Advanced Dungeon Generator

USER GUIDE

    - Degenerate Meshes
      A way to quickly modify the Transform component of your at-runtime generated Prefabs.
    - Blueprints
      A way to add components to your at-runtime generated meshes.

- **Dungeon Properties**
  A breakdown of all the elements that make up a Dungeon and how they influence its creation.

- **Dungeon Generator**
  A breakdown of the Dungeon Generator, the tool in charge of the in-game, at-runtime Dungeon creation; as well as important notes about the general rules and steps for Dungeon instances creation.

- **Dungeon Loader**
  Your Dungeons will be dynamically generated, the Dungeon Loader is the background element responsible for the correct loading of your Dungeons, and to ensure that everything goes smoothly!

- **Shape Visualizer**
  A sub-tool that lets you see the changes you made to your Dungeon Shapes in real time.

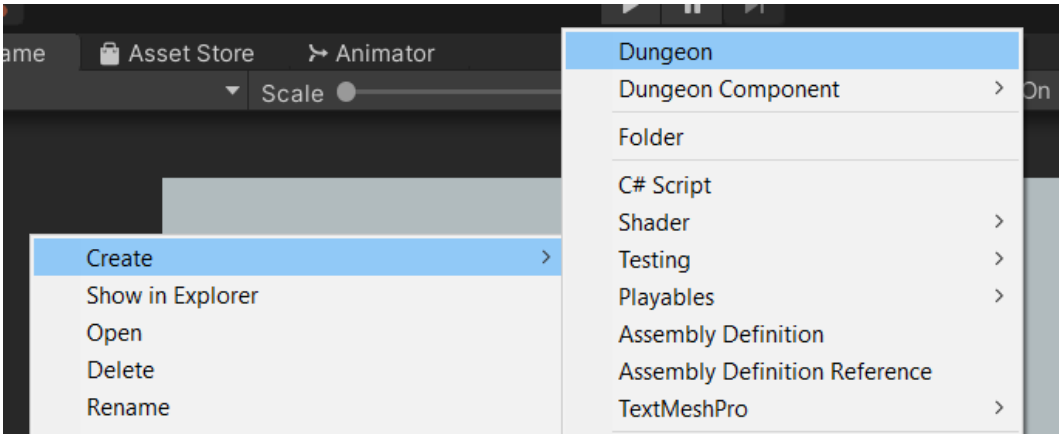
- **Security Check**
  An ever-present background checker to prevent catastrophic errors!

- **Demos**
  ADG includes lots of demos that can serve as starting points for your projects.

- **FAQ**
    - How do I add **Collisions** to the Walls?
    - Are my external Assets compatible with *ADG*?
    - How do I add a **NavMesh** to the Dungeon?

Creating a Dungeon

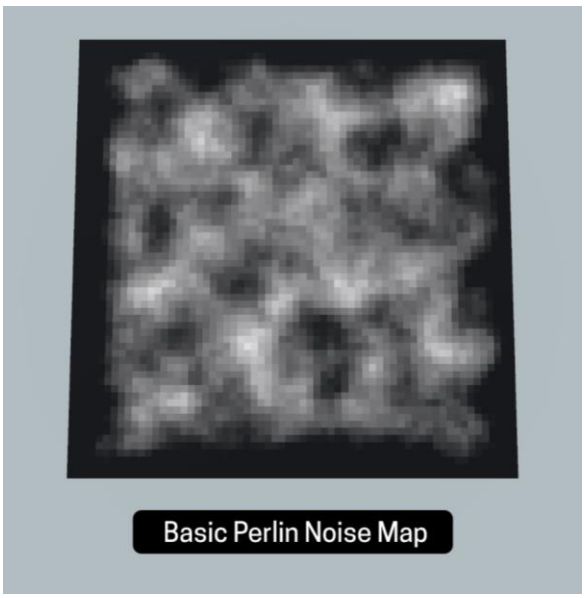
To create a new Dungeon or any of its Components, select the menu item **Assets > Create > Dungeon** or, in the Project Window, **Right Click > Create > Dungeon.**

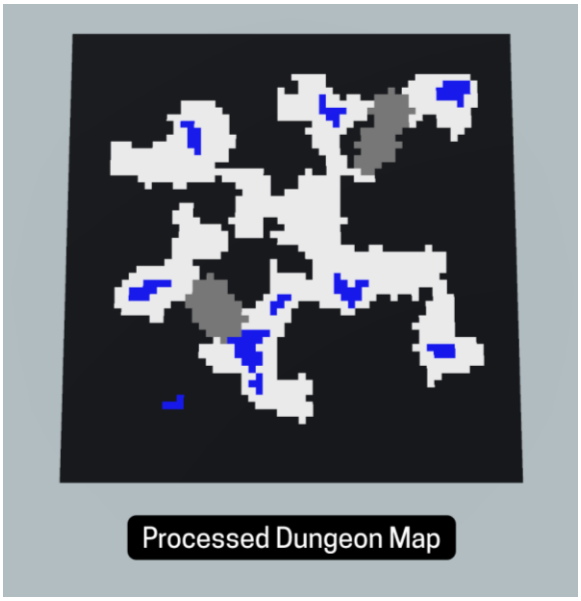


Key Concepts

At its core *ADG* is a **Square-Grid Tile Editor**.

By manipulating a Perlin Noise Map, *ADG* can grant different **Tile Types** and create unlimited **Dungeon Maps.**



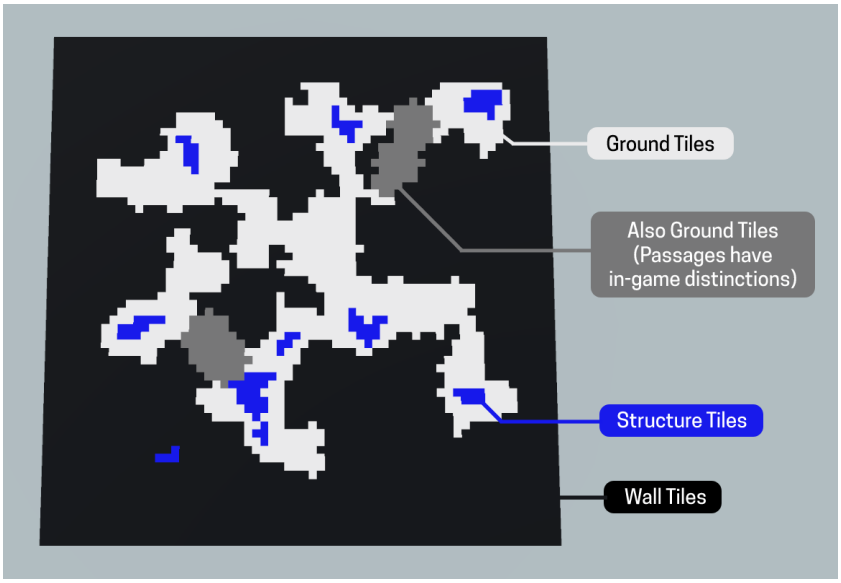


Each **Dungeon Map** can have its own properties, these are defined in the **Dungeon Shape** component.

A **Dungeon Map** is a diagram of an isolated and independent set of connected tiles.

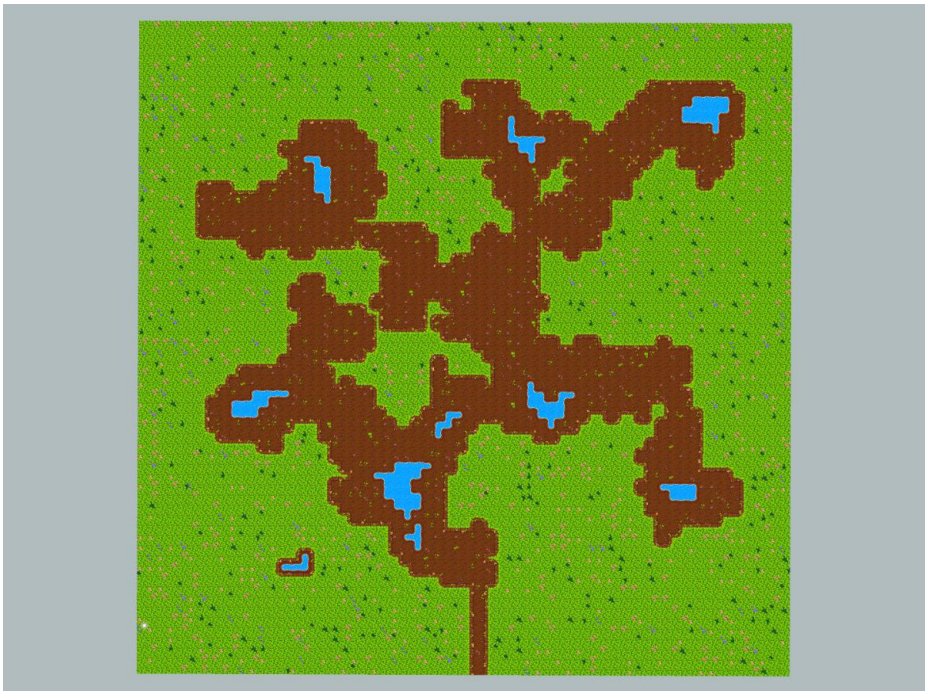
The arrangement created by these tiles, forms a fully traceable path composed of **3 different Tile Types**:

Wall Tile	Common obstruction tile, will border the entire map.
Ground Tile	Designated walkable tile, always linked to each other.
Structure Tile	**Optional tile**, designed as a ground decoration.

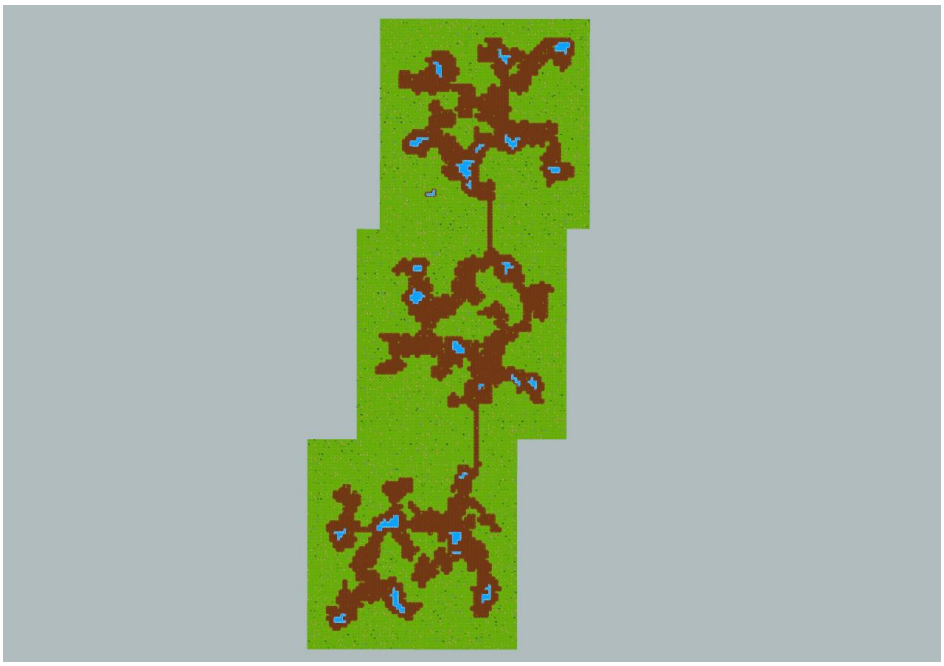


Each **Tile Type** can have its own properties, these are defined in the **Dungeon Brush** component.

In contrast, a **Dungeon Room** or **Dungeon Fragment** is the physical representation of a **Dungeon Map**.



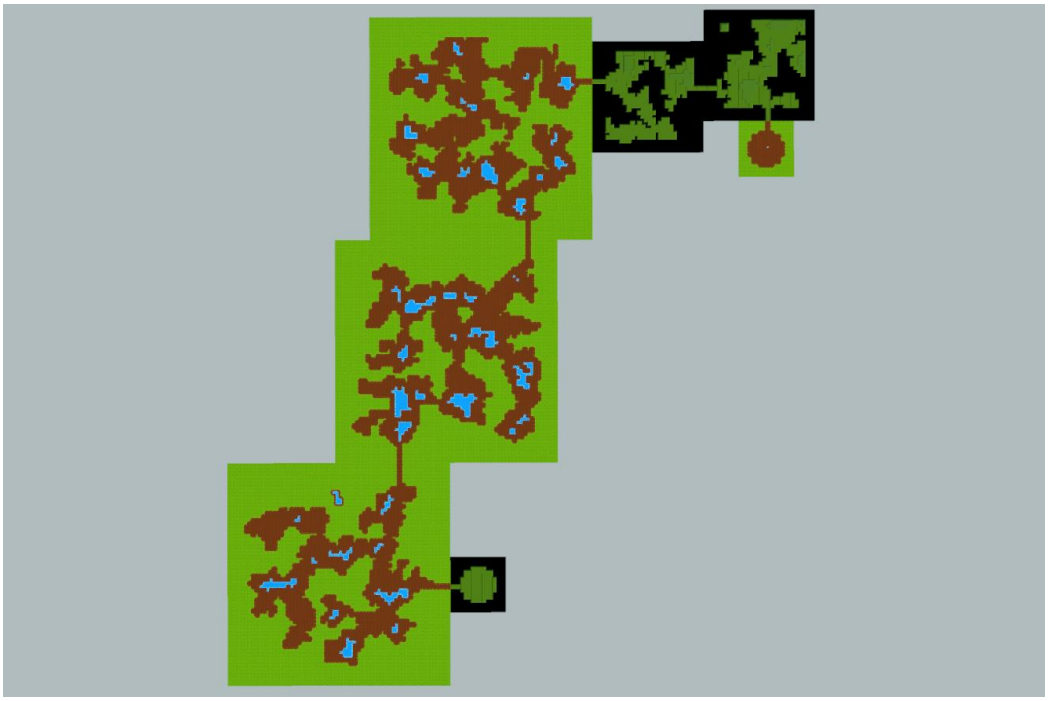
And a **Dungeon** is a successive and connected collection of **Dungeon Rooms.**



A **Dungeon** is defined by its individual **Components**. If you change the **Components**, you will change the **Dungeon**.

By design, *Advanced Dungeon Generator* **does not** allow you to concatenate **Dungeons**, however, *ADG* allows you to divide your **Dungeons** into different *segments*.

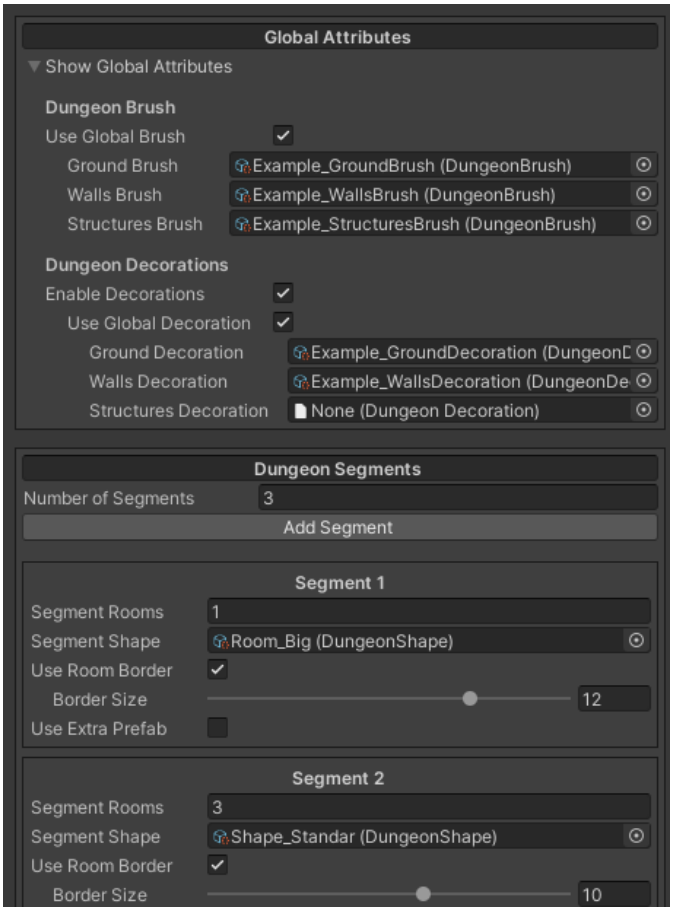
In a Dungeon, a **Dungeon Segment** is a group of **Dungeon Rooms** generated using the same **Components**.

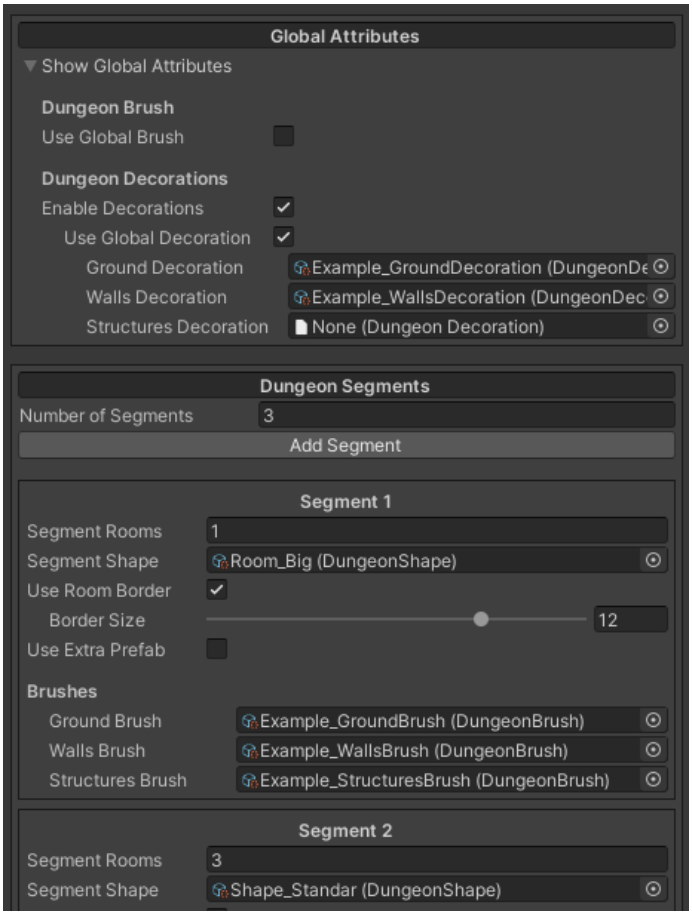


For clarification:

- You can assign **Global Components** to a segmented **Dungeon**.
- You can assign **Individual Components** to each **Dungeon Segment**.
- **You can also do both at the same time**, i.e., you can specify some Components to be the same in each Dungeon Segment while handling other Components individually.

*Here is an example of a **Dungeon** with several global and individual **Components**.*





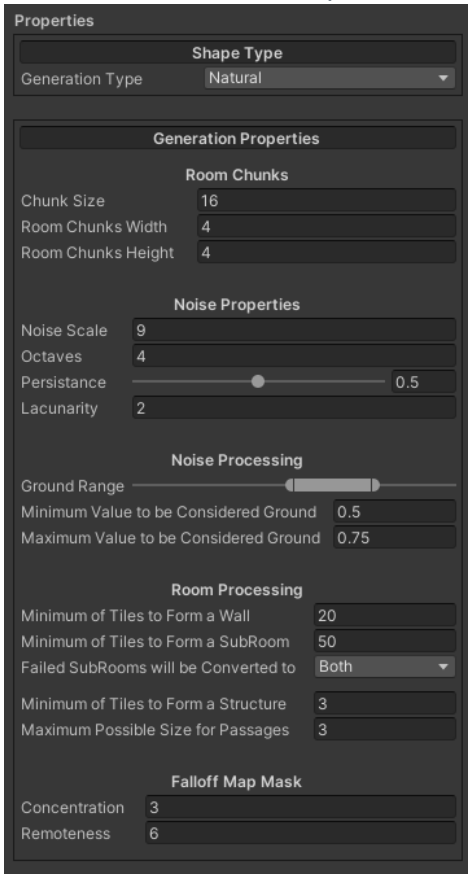
At the moment **there are only 3 main Dungeon Components**:

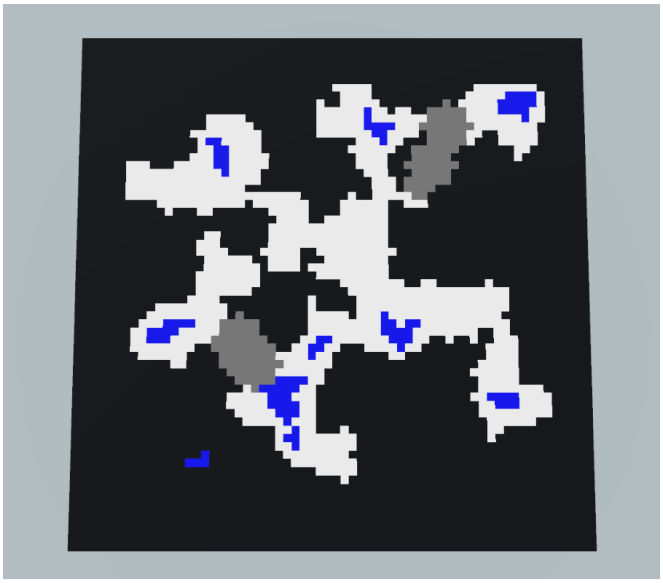
- **Dungeon Shapes**, which serve as a list of instructions to generate similar **Dungeon Maps**.
- **Dungeon Brushes,** that dictate the physical components of the Dungeon **Tile Types**.
- **Dungeon Decorations**, as an **optional** component that can create **non-interactable** objects that function as Dungeon ornaments

Dungeon Shape

The **Dungeon Shape** component is responsible for the creation of all **Dungeon Maps** requested by a **Dungeon** or **Dungeon Segment**, you can think of them as a set of successive instructions that generate similar **Dungeon Maps**.

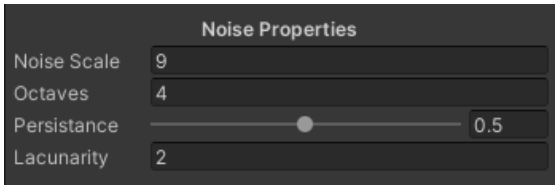
*Here is an example of a **Dungeon Map** generated by its **Dungeon Shape**.*

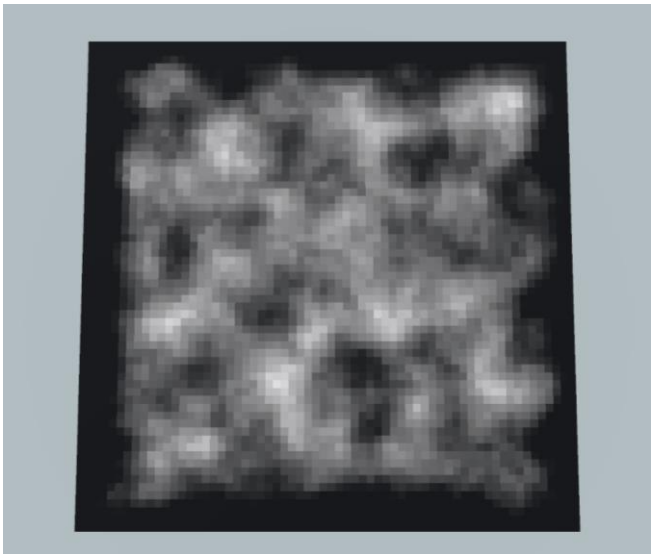




As stated above, a single **Dungeon Shape** is capable of generating several **Dungeon Maps**, this is possible because *ADG* uses a **Perlin Noise Map** as a map generation base.

*All **Dungeon Shapes** have a specific section for you to define the parameters of this noise.*

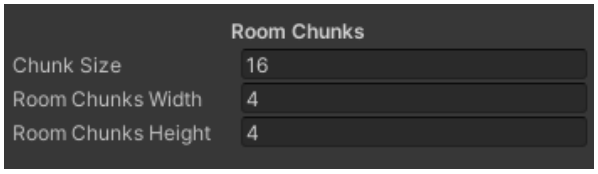




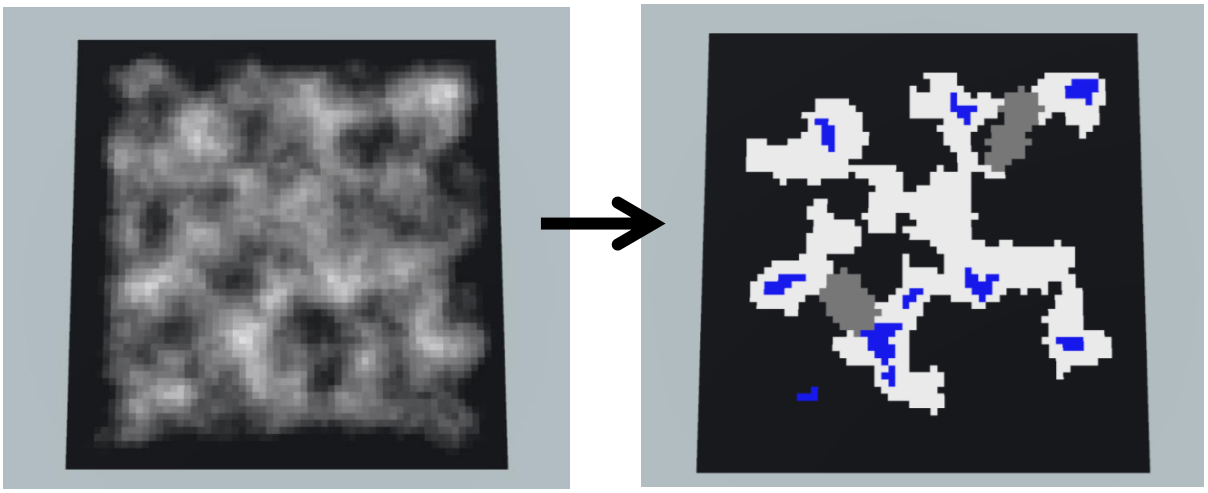
But before we continue with the noise processing, **how can we set the size of the map itself?**

Well, the dimensions of any **Dungeon Map** are strictly bound to how the **Dungeon Rooms** are generated at run-time, in order to improve game performance, *ADG* does not regenerate rooms block by block, but generates **chunks** of tiles at a time.

While you cannot directly define the size of your **Dungeon Rooms**, you can do it indirectly by defining in how many chunks the height and width of your rooms will be divided, and, the length of its square chunks.

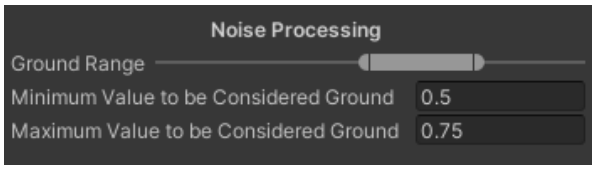


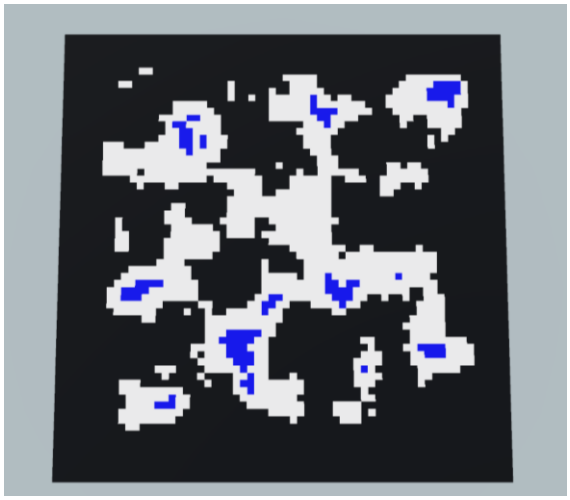
Now about the processing of the room itself, how do rooms go from noise to this?



Well, basically a **Perlin Noise Map** is a gradient map of values between 0 and 1, what the noise processing section does is to convert these intervals to functional **Tile Types**.

The **Ground Range** slider in the **Noise Processing section** represents the values that will be considered as **"Ground Tiles"** any value below this range will be considered as a **"Wal Tile"** and any value above this range will be considered as a **"Structure Tile".**

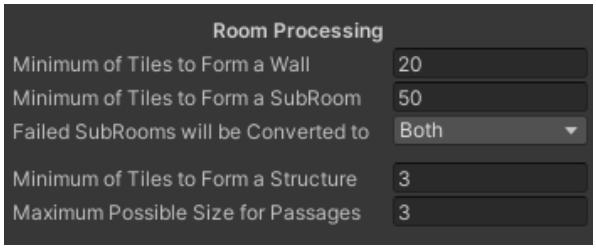


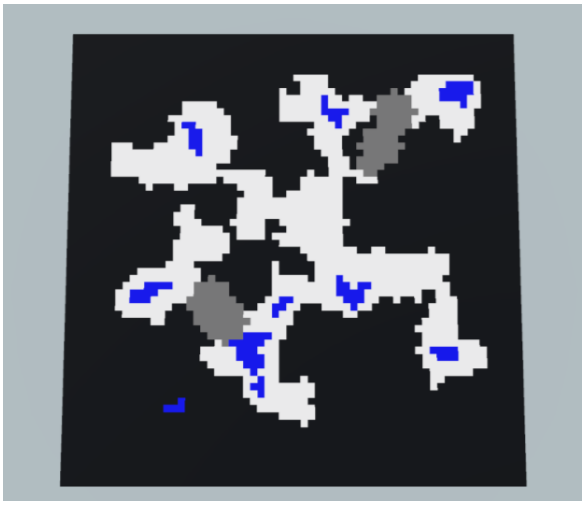


However, the room still doesn't look complete, and that's because **it still lacks a post-processing step.**

ADG will check the *connected tile clusters* and it will remove or change them according to the indications you define in the **Room Processing section**.

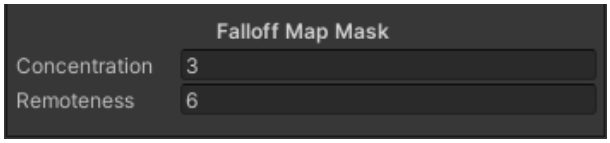
*ADG will also join all the **Ground Clusters** to make the rooms fully traceable.*

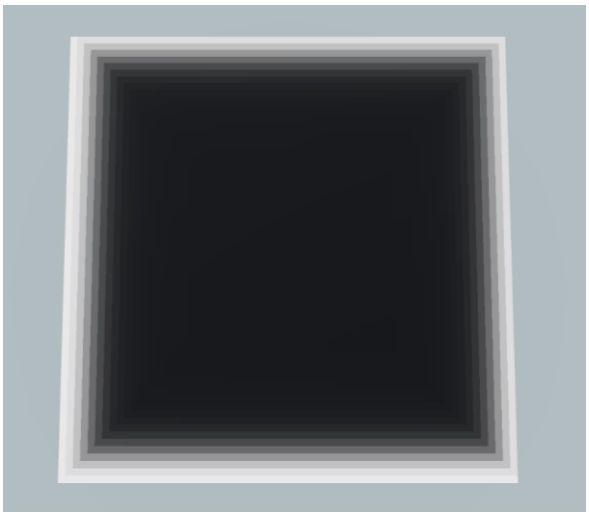




Finally, each of the **Dungeon Maps** will be blended with a **Falloff Map** to avoid open borders.

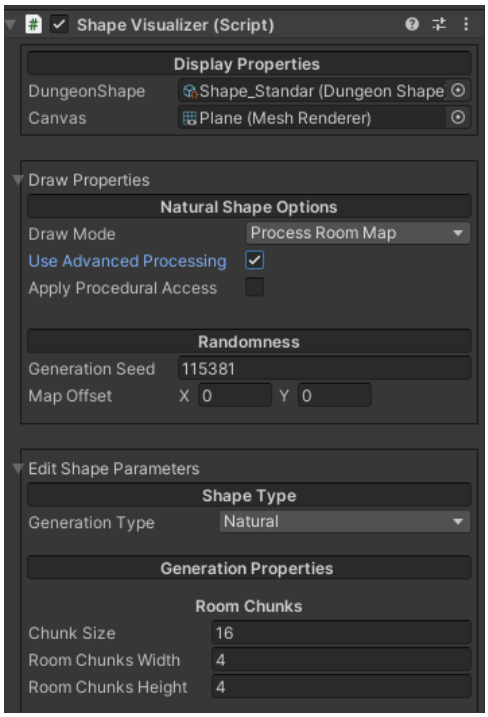
*You can modify the properties of this map in the **Falloff Map Mask** section.*

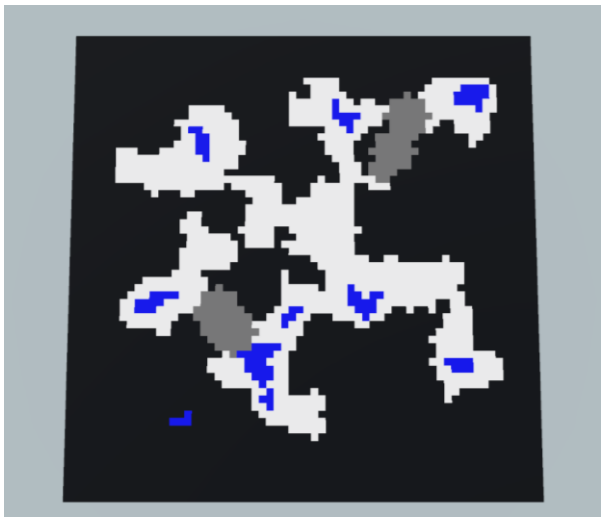




As you can see, having constant feedback on how changing parameters affects the creation of your **Dungeon Rooms** is quite useful, that's why *ADG* includes a sub-tool that lets you see your **Dungeon Shape** changes in real time!

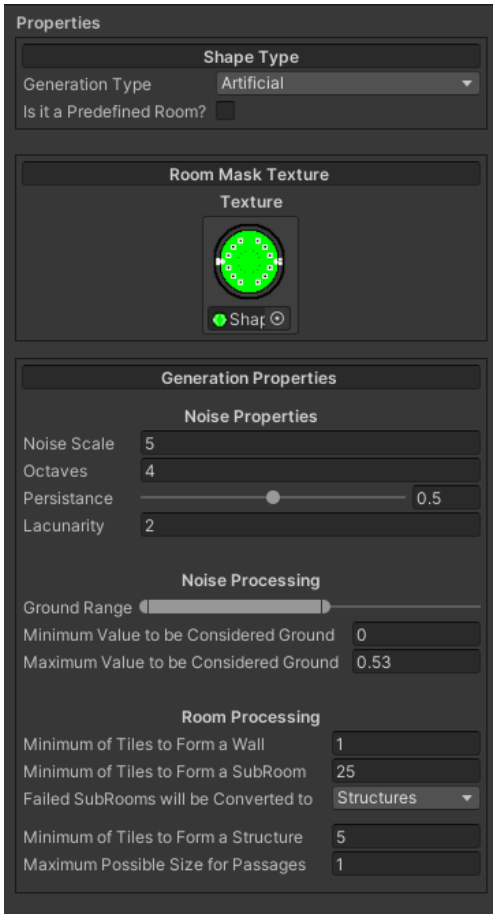
*You can see more in the **Shape Visualizer** section.*





---

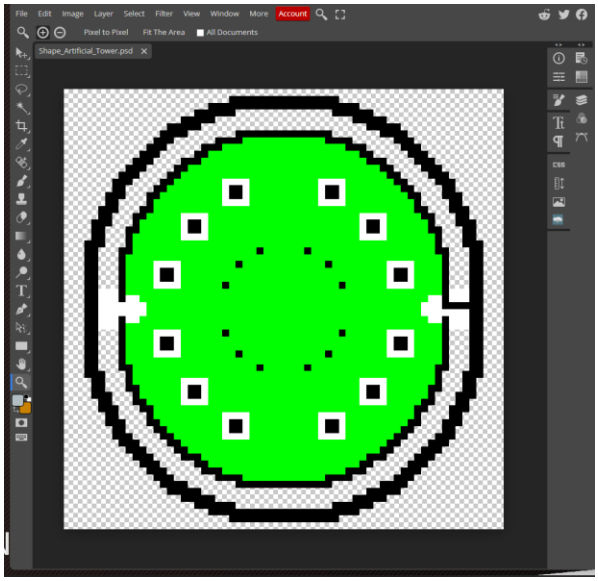
*On the other hand, there is still another type of **Dungeon Room Generation**.*



Unlike the **Natural Generation Type**, the **Artificial Generation Type** gives you total control over how your rooms will end up looking, letting you use an external image as a starting canvas.

ADG will read the colors of the image and interpret its pixels as **Tile Types** for **Dungeon Maps** creation.

In this example, only the green painted area will be used as accessible room space and the empty pixels will not create any in-game tiles.

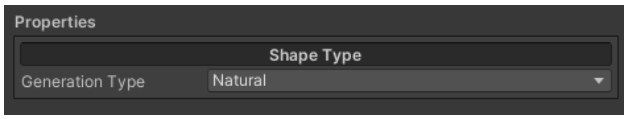


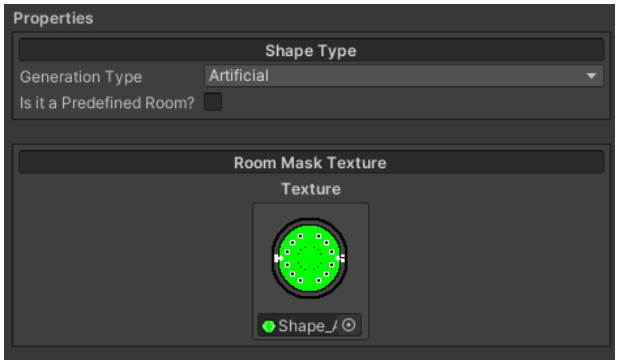
The next table could help you understand how the **Dungeon Shape** Script manages its **Tile Values**:

Tile Type	Description	Int Value	Read Color (For Image Import)	Write Color (As seen in the Shape Visualizer)
Ground	***The default walkable tile.*** By design, each dungeon chamber or dungeon sub-room has its own identifier, ranging from -1 to -inf. Their tiles inherit their identifier. The tiles linking these sub-rooms have a value of 0.	(-inf, -1) 0	White `#ffffff`	White (Grey for passages)
Walls	***The default obstruction tile.***	1	**Black** `#000000`	**Black**
Structures	***An optional multipurpose tile.***	2	**Blue** `#0000ff`	**Blue**
Empty Tile	***(Artificial generation only)*** **Specifies the absence of a tile.** No tile will be created in this position.	3	None (Alpha must be zero)	**Red**
Tile to Overwrite	***(Artificial generation only)*** These tiles function as a mask for the natural procedurally generated Dungeons.	4	**Green** `#00ff00`	None (Will be replaced by natural generation)

Dungeon Shape Properties Summary

Shape Type





Generation Type:
Defines the key properties needed for the creation of the Dungeon Rooms.

- **Natural:** A natural Dungeon Shape will use user modifications to alter a Perlin Noise Map in order to create the Dungeon Rooms.

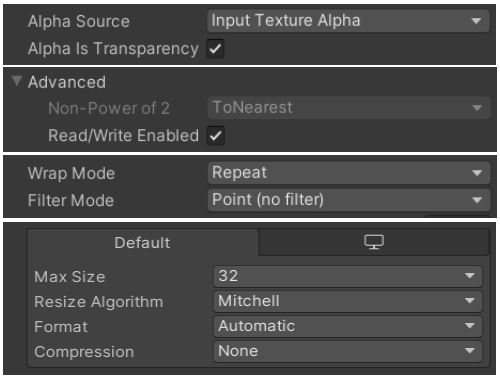
- **Artificial:** An artificial Dungeon Shape has nearly all the properties of a natural Dungeon Shape, but it can use a given image as an overlay mask for its natural generated Rooms.

  - **It's a predefined Room?**
    When enabled, the masking process will be ignored, and all the Dungeon Rooms generated by this Dungeon Shape will be copies of the given image.

  - **Room Mask Texture:**
    The image used in the masking process of an Artificial Shape.
    The mask image must have these properties:

    In the Image Editor:
    - Its dimensions must be multiples of 16, pixels; and both sides must have the same length.
    - The image has to follow the color rule from the above table.

    In the Unity Inspector:
    - Alpha is Transparency option must be checked.
    - Read/Write Enable advanced option must be checked.
    - Filter Mode option must be set to Point (no filter).
    - Max Size option must be set to the max size of the image.
    - Compression option must be set to None.

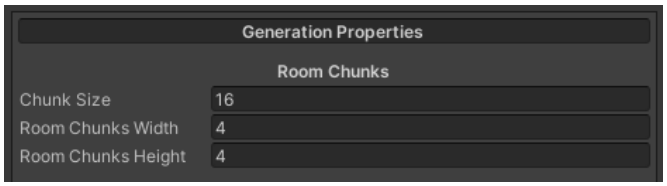


You can also apply the included **ADG_Map** preset!

Generation Properties

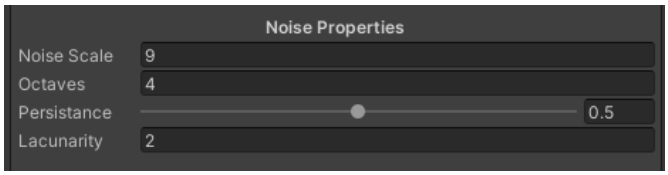
*You can change the set limits for the **Generation Properties** at the top of the **DungeonShape script**.*

Room Chunks (Natural Generation Only):
The Dungeon Room tiles will be grouped into Chunks to improve game performance. In an Artificial Shape, these values are implicit in the dimensions of the given image.



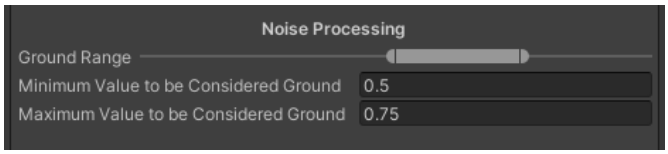
- **Chunk Size:** Determines the width and height, in tiles, of the individual chunks. *(Limited from 4 to 32)*
- **Room Chunks Width:** Determines the width, in chunks, of the Dungeon Rooms generated by this Dungeon Shape. *(Limited to a max of 8)*
- **Room Chunks Height:** Determines the height, in chunks, of the Dungeon Rooms generated by this Dungeon Shape. *(Limited to a max of 8)*

Noise Properties:
Set the properties of the base Perlin Noise Map from where all Dungeon Rooms are generated.



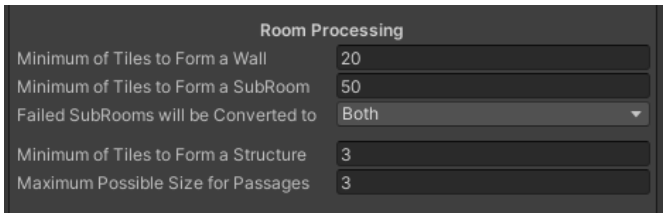
- **Noise Scale:** Set the distance at what the noise map is seen. *(Limited to a max of 128)*
- **Octaves:** Set the levels of detail the Perlin Noise Map will have. *(Limited to a max of 8)*
- **Persistence:** Set how much each octave contributes to the overall shape.
- **Lacunarity:** Set how much detail is added or removed at each octave. *(Limited from -4 to 4)*

Noise Processing:
The base Perlin Noise Map will be processed into an int matrix with only negative, 0, 1, or 2 values.



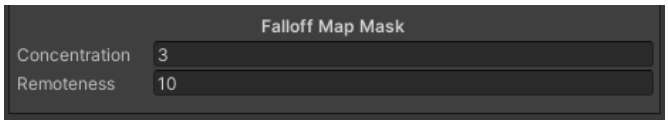
- **Ground Range:** This MinMax Slider represents the range of the 0 to 1 Perlin Noise Map values considered as Ground.
- **Minimum Value to be Considered Ground:** Any entrance of the 0 to 1 Perlin Noise Map below this value will be considered as a Wall.
- **Maximum Value to be Considered Ground:** Any entrance of the 0 to 1 Perlin Noise Map above this value will be considered as a Structure.

Room Processing:
The processed Perlin Noise Map will be post-processed to comply with the following Dungeon Room generation rules.



- **Minimum of Tiles to Form a Wall:** Minimum number of connected wall tiles required to be considered as an appropriate wall.
- **Minimum of Tiles to Form a SubRoom:** Minimum number of connected ground tiles required to be considered as an appropriate chamber or sub-room.
- **Failed SubRooms will be Converted to:** SubRooms with less than the minimum required tiles will be converted to this specification.
- **Minimum of Tiles to Form a Structure:** Minimum number of connected structure tiles required to be considered as an appropriate structure.
- **Maximum Possible Size for Passages:** Maximum size for the passages that will connect distant chambers or sub-rooms. *(Limited to a max of 16)*

Falloff Map Mask (Natural Generation Only):
Before processing the 0 to 1 Perlin Noise Map, a Fallow Map will mask it to avoid open borders. In an Artificial Shape, the user is responsible for setting the Room Borders in the given image.



- **Concentration:** Set how condensed the Fallow Map will be. *(Limited to a max of 128)*
- **Remoteness:** Set how remote to the center of the Room the Fallow Map will be. *(Limited to a max of 128)*

Dungeon Brush

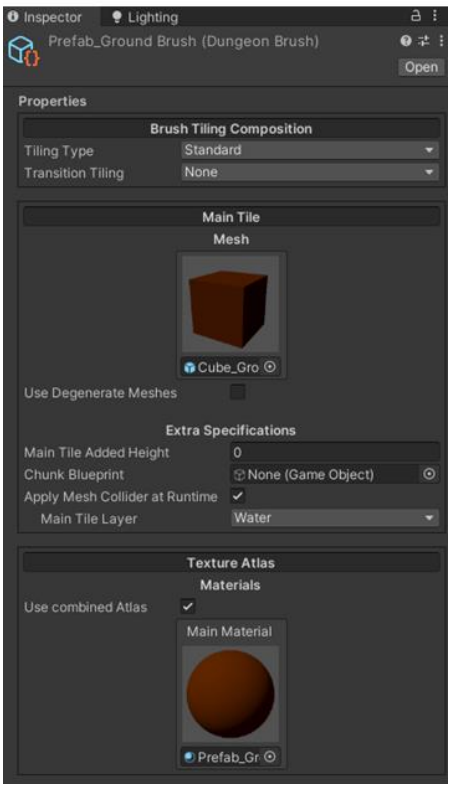
The **Dungeon Brush** component is responsible for the design of all **Dungeon Rooms Tiles** requested by a **Dungeon** or **Dungeon Segments**, contrary to its Shape counterpart, each **Tile Type** needs an assigned **Dungeon Brush.**

In other words, **Dungeon Brushes** are the containers of the physical properties of a concrete **Tile Type** *(Either Ground, Wall or Structure)*, and they determine the aesthetics and functionalities of the **Dungeon Rooms** defined by the **Dungeon Shapes**.

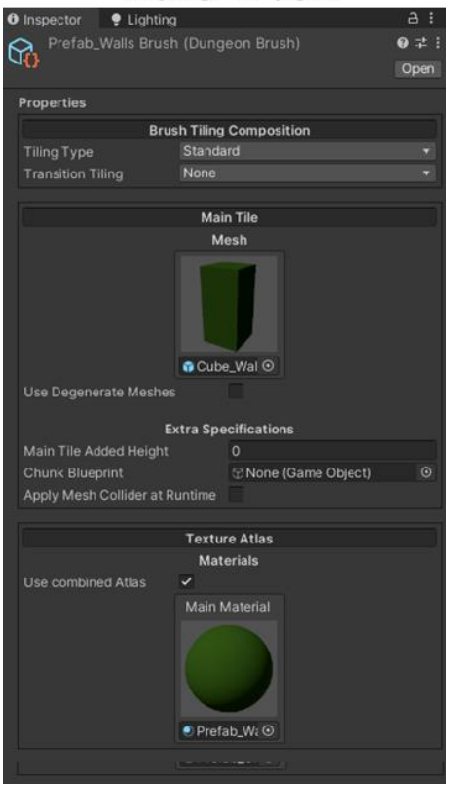
However, this doesn't mean that *ADG* will instantiate each given **Asset** as an individual **GameObject**, in order to improve performance and reduce in-game generation time, *ADG* will only use the referenced meshes of your assigned **GameObjects** and will deliberately ignore any other components that may be attached, including children.

For example, in the "***Prefab_Dungeon"* Demo**, there are 3 ***Standard Dungeon Brushes***, each **Dungeon Brush** has an assigned **Prefab** to be used in the creation of each of its **Tiles**.

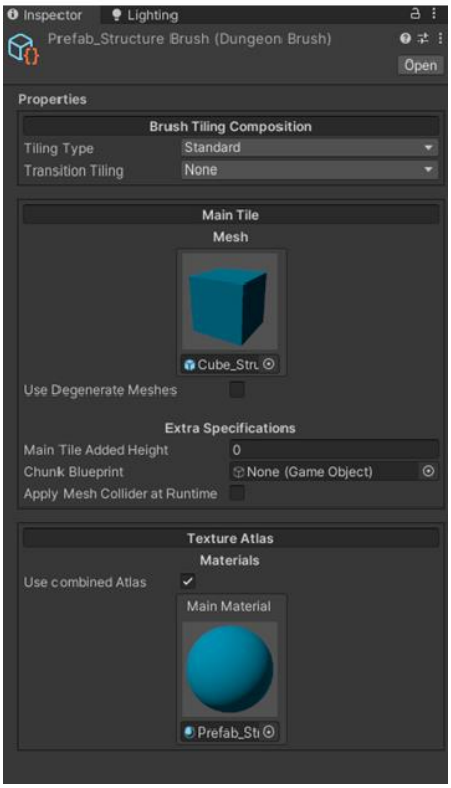
Ground Brush



Walls Brush

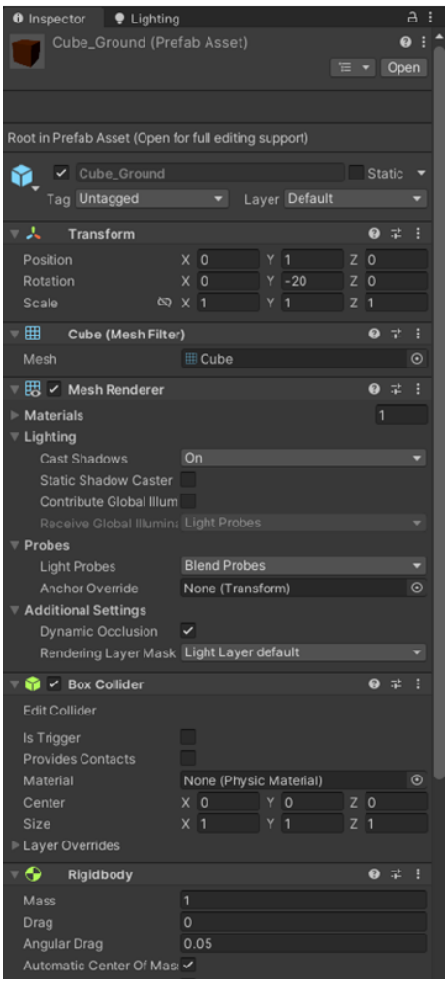


Structures Brush

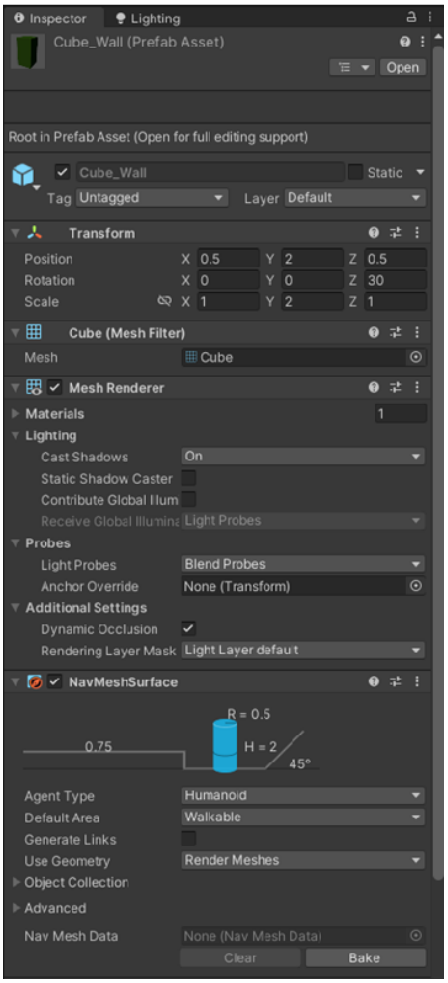


And each **Prefab** has its own specifications and components.

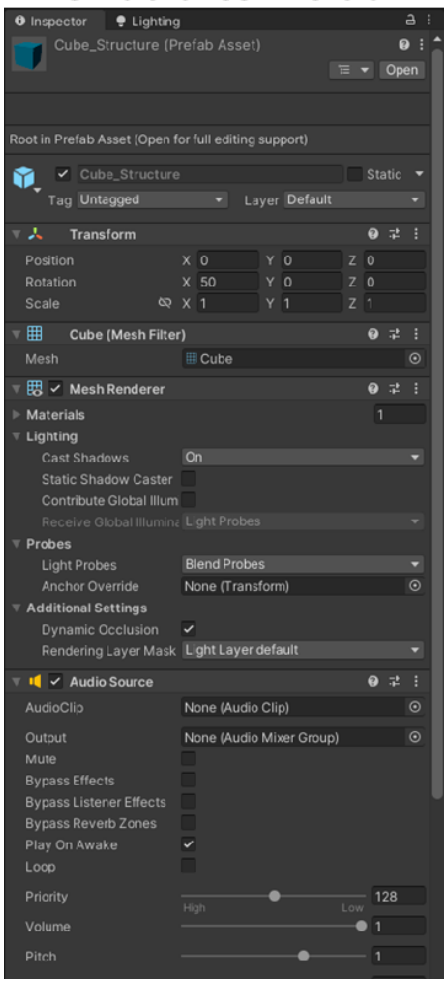
Ground Prefab



Walls Prefab



Structures Prefab



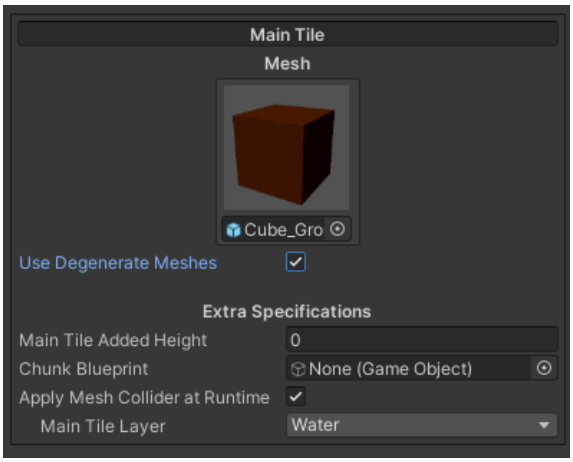
However, **NONE** of these components are taken into account when **Dungeons** are created at runtime. The **ONLY** component that matters is the Mesh Filter, **MESH**, and since all these **Prefabs** share the same mesh primitive, all Dungeon tiles are the same.

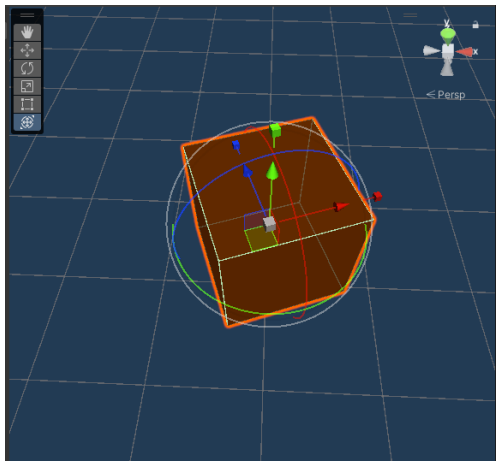
Please note that these design restrictions were made with game performance as the main priority and that, in theory, you can always modify your **Dungeon** meshes in external tools, such as Blender.

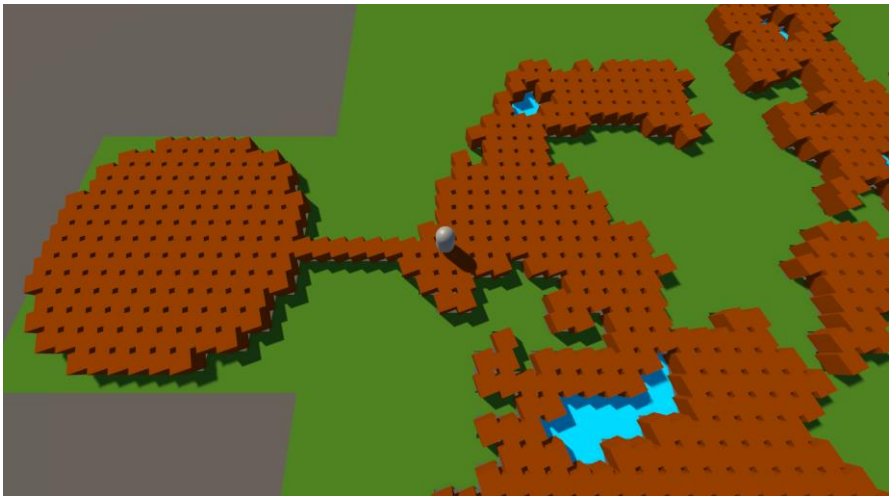
Even so *ADG* offers you some workarounds, in exchange for more processing resources, of course. For example:

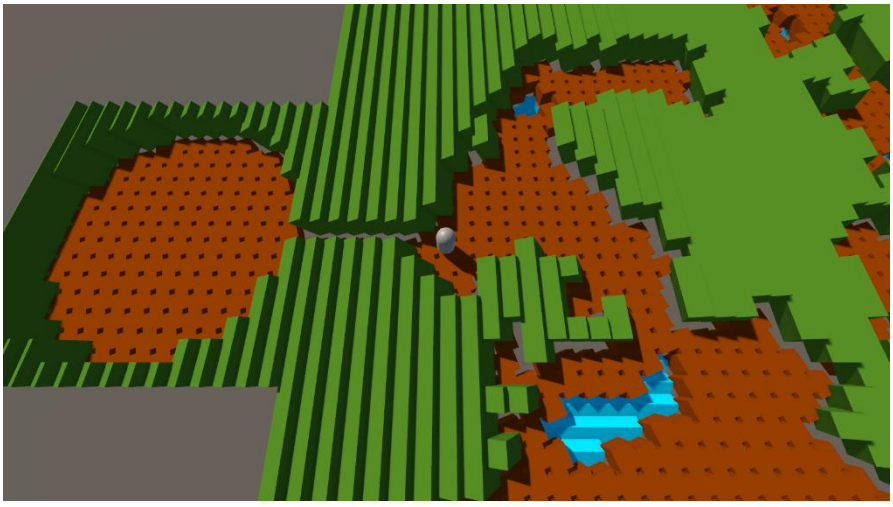
Degenerate Meshes

You can configure *ADG* to take into account the **Transform** component of your **Prefabs**, so you can modify your Meshes in the Unity editor itself.
Checking the **"Use degenerate meshes"** option in any **Dungeon Brush** will activate this processing.





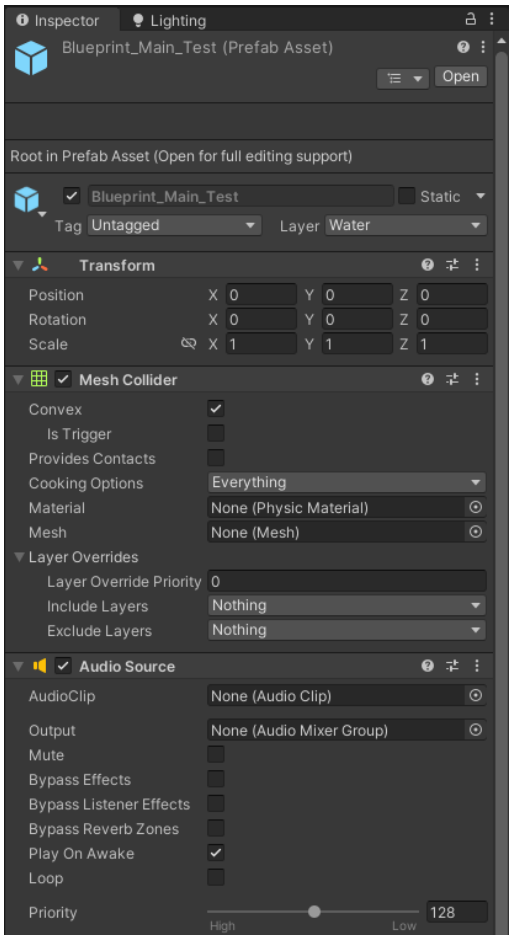


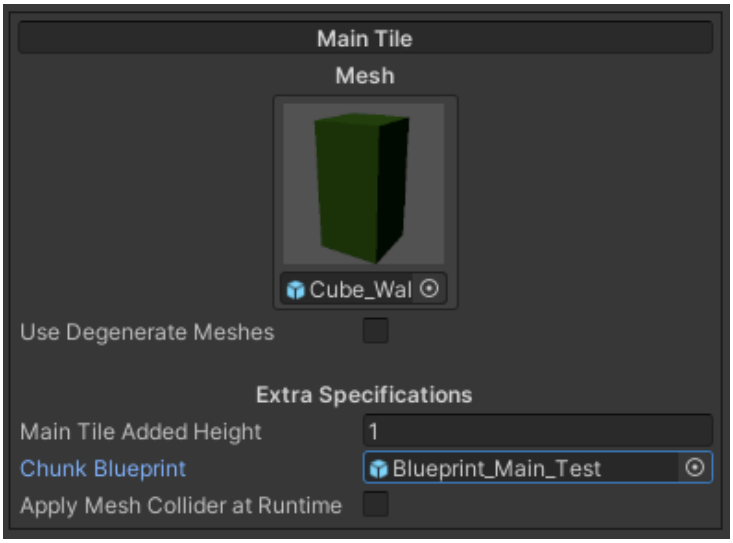


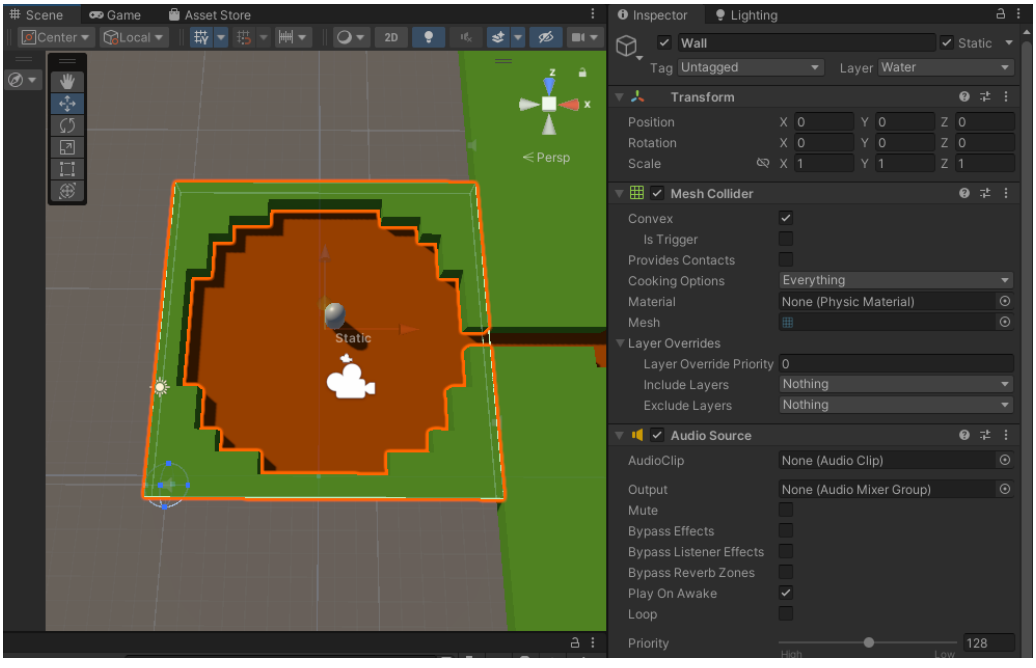
Blueprints

Degenerate Meshes will only take into account the **Transform component**, if you want to give your tile clusters any other components you will have to use **Blueprints.**

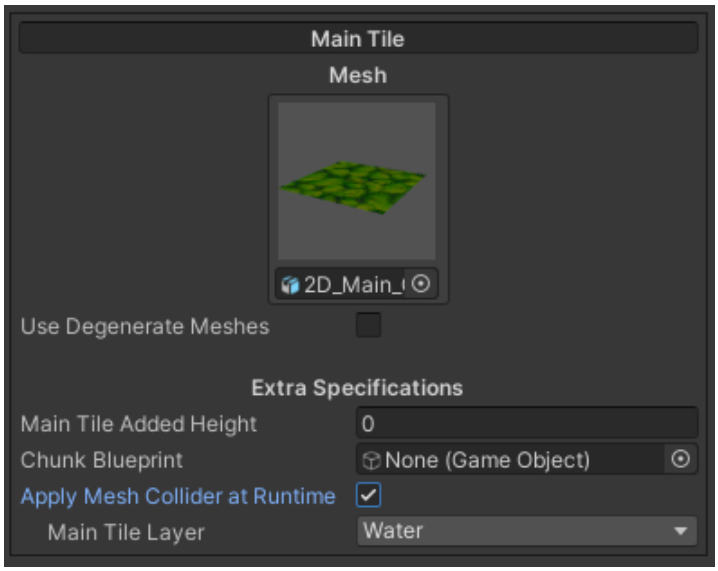
Blueprints are **Prefabs** that serve as Chuck templates. At runtime, all components of the given **Prefab** will be copied into their respective tile clusters.

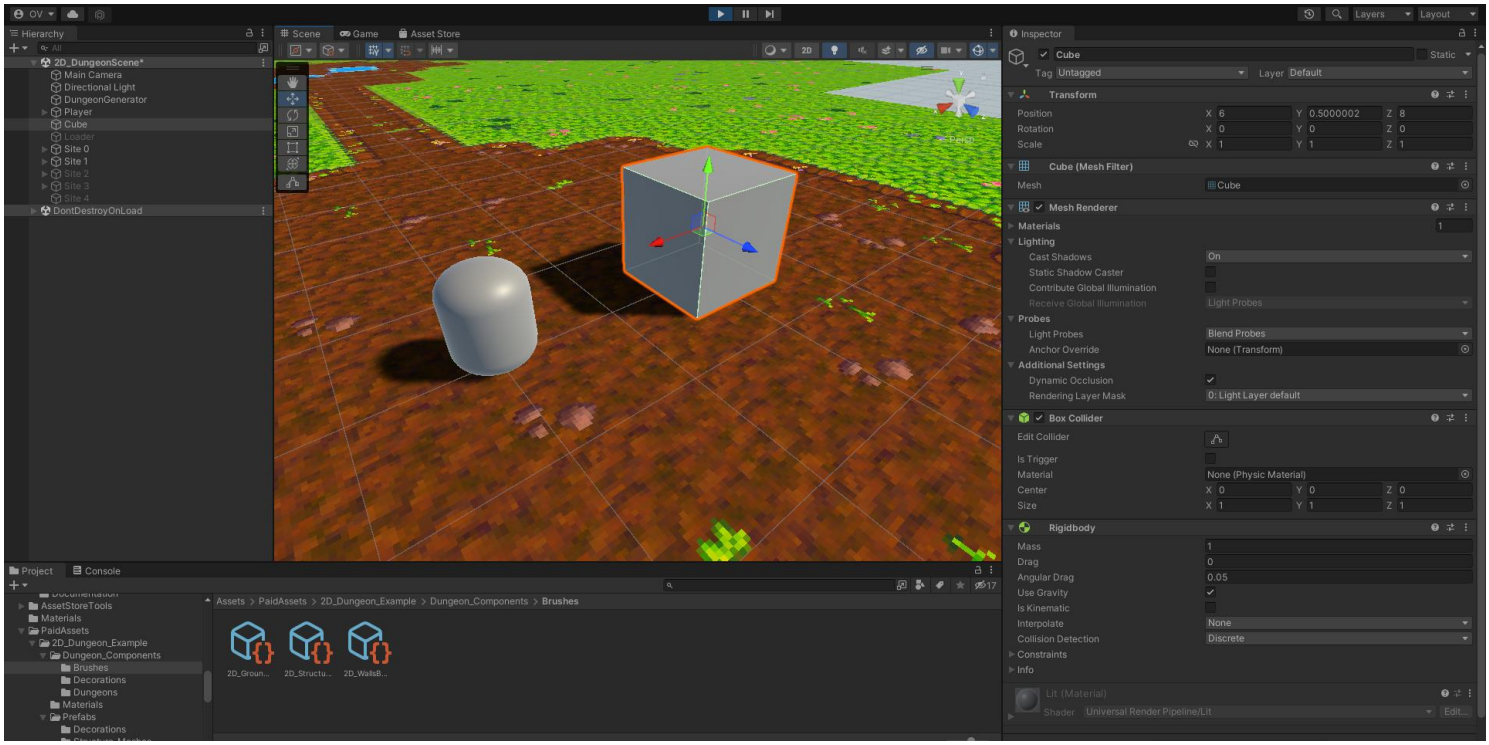






If you choose to ignore all of these options, *ADG* still allows you to give ***some essential properties*** to the tile clusters themselves, for example you can designate a **Mesh Collider component** as well as an **Object Layer** in your **Dungeon Brush**, this will provide your chunks with collisions, so your **Rigidbodies** don't fall through the ground.

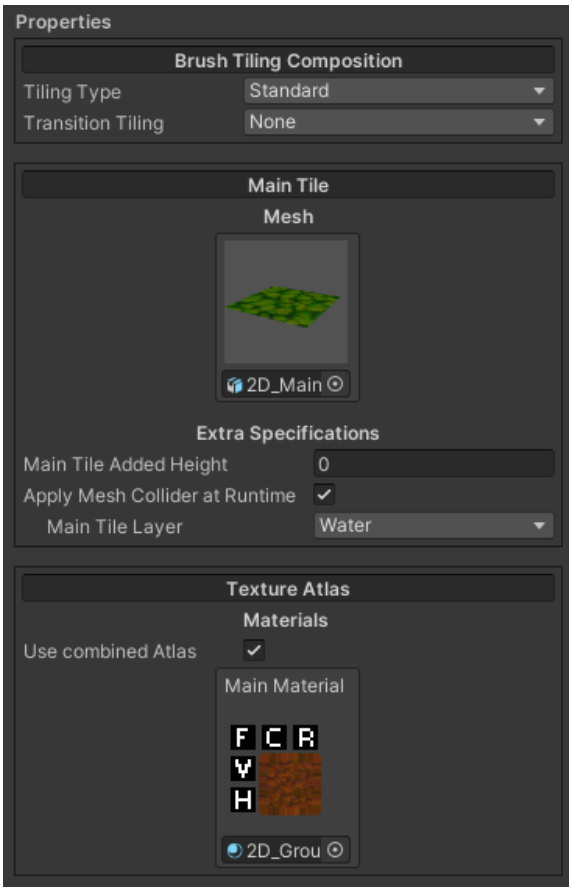


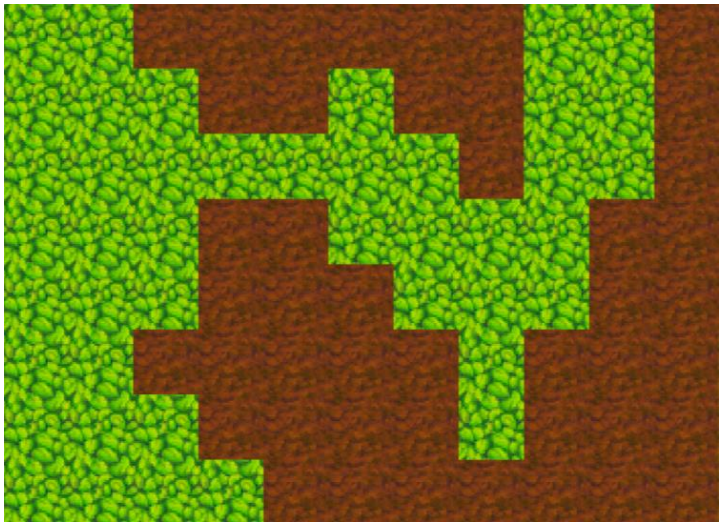


Dungeon Brush Types

Standard Tiling

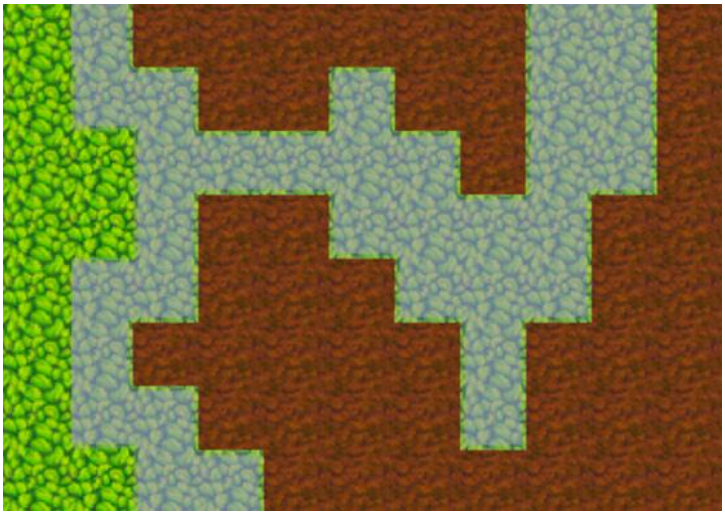
The **Standard Tiling** is the most basic type of **Dungeon Brush**, in it you just have to define a single Mesh and Material that will be used in each square meter tile.





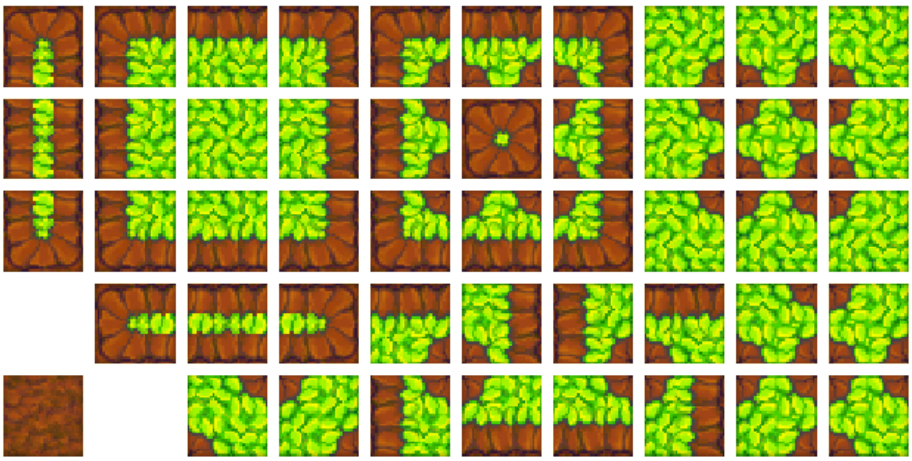
Is the most basic type of **Dungeon Brush** because it does not take **Transition Tile**s into account:

Transition Tiles are the tiles that are in contact with tiles from different **Tile Types**.



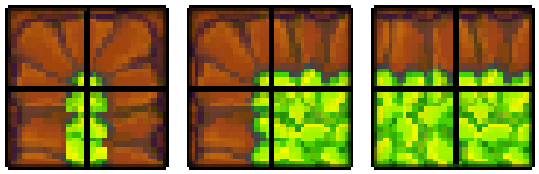
*To have a smooth transition between different **Tile Types** we need a way to manipulate these tiles*.

In order to create these **Transition Tiles**, *ADG* uses a slight modification of the **Blob Tileset.**

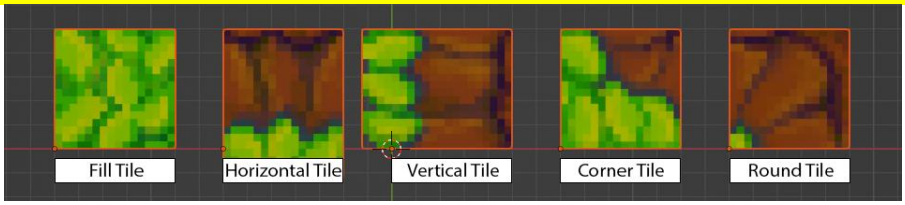


*If you have previously worked with **Tile Maps**, you are already familiar with the **Blob Tileset**.*

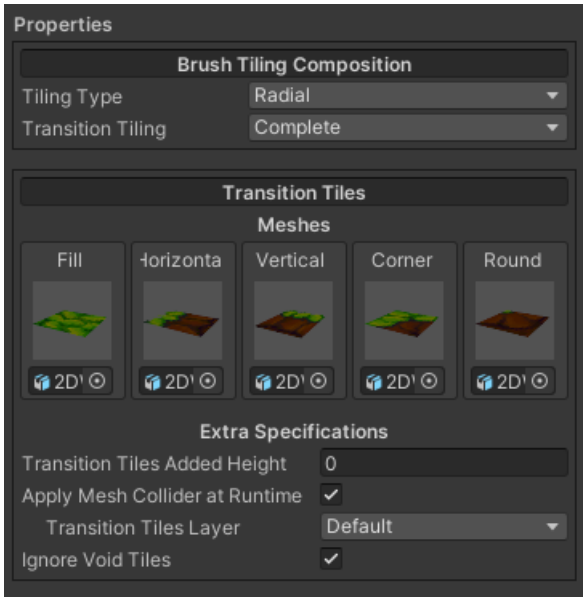
By design, *ADG* takes advantage of the similarity between tiles in the standard **Blob Tile Map**, dividing the tile formation into quarters.

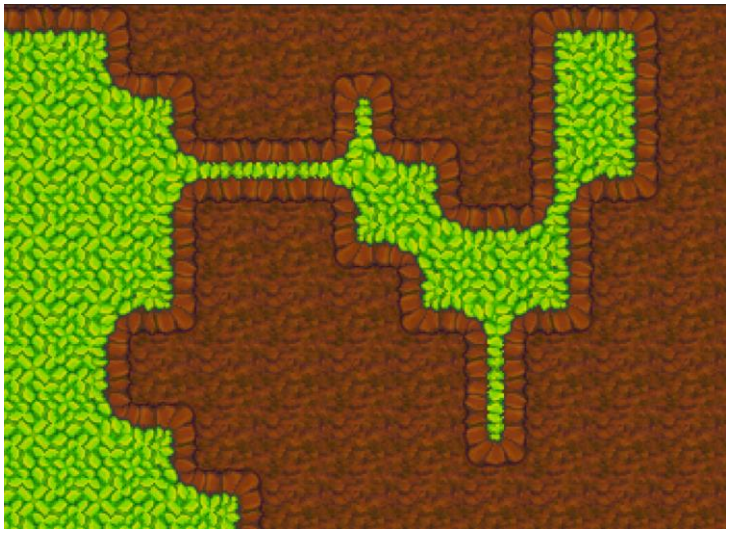


These are the ***"Sub-Tiles"*** that will be rotated and combined in order to create all the other tiles.



In *ADG*, this type of tiling is called **Radial Tiling.**



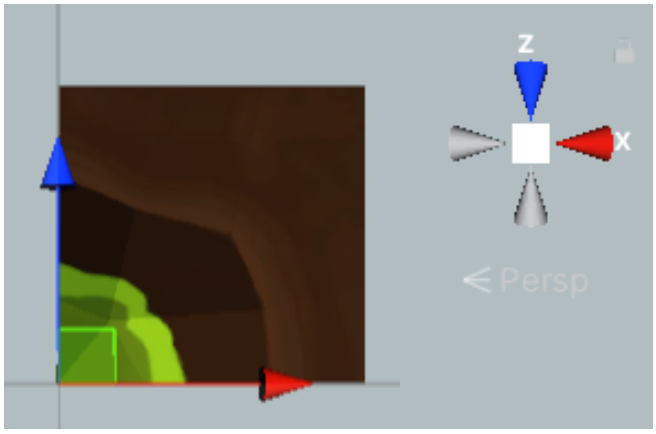


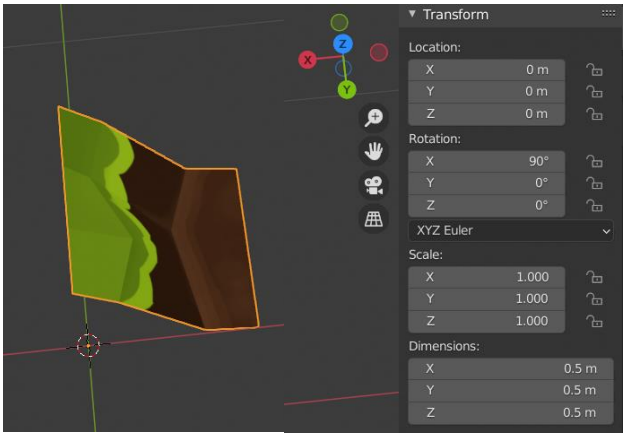
Radial Tiling

In the **Radial Tilling** every single tile is constructed by rotating and merging these 5 sub-tiles:

To accomplish this, the given meshes must comply with these specifications:

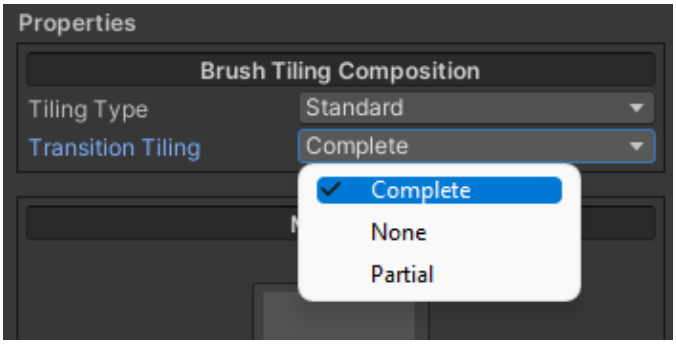
- The pivot point of the mesh must be in (0, 0, 0). *(Can be omitted on Degenerate Meshes)*
- The **actual mesh** should only be in the Unity (+X, +Z) quarter.
- The mesh X, Z dimensions should be (0.5 m x 0.5 m) maximum.
- All the meshes had to have radial symmetry. *(Can be omitted on Degenerate Meshes)*





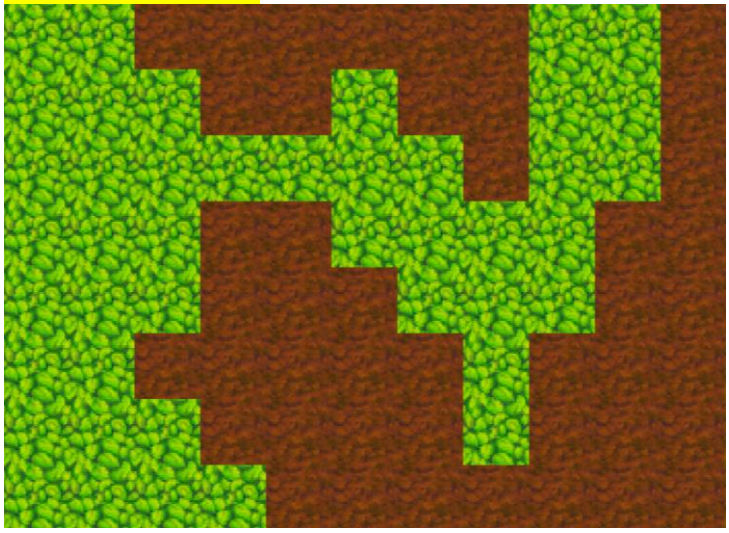
As stated above, these sub-tiles will replace all tile generation, which may be counterproductive as they are only necessary for the creation of **Transition Tiles**, that added to the relative inflexibility of the radial symmetry may leave you wondering, is there an option that gives me the best of both worlds?

Well, in fact you can decide what type of **Transition Tilling** will be use.

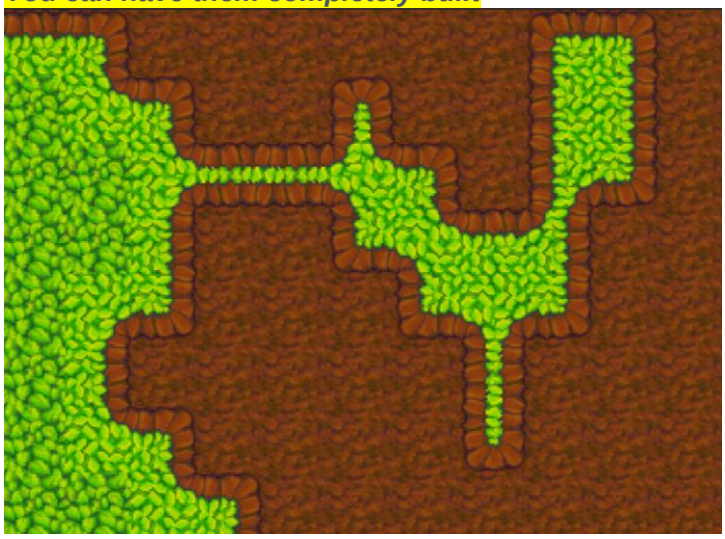


The **Transition Tiling option** will only affect how the **Transition Tiles** will be created.

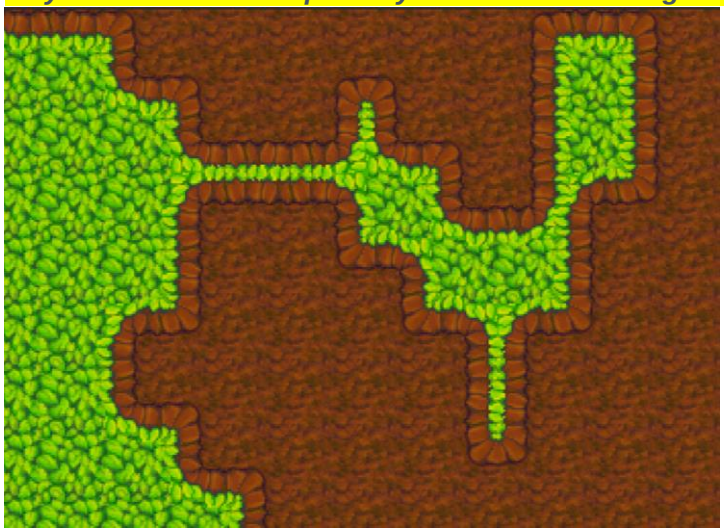
You can have none

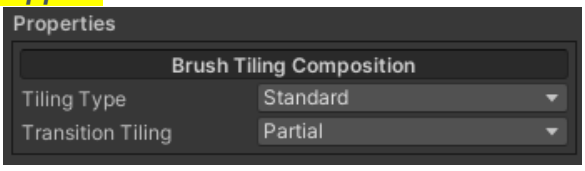


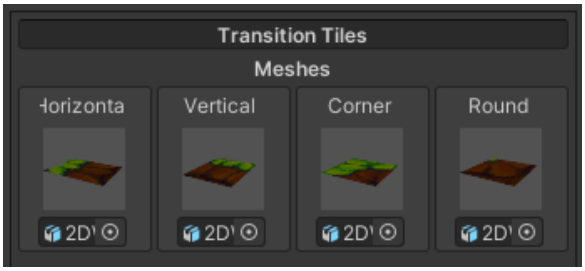
You can have them completely built



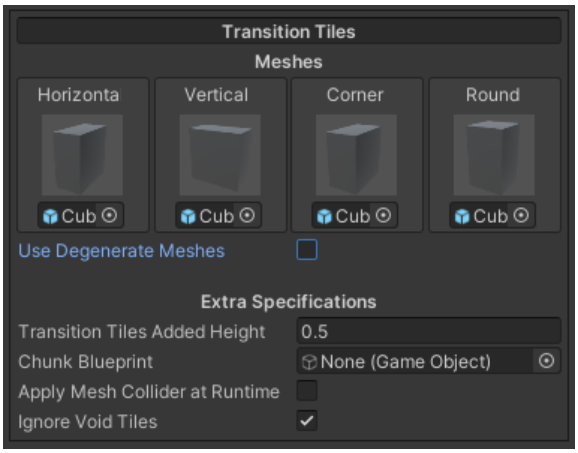
Or you can have them partially created while being overlapped.

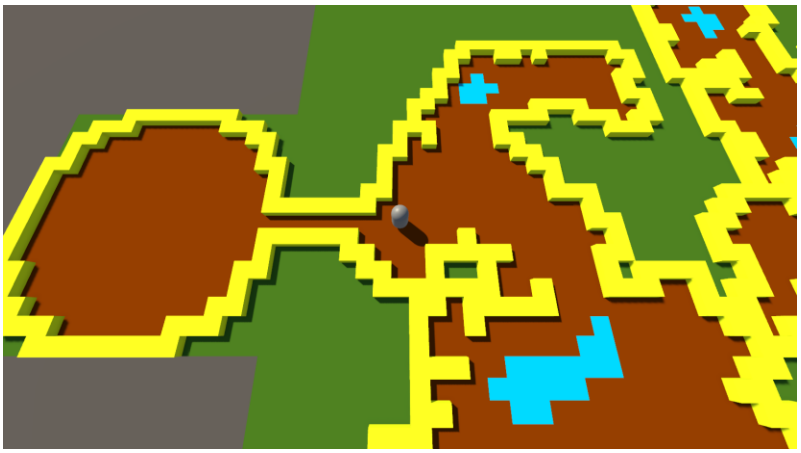


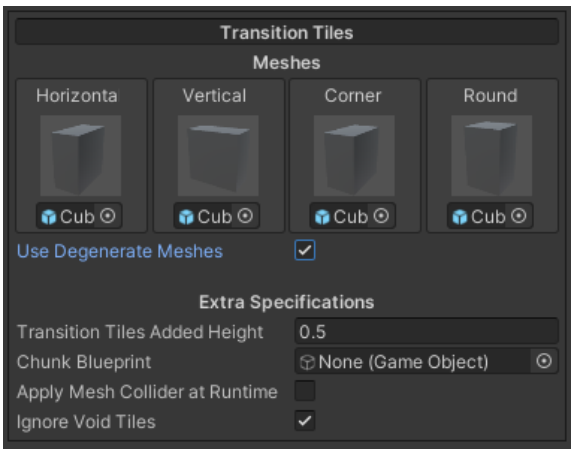


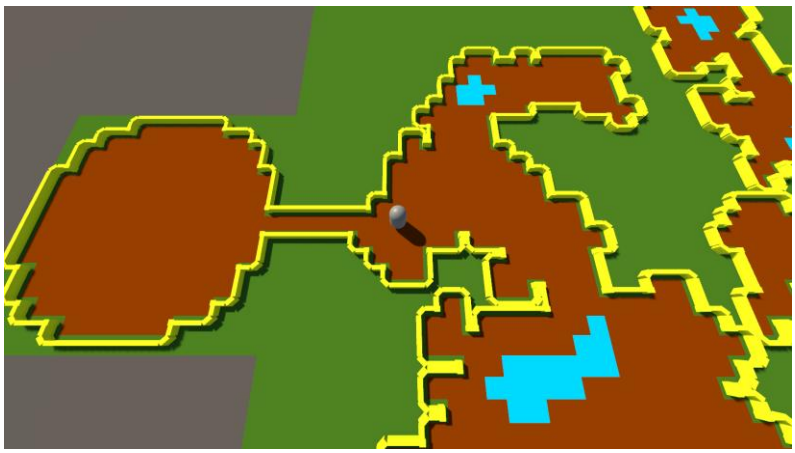


As a reminder, **Transition Tiles** also benefit from **Degenerate Meshes**.

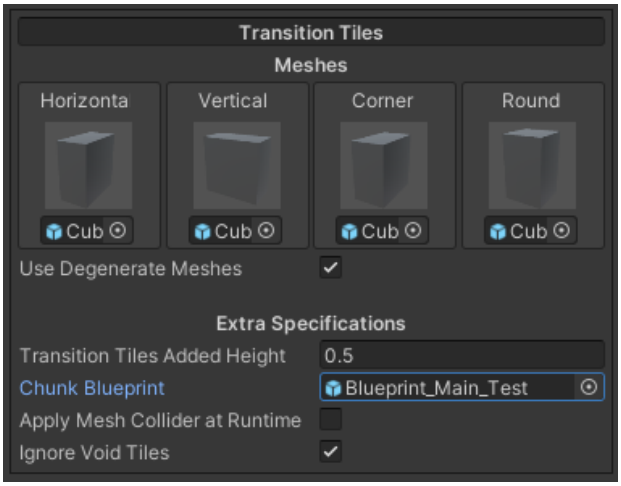


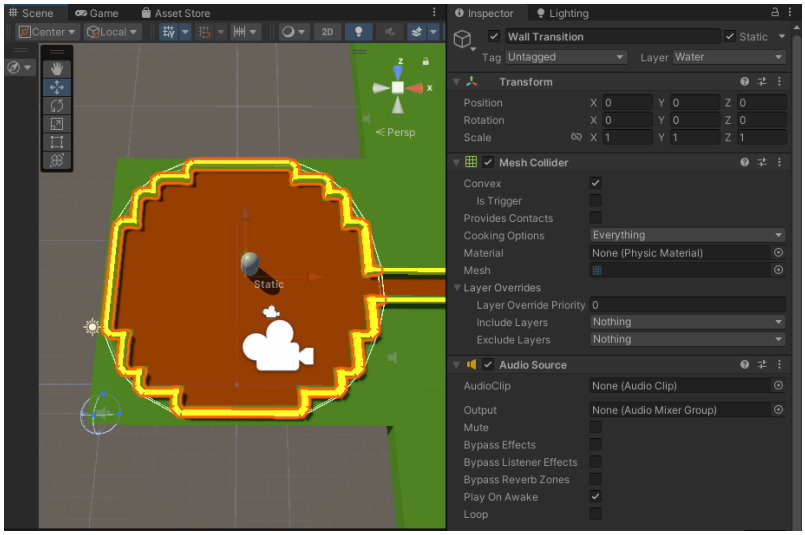






And **Blueprints**.

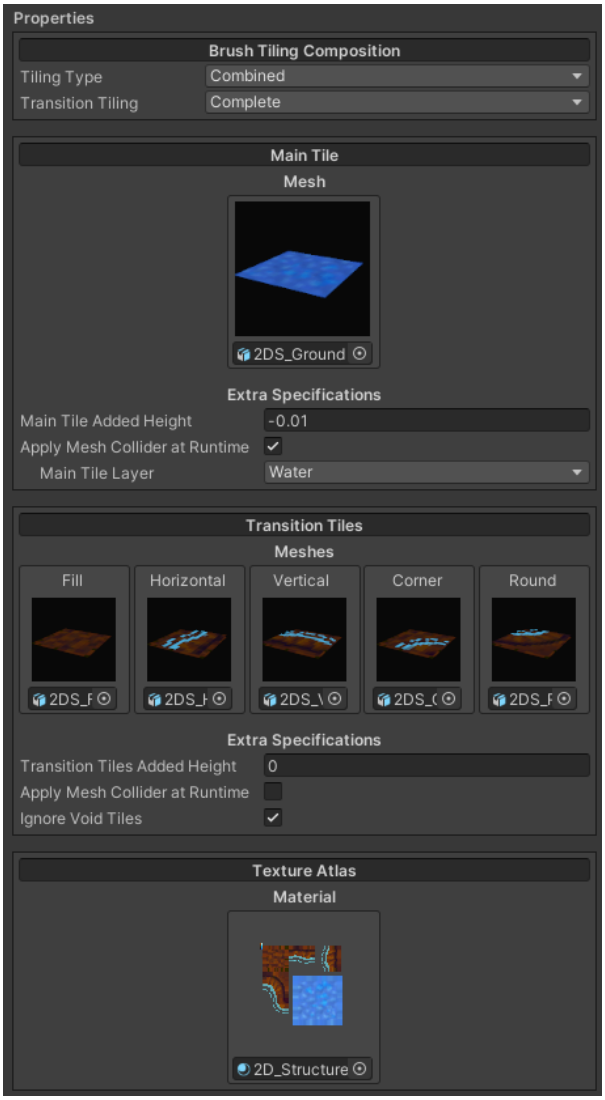


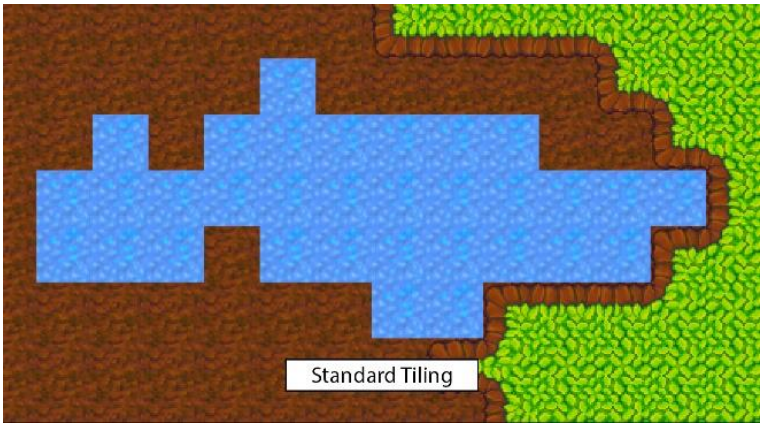


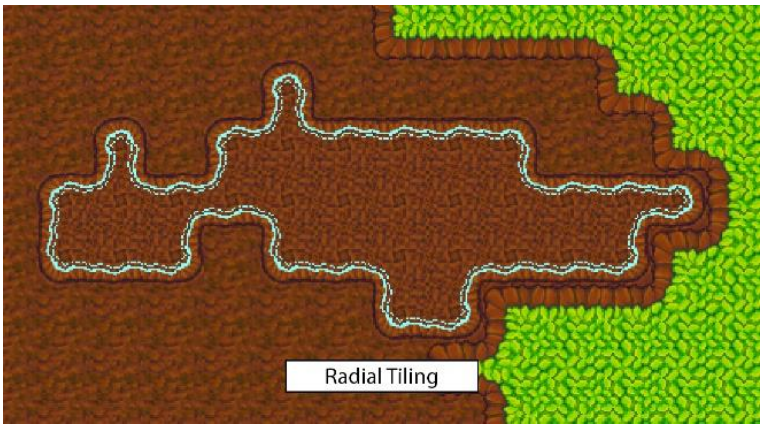
See more on the "***Prefab_Dungeon***" **Demo**

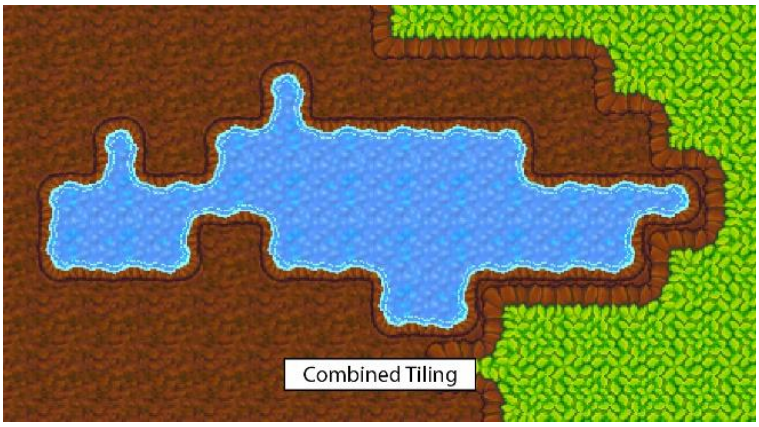
Combined Tiling

Now, the **Transition Tilling** isn't everything, because there may be some cases where you need **both Tilling Types** at the same time, mostly for **Structure Tiles**. In that case you can use the **Combined Tiling** option to create the Standard and the Radial Tiling at the same tile.

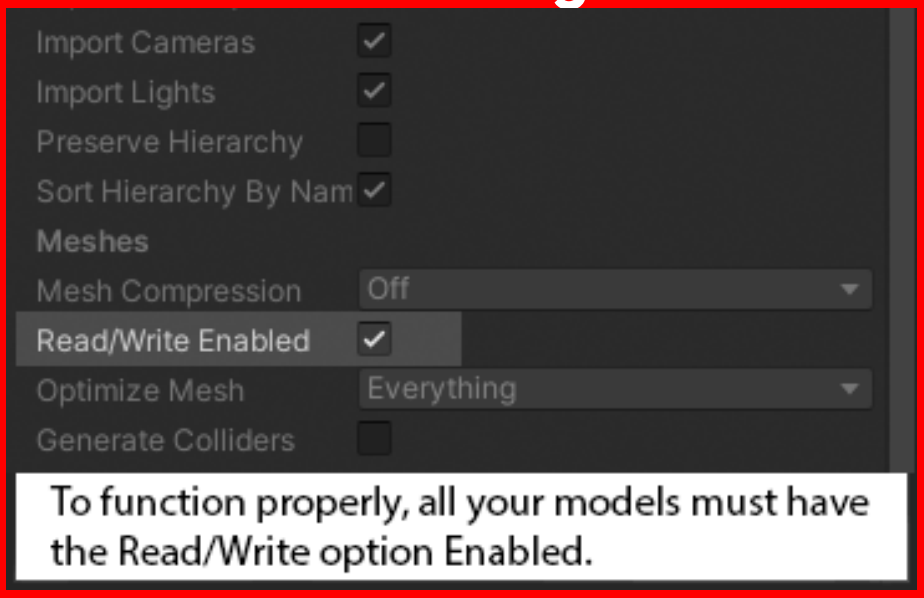




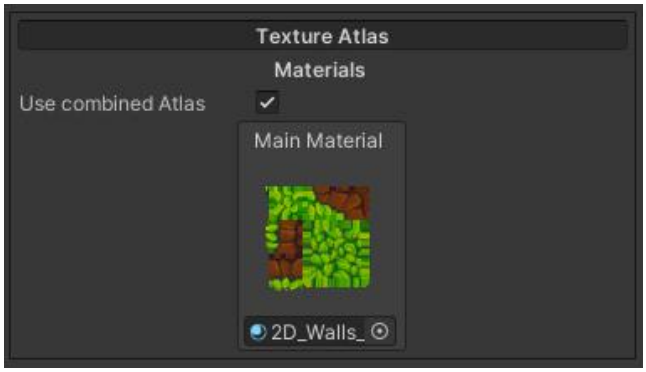




Warning!



Finally, the **Texture Atlas section** will define the **Material** to be applied to each of these Dungeon Brush tile clusters, because as stated at the start, *ADG* will only use the referenced meshes in your given **Assets** and will deliberately ignore any other components that may be attached, and a **Material** is an object **Component**.



I recommend you the **Use combined Atlas** option, but you can also assign one material to your **Main Tile** and another one to your **Transition Tiles**.

Dungeon Brush Properties Summary

Brush Tiling Composition

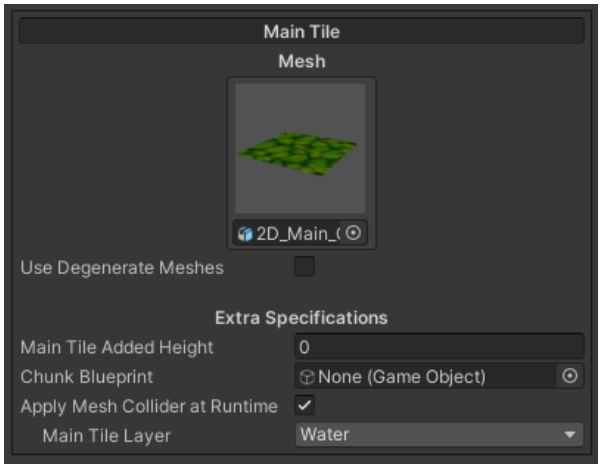
Tiling Type:
Defines how the Dungeon Brush tiles will be constructed.

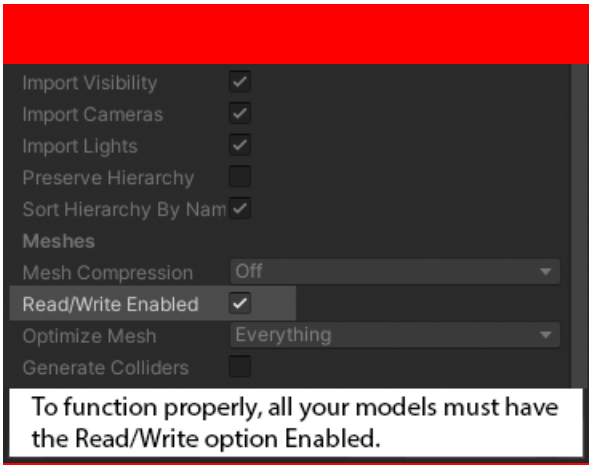
- **Standard:** The mesh of the Main Tile will be instantiated at each position of its type tiles.
- **Radial:** Instead of using a single object, each tile will be constructed by joining and rearranging common sub-meshes.
- **Combined:** The Standard and Radial tiling will be applied at the same time.

Transition Tiling:
Defines how the Dungeon Brush Transition Tiles will be constructed.

- **Complete:** All tiles in the Transition Area will be replaced by a sub-tile assembly, you can set these common blocks in the Transition Tiles section.
- **None:** There will not be any distinction between Flat Tiles and Transition Tiles.
- **Partial:** Instead of being replaced; the sub-tile assembly will be added on top of all the Transition Area tiles.

Main Tile





Mesh:

- The mesh to be instantiated at each square tile position.

Use Degenerate Meshes:

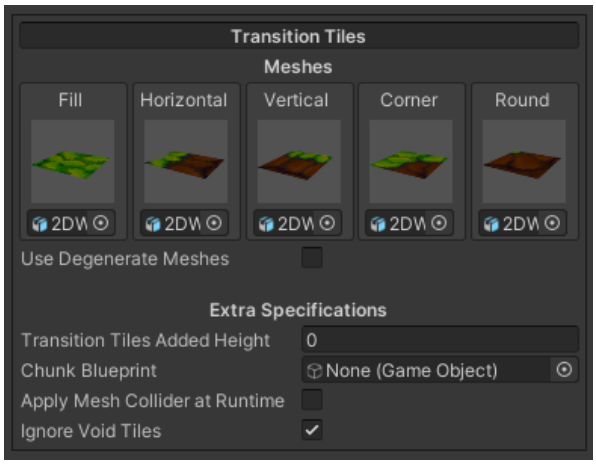
- When enabled, ADG will also take into account the Transform of the given Prefabs at Tile generation. This option is heavily resource-demanding, use it only for testing purposes.

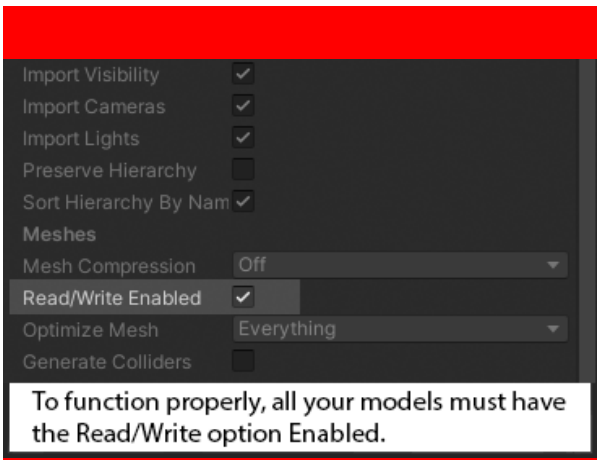
Extra Specifications:

- **Main Tile Added Height:** Extra height to be added to the Main tiles.
- **Chunk Blueprint:** All components of this object will be copied into each Main Tile Chunk. This option is heavily resource-demanding, use it only when absolutely necessary.
- **Apply Mesh Collider at Runtime:** If checked, a Mesh Collider will be applied to the Main tiles once they are generated.
- **Main Tile Layer:** Layer to be applied to the Main tiles.

Transition Tiles

A set of meshes that will be grouped and modified in order to build tiles.





Meshes:

- **Fill:** Solid Tile.
- **Horizontal:** Horizontal Divide Tile.
- **Vertical:** Vertical Divide Tile.
- **Corner:** Inner Curve Tile.
- **Round:** Outer Curve Tile.

Use Degenerate Meshes:

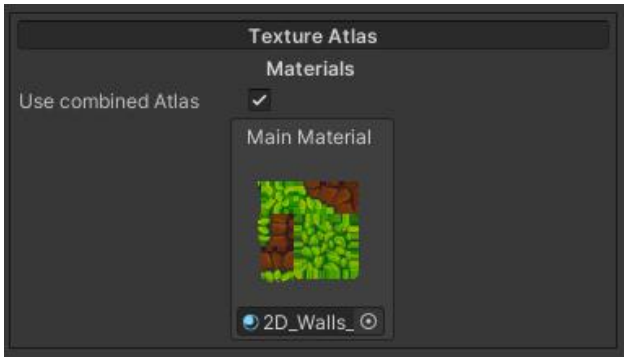
- When enabled, ADG will also take into account the Transform of the given Prefabs at Tile generation. This option is heavily resource-demanding, use it only for testing purposes.

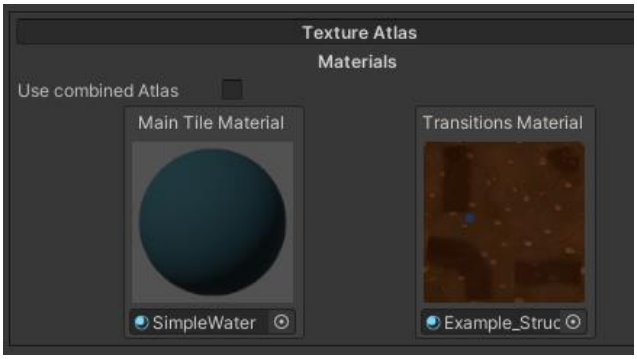
Extra Specifications:

- **Transition Tiles Added Height:** Extra height to be added to the Transition tiles.
- **Chunk Blueprint:** All components of this object will be copied into each Main Tile Chunk. This option is heavily resource-demanding, use it only when absolutely necessary.
- **Apply Mesh Collider at Runtime:** If checked, a Mesh Collider will be applied to the Transition tiles once they are generated.
- **Transition Tiles Layer:** Layer to be applied to the Transition tiles.
- **Ignore Void Tiles:** If checked, this Brush will not create Transition tiles around Empty tiles, extremely useful in Artificial Shapes.

Texture Atlas

The material that will be applied to all the meshes of this Dungeon Brush.





Materials:

- **Use combined Atlas:** A single material will be used for both the Main Tile and Transition Tiles.
    - **Main Tile Material:** Material to be applied to the main Tile.
    - **Transitions Material:** Material to be applied to the Transition Tiles.

Dungeon Decoration

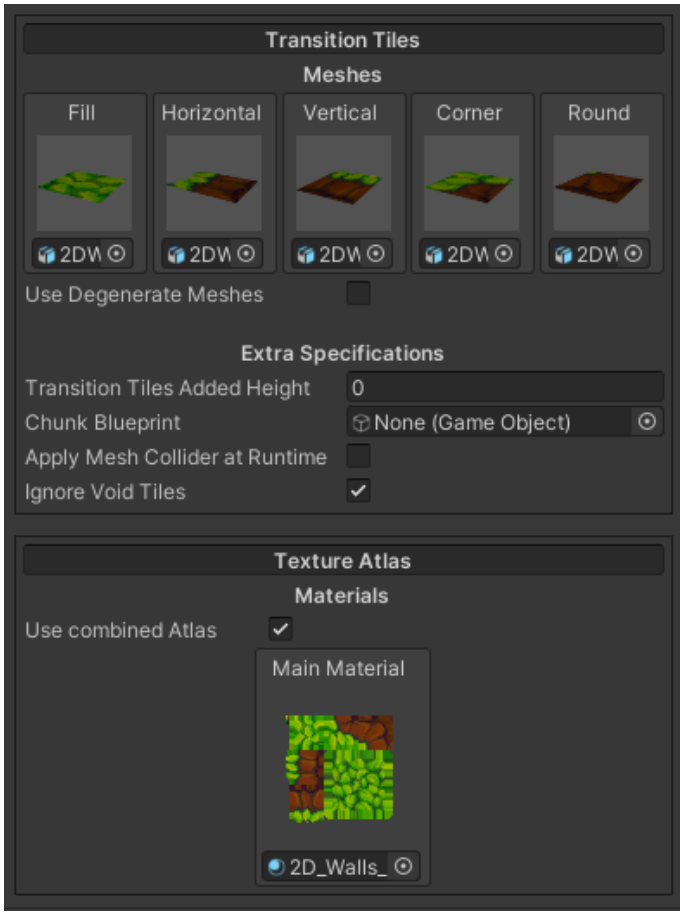
The **Dungeon Decoration** component is an **optional component** that allows you to generate in-game, at-runtime, non-interactable ornaments; and assign them to a specific **Tile Type**.

Like its Brush counterpart, it will deliberately ignore any other components that may be attached in the given **Assets** and will only use the referenced meshes.

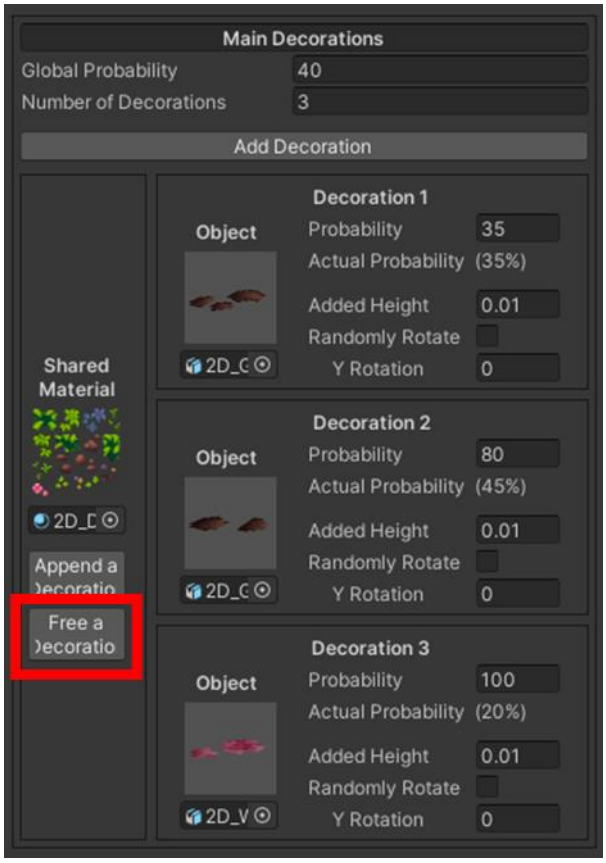
ADG will also, like its Brush counterpart, clump the generated meshes into chunk clusters, and will also allow you to give ***some essential properties*** to these clusters **(like a Shared Material)**

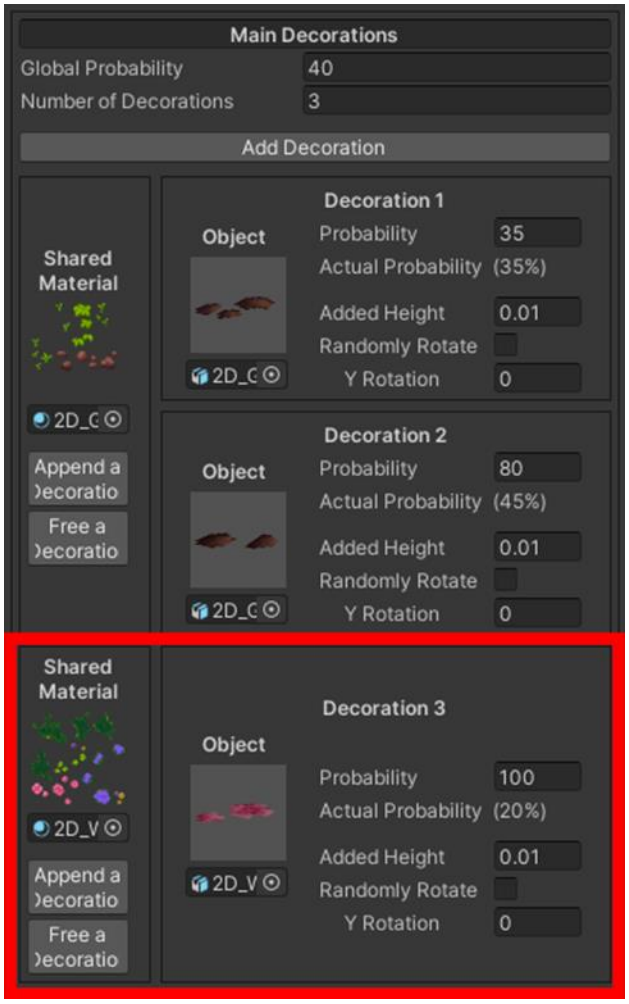
However, unlike its Brush counterpart, you can decide how many clusters to use and how big they are.

Elaborating:
A **Dungeon Brush** will only accept **one Material**, per section, for the creation of its Chunks. **By design**, there is no way to assign an individual material to each **Transition Tiles** of a **Dungeon Brush**, therefore, all individual **Transition Tiles** must share the same material in order to be generated correctly.



And, while it seems that **Dungeon Decorations** suffer from the same predicament you can in fact click on the **"Free a Decoration"** button to make a division between decorations and be able to assign another material to the given decorations.



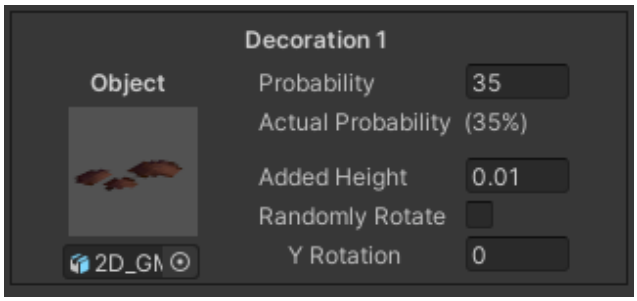


You can have as many sections as individual decorations!

*Even so, I recommend the use of **A Single Material** for each **Dungeon Decoration** because every new Material section will add a new **Draw Call**, but feel free to use as many sections as you deem necessary.*

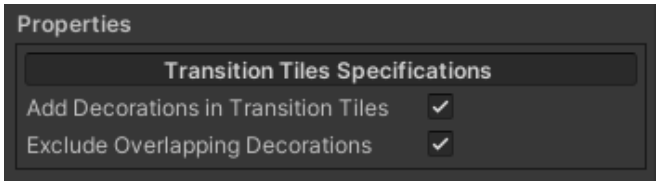
If you choose to use these **Dungeon Decorations**, *ADG* will try to add a **Decoration** at every single assigned **Tile Type**, tile, and its success will depend on its assigned probabilities.

You can also alter some properties of these individual instances, for example, you can decide if they will be randomly rotated or if they will have a fixed angle, as well as if they will have a change in height.



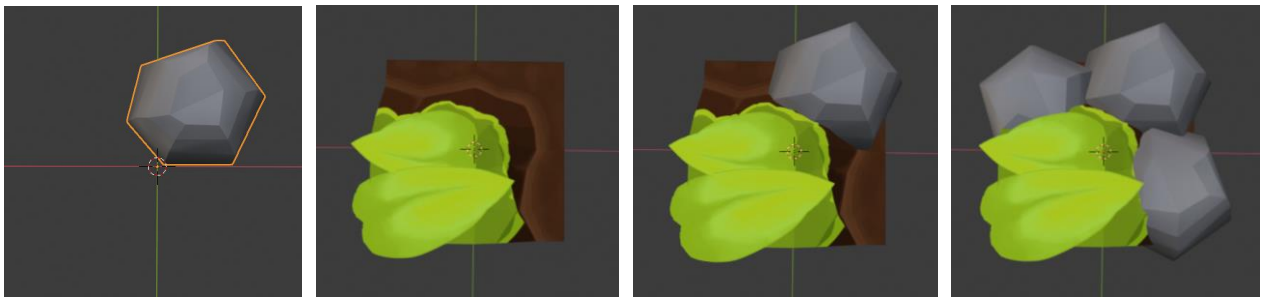
As stated above, *ADG will try to add a Decoration at every single assigned Tile Type, tile*, this also applies to **Transition Tiles**.

In the **Dungeon Decorations**, you can specify which **Decorations** will be used on the **Main Tiles** and which **Decorations** will be used on the **Transition Tiles,** or even if they will have decorations at all.

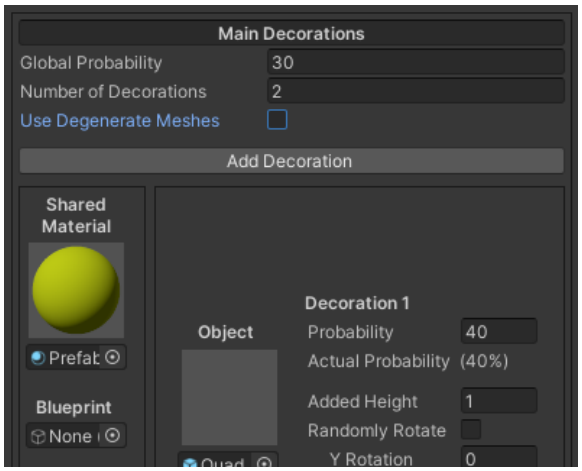


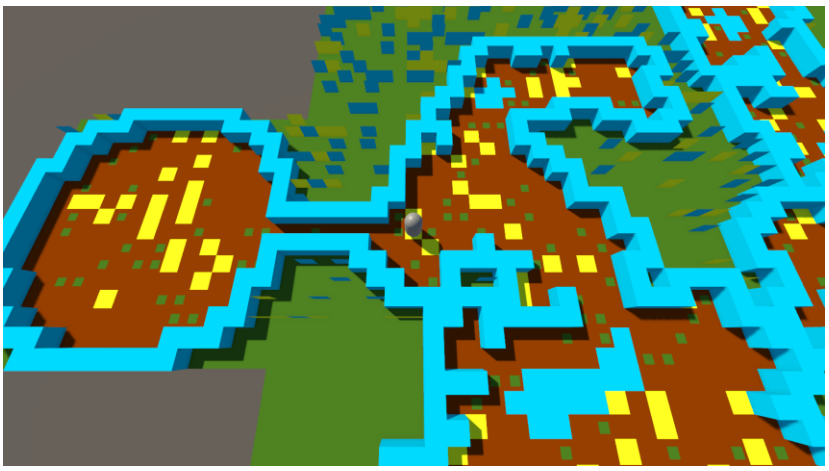
However, **Decorations** assigned to **Transition Tiles** are special, because they cannot rotate freely, they can only be generated in the same rotation as their associated **Transition Tiles** sub-tiles.

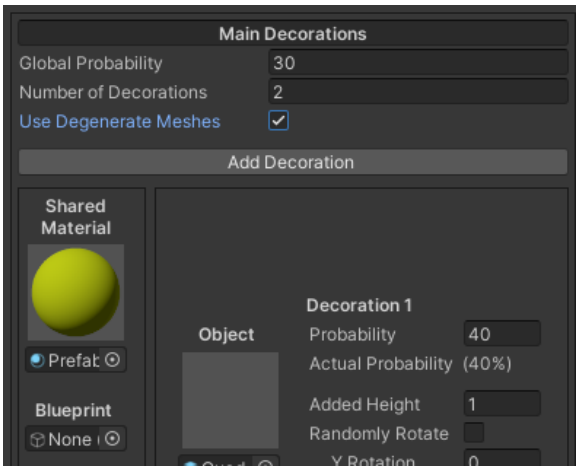
This means that **Decorations** assigned to **Transition Tiles** must have the same specifications as their **Dungeon Brush** sub-tiles counterparts.

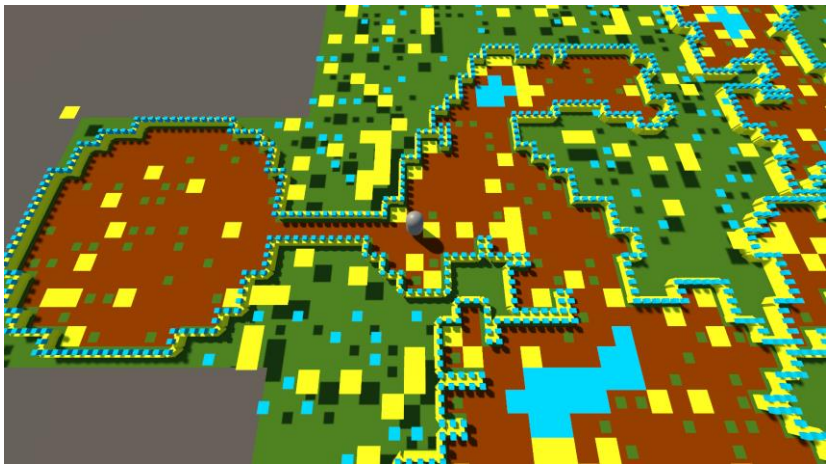


And finally, **Dungeon Decorations** also benefit from **Degenerate Meshes**.

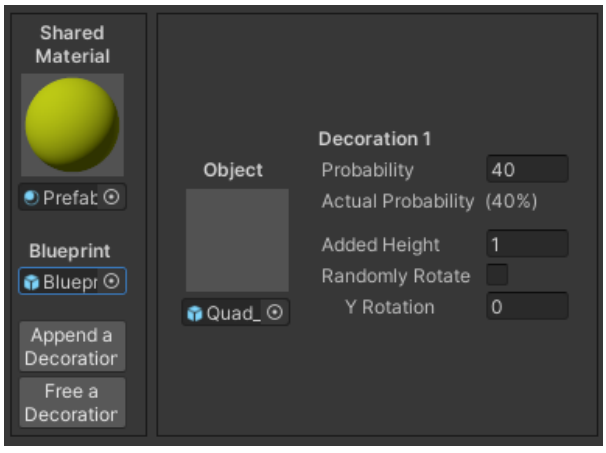


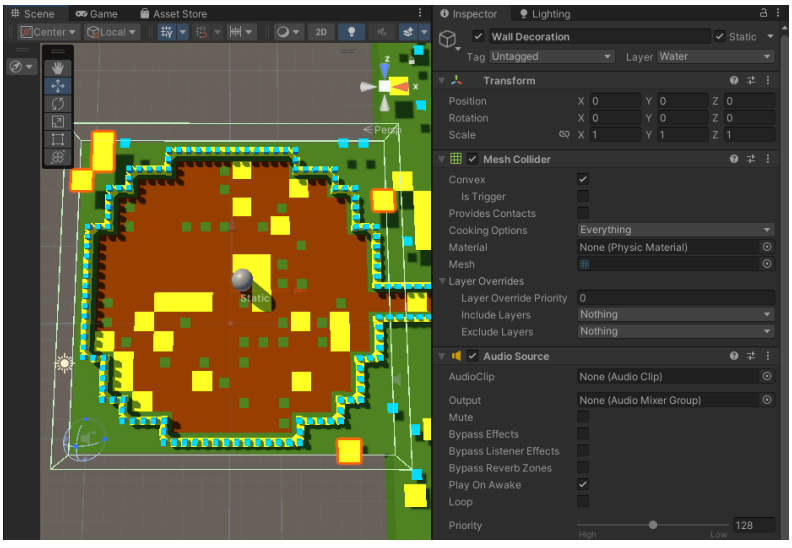






And **Blueprints**.



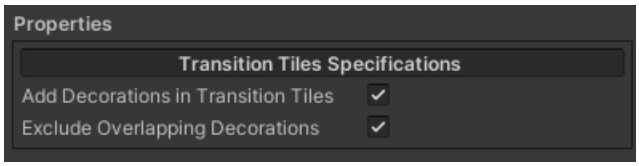


See more on the “***Prefab_Dungeon”*** **Demo**

Dungeon Decoration Properties Summary

Transition Tiles Specifications

Instructions on how to deal with Transition Tiles.



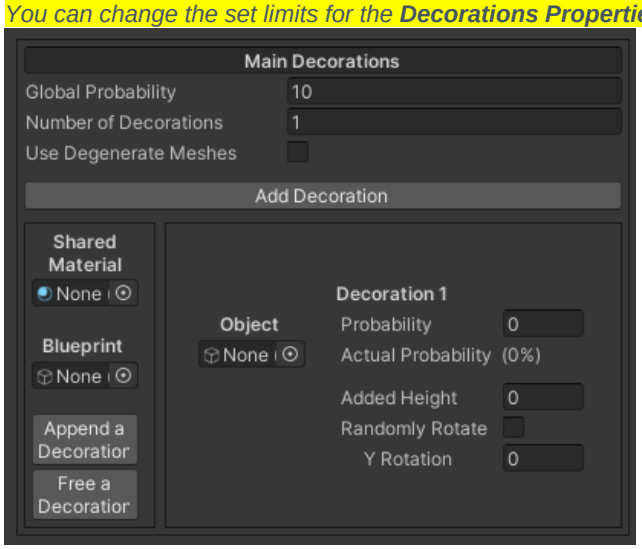
- **Add Decorations in Transition Tiles:** If checked, the Dungeon Decoration will also include a separate set of Decorations for the Transition Tiles.
- **Exclude Overlapping Decoration:** If checked, Main and Transition Decorations will not be generated on the same Tile.

Main Decorations

The set of Decorations for Main Tiles.

Main Decorations Set Properties:

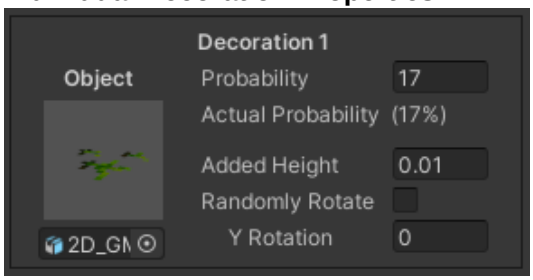
*You can change the set limits for the **Decorations Properties** at the top of the **DungeonDecoration** script.*



- **Global Probability:** The probability of even trying to create a Main Decoration.
- **Number of Decorations:** The number of decorations in this set.
  You can add a new Decoration by pressing the Add Decoration button, or you can set the specific number in the options box. *(Limited to a max of 8)*
- **Use Degenerate Meshes:** When enabled, ADG will also take into account the Transform of the given Prefabs at Tile generation. This option is heavily resource-demanding, use it only for testing purposes.

- **Shared Material:** Material to be shared among a set group of Decorations.
- **Blueprint:** All components of this object will be copied into each Decoration Chunk generated by this group. This option is heavily resource-demanding, use it only when absolutely necessary.
- **Decoration:** An individual Decoration.

Individual Decoration Properties:



- **Object:** The Decoration model.
- **Probability:** A representation of the overall probability of the Decoration.
- **Actual Probability:** The actual probability for this item to be generated. The probability of this item spawning is the probability of this decoration minus the previous one.

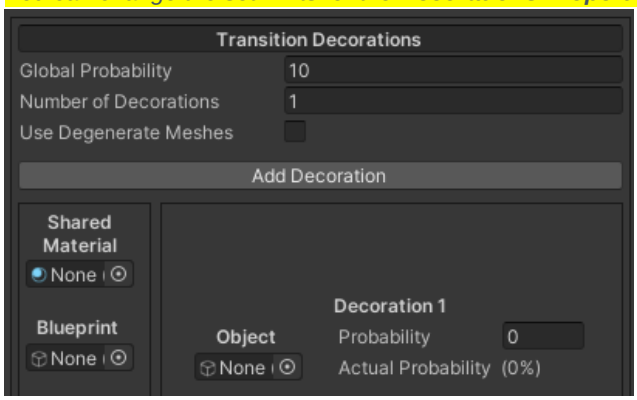
- **Added Height:** Height to be added to this decoration.
- **Randomly Rotate:** If checked, when generated, the Decoration will be randomly rotated on the Y axis.
- **Y Rotation:** The rotation, in degrees, that will be set on the Y axis of this Decoration when it is created.

Transition Decorations

The set of Decorations for Transition Tiles.

Transition Decorations Set Properties:

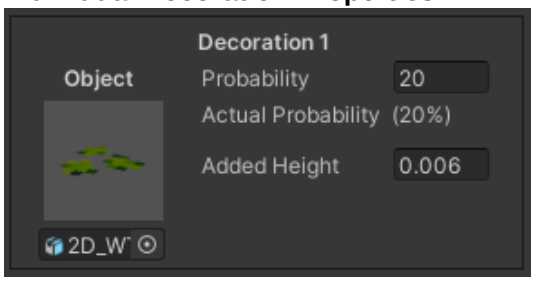
*You can change the set limits for the **Decorations Properties** at the top of the **DungeonDecoration** script.*



- **Global Probability:** The probability of even trying to create a Transition Decoration.
- **Number of Decorations:** The number of decorations in this set.
  You can add a new Decoration by pressing the Add Decoration button, or you can set the specific number in the options box. *(Limited to a max of 8)*
- **Use Degenerate Meshes:** When enabled, ADG will also take into account the Transform of the given Prefabs at Tile generation. This option is heavily resource-demanding, use it only for testing purposes.

- **Shared Material:** Material to be shared among a group of Decorations.
- **Blueprint:** All components of this object will be copied into each Decoration Chunk generated by this group. This option is heavily resource-demanding, use it only when absolutely necessary.
- **Decoration:** An individual Decoration.

Individual Decoration Properties:



- **Object:** The Decoration model.
- **Probability:** A representation of the overall probability of the Decoration.
- **Actual Probability:** The actual probability for this item to be generated. The probability of this item spawning is the probability of this decoration minus the previous one.

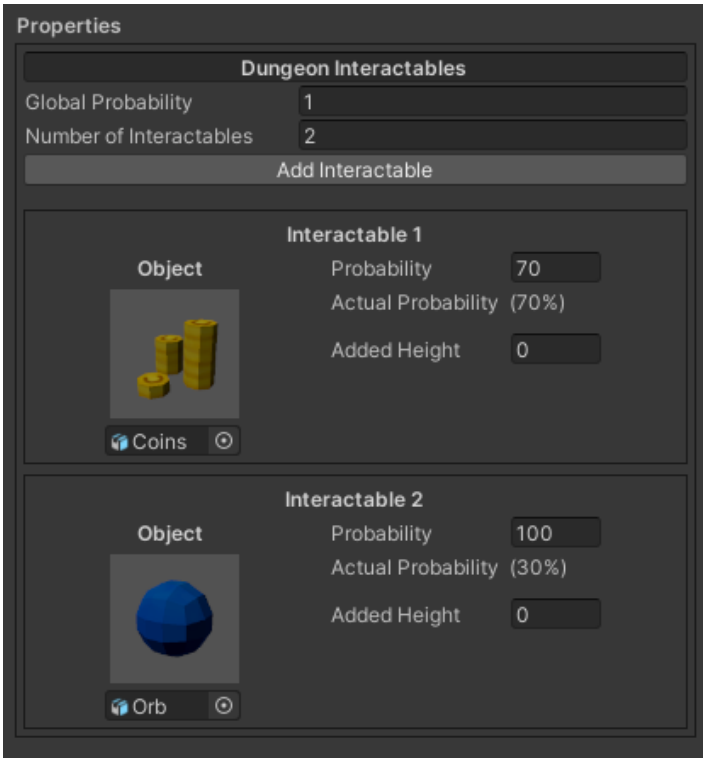
- **Added Height:** Height to be added to this decoration.

Dungeon Interactables

The **Dungeon Interactable** component is an **optional component** that allows you to generate in-game, at-runtime, **GameObjects** on a specific **Tile Type**.

Unlike all the other components, **Dungeon Interactables** are not based on clumping meshes into **Chunks** to save performance. *ADG* will just simply Instantiate the given **Prefabs** based in its spawn probability.

Dungeon Interactables are by far the most resource demanding feature in all *ADG*, is very util for pickups creation but use it very cautiously.



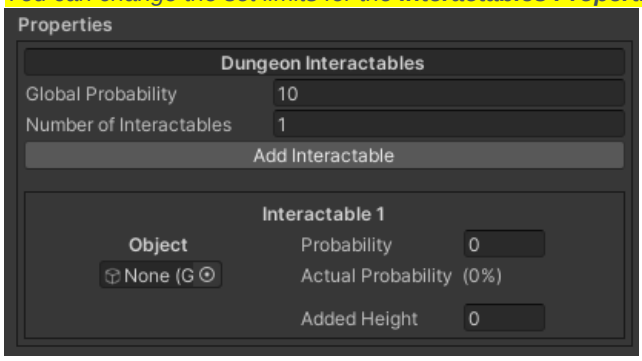
Dungeon Interactable Properties Summary

Dungeon Interactables

The set of Prefabs to be Instantiated by this Dungeon Interactable.

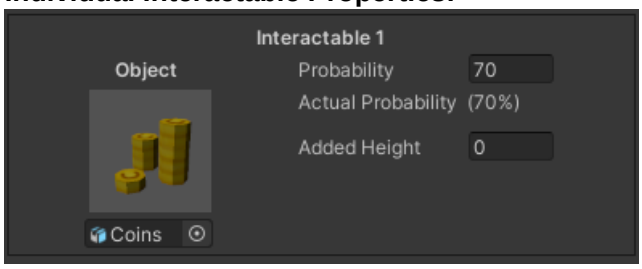
Interactables Set Properties:

*You can change the set limits for the **Interactables Properties** at the top of the **DungeonInteractables** script.*



- **Global Probability:** The probability of even trying to create a Interactable.
- **Number of Decorations:** The number of decorations in this set.
  You can add a new Decoration by pressing the Add Decoration button, or you can set the specific number in the options box. *(Limited to a max of 8)*

Individual Interactable Properties:



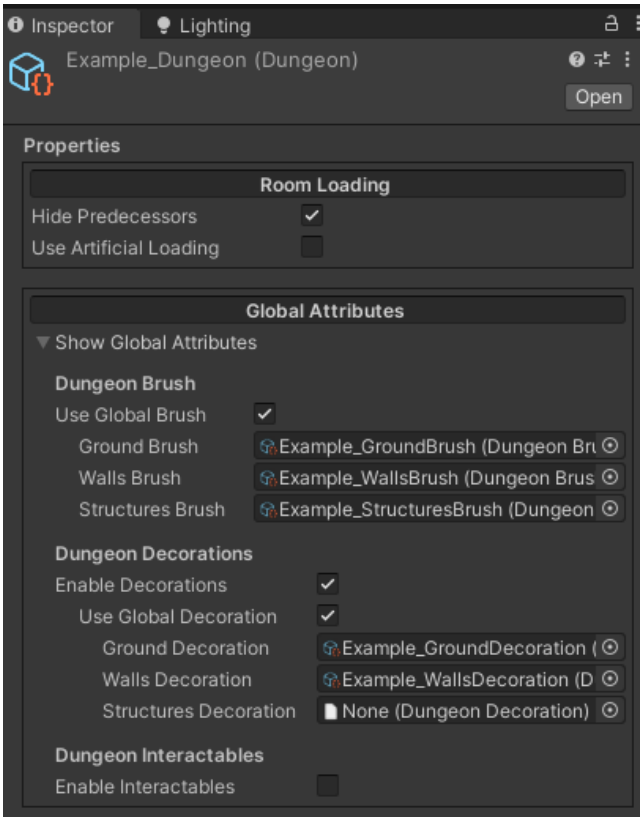
- **Object:** The Decoration model.
- **Probability:** A representation of the overall probability of the Decoration.
- **Actual Probability:** The actual probability for this item to be generated. The probability of this item spawning is the probability of this decoration minus the previous one.

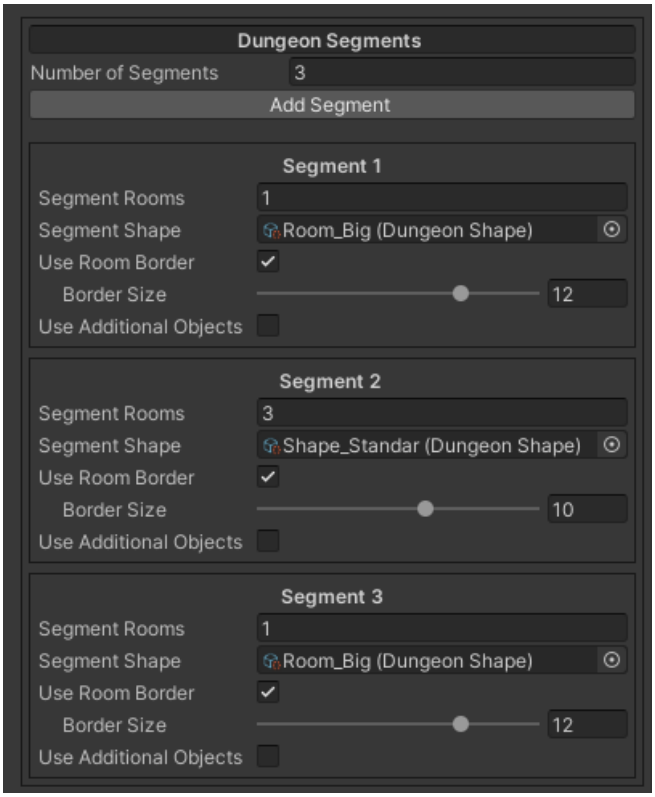
- **Added Height:** Height to be added to this decoration.

Dungeon

The **Dungeon** is an element that acts as a linker between **Dungeon Components**.

Its job is to specify how many **Dungeon Rooms** will be created under the same set of rules and how they will be joined and loaded at-runtime.





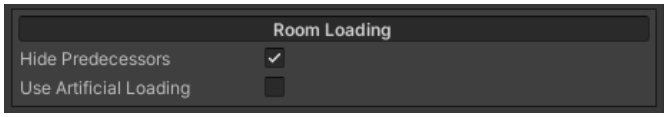
Remember:

- A **Dungeon** is defined by its individual **Components**. If you change the **Components**, you will change the **Dungeon**.
- **By design**, *Advanced Dungeon Generator* **does not** allow you to concatenate **Dungeons**, however, *ADG* allows you to divide your **Dungeons** into different *segments*.
- In a Dungeon, a **Dungeon Segment** is a group of **Dungeon Rooms** generated using the same **Components**.

Dungeon Properties

Room Loading

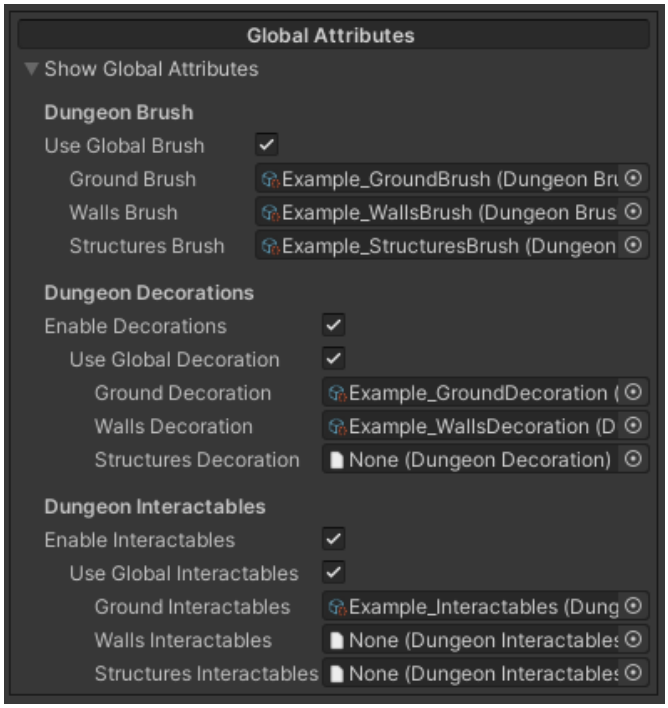
The Dungeon Generator will load and unload; build and destroy Dungeon Rooms as needed, you can specify certain aspects of this actions. Learn more about it in the Dungeon Loader section.



- **Hide Predecessors:** If checked, the rooms that are no longer visible on the screen will be hidden.
- **Use Artificial Loading:** If checked, the user will define the proper load and distribution of the Dungeon Rooms, otherwise, the Dungeon Generator will do it on its own.

Global Attributes

All Dungeons can be divided into segments. Each Dungeon Segment can have its own Dungeon Brushes and Dungeon Decorations. The Global Attributes section will help you set up common Dungeon Components in all segments.



- **Show Global Attributes:** A fold-out label that lets you see the global attributes of the Dungeon.

- **Use Global Brush:** When enable, these Brushes will be assigned to all Dungeon segments.
  - **Ground Brush:** This Brush will be applied to Ground Tiles.
  - **Wall Brush:** This Brush will be applied to Wall Tiles.
  - **Structure Brush:** This Brush will be applied the Structure Tiles.

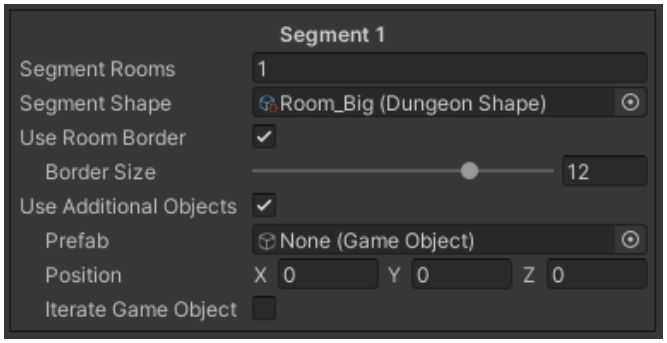
- **Enable Decorations:** When enabled, you can add Decorations to the Dungeon.
- **Use Global Decoration:** When enable, these Decorations will be assigned to all Dungeon segments.
  - **Ground Decoration:** These Decorations will be placed on Ground Tiles.
  - **Wall Decoration**: These Decorations will be placed on Wall Tiles.
  - **Structures Decoration:** These Decorations will be placed on Structure Tiles.

- **Enable Interactables:** When enabled, you can add Interactables to the Dungeon.
- **Use Global Interactables:** When enable, these Interactables will be assigned to all Dungeon segments.
  - **Ground Interactables:** These Interactables will be placed on Ground Tiles.
  - **Wall Interactables**: These Interactables will be placed on Wall Tiles.
  - **Structures Interactables:** These Interactables will be placed on Structure Tiles.

Dungeon Segments

*You can change the set limits for the **Dungeon Segments Properties** at the top of the **Dungeon** script.*

Number of Segments & Segment Properties:
The number of segments of this Dungeon. You can add a new segment by pressing the Add Segment button, or you can set the specific number in the options box. *(Limited to a max of 32)*



- **Segment Rooms:** The number of Dungeon Rooms in this segment. *(Limited to a max of 64)*
- **Segment Shape:** The Dungeon Shape for these segment rooms.

- **Use Room Border:** The rooms of this segment will have a smart border, very useful to avoid showing the raw edges of the room. However, the border system is still quite primitive and artifacts are likely to occur, use it with caution.
  - **Border Size:** The size, in tiles, of the room border *(Limited to the Segment Shape Chuck Size)*

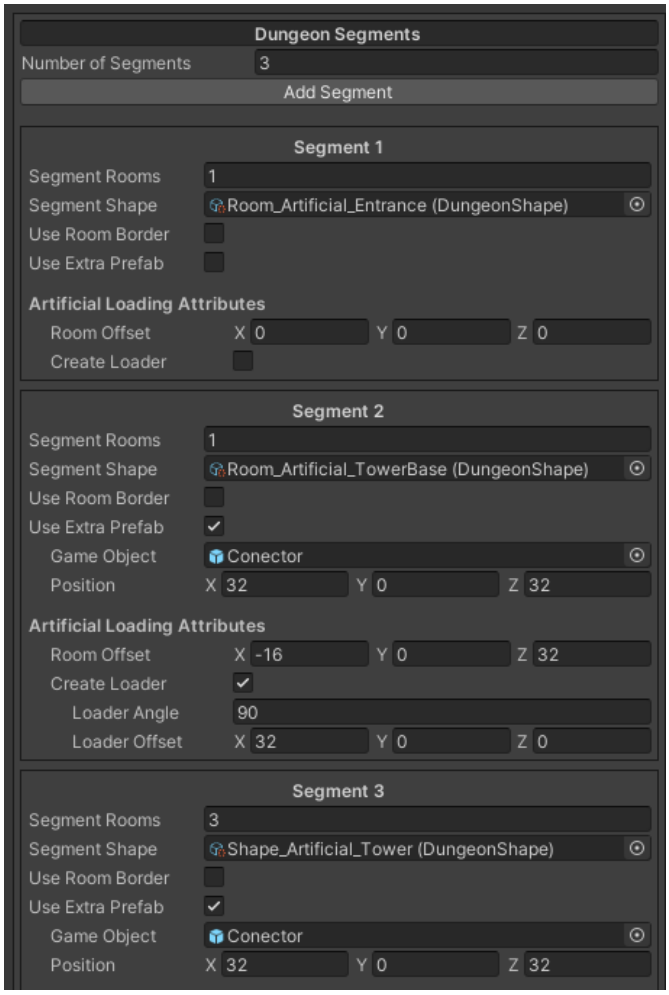
- **Use Additional Objects:** Will instantiate a given GameObject in each room of this segment.
  - **Prefab:** The GameObject to be instantiated with the Dungeon Room.
  - **Position:** Local position of the GameObject relative to its Dungeon Room position (Lower Left Corner).
  - **Iterate Game Object:** When enable, the given GameObject will be instantiated at each of the key points of the Dungeon Room, very useful if you want to define **Spawners**.

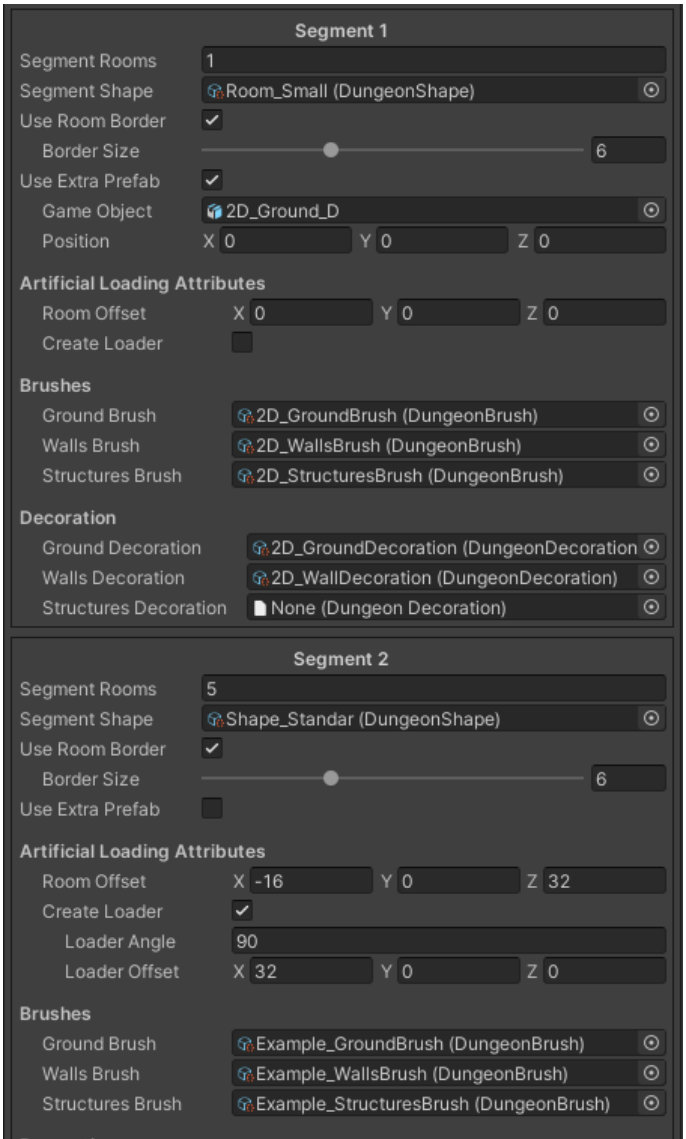
Artificial Loading Attributes:
These options will be available only if the Dungeon Room Loading is set to Use Artificial Loading. You can learn more about how it works in the **Dungeon Loader** section, or you can see it in use at the Dungeon Demos.

- **Room Offset:** The amount of displacement a Room of this segment has, relative to the previous one. To clarify, this action moves the Global Position of this Room from the Global Position of the previous one (The position of any Room starts at its Lower Left Corner).

- **Create Loader:** Will instantiate a custom loader in all the rooms of the segment.
  - **Loader Angle:** The Y angle at which the Room Size Loader will be.
  - **Loader Offset:** The amount of displacement the Room Size Loader will have, relative to its Room position.

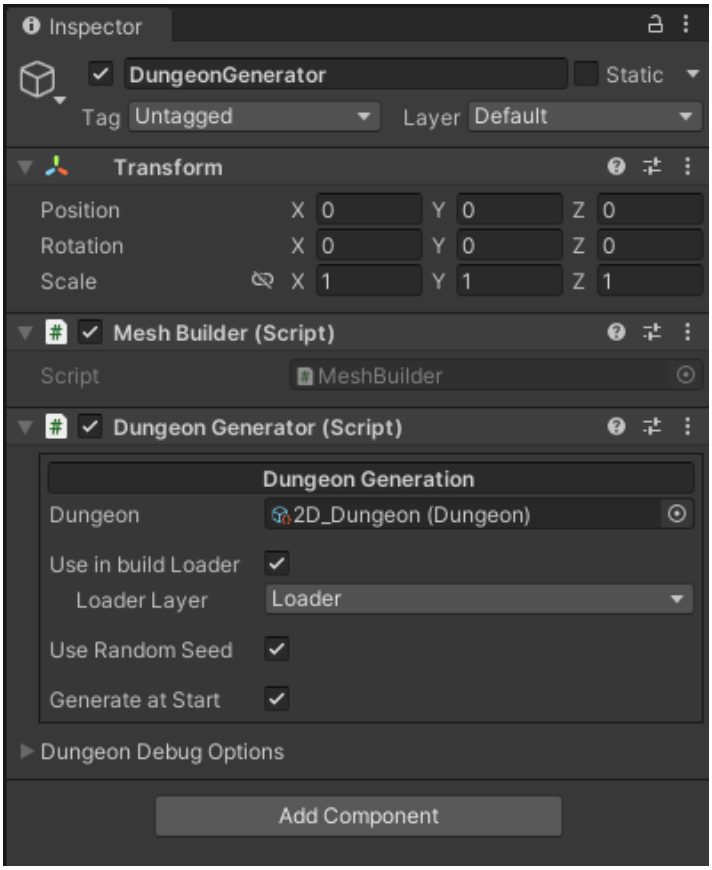
If a Dungeon is not using Global Attributes, each Dungeon Segment will ask you for its respective Dungeon Brushes and/or Dungeon Decorations.





Dungeon Generator

The **Dungeon Generator** is the element responsible for the creation of the in-game, at-runtime **Dungeon Rooms.** It works in conjunction with a specific **Dungeon** to build its predefined set of **Dungeon Rooms**.

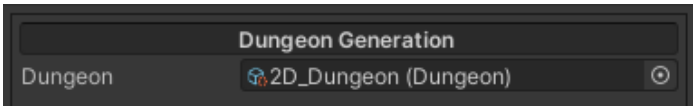


To create your in-game **Dungeon**, add an empty **GameObject** to your **Scene** and attach the **Dungeon Generator component** to it.

You will notice that another script will be added as well. The **Mesh Builder** script is a support construction script, and both need to be present in the same **GameObject** to function properly.

Note that there can only be one Dungeon Generator for Scene. If you have multiple **GameObjects** with the **Dungeon Generator** component attached, the extras will be destroyed at runtime.

Roughly speaking, the **Dungeon** property is the only one you will be changing. Once you assign a **Dungeon**; the **Dungeon Generator** will do the rest.



If you don't want the **Dungeon** to be loaded at the **Start**, you can still call its initialization from wherever you want, using the method:

`DungeonGenerator.Instance.DungeonStartTrigger();`

For example, here the Dungeon will be loaded only when the "L" key is pressed.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

Script de Unity | 0 referencias
public class TriggerTest : MonoBehaviour
{
    Mensaje de Unity | 0 referencias
    void Update()
    {
        if (Input.GetKeyDown(KeyCode.L))
        {
            DungeonGenerator.Instance.DungeonStartTrigger();
        }
    }
}
```

You can also delegate methods for when the **Dungeon** finishes loading, using the event:

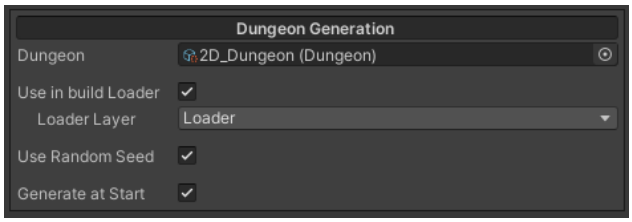
`DungeonGenerator.Instance.OnDungeonComplete += Method_Name;`

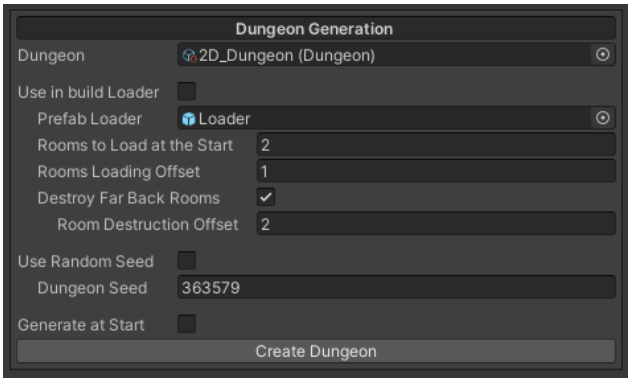
For example, here the player can move only when the Dungeon is fully loaded.

```
Mensaje de Unity | 0 referencias
private void Start()
{
    DungeonGenerator.Instance.OnDungeonComplete += StartPlayerReading;
}

2 referencias
public void StartPlayerReading()
{
    transform.GetChild(0).gameObject.SetActive(true);
    playerCanMove = true;

    Debug.Log("Im ready!");
}

Mensaje de Unity | 0 referencias
private void OnDisable()
{
    DungeonGenerator.Instance.OnDungeonComplete -= StartPlayerReading;
}
```

Dungeon Generator Properties

Dungeon Generation





- **Dungeon:**
  The Dungeon to be build.

- **Use in build Loader:**
  If checked, the Dungeon Generator will create the standard Loaders of the Dungeon on its own. Otherwise, the user will specify the prefab loader to use.
  - **Loader Layer:**
    The layer that will be assigned to all Dungeon Loaders.

    - **Prefab Loader:**
      The loader instantiated at every room. This prefab needs to have the Dungeon Loader script attach, you can learn more about it in the **Dungeon Loader** section.
    - **Rooms to Load at the Start:**
      The number of rooms loaded at the start of the dungeon creation *(2 by default).*
    - **Rooms Loading Offset:**
      When the loader triggers, the room to load will be the current one plus this number *(1 by default).*
    - **Destroy Far Back Rooms:**
      Rooms that are too far back will be destroyed *(True by default).*
    - **Room Destruction Offset:**
      When the loader triggers, the room to destroy will be the current one minus the Room Loading Offset and this number (*2 by default).*

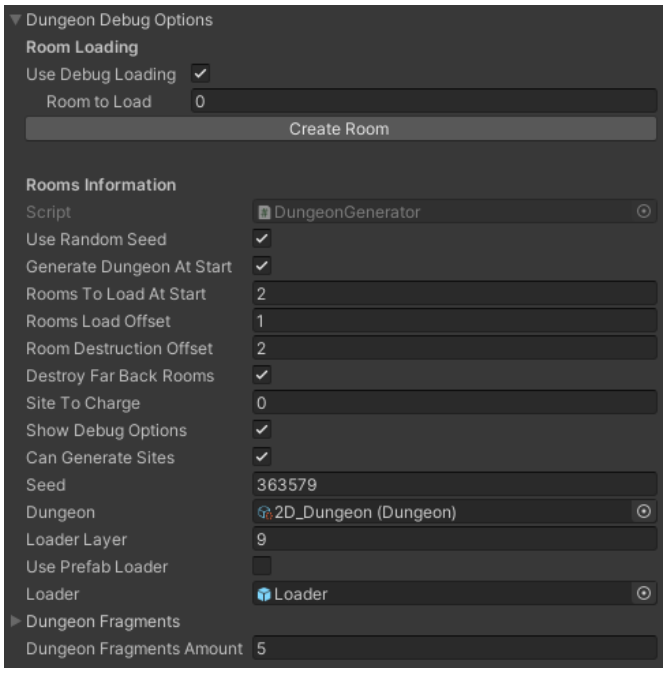
- **Use Random Seed:**
  If checked, the Dungeon Generator will use a random seed when creating the Dungeon.
  - **Dungeon Seed:** The seed that will be use at Dungeon generation.

- **Generate at Start**:
  If checked, the Dungeon will be generated as soon as the script is enabled.
  - **Create Dungeon:** Clicking this button will generate the Dungeon, you can call this method of the Dungeon Generator script by the **DungeonStartTrigger** name.

```
DungeonGenerator.Instance.DungeonStartTrigger();
```

Dungeon Debug Options

A set of extra options that will help you understand what is going on behind the generation of your Dungeons, it will also let you generate specific Dungeon Rooms.



Room Loading:

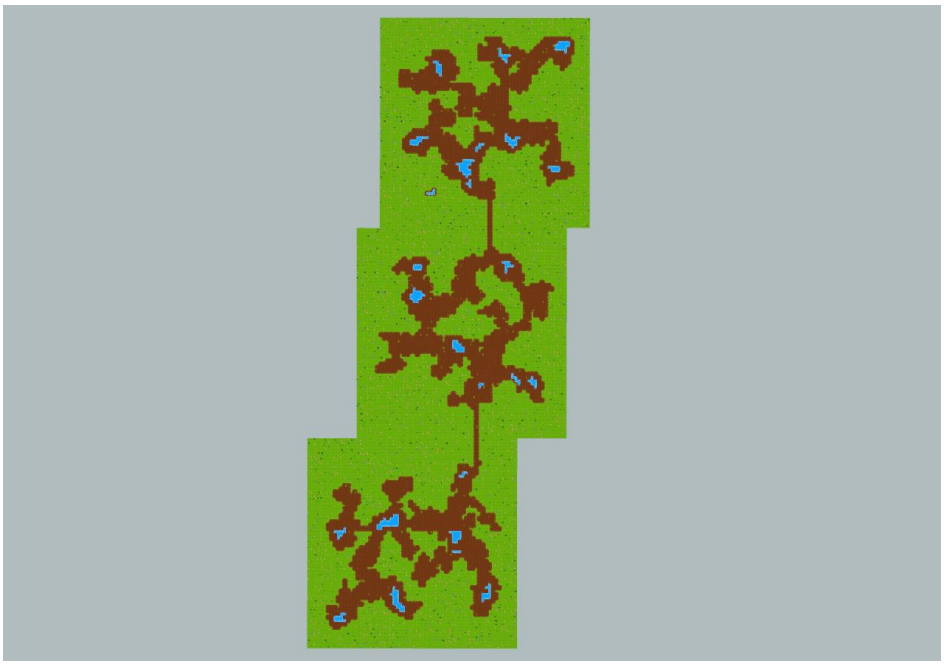
- **Use Debug Loading:** Gives you the option to load a specific room from the Dungeon.
- **Room to Load:** The ID of the room you want to load.
- **Create Room:** Clicking this button will generate the specified room.

Rooms Information:

- **Dungeon Fragments:** A visual representation of the generation values of all the Dungeons Rooms.

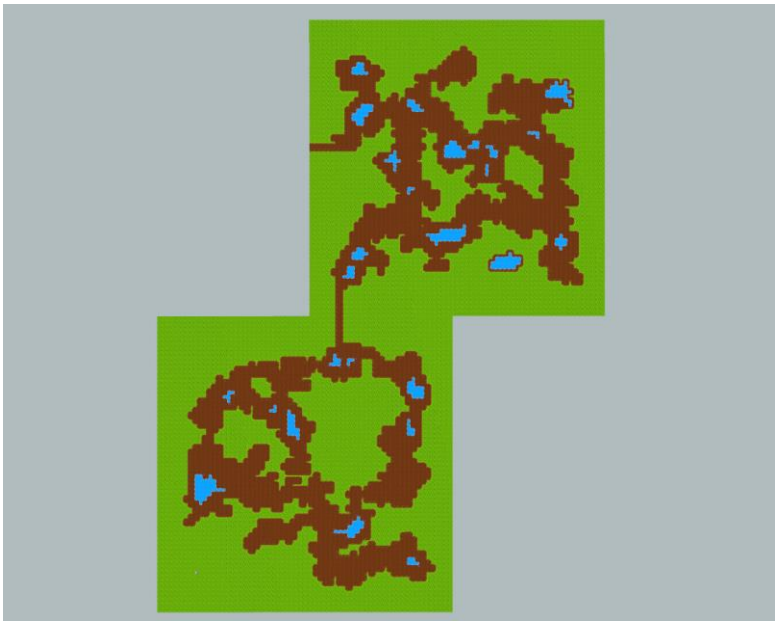
Dungeon Loader

A **Dungeon** is a successive and connected collection of **Dungeon Rooms.**



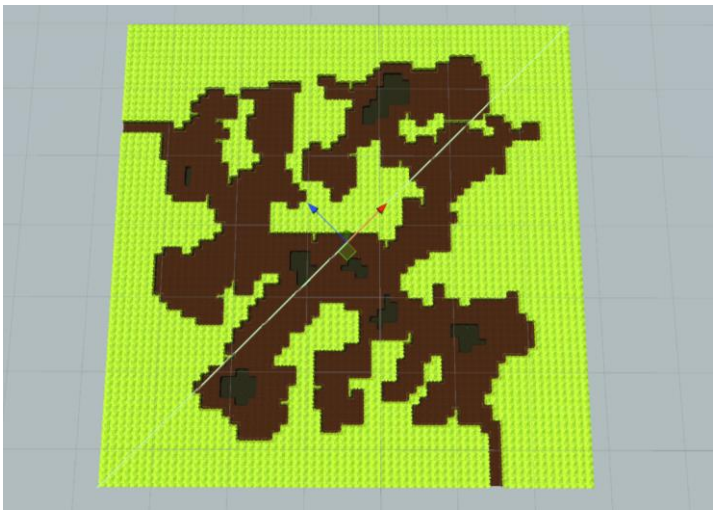
However, **there is no need to load all of them at once**.

Or to have them loaded when they are too far away.



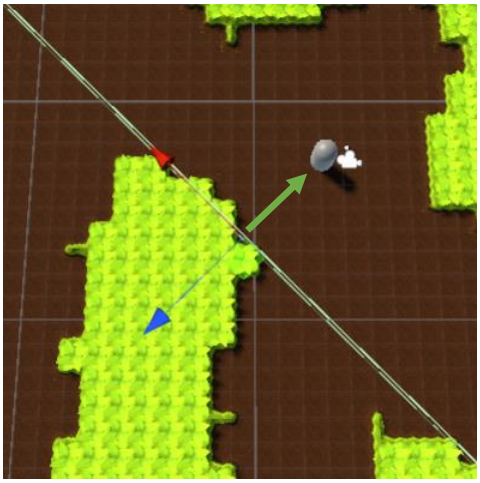
As you may have seen, the **Dungeon Generator** will only create some **Dungeon Rooms** when it is initialized. The rest will be generated dynamically.

To ensure this, the **Dungeon Generator** will create **Loaders**, room-size loaders between the entrance and the exit of the **Dungeon Rooms**.



These **Loaders** are **Box Colliders** that will use their position and the position of their triggers to decide which rooms are to be loaded and which are to be unloaded.

The rotation of these objects is important, the loaders are always looking towards the entrance of the rooms, and they are always on the lookout for other colliders trying to go through them.



The **dot product** of these two vectors is the value that will determine the **Dungeon Room** to load.

- If these vectors point in **opposite directions**, the room in front of the exit will be loaded, and the room in front of entrance will be unloaded.
- If these vectors point in **the same direction**, the room in front of the entrance will be loaded, and the room in front of the exit will be unloaded.

There are a lot of things that are done in the background when it comes to generating a **Dungeon**.

Making the entrances and exits of the rooms, placing rooms one after the other, creating the room loaders in the right way, all of this is done automatically in a standard session.

But, in **Artificially Loaded Dungeons**, the user will be responsible for all these.

In Artificially Loaded Dungeons:

- The entrances and exits of the rooms should be mapped in the **Mask Texture** of your **Dungeon Shapes**, whether those shapes are artificial or not.
- The rooms offset is calculated by the user and set at every **Dungeon Segment** in the **Artificial Loading Attributes**.
- The loader, its angle (the loader front, the blue arrow, should point to the entrance of the rooms), and its offset are calculated for each room by the user and set at every **Dungeon Segment** in the **Artificial Loading Attributes**.

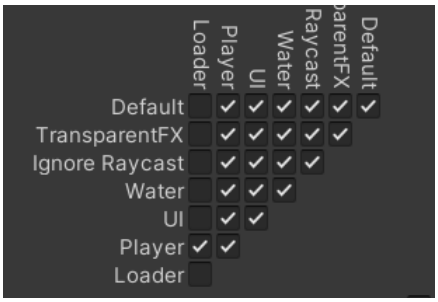
These new options will of course complicate the creation of **Dungeons**, but don't worry, these are very specific tools for very specific scenarios that will allow you to create amazing things.

You also have a demonstration on how to manage Artificially Loaded Dungeons in the Demos.

Loaders Behavior

The **Loaders** will react to **any** colliders trying to go through them, so you should select the **only layer** that will be assigned to all **Dungeon Loaders** in the **Dungeon Generator**, or set a custom **Loader**.

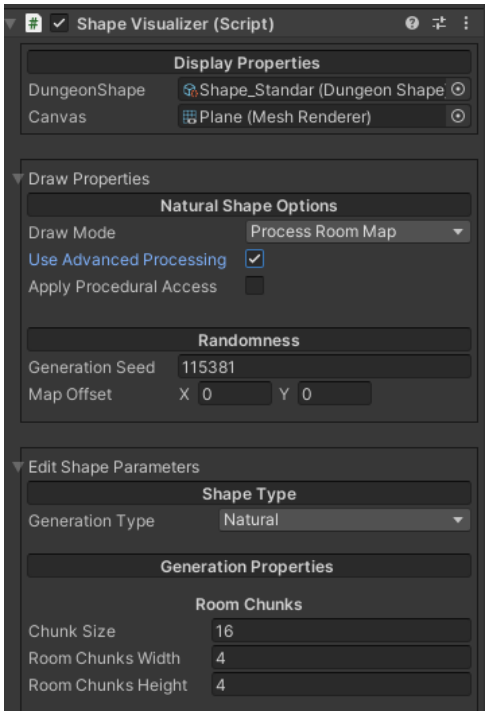
I recommend you to put the **Player** in a specific layer and then modify the **Unity Layer Collision Matrix (Edit > Project Settings > Physics)** so they can only interact with each other.

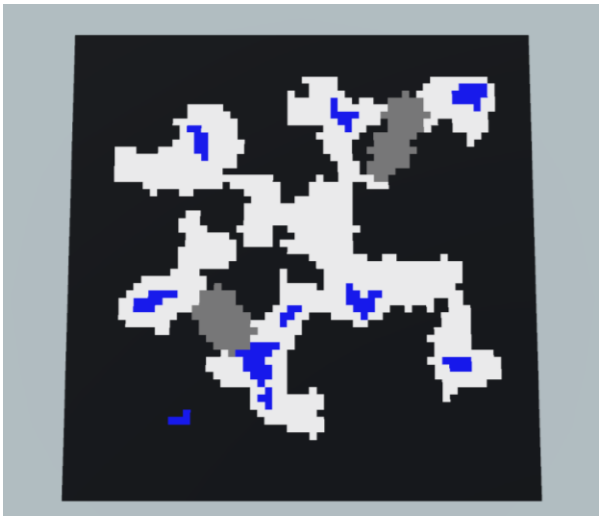


If you don't want to use the **Dungeon Generator** in-build loader system you can add your own **Loaders** *(There is an example Loader prefab in the Demos)*, by doing this you can also change the default loading settings and set how many rooms will be generated at the start, the rooms unloading offset, if the rooms too far back can be destroyed, etc.

Shape Visualizer

The **Shape Visualizer** script is an extra tool that lets you preview the **Dungeon Maps** that a **Dungeon Shape** can generate.

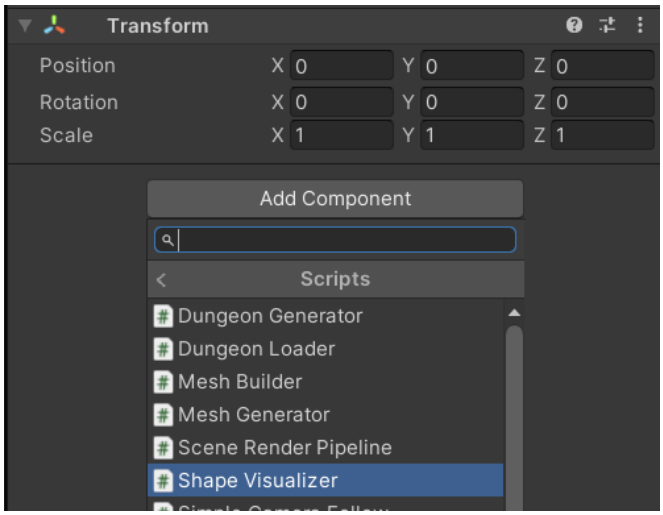




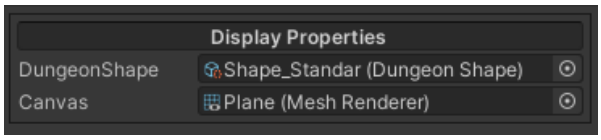
Setting the parameters of a **Dungeon Shape** can become a difficult task, especially if you don't have active feedback on how your changes affect the generation of the **Dungeon Rooms**.

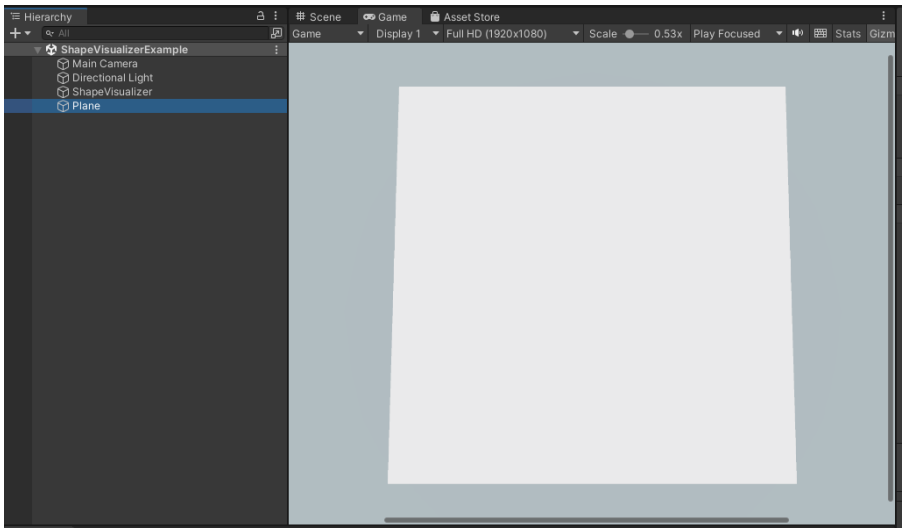
With the Shape Visualizer, you can see how your changes in the Dungeon Shapes affect their rooms on the fly!

*To use the **Shape Visualizer** tool, create a **GameObject** and add the **Shape Visualizer** script.*

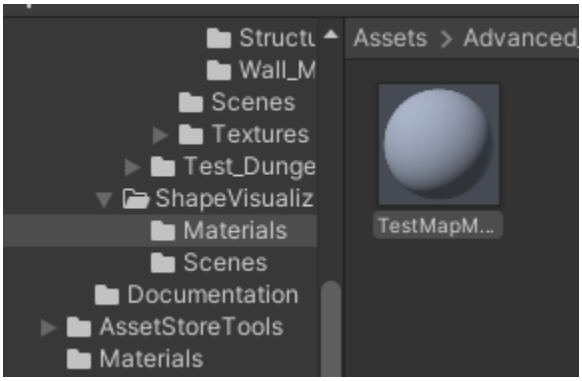


You will also need another **GameObject** to use as a canvas.





Remember that **this will change the texture of its material**, so create a specific material only for your visualizations.



You can see an example of the Shape Visualizer configuration in the Demos folder

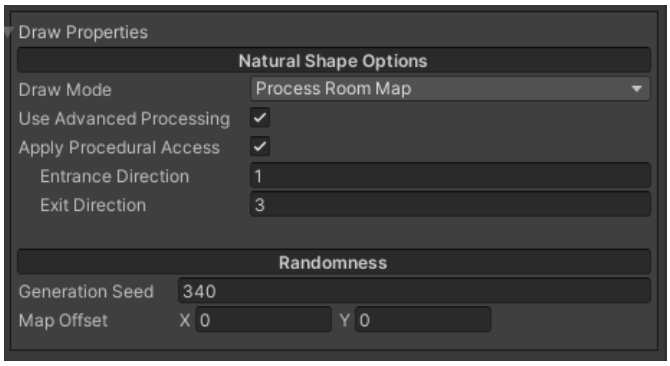
Shape Visualizer Properties

Display Properties

- **Dungeon Shape:** The Dungeon Shape that will be tested.
- **Canvas:** The GameObject on which the rooms will be drawn.

Draw Properties

The generation type of the Dungeon Shape will determine what drawing options will be available.



- **Draw Mode:** You can decide on a specific component of the rooms to draw.
  Depending on the Dungeon Shape, you can select between the Processed Room Map, the Basic Noise Map of the room, the Falloff Map of the rooms, and the Image given for the Masking Process of an Artificial Shape.

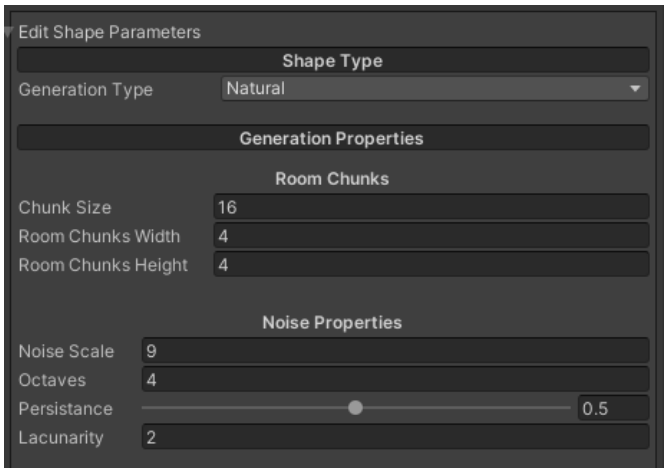
- **Use Advanced Processing:** Will show the post-processed room maps.
  A Processed room map will only show the raw Perlin Noise smoothing process, a post-Processed room map will show the sub-rooms now cleaned and joined.

- **Apply Procedural Access:** Will apply random entrances and exits to the room maps.
  - **Entrance Direction:** An entry will be created on this side of the room.
  - **Exit Direction:** An exit will be created on this side of the room.

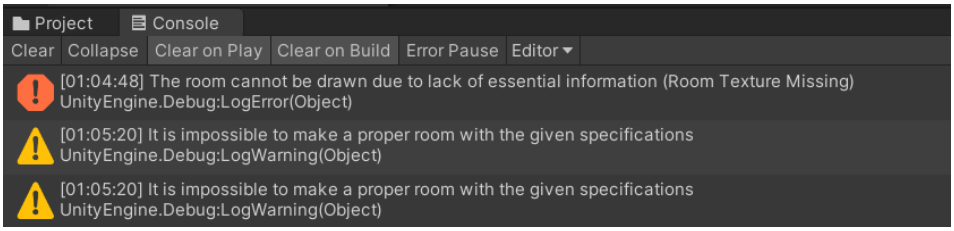
- **Generation Seed:** The seed of the base Perlin Noise map.
- **Map Offset:** The offset of the base Perlin Noise map.

Edit Shape Parameters

Here you can change all the properties of the selected Dungeon Shape.



The Shape Visualizer script will give you a custom error if the generator cannot create a room with the given specifications.



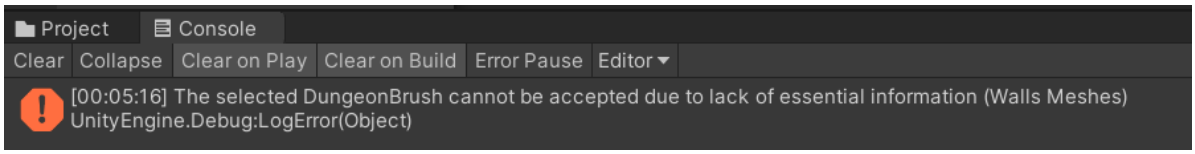
Update Options

- **Auto Update:** If checked, the canvas will be updated every time you change the Shape Parameters.
- **Draw Map:** The canvas will be updated when this button is pressed.

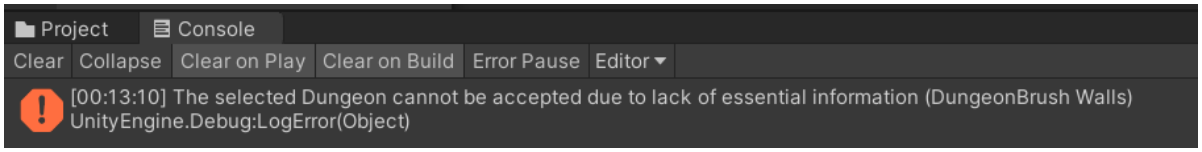
Security Check

Security Check is a small tool that will help you manage your **Dungeons**. It will behave as a background element in all your **Dungeon Components** and **will prevent you** from adding key elements that lack essential information.

For example, when creating a **Dungeon**, it will prevent you from adding any **Dungeon Brushes** that lacks **Meshes** and/or **Materials**.



It also works with the **Dungeon Generator**, so it will not accept a **Dungeon** that lacks **Dungeon Brushes** or **Dungeon Shapes**.



These are the elements that are considered to be essential information in each **Dungeon Component**:

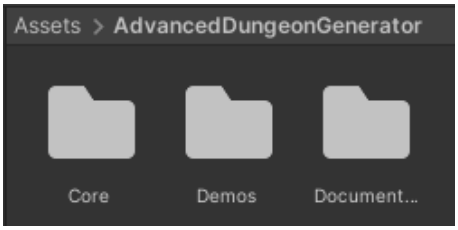
- **Dungeon Shape**:
  - The *Room Mask Texture* in all Artificially Generated Shapes.
- **Dungeon Brush**:
  - The *Main Tile Mesh* in all Brushes with Standard or Combined Tilling.
  - *Each Transition Tile Mesh* in all Brushes with Complete or Partial Transition Tilling.
  - The *Material* in the Texture Atlas for all Dungeon Brushes.
- **Dungeon Decoration**:
  - *Each Object Model* for the Decorations of the Dungeon Decoration.
  - *Each Material* for the Shared Materials of the Dungeon Decoration.
- **Dungeon**:
  - *Each Dungeon Shape* for all Dungeon Segments.
  - *The Dungeon Brush for Walls* in each Dungeon Segment.
  - *The Dungeon Brush for the Ground* in each Dungeon Segment.

Note that the Security Check script only checks the **Dungeon Components** validity when they are assigned. In theory, you can assign a valid component and then change it to make it invalid, please keep this scenario in mind.

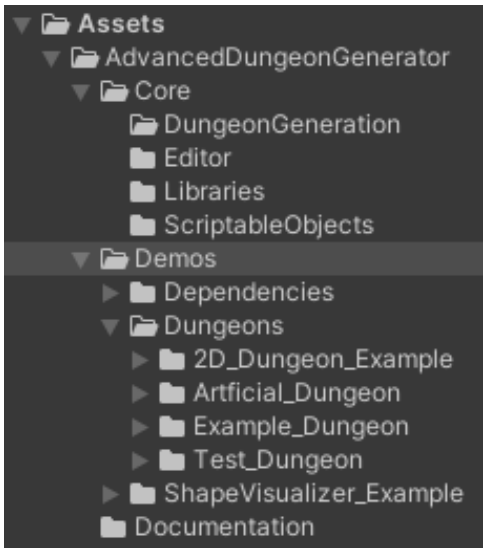
Demos

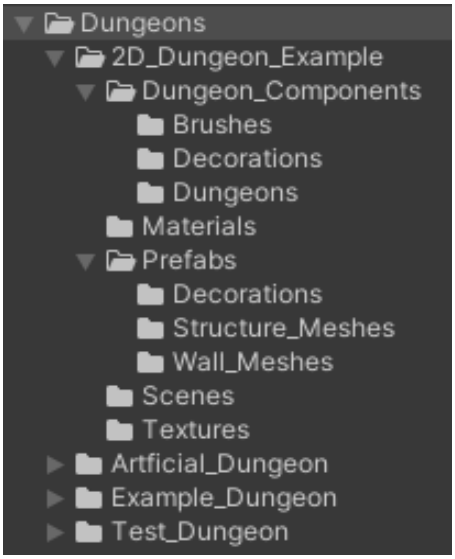
Advanced Dungeon Generator comes with several content folders. In one of these is this **Documentation**. There is also a folder full of **Demonstrations**.

The only folder required for *ADG* to work is the *Core folder*, the rest can be safely deleted or deselected at import.



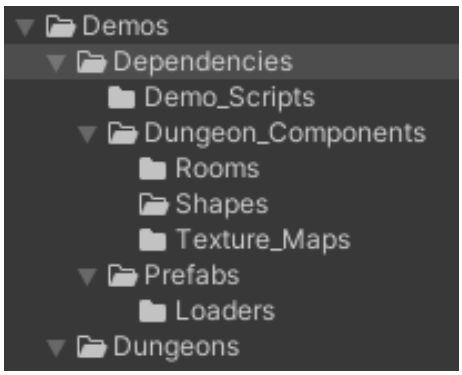
The **Demos** folder is segmented, each **Dungeon** example is encapsulated in its own folder and can be imported separately.





However, every time you want to import a **Dungeon** you also need to import the **Dependencies** folder.

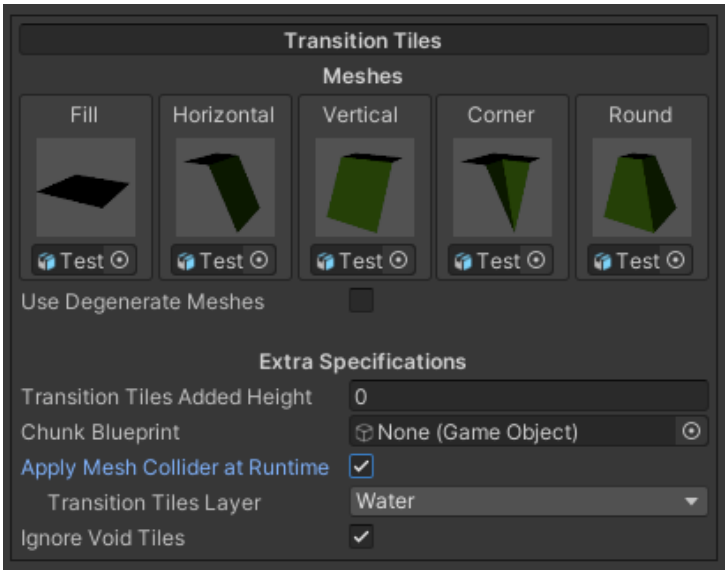
The **Dependencies** folder contains key components for ***all Dungeons***, including the **Shape Visualizer example**.



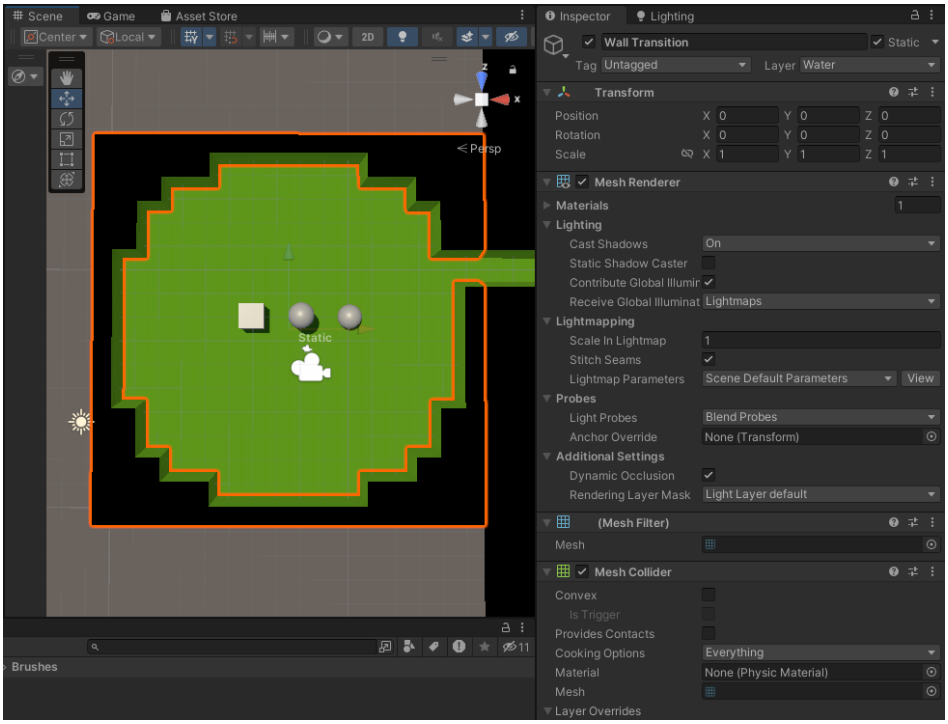
FAQ

▪ How do I add Collisions to the Walls?

You can enable the **"Apply Mesh Collider at Runtime"** option on all your **Wall Brushes**!



If you enable this option all the **Chunks** created by this **Dungeon Brush** will have a simple **Mesh Collid**er attached to them.



This is both applicable to **Main Tiles** and **Transition Tiles**.

▪ How do I add Custom Mesh Colliders to my Walls?

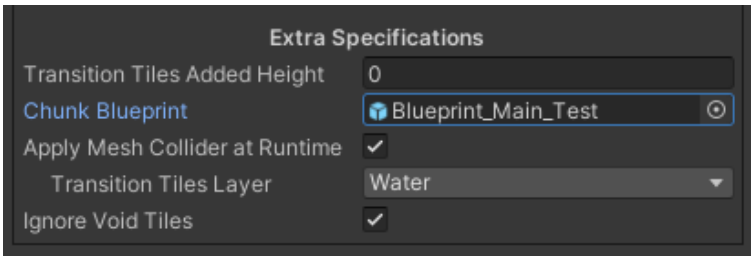
ADG's main priorities are **a stable performance** and **fast loading times**, to achieve these goals the individual aspects of all your given assets have to be sacrificed to make way for the formation of a cumulus, a conglomerate of parts, i.e., a **Chunk**.

You have to understand that *ADG* **does not** generate the **Dungeons** block by block, it actually uses your given assets to create giant individual blocks, i.e., **Chunks**.

All **Prefabs** lose its individual components in order to form a Chuck, if you want to add custom components to your **Dungeon** you will have to add custom components to your **Chucks** themselves.

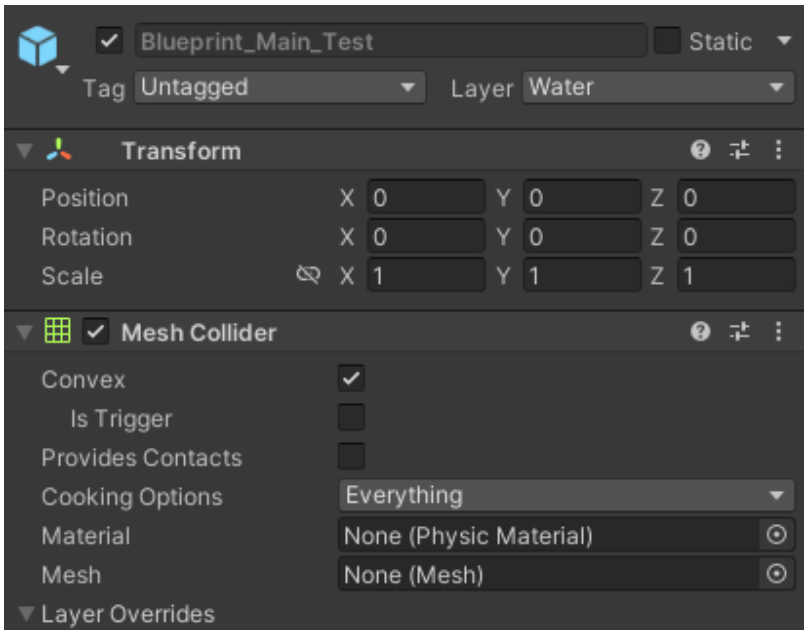
▪ How do I add components to the Chunks themselves?

Using **Blueprints**.



Blueprints are **Prefabs** that serve as **Chuc**k templates. At runtime, all components of the given **Prefab** will be copied into their respective tile clusters.

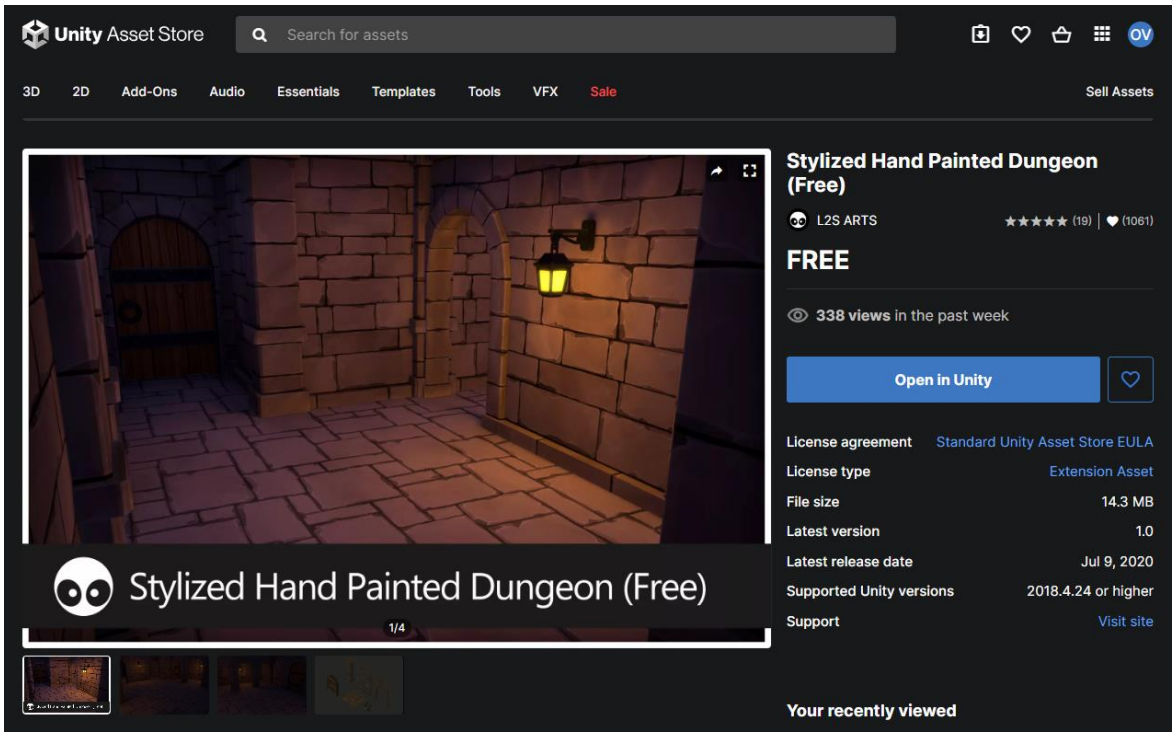
If you want to add a custom **Mesh Collider** to your **Walls**, you will have to create a new **Prefab** containing that custom **Mesh Collider** and assign it to your **Dungeon Brush**.



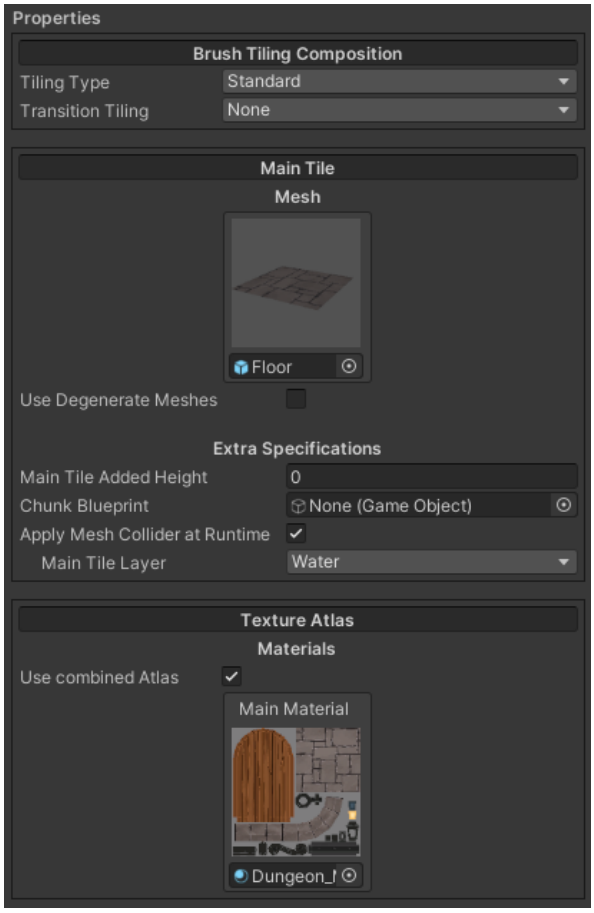
▪ Are my external assets compatible with ADG?

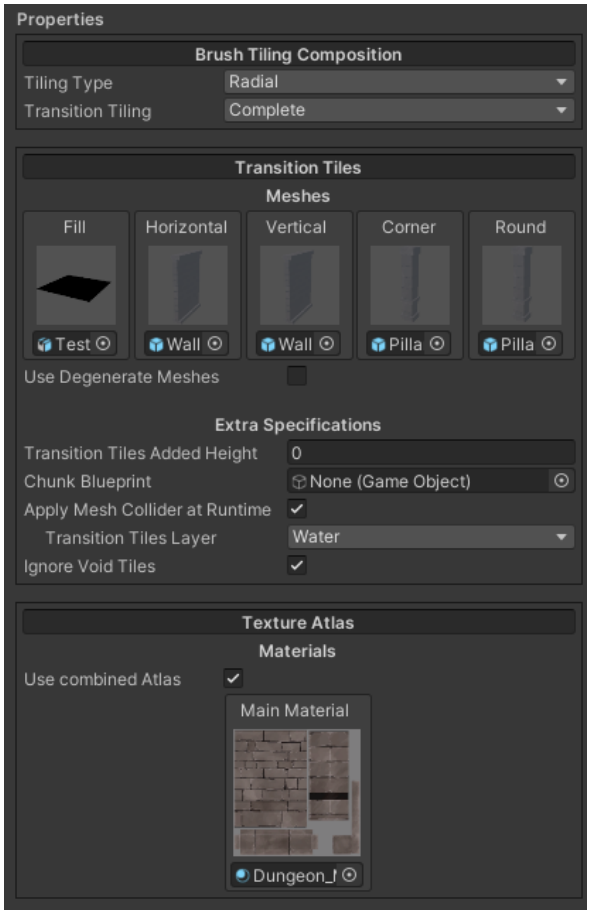
Yes, however, *ADG* requires some help from you, as the meshes you provide must meet certain requirements in order to function properly.

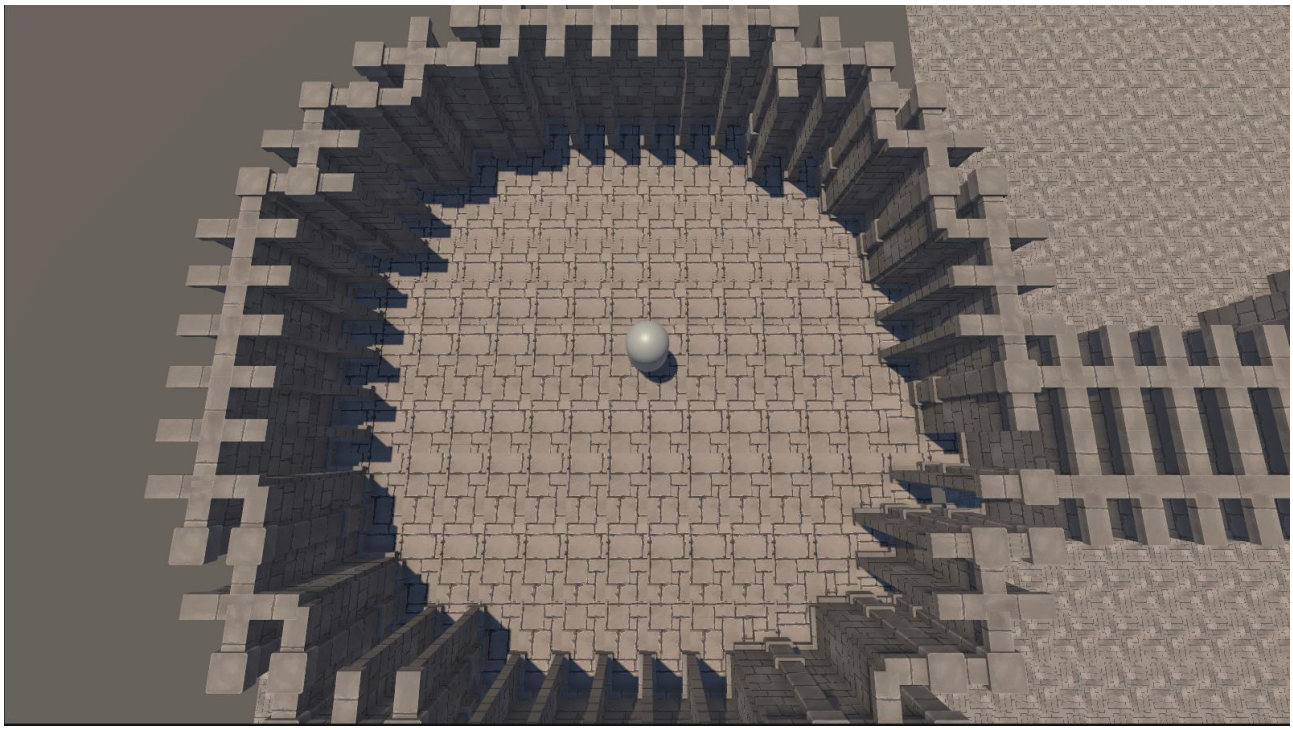
For example, I downloaded this amazing package from the Unity Asset Store:



But if I just assign **Prefabs** blindly, I end up with this **mess**:







All the elements used by *ADG* need to meet certain criteria.

For example, all your **Dungeon Brushes Transition Tiles** must comply with these specifications:

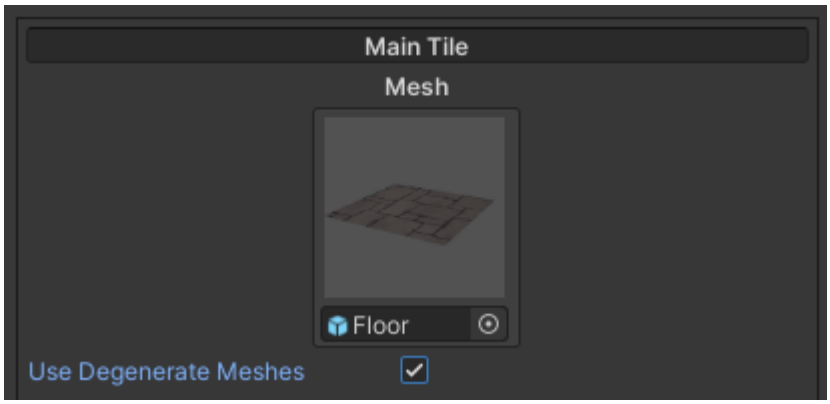
- The pivot point of the mesh must be in (0, 0, 0). *(Can be omitted on Degenerate Meshes)*
- The **actual mesh** should only be in the Unity (+X, +Z) quarter.
- The mesh X, Z dimensions should be (0.5 m x 0.5 m) maximum.
- All the meshes had to have radial symmetry. *(Can be omitted on Degenerate Meshes*

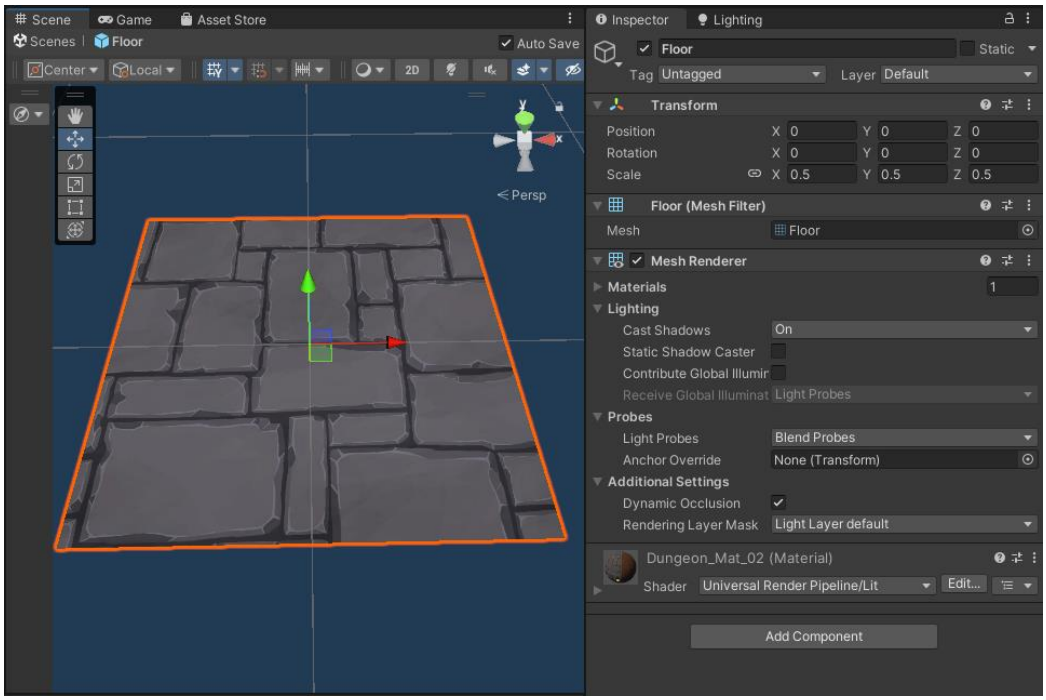
*All off this is specified in the **Dungeon Brush** section.*

And since *ADG* works on your meshes themselves you may have to modify your assets in external tools like **Blender**.

Even so, *ADG* offers you a workaround for these situations:

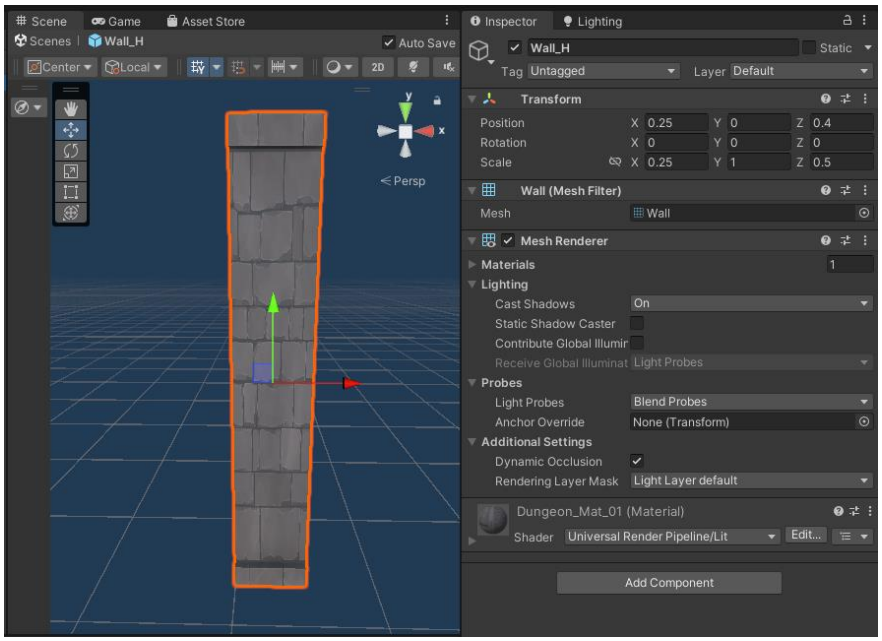
- If you enable the option **"Use Degenerate Meshes"**, you can change the **Prefab**s themselves to modify the creation of the Dungeon!

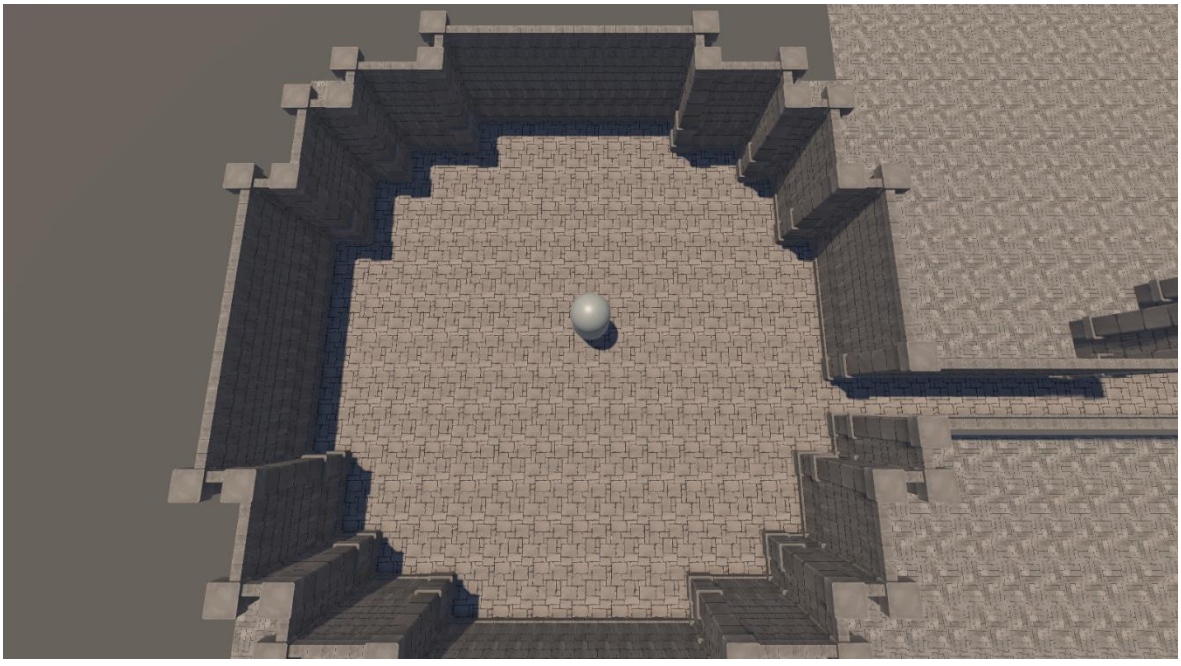




With this option enabled, you can easily create and modify **Prefabs** to meet the previously mentioned criteria:

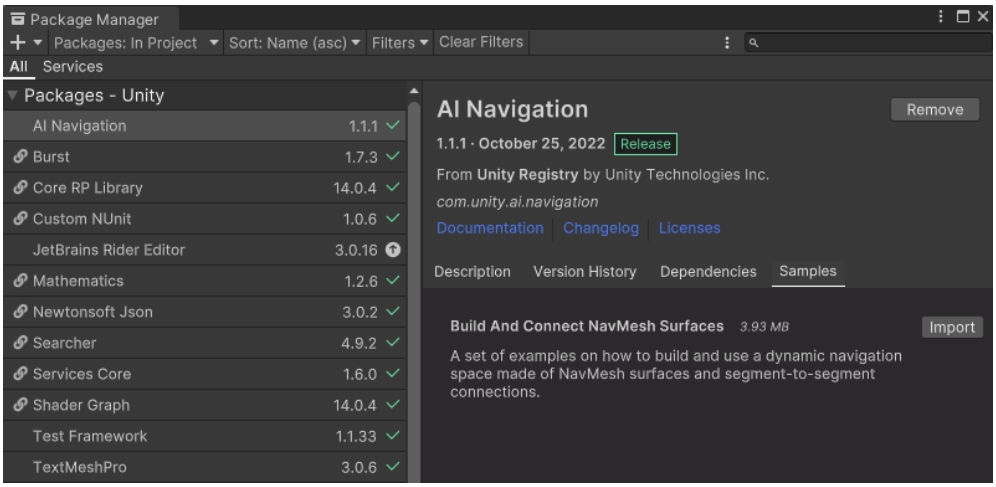
- The **actual mesh** should only be in the Unity (+X, +Z) quarter.
- The mesh X, Z dimensions should be (0.5 m x 0.5 m) maximum.



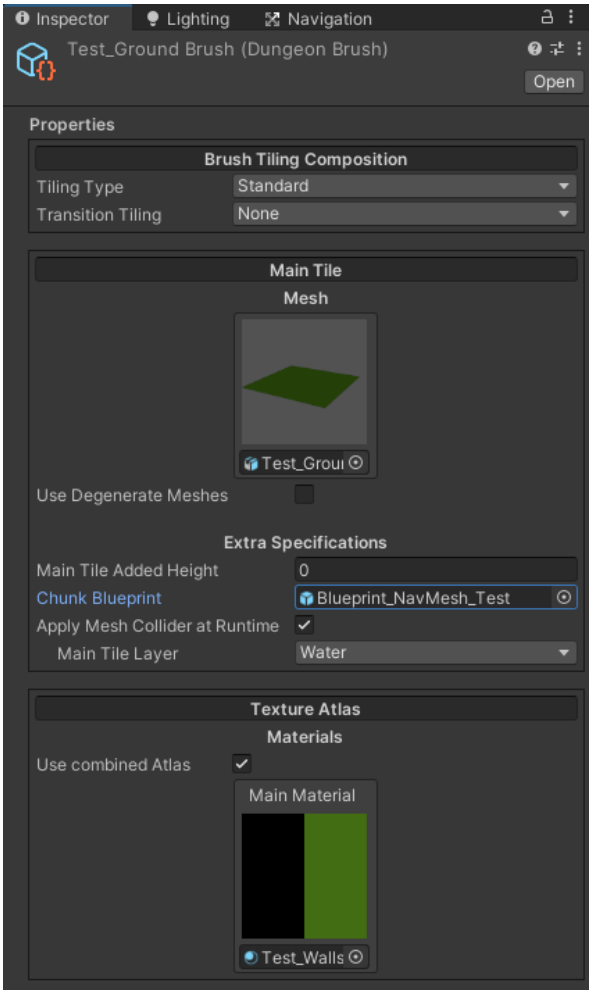


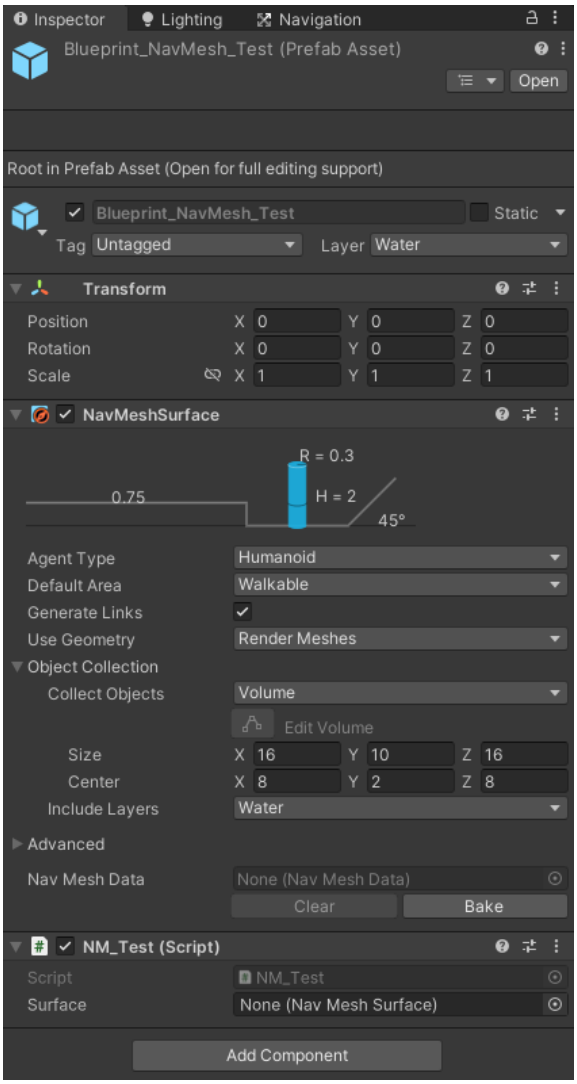
▪ How do I add a NavMesh to the Dungeon?

Please note that the default **NavMesh** version of Unity is extremely basic and will be deprecated. If you want high-level **NavMesh** components for building and using **NavMeshes** at runtime Unity literally forces you to download the **AI Navigation** package.



With this package, you can use **Blueprints** and a simple script to generate **NavMeshes** at runtime:

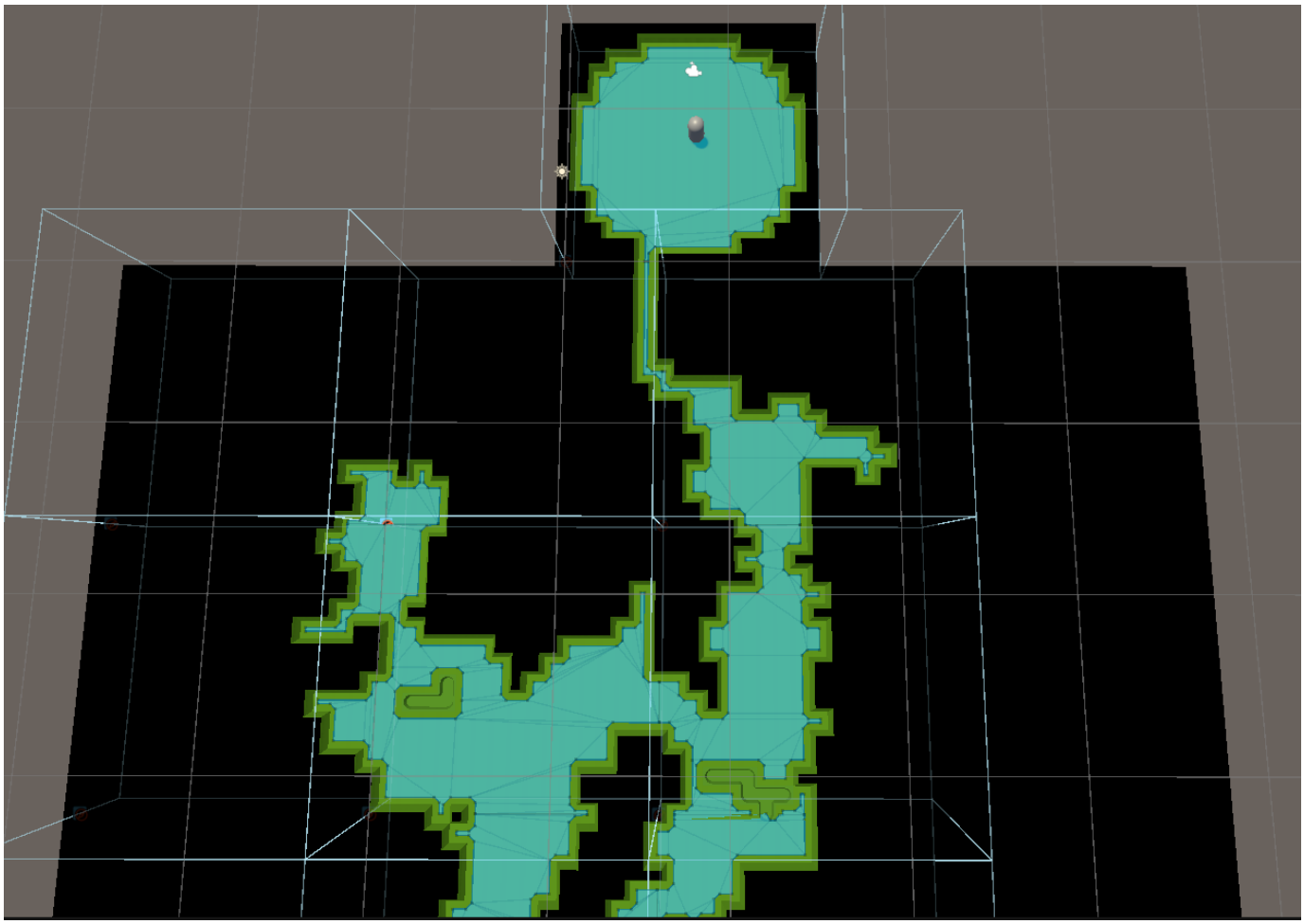




NM_Test.cs | DungeonEditor.cs | DungeonDecorationEditor.cs | MainLibrary.cs

Assembly-CSharp | NM_Test | Start()

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4   using Unity.AI.Navigation;
5
    Script de Unity (1 referencia de recurso) | 0 referencias
6   public class NM_Test : MonoBehaviour
7   {
8       public NavMeshSurface surface;
9
10      // Start is called before the first frame update
        Mensaje de Unity | 0 referencias
11      void Start()
12      {
13          surface = GetComponent<NavMeshSurface>();
14          surface.BuildNavMesh();
15      }
16  }
17
```



Please keep in mind that this **Unity** feature is still in heavy development, and therefore, its results tend to be inconsistent and its **Documentation** is rather lacking. **If possible, use anything else.**