

# KARTOGRAPH

## PROCEDURAL LEVEL GENERATOR

By Juan Rodriguez. 2024

Index

[Introduction](#)

[Requirements](#)

[Usage](#)

[Level Generator](#)

[Sections](#)

[Bounds](#)

[Exits](#)

[Advanced Exits](#)

[Ends](#)

[Special Rules](#)

[Support](#)

[Examples](#)

# Introduction

Kartograph is an asset for Unity Game Engine that allows creators to quickly build procedural levels by connecting previously defined prefabs together. The tool provides a lot of control to customize the creation process. Here is a complete list of features in this tool:

- Full control of each section, including shape and size.
- Seed input for generating specific levels and/or daily runs and challenges.
- Full control of the types of sections that are to be built each iteration.
- Optional control for specifying which exit generates which section.
- Versatile rule system which allows the user to limit or force the amount of specific sections.
- Virtually unlimited capacity of sections and section types.
- Efficient generation algorithm written in clean code.
- Support for own implementation, all code is inheritable.
- Support for 3D and 2D maps for any type of games.
- Support for every shape, not confined to a grid or 90-degree layouts.
- Lightweight assemblies, no need to recompile Kartograph code while working on the game.
- Full debug mode (Only available in pro version).
- Generate-in-editor button (Only available in pro version).
- Developer support (Only available in pro version).

It is highly recommended to also watch the video tutorials:

- [Basic Tutorial](#)
- [Advanced Tutorial](#)

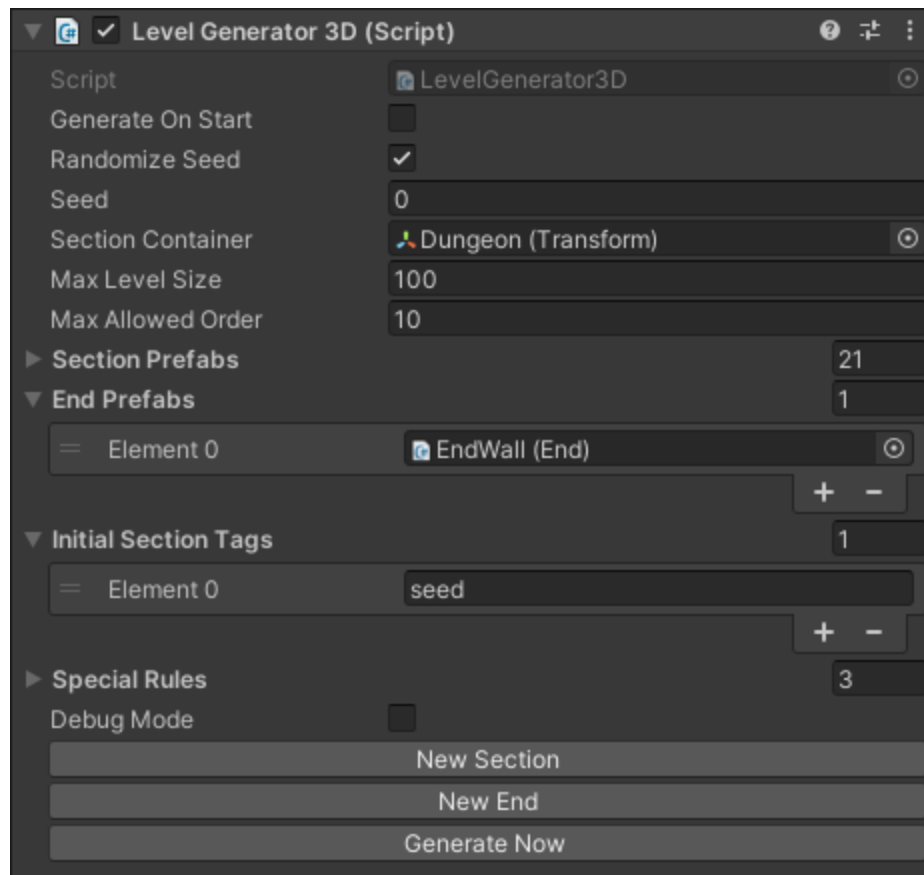
## Requirements

This asset uses syntax and features from C# 6 so it is required to configure the project to use .Net Framework 4.x

# Usage

## Level Generator

Let's get started with how to use the level generator, first of all, create a game object which will be your level generator object, it can act as the level container itself. Add a component called LevelGenerator(3D or 2D) to it.



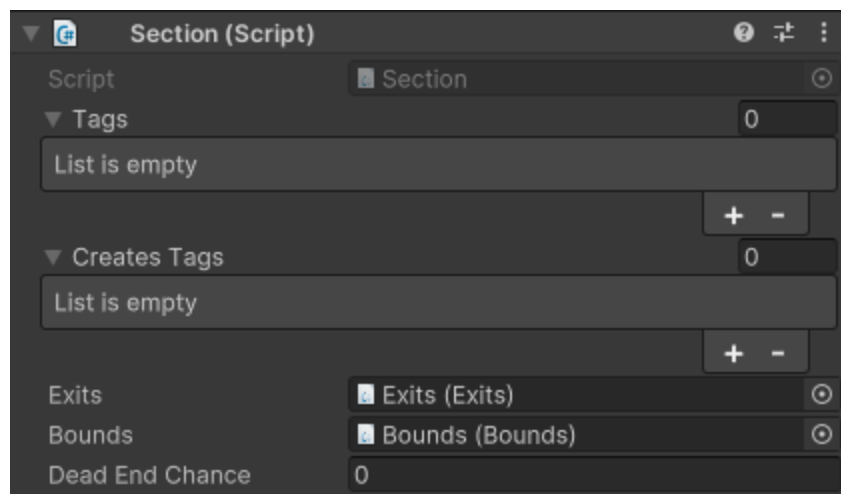
This will be our main component, and from here we will build the rest of the items, let's go through the list of fields we have here:

1. **Generate on Start (bool):** will define if the generate method is called on the Start event.
2. **Randomize Seed (bool):** will define if the seed is randomized on the generation event.
3. **Seed (int):** integer number to define the seed, use this number to generate specific levels.
4. **Section Container (transform):** This field is the referenced transform where all generated sections will be contained in the hierarchy, this field is optional, if left unassigned, the generator game object will be the parent transform
5. **Max Level Size (int):** The maximum amount of sections to generate.

6. **Max Allowed Order (int):** The maximum length (in sections) of branches from the original section. Setting a high number will result in longer paths while setting it shorter will result in a hive-like structure.
7. **Section Prefabs (list of sections):** Sections will be loaded as prefabs in this collection.
8. **End Prefabs (list of ends):** End prefabs will be loaded as prefabs in this collection.
9. **Initial Section Tags (list of strings):** The tags that will define which section is used as the start of the level, note that this does not mean the character player is forced to start here.
10. **Special Rules (list of rules):** Here we'll define special rules to force some aspects of the generation process. More on this below.
11. **Debug Mode (bool):** when active, it will dump all the generation data into the console. This data will have useful information to track the generation process step by step. This feature is **only available in the Pro version**.
12. **Add Section Template:** Will add a new section template to the scene, loaded with the basic structure for you to work on.
13. **Add End Template:** Will add a new end template to the scene, loaded with the basic structure for you to work on.
14. **Generate Now:** This button will call the generate method in-editor. This feature is **only available in the Pro version**.

## Sections

While the Level Generator component is the heart of our system, the sections are the veins. A section can be a corridor, a room, a cave chamber, a spawn section, even an entire open area. This system is designed to adapt to any type of level requirement. Once you click the **Add Section Template** button, a new object will be created in the scene, this is your section template.



Sections require special attention, as you will be spending most of the time creating these, let's go through the fields:

1. **Tags (list of strings)** : These are the tags the section will have, you may want to set one or multiple tags to a section, maybe a section can behave as a room and a corridor at the same time, this provides a lot of freedom when designing the generation constraints.
2. **Creates Tags (list of strings)** : These are the tags that the room will create on its exits, a common rule is to have rooms generating corridors and corridors generating rooms for example
3. **Exits (component Exits)** : This field is assigned by default, its a reference to the transform that will contain the room's exits
4. **Bounds (component Bounds)** : This field is assigned by default, its a reference to the transform that will contain the room's bounding boxes
5. **Dead End Chance (int)** : Chance in 100 to generate an end in each exit as opposed to a new section.

Once you have finished creating your section, save it as a prefab in your Project view.

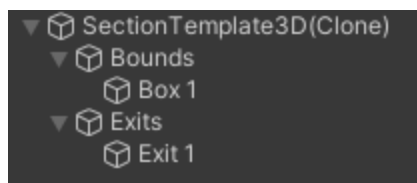
## Bounds

Bounds are simple colliders used to outline the shape of the section; this will come into play afterwards in the level generation as they prevent sections from overlapping each other. It is recommended you use a single box collider that covers the entire section, although you can use more colliders to outline a complex shape or you can choose to be more permissive with your overlapping to reduce the space between your sections. It is extremely recommended that you use the collider component bounds instead of the transform scale. Please refer to the tutorial video or example prefabs for more information.

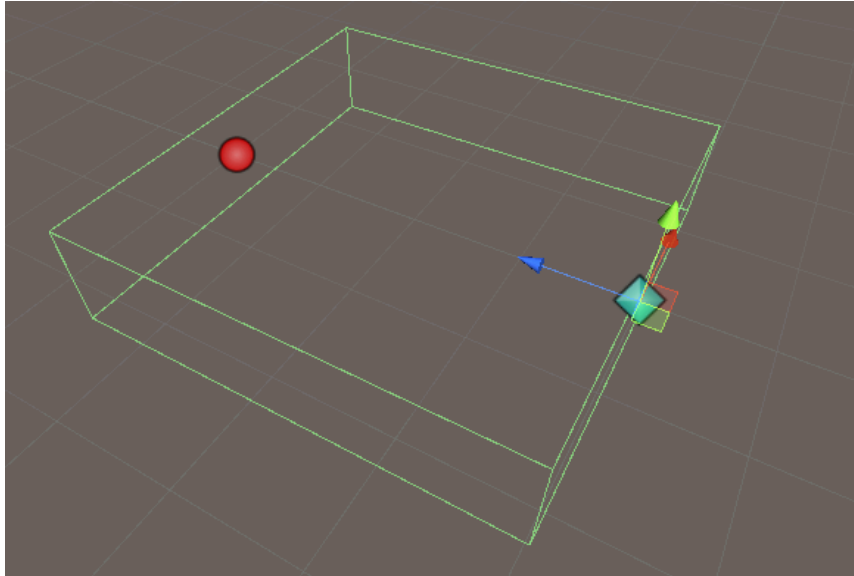
For 2D maps and sections you must use 2D Colliders.

## Exits

Exits will set the position and rotation of a new section, keep in mind that a section can have many exits but only one entrance, the entrance being the (0,0,0) of the section, and the entrance will be placed in the same position as the predecessor exit.



If you want to add more exits, you can clone Exit 1 game object and move it accordingly. Keep in mind that the rotation will also affect how the sections are placed, always keep the exits facing outwards (blue vector indicator).



The teal diamond represents the (0,0,0) of the section while the red circle represents an exit. Please refer to the tutorial videos or example prefabs for more information.

## Advanced Exits

Advanced exits are components used to provide a tag override when generating next sections. For example, a room can have “corridor” and “trap” as its **Creates Tags** setting, but if the section has an Advanced Exit with “treasure” as a tag, it will attempt to create a section with “treasure” as tag on that exit.

To add an Advanced Exit simply create a new exit and add the component **AdvancedExit** to it, then set the tags as you would with a section.

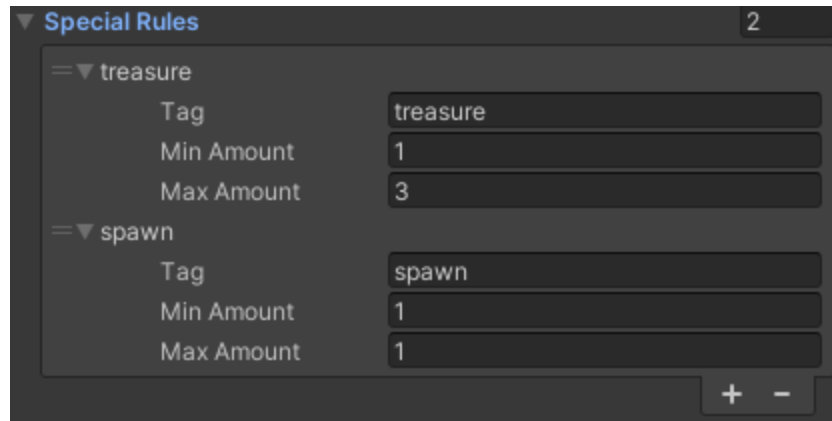
## Ends

Ends are important because you want to prevent the player from getting out of the map, Ends close off exits that are not being used to generate a new section. Once you click the Add End Template button, a new game object will be created with the template to create a new End, this follows exactly the same logic as with sections but ends have no exits. Think of ends as walls that cover holes which could have been exits.

Once you have finished creating your End, save it as a prefab in your Project view.

## Special Rules

The last feature of this tool is the special rules system, which allows you to control certain constraints on the generation process. A rule is defined by a tag and two numbers.



The screenshot shows a user interface titled "Special Rules" with a count of 2 rules. It contains two rule entries, each with a tag and two numerical values for minimum and maximum amounts.

Tag	Min Amount	Max Amount
treasure	1	3
spawn	1	1

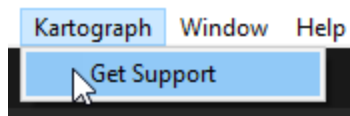
At the bottom right of the interface are plus (+) and minus (-) buttons to add or remove rules.

Simple enough, this set of data will tell the system how many sections of a certain tag must be created and how many can be created. In the given example, the rules define that one spawn room **MUST** exist in the level, and up to one treasure room, but there can be none. You can set up as many rules as you like.

# Support

If you have any issues with Kartograph you can email me at [juan.rod707@gmail.com](mailto:juan.rod707@gmail.com) with your concern.

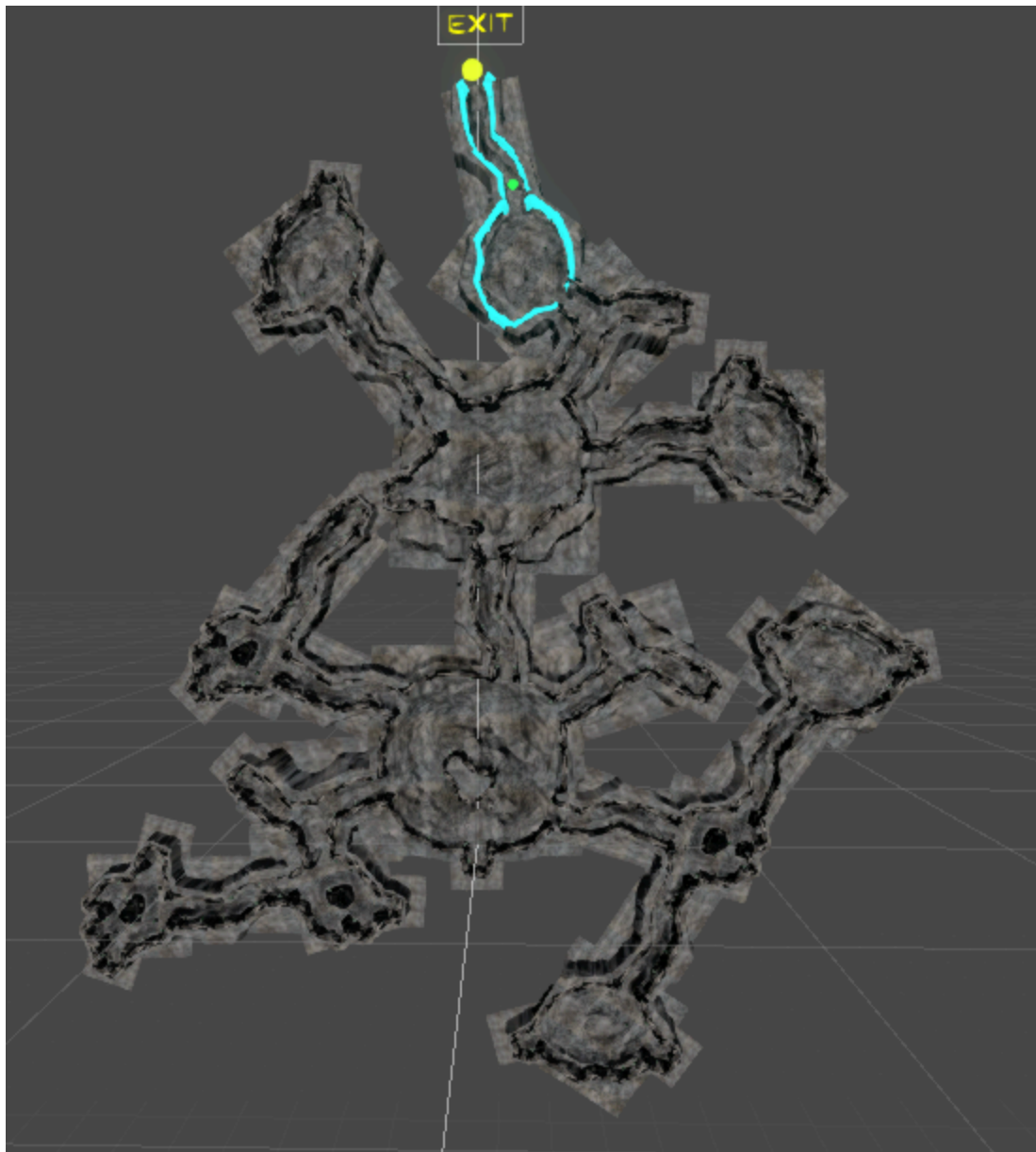
As part of the **Pro version** you have access to an exclusive support key that will give you support priority. To access this key click on the **Kartograph -> Get Support** context menu.



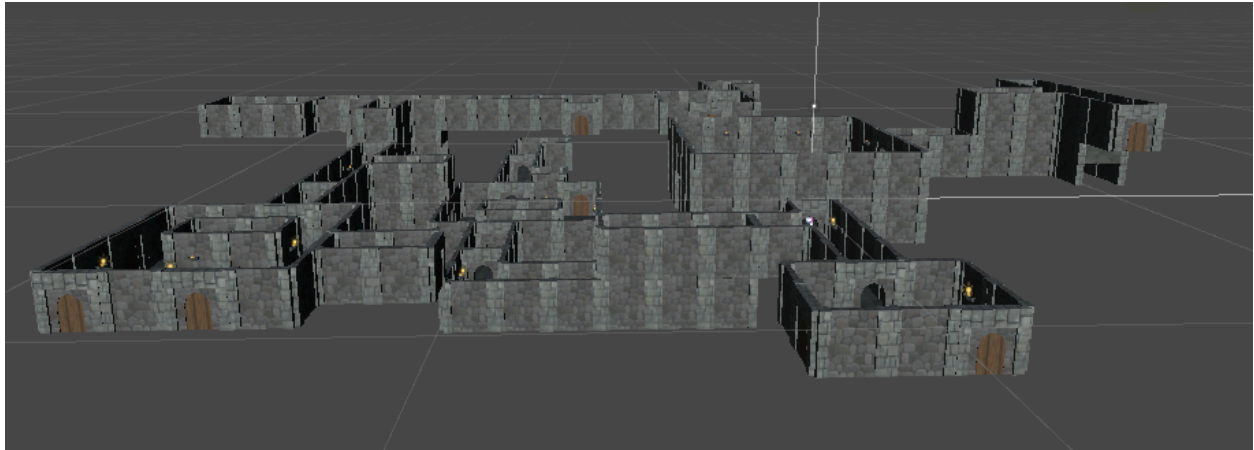
A pop up window will appear with instructions to get support.



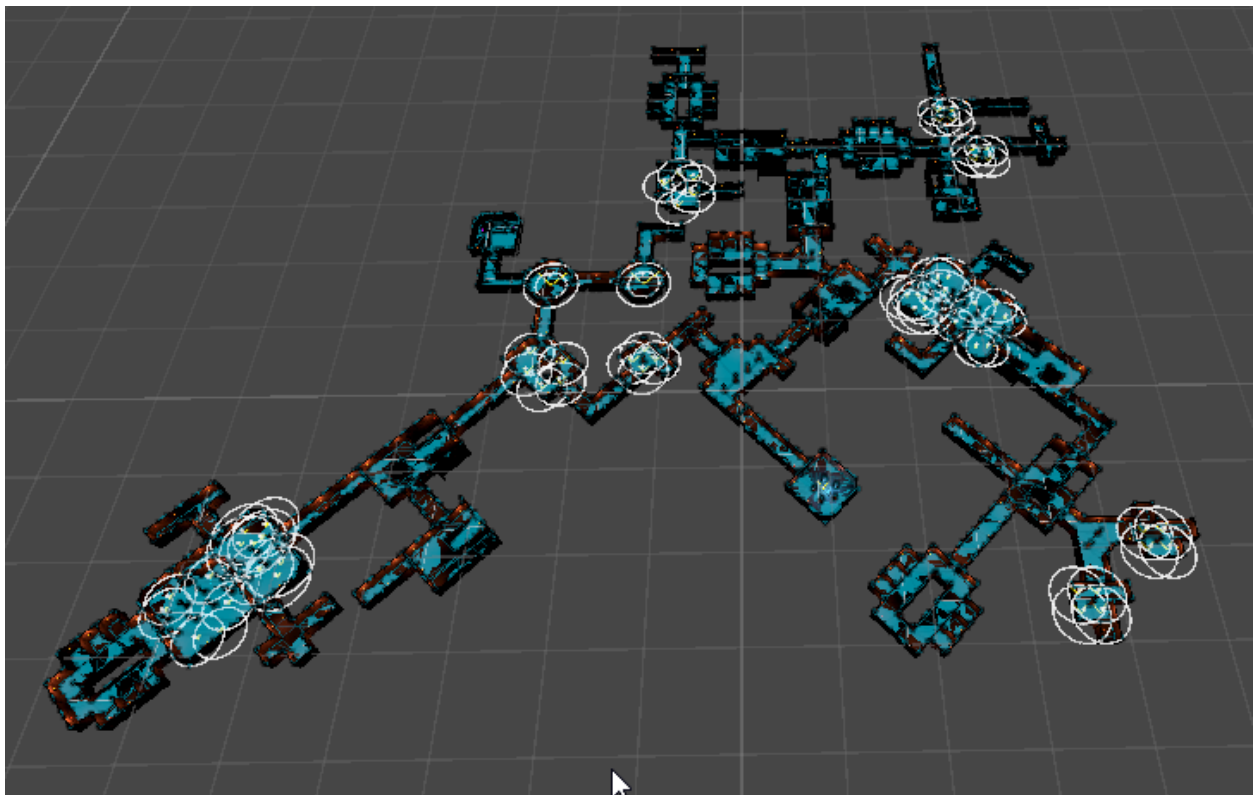
## Examples



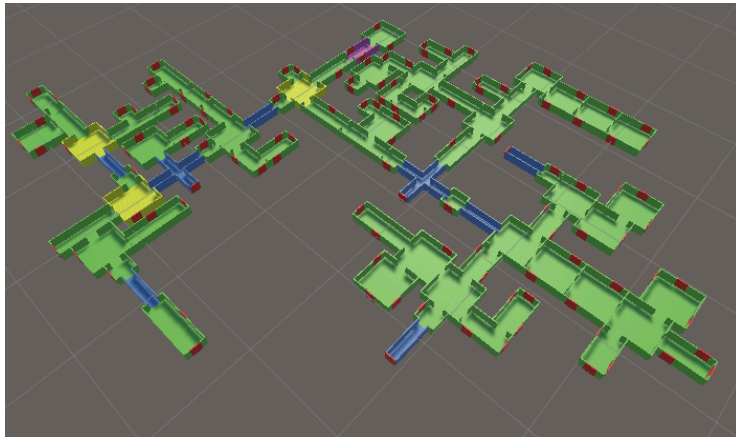
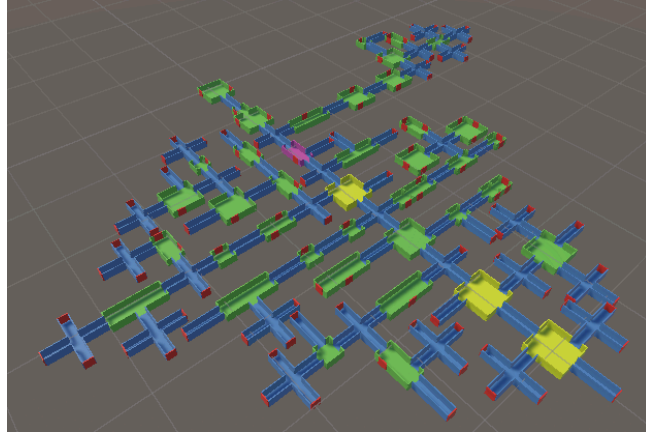
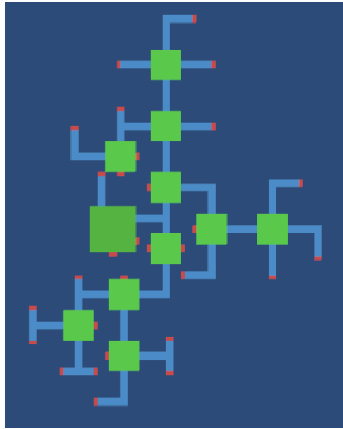
Cavern system for a side view exploration game, Kartograph does not limit you to 90/45 degree maps.



Dungeon with multiple vertical levels.



Supports any kind of navigation, including dynamic NavMesh.



Screenshots of levels created with the demo scenes and assets.

