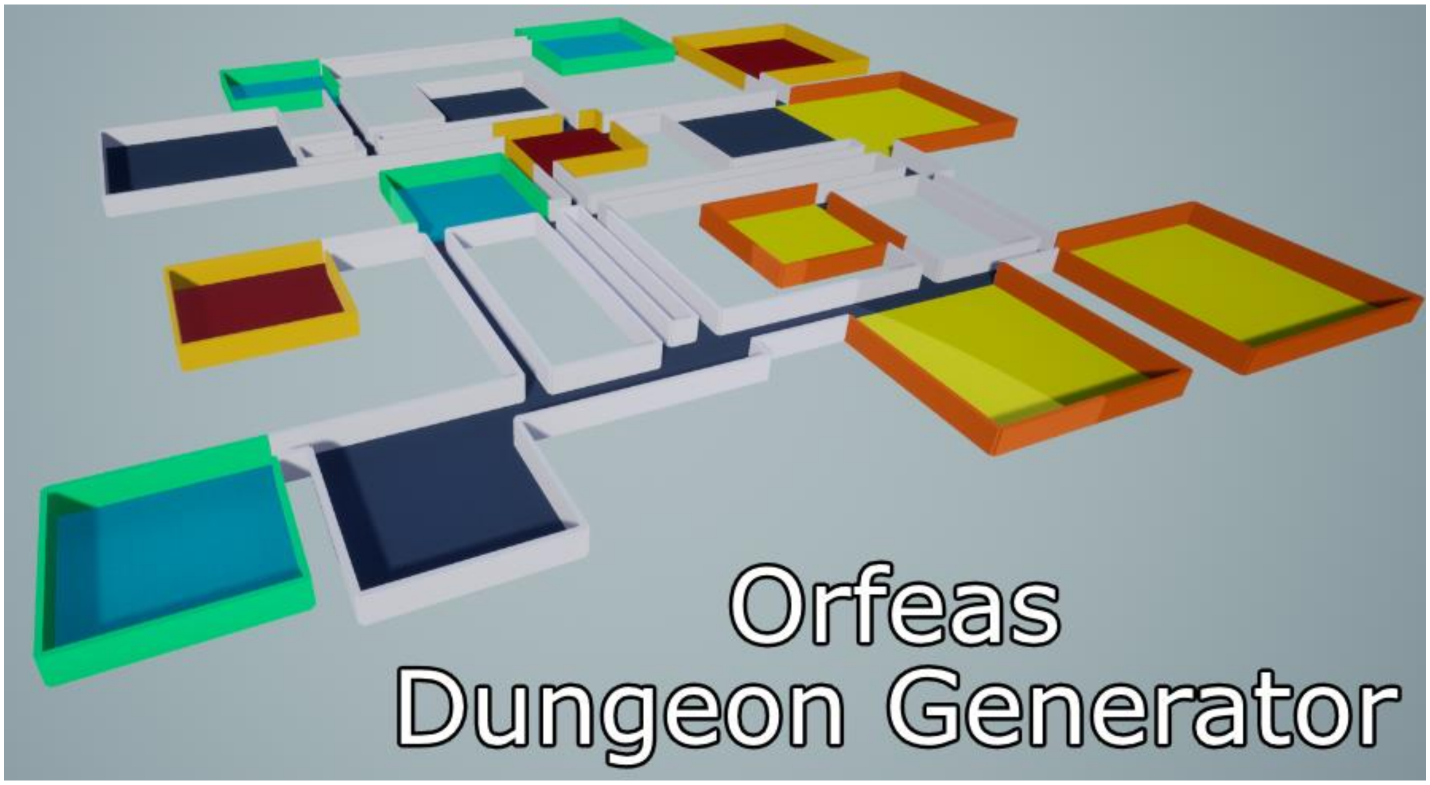
Orfeas Dungeon Generator



A handy dungeon generator for Unreal Engine 4. Can be used in editor and at runtime.

Date: June 10, 2022

Version 1.1

Changelog

- Split Data Table from Generic Floor & Wall settings
- Fixed buggy pivot offsets for floor and walls meshes

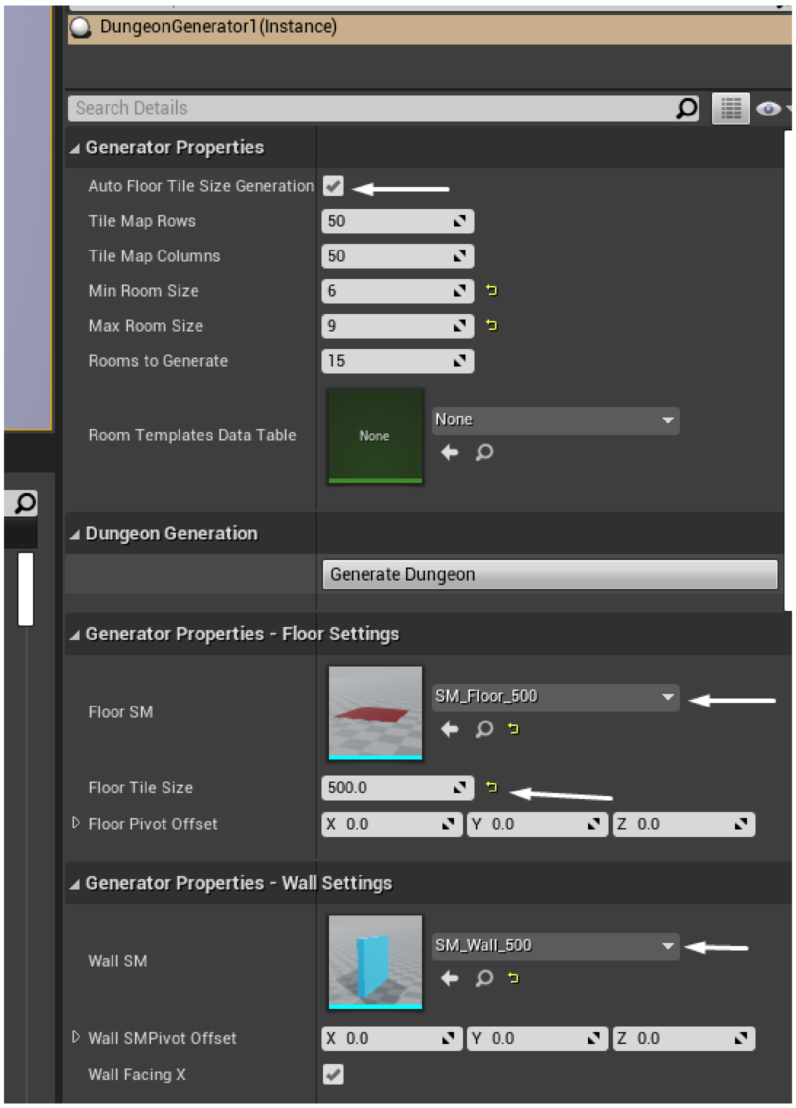
Contents

1. How to use

In order to use the dungeon generator:

- Place a Dungeon Generator actor from the “Place Actors” panel
- From the details panel:
    - Assign a static mesh to FloorSM (the static mesh for each floor tile)
    - If the plugin doesn’t automatically detect the correct size, turn off the Auto Floor Tile Size Generation property and manually assign the correct value to Floor Tile Size
    - Assign a static mesh to WallSM (the static mesh for each wall). **The plugin assumes that wall extents match your assigned tile** meaning that if your tile size is 500 then your wall’s static mesh width should be 500 units as well.
- Click the Generate Dungeon button

Here’s a screenshot summing up the following steps:



DungeonGenerator1 (Instance)

Search Details

Generator Properties

Auto Floor Tile Size Generation	✓
Tile Map Rows	50
Tile Map Columns	50
Min Room Size	6
Max Room Size	9
Rooms to Generate	15
Room Templates Data Table	None

Dungeon Generation

Generate Dungeon

Generator Properties - Floor Settings

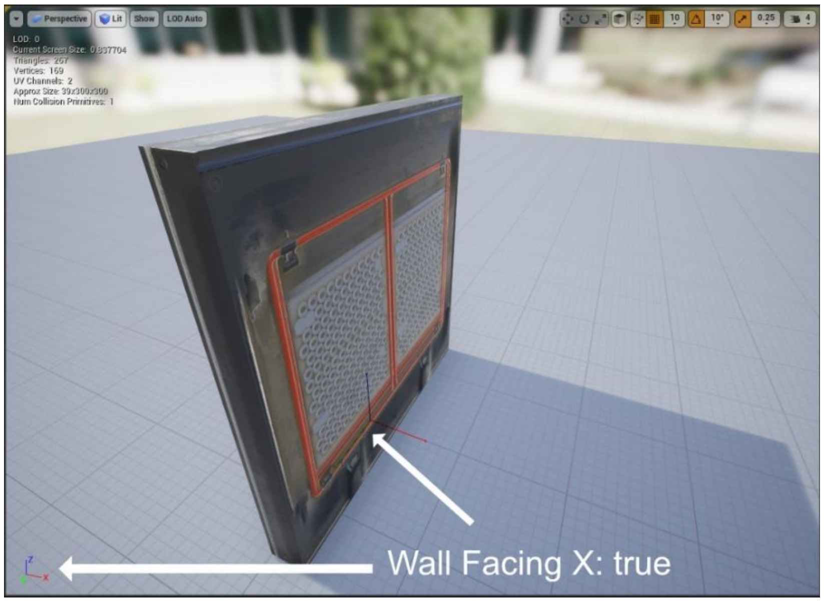
Floor SM	SM_Floor_500
Floor Tile Size	500.0
Floor Pivot Offset	X 0.0 Y 0.0 Z 0.0

Generator Properties - Wall Settings

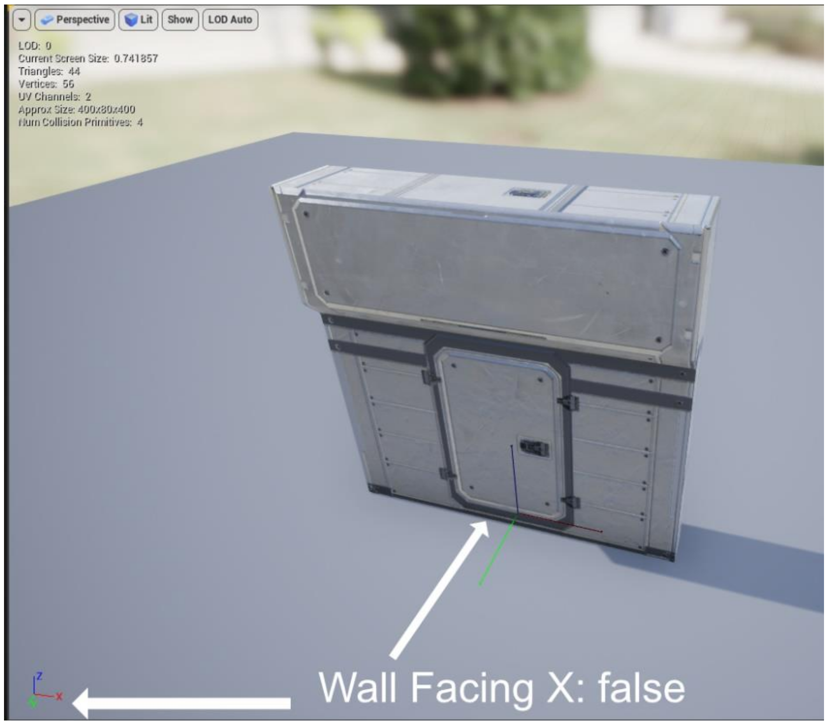
Wall SM	SM_Wall_500
Wall SMPivot Offset	X 0.0 Y 0.0 Z 0.0
Wall Facing X	✓

1.1 Configuring wall meshes

Depending on the imported wall mesh, you may have to modify the Wall Facing X property, found under Generator Properties – Wall Settings. To determine the correct value for your particular case, open up your wall mesh and see if your wall is facing the X axis. If that's the case, make sure to mark this option as true (false otherwise). An example of two meshes using this option:



1: Wall-Facing-X:True



2:Wall-Facing-X:False

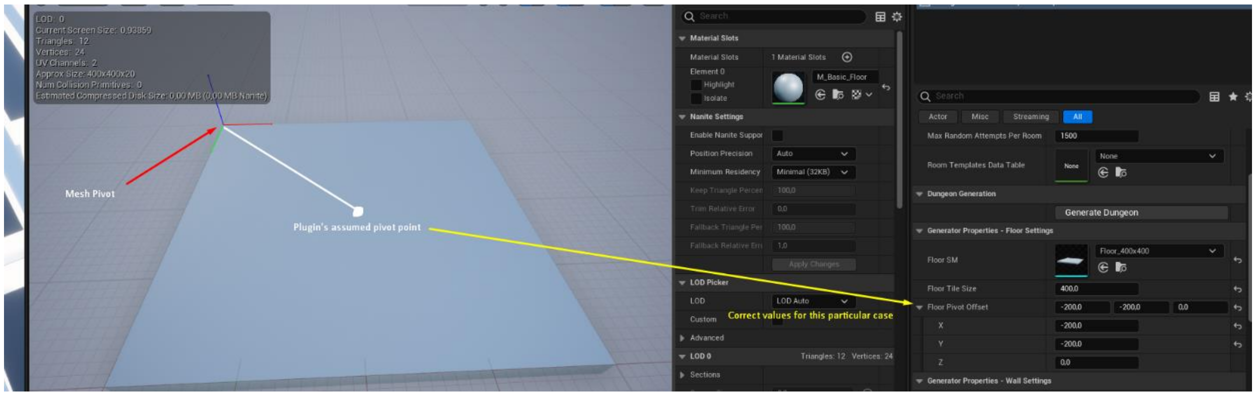
1.2 Adjusting Pivot Offsets for Floors and Walls

By default, the plugins assumes that every floor tile has a pivot point on its center and every wall mesh has its pivot point on the lower center (check Figure 1 where pivot point is displayed). Sometimes, floor tiles have their pivot point on a single corner and wall meshes have pivot points either at the lower left or lower right corner. If you meshes' pivot points aren't centered use the properties:

- Floor Pivot Offset and set a relative value to offset the pivot to the correct location for your mesh (see example after this paragraph)
- Wall SM Pivot Offset performs the same pivot offset but for walls only

1.2.1 Pivot Offset Examples

In the following example our floor tile is 400x400. Since the pivot point is on the left corner (check the following image) we need to tell the plugin that the pivot offset for this case is (-200, -200) (basically the tile's size / 2)


Mesh Pivot
Plugin's assumed pivot point
Correct values for this particular case

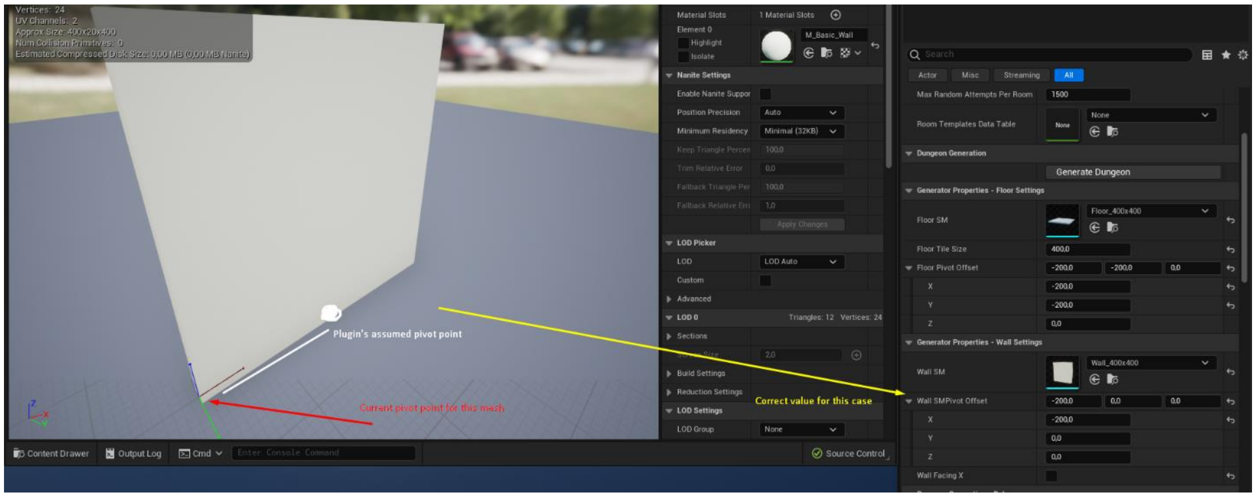
LOD: 0
Current Screen Size: 0.93859
Triangles: 12
Vertices: 24
UV Channels: 2
Approx Size: 400x400x20
Num Collision Primitives: 0
Estimated Compressed Disk Size: 0.00 MB (0.00 MB Nanite)

Material Slots	1 Material Slots
Element 0	M_Basic_Floor
Enable Nanite Support	
Position Precision	Auto
Minimum Residency	Minimal (32KB)
Keep Triangle Percent	100.0
Trim Relative Error	0.0
Fallback Triangle Percent	100.0
Fallback Relative Error	1.0
LOD	LOD Auto
LOD 0	Triangles: 12 Vertices: 24

Max Random Attempts Per Room	1500		
Room Templates Data Table	None		
Floor SM	Floor_400x400		
Floor Tile Size	400.0		
Floor Pivot Offset	-200.0	-200.0	0.0
X	-200.0		
Y	-200.0		
Z	0.0		

3: Adjusted Pivot Offset - See image FloorTilePivotOffset.png in Images folder

Using the same logic as above, we adjust the wall's pivot offset as well:


Plugin's assumed pivot point
Current pivot point for this mesh
Correct value for this case
Content Drawer
Output Log
Cmd
Enter Console Command
Source Control

Vertices: 24
UV Channels: 2
Approx Size: 400x20x400
Num Collision Primitives: 0
Estimated Compressed Disk Size: 0.00 MB (0.00 MB Nanite)

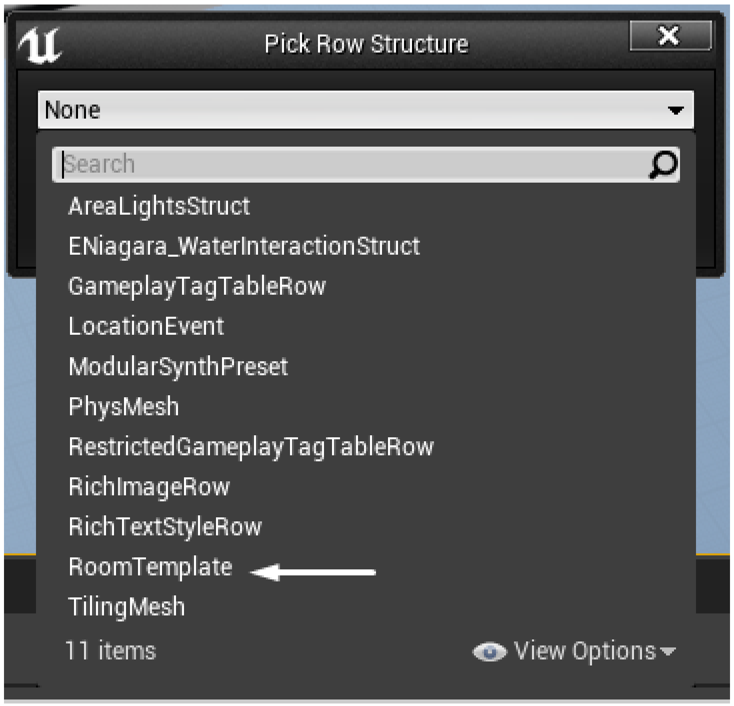
Material Slots	1 Material Slots
Element 0	M_Basic_Wall
Enable Nanite Support	
Position Precision	Auto
Minimum Residency	Minimal (32KB)
Keep Triangle Percent	100.0
Trim Relative Error	0.0
Fallback Triangle Percent	100.0
Fallback Relative Error	1.0
LOD	LOD Auto
LOD 0	Triangles: 12 Vertices: 24
LOD Group	None

Max Random Attempts Per Room	1500		
Room Templates Data Table	None		
Floor SM	Floor_400x400		
Floor Tile Size	400.0		
Floor Pivot Offset	-200.0	-200.0	0.0
X	-200.0		
Y	-200.0		
Z	0.0		
Wall SM	Wall_400x400		
Wall SMPivot Offset	-200.0	0.0	0.0
X	-200.0		
Y	0.0		
Z	0.0		
Wall Facing X			

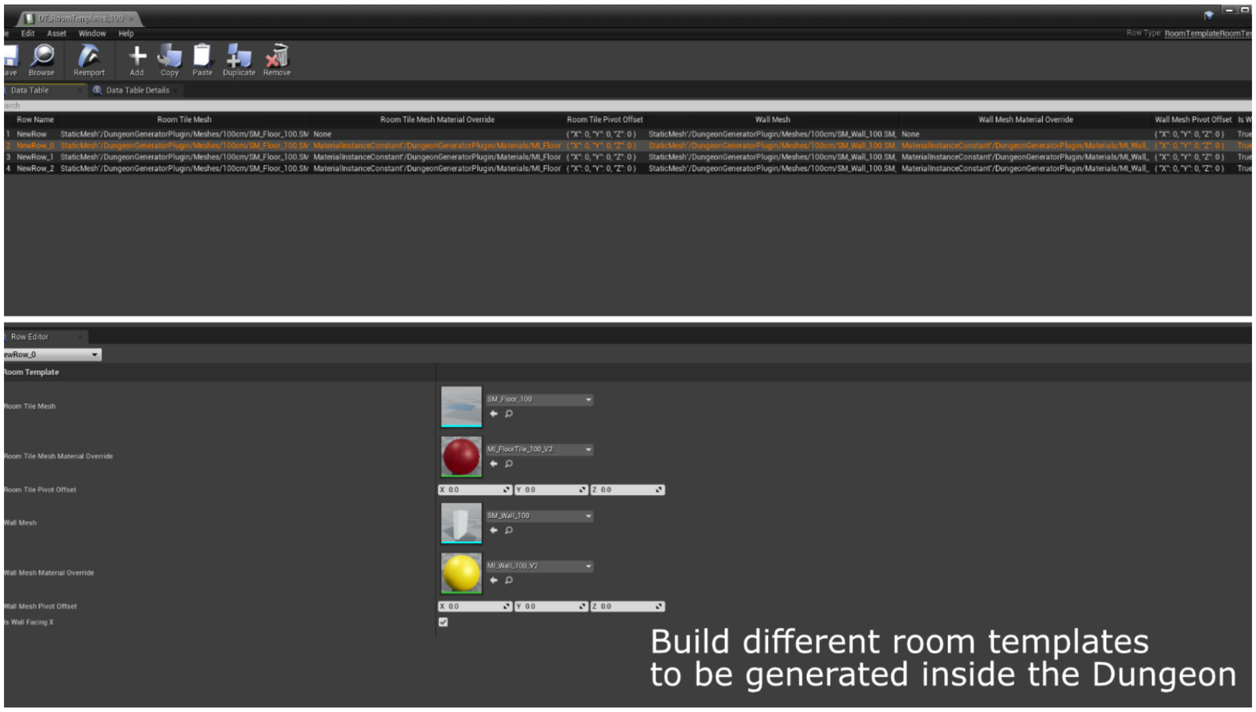
4: Adjusted Pivot Offset for Wall Mesh - See image WallPivotOffset.png in Images folder

2. Using the Room Templates Data Table

This dungeon generator offers the option to use different floor and wall meshes for different rooms. To use this, create a new data table and choose the RoomTemplate type:



Then, proceed on adding your floor static mesh as well as the walls for this particular room:



	Row Name	Room Tile Mesh	Room Tile Mesh Material Override	Room Tile Pivot Offset	Wall Mesh	Wall Mesh Material Override	Wall Mesh Pivot Offset	Is W
1	NewRow	StaticMesh'/DungeonGeneratorPlugin/Meshes/100cm/SM_Floor_100.SM	None	{ "X": 0, "Y": 0, "Z": 0 }	StaticMesh'/DungeonGeneratorPlugin/Meshes/100cm/SM_Wall_100.SM_	None	{ "X": 0, "Y": 0, "Z": 0 }	True
2	NewRow_0	StaticMesh'/DungeonGeneratorPlugin/Meshes/100cm/SM_Floor_100.SM	MaterialInstanceConstant'/DungeonGeneratorPlugin/Materials/MI_Floor	{ "X": 0, "Y": 0, "Z": 0 }	StaticMesh'/DungeonGeneratorPlugin/Meshes/100cm/SM_Wall_100.SM_	MaterialInstanceConstant'/DungeonGeneratorPlugin/Materials/MI_Wall_	{ "X": 0, "Y": 0, "Z": 0 }	True
3	NewRow_1	StaticMesh'/DungeonGeneratorPlugin/Meshes/100cm/SM_Floor_100.SM	MaterialInstanceConstant'/DungeonGeneratorPlugin/Materials/MI_Floor	{ "X": 0, "Y": 0, "Z": 0 }	StaticMesh'/DungeonGeneratorPlugin/Meshes/100cm/SM_Wall_100.SM_	MaterialInstanceConstant'/DungeonGeneratorPlugin/Materials/MI_Wall_	{ "X": 0, "Y": 0, "Z": 0 }	True
4	NewRow_2	StaticMesh'/DungeonGeneratorPlugin/Meshes/100cm/SM_Floor_100.SM	MaterialInstanceConstant'/DungeonGeneratorPlugin/Materials/MI_Floor	{ "X": 0, "Y": 0, "Z": 0 }	StaticMesh'/DungeonGeneratorPlugin/Meshes/100cm/SM_Wall_100.SM_	MaterialInstanceConstant'/DungeonGeneratorPlugin/Materials/MI_Wall_	{ "X": 0, "Y": 0, "Z": 0 }	True

ewRow_0

oom Template

oom Tile Mesh	SM_Floor_100
oom Tile Mesh Material Override	MI_FloorTile_100_V2
oom Tile Pivot Offset	X 0.0 Y 0.0 Z 0.0
all Mesh	SM_Wall_100
all Mesh Material Override	MI_Wall_100_V2
all Mesh Pivot Offset	X 0.0 Y 0.0 Z 0.0
s Wall Facing X	✓

Build different room templates
to be generated inside the Dungeon

You can assign a material override for each individual mesh. This is used to generate different looking rooms with the same mesh!

2.1 Version 1.1 settings for Data Table

The data table assumes that the imported meshes have the same dimensions. The 1st line of the data table acts as a "generic" template, meaning that the 1st row containing a floor and wall mesh will be automatically used for all corridors as well as some rooms. In case you assign a data table, the plugin will take a random row and attempt to spawn a room. Unlike version 1.0 you no longer need to assign the "FloorSM" and "WallSM" properties in the details panel for the data table to work!

Version 1.1 is only available in UE5+ so in case you're using the plugin in version 4.27 check Section: 2.2 Version 1.0 settings for Data Table (Deprecated as of version 1.1) to adjust your settings.

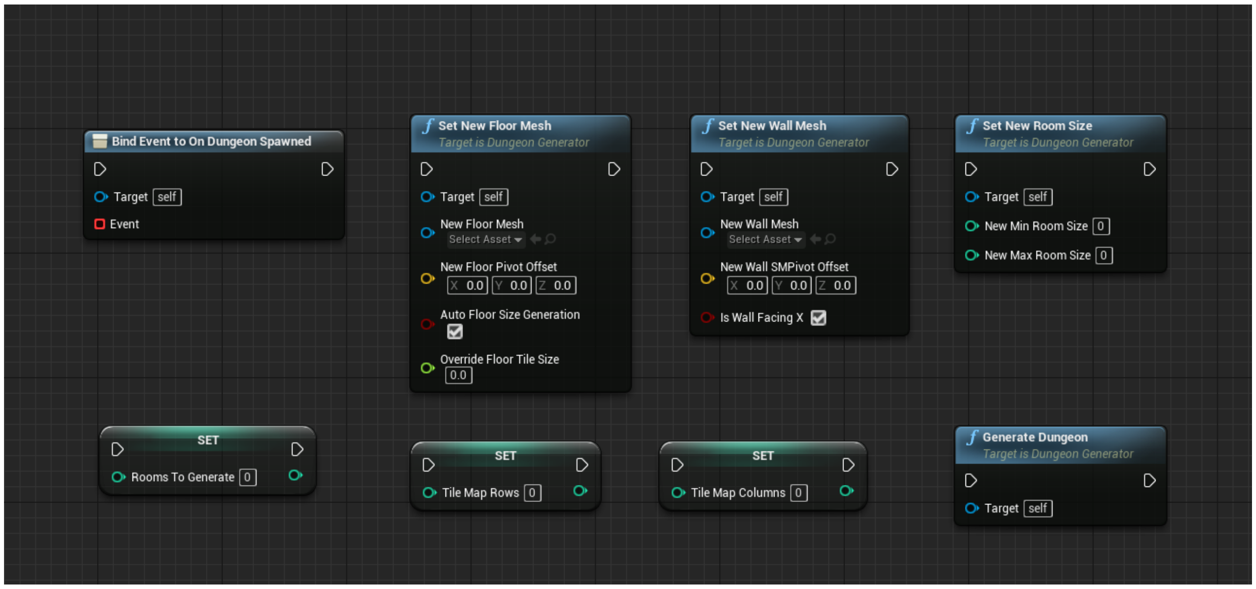
2.2 Version 1.0 settings for Data Table (Deprecated as of version 1.1)

The generator assumes that every floor tile is matching the "FloorSM" tile size you added on the details panel.

Once you have configured your data table, assign it to the details panel and generate a new dungeon. The generator will automatically try to use the assigned data table. Moreover, all the corridor floor tiles and walls will fall back to the default "FloorSM" and "WallSM" properties so **don't forget to assign these properties whether you're using the Data Table or not**.

3. Configuring the generator at Runtime

The dungeon generator actor provides the following BP functions and properties:



Moreover, you can use the OnDungeonSpawned callback to call any custom code when the generator has finished spawning various meshes.

4. A brief overview of what’s happening behind the scenes

The plugin contains two C++ classes:

- A Tile Matrix class, responsible for generating locations of rooms and walls (therefore designing the whole dungeon)
- A Dungeon Generator class, which takes all the input from the Tile Matrix class and spawns the assigned meshes in the world, bringing the whole dungeon to life!

The tile matrix class initially creates a NxY matrix of Booleans. Afterwards, it randomly assigns true values to various elements on the matrix. Whenever a new element of the matrix is selected, the tile matrix tries to “fit” a room into the nearby tiles if they’re not already occupied. If a room can fit in the selected space, the tile matrix proceeds on marking all the involved elements as true, otherwise the elements of the matrix remain unchanged and a room isn’t spawned.

Each time the Tile Matrix creates a new room somewhere, it also connects it automatically to the previous added room using the shortest available path.

Once all required tiles in the matrix have been marked as “true” (which means at this point that we have generated all the rooms and the connections between them), we’re going through each occupied tile and peek its nearby tiles. For the nearby tiles that aren’t occupied, we need to spawn walls (since the tile matrix didn’t generate floor to that location), therefore the tile matrix also stores these locations to an array and also marks the orientation of the walls. The orientation comes in handy when we need to rotate walls based on their world location (see section 1.1)

Once the Tile Matrix completes the whole process, it provides a list of locations for all the floor tiles and wall locations to the Dungeon Generator Actor. Then, the dungeon generator removes any previously spawned meshes and creates the new meshes on the provided locations.