

# *FIMPOSSIBLE* *CREATIONS*

## PROCEDURAL GENERATION GRID USER MANUAL

### About Procedural Generation Grid

---

- Procedural Generation Grid is a system which uses grids to define fields and paths for procedural generation algorithms. [I recommend checking the tutorials!](#)
- System is designed to be flexible and able to generate very various and complex objects/places. **Have full control on the procedural generation logic!**
- It can be used to generate procedurally whole game levels at runtime or for editor use to quickly paint levels for speed up level design process.
- System is using something like visual scripting, so you don't need to know how to code (but you can still take benefits out of it) you just need to plan logics for your needs.
- Procedural Generation Grid is right now **in Beta Stage** so, **many features may change, new features will come** and some small elements are not fully finished but the developmental stage is completed enough to be functional in many cases.
- Package is providing additional packages for randomly placing smaller objects in physical space using Unity's collision system (Object Stamper and Pipe Generator)
- Package is not dedicated to generating giant worlds (it still can work like that), it's focusing on smaller detailed areas like room interior with furniture, tunnels with many elements, presets for painting modular walls etc.

*Contact and other links you will find in Readme.txt file*

# Index

---

## 1: The Procedural Grid Generation System:

- Grid Generation Process (3)
  - Field Setups/Presets (4)
  - Field Designer Window (5-8)
- 

## 2: Designing Your Field Setup:

- Your first Field Setup (9-10)
  - Workflow hints (10)
  - Optimizing Generation Duration (11)
- 

## 3: Using many Field Setups:

- Build Plan (12)
  - Example Generating Components Descriptions (1)
- 

## 4: Additional:

- Objects Stamper (13)
  - Objects Multi Emitter (13)
  - Pipe Generator (13)
  - Checker Field Framework (13)
- 

## 5: Advanced Specification:

- Global Rules (14)
  - Field Setup Commands (14)
  - Field Setup Variables (14)
  - Injecting (14)
- 

## 6: Coding Custom Spawn Rules (15-16)

---

## 7: Build Planner and Node Graph (17-24)

- Build Planner (18)
- Field Planner (19-20)
- Node Graph (21-24)
- Build Planner Executor (24)
- Field Setup Compositions (25)

# 1: The Procedural Grid Generation System

## Grid Generation Process (Field Setups




Before working with Procedural Generation Grid you need to know what are the fundamentals of its logic.



System runs spawning rules for every cell of the grid provided by the user.

**The order of provided grid cells modifiers (  ) is very important!**

For example: when you want to align some furniture underneath the walls, first you need to spawn walls. Furniture requires information where and how are placed walls, this information will appear after calling wall spawning modifiers.

When you want to create more complex architecture then sometimes logic can get more complicated, keep that in mind.

Field Setup  is the main asset type of the whole package, it contains groups (packs ) of grid modifiers  which are spawning or removing objects generated on grid.

You can group modifiers, you can find it very useful. As example: you can make a group  of field modifiers  which spawns the whole interior room with props inside. You can do separated Groups of Modifiers, one with base interior spawn like: walls, floors, ceiling etc. other group with light lamps, or other cosmetics and another Modifiers Group with props like furniture, debris etc.

Then in Field Setup you can freely assign groups to be executed, for example group of walls as first, then run cosmetics group and in the end select your props modifiers group.

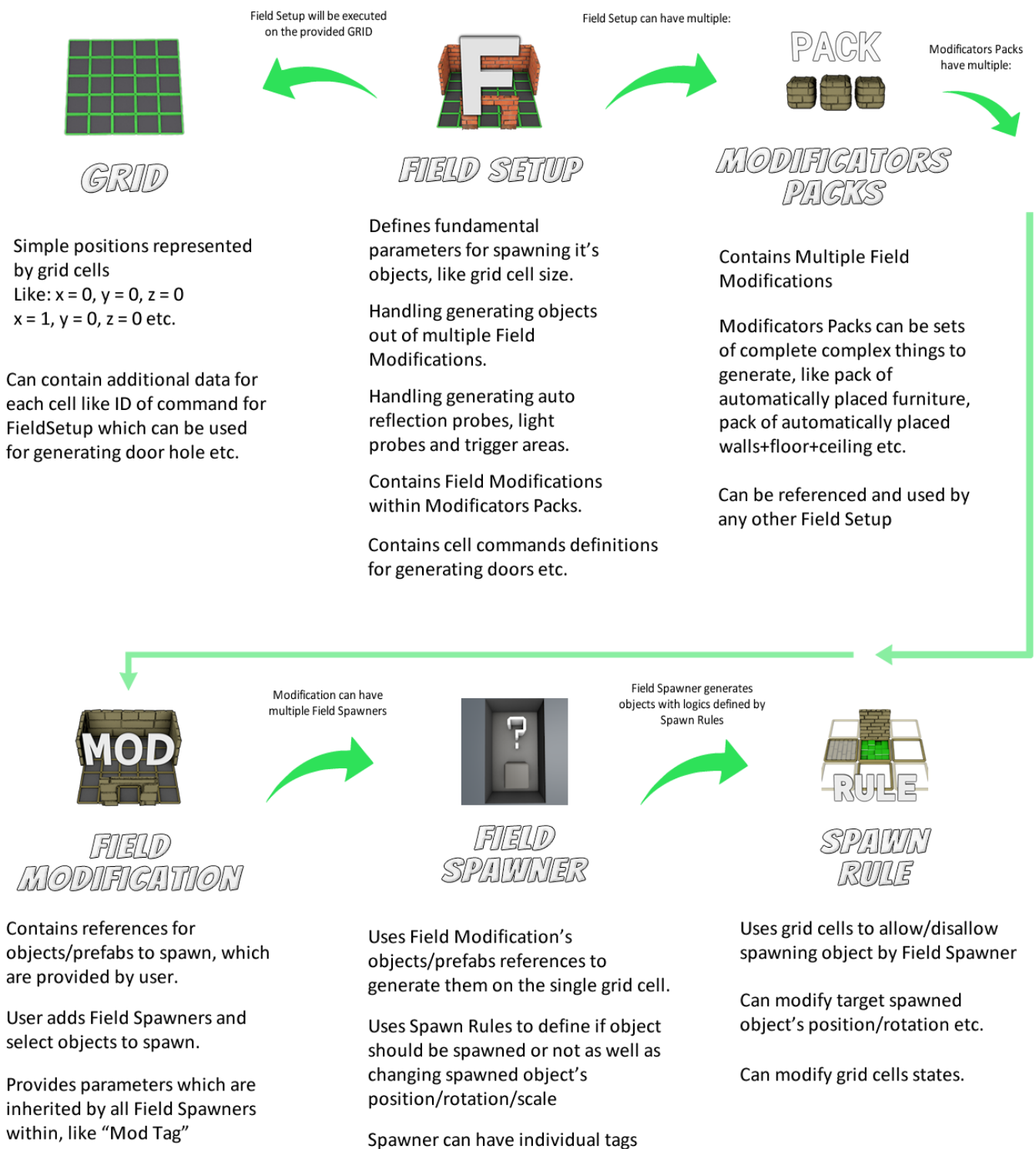
Field Modifiers can have variant files, variants can be created using the “V” button which you can find in Field Designer Window or Inspector window of Modifiers Pack. Variant means that Field Modifier is using all rules of parent Field Modifier but can change prefabs setted inside it, so you don't need to set up rules multiple times when you just need to change the look of some walls as an example.

**Procedural Generation Grid system is not generating game objects during cell checking/spawn rule execution process**, it spawns them at the end, all spawn data is contained within additional classes which are used after all Field Setup's rules are computed.

[You can watch the fundamentals tutorials here.](#)

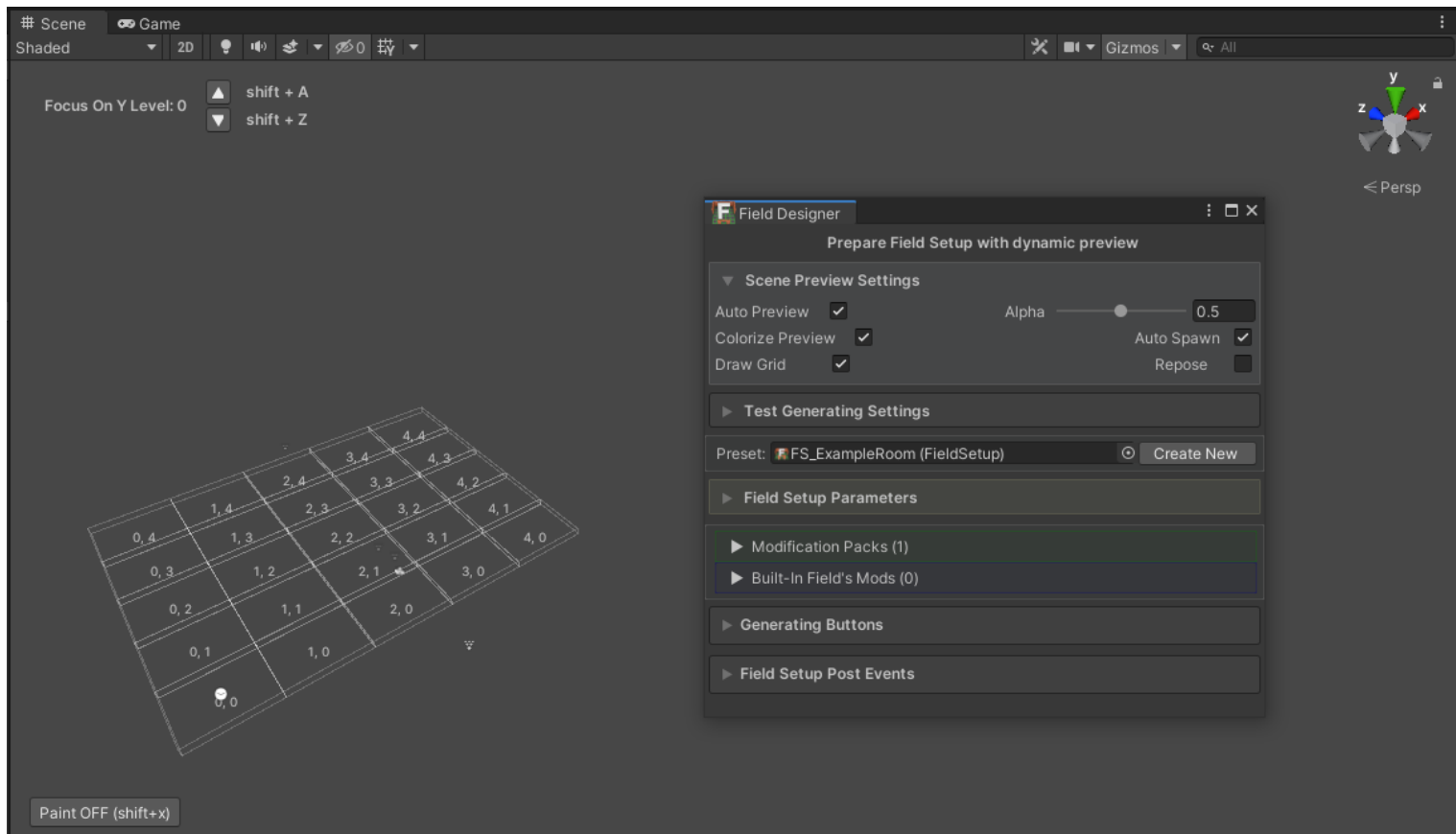
## Field Setups/Presets:

Package offers preset structure which can help you a lot.  
Take a quick look on relations between different instances:



In most cases you will use **Field Designer Window** which you can open through “Window->Impossible Creations->Level Design->Grid Field Designer”

**Field Setup** is the root of what is spawned on the provided grid.



**Field Designer Window is dedicated to focus on spawning logics**, not on creating gameplay shaped grid. Debug grid visible when the Field Designer Window is opened will disappear and be forgotten after closing the window. **You should edit Field Setup files only through Field Designer Window** (it opens automatically when double-click on Field Setup File)

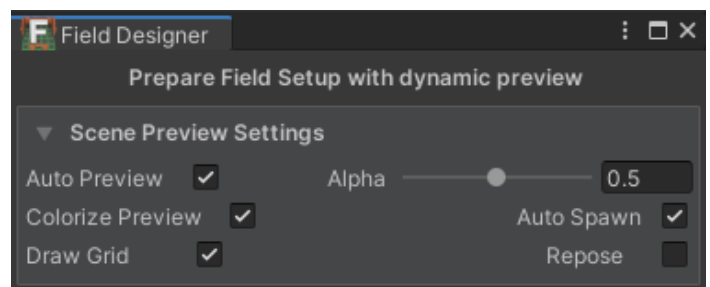
## Scene Preview Settings:

Configure how you want preview grid generation on gizmos or with spawning prefabs on the scene (slower).

(**Repose** is changing position of Field Designer after scripts compilation)

## Test Generating Settings:

Configure on what grids you want to test your spawning logics, you can simply generate random scaled rectangle grids, or paint on scene grid as you like or generate branched tunnels.

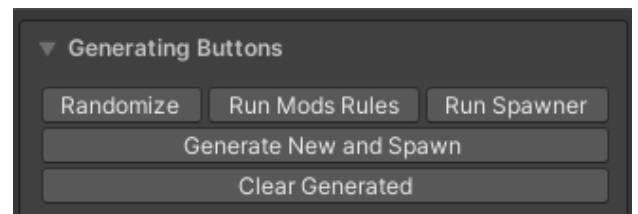


Grid will be generated each time you change any rule through the inspector window if **“Auto Spawn” is enabled** or you can trigger it with **“Generating Buttons”** lower in the Field Designer window.

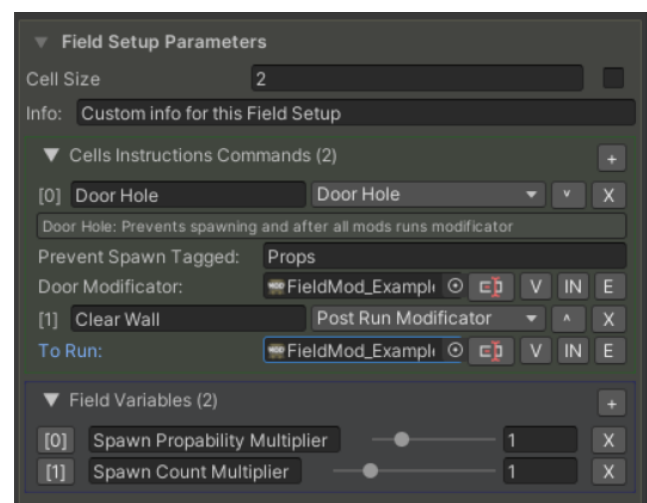


## Field Setup Parameters:

Configure field to objects you will spawn on grid, **it's best to fit CellSize to floor/wall tiles** (but you can make it smaller and spawn floor/wall tiles with grid scaleUp features - advanced)

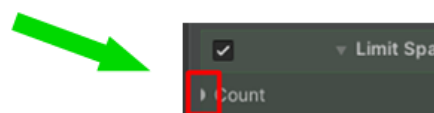


**Cell Instructions Commands** can be used when the grid on which Field Setup is used provides additional data on which cells commands should be used. It's really useful for making doors / door holes etc. but can also be used for many other things.

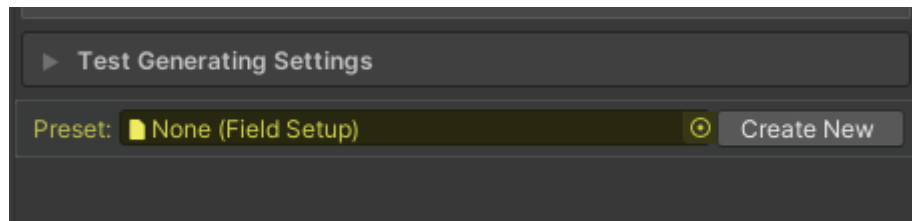


**Field Variables** can be used for controlling some rules parameters from FiledSetup level or through injecting (described later)

Some rules parameters have possibility to be used by Field Variable, you can identify them by a half white dot which must be pressed to assign variable:



To work with a field designer you must create a FieldSetup file inside your project directory, the best place them in a new directory since you probably will create a few of them later.

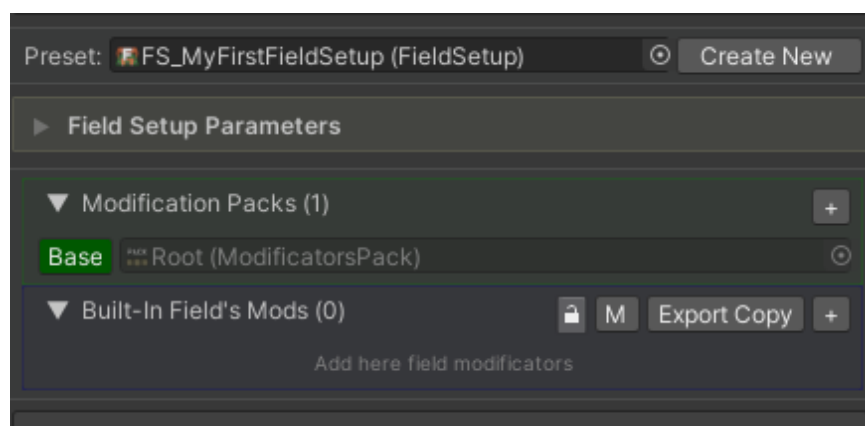


After creating the **FieldSetup** preset file you will see the “**Modification Packs**” list. (FieldSetup is created with a root (built-in) modifiers pack but can use multiple other packs in a shared way)

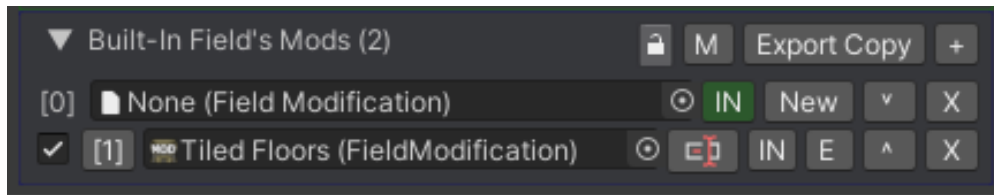
Inside the box below there are displayed **Field Modifiers** which are inside currently selected **Modifiers Pack**.

After creating the preset there is just a root modification pack and no **Field Modifiers**.

To know what are the buttons on the right, just enter on them inside the Unity Editor and a tooltip will appear.



Now you can start adding Field Modifiers by pressing the “+” button.



You can keep Field Modifiers inside Modifiers Pack (in root pack then also inside FieldSetup without creating additional file inside project directory) In most cases you will use “**IN**” button to create a new **FieldModifier** and put it automatically inside **ModifiersPack**.

**With toggle on the left** you will quickly disable / enable Modifier, very helpful when designing rules.

**With “^” and “v” buttons you will change order of modifiers which is very important**, since each modifier have only access to field state on modifiers executed before it.

**To know what other buttons are doing, just enter on them inside Unity Editor and tooltip will appear.**

**When you have modifiers, just hit buttons on the left “[0]”, “[1]” etc.** to display Field Modification inside the inspector window.

Clicking on these buttons with the right mouse button will display additional options for duplicate modifier, export copy or variant.



## 2: Designing Your Field Setup

### Your first field setup:

Open Field Designer Window and create a Field Setup preset using the “New” button. Unfold “Built-in Field’s Mods” to display Field’s modifiers list.

Hit “+” button to add a new field in list, hit “IN” to generate a new Modifier inside Field Setup’s asset file.

Hit the “0” button or double click on the Modifier field to display it’s options.

### Field Spawners:

Inside the Field Modifier window you can assign prefabs for spawning.

Hit the “Add Spawner” button and select the prefab you want to generate.

Without any rule setted, the spawner will generate selected prefabs on every cell of the provided grid.

Now you can add rules to define how your objects should be generated.

Field spawners can have assigned tags for rules to detect wanted objects on cells, you can assign multiple tags by using “,” commas like

Prop,Lamp,OnCeiling. Tag assigned to FieldModifier will be applied to all spawners within.

[I recommend checking the second part of the main tutorial playlist.](#)

### Field Rules:

When you hit “Add Rule” button you will see different categories to choose from. You will find rules for moving/rotating spawned objects, for allowing to generate in certain conditions or more complex rules which are doing many things at once.

Each rule contains description which can be displayed when you **enter on it’s header inside inspector window** (after adding it to spawner)

**You can reverse/negate rule logics**, which is pretty important, sometimes you might want to get opposite effect from some rule, then just click ‘#’ button to negate rule logic. This feature is very useful and it’s worth remembering.

**You can do the same with tags**, adding “!” before tag name will make rules affect every tag excluding ones with “!”.

Example: “!Floor,!Ceiling” so all spawns without Floor and Ceiling tag will be gathered by rule algorithm.

## Short description for each Rules Category:

Transforming: Rules which change only position, rotation or scale.

Quick Solutions: More complex rules with algorithms which replaces usage of multiple other rules to get desired effect with just one rule.

Placement: Checking conditions related to cells placement on grid.

Other: Just helper decorators, not changing anything in grid.

Modelling: Rules related to changing generated objects appearance etc.

Field And Grid: Rules related to field variables, other fields, grid size etc.

Count: Rules which are limiting count of spawned objects.

Collision: Rules related with collision check (procedural generation grid is using mostly bounding box collision, no raycasting)

Cells: Rules which are modifying other cells, like removing spawns, preventing generating for selected ones etc.

## Setting up props:

For a more practical presentation on how to set up props on your grid [check the tutorial videos on Youtube](#).

**Here are some tips**: when you want to set some **debris** on the floor, use just randomized positioning with Transforming rules, separate spawns with Placement->Cell Distance rules using debris spawner name and in addition use Count->Spawning Probability to avoid too many objects. Also don't forget to set "Cell Check Mode" of spawner to random for additional randomization.

If you want to generate **wardrobes or other furniture**, use Full Transforming->Get Coordinates and gather coordinates from walls, adjust positioning and rotation plus add similar count limits / distance separation like with debris.

If you want to generate **groups of objects like a pile of card boxes**, prepare a "Multi Spawn Emitter" component and create a prefab out of it then spawn it in a similar way like wardrobes.

If you want to generate **multiple objects in one cell, like narrow shelves**, try using Quick Solutions->Sub Spawner to generate additional objects in the same cell but with other customized rules.

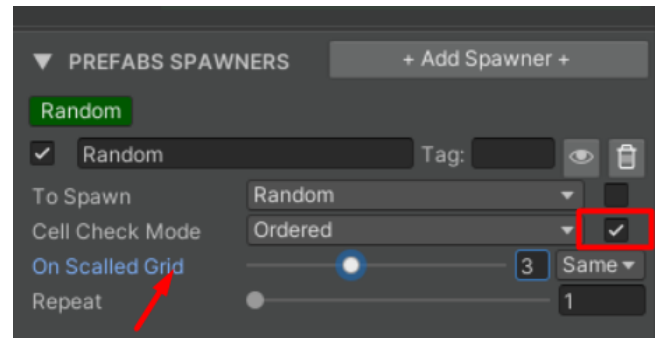
If your model's origin/pivot is not starting in center/middle but in corners, you can adjust it with some special nodes, [check this tutorial for more info](#).

## Complex Cases:

If you want to **create curved walls with your levels or for example floors with leaks on texture aligned to walls**, use the power of Quick Solutions->Wall Placer and Floor Placer components.

If you want to replace **walls with damaged versions every few wall segments**, you can use Full Transforming->Get Coordinates to align with other wall, then Cells->Remove in position to remove previous wall and replace it with damaged version, then use similar conditions like debris in previous manual section, to spawn them every few segments.

In case you want to **create a giant underground tunnel and cover small objects with it**, you can try using a scaled grids feature.



You can switch scaled grid for spawner with the toggle next to “Cell Check Mode”  
Use Grid Painting to see how it works.

### Common Logic Rules Nodes:

Most commonly used nodes are:

Full Transforming -> Get Coordinates

Quick Solutions -> Wall Placer, Floor Placer

Placement -> Distance to other, Check Cell Neighbours, If Cell Contains Tag

Count -> Spawning Probability

Cells -> Remove in Position

### Optimizing:

If your generating process takes too long, you can optimize it a bit with approach like follows:

Using “Cell Check Mode” called “Total Random” is not optimal, it’s shuffling all grid cells, when you use Modifier on a large grid it’s better to use other cell check modes.

It's best to set simplest logics as first in spawner, if the first simple check is not allowed to spawn then the rest logics are ignored and will not cause CPU overhead.

Simple logic nodes have “Lightweight” suffix in tooltip when you enter node's header title.

# 3: Using Many Field Setups

## **Build Plan (Legacy - Check BUILD PLANNER for new approach):**

Build Plan can be filled with customized settings for multiple Field Setups.

It can be used by other components to generate multiple grids.

You can prepare Field Setups for generating whole rooms like guest room, bathroom, kitchen etc. and use other components to generate them in the wanted setup.

Check example scenes of generating dungeon game, which is using build plan to generate key room, boss room etc. restrict their position to be near exit doors etc.

[Also check this tutorial video for some more explanations.](#)

## **Example Generating Components (Legacy):**

**Building Plan Generator** (randomized generating): It's an old component which was made at the alpha stage of this package. It's not using the newest "CheckerField" framework but can be found useful for some users. It's trying to generate corridors and rooms out of the provided room inside the Build Plan preset.

**Facility Generator** (randomized generating): It's similar to BuildinPlanGenerator but uses a "CheckerField" framework which supports rotating non-rectangular room shapes to fit them to generated corridors and rooms.

**Grid Painter** (manual level design): It's component which allows to paint cells on scene, paint command guides very useful to set up doors between other Grid Painters.

**MiniCityGenerator** (randomized generating): Simple component to generate roads and buildings around them.

**RectangleOfFieldsGenerator** (controlled generating): Generates multiple FieldSetups within a custom rectangle shape, useful when generating house buildings with rooms inside them.

**SimpleDungeonGenerator** (randomized generating): Very similar to Building Plan Generator and Facility Generator but with some different rules.

**SimpleFieldGenerator** (randomized generating): Simple code example of generating objects with provided FieldSetup preset.

## 4: Additional



### **Object Stamper:**

Object Stamper is a component which can spawn objects on ground using raycasting with randomization. There can be random prefabs and each can have its own probability level to be chosen to spawn.

Add the ObjectsStampEmitter component and create a Stamper Preset using the “New” button, open the preset and add prefabs through the inspector window.

Adjust random position/rotation/scale settings and raycasting settings.

Check restrictions and overlapping configuration to spawn objects according to your needs. Now you can hit the “Randomize Preview” button to see ghost gizmos of target object to spawn, you can create object with this component as prefab and use with PGG or place around the scene to spawn randomized objects with raycasted placement.

### **Object Multi Emitter:**

It's using an Object Stamper preset or multiple presets to spawn multiple objects in configured areas. It is really useful when you want to generate piles of objects or objects on wardrobes, tables.

It can make generated spaces much more fulfilled.

### **Pipe Generator:**

Component which uses modules of any shape to generate path towards target or random position. Modules must be prepared, meaning must have defined points to which other modules can join with target out rotation at target point.

It can make generated spaces much more interesting.

### **Checker Field Framework (for programmers):**

Checker Field classes are providing workflow for generating grids.

This framework is easy generating custom build plans by auto cell-path generating, quick cell collision checking, snapping cells or groups of cells from one to another.

Check example generator components commented code lines for more.

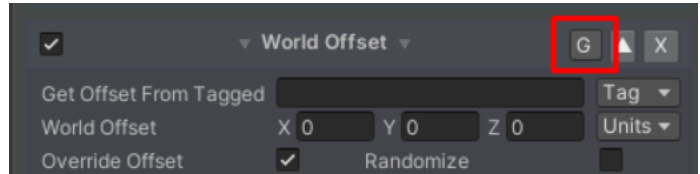
## 5: Advanced Specification:

### Global Rules:

Some rules have option to be setted as global ("G" button)

When enabled, all spawners in Field Modifier will have this rule

applied. Also when execution order matters, global rules setted lower will be queued to be executed after all other rules in spawner, global rules on the top will be queued to be executed before all other rules.

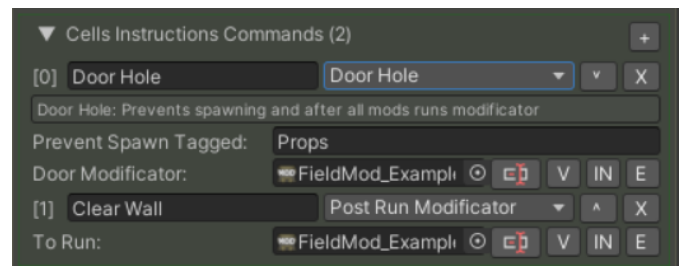


### Field Setup Commands:

Field setup can have assigned

Instruction Commands definitions which can be used by Grid Painter and custom coded grid generators to create door holes in walls by replacing wall model

with doorhole wall model. You can also use commands to just spawn additional objects on the grid like painting fences etc. This Cell Instructions opens many new possibilities for designing your Field Setup.



### Field Setup Variables:

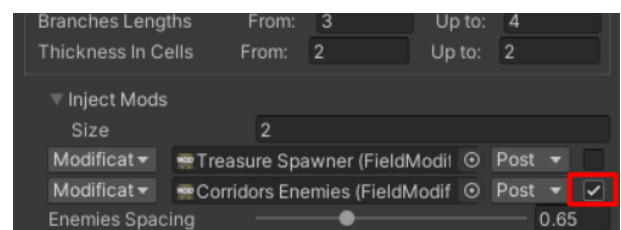
Field Variables can be used by spawn rules to change their parameters.

In most cases you would like to use field variables when injecting modifiers to other field setups.



### Injecting:

Injecting means running Field Modifier / Group of them as addition to Field Setup without adding them to the Field Setup preset file.



For example: you have Field Modifier which spawns enemies, you want to inject enemies spawner to some rooms, but in room A you want to have a lot of enemies and in room B just a few of them. You can use the Distance Check rule for setting spacing between enemies and assign to it Field Variable (assigning? -> bottom of page 6) then override variables with the injecting field and change the field's variable value up or down. (To assign variable modifier must be child of Field Setup with needed variables)

## 6: Coding Custom Spawn Rules

If you want to code custom rule to be used in your Field Modifier Spawner, go to:  
**Right mouse button somewhere inside project directory -> Create -> Scroll and find Impossible Creations -> Procedural Generation -> Scripting Templates -> Choose**  
Template, name it and open automatically creates .cs file.

(If you use Assembly Definitions this file must be created in PGG's directory or directory with Assembly Definition referencing to PGG assembly)

Right away you can add your rule inside spawners under category "Custom"

You can change it by setting a different namespace for the class file.

Each spawn rule must have at least one visible public variable to avoid GUI errors!

Now depending on what your rule will do, you will have access to the current state of the grid, to the spawn data grid is attempting to generate, Field Setup file when you override right methods.

You should choose correct **EProcedureType** for your rule:

**Procedure** - Conditions + changing spawn properties

**Rule** - Conditions for spawning (override just CheckRuleOn() )

**Event** - Just changing spawns properties (override just CellInfluence() )

**OnConditionsMet** - Will be executed at the end only when all conditions are met

**Coded** - Running CheckRule and OnConditionsMet to give wide variety access

And override right methods:

**CheckRuleOn()** Executed on every cell until conditions are met

Use variable CellAllow to make rule allow or disallow spawn

**CellInfluence()** Executed only once when occupied cell is known after running all rule checks




**OnConditionsMetAction()** Executed at the end of all rules if all conditions met

Check code of other rules to get acquainted with grid logic.

I suggest checking simpler ones at first, like SR\_PushPosition, SR\_SpawningPropability.

Optional: To support  
**accessing spawns of scaled  
grid's cells** you can use code  
like this:

```
cell.CollectSpawns(FieldSpawner.ESR_CellHierarchyAccess.)
```

 HigherAndSame  
 LowerAndSame  
 SameScale

but if you don't need scaled grid support you can use method like:

`cell.GetSpawnsJustInsideCell()`

Optional: To support **Field Setup variables**, you must add

`public SpawnerVariableHelper VariableHandlerName;` below your rule's variable, it will create this half-dot on the left of the property inside the inspector window.

Then you can use it in code like: `VariableHandlerName.GetValue(defaultValue);`

to make an effect in the rule algorithm, but you need to do one more thing to make all of that work.

You need to **override `GetVariables()`** method and return your variable handlers in it.

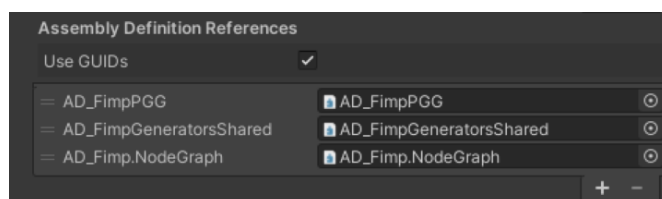
You can use quick method if you need to use just one field variable handler in your rule: `return VariableHandlerName.GetListedVariable();`

If you want to code your own rules inside a separated assembly definition, first you need to import the 'Fimpossible Assembly Definitions' pack and then use in your own assembly definition a few references.

If you need just to code custom field modify nodes, all you need is **AD\_FimpPGG**

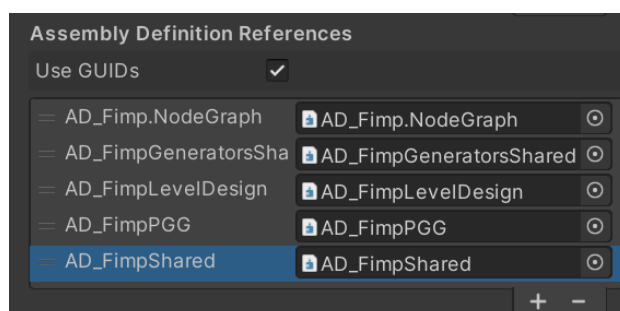
But if you want to use some of PGG helper utilities, add **AD\_FimpGeneratorsShared**

If you want to code graphs nodes, you need to add **AD\_Fimp.NodeGraph**



To access more utilities you can add **AD\_FimpShared** and **AD\_FimpLevelDesign**

So the max assembly definition list would look like this:






## 7: Build Planner (and Node Graph)

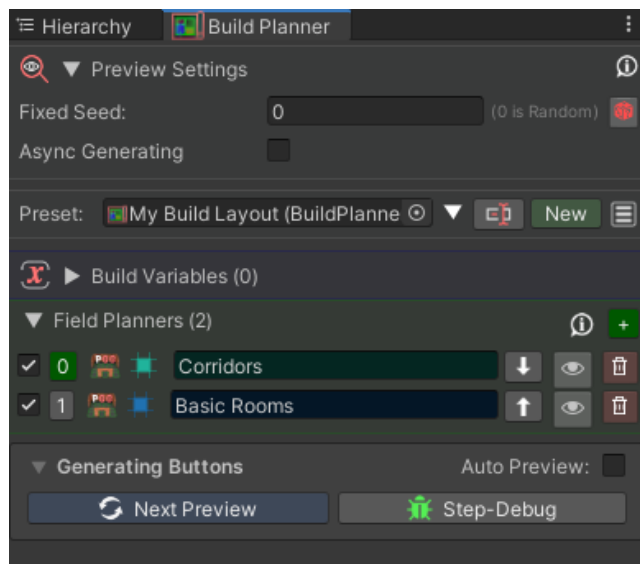
The Build Planner is in the early stage of development and it will be successfully updated with new features until the end of the beta stage of the procedural generation grid package.

Build Planner is defining areas to spawn objects and is providing guides for special spawning like doorway cells etc.


In the future versions, Build Planner node graph will be enough to replace all of the current generator scripts like “Facility Generator”, “Dungeon Generator” etc. which will become deprecated (but still available in the “PGG Legacy Coded Generators” unitypackage)



With a prepared **Build Planner Preset**  you need a game object on the scene with **Build Planner Executor** component which you use for gameplay / editor usage.

To start working with Build Planner, go to **Window -> Fimpossible Creations -> PGG Build Planner** (Build Layout)




With the build planner window you will quickly preview your work and prepare how many grids you want to generate, using **Field Planners** .

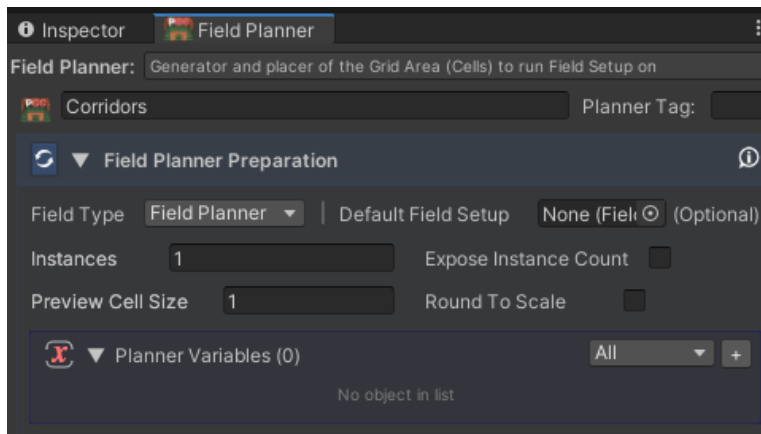
Prepare the main variables for every field planner's node graph .

You will also define order of the field planners (it works in the similar way as with field modifiers  in the field setups ).

The preview of build planner is visible on the scene center, it's good to use it on the empty scene in your project. If your scene view camera is far from preview position, the camera button will appear next to the “Preview Settings” bookmark.

When you select the Field Planner  the new window will open in which you will prepare a field / grid to generate and place on the build plan layout.

## First foldout of the Field Planner is “Field Planner Preparation”



There you define type of the field planner (right now only ‘Field Planner’ type is available, in the future there will be more)

You can select the default Field Setup which will be automatically chosen later, in the build planner executor component on the game scene.

You can define how many instances of the field planner should appear on the build plan. Instances are like duplicates / copies, it requires node graph logics prepared for use of multiple instances.

Expose instance count will make instance count editable later in the build planner executor component on the game scene.

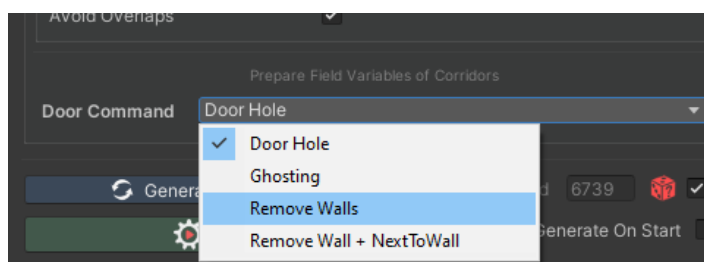
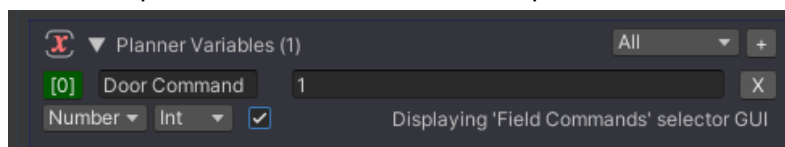
**Preview Cell Size** is the scale of a single cell of target grid area. Right now it’s recommended to use field setups of the same cell size but in the future support for different scaled grids will be improved.

**Round to scale** will automatically round position of the field to not be fractioned, when cell size is 2 then position  $x = 2.4$  will be rounded to 2.0. It can be helpful when designing some types of the build layout.

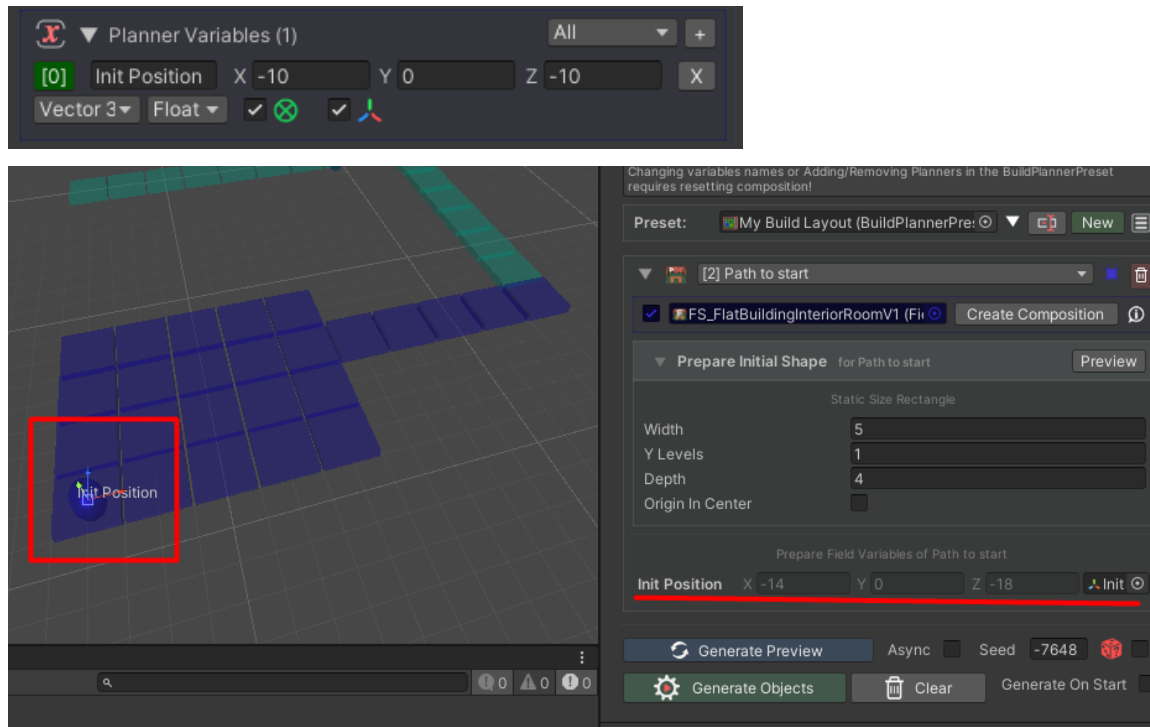
Finally you can define **variables** which will be used by the node graph and will be visible and editable later in the build planner executor component.

Useful tips for the variables:

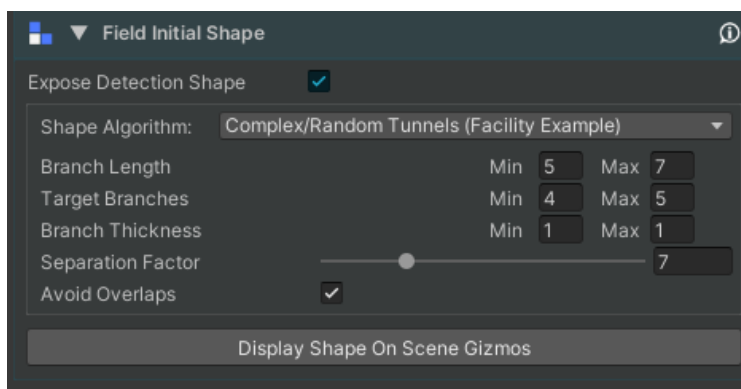
When you set type “number”, “int” there will appear a toggle which will make the variable appear later in the build planner executor as the field setup command selector.



When you use the Vector3 position you can enable handles toggles to display helpful gizmos on the scene, or choose some transform position to drive your node graph logic!



## Second Foldout is “Field Initial Shape”



There you define the starting shape for your field planner which is generated with the chosen algorithm. There few algorithms are provided but in the future versions there will be more and you will be able to code your own initial shape generator algorithm.



In some cases you will not need any shape because you can generate it with the node graph. As example: you will not need initial shape if you will use some field planner just to generate a path-finder connection between two field planners.


You can toggle “Expose Detection Shape” to make it editable later in the build planner executor component.

## And the last part is node graph section

There you can display a node graph and work with it, or open the same graph in the separated window if you will.

Every Field Planner owns two graphs. **“First Procedures”** and **“Post Procedures”**.

Since version 1.6.4 every Field Planner can contain multiple custom node graphs which can be displayed with  button and renamed/removed using  button.

First Procedures graphs are called as ordered in the Build Planner Preset 

It's dedicated to execute here main layout placement logics.

Post Procedures are called again as ordered in Build Planner Preset, but called after all graphs in First Procedures done their job. So Post Procedures are dedicated for creating final relations between field planners, when the whole layout is prepared and defined. When you know where each of the rooms will be placed, you can define relations like preventing spawning windows between rooms on opposite sides etc.

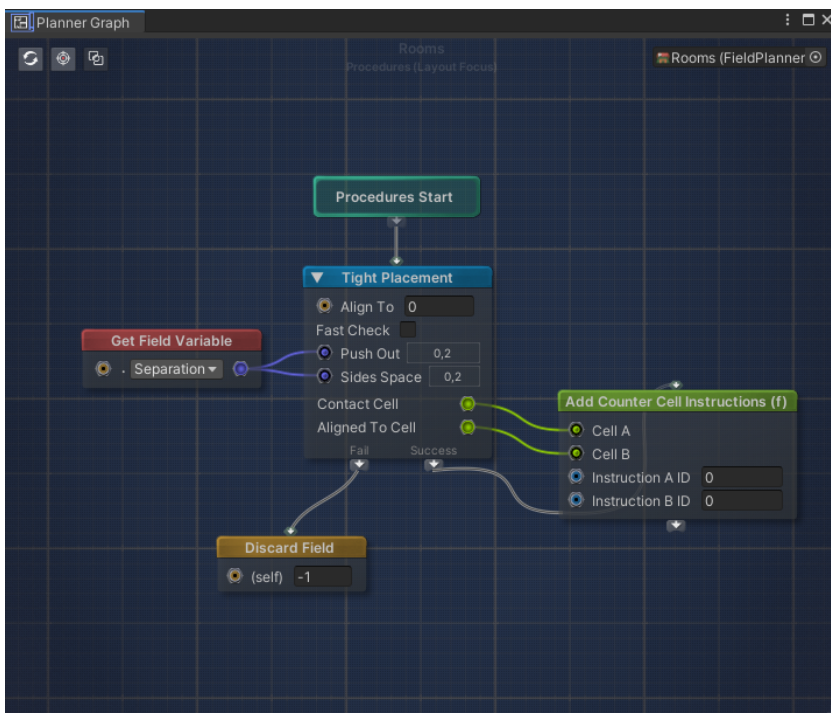
The GUI element next to “Post Procedures” is just a shortcut for switching between field planners stored in the build planner preset.

## The Field Planner Node Graph:

With a node graph you will define where grid areas will be placed.

You can generate additional shapes from one point to another and join/subtract cells of grids, prepare cells instructions to be added into grids, and design it to your project requirements.

In the future updates this documentation for node graphs will expand.



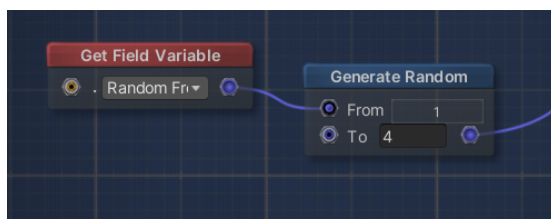
Node graph starts its logic with the “Procedures Start” node which needs to be connected with some other node.

When node is triggered, it runs its coded logic, reads values from connected ports and uses them.

When a node finishes its job the trigger signal moves forward to the next connected node until there is no connection to the next node in the graph.



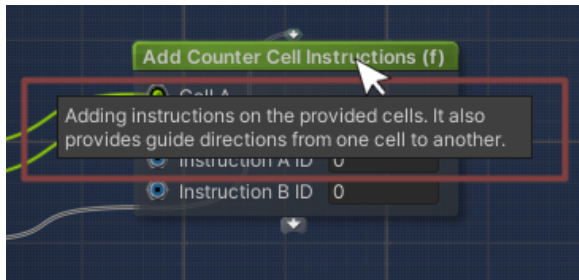
Some of the nodes contain arrow blocks on the top and on the bottom, that means this is “Execution Node” and it will call its main logic when some trigger connection reaches it.



Nodes with ports only will compute their logics when some of the execution nodes call them during the generating build layout preview process.

**When connecting some ports you can see different values in the node than it will be computed, when graph logics will reach the port (preview needs to be triggered) values will be refreshed.**

Graph is expanding its canvas in the right-down direction, meaning it will increase its size automatically when you put some nodes near to the low or right edge of view.



When the plugin will reach the ending stage of the BETA there will be provided description for each graph node. For now you must rely on the examples and provided tooltips on the node header / parameter fields.

Many of the nodes have the unfold option ▲ when you click on it, it will display more advanced options of the node.

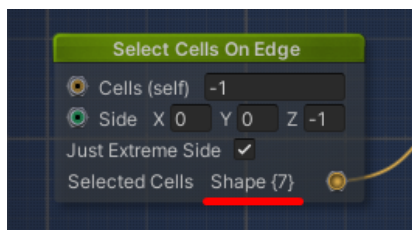
**The orange ports** are 'Planner Ports' which can receive different values, to convert them into Field Planners / Checker Fields.

Receiving a number value will convert it into a field planner with an index of the provided number. Receiving vector value will convert it into  $x = [\text{index}]$   $y = [\text{Duplicate Index}]$   $z = [\text{Sub-Field Index}]$ . It can also receive FieldPlanner references, CheckerField references or Cell references.

When outputting, the port can return FieldPlanner or CheckerField3D class.

It can be also returned as ICheckerReference interface type, which in the future may become just one type which the port will return.

If the port is returning CheckerField, a free grid without relation with any Planner, it will display "Shape" on the GUI with the grid cells count inside {} braces.



Shape can be treated as a separated list of cells.

If you want to do something with shape cells, you can iterate through them, but to read their internal datas like 'Cell Data' or 'Command Instructions' of some grid, you can't read it since shape cells have erased data of the grid the cells was selected from. In such case you can use the shape cells to get their world position and then read cell of the grid with the data, to read it. (in the future versions of the plugin it will not be required)

When the port is displaying values using [] brackets, that means it's referencing the Field Planner. If there are two [] brackets, that means it's referencing a duplicate instance of field planner.



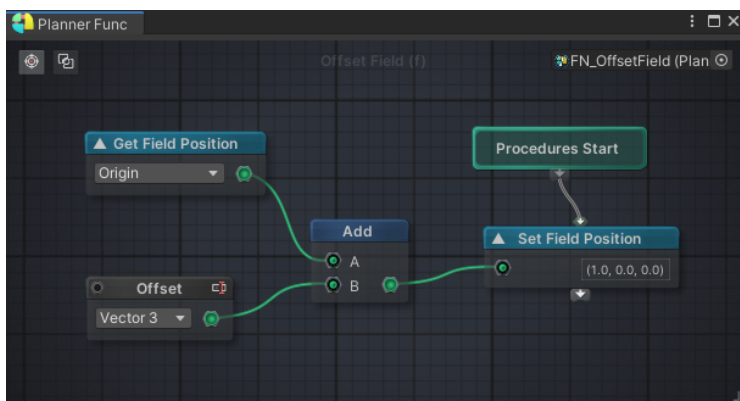
There: referencing to [0] index planner and [2] duplicate, so to the Instance number 3.

Instance zero is original planner, instance 1 is first duplicate of the original planner.  
Value -1 for duplicate means it's no duplicate.

There are also Sub-Fields which are displayed with Sub[] text, where Sub[NR] is sub-field index.

You can provide vector 3 value to the planner port to choose manually some field, which can be useful when debugging. Check the section above for more info about vector reading.

## Function Nodes:




Function nodes are groups of coded nodes condensed into one simple node which can be used in the field planner graphs or inside another function nodes. Function nodes are identified with "(f)" in header name, quick double clicking left mouse button will open function node graph. (there is also option for it when hitting right mouse button on the node in node graph)

You can create new function node by hitting right mouse button on some node on any graph and choosing "Create Custom Function (f) Node" option or by clicking right mouse button in the project browser and going to Create->Fimpossible Creations->Procedural Generation->Create Build Planner Function Node

Function nodes can use nodes which planner graphs can't use.

These are "Input", "Parameter" and "Output" nodes, which will appear on the node drawer of the prepared function node.

Hitting the  icon will allow you to change the name of the port display name. (you can do it also through inspector window which will appear when selecting node)



Beware for changing node count/names/types after adding one of them to the other node graphs, the function node refresh will be improved in the future versions!

## Node Graph Tips:

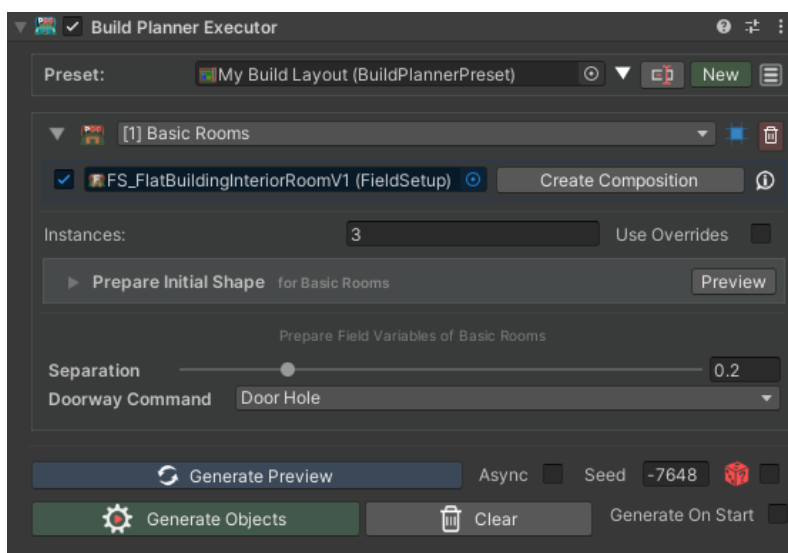
Node Graph offers some shortcuts.

If you keep pushing the 'A' button on your keyboard and hit the left mouse button on the grid canvas, a math "Add" node will be created. When you keep pressing the "1" number key, there will be a value node. There are prepared many shortcuts for different keys you can check (it will be described in the manual when beta stage of the package will be reaching end)

Pressing the 'R' key on your keyboard and hitting the middle mouse button will trigger Build Planner Window refresh, you can also double click the middle mouse button to do the same.

## Build Planner Executor:

Build Planner Executor is a component which you should add to the new empty game object. Generated structures will be setted as child objects of the executor object in the scene hierarchy and generated in its space, so you can freely move / rotate the executor object on the scene.



Build Planner Executor will create GUI accommodated with the build planner preset names, variables and field planners names / variables / preparation.

Build Planner Executor is generating objects using **Grid Painters**. So when you hit "Generate Objects" all generated areas will be accessible for debugging, further editing and free moving/rotating on the scene.

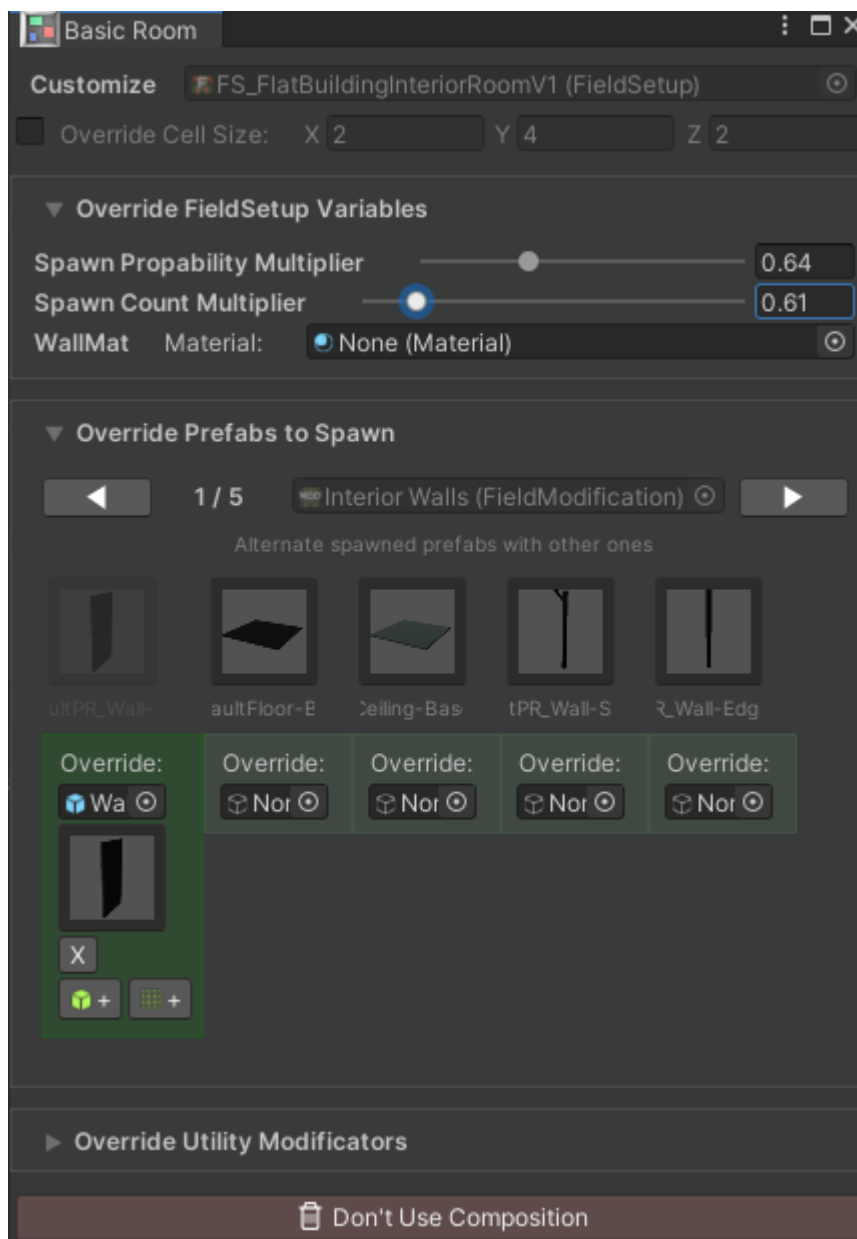
**Executor is generating build planner preset logics with changed cell size as the Field Setups grid cell sizes so the result can look different than with the Build Planner Window preview if "Preview Cell Size" is different!**



## Field Setup Compositions:

When spawning objects with Build Planner Executor or Grid Painters now you have the possibility to alternatively compose the Field Setups.

You can easily replace spawned prefabs with new chosen ones, replace some parameters / variables or disable modifiers/packs individually.



In the future the manual will expand more!

If you like this package please visit my asset store page for more or write a review for this asset ;)