

# Semi Procedural Map Generator

**This is a incomplete verssion of the documentation.  
to help you to understand how the generator work**

# Introduction

The "Semi Procedural Map Generator" asset is a tool that allows you to create a map from a blueprint in order to have a faster and more detailed generation.

The Lvl\_Tutorial map you will find in the asset contains basic explanatory information and animations to help you understand how the asset works. This document will explain in more detail how it works.

Before we start we need to understand how this asset is different from a classic map generator.

In order to create a procedural map it is usually customary to use tiles. These tiles are pieces of floor, walls or other elements that can be repeated.

The first step is to find the location of the rooms and then spawn the floor piece by piece. While checking that no room overlaps.

*It is important to note that spawning an element takes time which will accumulate with the number of spawns. (each piece of floor, wall or scenery element will slow down the map generation more and more).*

After the generation of the floor the shortest path between each room will be calculated to create the corridors. Then come the creation of the walls and the generation of the interiors (light/personage/furniture/decoration....).

Although this method allows to have an almost infinite number of possibilities it has many weak points such as the speed of generation, the ability to have detailed scenery and the least personalization which requires a very good mathematical skills.

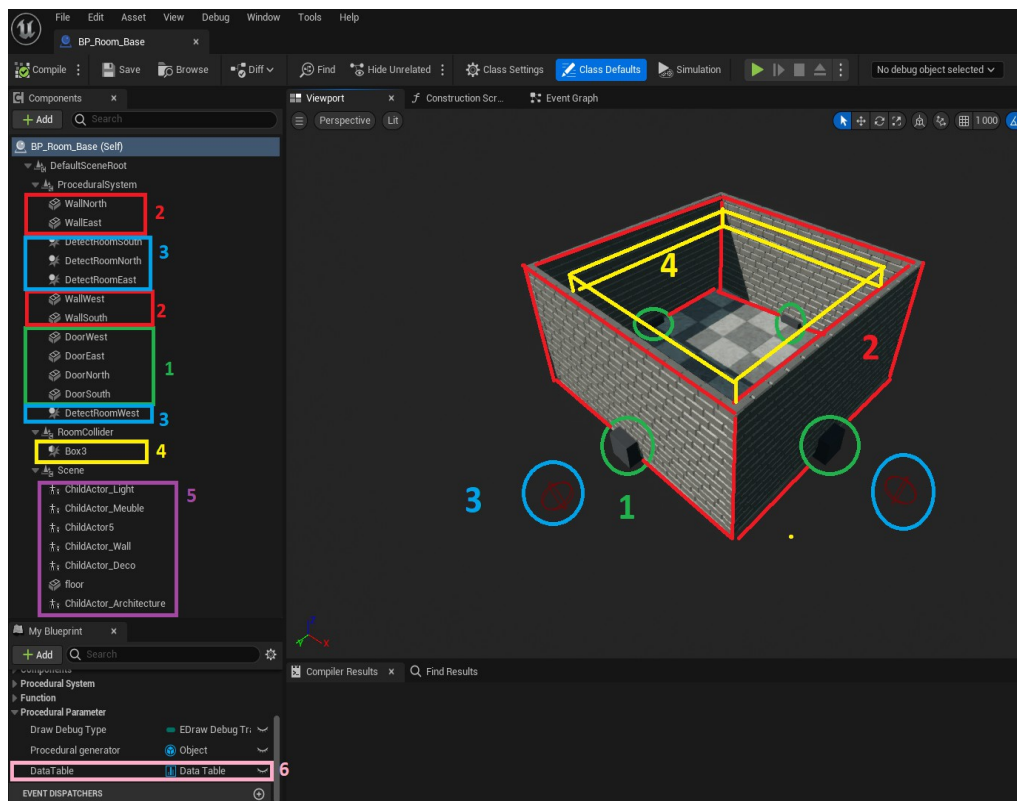
The "Semi Procedural Map Generator" asset requires more work however you can achieve a very large number of possibilities while having much more control over the content of the rooms.

# How does it work?

## 1 Room Generation

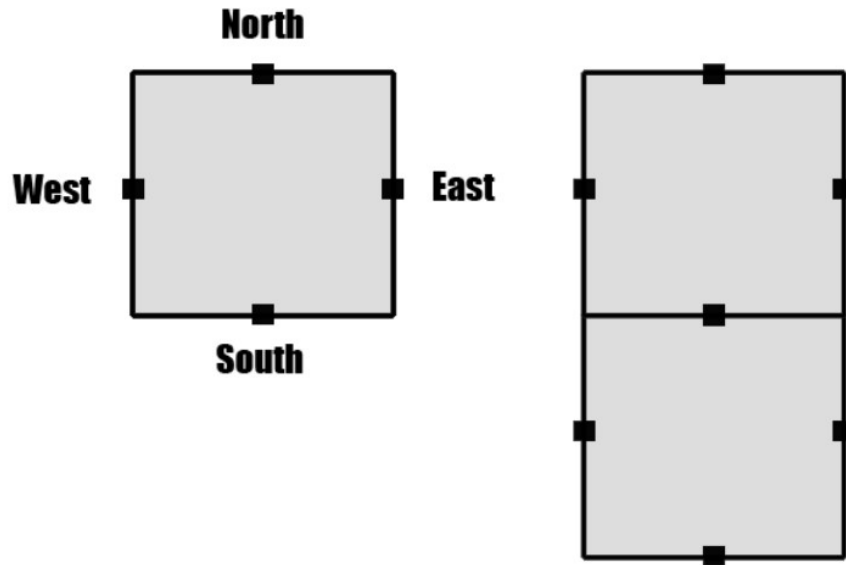
First of all the rooms need several things.

Let's take a look at the blueprint that serves as a basis for the creation of the rooms "BP\_Room\_Base".



### -1 Doors:

The cubes are used to find the location of the doors in the room. There are necessarily four doors (East, West, North, South). This will be used to find where to place the next room using the door as a snapping point. And also where to place the blueprint of the door.



In order to connect one room to another and to find its location we will use the doors. The logic is simple:

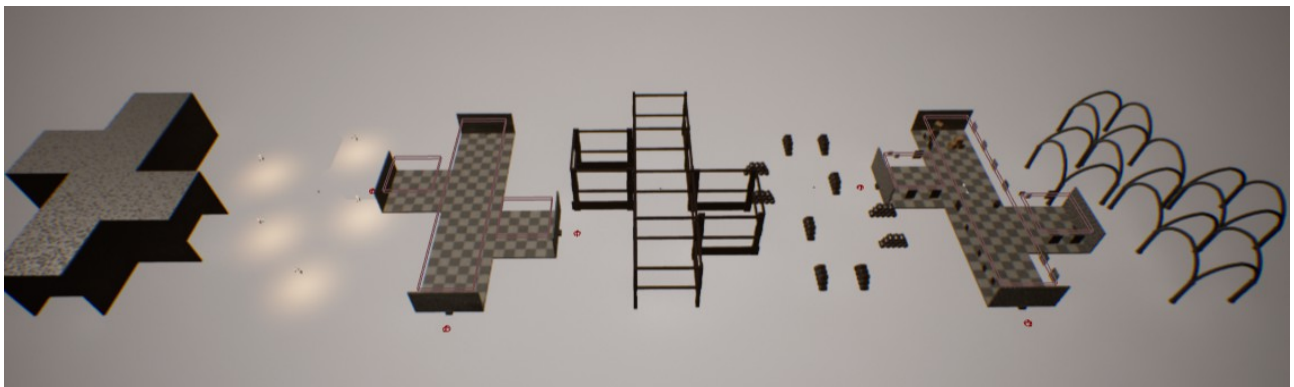
If we create a new room to the north of the current room it means that the door to the north must be the door to the new room and that in the new room it is the door to the south that opens to the old room (like a puzzle). All that's left is to roll a four-sided dice to see which door will open onto the next room.

It is important to understand one thing unlike a more classical method. Walls, decorations, furniture... are preplaced in blueprints. Each room can have several furniture configurations prepared in advance and will load one randomly from a list (DataTable).

Example with one room:



It's the same room base but the walls, the decoration and the light change each time. Here is what the different blueprints look like when they are not stacked on top of each other.



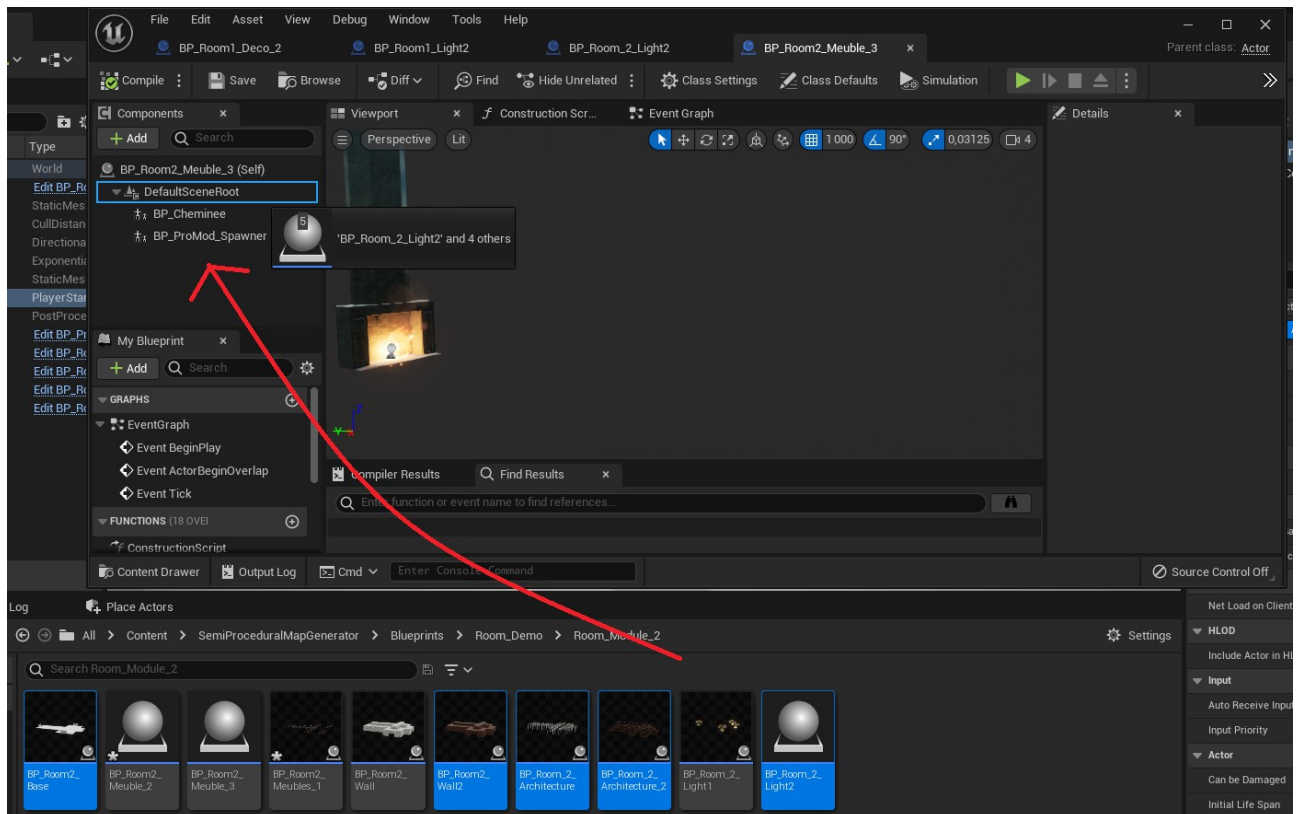
This method allows us to have better performance because each element has been prepared in advance and we don't need to calculate their position every time and there is no spawn to perform. And finally it is possible to have detailed environments without the risk of an element appearing out of place. It's also possible to make specific rooms like a store, a boss room that adapts to the boss inside, etc.

## Important note:

The blueprints that serve as backgrounds will be positioned at the coordinate (0,0,0) relative to the room.

A good way to know where and how to place the elements is to open the blueprint you want to edit and then drag the other blueprints relative to the room.

## Example :

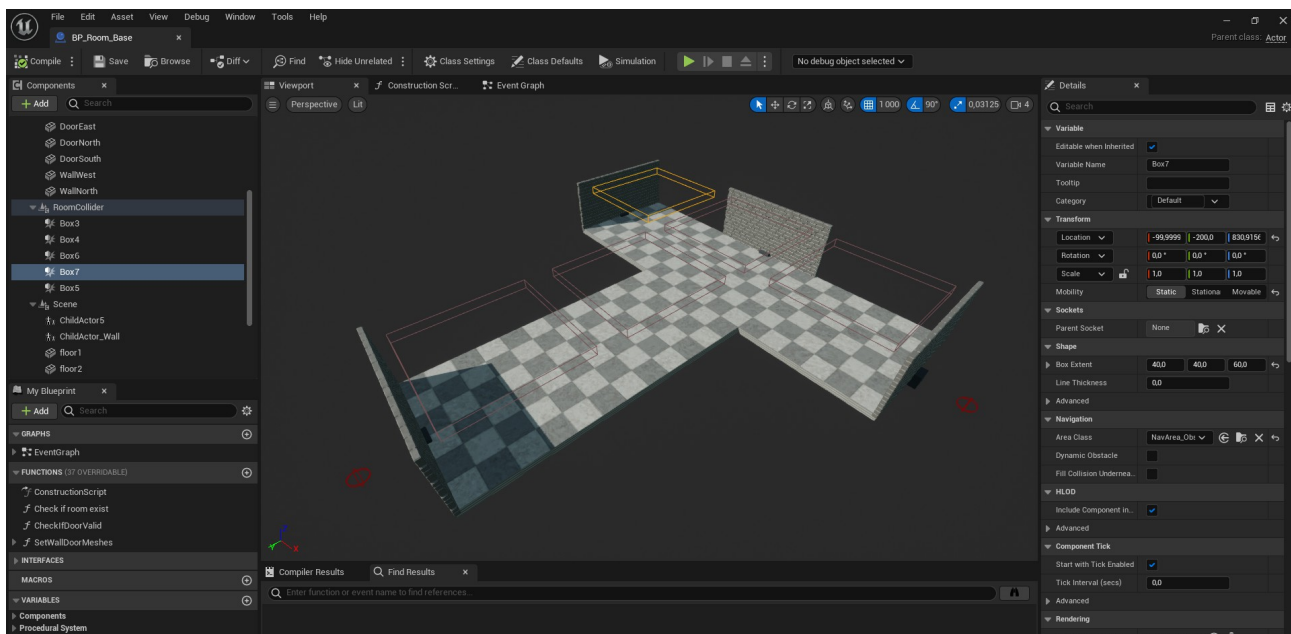
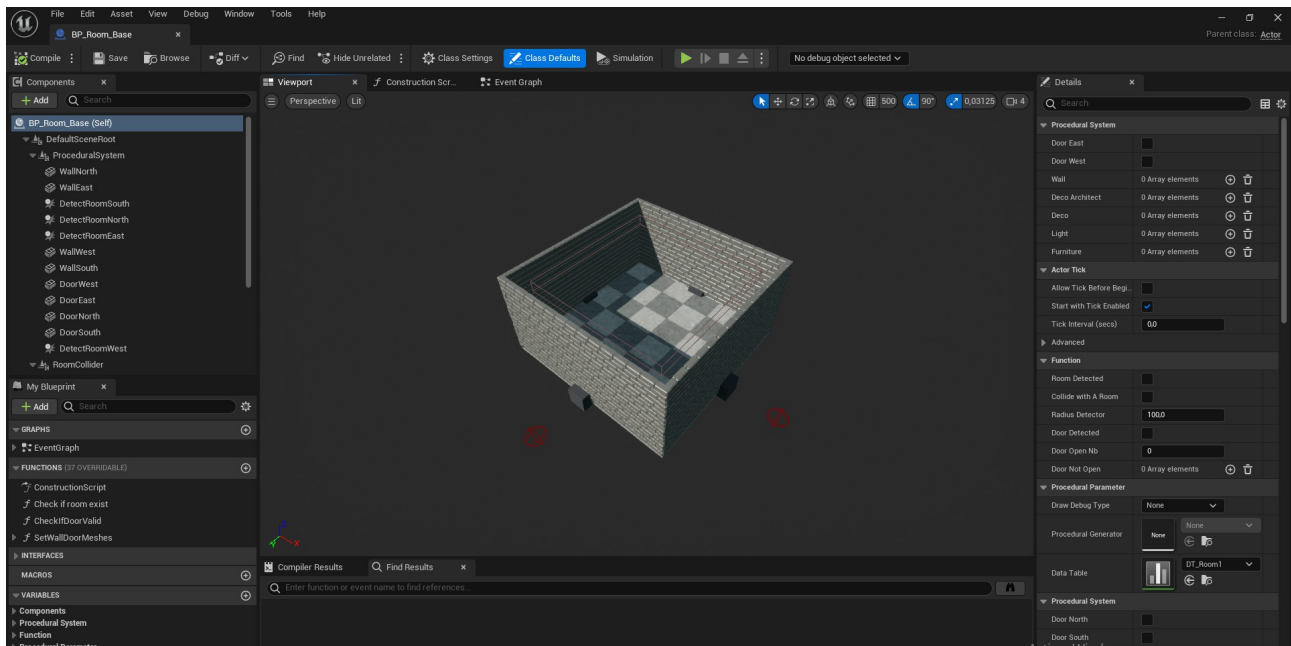


In this example I want to create a new blueprint containing a new configuration for the furniture. In order to know where the pillars, the floor and the walls are I drag the blueprints containing those information to the root of the blueprint being edited. This way the floor, the walls and the rest will be perfectly placed as ChildActor and will only need to be removed once finished.

## Step: 1

We will create a T-shaped room.

To do this, we make a copy of the BP\_Room\_Base, then we move the elements.



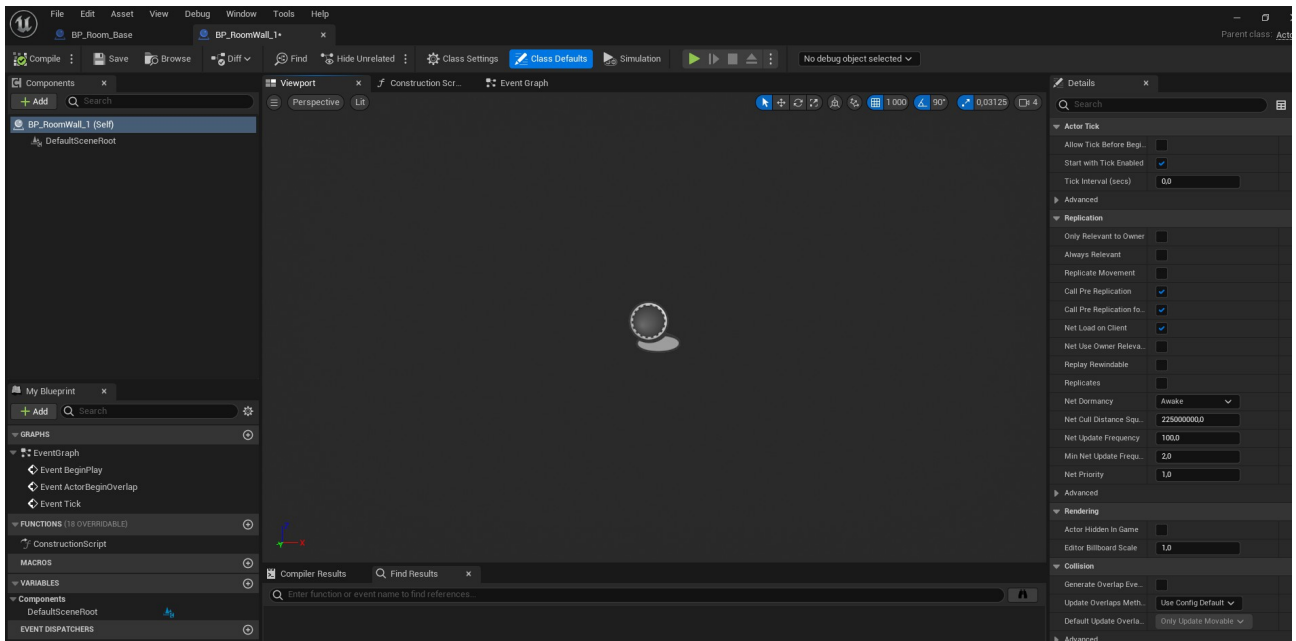
Once all the doors are placed, we copy the floor and the box collider.

Any shape can be made, but remember that the box collider must cover the entire surface of the room.



## Step2 :

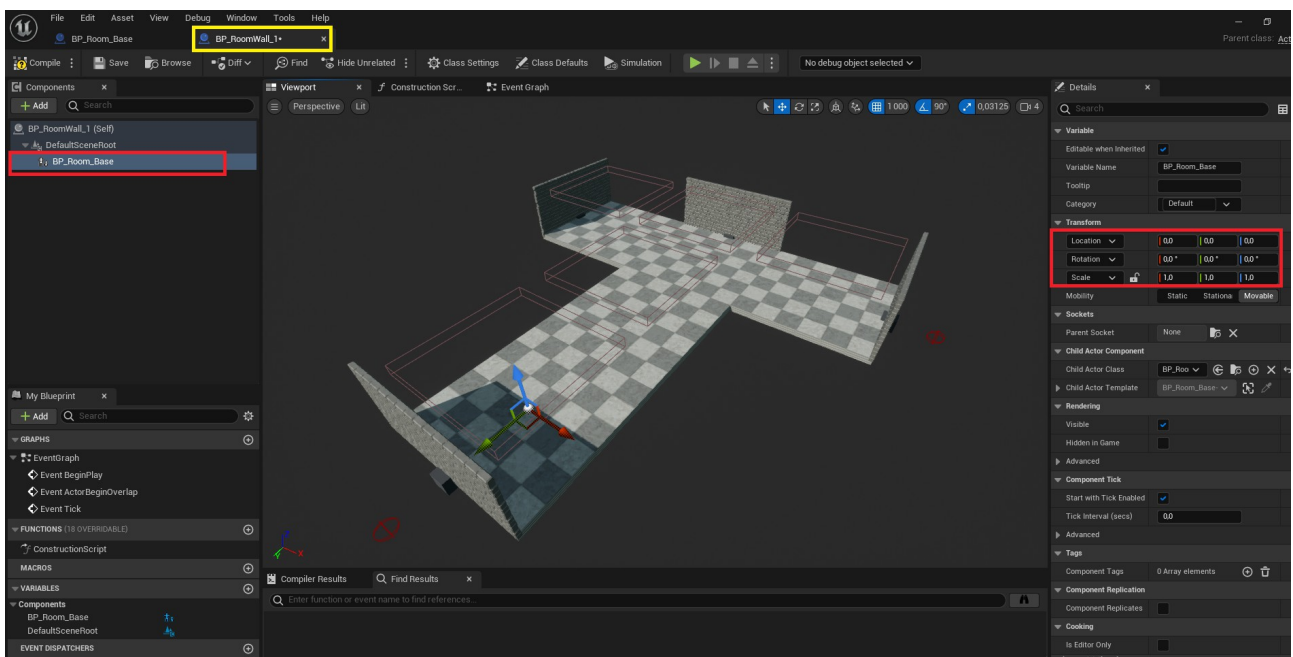
Now that the room shape is created, we will create the wall.  
To do this, create an empty blueprint actor.



Then drag and drop the plane with your room shape into the new blueprint.  
It must be at the location (x:0 y:0 z:0).

## Why do this?

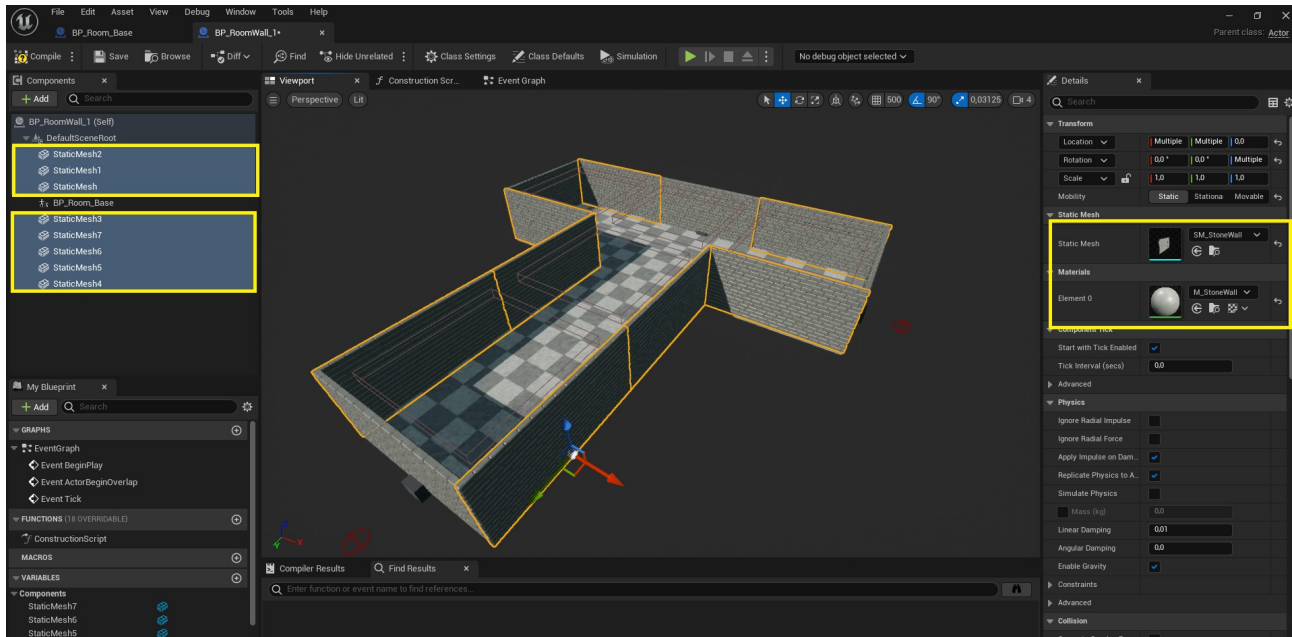
It is only to see if your elements are placed the way you want. When the room spawns the elements, they will be in exactly the same place (you don't want a table to be placed in a wall or under the floor).



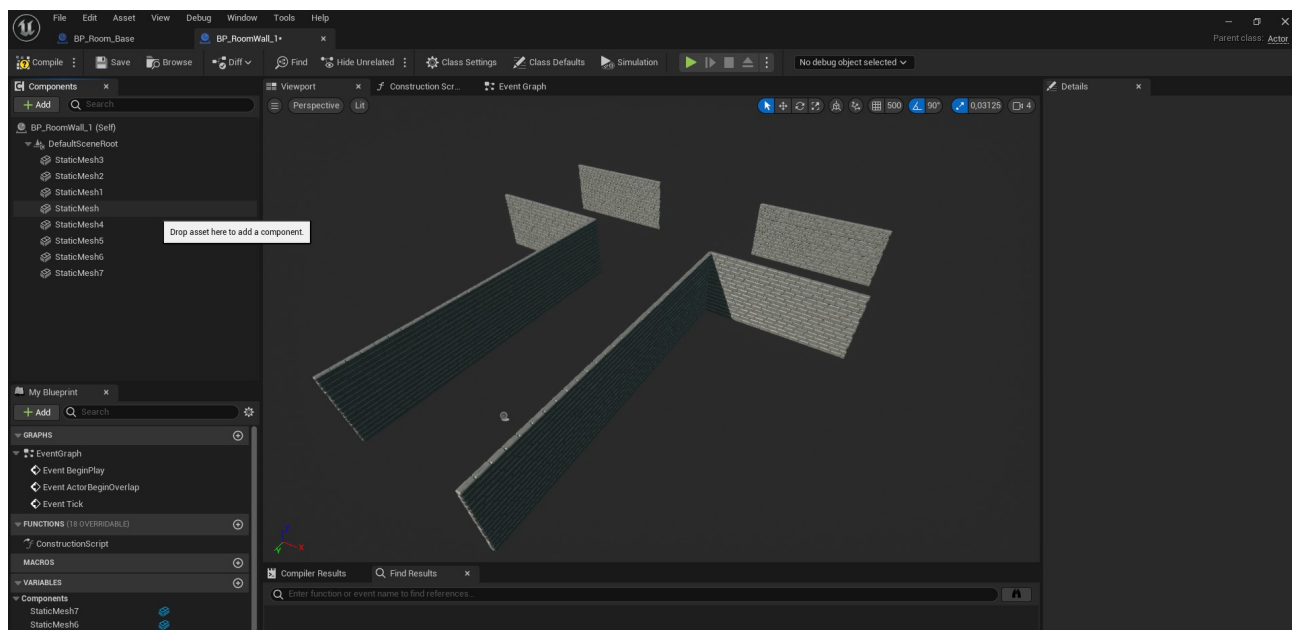
Then, you will place the walls as you wish.



In this example I used static meshes, but you can also create instantiated meshes if you want to optimize your room.



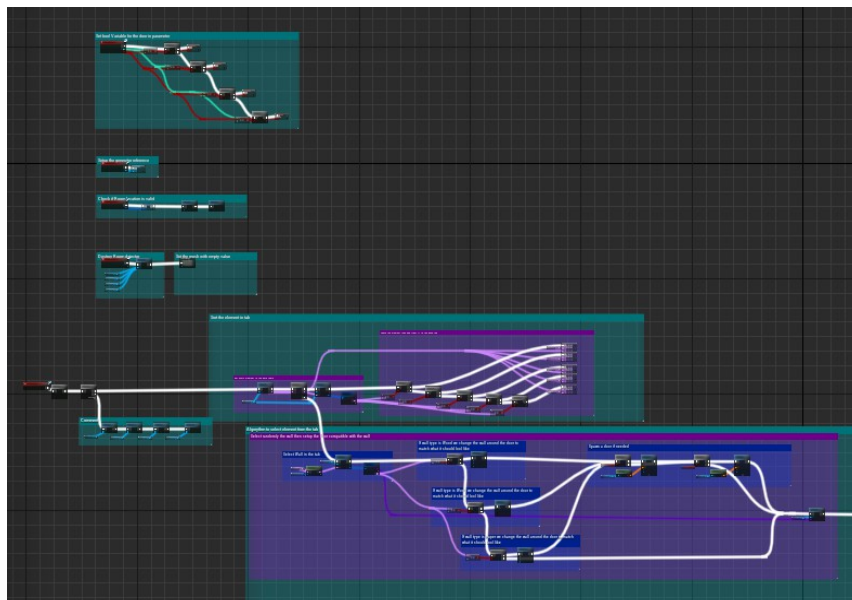
Then delete the BP\_Room\_Base in your blueprint to get this.



This way, if we spawn these walls from the BP\_Room\_base, they will be in the right place.

The screenshot shows the Unreal Engine 4.26.2 interface. The top menu bar includes File, Edit, Asset, View, Debug, Window, Tools, and Help. The top toolbar contains buttons for Compile, Save, Browse, Diff, Find, Hide Unrelated, Class Settings, Class Defaults, Simulation, and a play button. The Components panel on the left shows a hierarchy with BP\_RoomArchitecture (Self) at the top, followed by DefaultSceneRoot, BP\_RoomWall\_1, and BP\_Room\_Base. The Transform panel on the right shows the location, rotation, and scale of the selected component. The main viewport displays a 3D model of a room with a checkered floor and stone walls.

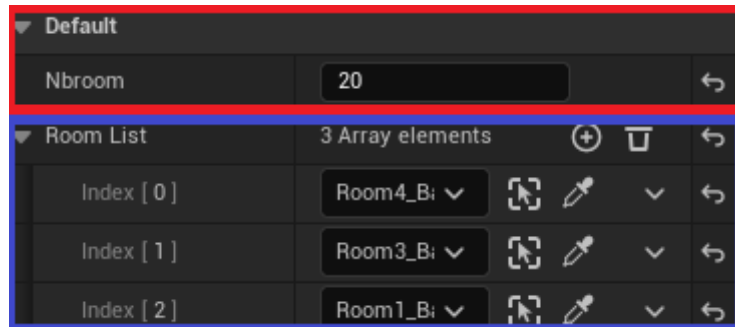
Note: the blueprints are fully commented and may have an answer to your question.



### **3 Map generation :**

Now that we know how the generation of the rooms works we can tackle the generation of the map.

To create the map we will use the blueprint : BP\_Procedural\_Map\_Generator.  
Here are the parameters that the blueprint takes.



#### **NbRoom :**

The number of rooms that the generator will create.

The more rooms you ask for the longer the generation time will be.

With a 7 years old processor here are my generation times.

20 room : 0,6s

100 room : 2,5s

500 room : 17s

1000 room : 30s

**CAUTION: If you encounter an infinite loop, it may be due to too many room. In this case, try increasing the "Maximum value for the number of loop iterations" in the project settings.**

#### **Room List :**

The list of BP\_Room\_Base that will be used to create the map.

They must be placed in the lvl and given as reference to the map generator.

(I'll look for another easier way for a next update).

#### **Room Detector :**

Is a references to a BP\_RoomDetector placed on the map. It will generate the interior of the room close to the player.

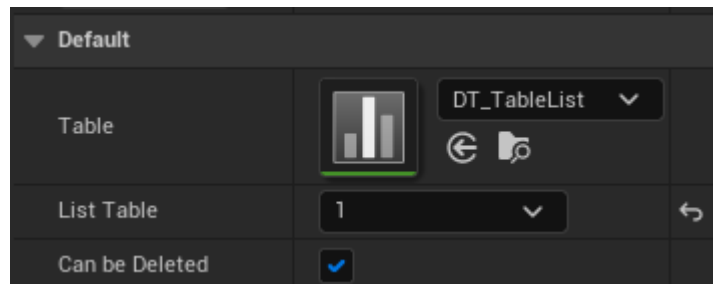
**Seed :** The seed is a random number that will affect the randomness. If you like a map keep the seed to use it again later.

**Use Specific Seed :** By default a random seed is created each time a map is created but if you prefer to use a seed. Enable this boolean then set the seed.

### **Randomize scenery elements:**

The BP\_Item\_Spawner blueprint will select a random Actor from a DataTable given in parameter and display it at its location and rotation.

This allows to randomize which actor will be displayed at this location.



### **Table :**

DataTable in which the blueprint will select an actor to display.

**Note: The DataTable contains blueprint actors which means that it is possible that the objects displayed have blueprint code and interaction such as physics they are not just meshes.**

### **List Table :**

Used by the editor to select which actor to display in the list.

### **Can Be Deleted :**

It is a boolean if this one is true there is a 1 in 5 chance that nothing will be displayed instead of the actor.