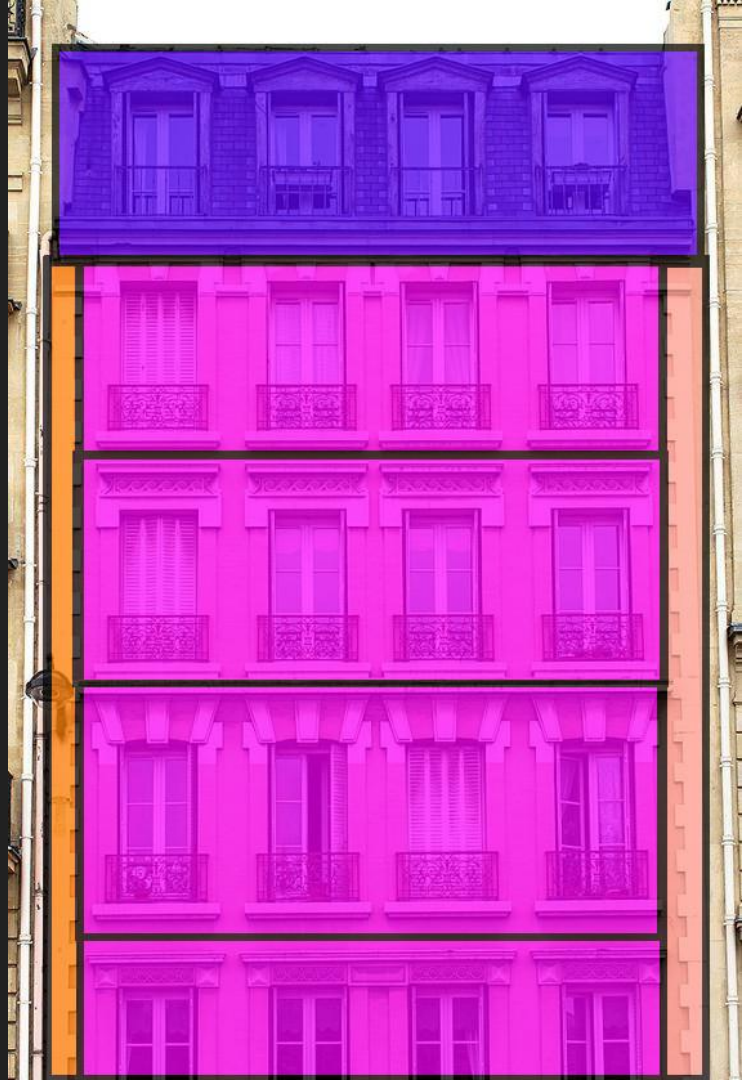
Real Life Example



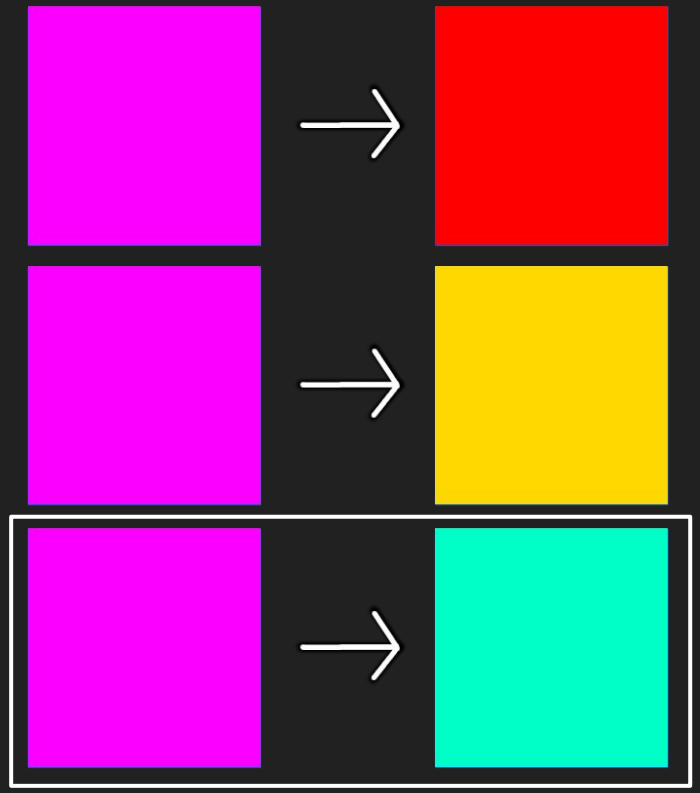


Real Life Example

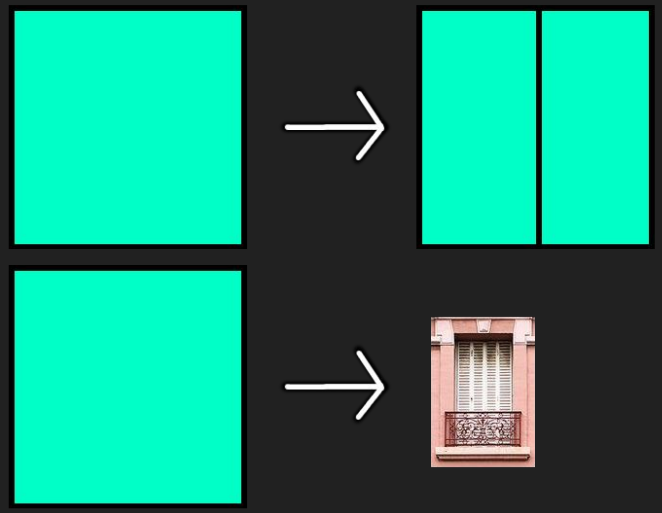




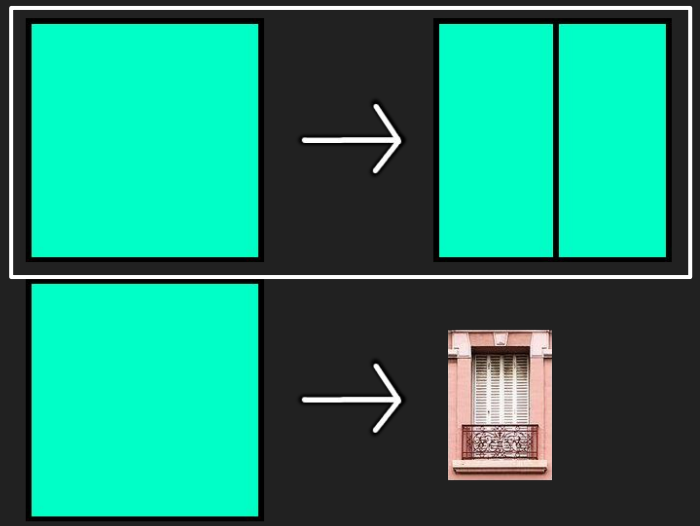
Real Life Example



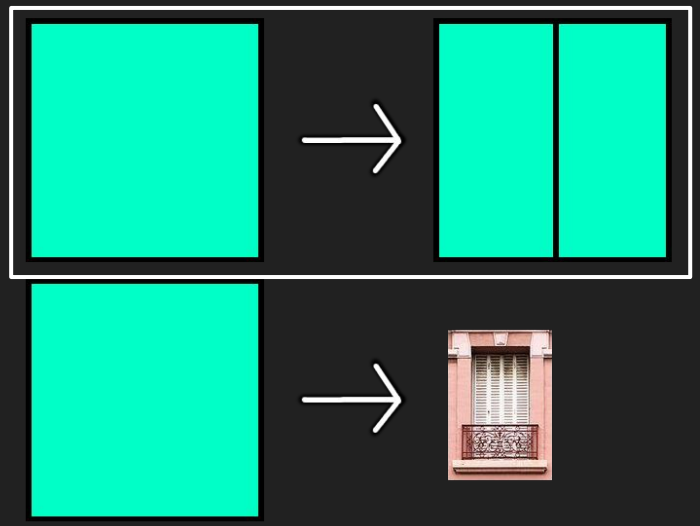
Real Life Example



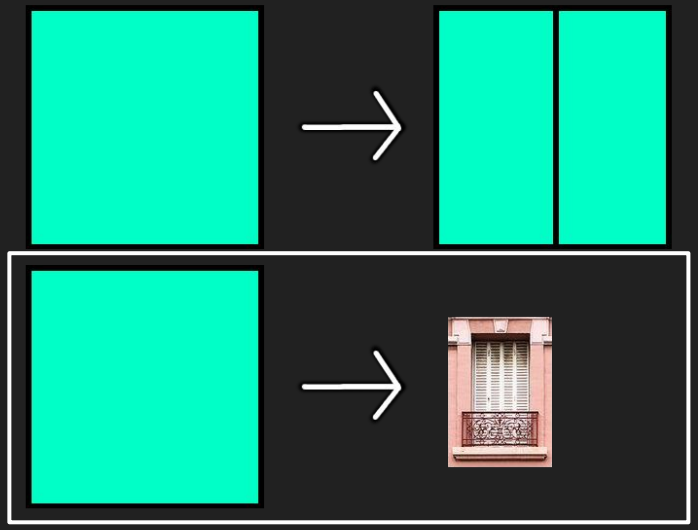
Real Life Example



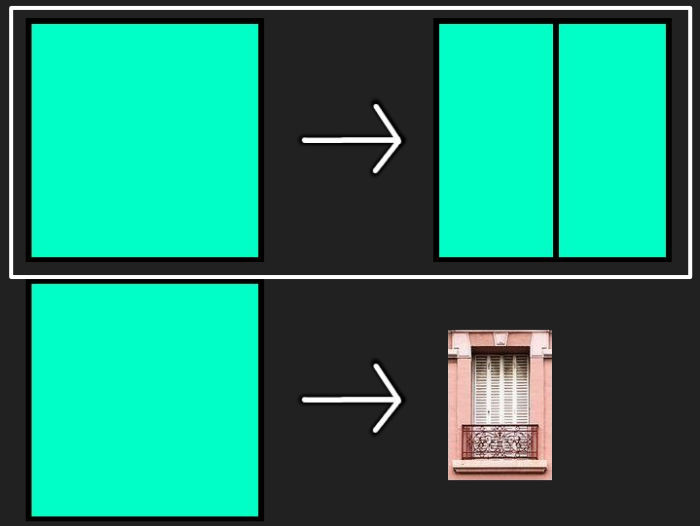
Real Life Example



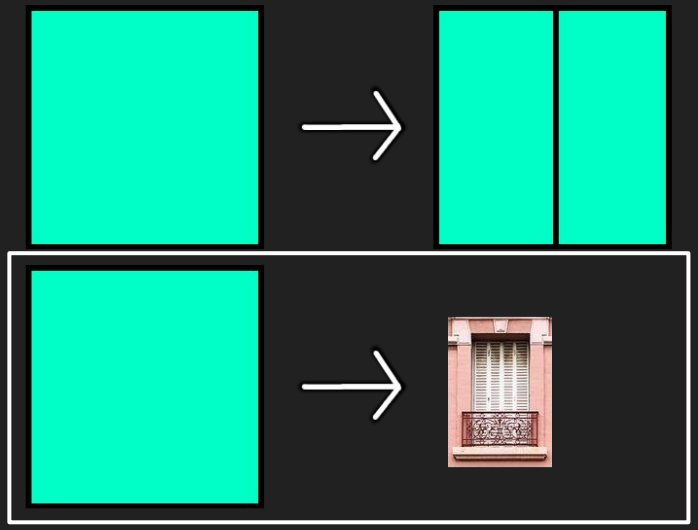
Real Life Example



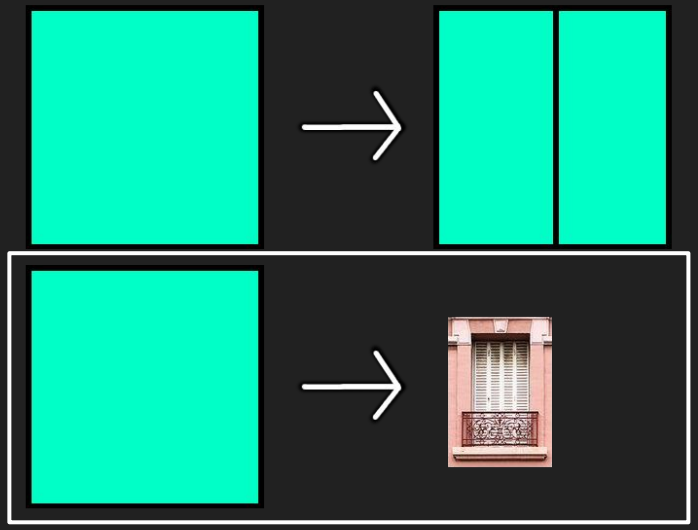
Real Life Example



Real Life Example



Real Life Example



Generating with Spacial Grammars

For each region of space:

- Select a substitution rule (if any)
- Figure out how to divide up the space
- Divide region into the new regions

Generation will terminate when no more rules can be applied

Rule Selection

Common ways to select rules:

- Based on region type
- Based on region size or shape
- Randomly
- (and more)

Dividing Up Space

Can get more variety by dividing space differently:

- Random range of positions
- Divide along random axis
- Divide along largest axis
- Divide along smallest axis
- (and more)

Generating Stuff with Grammars Works Great…

Generating Stuff with Grammars Works Great…

Unfortunately...

Generating Stuff with Grammars Works Great…

Unfortunately…

Grammars aren’t magic

Limitations to Using Spacial Grammars

Like language, blindly expanding grammar rules produces nonsense:

- **The broken seat accumulates the thought.**
- **The few linen translates the curve.**
- **Why does the cork involve the abhorrent chance?**
- **How does the motionless land formulate the swim?**

Limitations to Using Spacial Grammars

- Can generate bad results
- A lot of duplicate structure
- Decisions are made in isolation from other parts of the architecture
- Space doesn’t always divide up nicely
- Hard to do with general shapes

Limitations to Using Spacial Grammars

Solution: Can generate bad results

- Strict rules can be used to limit bad results
- Results can be processed to correct problems

Limitations to Using Spacial Grammars

Solution: A lot of duplicate structure

- Solved this by making macro-like functions to generate parts of the grammar
- Might be worthwhile exploring using the functions directly

Limitations to Using Spacial Grammars

Solution: Decisions are made in isolation from other parts of the architecture

- Separated aspects of the level generation in to separate parts
- Some of the parts consider the level as a whole

  (more details coming soon)

Limitations to Using Spacial Grammars

Solution: Space doesn’t always divide up nicely

- Always have default rules to fall back on
- Test and iterate

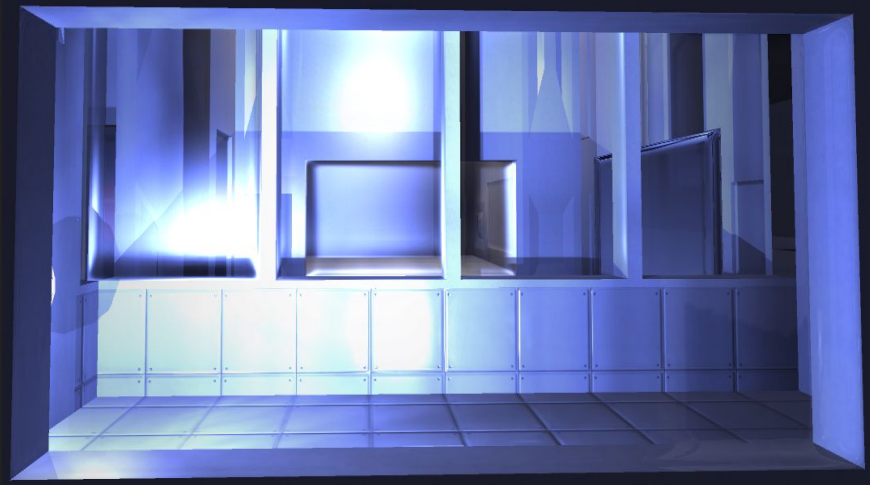
Limitations to Using Spacial Grammars

Solution: Hard to do with general shapes

- Avoiding the problem by always dividing into axis aligned boxes

Generator Overview

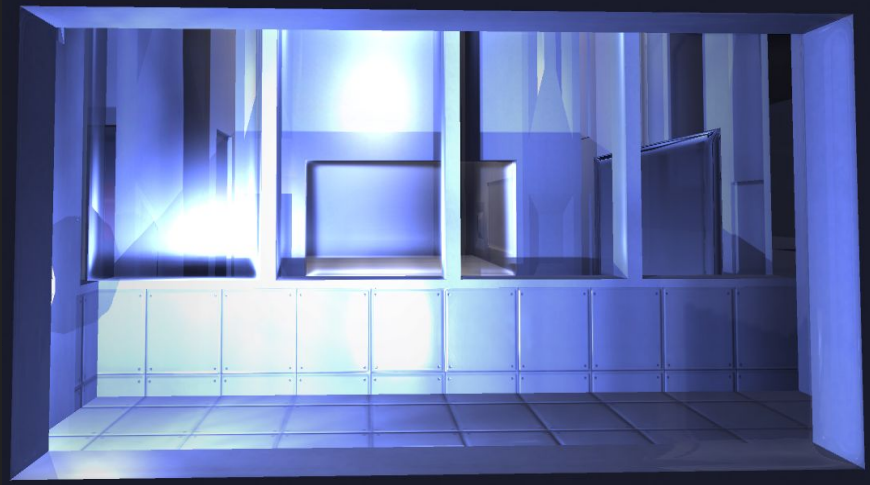
- The generation process does not purely use grammars
- The process to generate a levels is broken down into smaller steps
- This helps to keep things simple and to overcome some of the limitations of using a pure grammar



Generator Terminology

Region

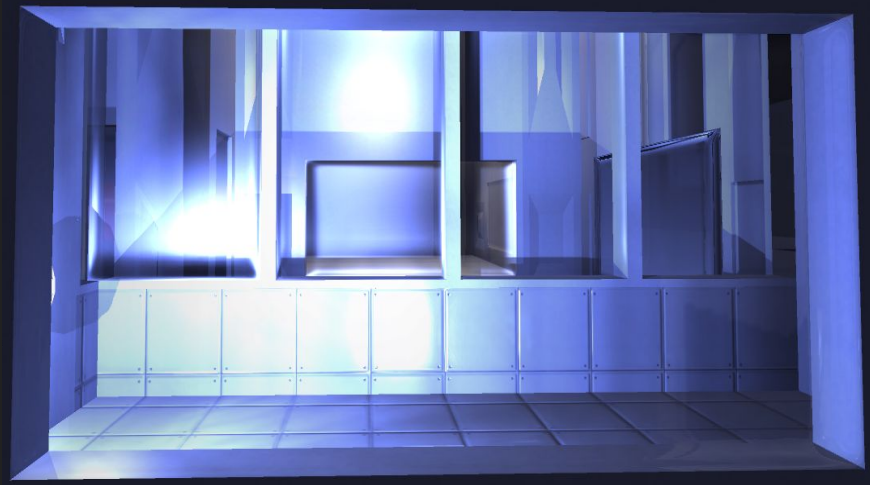
- AABBs with a type associated with them
- Responsible for creating things like:
  - Building shape
  - Hallway sections
  - Rooms
  - Elevators and staircases
  - Etc.



Generator Terminology

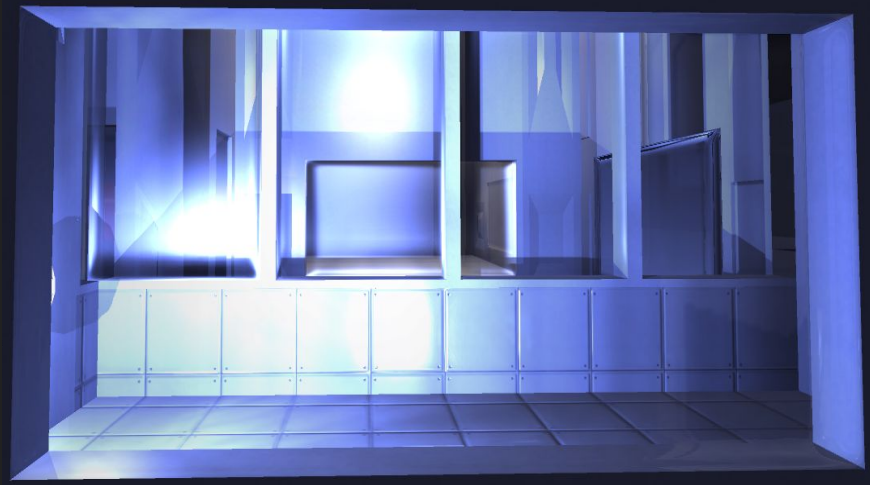
Surface

- AABBs with a type associated with them
- Will generate meshes and portals
- Responsible for creating things like:
  - Walls
  - Floors
  - Windows
  - Doors
  - Etc.



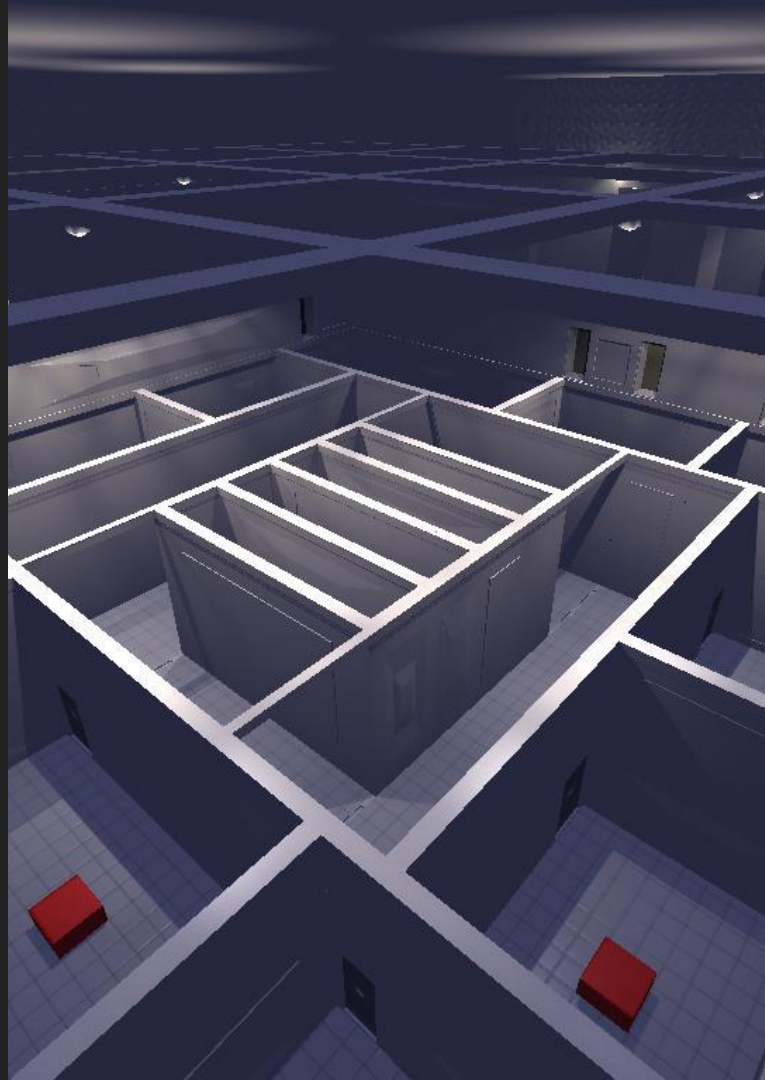
Generator Overview

1. Divide up space in to regions
2. Select surfaces
3. Fix problems with region connections
4. Create surfaces
5. Place objects

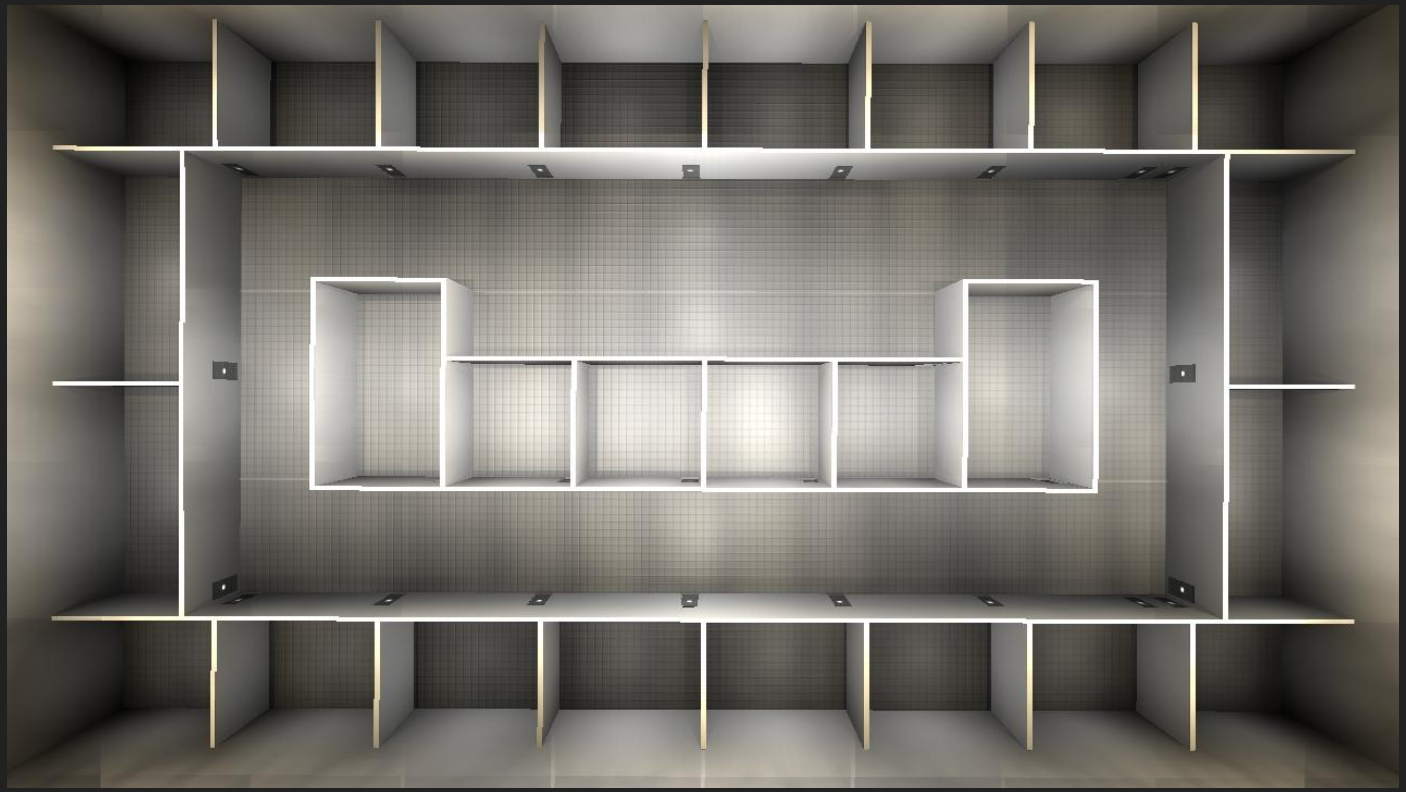


Region Generation

- Divides 3D space in to smaller parts using a spatial grammar
- Each region will be an AABB
- All divisions will be axis aligned



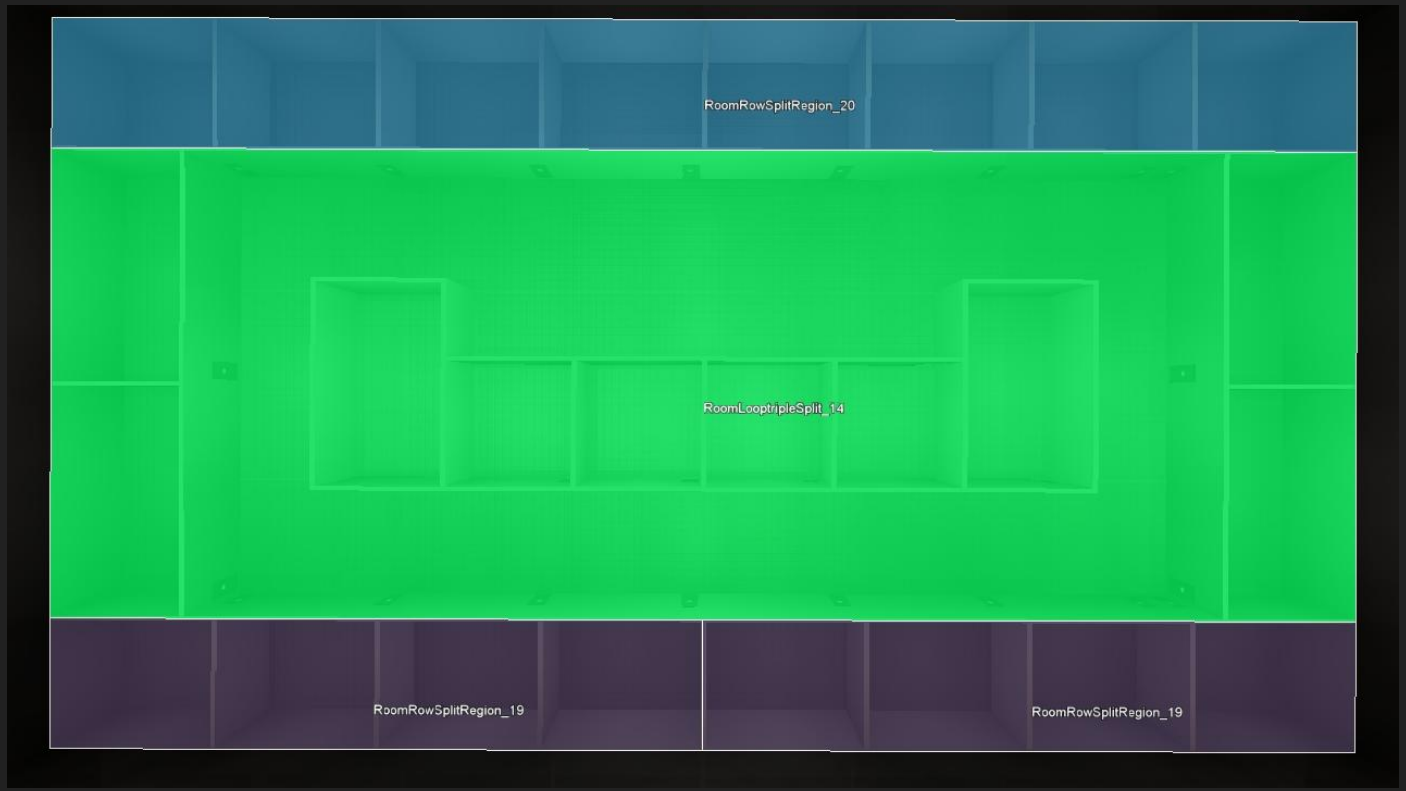
Region Generation Example



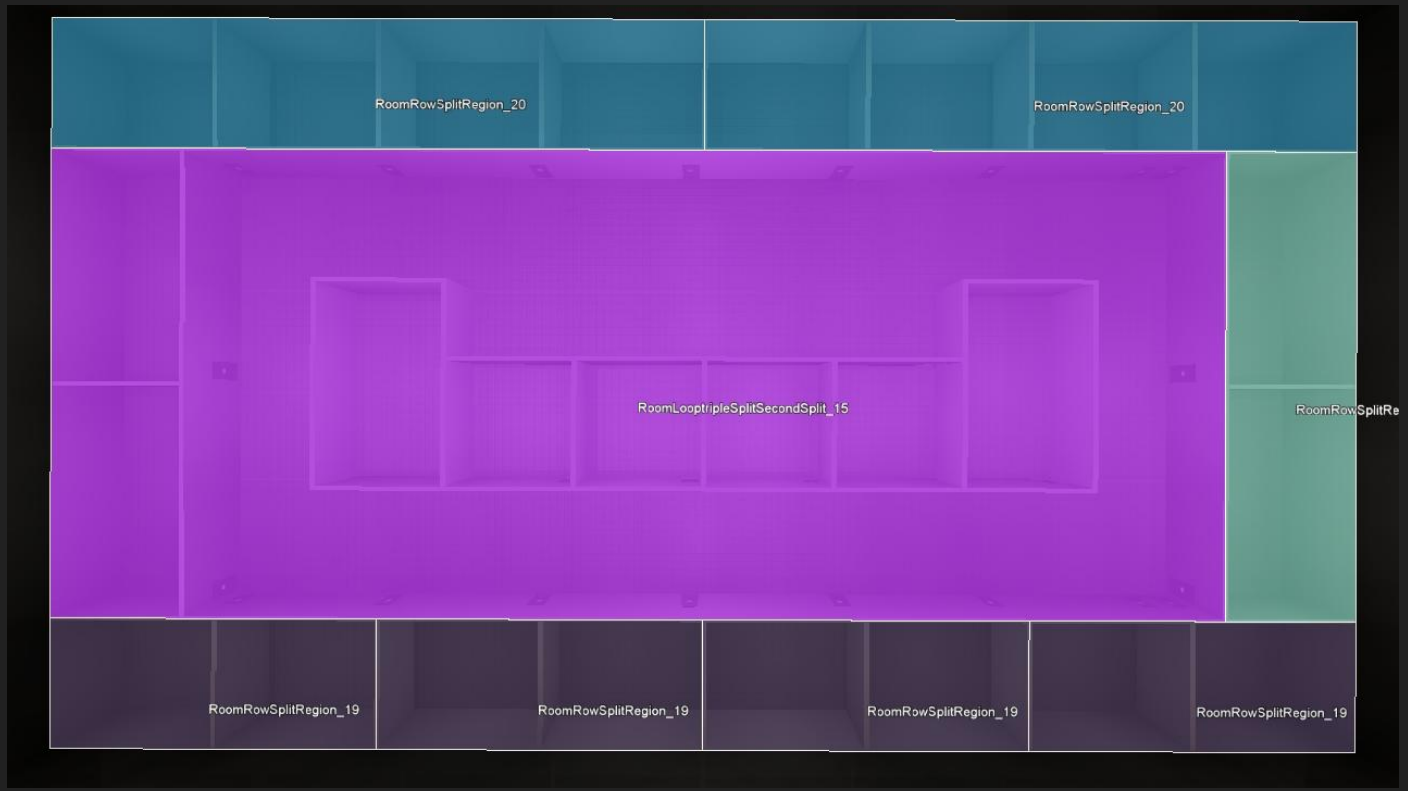
Region Generation Example

Region Generation Example

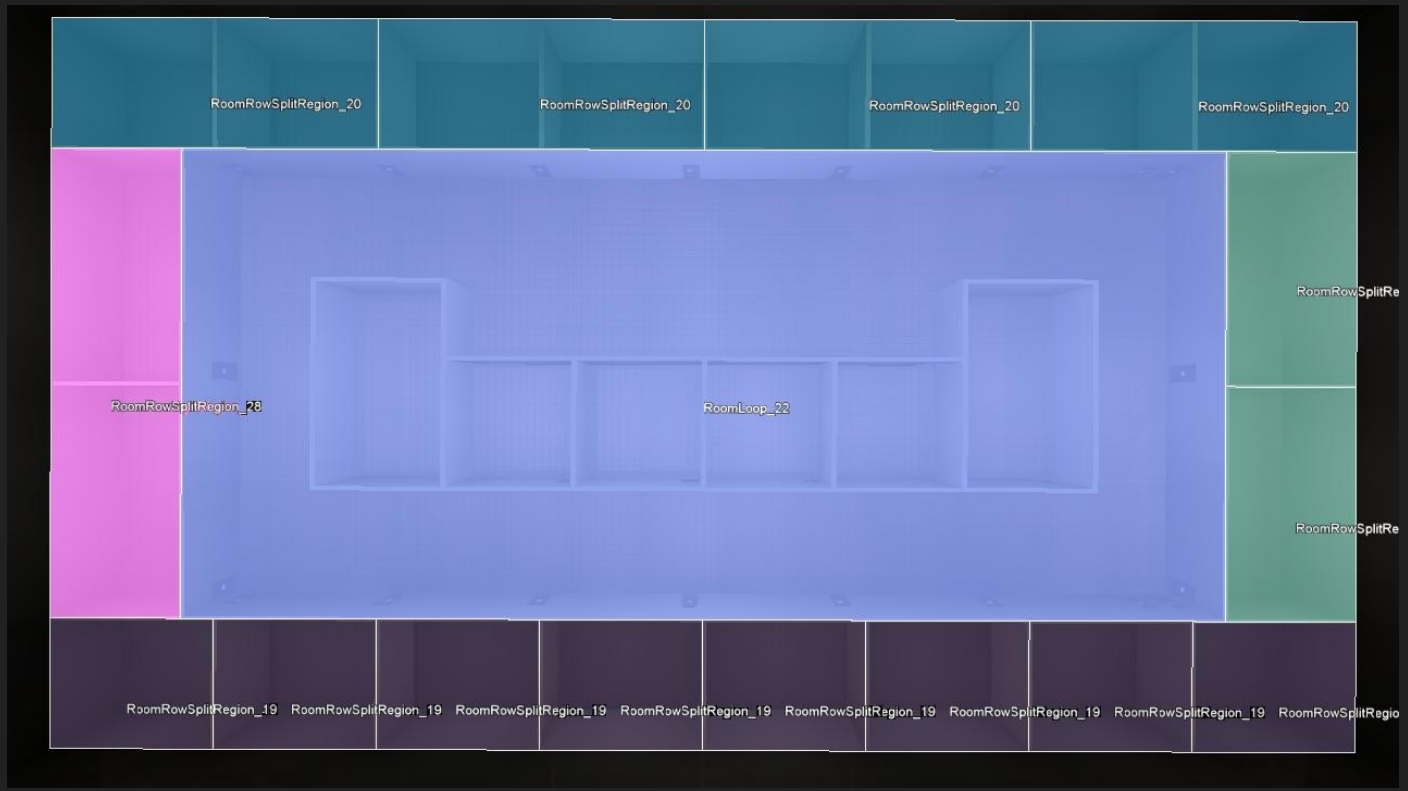
Region Generation Example



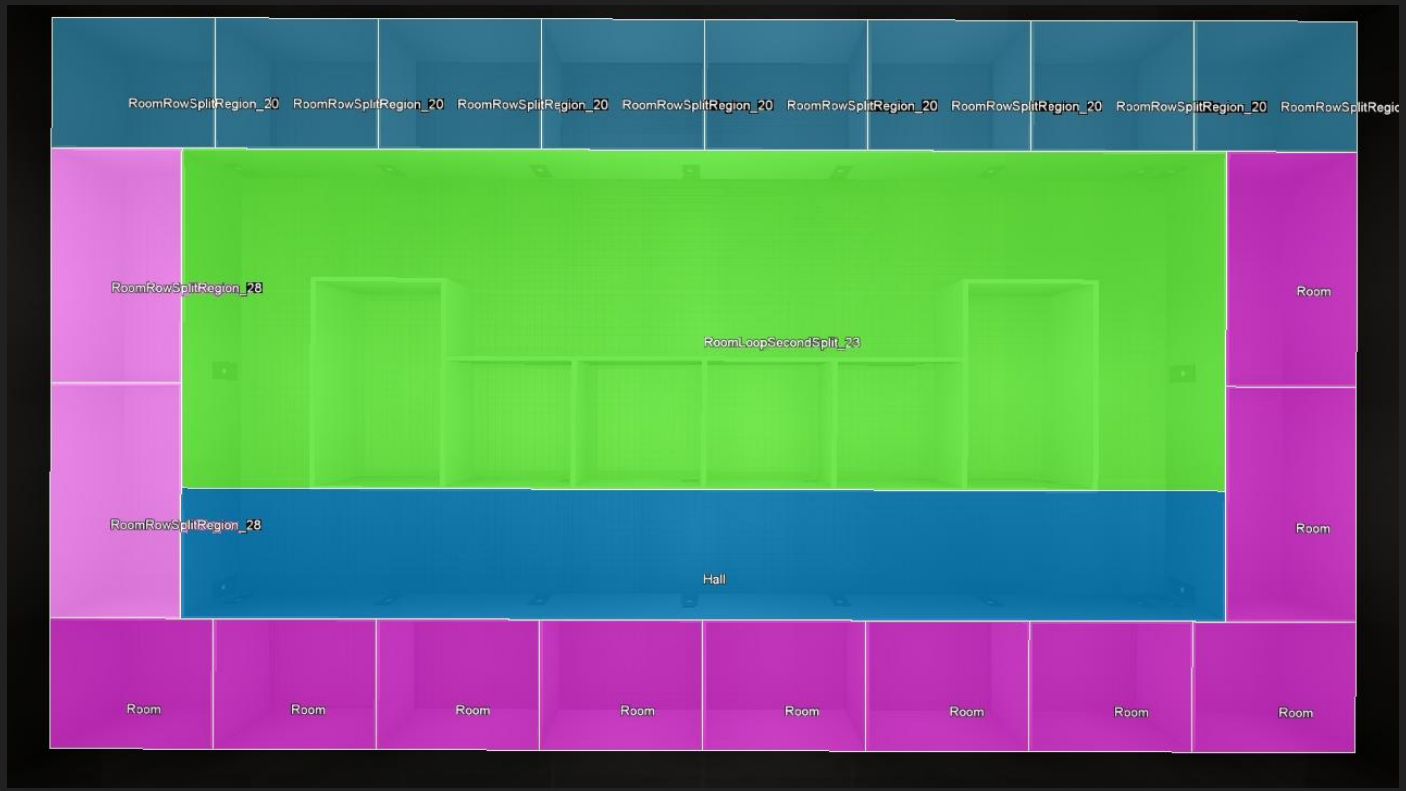
Region Generation Example



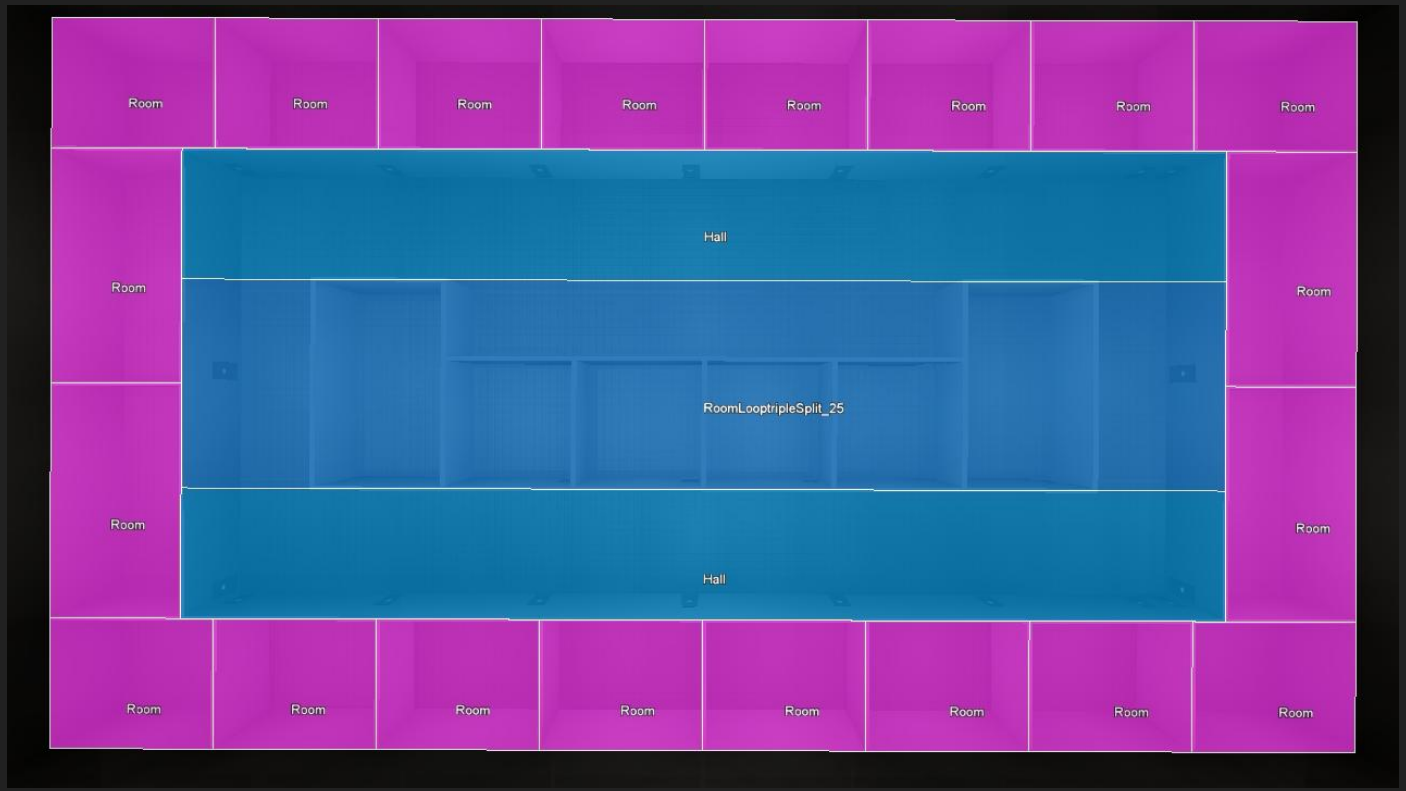
Region Generation Example



Region Generation Example



Region Generation Example



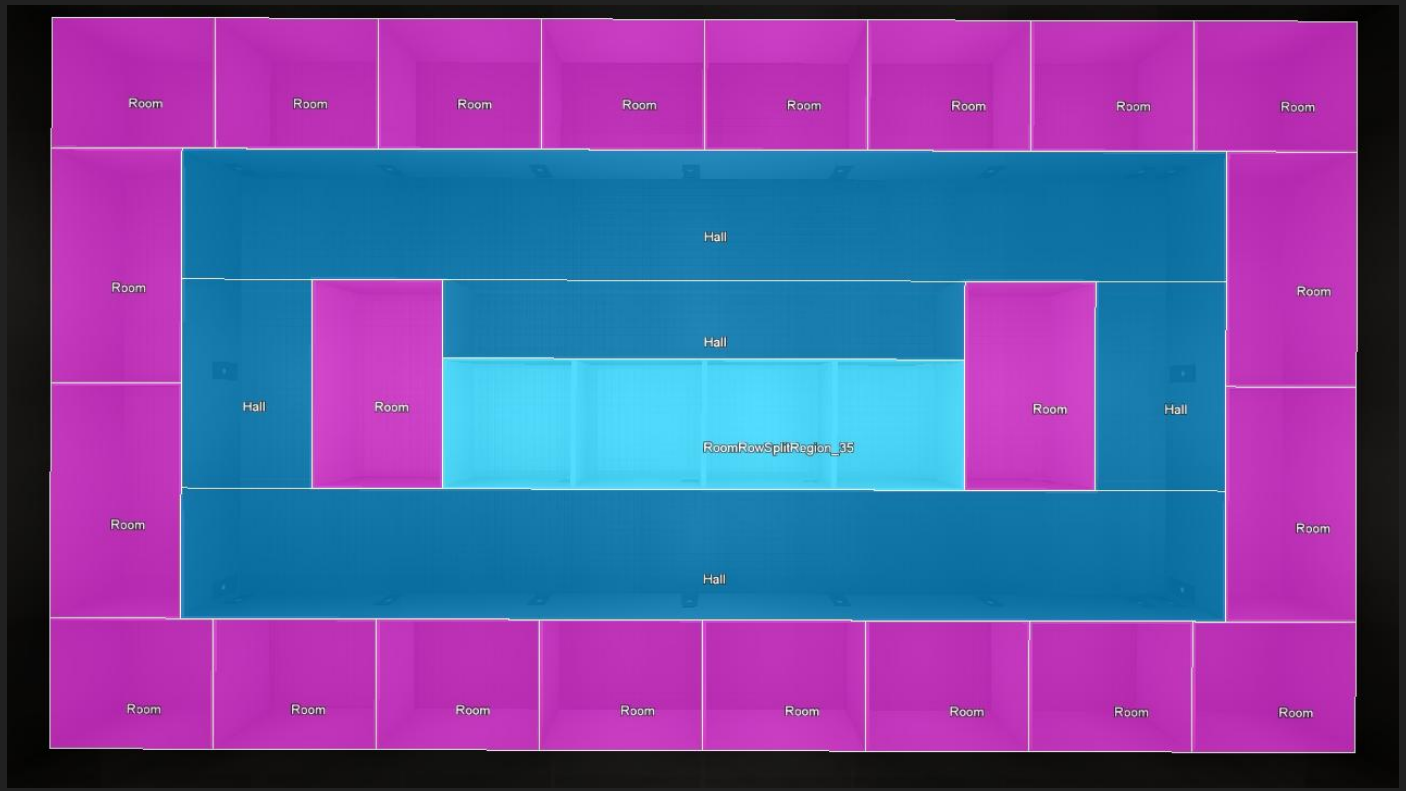
Region Generation Example

Region Generation Example

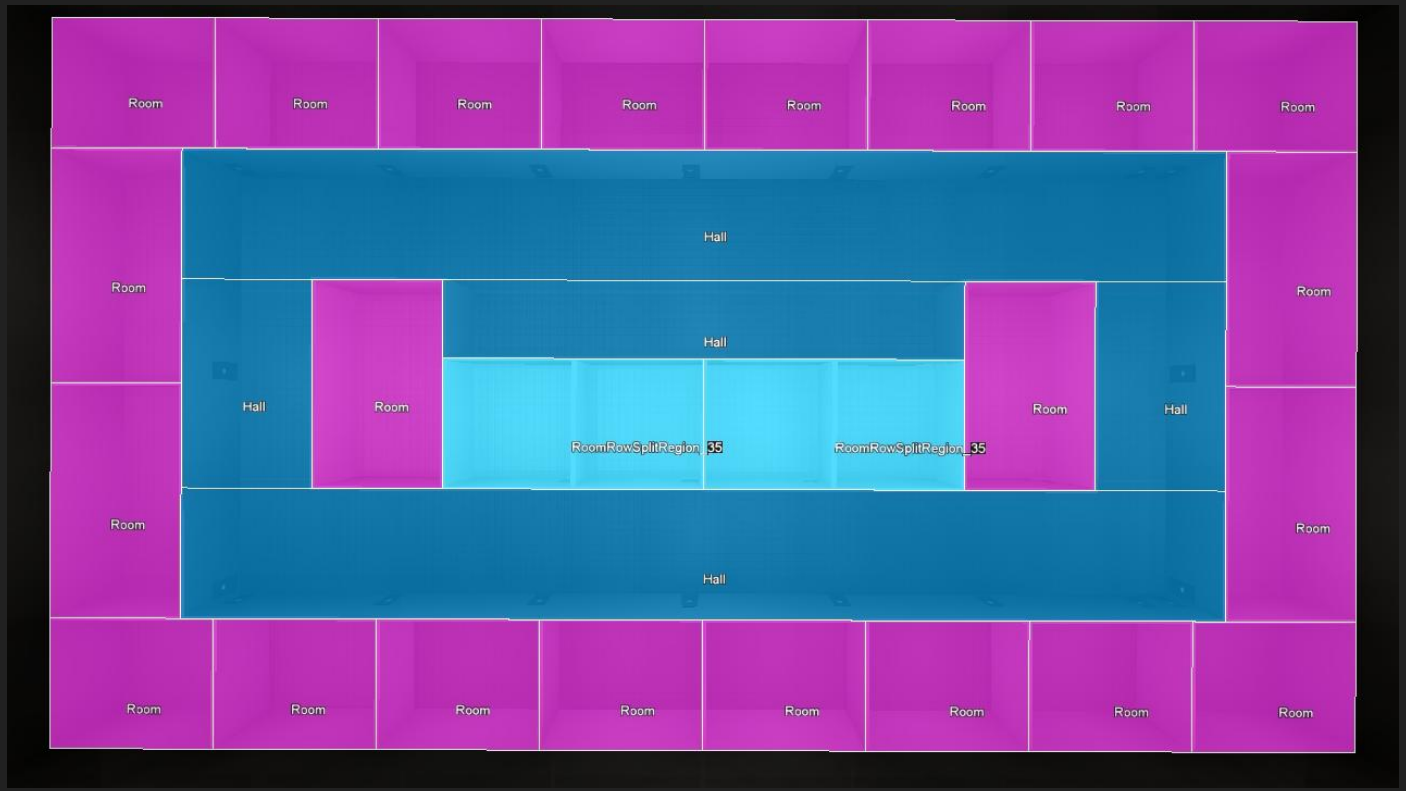
Region Generation Example

Region Generation Example

Region Generation Example



Region Generation Example

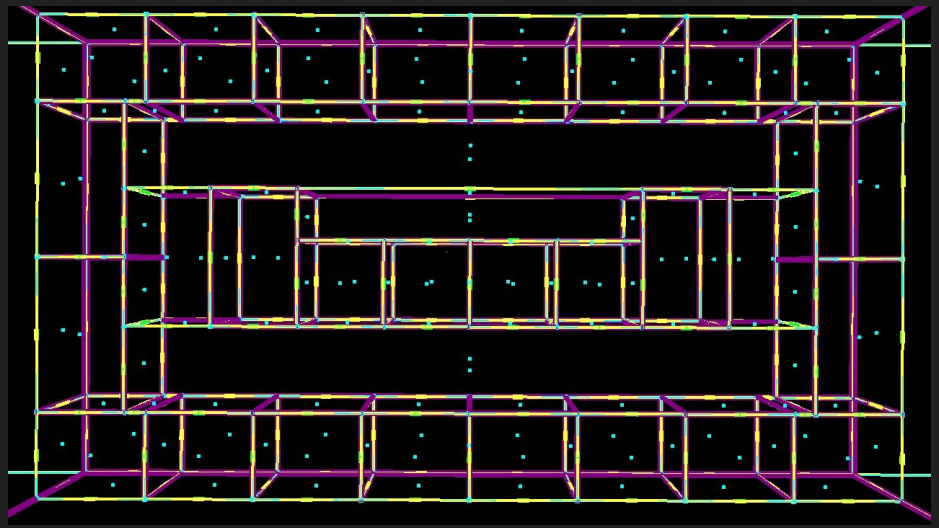


Region Generation Example

Region Generation Example

Generating Region Connection Info

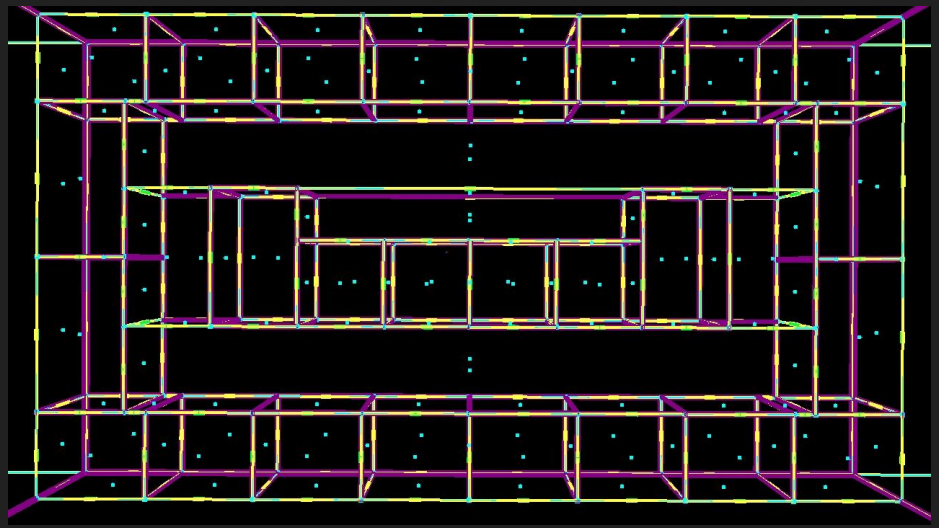
- Need to track how regions are connected
- The connection info is updated each time a region is split
- Produces a graph structure that will be used to solve connection problems, and generate geometry



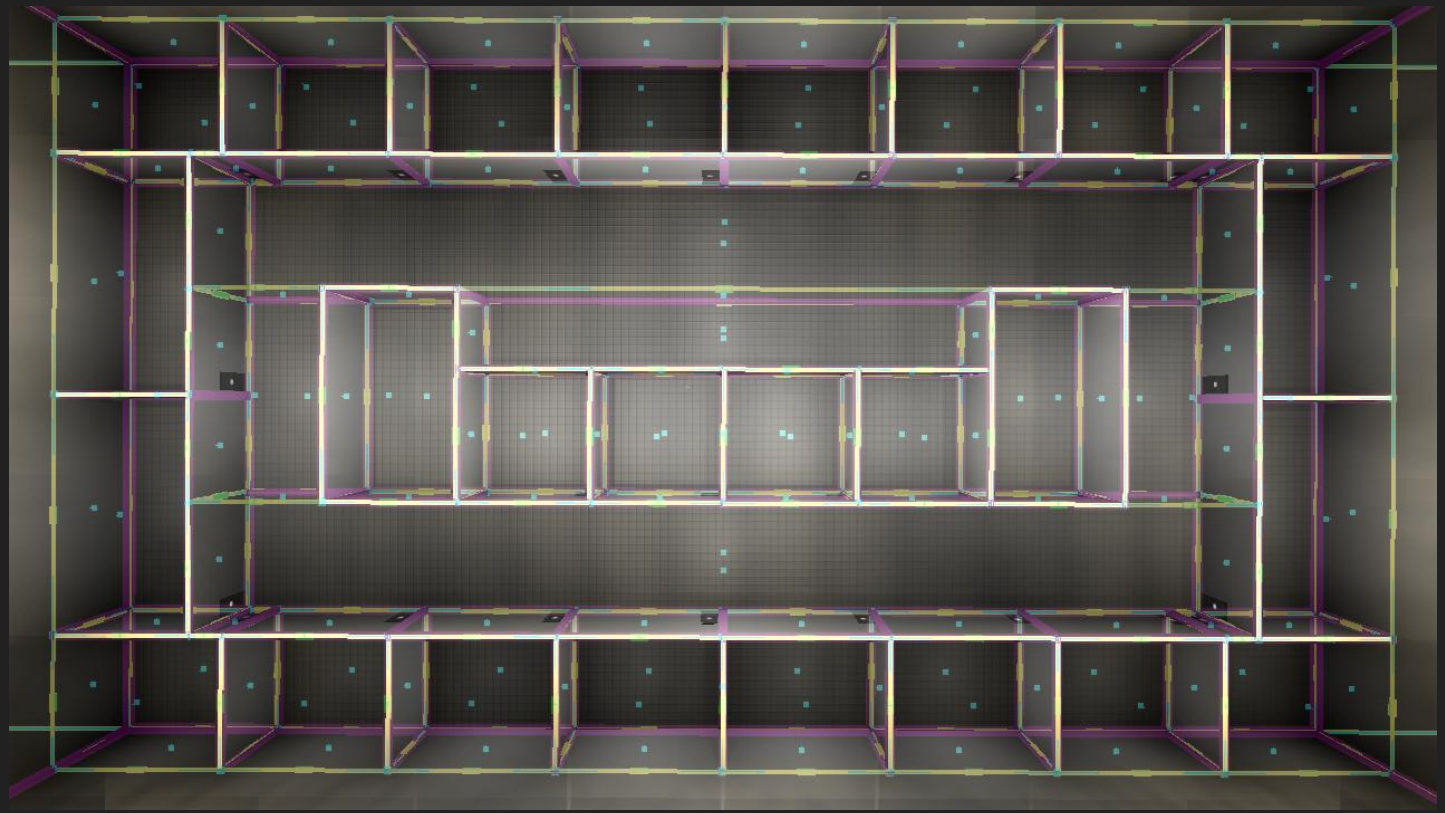
Generating Region Connection Info

This tracks:

- Connecting faces between regions
- Connecting edges between faces
- Connecting vertices between edges



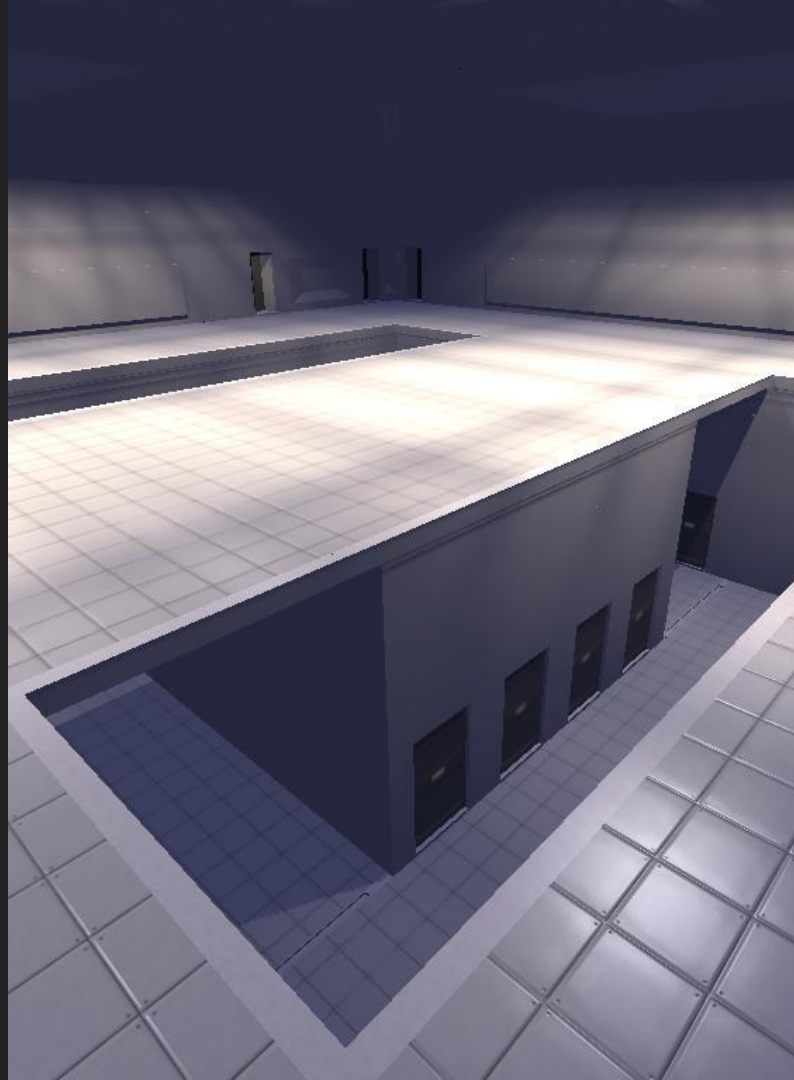
Generating Region Connection Info



Connecting Surface Selection

- Initial base surface types will be selected for the connecting faces, edges, and vertices

- Surfaces selected based on a number of rules

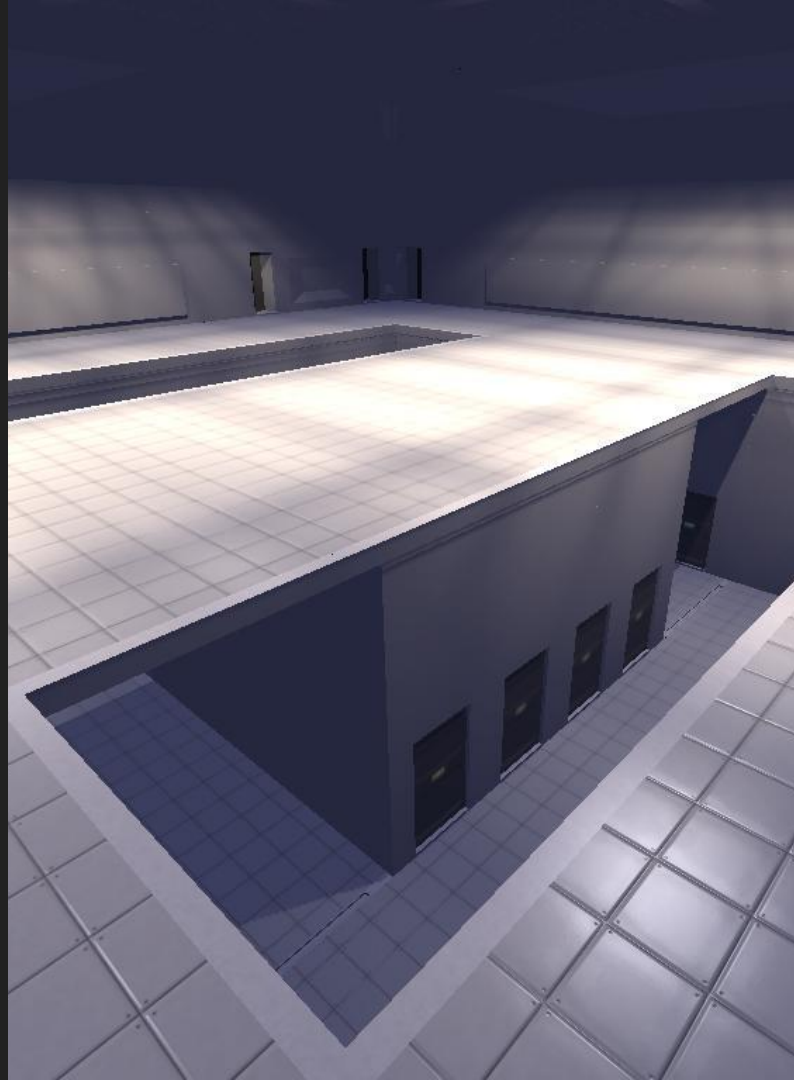


Connecting Surface Selection

Selection rules based on:

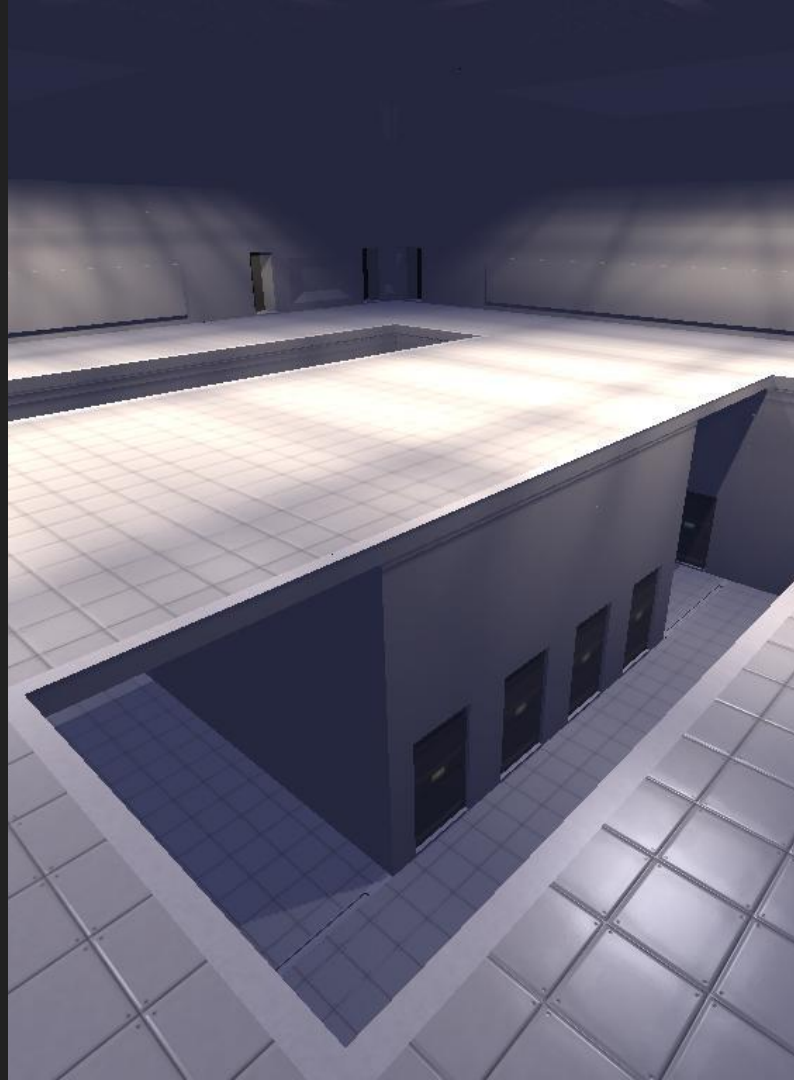
- Connecting types
- Dimensions of connection
- Orientation of connection
- Whether surface can be traveled through

There will always be a fallback surface if all rules fail.

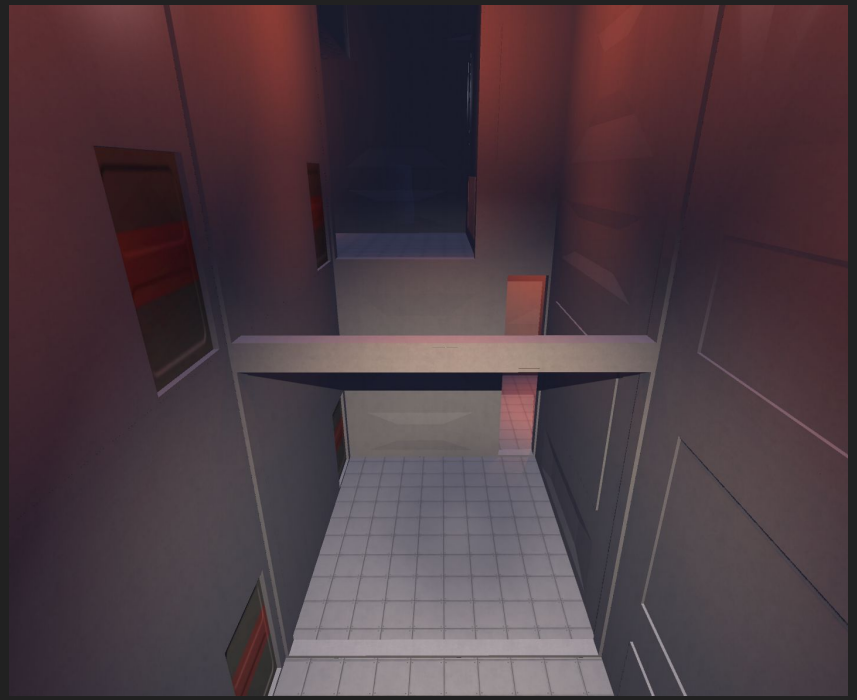


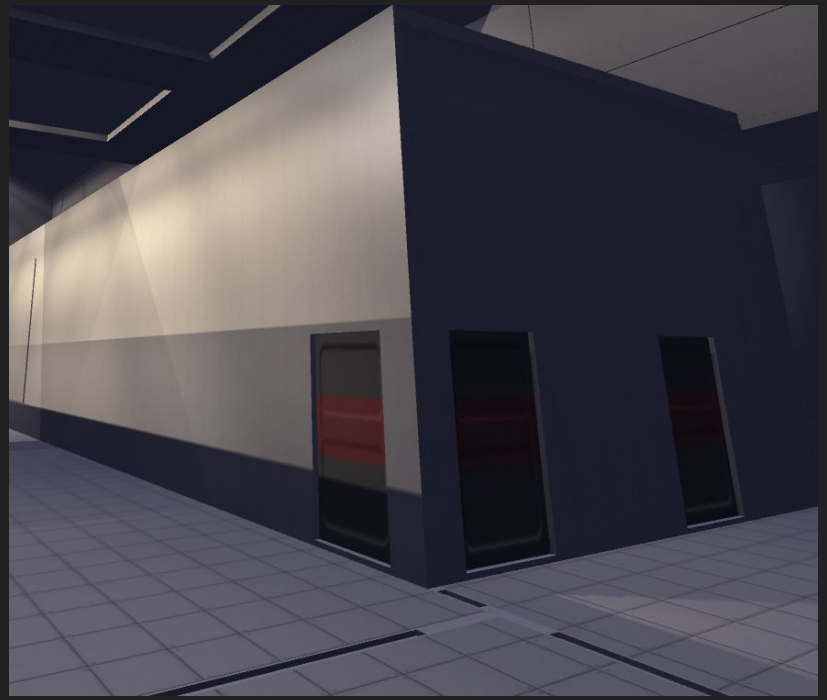
Connecting Surface Selection

- Solid walls between rooms
- Empty space between halls
- Wall with door between halls and rooms
- Wall with windows between exterior regions and halls/rooms
- Floor with hole and railings to connect hallways on the XY plane
- etc.



Sometimes problems occur with simple rules...

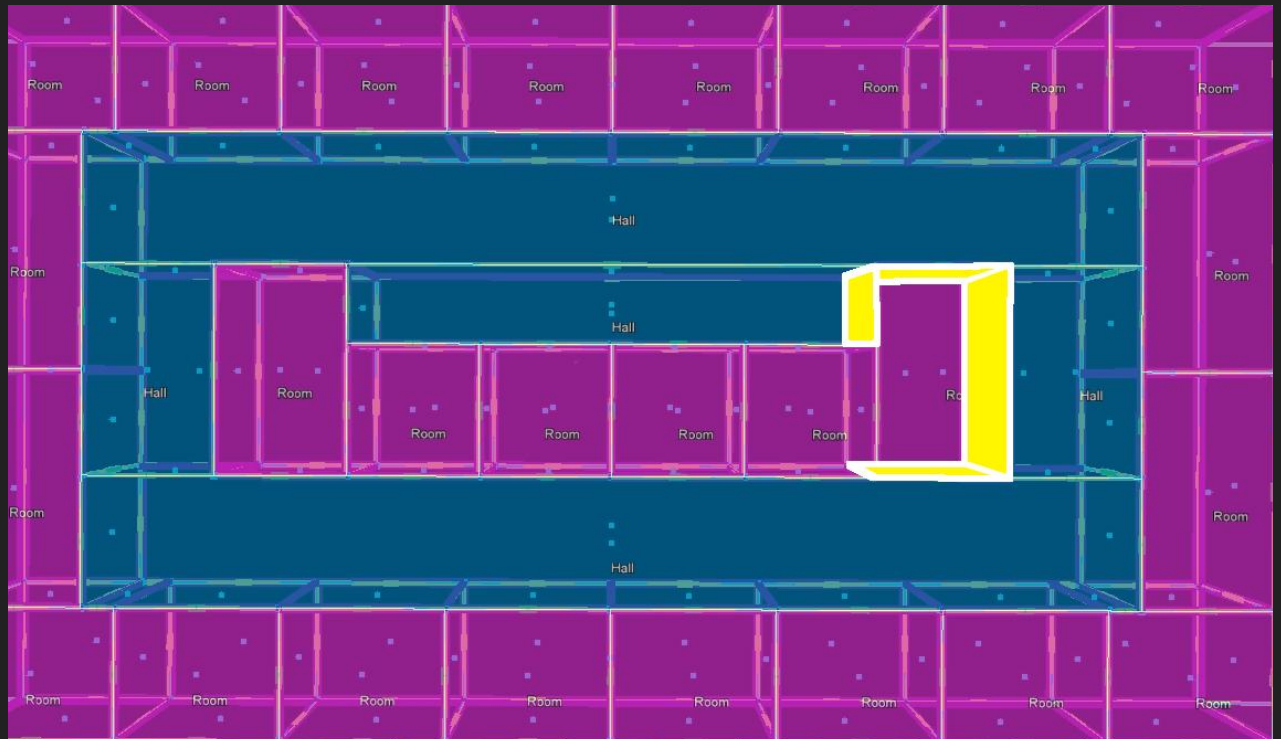




Solving Connection Problems

- Ensure better flow through the level by traversing over faces, and adding / tweaking connections
- Mostly want to add connections to fix disjoint regions

Common Problem Situation



Solving Connection Problems

- Specify how many travel connections regions should have and remove them as necessary
- Currently removing them in a weighted random fashion
- Only removing them if it won’t cause regions to become disjoint

Surface Generation

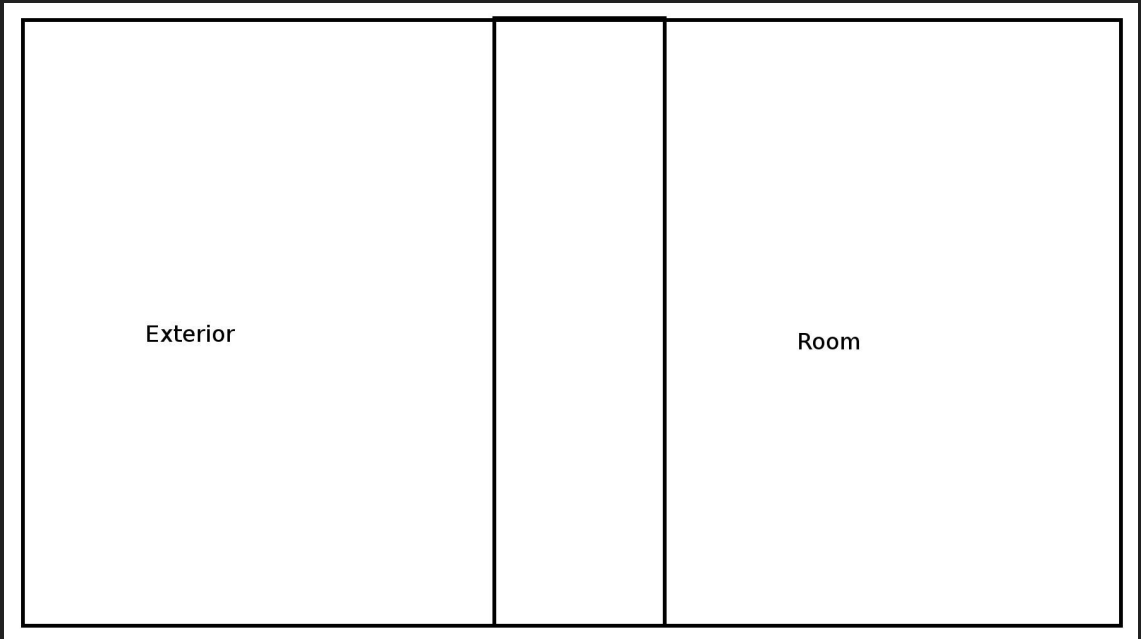
- Generated by dividing space with a grammar

- Terminal surfaces create:
  - Mesh geometry
  - Portals to connect regions
  - Physics collisions

Surface Generation

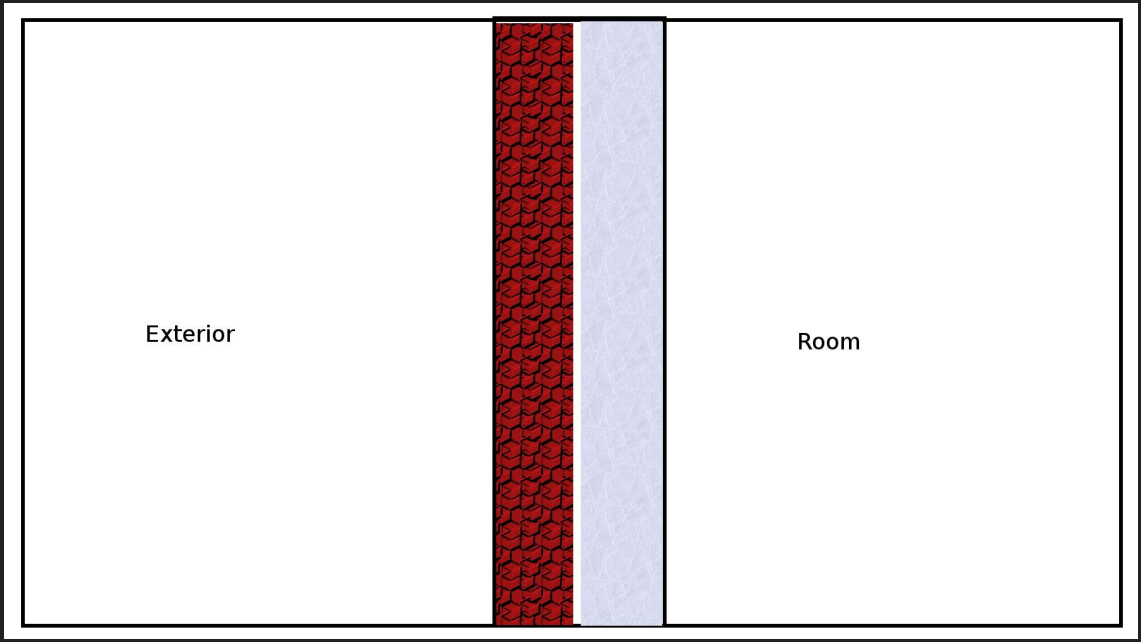
- Dividing up surfaces is a little bit more complicated than regions
- Typically want to generate different things for floors, ceilings, room specific walls, etc
- Surfaces are responsible for generating portals though

Surface Generation - Without Portals



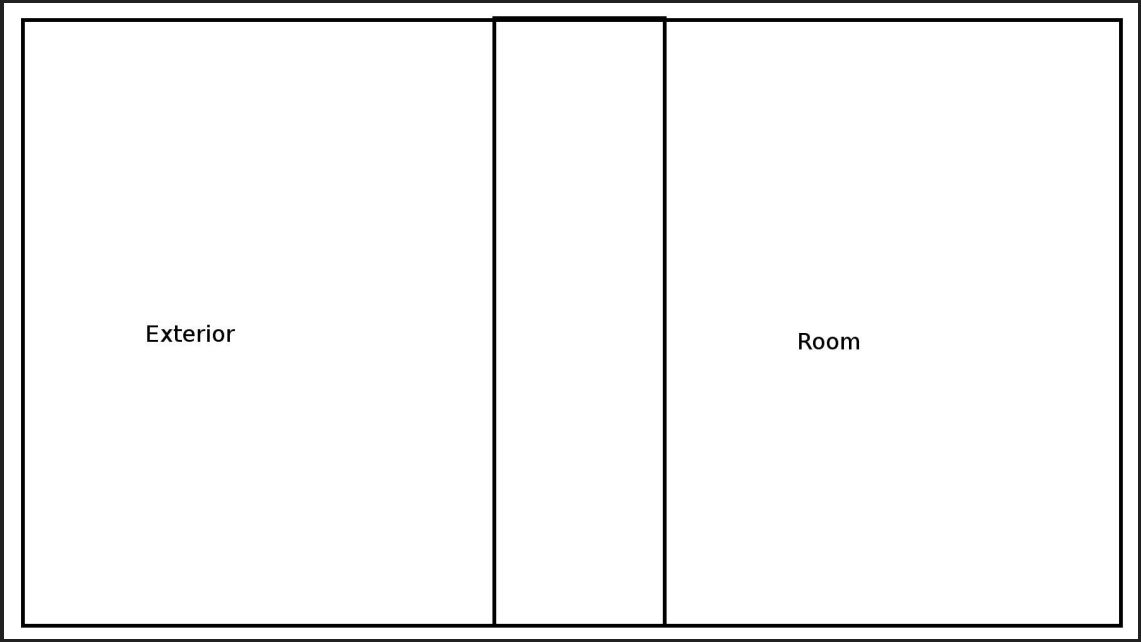
Surface generation without portals is straightforward.

Surface Generation - Without Portals



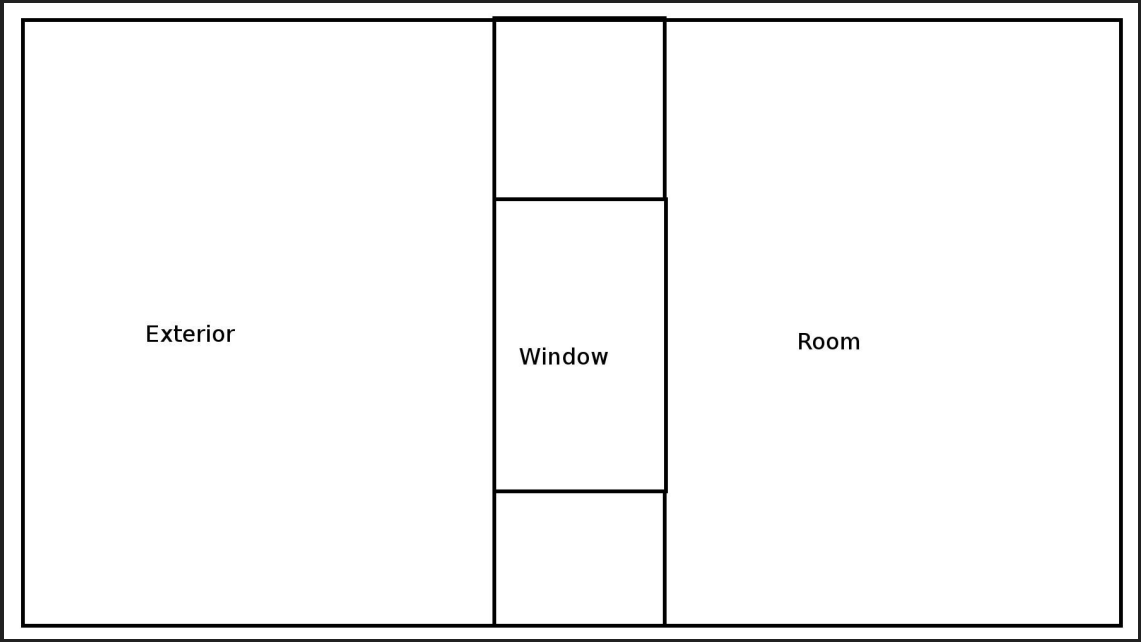
Surface generation without portals is straightforward.

Surface Generation - With Portals



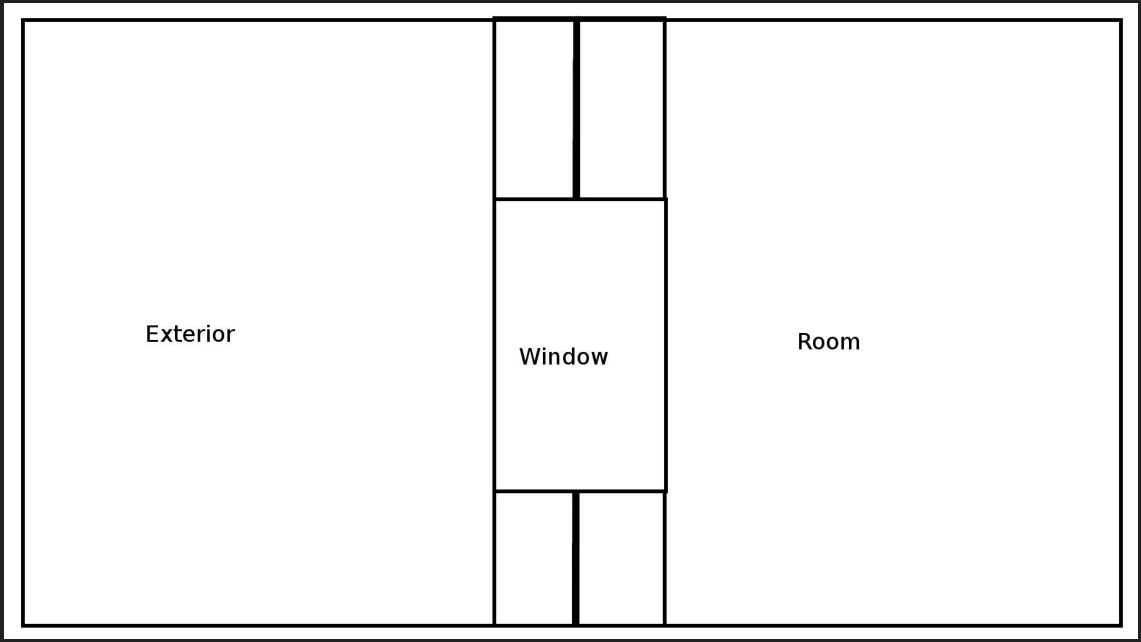
Portals make things a bit more complicated

Surface Generation - With Portals



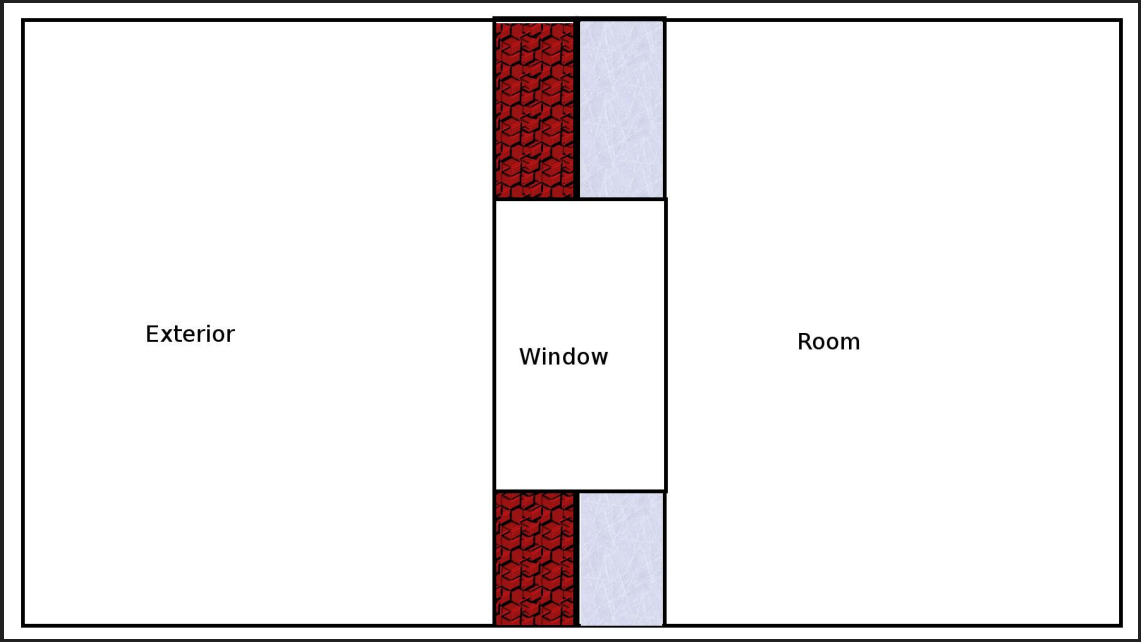
Expand only the parts of the grammar that could contain a portal

Surface Generation - With Portals



Everything left will be safe to split up without ruining the connections

Surface Generation - With Portals

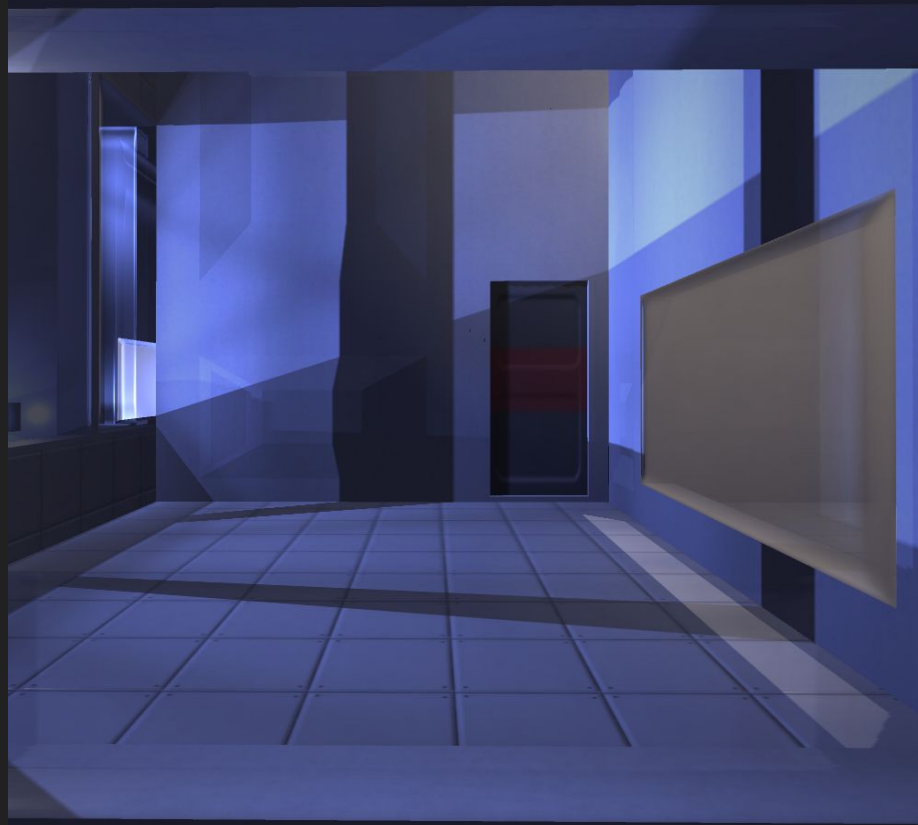


Each side can be generated in isolation now

Discussion

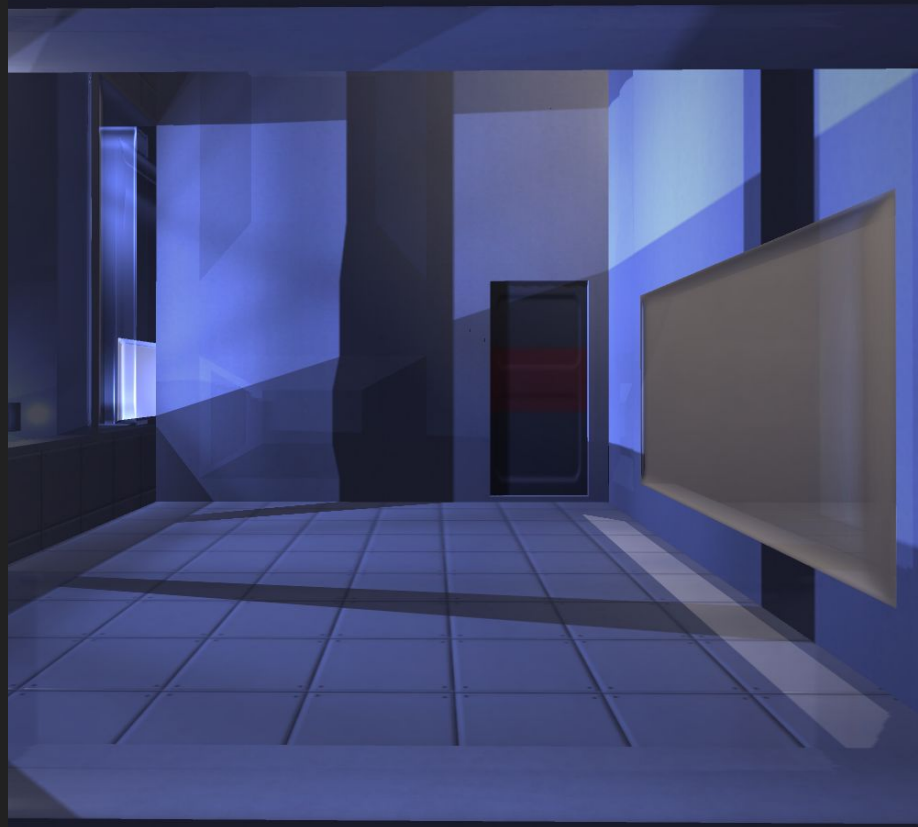
- Happy with the results so far
  - Versatile
  - Fast
  - Only requires simple meshes

- Biggest complaint: Authoring in plain text can be difficult



Discussion

- Generated levels aren’t as good as hand authored levels
- Lots of code
- Fairly easy to test parts in isolation
- Recommend automated testing



Questions?

Evan Hahn

Email: evan.hahn@snowedin.ca

Twitter: @ehahnda

Web: snowedin.ca