Dungeon Architect for Unity

Getting Started

Create your first Dungeon

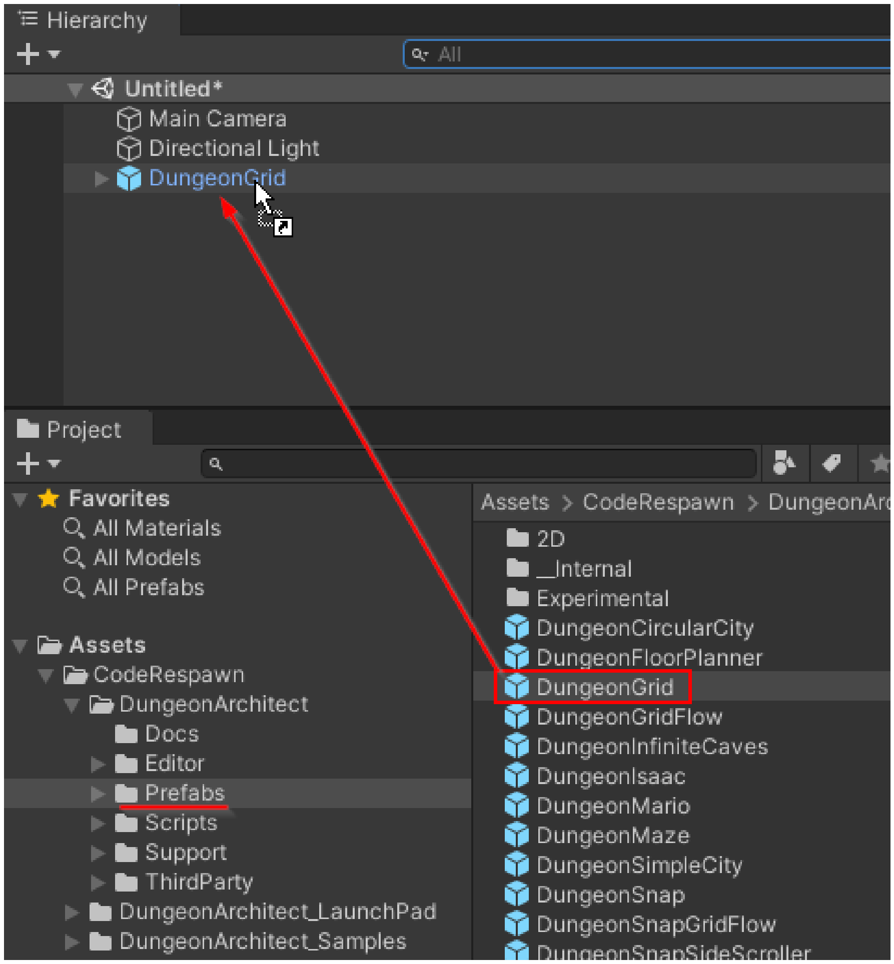
Install Dungeon Architect

Install and import Dungeon Architect from the Asset Store. You should see the following folders:

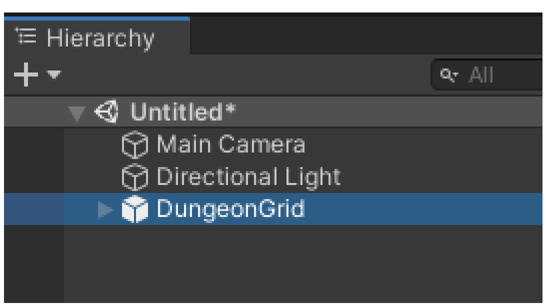
Setup Dungeon Prefab

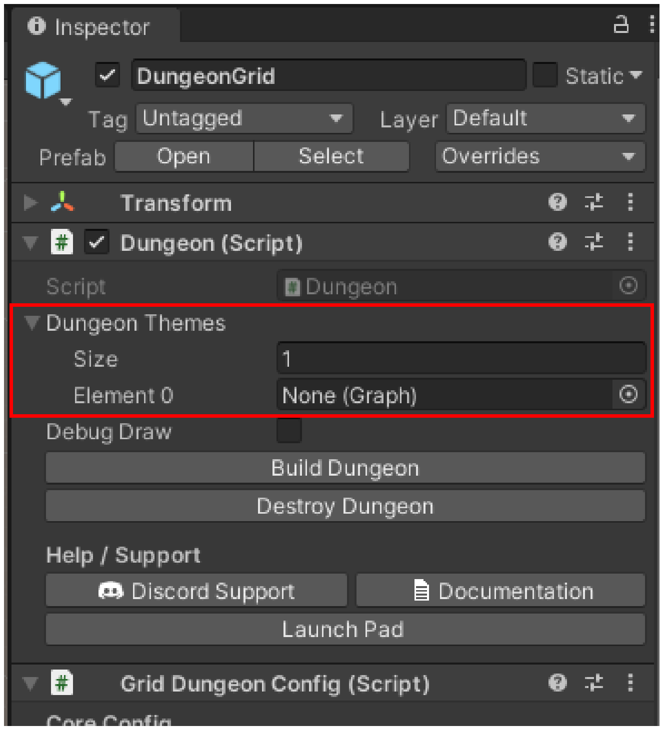
Create a new Scene

Navigate to `CodeRespawn > DungeonArchitect > Prefabs` and drop in the `DungeonGrid` prefab on to the scene



Select the dungeon game object and inspect the properties. We'll need to assign a new theme before we can build the dungeon

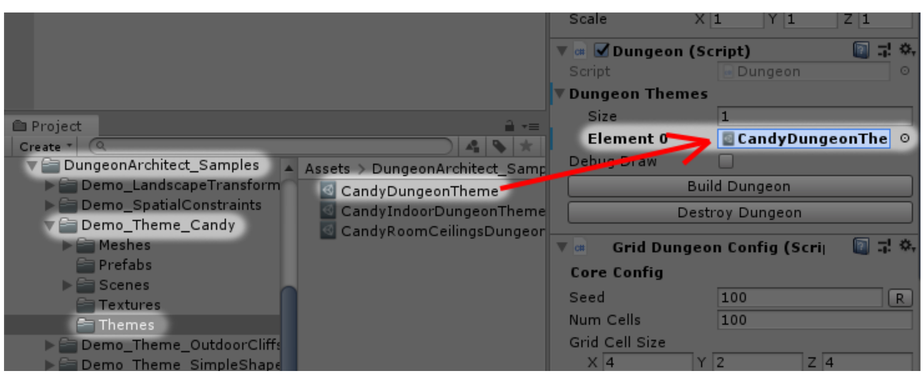




Assign Theme

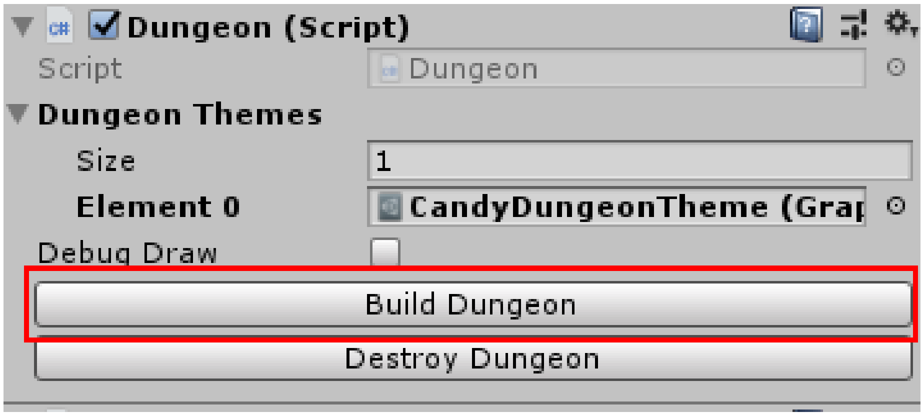
Assign an existing theme to the dungeon actor

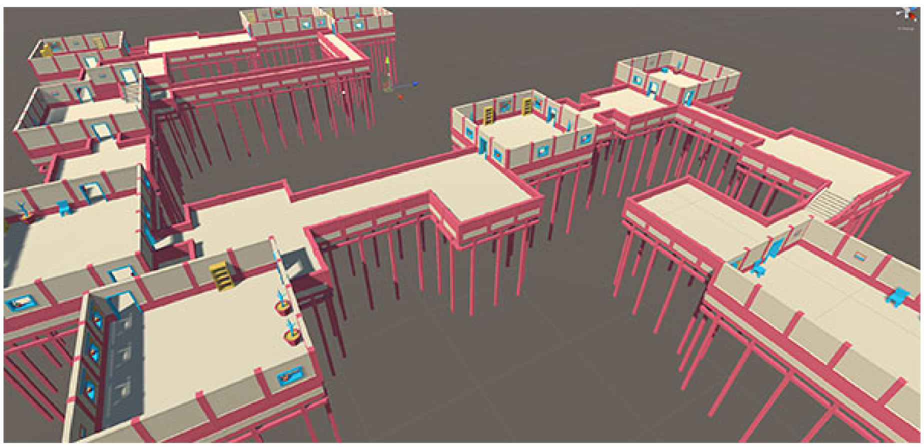
- Navigate to `Assets\DungeonArchitect_Samples\Demo_Theme_Candy\Themes`
- Assign theme file named `CandyDungeonTheme` as shown in the image below



Build Dungeon

Select the `DungeonGrid` game object and click the `Build Dungeon` button in the Inspector window





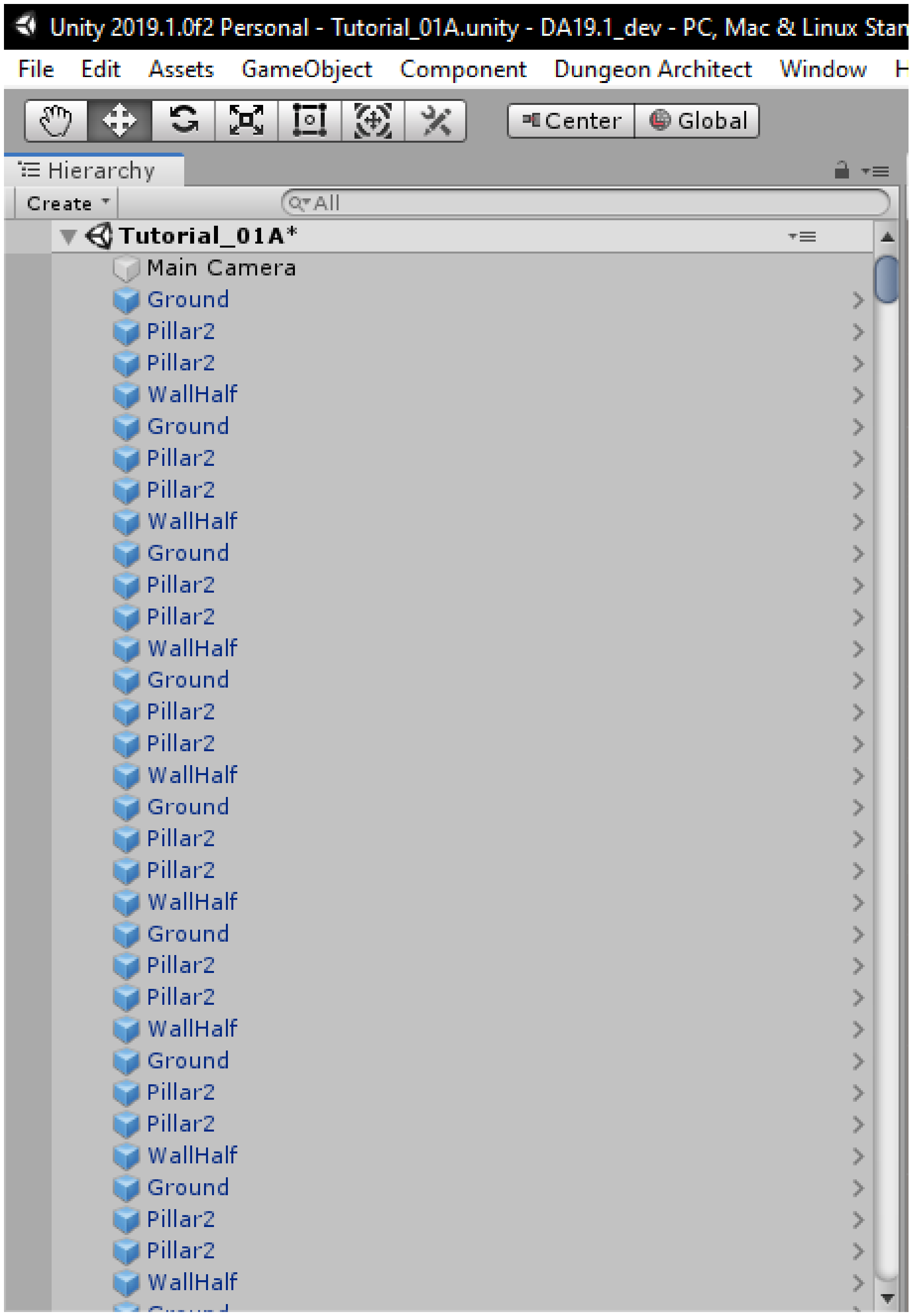
Randomize Dungeon

Select the `DungeonGrid` game object and change the Seed value in the configuration. Changing this value will create a dungeon with a different layout

Click `Build Dungeon` button to rebuild the dungeon with the new seed

Organization

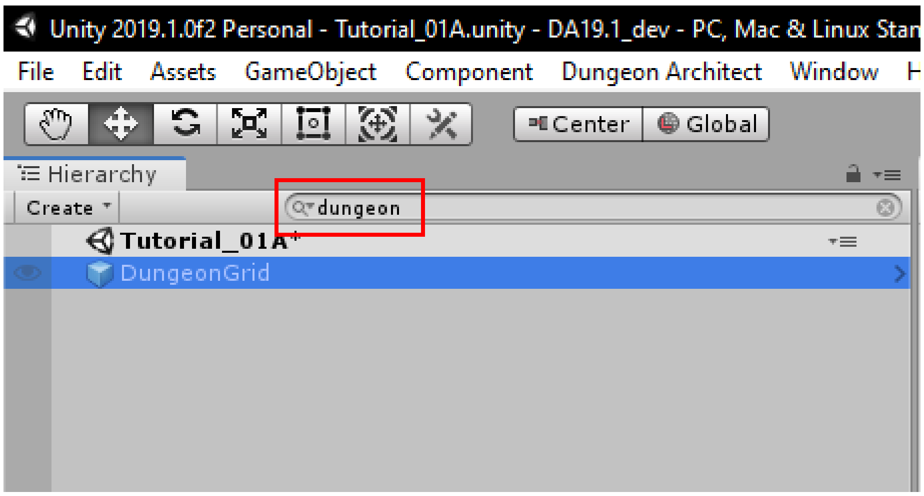
All the dungeon objects are created on the root hierarchy and makes it difficult to organize. We’ll configure it so all objects are spawned under a certain game object



Lets destroy this current dungeon, configure it for better organization and then rebuild

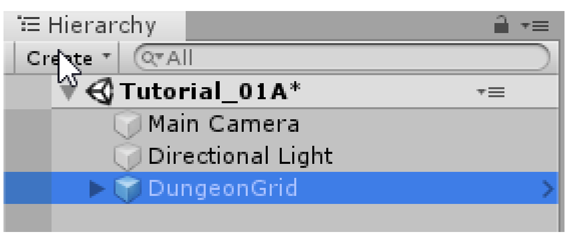
Figure 1: Create an empty game object

Destroy Existing Dungeon Search for `dungeon` on the hierarchy search box



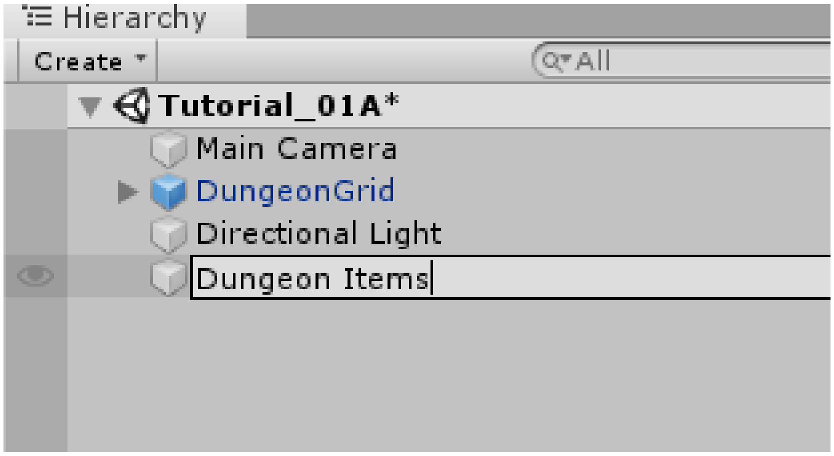
Select the DungeonGrid game object and click the `Destroy Dungeon` button

Clear out the search text box in the hierarchy. Your hierarchy should now look like this

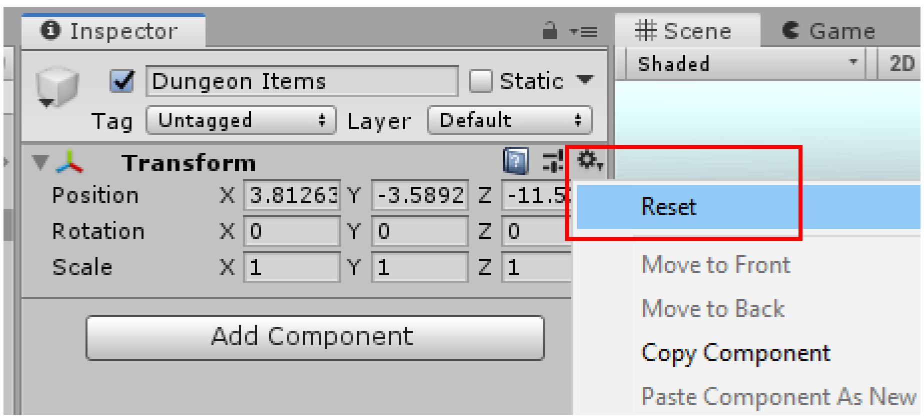


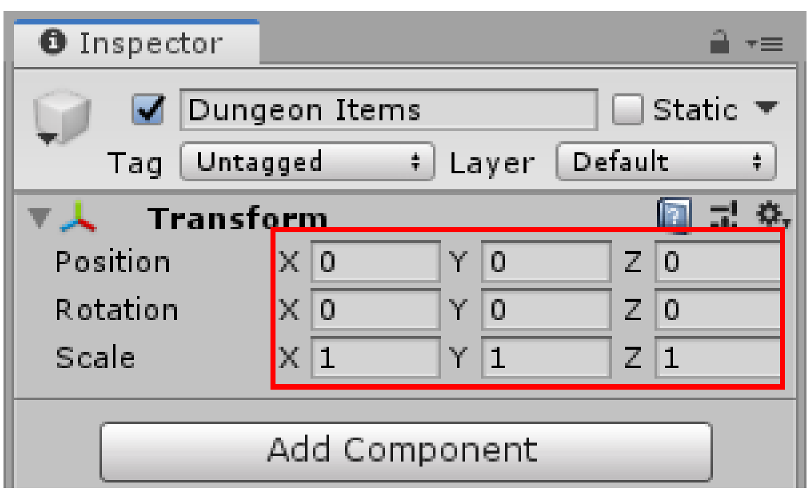
Configure Parent Object Create an empty Game Object. All our dungeon items will go inside this parent object

Rename the parent object (e.g. `Dungeon Items`)

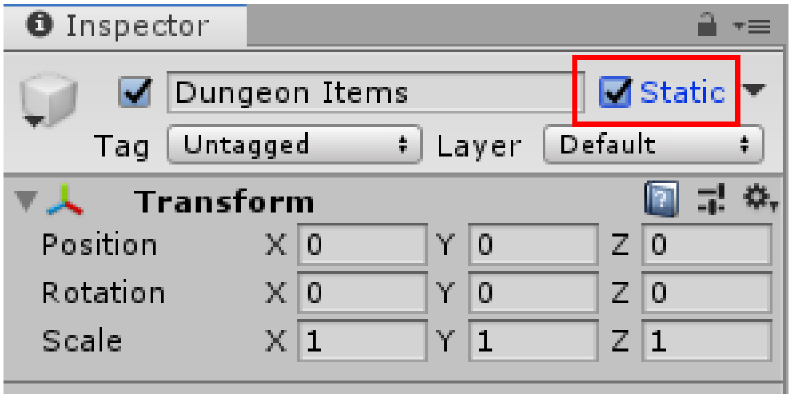


Select the parent object and **Reset the transform**

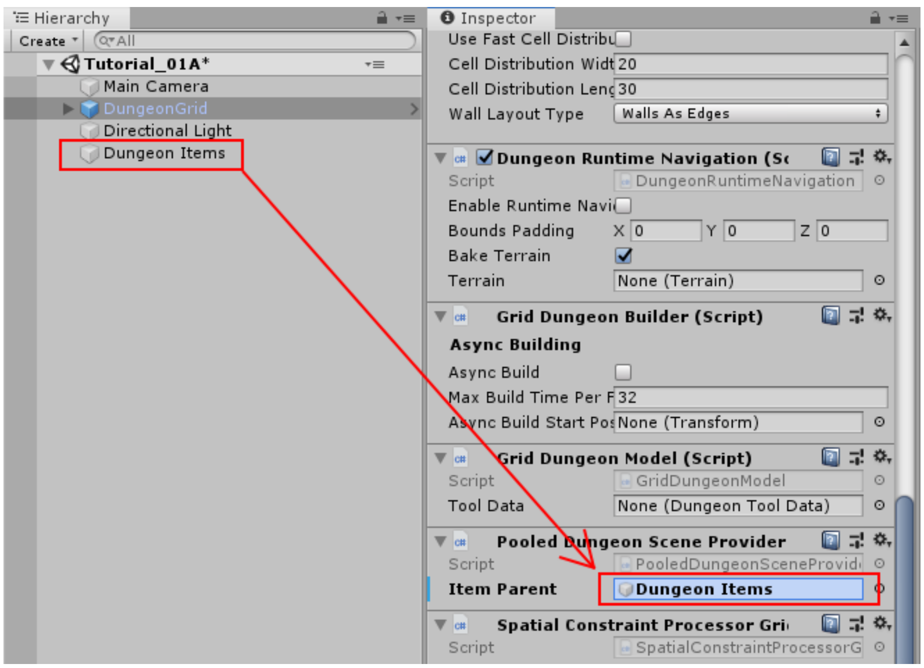




Select the parent object and set it to **static**

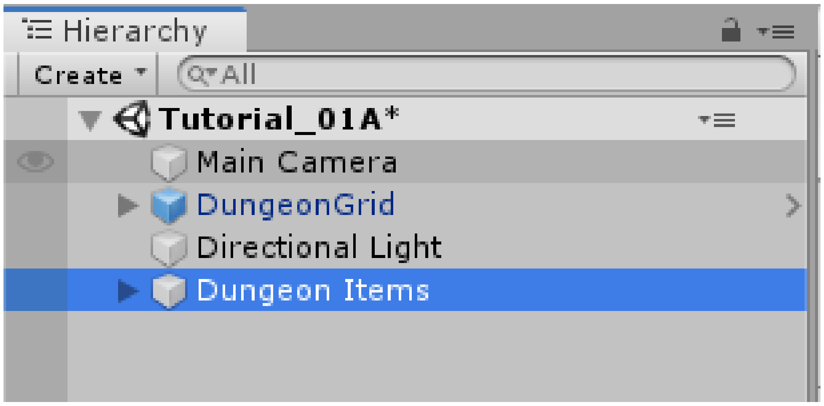


Assign the parent object to the GridDungeon game object



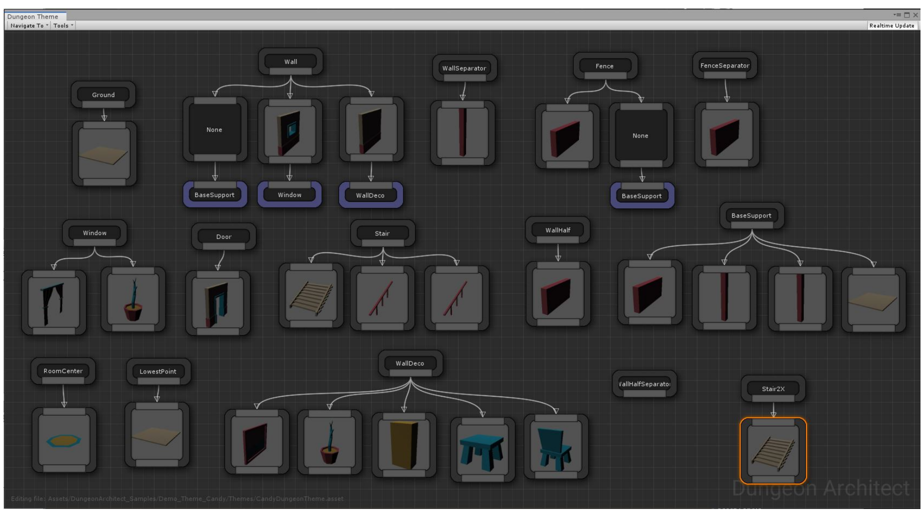
Rebuild Dungeon Select the GridDungeon game object and click `Build Dungeon`

All your dungeon game objects will be organized under the parent object



Design your first Theme

A theme file is a mapping between **marker** names (like Walls, Ground, Door etc) and the meshes that you provide. The prefabs you map here will be used to build your dungeon



In the previous section, we created a dungeon with an existing theme file (CandyDungeonTheme). In this section, we'll create one ourselves

Create a Theme File

Create a new theme file from either the Main menu or the Create menu

This will create a theme file in the current open directory in the Projects window

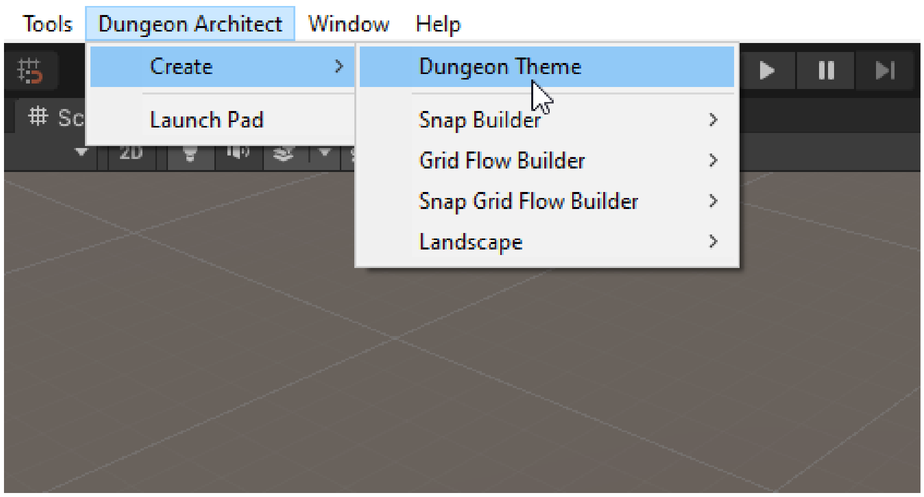


Figure 2: Create from Main menu

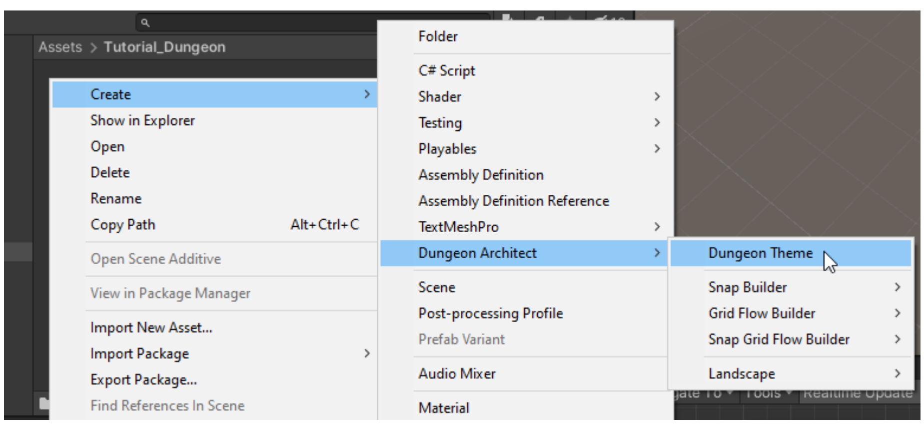
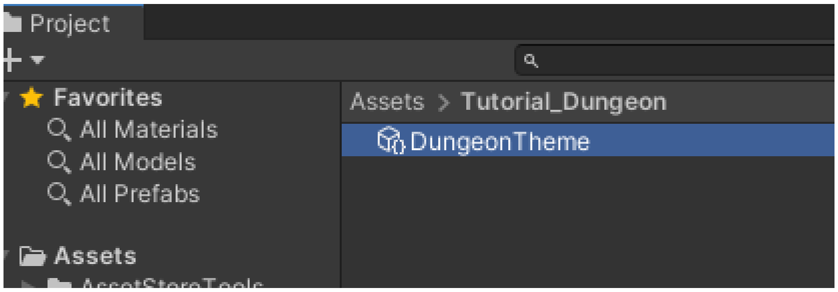
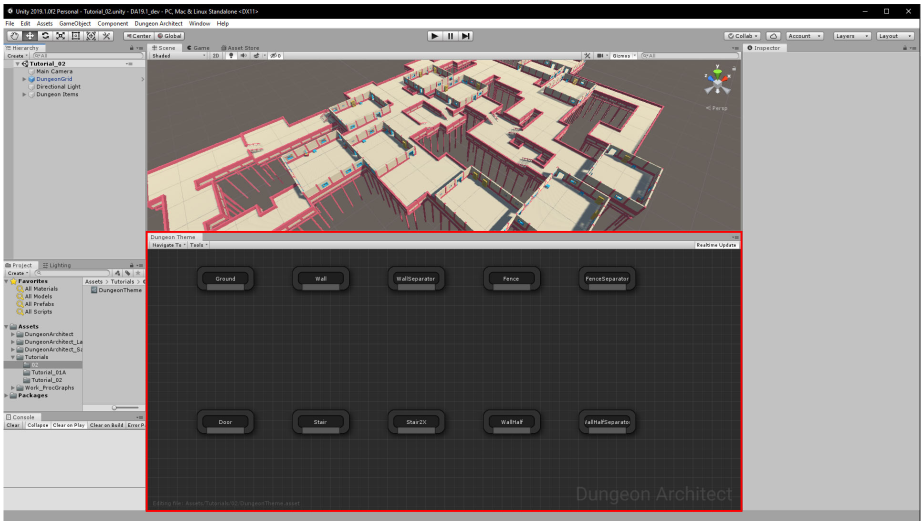


Figure 3: Create from Create menu



Open the Theme File

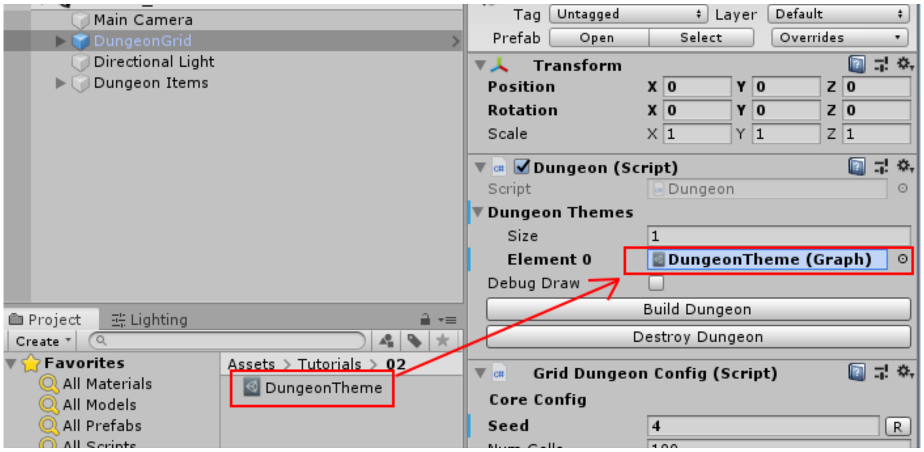
Double-click the theme file to open the **Theme Editor**. Dock the theme editor so you can see both the Scene view and the Theme Editor at the same time



Assign the Theme File

Destroy the existing dungeon by selecting `DungeonGrid` game object and click `Destroy Dungeon` button

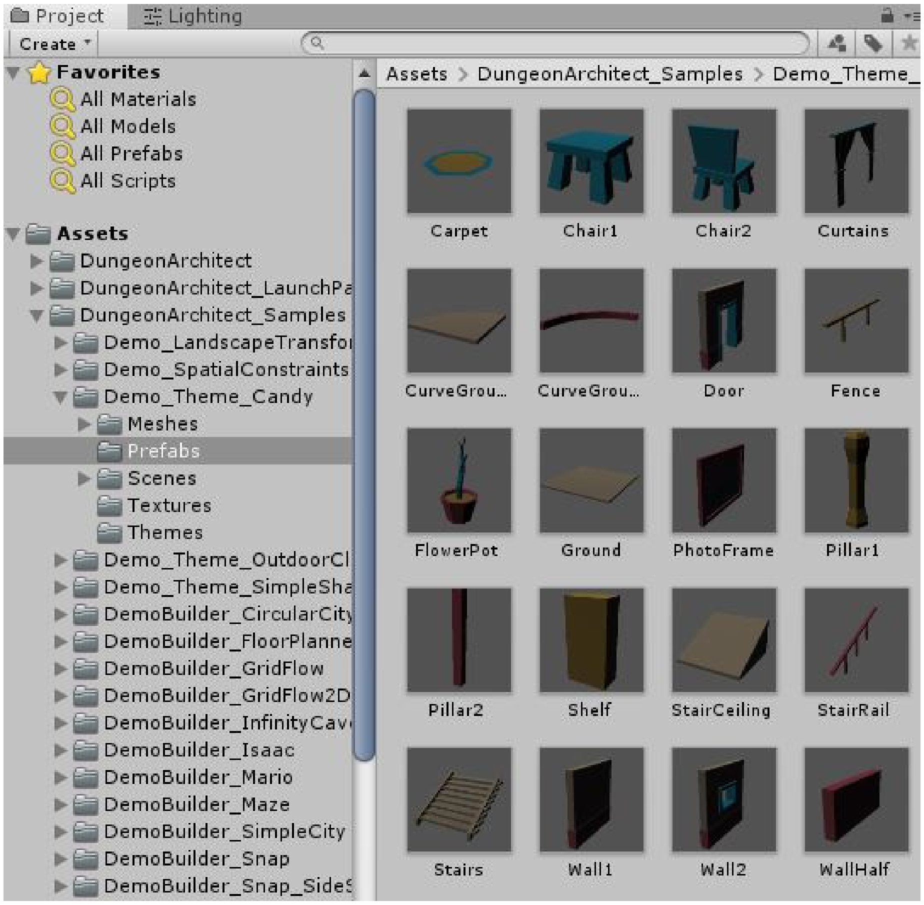
Assign the theme file that you just created, on to the DungeonGrid game object



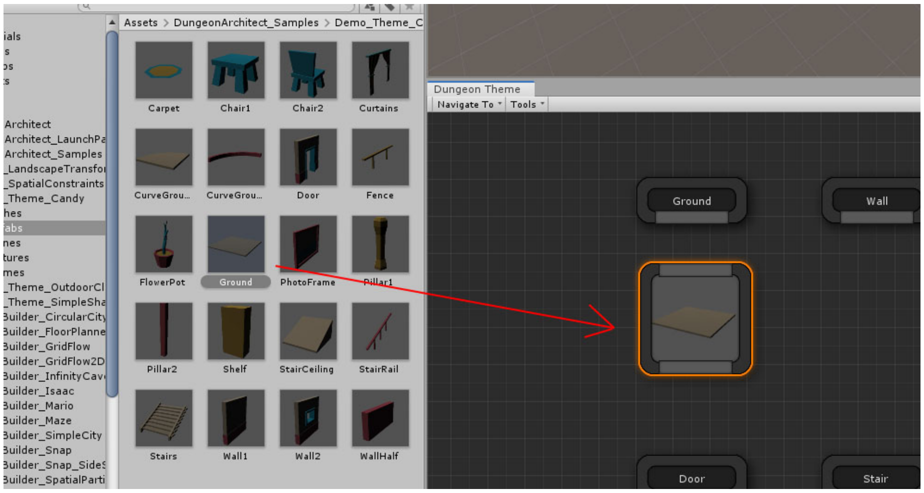
Click `Build Dungeon` and nothing should appear. That's because we haven't added prefab mappings on the theme yet

Design the Theme

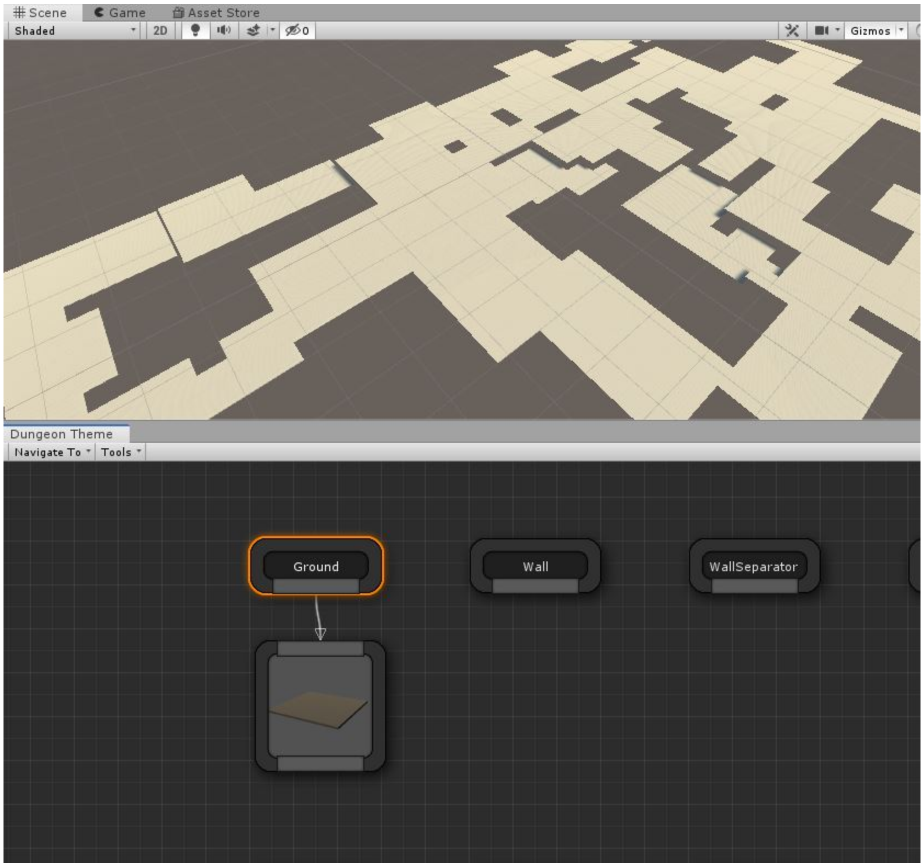
Navigate to `Assets\DungeonArchitect_Samples\Demo_Theme_Candy\Prefabs`. This folder contains a set of mesh prefabs we can use for our dungeon



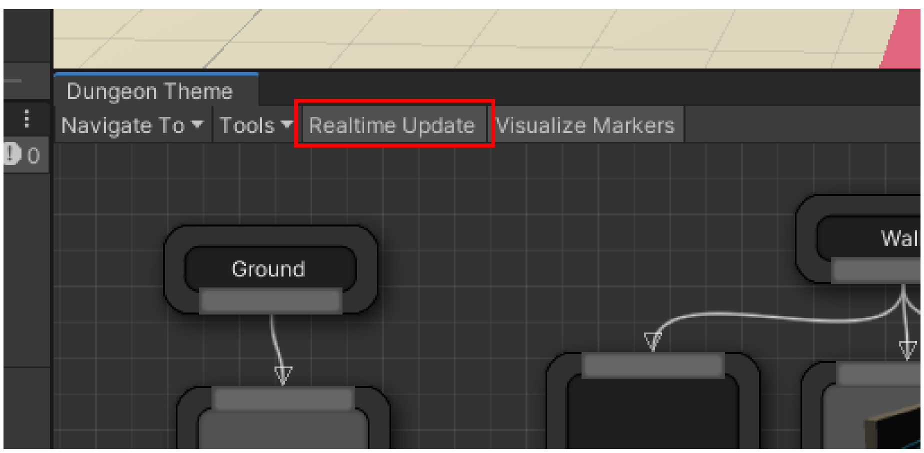
Add Ground Drag-drop the ground prefab mesh on to the Theme Editor



Link up the mesh node with the `Ground` marker node. When you do, you should see a live preview on the scene view with the dungeon using this ground mesh

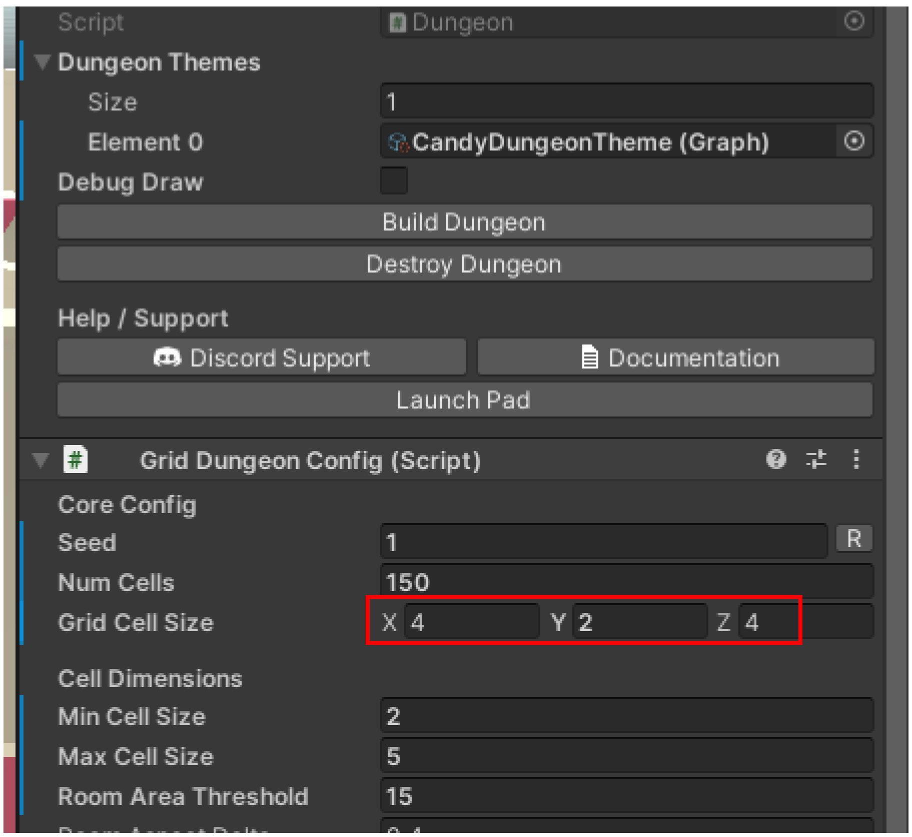


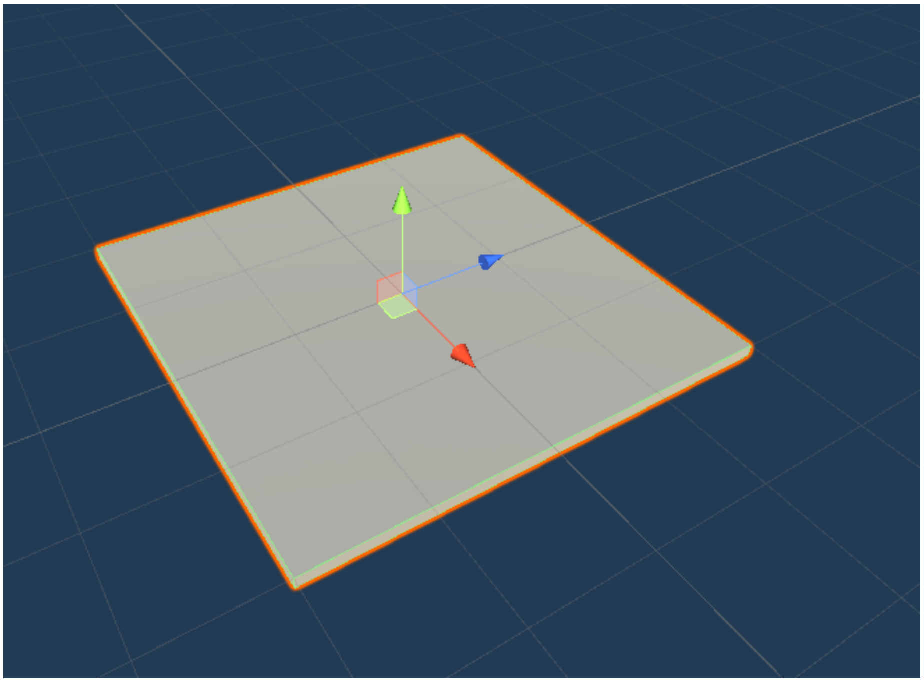
For the live preview to work, make sure the “Realtime Update” button is enabled in the Theme Editor



Another criteria for the live preview to work is that a dungeon in the scene should reference the theme that is currently being edited in the theme editor

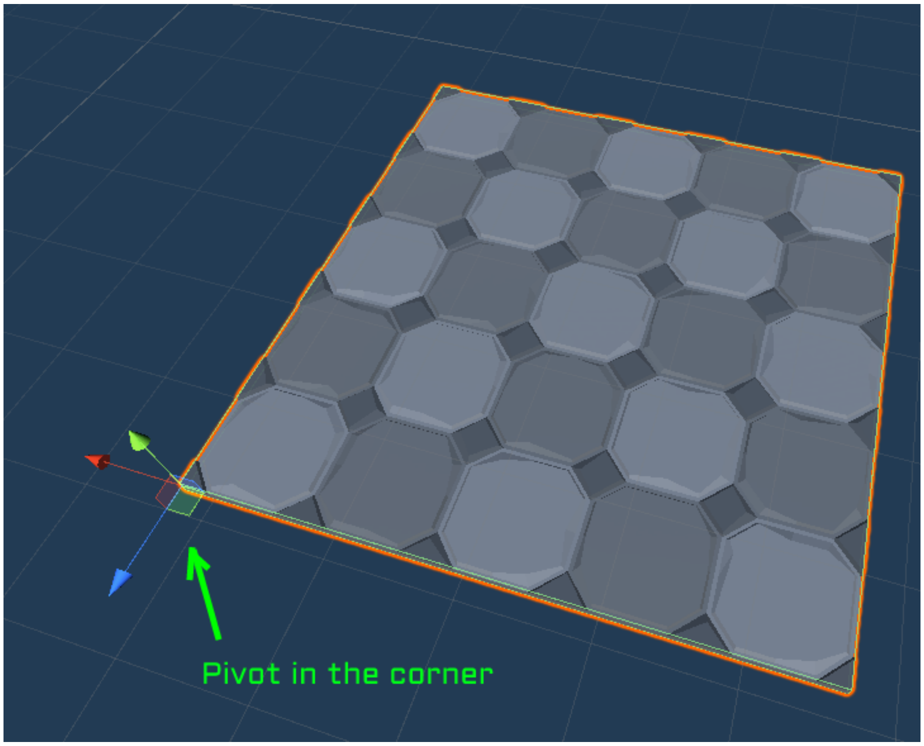
Adjust Object Transform Dungeon Architect can adapt to any modular asset. In our example, the dungeon's grid size is set to `(4, 2, 4)`. The ground prefab that we used has the same size (4x4) and the pivot is in the center. So it will fit nicely without any change





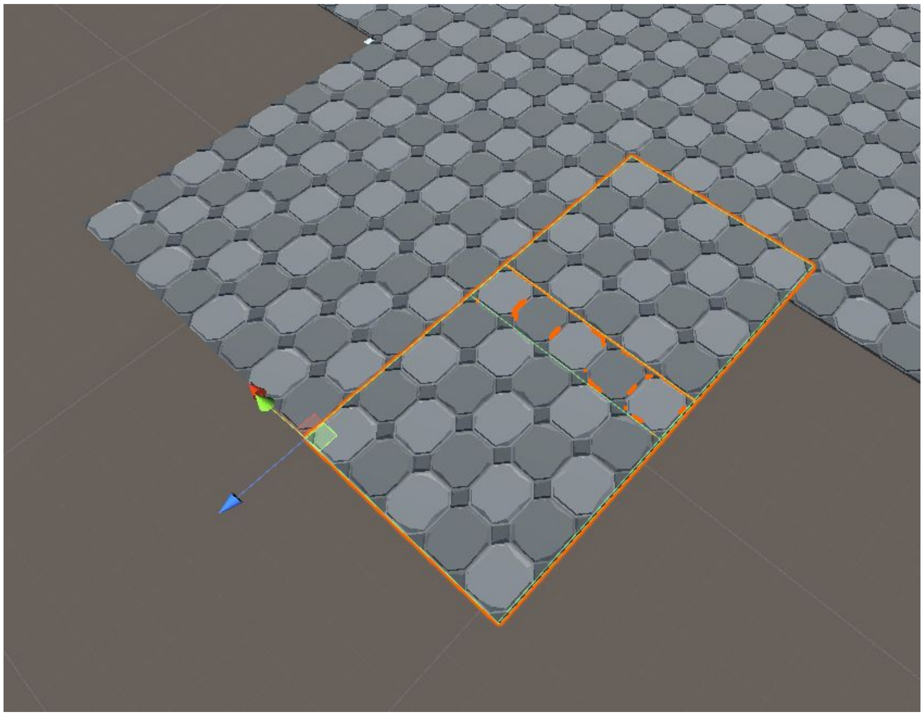
However, this might not always be the case. Your prefabs might have its pivot in a different position (like in the corner) and the size may be different than the grid size we've defined in the dungeon game object.

Here's an example of a modular ground prefab from one of the Synty asset.



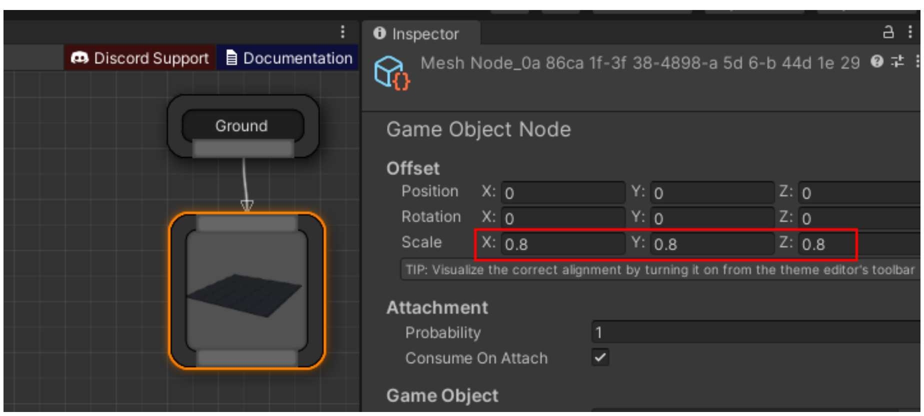
The Pivot is in the corner and the size of the tile is 5x5 units

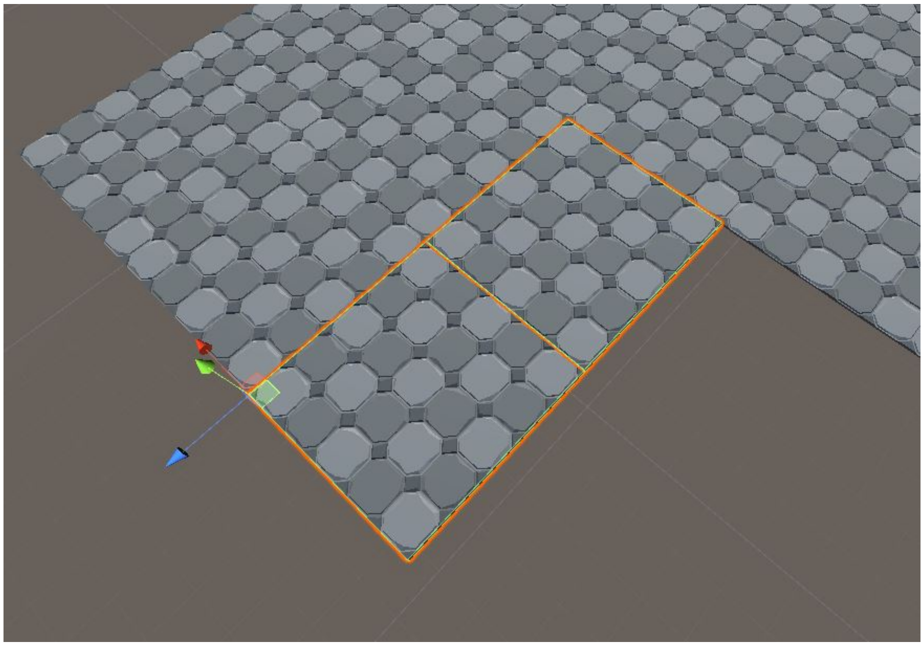
If we were to drop this in the theme editor and link it to the ground marker, we'd see this



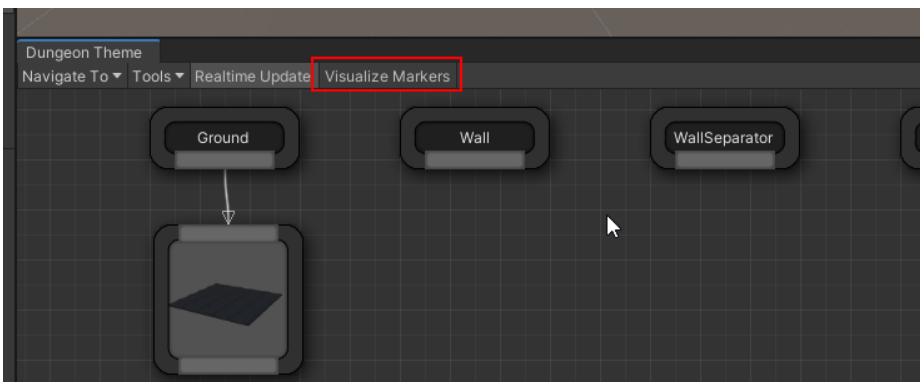
The tiles overlap since the asset is bigger than the grid size (we chose the grid size to be 4 and the asset size is 5)

Select the mesh node and set the scale to `0.8` *(since 4/5 = 0.8)*

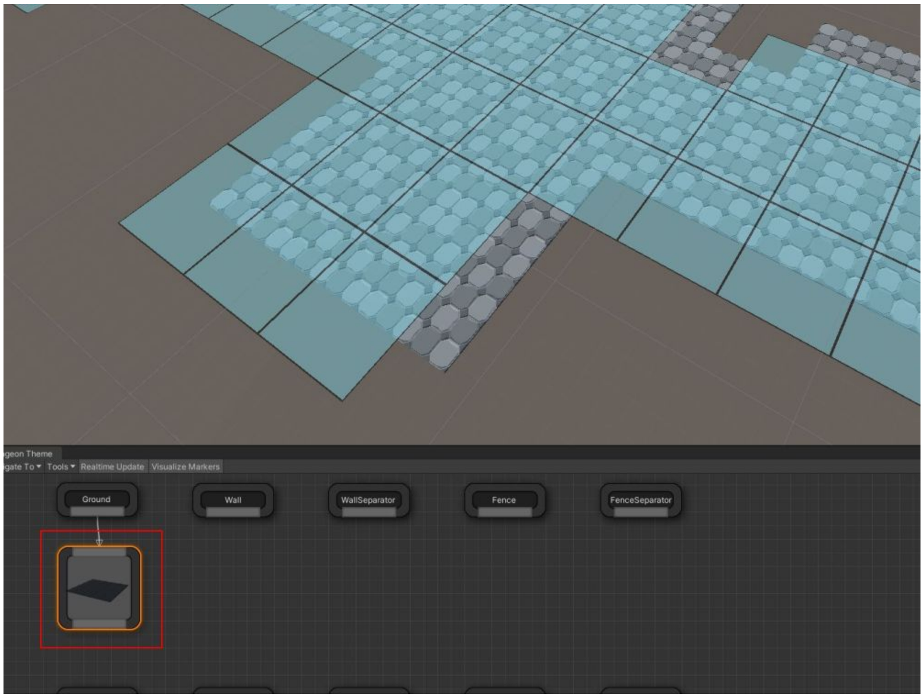




Now we need to align the position. Click `Visualize Markers` from the theme editor's toolbar

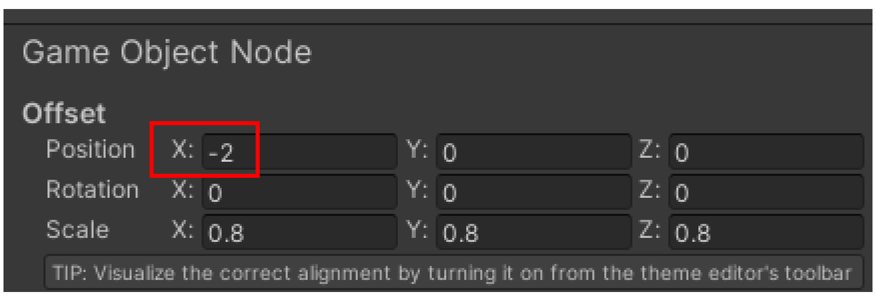


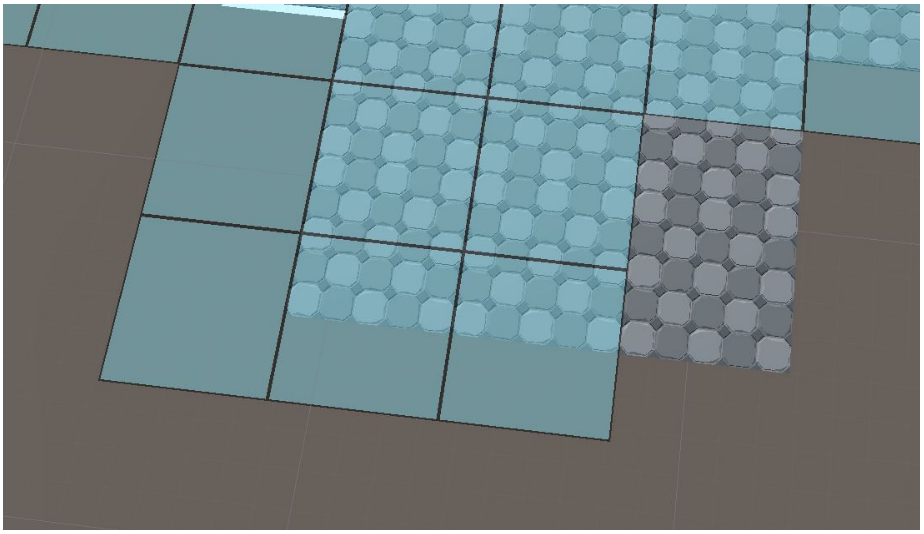
Select the ground prefab node or the `Ground` marker node to visualize the correct position of where your meshes should be



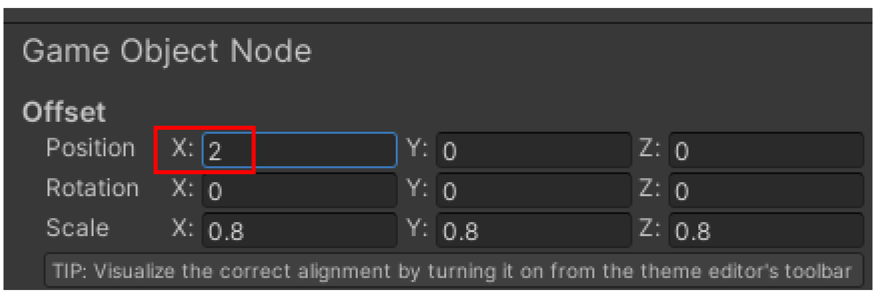
Use a corner as a reference for alignment, like in the image above

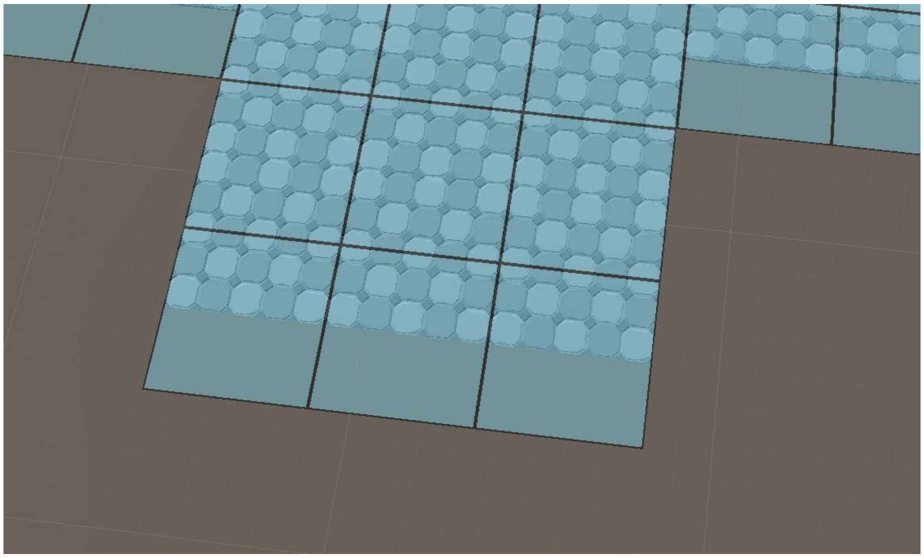
We'll move it along X by -2 and see where it goes



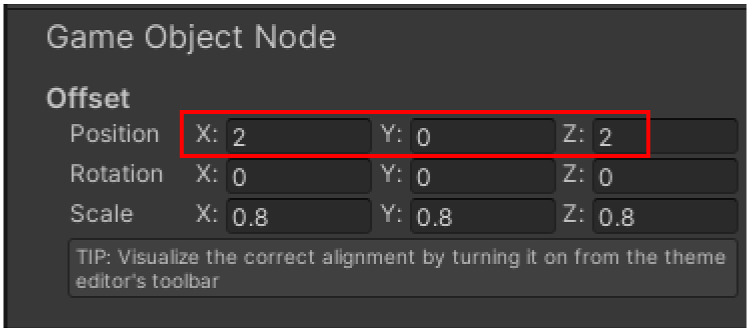


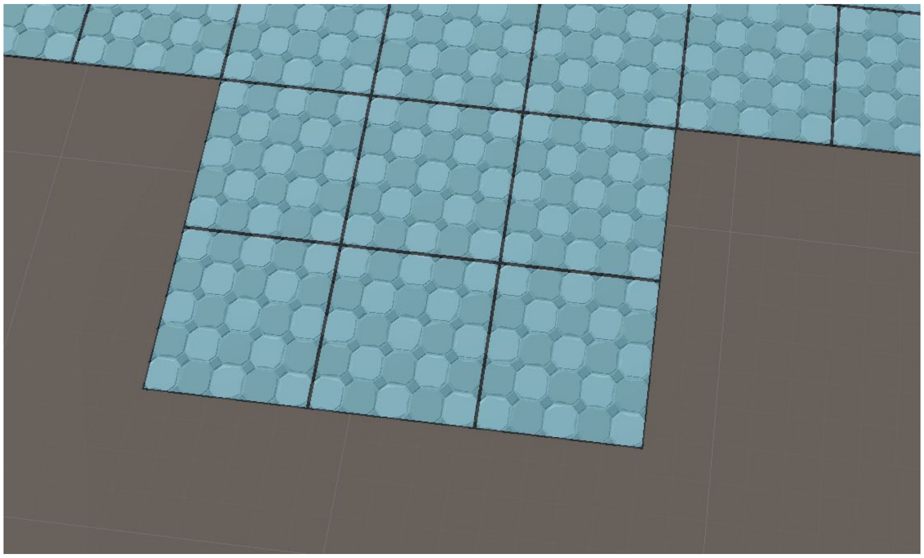
That doesn't seem right. Move the X along 2 instead and it fits correctly along the x-axis





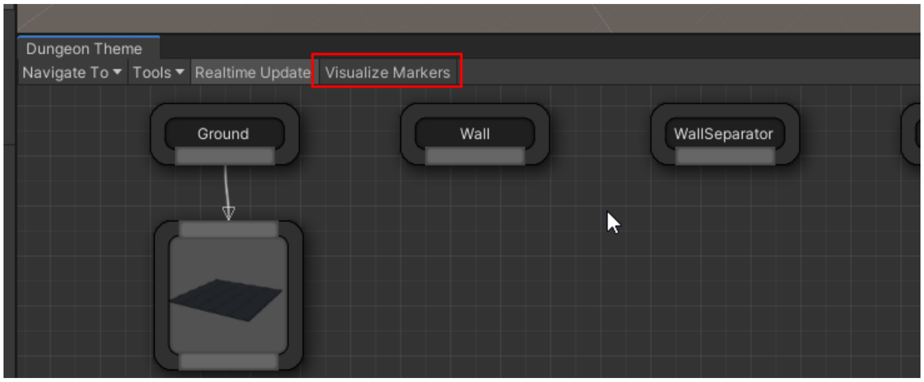
Move along Z by 2 units to bring it to the correct position

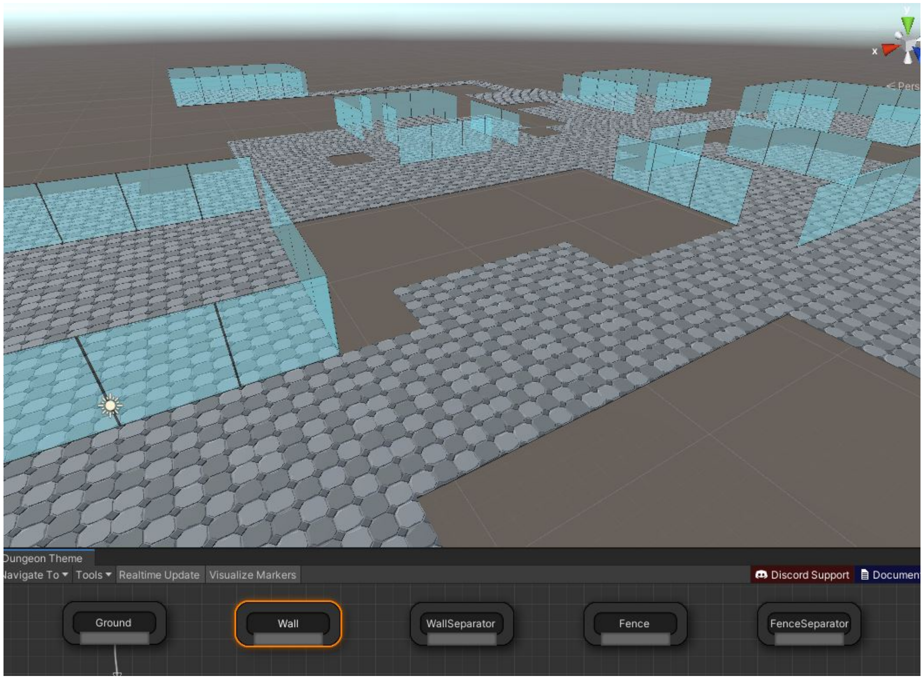


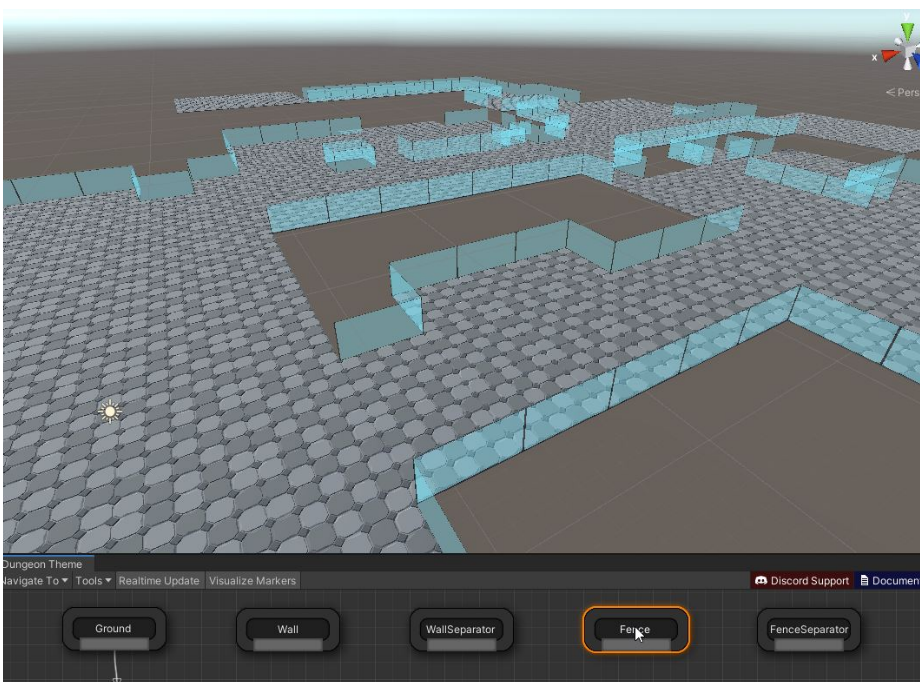


Marker Visualizer As we've seen in the previous section, the theme editor's *Marker Visualizer* is a useful feature while designing your dungeons. It will show the alignment guides whenever you select a built-in marker node (like Ground, Wall etc) or any node connected underneath it

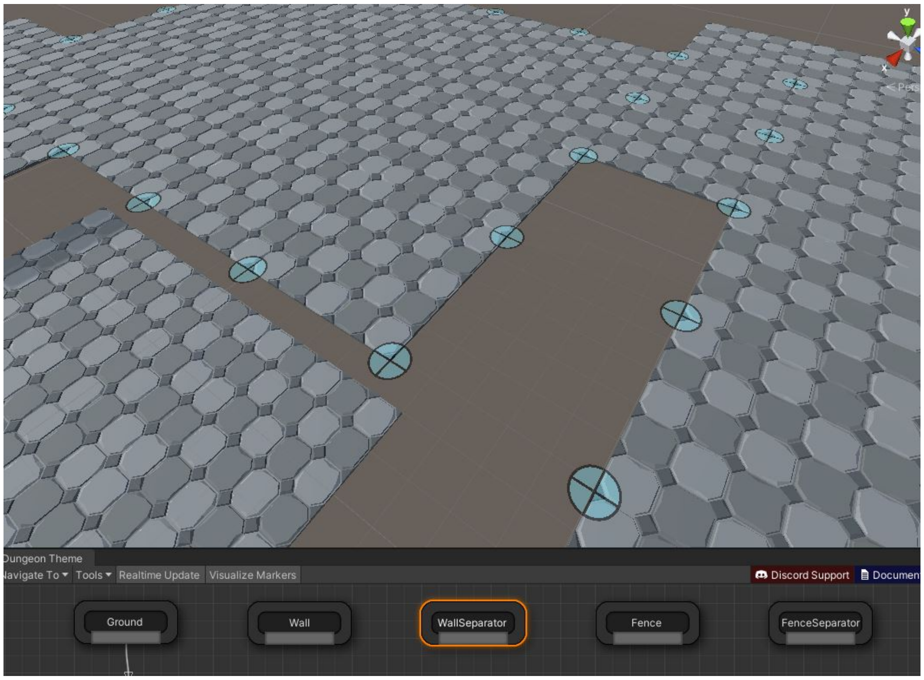
Turn it on from the Theme Editor's toolbar and select a built-in marker node or any node underneath it to visualize that marker's expected alignment



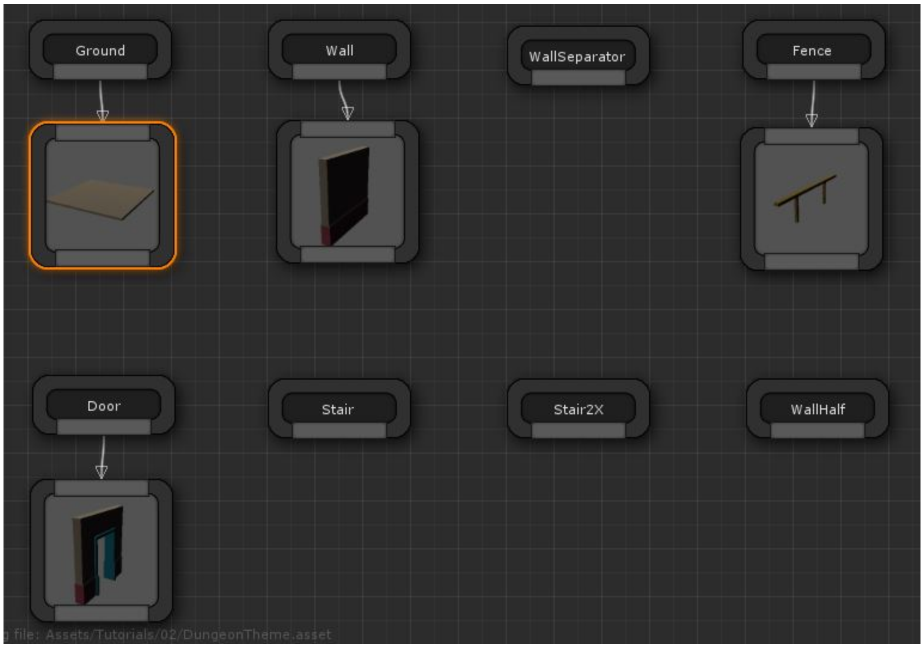

Dungeon Theme
Tools
Realtime Update
Visualize Markers
Discord Support
Ground
Wall
WallSeparator
Fence
FenceSeparator

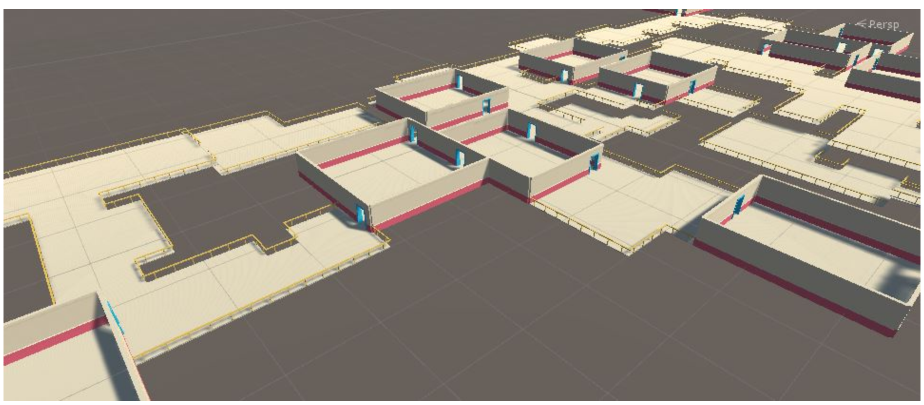

Dungeon Theme
Tools
Realtime Update
Visualize Markers
Discord Support
Ground
Wall
WallSeparator
Fence
FenceSeparator


Dungeon Theme
Navigate To
Tools
Realtime Update
Visualize Markers
Discord Support
Documen
Door
Stair
Stair2X
WallHalf
WallHalfSeparato

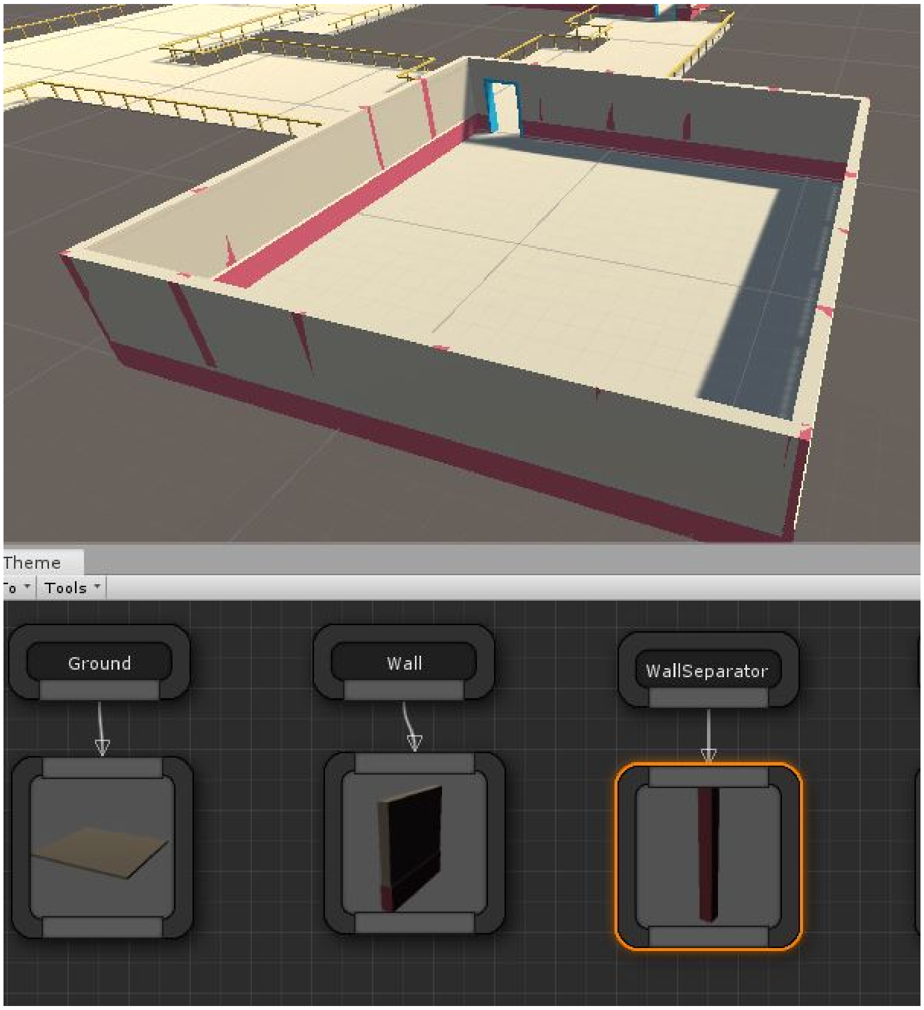

Dungeon Theme
Navigate To
Tools
Realtime Update
Visualize Markers
Discord Support
Documen
Ground
Wall
WallSeparator
Fence
FenceSeparator

Add More Prefabs Go ahead and add prefabs under the following markers: **Wall**, **Fence** and **Door**

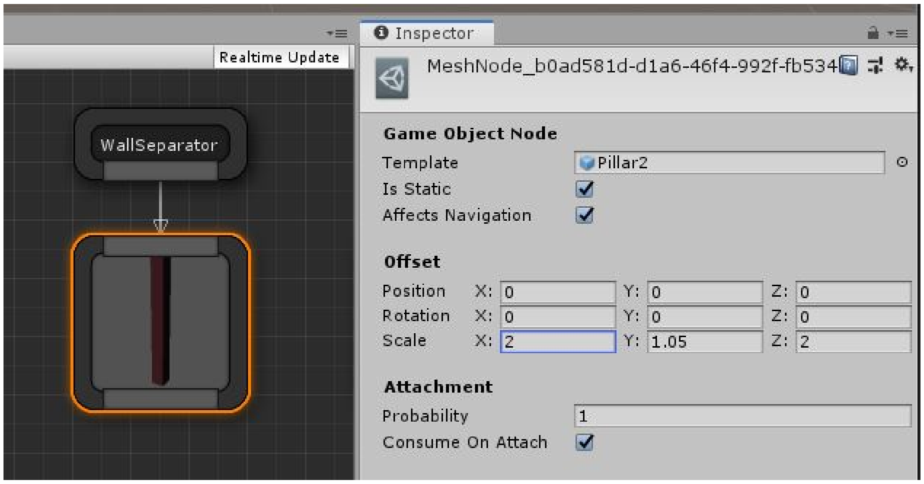


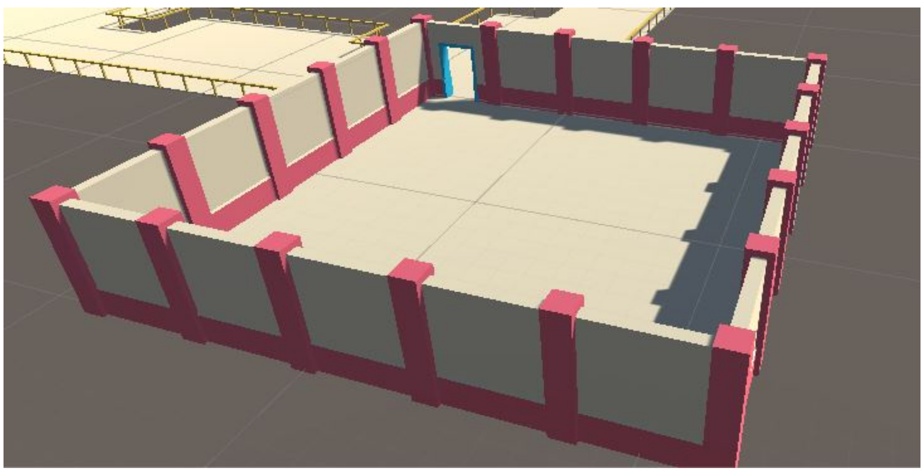


Add Wall Pillars Drag drop the `Pillar2` prefab to the theme editor and link it to the `WallSeparator` marker node

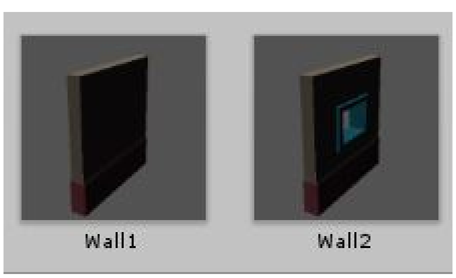


We’ll need to make this pillar a bit bigger. Select the node you just dropped and modify the Scale parameter under Offset category



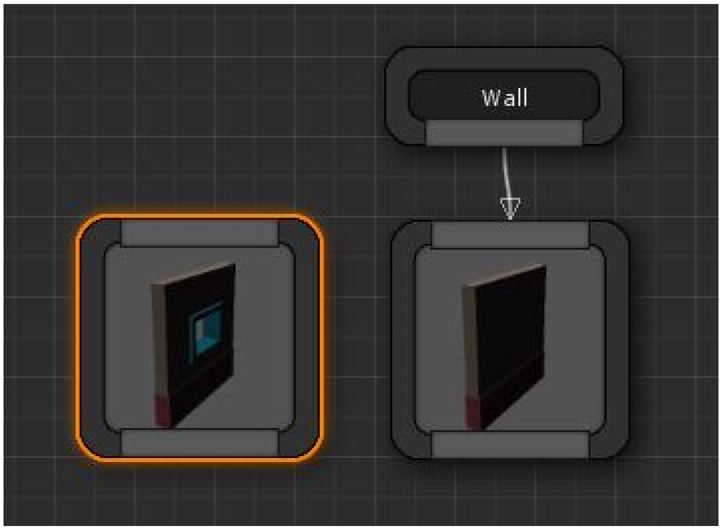


Add Windows We have two wall meshes in the samples folder

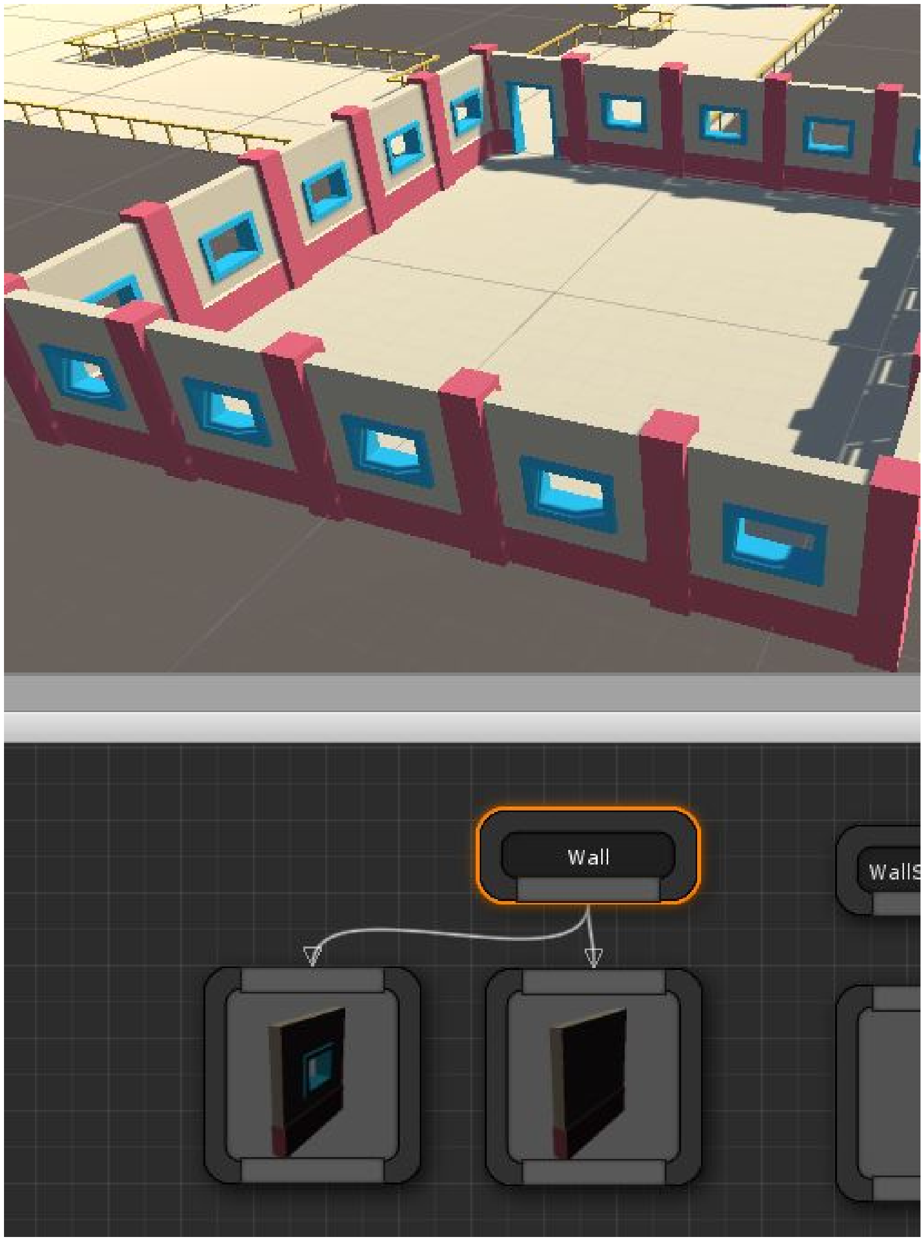


The other one (`Wall2`) has a window. Lets configure the theme to sometimes use this second mesh so we have windows

Drag drop `Wall2` prefab on to the theme editor and place it **before** (left of) the existing wall prefab node



When you connect this to the `Wall` marker node, you'll notice it has picked up the window node for all the walls



This is because Dungeon Architect starts executing the nodes from left to right. When the condition was satisfied to pick the first node, it stopped execution and never came to the second node.

There are multiple ways you can control this condition, the simplest being adjusting the probablity of selection.

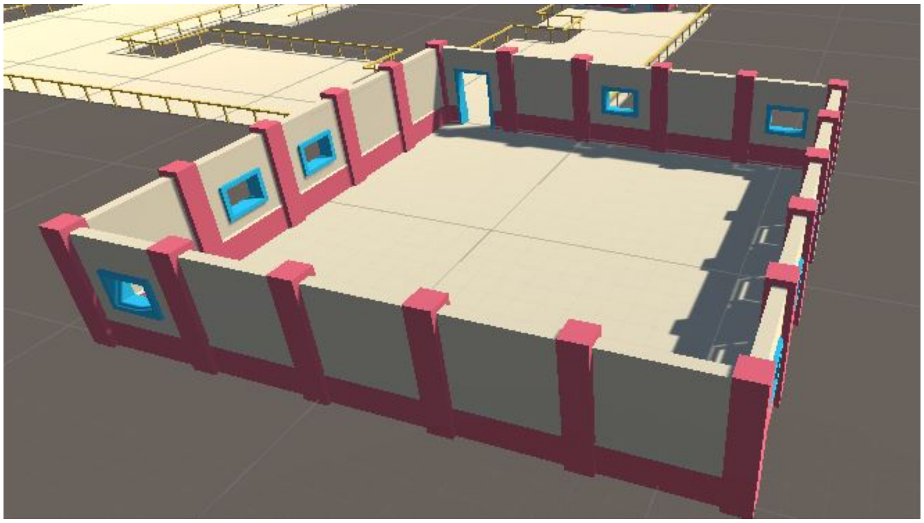
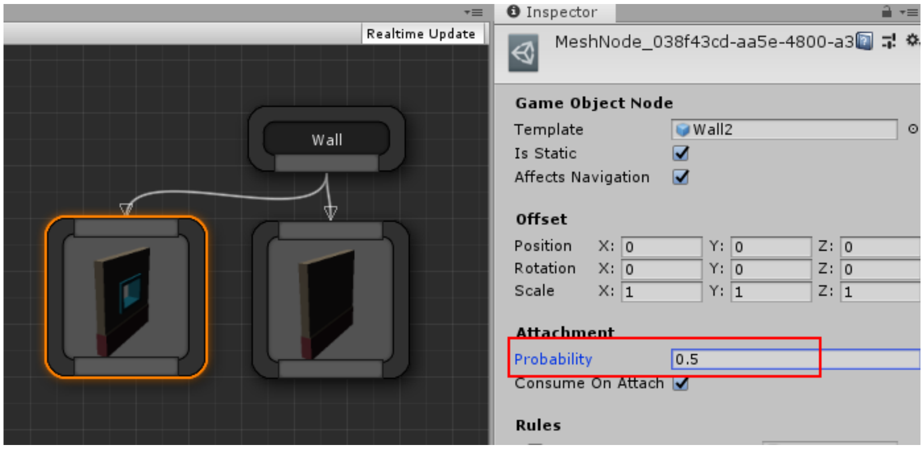
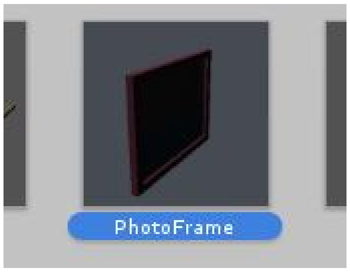


Figure 4: Half the walls have windows

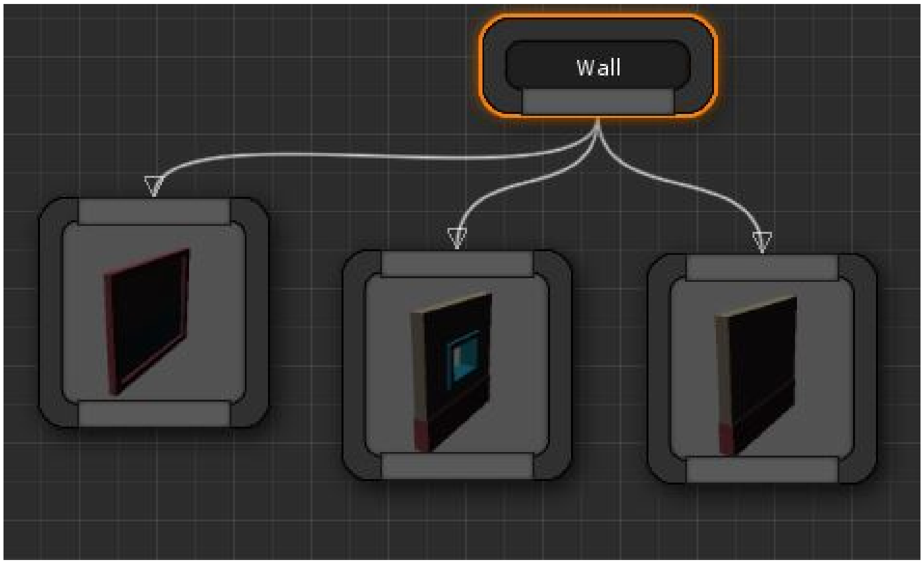
Select the node you just dropped and change the probability to `0.5` (this would mean it gets selected 50% of the time). The other 50% of the time, it would not be selected and the execution would then move to the next node, and hence selecting the non-windowed wall node



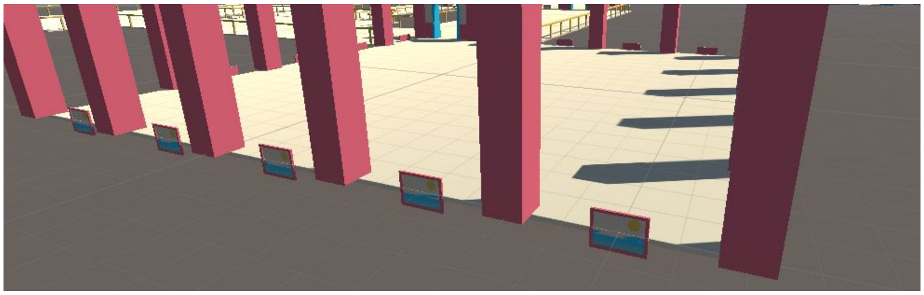
Add Wall Decorations There's a photo frame prefab we'd like to attach to every wall



Drag drop this prefab to the theme editor **before** the two existing nodes and link it to the `Wall` marker node

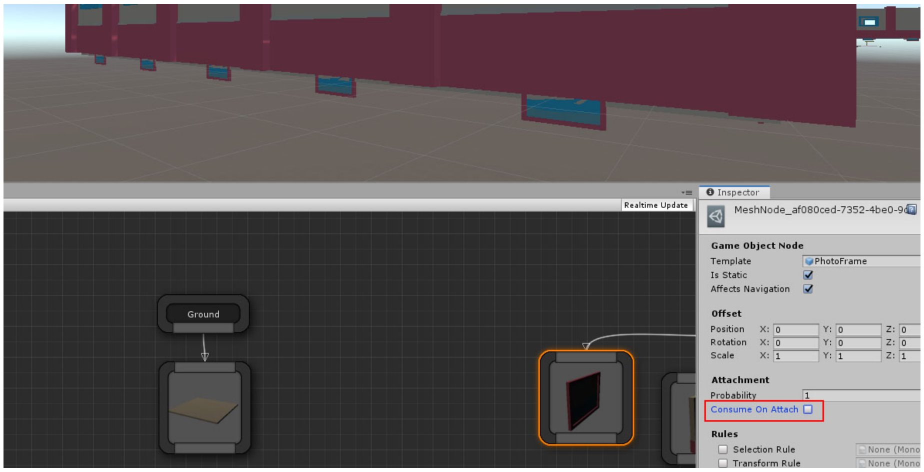


This will cause all the walls to disappear and be replaced with this photo frame



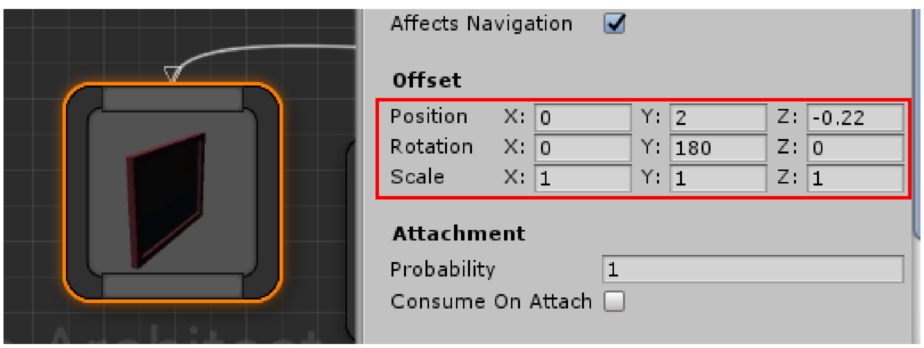
This is because once the photo frame was selected, the execution stopped and the the wall nodes further down the line were not executed.

Select the photo frame and uncheck the flag “Consume on Attach”. This will cause the execution to continue further even though this node was selected by the theming engine

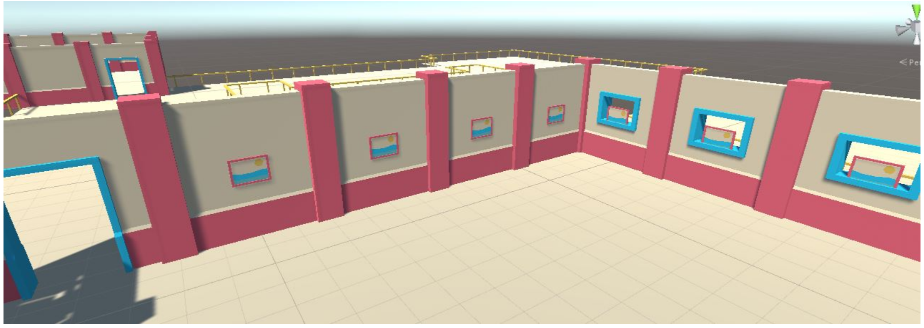


Lets adjust the offset of the photo frame (position and rotation) to make it properly align with the inner walls

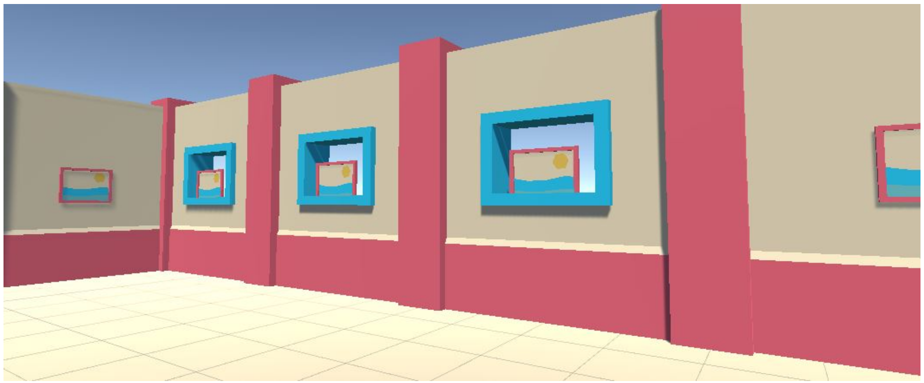
Select the photo frame node and change the position to `(0, 2, -0.22)` and rotation to `(0, 180, 0)`



The photo frame is aligned with the walls

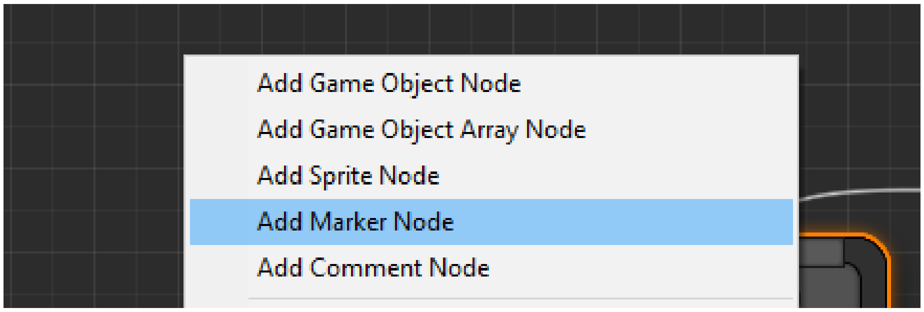


Marker Emitters We have an issue with the photo frames. They also spawn near windows

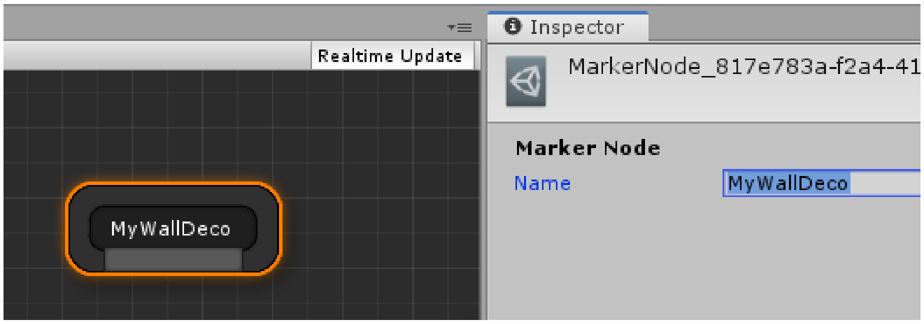


`Marker Emitters` allow you to emit marker names from any of your dropped prefab nodes. This means, we can define a new marker node (e.g. `MyWallDeco`) and then emit that marker from the wall node that doesn't have a window (`Wall1` prefab). All our wall decorations can now go under this `MyWallDeco` marker and it will show up only near solid walls

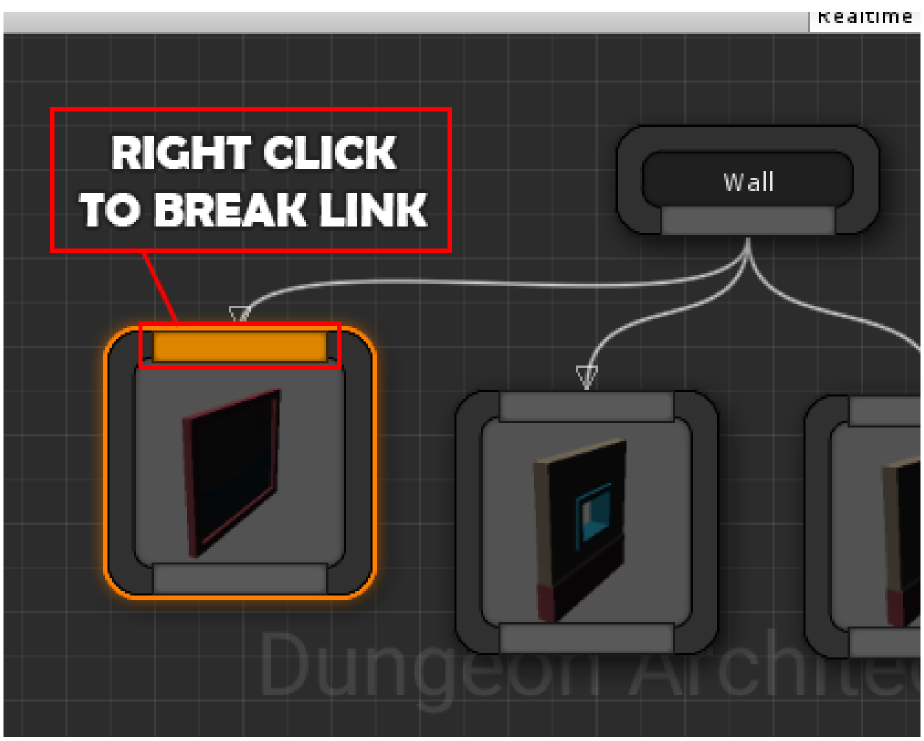
Right click on an empty area in the theme editor and select `Add Marker Node`



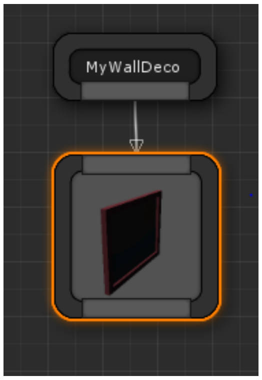
Select the node and change its name to `MyWallDeco`



Break the link to the photo frame

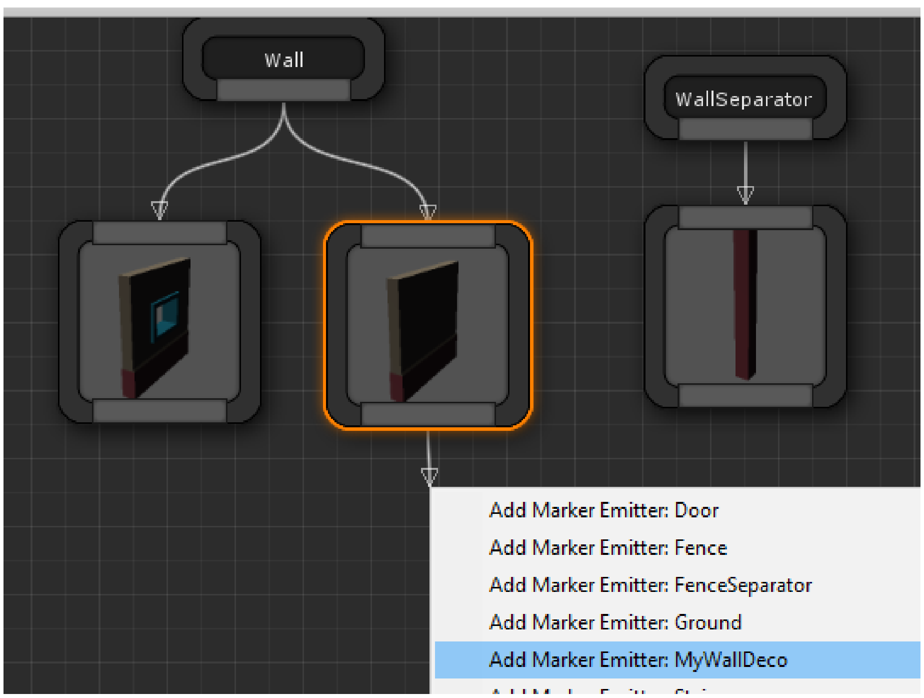


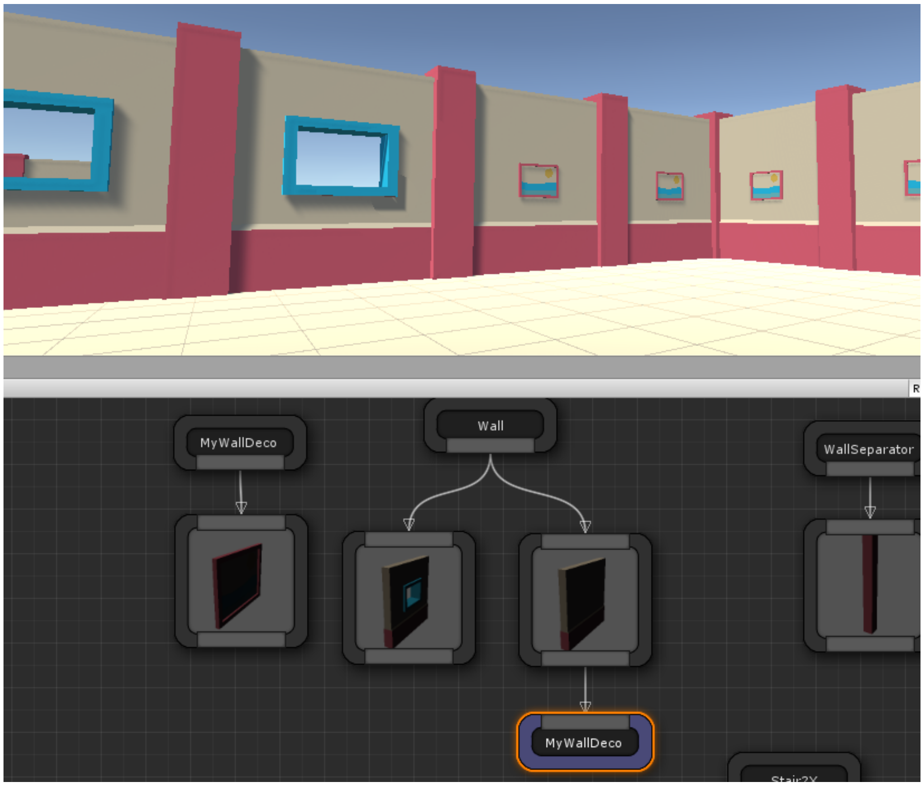
Connect this under `MyWallDeco` marker node. All the future wall decorations can also go under this marker



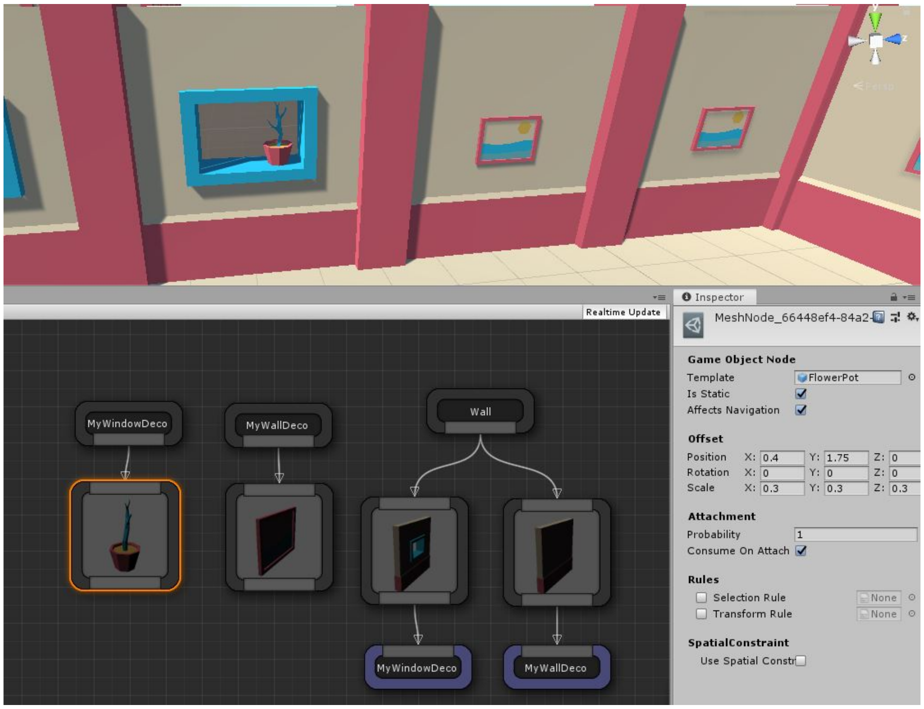
Now emit this marker from the wall node that doesn't contain a window

Drag a link out of the bottom of the solid wall prefab node and release the mouse in an empty area





You can follow the same method to create another type of decoration (e.g. MyWindowDeco) and emit it from under the windowed wall node. In this example, I've added a flower pot in the windows



Recap In this section we learnt the following:

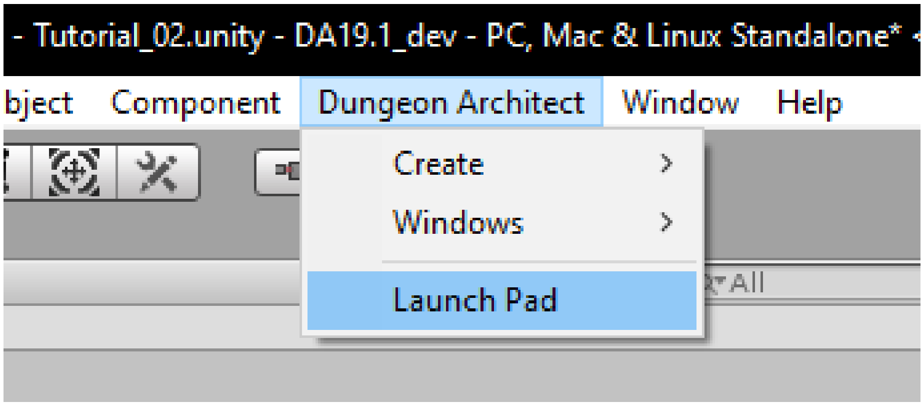
- *Probablity* - Controls the percentage chance of a node being selected. A value of 1 means 100% selection chance. A value of 0.25 means 25% selection chance
- *Execution Order* - The theme engine executes all the nodes under a marker node from left to right. If it selects a certain node, it stops executing, unless the `Consume on Attach` flag is unchecked
- *Marker Emitters* - You can create complex hierarchies with your own marker nodes, giving you more freedom to decorate your dungeons

Launch Pad Window

Use the Launch Pad window to setup new dungeon scenes, browse the samples, clone from templates and much more

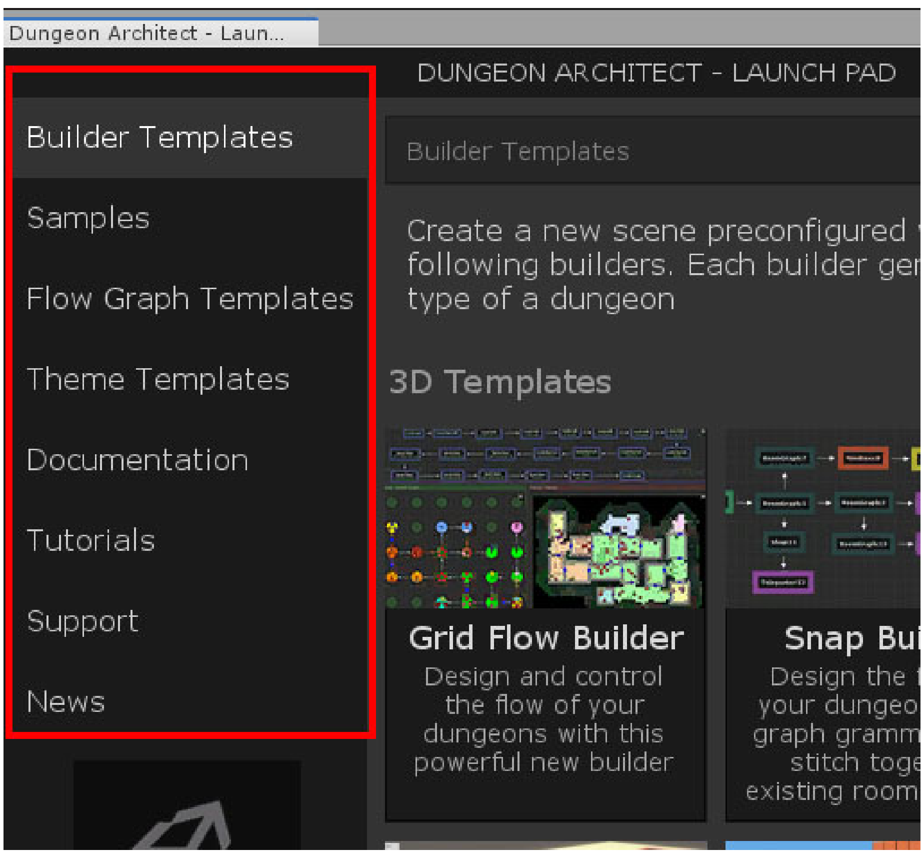
Open Launch Pad

From the Main menu, open the Launch Pad window `Dungeon Architect > Launch Pad`



Navigation

Select the various sections from the left.



Use the navigation bar on the top to go back to a previous page.

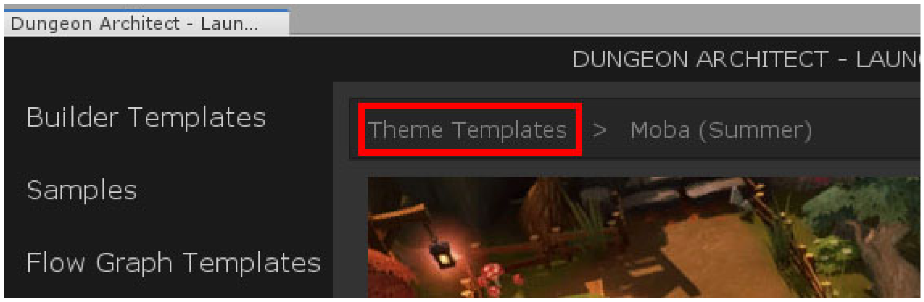


Figure 5: Navigation bar

This is useful for retaining the scroll positions of the previous page (especially for larger pages like the Samples section)

Builder Templates

Dungeon Architect supports many different types of dungeon layout methods and is designed in a way that new layout methods can be easily added in the future

These layout methods are called `Dungeon Builders` or **Builders** in short

This section lets you create a new scene preconfigured with one of the builder templates. Click on any of the builders. In the next screen, click the `Clone Scene` button

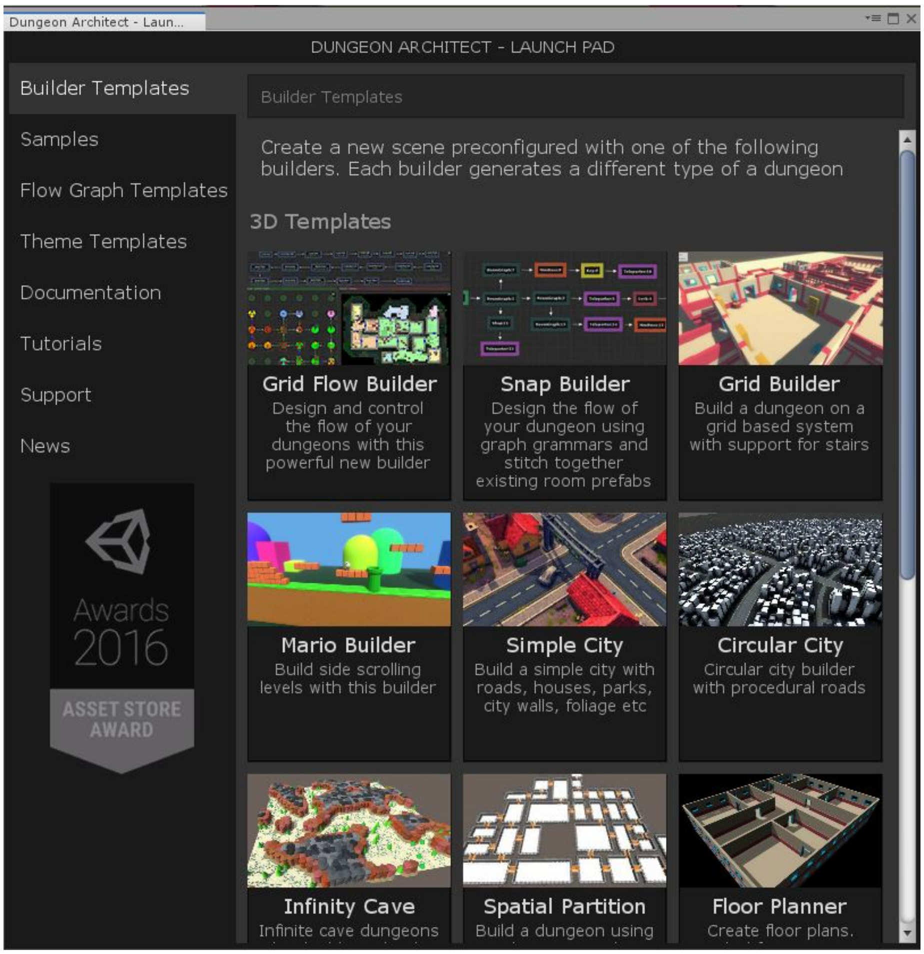
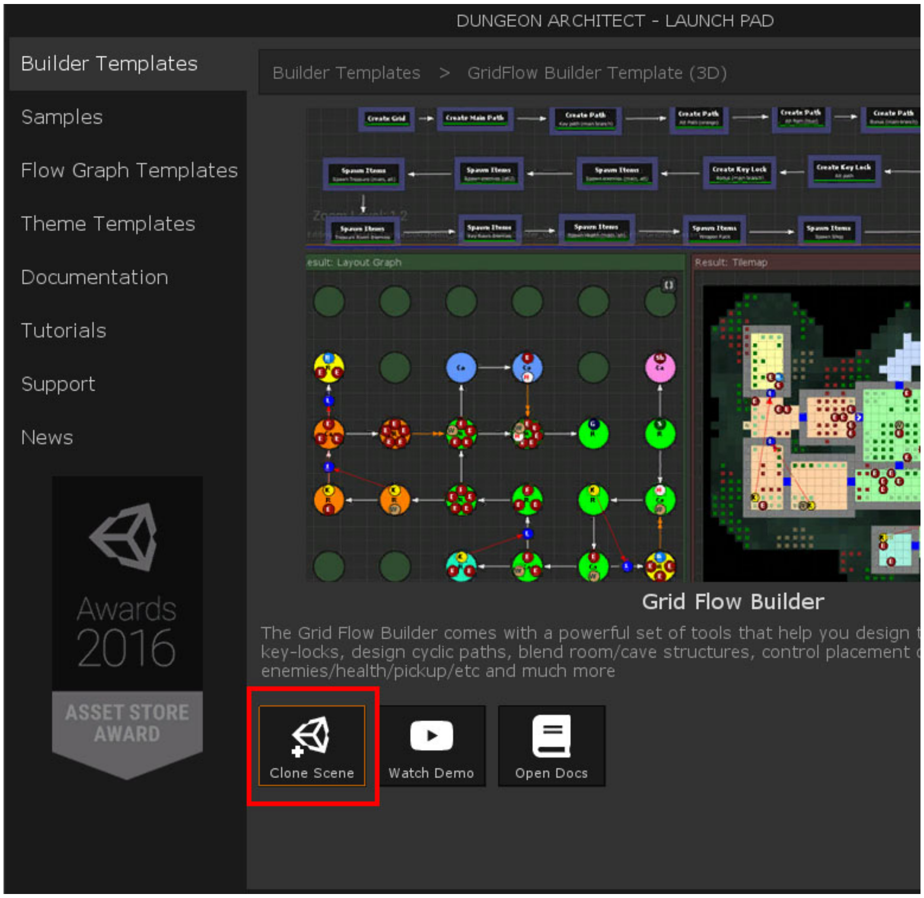


Figure 6: Dungeon Architect - List of Builders

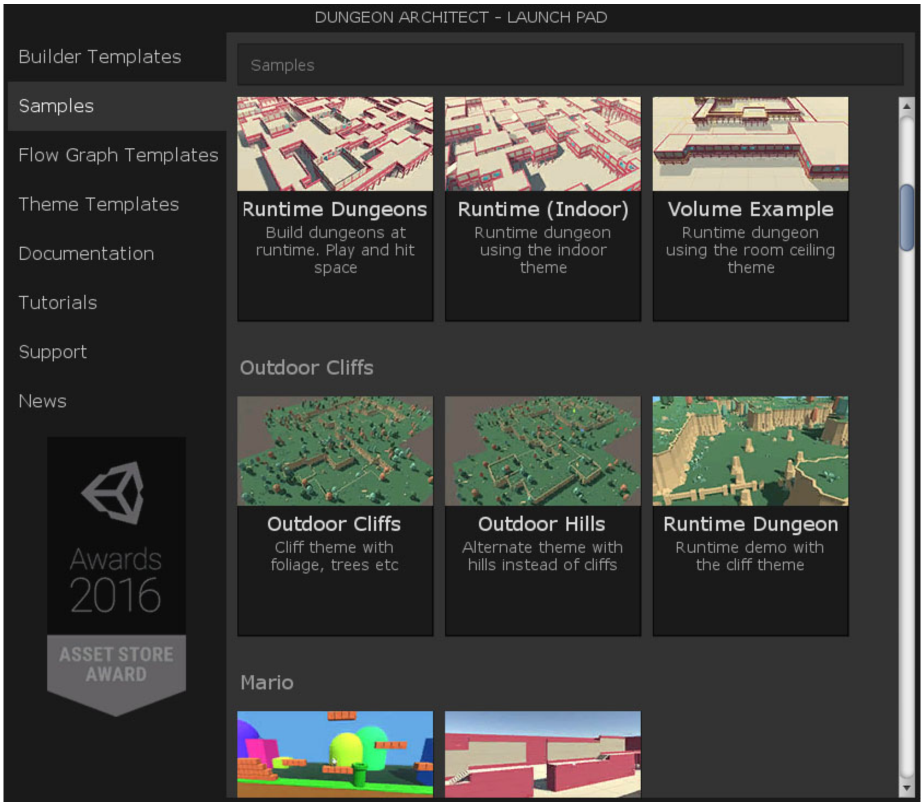


This will create a new scene based on the template, fully configured with the appropriate dungeon. It will also clone a starting theme file (and any other flow graph assets) and set everything up.

Choose a folder to saved your scene file. Once saved, the launcher would do the following:

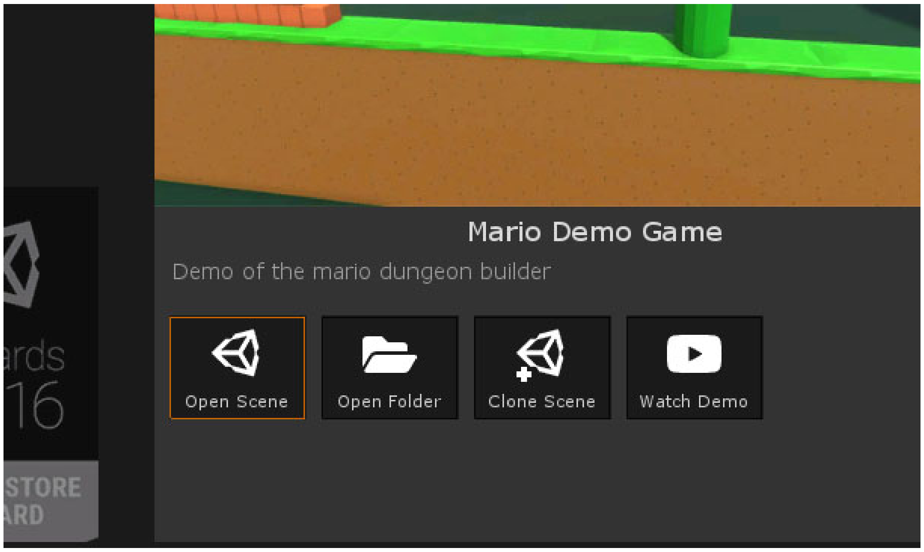
- Open the new scene
- Open any theme editor windows associated with the referenced assets (Theme Editor, Flow Graph Editors etc)
- Select the dungeon game object (so you see the properties in the inspector by default)

Samples

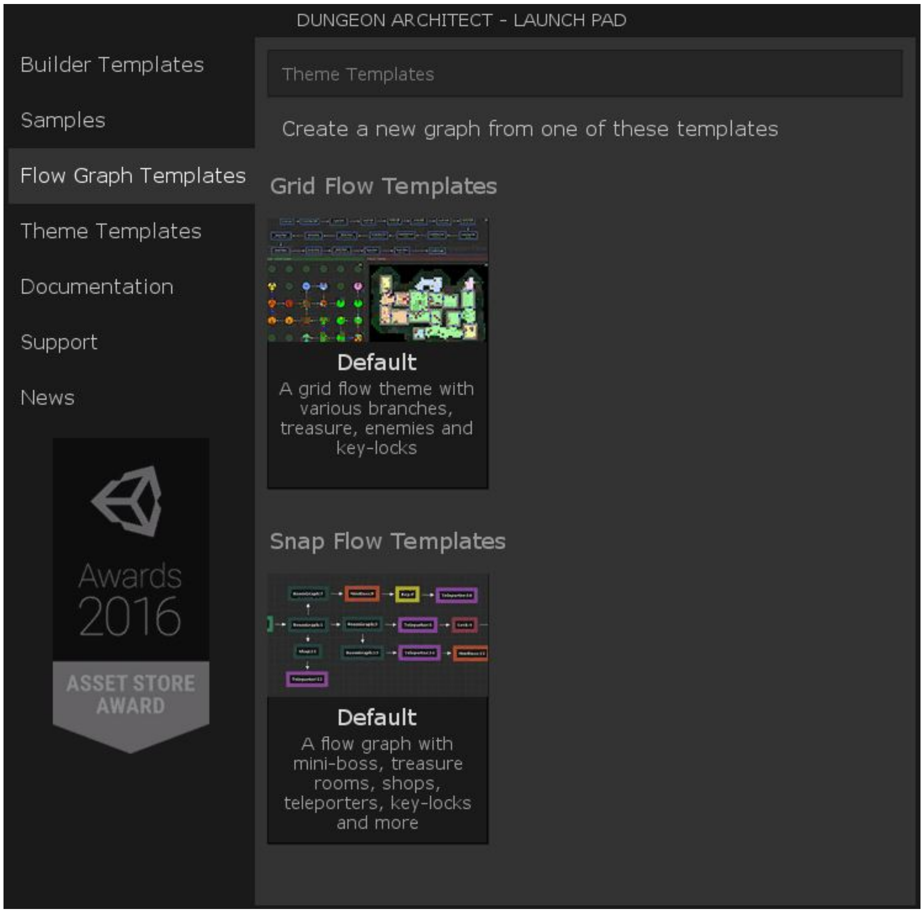


There are tons of samples to explore. Select a sample you like and perform one of the following actions

- **Open Scene**: Opens the sample scene (usually under DungeonArchitect_Samples folder)
- **Open Folder**: Opens the folder containing the scene
- **Clone Scene**: Clone a scene and also clone over the referenced assets (themes, flow graphs etc) so you can modify them without affecting the sample scene
- **Watch Demo**: Watch a video, if it exists


Mario Demo Game
Demo of the mario dungeon builder
Open Scene
Open Folder
Clone Scene
Watch Demo

Flow Graph Templates



Flow graphs allow you to control the flow of your dungeon (more on this in the later tutorial sections). This section contains a list of flow graph templates you can use as a starting point for your project

Theme Templates

Clone one of the many themes and use it in your project or as a starting point for a new theme

Select a theme and clone it

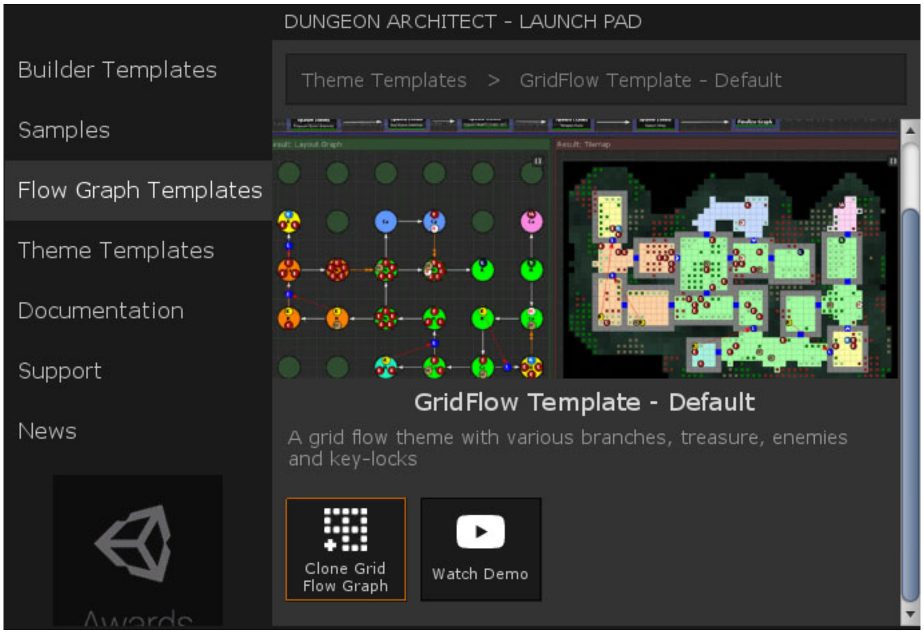


Figure 7: Clone Grid Flow Graphs

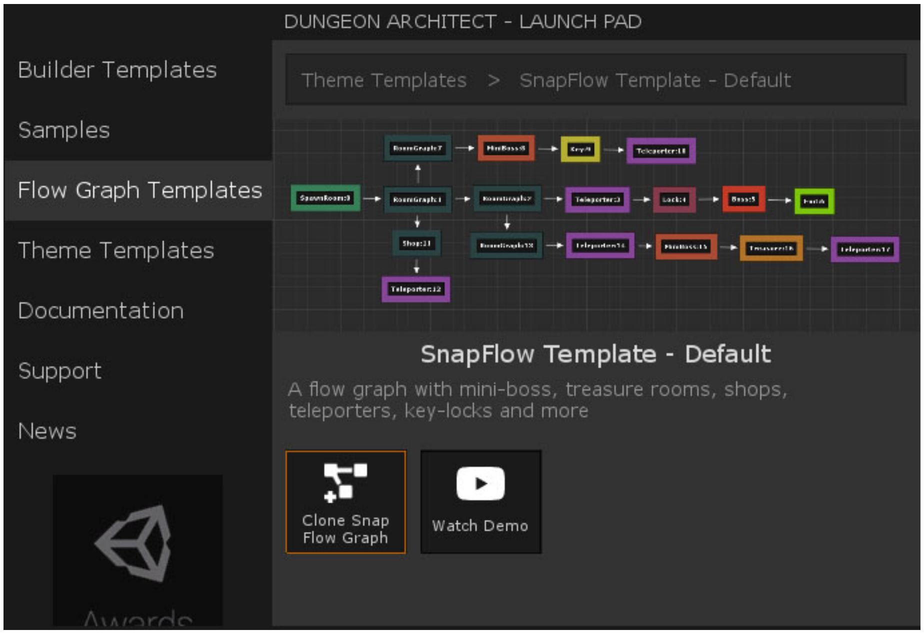


Figure 8: Clone Snap Flow Graphs

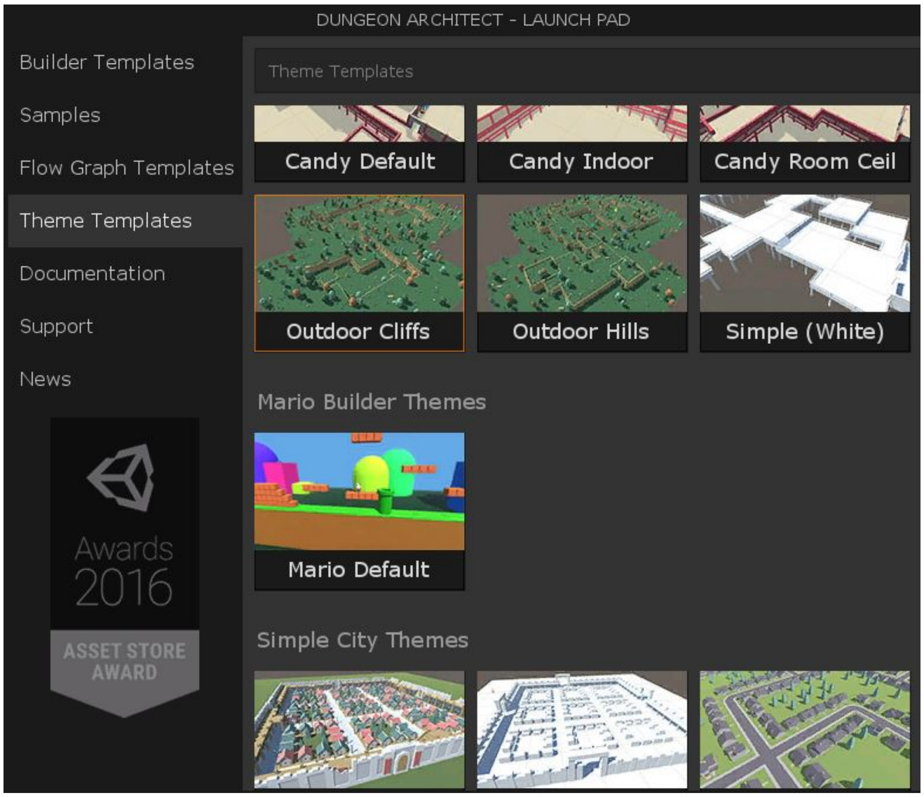
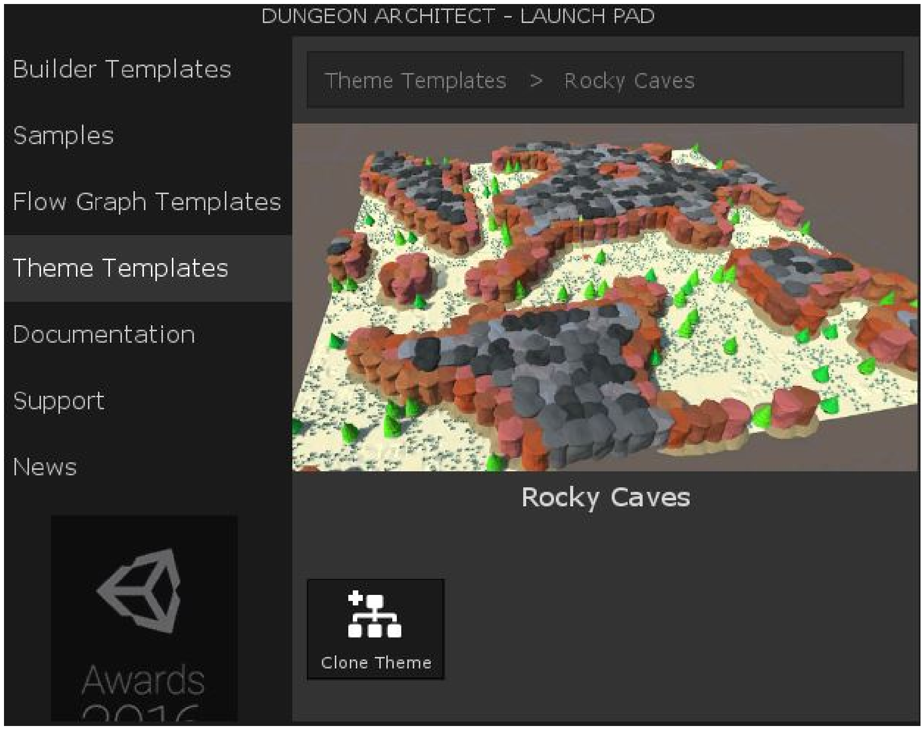
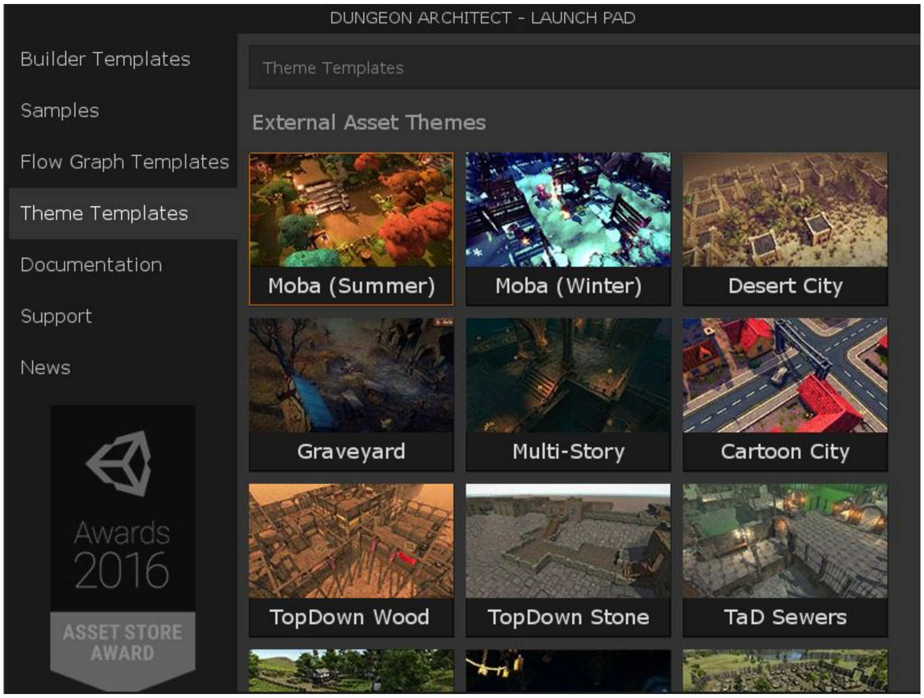
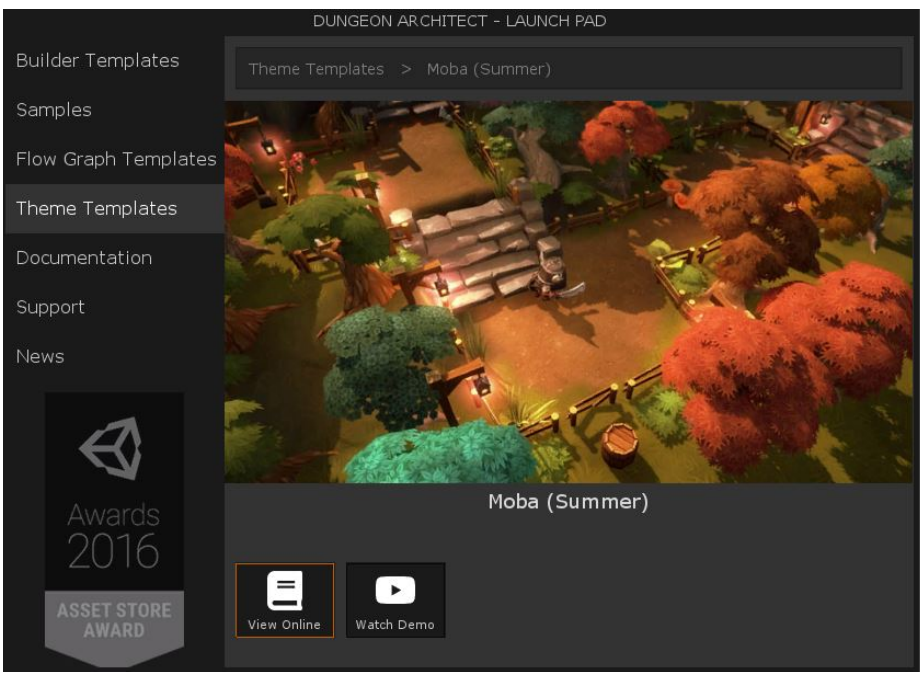


Figure 9: Theme template browser



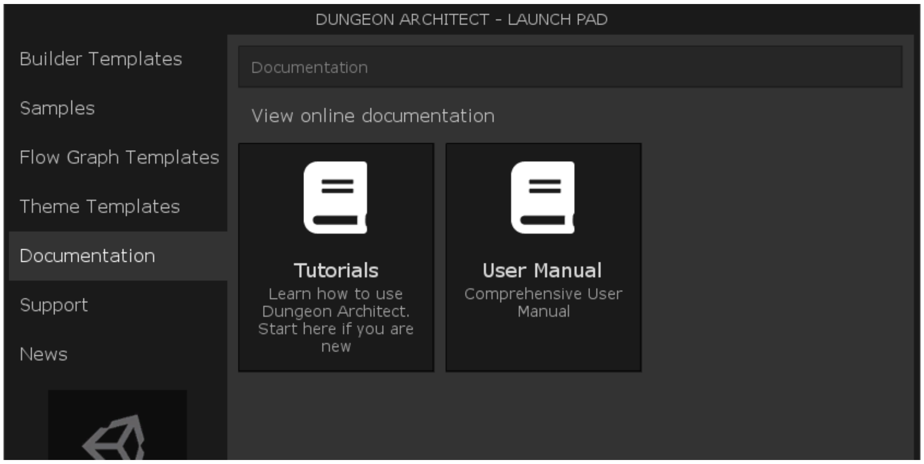
There's also a category on `External Themes`, the ones that use external paid art assets from the Asset Store. You can use these themes if you also own the art asset.


DUNGEON ARCHITECT - LAUNCH PAD
Builder Templates
Samples
Flow Graph Templates
Theme Templates
Documentation
Support
News
Awards 2016
ASSET STORE AWARD
Theme Templates
External Asset Themes
Moba (Summer)
Moba (Winter)
Desert City
Graveyard
Multi-Story
Cartoon City
TopDown Wood
TopDown Stone
TaD Sewers


DUNGEON ARCHITECT - LAUNCH PAD
Builder Templates
Samples
Flow Graph Templates
Theme Templates
Documentation
Support
News
Awards 2016
ASSET STORE AWARD
Theme Templates > Moba (Summer)
Moba (Summer)
View Online
Watch Demo

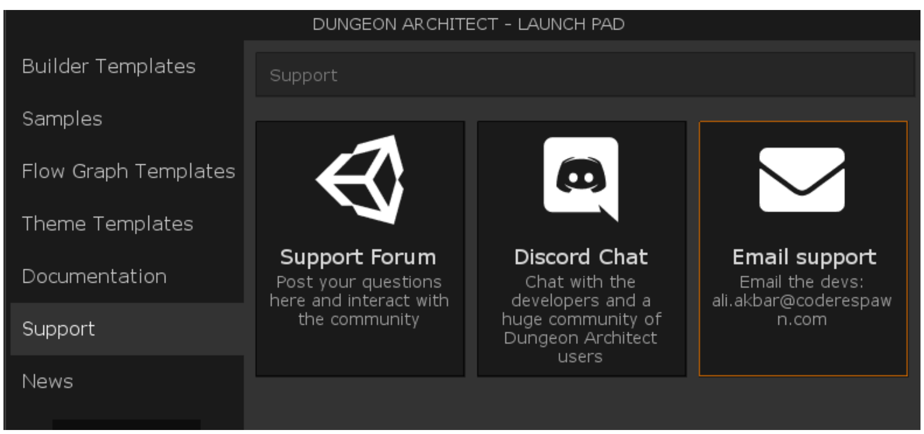
Documentation

Links to various online documentation, including this one



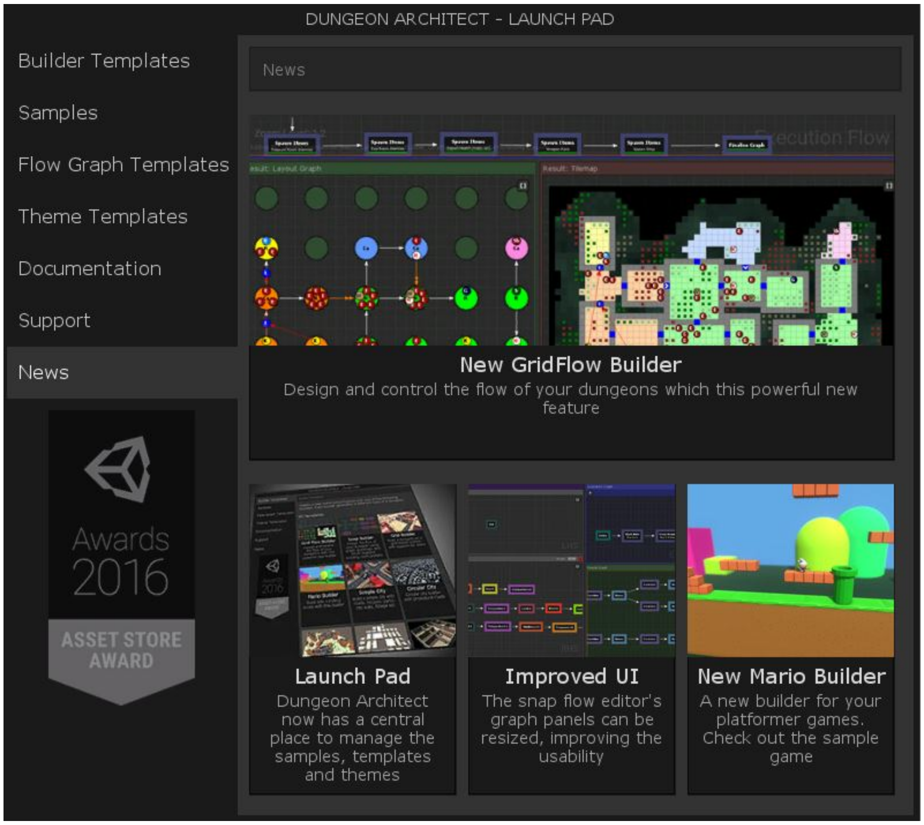
Support

Reach the developers through any one of these channels. Interact with the community and the devs in Discord chat and forums or reach directly through email



News

Dungeon Architect News! Find out whats new since the last update



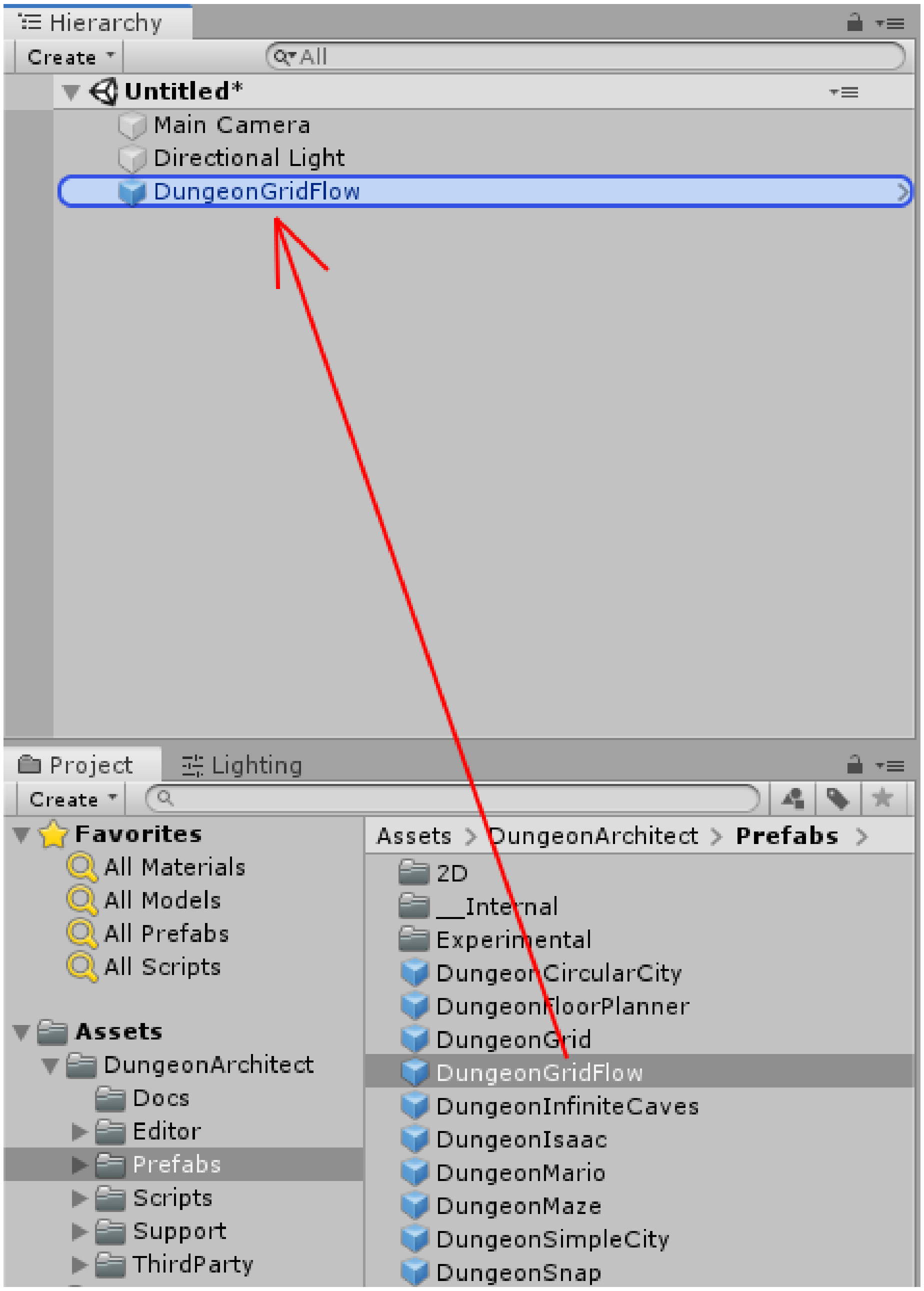
Grid Flow Builder

Create a Grid Flow Dungeon

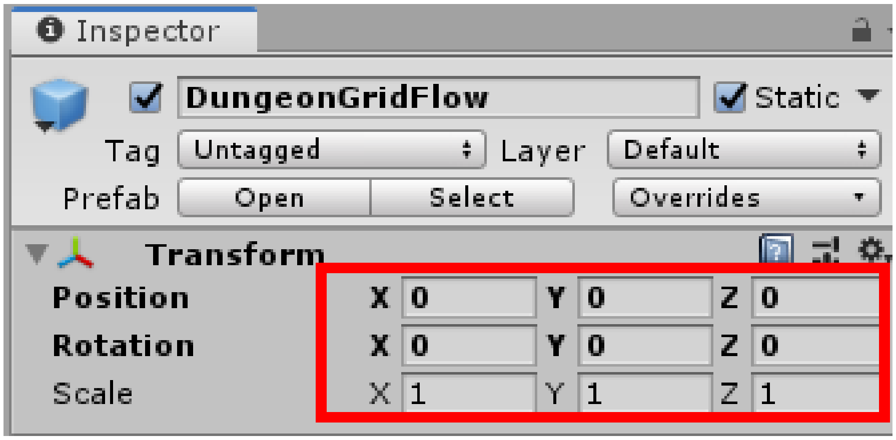
The Grid Flow Builder offers a rich set of tools to control the flow of your dungeons and item placement

Setup Dungeon Prefab

Create a new scene. Navigate to `Assets/DungeonArchitect/Prefabs` and drop in the DungeonGridFlow prefab on to the scene

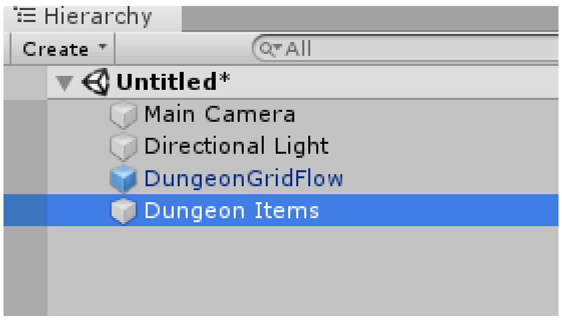


Select the DungeonGridFlow game object you just dropped and reset the transform

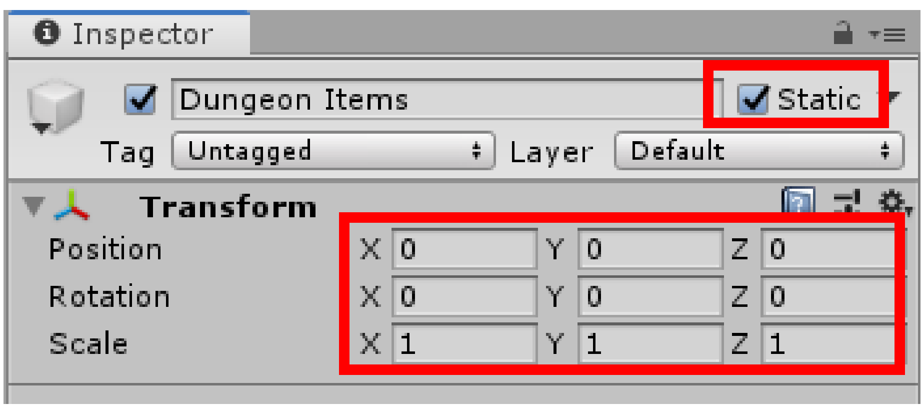


Setup Parent Object

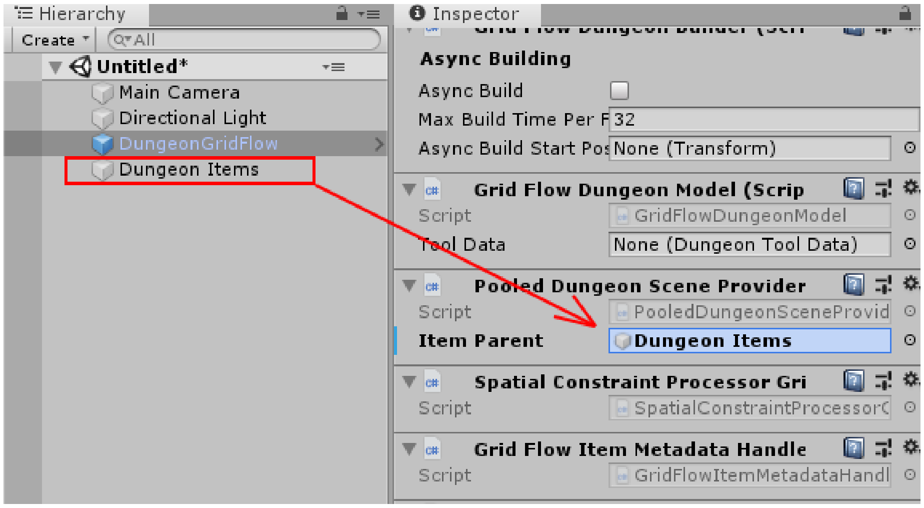
Create a new Parent object where all the spawned dungeon items will go in.



Reset the parent object's transform and set it to static



Assign the parent object

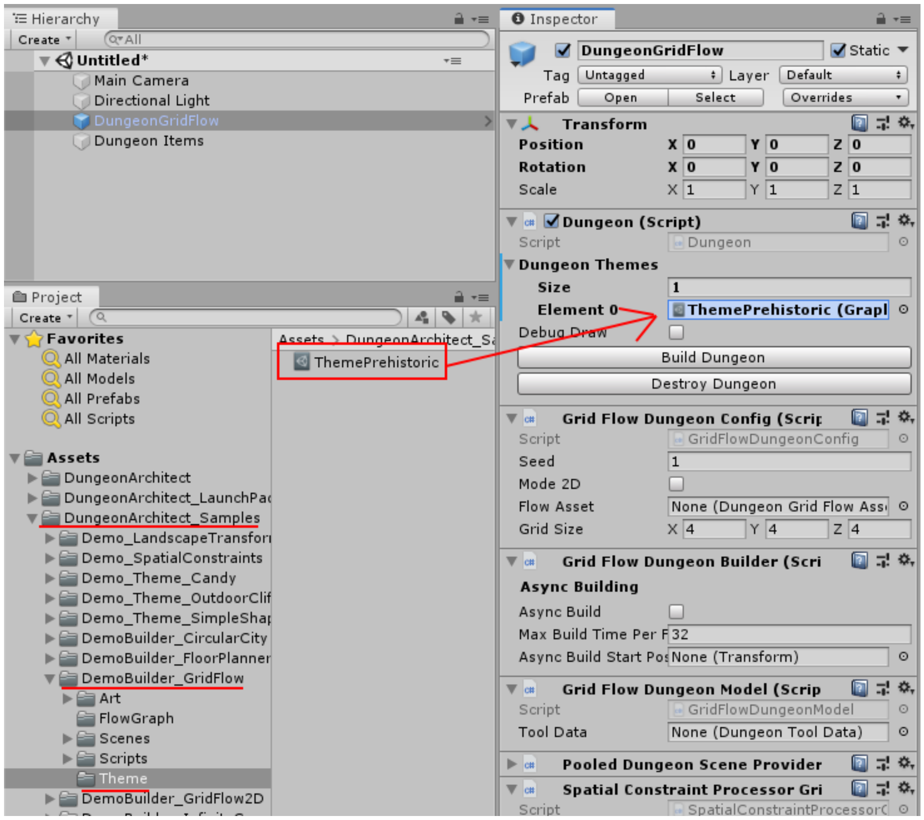


This makes sure all the spawned dungeon objects are placed under this parent object

Setup Theme

There's a theme available in the samples folder which we'll use for this tutorial section

Navigate to `Assets\DungeonArchitect_Samples\DemoBuilder_GridFlow\Theme` and assign the theme `ThemePrehistoric` to the DungeonGridFlow game object



Setup GridFlow Graph

This builder requires another asset called the `Grid Flow Graph`. This is a graph that helps you control the flow of your dungeon. In this section, we'll use an existing graph from the samples folder

Navigate to `Assets\DungeonArchitect_Samples\DemoBuilder_GridFlow\FlowGraph` and assign the graph `DemoGridFlow` to the `DungeonGridFlow` game object's `Flow Asset` property

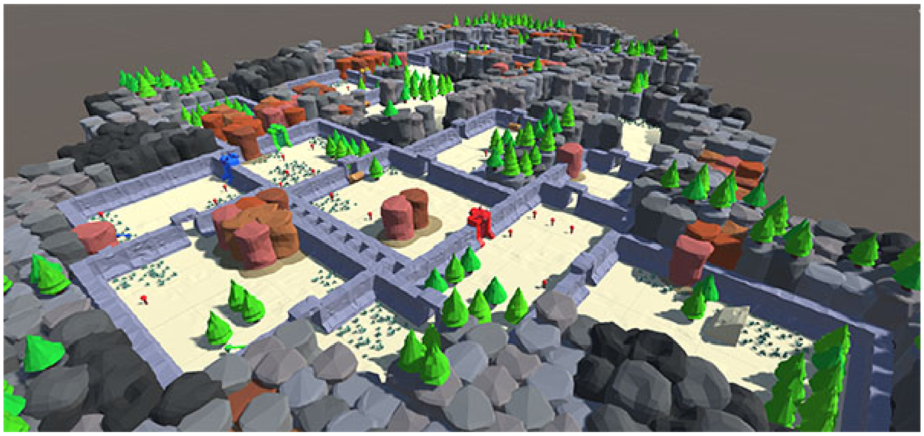
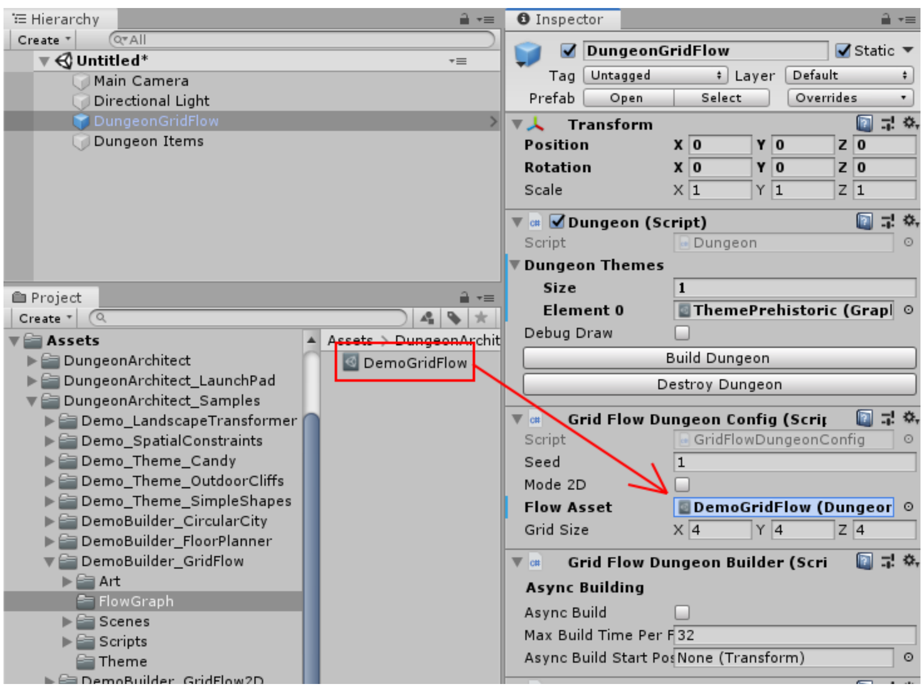


Figure 10: GridFlow dungeon built using the Prehistoric theme



Build Dungeon

Select the `DungeonGridFlow` game object and click `Build Dungeon` button from the inspector window

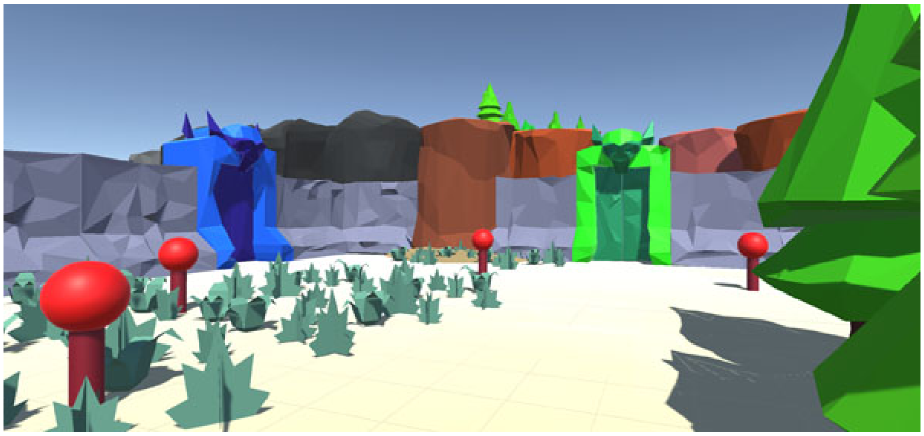


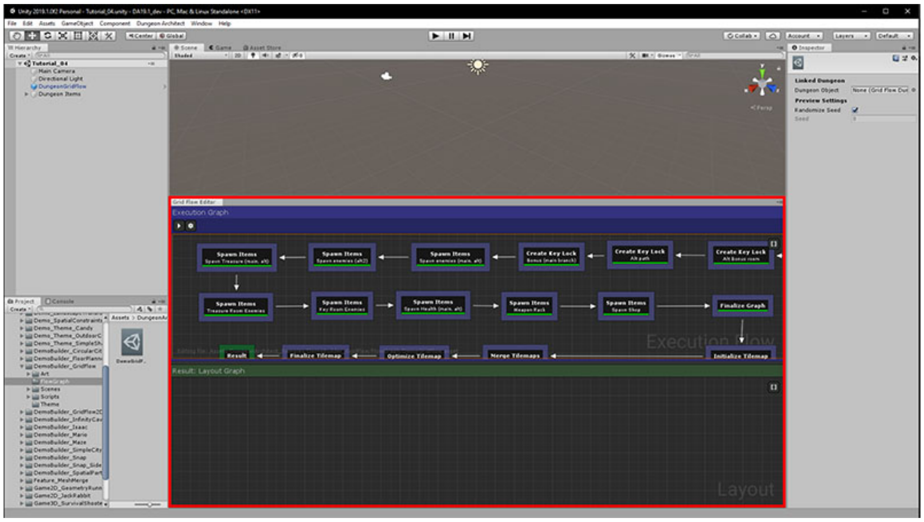
Figure 11: GridFlow dungeons support key-locks

Open Grid Flow Editor

Let’s open the GridFlow asset in the editor:

Navigate to `Assets\DungeonArchitect_Samples\DemoBuilder_GridFlow\FlowGraph` and double click on `DemoGridFlow`.

Dock the editor window so you see both the scene view and the grid flow editor



Click the `Build` button on the top left of the flow editor to build a new dungeon in the Grid Flow Editor

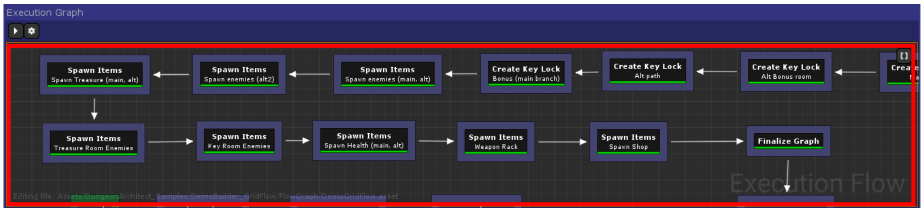
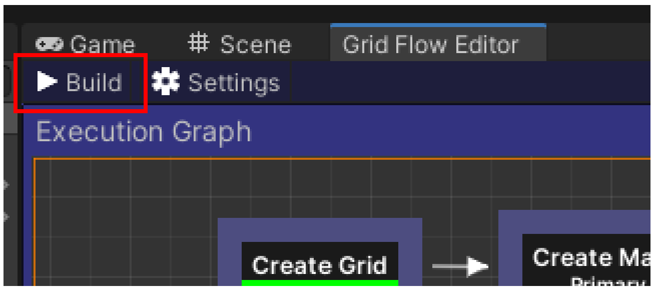
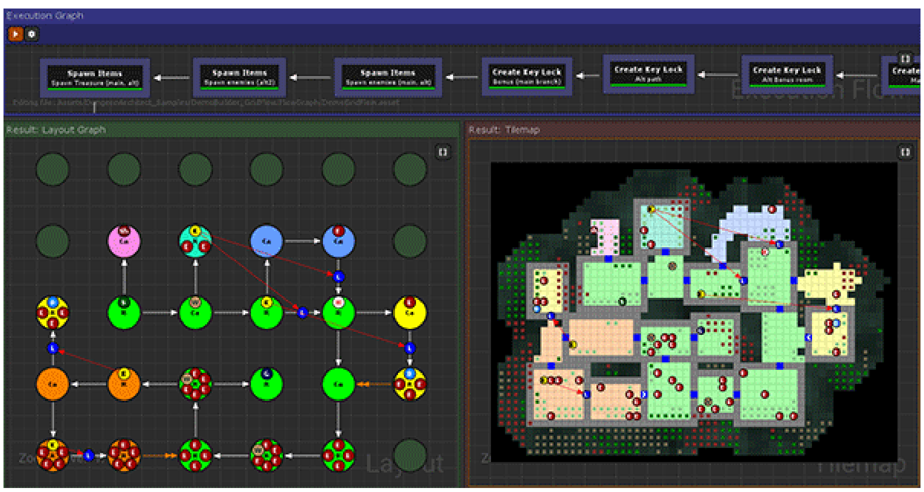


Figure 12: Click anywhere in the empty area





Link Editor with Dungeon

We are going to link up the dungeon we have on the scene (`DungeonGridFlow` game object) with the Grid Flow Editor so when we generate a new dungeon in the editor, it syncs up the dungeon on the scene

Click an empty space (grey area) in the Execution Graph

This will show up the execution graph properties in the Inspector window

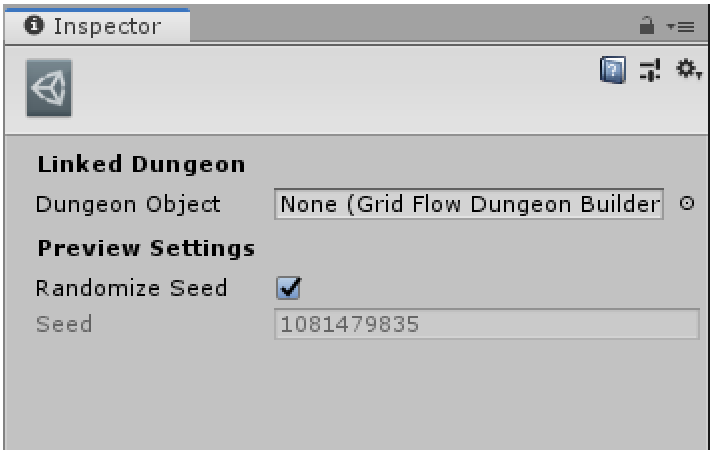
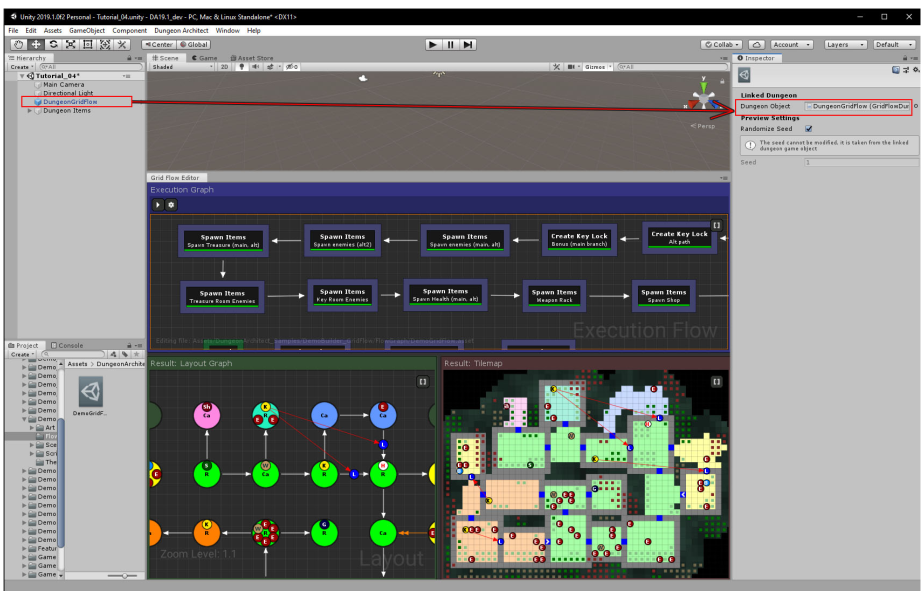
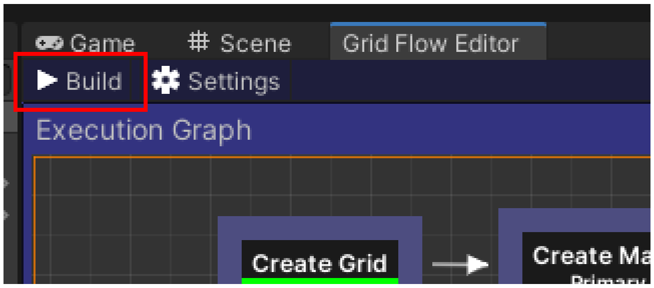


Figure 13: Execution Graph properties

Assign the dungeon game object by dragging the DungeonGridFlow over



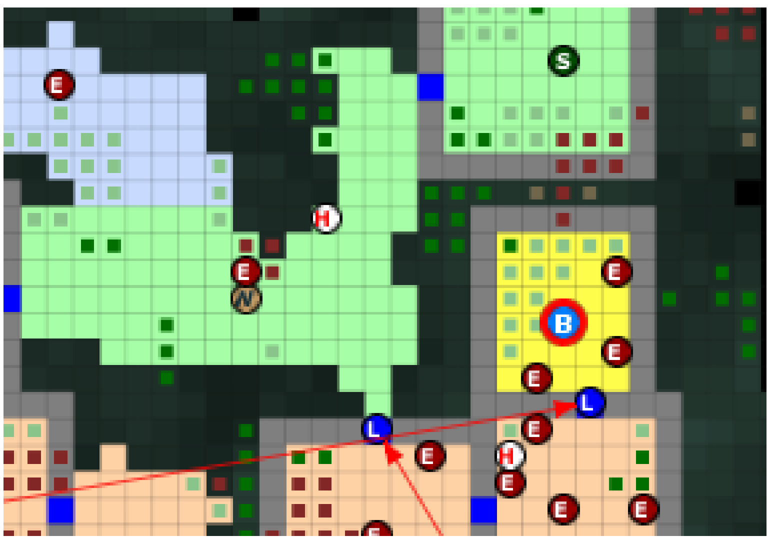
Now build the dungeon again by clicking the `Build` button in the Execution Graph Window



This will recreate the dungeon in the scene view. The dungeon in the editor window is now synchronized with the dungeon in the scene view

If you double click on any of the tiles in the Tilemap window, the scene view should focus on that tile/item

Double click on the Bonus Item (B) in the tilemap.



The scene view should zoom in on the treasure chest

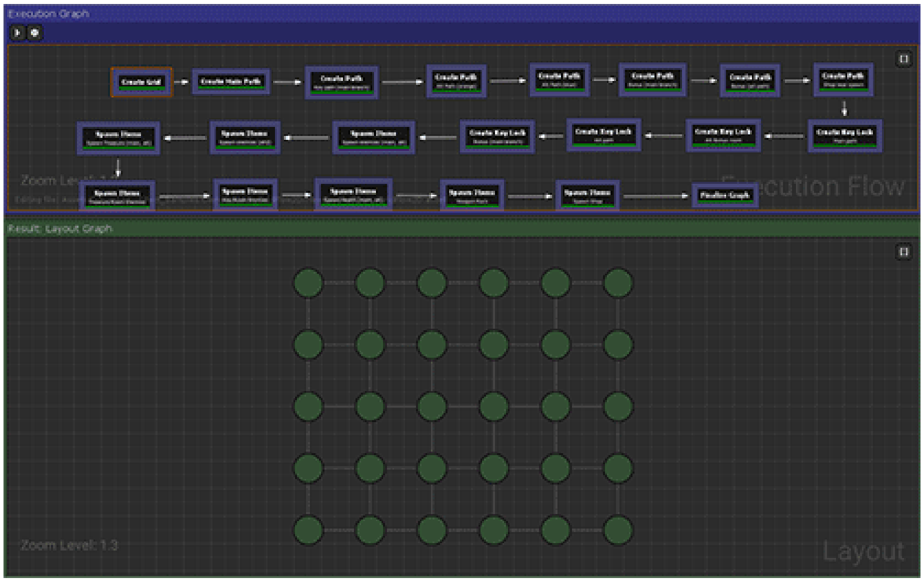


Figure 14: Select a node to preview the build process

Explore Grid Flow Graph

After you've built a dungeon in the editor (by hitting the build button on the top left), you can select each node and see how the dungeon layout was built, as shown in the lower preview panels

Design a Grid Flow Graph

In the previous section, we used an existing Grid Flow graph. In this section, we'll design one ourselves.

Setup

Destroy the existing dungeon and clear out the Flow Asset that we assigned earlier

Create a new Grid Flow asset from either the Create menu or the Main Menu

Rename to something appropriate and double click the grid flow asset to open it in the editor

We won't be needing the scene view for some time. Dock the editor so we have more working area

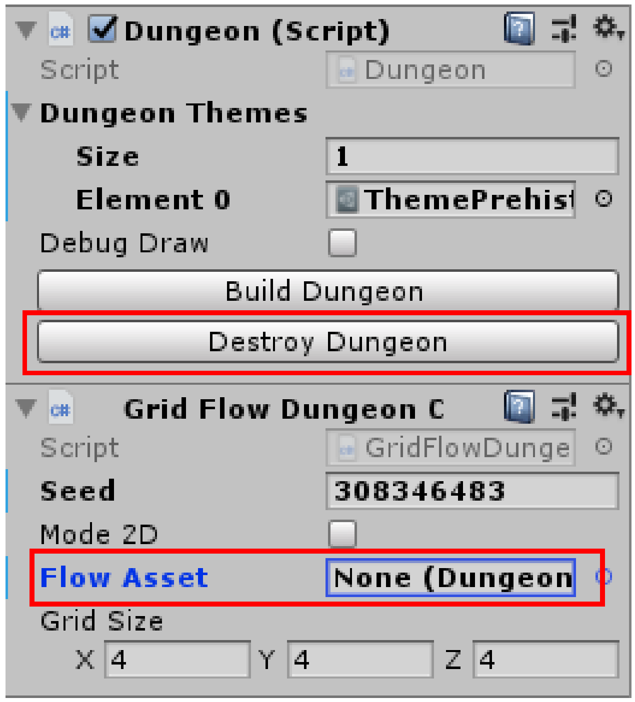


Figure 15: DungeonGridFlow game object properties

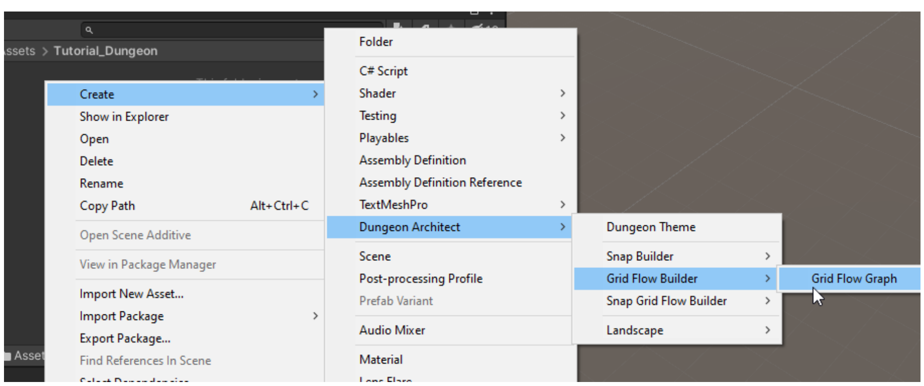


Figure 16: Create Menu

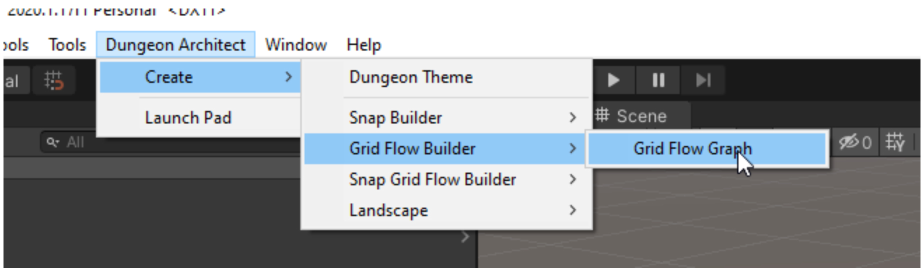
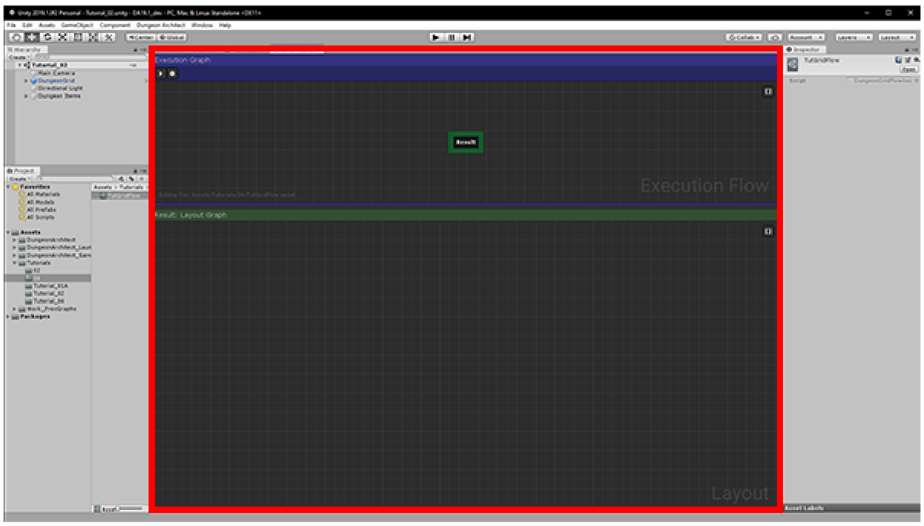


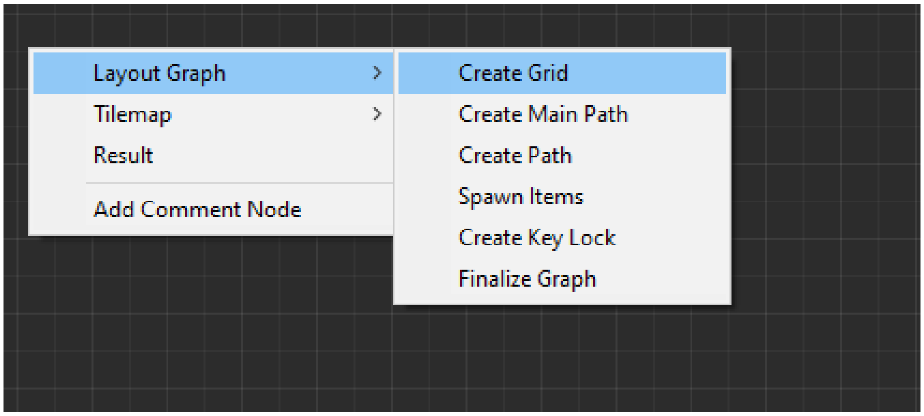
Figure 17: Main Menu

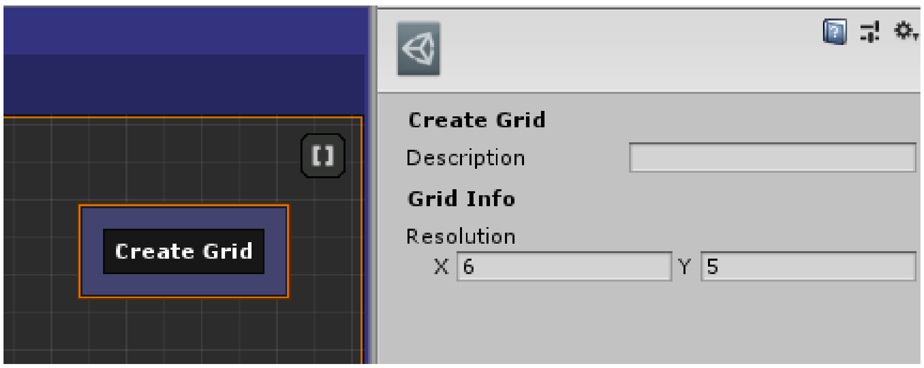


Notice that there is only one node in the Execution graph, the `Result` node. Our final output should be connected to this node

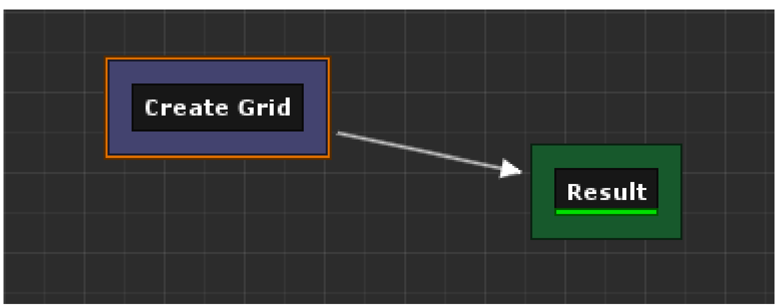
Create Grid

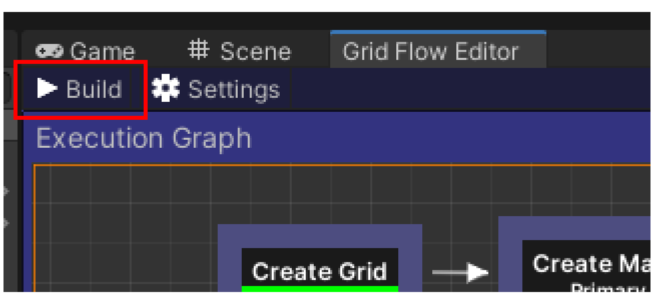
Right click on an empty area in the Execution Graph and from the context menu select `Layout Graph > Create Grid`



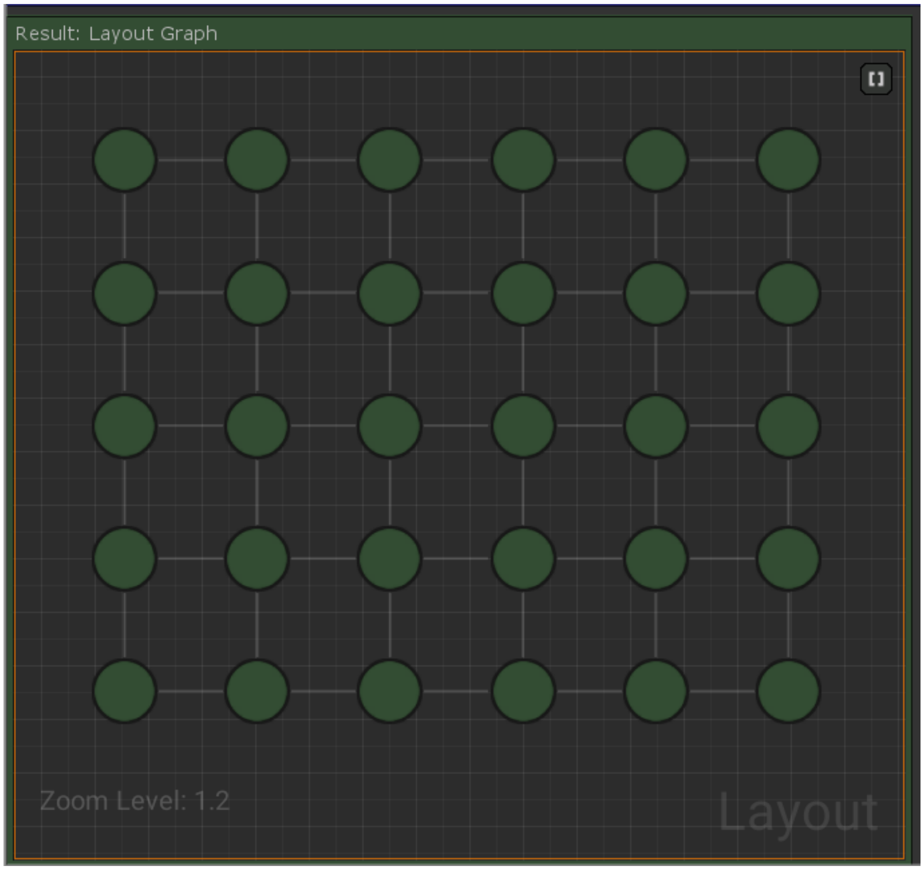


Connect this node to the `Result` node and hit build





This node creates an initial grid to work with

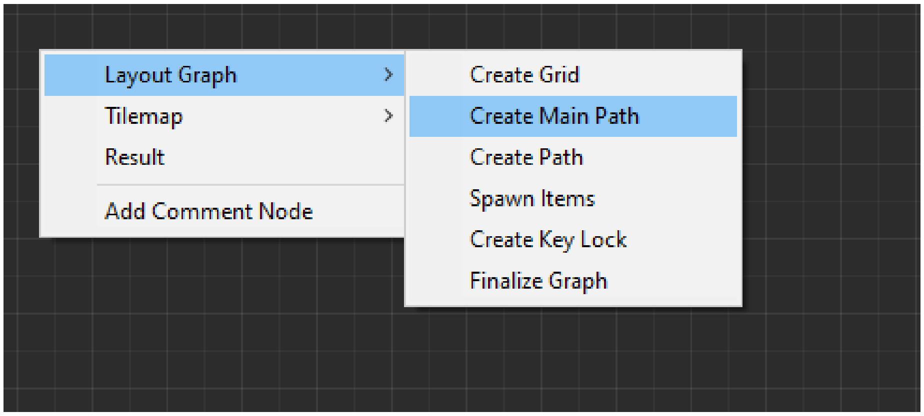


In this builder, you first design your level in an abstract layout graph like this and then move the final result to a tilemap

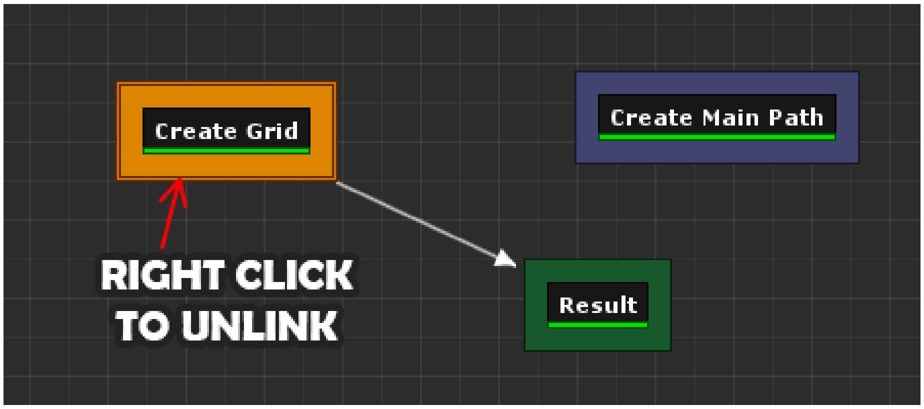
Create Main Path

We'll next create a main path within this grid. The main path has a spawn point and goal

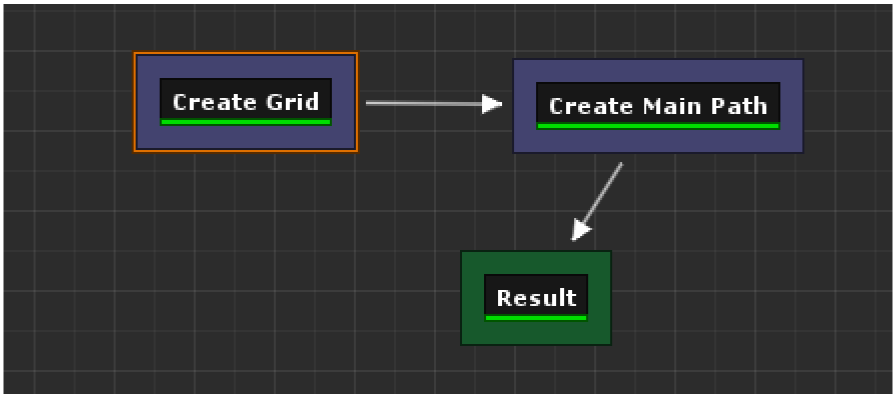
Create a new node `Layout Graph > Create Math Path`



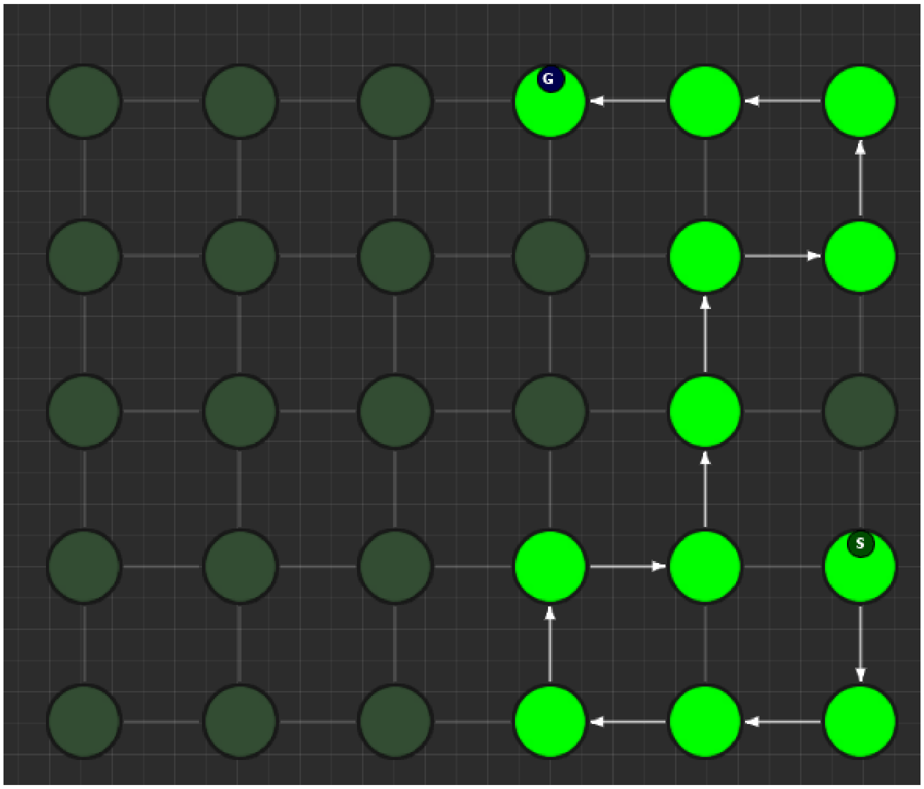
Unlink the `Create Grid` node from the `Result` node (do this by right clicking on the node's orange border)



Link the nodes up like below and click Build

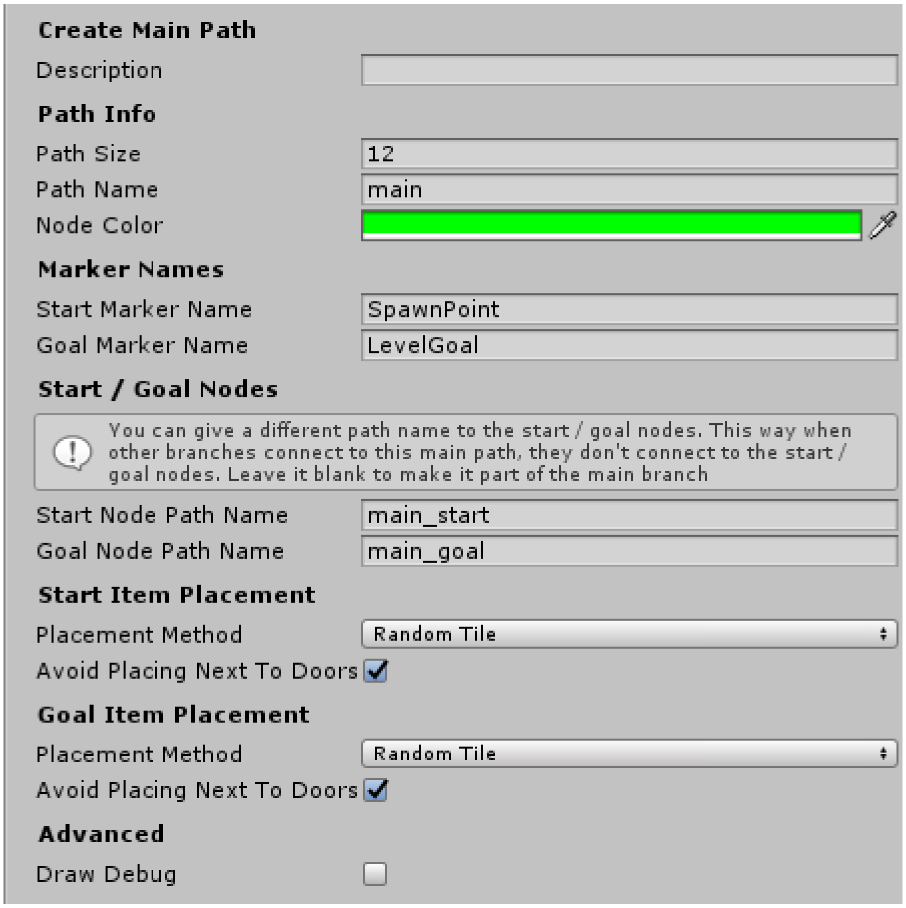


This node creates a main path in the grid. Keep clicking the build button for different result



If you do not see random results when you click build, make sure randomize is enabled. Enable this by clicking on an empty area in the Execution Graph to show the properties. In the inspector, select `Randomize Seed`

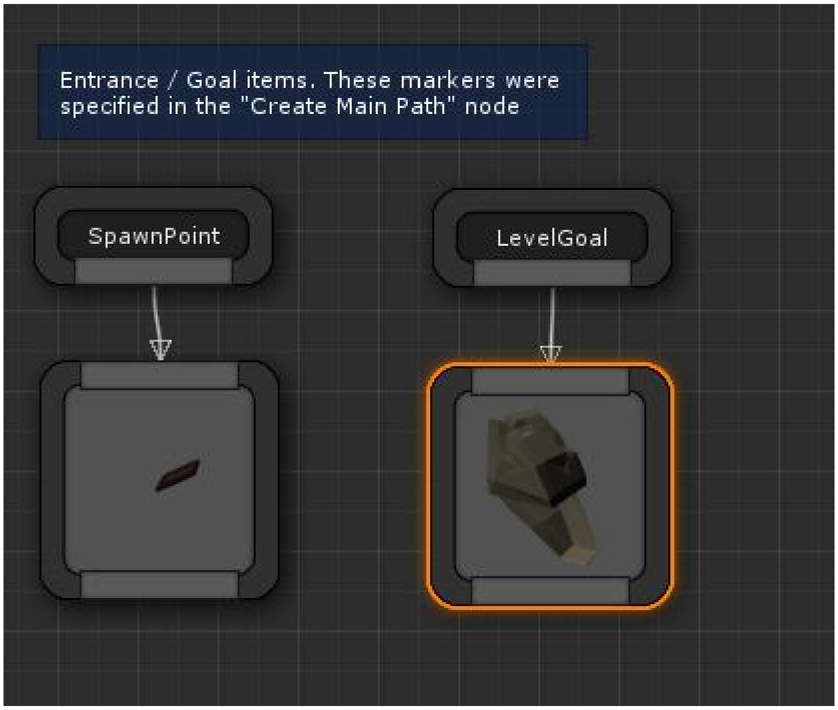
Select the `Create Main Path` node and inspect the properties



We'll leave everything to default for now

Notice the `Path Name` parameter is set to `main` This is the name of the path and we will be referencing this path in the future nodes with this name

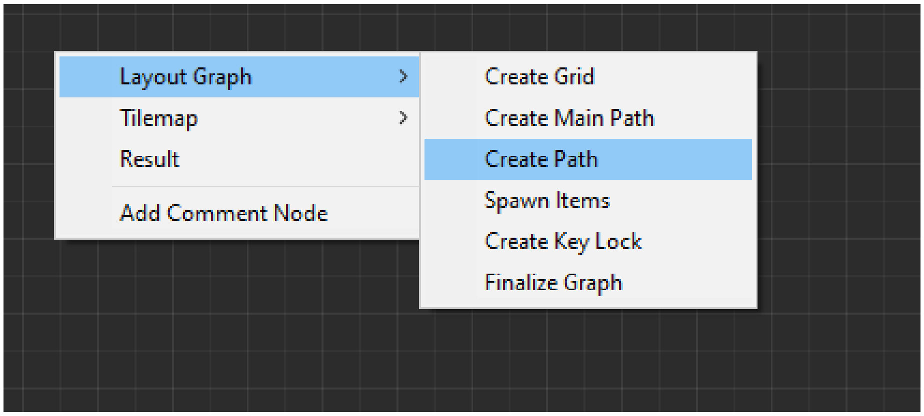
You can adjust the size of the path. `Start Marker Name` and `Goal Marker Name` lets you specify a name for the markers. You can then create these markers in the theme file and add any object you like. In the `Prehistoric` theme, there's a marker already created with these names and a player controller is placed under `SpawnPoint` marker and a level goal handler prefab is placed under `LevelGoal` marker



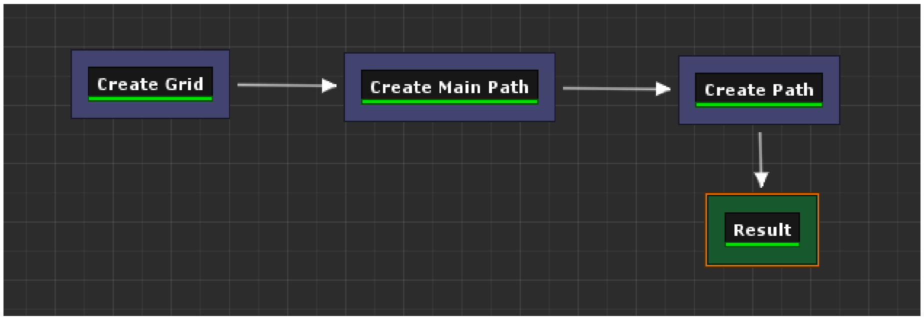
Create Alternate Path

We'll next create an alternate path pathing off the main path so the player has another way of reaching the goal

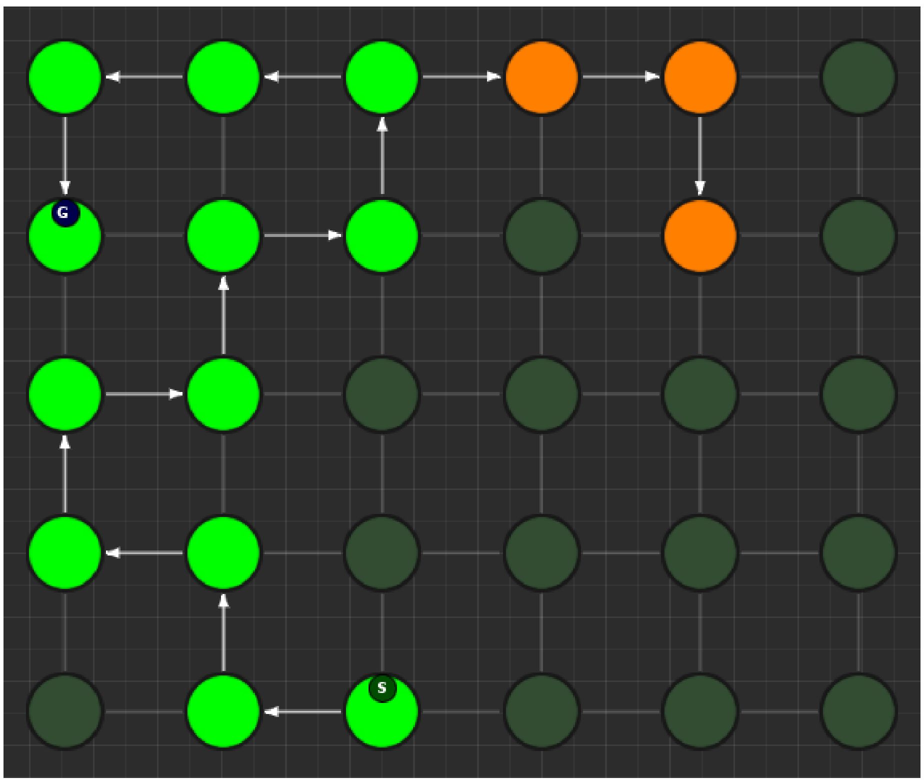
Create a new node `Layout Graph > Create Path`



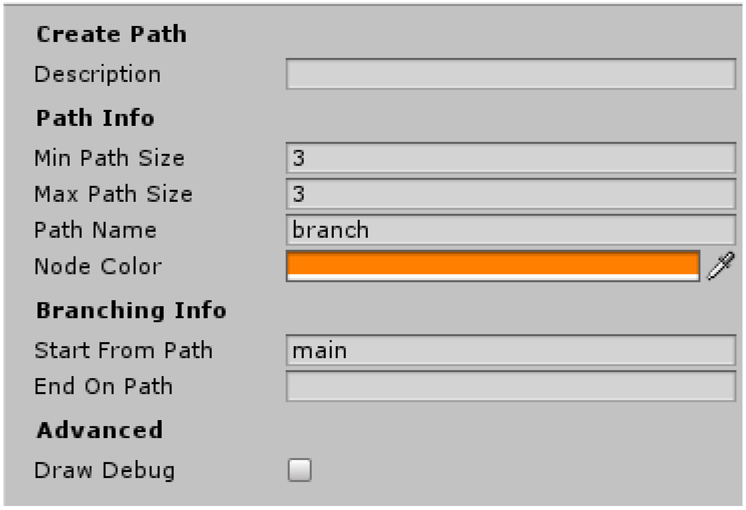
Connect the nodes together like below



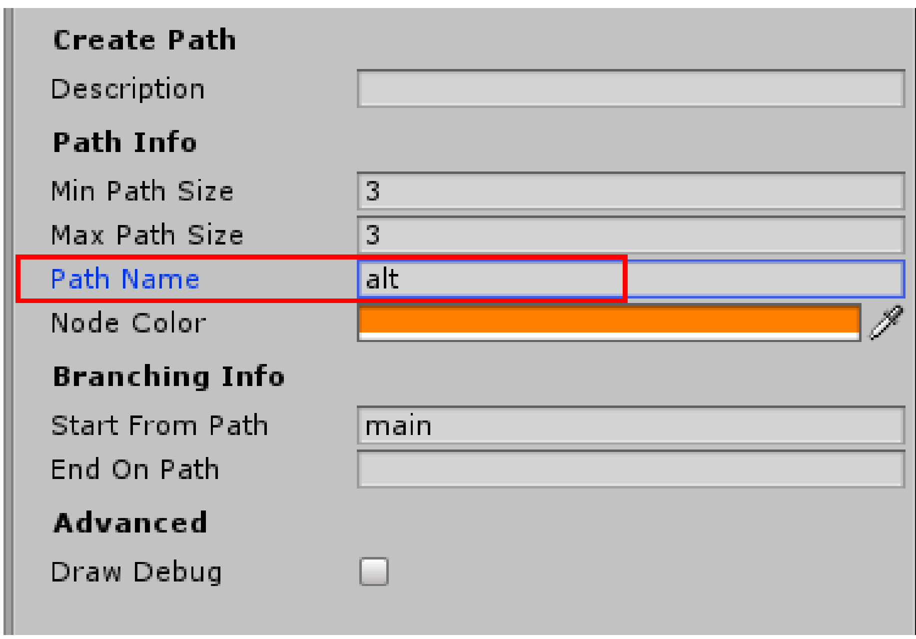
Leave all the properties as default and click build



Select the `Create Path` node and inspect the properties



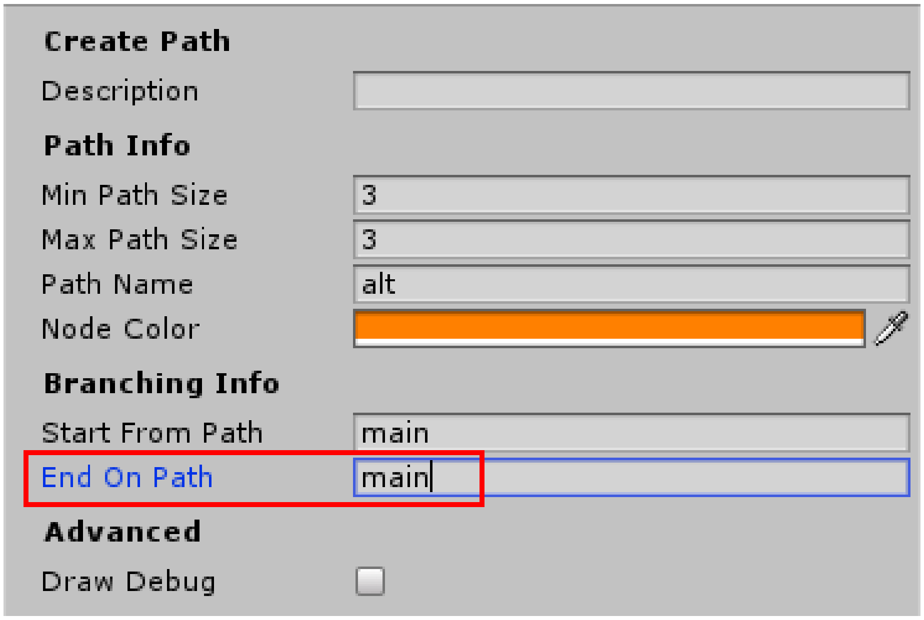
Change the `Path Name` from `path` to `alt`. We will be referencing this path as `alt` in the future



You can specify the paths from which this path should start and end. The `Start From Path` parameter is set to `main`, referencing the main path we created in the previous section

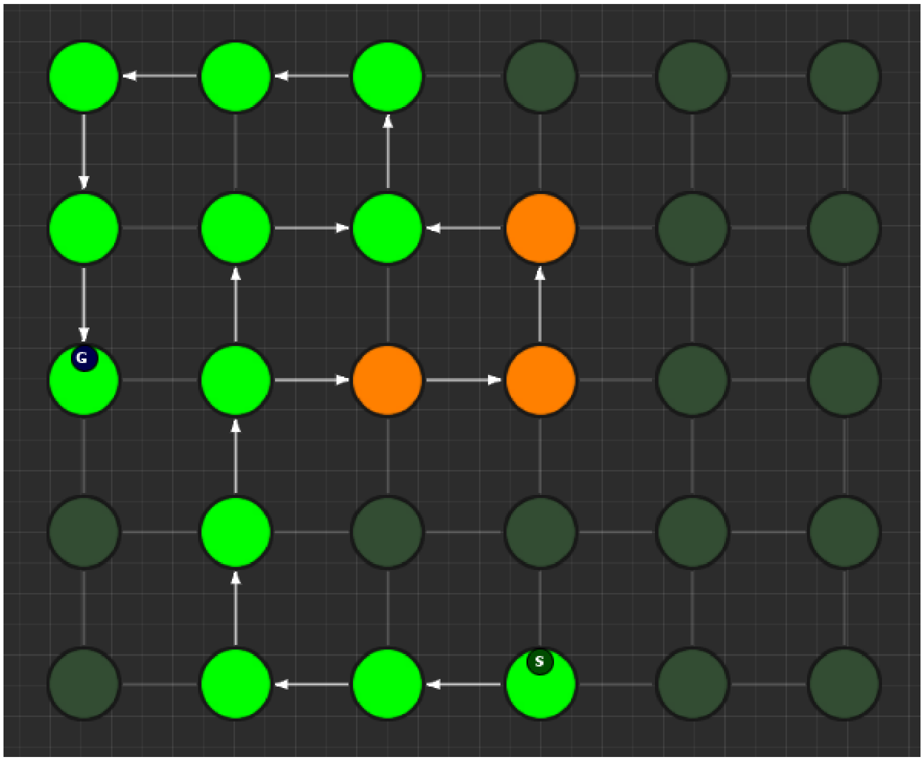
The `End On Path` is left empty, so the end of this path doesn't connect back to anything. We'd like this path to connect back to the main path.

Set the `End On Path` parameter to `main`

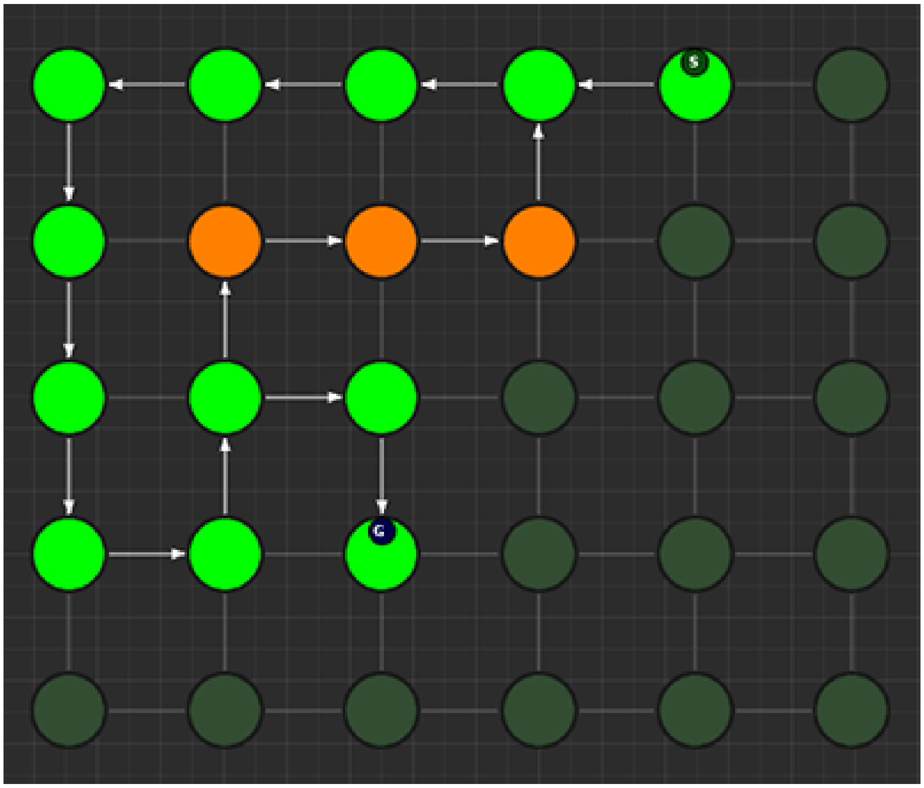


Parameter | Value **Min Path Size |** 3 **Max Path Size |** 3 **Path Name |** alt **Node Color |** orange **Start From Path |** main **End On Path |** main

This will make the alternate path (orange) connect back to the main path (green)



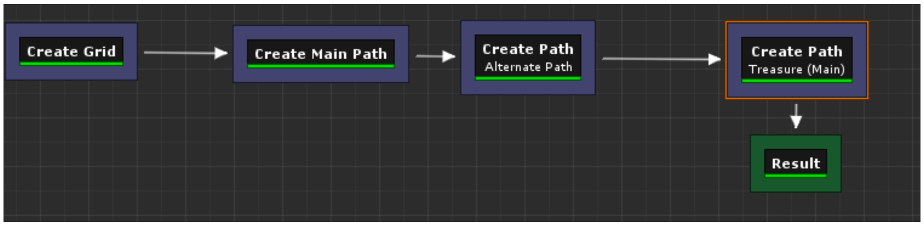
Keep clicking build for different results

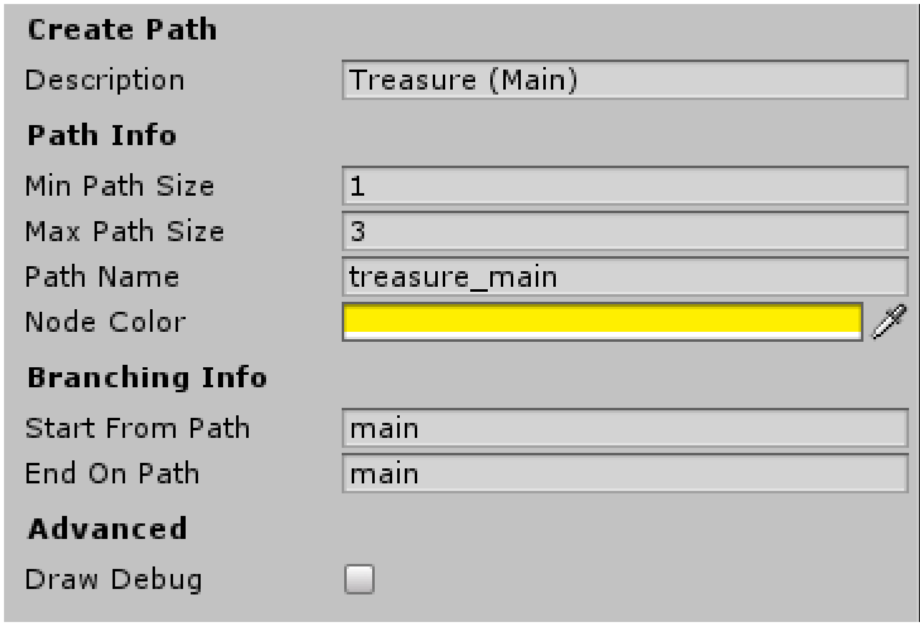


Create Treasure Room (Main)

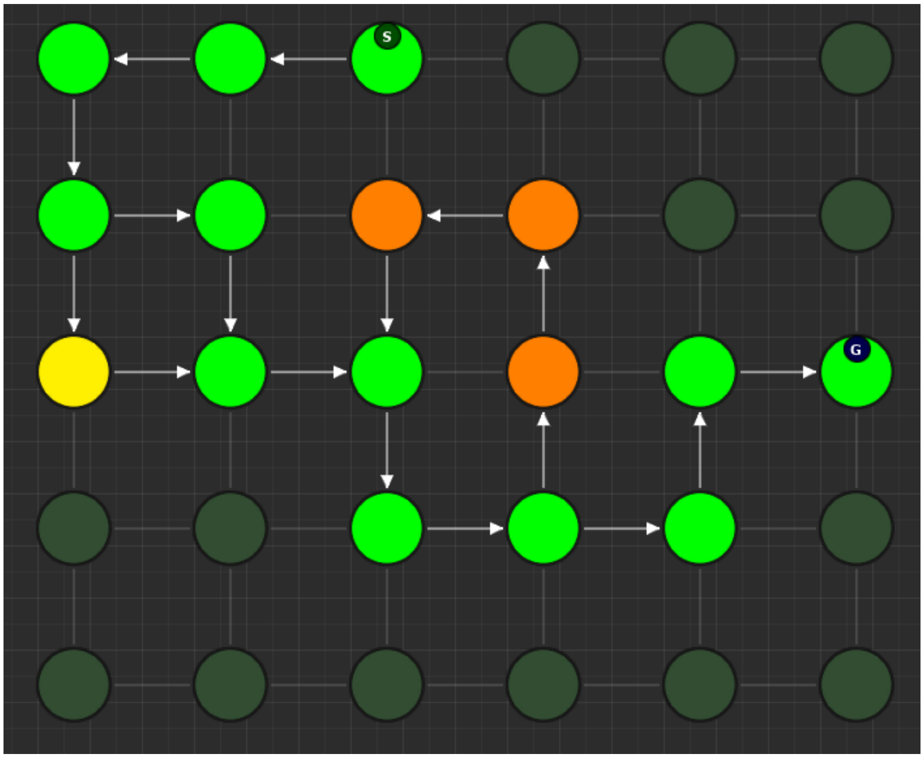
We'll add a treasure room connected to the main path

Add a new node `Layout Graph > Create Path` and set it up as follows:





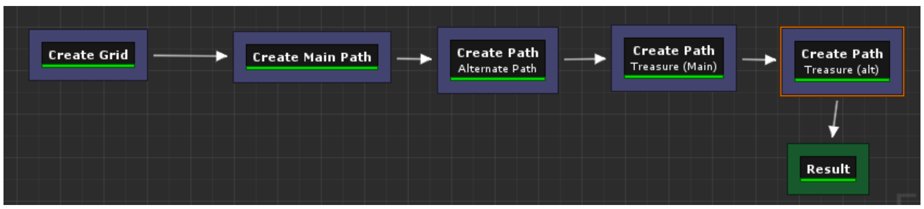
Parameter **|** Value **Min Path Size |** 1 **Max Path Size |** 3 **Path Name |** treasure_main **Node Color |** yellow **Start From Path |** main **End On Path |** main

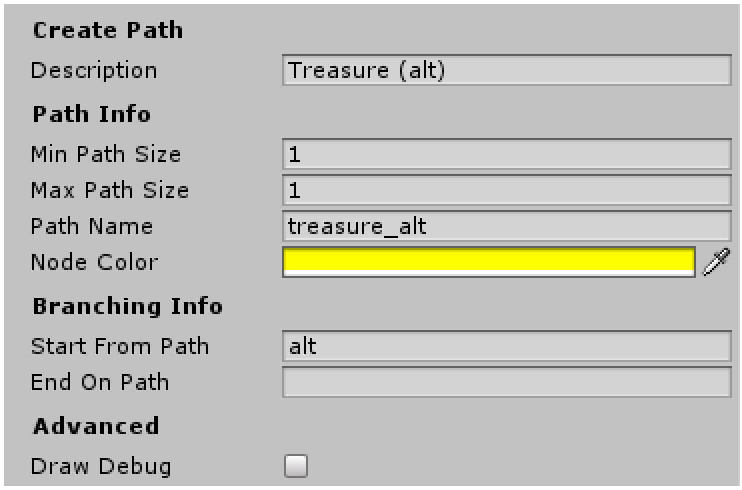


Create Treasure Room (Alt)

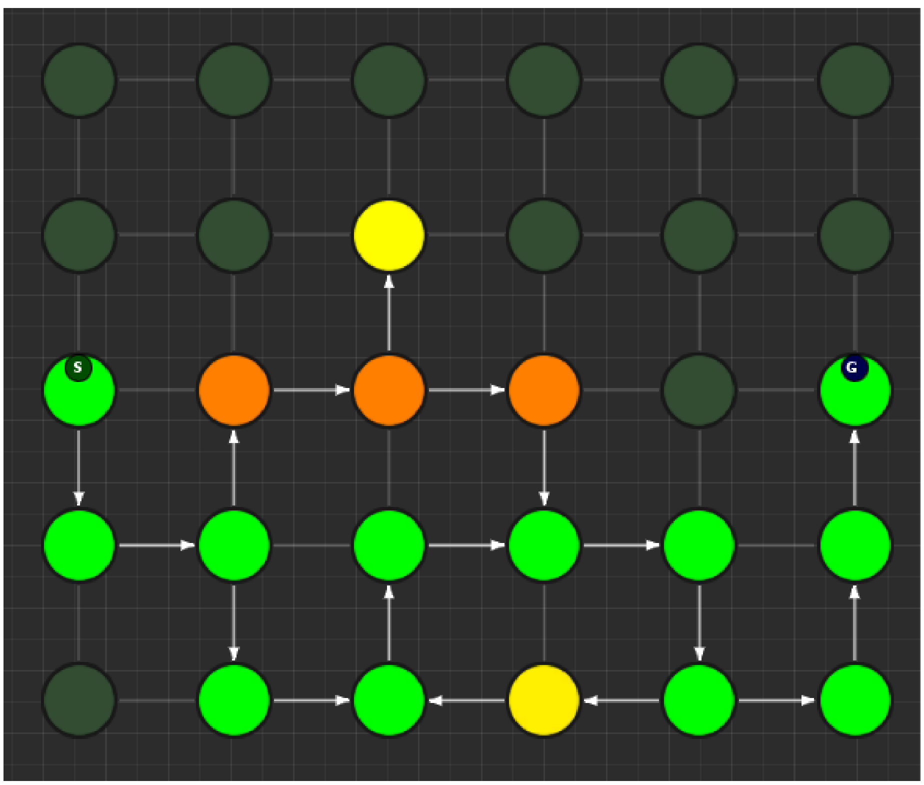
We'll add another treasure room connected to the `alt` path but keep the `End On Path` parameter empty so it doesn't connect back to anything:

Add a new node `Layout Graph > Create Path` and set it up as follows:





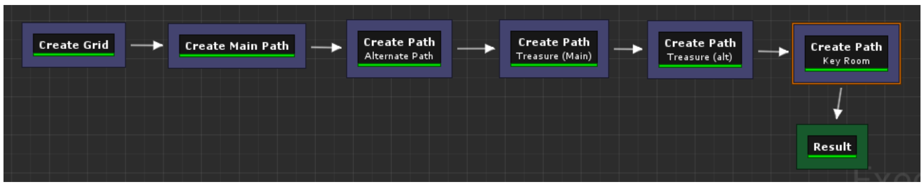
Parameter | Value **Min Path Size** | 1 **Max Path Size** | 1 **Path Name** | treasure_alt **Node Color** | yellow **Start From Path** | alt **End On Path** |

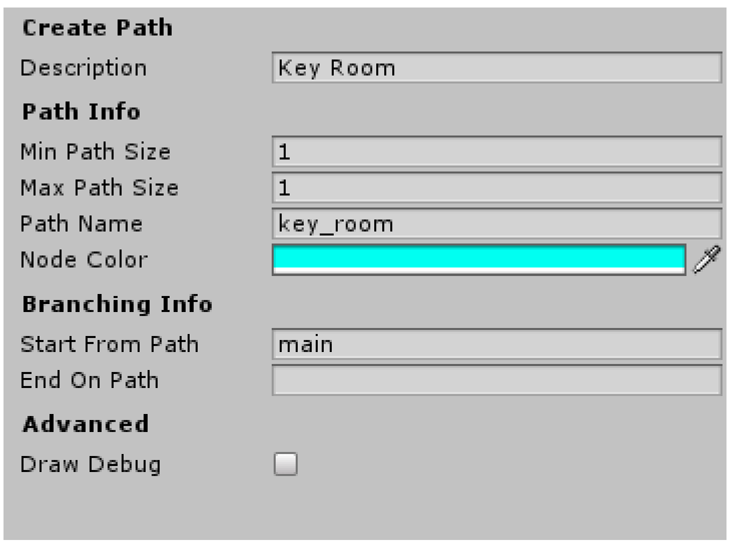


Create Key Room

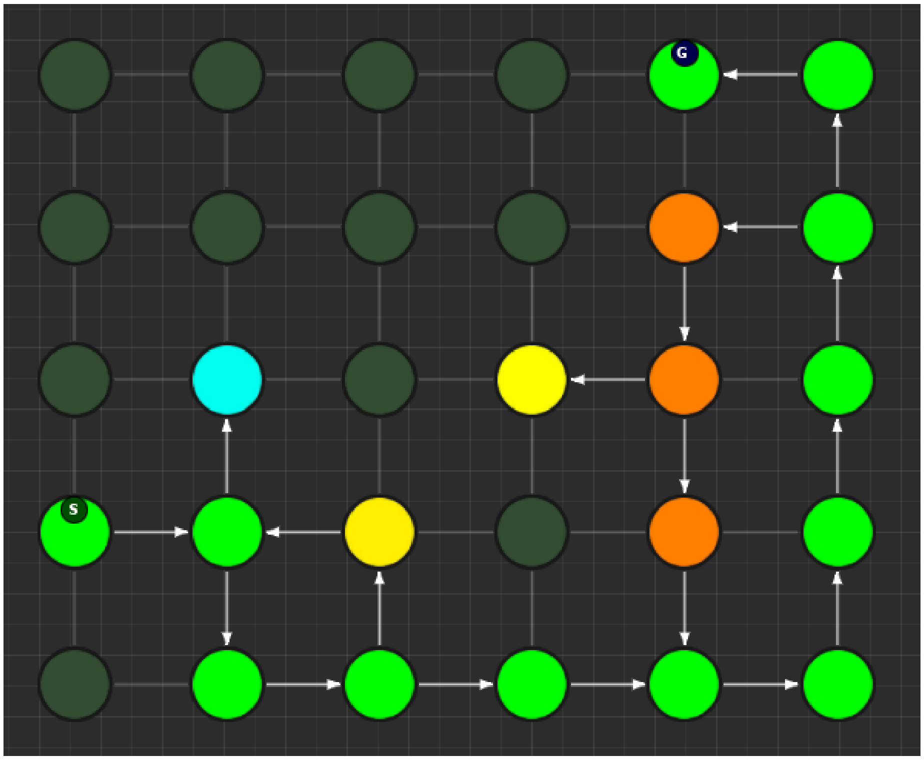
We'll create a room connected to the main path which will act as the key room. We'll later configure this room to have a key that opens up a lock in the main path. It will also have a NPC (key guardian) guarding the key

Add a new node `Layout Graph > Create Path` and set it up as follows:





Parameter **|** Value **Min Path Size |** 1 **Max Path Size |** 1 **Path Name |** key_room **Node Color |** cyan **Start From Path |** main **End On Path |**

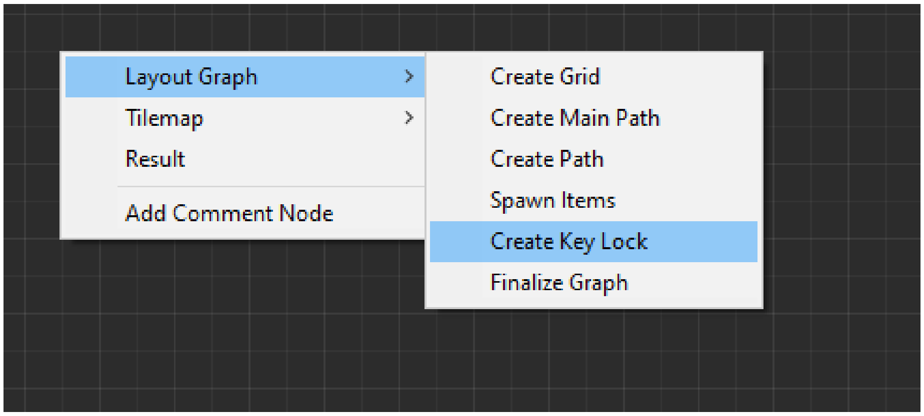


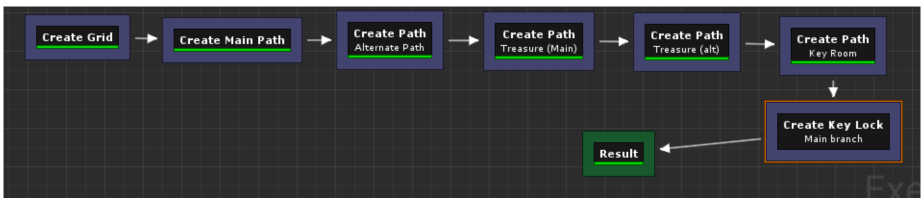
We’ve named this path `key_room`. It will be referenced later on when creating the key locks

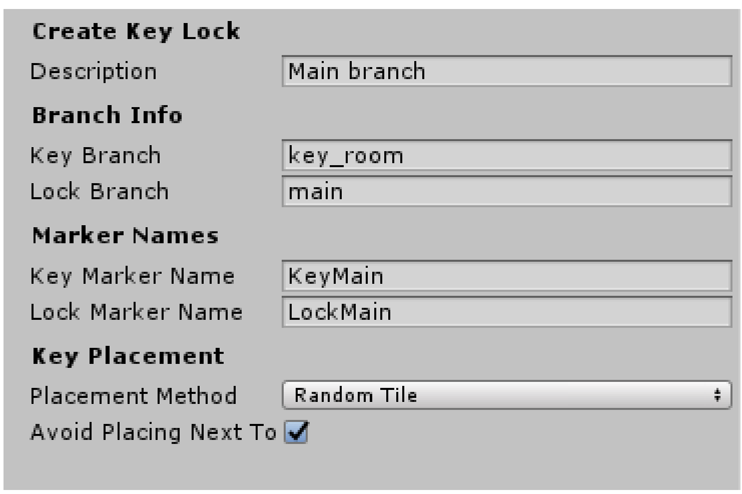
Create Key-Lock (Main)

We’ll next create a key-lock system on the main path. Our key will go on the Key Room we created earlier (`key_room` path) and the lock will be somewhere in the main branch (`main` path)

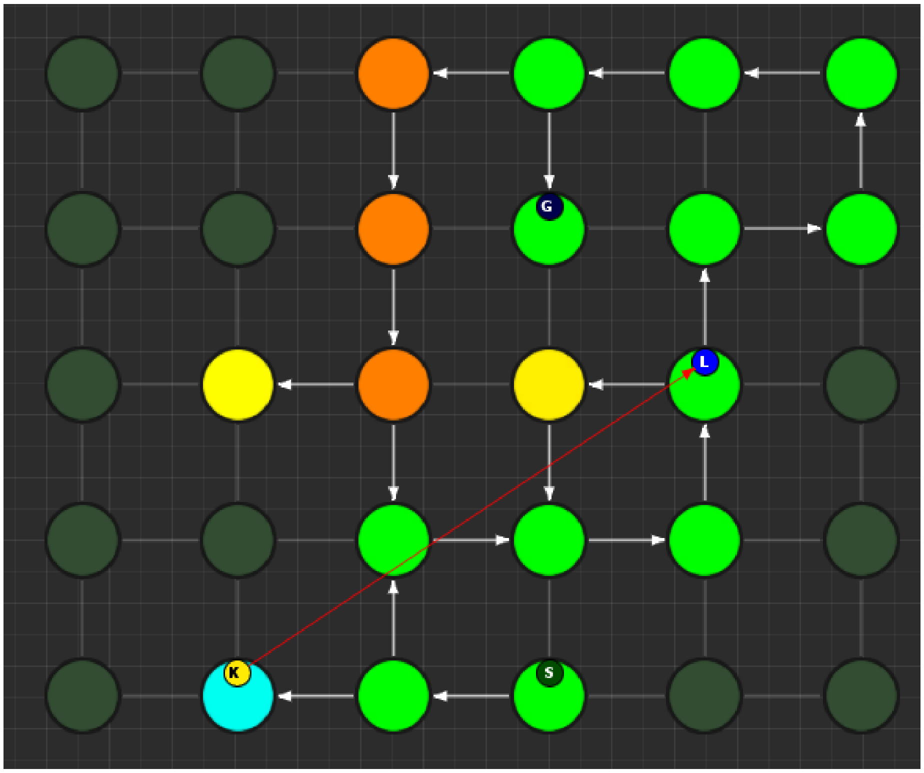
Add a new node `Layout Graph > Create Key Lock` and set it up as follows:







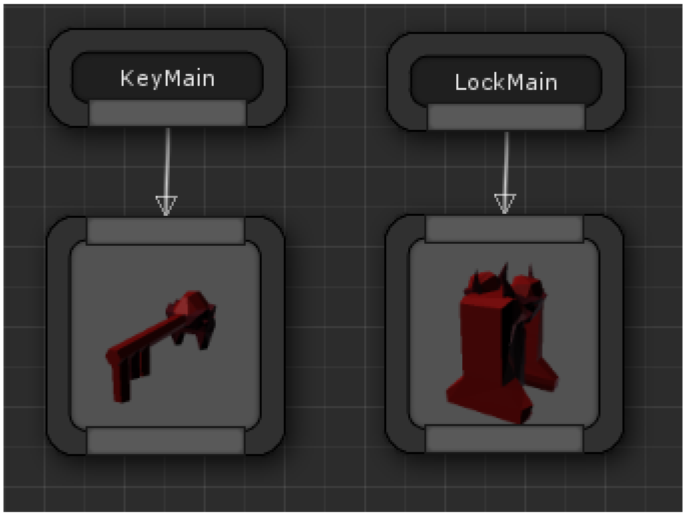
Paramter | Value **Key Branch** | key_room **Lock Branch** | main **Key Marker Name** | KeyMain **Lock Marker Name** | LockMain



Specify the `Key Branch` as `key_room` and `Lock Branch` as `main`

Set marker name for the key as `KeyMain` and lock as `LockMain`. Then in the theme file, you'd create marker nodes with these names and add your key and locked gate prefabs.

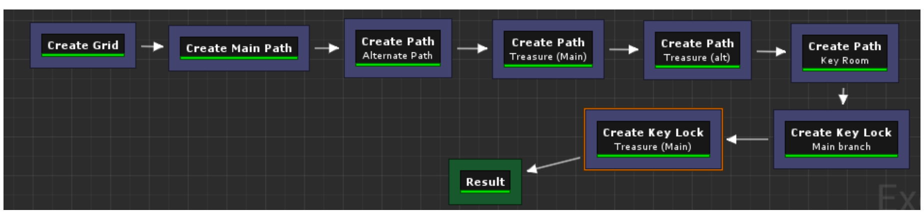
The prehistoric theme already has these setup

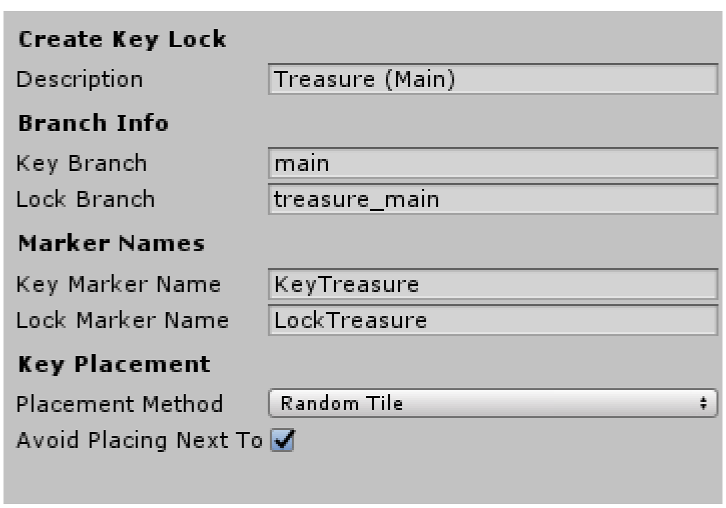


Create Key-Lock (Treasure Main)

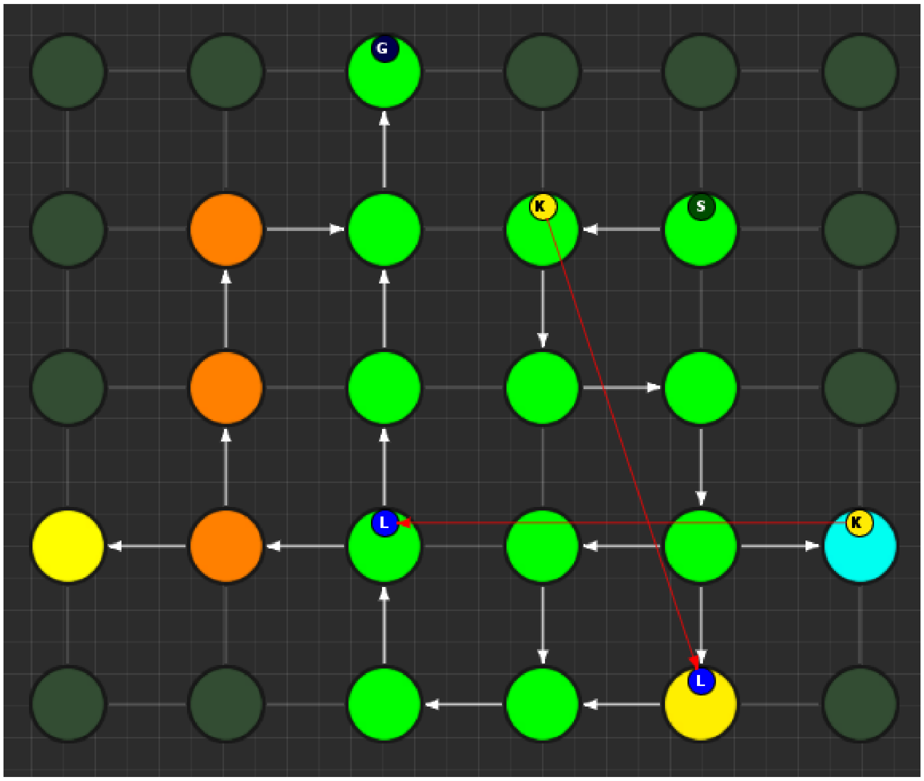
We need a key-lock to guard the treasure room in the main branch

Add a new node `Layout Graph > Create Key Lock` and set it up as follows:



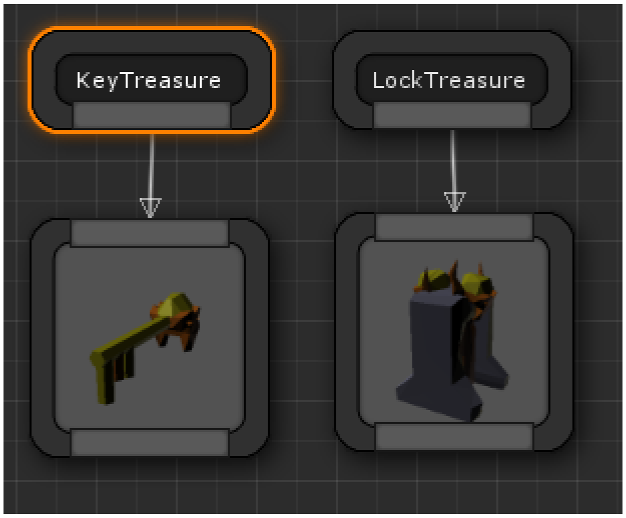


Parameter | Value **Key Branch** | main **Lock Branch** | treasure_main **Key Marker Name** | KeyTreasure **Lock Marker Name** | LockTreasure



Set marker name for the key as `KeyTreasure` and lock as `LockTreasure`. Then in the theme file, you'd create marker nodes with these names and add your key and locked gate prefabs.

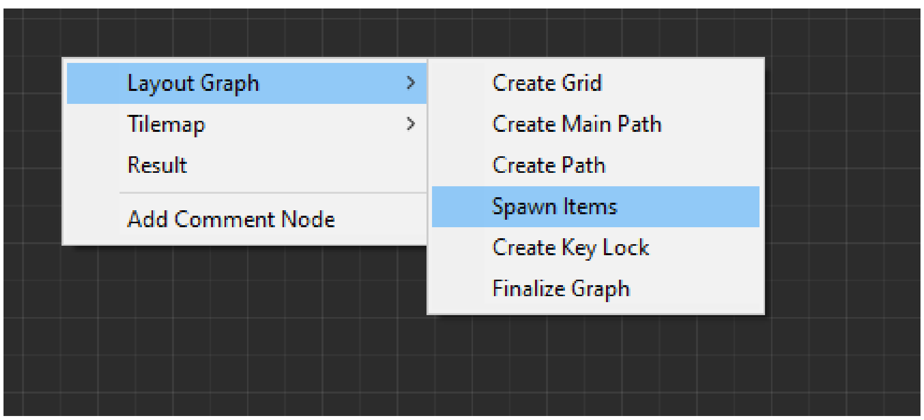
The prehistoric theme already has these setup

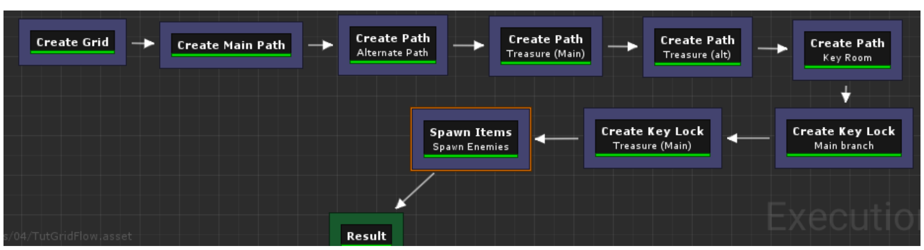


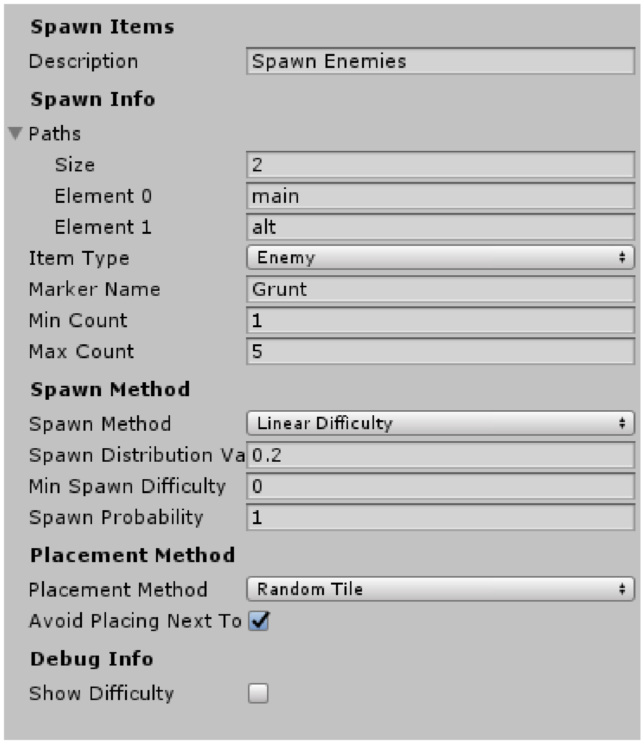
Spawn Enemies (Main, Alt)

We'll use the `Spawn Items` node to spawn enemies on the `main` and `alt` paths

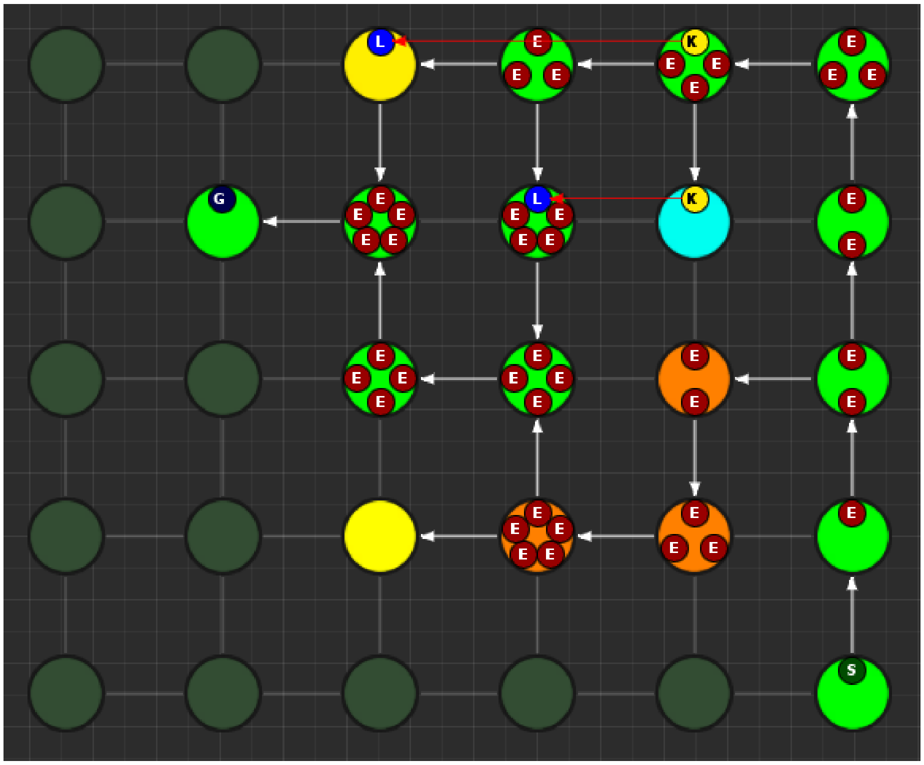
Create a new node `Layout Graph > Spawn Items` and set it up as follows:





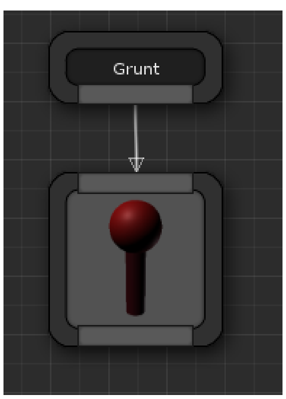


Parameter **|** Value **Paths |** main, alt **Item Type |** Enemy **Marker Name |** Grunt **Min Count |** 1 **Max Count |** 5



This will spawn enemies in the nodes, gradually increasing the number of enemies based on the difficulty. The difficulty increases as we get closer to the goal. You can control this from the `Spawn Method` properties. Leave it to default for now

We've specified the marker name as `Grunt` and an appropriate marker node should be created in the theme file so we can spawn prefabs under it. The pre-historic theme already has this marker

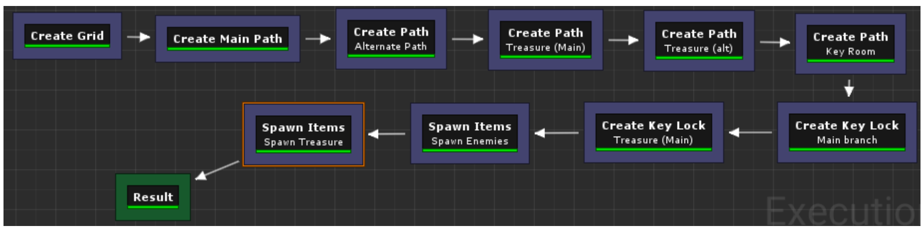


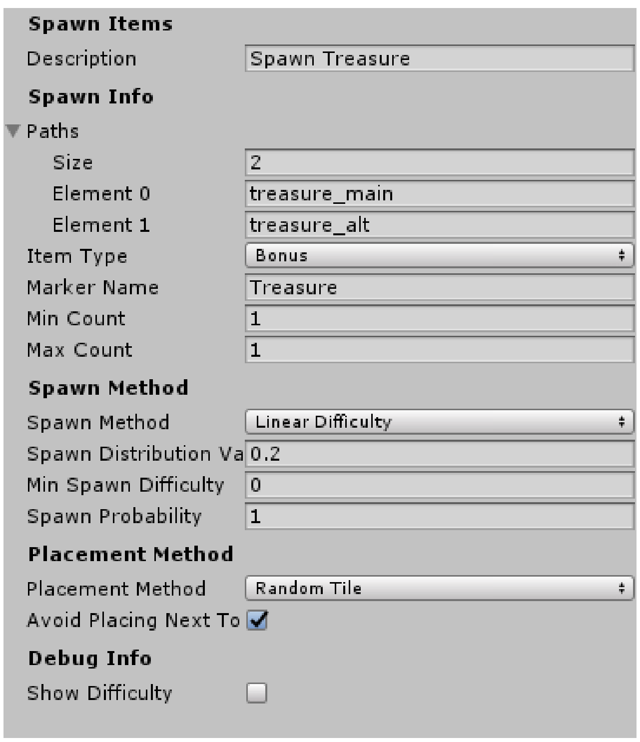
You can control the placement of items (in the tilemap) from the `Placement Method` property section. Leave it to default for now

Spawn Bonus (Treasure Chests)

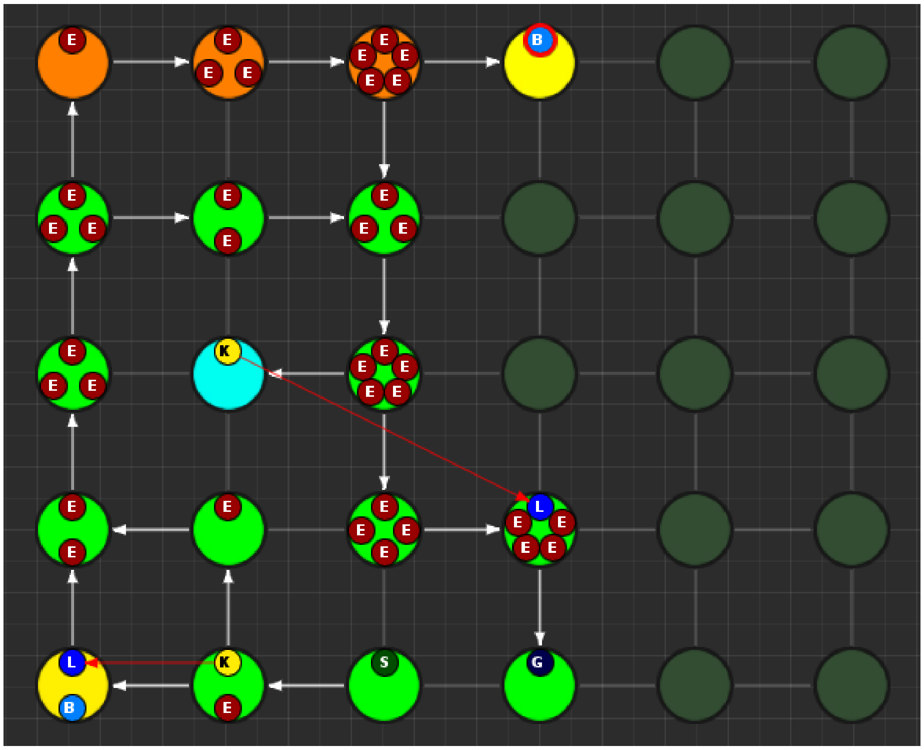
Spawn treasure chests in your bonus rooms using the `Spawn Items` node

Create a new node `Layout Graph > Spawn Items` and set it up as follows:





Parameter | Value **Paths** | treasure_main, treasure_alt **Item Type** | Bonus **Marker Name** | Treasure **Min Count** | 1 **Max Count** | 1 **Min Spawn Difficulty** | 1



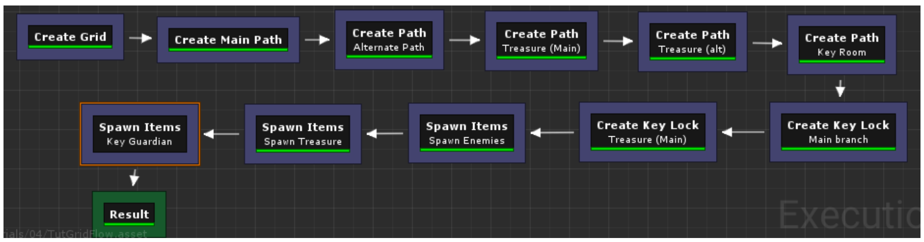
We’ve specified the marker name as `Treasure` and an appropriate marker node should be created in the theme file so we can spawn prefabs the treasure chest under it. The pre-historic theme already has this marker

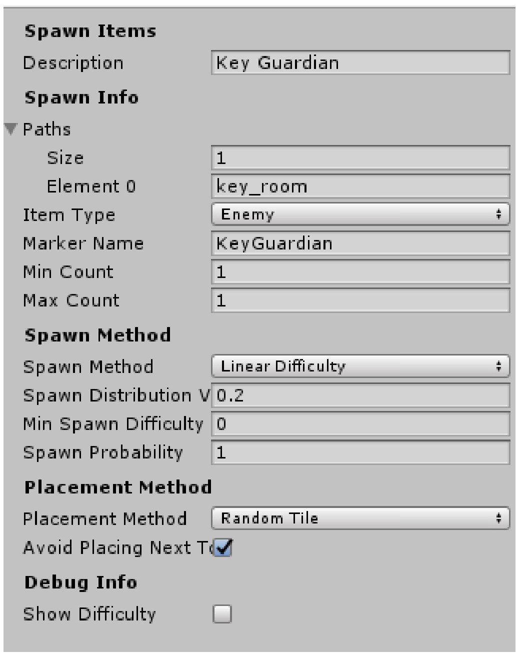
The `Min Spawn Difficulty` is set to `1`. The first node in the branch will have a difficulty of `0` and the last node `1`. Sometimes, the yellow branch may be 3 nodes long. Since we want the chest to occur only on the last node, we’ve set this value to `1`

Spawn Key Guardian

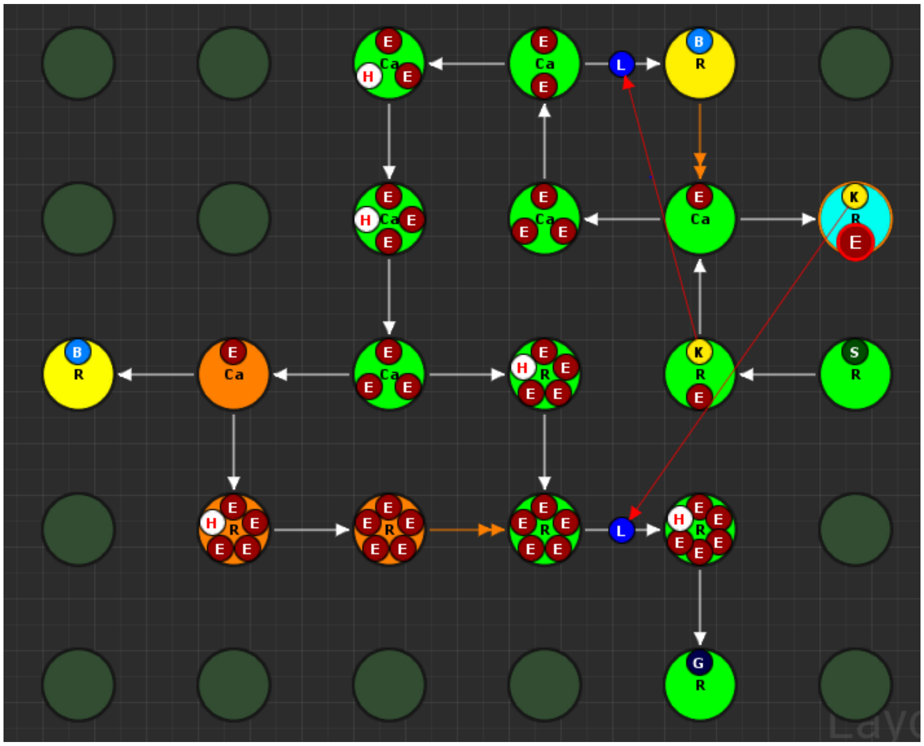
We'll add an NPC in the Key room guarding the key

Create a new node `Layout Graph > Spawn Items` and set it up as follows:





Parameter | Value **Paths** | key_room **Item Type** | Enemy **Marker Name** | KeyGuardian **Min Count** | 1 **Max Count** | 1



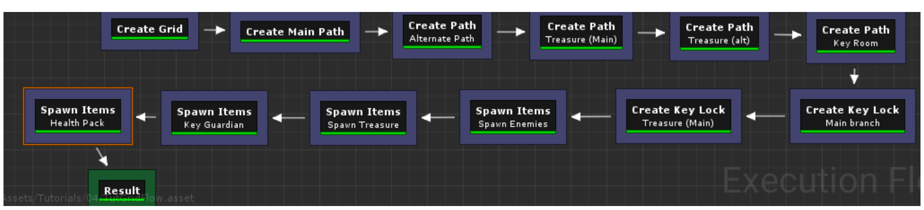
You'll need to create a marker named `KeyGuardian` in the theme file and place your NPC prefab under it. This marker doesn't exist in the `Prehistoric` theme and you'll need to create it yourself if you want to visualize it

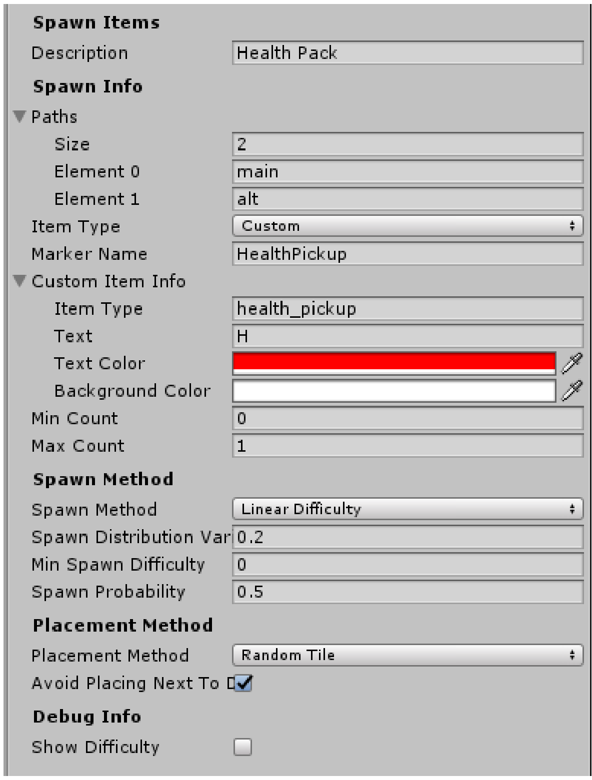
Spawn Health Pack

We'll use the `Spawn Items` node to spawn a few health pickups along the `main` and `alt` paths

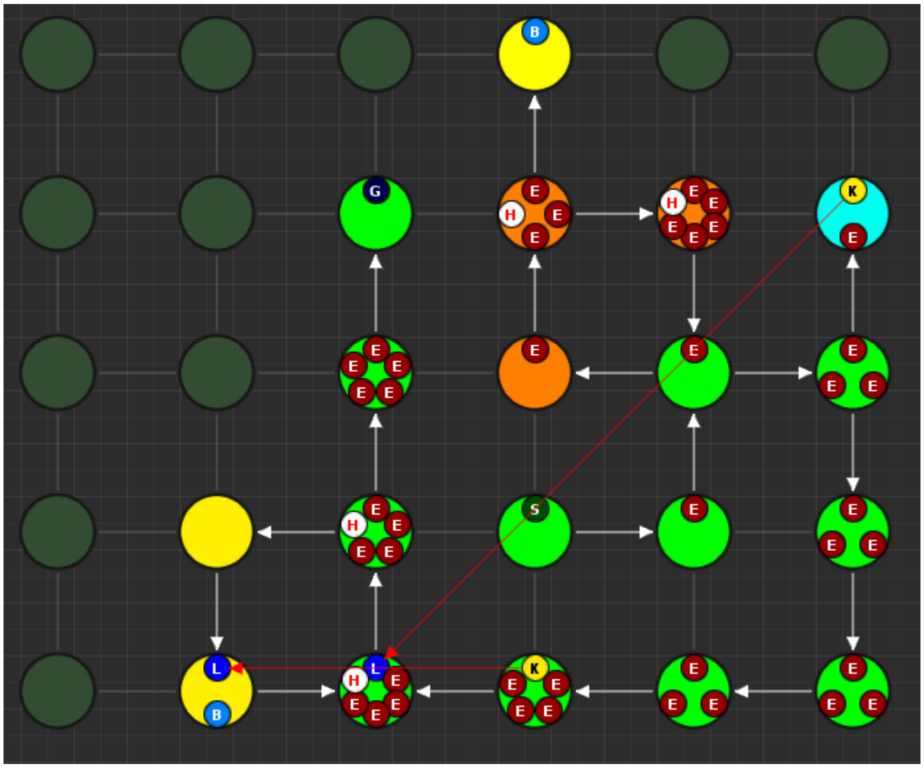
This section also shows you how to use the `Custom` Item Type

Create a new node `Layout Graph > Spawn Items` and set it up as follows:





Parameter | Value **Paths** | main, alt **Item Type** | Custom **Marker Name** | HealthPickup **Min Count** | 0 **Max Count** | 1 **Spawn Probability** | 0.5 Custom Item Info > **Item Type** | health_pickup Custom Item Info > **Text** | Health Custom Item Info > **Text Color** | [Red] Custom Item Info > **Background Color** | [White]



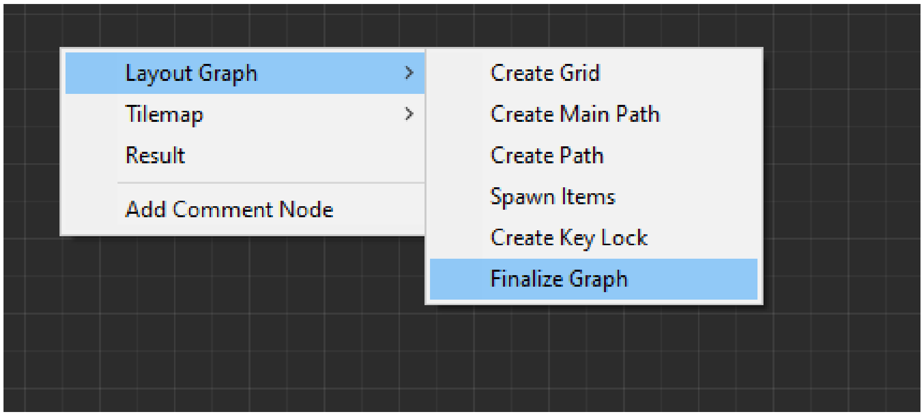
:::note You'll need to create a marker named `HealthPickup` in your theme file and add your health pack prefab :::

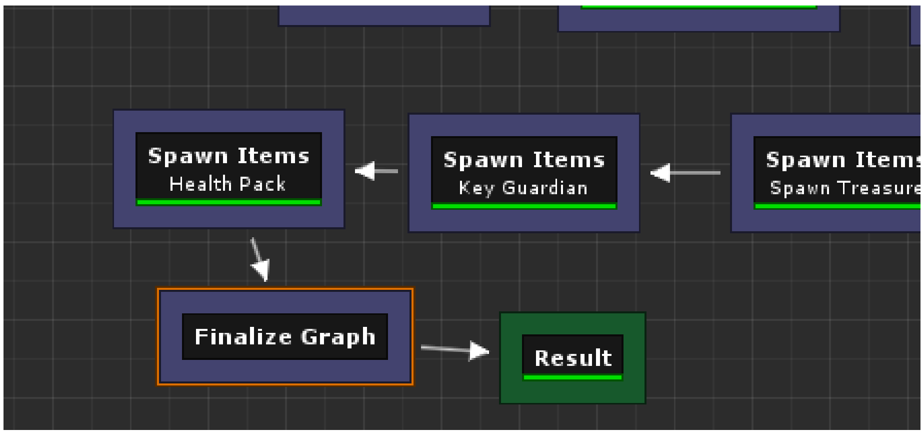
Finalize Layout Graph

After we are done designing the layout graph, we'll need to finalize it with the `Finalize Graph` node. This node does a few things:

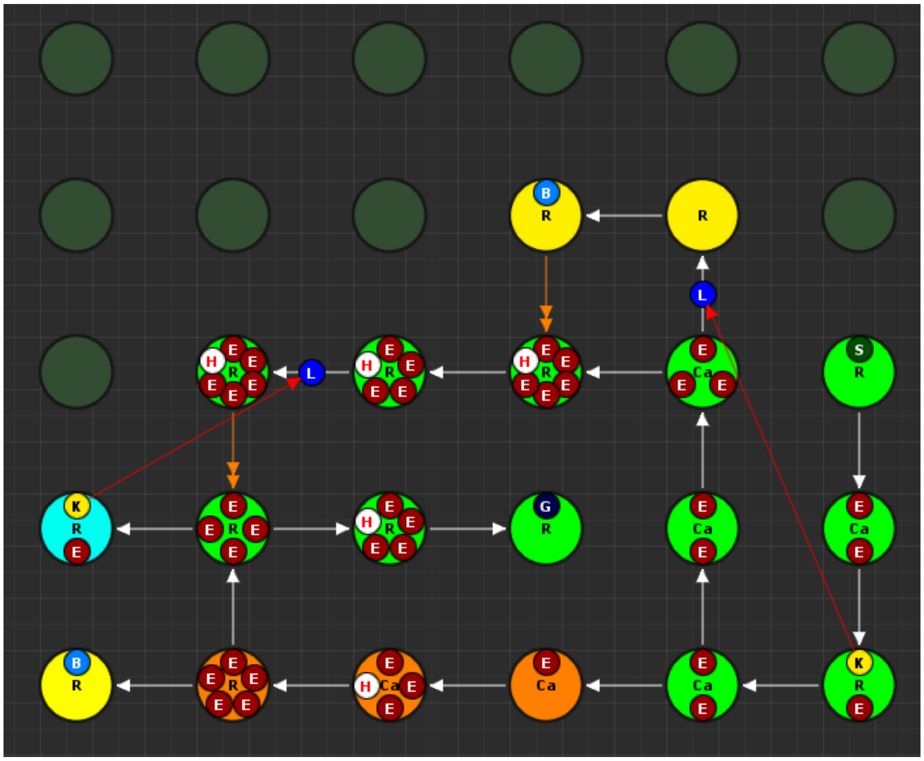
- Move the locks from the nodes on to the links
- Create one way doors (so we don't go around locked doors)
- Assign room types (Room, Corridor, Cave)

Create a new node `Layout Graph > Finalize Graph` and set it up as follows:

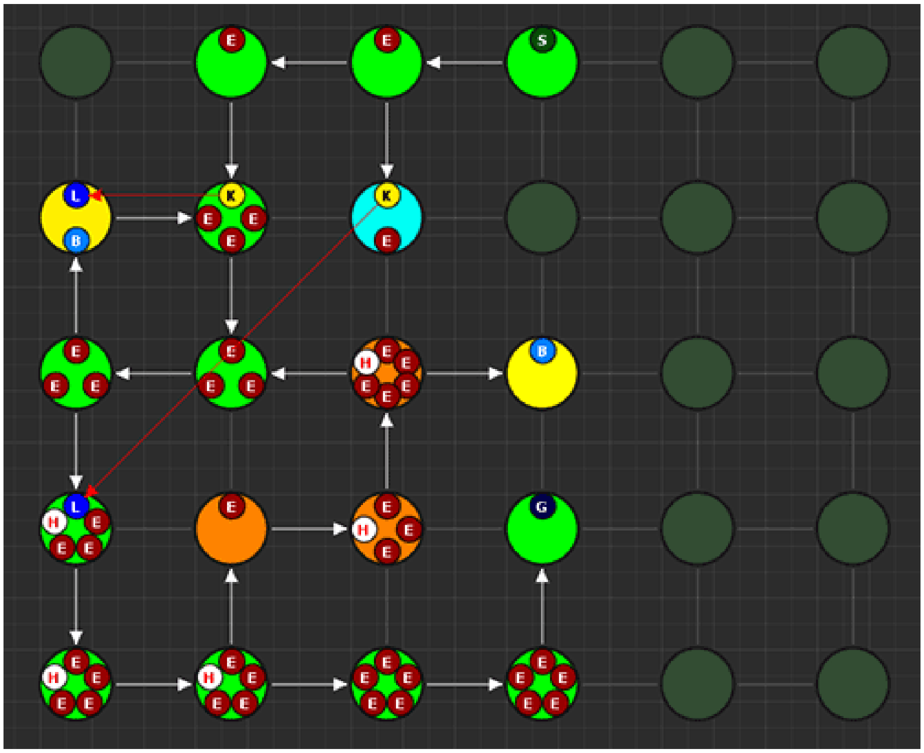




Leave all the properties to default



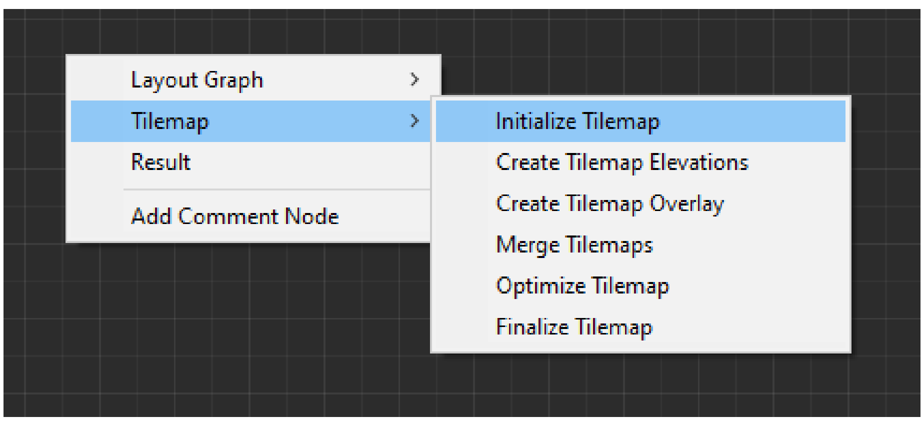
B
R
R
H E E R E E E
L
H E R E E
H E E R E E E
E Ca E E
L
S
R
K
R
E
E E R E E
H E R E E E
G
R
E Ca E
E Ca E
B
R
E E E R E E
E H Ca E E
E
Ca
E Ca E
K
R
E

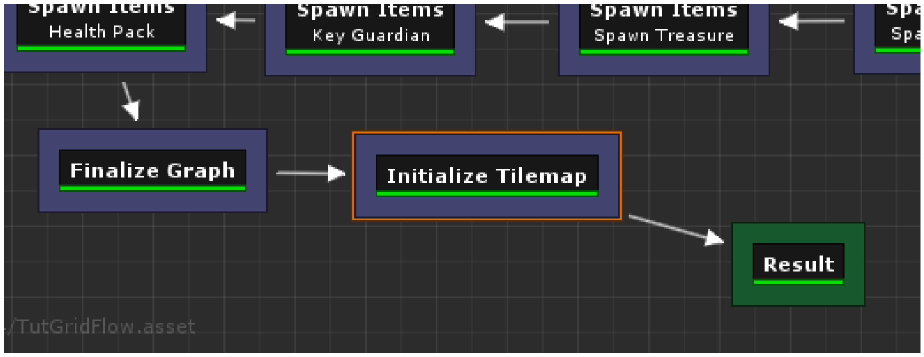


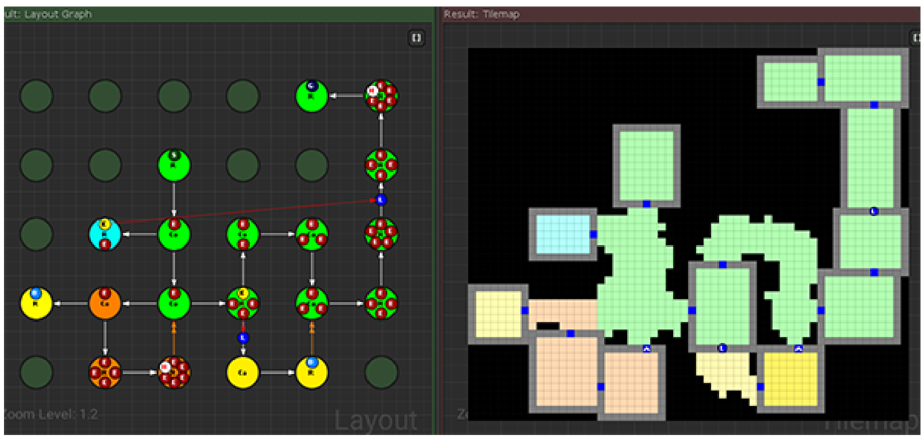
We are now ready to create a tilemap from this

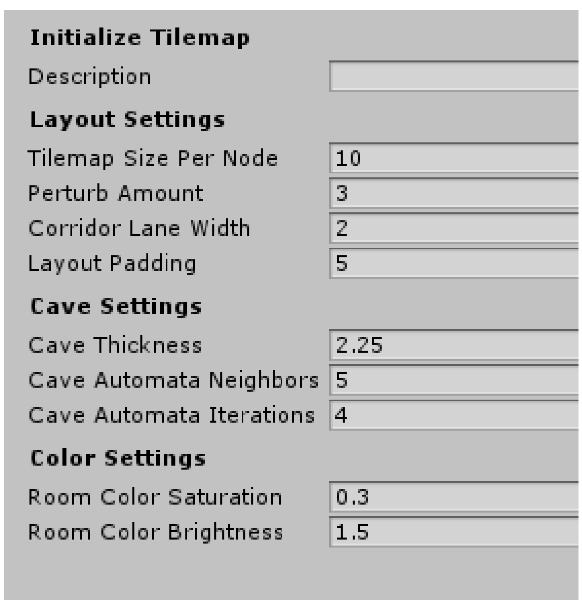
Initialize Tilemap

Create a new node `Tilemap > Initialize Tilemap` and set it up as follows:









You can control the thickness of the caves from the `Cave Thickness` parameter. Each node on the layout graph gets converted into rooms in the tilemap.

The parameter `Tilemap Size Per Node` controls how many tiles are used to generate a room from the node. Bump this number up if you want more space in your rooms

If you want a more uniform grid like look on your rooms, bring the `Perturb Amount` close to `0`

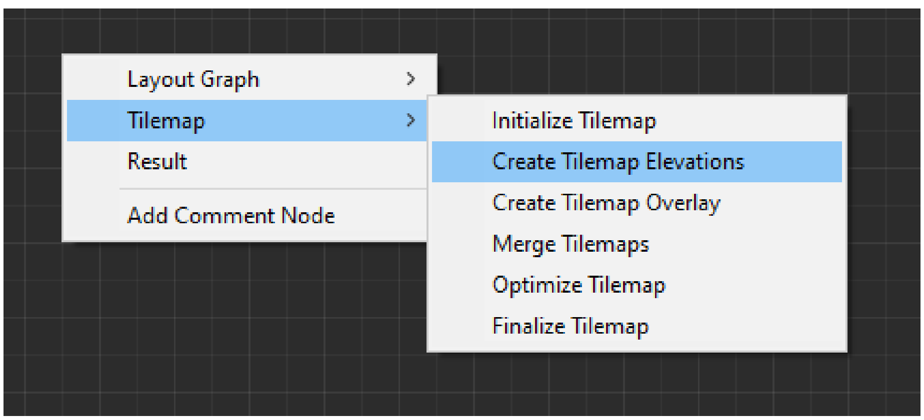
`Layout Padding` adds extra tiles around the dungeon layout. Set to `5` so we can apply some decorations outside the dungeon bounds

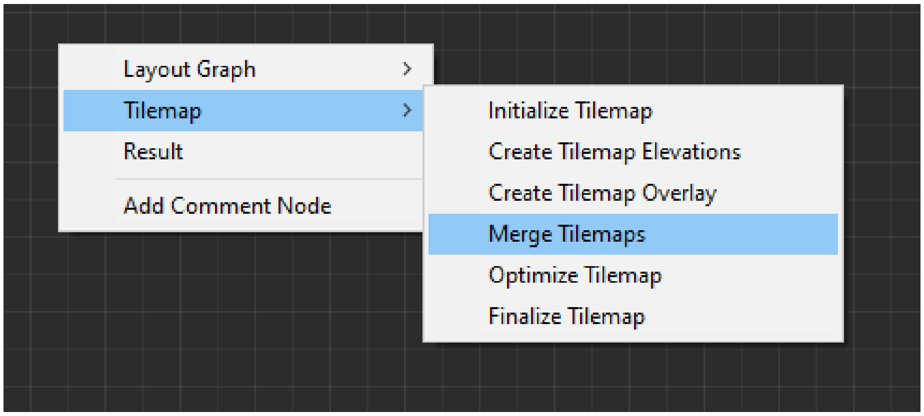
When you select a node on the layout graph, the tiles that belong to the node light up. This is controlled by the `Color Settings` parameters

Add Background Elevation

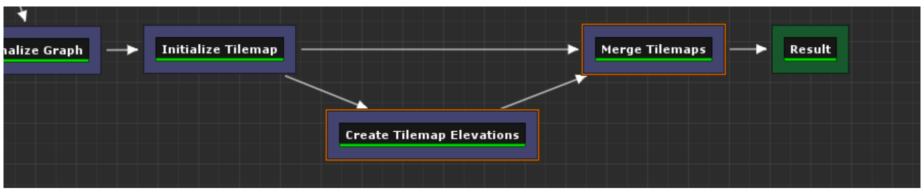
We are going to create overlays and merge them with the original tilemap. Create the following two nodes:

- Create a node `Tilemap > Create Tilemap Elevations`
- Create a node `Tilemap > Merge Tilemaps`





Link them up like below:



Update the properties

Parameter | Value **Noise Frequency** | 0.1 **Num Steps** | 8 **Min Height** | 0.5 **Max Height** | 3.5 **Sea Level** | -1

We've specified the marker name as `Rock`. If you place objects under the specified marker node in the theme editor, they will show up on these tiles at the given height

The Min/Max height is logical and will be mulitplied by the dungeon config's Grid Size Y value. If the GridSize is `(4, 2, 4)` in the DungeonGridFlow game object's config and the tile height happens to be `2.5`, the actual placement will be on `2.5 * 2 = 5`

Add Tree Overlays

We'll overlay trees on our dungeon using a noise parameter. These overlays will be placed such that they will not block the main path

Create a node `Tilemap > Create Tilemap Overlay`

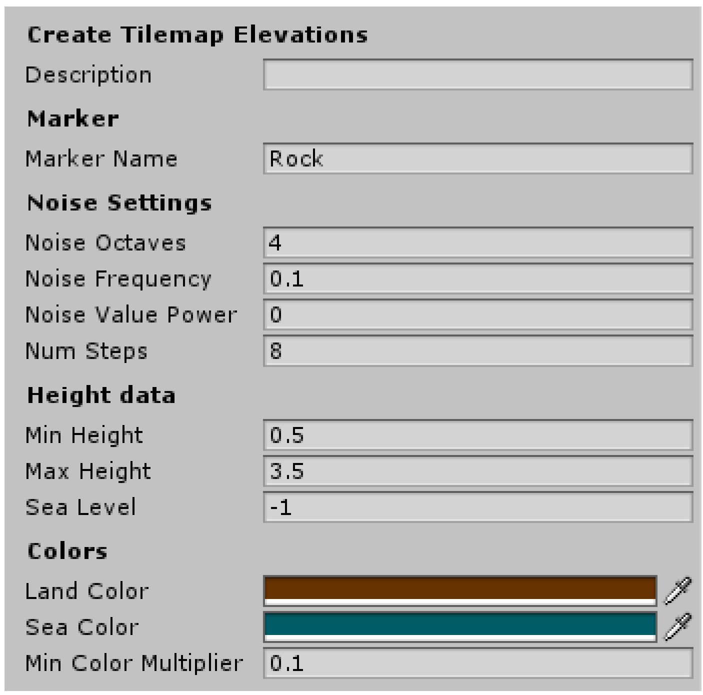


Figure 18: Create Tilemap Elevation properties

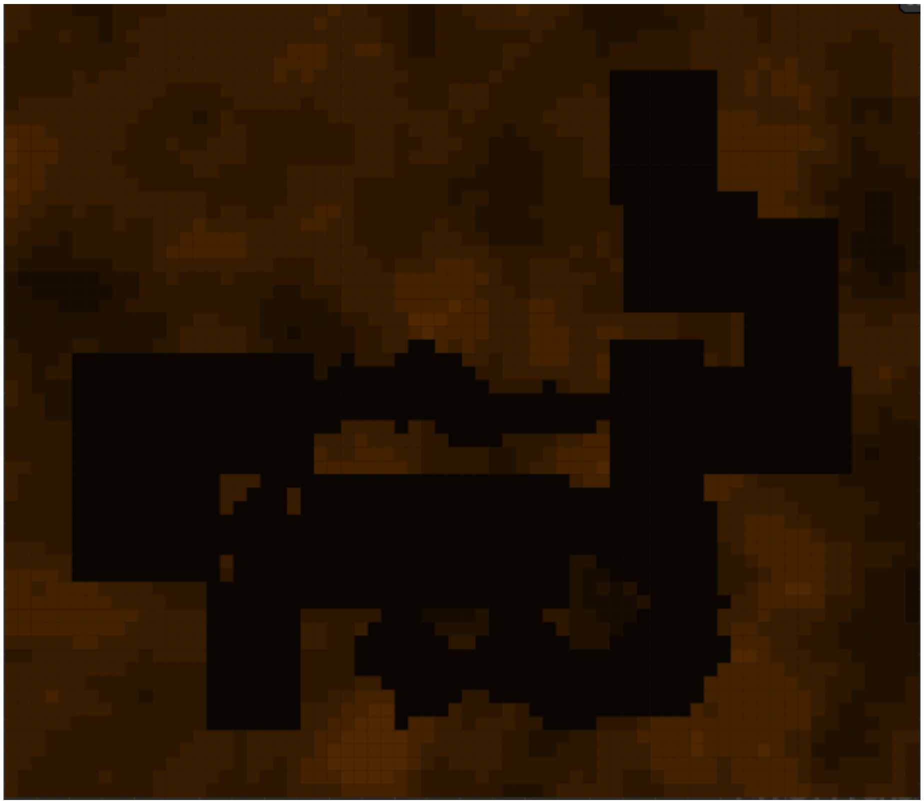


Figure 19: Create Tilemap Elevation Node Result

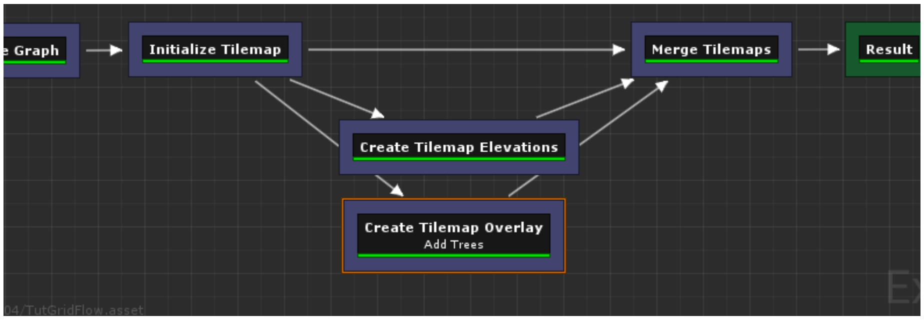
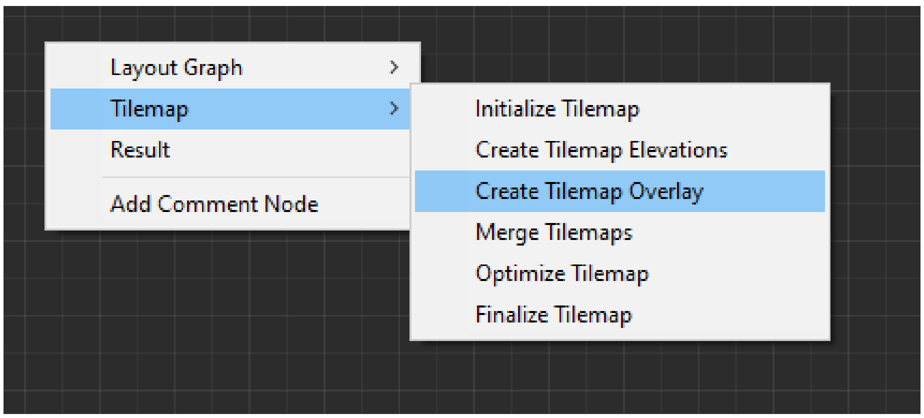


Figure 20: Create Tilemap Overlay Node Connection



Noise Settings

Parameter | Value Noise Frequency | 0.2 Noise Max Value | 1.5 Noise Threshold | 0.75 Min Dist From Main Path | 1

Merge Config

Parameter | Value Max Height | 1

Finalize Tilemap

Finalize the tilemap to complete the grid flow graph

Create a node `Tilemap > Finalize Tilemap`

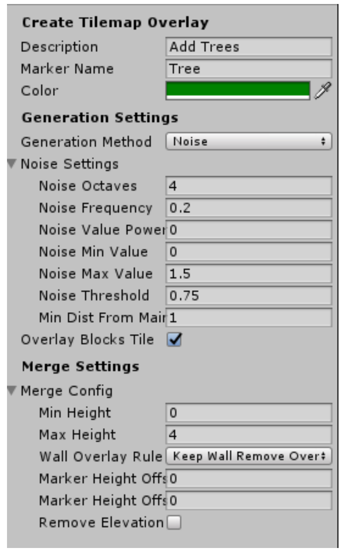


Figure 21: Create Tilemap Overlay Node properties

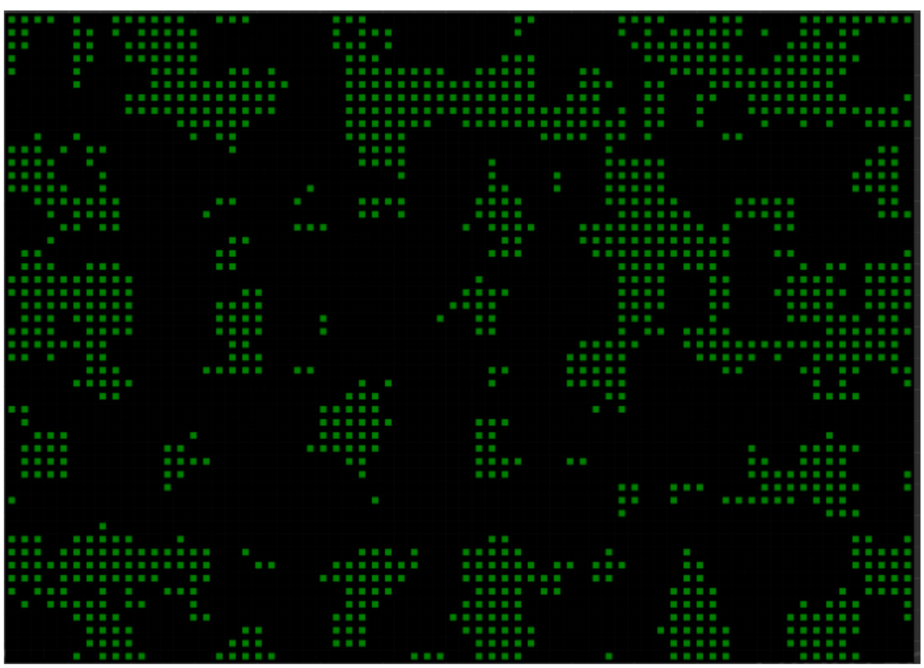


Figure 22: Result of the Create Tilemap Overlay Node

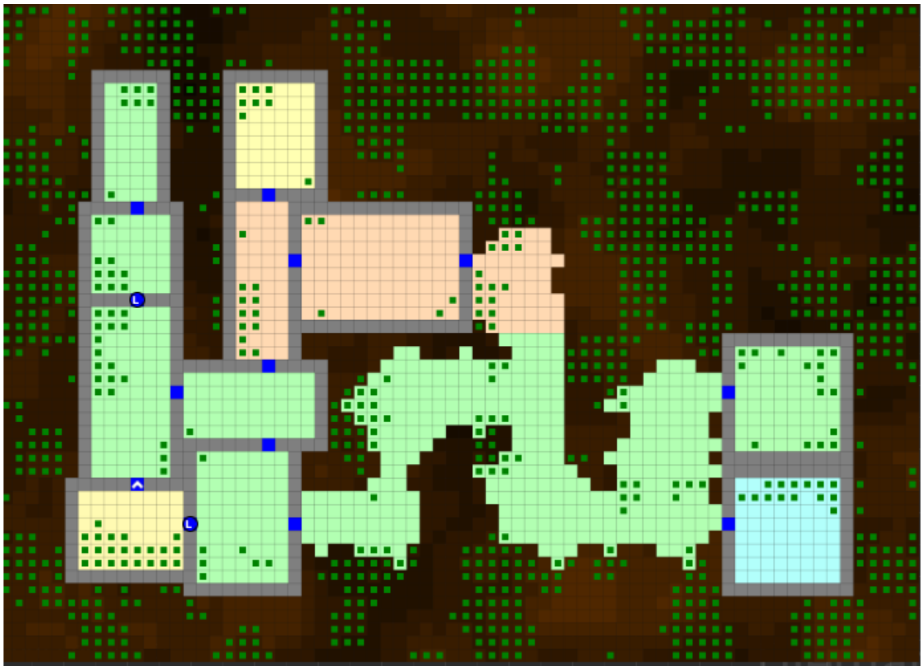
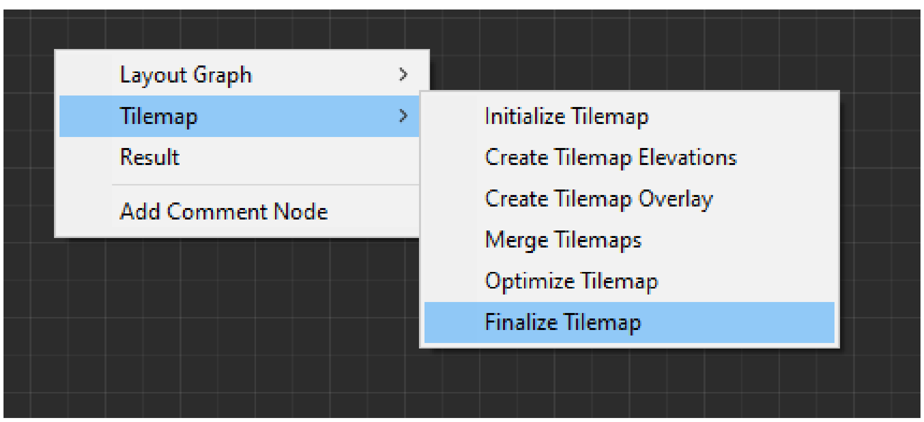
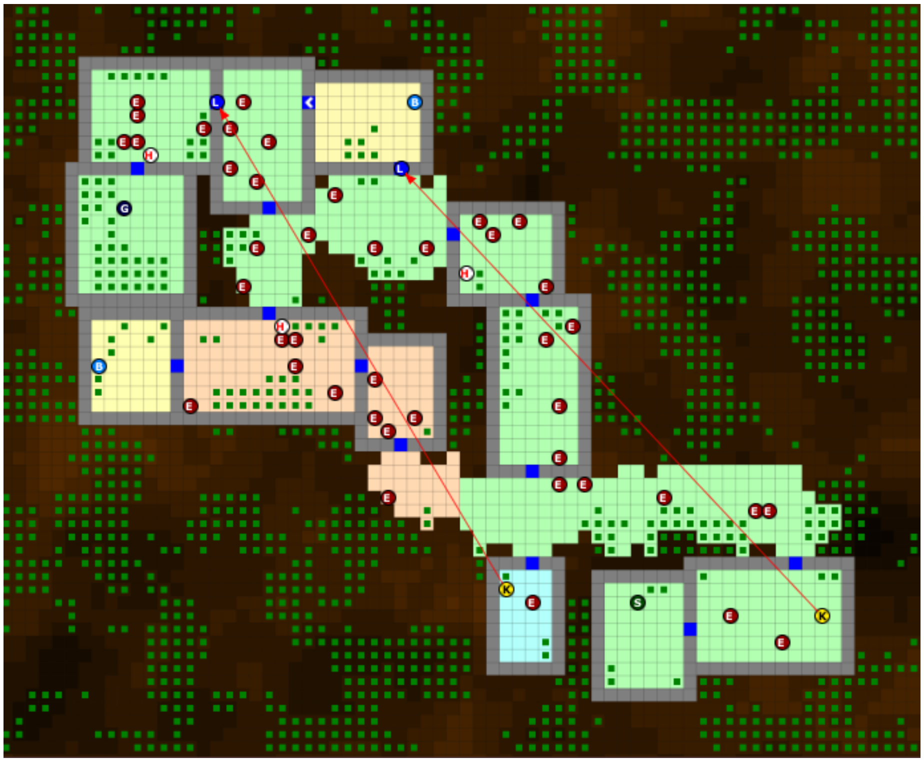


Figure 23: Merged result

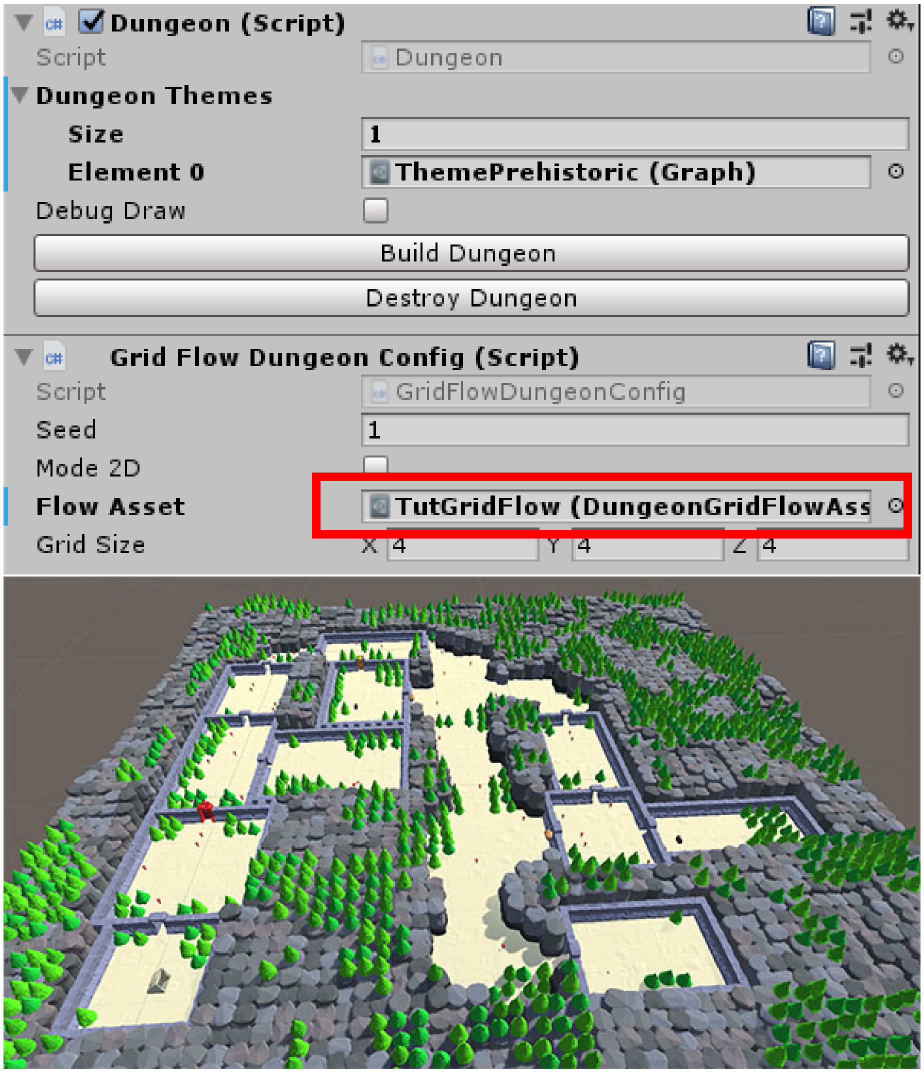




Finalize Tilemap node places all the items on to the tilemap (enemies, keys, bonus etc)

Build Dungeon

Assign this grid flow graph to your DungeonGridFlow game object and click `Build Dungeon`

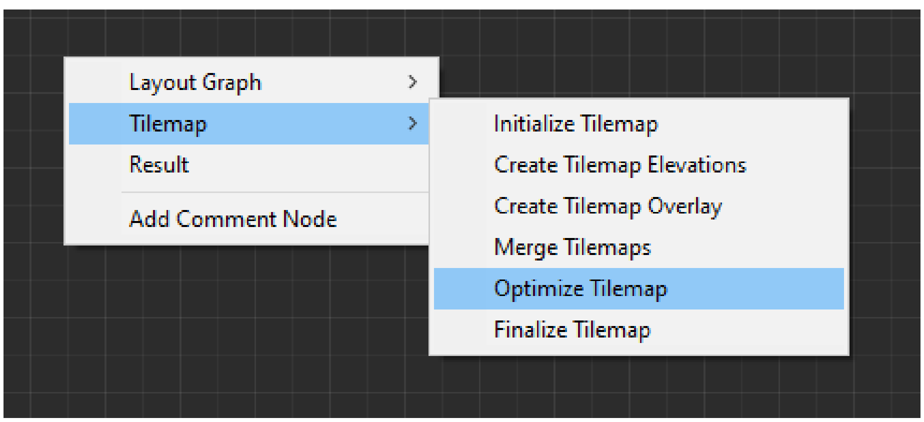


Optimize Tilemap

When the tilemap based level is generated, there are many tiles that the player might never see, as they are far away from the dungeon layout

The `Optimize Tilemap` removes tiles that are away from the specified distance from the dungeon layout bounds

Create a node `Tilemap > Optimize Tilemap`



Connect it before the `Finalize Tilemap` node like below:

Figure 24: Optimize Tilemap Node Result

Figure 25: Optimize Tilemap Before / After

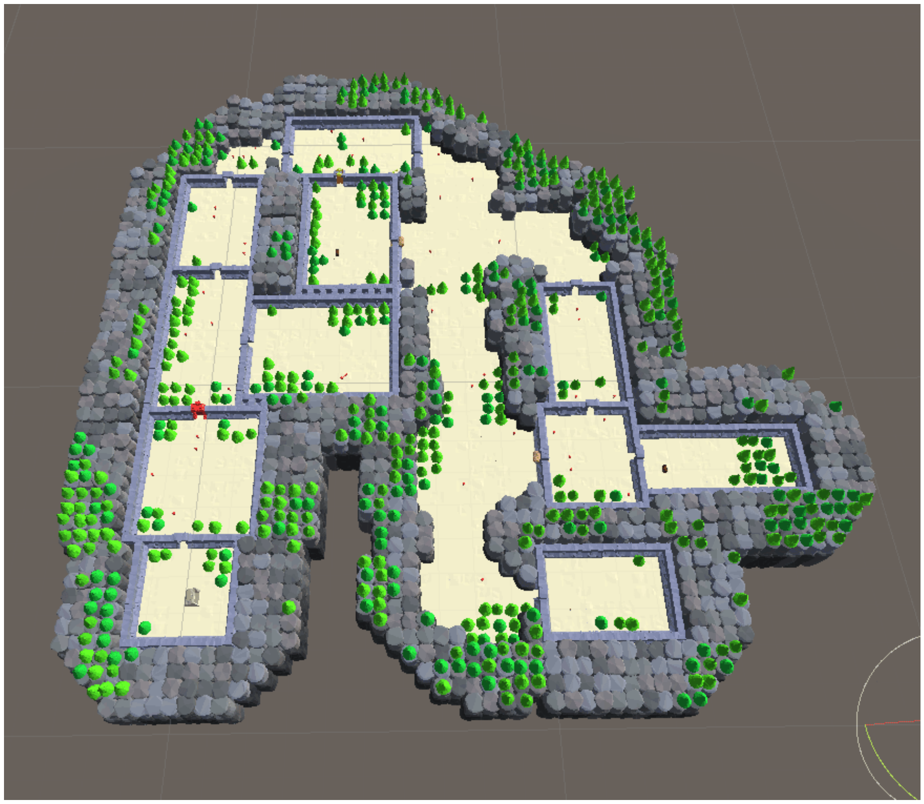
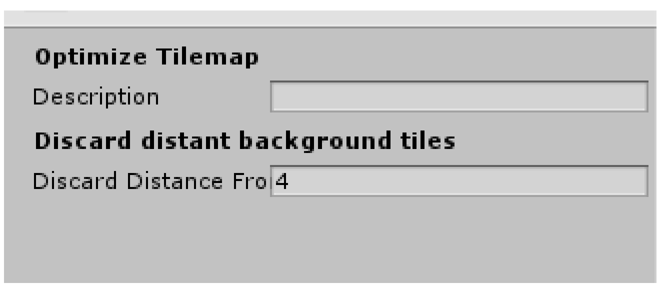


Figure 26: Optimize Tilemap Before / After



Rebuild the dungeon in the scene view

Key Lock System

The spawned Key and Lock game objects will have the following components attached to it by Dungeon Architect

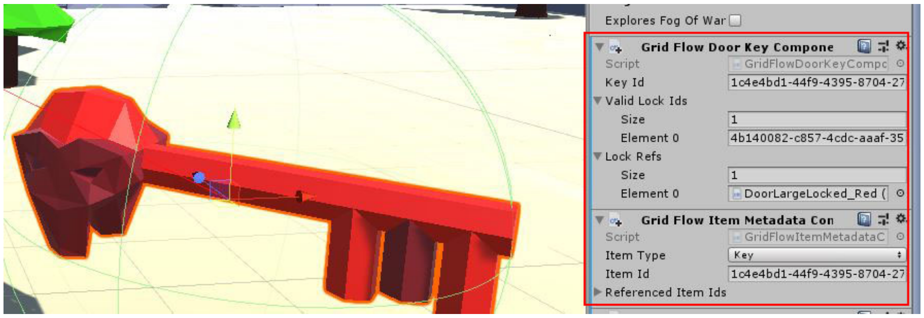


Figure 27: New Components attached to the Key Prefab

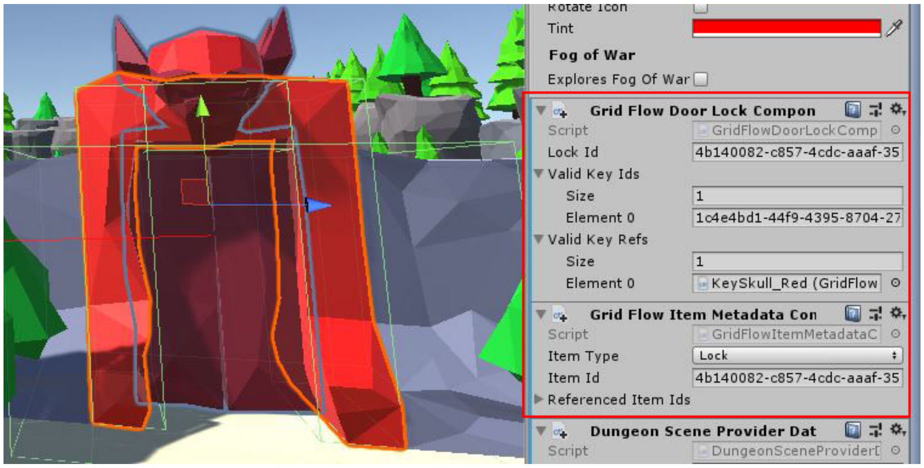
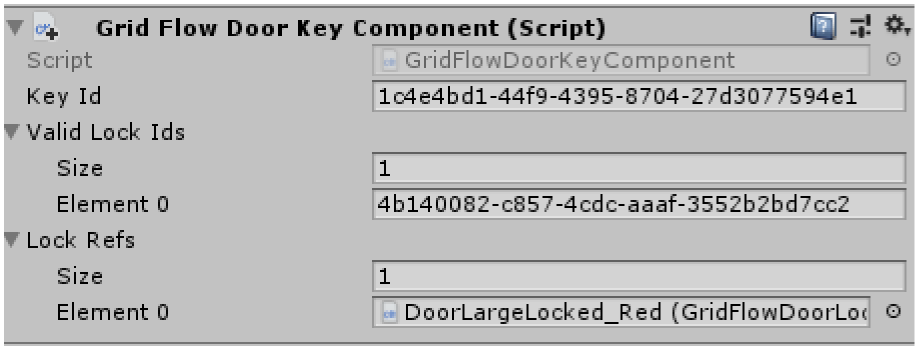


Figure 28: New Components attached to the Locked Door Prefab

Key Component

The builder will attach a new component `GridFlowDoorKeyComponent` to the spawned key prefab



This component contains the KeyId and a reference to all the locks that this key can open

Parameter | Value **Key Id** | The Key Id **Valid Lock Ids** | List of Lock Ids that can be opened by this key **Lock Refs** | References to the spawned lock game objects that can be opened by this key

Lock Component

The builder will attach a new component `GridFlowDoorLockComponent` to the spawned lock prefab

This component contains the LockId and a reference to all the keys that open this lock

Parameter | Value **Lock Id** | The Lock Id **Valid Key Ids** | List of Key Ids that open this lock **Valid Key Refs** | References to the spawned key game objects that open this lock

Sample

Game Sample Scene: `Assets/DungeonArchitect_Samples/DemoBuilder_GridFlow/Scenes/Gr`

The GridFlow game sample contains a working example of how you can implement a key lock system. There are many ways of implementing this, this sample shows one such way.

The Sample has the following scripts:

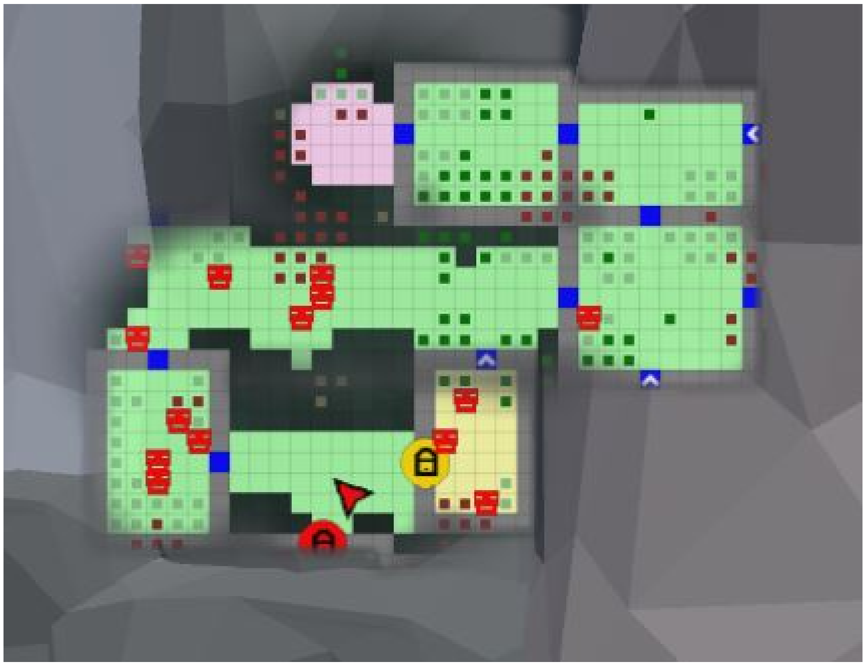
- Inventory: Saves the picked up keys in the inventory
- LockedDoor: A script that implements the door opening logic. This script is added to the locked door prefab. When something collides with the door trigger, it checks if it has an inventory. If it does, it checks if the inventory contains any of the valid keys that can open this door

LockedDoor script location: `Assets/DungeonArchitect_Samples/DemoBuilder_GridFlow/Scri`

```
bool CanOpenDoor(Collider other)
{
    var inventory = other.gameObject.GetComponentInChildren<Inventory>();
    if (inventory != null)
    {
        // Check if any of the valid keys are present in the inventory of th
        foreach (var validKey in validKeys)
        {
            if (inventory.ContainsItem(validKey))
            {
                return true;
            }
        }
    }
    return false;
}
```

Mini-Map

Display a 2D minimap with fog of war



The `DungeonGridFlow` prefab already comes pre-configured with the minimap. This is done with the `GridFlowMinimap` component:

Parameter | Description **Update Frequency** | Control the frequency of minimap updates. The updates can run at a lower fps for better performance **Enable Fog of War** | Hides parts of the map that is not explored yet **See Through Walls** | If this is disabled, unexplored area behind a wall will not be made visible. This works if Fog of War is enabled **Minimap Texture** | The Render Target texture that the minimap will be rendered on **Icons** | The icons to overlay on special tiles

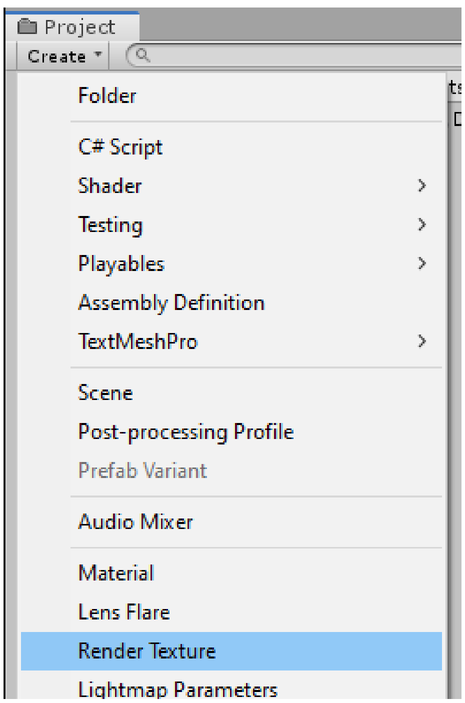
Init Mode values:

Parameter | Description **On Dungeon Rebuild** | The minimap layout texture is regenerated when the dungeon rebuilds **On Play** | The minimap layout texture is generated when you start play **Manual** | The minimap layout texture is generated only when you manually build it from script

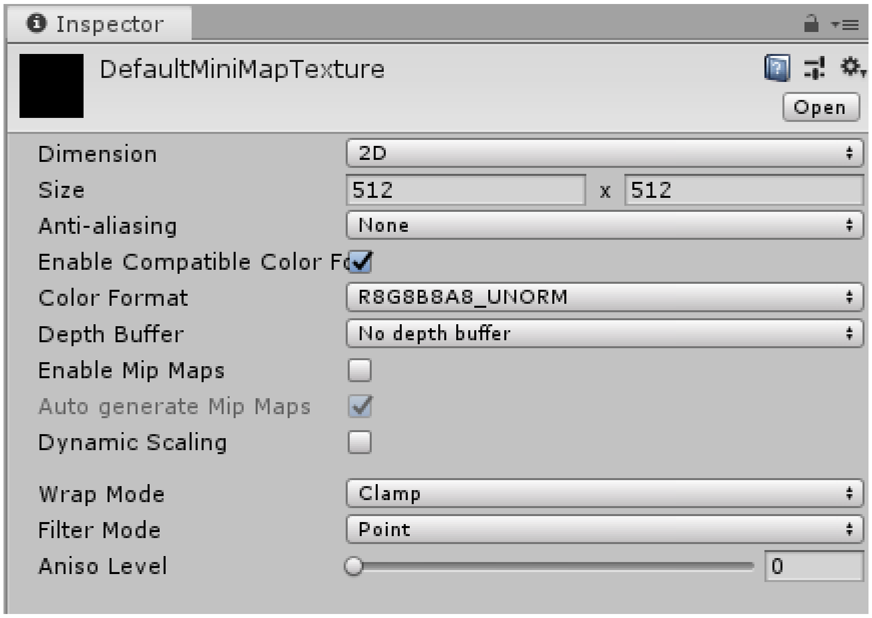
Setup

The minimap requires you to provide a Render Texture asset in the `Minimap Texture` property. The minimap will be rendered in this texture. You can then apply this texture anywhere (in your UI elements, in a mesh etc)

Create a new Render Texture asset. Use the Create menu in the Project window: `Create > Render Texture`



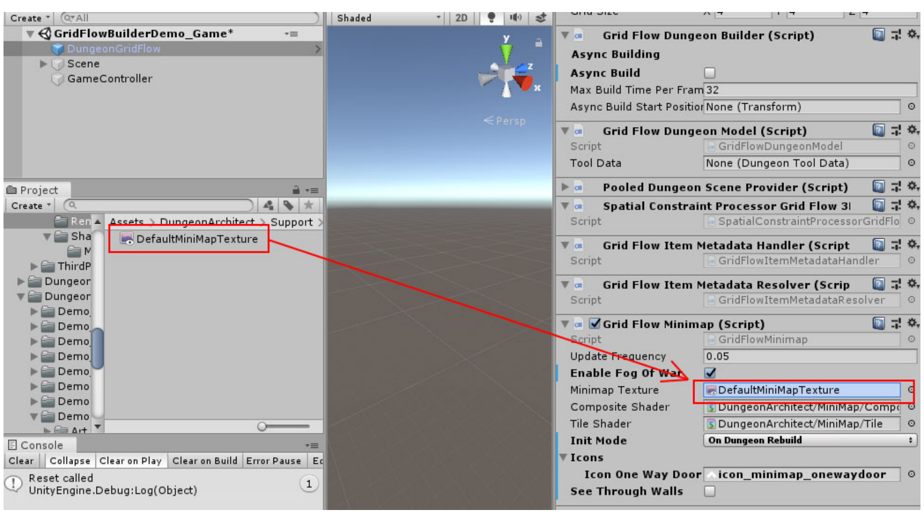
Select the Render Texture asset and inspect the properties



Change the following:

Parameter | Description **Size** | Change to 512x512 (or the quality you are comfortable with) **Depth Buffer** | No Depth buffer (we don't need it here) **Filter Mode** | Point (so we get sharp tile edges instead of a blurry image)

Assign this `Render Texture` asset to your DungeonGridFlow game object's minimap component

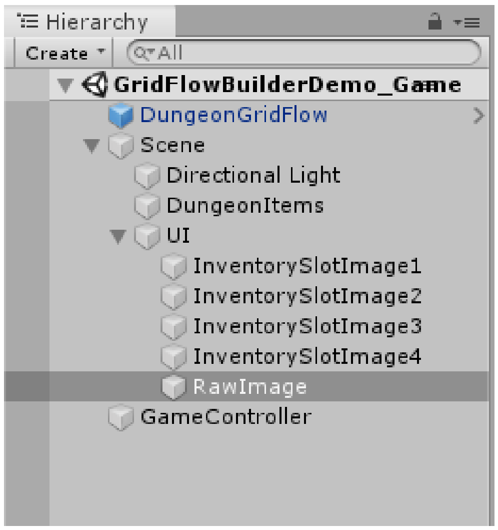


Dungeon Architect will automatically update this texture based on the specified `Update Frequency`. You can assign this texture anywhere on your UI. You can also attach it on a mesh

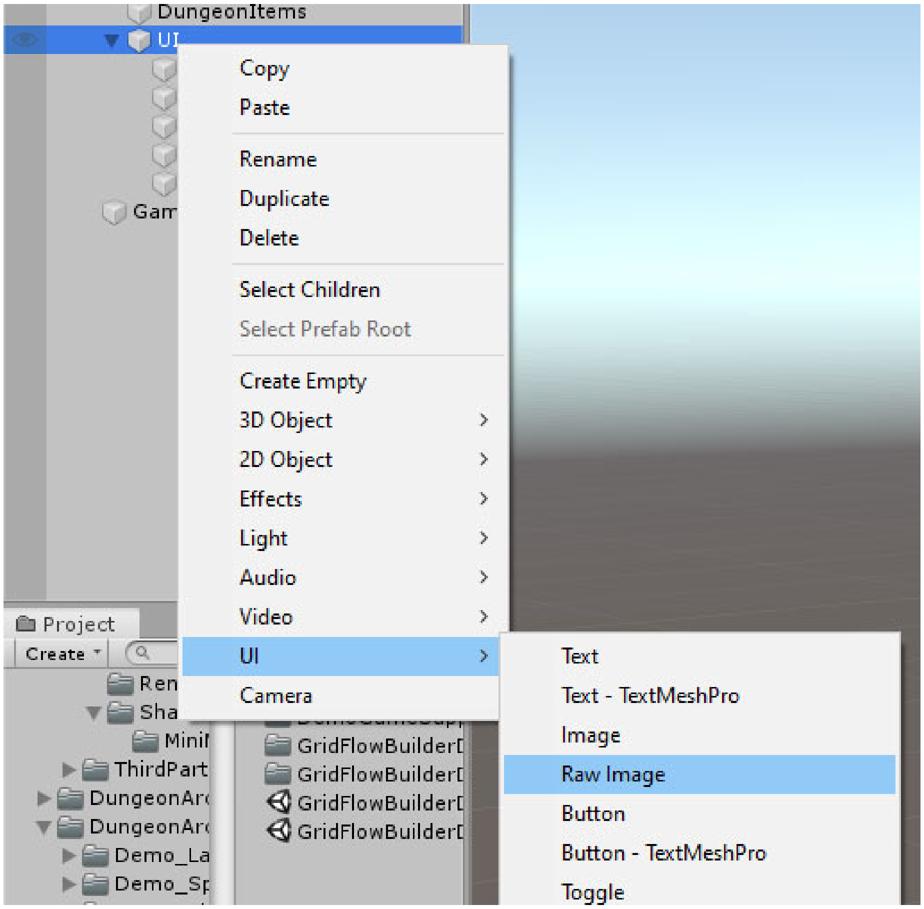
Show in UI

Open the game sample scene: `Assets/DungeonArchitect_Samples/DemoBuilder_GridFlow/Sc`

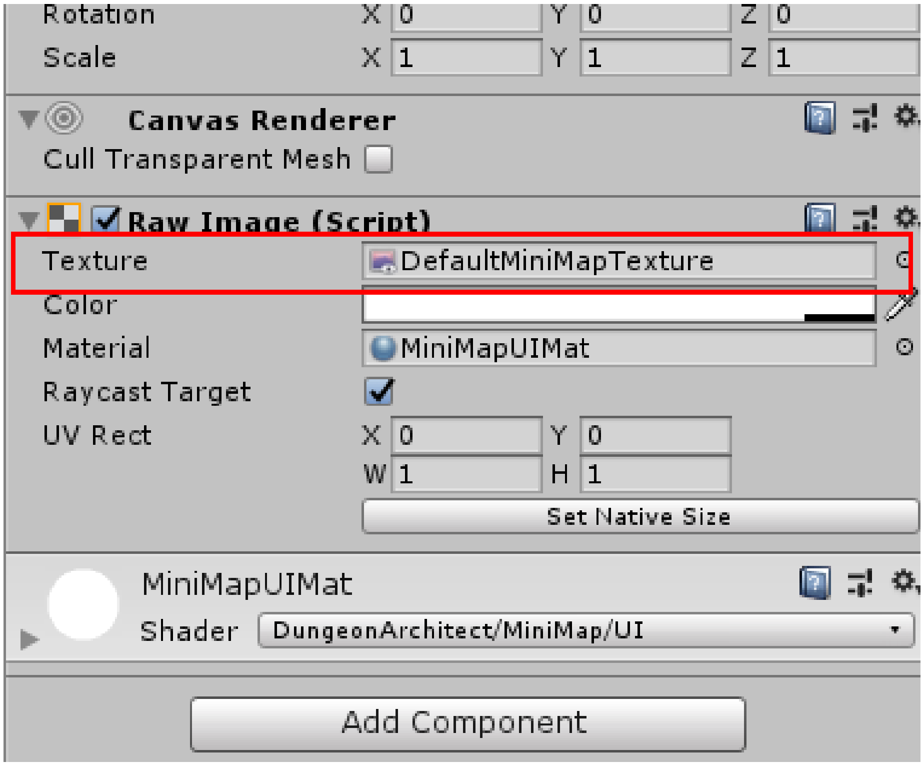
There's a UI canvas in the hierarchy. Expand and inspect it:

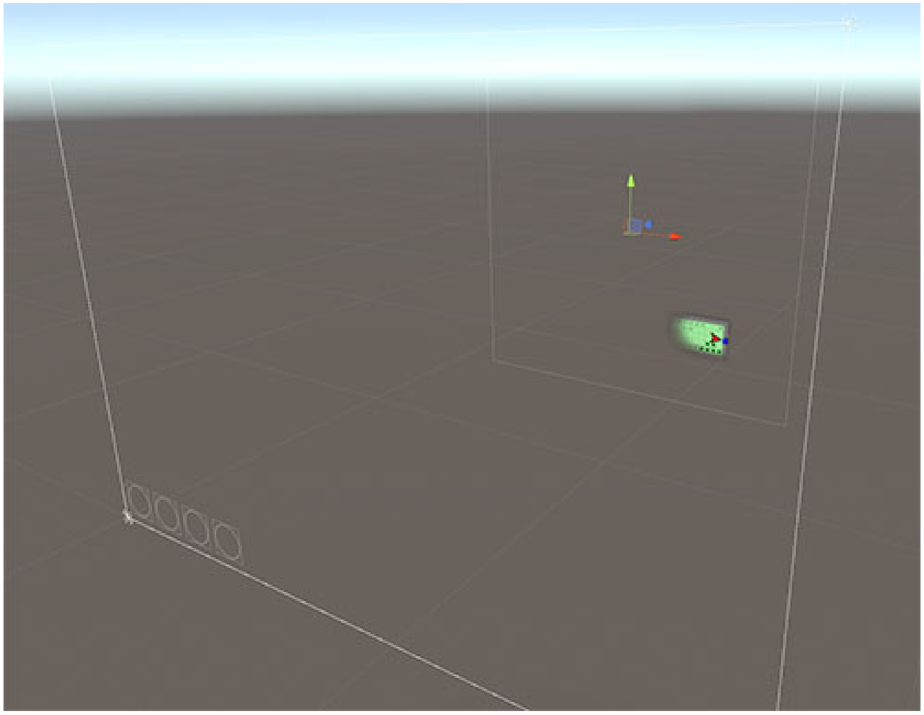


There is a `RawImage` Canvas Item in there. It was created like this:



Select the RawImage item and configure it like this:


Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Canvas Renderer
Cull Transparent Mesh
Raw Image (Script)
Texture DefaultMiniMapTexture
Color
Material MiniMapUIMat
Raycast Target
UV Rect X 0 Y 0
W 1 H 1
Set Native Size
MiniMapUIMat
Shader DungeonArchitect/MiniMap/UI
Add Component



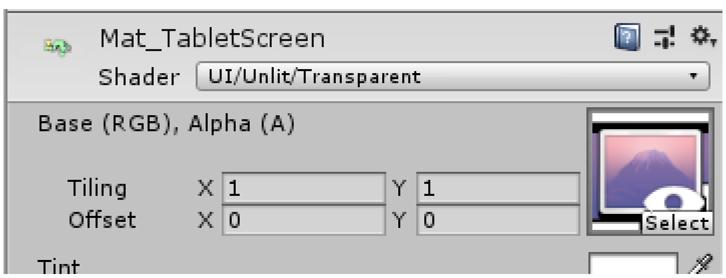
The Render Texture was assigned there so it will show our minimap

Add to a Material

While playing the sample game, if you look down, you notice the player holding a map in the hand (like in Minecraft). This map shows the minimap in realtime

The texture was simply added to an unlit material, and the material was then applied to that mesh

Create a Material as below:



- Set the Shader to `UI/Unlit/Transparent`
- Set the texture to your Render Texture asset

You can now apply this material anywhere (e.g. in a large billboard in your world, a small map that the player holds, dashboard of a vehicle etc)

:::note See Also Check the sample game to see how this was done

Asset | Path **Tablet Prefab** | *Assets/DungeonArchitect_Samples/DemoBuilder_GridFlow/Art/Prefa* **Material** | *Assets/DungeonArchitect_Samples/DemoBuilder_GridFlow/Art/Materials/Mat_TabletS*
:::

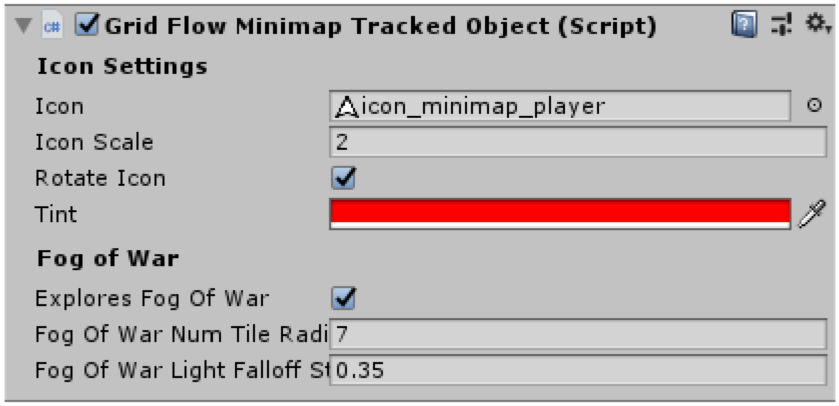


Figure 29: Added to the Player prefab

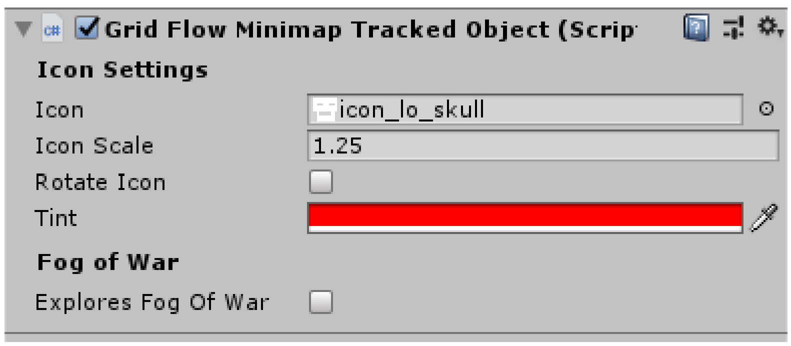


Figure 30: Added to the Enemy prefab

Minimap Tracked Objects

The minimap can track any object in the scene. You do this by adding the `GridFlowMinimapTrackedObject` component to the desired prefab

It has the following features:

- The tracked object can explore the minimap (e.g. player and allies)
- Specify an icon, color and scale of the object in the minimap
- the icon can rotate to indicate the game object's Y rotation (good for player game objects)

You'd want to turn on `Explores Fog of War` only for the player and other relavant objects. The icon can be greyscale and you can apply a tint on it with different colors (e.g. on key icon but different colors applied to the red key prefab, blue key prefab and so on)

:::note See Also Check the sample game prefabs to see how the component was configured

Asset | Path Player Controller | `Assets/DungeonArchitect_Samples/DemoBuilder_GridFlow/Sc`
Grund NPC | `Assets/DungeonArchitect_Samples/DemoBuilder_GridFlow/Scenes/DemoGameS`
Key (Red) | `Assets/DungeonArchitect_Samples/DemoBuilder_GridFlow/Art/Prefab/KeySkul`
Key (Blue) | `Assets/DungeonArchitect_Samples/DemoBuilder_GridFlow/Art/Prefab/KeySku`
Door (Yellow) | `Assets/DungeonArchitect_Samples/DemoBuilder_GridFlow/Art/Prefab/Door`
Door (Green) | `Assets/DungeonArchitect_Samples/DemoBuilder_GridFlow/Art/Prefab/Door`
::: # Snap Flow Builder

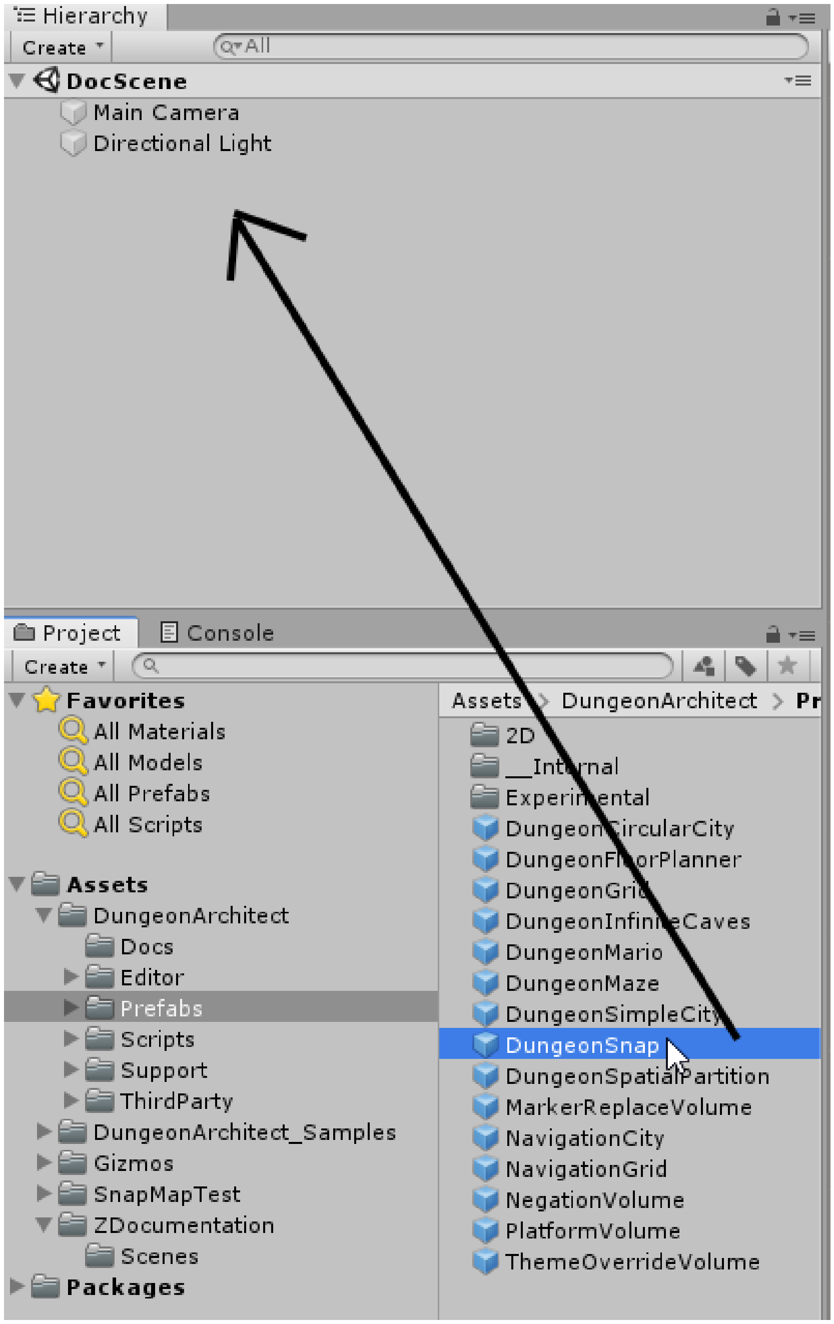
Setup Snap Dungeon

The Snap Builder generates a dungeon by stitching together pre-built rooms prefabs. The rules for stitching them is controlled by Graph Grammars

In this page, we'll walk through the creation process.

Preparing the Scene

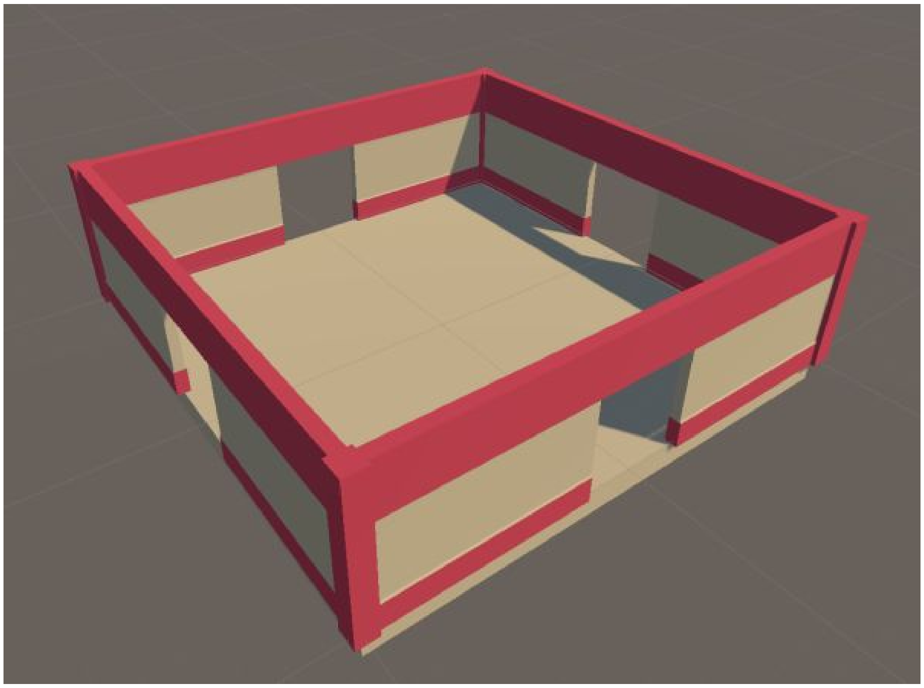
Create a new scene and drop in a Dungeon Snap game object. This will allow you to build snap dungeons



Creating Module Prefabs

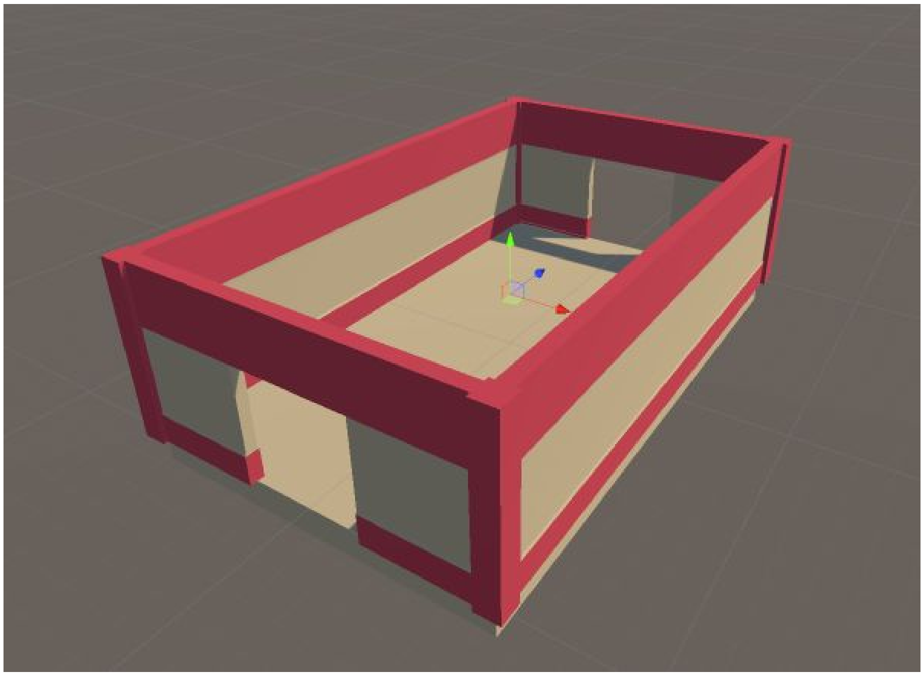
A module is a prebuilt prefab of a area (like room, corridor etc)

Design a room in the editor so we can turn it into a prefab for use with the snap builder



We've left holes at places where we want a possible door. We'll place a special Connection prefab later at these places to let Dungeon Architect know that it can stitch the rooms from these points

Go ahead and create a few more module prefabs. We've created one for a corridor below



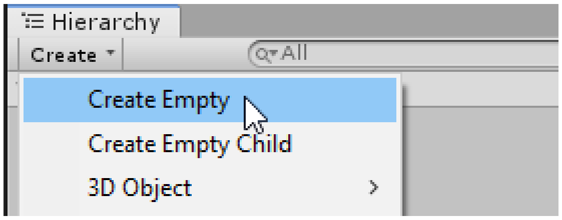
Connections

A Snap Connection tells DA how to stitch the room modules together. They are usually the Door Entry / Exits.

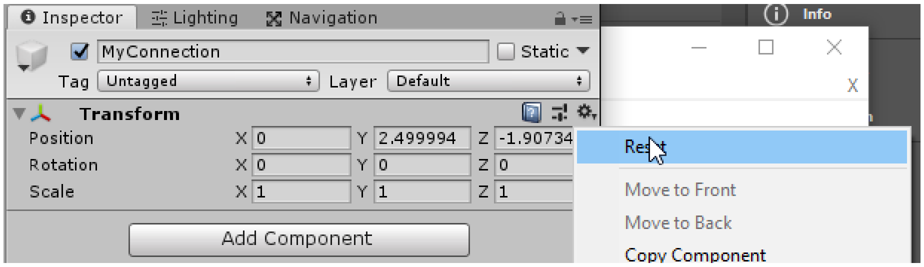
The connection also contains references to two assets, a references to a Door prefab and a reference to a Wall prefab.

If DA stitches another module through that connection point, it would place the specified Door prefab in that place. Otherwise it would fill up the gap with the specified Wall prefab

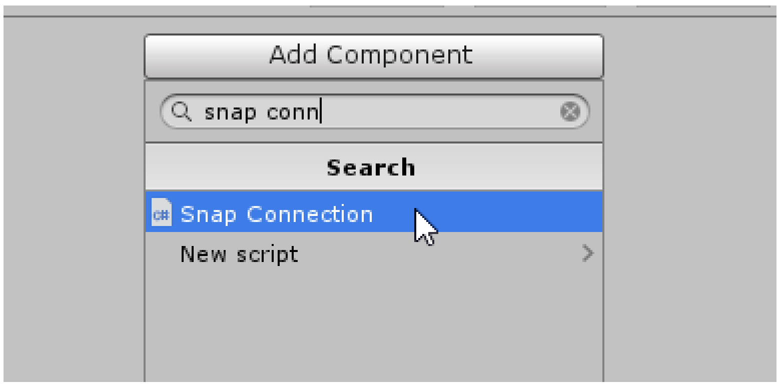
Design a new Connection prefab by creating an emtpy Game Object



Reset the transform of the newly created emtpy Game Object and rename it

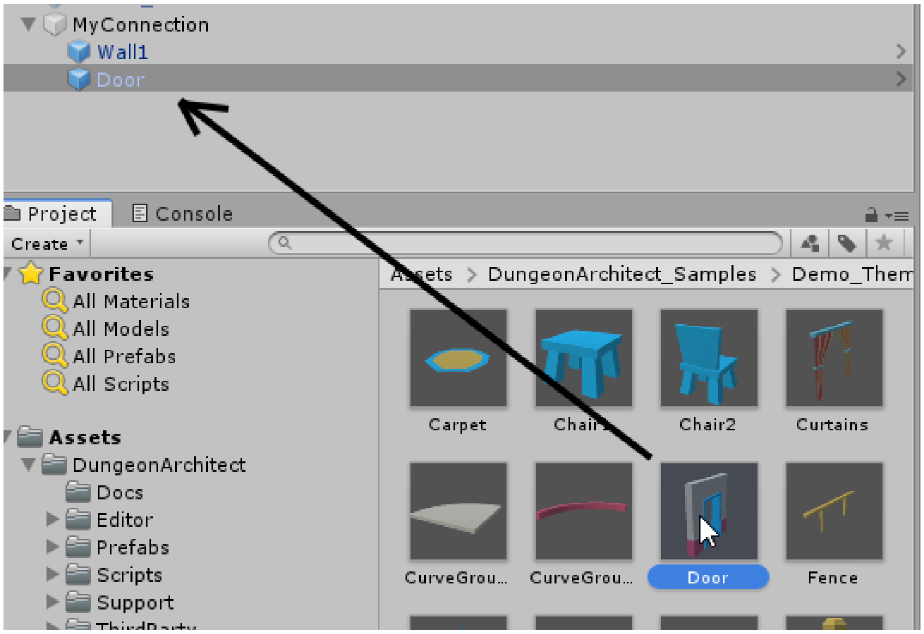


Add a SnapConnection script to it

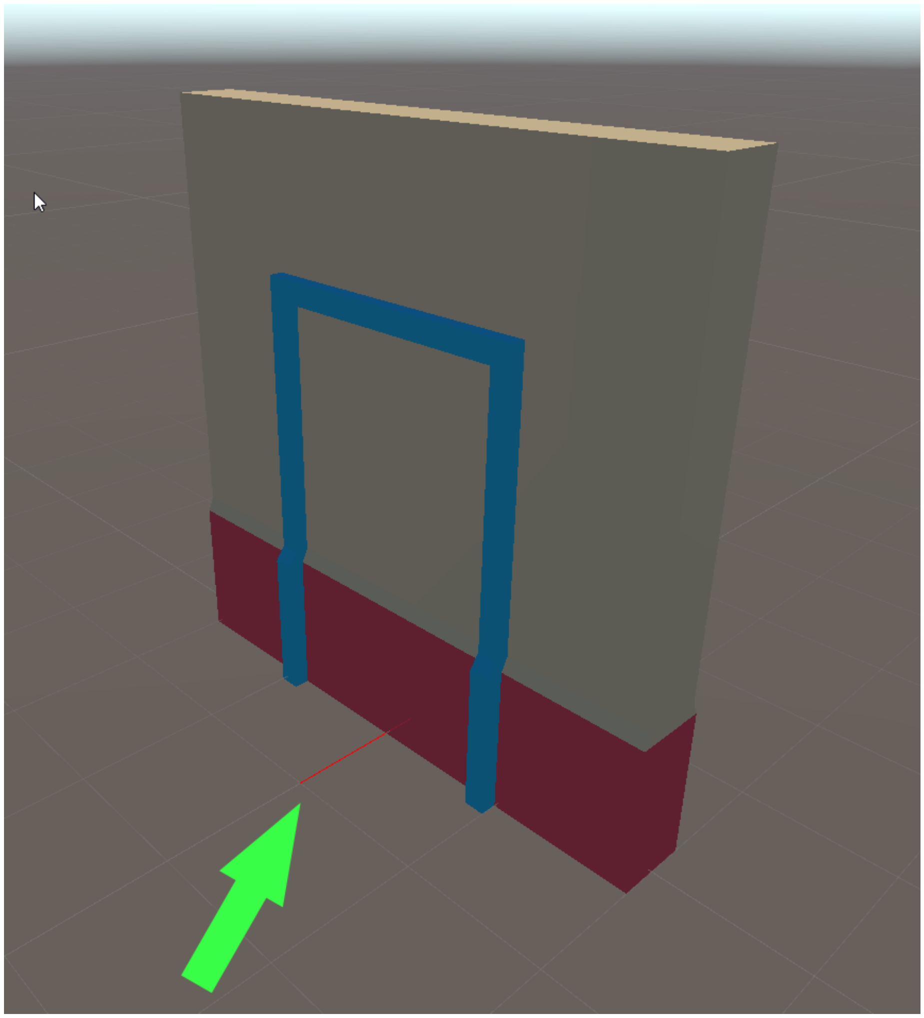


This script takes references to the above two mentioned prefab references, one for door and another for wall

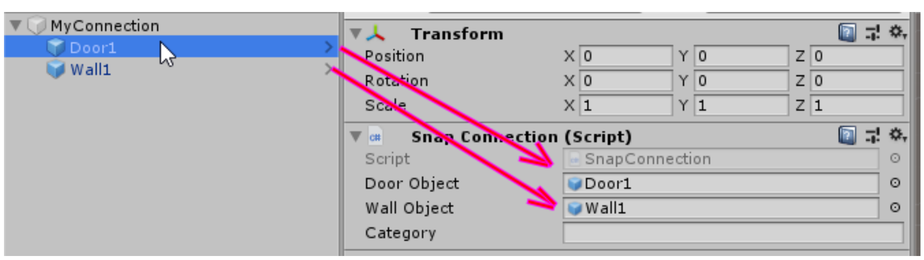
Drop your Door and Wall prefabs under the Connection prefab



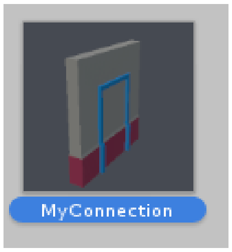
Align the door and wall prefabs so the Red line is perpendicular to it



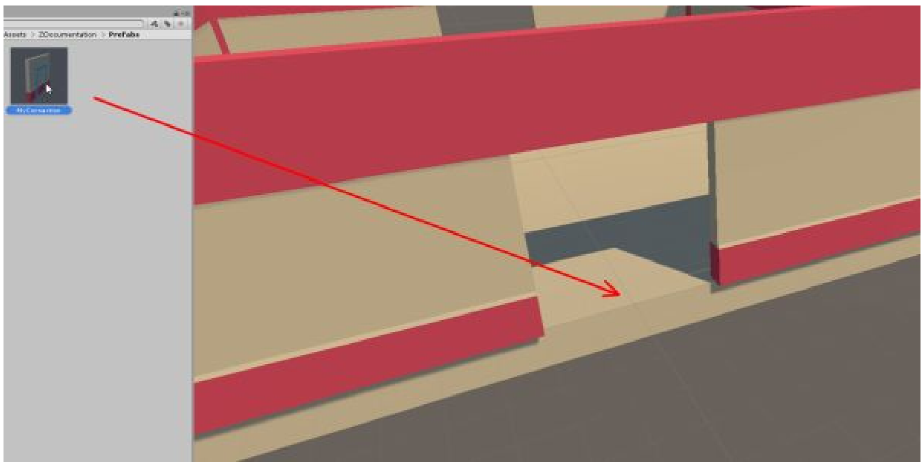
Now set these child door / wall prefab references on the SnapConnection script



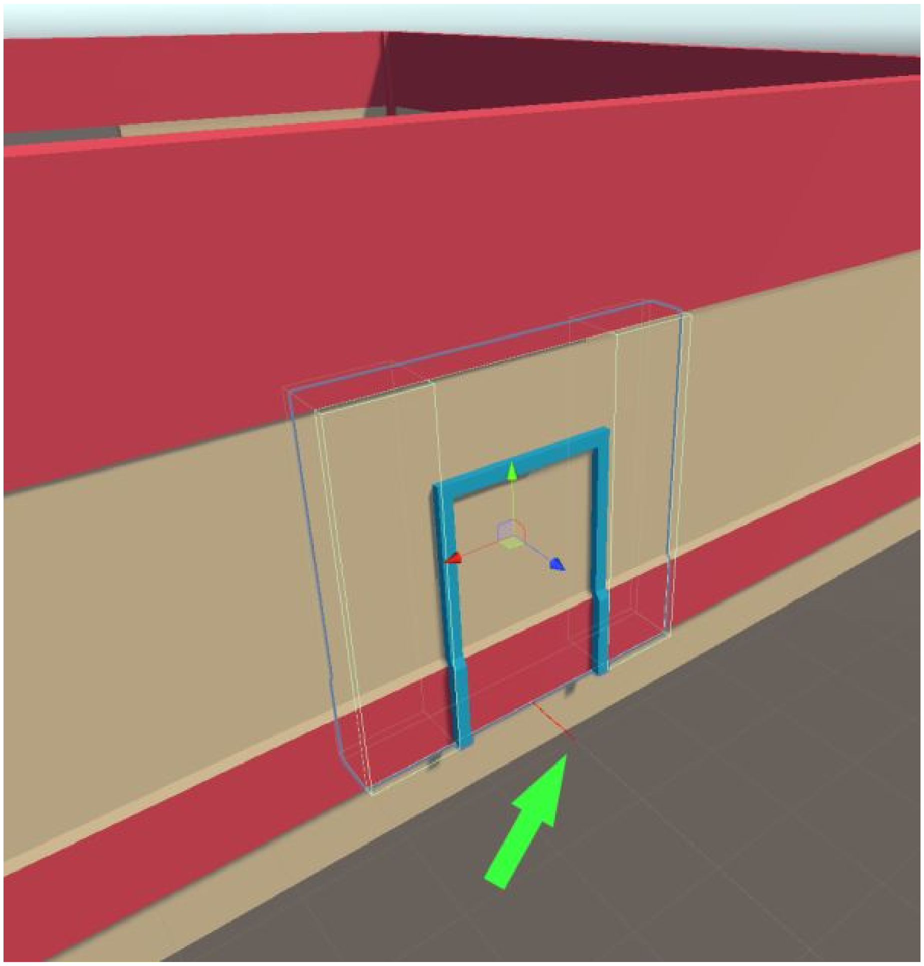
Your connection is ready. Turn this into a prefab so we can reuse this in our modules



Drag and drop the connection prefab on your previously generated modules. Make sure the red line points outwards from the opening

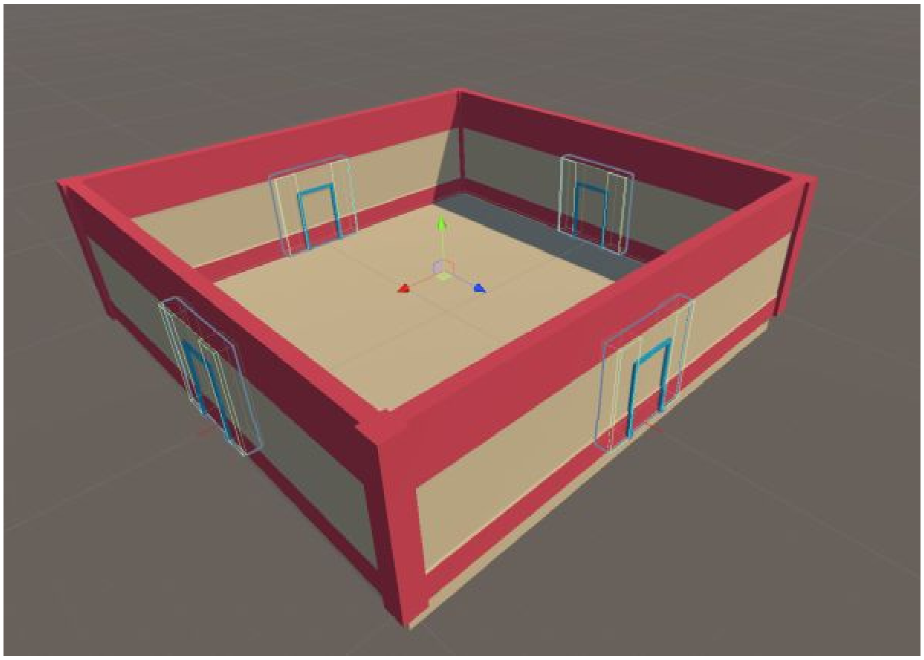


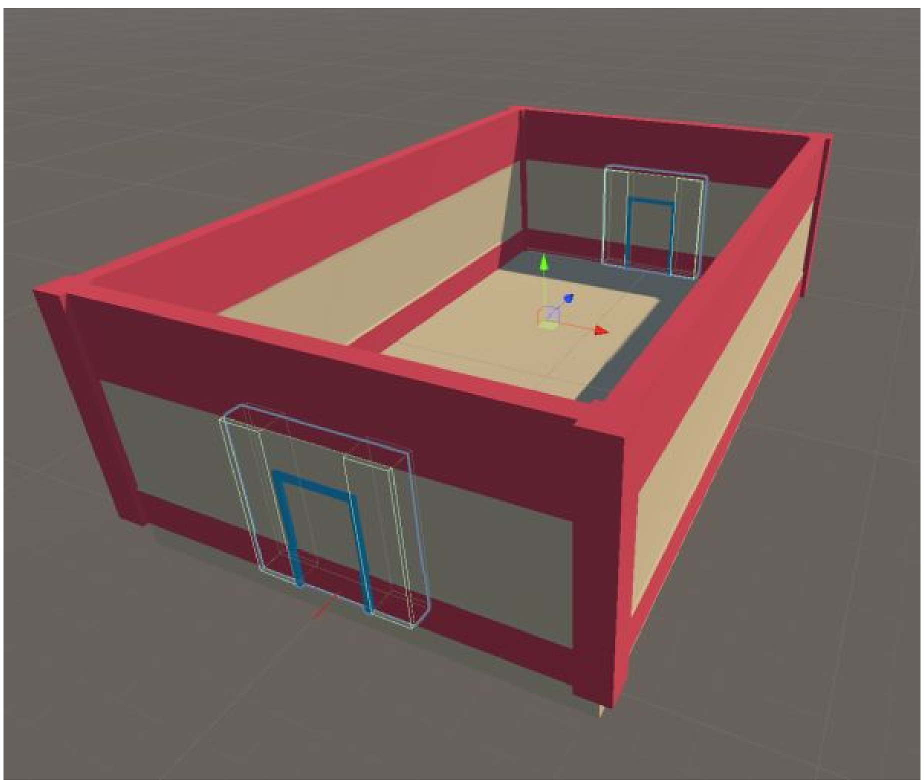
Make sure the red line is pointing outwards and is on the edge of the module bounds



It is a good practice to design with the snap settings (Edit > Snap Settings > Snap All Axis)

Repeat by drag-dropping on all the door openings. Do this for all the other modules as well (like the corridor module)



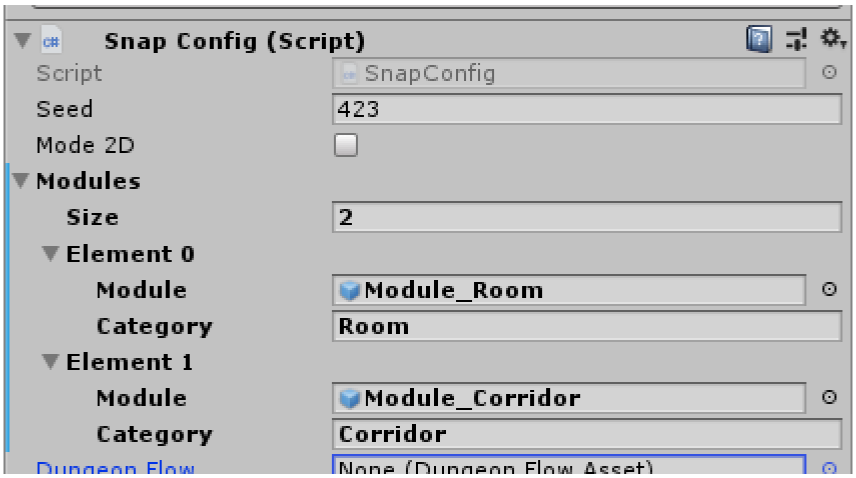


Save/Update your module prefab

Register the Modules

Register your modules in the `DungeonSnap` game object so Dungeon Architect can use it to build the dungeon

Inspect the DungeonSnap Game Object



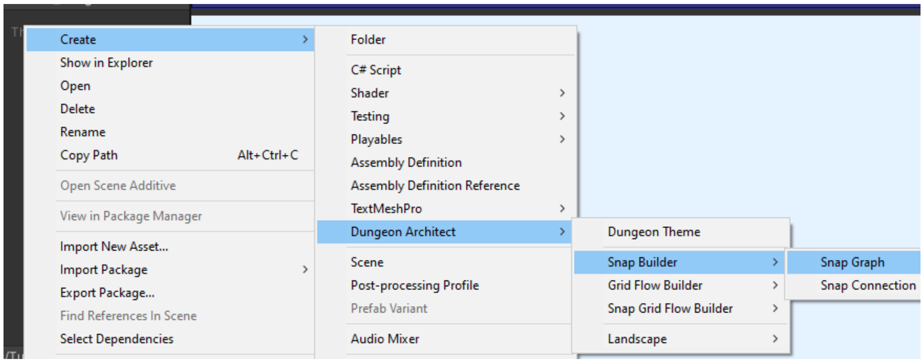
Here we've registered the two modules and assigned a Category to them (e.g. Room, Corridor, TreasureRoom, MiniBoss, MainBoss, SpawnRoom, Exit etc).

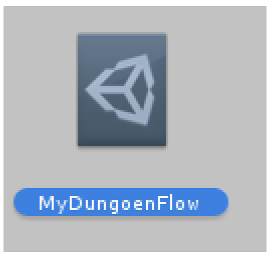
You can have multiple module prefabs assigned to the same category. These categories are used in the Dungeon Flow graph to design a procedural layout graph for your dungeon

Design Snap Flow Graph

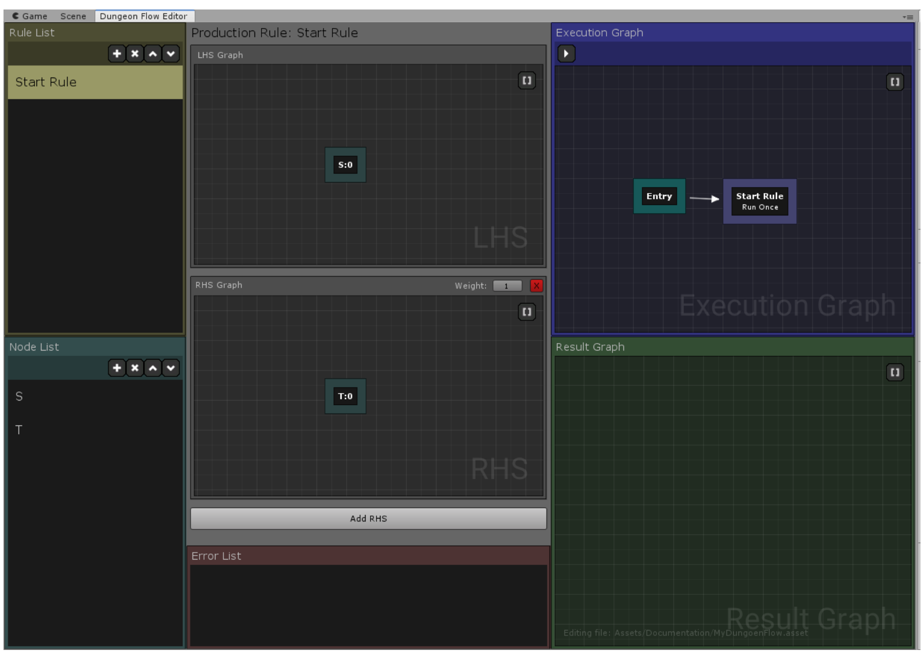
A Dungeon Flow graph allows you to control the layout of your dungeons using Graph Grammars. You can generate interesting graphs with simple rules

Create a new Dungeon Flow Asset by right clicking on the Projects window. (This can also be done from the Create menu in the Projects window or Assets > Create from the editor's main menu)



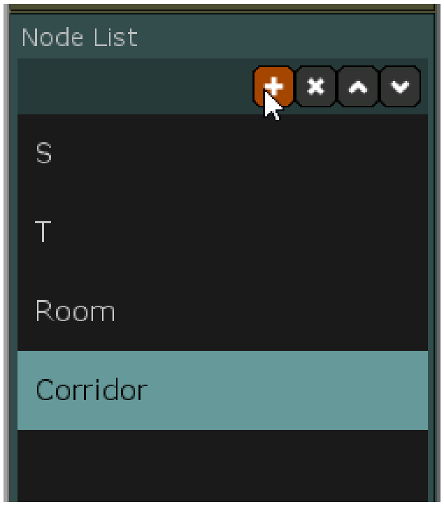


Double click the asset to open the Dungeon Flow Editor. Doc this window in a large area

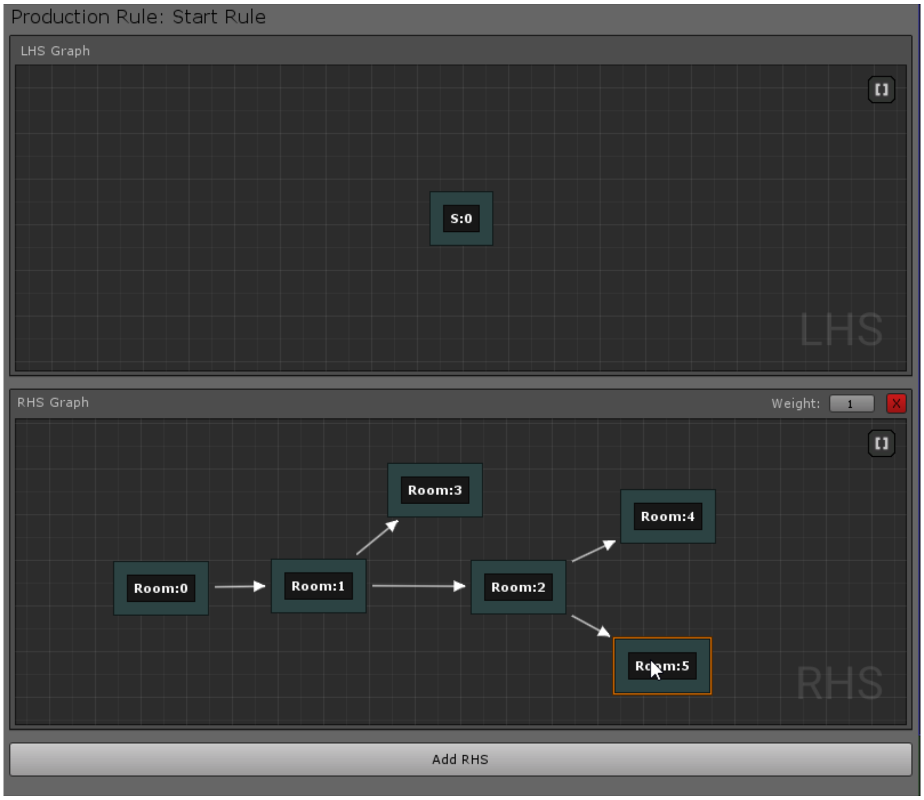


Add two new nodes **Room** and **Corridor**. You can change the name of the nodes from the inspector window

These names map to the names you specified on the Module registration in the DungeonSnap game object

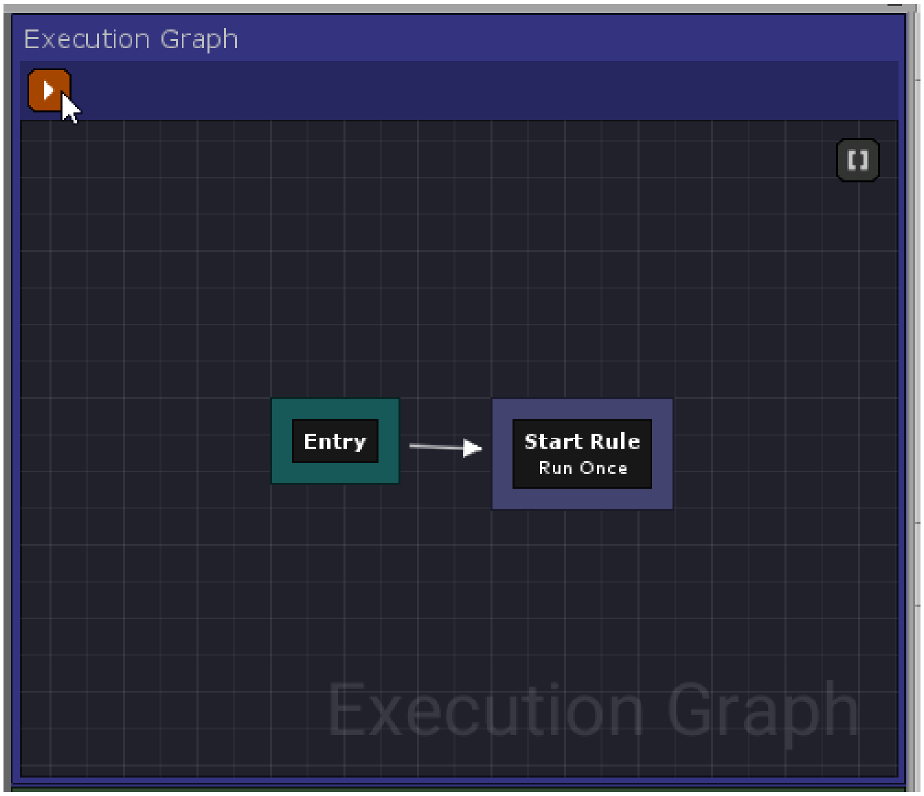


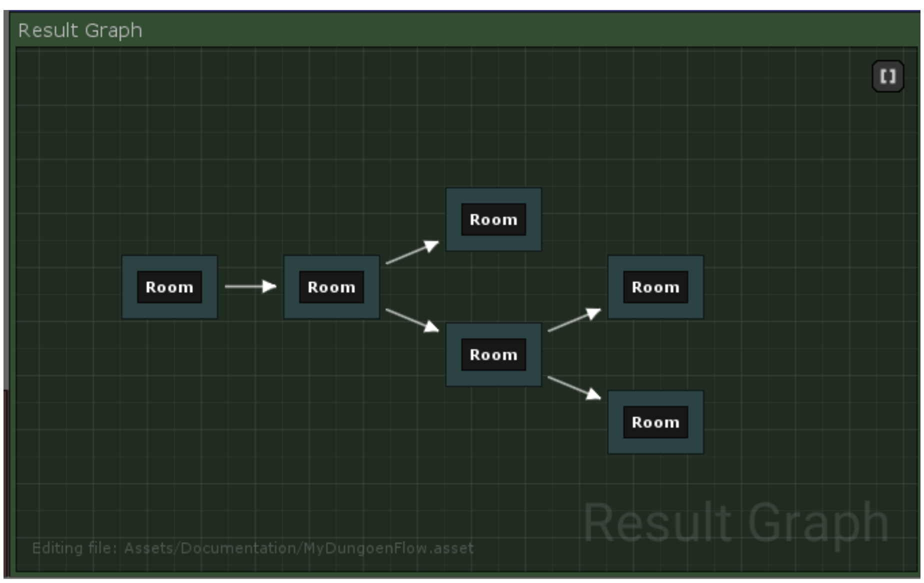
Select the *Start Rule* and on the RHS, delete the default T node and drop in a few Room nodes like this:



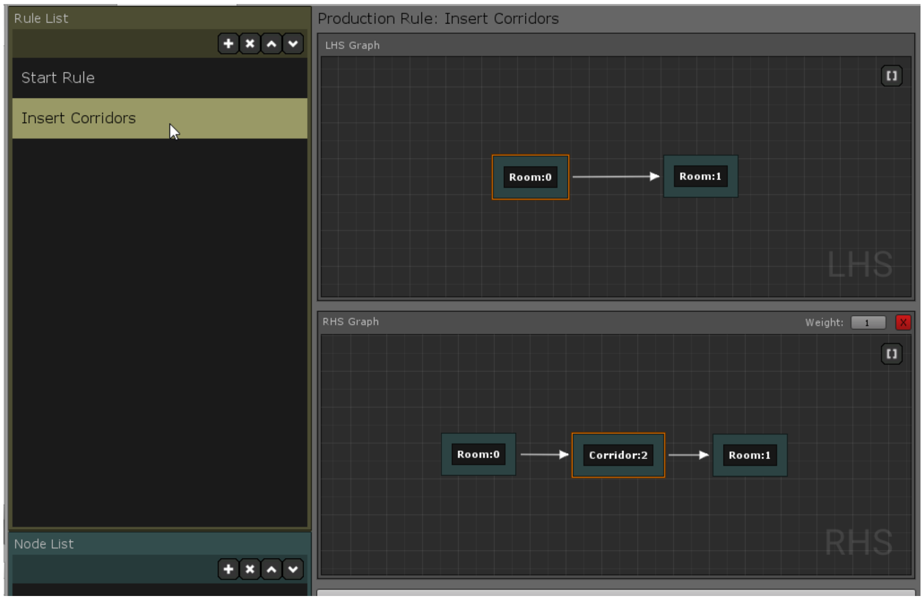
Cycles are not supported by the SnapMap builder

Execute the rule and see how the final graph is generated. You do this by clicking the Run icon on the Execution graph panel





We’d like to insert Corridors between the rooms. Create another rule and give it a name (e.g. *Insert Corridors*)

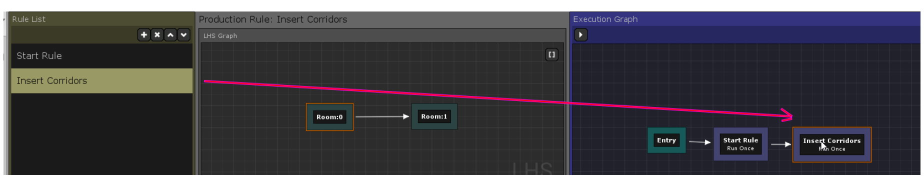


On the LHS, we want to find a patterns where two rooms are connected to each other like this (Room -> Room) and have it replaced with (Room -> Corridor -> Room)

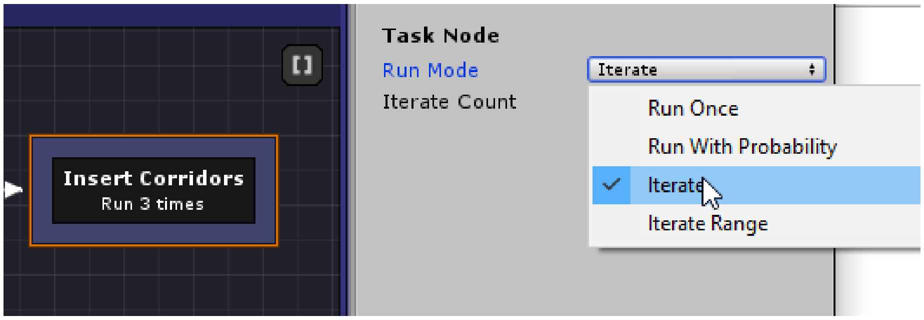
The Graph Grammar will find a pattern you specify on the LHS and replace it with the one you specify on the RHS

The Indices on the nodes (e.g. Room:0, Room:1) are important that helps in correct mapping. Since we properly specified 0 and 1 indices on the RHS, it knows the direction of the newly created links to the corridor. This will be covered in detail in the full documentation soon

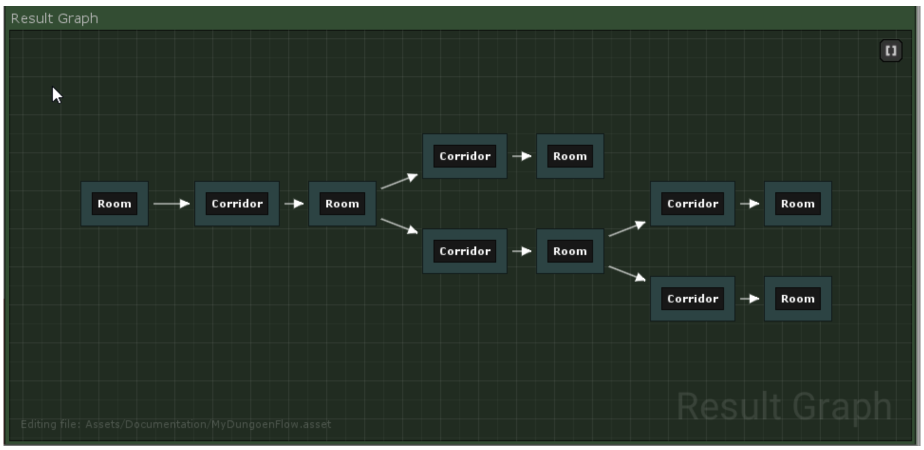
You control how your rules are run from the **Execution Graph**. Drag drop your newly created **Insert Corridor** rule on to the execution graph and connect it after the *Start Rule*.



Select the newly placed node and from the details panel, change the execution mode to Iterate and set the count to 2 or 3 (This makes the rule run multiple times since the newly replaced Room nodes wont map with the adjacent older Room nodes by design and need to be run again)



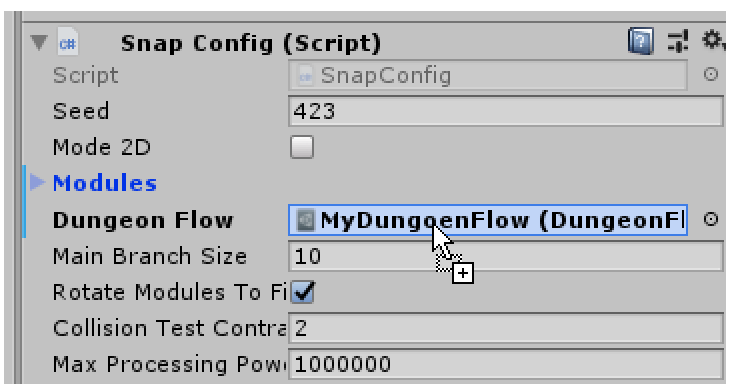
Execute the grammar and you'll now see corridors between your rooms



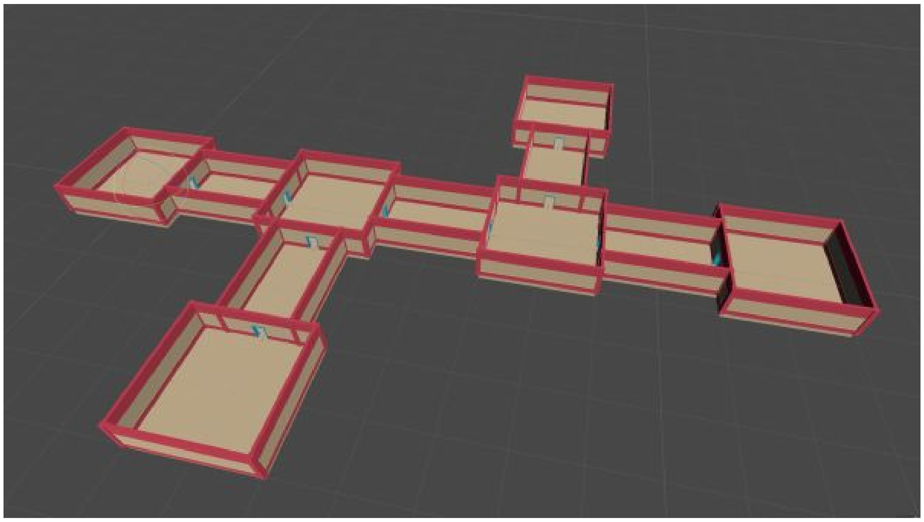
We will use this Dungeon Flow graph grammar to generate our snap dungeons

Build the Dungeon

Assign the `Dungeon Flow` assets to the DungeonSnap game object

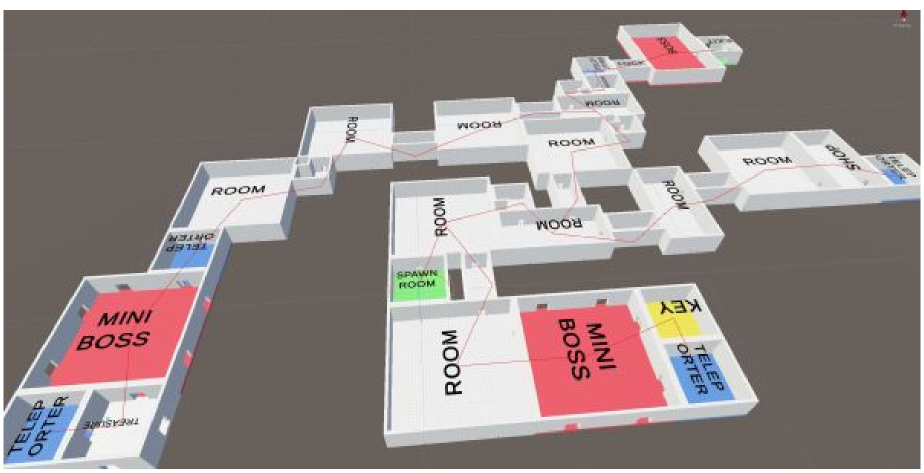


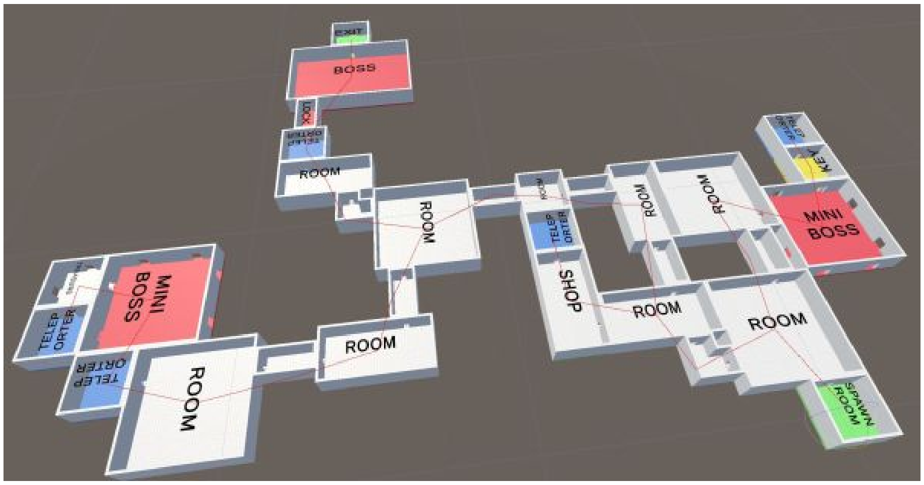
Hit `Build Dungeon`. Randomize the seed and get different configurations that satisfy the layout graph you defined in the flow asset. Change the Dungeon Flow graph and experiment further

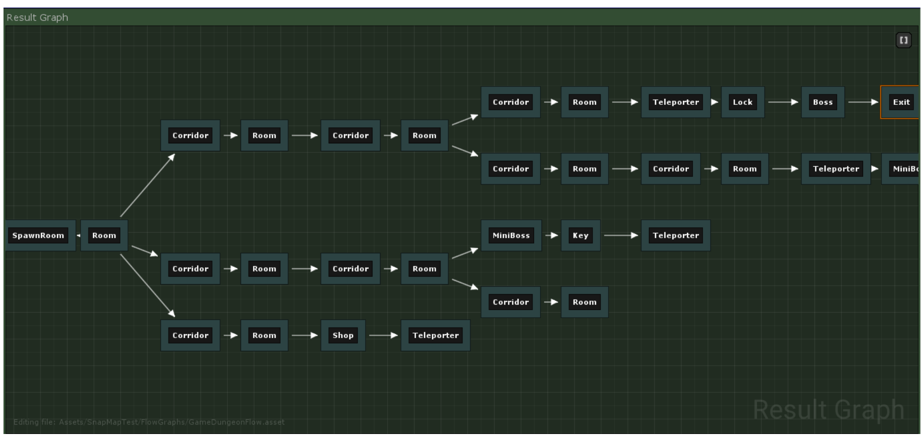


Explore the Sample for a more complex and complete dungeon map with multi key-lock system, treasure rooms guarded by miniboss, exit guarded by a Boss room which requires a key to unlock

Location: `DungeonArchitect_Samples\DemoBuilder_Snap\Scenes\DemoScene`







Snap Grid Flow Builder

Introduction

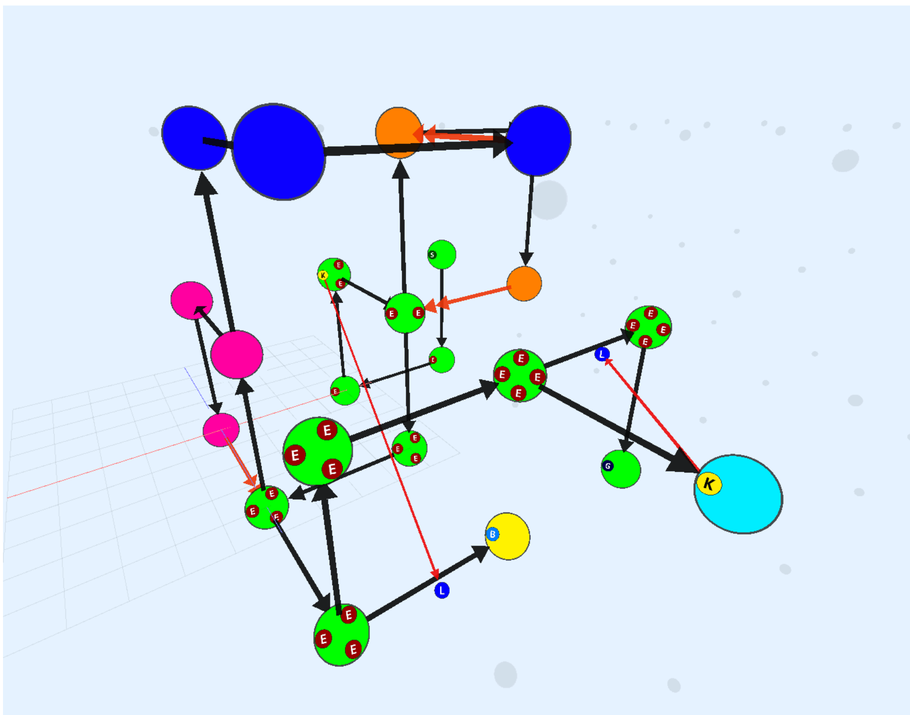
Features

The **Snap Grid Flow Builder** allows you to stitch snap modules using a flow graph. You can create many types of levels with this builder. Some examples include: * FPS/TPS with multiple floors * Side Scrollers * Top-Down dungeon crawlers (like diablo) * Cities (with custom road paths, subways etc) * Race Tracks

Since this builder uses parts of the Snap builder and the Flow framework, you can do the following: * As an artist, have complete control over the design of the individual snap rooms. Use vertex painting, foliage, landscapes, custom gameplay elements etc. on your snap room modules * Use the flow framework to design procedural layouts, leveraging all the existing features like cyclic-paths, key-locks, one-ways doors, item spawners etc * Use Level streaming to keep the framerate high, even with lots of dynamic lights

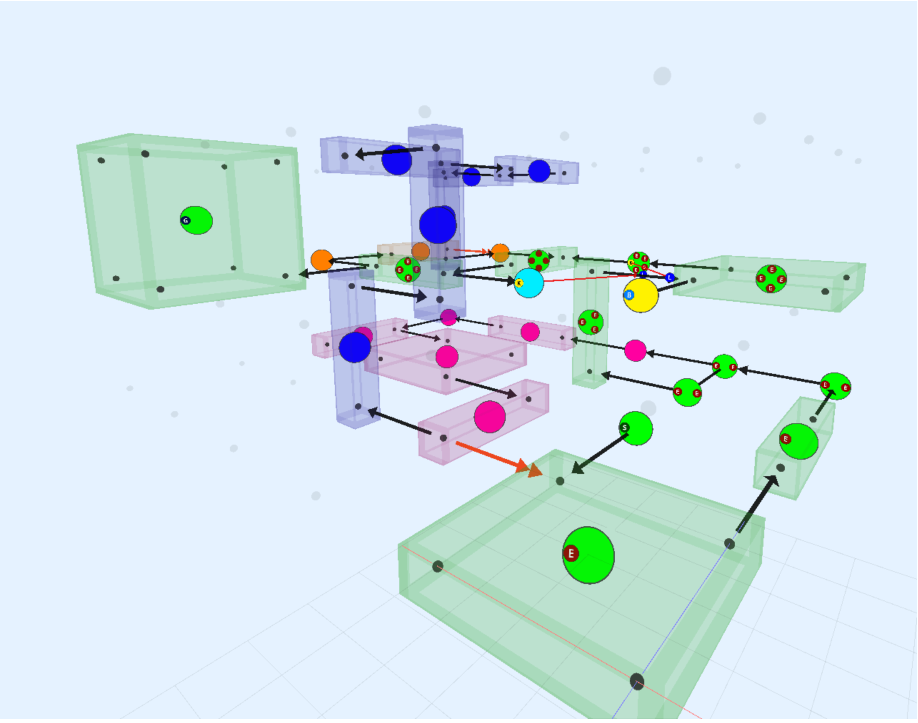
3D Flow Graph

You'll design the flow graph in a 3D Layout Grid

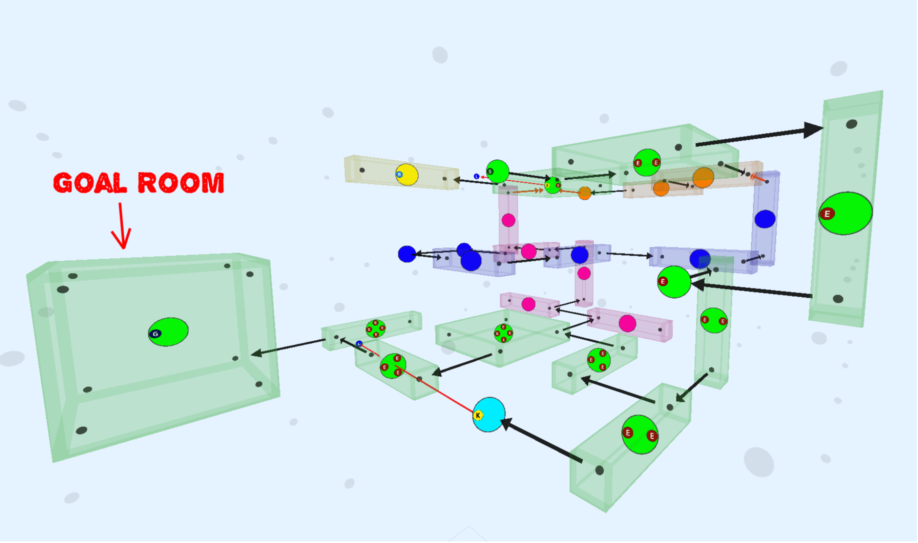


The snap rooms are required to be of a fixed size (that is chosen by you), so they all can fit nicely and can be stitched inside the 3D grid.

However, you may also design your snap rooms to span multiple nodes in the flow graph. The flow framework is smart enough to identify these rooms and use them appropriately in the flow graph:



In this example, the goal room was designed to be 2x2x2 the size of the chunk. The flow framework identified it and created a larger node appropriately while building the flow graph



The flow framework will also read the available doors you've setup in your snap modules and use that configuration to grow the graph. So you are free to leave out the doors that you don't want while designing your snap module

This means, you don't need to leave space for doors on each side of the room. You can safely wall them off with your art assets, and it won't grow from there

Modules

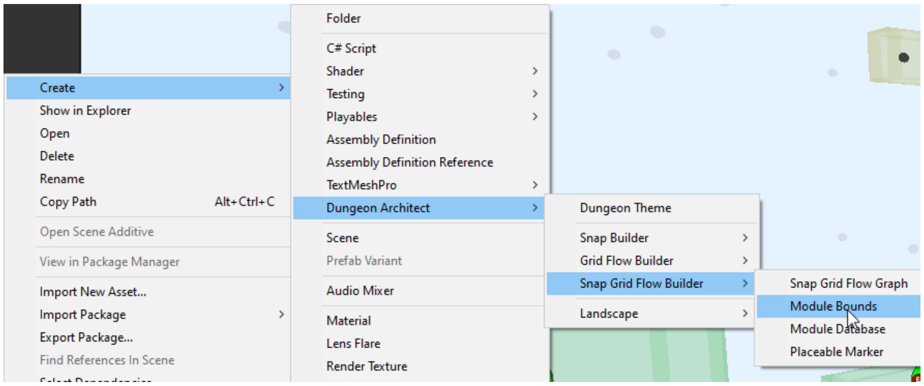
A room is designed in a separate prefab, and it is called a `Module`

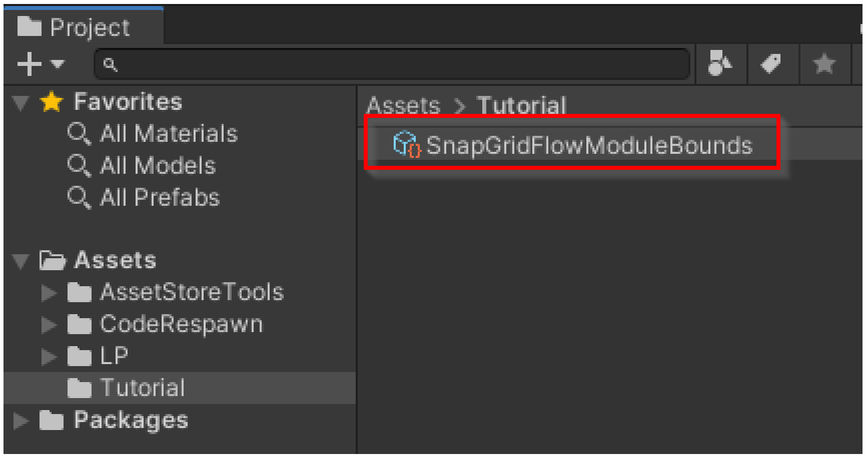
All Modules in the Snap Grid Flow builder have a fixed size called the `Chunk Size`. Your module size can be in mulitples of this size (e.g. you can create a taller room of size `1x2x1` or a bigger room `3x2x3`)

Module Bounds

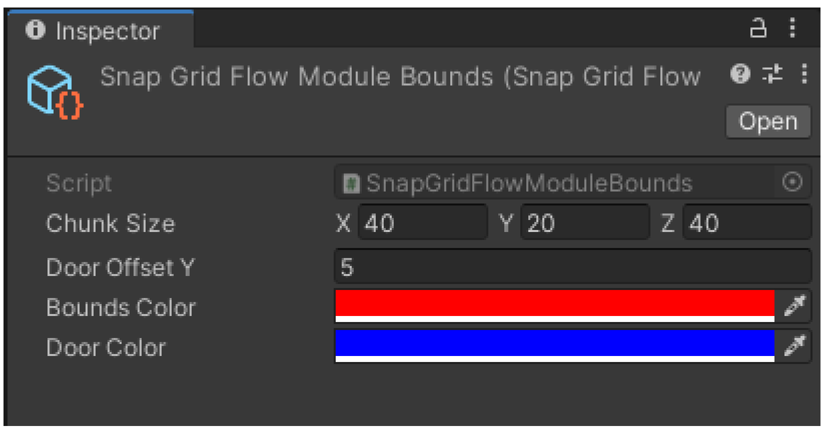
Start by creating a `Snap Grid Flow - Module Bounds` asset. You will assign the size of your modules here and use it while designing the modules

In the Projects tab, navigate to the desired folder and create the `Snap Grid Flow Module Bounds` asset





Double click the asset to open the editor * Set the `Chunk Size` to something like `(40, 20, 40)`. The `X` and `Z` has to be the same if you want your modules to support rotation. In this case, we've provided `40` for both * Set the `Door Offset Y` to `5`.



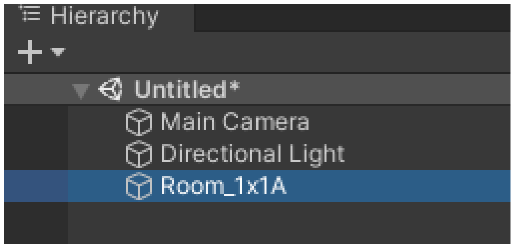
`Door Offset Y` determines how high the door is from the base of the module. It is a good idea to leave some gap, so you have space to create some pits or lower areas

You'll use this asset on each of the module prefabs you design

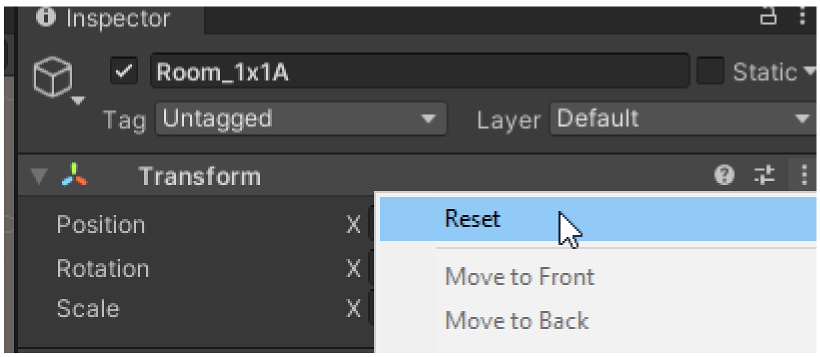
Create a Module

Modules are rooms that are designed and saved into a prefab. We'll design a new room module prefab

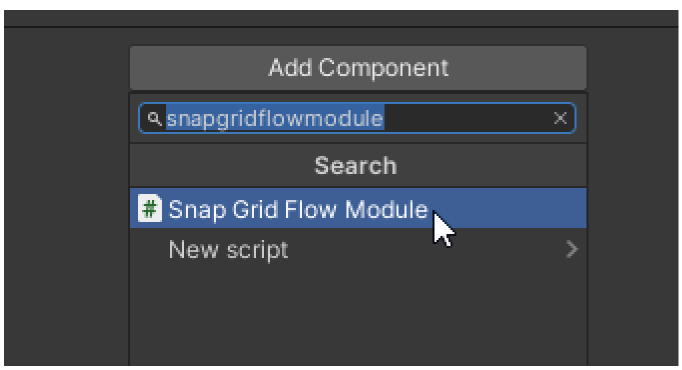
- Create a new scene
- Create an empty game object and name it to `Room_1x1A`

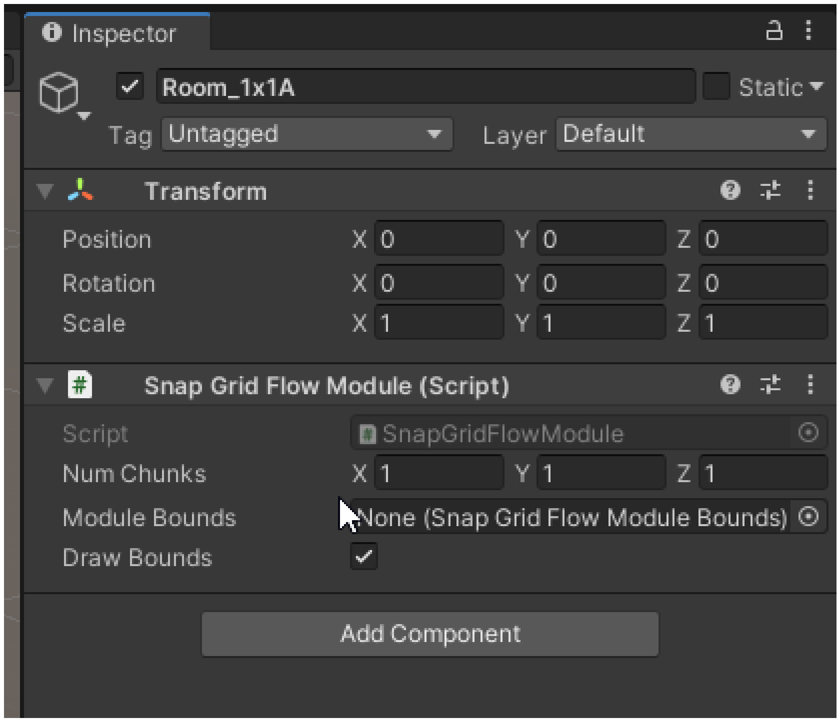


- Reset the prefab transform

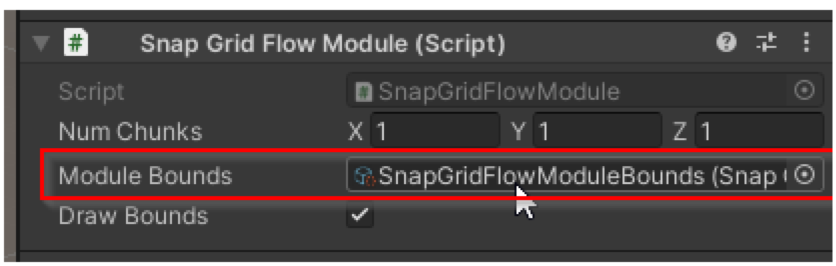


- Add the `Snap Grid Flow Module` component to it

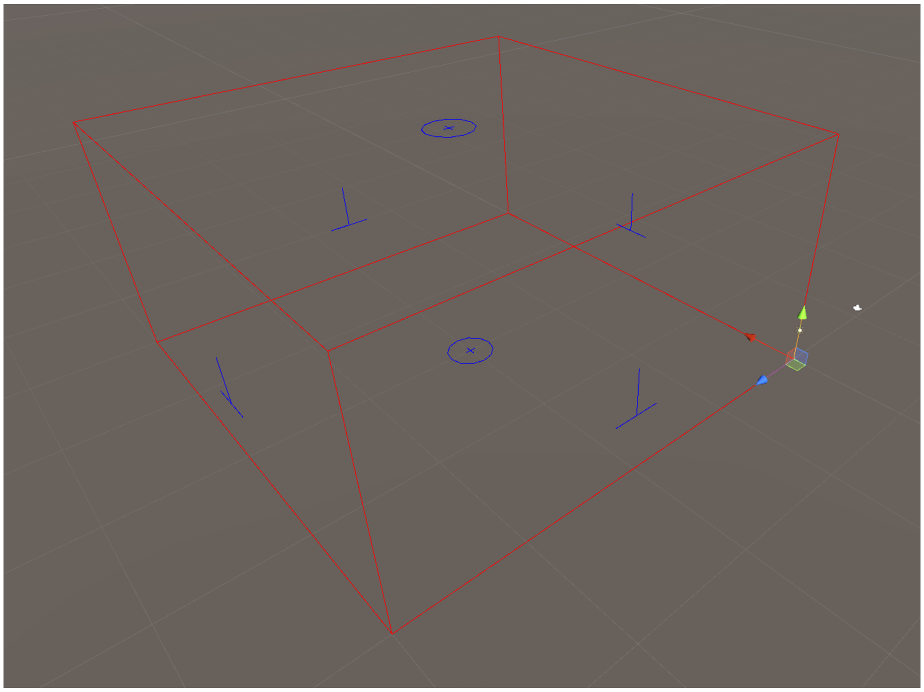




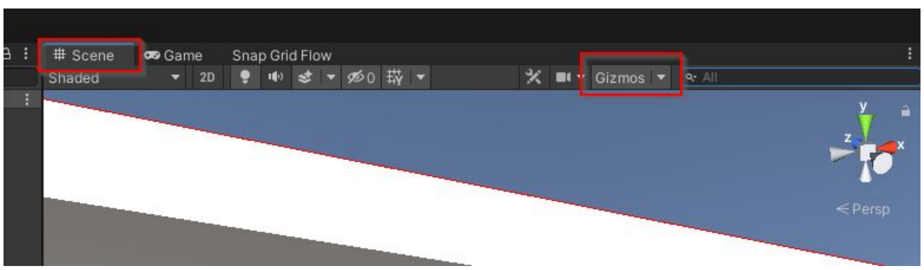
Assign the module bounds asset we created earlier



Your module prefab will now provide visual information that will help you design your room



:::warning Note If you don't see the red lines, make sure the `Gizmos` button is pressed in the Scene View tab's toolbar

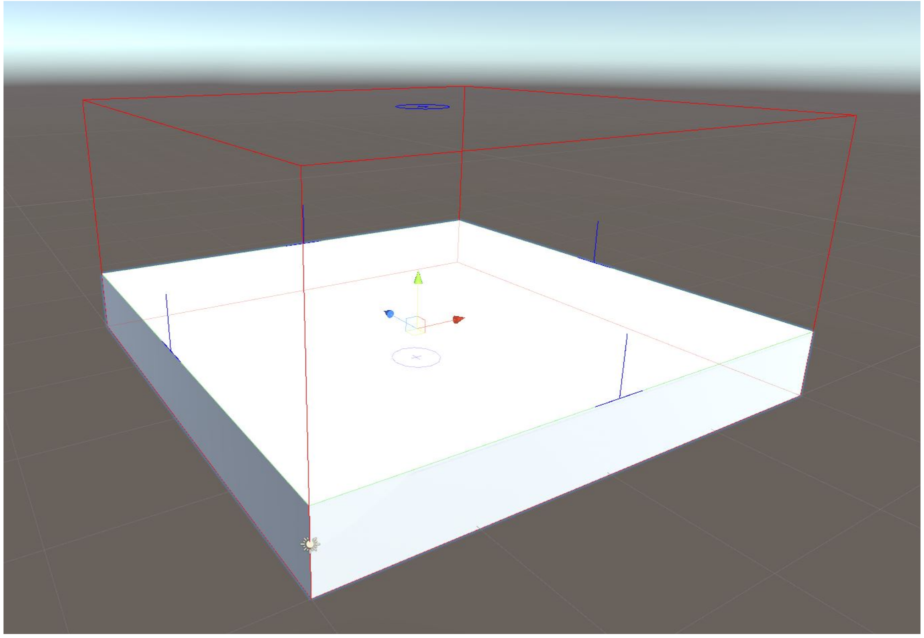


:::

The red wireframe indicates the bounds of your module. You may fill this up in any way you like.

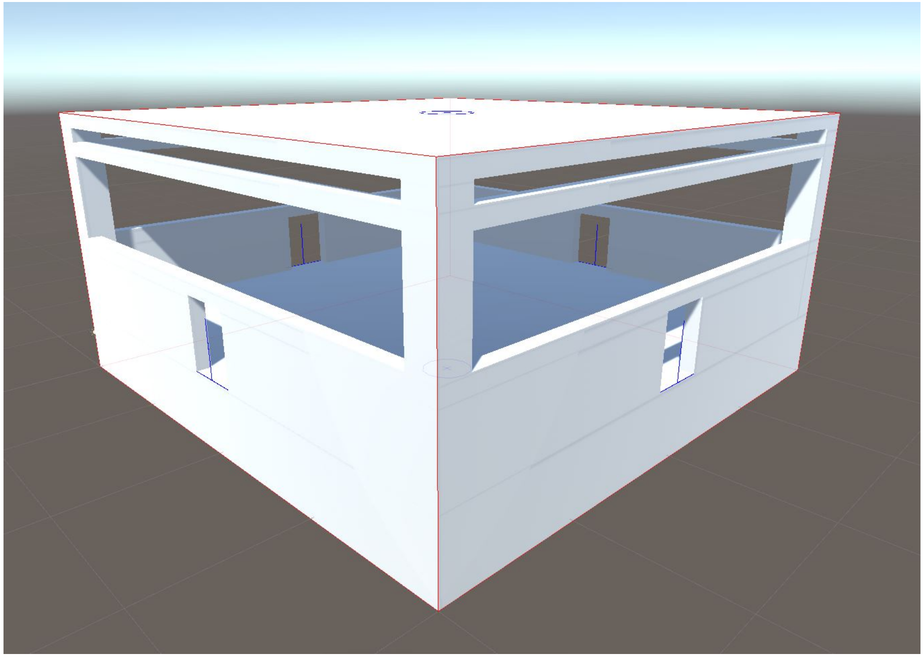
The blue indicators show the possible locations of the doors

Design your module inside the level bounds.

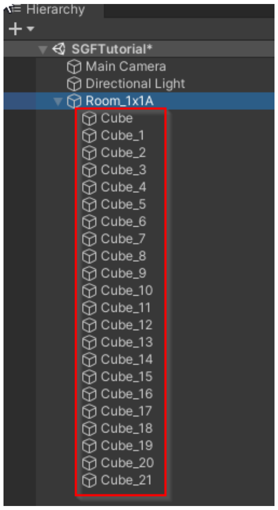


Note how the ground mesh was aligned to a height where it matched the door indicators (blue lines). These blue lines were previously configured to be 5 units above the module's base (`DoorOffsetY`)

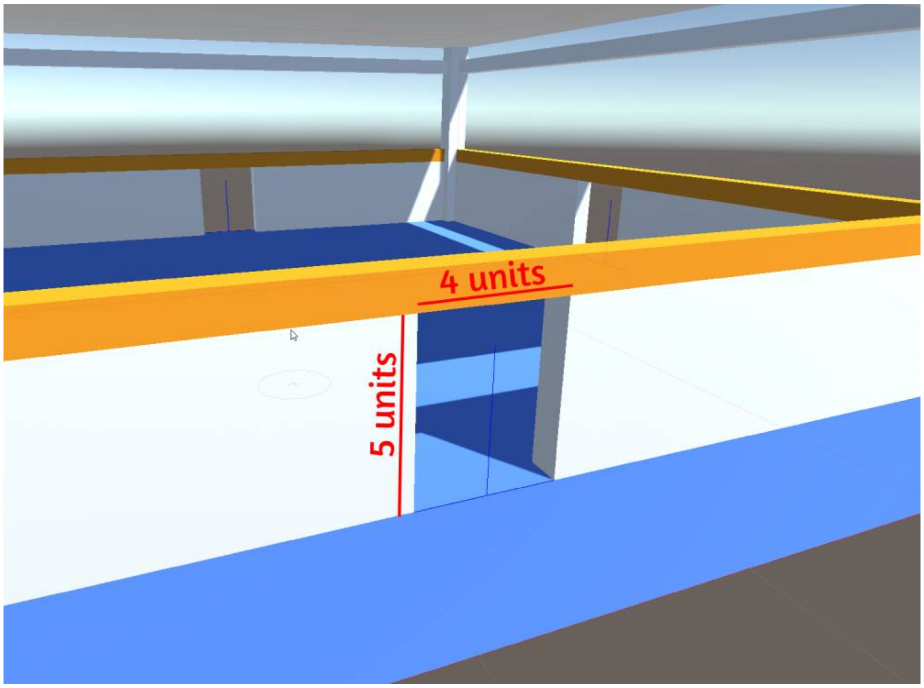
Go ahead and design the rest of the room in any way you like



Make sure all your game objects are inside the module game object

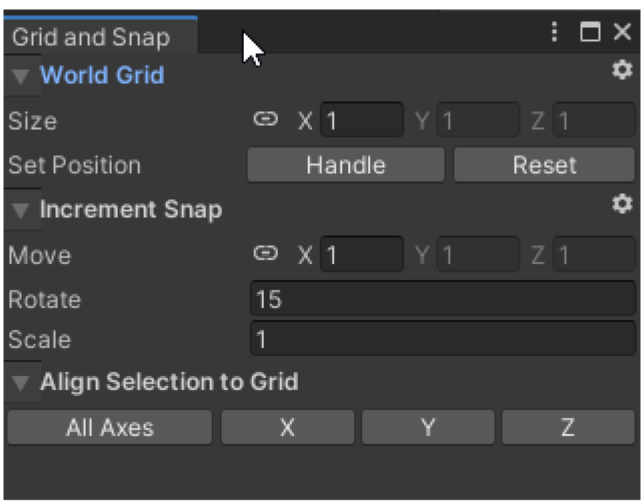


In this simple example, we want to have doors on all the 4 sides. A gap of `4 wide / 5 high` was left out at the door openings. Our door and wall assets will eventually be of this size to fill up the gap



Do this for all the four doors

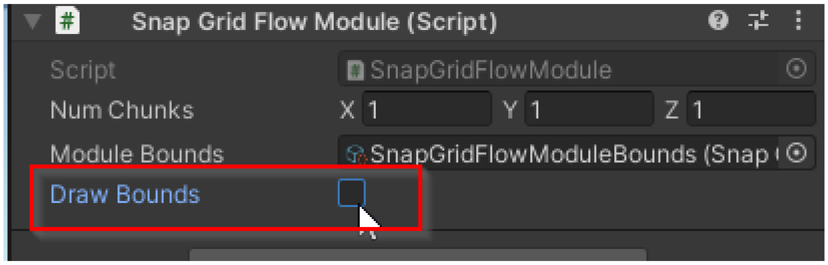
While designing the rooms, it is always a good idea to use Grid snapping (`Edit > Grid and Snap Settings`)



More info on snap settings here

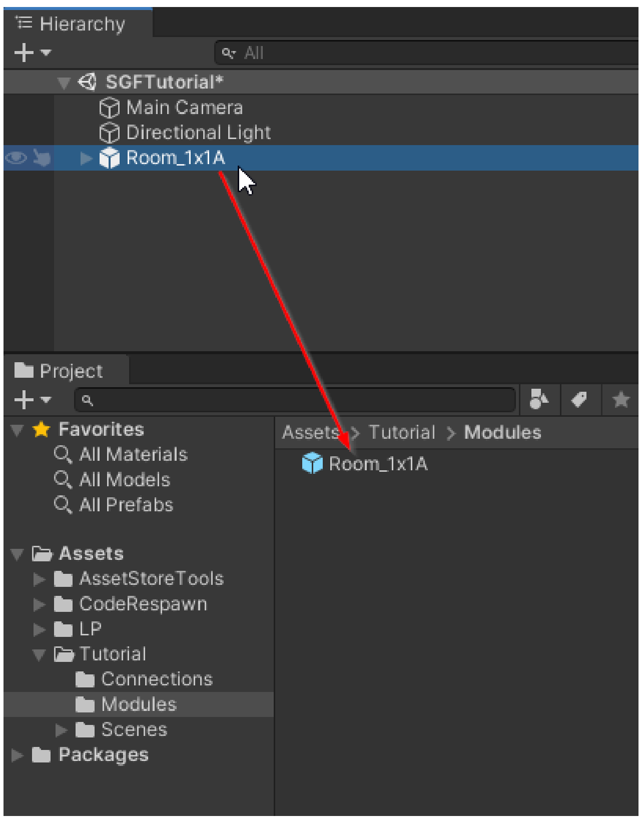
You can always turn off the module bound visuals (the red box) if it gets in the way. Do this by unchecking `Draw Bounds` in the `Snap`

`Grid Flow Module` component

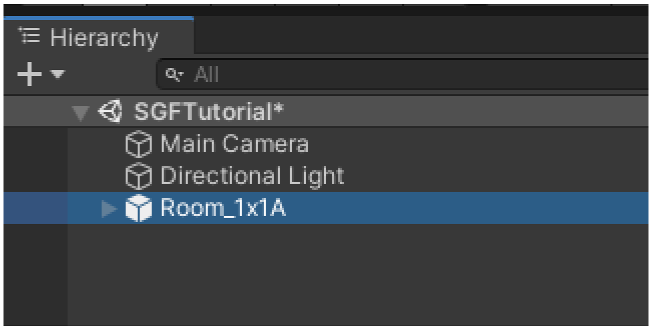


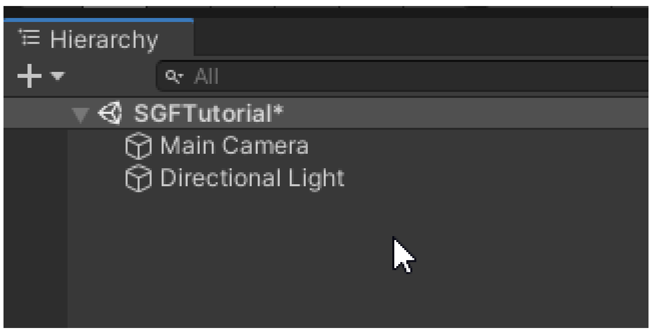
Save as Prefab

Select the module game object `Room_1x1A` and save it as a prefab in some folder



Now that we have it saved as a prefab, delete the module game object from the scene





In the next section, we'll create a `Connection` and add it near the door openings## Connections

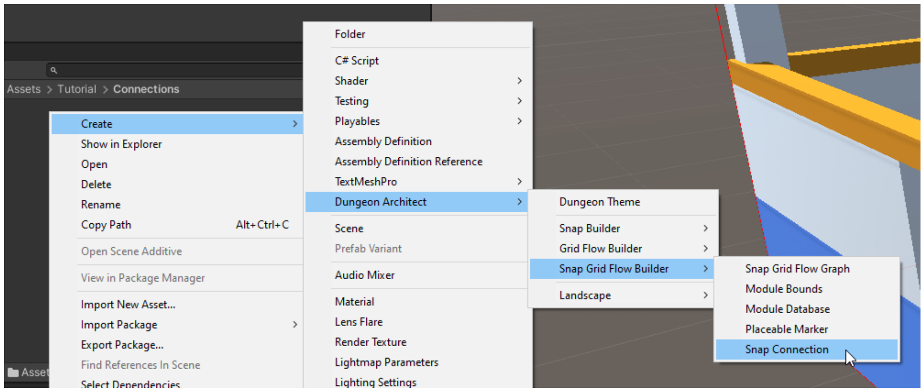
A `Connection` is a stitching point that the Snap framework uses to join rooms together. A connection contains a reference to a Door asset and a Wall asset.

If two rooms are stitched together through this stitching point, a door will be displayed in that position, otherwise the empty space will be walled off by the specified Wall asset

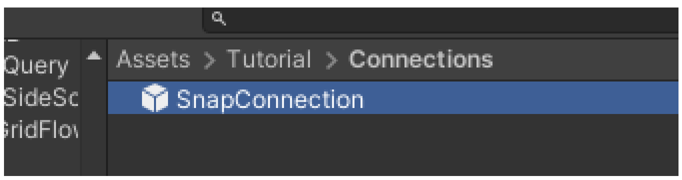
Create a Connection Asset

Create a new `Snap Connection` asset in a folder

`Create > Dungeon Architect > Snap Grid Flow Builder > Snap Connection`

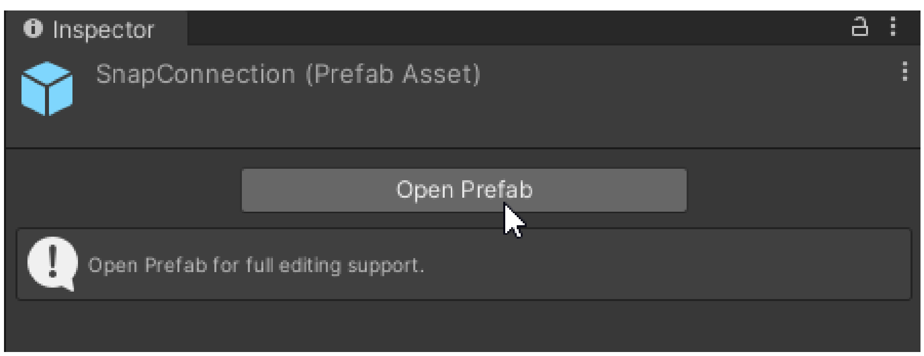


This will create a snap connection prefab asset that you can open and customize

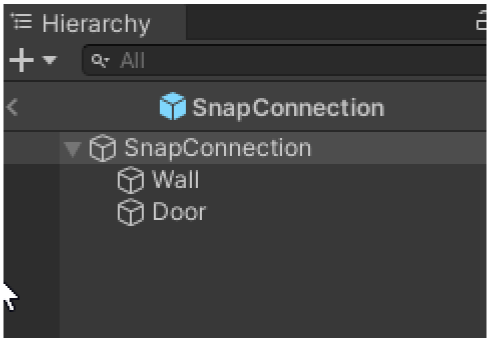


Customize Connection Asset

Select the newly created snap connection asset and click `Open Prefab` from the inspector tab



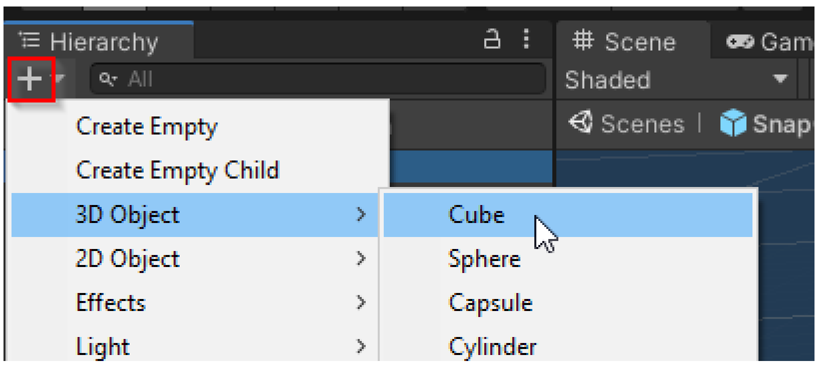
After you've opened the snap connection prefab, you'll see that it is setup like this:



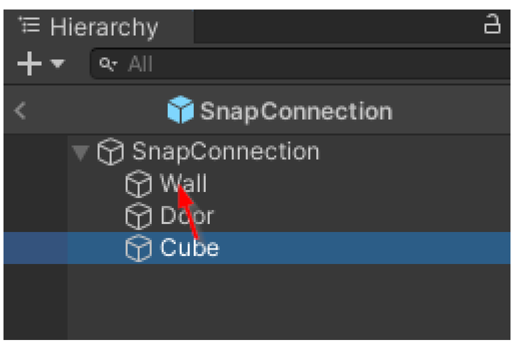
You will place your wall assets under the `Wall` game object and the door assets under the `Door` game object

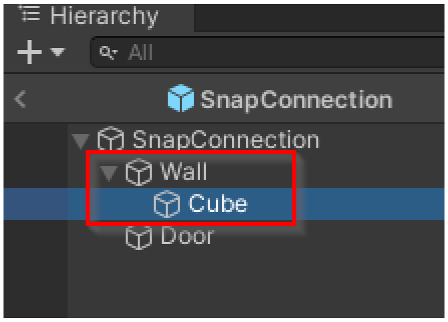
Dungeon Architect will take care of automatically showing the appropriate game object (wall or door) and hide the other one

Setup Wall Asset In this example, we'll use a simple cube as a wall. Create a new cube mesh from the create menu



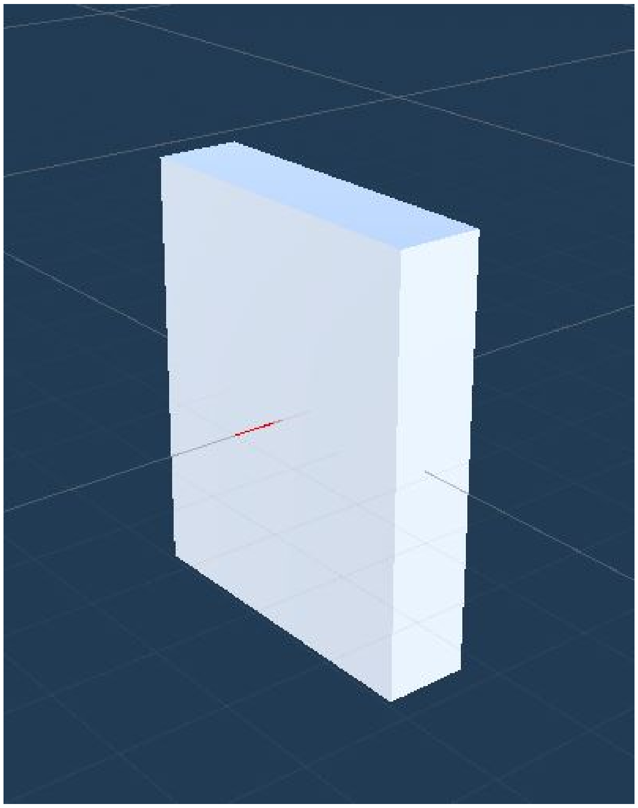
Parent this cube under the `Wall` game object.





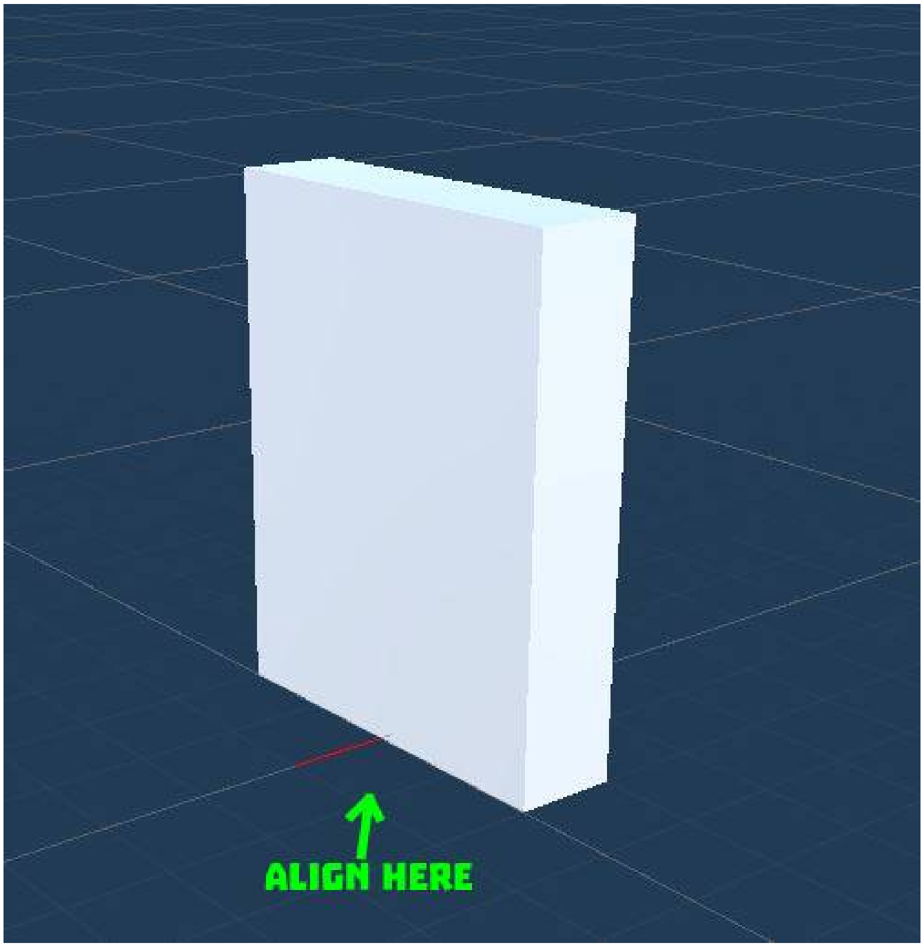
We need the wall to be 4 units wide and 5 units tall (to cleanly cover up the gap we created in our modules previously)

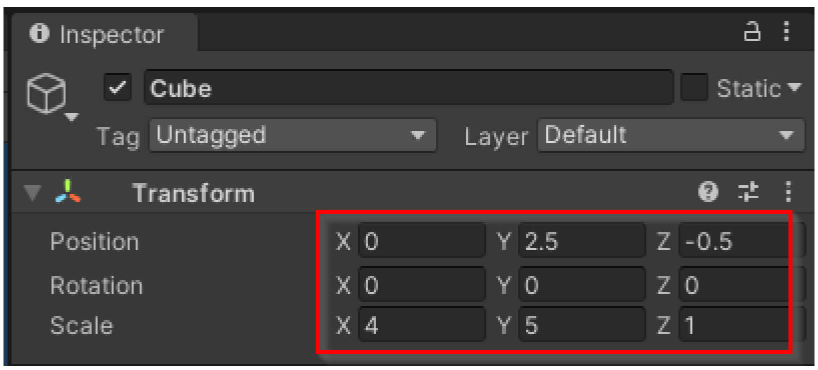
Select the cube mesh and set the scale to `(4, 5, 1)`



The wall should be moved appropriately so that: * The red line is at the bottom-center of the wall * The wall should be behind the red line's origin point

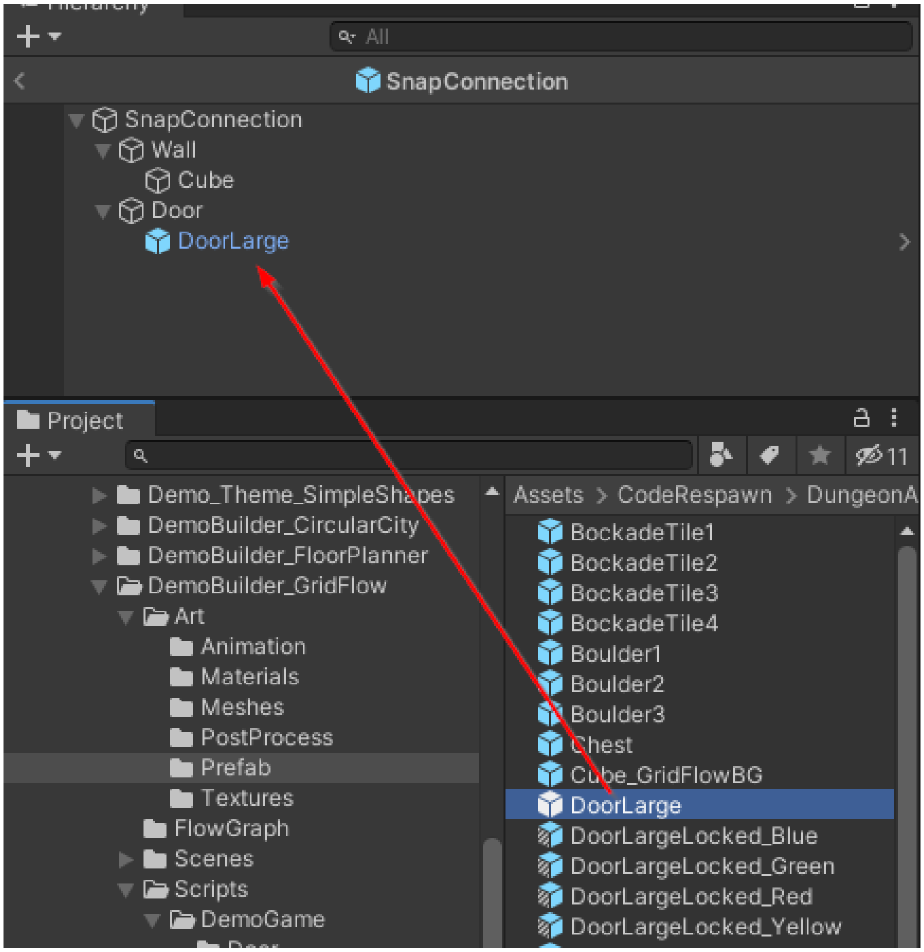
Move the wall up and a bit back by setting the location to `(0, 2.5, -0.5)`

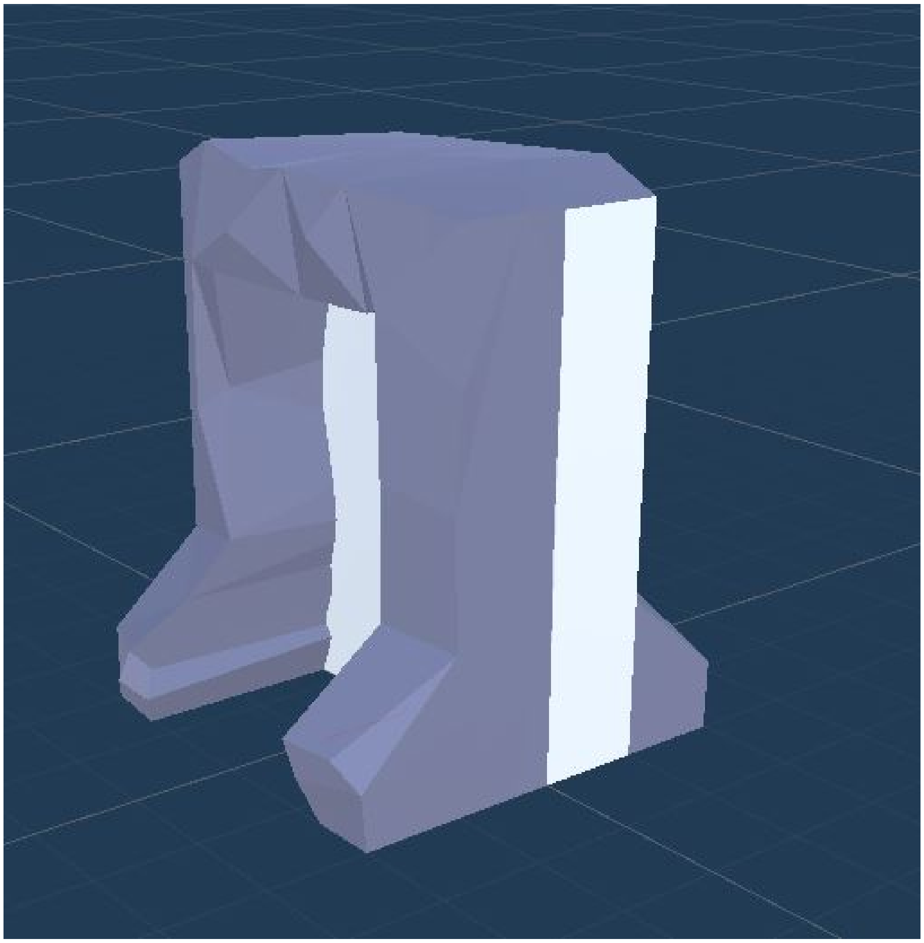




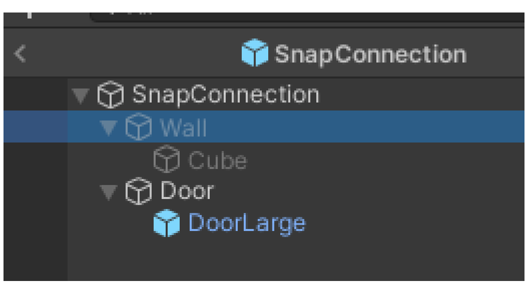
Setup Door Asset There's a simple Door prefab that comes bundled with the samples. We'll use that here, however, feel free to use your own door prefabs

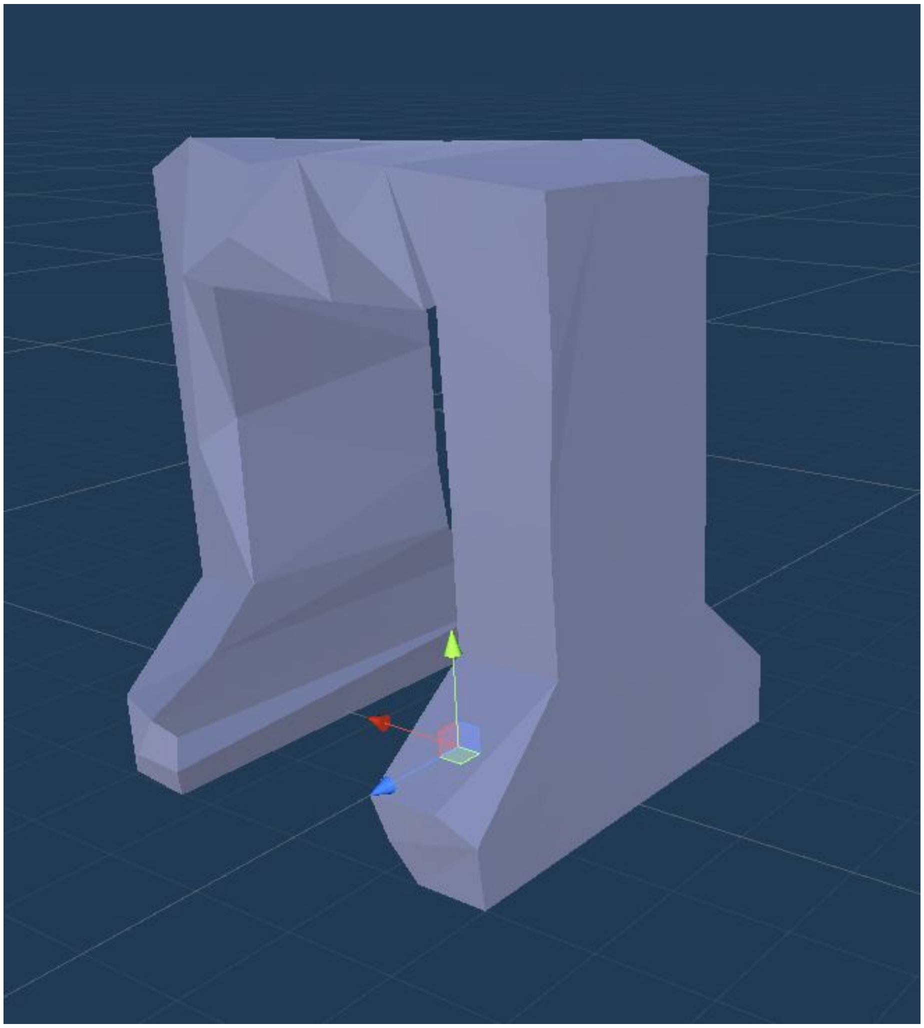
Navigate to `Assets\CodeRespawn\DungeonArchitect_Samples\DemoBuilder_GridFlow\Art\P` and drop in the `DoorLarge` prefab under the `Door` game object





Go ahead and hide the `Wall` game object. Dungeon Architect will take care of making it visible where needed



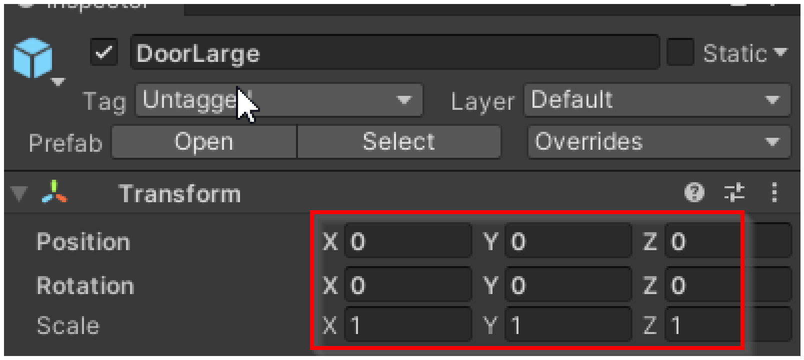


:::warning Important You may safely hide the main `Wall` or `Door` game objects but do not hide what is underneath it. For example, do not hide the `Cube` or the `DoorLarge` game objects :::

The rules for aligning the door with the red line are a bit different

You should move the door asset so that: * The red line is at the bottom-center of the door * The origin point of the red line should be at the center of the door

Our door prefab is already of the correct size (5 units tall and 4 units wide) and the pivot is in the right position. So reset the transform of the `DoorLarge` game object

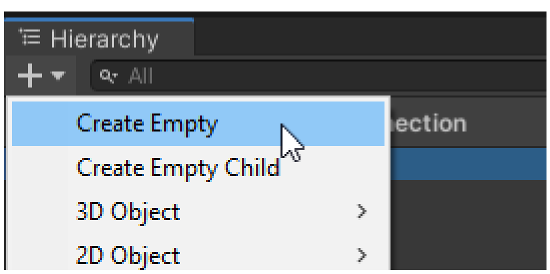


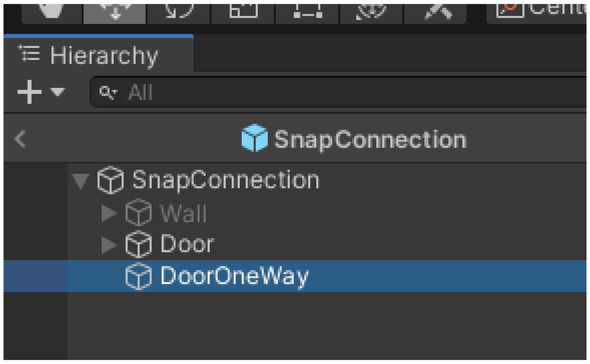
You'd want your door asset to be twice as thick as the walls. This is because when two adjacent modules are stitched together, we have two walls from each modules and a door that is twice as thick as the walls will cleanly fill up the gap

Setup One-way Door Asset Some doors will be promoted to one-way doors. This is done so that the player doesn't bypass a locked door and enter through another nearby door. You'll need to provide a door prefab that opens only from one way

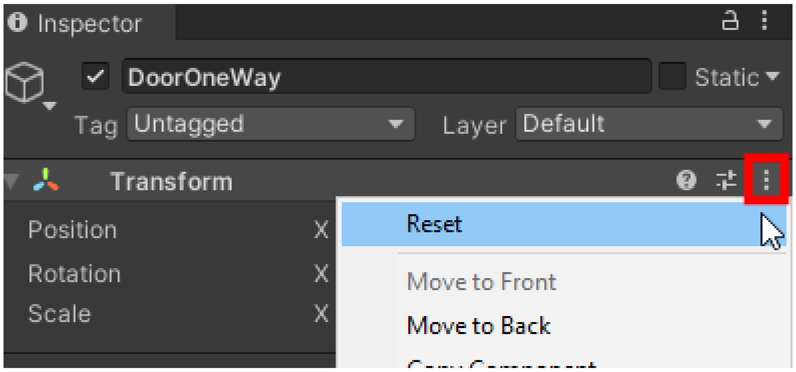
Right now, the snap connection is setup to support a `Door` and a `Wall`. We are going to add support for a one-way door.

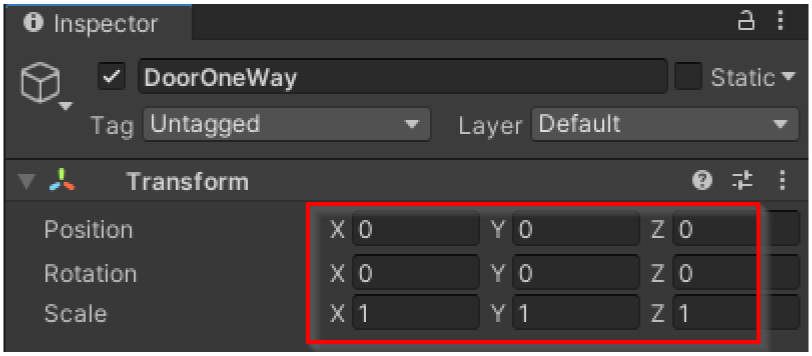
1. Create a new game object and name it `OneWayDoor`. This should stay alongside the `Door` and the `Wall` prefabs



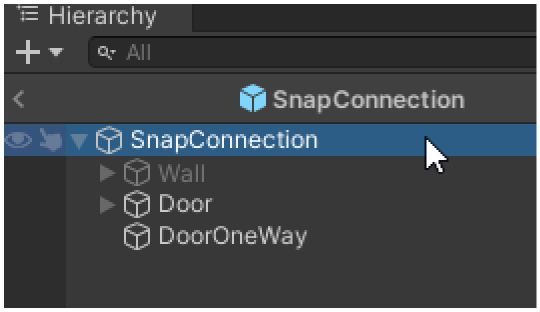


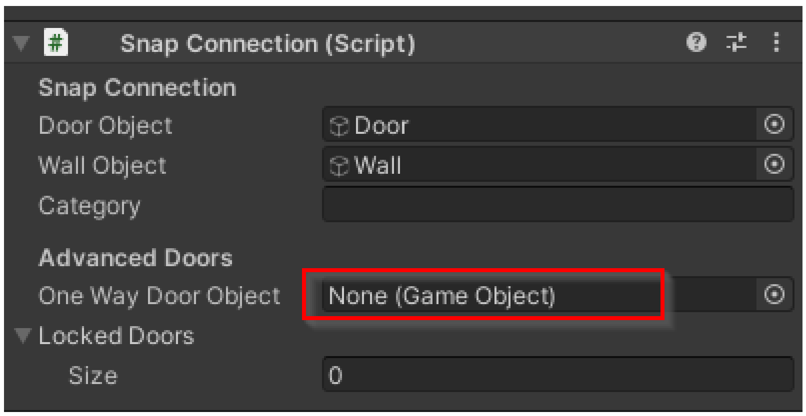
2. Select the `DoorOneWay` game object and reset the transform



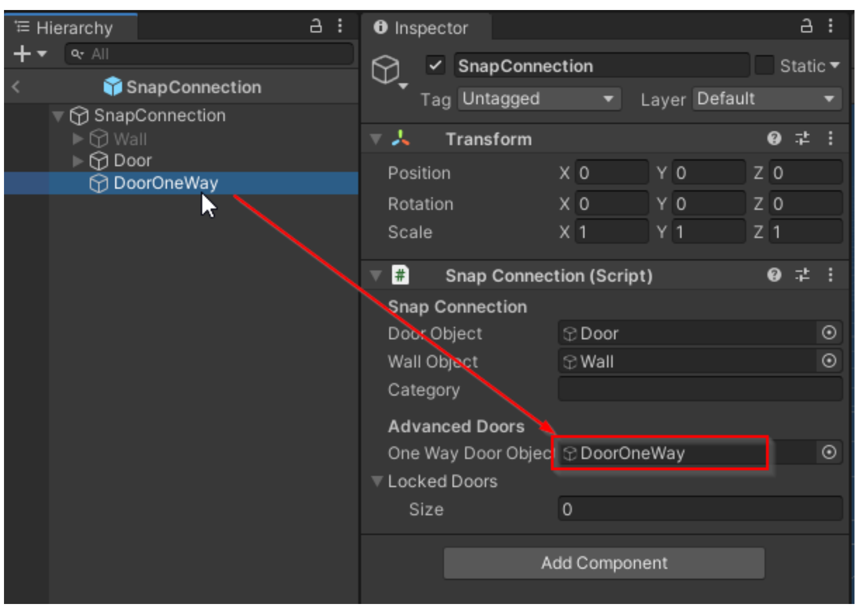


3. Let the SnapConnection know that this game object represents a one-way door. Select the root SnapConnection game object and inspect the properties



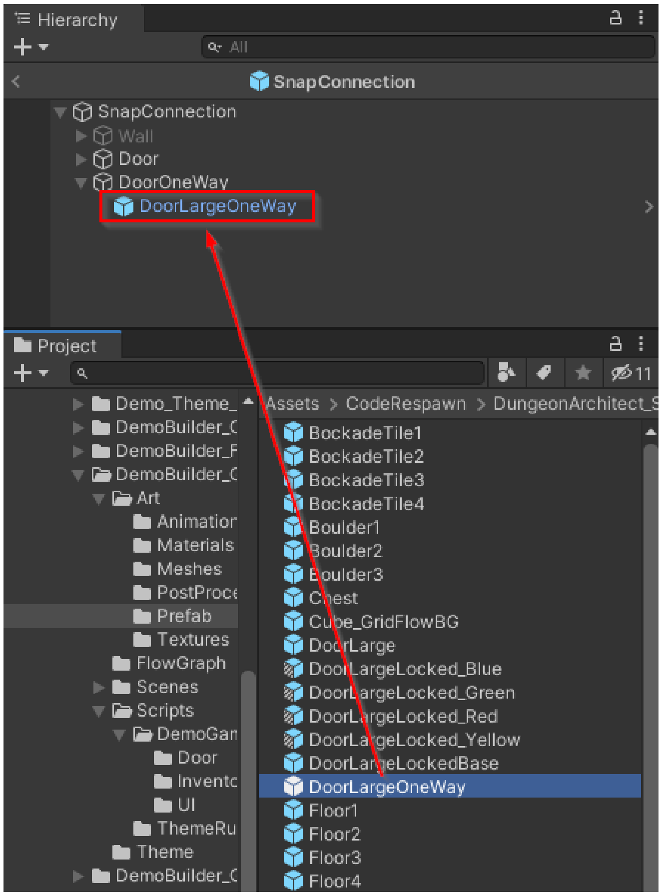


We'll assign our newly created `DoorOneWay` game object here

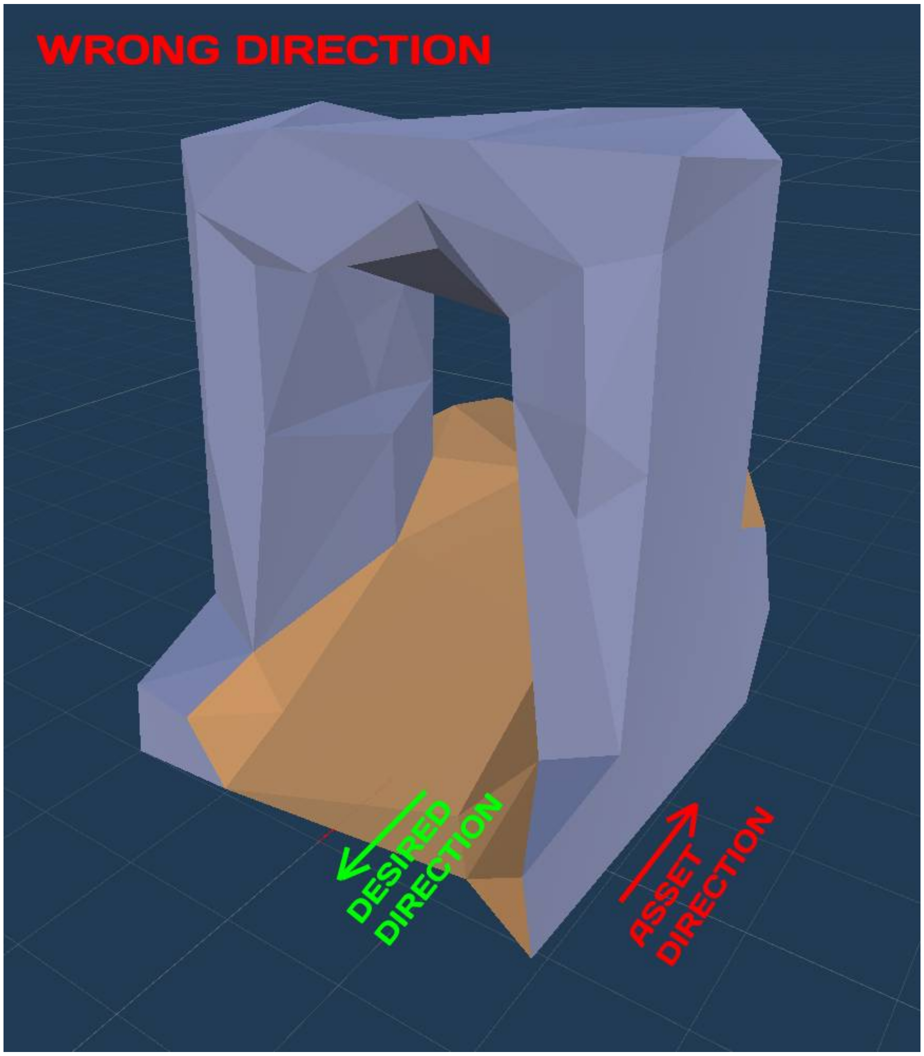


We can now place our one-way door assets under this. Navigate to `As-`

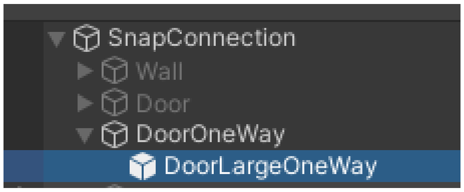
`sets\CodeRespawn\DungeonArchitect_Samples\DemoBuilder_GridFlow\Art\Prefab` and drop in `DoorLargeOneWay` prefab as shown below

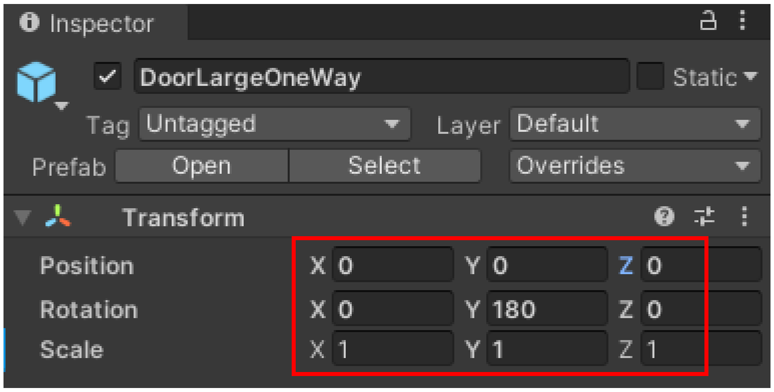


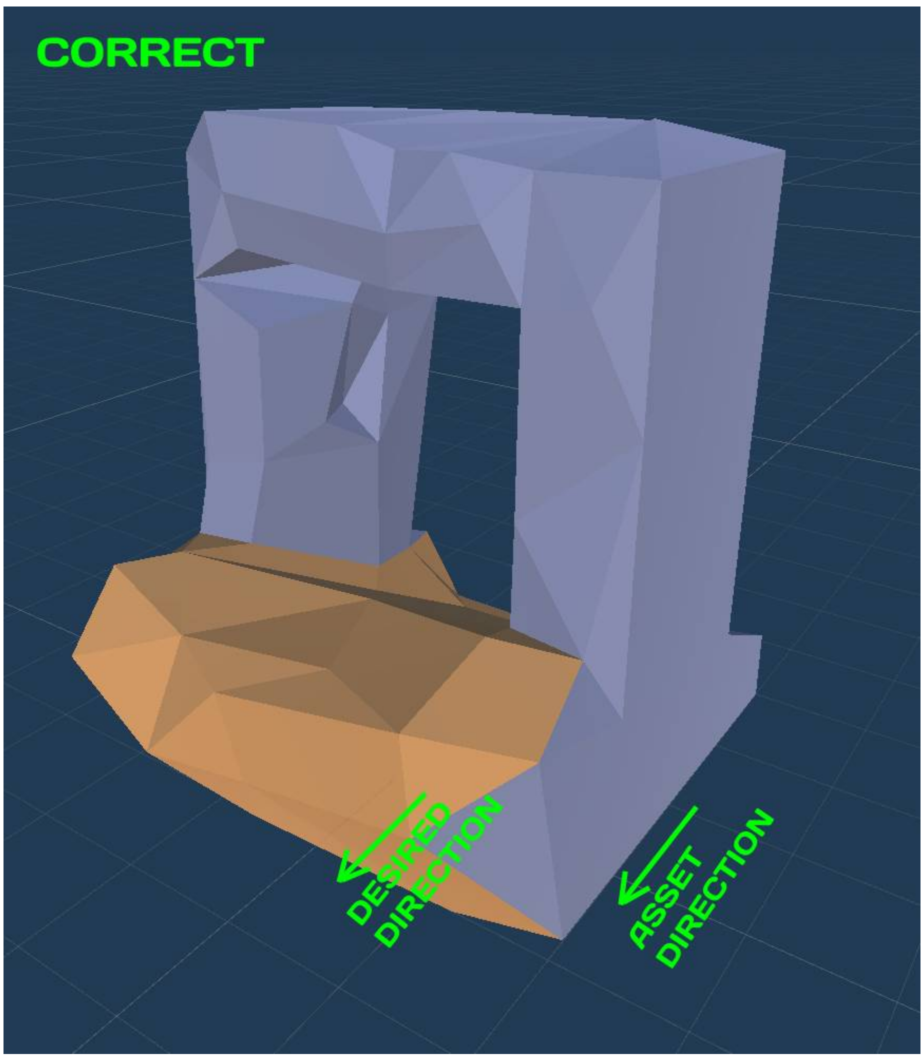
The alignment rule of a one-way door is similar to a door. The direction in which we are allowed to go through the door should follow the red line outwards



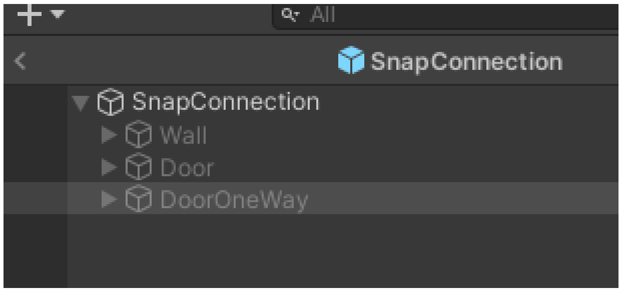
Reset the transform of the asset and rotate it along Y by `180` degrees since it is facing the wrong way



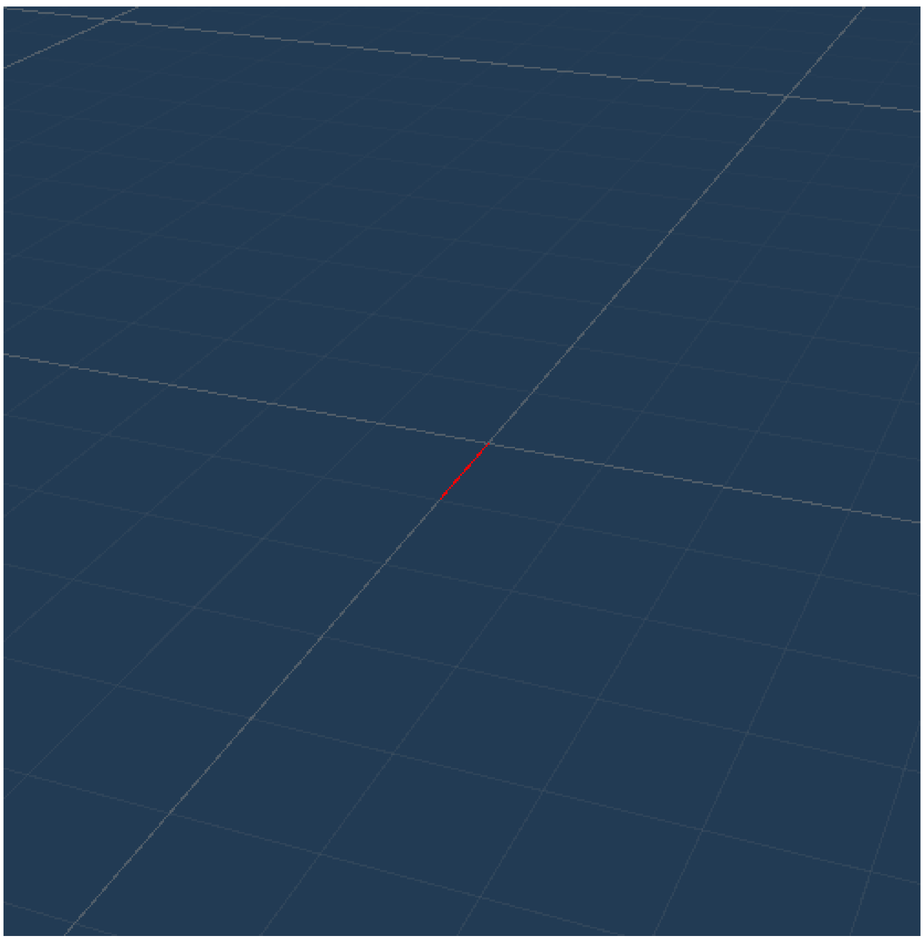

Inspector
DoorLargeOneWay
Static
Tag Untagged
Layer Default
Prefab Open Select
Overrides
Transform
Position X 0 Y 0 Z 0
Rotation X 0 Y 180 Z 0
Scale X 1 Y 1 Z 1


CORRECT
DESIRED DIRECTION
ASSET DIRECTION

Hide the `Door` and `DoorOneWay` game objects

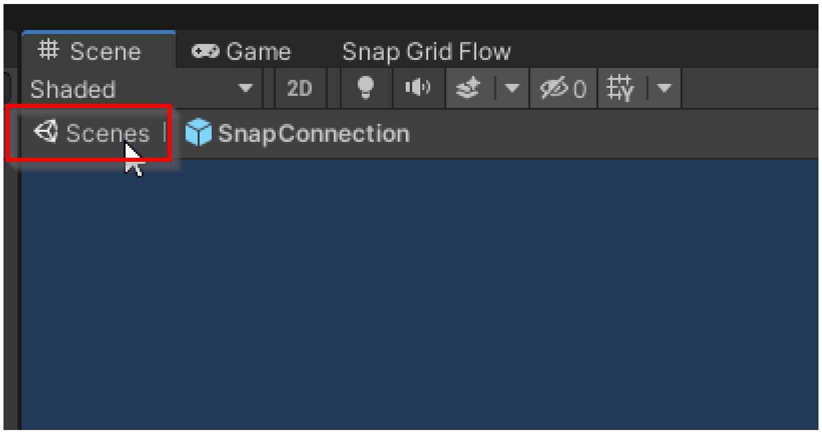


Make sure you hide the outermost object, namely `Wall`, `Door` and `DoorOneWay`



Our snap connection is now complete and we are ready to place them in our snap modules

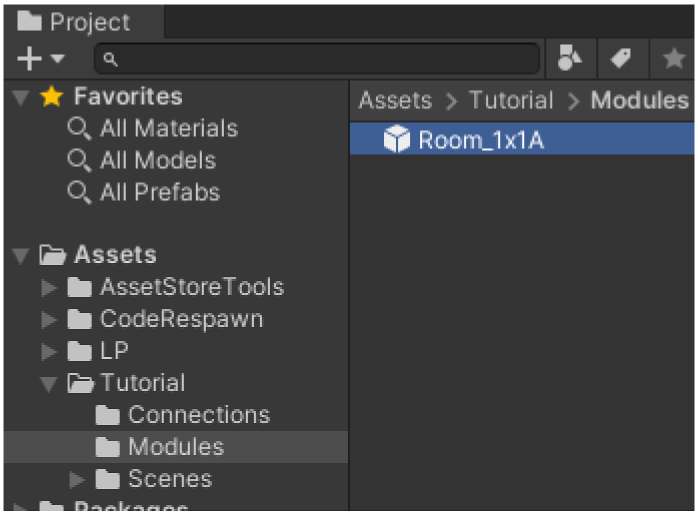
Exit out of the snap connection prefab and return to the scene by clicking the `Scene` button on the viewport

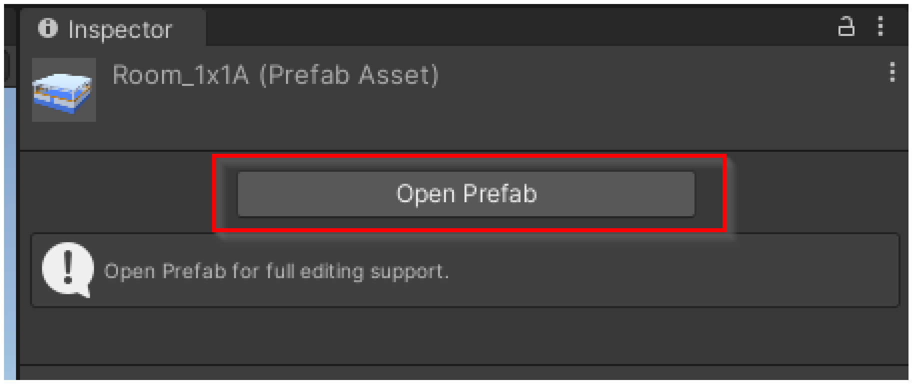


Place Connections on Modules

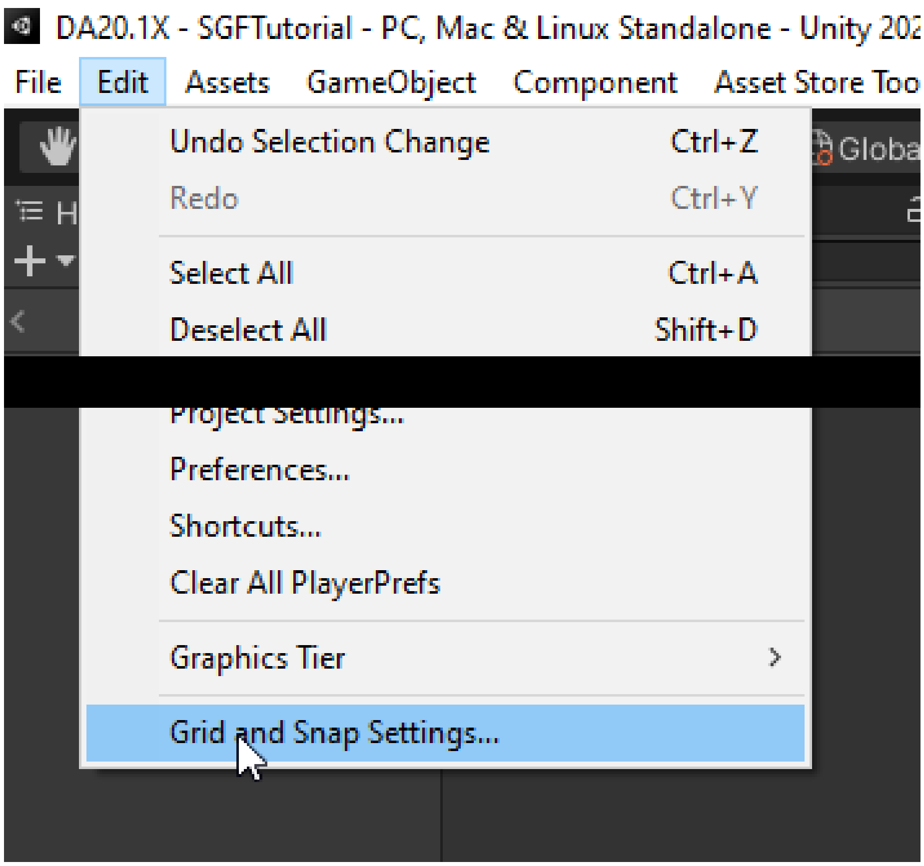
It's now time to use this snap connection on our modules

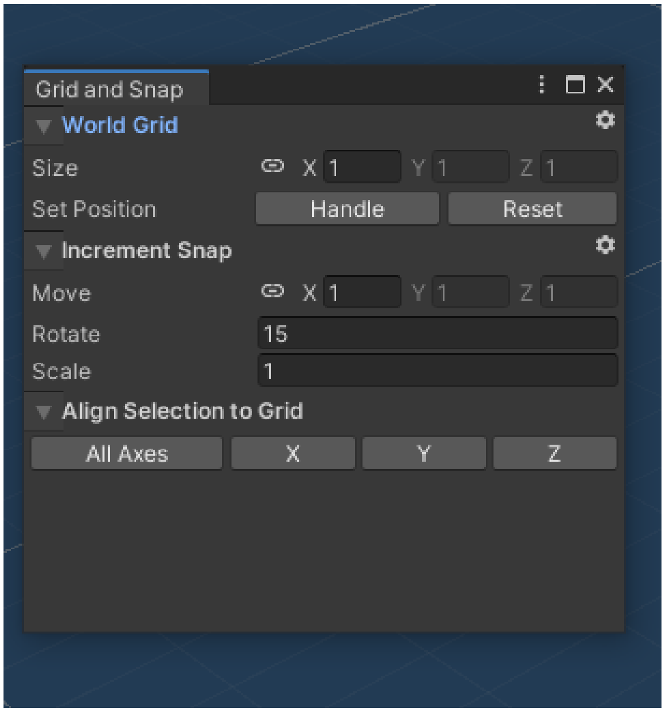
Open up the `Room_1x1A` module prefab we created earlier



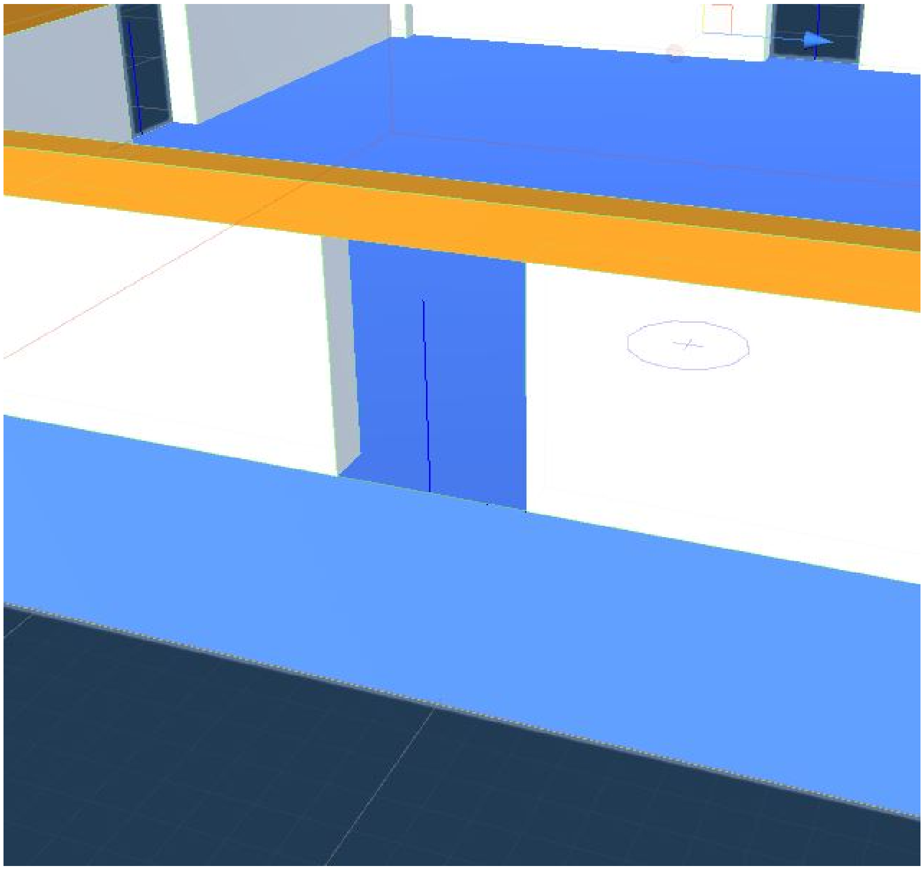


Open the Grid and Snap window so we can align our snap connections correctly. Navigate to Edit > Grid and Snap Settings

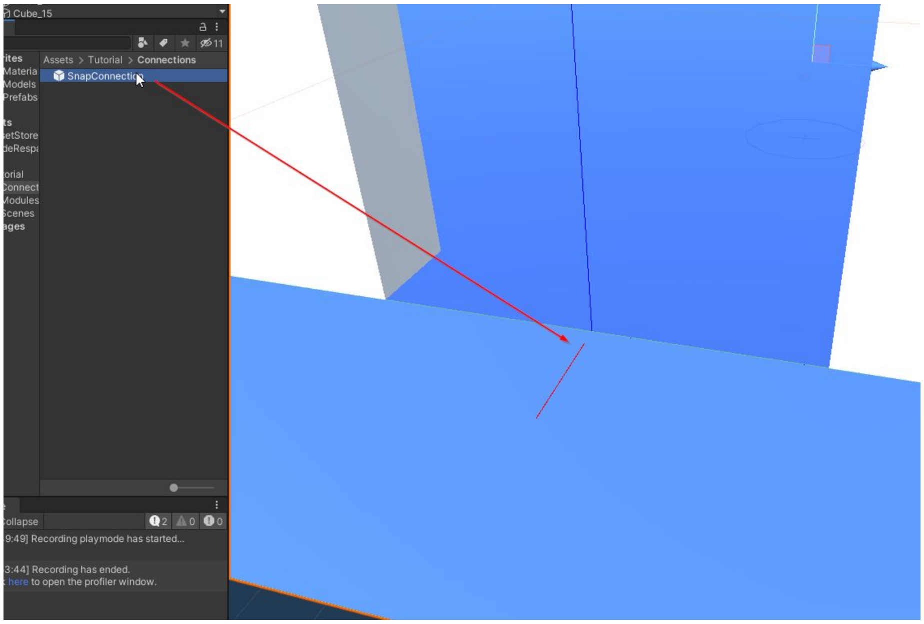




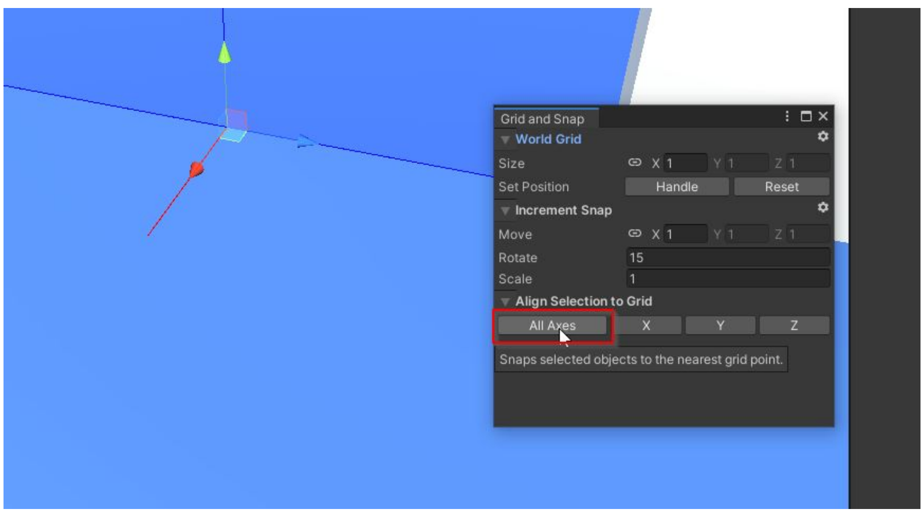
Move over to one of the doors in the module prefab



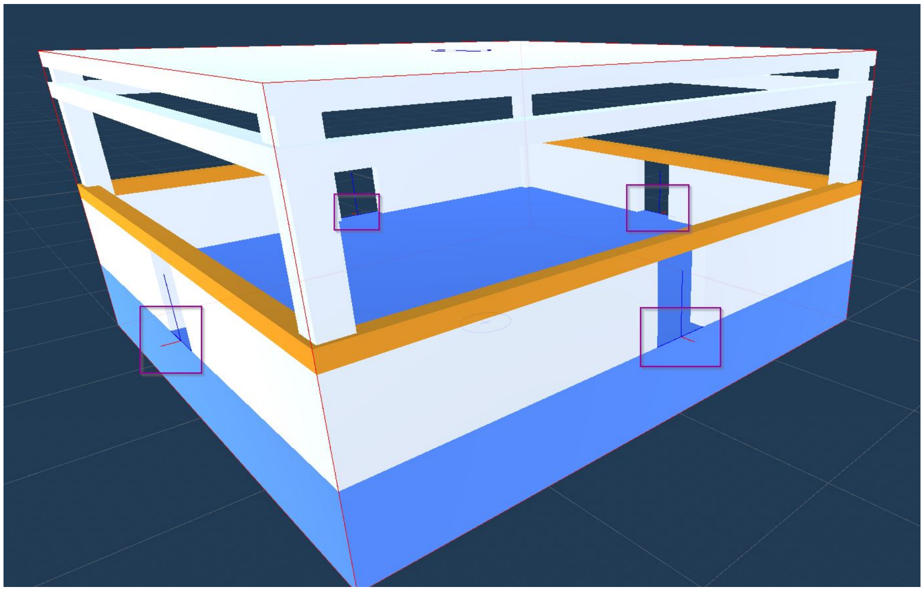
Drag and drop the snap connection and align the snap connection origin point in the blue door position. Make sure the red line points outward



The position and rotation might be off by a slight margin. Select the snap connection actor and click `All Axes` button on the Grid and Snap window and it should cleanly snap at the door position



Repeat this for all the 4 door openings



Module Database

A Module Database is a registry of all the available modules that Dungeon Architect can use to stitch the dungeon

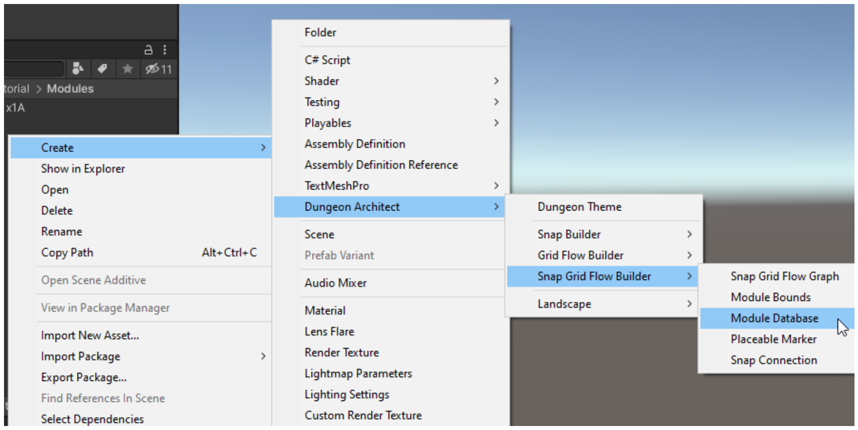
It also contains an acceleration structure with all the necessary information pre-calculated in the editor, so the stitching is fast at runtime

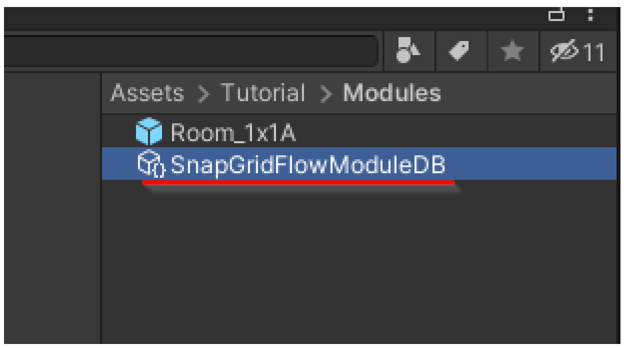
You may have different module databases to generate dungeons with different art styles (e.g. Sci-Fi spaceships, Medieval castles etc)

Create a Module Database

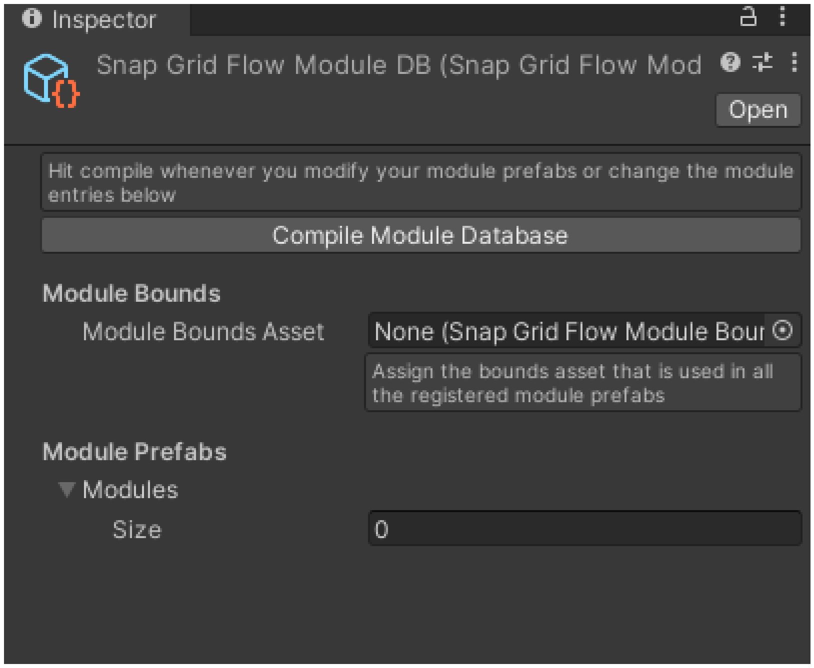
We'll create a new module database asset and register the module that we created in the previous sections

1. Move to an appropriate folder and create a module database from the Create menu:

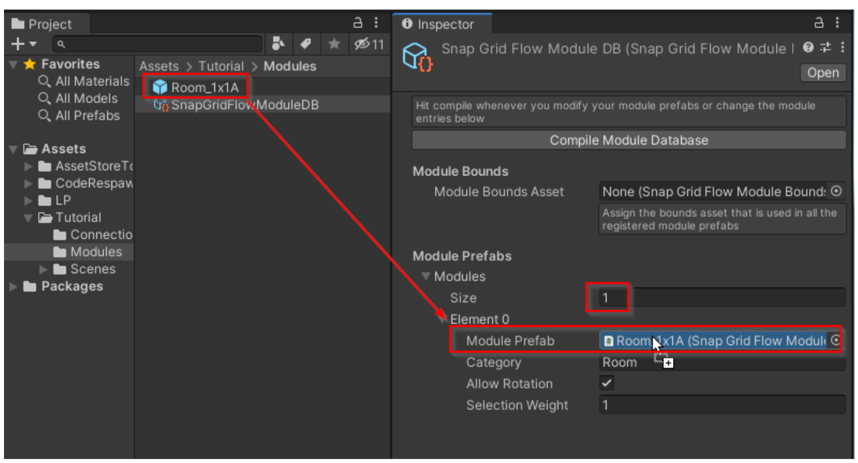




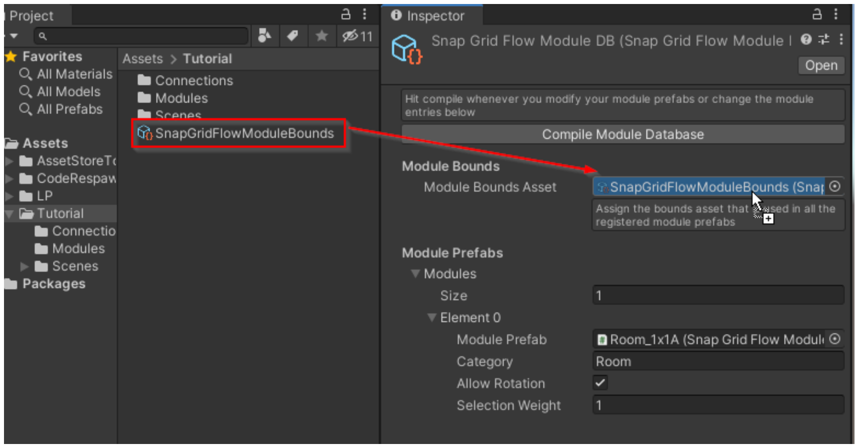
2. Select the module database and inspect the properties



3. Register your modules in the `Modules` array

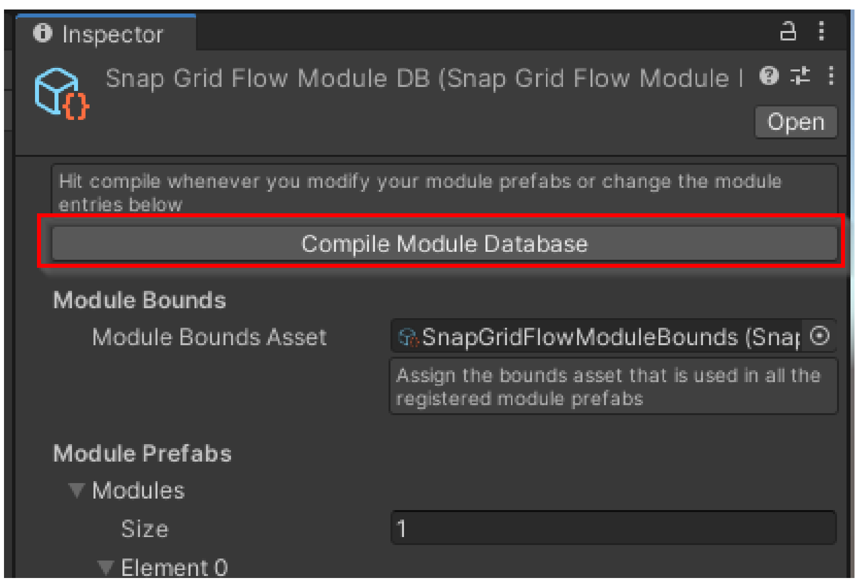


4. You'll also need to provide the bounds asset that is used in all of the registered module prefabs



All the modules registered under this module database should use this same bounds assets

5. Finally, Click `Compile Module Database` whenever you make any changes to either the module or the module database. This will do some internal calculations in the editor so your dungeons can build fast at runtime



:::warning Important This is an important step. Do not forget to recompile the module database whenever you make any changes to it or the modules themselves :::

Create Flow Graph

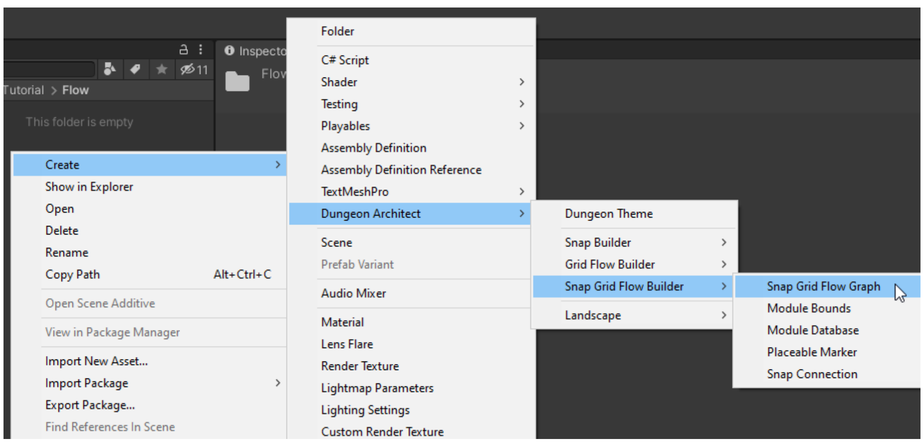
Design the layout of your procedural dungeons using the flow editor. Then create an infinite number of procedural dungeons that follow this layout rule.

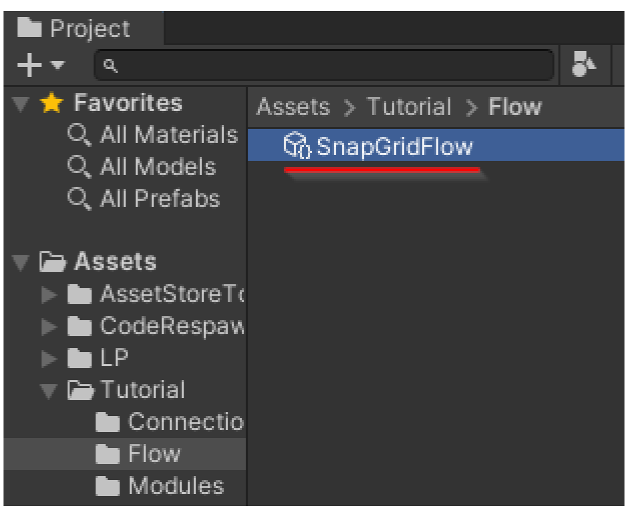
Create cyclic-paths, key-locks, teleporters, shops, treasure rooms, boss rooms and much more

Create a Snap Grid Flow Asset

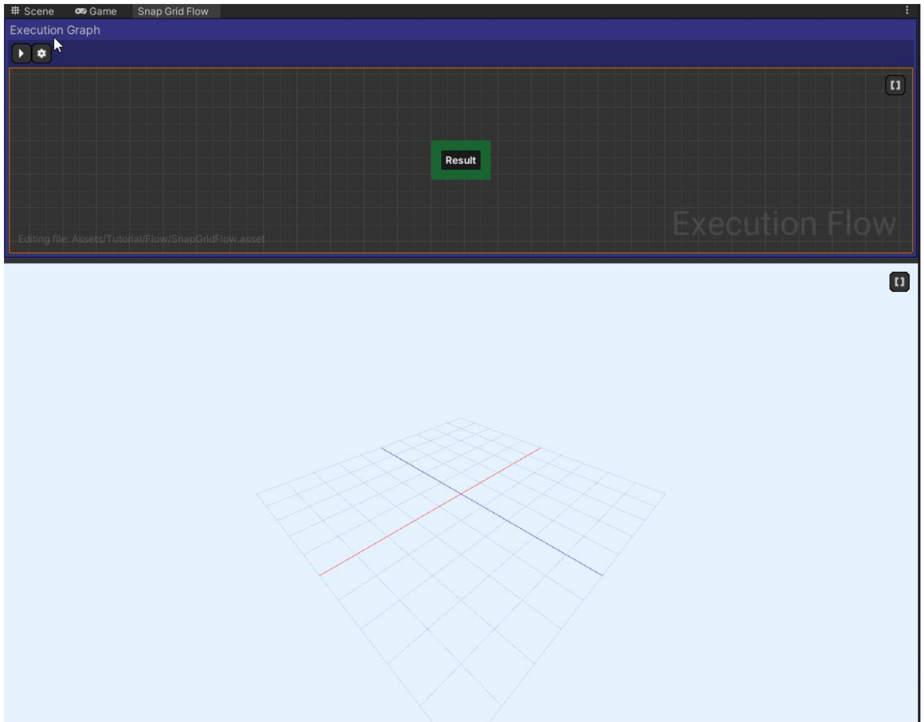
Move to an appropriate folder and create a new `Snap Grid Flow Graph` asset from the Create Menu

`Create > Dungeon Architect > Snap Grid Flow Builder > Snap Grid Flow Graph`





Double click the asset to open up the flow graph editor



The top panel is the Exection Flow Graph where you would be designing your dungeon flow. The resulting layout graph is shown in the 3D viewport below

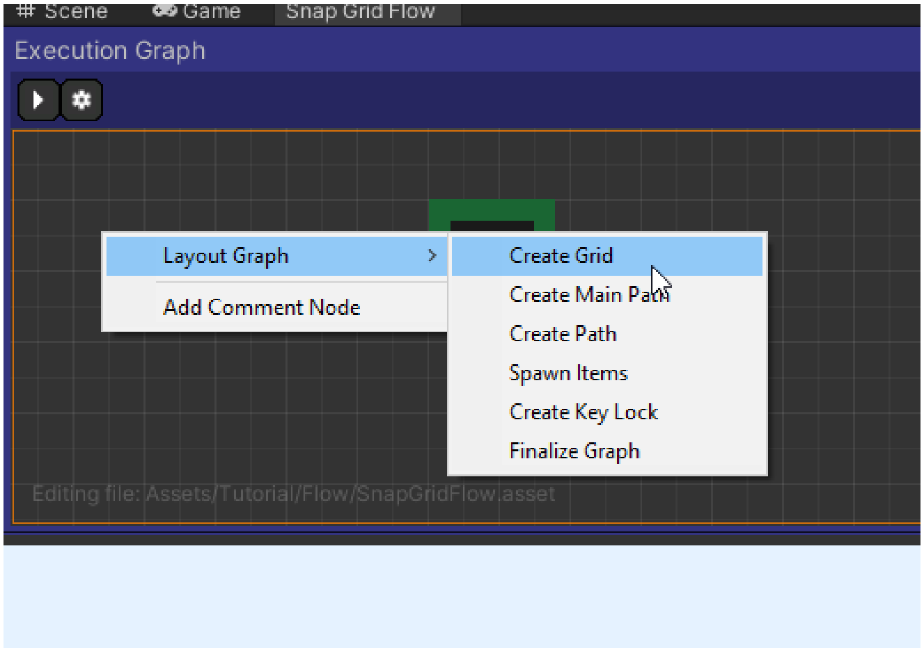
3D Viewport Navigation The navigation is similar to what you'd expect from Unity's scene view

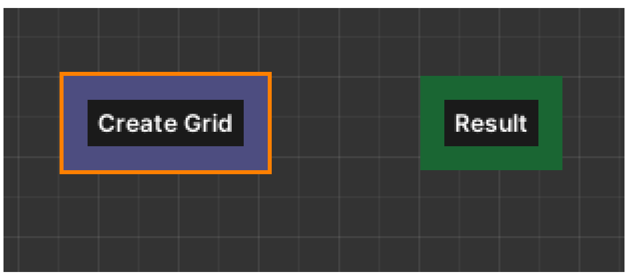
- Hold right mouse button and move to look around.
- Hold right mouse button and WASD to move
- Hold right mouse button and Q to move up, E to move down

Create Grid

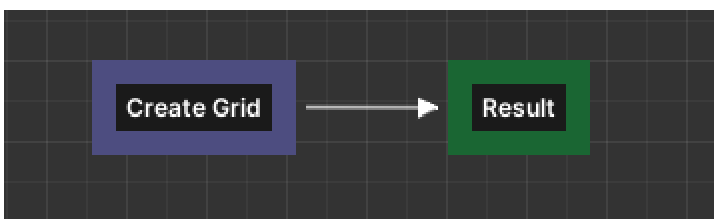
Right click on the execution graph and add a `Create Grid` node from the context menu

`Layout Graph > Create Grid`



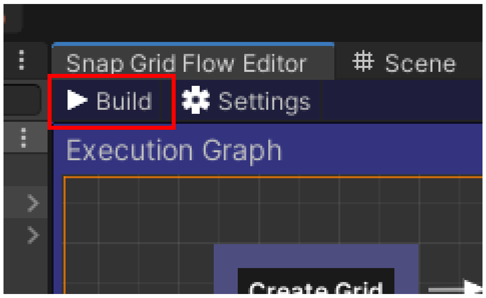


Link the `Create Grid` node to the `Result` node

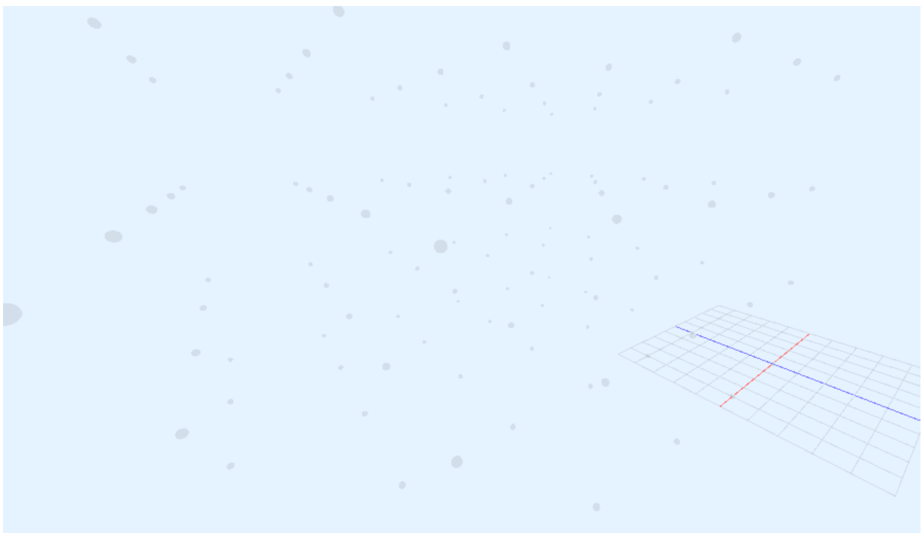


To create a link, hover the mouse pointer over the border of the `Create Grid` node until it turns yellow. Then drag a link out of it and attach it to the border of the `Result` node

Hit the `Build` button in the Execution Graph Panel



Move around in the 3D viewport and should should see a faint set of dots that represent the initial grid that this node has created. This is our work area and our dungeon layout will grow in it



Select the `Create Grid` node and inspect the properties

You may adjust the size of your initial work area here.

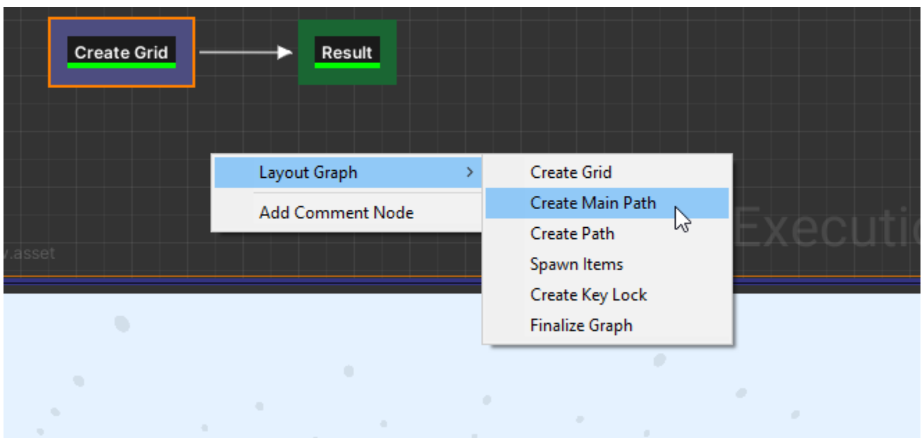
There are a few things to consider when choosing the size of your grid * Larger initial grids will require more processing power.
* Sometimes, it's better to have a smaller grid so it creates a nice tightly packed layout.
* However if the grid is too small, the paths will not have any room to grow

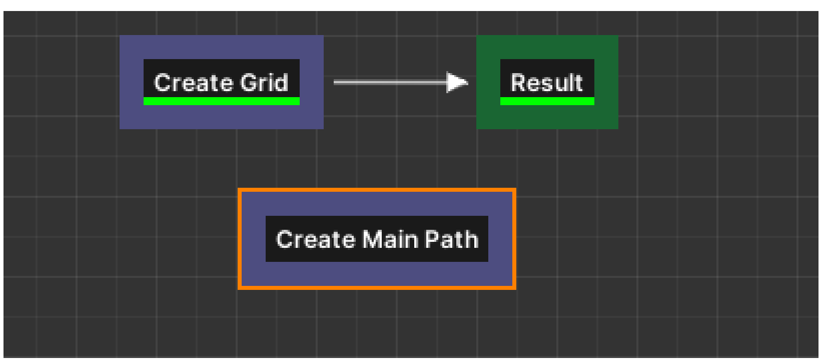
You should come back to this node and adjust it as needed. For now, we'll leave it to default

Create Main Path

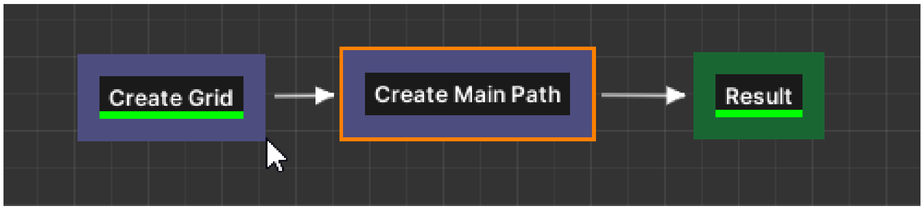
Add a `Create Main Path` node

`Layout Graph > Create Main Path`



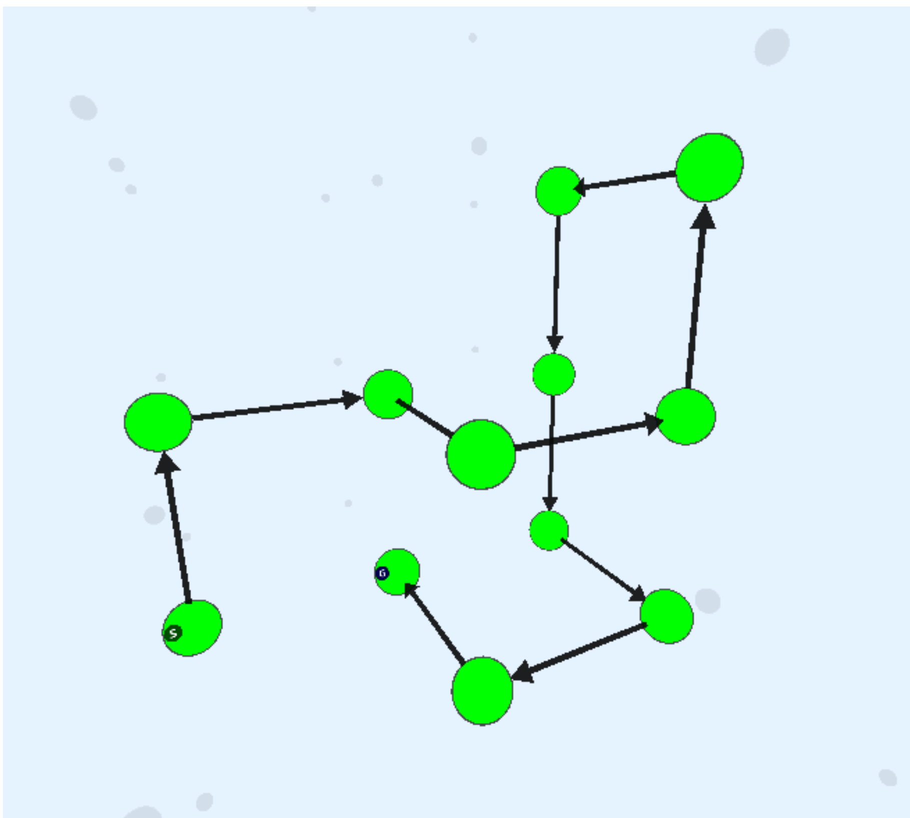


Break the previous connection we made to the result node and link it like this



To break the link, hover the mouse over the `Create Grid` node's border till it turns yellow. Then right click

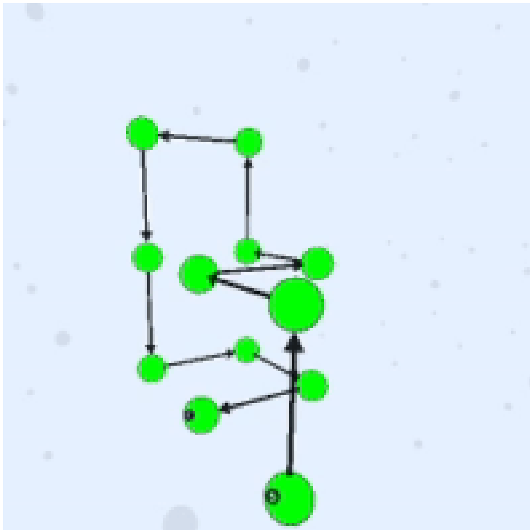
Hit build



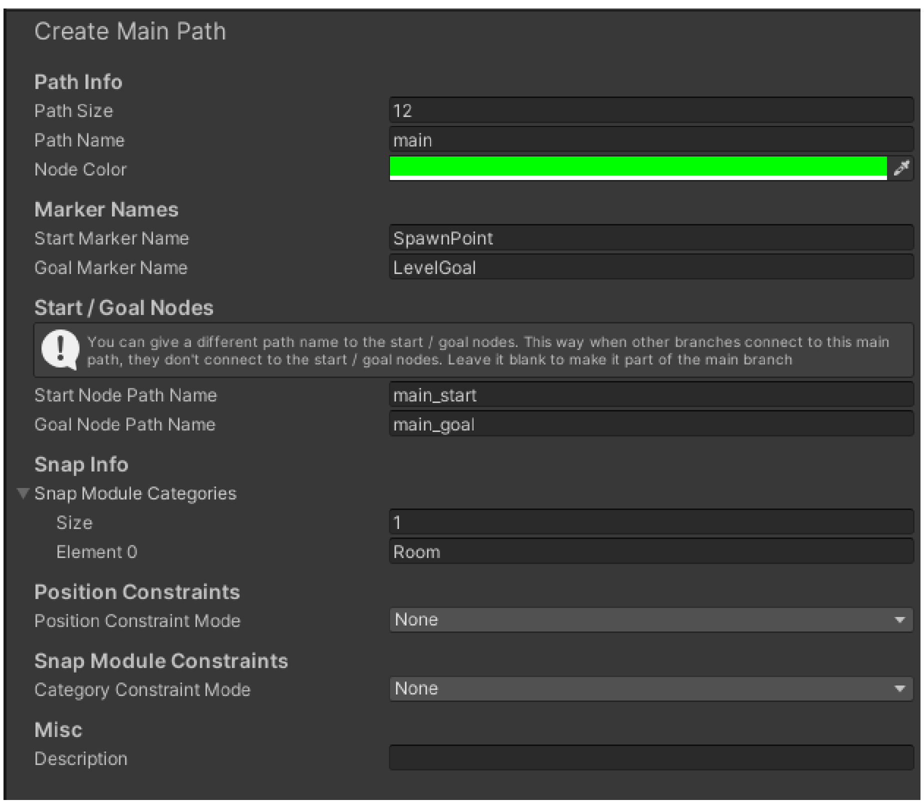
Keep hitting build for different results

More Viewport Controls

- Hold Alt + Hold Left Mouse button + Move mouse to orbit around the selected nodes
- Press F key to focus on the active nodes and reset the orbit pivot



Main Path Properties Select the `Create Main Path` node and inspect the properties

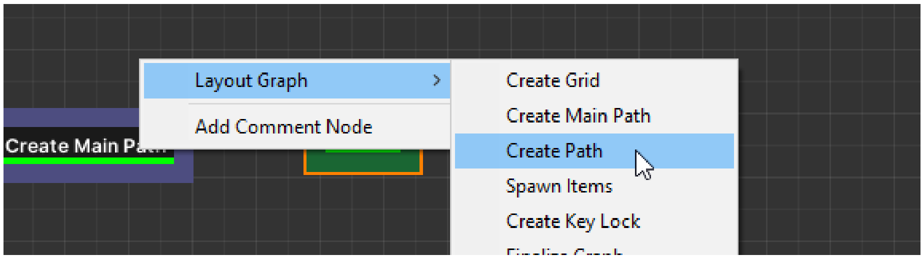


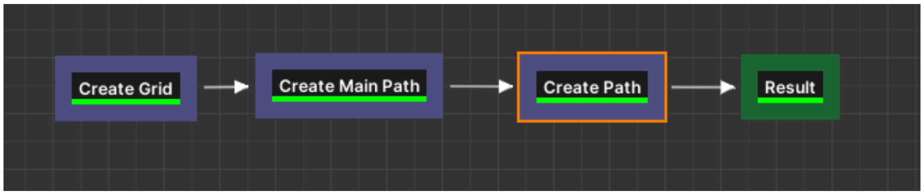
Parameter | Description Path Size | Control the size of the path with the variable Path Name | Each path in the flow system can be later referenced using its path name. The name of this path is set to `main` Snap Module Categories | When you registered a room in the module database, you specified a category (which defaulted to `Room`). This allows you to control the module prefabs that would be used from that list to stitch the path Node Color | Adjust the preview color of the nodes created in this path Start/End Marker Names | These values allow you to insert your prefabs into the start / end rooms using the theme engine. More on this later Start/End Node Path Name | Override the path name of the first and the last node in the path. This allows you to give a unique path name to your spawn room and goal room Position Constraints | Control the position of the nodes in the path with your own scripts and rules. More on this in the later sections Snap Module Constraints | Override the snap modules for any of the nodes in the path with your own scripts and rules. More on this in the later sections

We'll leave all the properties unchanged for now

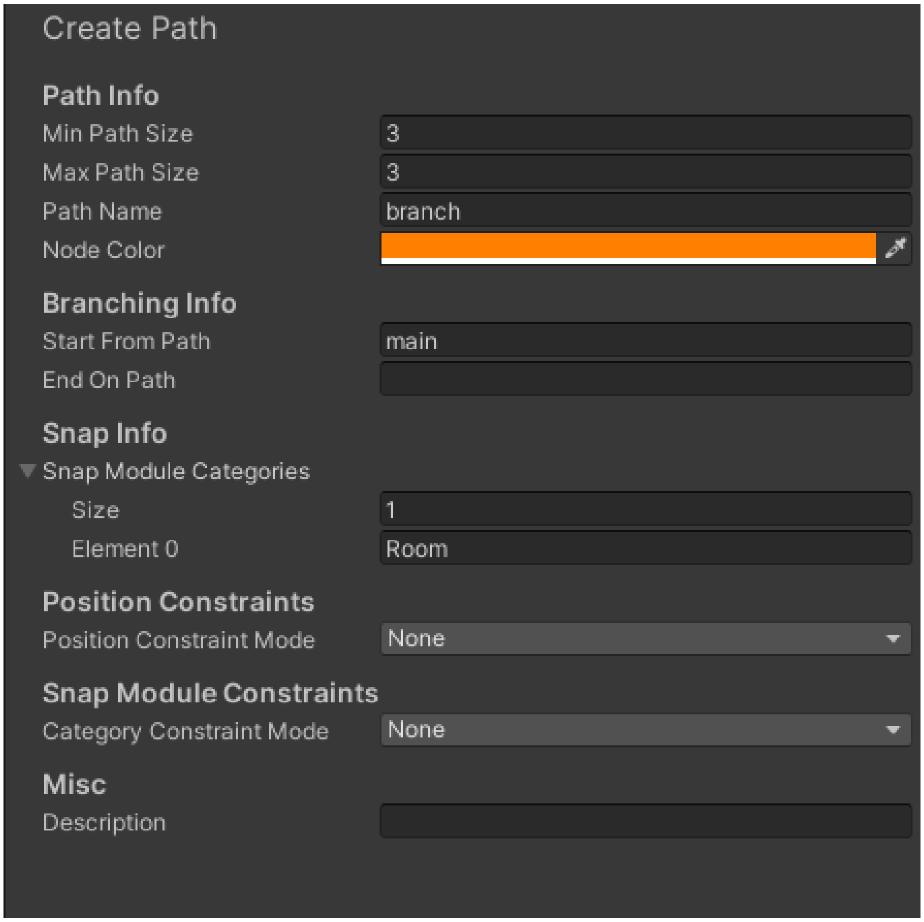
Create Alternate Path

Add a node `Create Path` and connect it as below

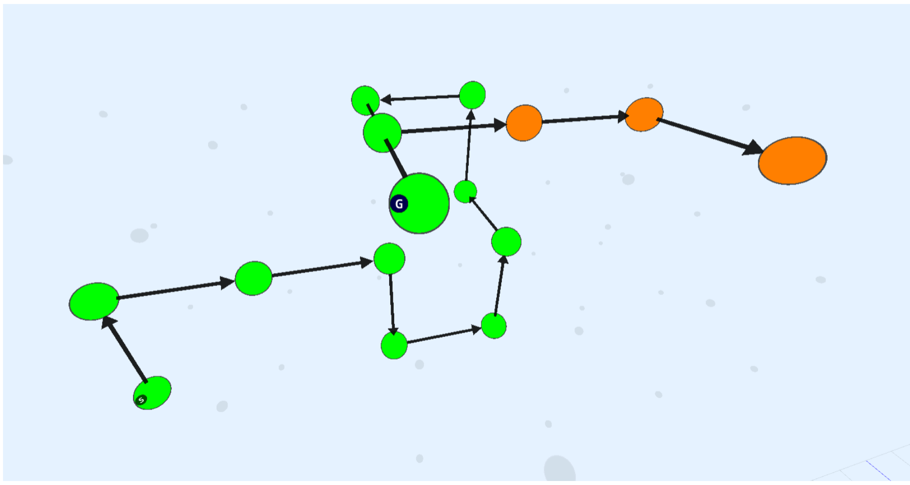




Select the node and inspect the properties

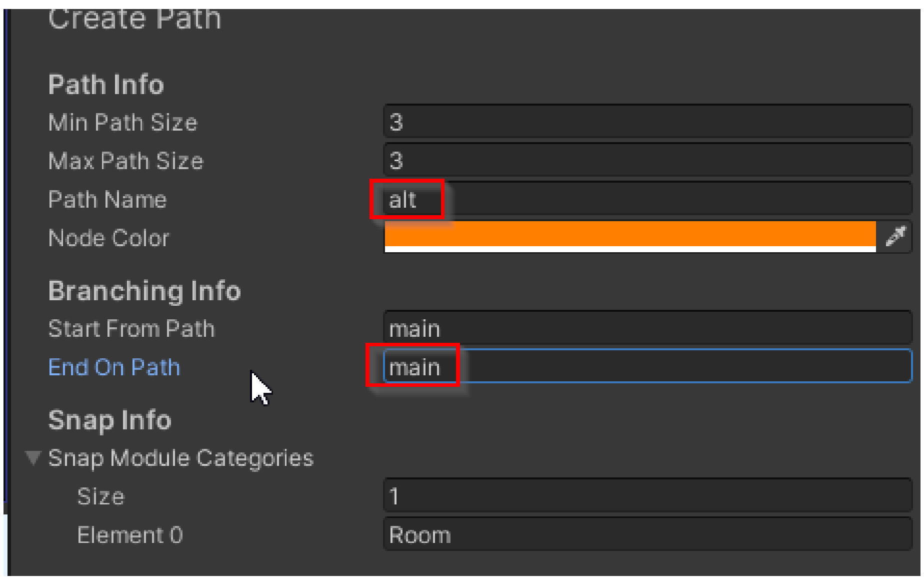


Build the graph, and you'll see a new branch (orange) emitting out of the main branch (green)



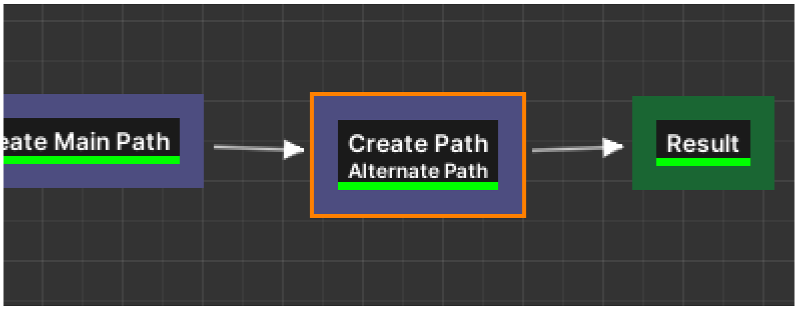
This new orange path branched out of the main (green) path because we have configured it to do so by setting the `Start from Path` parameter to `main`

We would like to have this new path merge back into the main path. Set the `End on Path` parameter to `main`. We also want to reference this new path as `alt`



Control the size of the path with the `Min/Max Path Size` parameter

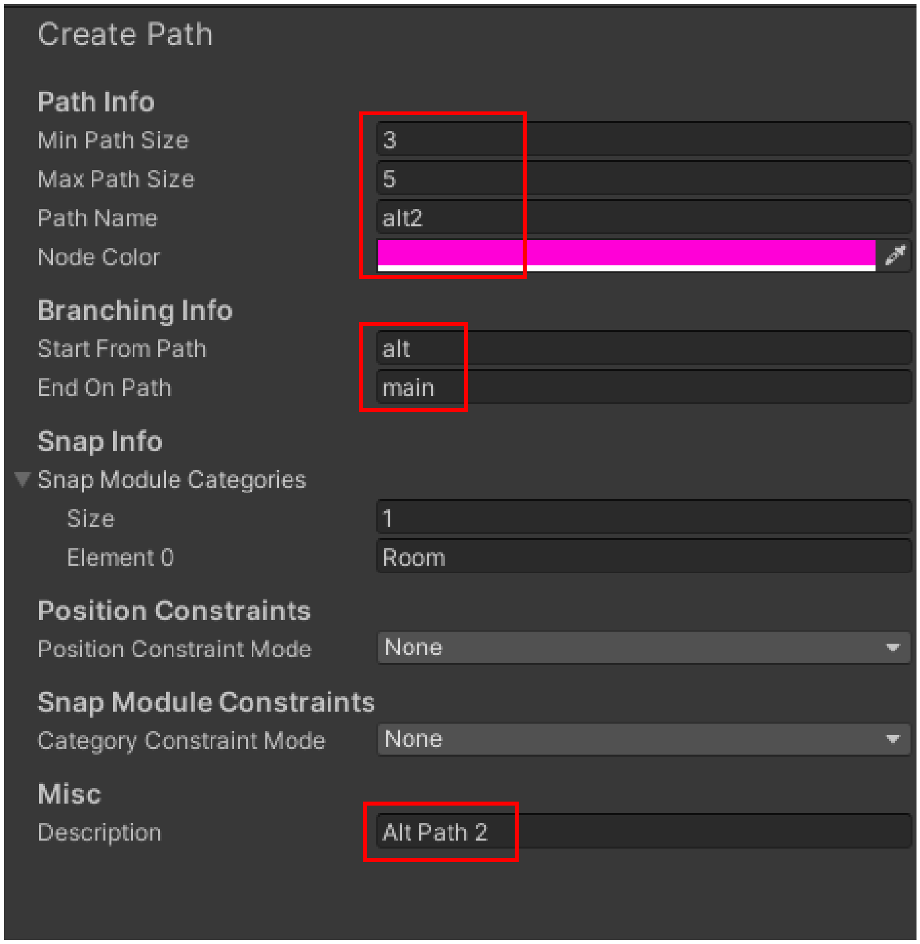
Add a description to this node



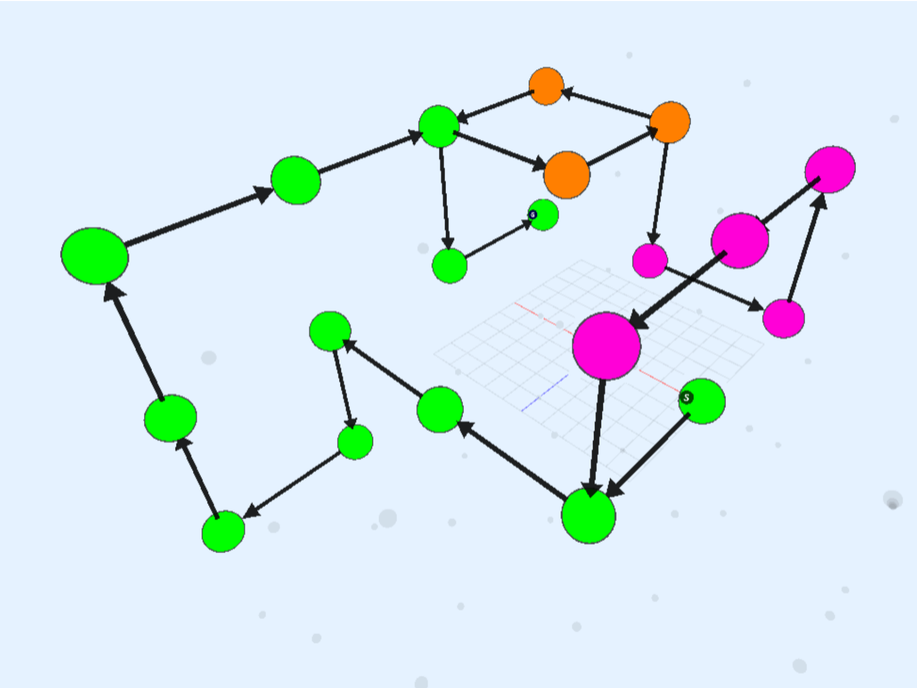
It’s a good practice to add description to your nodes, as it becomes easier to manage them when you graph grows bigger

Create Another Path

We’ll create another path originating from the ‘alt’ path and merging back into the ‘main’ path



The path originates out of the `alt` path (orange) and merges beack into the `main` path (green)

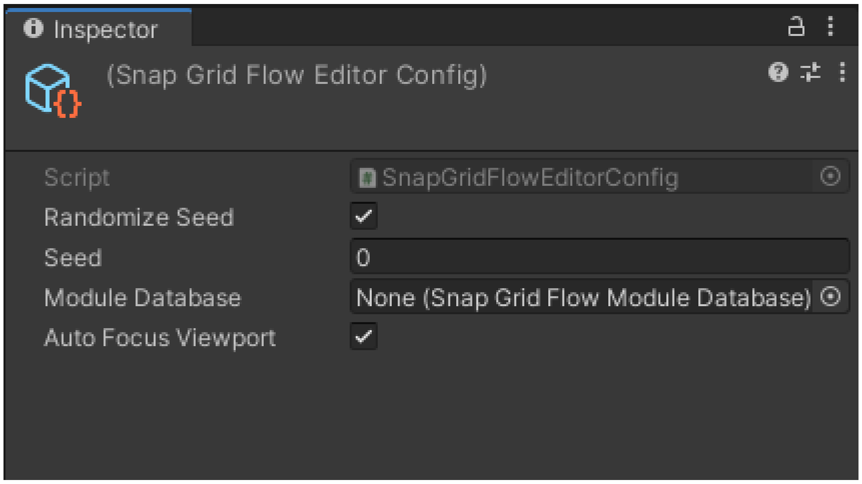


Assign Module Database

The flow editor doesn't have a module database assigned, so it doesn't really know how our modules look like

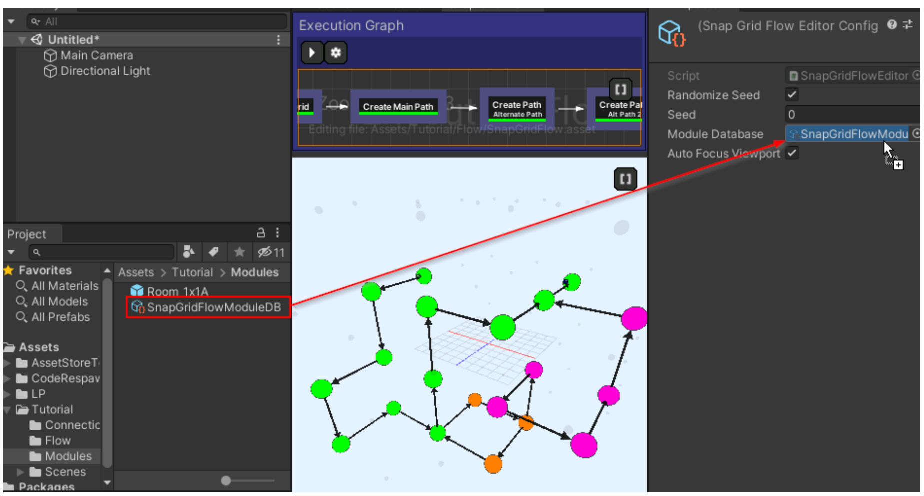
Let's assign the module database in the editor settings, so it creates a layout graph compatible with how we've set up the modules

Click an empty area in the Execution Flow Graph to view the editor properties

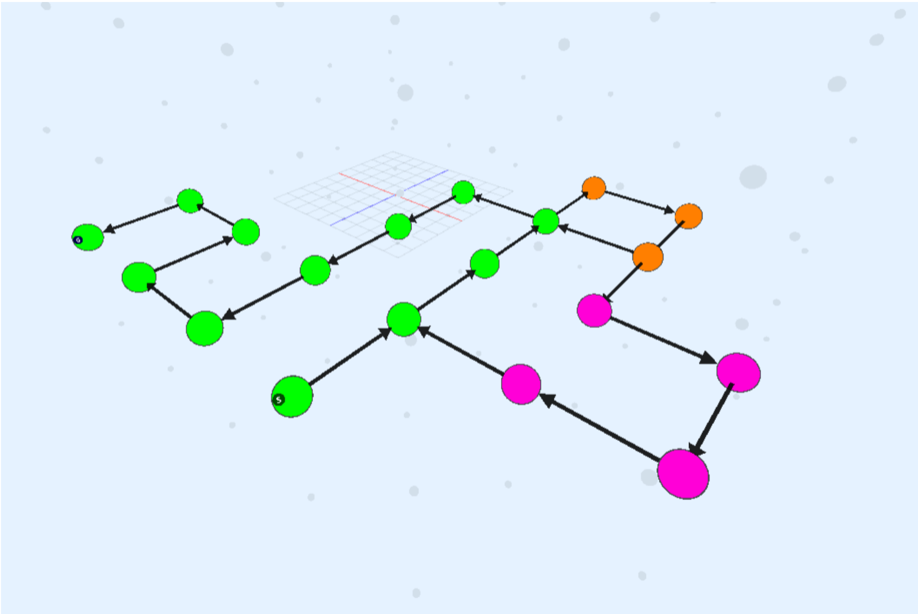


Property | Description Randomize Seed | Randomize the layout everytime you build Seed | The current seed that was used to build the dungeon Module Database | Specify a module database asset to build the flow graph compatible with the regsitered module prefabs Auto Focus Viewport | Whenever you select a node in the execution graph, the camera auto-focuses on all the active nodes. Uncheck to disable this

Assign the module database we created earlier



Rebuild and have a look at the layout graph now



You'll notice that the flow graph is built on a single floor, which is consistent with the way we've designed our registered modules. We have designed a room which, although it goes out in the four horizontal directions, there's no way of moving up or down.

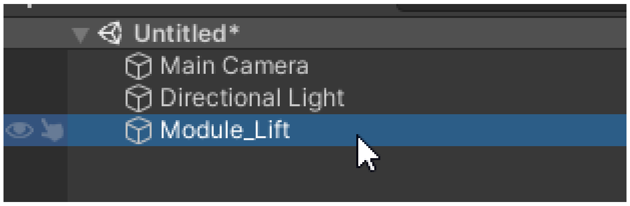
If your build fails, make sure you've compiled your module database and saved the asset

Create a Lift Module

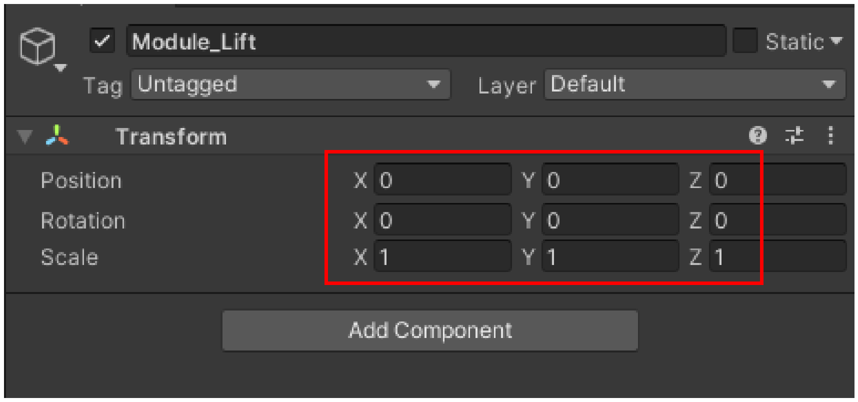
Let's create a module that lets us move to another floor

Design Lift Module Create a new module as we've done in the previous Modules section

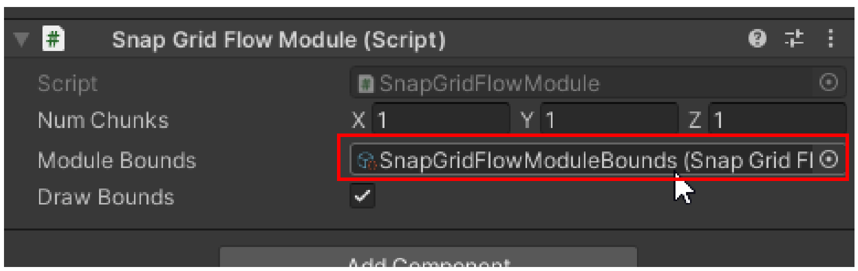
1. Switch to a new scene and create a new game object and name it `Module_Lift`



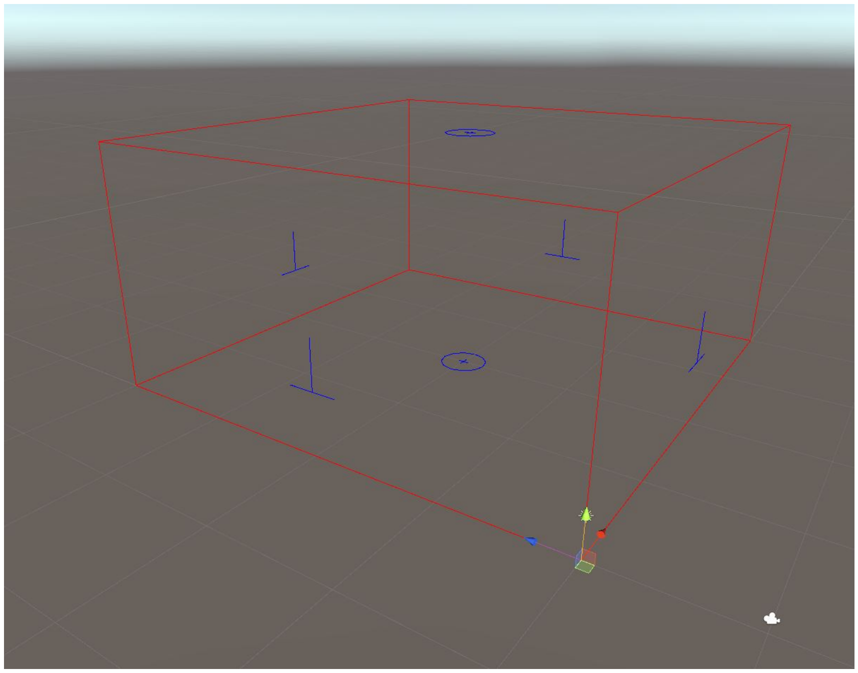
2. Reset the transform



3. Add a `Snap Grid Flow Module` component and assign the module bounds asset

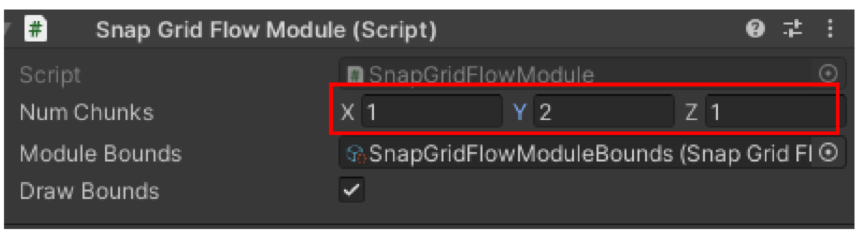


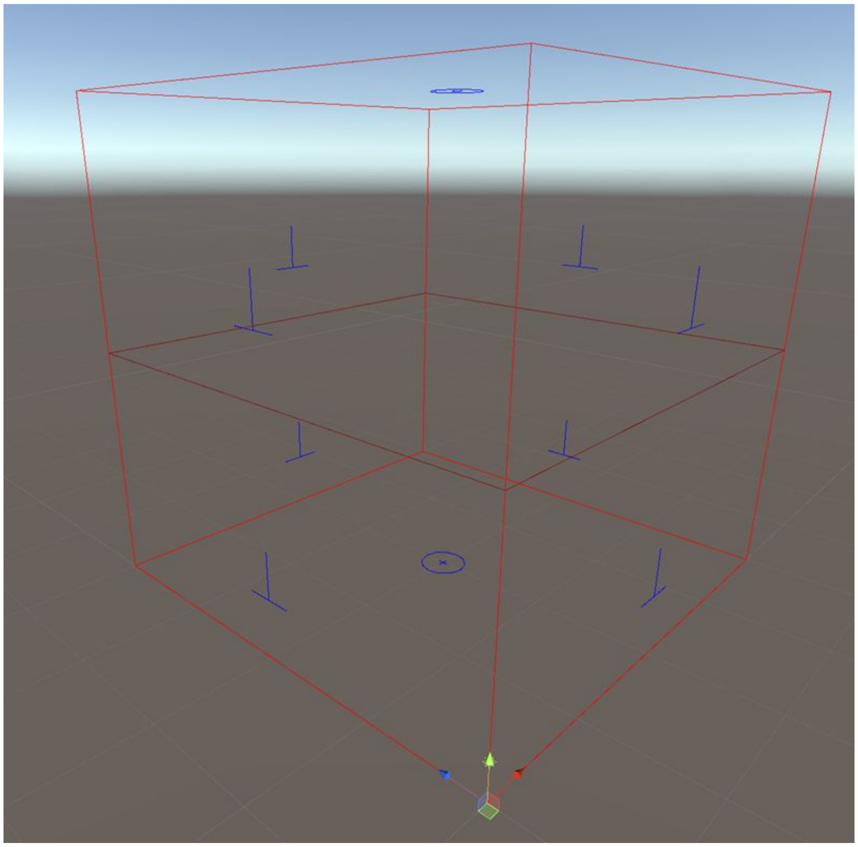
You should see the bounds of the module



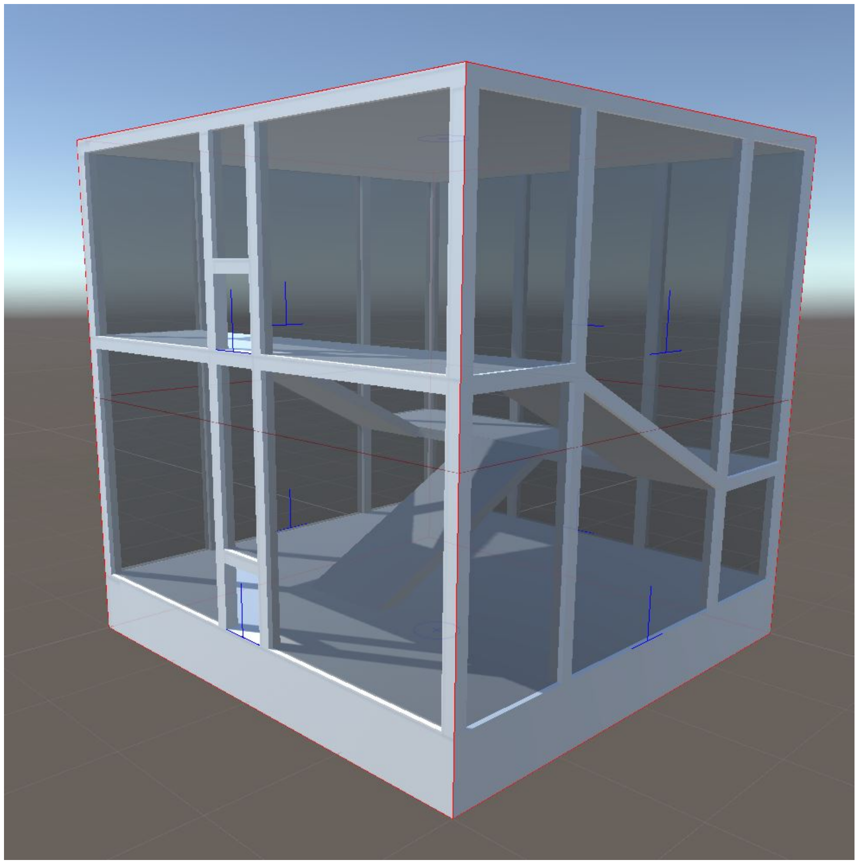
4. We want the module to span two node vertically

   Set the Num Chunks parameter to `(1, 2, 1)`

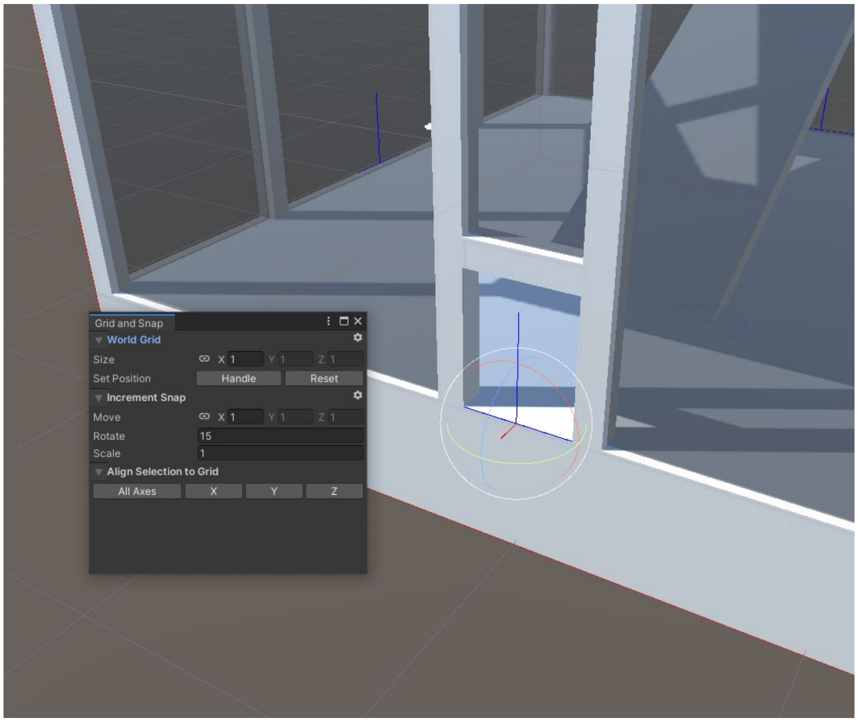


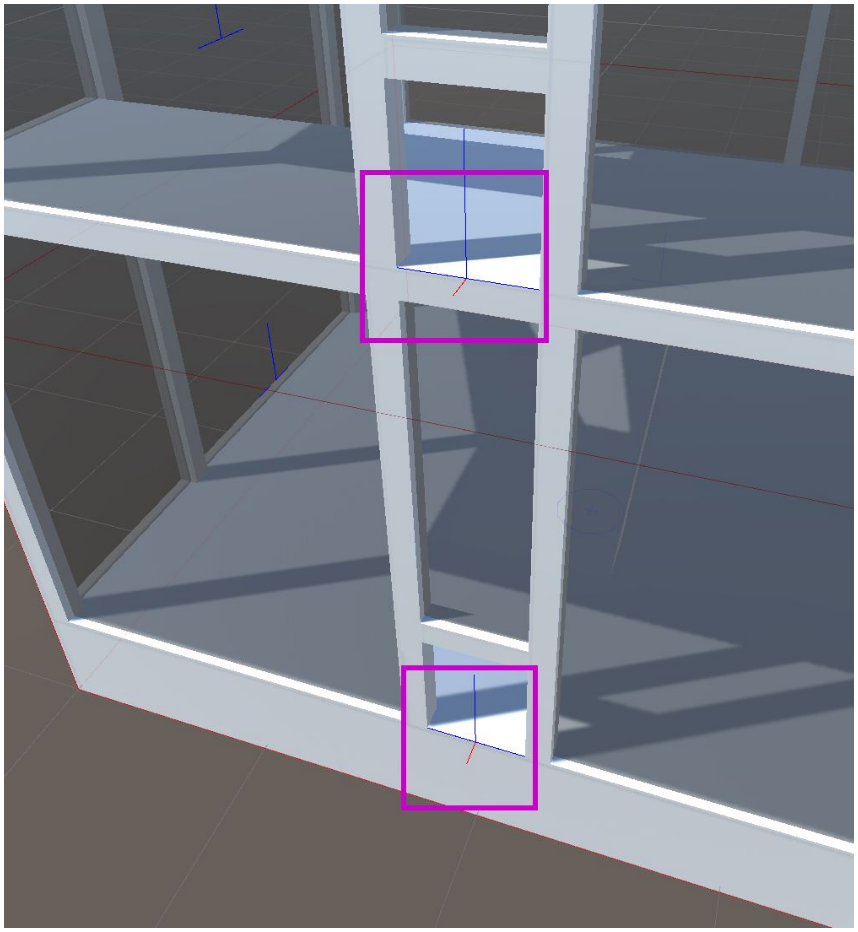


5. Go ahead and design your lift module in any way you like. Leave two openings on the same side of the room, one below and another on top

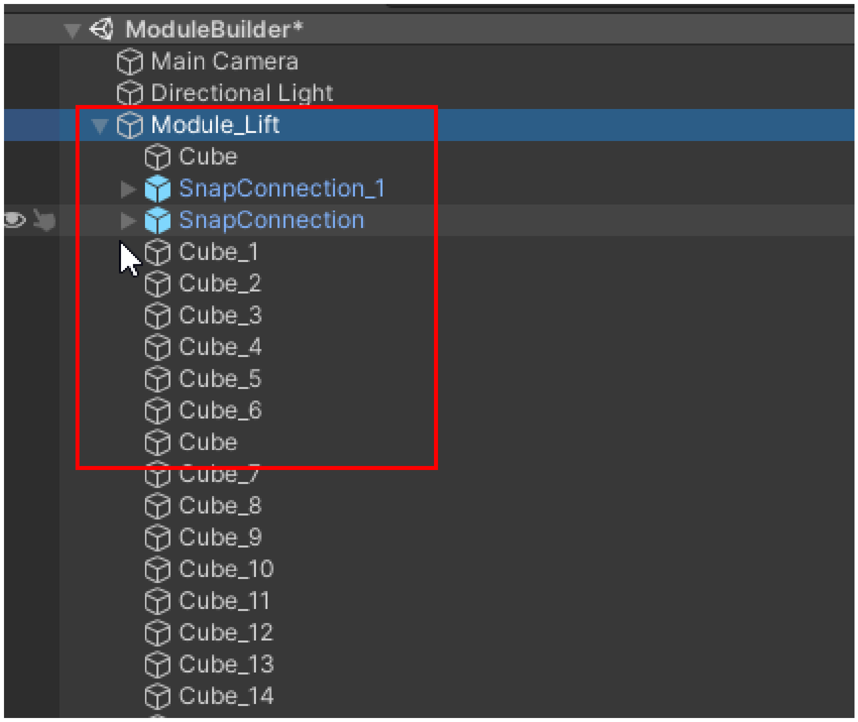


6. Drop in two snap connection prefabs and make sure they are facing outwards. Align them correctly using the `Grid and Snap` window

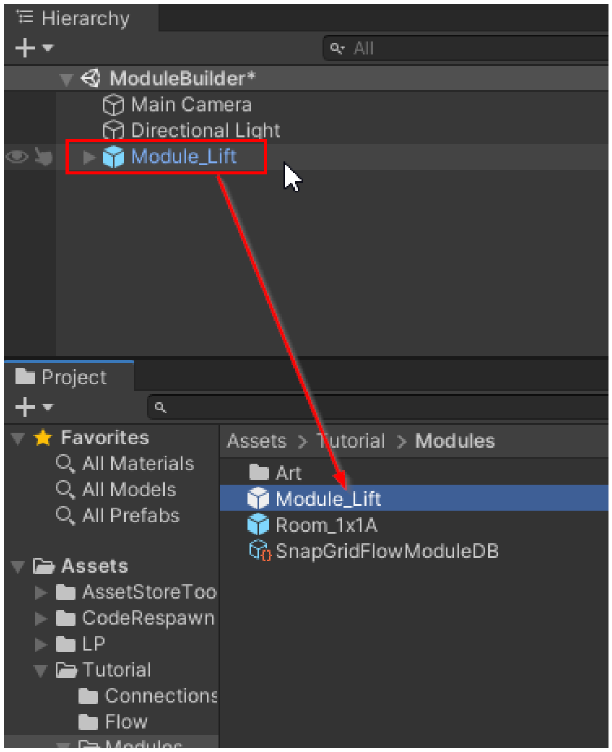

Grid and Snap
World Grid
Size
X 1
Y 1
Z 1
Set Position
Handle
Reset
Increment Snap
Move
X 1
Y 1
Z 1
Rotate
15
Scale
1
Align Selection to Grid
All Axes
X
Y
Z



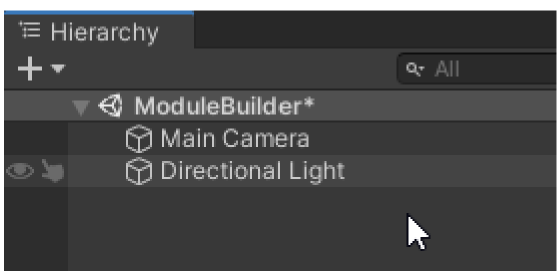
7. Make sure all the objects that make up this room are inside the module game object



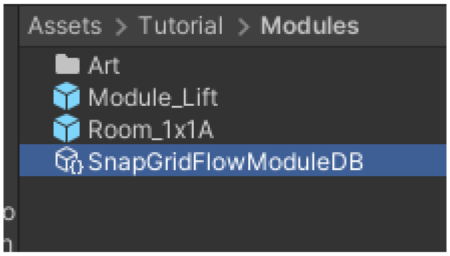
8. Save this module lift as a prefab

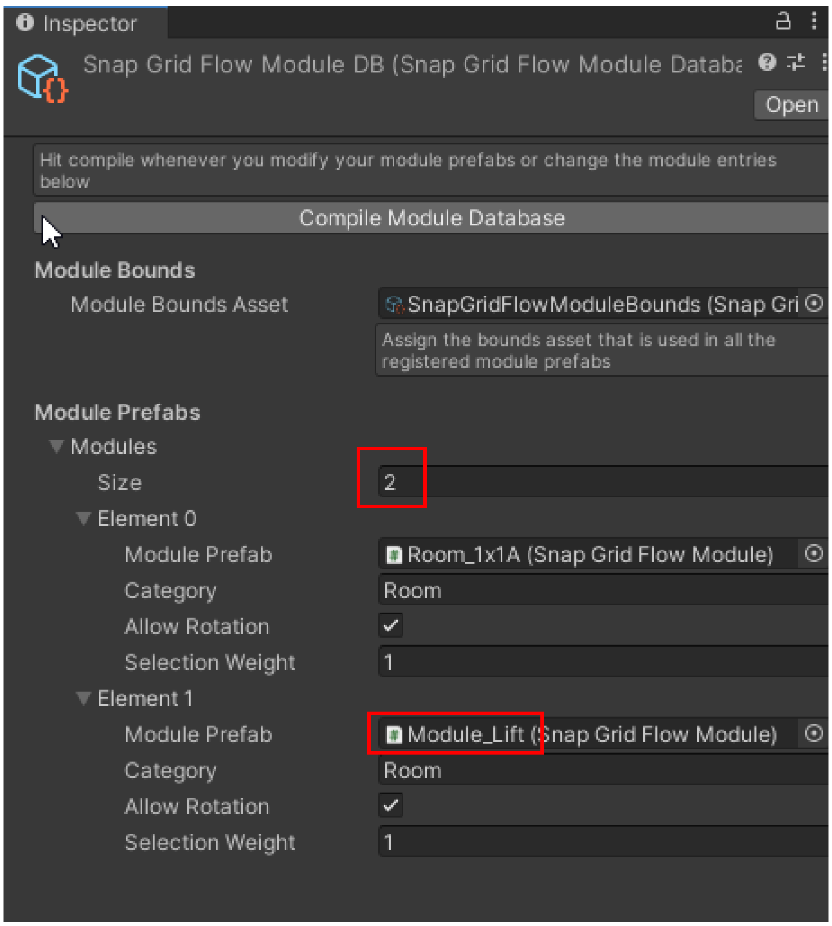


9. Delete the Module_Lift game object from the scene as we no longer need it

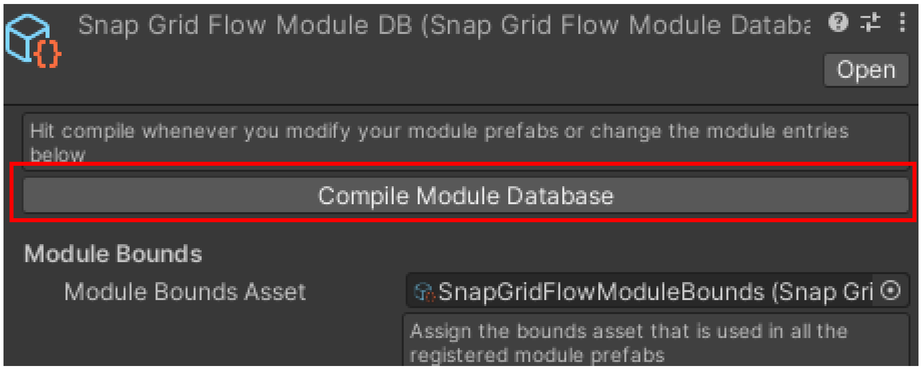


Register Lift Module Register this module in the Module database. Select the module database asset and in the properties, register our new lift module

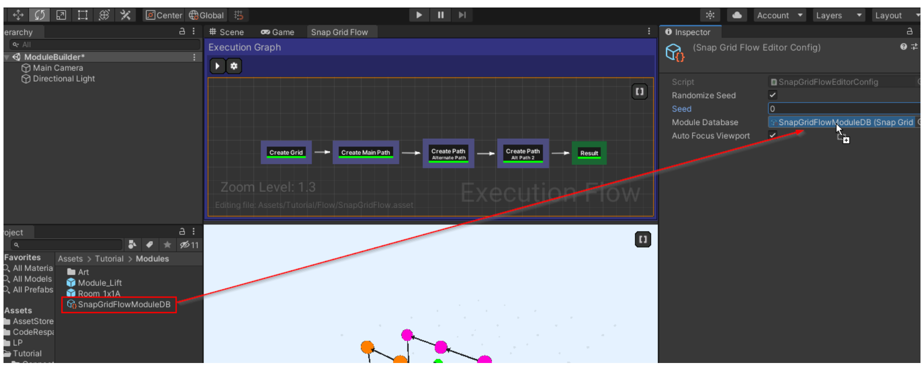




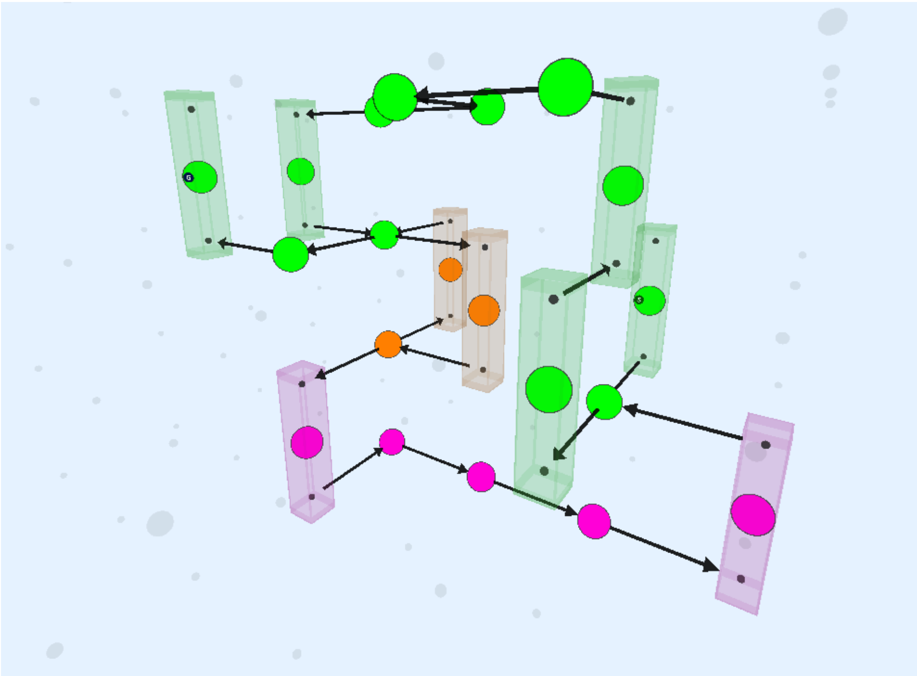
Since we've modified the module database, hit `Compile Module Database` and save the asset

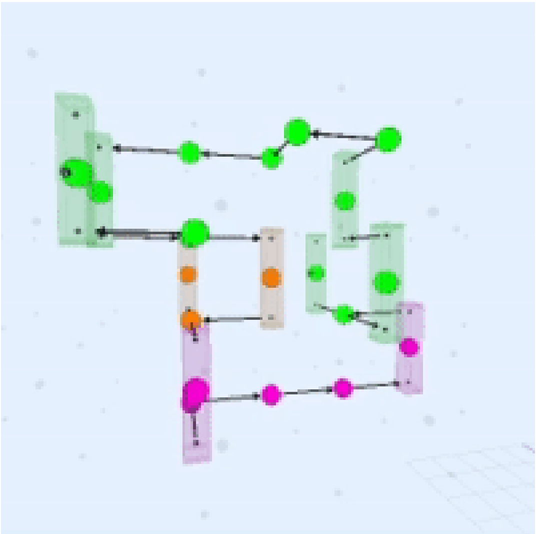


Open up our flow graph editor and reassign the Module database in the Editor Settings, as we've done previously

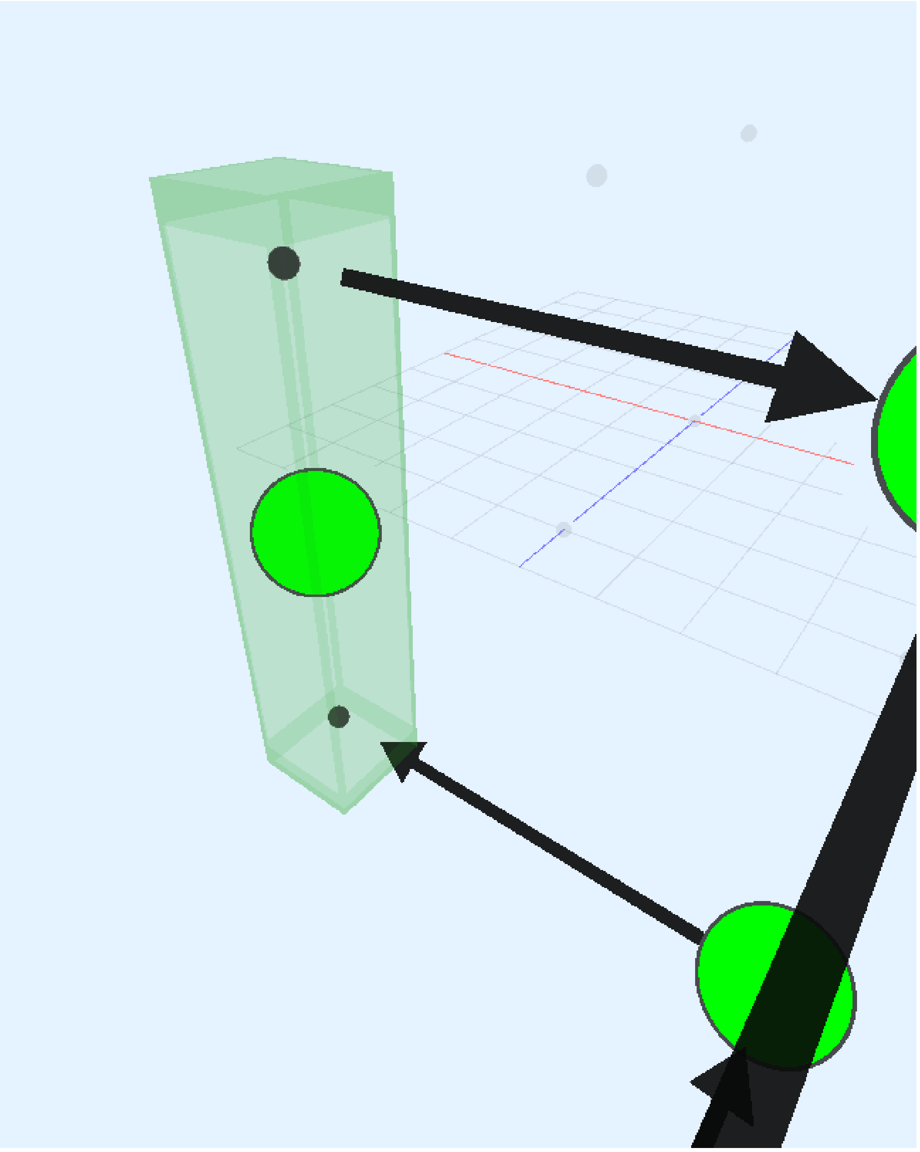


Click build and now, the flow graph uses the new lift module to move to other floors

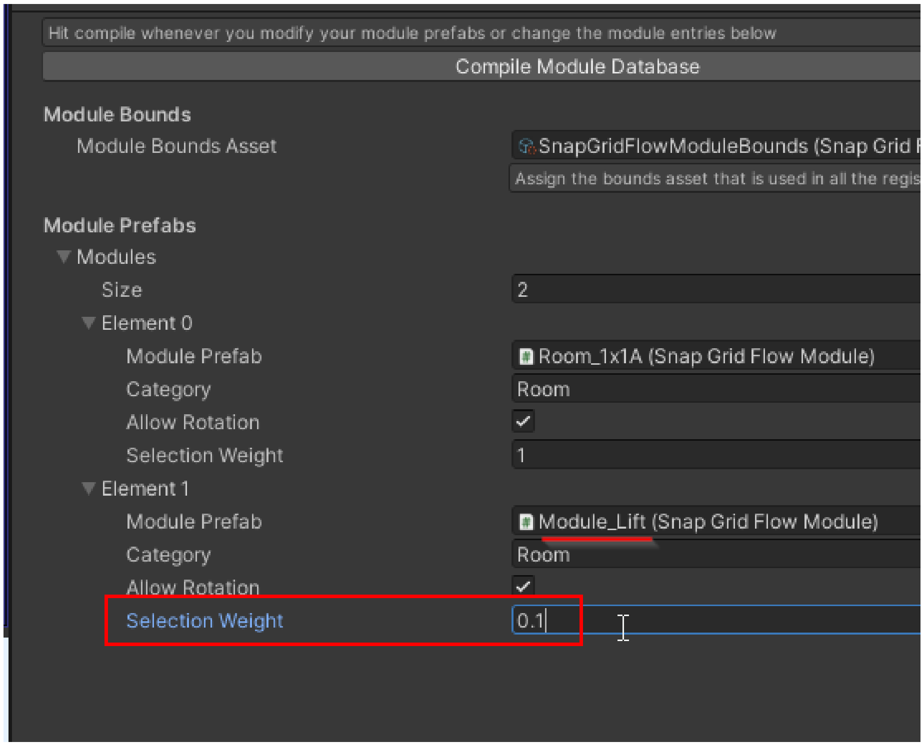




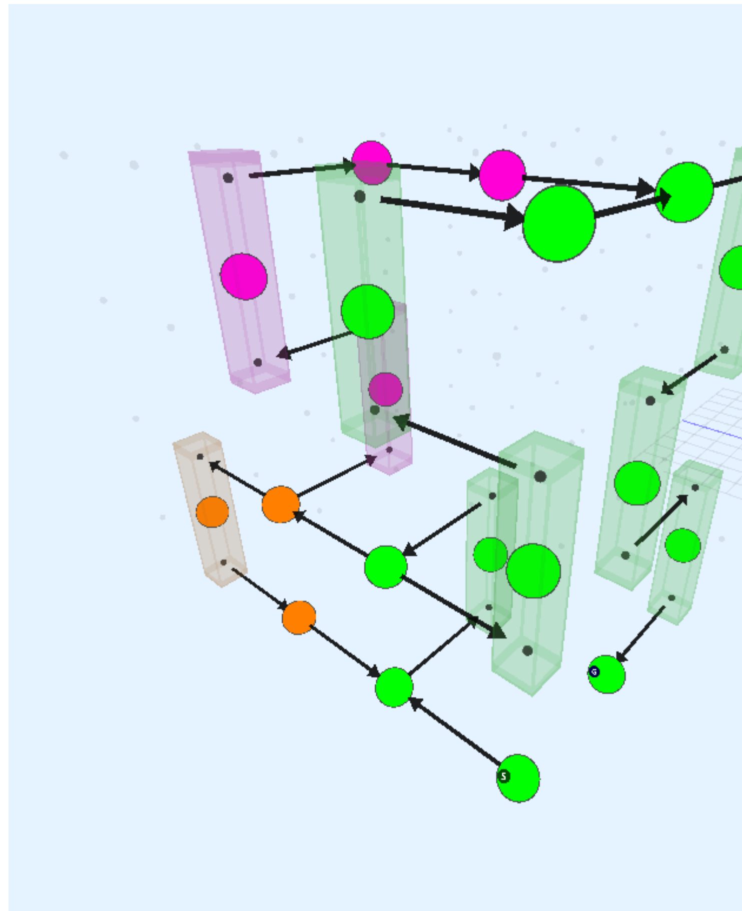
Notice how the links enter and exit out of the same side, since this is how we've set it up in our module design



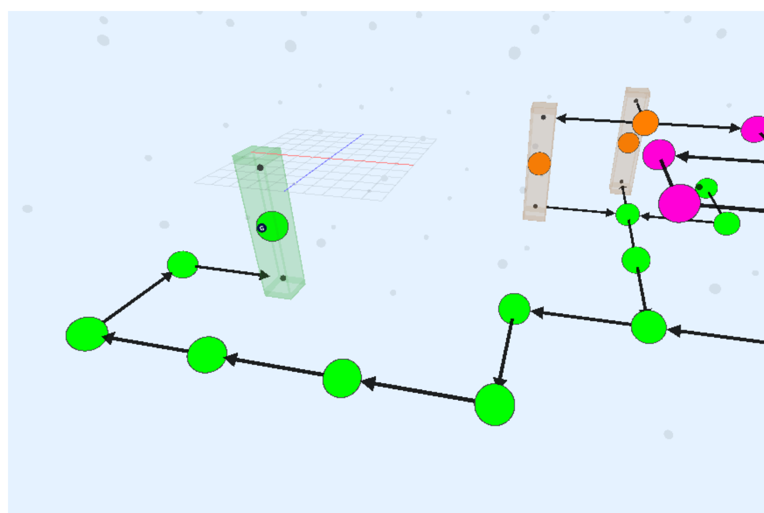
Adjust Selection Probability We don't want the lift to show up too often. Open up the Module database and set the selection weight to `0.1` on the lift module entry



Hit compile whenever you modify your module prefabs or change the module entries below
Compile Module Database
Module Bounds
Module Bounds Asset
SnapGridFlowModuleBounds (Snap Grid
Assign the bounds asset that is used in all the regis
Module Prefabs
Modules
Size
2
Element 0
Module Prefab
Room_1x1A (Snap Grid Flow Module)
Category
Room
Allow Rotation
Selection Weight
1
Element 1
Module Prefab
Module_Lift (Snap Grid Flow Module)
Category
Room
Allow Rotation
Selection Weight
0.1



Before (with `selection weight 1.0`):



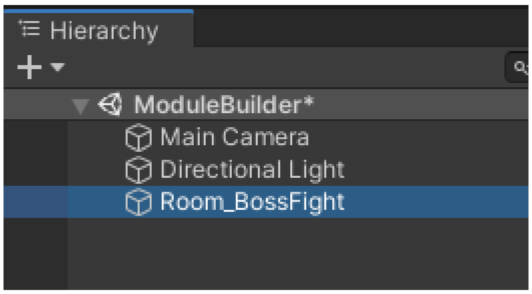
After (with `selection weight 0.1`):

Create a Goal Room

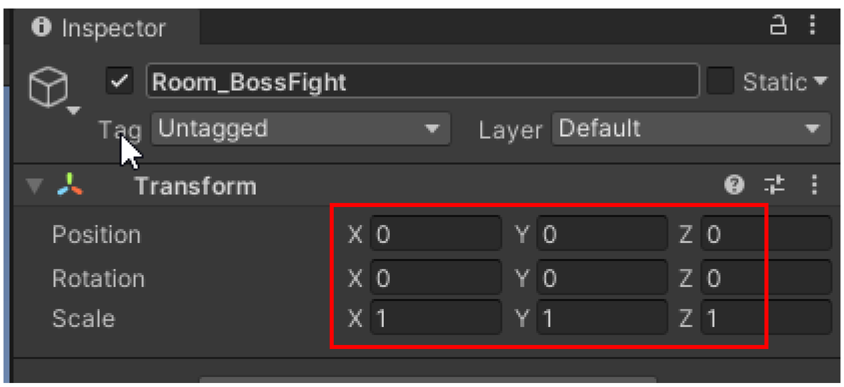
We'll create a large 2x2x2 room for the boss fight and register it under the category `Boss`.

Design the Goal Room Prefab

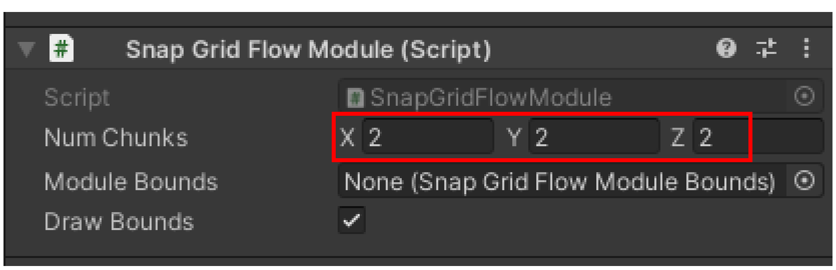
1. Open a new empty scene, create an empty game object and name it `Room_BossFight`



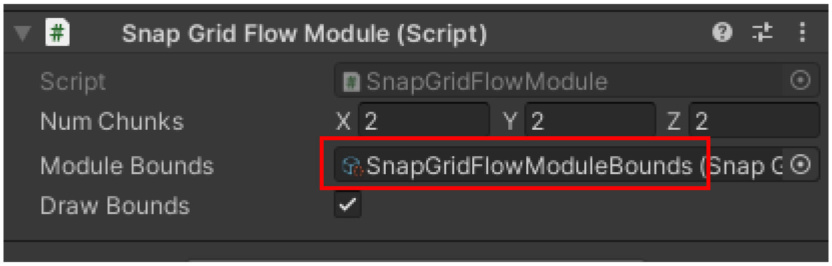
2. Reset the transform



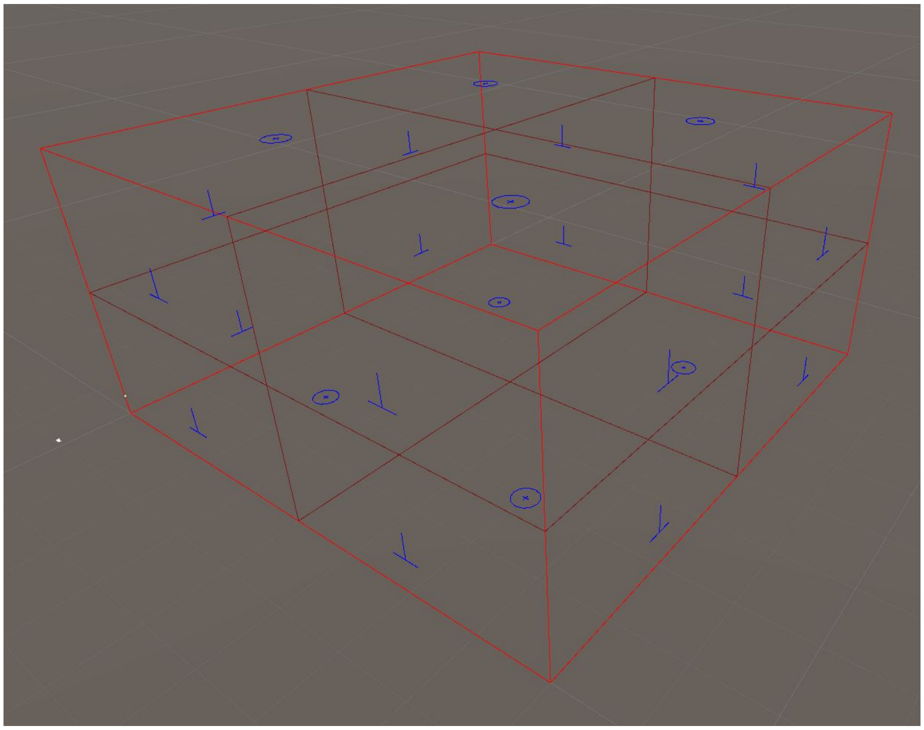
3. Add a `Snap Grid Flow Module` component and set the num chunks to `(2, 2, 2)`



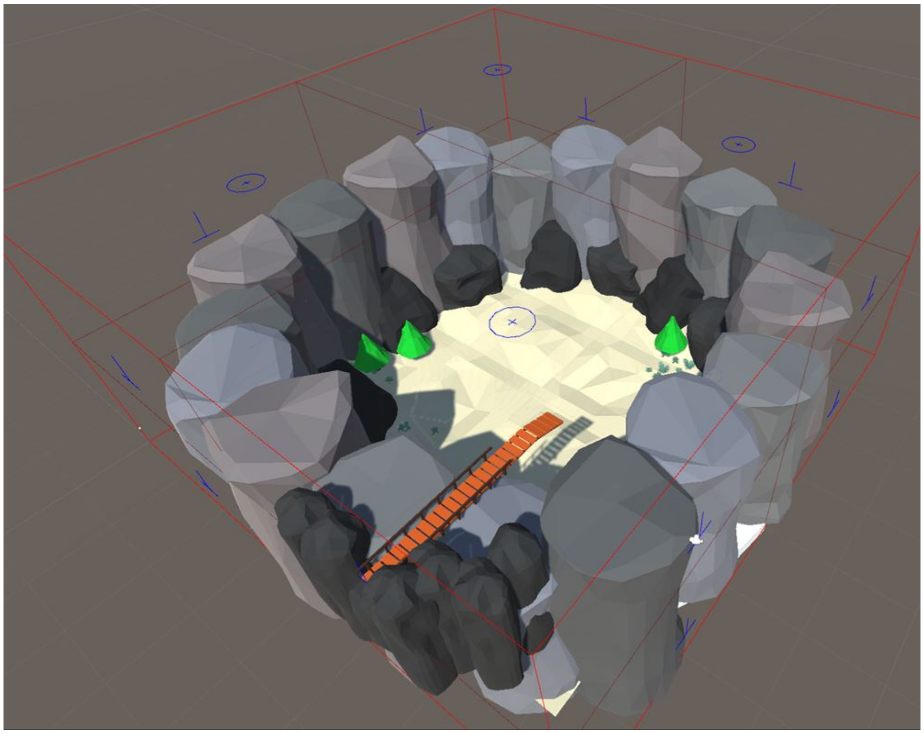
4. Assign the module bounds



You should now see a the bounds visuals in the scene view. You have a large area to design your boss fight arena

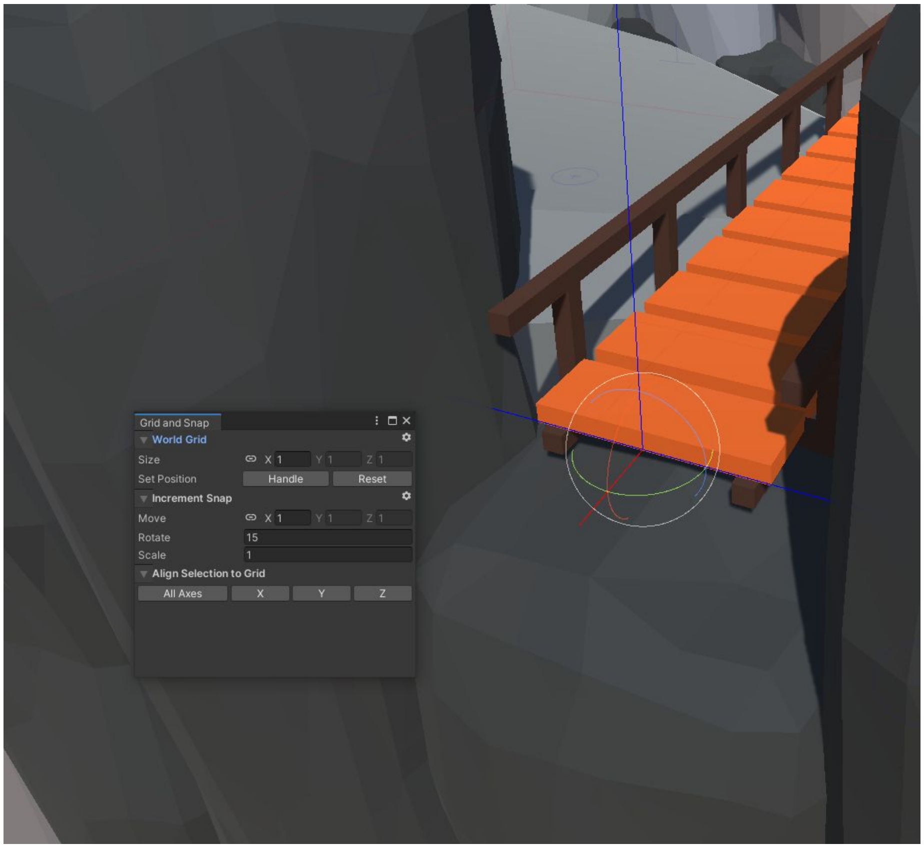


5. Design the module in any way you like.

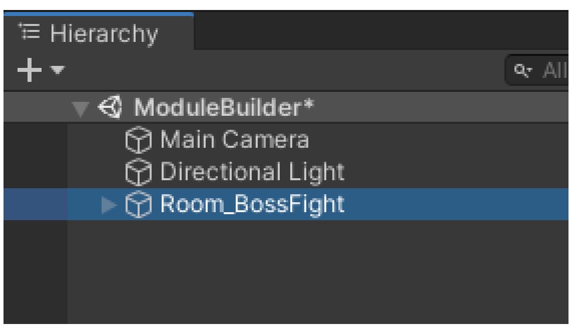


In our example, we'll keep only one opening on the top floor, and the player falls down to the arena to fight the boss

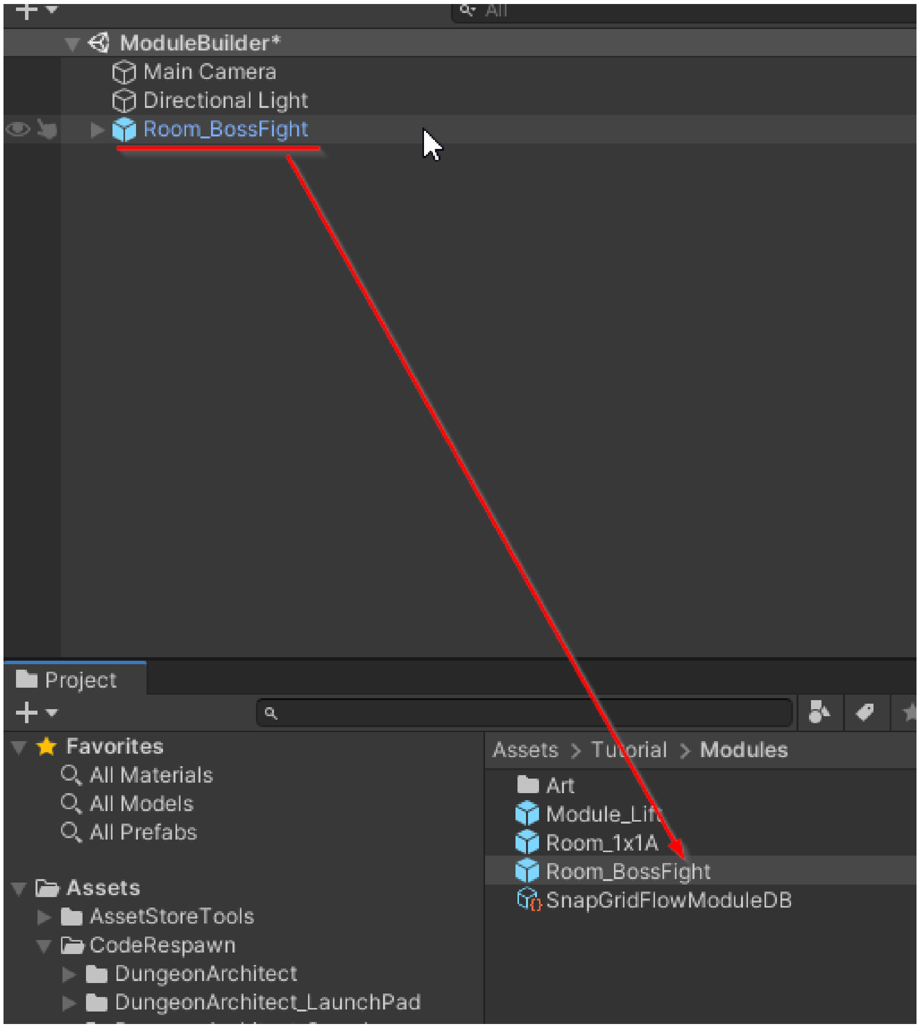
6. Add the snap module near the opening and snap it to the correct position



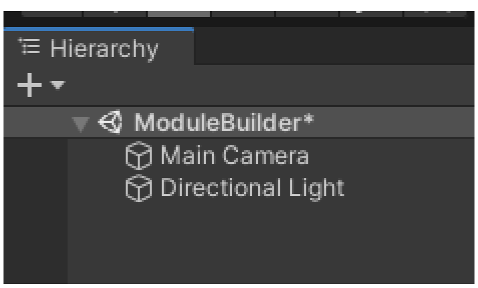
7. Make sure all your objects are inside the module prefab



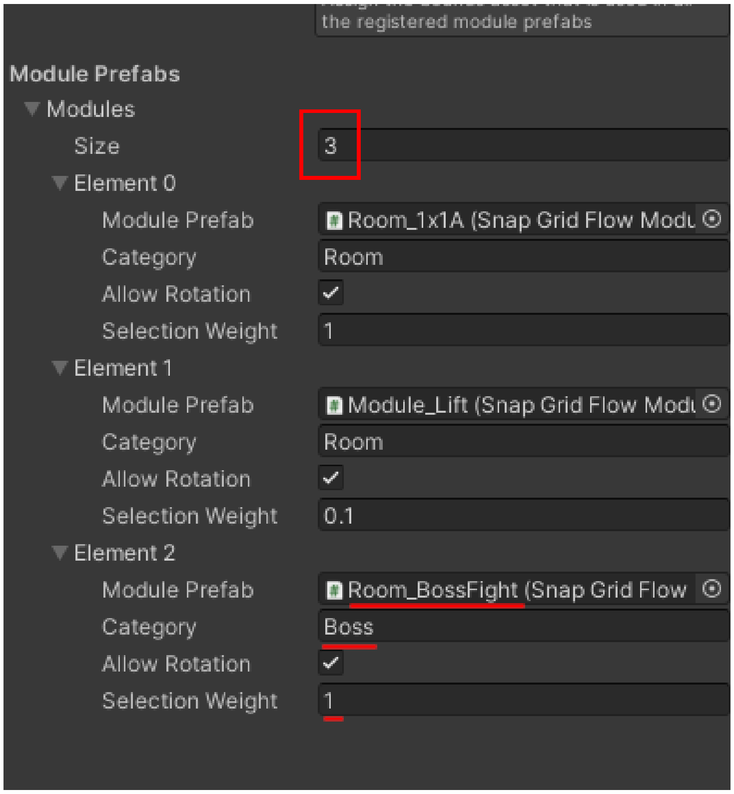
8. Turn this into a prefab



9. Delete the module game object from the scene



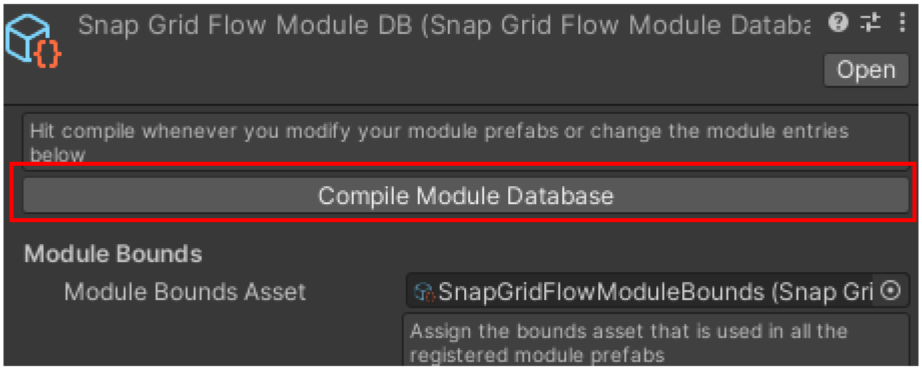
Register the Module Register this module in the `Module Database` with the category `Boss`



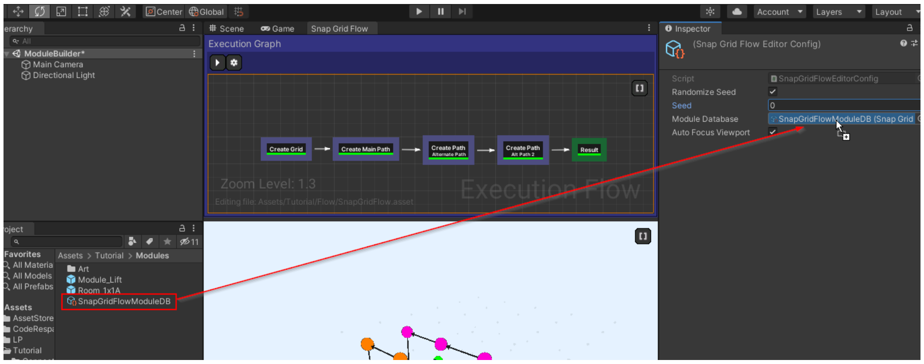
Make sure the following parameters are set

Parameter | Value Module Prefab | Room_BossFight Category | Boss Selection Weight | 1

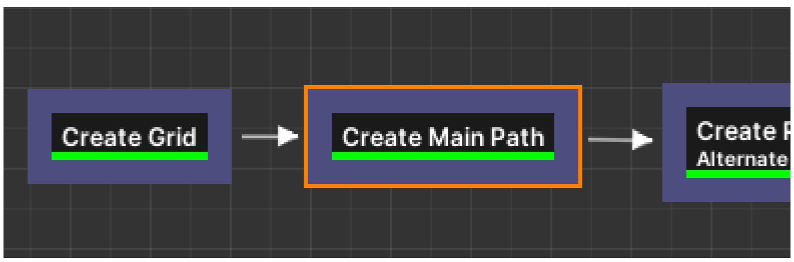
Since we've modified the module database, hit `Compile Module Database` and save the asset

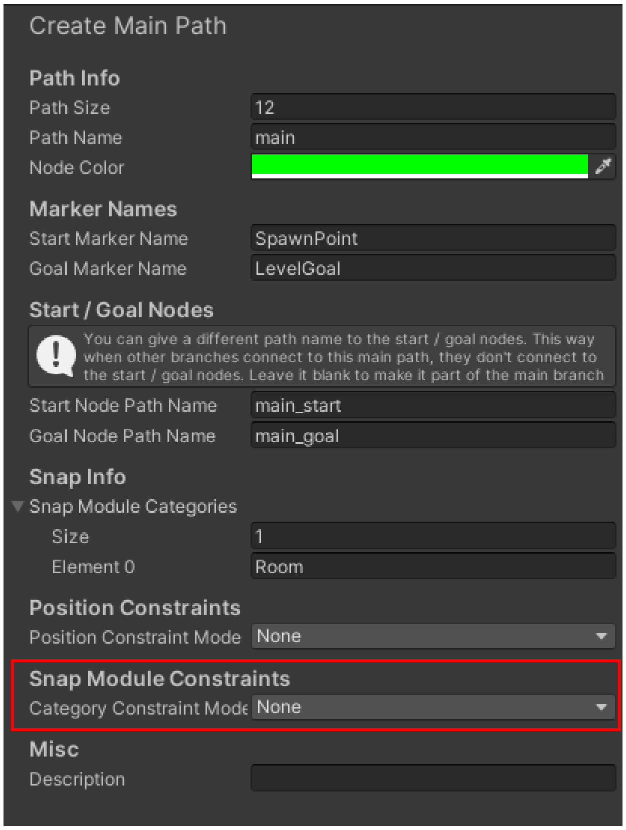


Open up our flow graph editor and reassign the Module database in the Editor Settings, as we've done previously



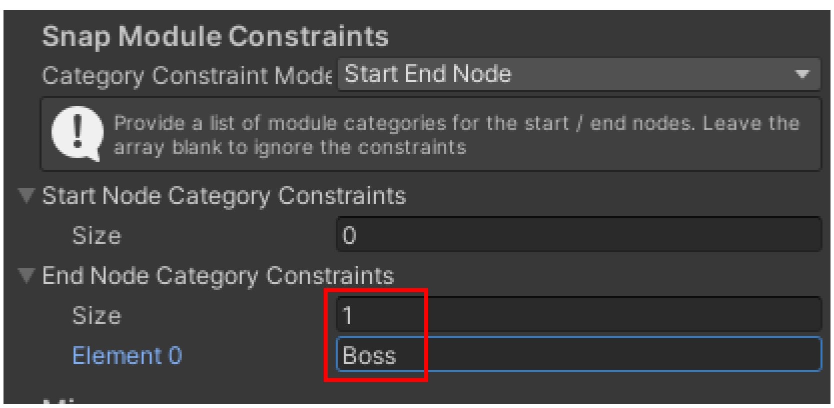
Select the `Create Main Path` node and inspect the properties in the Details panel





Change the `Category Constraint Mode` to `Start End Node`

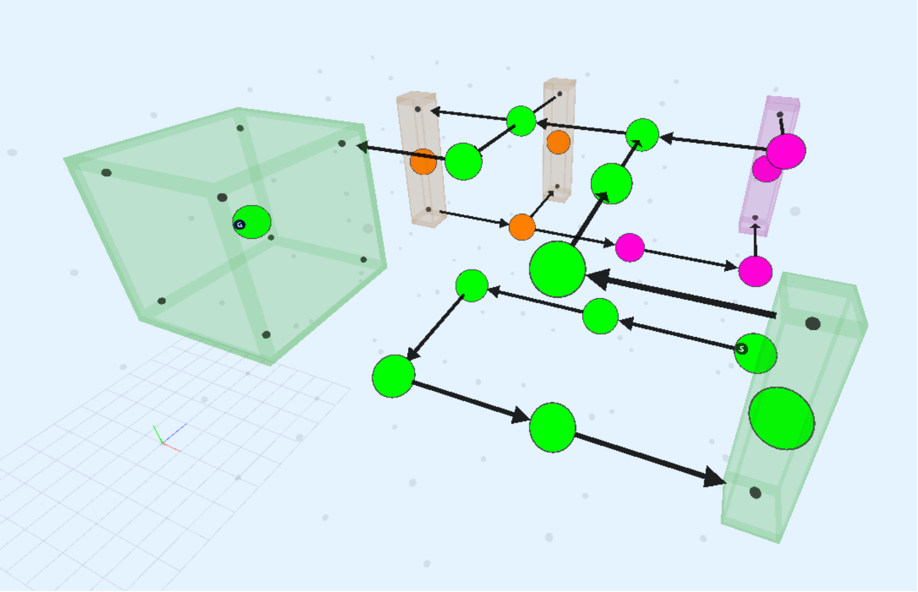
This allows you to override the category of the start and the end nodes.



This will force the flow system to pick up modules that are registered under the `Boss` category. We previously registered our boss room with this category

You can have more than one module registered under the same category. For example, you may have 3 boss rooms registered in the module database under the category `Boss` and it would randomly pick one while building it

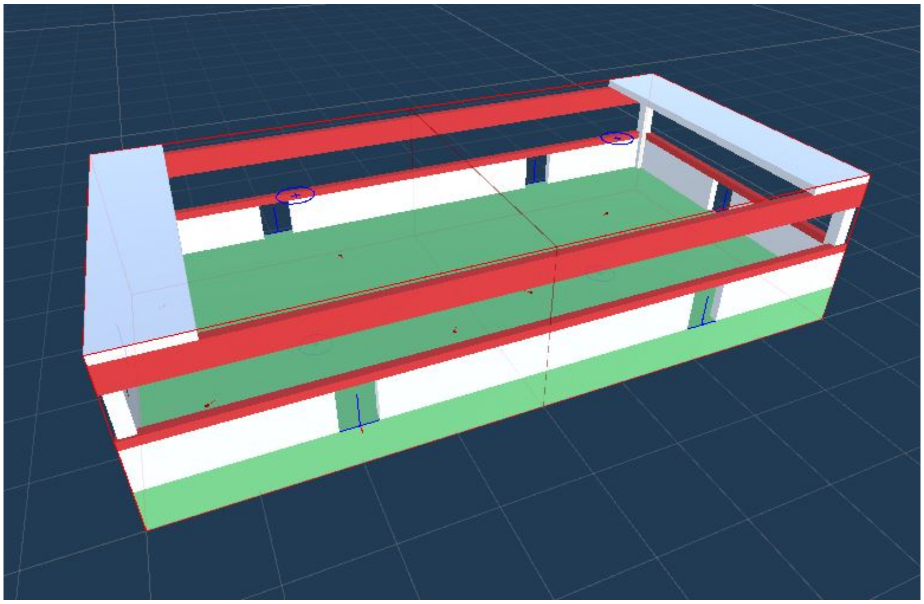
Build the flow graph and have a look at the layout graph

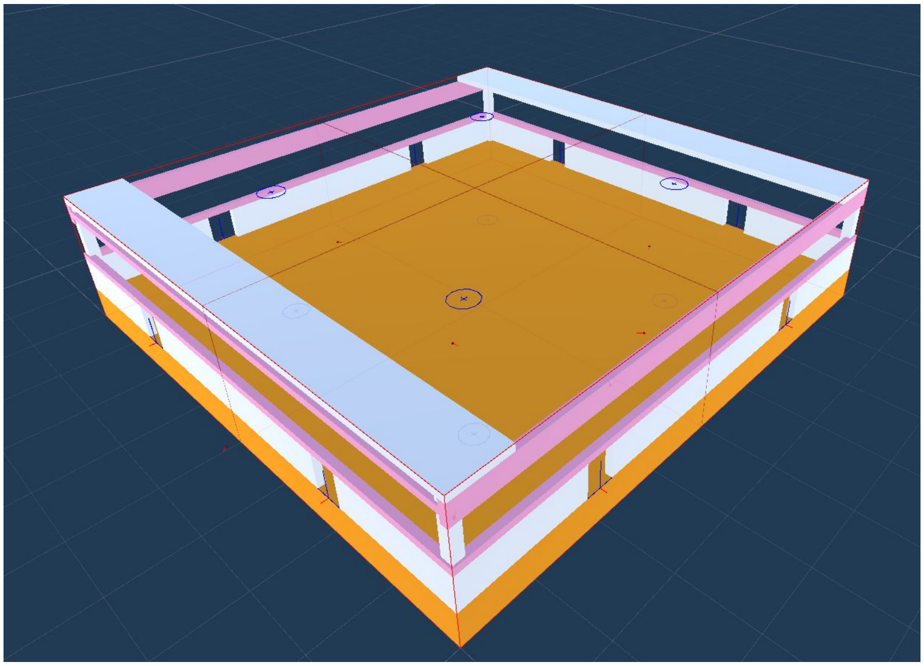


The last node in the main path has a size of `2x2x2` to accommodate our goal module. It also enters from the correct position

Create More Room Modules

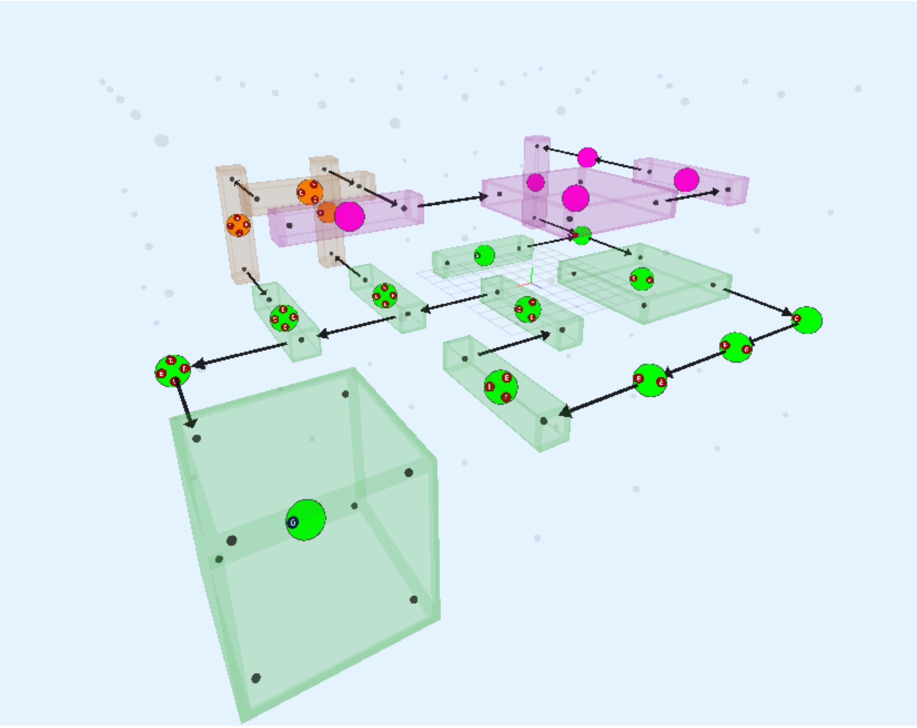
Add a few more `Room` modules of size `2x1x1` and `2x1x2`





Register them with the module database under the category `Room` and click `Compile Module Database`. Adjust their weights to control how often they appear

Rebuild the flow graph

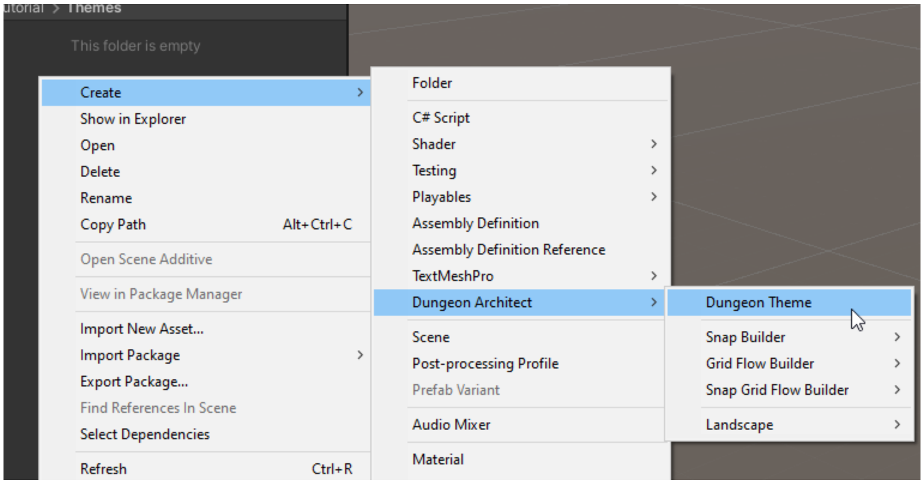


Build Dungeon

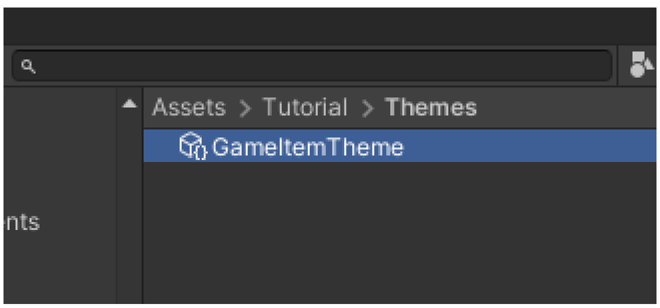
It’s time to use everything we’ve created to build a dungeon

Create Theme file

Create an empty theme file somewhere in the content browser. We’ll visit this later to spawn items in our modules (like NPCs, Spawners, Pickups, player prefab etc.)

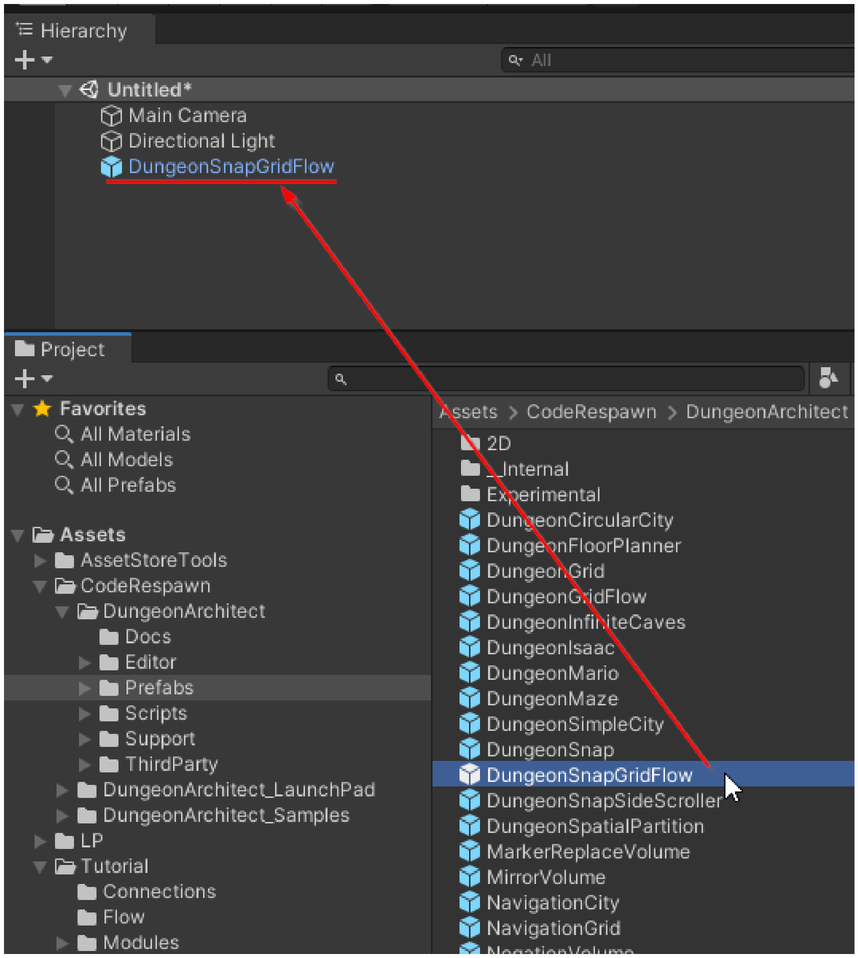


Rename it to `GameItemTheme`

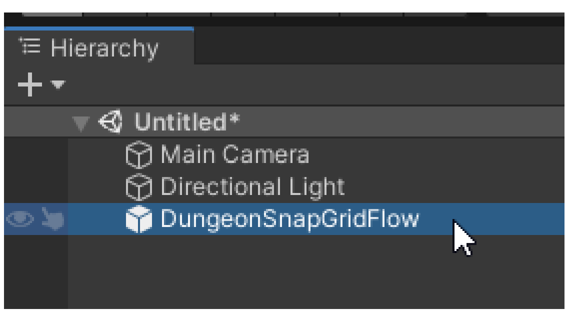


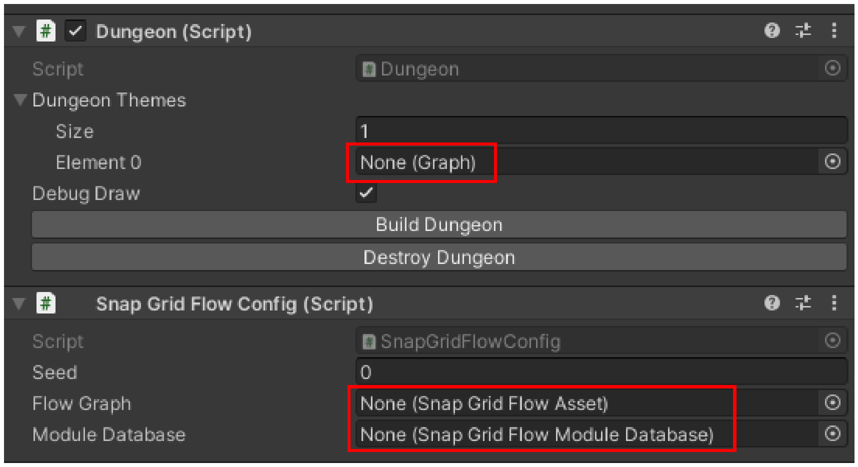
Setup Dungeon Game Object

1. Create a new scene and drop in a `DungeonSnapGridFlow` prefab from `Assets > CodeRespawn > DungeonArchitect > Prefabs`



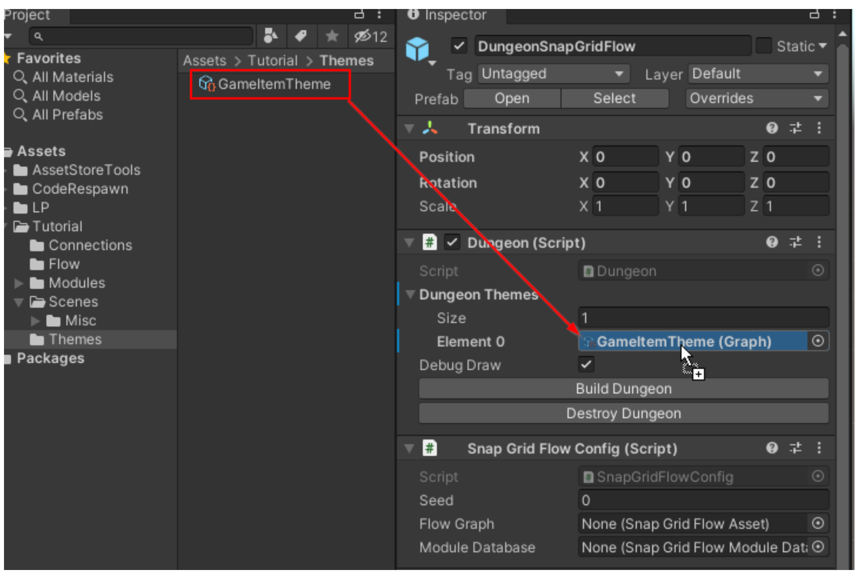
2. Select the `DungeonSnapGridFlow` game object and inspect the properties



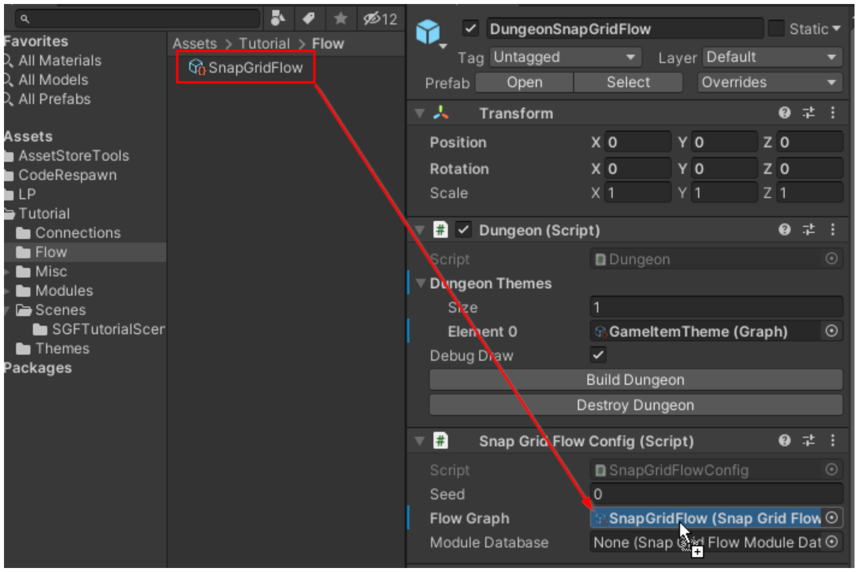


We'll assign the three assets we've created earlier

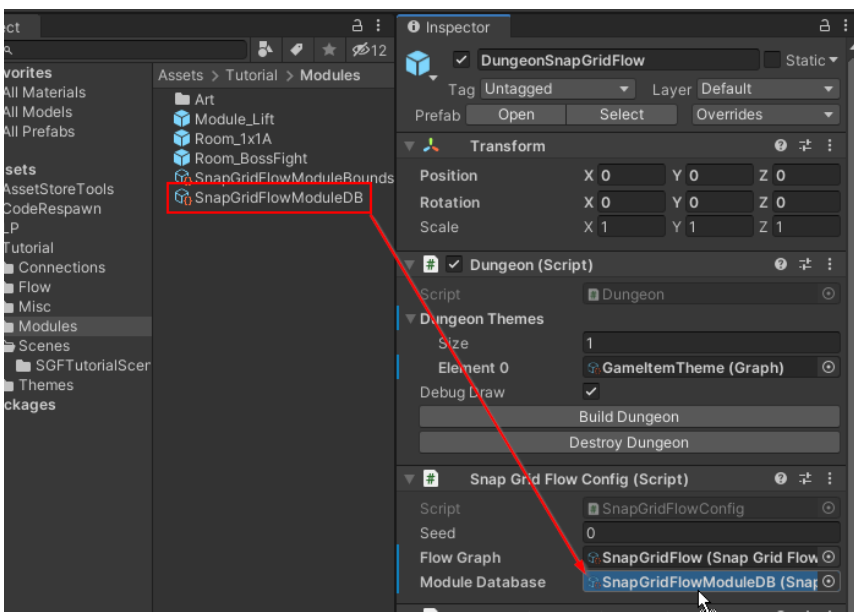
3. Assign the `GameItemTheme` create created above



4. Assign the Snap Grid Flow Graph

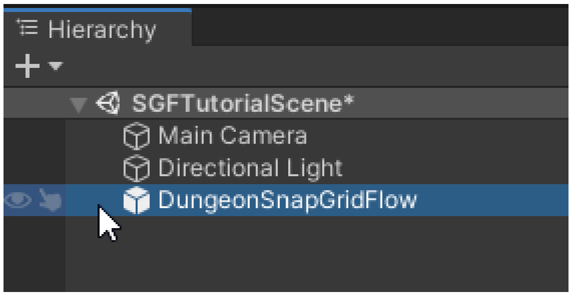


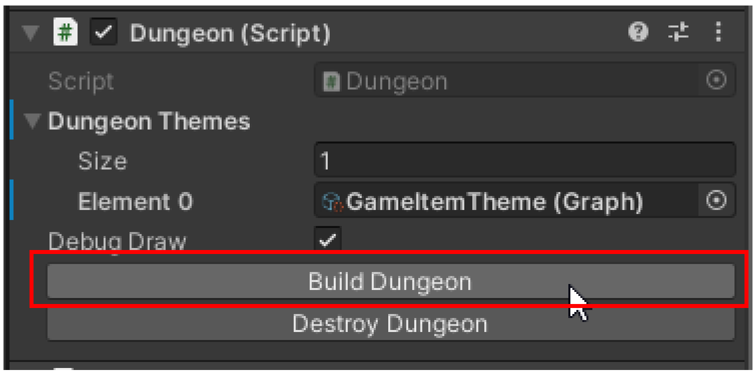
5. Assign the Module Database

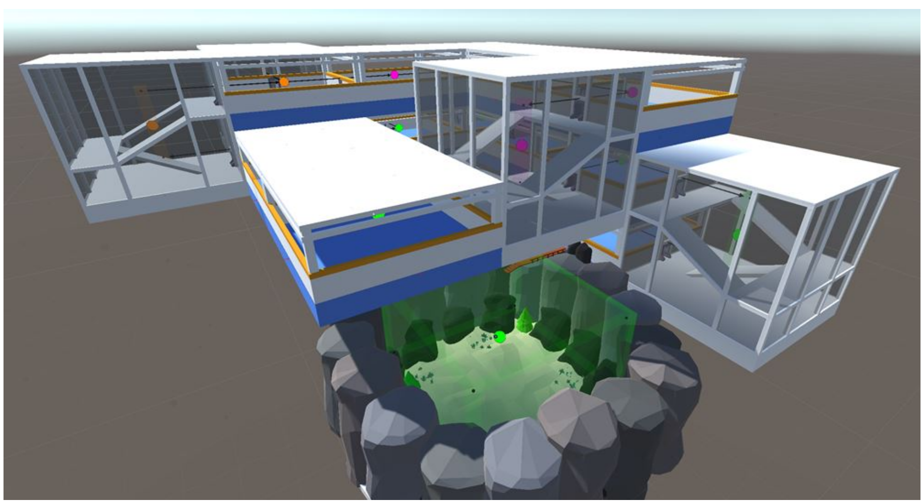


Build Dungeon

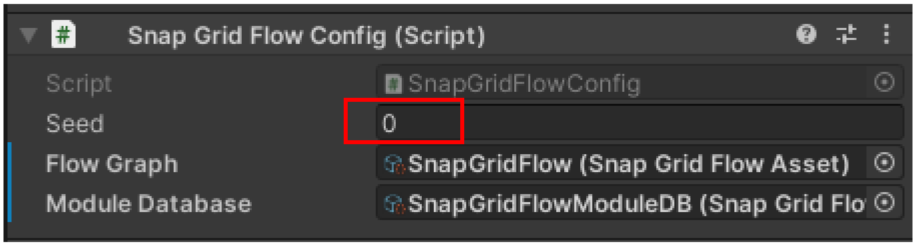
Select the `DungeonSnapGridFlow` game object and click `Build Dungeon`

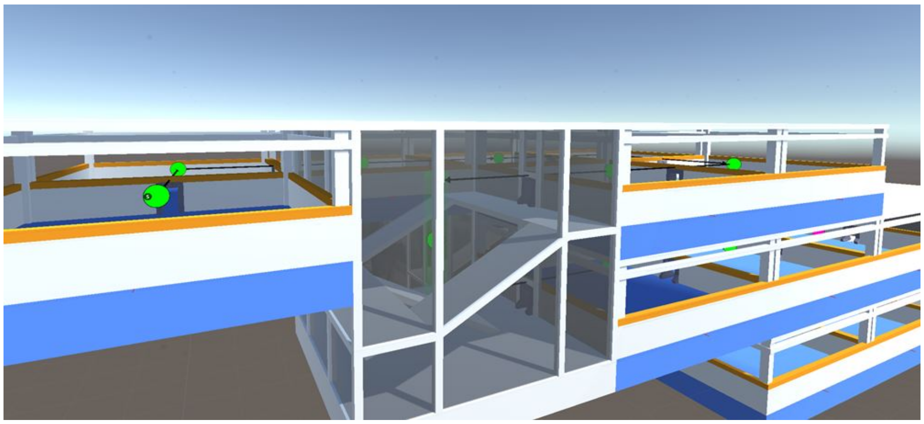






Change the seed and click build again to get a different dungeon

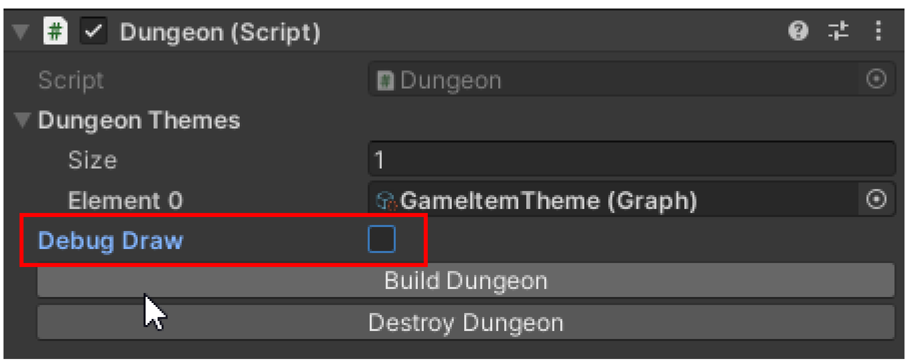


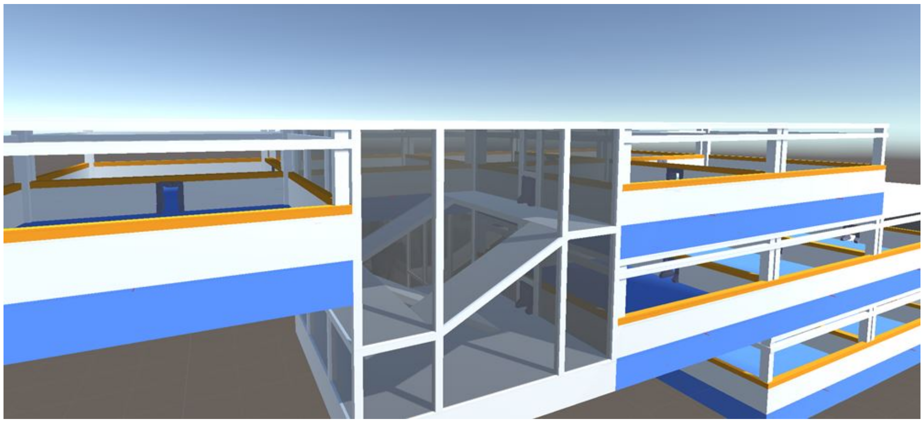


Debug Draw

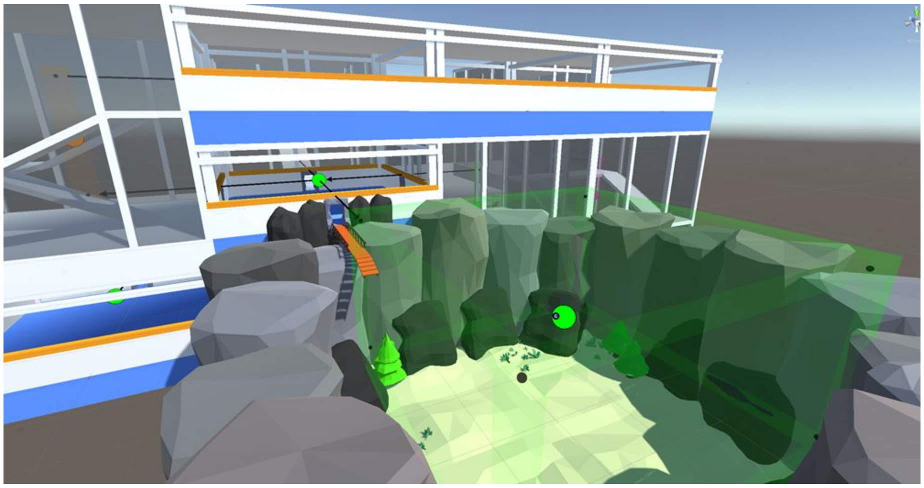
You get a debug overlay of the layout graph rendered by default when you build the dungeon. You'll want to turn this off in your final dungeon

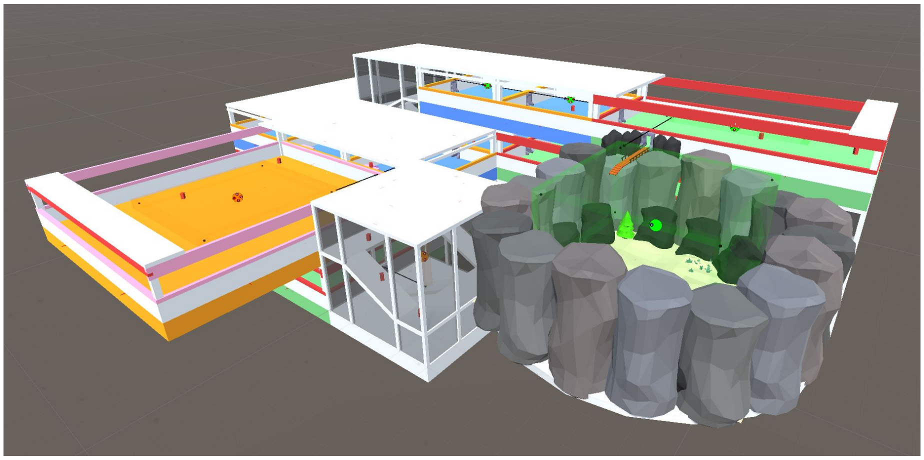
Do this by unchecking the `Debug Draw` check box and rebuild the dungeon





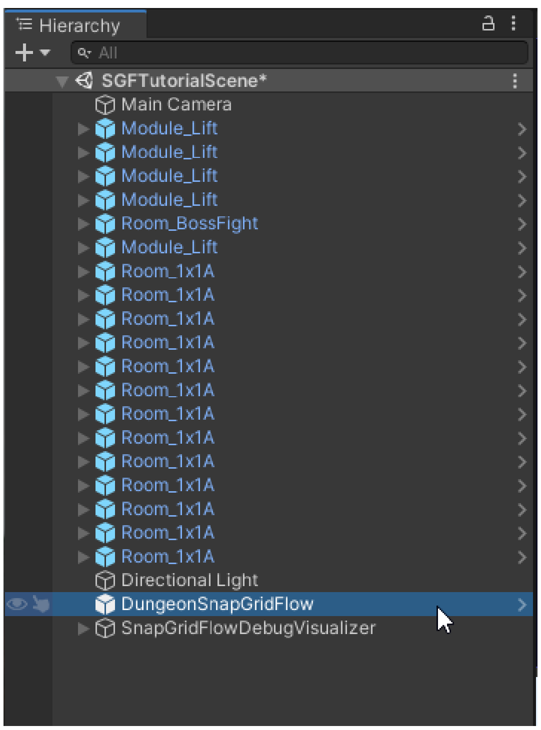
Let’s keep the `Debug Draw` check box on for now so we can see the layout graph overlayed in the scene





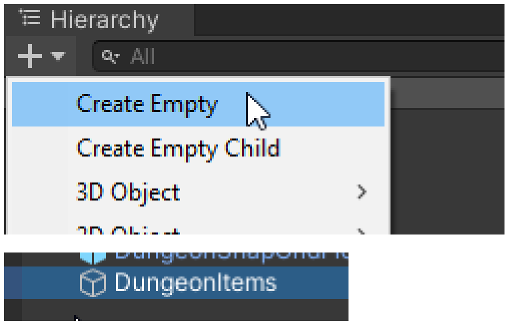
Keep Things Organized

When you build the dungeon, it clutters up the hierarchy

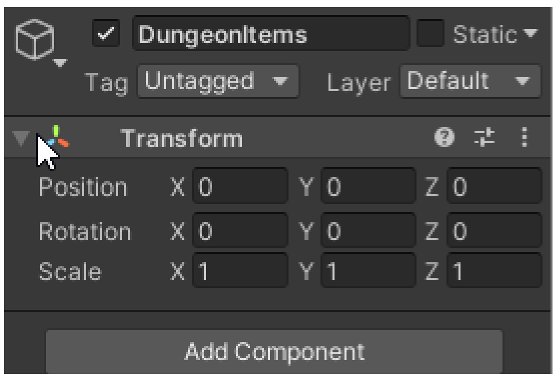


We'll configure it so that our dungeon is built under a certain game object and won't clutter the root.

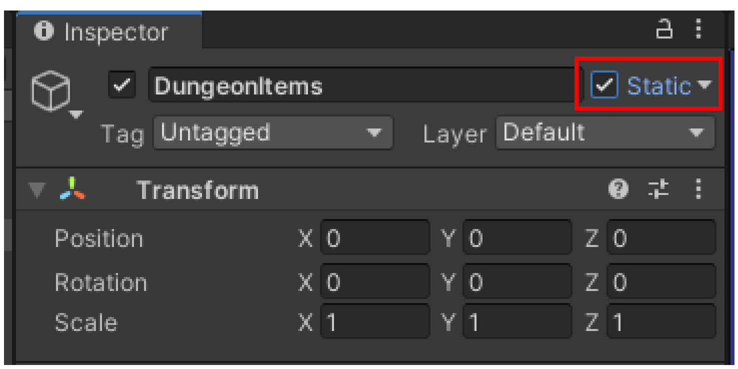
1. Create a new empty game object and name it `DungeonItems`



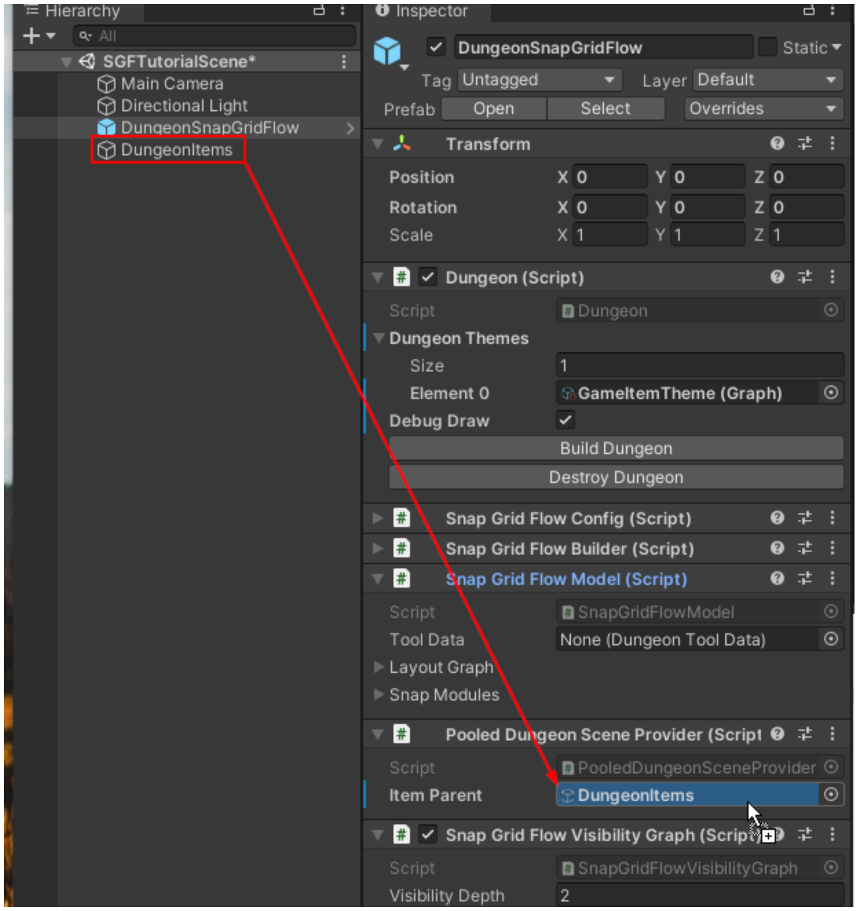
2. Reset the transform



3. Set it to **static**



4. Assign this game object to the *DungeonSnapGridFlow* gameobject's *Pool Dungeon Scene Provider* component



5. Click `Build Dungeon` again and our dungeon will be built under the `DungeonItems` game object

Save Map

We've set up our dungeon game object. Save this scene somewhere, we'll revisit it later ## Placeable Markers

In this section, we'll spawn items like NPCs, power ups etc in our dungeon.

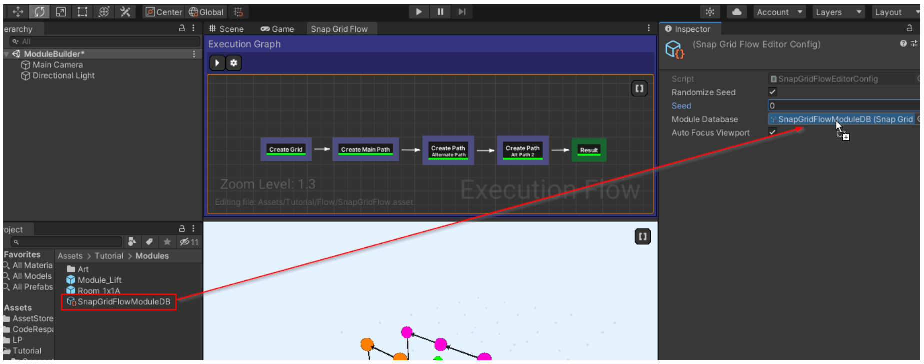
For this, we'll do the following:

- Use the `Spawn Item` node in the flow graph. This will add items in the layout graph
- Create `Placeable Marker` assets and drop a few of them in the snap modules so that the builder can spawn them at those places when necessary

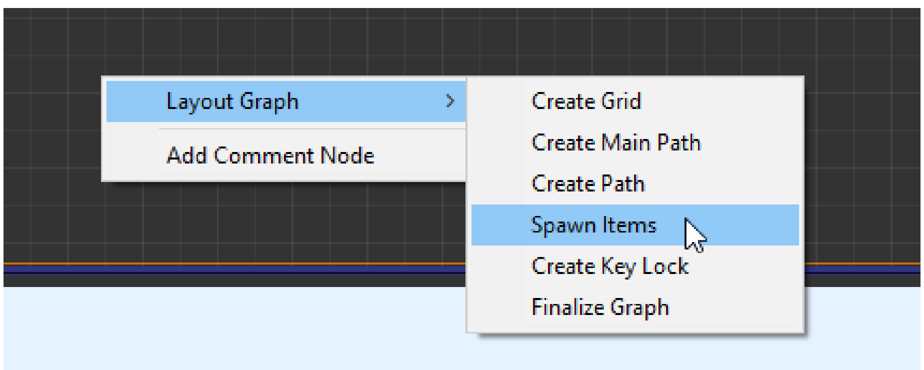
- Use the Theme engine to spawn the actual prefabs at those marker locations

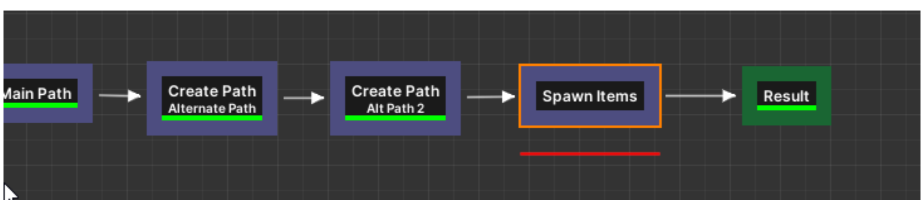
Spawn Items in Flow Graph

Open up our flow graph editor and reassign the Module database in the Editor Settings, as we've done previously

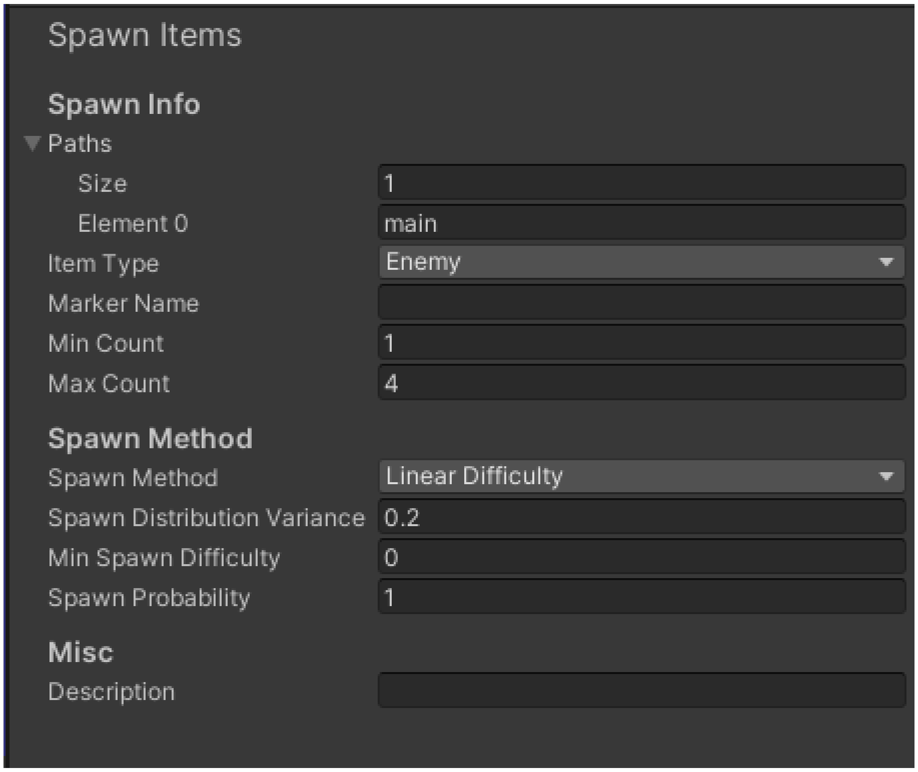


1. Create a new `Spawn Items` node and link it up as shown below:

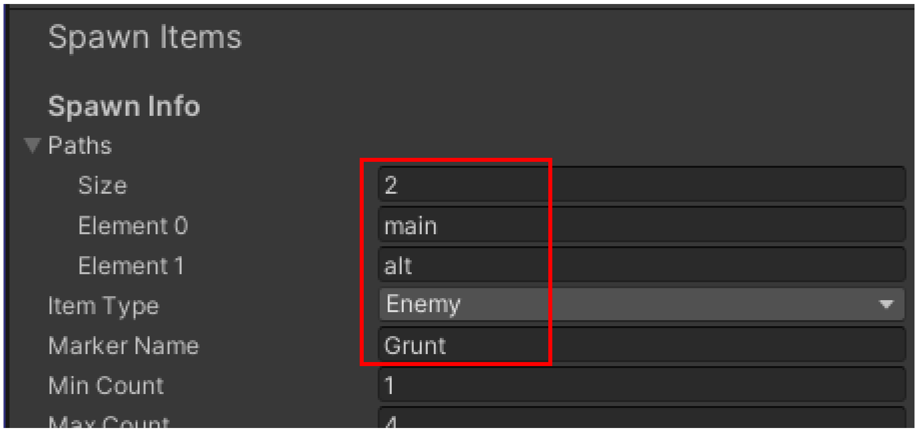




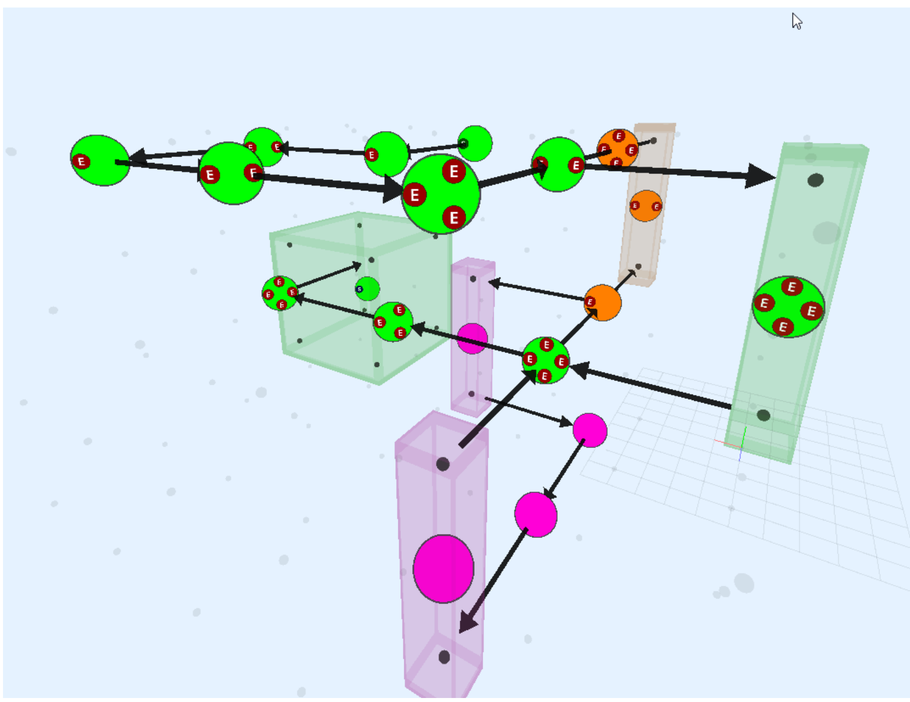
2. Select the `Spawn Items` node and inspect the properties.



3. We want to spawn enemies in the `main` path (green) and the `alt` path (orange).
    - Add two entries to the `Paths` array and set the values to `main` and `alt`
    - Set the `Item Type` to `Enemy`
    - Set the marker name to `Grunt`. Later in the theme file, we'll create a marker node named `Grunt` and place our NPC prefabs under it

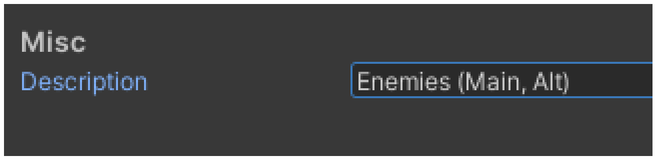


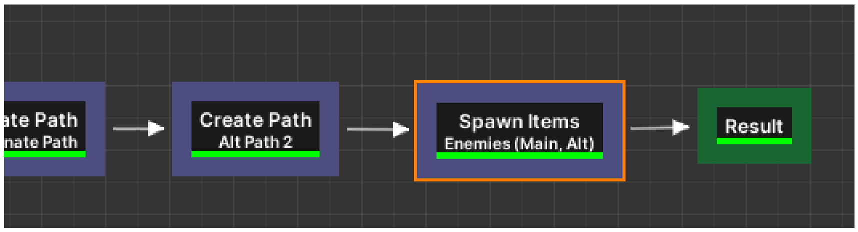
4. Rebuild the flow graph



The nodes now in the green path `(main)` and orange path `(alt)` have red enemy items

5. Add a description to this node





Placeable Markers

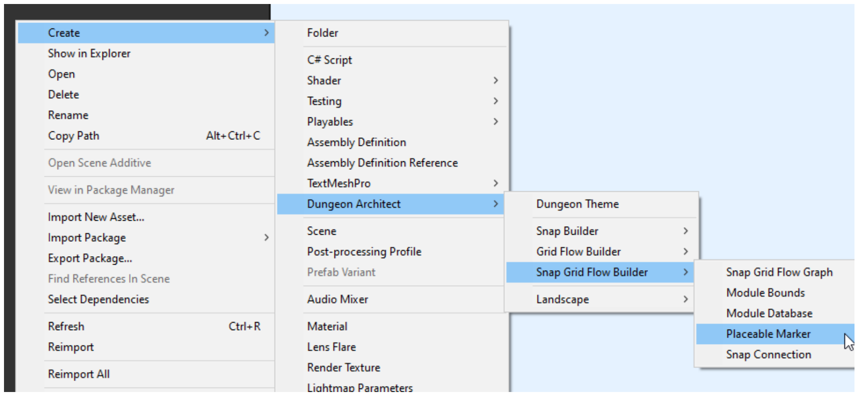
A placeable marker is an prefab you create, which you can then drag and drop anywhere on your modules. You can then use the theme file to spawn objects at that location.

A placeable marker prefab can contain more than one marker name. For example, a placeable marker prefab named `PM_Enemies` may contain a list of marker names like (`Grunt`, `FireTroll`, `IceTroll`, `Goblin`). In your snap module, you'd place these markers in appropriate locations (say 10 different locations within the room module).

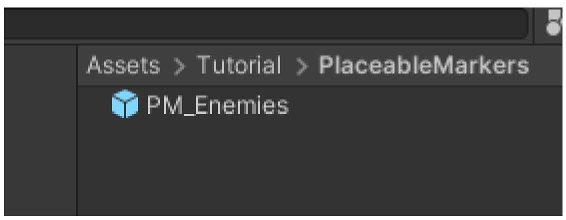
If the dungeon builder needs to spawn a `Grunt` marker 4 times inside the room, it will first find all the existing and compatible marker assets placed in the room. In this case `PM_Enemies` would be compatible since it contains a `Grunt` marker. Since we have 10 of these in the room module, it will randomly pick 4 from them and use the theme file to spawn the grunt prefab

Create Prefab

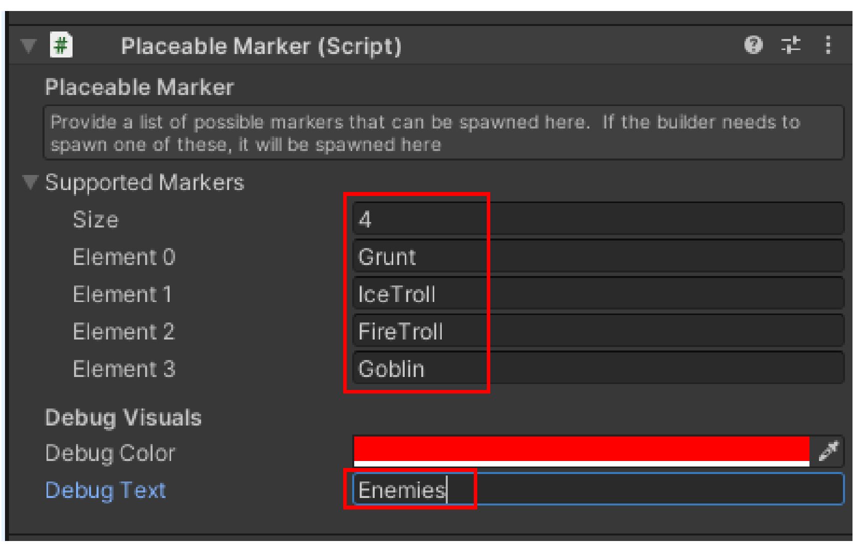
1. Move to an appropriate folder and create a Placeable Marker prefab from the create menu



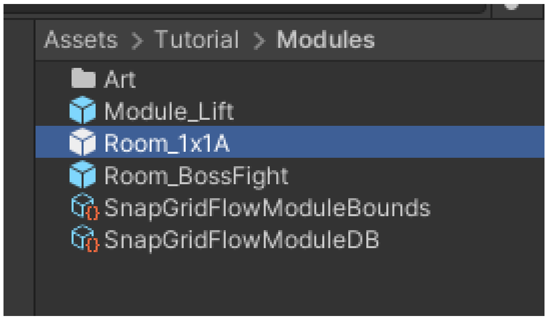
2. Rename it to `PM_Enemies`

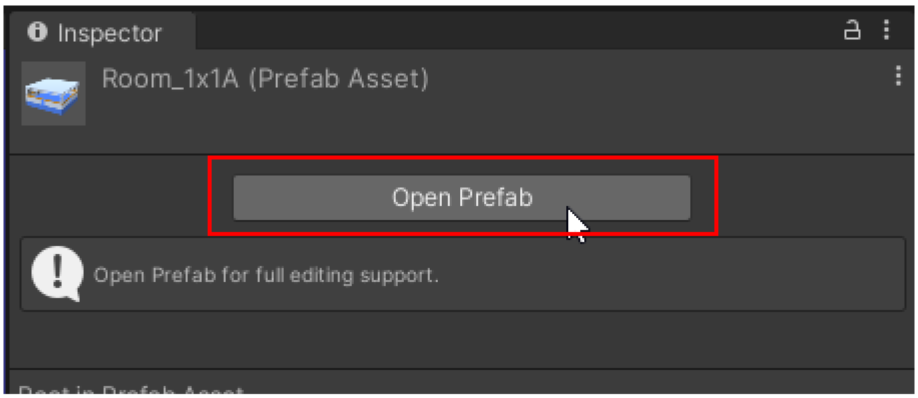


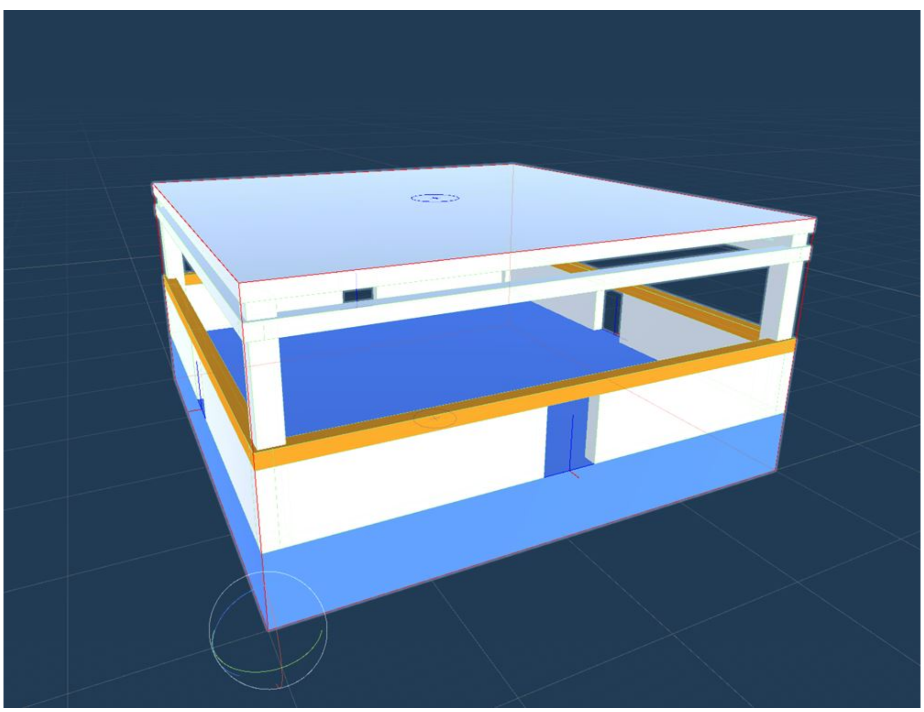
3. Select the `PM_Enemies` prefab and inspect the properties. Add a `Grunt` marker (since we specified this earlier in the `Spawn Items` node). Add a few more markers for future use like `IceTroll`, `FireTroll`, `Goblin`



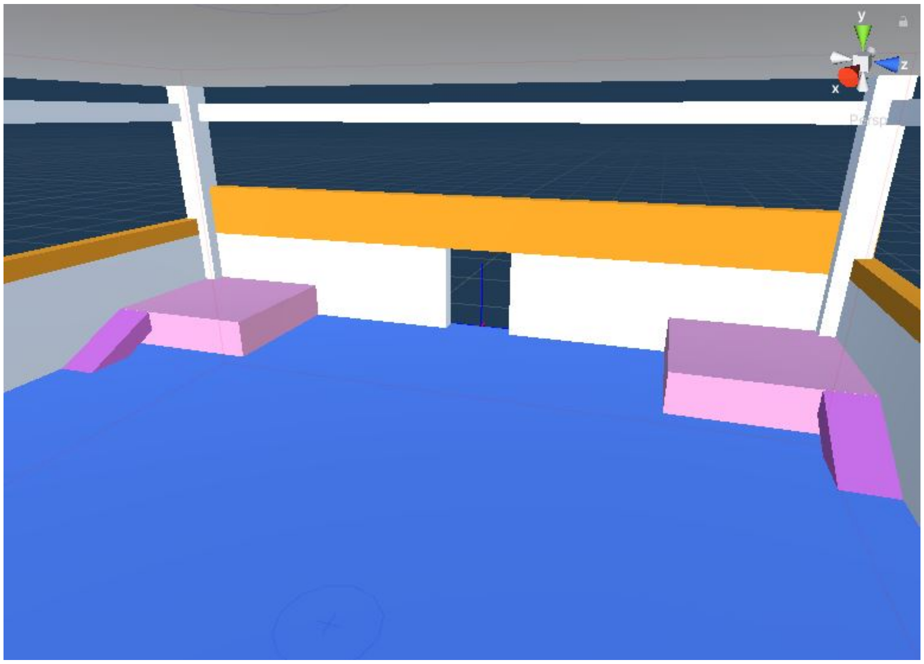
Add to Snap Modules Open up the previously created room module


Assets > Tutorial > Modules
Art
Module_Lift
Room_1x1A
Room_BossFight
SnapGridFlowModuleBounds
SnapGridFlowModuleDB


Inspector
Room_1x1A (Prefab Asset)
Open Prefab
Open Prefab for full editing support.

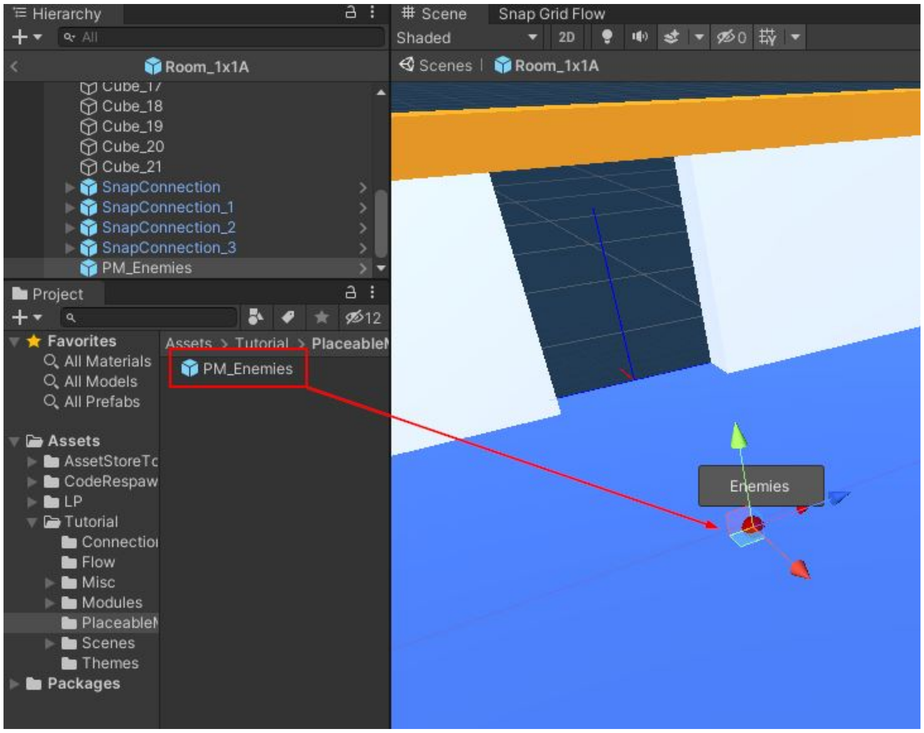


We went ahead and added a bit of geometry int he room

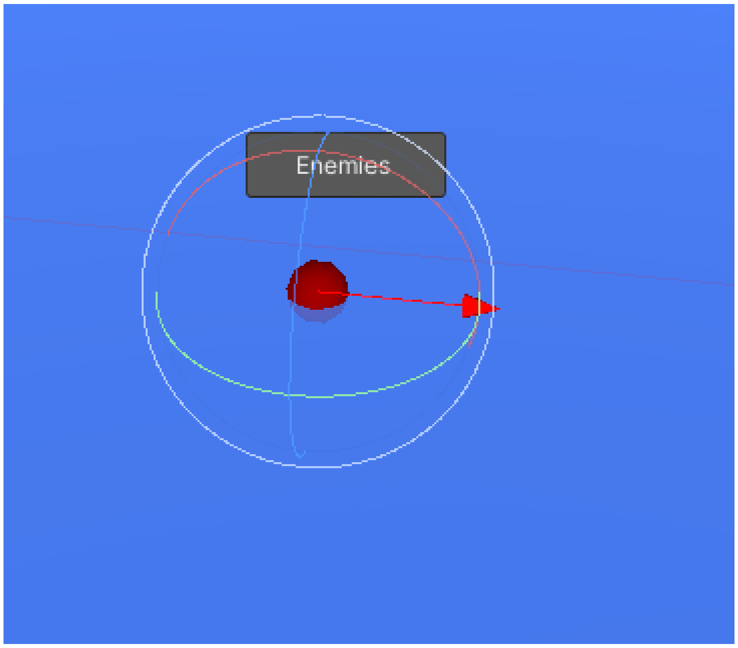


The snap system gives complete freedom to the artist to design the room as they see fit. In that same spirit, the artist should also have control on where the markers spawn. This is where placeable markers come in

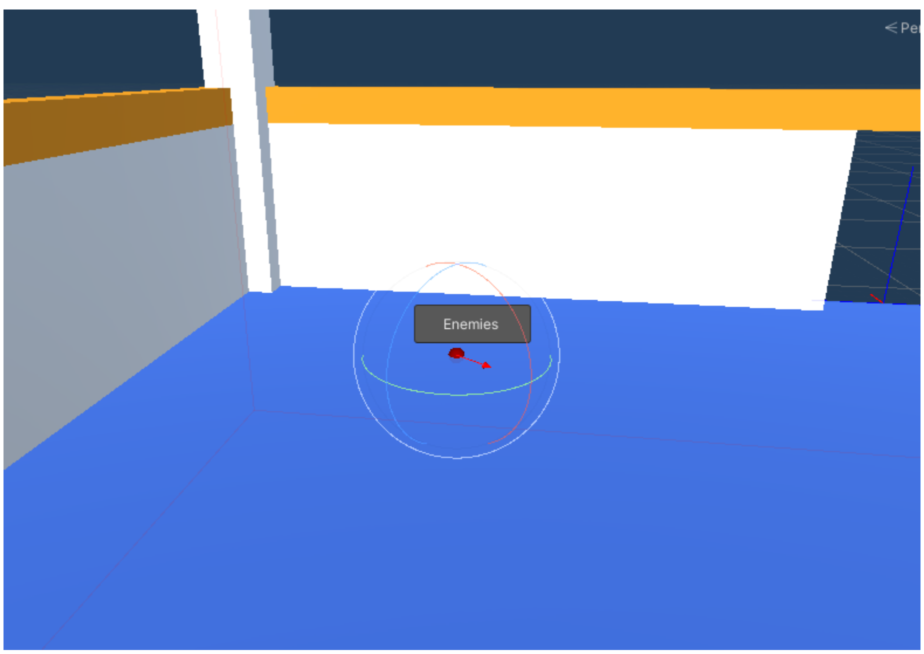
Drag drop the placeable marker prefab that you’ve created before, on to the scene



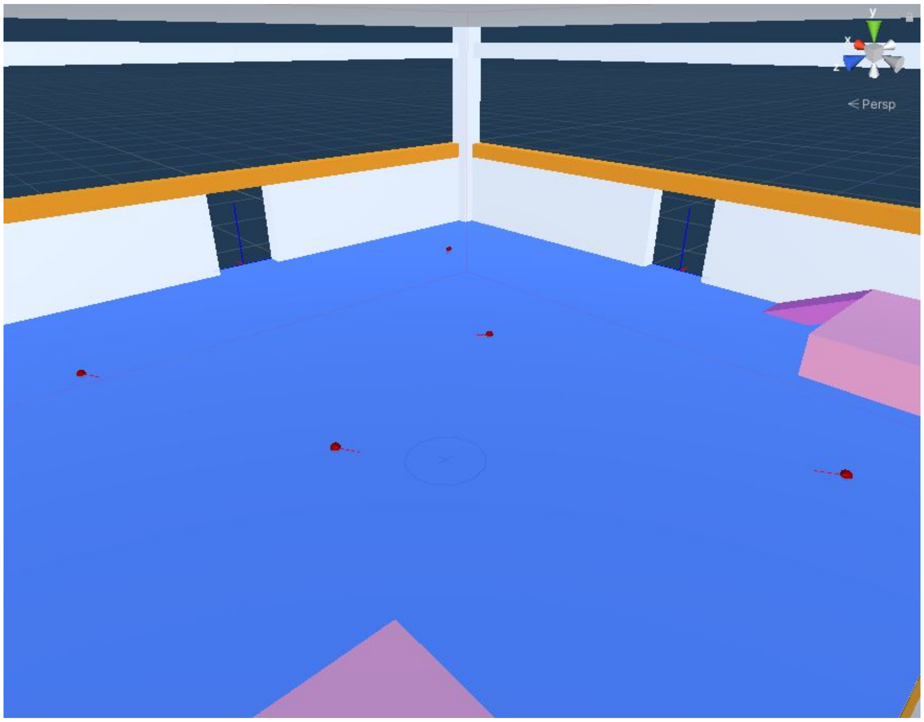
This will spawn a placeable marker game object on the scene. When selected, it shows the descripton (`Debug Text`) of the placeable marker.



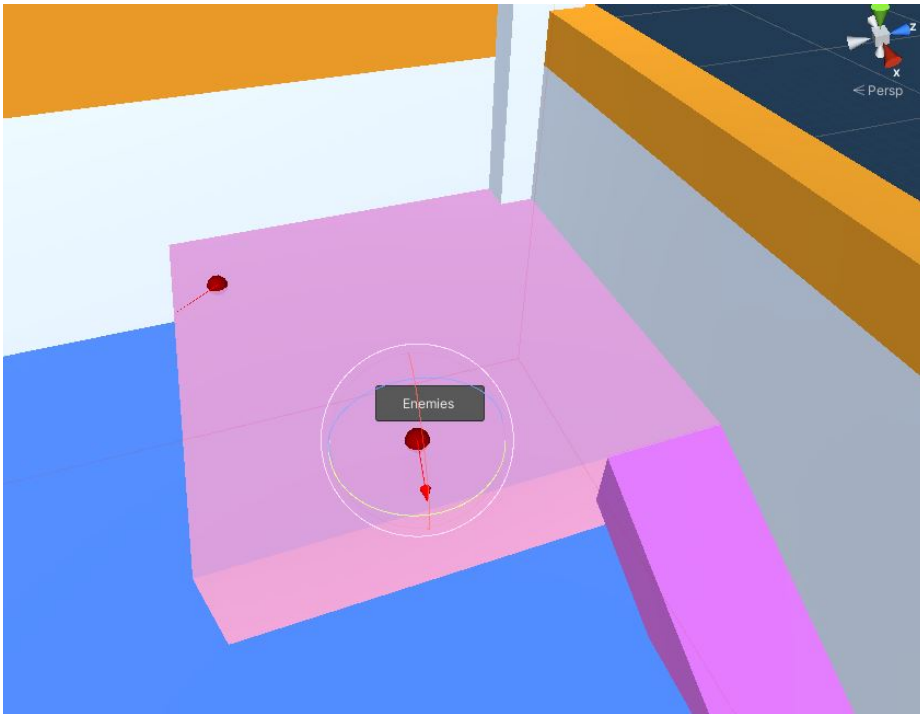
Rotate the actor as needed. The red arrow shows the orientation of the marker. When the theme engine spawns an actor here, it will do so with this rotation

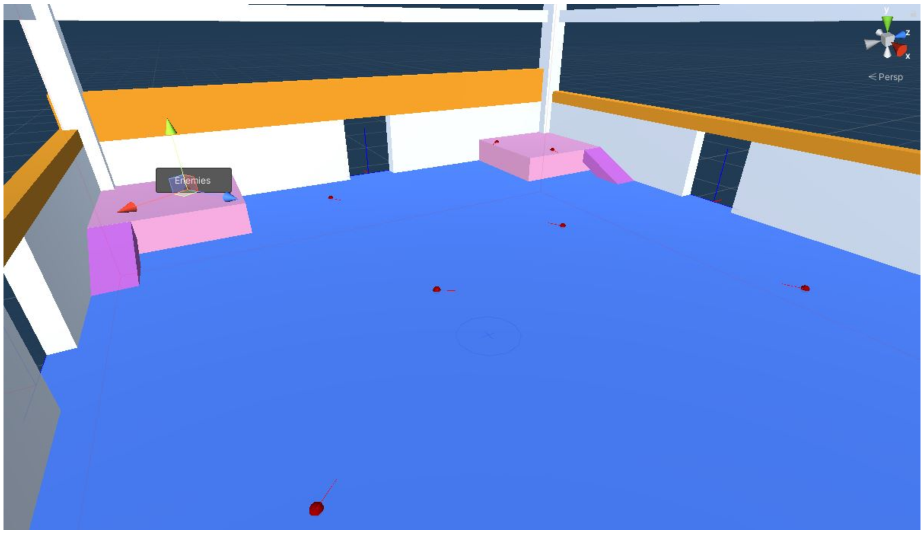


Add a few more markers. Add at least 4 markers, since we are adding a maximum of 4 enemy items per node in the flow graph using `Spawn Item`, but adding more is always better



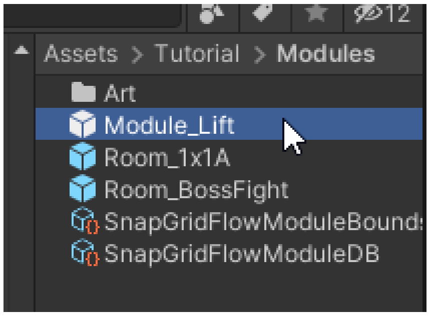
We'll add a few more on top of the ramp

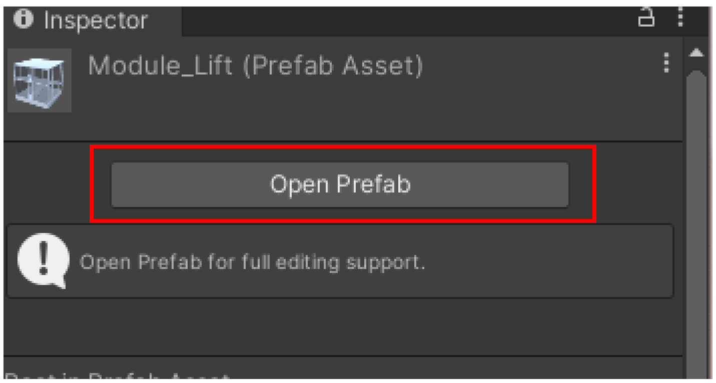


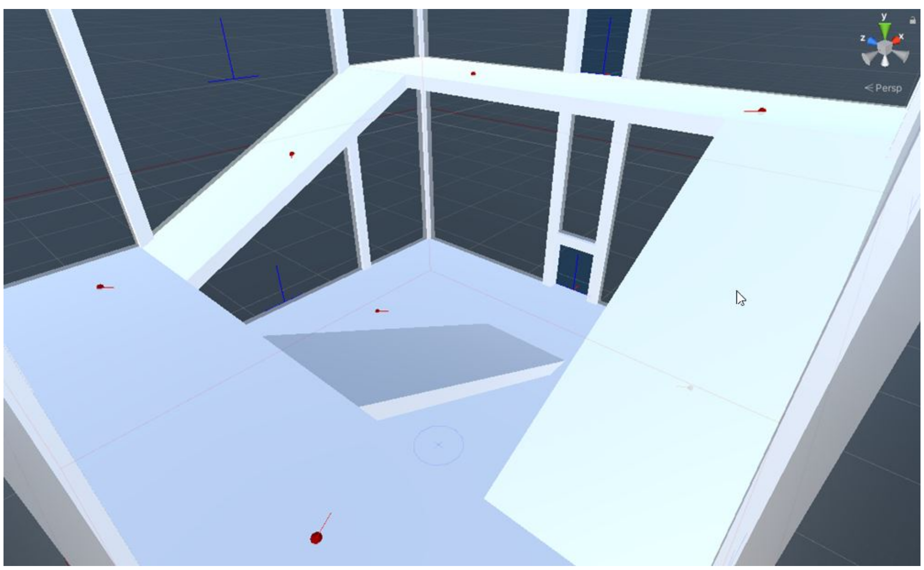


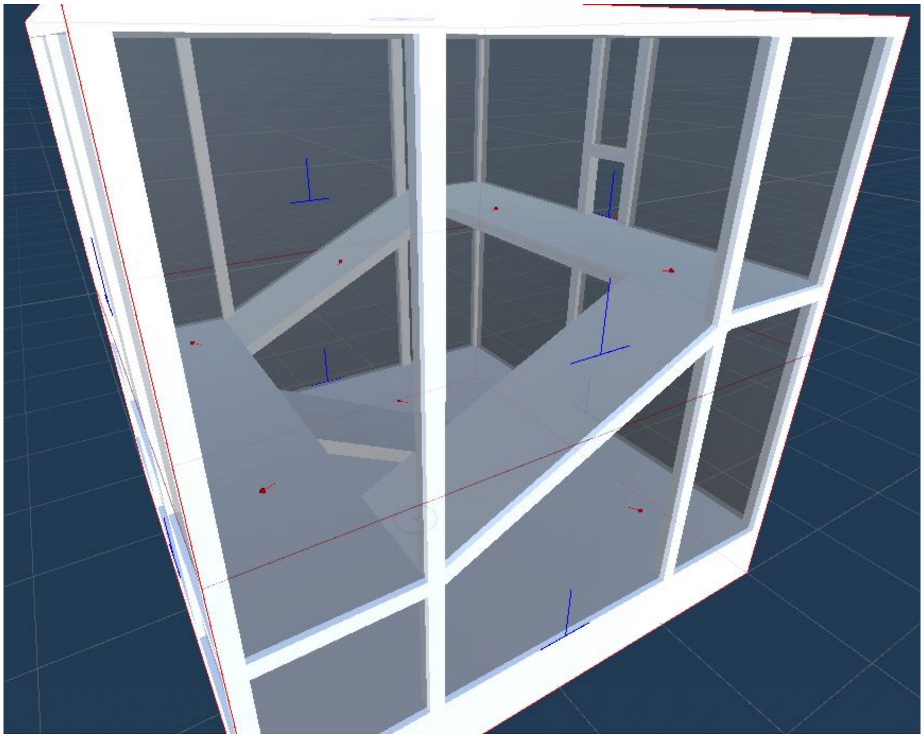
Do this for all the modules you've created so far that would need this marker in it

Open up the Lift module and add a few more placeable markers there

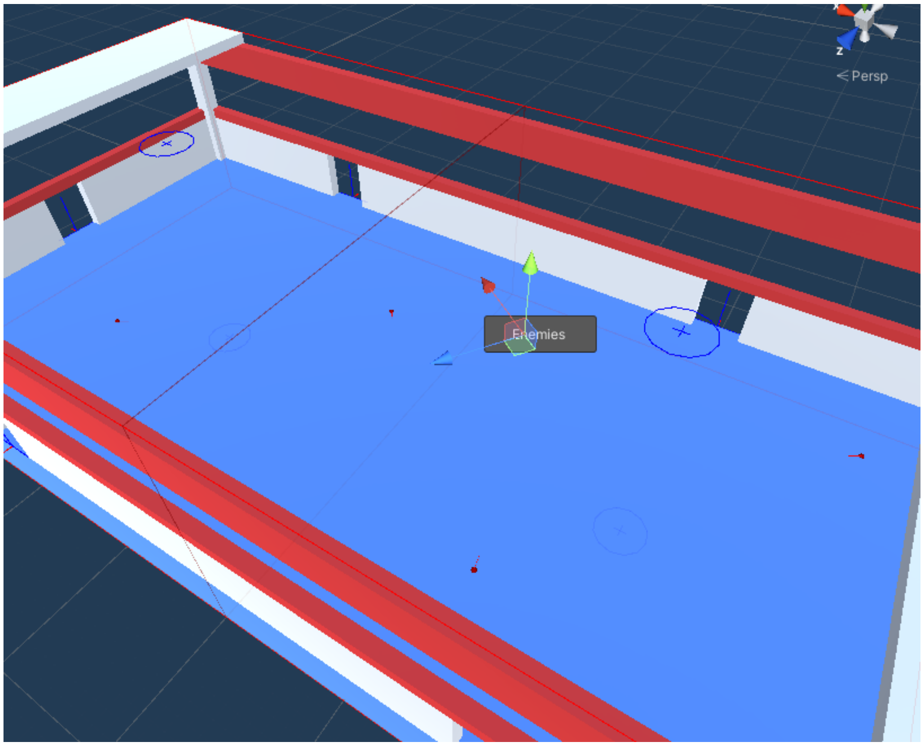

12
Assets > Tutorial > Modules
Art
Module_Lift
Room_1x1A
Room_BossFight
SnapGridFlowModuleBound
SnapGridFlowModuleDB

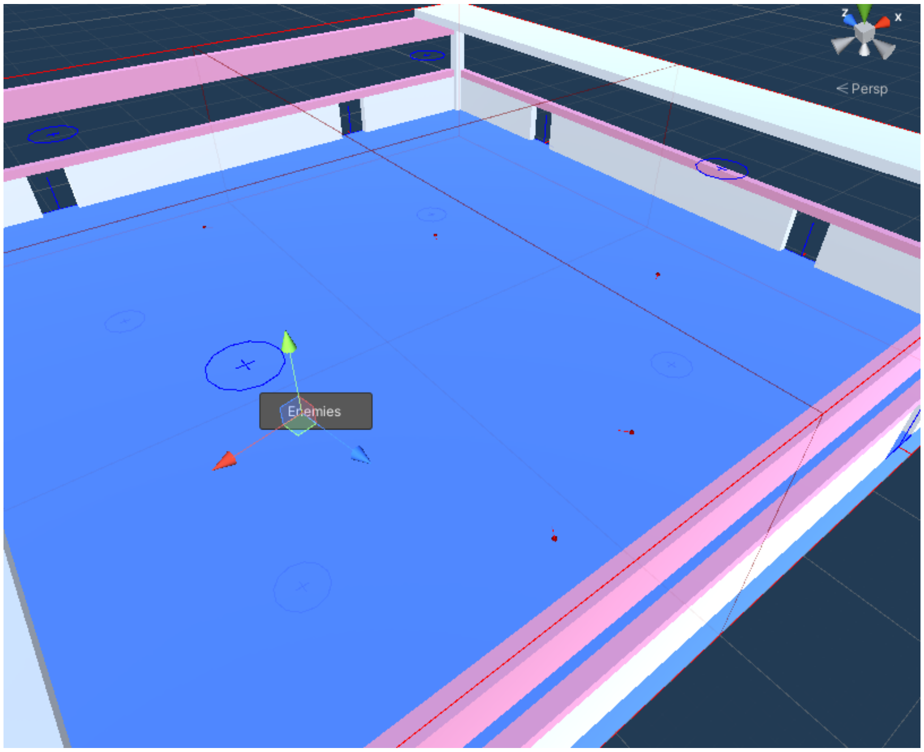

Inspector
Module_Lift (Prefab Asset)
Open Prefab
Open Prefab for full editing support.


Persp



Open up the 2x1x1 and 2x2x2 Room modules and add the markers there as well

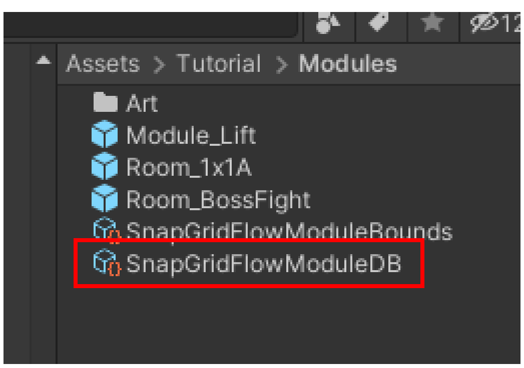

Persp
Enemies

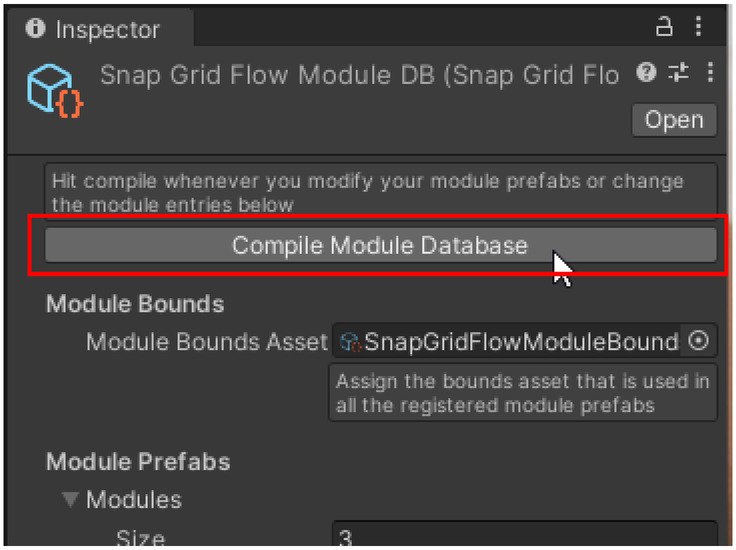


Rebuild Module Database Cache

If you add / remove a placeable marker from a snap module, you'll need to rebuild the module database cache

Select the Module database asset that we created in the previous section and in the inspector, click `Build Module Cache` button. Save the module database asset



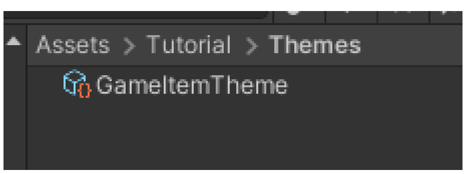


:::warning IMPORTANT This is an important step. Remember to rebuild the cache when needed :::

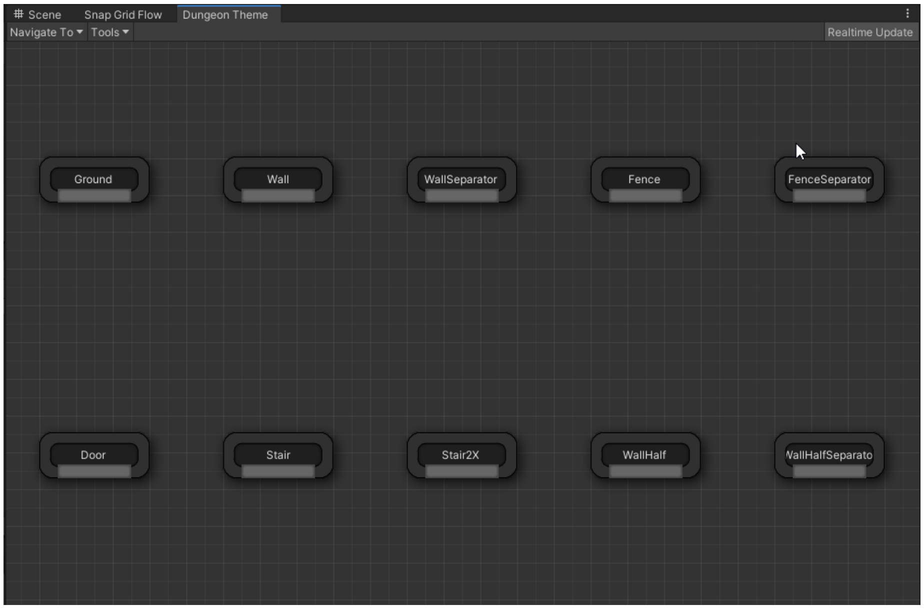
Update Theme File

We'll use a theme file to actually spawn our enemy prefab.

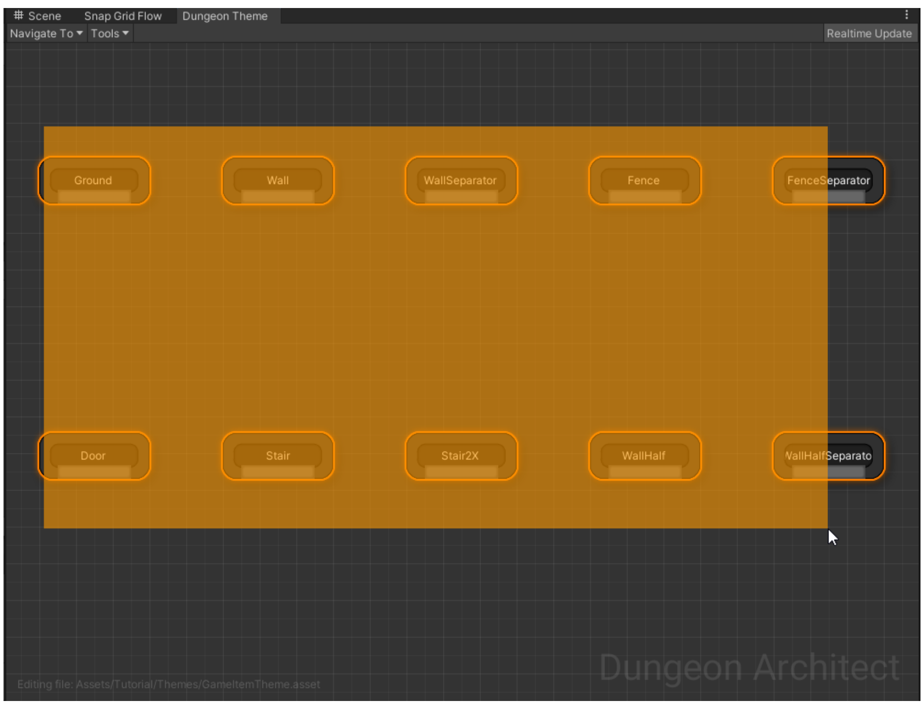
Open Theme Editor Open up the existing theme file we created in the previous section.

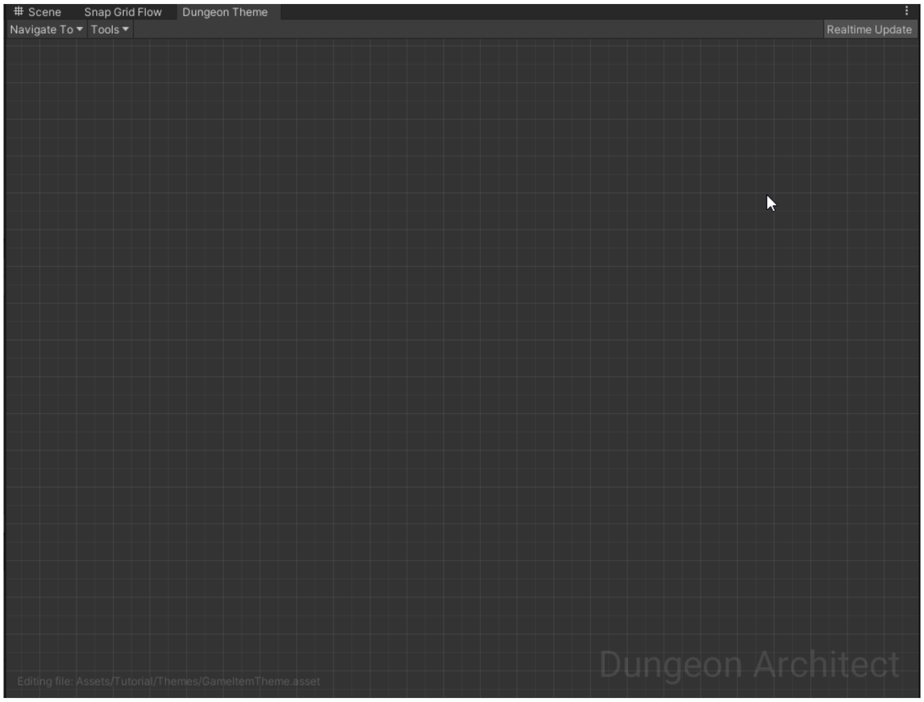


Double click on the theme asset to open the theme editor



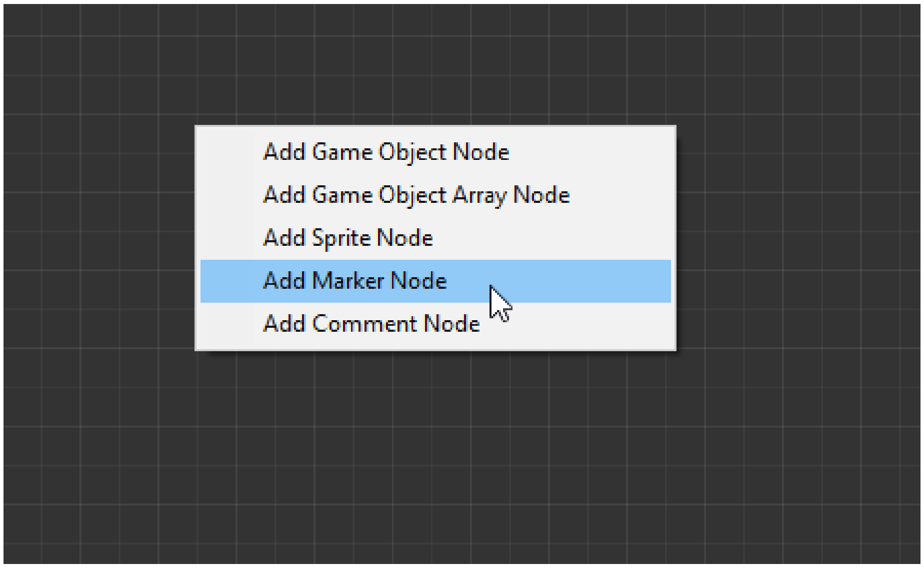
Box select all the marker nodes and delete them



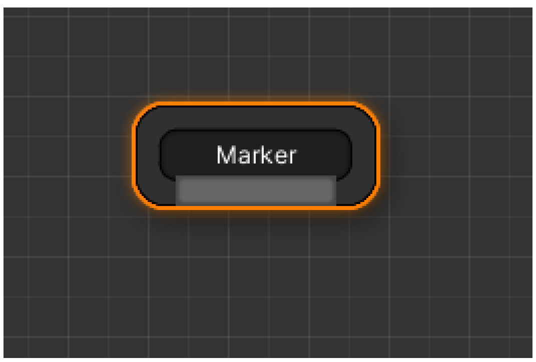


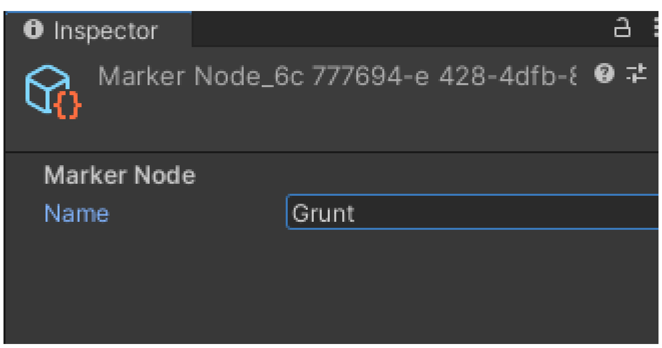
Create Marker In the previous section, we assign the marker name of `Grunt` in the `Spawn Items` node. In the theme file, we'll create a new marker node and name it `Grunt`

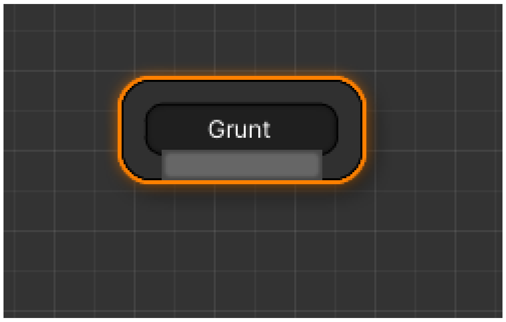
Right click on the graph and choose `Add Marker Node`



Select the marker node and from the inspector, change the Marker name to `Grunt`

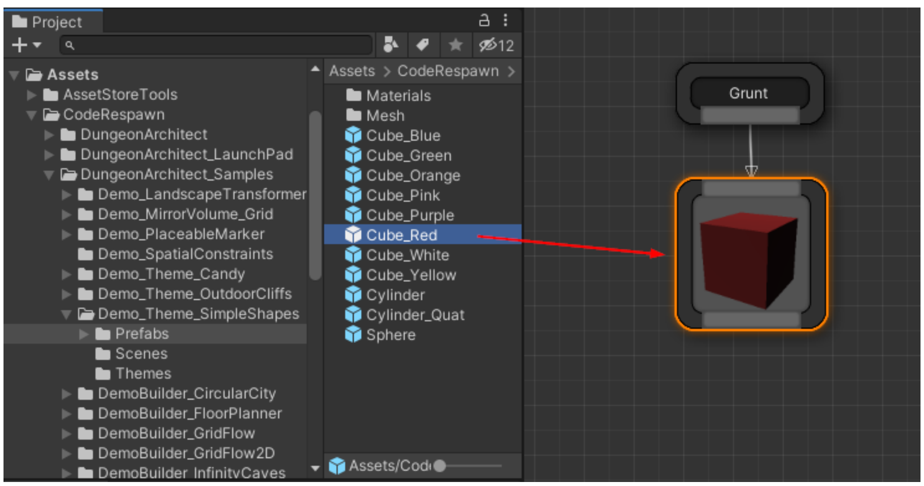




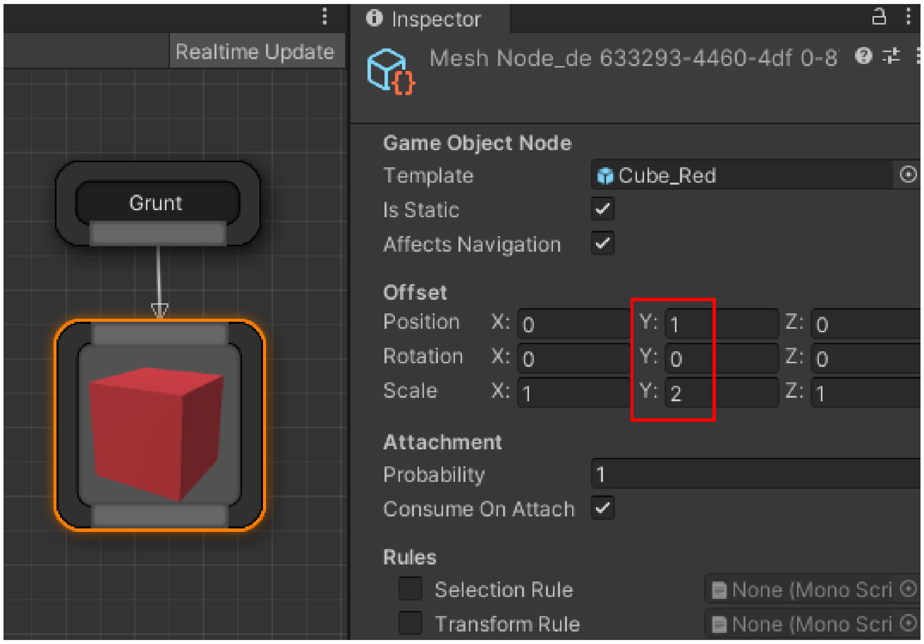


Add Enemy prefab Add your NPC character prefab here. For this tutorial, we'll add a cube and adjust the size and scale

Navigate to `Assets > CodeRespawn > DungeonArchitect_Samples > Demo_Theme_SimpleShapes > Prefabs` and drop in `Cube_Red` and connect it the `Grunt` marker node to it



Select the cube node that you just dropped and adjust the scale and position

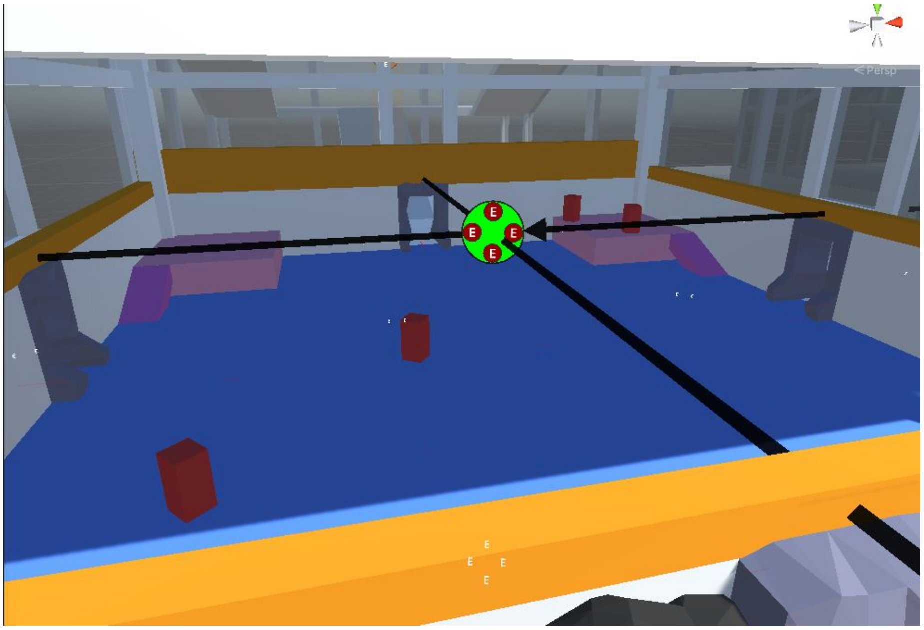


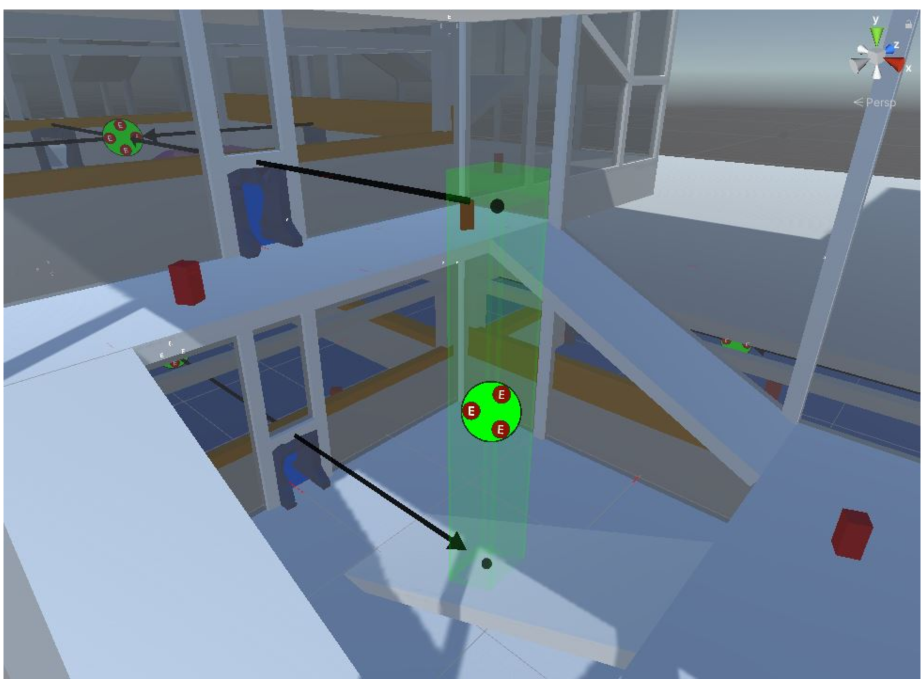
Save and close the Theme Editor

Build Dungeon

Open the map where we previously configured our dungeon and build it.

You'll see that enemies start to spawn at the locations where you've placed the markers

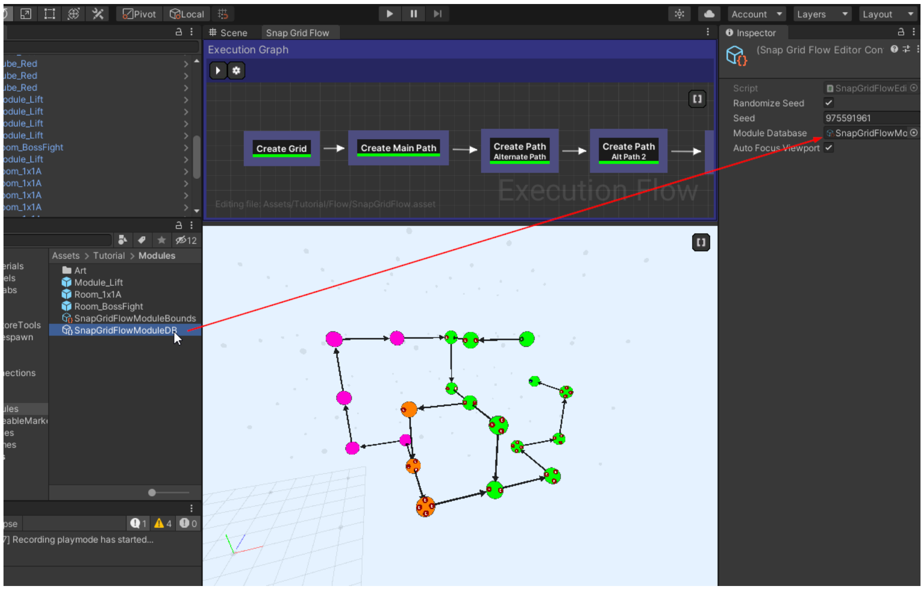




You can use this system to spawn anything (treasure chests, weapon racks, power ups or any gameplay prefab)## Setup Key-Locks

In this section, we'll add a few key-locks to our dungeon using the flow framework

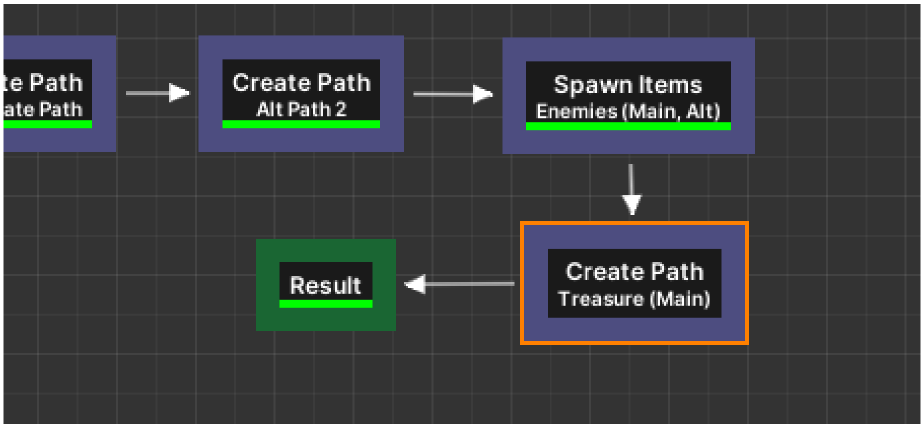
Open up the flow graph and assign the module database like before



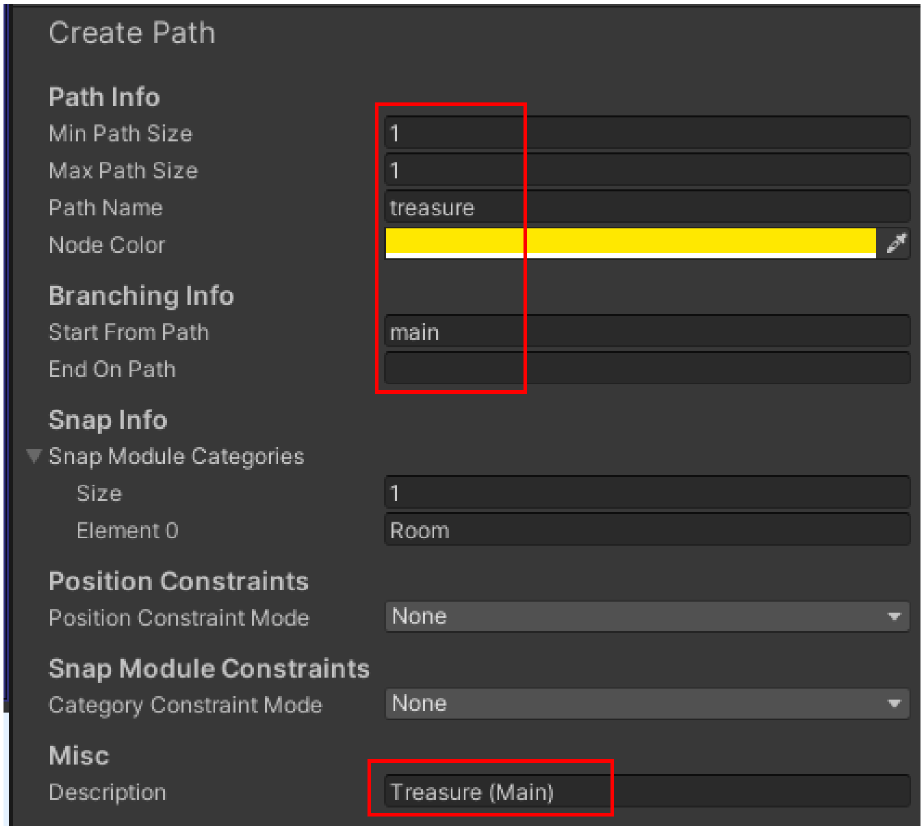
Create Treasure Room

We'll create a treasure room that is attached to the main path. We do this by creating a new path (of length 1) that emits out of the `main` path

Add a new `Create Path` node and link it to the end as shown:

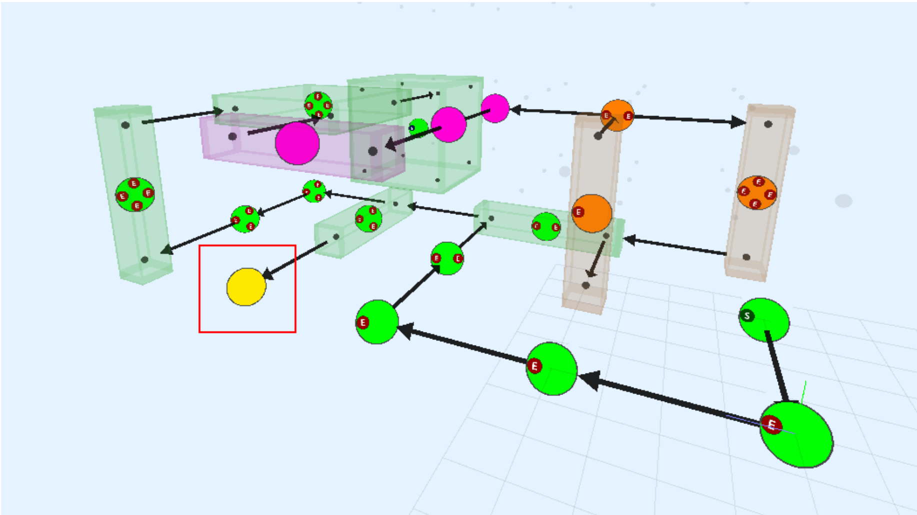


Select the node and inspect the properties



- Set the Min/Max size to `1` since we want a single room to hold the key
- Set the path name to `treasure`. We'll use this id later to place a key here
- `Start from Path` is set to `main` so this room is connected to the main path
- We want the treasure room to be isolated and don't want it to converge back to another path. Leave the `End on Path` parameter blank
- Change the `Node Color` to anything you like

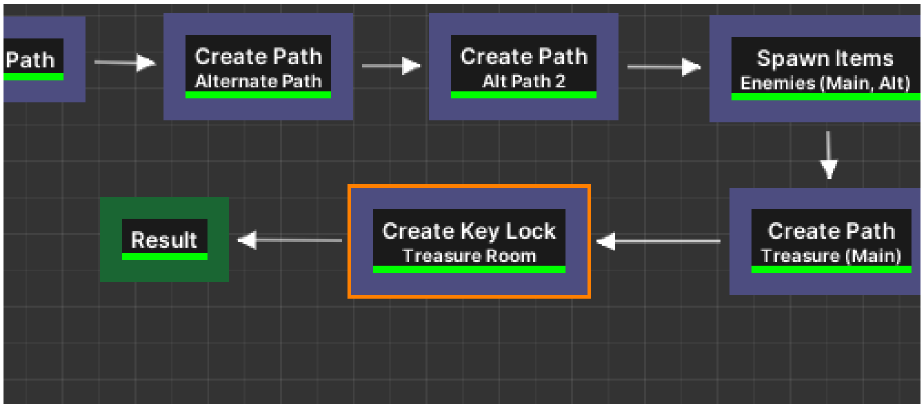
Hit `Build` and you'll see a new treasure room created



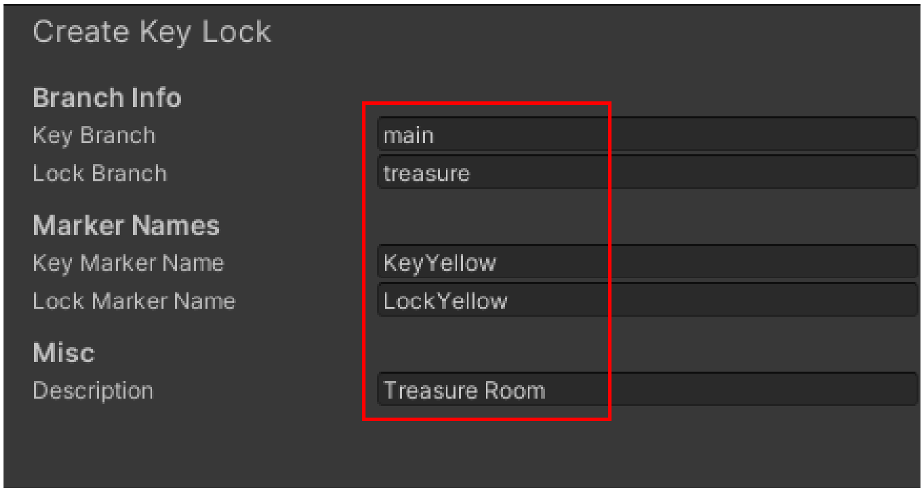
Treasure Room Key/Lock

We want to lock the `treasure` room and have the key somewhere in the `main` path

Add a new `Create Key/Lock` node and link it to the end like shown below:

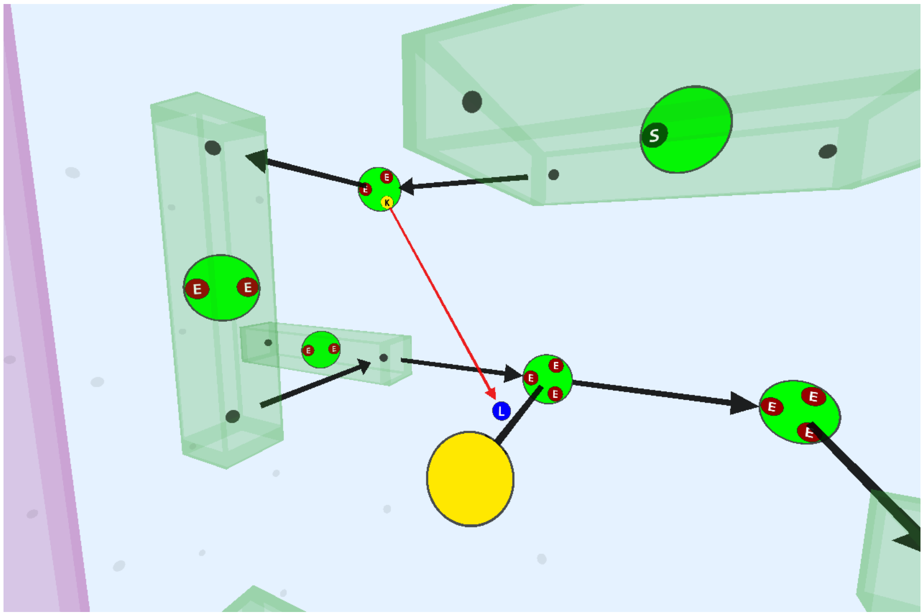


Select the `Create Key/Lock` node and inspect the properties



- `Key Path` - Since we want the key to be in the main path, set this to `main`
- `Lock Path` - Since we want to lock the treasure path, set this to `treasure`
- `Key Marker Name` - We'll use the **Theme Editor** to spawn the key prefab. Set the maker name to anything you like, however you'll need to create a corresponding marker node with the same name in the theme file to spawn the key prefab. For this tutorial, set this to `KeyYellow`
- `Lock Marker Name` - We'll use the **Snap Connection** prefab to spawn the locked door prefab. Set the marker name to anything you like, however you'll need to create a corresponding mapping in the snap connection to spawn the locked door prefab. For this tutorial, set this to `LockYellow`

Hit `Build` and inspect the layout graph

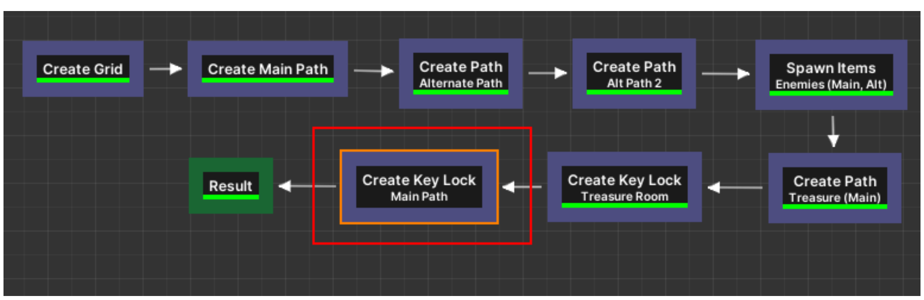


A blue lock item was created on the link that connects to the yellow treasure node. The key was placed somewhere in the main path (green). The red arrow shows the key-lock relationship

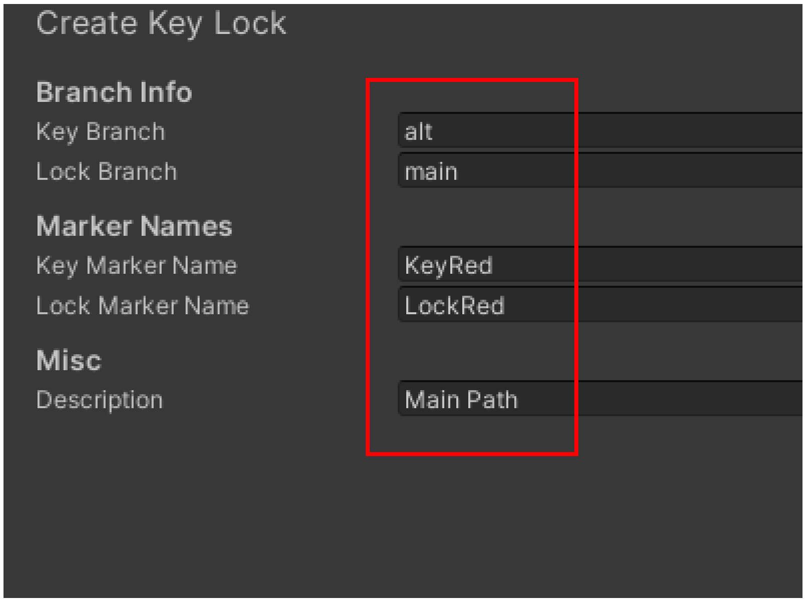
Main Path Key-Lock

We'll create another key lock for the main path. The key would be in the `alt` path (orange) and the lock would be somewhere in the `main` path (green).

Add a new `Create Key/Lock` node and link it to the end as shown below:

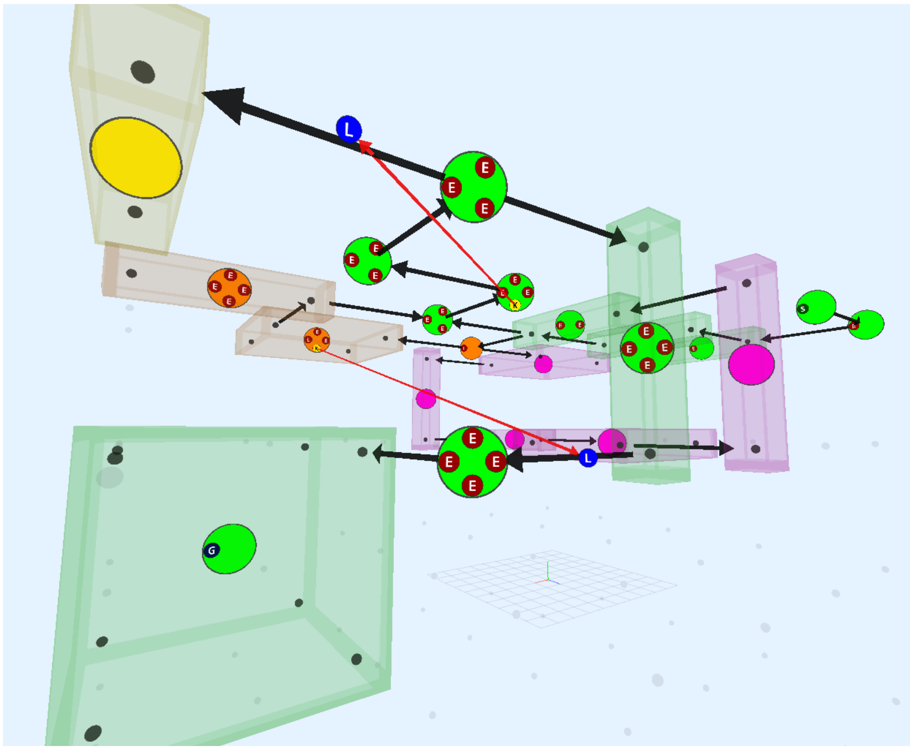


Update the parameters:



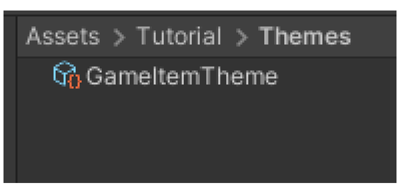
Parameter | Value Key Path | alt Lock Path | main Key Marker Name | KeyRed Lock Marker Name | LockRed Description | Main Path

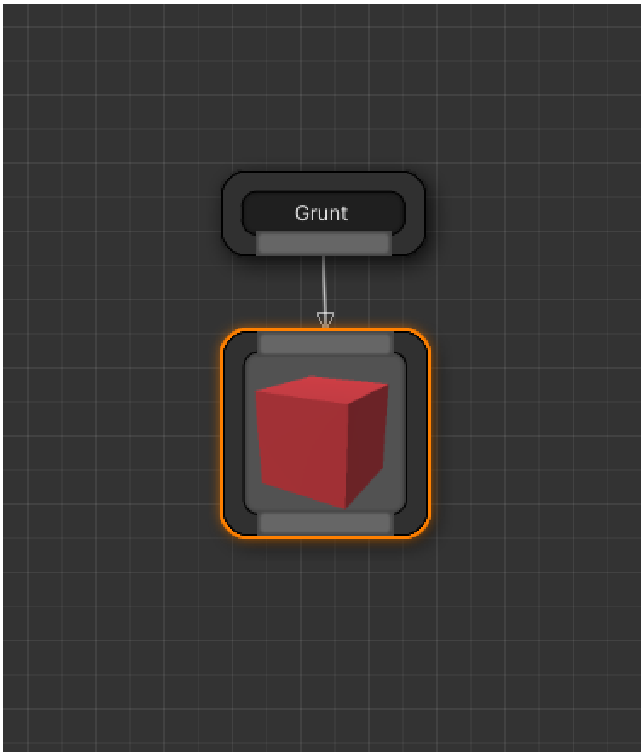
Hit `Build` and inspect the layout graph



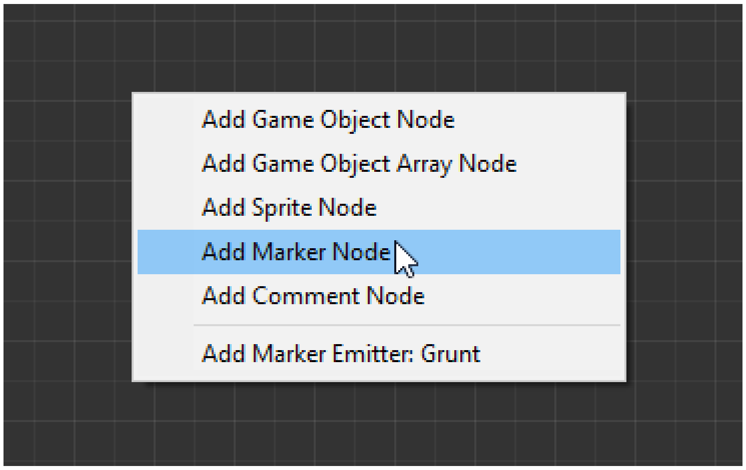
Spawn Keys

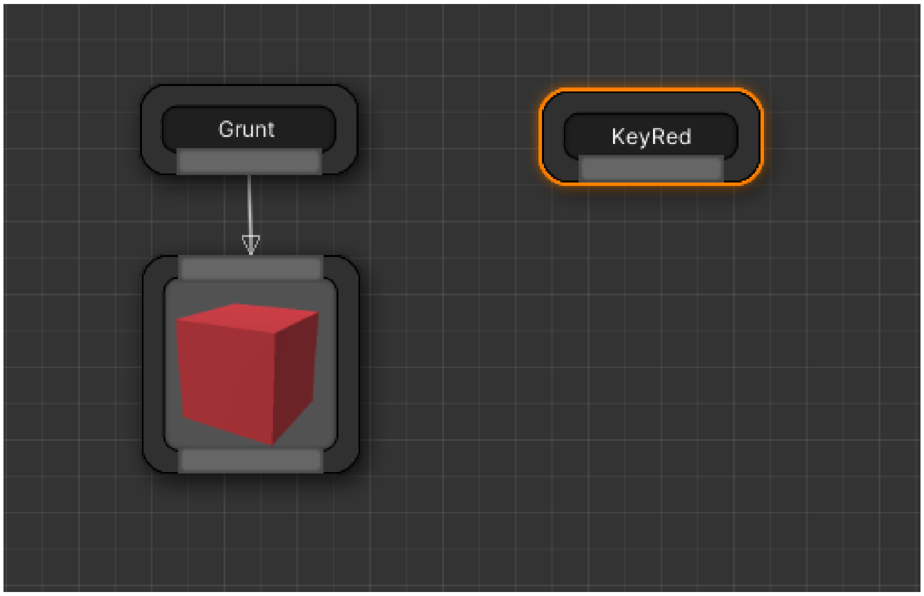
Setup Theme File Open up the theme file we created earlier



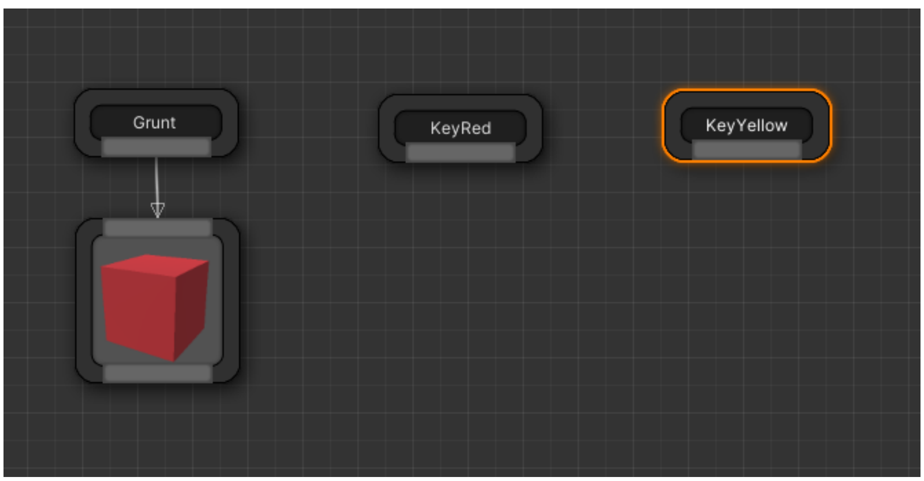


Create a marker node ane rename it to KeyRed





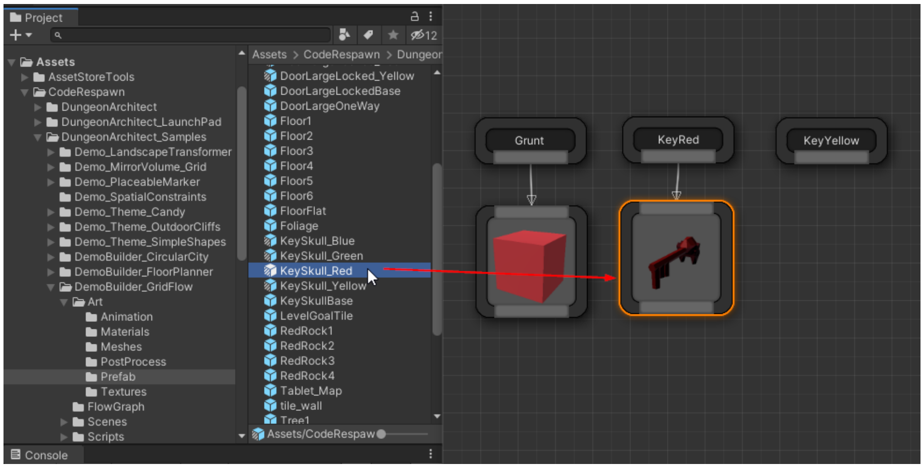
Create another marker node and name it `KeyYellow`

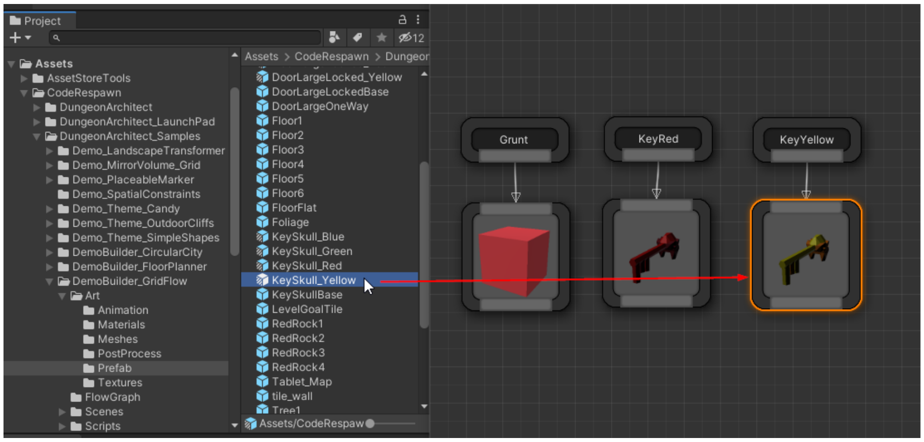


We’ll place our key prefabs under these.

The maker names are case-sensitive. So make sure you capitalize them correctly

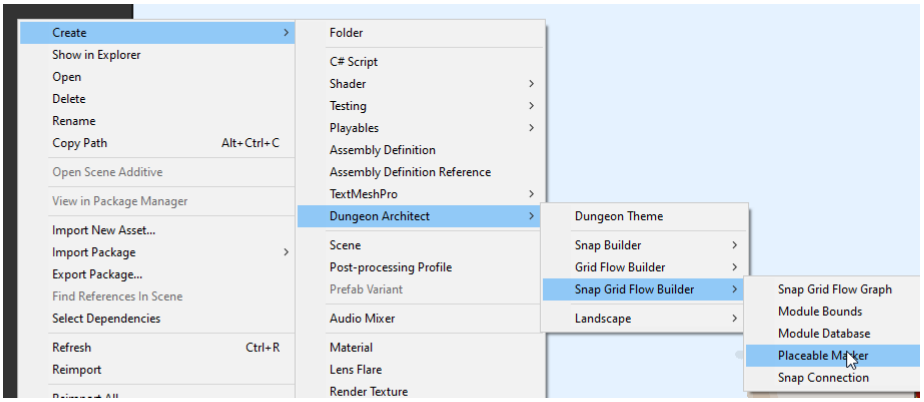
Navigate to `Assets\CodeRespawn\DungeonArchitect_Samples\DemoBuilder_GridFlow\Art\P` and drop in the `KeySkull_Red` and `KeySkull_Yellow` prefabs on to the theme editor and link them up

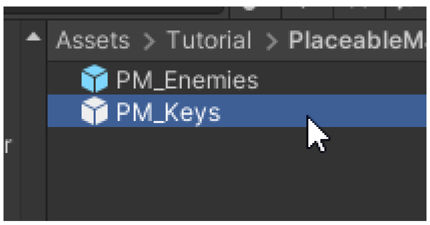




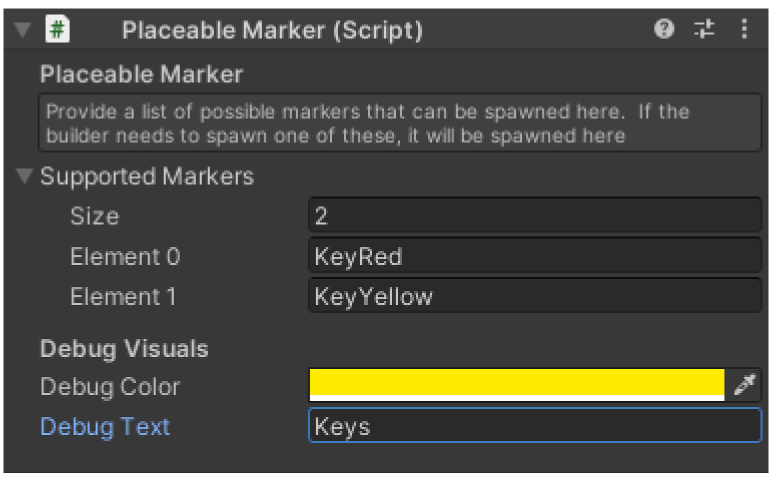
With this mapping, we've defined **what** key prefab to spawn. Next, we'll create a placeable marker asset to define **where** to spawn these inside the modules

Create a Placeable Marker Create a new placeable marker asset and name it `PM_Keys`





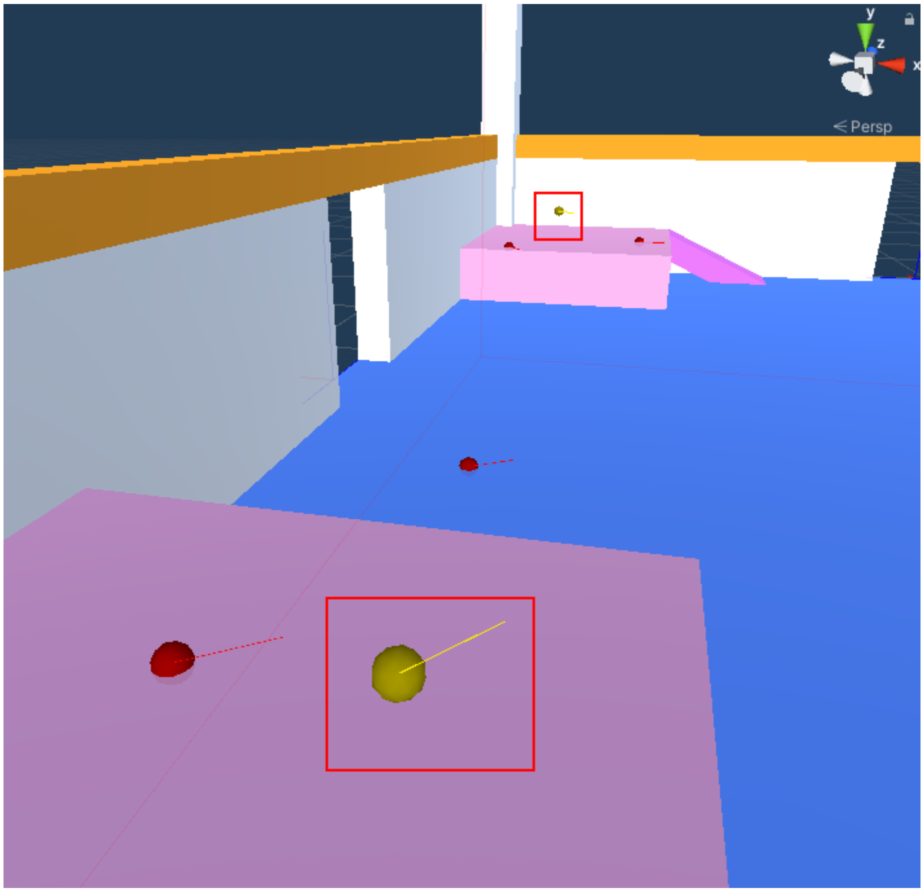
Select the PM_Keys asset and inspect the properties



- Add two markers KeyRed and KeyYellow
- Set the *Debug Color* to yellow
- Set the *Debug Text* to Keys

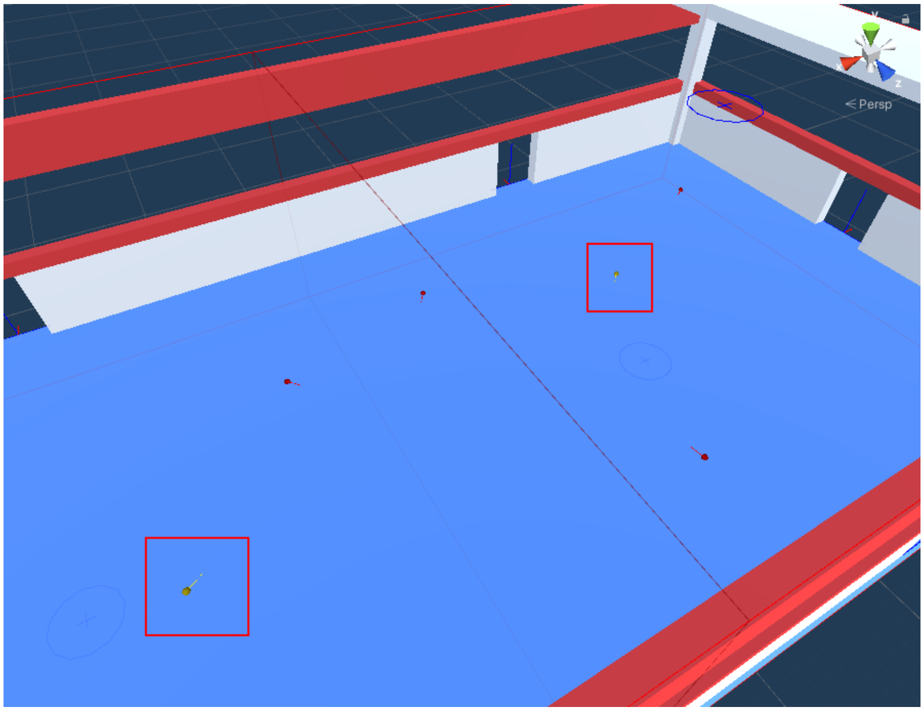
Place Key Markers Open up all your room modules and drop in the PM_Key placeable marker asset, similar to what we've done with the enemy placeable marker in the previous section

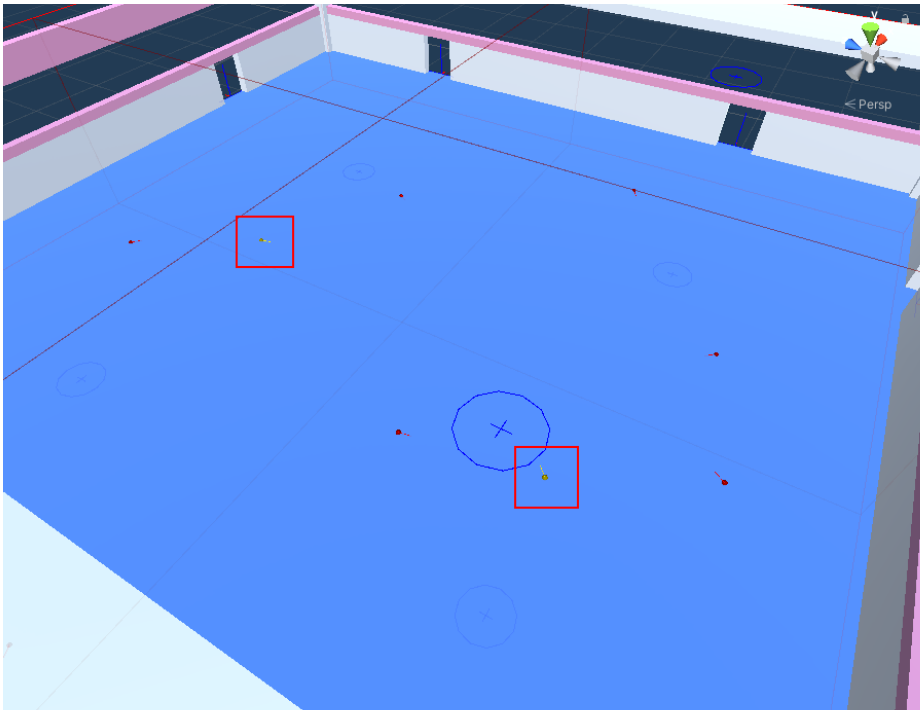
Drop in at least 2 of these in a room, in case if both the `Red` and the `Yellow` keys spawn in the same room

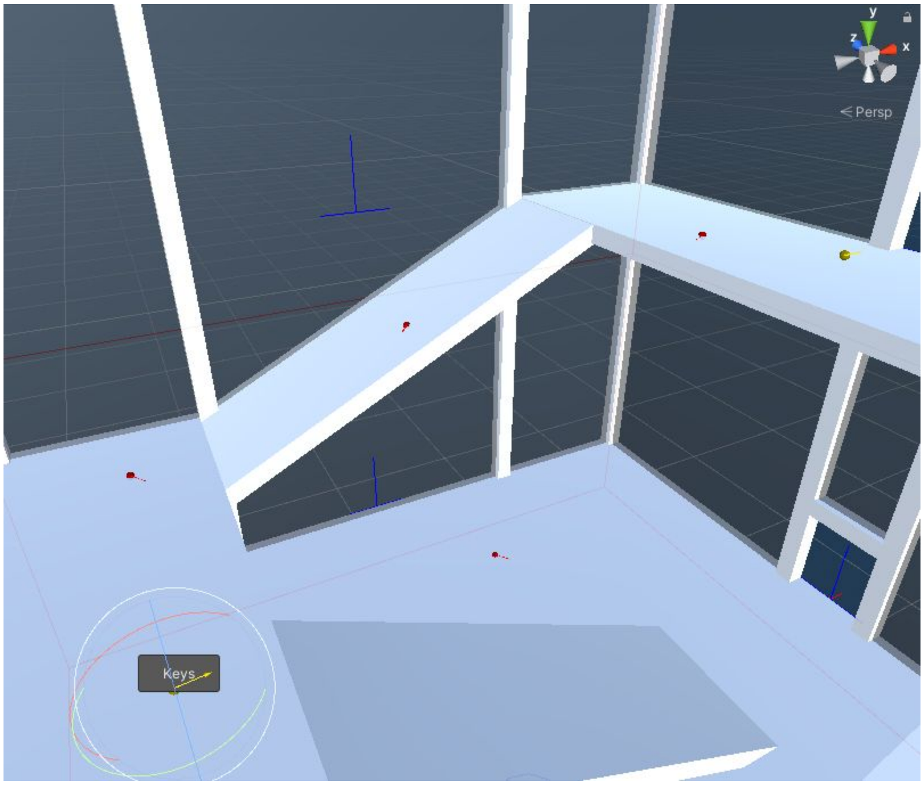


Either move this placeable marker game object up by 1 unit (so the key doesn't spawn buried half into the ground), or move the key up in the theme editor by selecting the key nodes and moving it up at Y by 1 unit

Add to the other room modules as well

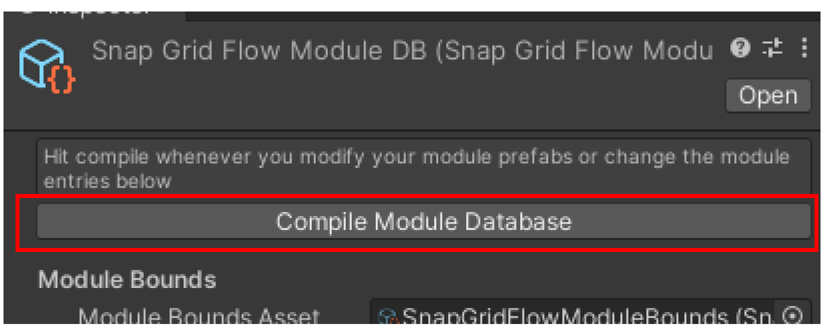

Persp


Persp



Recompile module database Since we've modified the module's markers, we need to rebuild the module database cache

Select the module database and click `Build Module Cache`



The system now knows where to spawn the keys in the snap module and what prefabs to spawn at those locations

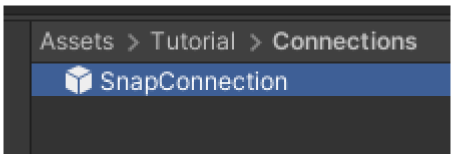
Spawn Locks

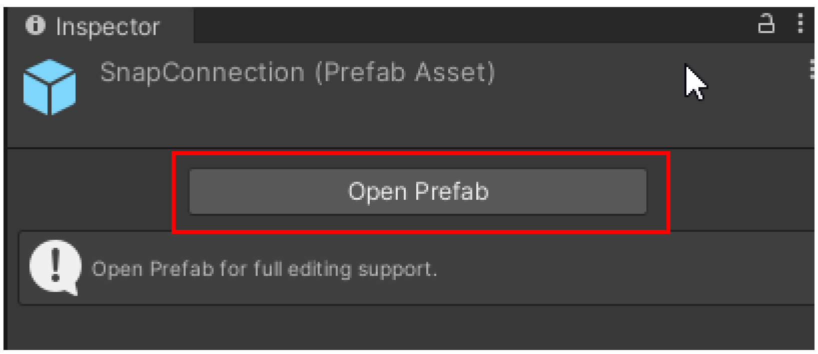
Locks are door prefab with locking support (optionally with different visuals based on your art asset)

We'll setup these locked door prefab in the snap connection

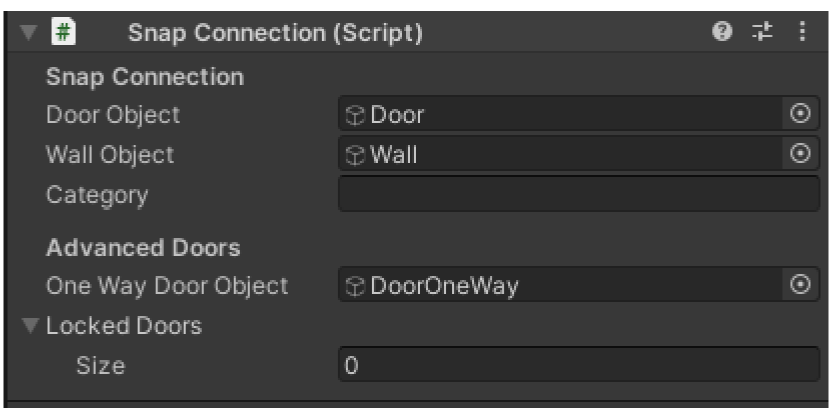
Keys are mapped in the **Theme** editor. Locks are mapped in the **Snap Connection** prefab, since they deal with doors

Open Snap Connection Prefab Open the `Snap Connection` prefab that we created in the earlier section



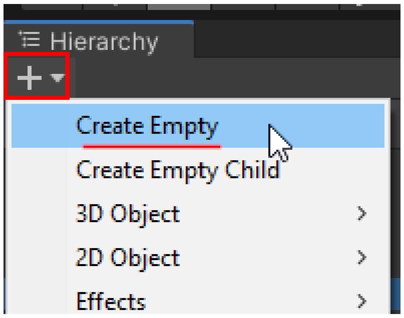


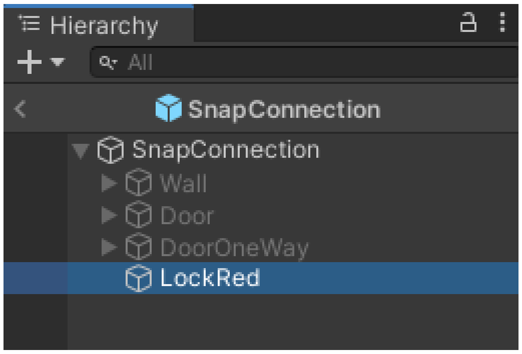
Inspect the properties of the prefab



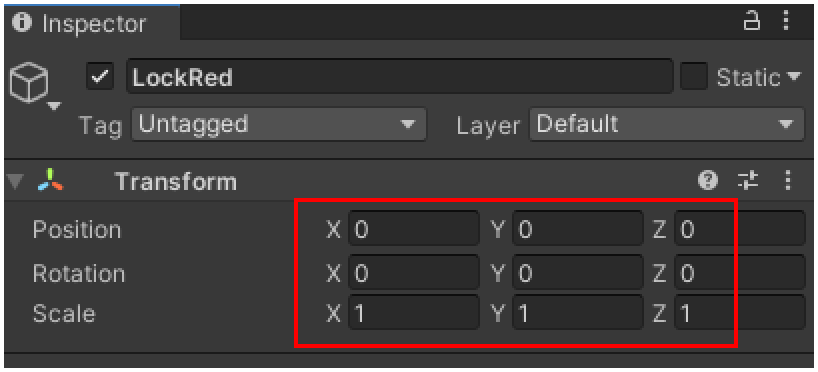
Setup Locked Door Similar to the way we've previously setup the one-way door, we'll setup two locked doors

Create an empty game object and rename it to `LockRed`. Place it along side the `Wall`, `Door` and `DoorOneWay` game objects



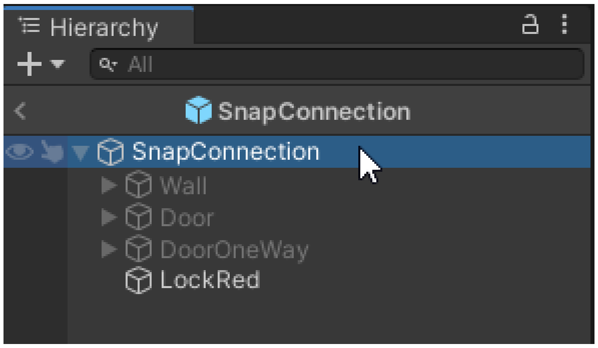


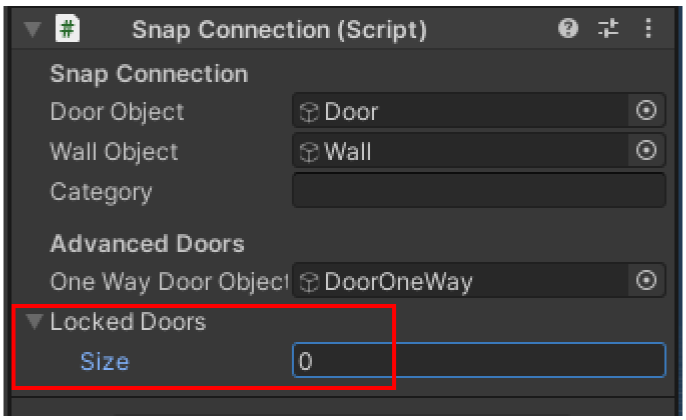
Reset the transform



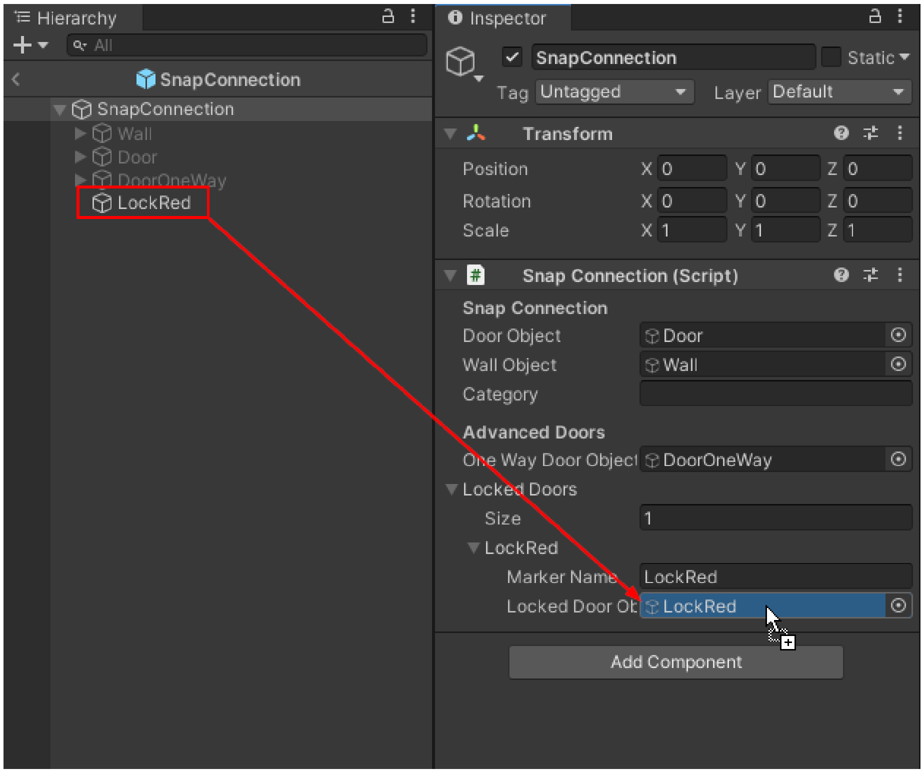
We'll now register this game object as a locked door with the marker name as `LockRed`

Select the root Snap Connection object and inpect the properties

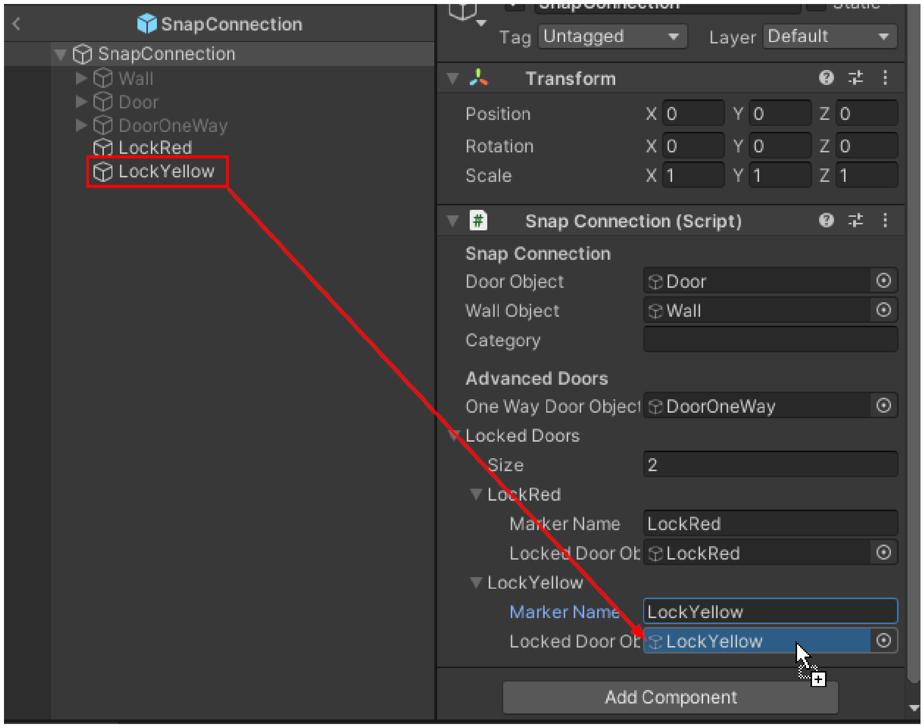




Add a new entry to the *Locked Doors* array and set the marker name to LockRed and assign the game object to it

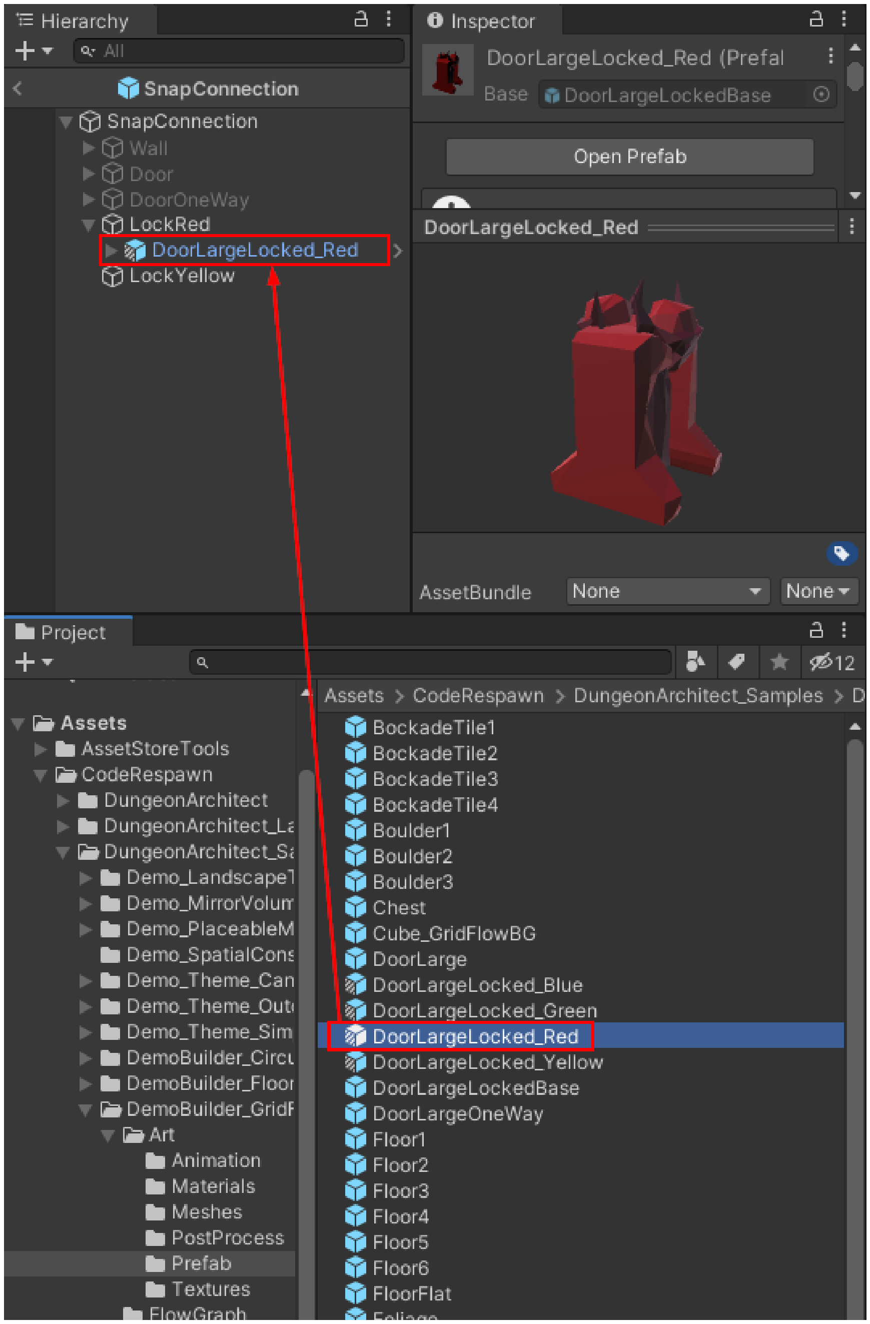


Repeat the same for the yellow lock. Set the marker name to `Lock-Yellow`, create a new gameobject, reset the transform and assign it

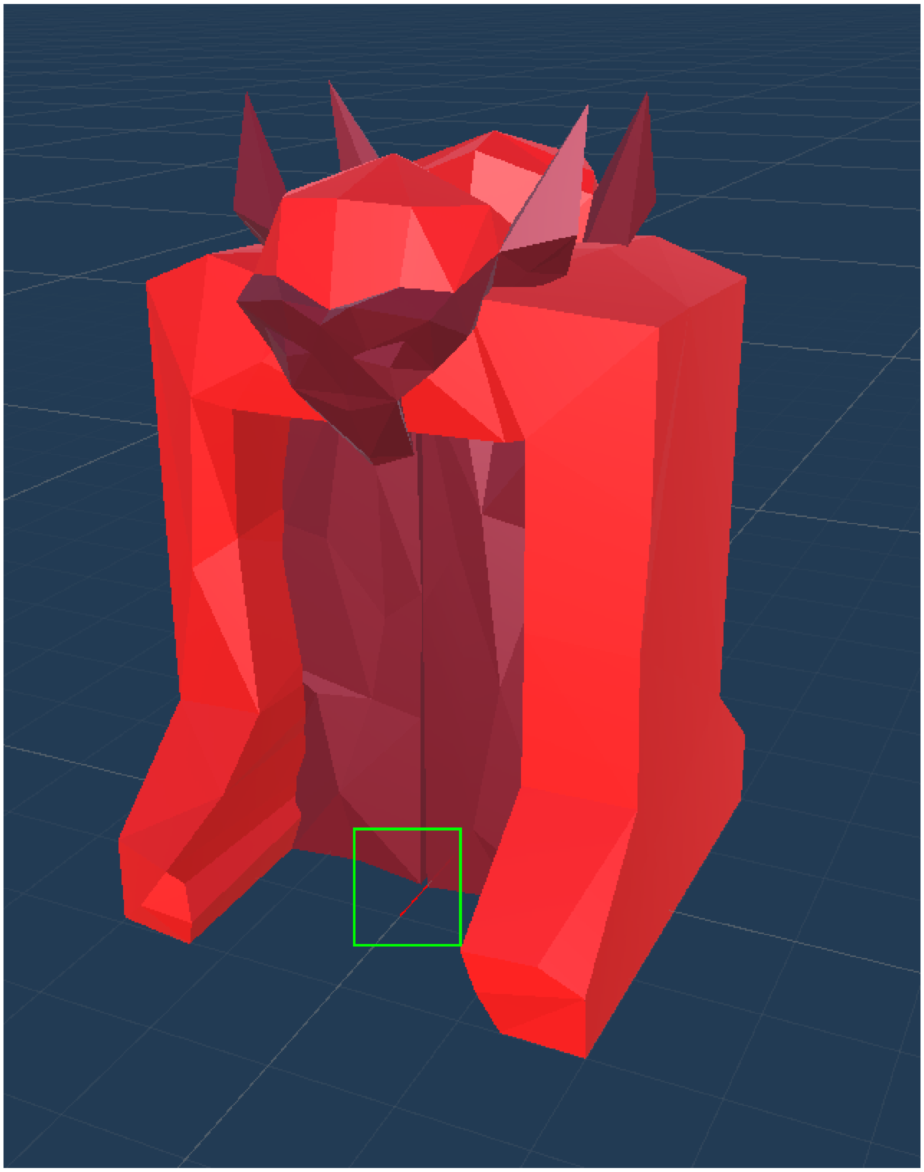


We used the names `LockRed` and `LockYellow` since this is what we assigned in the *Create Key/Lock* node in the flow graph

Assign Lock prefabs Navigate to `Assets\CodeRespawn\DungeonArchitect_Samples\DemoBu` and drop in `DoorLargeLocked_Red` prefab as a child of `LockRed` game object

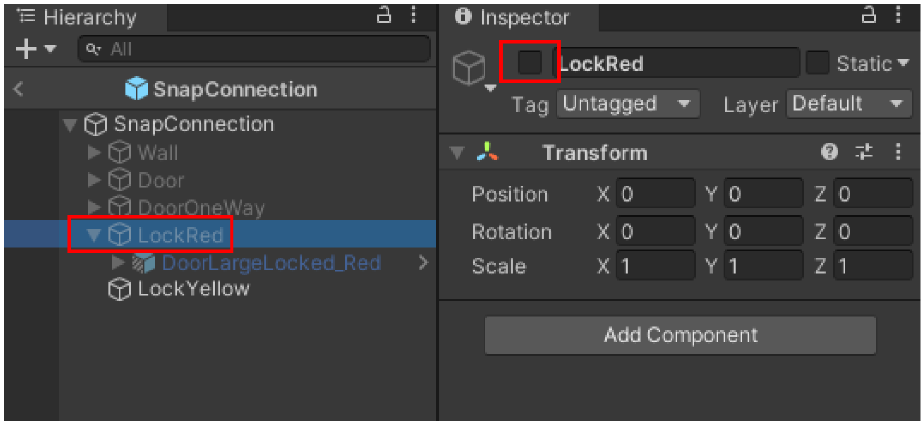


The alignment rules are the same like other doors. Reset the transform of the red door prefab and it should align correctly (red line should face outwards and the origin is in the bottom-center)

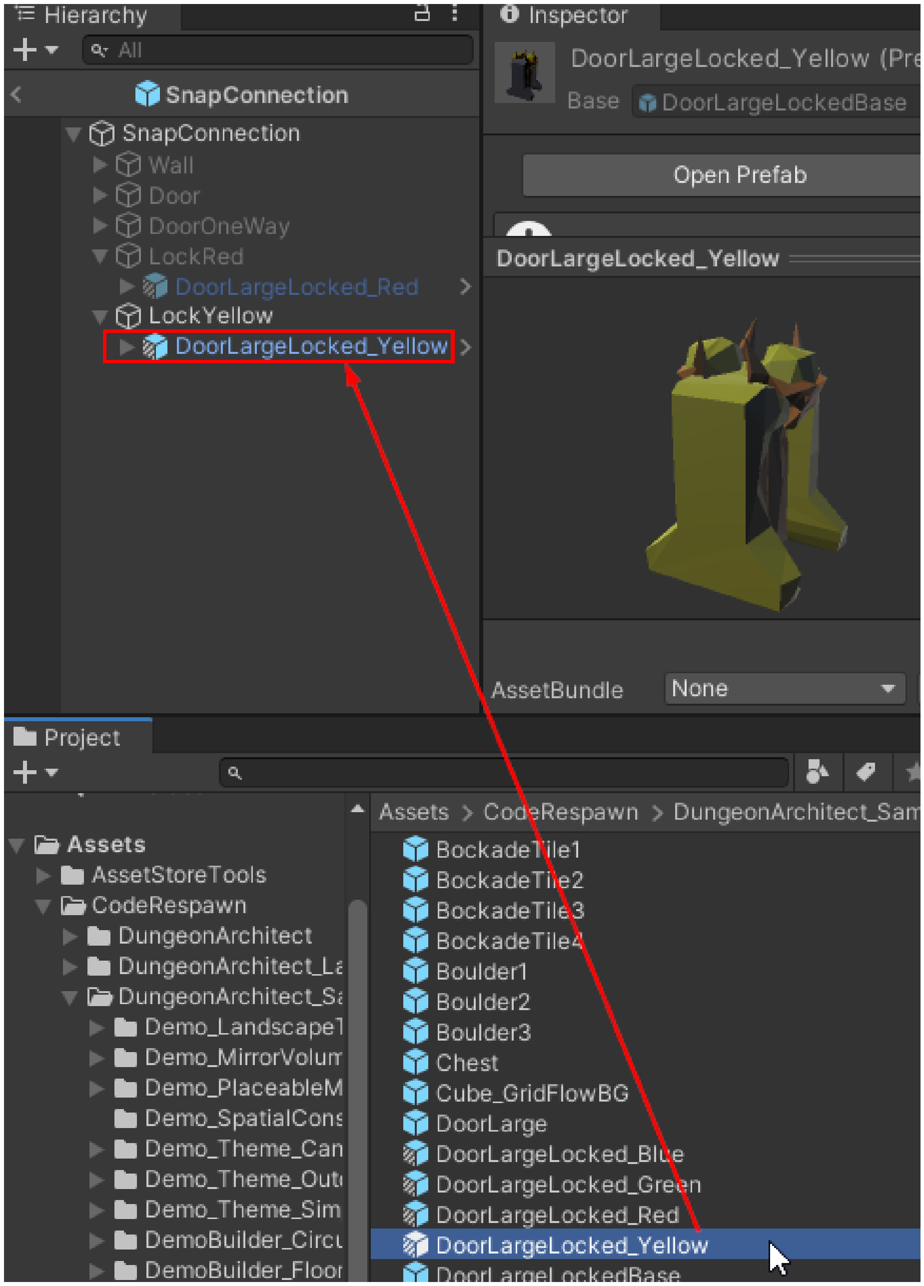


Do the same for the yellow lock.

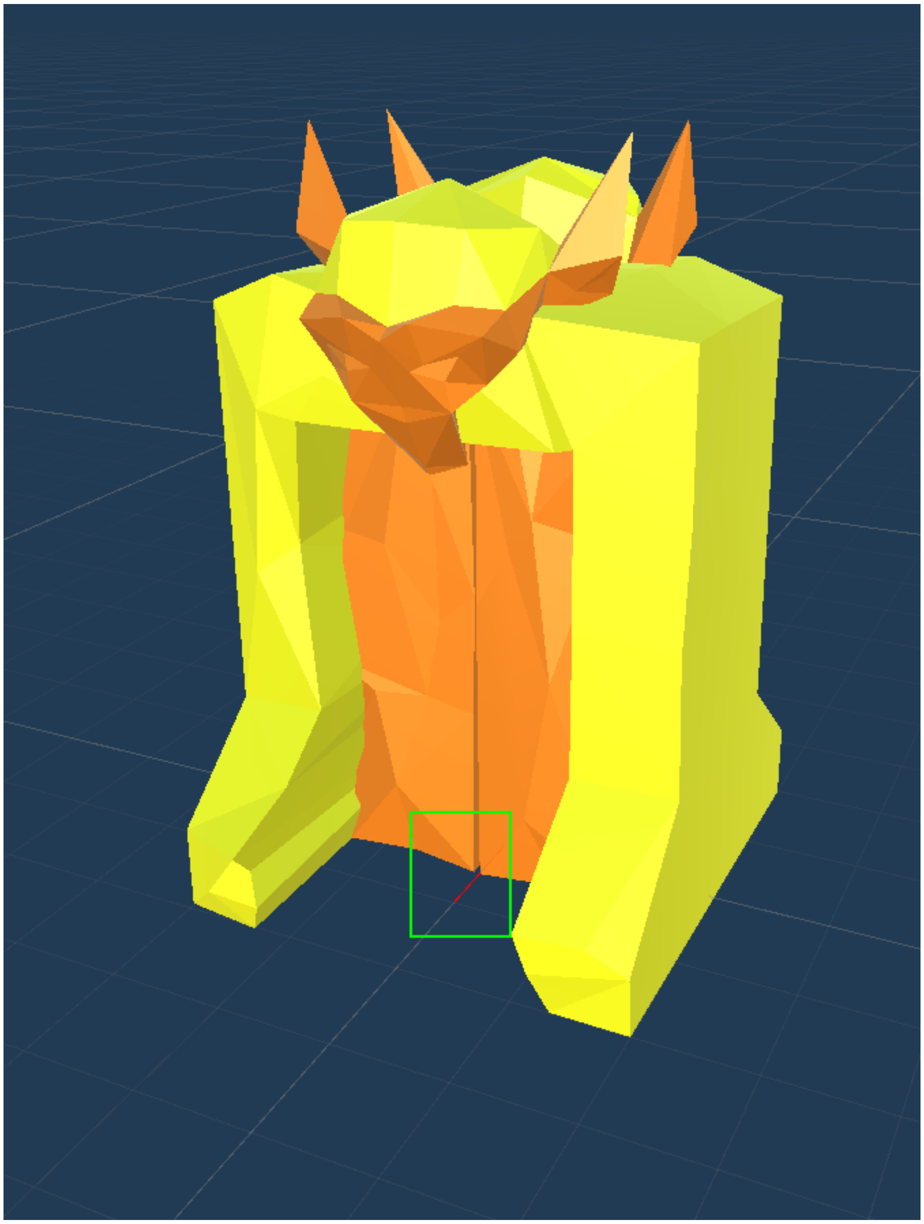
Hide the `LockRed` game object



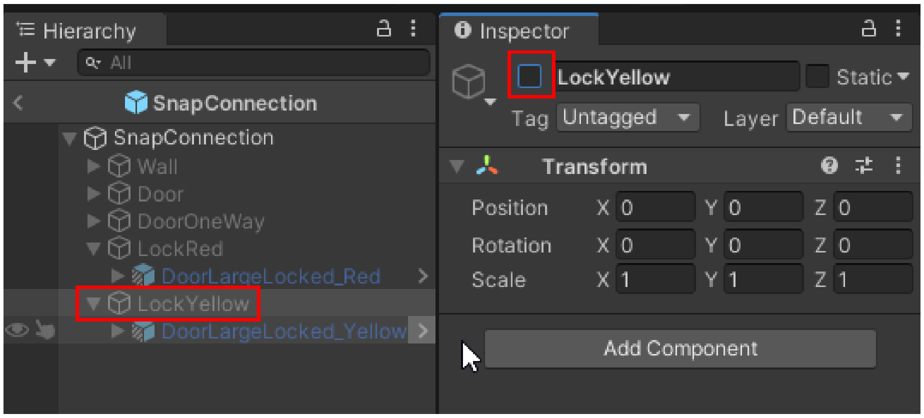
Drop in the `DoorLargeLocked_Yellow` prefab under the `LockYellow` gameobject



The alignment rules are the same like other doors. Reset the transform of the yellow door prefab and it should align correctly (red line should face outwards and the origin is in the bottom-center)



Hide the `LockYellow` game object



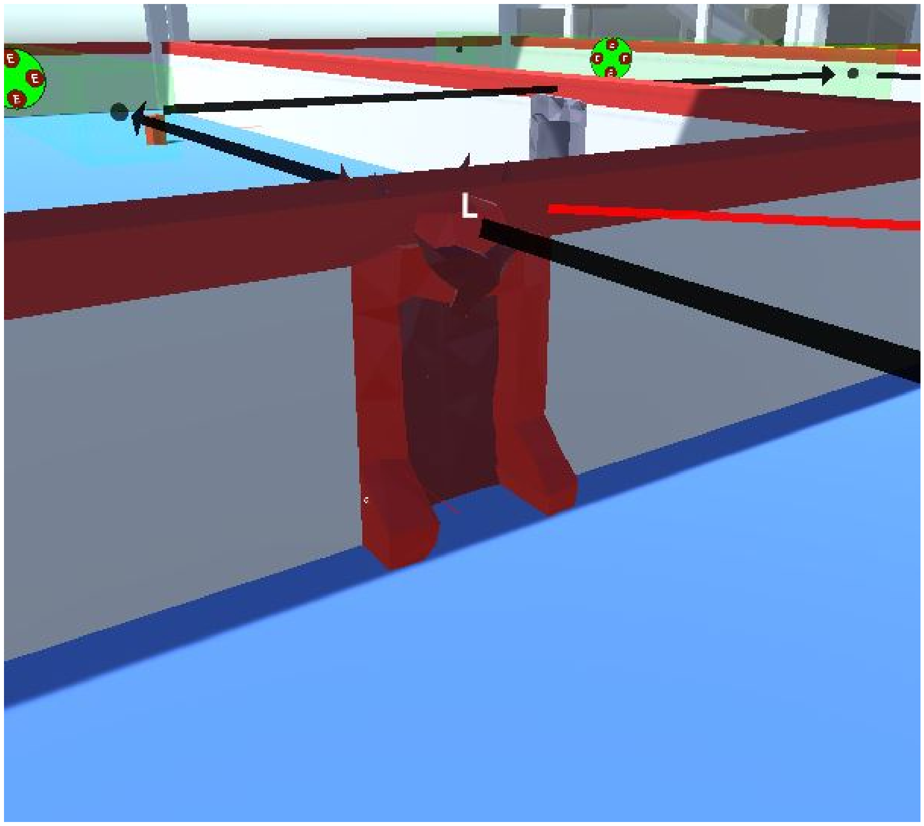
Make sure you hide the outermost `LockRed` and `LockYellow` game objects and not the `DoorLargeLocked_Red` and `DoorLargeLocked_Yellow` game objects

There is no difference between the `DoorLargeLocked_Red` and `DoorLargeLocked_Yellow` prefabs other than the visuals and they are both variants of the same parent prefab. The flow system will automatically supply it with the valid key ids when the dungeon is built and a common logic is used to check if we can open the door. More on this later

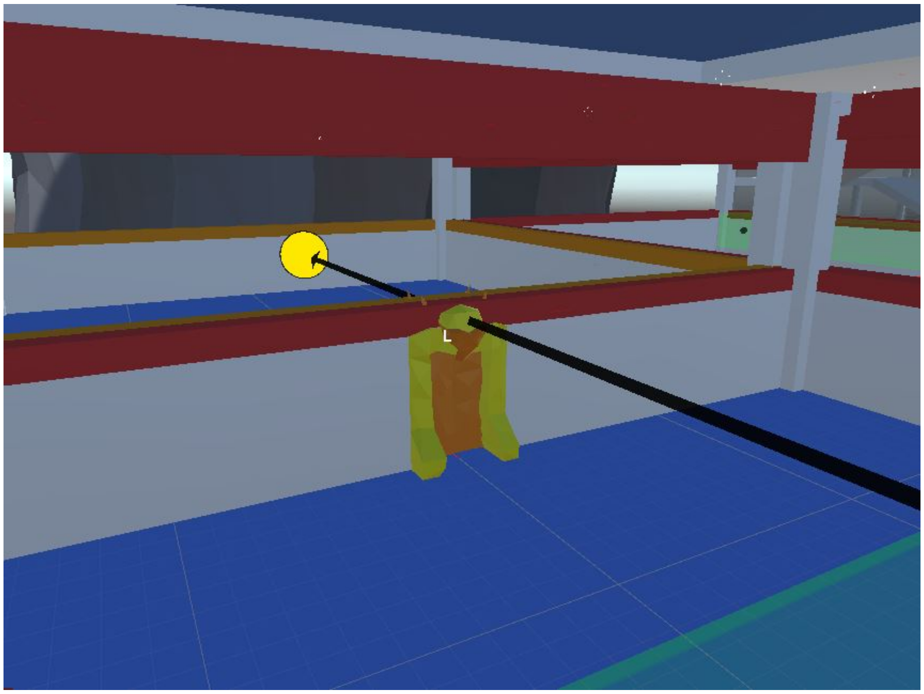
Build dungeon

Open the scene where we previously set up our dungeon. Rebuild the dungeon

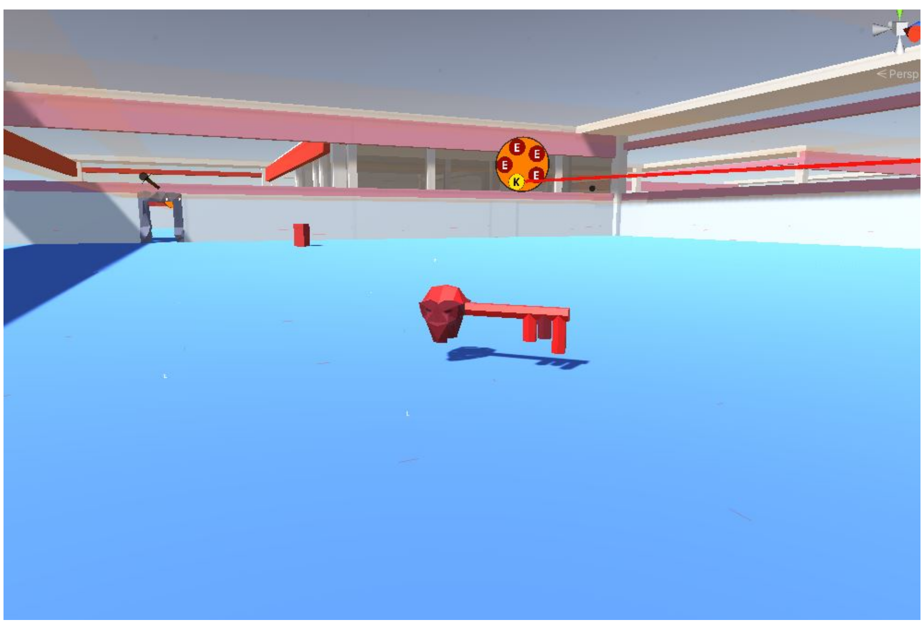
Red lock guarding the main path:



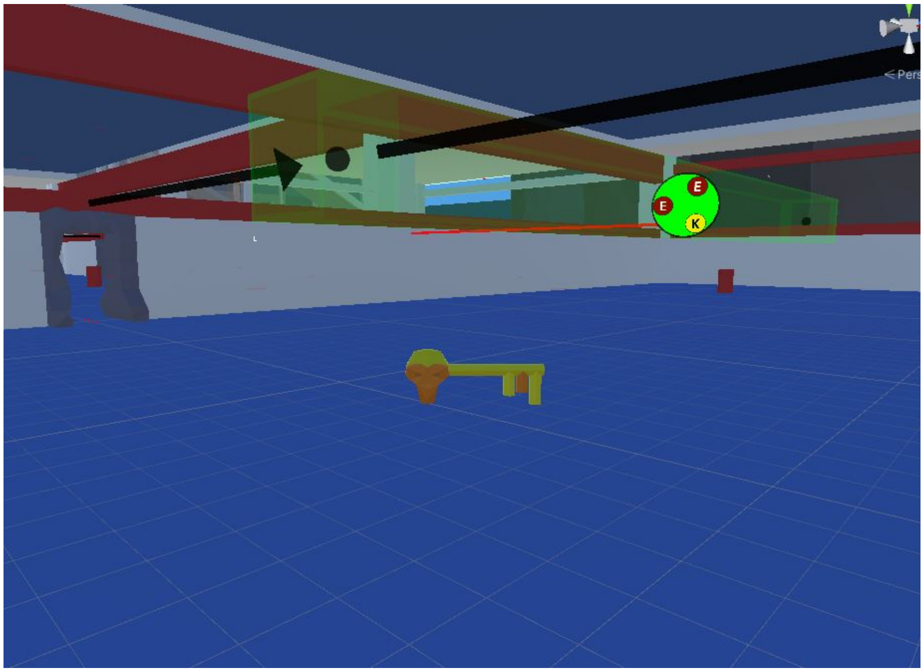
Yellow lock guarding the treasure room:



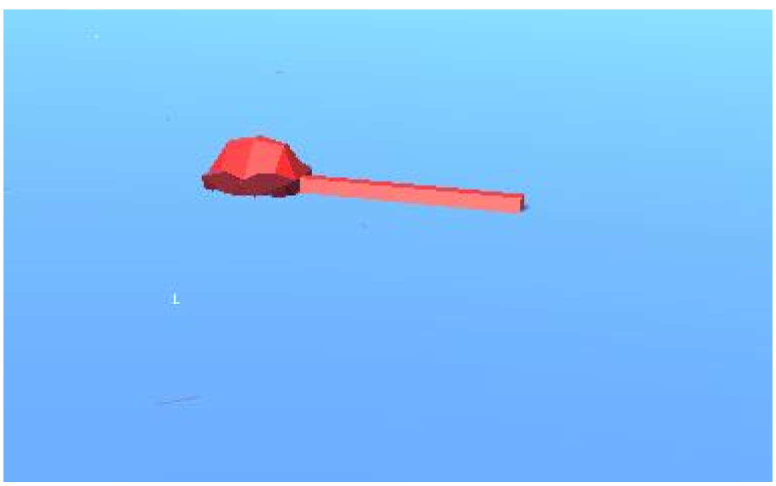
Red Key:



Yellow Key:



If your keys are shown half buried into the ground or below the ground, adjust their offset from the theme editor, or move the placeable marker game object up

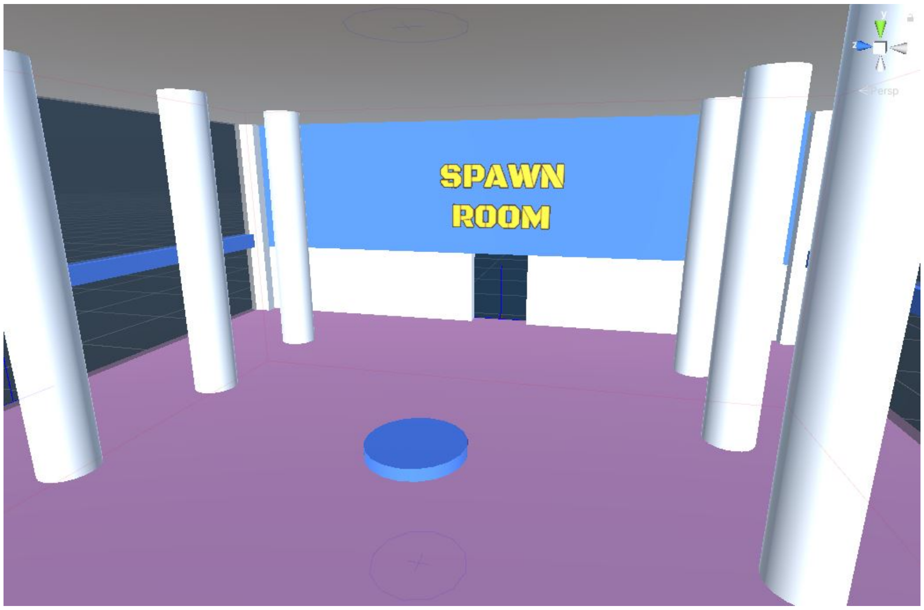


Create Spawn Room

We're going to create a spawn room and place a marker there to spawn our player prefab

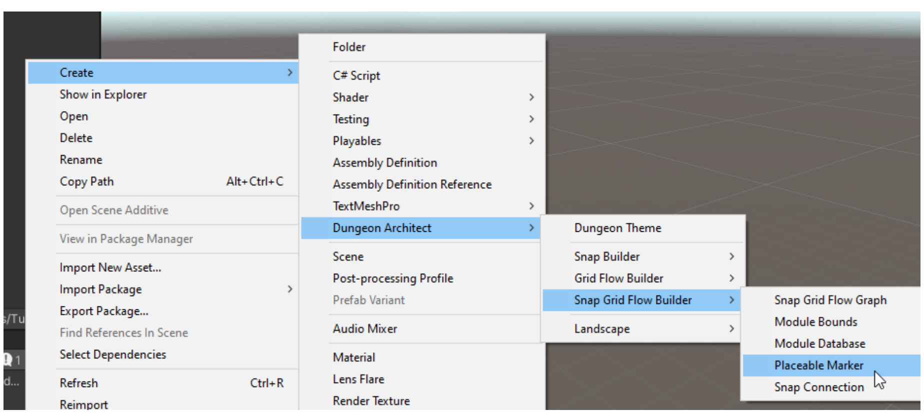
Spawn Room

Create a spawn room module like before and leave a few connection points open

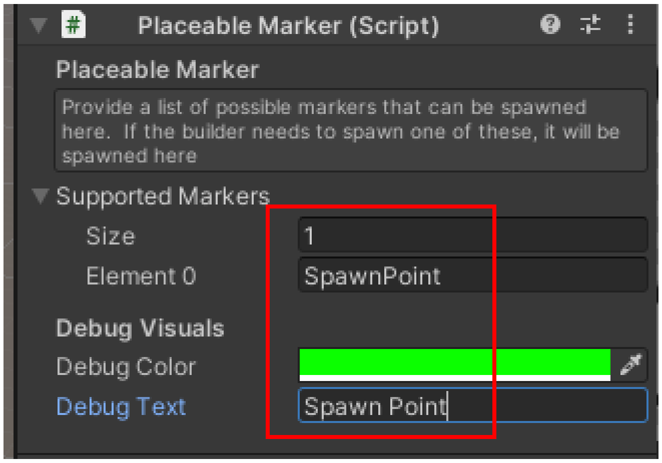


In this example, we've created a `1x1x1` spawn room with a single connection point

Create SpawnPoint Marker We want the theme engine to spawn our player prefab in the spawn room. Create a placeable actor asset named `PM_SpawnPoint`

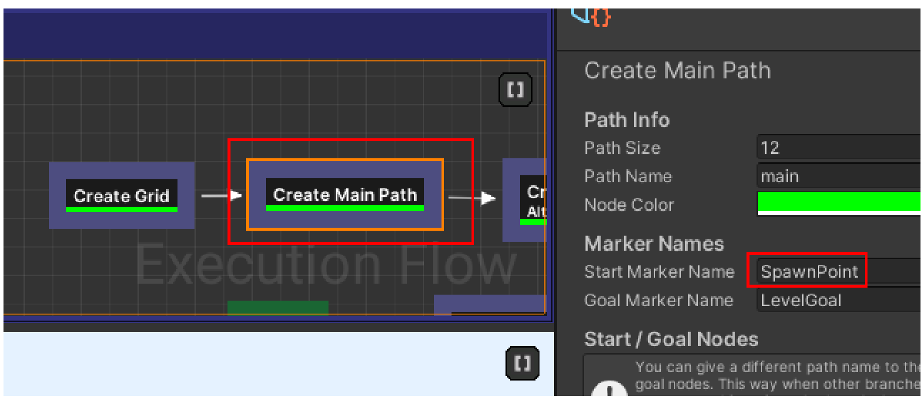


Select the `PM_SpawnPoint` asset and inspect the properties.



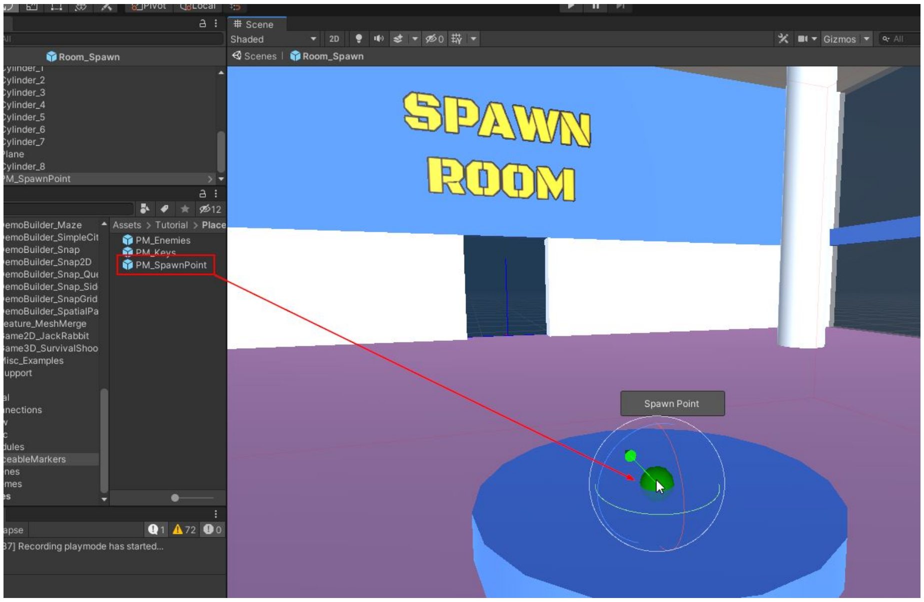
- Add a marker named `SpawnPoint` in the *Supported Markers* array
- Change the debug color to green
- Set the debug text to something descriptive, like `Spawn Point`

We set the marker name to `SpawnPoint` because this is what was specified in the flow graph's `Create Main Path` node



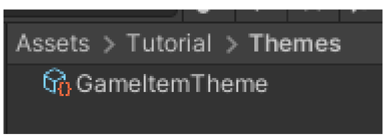
Save and placeable marker prefab

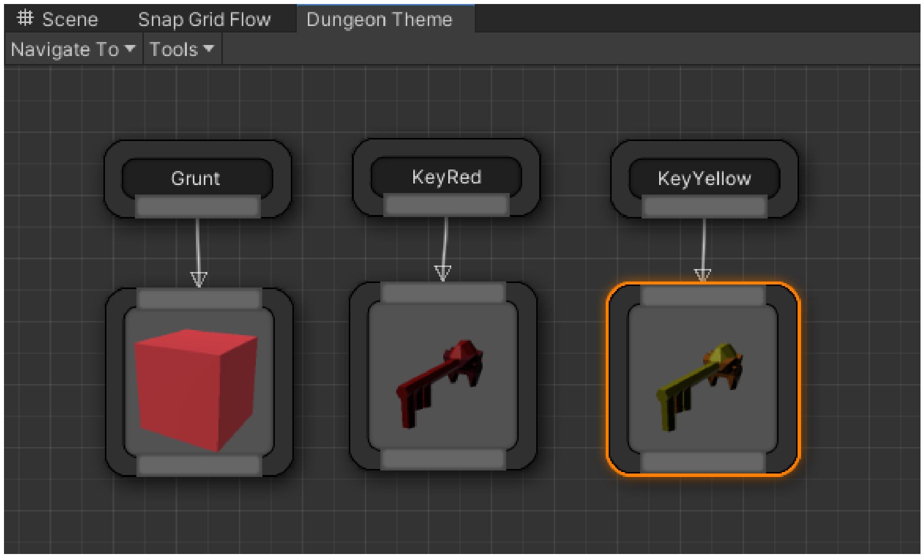
Add SpawnPoint Marker Open the Spawn Room module prefab and drop this placeable marker asset somewhere appropriate



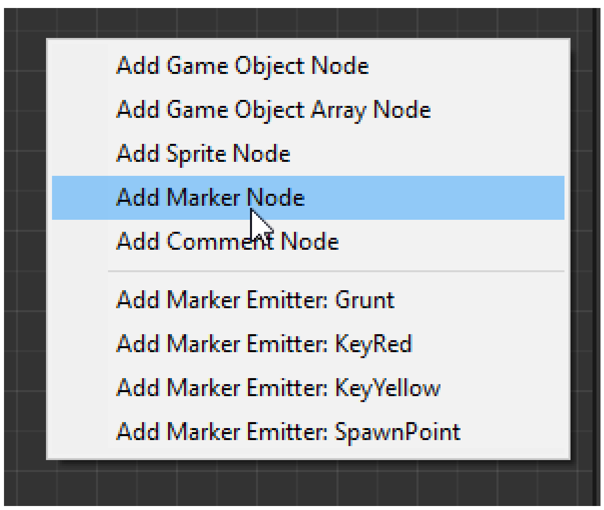
Close the prefab and return to the scene

Add Player prefab Open the theme file we created previously.



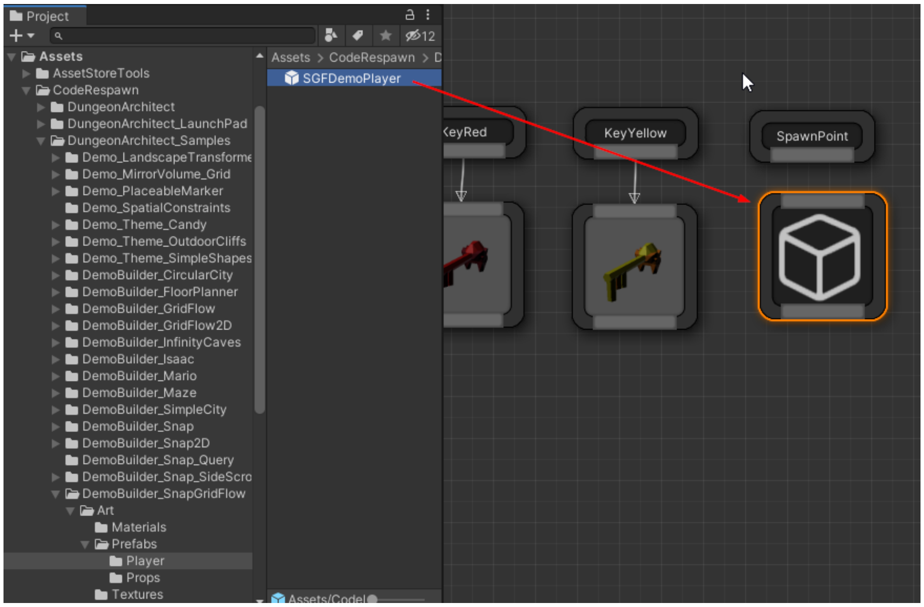


Create a new Marker node and rename it to `SpawnPoint`

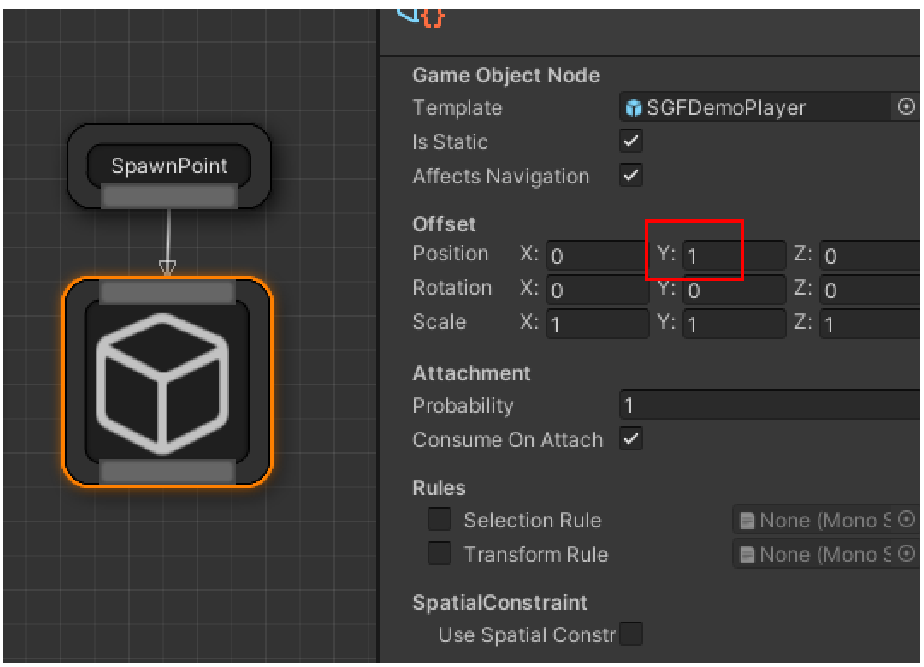


Add a player prefab. We already have a player prefab setup with some fps controls. Drop it from the samples folder Navigate to `Assets > CodeRespawn > DungeonArchitect_Samples`

> `DemoBuilder_SnapGridFlow > Art > Prefabs > Player` and drag-drop `SGFDemoPlayer` on to the theme editor

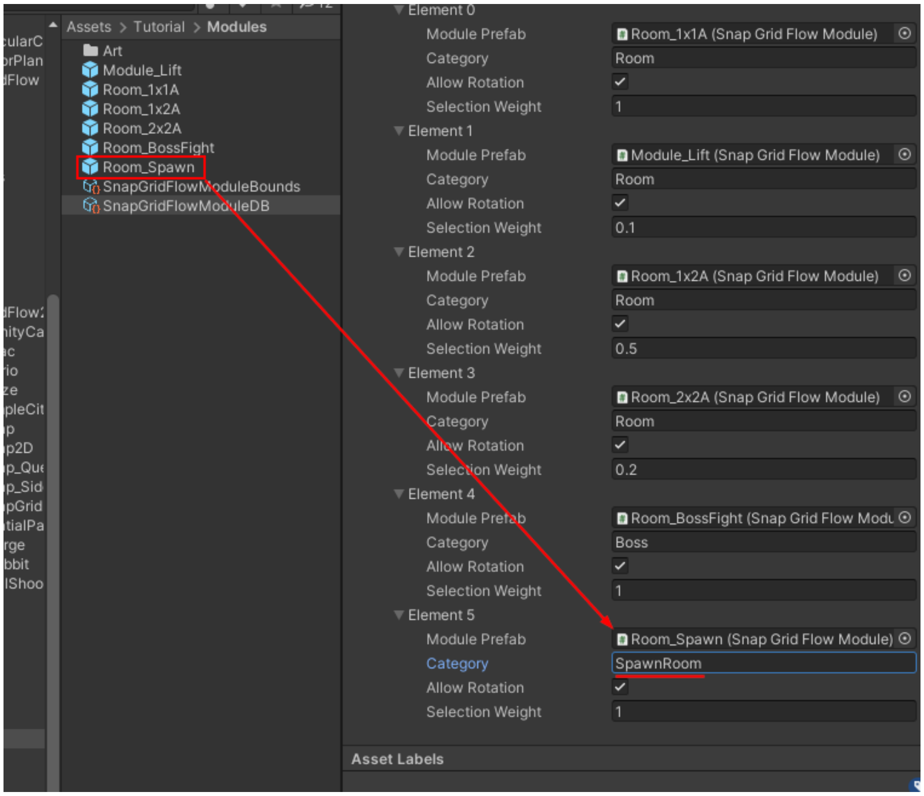


We want the player prefab to spawn 1 unit high (since the placeable markers was placed on the ground)



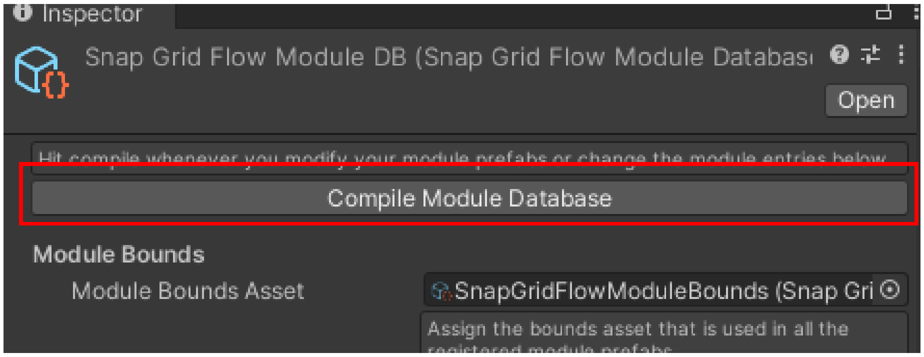
Save and close the theme editor

Register Spawn Module Open up the module database and register this spawn room module



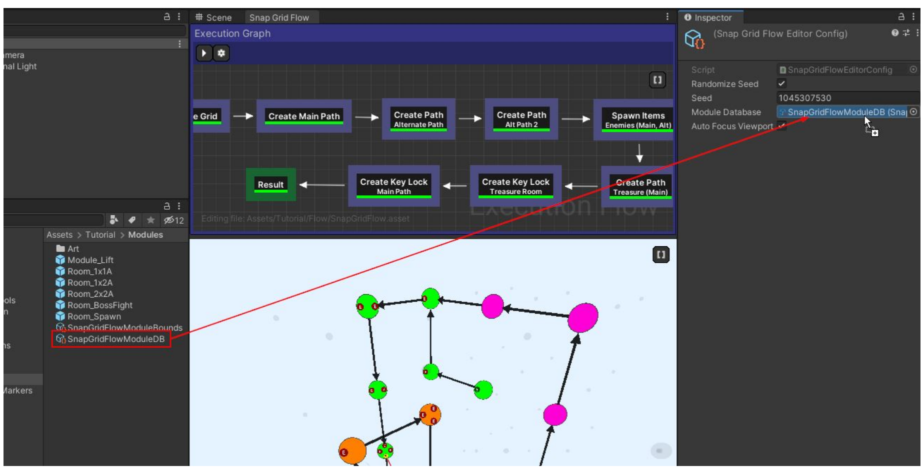
Set the category name to `SpawnRoom`. We'll use this category name in the flow graph shortly, to force it to use our spawn room while building the main path

Recompile the module database cache

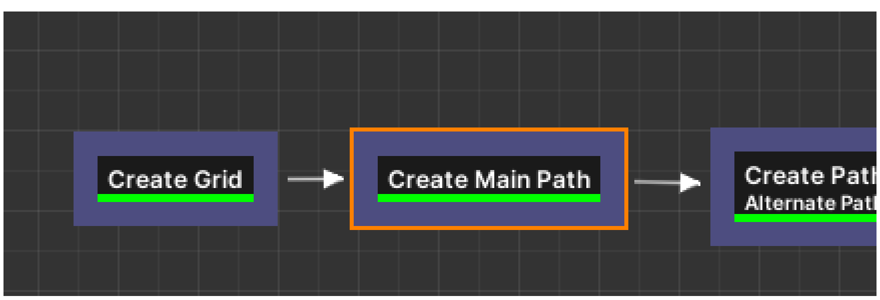


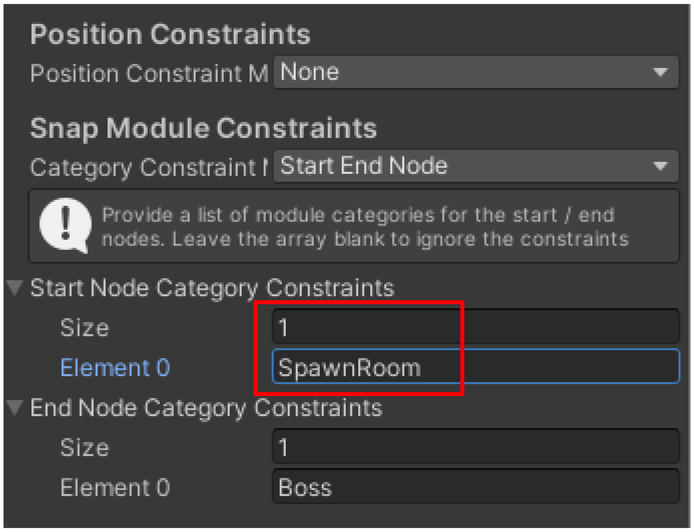
Save the module database

Update Flow Graph Open up our flow graph editor and reassign the Module database in the Editor Settings as before



Select the `Create Main Path` node and inspect the properties





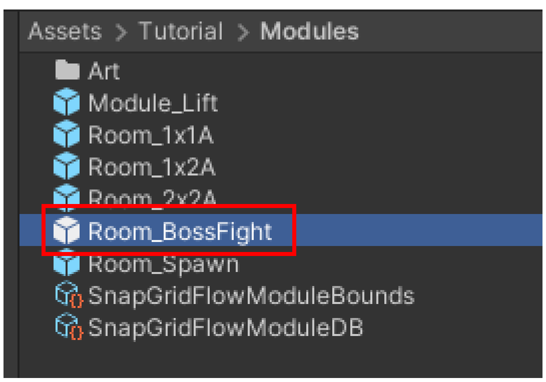
Add an entry to *Start Node Category Constraints* and set it to `SpawnRoom`.

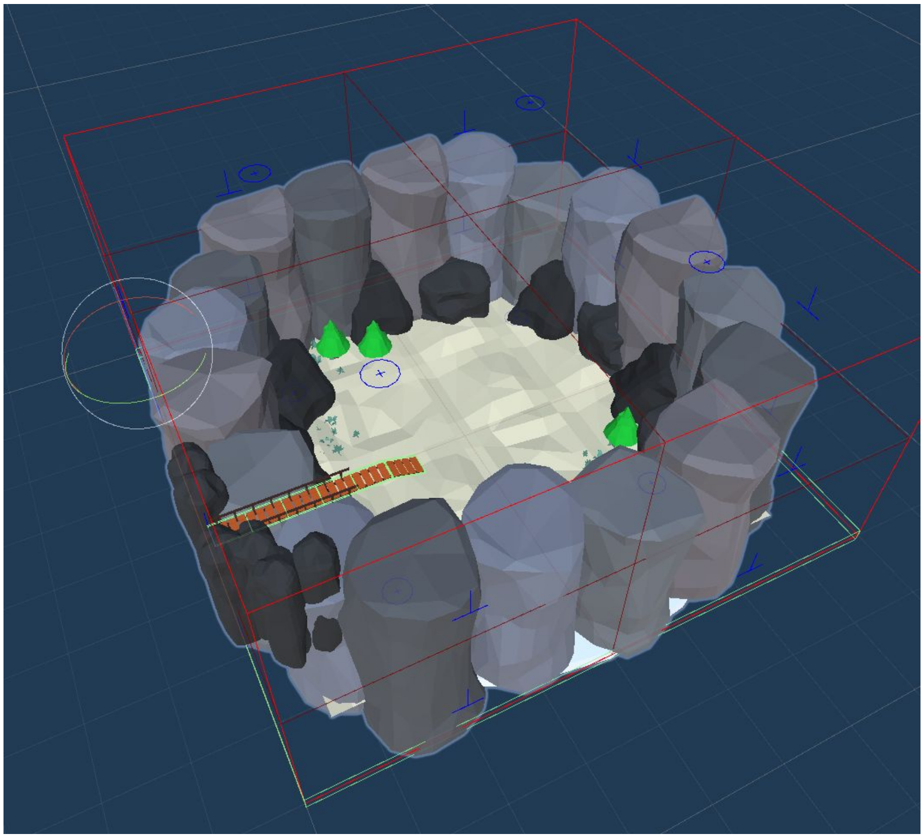
This will make the flow editor choose the start room registered in the module database with the specified category. As of now, we have only one spawn room, feel free to register more modules with the same name to have it randomly pick one spawn room

Hit build in the flow editor and make sure it generates a flow graph correctly

Goal Room

Open up the Goal room prefab we created earlier

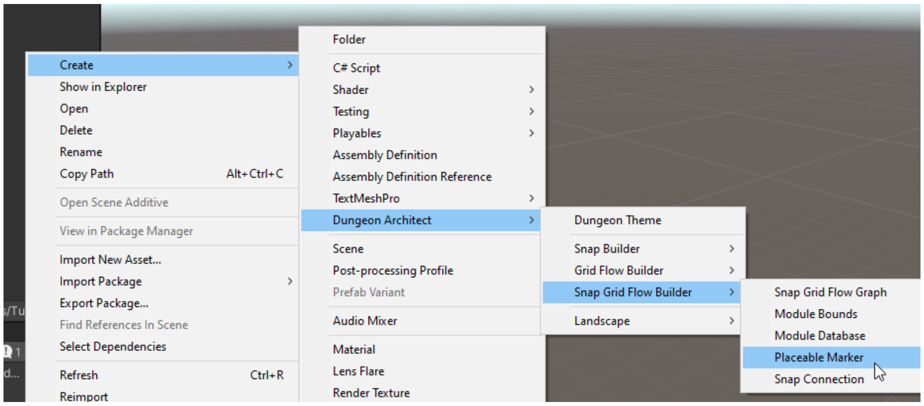


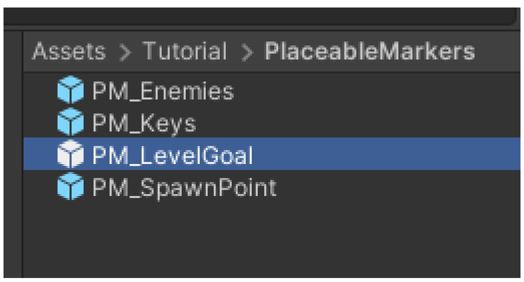


We'll create a placeable marker prefab that supports the marker name `LevelGoal`. We'll then use the theme engine to spawn our level goal prefab (e.g. it could be an artifact that takes you to another dungeon)

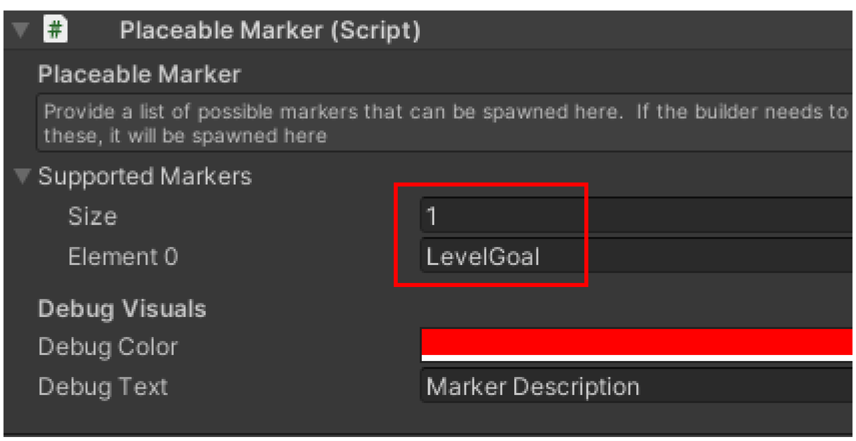
This step is optional and you can skip this section if you don't want to spawn the Level Goal prefab from the theme engine. You may directly place the level goal object inside the goal room module

Create LevelGoal Marker Create a Placeable Marker asset and name it `PM_LevelGoal`

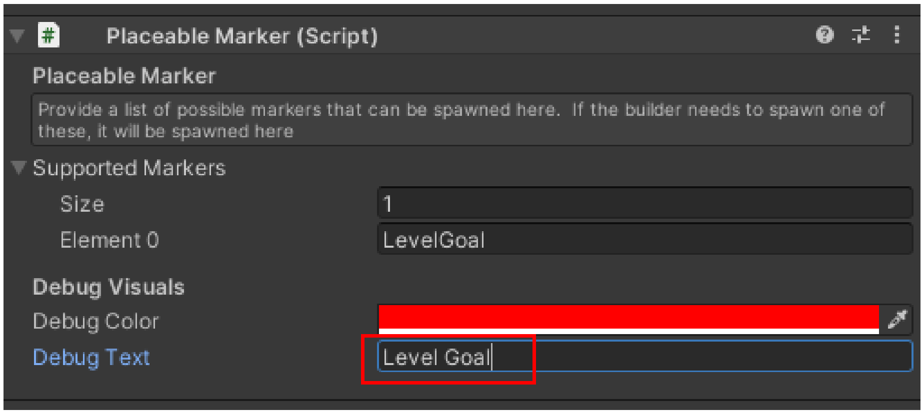




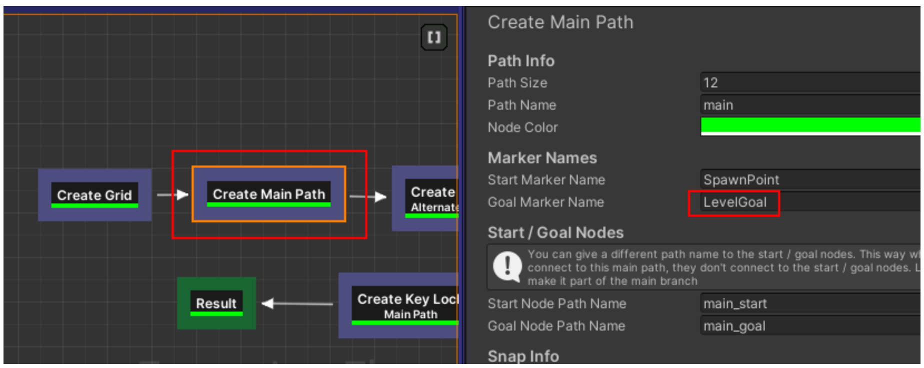
Inspect the properties. Add `LevelGoal` as a *Supported Marker*



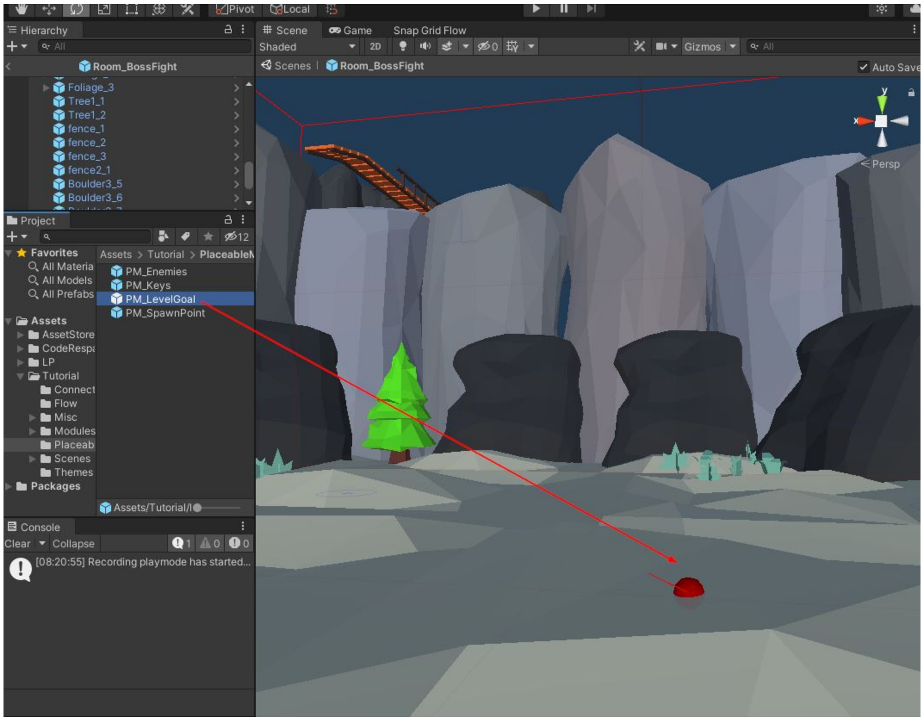
Change the *Debug Text* to something descriptive



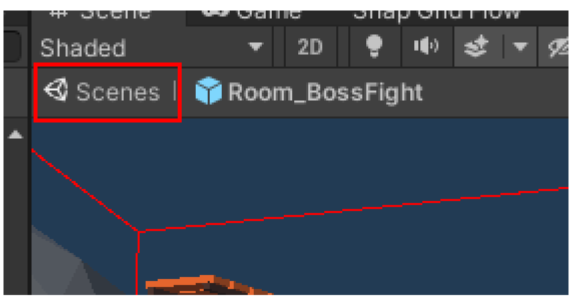
We created a marker with the name `LevelGoal` since this is what we specifed in the flow graph's *Create Main Path* node



Add LevelGoal Marker Add the `PM_LevelGoal` marker to the goal module. Drag and drop it somwhere appropriate in your goal module

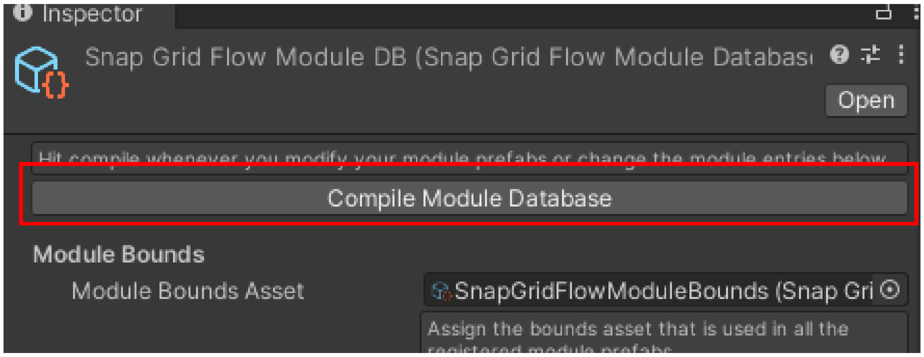


Close the goal room prefab and return back to the scene

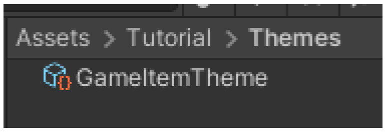


Since we’ve modified the markers in the module prefab, we’ll need to recompile the module database cache

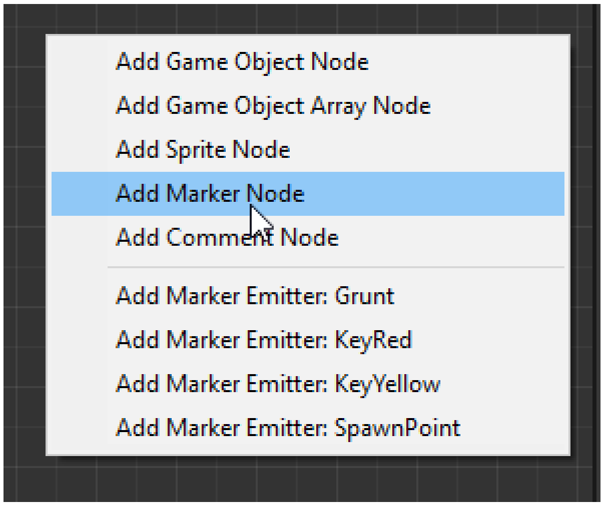
Open the module database and click `Compile Module Database` button

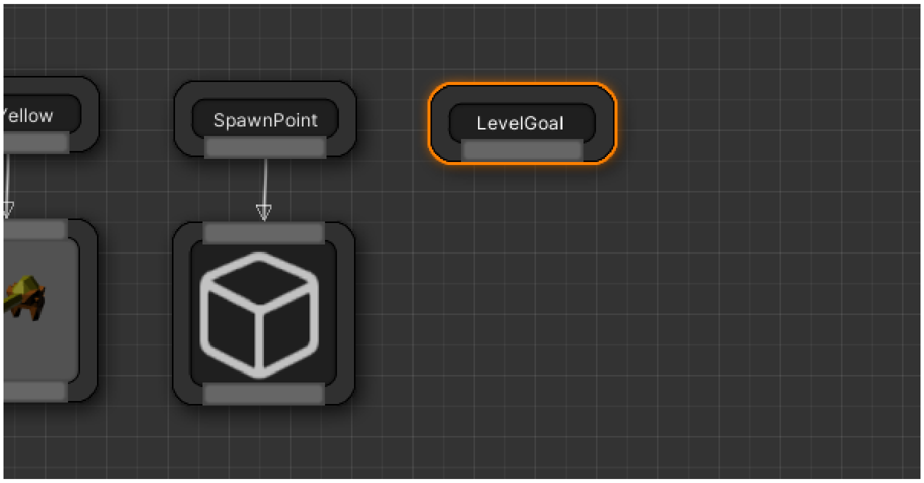


Add LevelGoal Prefab Open the theme file we created previously.

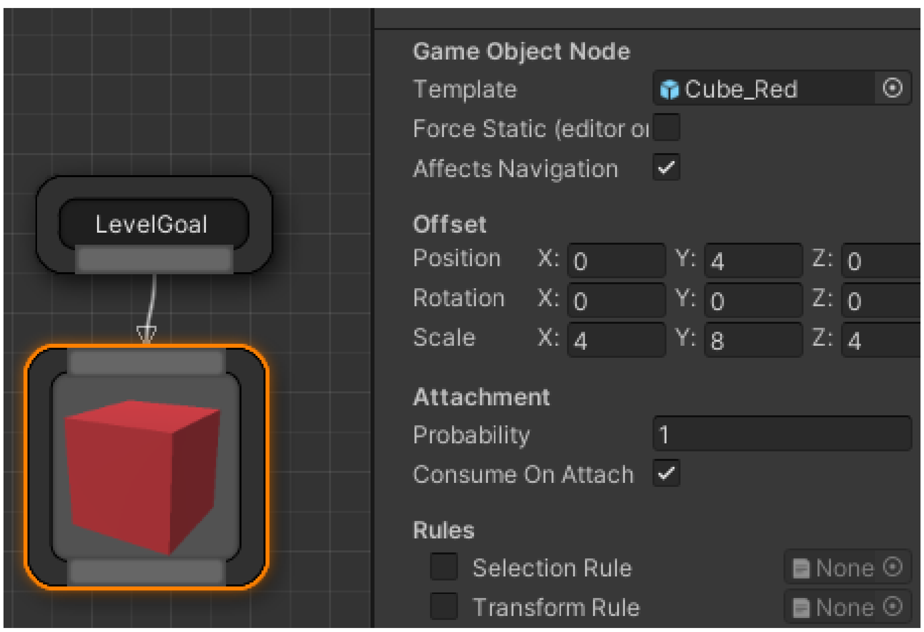


Create a new Marker node and rename it to `LevelGoal`





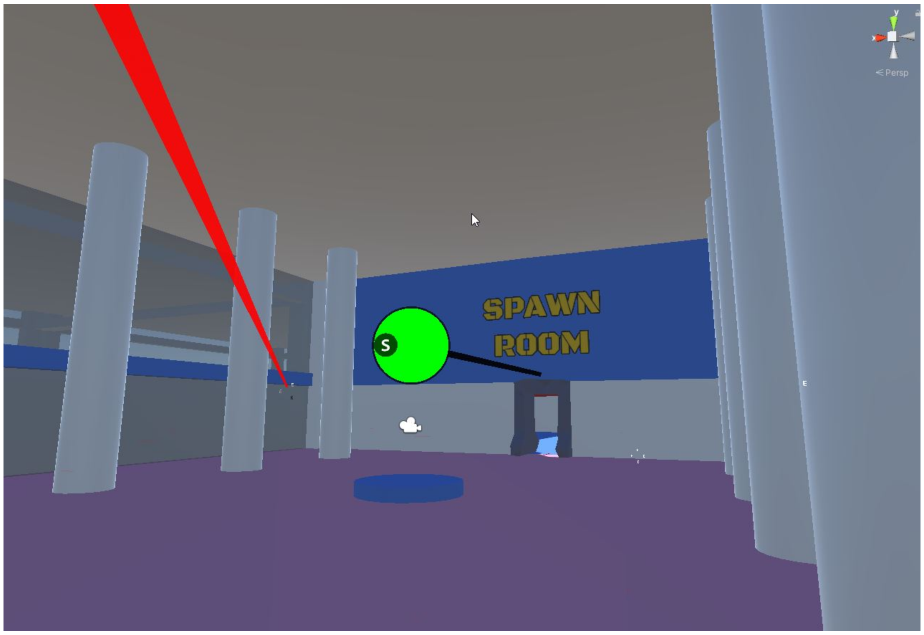
Add your level goal prefab here. We'll use a simple cube for this example that will represent the final boss



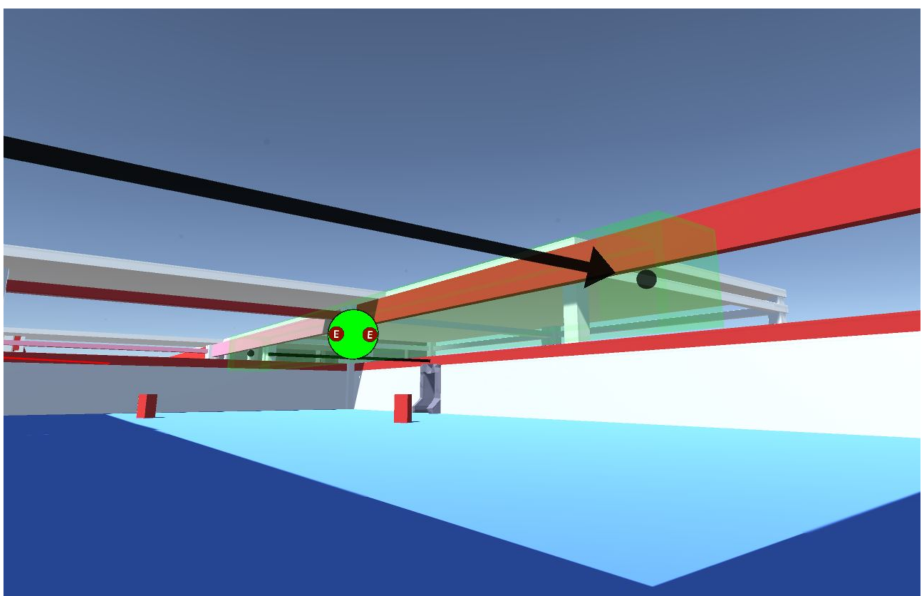
Build dungeon

Open the scene where we previously set up our dungeon. Rebuild the dungeon

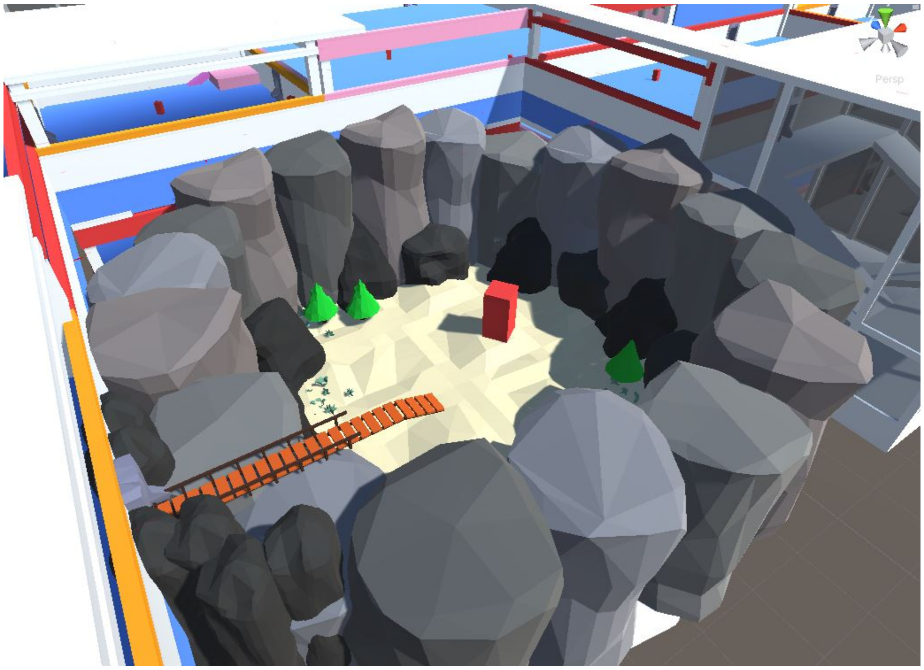
You should see the spawn room, and a `PlayerStart` actor spawned at the correct place



You should be able to play your game and move around with the player prefab



Level Goal prefab:

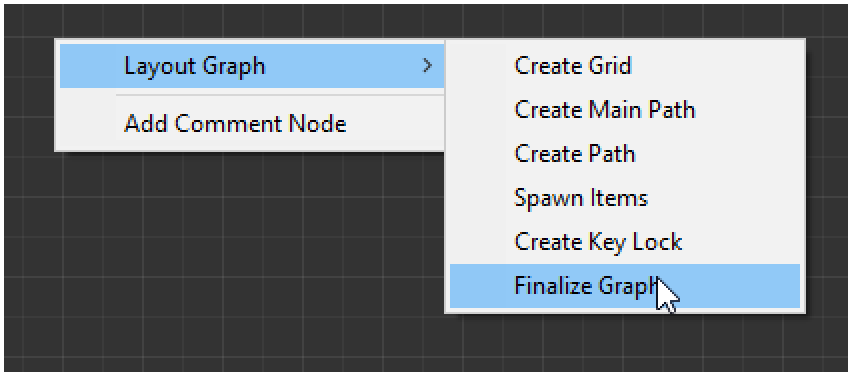


Finalize Graph

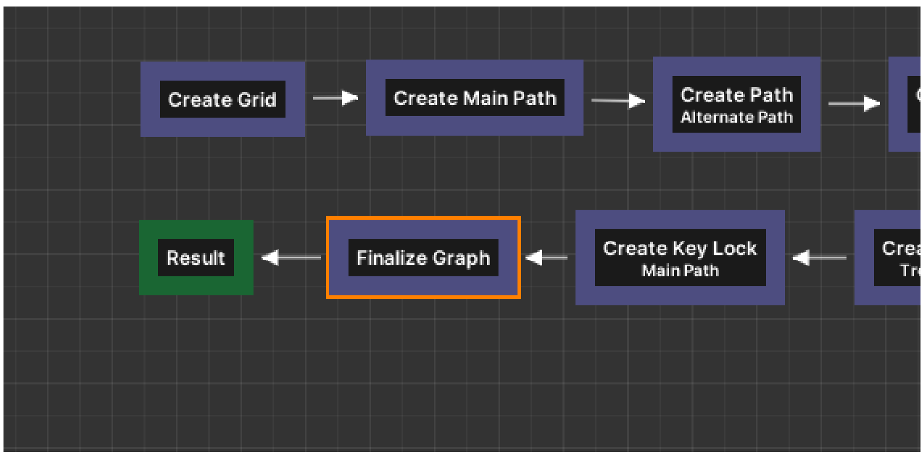
The final node of your flow graph design should always be the `Finalize Graph` node. This node does the following: * Strategically promote some doors to `one-way` doors. This is done to keep the player from bypassing locked doors by entering from another nearby door. This may also be done to keep the player from entering another path from the opposite direction. It will always create a playable level * Remove unused links from the layout graph

Add Finalize Node

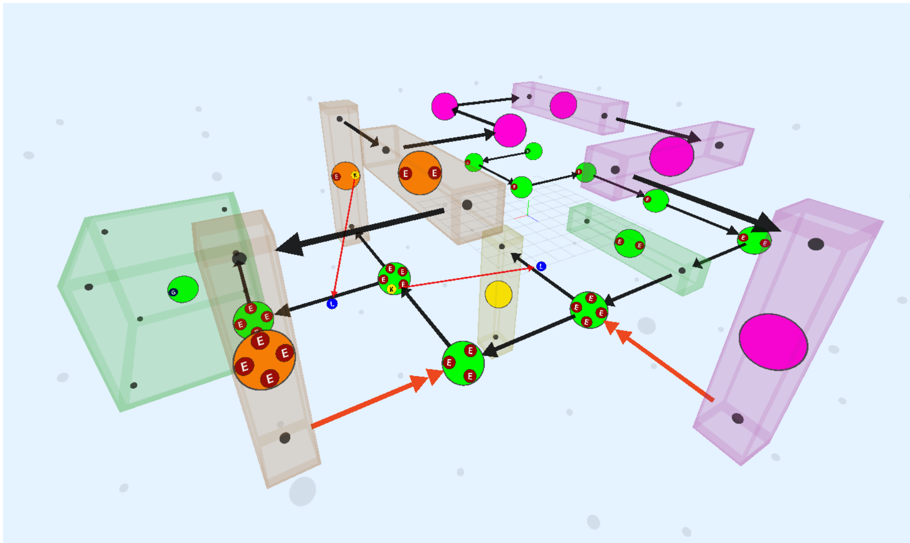
Open the flow graph we designed earlier and add a `Finalize Graph` node



Link it before the `Result` node as shown below

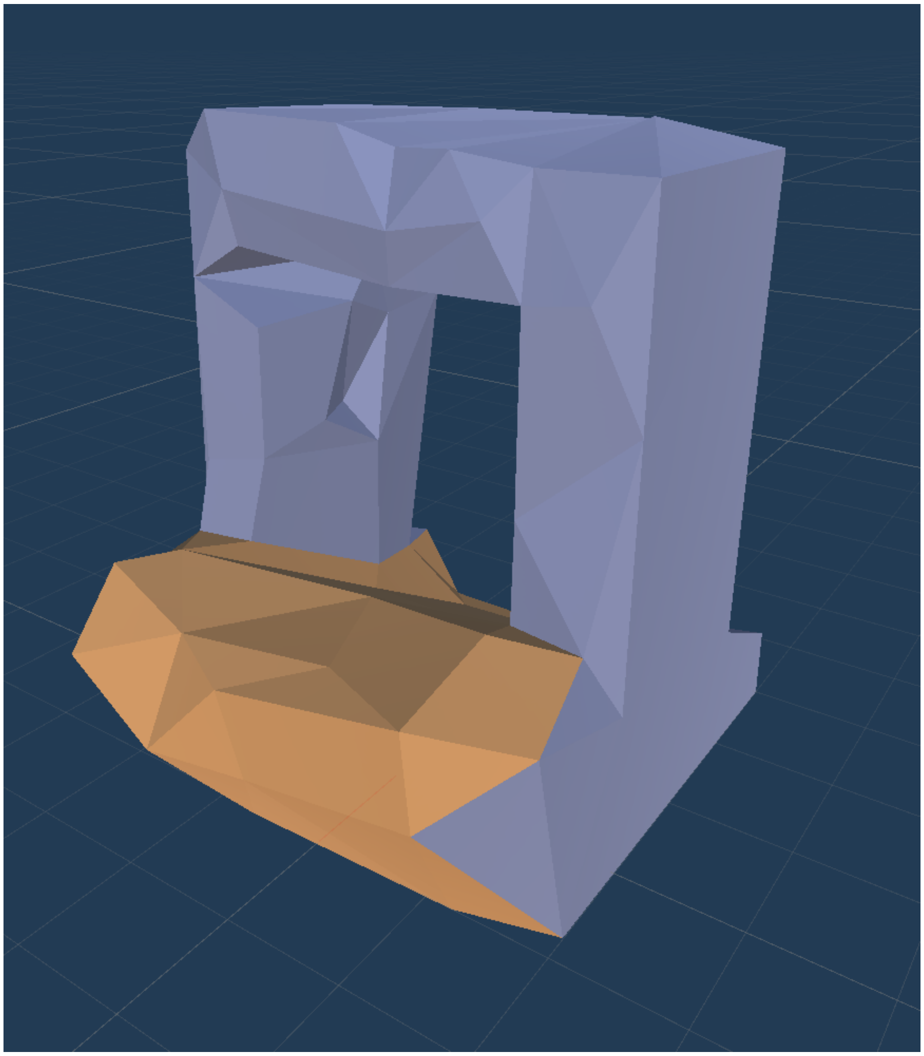


Build the graph and have a look at the layout graph



The orange double head arrows indicate one-way doors

We have already specified a one-way door asset in the snap connection prefab previously

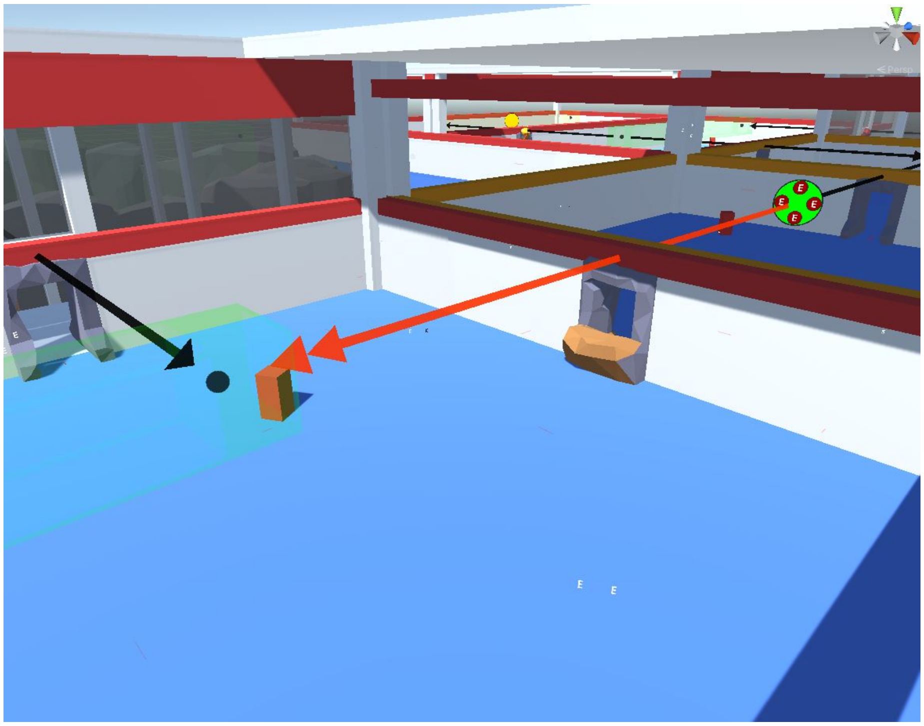


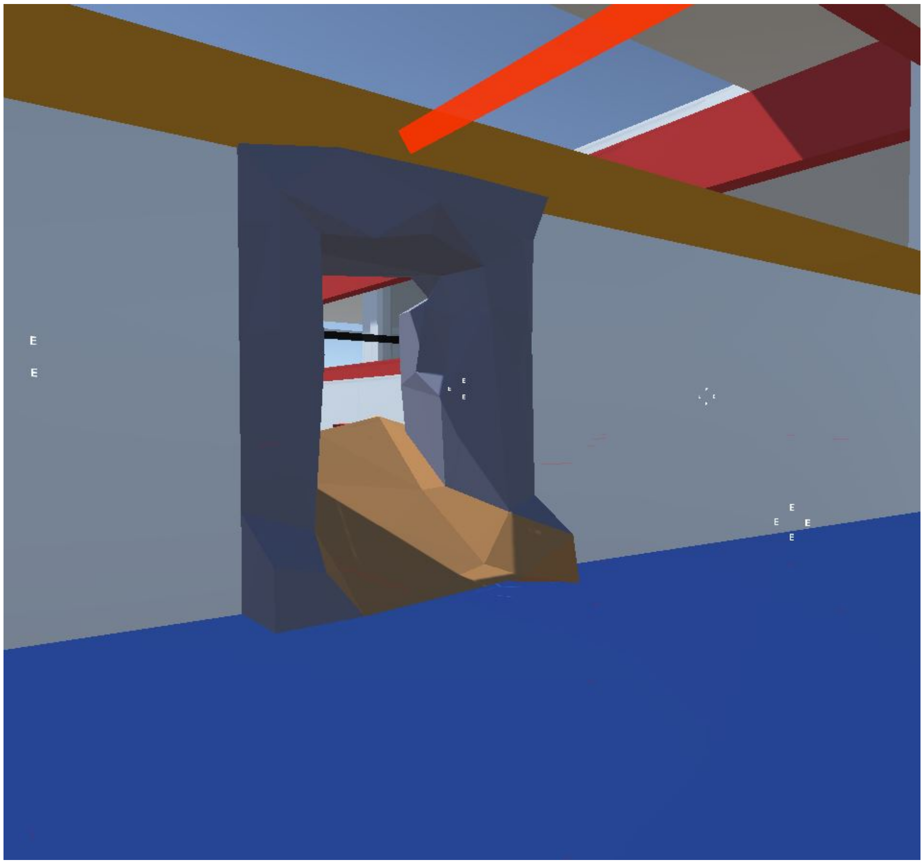
This one-way door prefab will be used in those locations

Build the dungeon

Open the scene where we previously set up our dungeon. Rebuild the dungeon

You should see one-way doors spawn where needed





We have now created a fully playable level and this wraps up our dungeon flow design. Feel free to add more path or play around with your own design

In the next section, we'll look at how to set up gameplay where we will build a random dungeon at runtime, move the player to the spawn room and have the player character (first person, third person etc) move around the map, pick up keys, open locked doors and more

Setup Gameplay

We want to be able to open locked doors after we've picked up the keys. For this to happen, we need to implement a few things: * Pick up the keys when we touch them and place it in our inventory * When we get close to a locked door blueprint, check the keys in our inventory and find out if any one of them can open it * A UI to show the keys in our inventory

Player Inventory

You may create your inventory in any way you like. A simple inventory is implemented for the sample character

Check the `Inventory` class in this path: `Assets\CodeRespawn\DungeonArchitect_Samples\Demo`

It contains four inventory slots. Each inventory slot contains an item id and a preview texture to show the item in the UI

There's also a `PickableItem` class in this path: `Assets\CodeRespawn\DungeonArchitect_Sample`

Pickable Item

If there's an object that you'd the player to pick (e.g. keys), add a `PickableItem` component to it. Whenever a gameobject with an inventory touches this object (e.g. a player), it would add the item id to the next free slot of the inventory and destroy the pickable object

This is how the keys get added to the inventory and disappear when you touch them

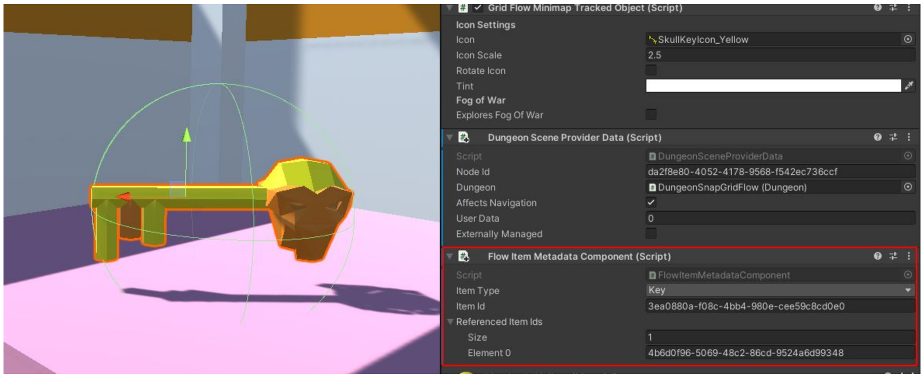
Pickable items have an icon texture that you specify in your prefabs. This texture is added to the inventory slot so it can be shown in the inventory UI

Locked Doors

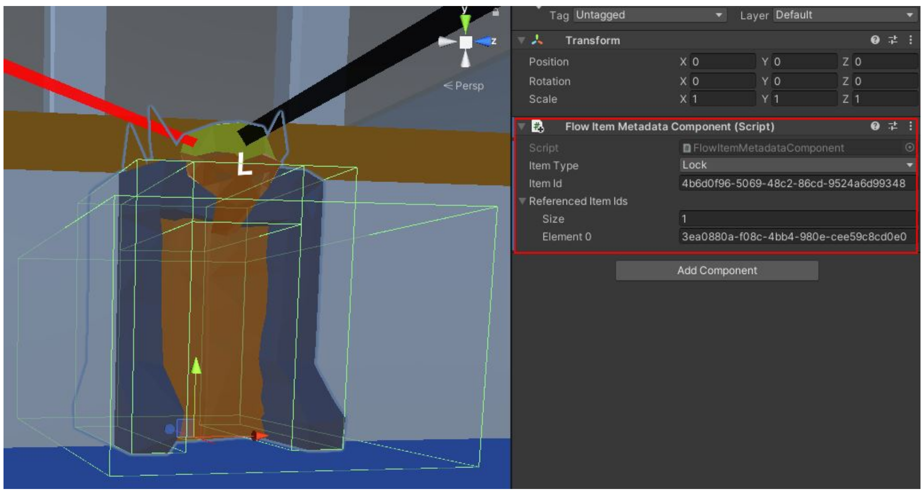
You are free to implement this in any way you like. A sample is provided in `LockedDoor.cs` file here: `Assets\CodeRespawn\DungeonArchitect_Samples\DemoBuil`

When Dungeon Architect spawns the Key and Locked Door prefabs, it will automatically insert a `FlowItemMetadataComponent` to it with the item id. This component will also contain referenced item ids. A key item will reference the lock item and vice versa. So if you are reading a locked door item metadata, the reference item ids will contain a list of key ids that can open this lock

Key Metadata:



Lock Metadata:



Notice how the key references the lock id and the lock references the key id

This component was added automatically by the flow framework when the dungeon was built

When ever we pick a key, the key item ids are added to our inventory. When a player enters the trigger volume of a locked door, we simply check if the inventory contiains any of the key ids that this lock can open

```
bool CanOpenDoor(Collider other)
{
    var inventory = other.gameObject.GetComponentInChildren<Inventory>();
    if (inventory != null)
    {
        // Check if any of the valid keys are present in the inventory of the co
        foreach (var validKey in validKeys)
```

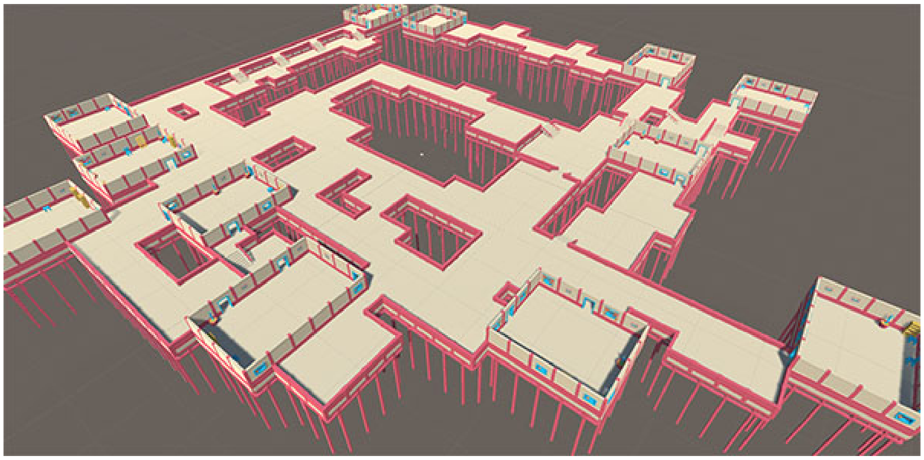
```
        {
            if (inventory.ContainsItem(validKey))
            {
                return true;
            }
        }
    }
    return false;
}
```

Grid Builder

Introduction

Grid Builder

The `Grid Builder` generates a dungeon by scattering rooms across the map and connecting them with corridors. The builder supports height variations (stairs)

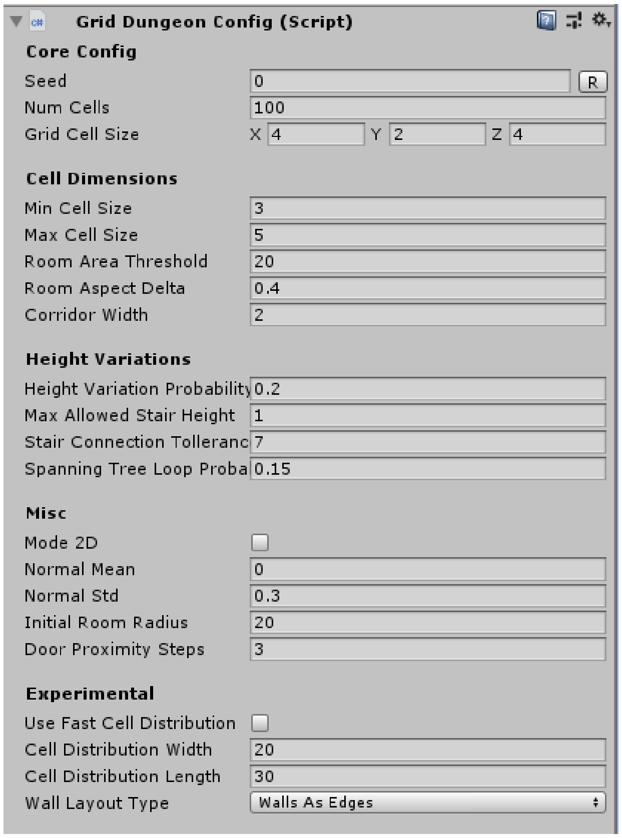


We've used this dungeon builder in the previous section Create your first Dungeon and Design your first Theme

In the following sections, we'll explore more about this builder

Properties

Continuing on the scene created in the section Design your first Theme, open the scene and select the `DungeonGrid` game object and inspect the properties



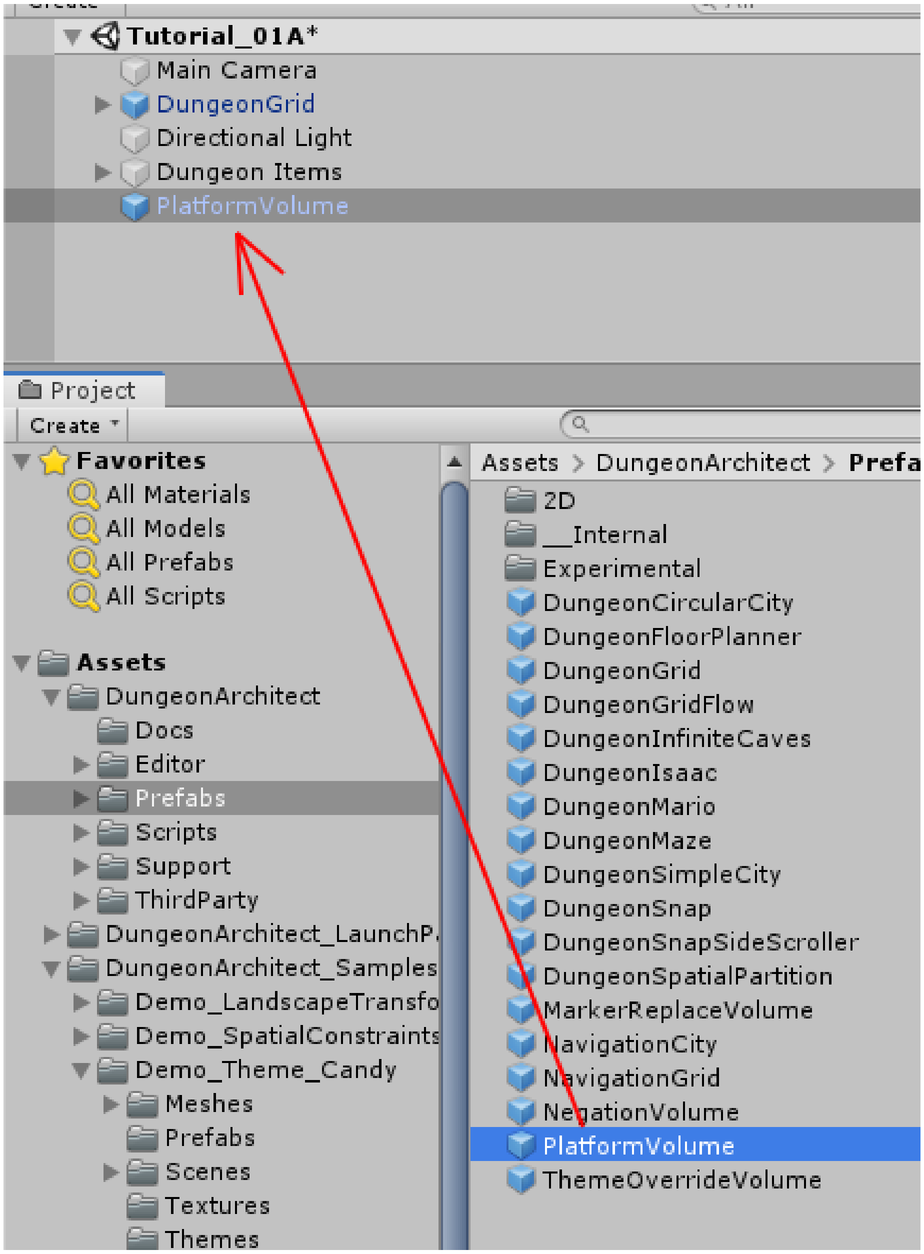
- Change the `Seed` parameter to build a different dungeon layout
- Set the `Grid Cell Size` parameter according to your moduler art asset. If the ground mesh is 4x4 and the stair mesh height is 2, set this to `(4, 2, 4)`
- The dungeon creation method first creates a number of cells (defined by `NumCells`) and spreads them across the scene.
- The size of these cells is define by the parameters `Min Cell Size` and `Max Cell Size`
- Some of these cells will be promoted to Rooms and the rest will be promoted to Corridors. If the cell area is large enough (parameter `Room Area Threshold`), that cell is promoted to a `Room`, otherwise a `Corridor`
- The rooms are connected together with corridors
- Control how much height variation is allowed with `Height Variation Probability`
- A new Stair will not be created between two tiles if there's another stair nearby that if traversed, takes N steps to reach this cell. This value is controlled by `Stair Connection Tollerance`. Bump this number up if you want fewer stairs

- Maximum allowed stair: Determine how high a stair tile can get. If set to one, the builder will create stairs that go only one level up
- Both Rooms and Corridors have a `Ground` marker. Rooms are surrounded by `Wall` markers while the corridors are surrounded by `Fence` marker

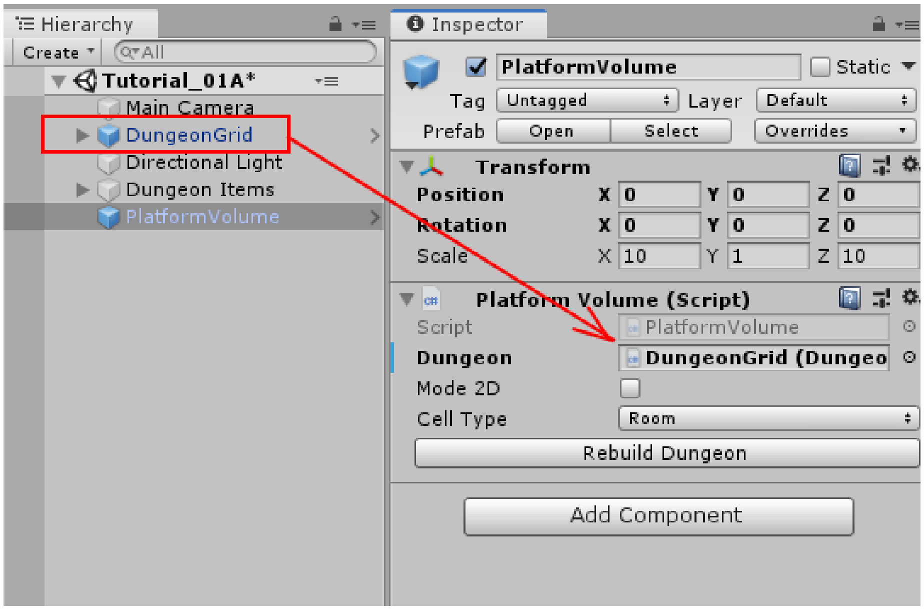
Platform Volume

Platform Volumes let you control the placement of the rooms or corridors. You do this dropping in a platform volume on to the scene and resizing / positioning it on the scene and you room will be built around it.

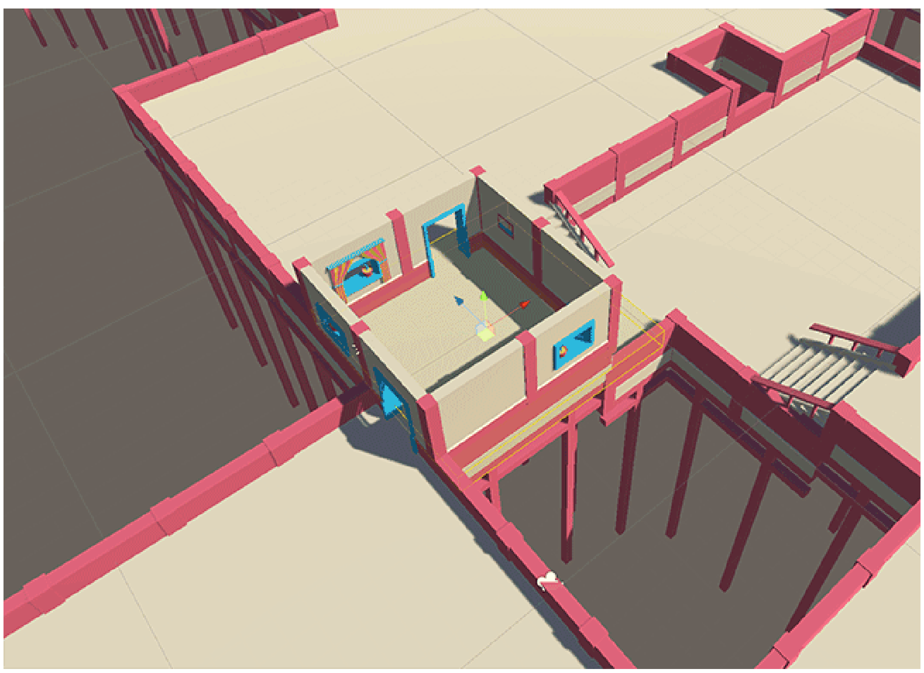
Navigate to `Asset/DungeonArchitect/Prefabs` and drag drop the `PlatformVolume` prefab on to the scene



Select the PlatformVolume game object and set the Dungeon reference



Click the button `Rebuild Dungeon`

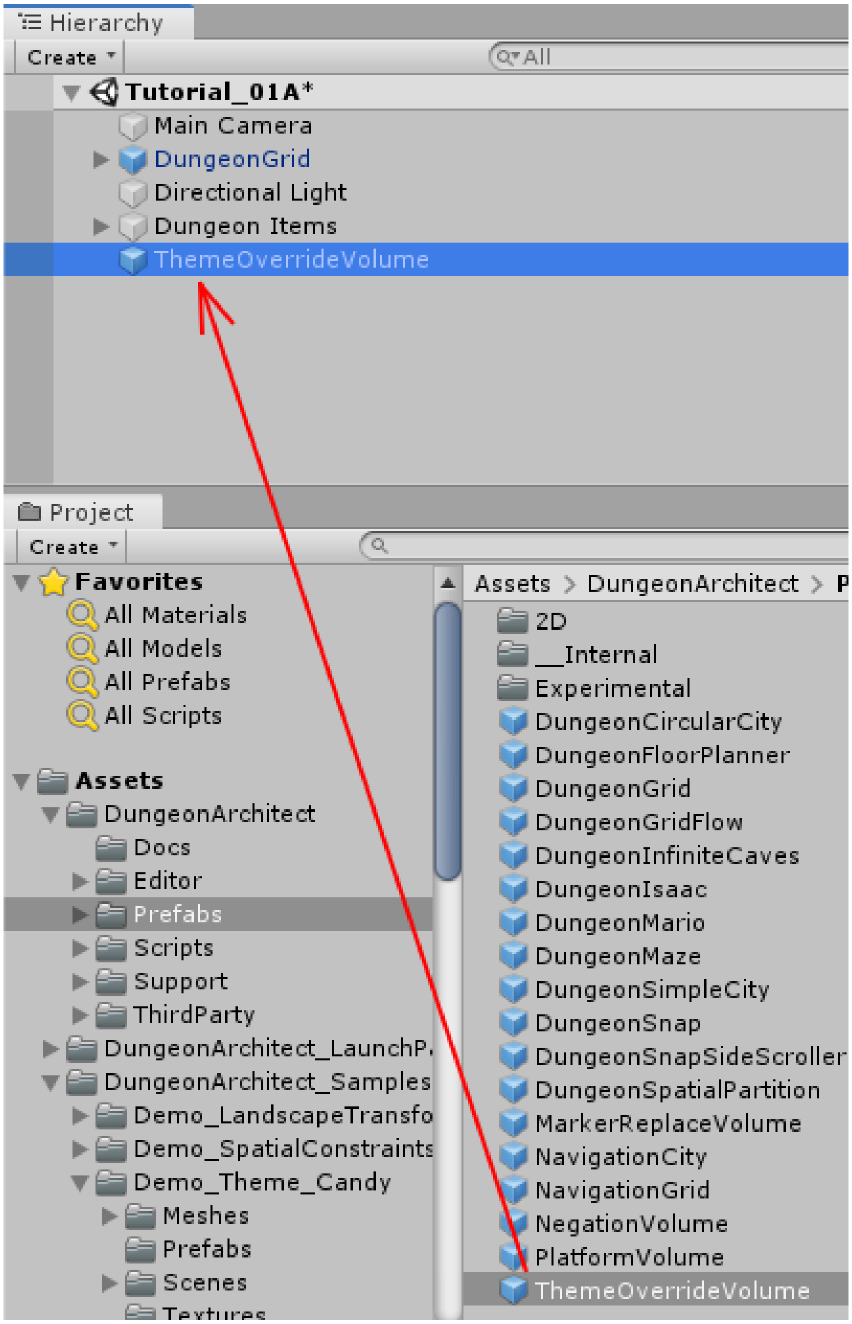


Move the `Platform Volume` and scale it to control the position and size of your room. You can have multiple platform volumes in the scene. Check the samples in the Launch Pad for more examples

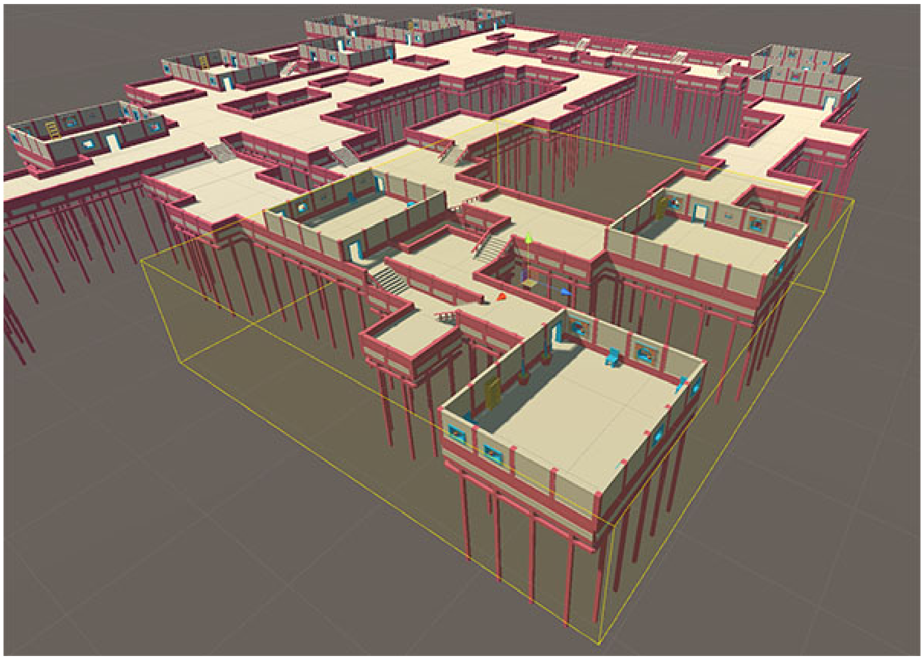
Theme Override Volume

Theme Override Volumes let you apply another theme on certain portions of your dungeons that are covered by this volume. These are useful for adding variations to your dungeons.

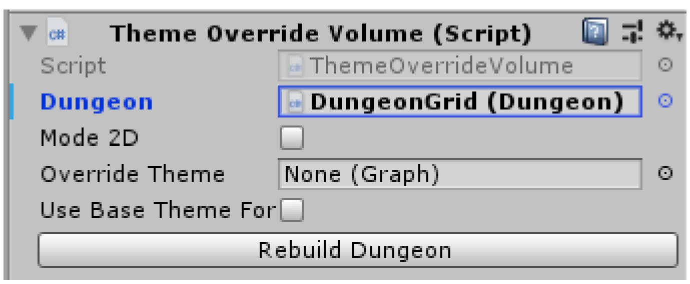
Navigate to `Asset/DungeonArchitect/Prefabs` and drag drop the `ThemeOverrideVolume` prefab on to the scene



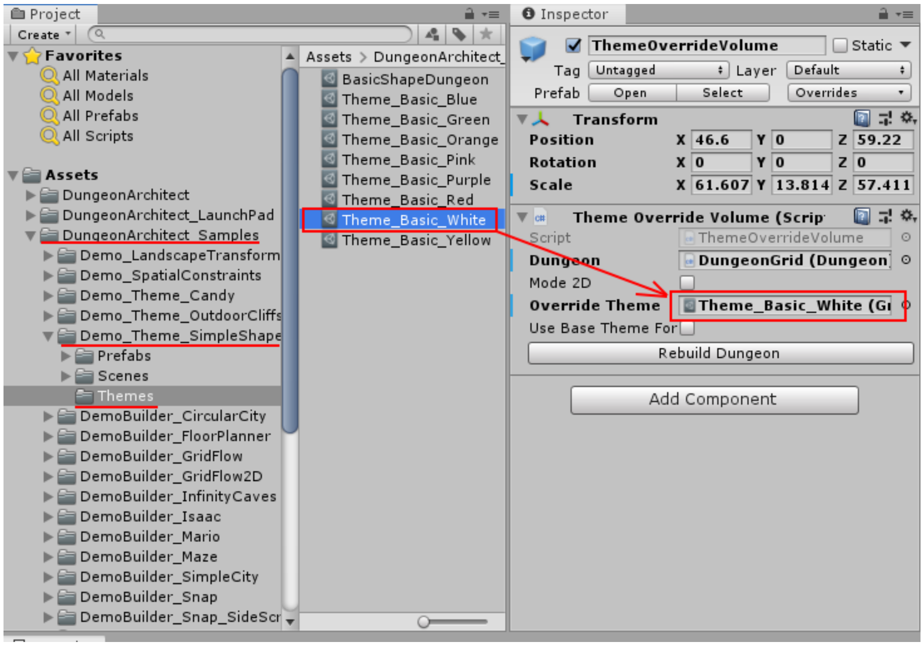
Move and scale it to cover a certain portion of the dungeon



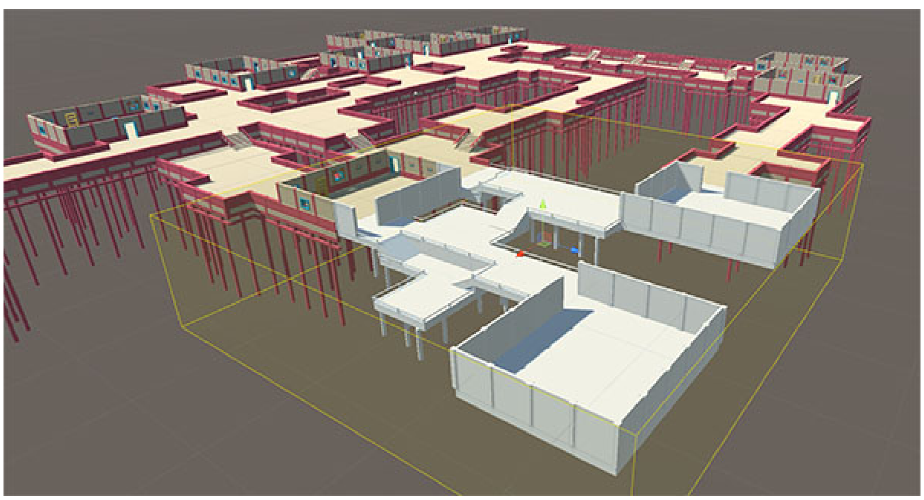
Select the theme override volume you just dropped and inspect the properties and assign the DungeonGrid reference



Navigate to `Assets/DungeonArchitect_Samples/Demo_Theme_SimpleShapes/Themes` and assign the theme `Theme_Basic_White` to the Theme Override Volume



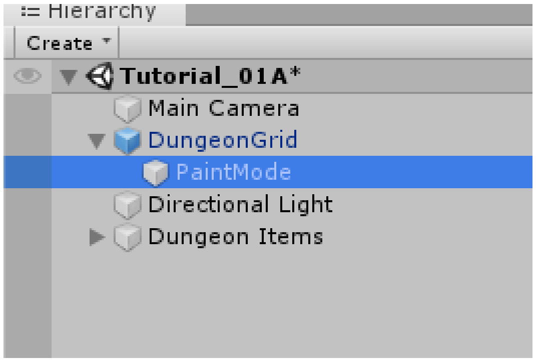
Click `Rebuild Dungeon`



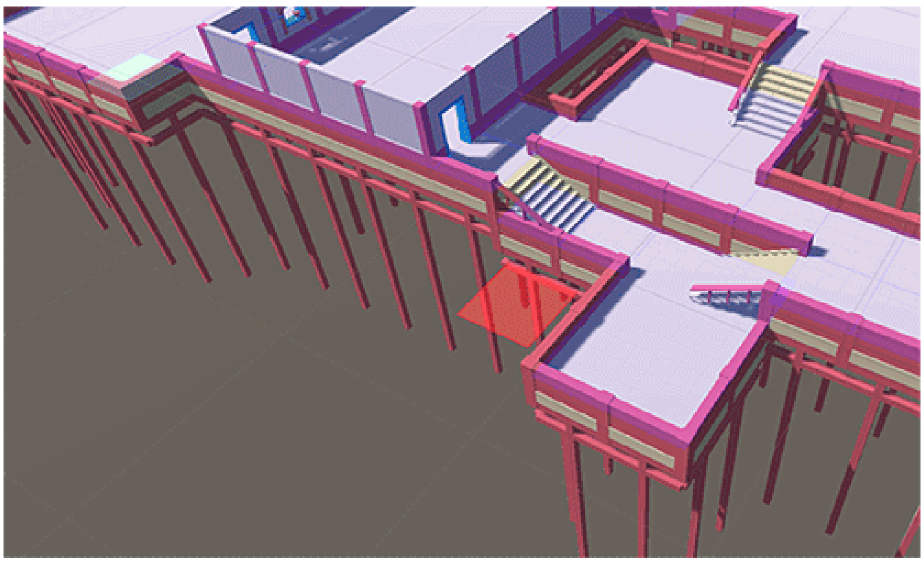
Paint Mode

You can paint your own dungeon layout on top of the procedural dungeon

Select the DungeonGrid prefab and expand it. Select the `PaintMode` game object. This will activate the `Paint Mode`



`Left click` and drag to draw dungeon cells. `Shift + Left click` to delete cells. `Scroll wheel` to move the cursor up/down

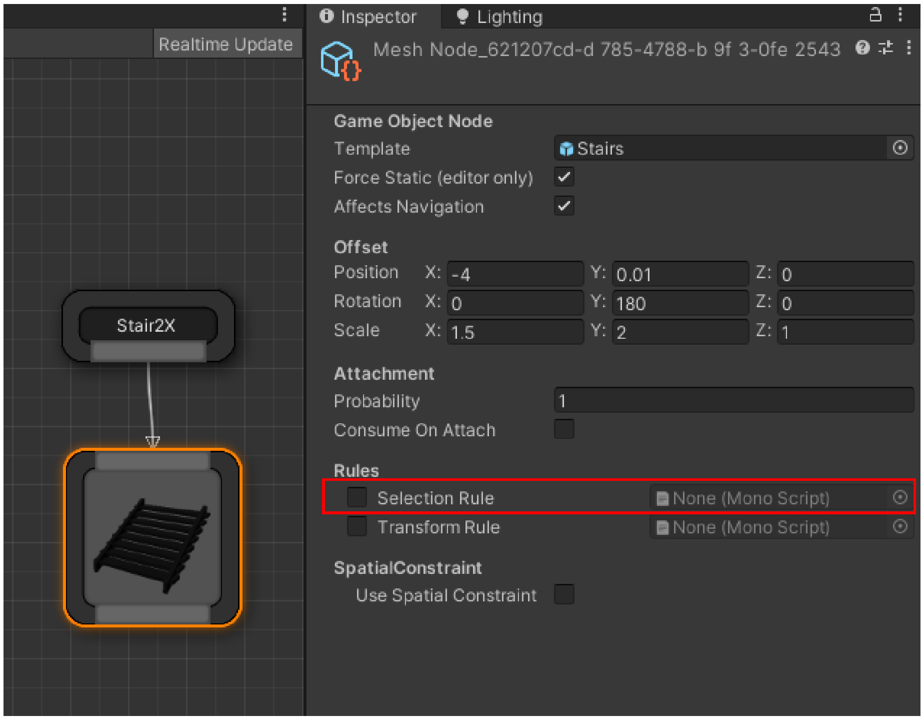


Advanced Theming

Selection Rule

Overview

We use the probability parameter to decide if we want to select and insert a certain object into the scene. If you need more control, you could write your own selection rules scripts



API

Create a C# script that inherits from `DungeonArchitect.SelectorRule`

Override the following method

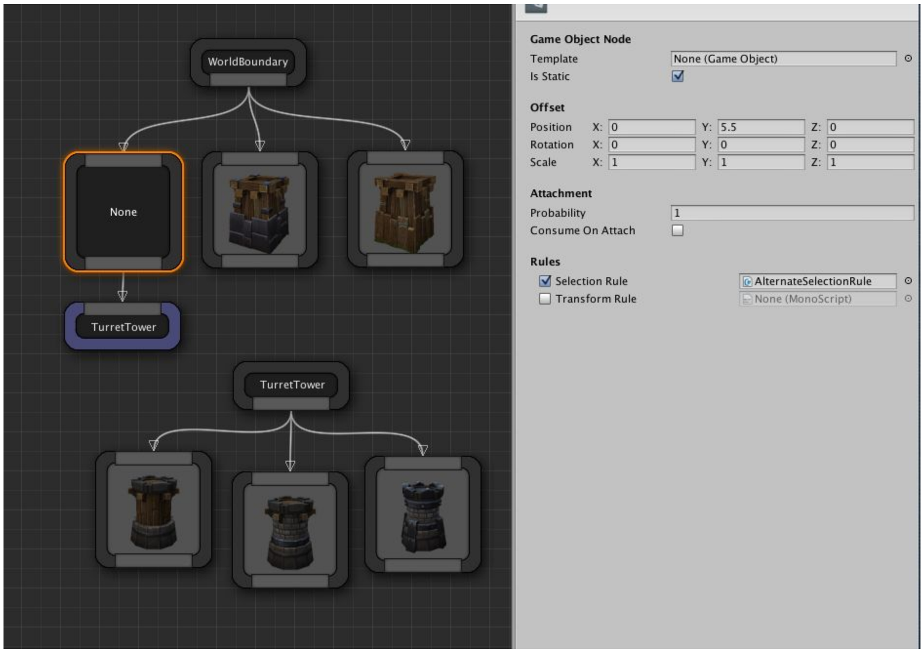
```
bool CanSelect(PropSocket socket, Matrix4x4 propTransform, DungeonModel model, S
```

Parameter | Description socket | The information about the marker propTransform | The final transform of the object that will be inserted model | The dungeon model object. You'll want to cast it to the approprirate model (e.g. `GridDungeonModel` and read the layout info if needed) random | The random stream. If you rely on any randomness, this object should be used to create consistent results in the dungeon

Example

In this example the towers are too crowded and close to each other.

A selector rule is created to select alternate cells

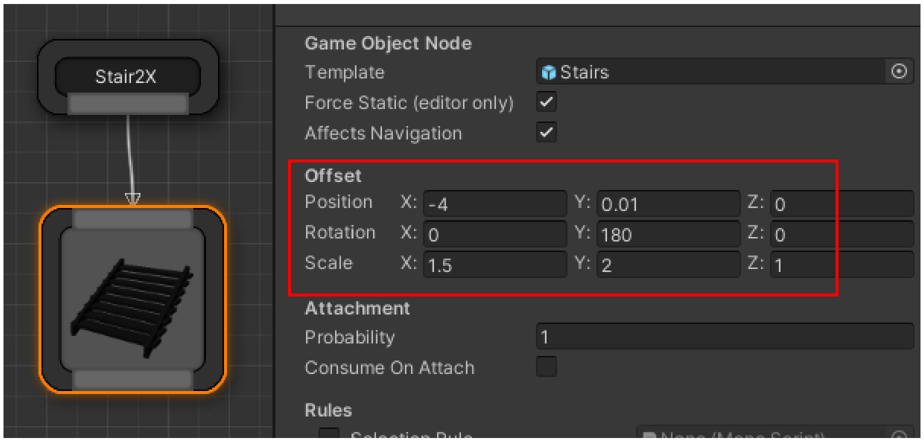



```
using UnityEngine;
using System.Collections;
using DungeonArchitect;

public class AlternateSelectionRule : SelectorRule {
    public override bool CanSelect(PropSocket socket, Matrix4x4 propTransform, D
        return (socket.gridPosition.x + socket.gridPosition.z) % 2 == 0;
    }
}
```

Transform Rule

Dungeon Architect lets you specify offsets to your visual nodes to move/scale/rotate them from their relative marker locations.



However, if you want a more dynamic way of applying offsets (based on scripts), you can do so with a *Transform Rule*. This can be very useful for adding variations to your levels for certain props

You can create new transform rules by implementing the TransformationRule class under the DungeonArchitect namespace

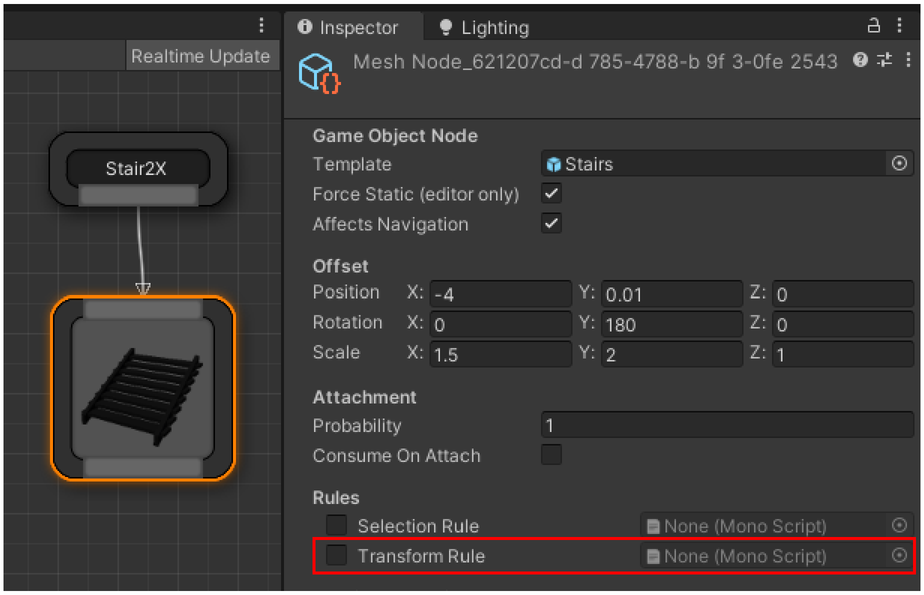
```
using UnityEngine;
using System.Collections;
using DungeonArchitect;
using DungeonArchitect.Utils;

public class RandomRotYTransformRule : TransformationRule {

    public override void GetTransform(PropSocket socket, DungeonModel model, Mat
        base.GetTransform(socket, model, propTransform, random, out outPosition,

        // Your transform logic here.
        // Update the outPosition, outRotation or outScale if necessary
    }
}
```

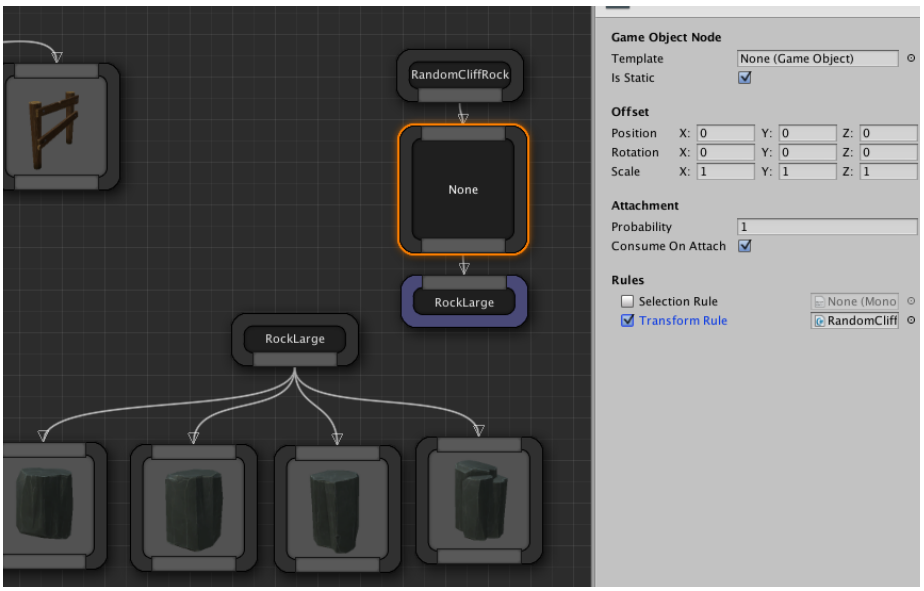
Attach this script to the theme node



Example #1

In this example, the cliff rocks are facing the same direction and do not look organic

After applying the transform script to the rock node, it looks much better




```
using UnityEngine;
using System.Collections;
using DungeonArchitect;
using DungeonArchitect.Utils;

public class RandomCliffTransformRule : TransformationRule {
    public override void GetTransform(PropSocket socket, DungeonModel model, Mat
        base.GetTransform(socket, model, propTransform, random, out outPosition,

        // Randomly rotate along the Y-axis
        var angle = random.NextFloat() * 360;
        var rotation = Quaternion.Euler(0, angle, 0);
```

```
        outRotation = rotation;

        // Slightly translate the node
        var variation = new Vector3(0.25f, -1, 0.25f);
        outPosition = Vector3.Scale (random.OnUnitSphere(), variation);
    }
}
```

Example #2

In this example a small random rotation is applied to ground tiles. This might be useful while creating ruins when laying down broken tile meshes

```
using UnityEngine;
using System.Collections;
using DungeonArchitect;
using DungeonArchitect.Utils;

public class BrokenTilesTransformRule : TransformationRule {

    public float maxAngle = 5;

    public override void GetTransform(PropSocket socket, DungeonModel model, Mat
        base.GetTransform(socket, model, propTransform, random, out outPosition,

        var rx = random.Range(-maxAngle, maxAngle);
        var ry = random.Range(-maxAngle, maxAngle);
        var rz = random.Range(-maxAngle, maxAngle);
```

```
        outRotation = Quaternion.Euler(rx, ry, rz);
    }
}
```

Example #3

In this example, the outer trees are spawned in the same height as the dungeon layout

We have a terrain that Dungeon Architect modifies and its steepness value is controlled by the user using a curve.

So, we would like to clamp this tree's base on the dynamic terrain.

This is done by finding the height of the terrain at that location, and creating an offset such that the tree would move up or down to properly clamp on it

```
using UnityEngine;
using System.Collections;
using DungeonArchitect;
using DungeonArchitect.Utils;

public class ClampToTerrainTransformRule : TransformationRule {

    public override void GetTransform(PropSocket socket, DungeonModel model, Mat
        base.GetTransform(socket, model, propTransform, random, out outPosition,

        var terrain = Terrain.activeTerrain;
        if (terrain == null) {
            return;
        }

        var position = Matrix.GetTranslation(ref propTransform);
        var currentY = position.y;
        var targetY = LandscapeDataRasterizer.GetHeight(terrain, position.x, pos

        // Apply an offset so we are touching the terrain
        outPosition.y = targetY - currentY;
    }
}
```

Item Spawn Listener

Item Spawn Listeners get notified of every game object that is spawned by the theme engine. This allows you to modify the spawned objects and perform post processing on them (e.g. set metadata, add / remove extra components etc)

You'll need to inherit from `DungeonItemSpawnListener` and implement the following function

```
void SetMetadata(GameObject dungeonItem, DungeonNodeSpawnData spawnData)
```

Then add this script to the dungeon game object

```
using DungeonArchitect;
using UnityEngine;

public class FlowItemMetadataHandler : DungeonItemSpawnListener
{
    public override void SetMetadata(GameObject dungeonItem, DungeonNodeSpawnDat
```

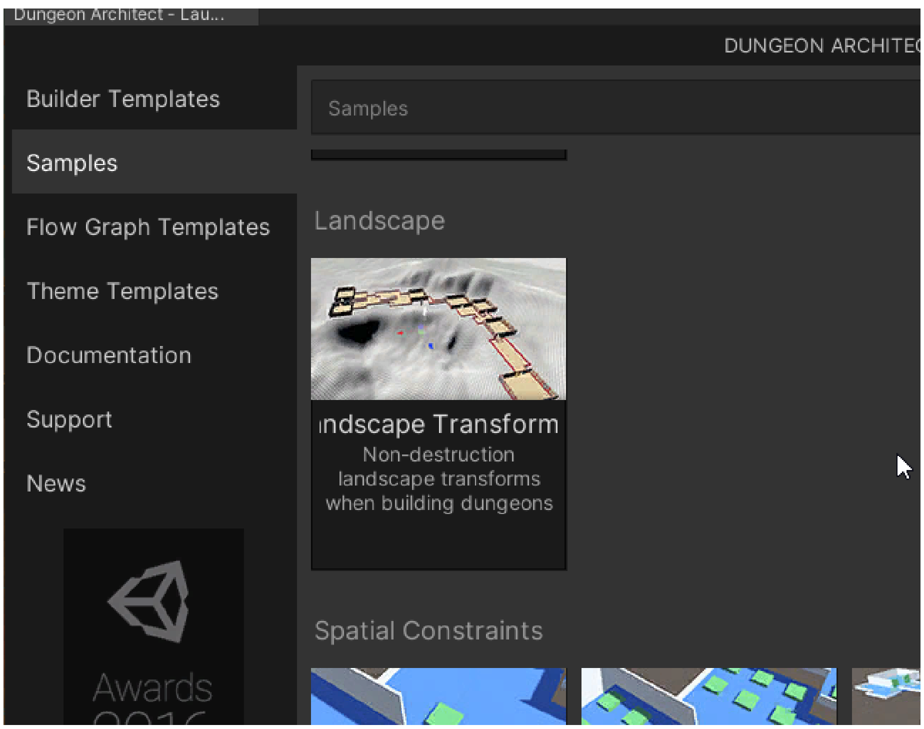
```
    {
        if (dungeonItem != null)
        {
            dungeonItem.AddComponent<...>();
        }
    }
}
```

Advanced Dungeons

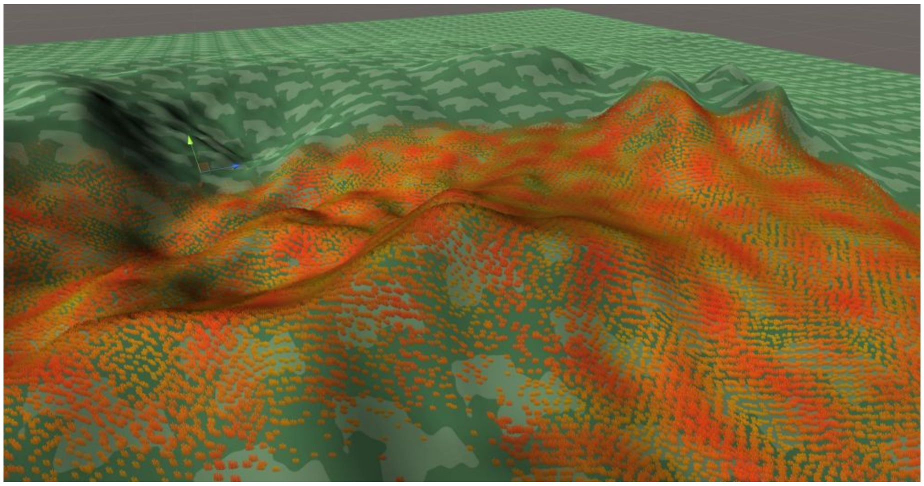
Landscape Transformer

Dungeon Architect supports non-destructive landscape transformations around the generated dungeon

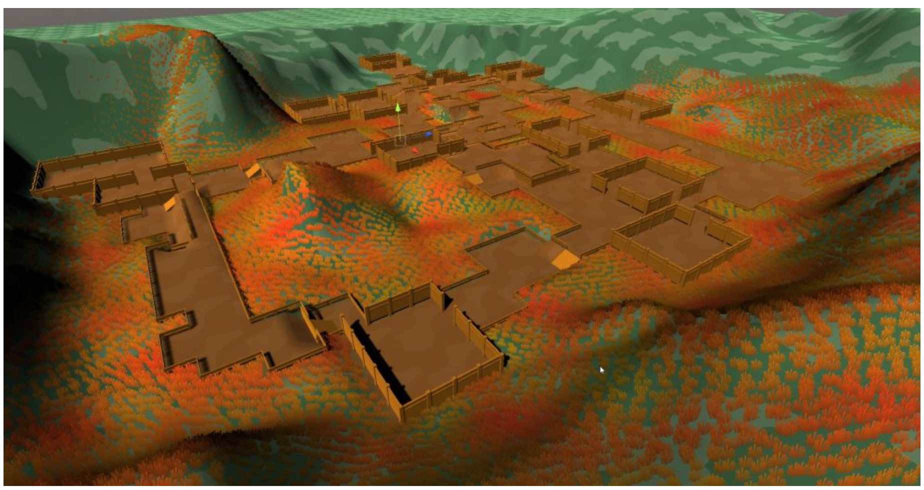
To see this in action, clone the Landscape transformer sample from the launchpad



You can build your dungeons on top of an existing landscape that has painted textures and foliage



The heightfield, paint and foliage around the dungeon would be modified



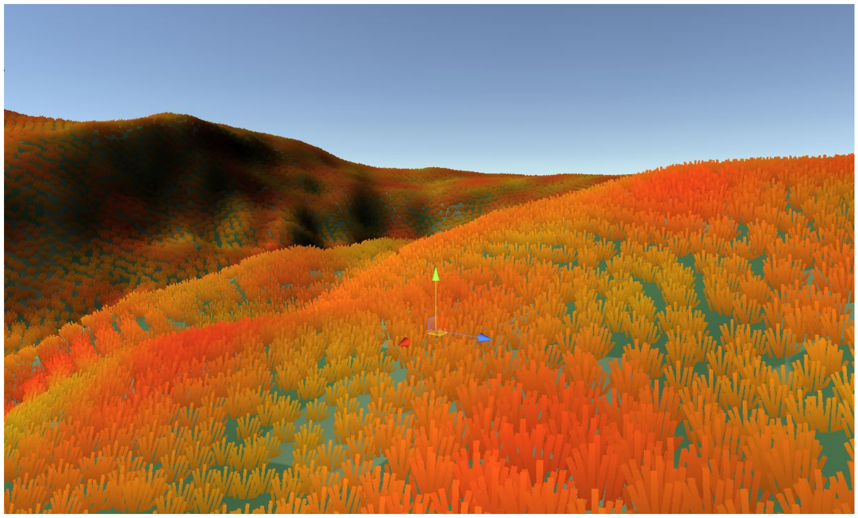
This is non destructive, so when you destroy your dungeon, the original landscape data (height, paint, foliage etc) in that area is restored

Right now, the Grid Builder and City builder support landscape transformations

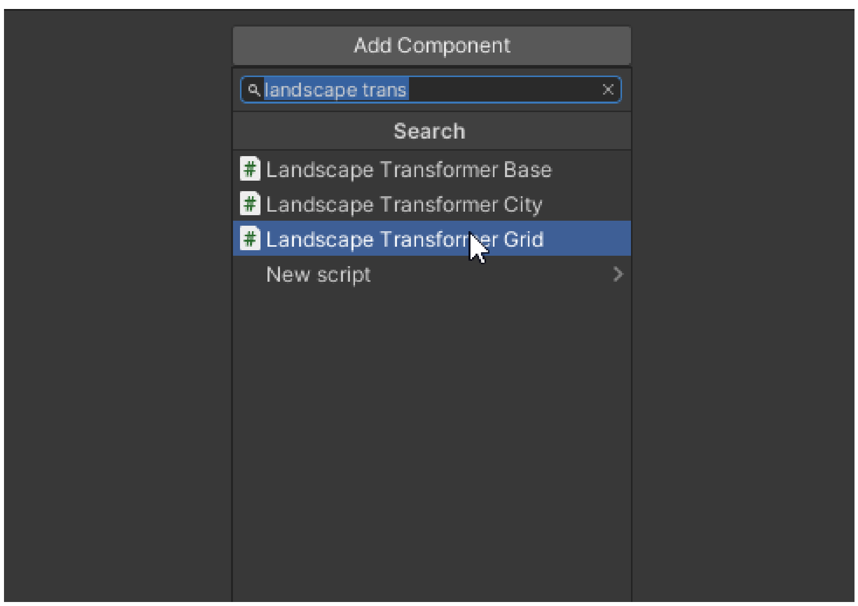
Setup Landscape Transformer

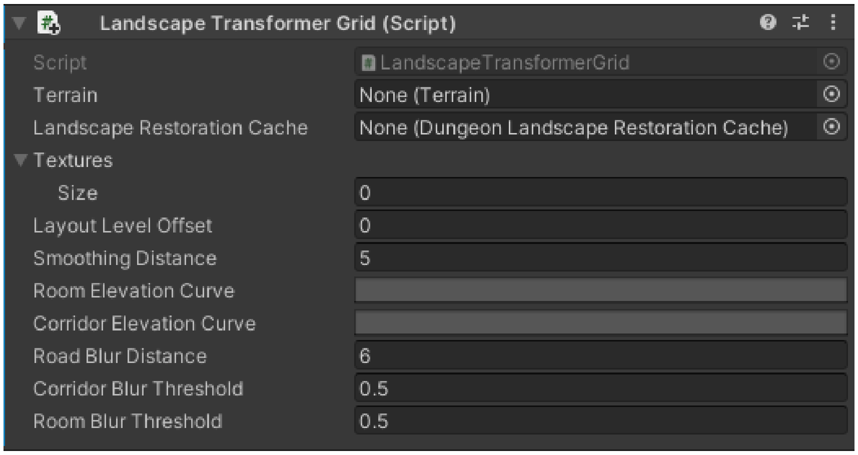
To add support for landscape transformations, perform the following,

1. Move your dungeon game object to the location where you'd like to build your dungeon

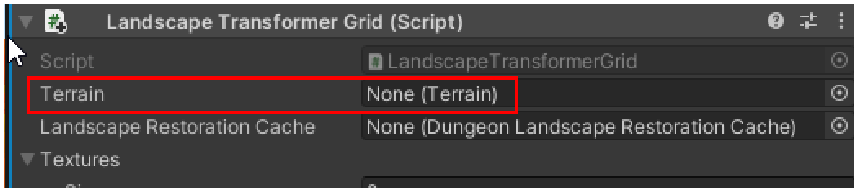


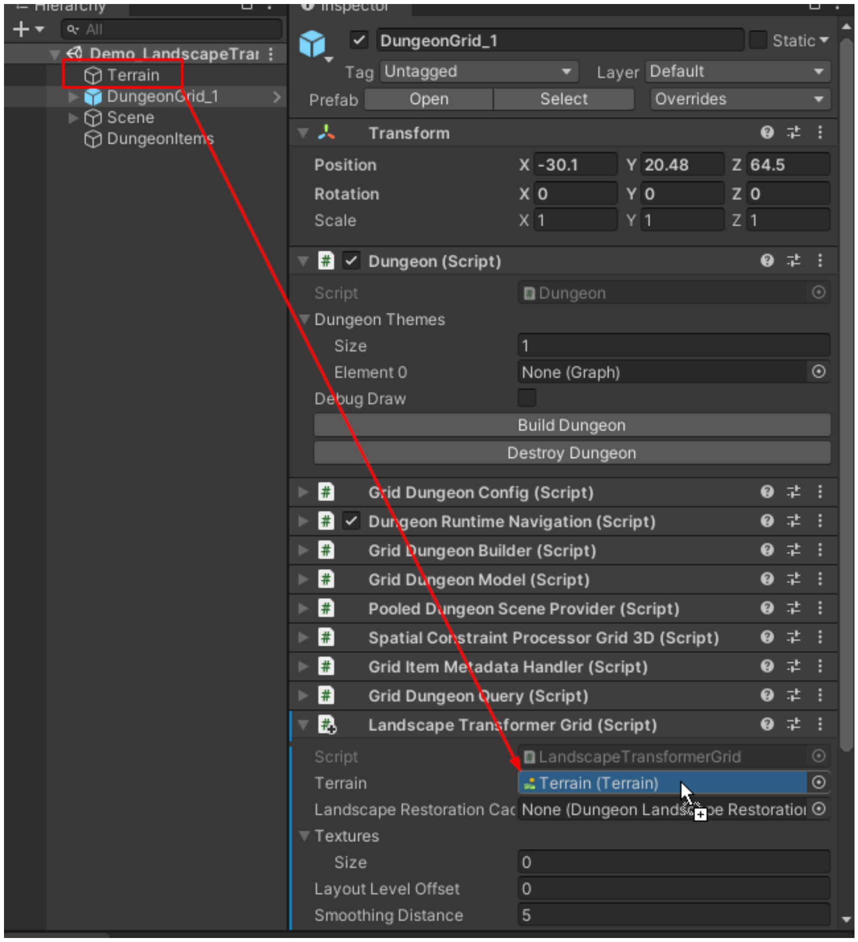
2. Add the `LandscapeTransformerGrid` or the `LandscapeTransformerCity` component to your dungeon game object, depending on the builder





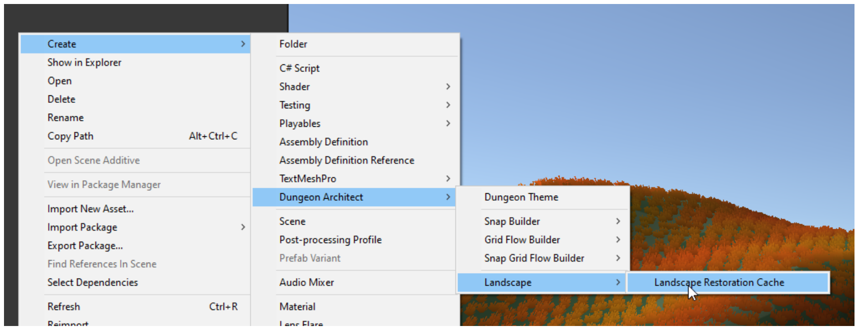
3. This script requires the terrain game object reference so it can modify it.

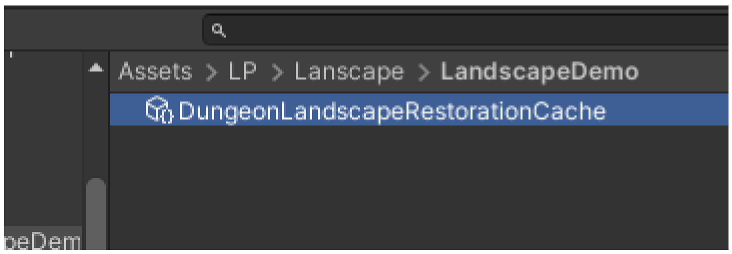




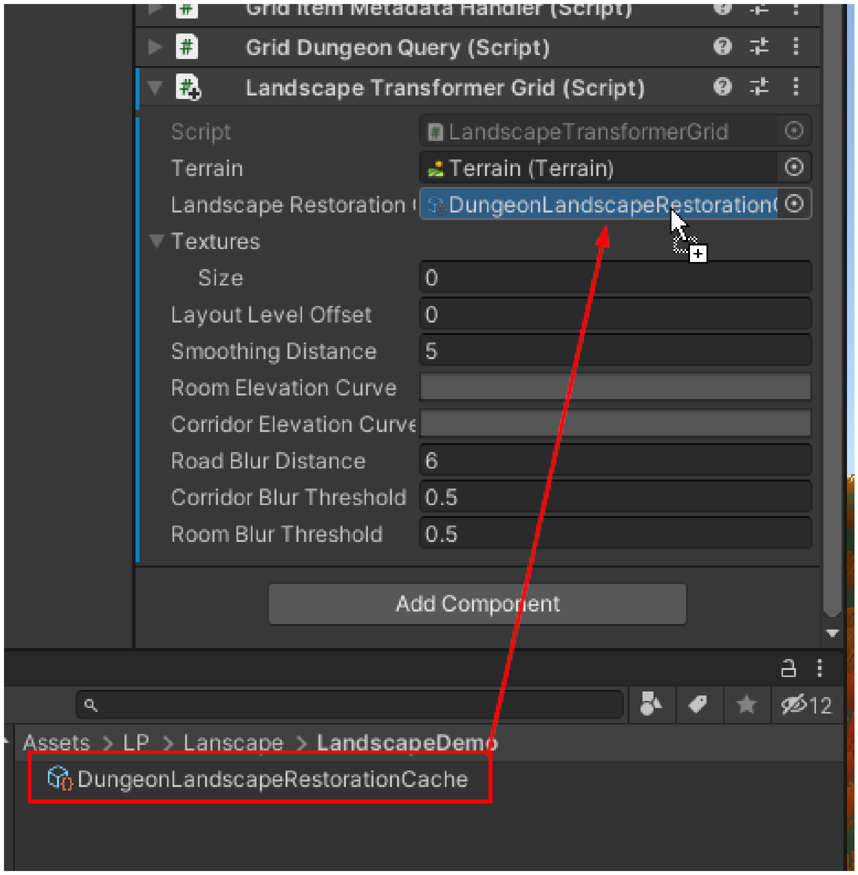
4. To support non-destructive landscape modification, Dungeon Architect needs to save the state of the landscape in the area where it modifies it, so that it can restore it later on (when the dungeon is destroyed or modified)

   Create a landscape restoration cache asset



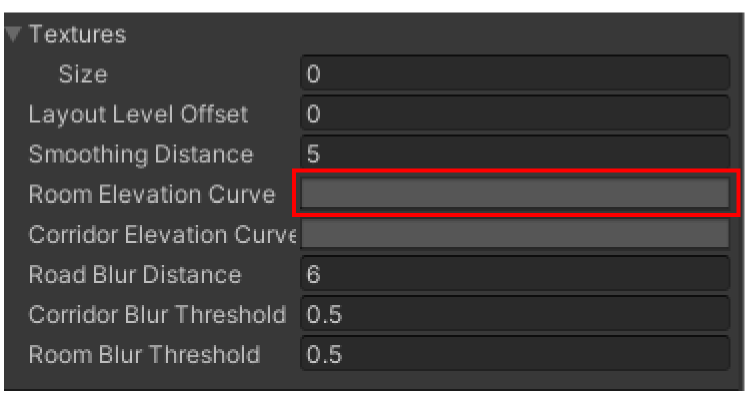


5. Assign this to the Lanscape Transformer script

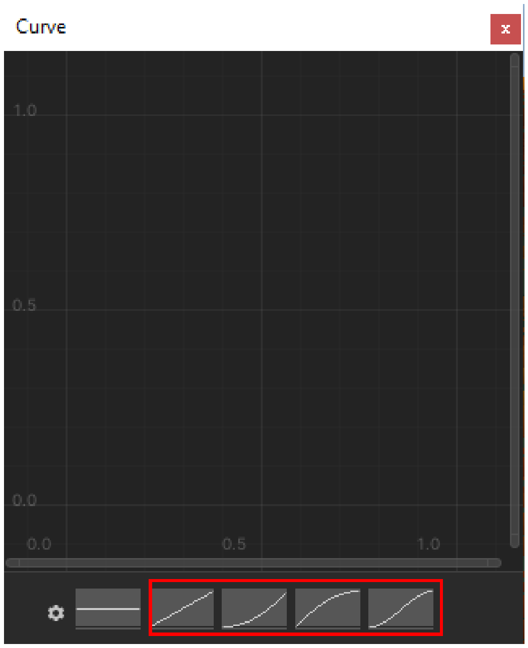


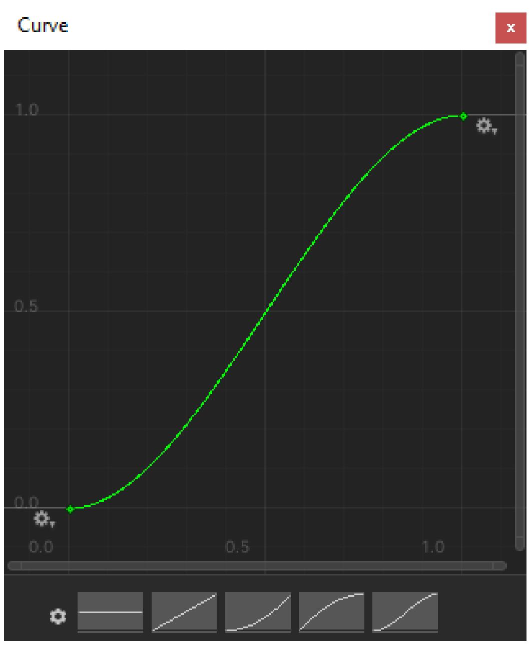
6. Assign the Elevation curves so we have a smooth transition from the existing terrain to the dungeon

   :::warning Important Unity will leave this blank by default and you need to assign it some value for it to work :::

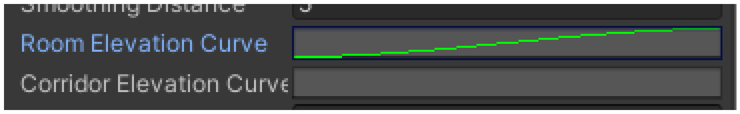


Click on blank area to open up the curve editor. Choose one of the values highlighted below

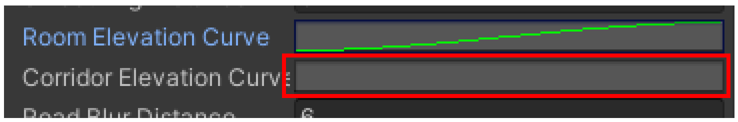


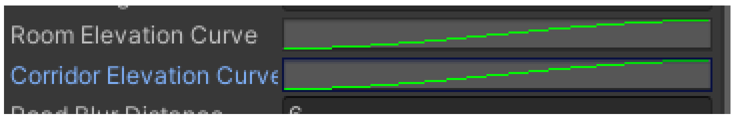


Close the curve editor

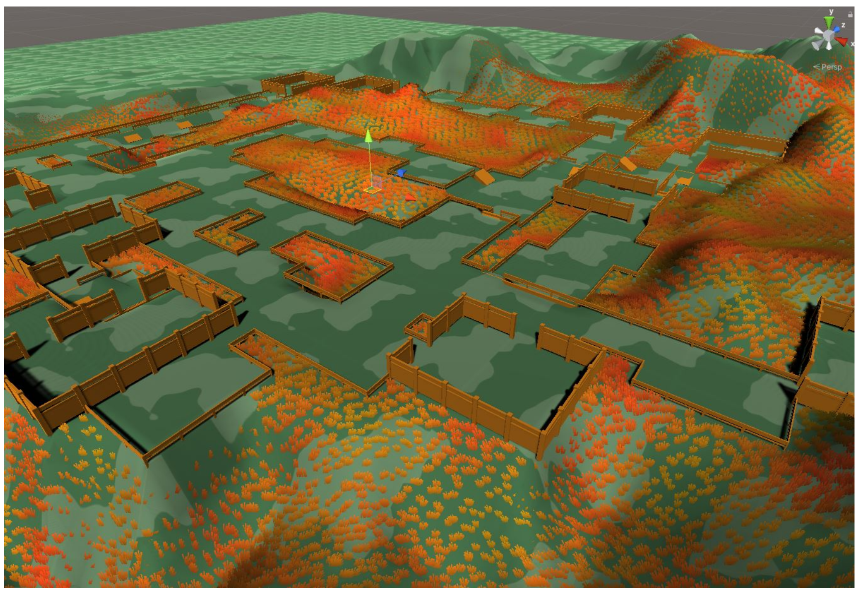


Do the same for Corridor Elevation Curves

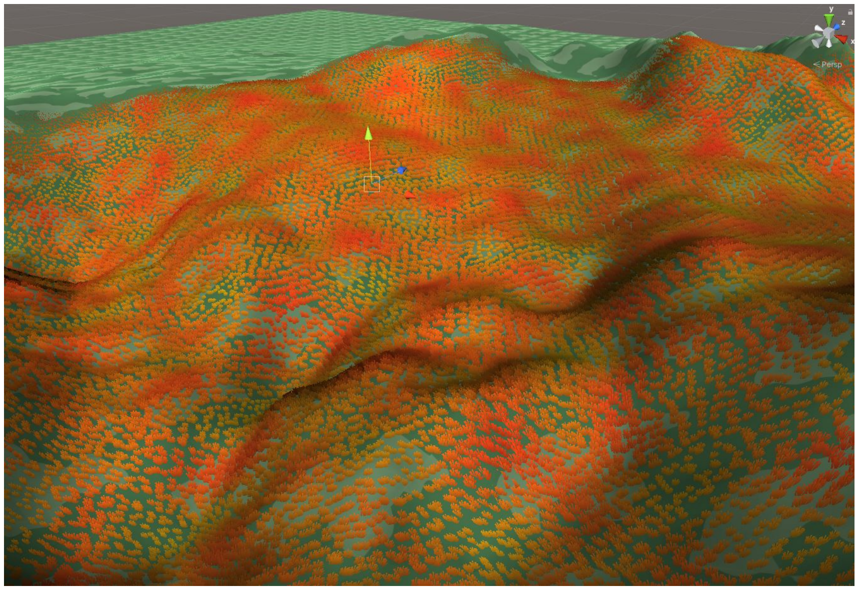




7. Build the dungeon

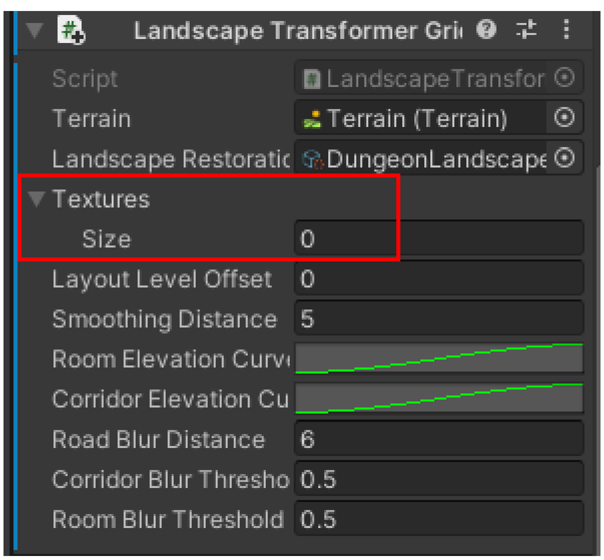


Destroy the dungeon to restore the terrain back

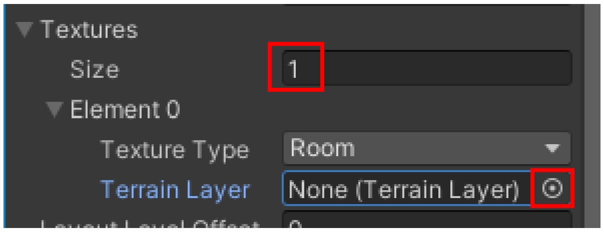


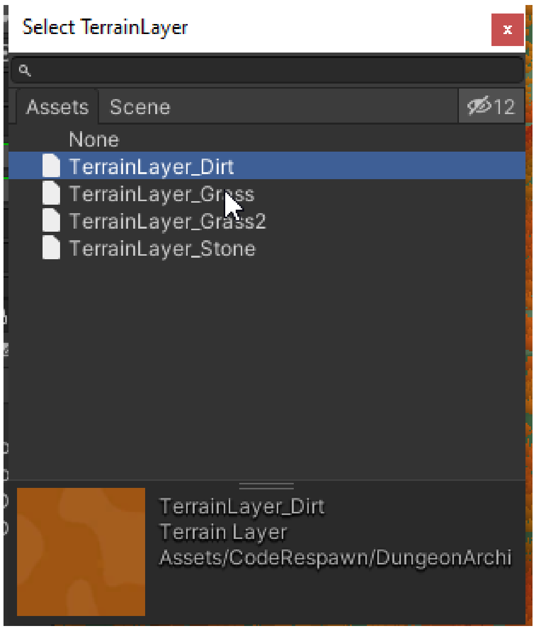
Setup Paint Support

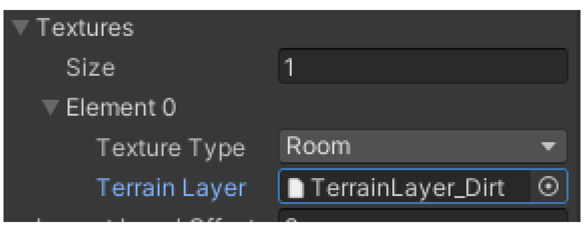
You may paint your terrain around the rooms, corridor and cliffs by applying your existing terrain layers. Since you already have a terrain, you would have setup terrain layers to paint your terrain. If not, find more info about Terrain Layers here. We'll paint the rooms with a certain terrain layer



Create a new Texture entry and add an existing Terrain Layer



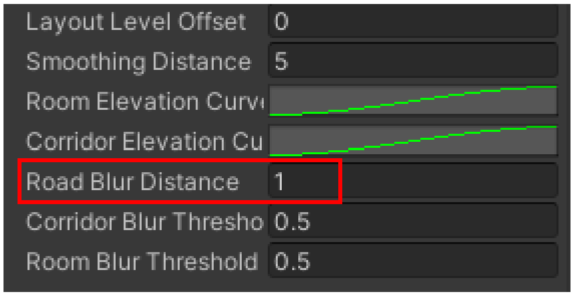




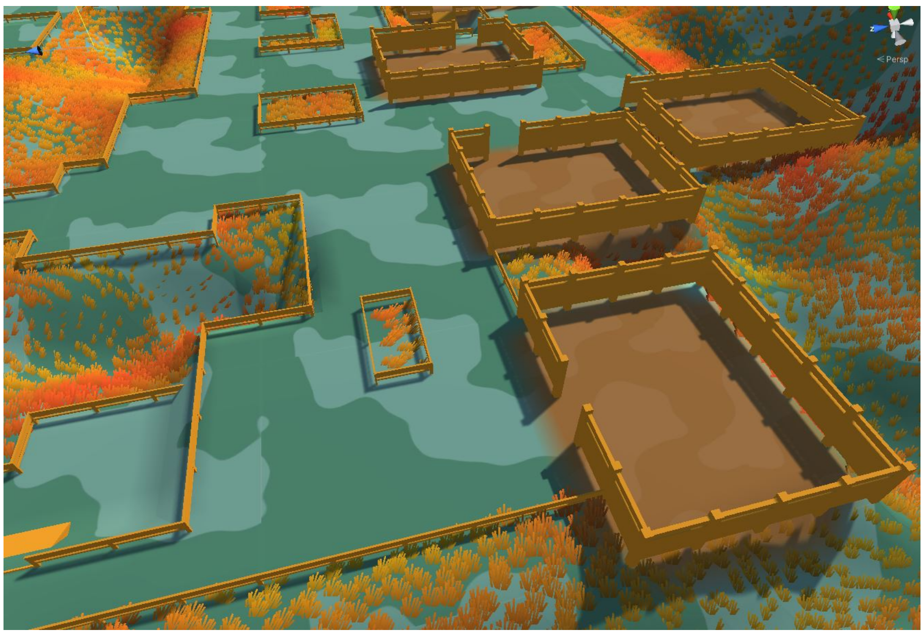
Set the *Texture Type* to `Room` or `Corridor`.

`Cliff` Texture Type is not supported for the time being and should not be used

Set the *Road Blur Distance* to `1`



Now build your dungeon



Add a corridor texture