# Procedural Level Generator

By Juan Rodriguez. 2019

## Index

## Introduction

Procedural level generator is an asset for Unity 3D 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 top down or platformer games
- Support for every shape, not confined to a grid or 90-degree layouts

It is highly recommended to also watch the video tutorial:
https://www.youtube.com/watch?v=9Dh0oLWIHPE

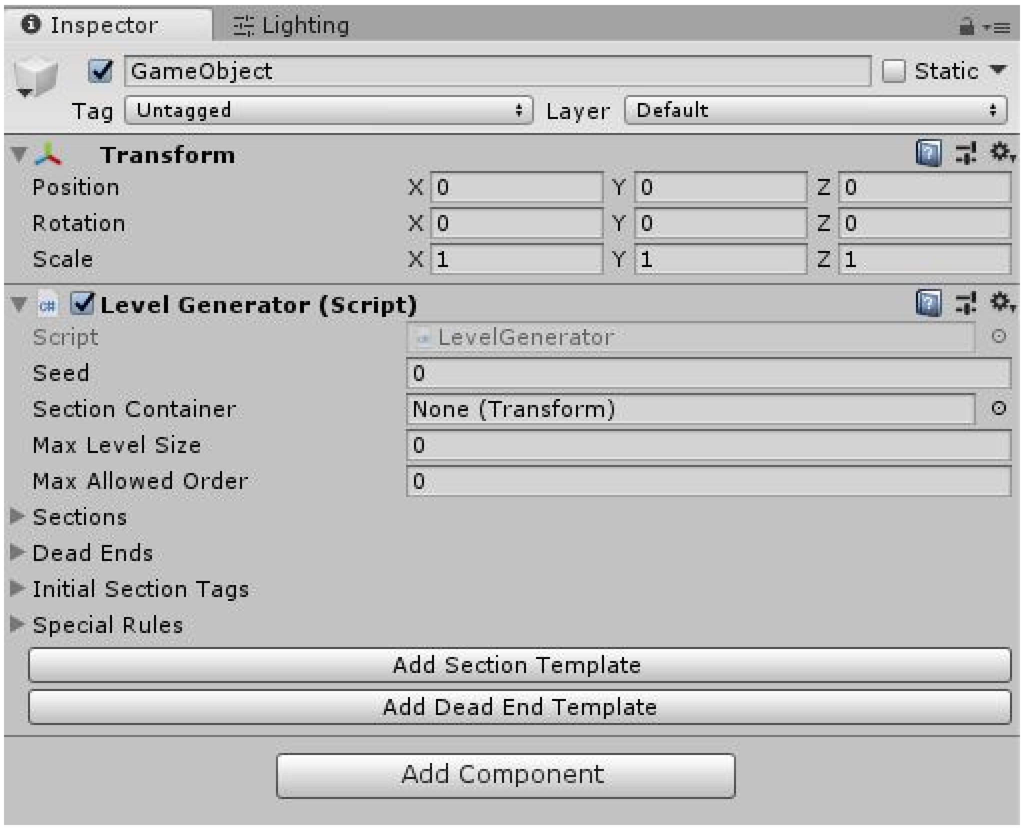
## Requirements

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 to it.



GameObject
Tag: Untagged   Layer: Default

Transform
Position X 0 Y 0 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1

Level Generator (Script)
Script: LevelGenerator
Seed: 0
Section Container: None (Transform)
Max Level Size: 0
Max Allowed Order: 0
Sections
Dead Ends
Initial Section Tags
Special Rules

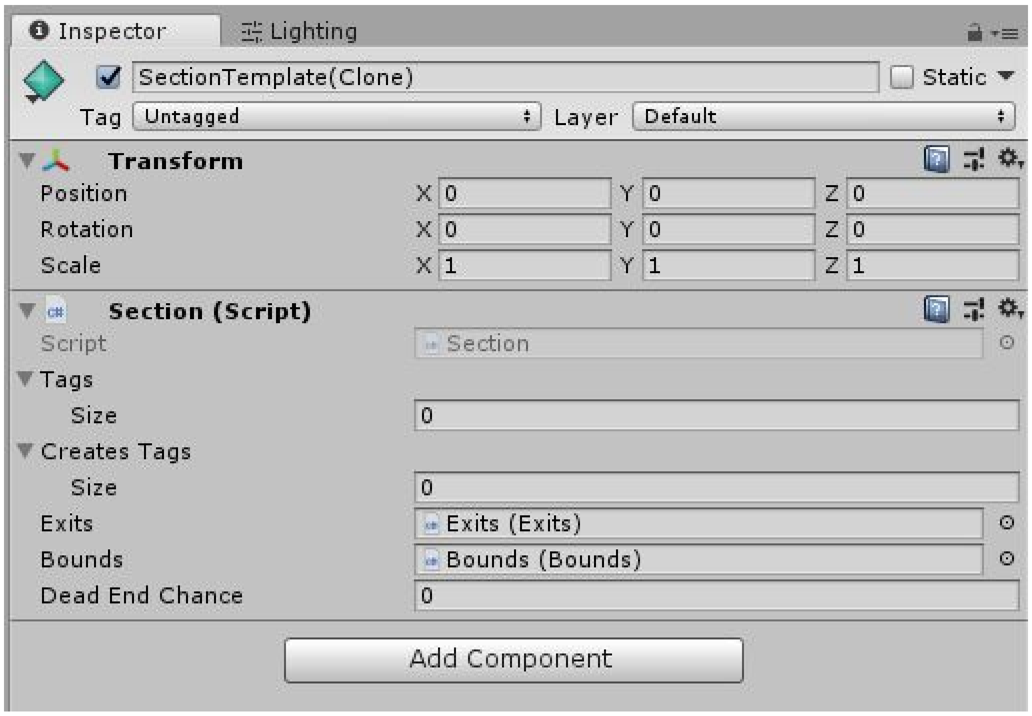
This will be our main component, and from here we will build the rest of the items, let's go through the list of fields we have here:

1. **Seed (int)** : integer number to define the seed, if left 0 the generator will randomly create a seed for the generation which will be displayed in this field
2. **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 will be the parent transform
3. **Max Level Size (int)** : The maximum amount of sections to generate
4. **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

5. **Sections (list of sections)** : Sections will be loaded as prefabs in this collection
6. **Dead Ends (list of dead ends)** : Dead end prefabs will be loaded as prefabs in this collection
7. **Initial Section Tags (list of strings)** : The tags that will define which section is used as the starting section of the level.
8. **Special Rules (list of rules)** : Here we'll define special rules to force some aspects of the generation process. More on this later.
9. **Add Section Template** : Will add a new section template to the scene, loaded with the basic structure for you to work on
10. **Add Dead End Template** : Will add a new dead end template to the scene, loaded with the basic structure for you to work on

## 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.



SectionTemplate(Clone)
Static
Tag Untagged
Layer Default

Transform
Position X 0 Y 0 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1

Section (Script)
Script Section
Tags
Size 0
Creates Tags
Size 0
Exits Exits (Exits)
Bounds Bounds (Bounds)
Dead End Chance 0

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

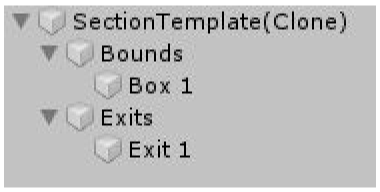
5. **Dead End Chance (int)** : Chance in 100 to generate a dead end in each exit as opposed to a new section

## Bounds

Bounds are simple colliders used to outline the shape of the section, this will come in 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. Please refer to the tutorial video or example prefabs for more information.

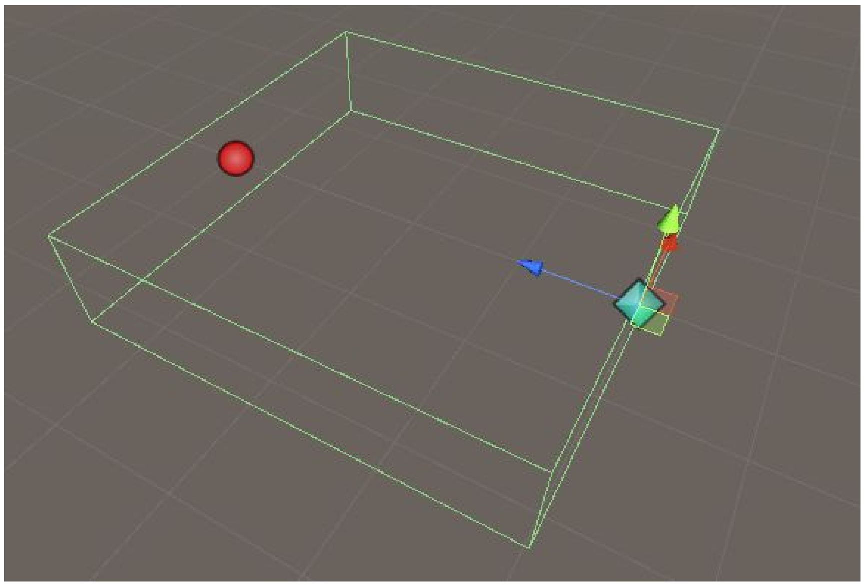
## Exits

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.



```
SectionTemplate(Clone)
  Bounds
    Box 1
  Exits
    Exit 1
```

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 video 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.

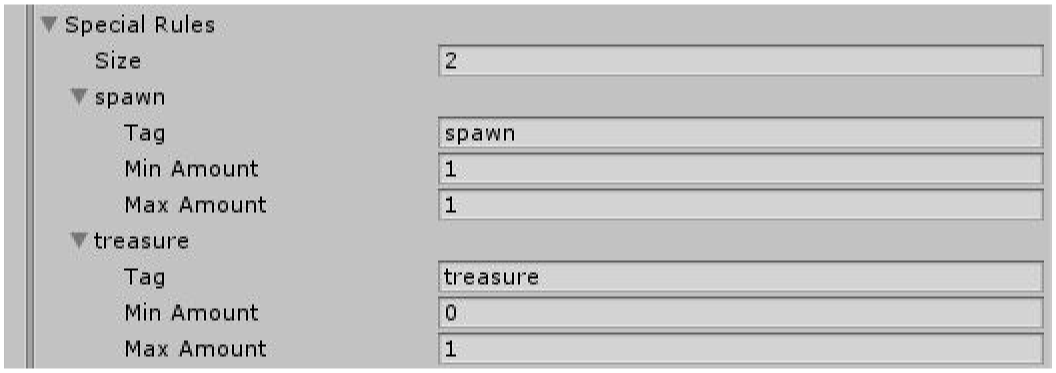
To add an Advanced Exit simply create a new exit and add the component **AdvancedExit** to it, then set the tags as you would with a section.

## Dead Ends

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

# Special Rules

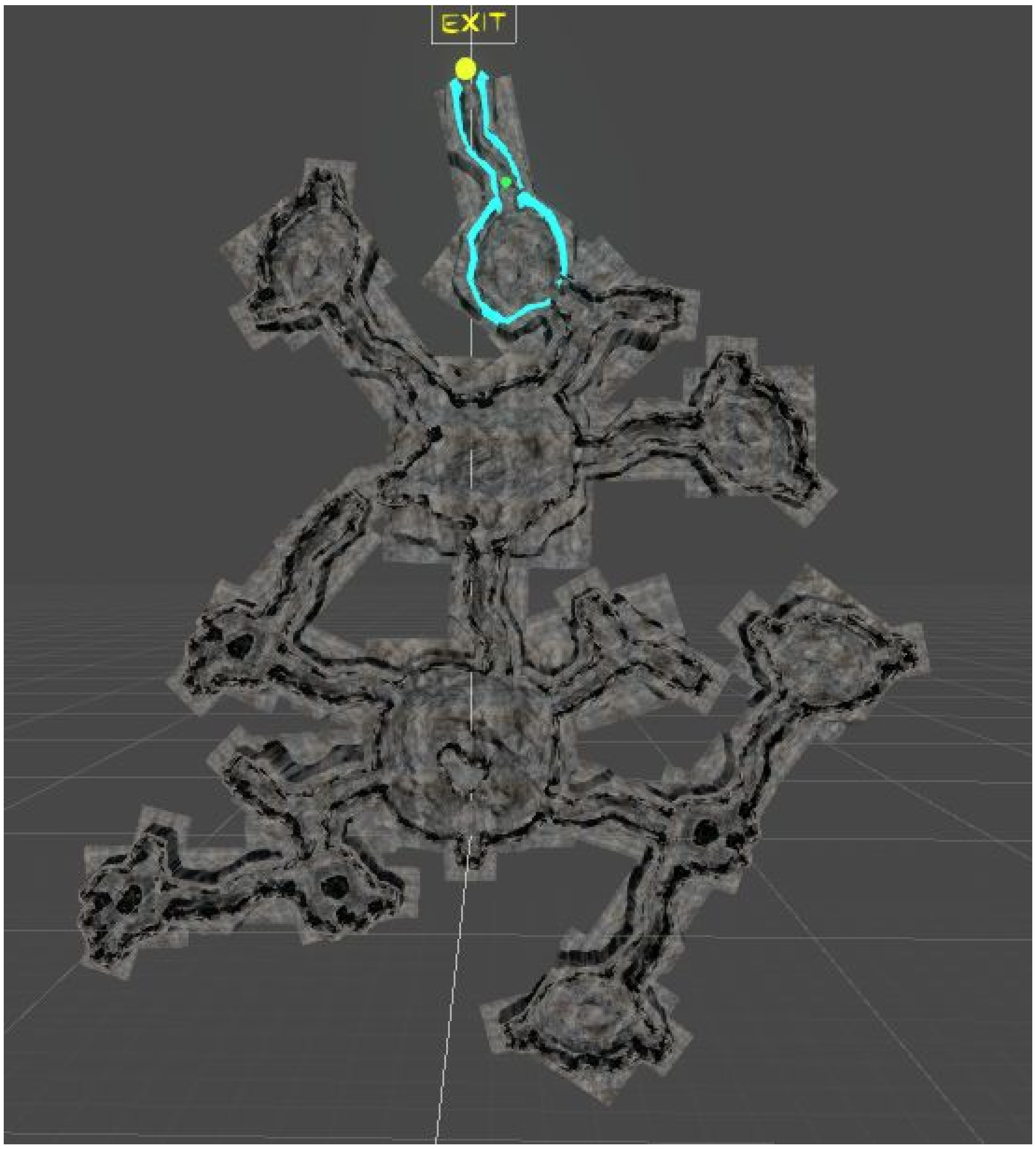
The last feature of this tool is the special rules system, which allows you to control certain constraints on the generation process. A rule is defined by a tag and two numbers.



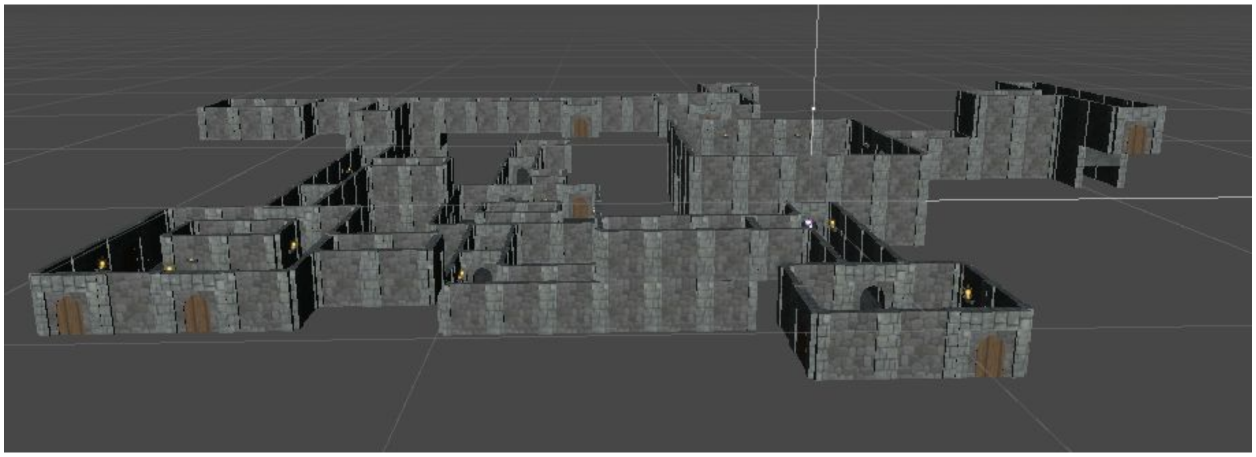
| | |
|---|---|
| Size | 2 |
| spawn | |
| Tag | spawn |
| Min Amount | 1 |
| Max Amount | 1 |
| treasure | |
| Tag | treasure |
| Min Amount | 0 |
| Max Amount | 1 |

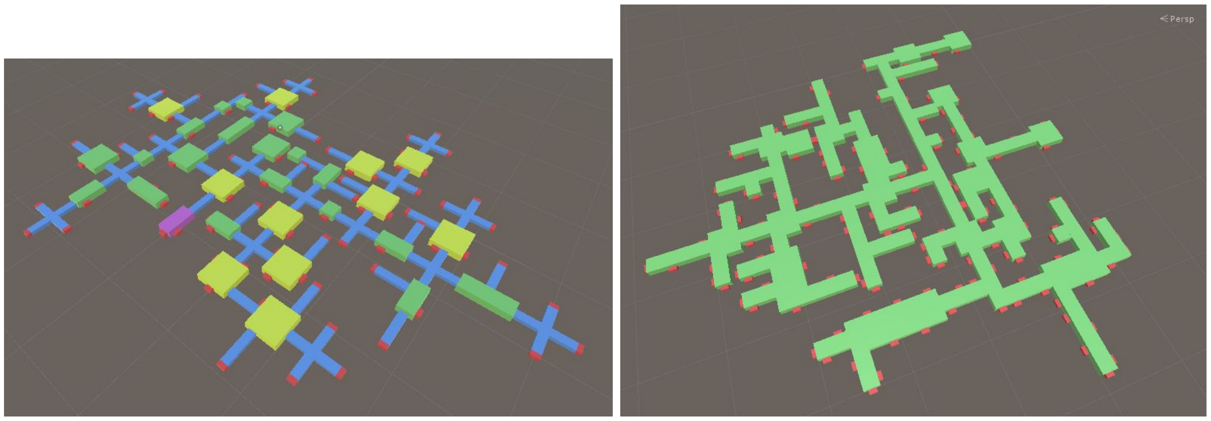
Simple enough, this set of data will tell the system how many sections of a certain tag must be created and how many can be created. In the given example, the rules define that one spawn room MUST exist in the level, and up to one treasure room, but there can be none. You can set up as many rules as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

## Examples



Cavern system for a side view exploration game


Dungeon with multiple levels



Both screenshots are from the demo scene provided in the asset, the difference was achieved by just setting the rooms to create other rooms instead of corridors.