Dungeon Architect User Guide

for Unity 5

1 Introduction

Dungeon Architect is a plugin for Unity that helps in streamlining the level creation process. It allows you to quickly create levels procedurally (or manually) by blocking out volumes and having the plugin build the environment automatically around it. This document introduces you to the various features of Dungeon Architect

2 Generation Overview

A dungeon is generated in two phases

- Layout Generation
- Visual Generation

2.1 Layout Generation

In this phase, only the layout of the dungeon is generated in memory. No meshes or actors are actually spawned.

Next, the level is populated with invisible points called **Markers** around the generated layout. A **marker** has only a name and a transformation in the 3D space.

In the above image, after the layout has been built, the dungeon builder has populated the level with marker points around the layout of the dungeon

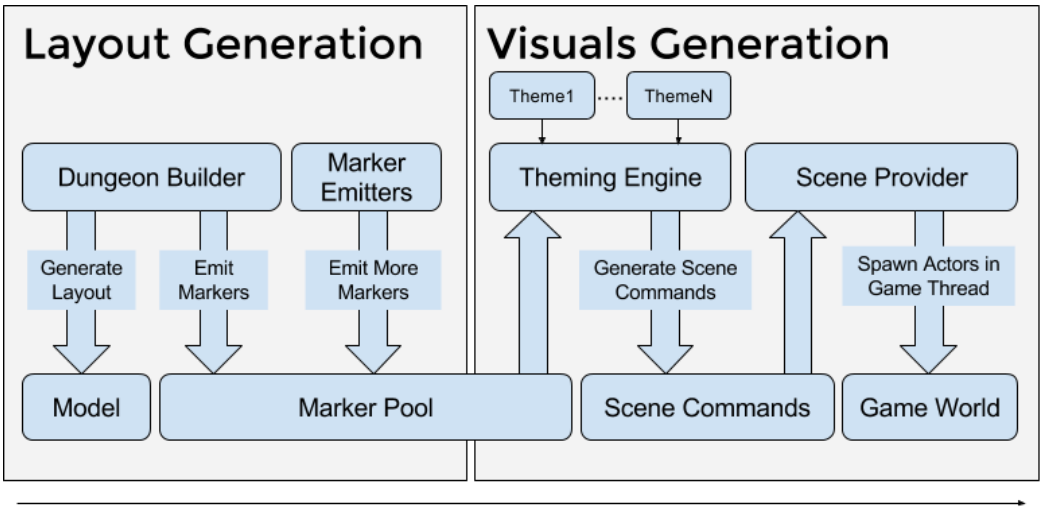


Figure 1: Architecture

2.2 Visual Generation

This phase spawns the actors in your scene. It takes all the marker points inserted in the previous phase and replaces them with actors (meshes, lights, blueprints etc) that you have mapped in your theme files

The theming engine is executed for each marker inserted in the Layout generation phase. In the above example, when a `Ground` marker is encountered, it would look for a `Ground` marker mapping in your theme file and replace the marker with the meshes you have mapped to it

The advantage of this data driven theme based approach is that theme files can be swapped to give your dungeon a completly different look. Theme files can also be shared across mulitple projects / teams

Multiple themes can also be used within the same dungeon to create variations

3 Dungeon Prefab

A dungeon prefab is used to build your dungeons. Drop the dungeon prefab into the scene and reset it's transform

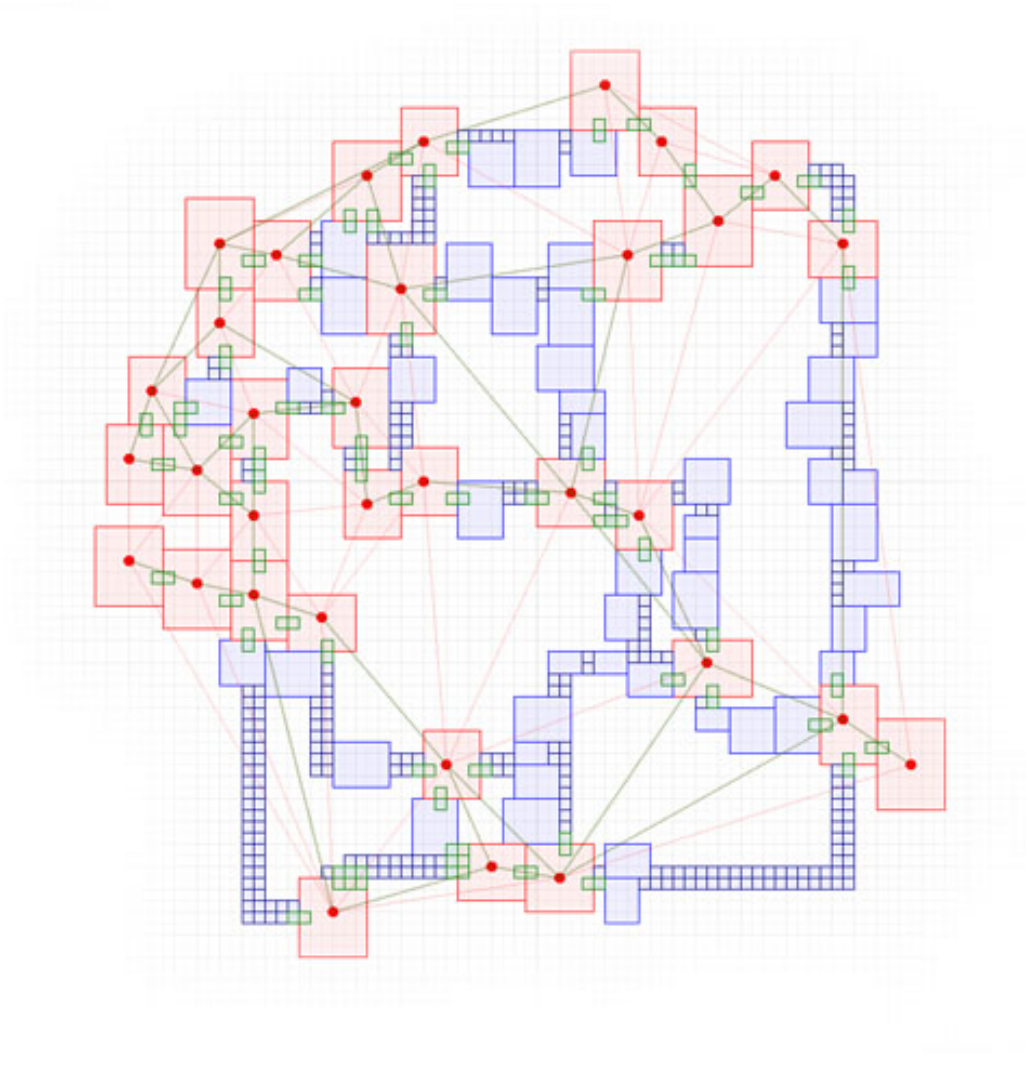


Figure 2: Layout of the dungeon in memory

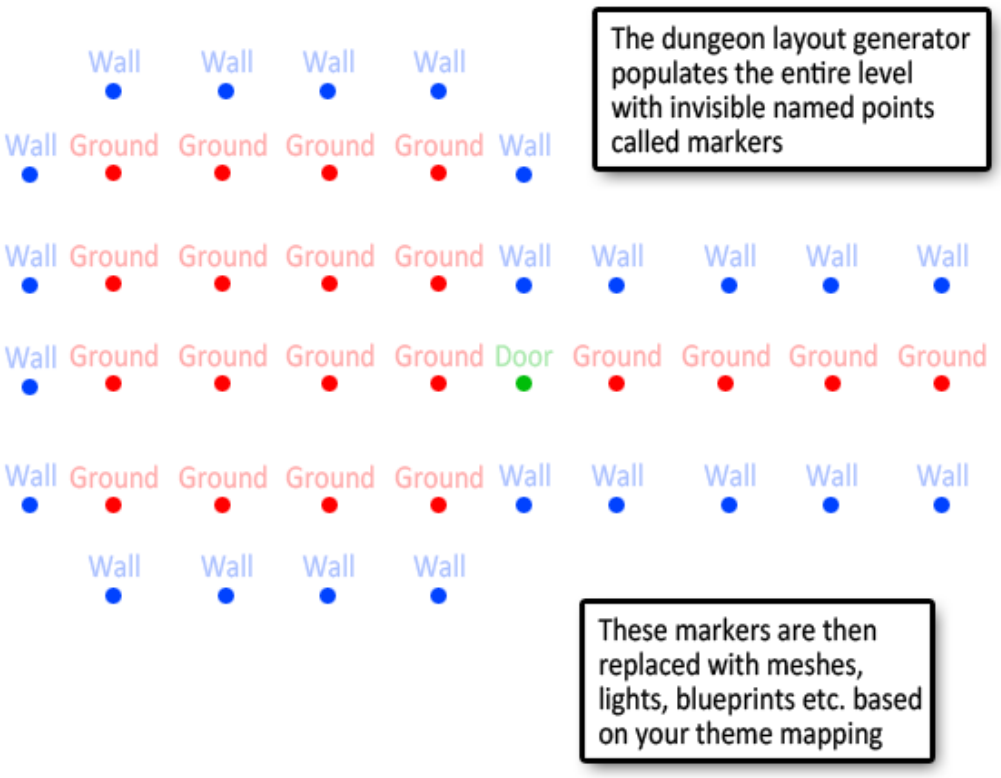


Figure 3: Markers populated for a sample room

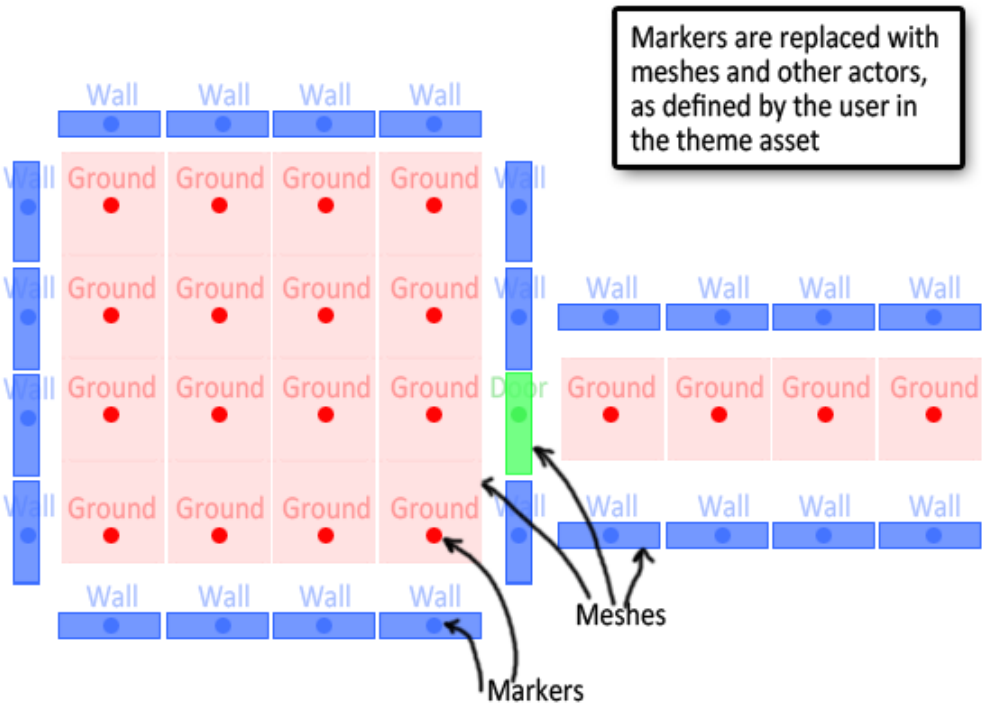


Figure 4: Actors spawned by a theme mapping

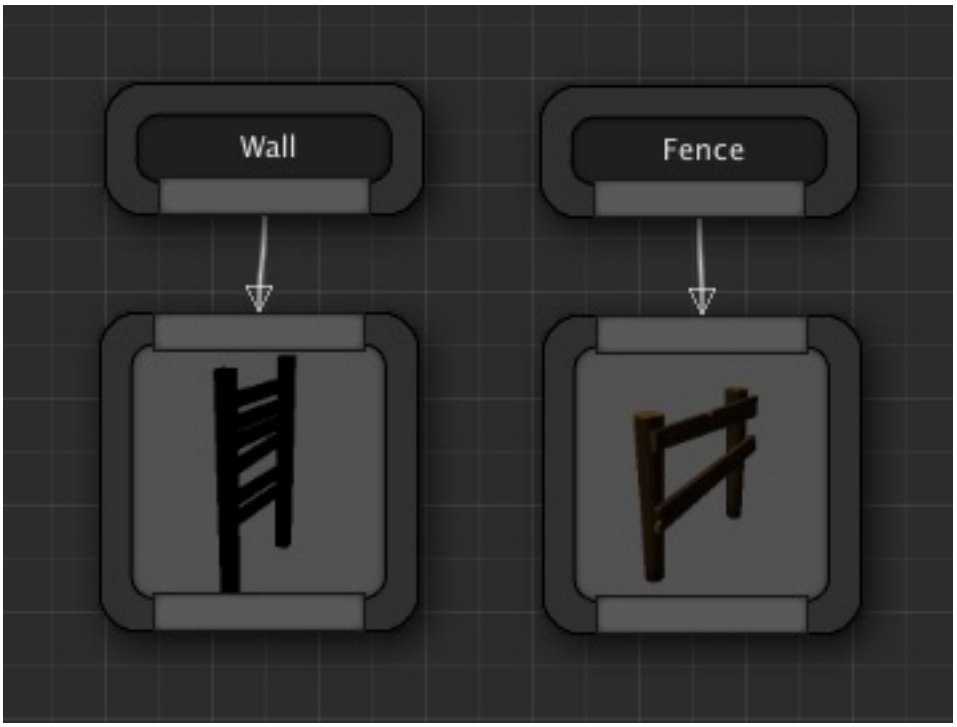


Figure 5: Sample Theme Mapping

Figure 6: Result after theme mapping

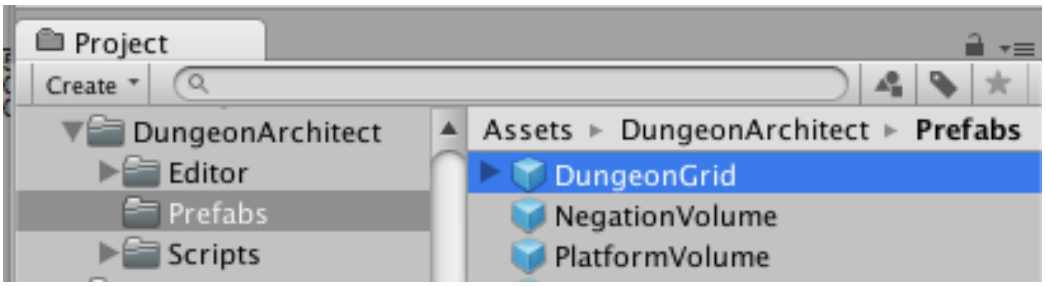


Figure 7: Dungeon Prefab

The dungeon game object generate a procedural layout for your dungeon based on the various configuration paramters.
After the layout has been generated, it spawns meshes, lights, blueprints etc, based on the mappings you have defined in the Theme file. This way you can define what meshes needs to be attached to the floors, walls, ceilings, etc

3.1 Properties

The Dungeon game object lets you perform various actions on your procedural dungeon. Select the Dungeon game object and have a look at the configuration in the Inspector window

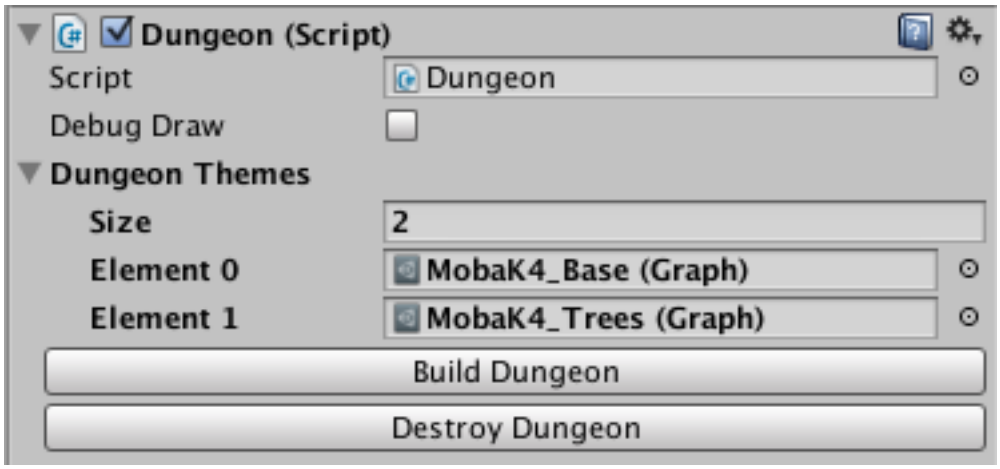


Figure 8: Dungeon Actor Properties

Build Dungeon: Builds a procedural dungeon. You need to define atleast one theme before you build

Destroy Dungeon: Destroys an existing dungeon owned by this actor. If you want to rebuild a dungeon after modifying the theme, there is no need to destroy first and you can directly click build

Themes: A theme file lets you design the look and feel of your dungeon. The theme editor lets you interactively design your own themes. You need to specify atleast one theme file before you can build your dungeon (sample content comes with many pre-created themes to get you started)

Debug Draw: Draws debug information in the scene view

3.2 Config Parameters

Select the Dungeon Actor and navigate to the Inspector window

The various config parameters determine how the layout of the dungeon is generated procedurally. The default layout generator algorithm is implemented based on the excellent writeup by TinyKeep's author Phi Dinh

The various parameters are:

- **Seed**: Changing this number would completely change the layout of the dungeon. This is the base random number seed that is used to build the dungeon. There is a convenience function to randomize this value (button labeled R)
- **Num Cells**: The number of cells to use while building the dungeon. You will not see these cells in the final result. A larger number would create a bigger and more complex dungeon. A number of 100-150 builds a medium to large sized dungeon. Experiment with different ranges
- **Grid Cell Size**: The dungeon generator works on a grid based system and required modular mesh assets to be placed on each cell (floors, walls, doors etc). This important field specifies the size of the cell to use. This size is determined by the art asset used in the dungeon theme designed by the artist. In the demo, we have a floor mesh that is 400x400. The height of a floor is chosen to be 200 units as the stair mesh is 200 units high. Hence the defaults are set to 400x400x200. You should change this to the dimension of the modular asset your designer has created for the dungeon

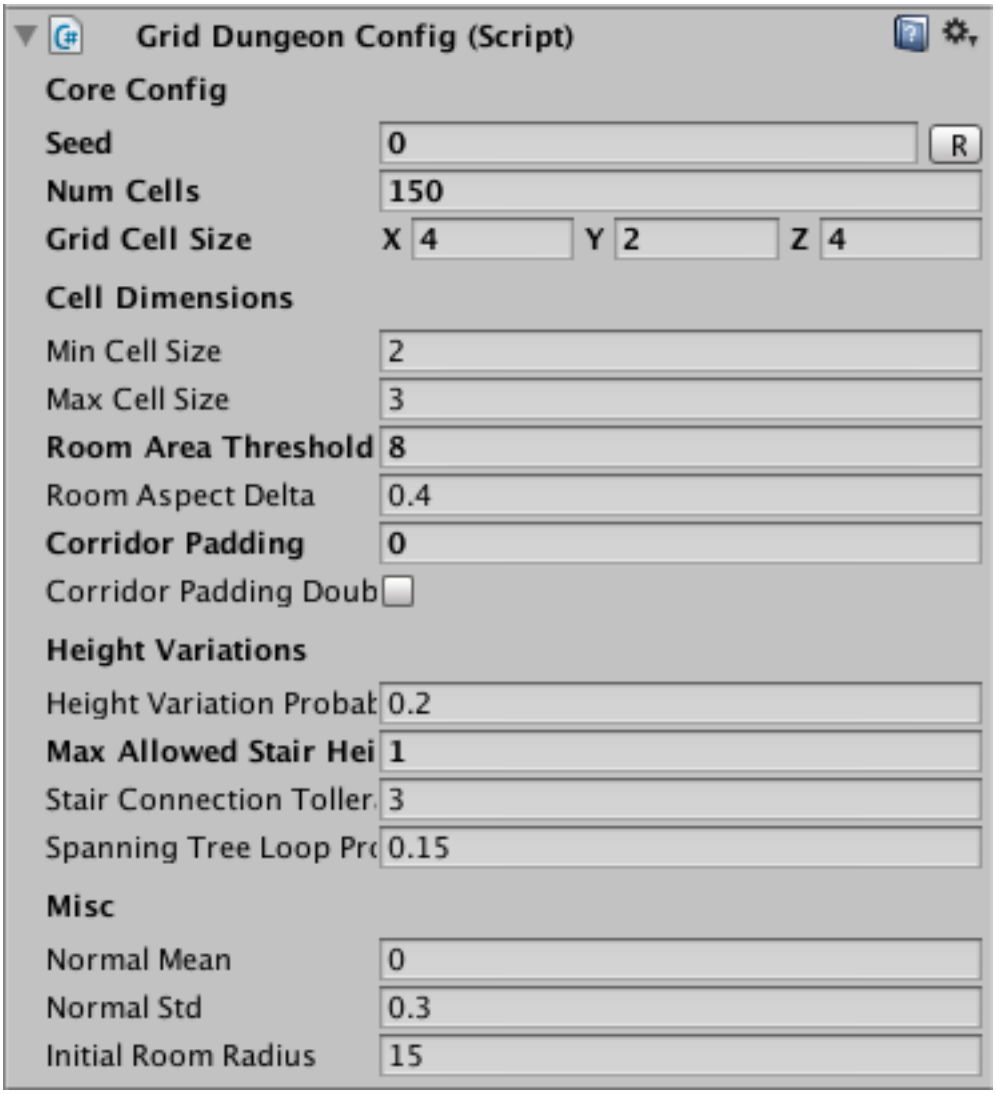


Figure 9: Dungeon Config Parameters

- **Min/Max Cell Size**: This is how big or small a cell size can be. While generation, a cell is either converted to a room, corridor or is discarded completely. The Cell width / height is randomly chosen within this range
- **Room Area Threshold**: If a cell size exceeds past this limit, it is converted into a room. After cells are promoted to rooms, all rooms are connected to each other through corridors (either directly or indirectly. See spanning tree later)
- **Room Aspect Delta**: The aspect ratio of the cells (width to height ratio). Keeping this value near 0 would create square rooms. Bringing this close to 1 would create elongated / stretched rooms with a high width to height ratio
- **Corridor Padding:** The extra width to apply to one side of a corridor
- **Corridor Padding Double Sided:** Flag to apply the padding on both sides of the corridor
- **Height Variation Probability**: Tweak this value to increase / reduce the height variations (and stairs) in your dungeon. A value close to 0 reduces the height variation and increases as you approach 1. Increasing this value to a higher level might create dungeons with no place for proper stair placement since there is too much height variation. A value of 0.2 to 0.4 seems good
- **Max Allowed Stair Height**: The number of logical floor units the dungeon height can vary. This determines how high the dungeon's height can vary (e.g. max 2 floors high). Set this value depending on the stair meshes you designer has created. In the sample demo, there are two stair meshes, one 200 units high (1 floor) and another 400 units high (2 floors). So the default is set to 2
- **Spanning Tree Loop**: Determines how many loops you would like to have in your dungeon. A value near 0 will create fewer loops creating linear dungeons. A value near 1 would create lots of loops, which would look unoriginal. Its good to allow a few loops so a value close to zero (like 0.2 should be good)
- **Stair Connection Tolerance**: The generator would add stairs to make different areas of the dungeon accessible. However, we do not want too many stairs. For e.g., before adding a stair in a particular elevated area, the generator would check if this area is already accessible from a nearby stair. If so, it would not add it. This tolerance parameter determines how far to look for an existing path before we can add a stair. Play with this parameter if you see too many stairs close to each other, or too few

- **Normal Mean / Std**: The random number generator used in the dungeon generator does not use a uniform distribution. Instead it uses a normal distribution to get higher frequency of lower values and fewer higher values (and hence fewer room cells and a lot more corridor cells). Play with these parameters for different results
- **Initial Room Radius**: Internal Usage. Keep to a low value like 10-15

4 Theme Overview

A theme file lets you design the look and feel of you dungeons with an intuitive graph based approach

Themes are saved as a separate assets on disk. Dungeon Architect also provides an interactive editor to help you design beautiful levels

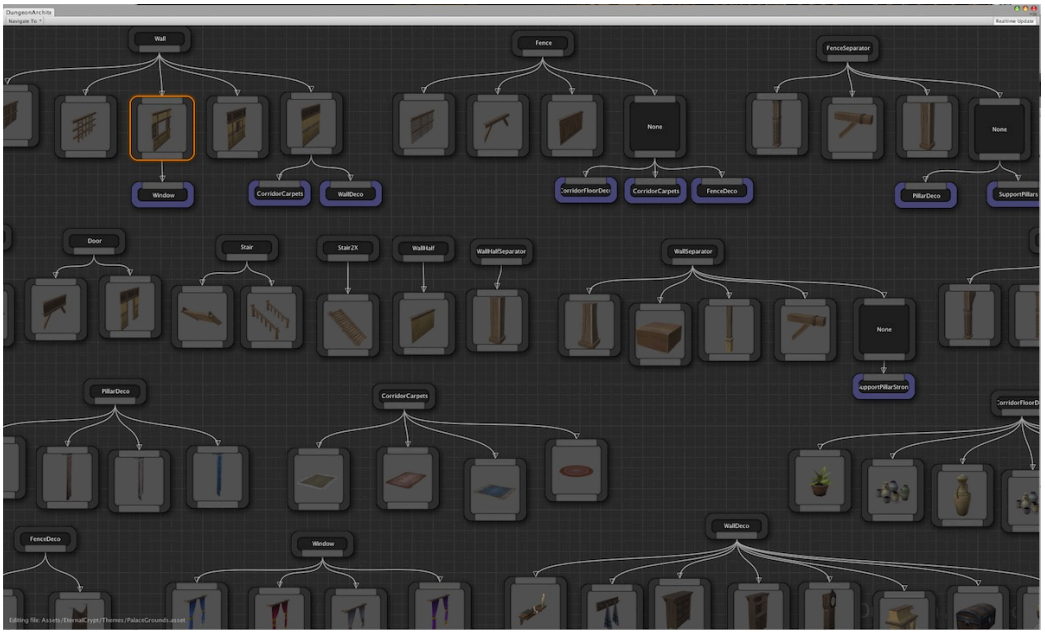


Figure 10: An example theme

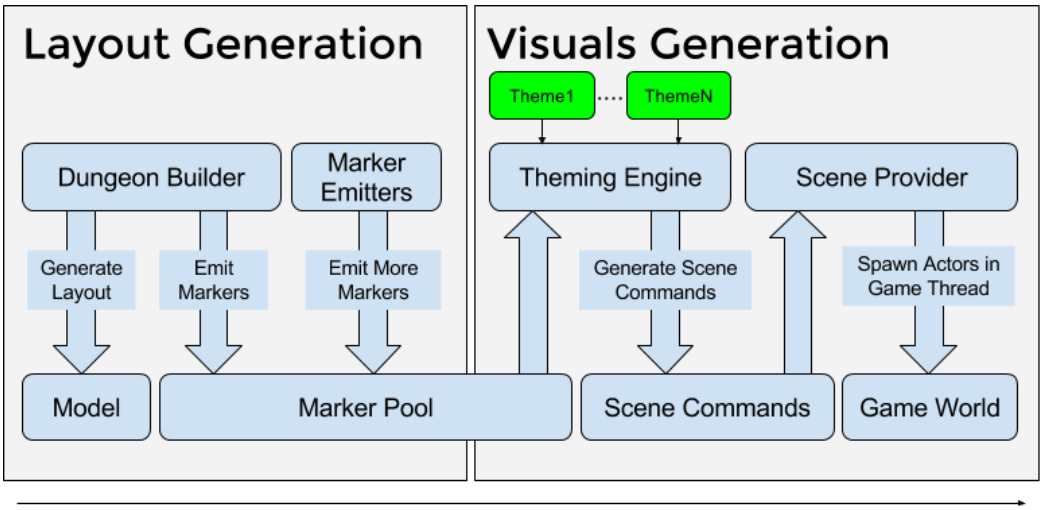


Figure 11: Themes are used by the Theming Engine

4.1 Create a Theme

To create a new theme, right click in the content browser and choose “Dungeon Theme”

5 Theme Editor

Double click a dungeon theme asset to open it in the `Theme Editor`

5.1 Interactive Editing

As you design your theme, the scene view automatically gets updated based on your theme graph mapping. To make this happen, you need to have a dungeon game object in the scene with the current theme being edited applied to it

Whenever you change the theme, the theme editor would search for a dungeon game object in the scene (that has this theme applied to it) and rebuild it. This way, you get an immediate visual feedback while designing the look and feel of you levels

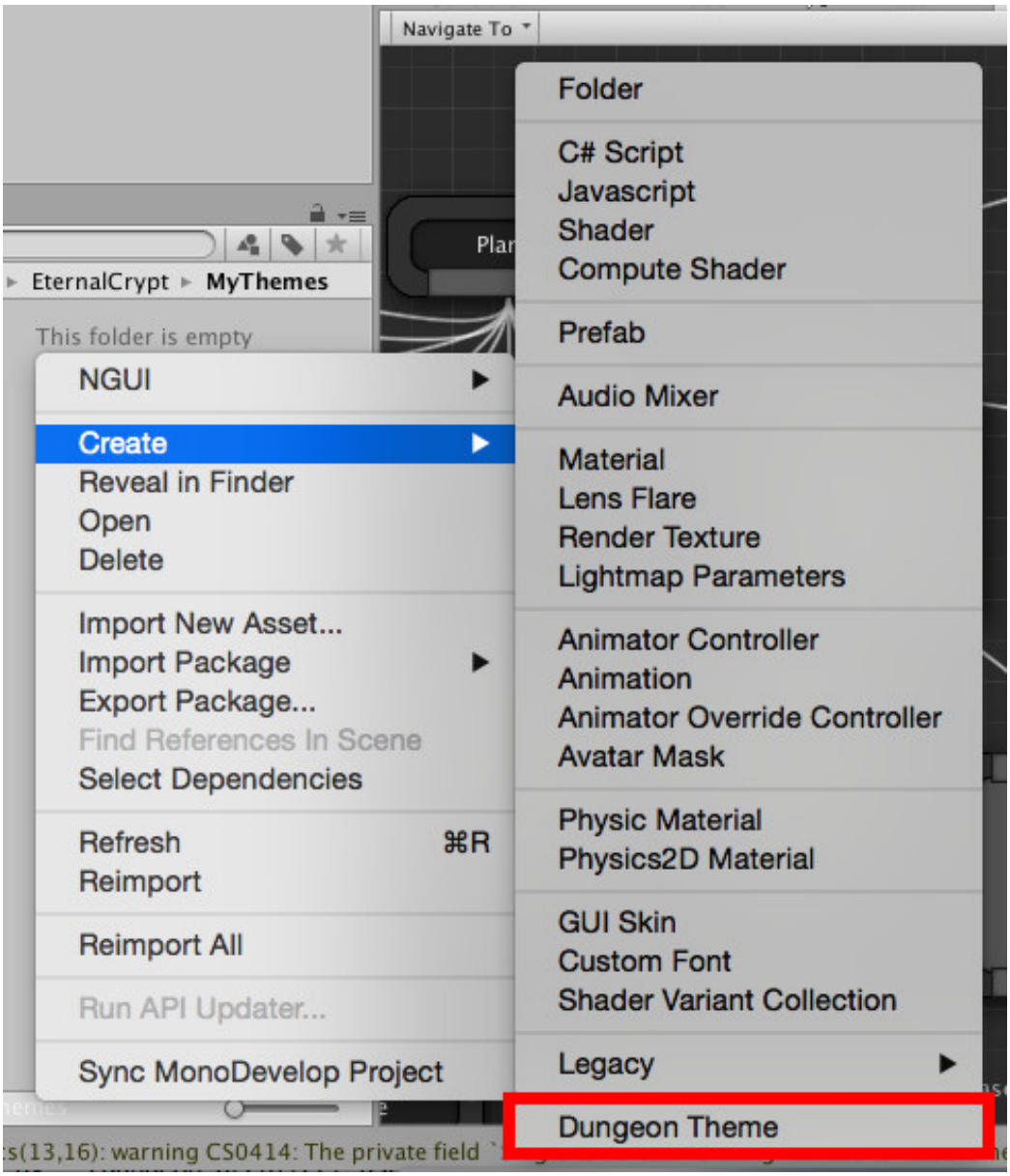


Figure 12: Create a dungeon theme

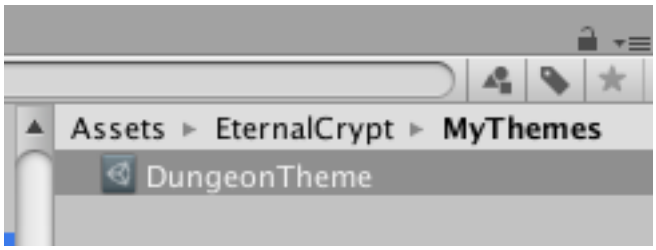


Figure 13: Dungeon Theme Asset

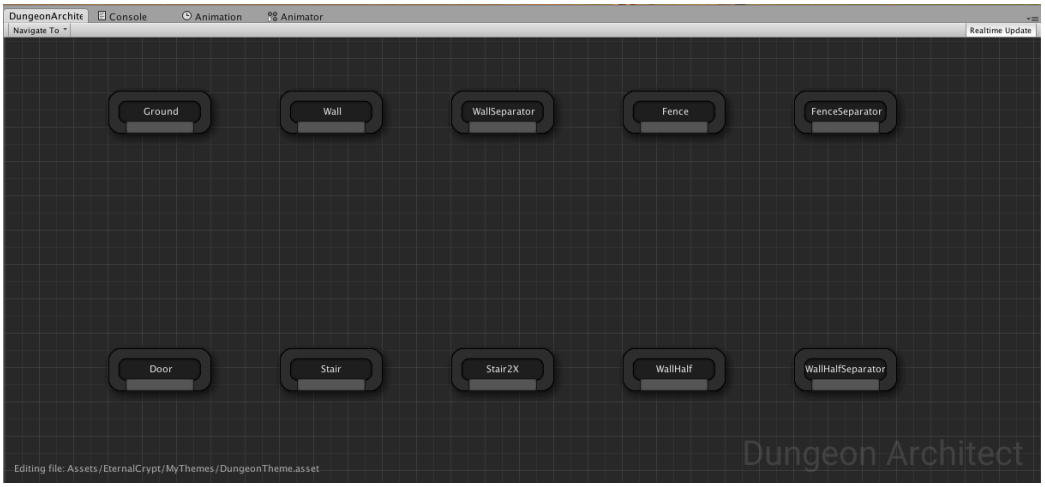


Figure 14: Dungeon Architect Theme Editor

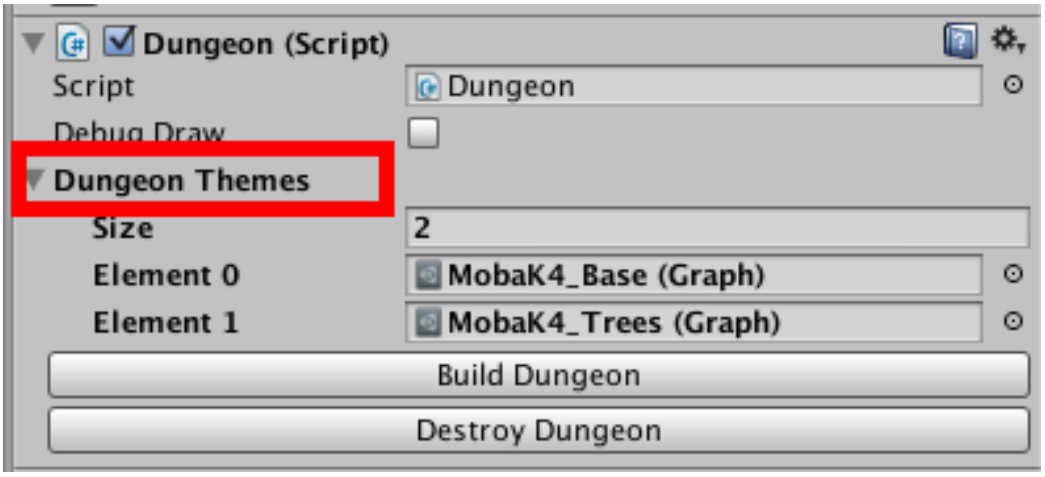


Figure 15: Dungeon Architect Theme Editor

6 Theme Nodes

A Theme can have 3 category of nodes

- **Marker Nodes**
- **Visual Nodes**
- **Marker Emitter Nodes**

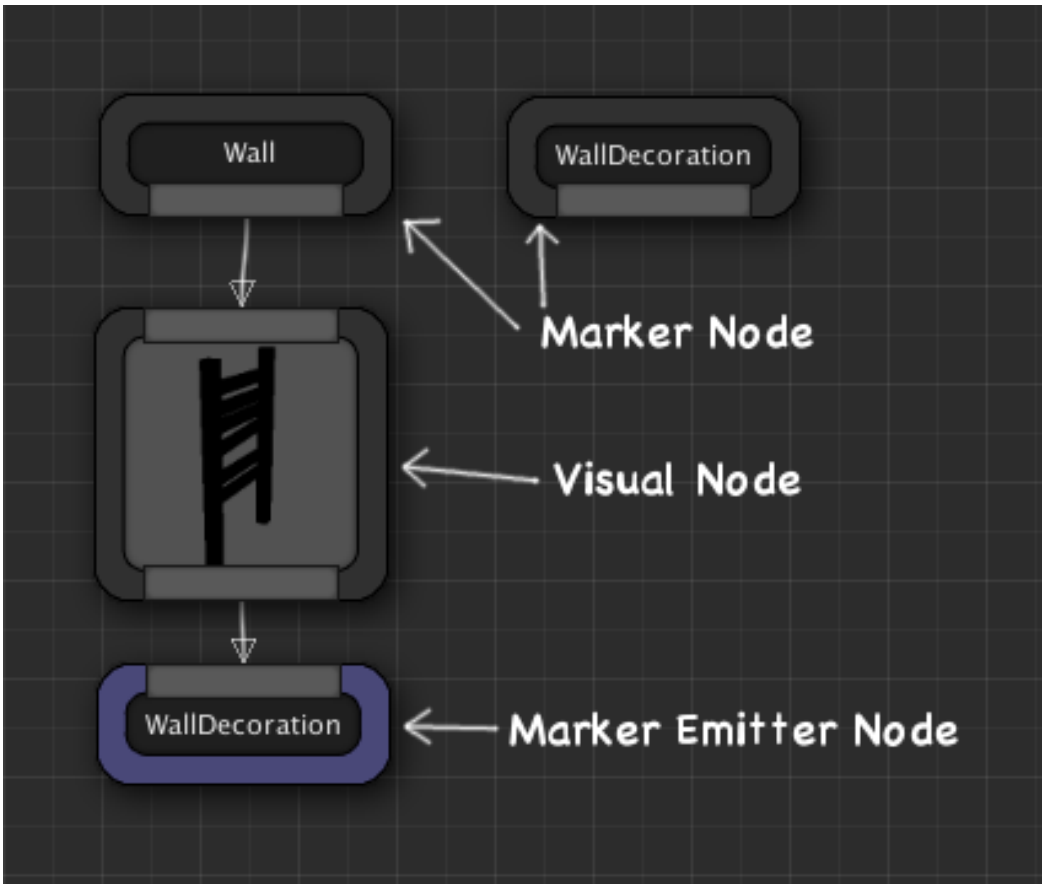


Figure 16: Theme Node categories

6.1 Marker Nodes

After the layout generation phase, the scene would be scattered with invisible named points called **Markers**. Then, for every marker point in the scene, the theming engine looks for a corresponding **Marker Node** with that marker name. If found, it would start executing all the nodes defined below the marker node.

For e.g., if you have a marker node named `Ground`, it would be invoked for every `Ground` marker found in the scene. Once invoked, the theming engine executes all the nodes defined below it from left to right untill a certain condition is met

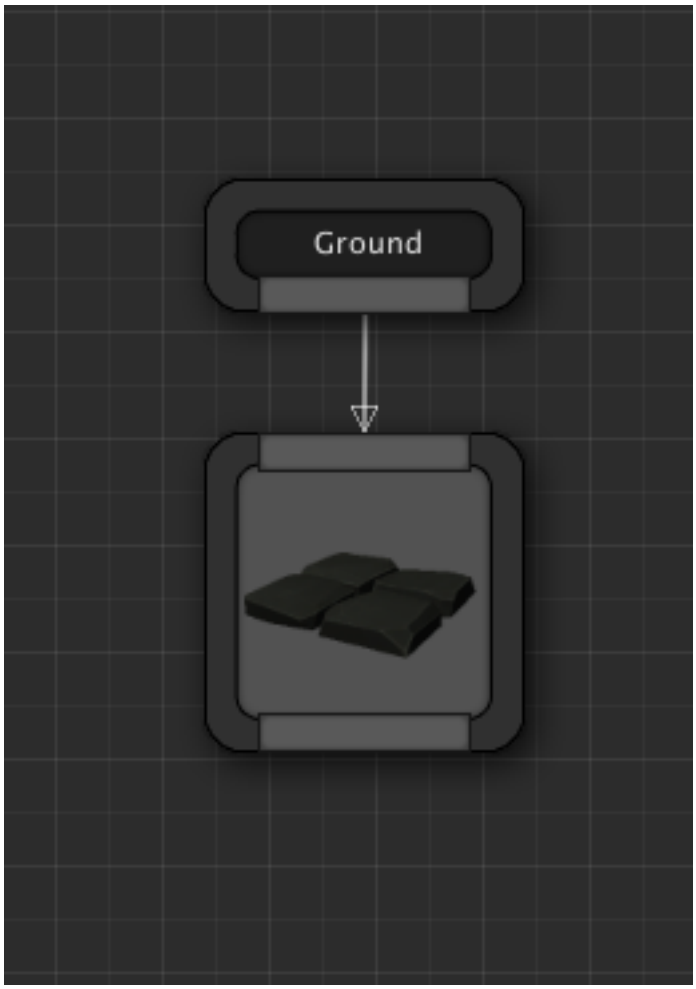


Figure 17: Ground Node

In the above example:

1. In the first phase, the layout builder has populated the map with ground markers, wherever a ground mesh was expected.

2. Then in the next phase, the theming engine ecounters the Ground marker while iterating through all the markers in the scene
3. It then looks for a **Marker Node** named Ground in the theme graph
4. Once found, it executes the visual nodes defined below it, starting from left to right

When you create a new theme asset, the theme graph comes with a set of default marker nodes.

You can define new marker nodes and build your own hierarcy for advanced theming

Names of custom marker nodes can be changed by double clicking on them, or from the details tab

6.1.1 Creating marker nodes

To create a marker node, right click anywhere in the empty area and choose **Add Marker Node**

6.2 Visual Nodes

Visual nodes are used for spawning visual objects into the scene (e.g. any game objects, sprites etc). They are usually attached to a marker node and executed whenever an marker with that name is encountered in the scene. When executed, it spawns a game object defined within it and places it in the scene where the marker was encountered

You can create the following visual nodes:

- **Game Object Node** - Spawns any type of a game object. Expects a game object template (e.g. prefabs)
- **Sprite Node** - Spawns a sprite for your 2D games. Expects a sprite reference. Also have sprite specific properties

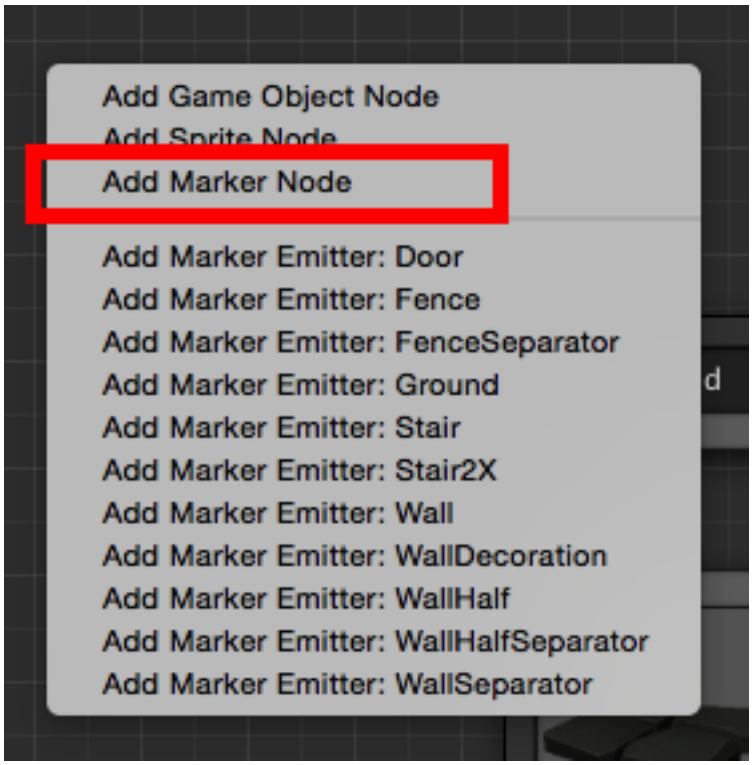


Figure 18: Add new marker node from the Context Menu

6.2.1 Creating visual nodes

There are several ways to create a new visual node:

Drag and drop a game object from the *Project* window on to the theme editor to create a Game Object or Sprite node

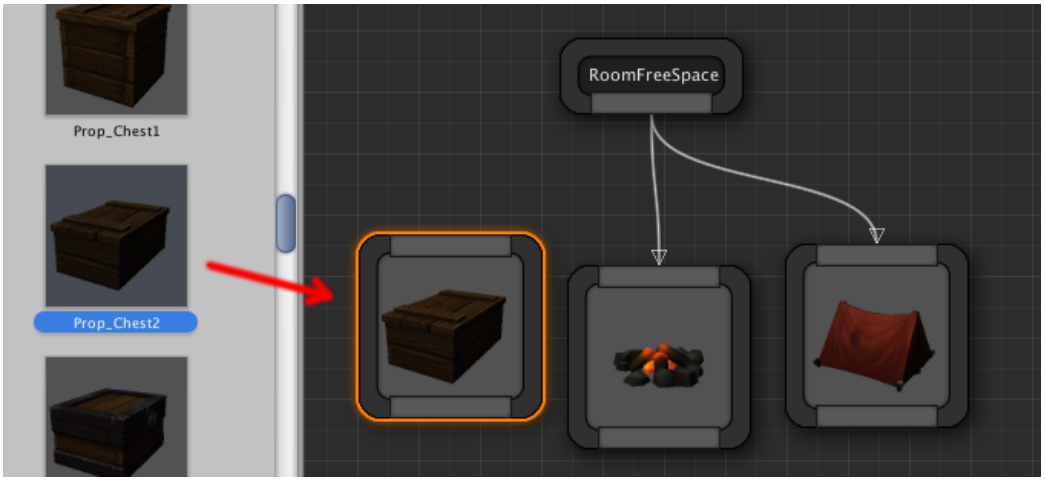


Figure 19: Drag an drop prefabs into the theme editor

Alternatively, drag a link out of the marker node you intend to attach it on and select the appropriate visual node you desire

Then select the node and assign the game object template from the inspector window

6.3 Marker Emitter Nodes

Marker Emitters emit new markers into the scene. These nodes are attached to visual nodes and if the parent visual node is executed, it would insert a named marker into the scene.

Marker Emitter nodes are similar in apperarance to Marker Nodes. However, they are purple in color and have an input pin, instead of an output pin

In the above example, the `Wall` Marker has 3 Mesh nodes attached to it with probability such that any one for the 3 would be randomly chosen.
One of the 3 meshes has a window in it and we would like to decorate that mesh with

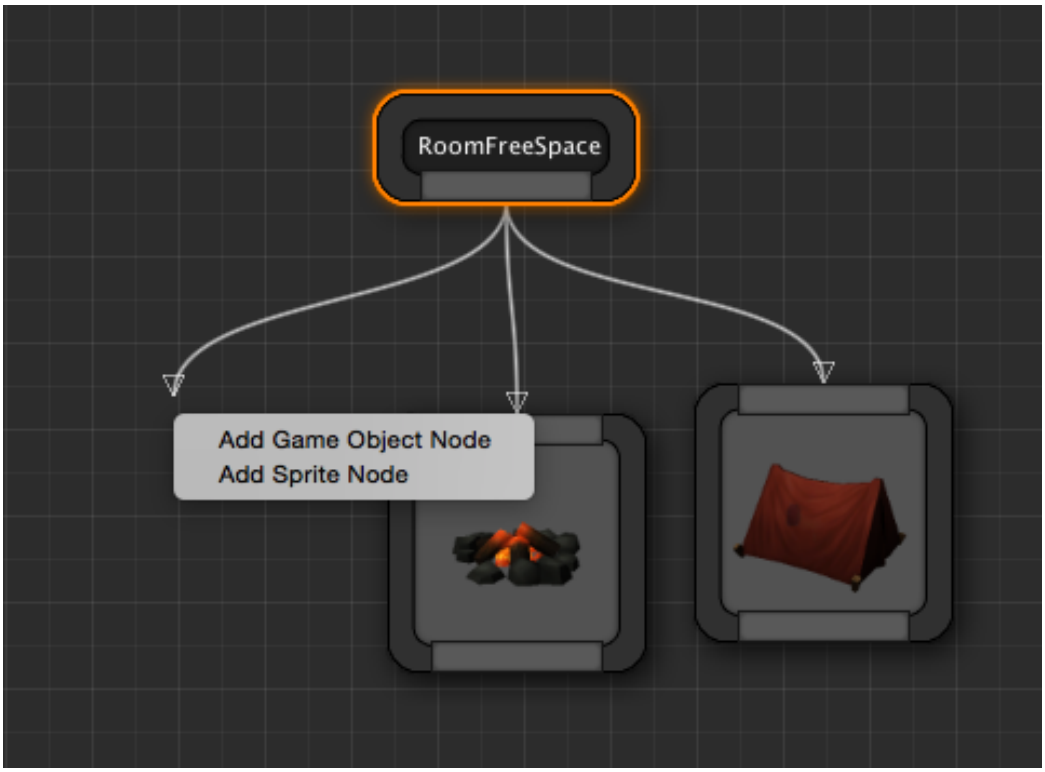


Figure 20: Drag a link from existing marker nodes

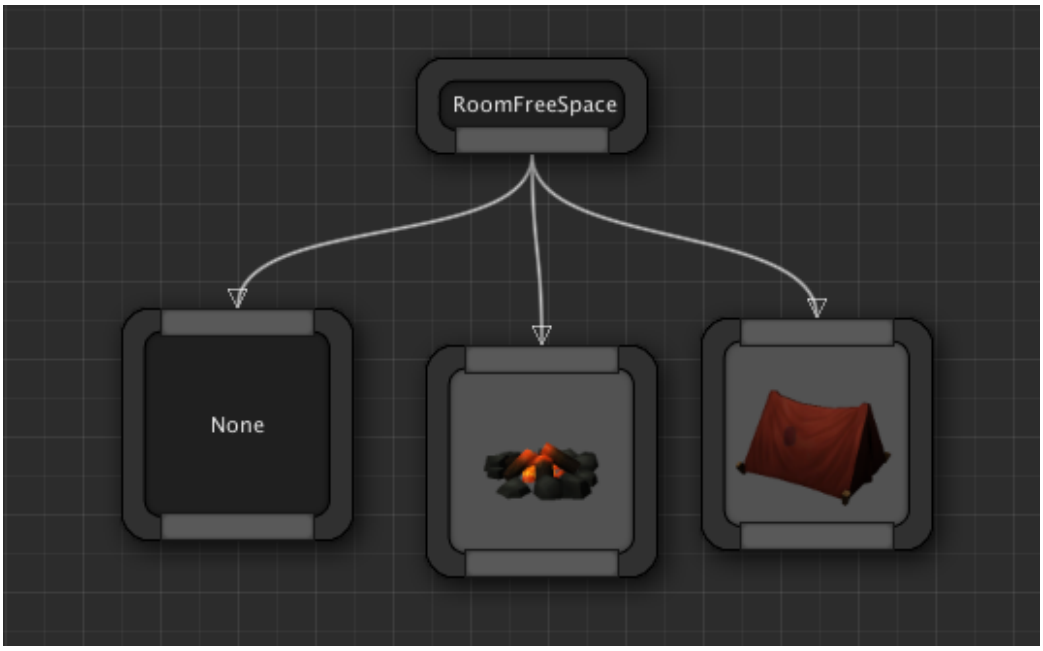


Figure 21: This would create an empty visual node

curtains, but only if that node is selected. So, we define a new Marker named `Curtains` (can be any name) and attach curtain meshes to it. Then we **emit** a `Curtain` Marker Node from the desired visual node. Hence, if the mesh in the middle is executed, it would also insert a marker named *Curtain* in its position. Then the theming engine would execute everything beneath the `Curtain` marker and pick a random curtain and attach to the wall

This ability of defining your own hierarchy lets you design powerful themes for your levels

6.3.1 Creating marker emitter nodes

To create a **Marker Emitter** Node, drag a link out of a visual node and select a marker name you would like to emit

Alternatively, right click anywhere in the empty area and expand the `Marker Emitters` category and click choose a marker to emit

You can create a marker emitter for any of the existing markers in the scene

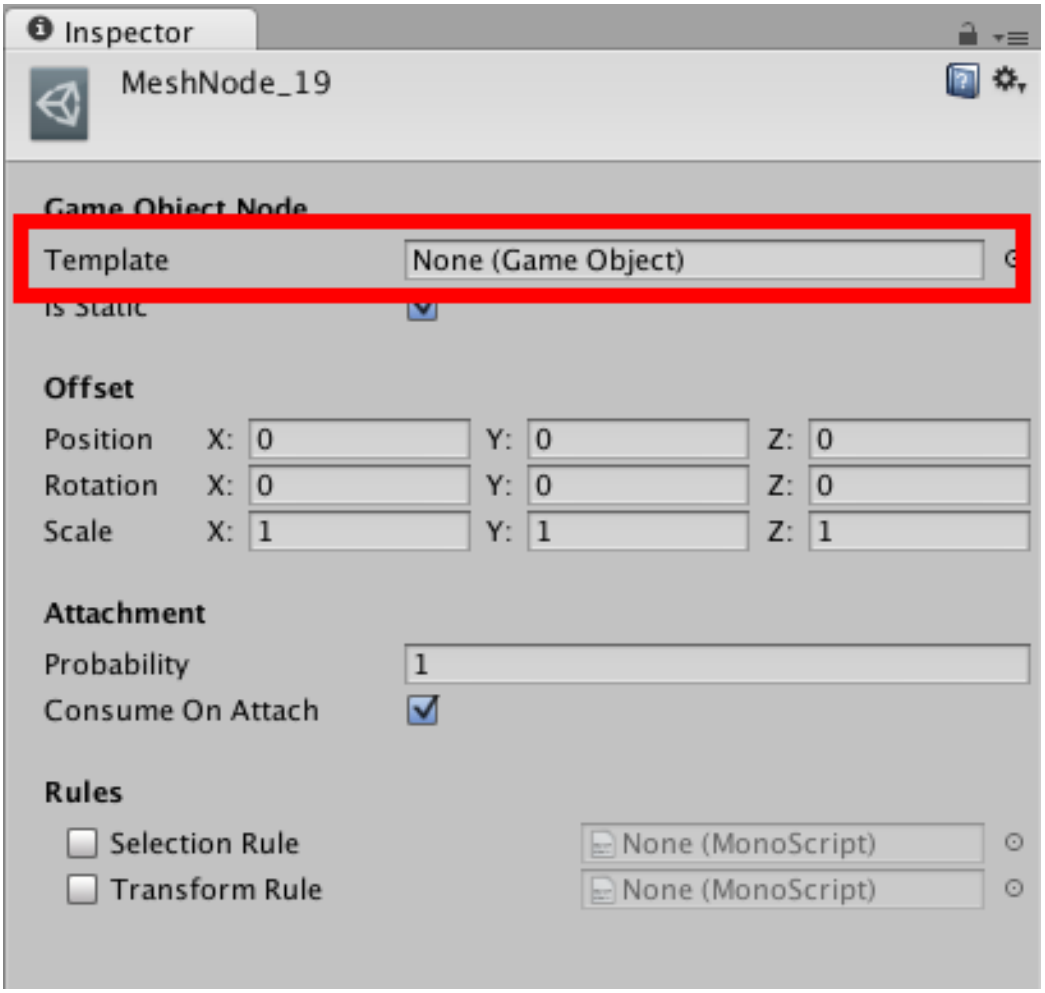


Figure 22: This would create an empty visual node

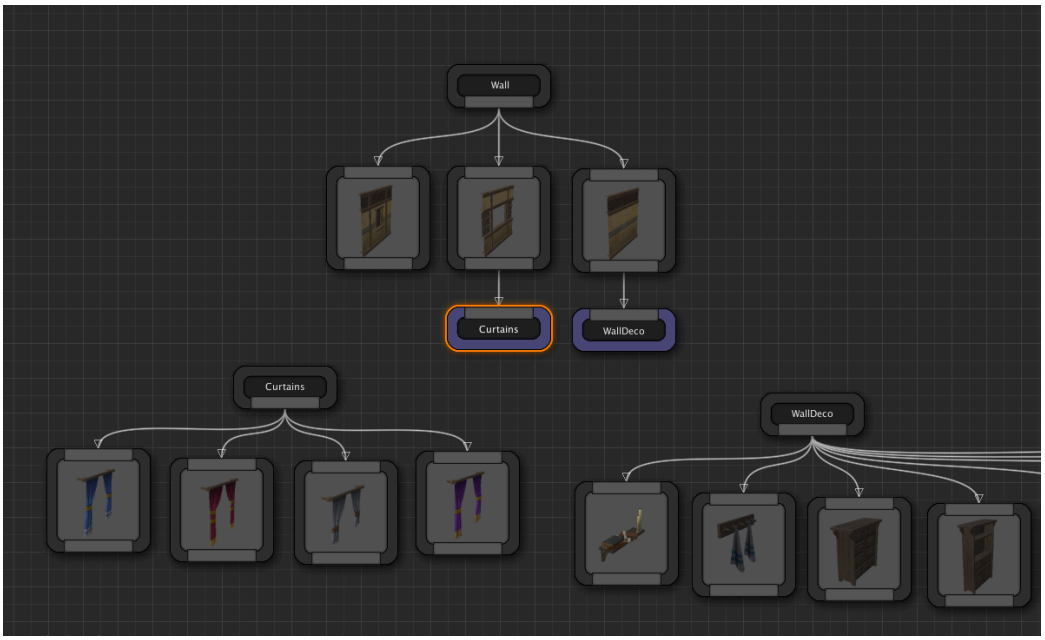


Figure 23: Marker Emitter Sample

6.3.2 Cycles

Cycles are not allowed when you emit markers since we do not want to continuously emit markers in an infinite loop

The editor takes care of not allowing cycles and notifies you with a user-friendly message when you attempt to create a connection with a marker emitter that might cause a loop

7 Theme Node Properties

A node in the theme graph can be customized from the Details Tab

7.1 Visual Nodes

Select a visual node (e.g. a Game Object node) and have a look at the details tab:

Figure 24: Sample Dungeon Scene

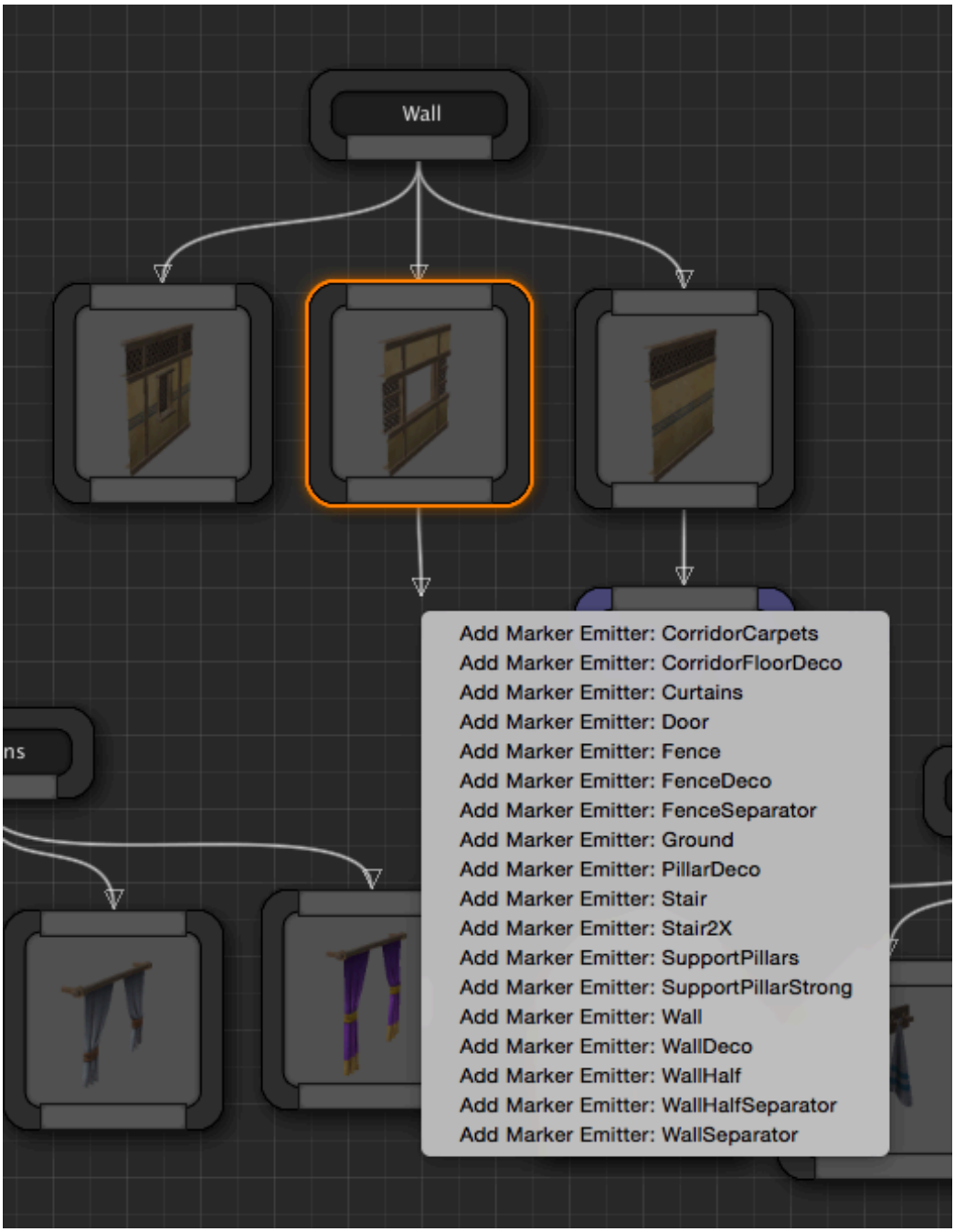


Figure 25: Choose a Marker to emit from the filtered context menu

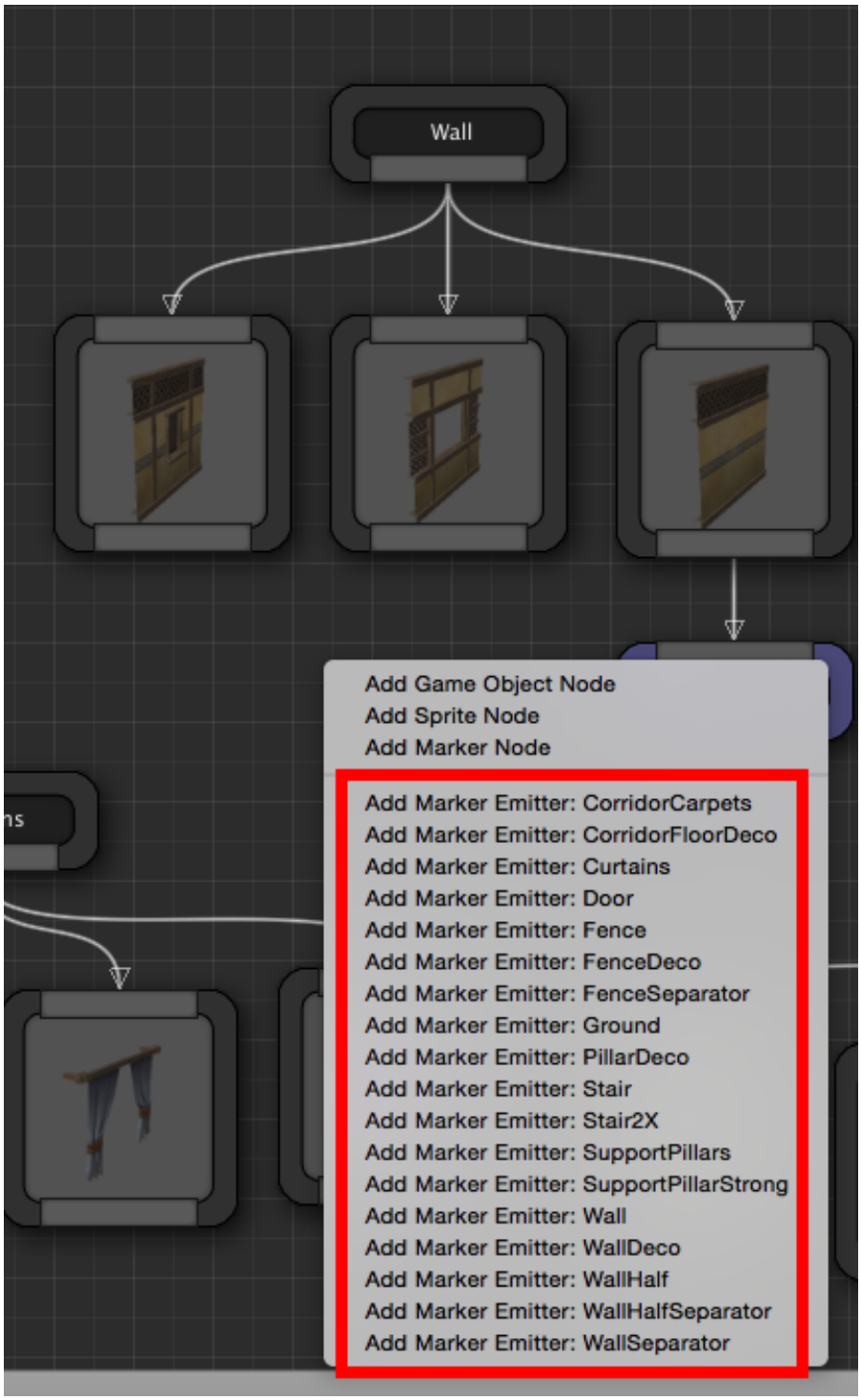


Figure 26: Choose a Marker to emit from the context menu

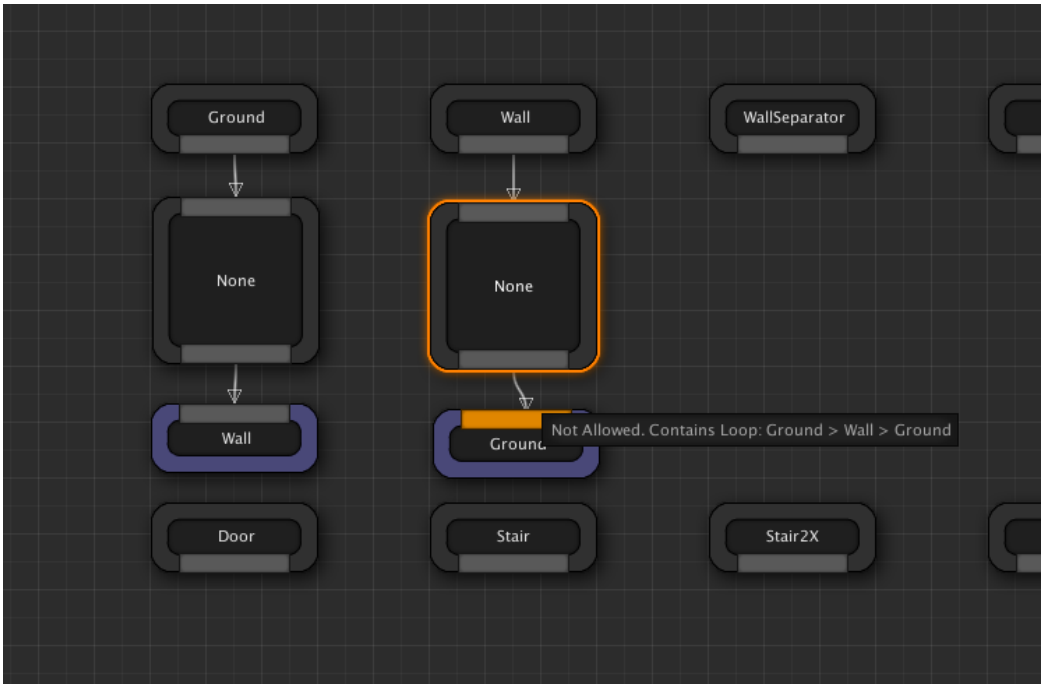


Figure 27: Cycles not allowed

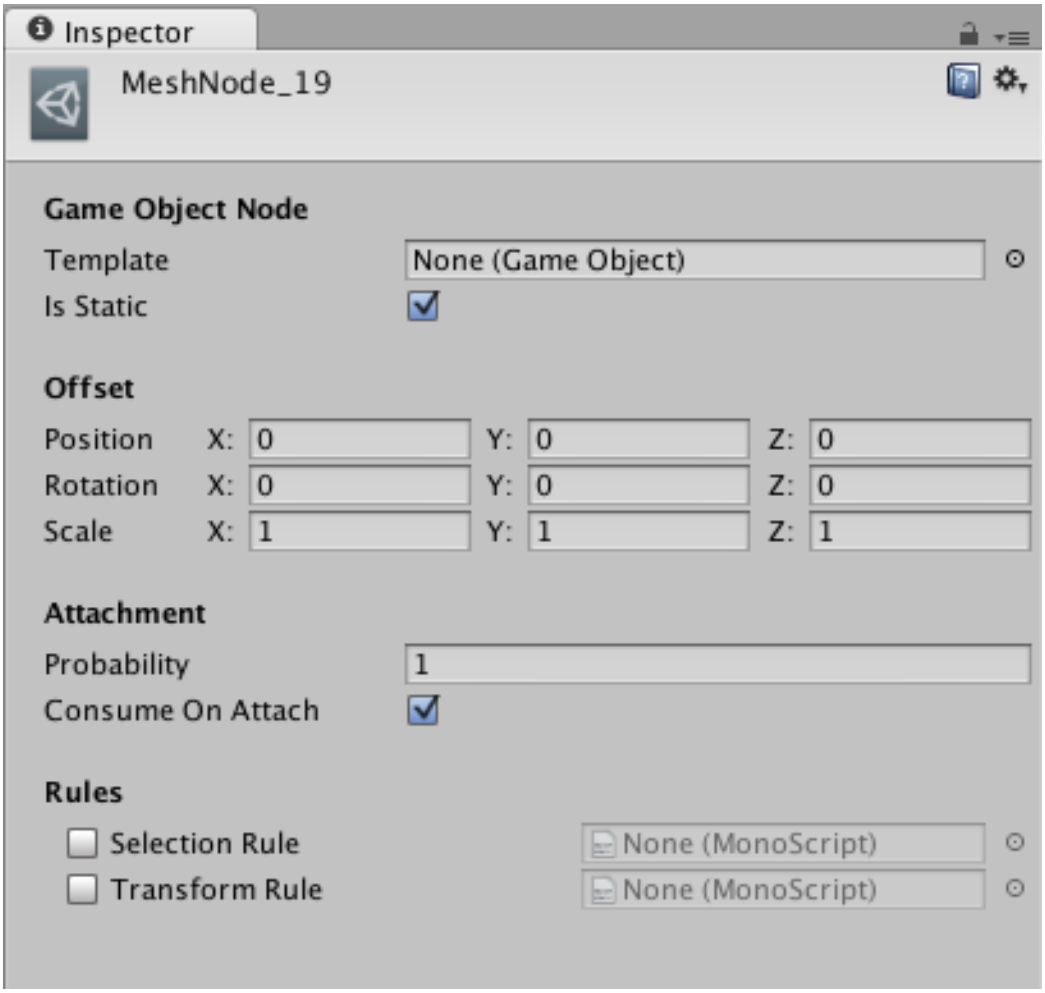


Figure 28: Game Object Node properties

The properties that are common to all Visual nodes (Game Object, Sprite etc) are explained below:

Offset: Apply transformation offset to your visual object relative to the marker location. This is a very useful property while designing your theme. If the pivot of the mesh your artist has designed isn't where you wish it were, you can easily adjust it here to translate / rotate it around. You can also scale objects if they are too small / big. While designing your theme, you'll find this property useful to re-position your visual nodes, if required.

Probability: This is the probability of attachment. When this node is executed, the theming engine looks at this variable and rolls a dice and decides whether to insert this visual object into the scene or not. If this value is 1.0, then it would insert it 100% of the time. If the value is 0, then it would not insert it since selection probability 0%. If it is 0.5, then it would insert it 50% of the time

Consume on Attach: If the visual object was indeed spawned into the scene (based on the probability above), the theming engine would then look at this flag to decide if we need to execute the next sibling visual node. If it is checked, then execution stops for this marker. If it is unchecked, the next sibling gets processed. *Affinity* and *Consume on Attach* can be combined to create interesting possiblities in your theme

Selection Logic: Lets you define selection logic scripts. You have seen an example above of node selected based on random probability (Probability property). The selection process can be far more power than a simple random probability based selection. You can define you own behavior scripts and assign it here so your custom logic can decide if a node is to be selected or not. More details below

Transform Logic: In the Offset property as seen above, you can define a static offset transformation to move/scale/rotate the visual object from the marker position. With Transform Logic, your behavior scripts can provide dynamic tranform offsets based on a logic. For e.g., you might want to rotate/scale/translate a tree randomly to give natural variation instead of having them all face the same direction. More on this later

7.1.1 Game Object Node

A game object not lets you instantiate any type of game object on the scene

Template: Specify a game object template (prefab) to spawn in the scene. The node's thumbnail will update to reflect the game object assigned here

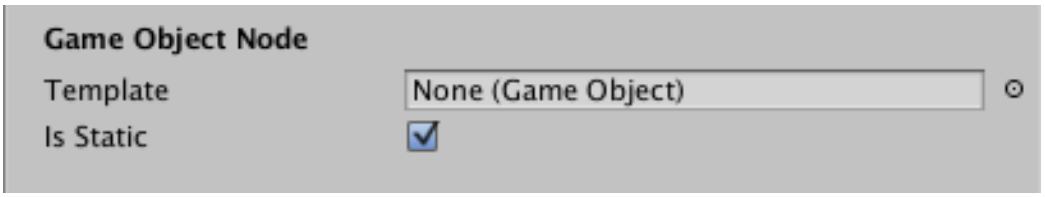


Figure 29: Game Object Node specific properties

Static: Set this if you want to make your object static. If you are spawning dynamic objects (like NPCs), then uncheck this flag

7.1.2 Sprite Class Node

If you are making a 2D game, you'll use **Sprite nodes** to build your scene

Here are the sprite specific parameters:

Sprite: Assign the sprite you would like to spawn with this node. The thumbnail of the node displays this sprite, if assigned

Color: The color tint to assign on the sprite node

Material Override: Specify the different material to use on your sprite (e.g. translucent, masked etc). If unassigned, the default material would be used that is spawned with Unity's sprite object

Sorting Layer Name: The name of the 2D sorting layer used with Unity's 2D framework

Order in Layer: The order this mesh should appear in the layer. This value is set in Unity's 2D sprite object

Static: Set this if you want to make your object static. If you are spawning dynamic objects (like NPCs), then uncheck this flag

7.2 Marker Node

You can change the name of a marker node by setting it's **Name** field

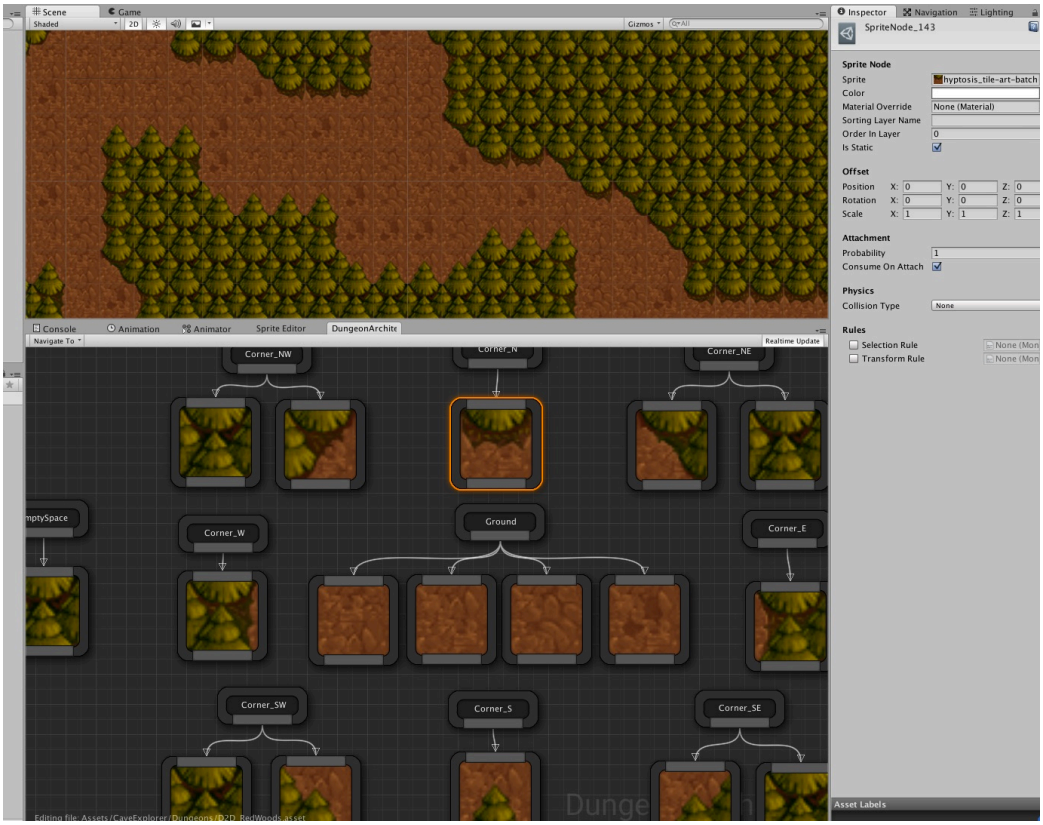


Figure 30: 2D Procedural Scene

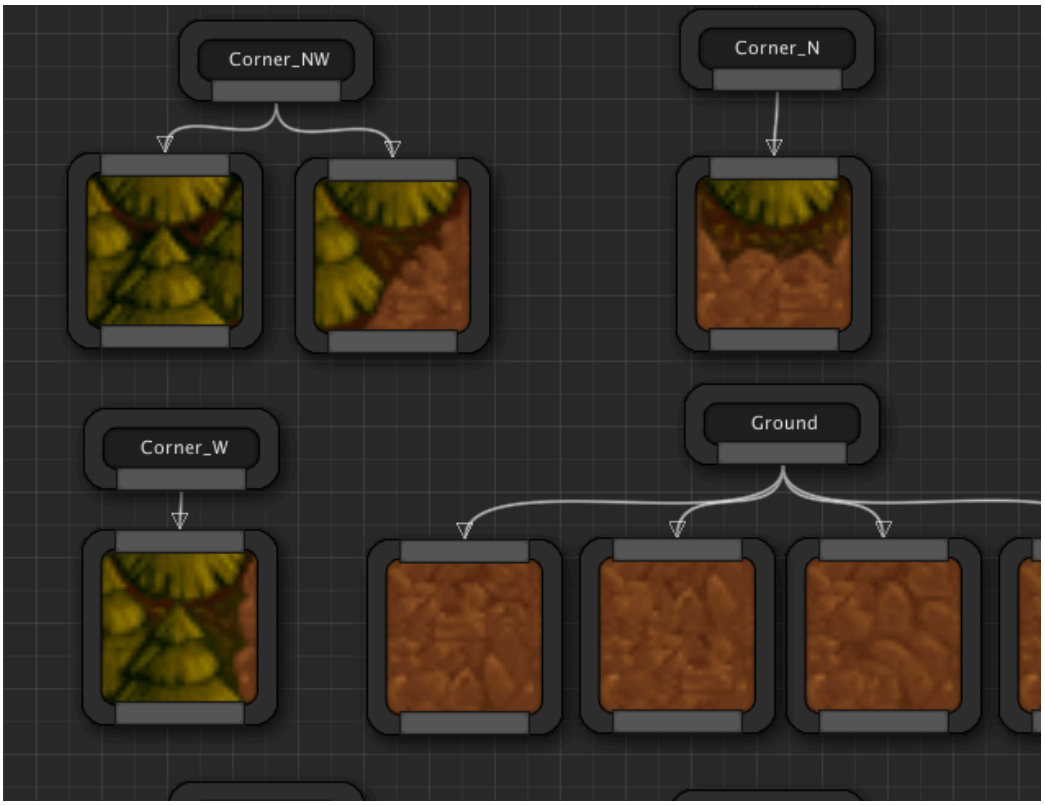


Figure 31: Sprite Nodes

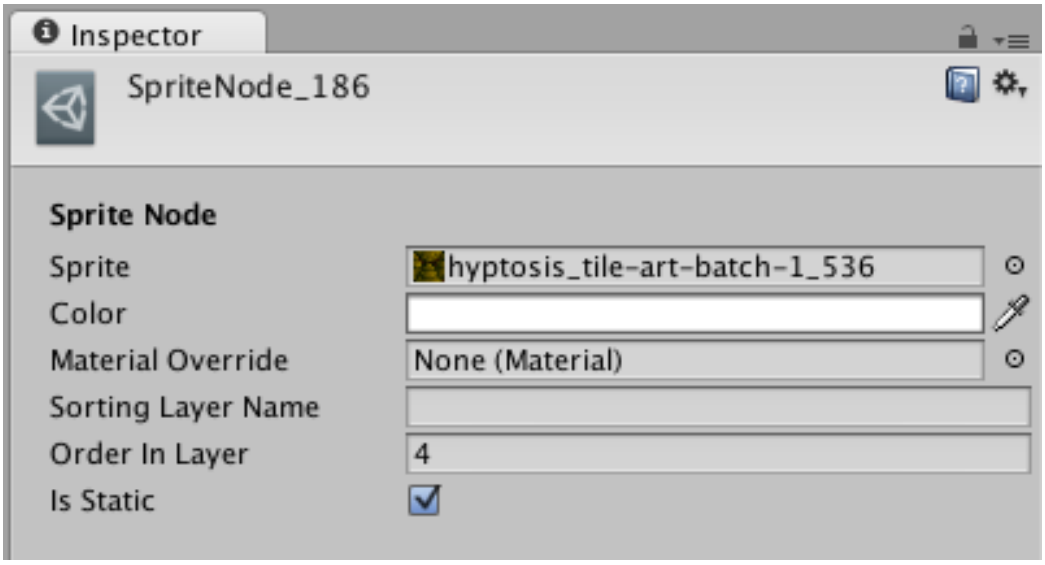


Figure 32: Sprite Node Properties

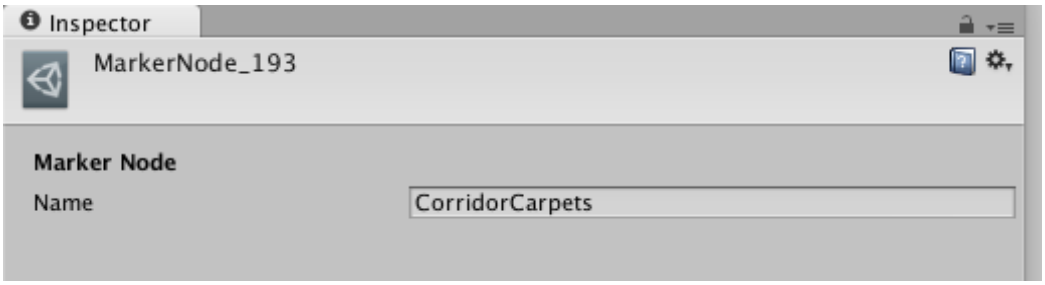


Figure 33: Marker Node Properties

7.3 Marker Emitter Node

When you emit a marker, you can apply an offset to the emitted marker in the **Offset** field

Figure 34: Marker Emitter Node Properties

8 Rules

You can attach script to add logic on the theme nodes for more control. There are two types of rules you can attach to Visual nodes

8.1 Selection Rule

A selection rule is a behavior script that is used to decide if the current node is to be attached to the scene. This rule replaces the default **Probability** property that is used for randomly deciding if visual node needs spawning based on a probability.

Selection rules gives you more power, when you need it. In the rule's script logic, you can query the dungeon model and determine if this node should be inserted into the scene

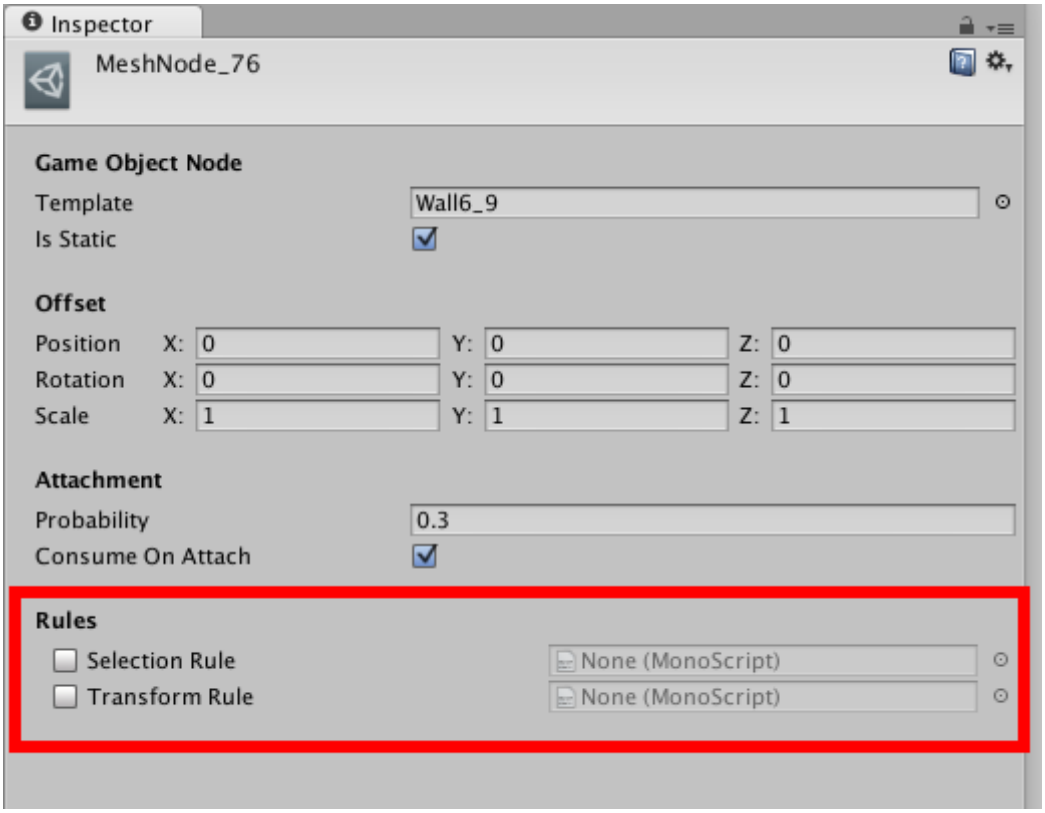


Figure 35: Visual Node Rules

8.1.1 Using Selection Rules

To assing an existing rule into the node, Check the **Use Selection Logic** property and drop in the *Selection Rule* script you would like to attach to the node

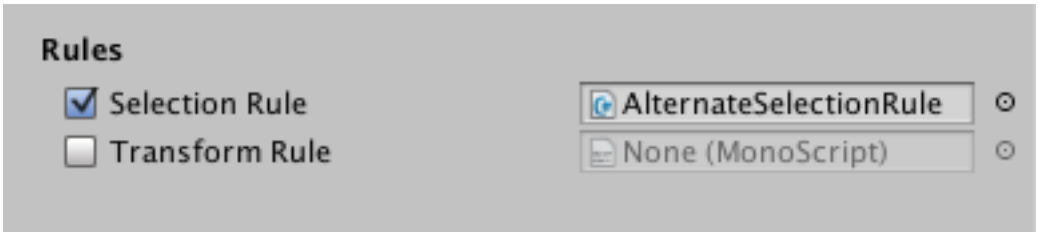


Figure 36: Assign an existing Selection Rule

You can create new Selection Rules by overriding the `AlternateSelectionRule` class under the `DungeonArchitect` namespace

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

public class MySelectionRule : SelectorRule {
    public override bool CanSelect(PropSocket socket, Matrix4x4 propTransform, DungeonM
        bool selected = false;
        // Your selection logic here

        return selected
    }
}
```

8.1.2 Example #1

This theme decorates the sides of the walls with props. Sometimes, they get in the way and block the doors.

A selection rule is used to query the dungeon model and check if it is near a door. If so, it returns false indicating that we don't want to insert it here

```
using UnityEngine;
```

Figure 37: Decorative props blocking the door pathway

Figure 38: Decorative props removed near doors

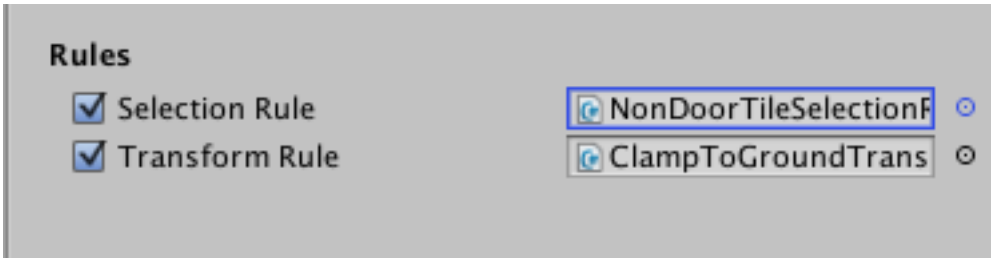


Figure 39: Rule to avoid creation of props near doors

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

public class NonDoorTileSelectionRule : SelectorRule {
    public override bool CanSelect(PropSocket socket, Matrix4x4 propTransform, DungeonM
        if (model is GridDungeonModel) {
            var gridModel = model as GridDungeonModel;
            var config = gridModel.Config as GridDungeonConfig;
            var cellSize = config.GridCellSize;

            var position = Matrix.GetTranslation(ref propTransform);
            var gridPositionF = MathUtils.Divide (position, cellSize);
            var gridPosition = MathUtils.ToIntVector(gridPositionF);
            var cellInfo = gridModel.GetGridCellLookup(gridPosition.x, gridPosition.z);
            return !cellInfo.ContainsDoor;
        } else {
            return false;
        }
    }
}
```

8.1.3 Example #2

In this Diablo like dungeon level, the way our camera is setup, we don't want a room wall to block our view when we are inside a room

So we create non-view blocking fences instead of walls at certain wall facing directions

In the above theme, the rule is attached to the first node, and if true, it would emit a **RoomblockingWall** marker which would create a wall and decorative props. Otherwise, it would proceed to the next node, which emits a **Fence** marker and would create the fence meshes defined under it

This is done with a simple rule that checks the direction of the wall and decides if the view would be blocked from here

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

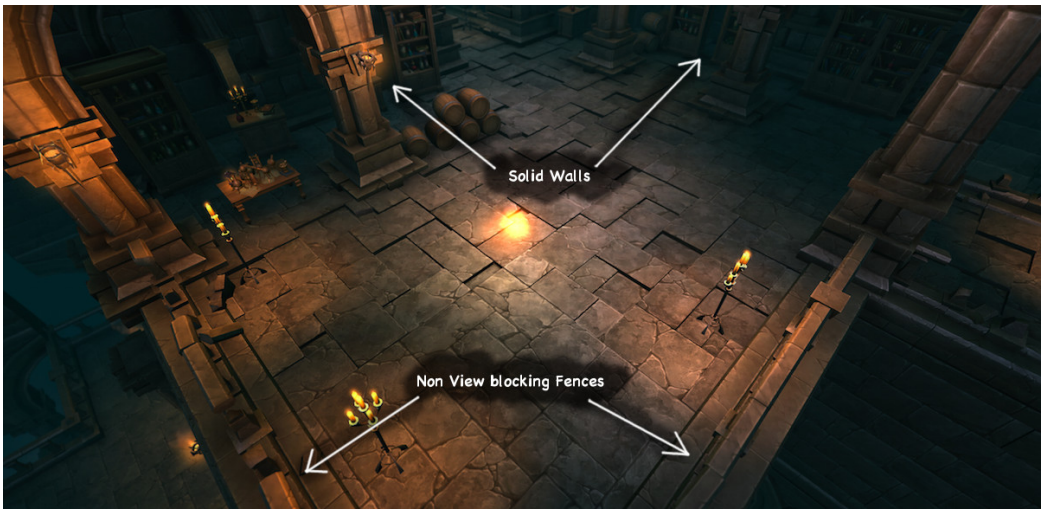


Figure 40: Rule to disallow wall creation in the +X and +Z-axis

Figure 41: View not blocked by walls

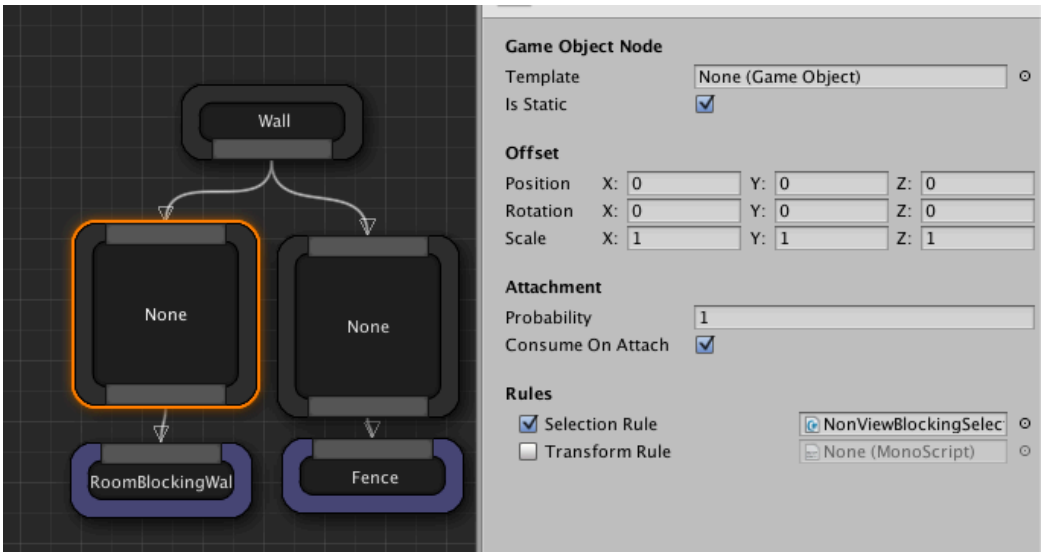


Figure 42: Rule assignment to the wall node

```
using DungeonArchitect;
using DungeonArchitect.Utils;

public class NonViewBlockingSelectionRule : SelectorRule {
    static Vector3[] validDirections = new Vector3[] {
        new Vector3(1, 0, 0),
        new Vector3(0, 0, 1),
    };

    public override bool CanSelect(PropSocket socket, Matrix4x4 propTransform, DungeonM
        var rotation = Matrix.GetRotation(ref socket.Transform);
        var baseDirection = new Vector3(1, 0, 0);
        var direction = rotation * baseDirection;
        foreach (var testDirection in validDirections) {
            var dot = Vector3.Dot(direction, testDirection);
            if (dot > 0.707f) return true;
        }
        return false;
    }
```

```
}
```

8.1.4 Example #3

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

Figure 43: Towers are too 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, DungeonM
        return (socket.gridPosition.x + socket.gridPosition.z) % 2 == 0;
    }
}
```

The above logic uses a checker board pattern, where you sum the X and Y position and return true if it is an even number

Figure 44: Select alternate cells

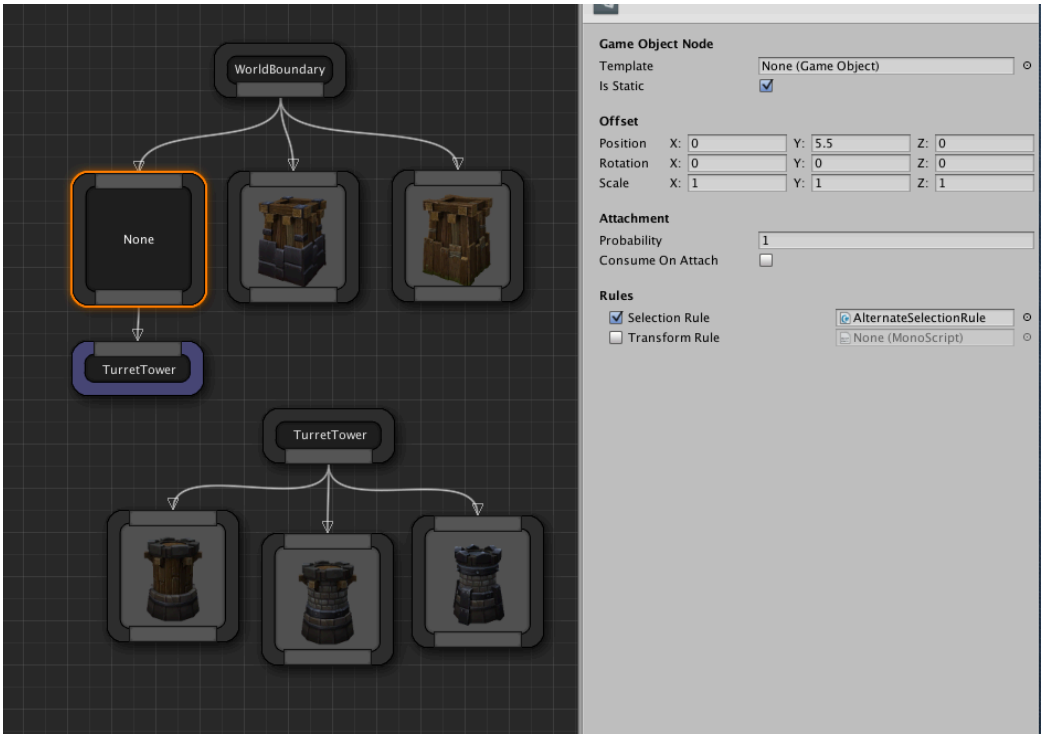


Figure 45: Rule assignment

8.2 Transform Rule

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

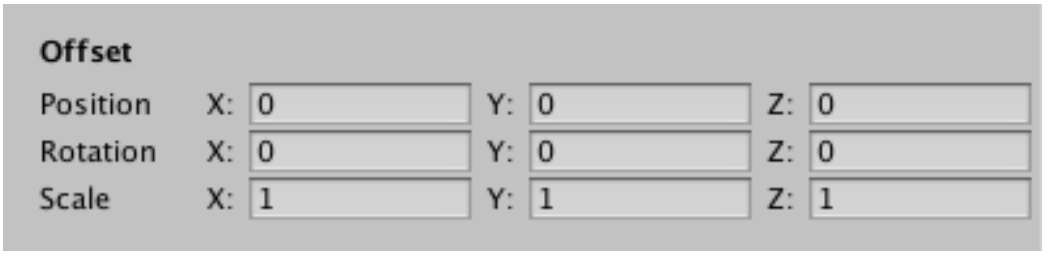


Figure 46: Static node Offset

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 to add variations to your levels for certain props

8.2.1 Using Transform Rules

To assing an existing rule into the node, Check the **Use Transform Logic** property and select the rule you would like to attach to the transform script

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, Matrix4x4
        base.GetTransform(socket, model, propTransform, random, out outPosition, out ou

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

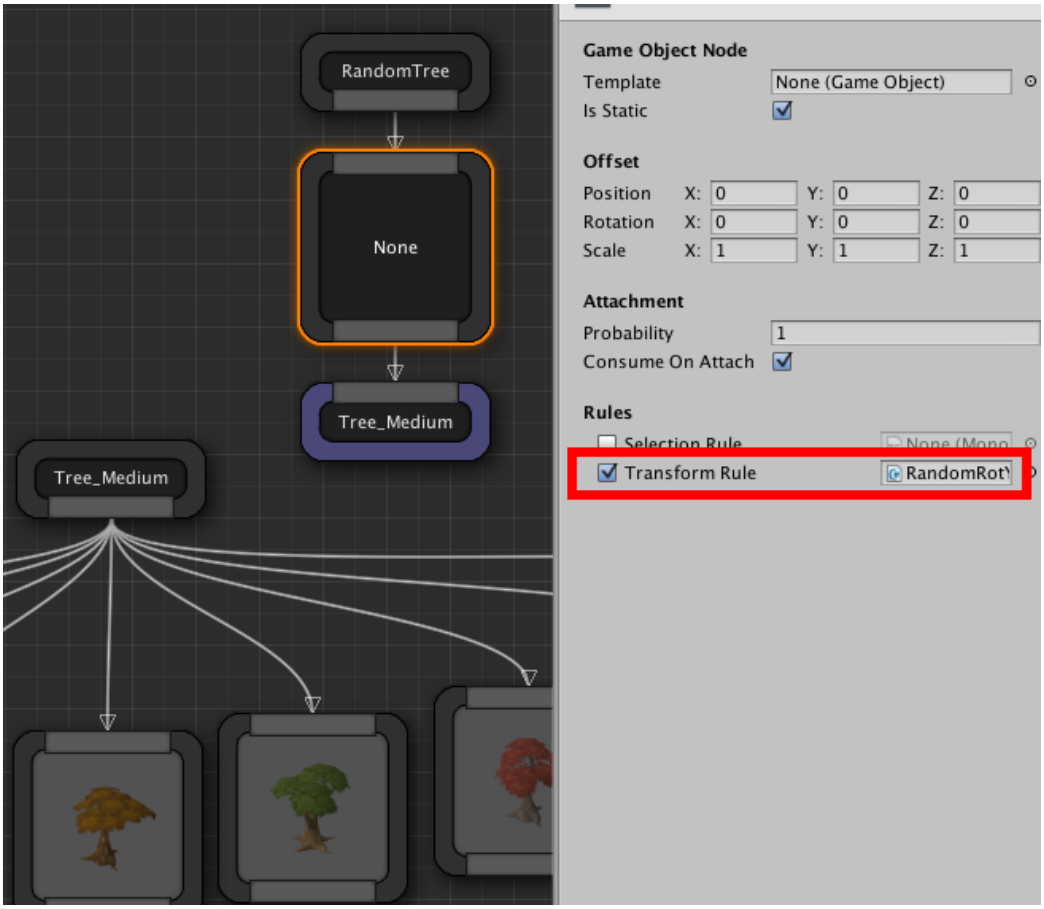


Figure 47: Assigning a Transform Rule

```
    }
}
```

8.2.2 Example #1

In this example, the cliff rocks are facing the same direction and look boring and unnatural

Figure 48: Rocks without transform rules

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

public class RandomCliffTransformRule : TransformationRule {

    public override void GetTransform(PropSocket socket, DungeonModel model, Matrix4x4
        base.GetTransform(socket, model, propTransform, random, out outPosition, out ou

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

Figure 49: Rocks randomly rotated and slightly translated

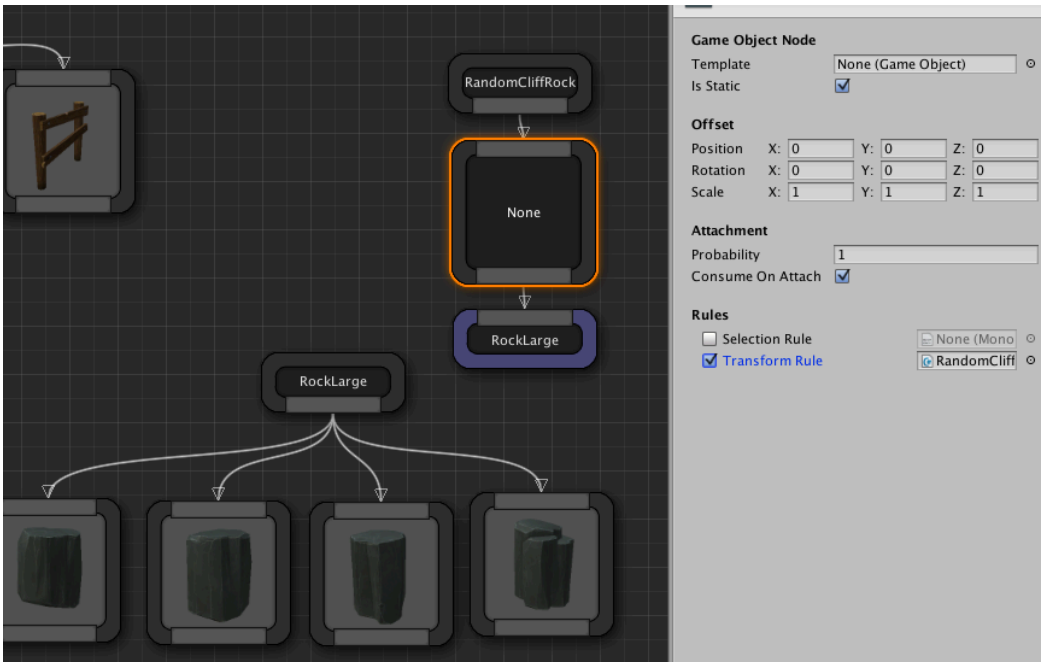


Figure 50: Rule assignment on the rock's base node

```
        outRotation = rotation;

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

A similar rule is applied to trees to randomly rotate them along the Y-axis and randomly scale them slightly

8.2.3 Example #2

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

Figure 51: Tree spawned near the dungeon layout

However, we also have a terrain that Dungeon Architect modifies, whose steepness value is controlled by the user using a curve.

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

Figure 52: Proper offset applied to move it to the terrain ground

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, Matrix4x4
        base.GetTransform(socket, model, propTransform, random, out outPosition, out ou

        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, position.z

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

8.2.4 Example #3

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

Figure 53: Transform rule applied to ground tiles

```
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, Matrix4x4
        base.GetTransform(socket, model, propTransform, random, out outPosition, out ou

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

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

9 Paint Mode

Dungeon Architect also allows you to paint your own dungeon layouts with an editor extension. This gives you more artistic control as you are no longer restricted by what the procedural algorithm creates for you

To Activate the Paint Editor mode and start painting, expand the DungeonGrid game object and select the PaintMode game object. This would change your editor's scene view into **Paint Mode**

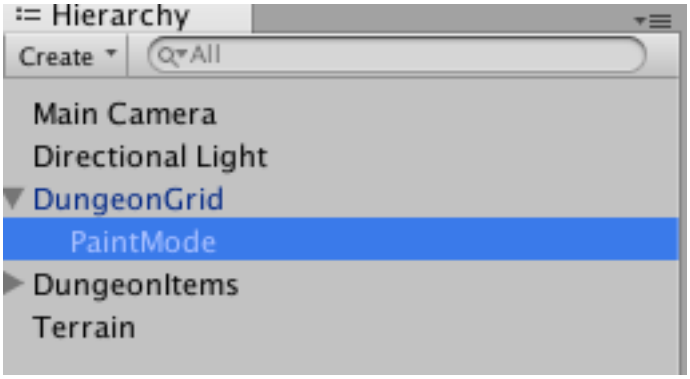
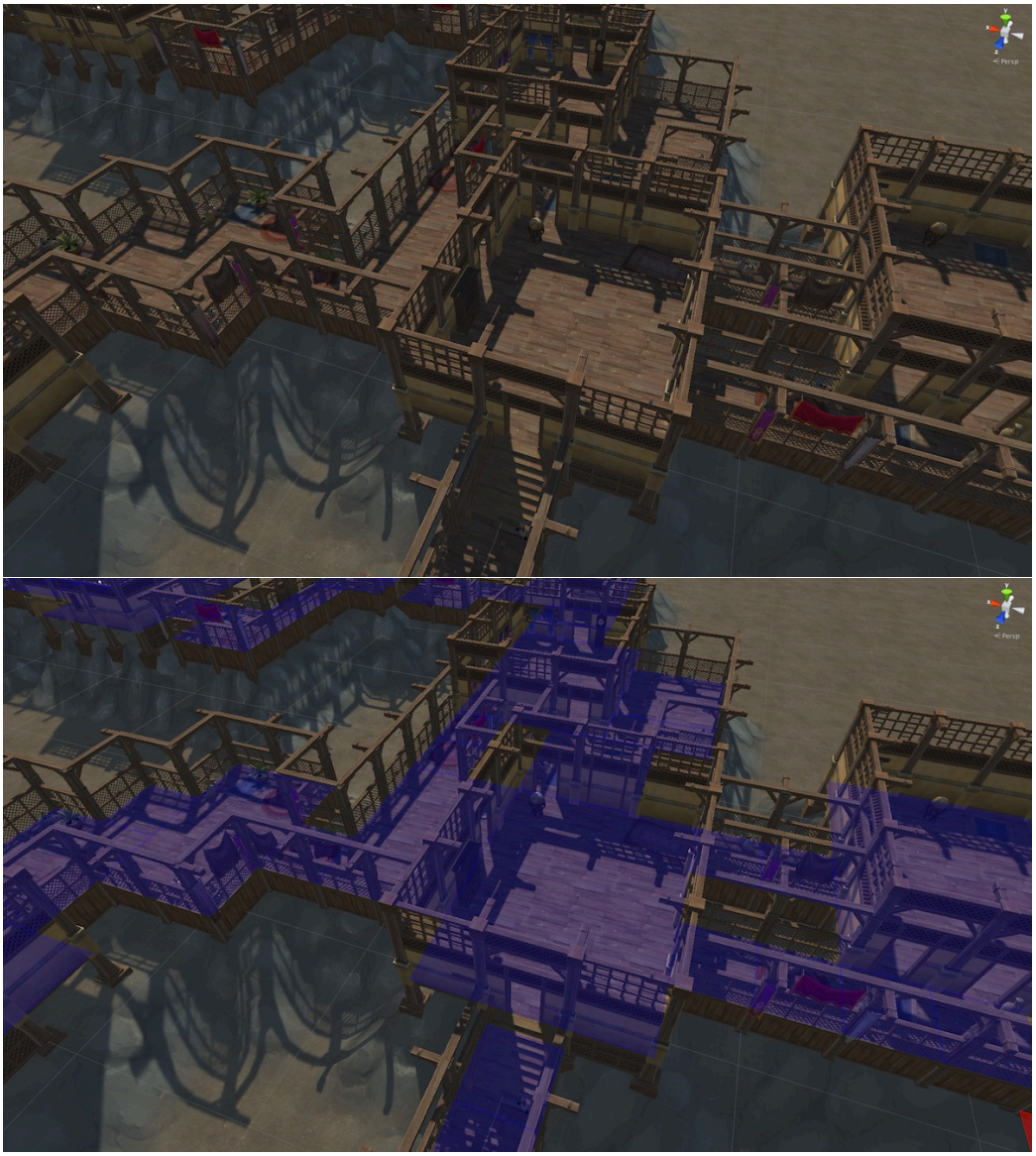


Figure 54: Activate Paint Mode

When you are in the Paint Mode, the Scene View shows the layout of your dungeon in Blue



You can now paint your layout on the Scene View

- **Left Click:** Paint layout
- **Shift + Left Click:** Delete painted layout
- **Mouse Wheel:** Change paint height

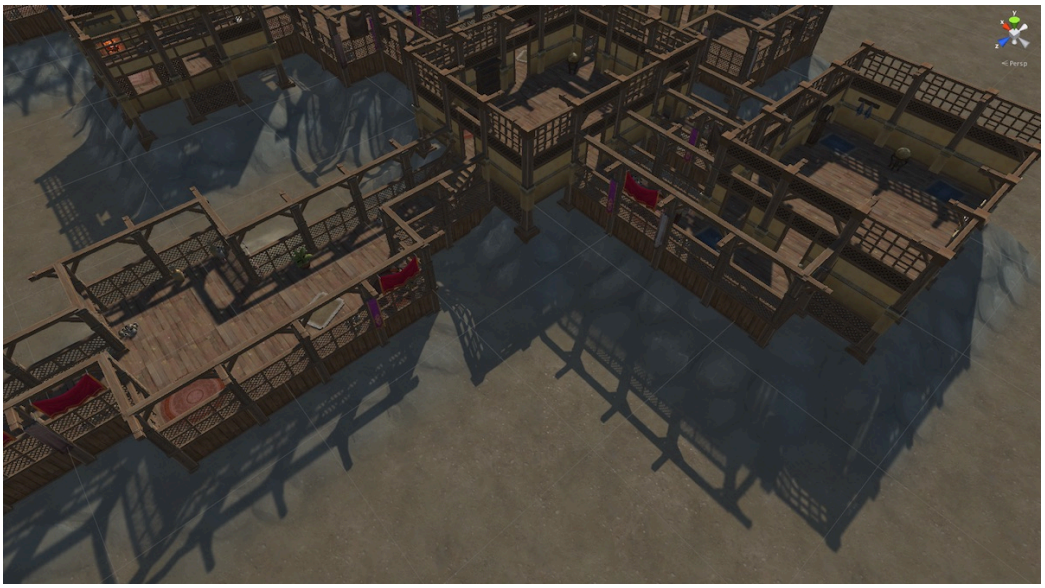


Figure 55: Starting Scene

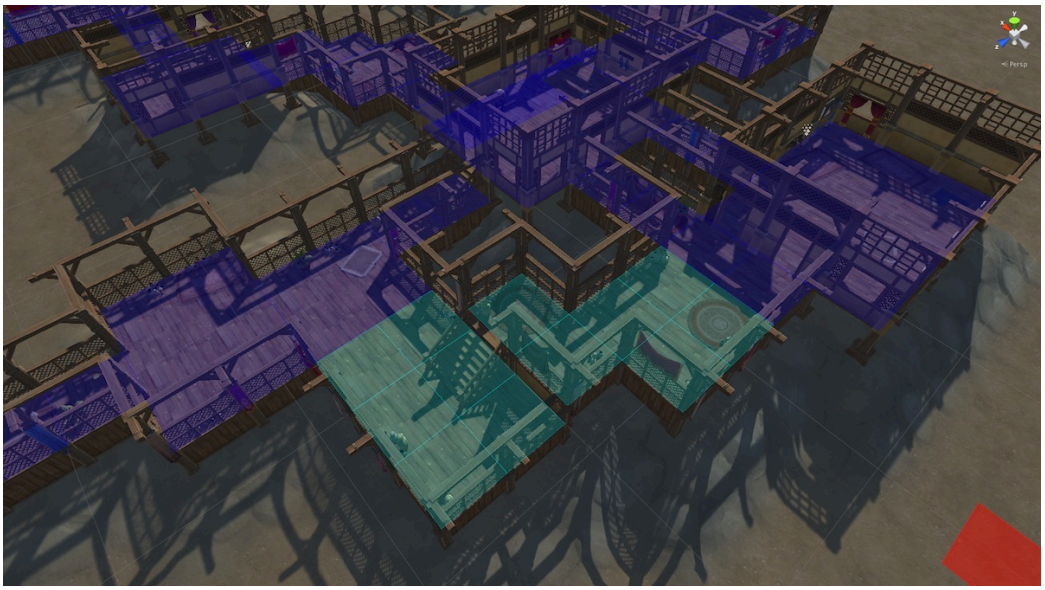


Figure 56: Painted cells show up in Cyan color

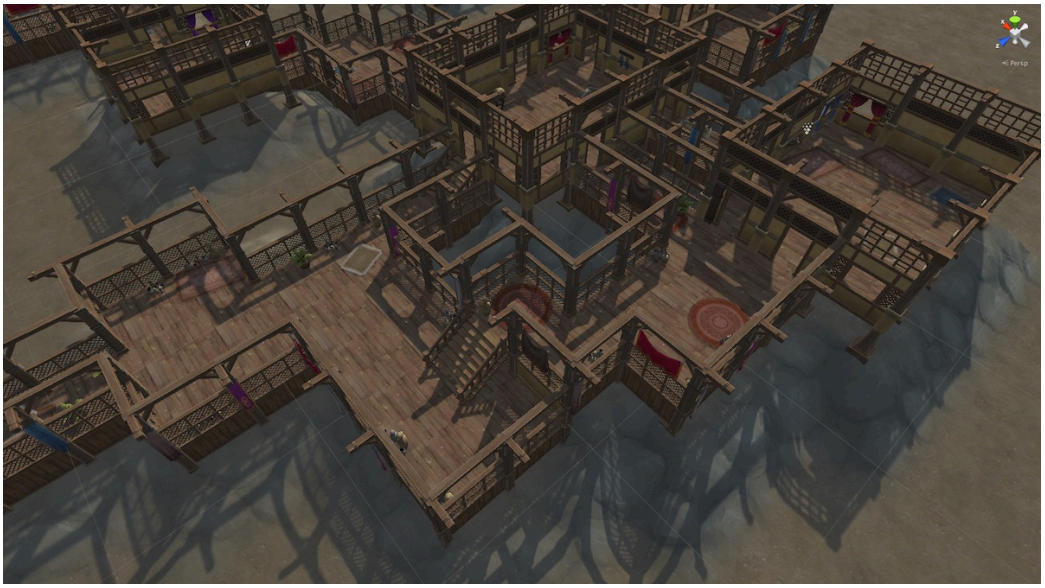


Figure 57: Modifed layout

Another Example: 

9.1 Paint Mode Properties

When you select the Paint mode game object, you can set various parameters to control the paint tool:

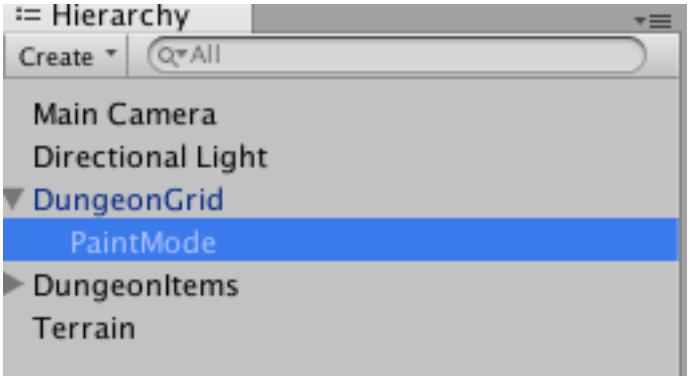


Figure 58: Paint Mode Game Object

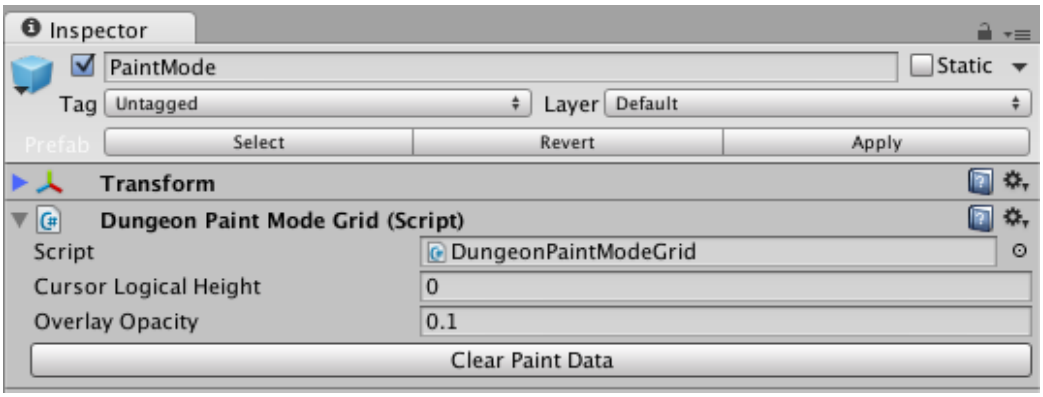


Figure 59: Paint Mode Properties

9.2 Non-Procedural Painting

If you do not want any procedural content to be generated when you paint your level, then set the **Num Cells** property to `0` in your Dungeon Actor's Configuration section

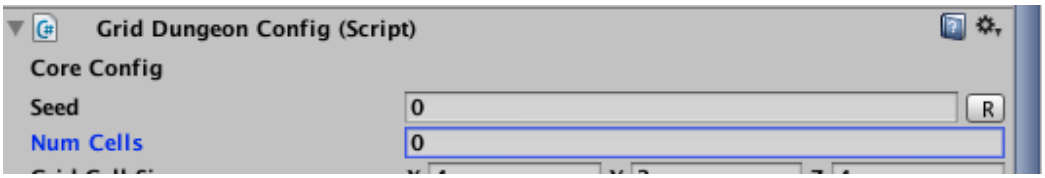


Figure 60: Stop procedural layout generation

This way the dungeon would have an empty layout, allowing you to paint from scratch

If you do want procedural content but want to remove certain procedural areas that are getting in your way, then use a *Negation Volume*

10 Volumes

Dungeon Architect provides various volumes to help you influence your dungeon as per your requirements.

You can find the various volume prefabs under `Assets/DungeonArchtitect/Prefabs`

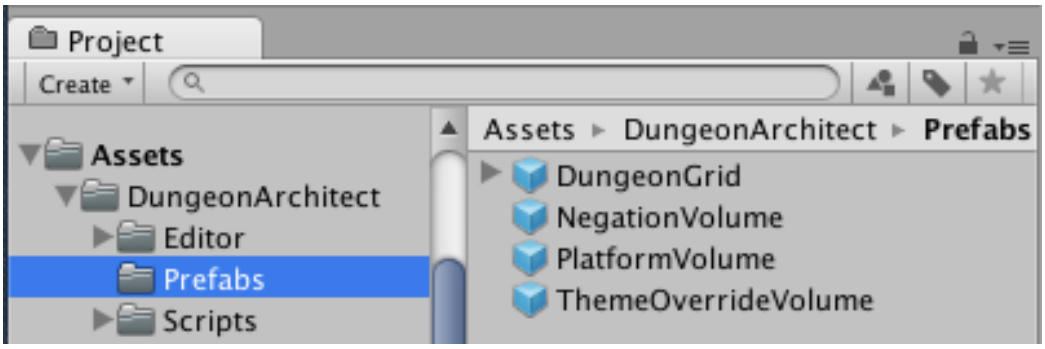


Figure 61: Platform Volume Prefab

10.1 Platform Volume

Place a platform volume anywhere in the scene and Dungeon Architect would adjust the dungeon layout and create a platform (room or corridor) at that location. Scale the volume along the XZ plane to change the size of the generated platform. You can move the platform volume with the move tool to the desired location. (Rotation is not supported)

This gives you artistic control and lets you manipulate the dungeon to suit your needs

To place a platform volume, navigate to `Assets/DungeonArchtitect/Prefabs`

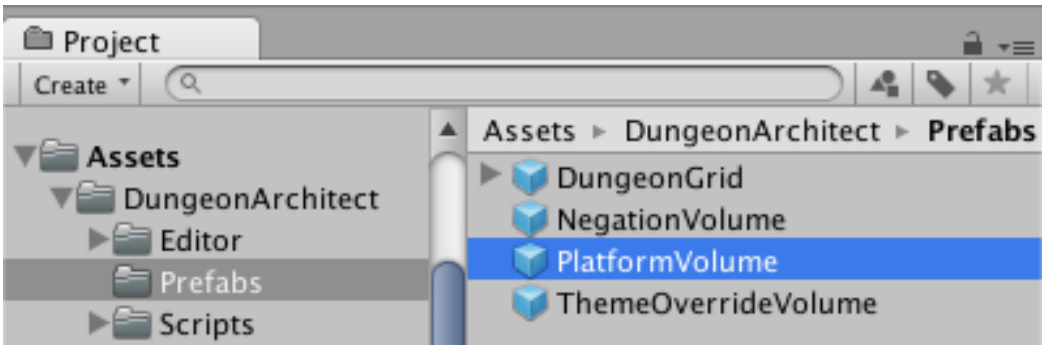


Figure 62: Platform Volume Prefab

Drag and drop the Platform Volume Prefab into the scene view

Select the platform volume and have a look at it's properties

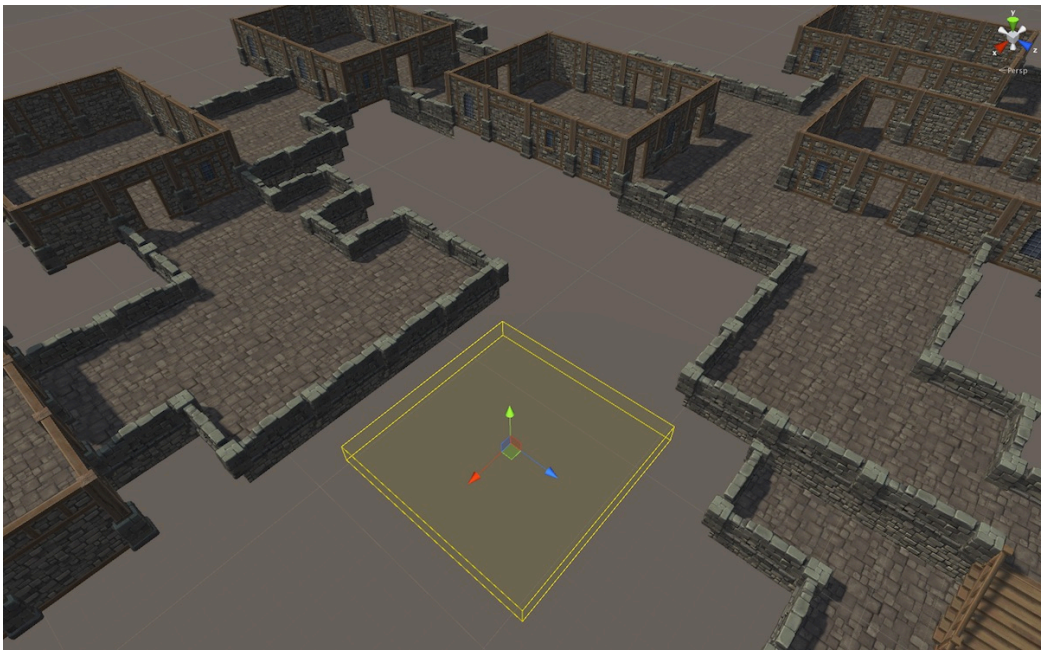


Figure 63: Platform Volume Prefab

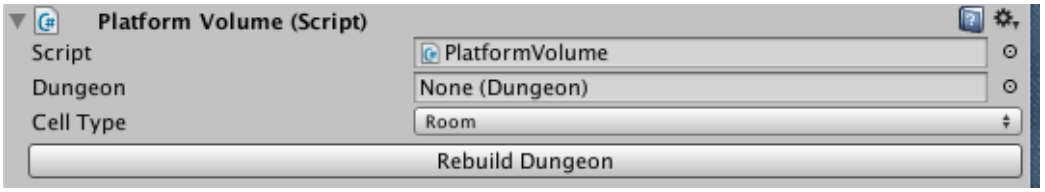


Figure 64: Platform Volume Properties

The Volume needs to know which dungeon the volume belongs to (DA Supports mulitple dungeons within the same scene).

Assign the dungeon you'd like this volume to affect in the **Dungeon** field

Select the type of cell to create on this platform's location (Room or Corridor)

Corridors form isolated platforms in the dungeon which merge nicely with existing corridor cells

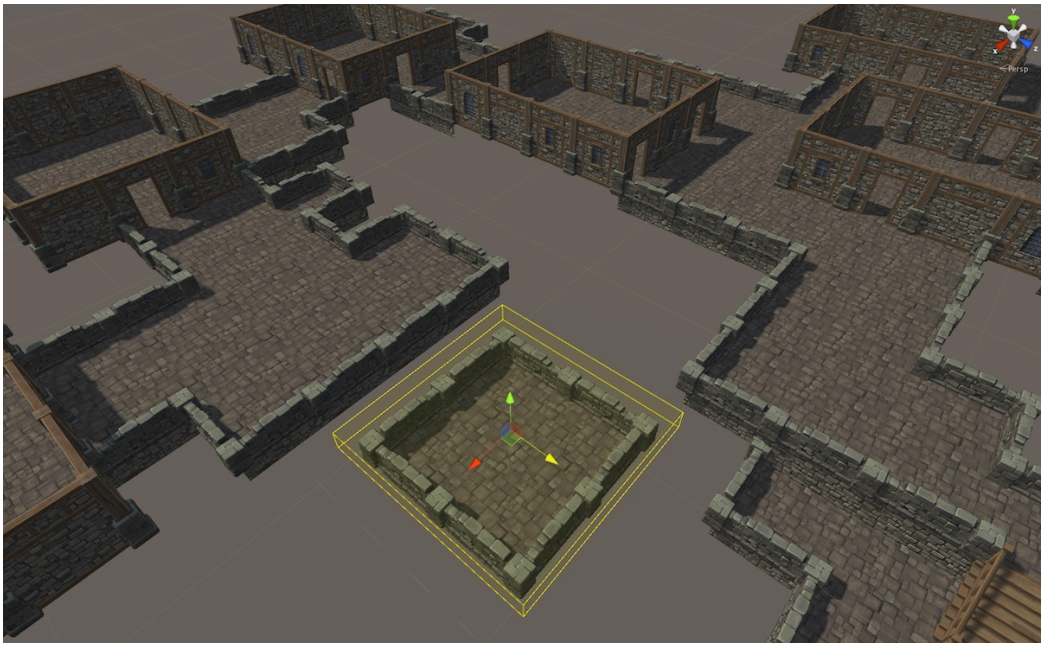


Figure 65: Corridor platform

Rooms always connect to atleast one other room in the dungeon. Changing the Cell type to *Room* creates this result

A button to rebuild the dungeon is provided for convenience. It rebuilds the dungeon in the scene

10.2 Theme Override Volume

Give certain areas of you dungeons a different look and feel. Layout inside this volume would use the theme defined by this volume.

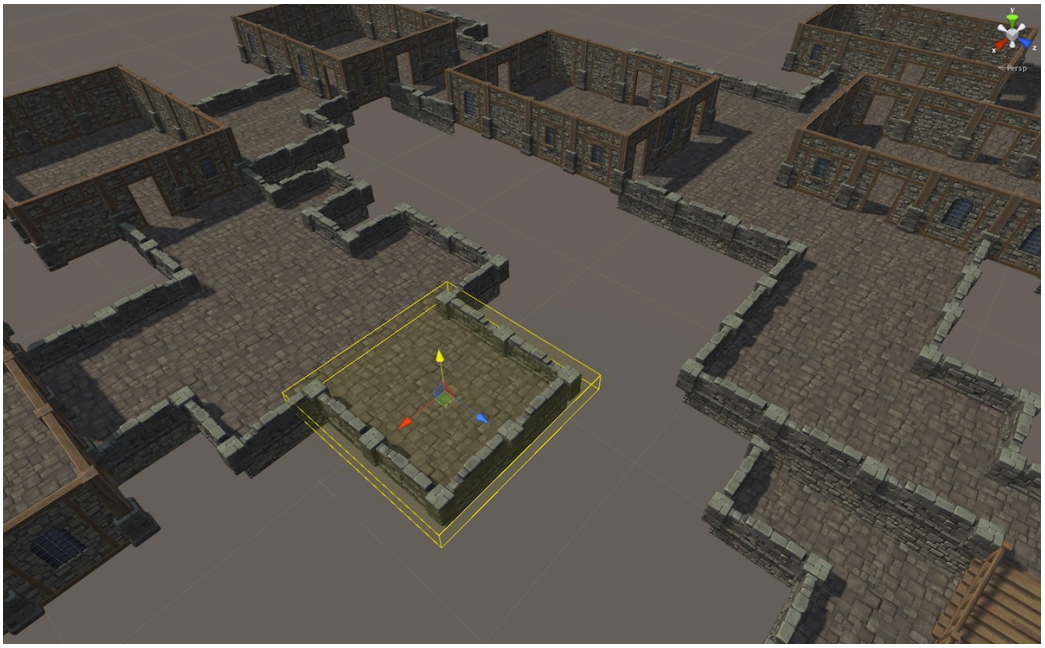


Figure 66: Merges nicely with existing procedural layout

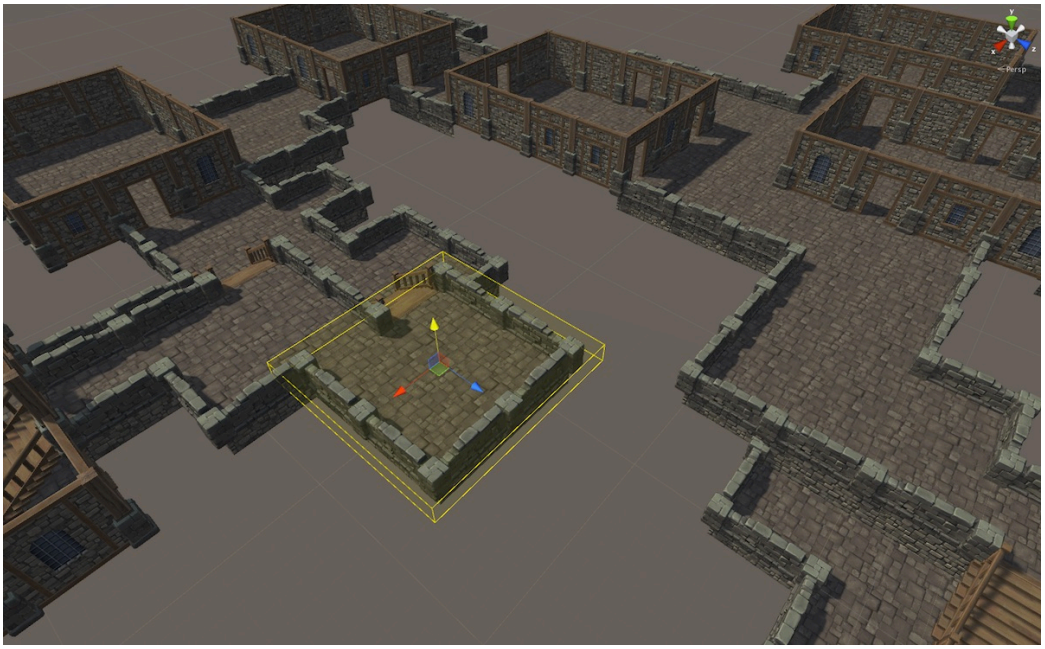


Figure 67: Volume moved up along the Y-axis

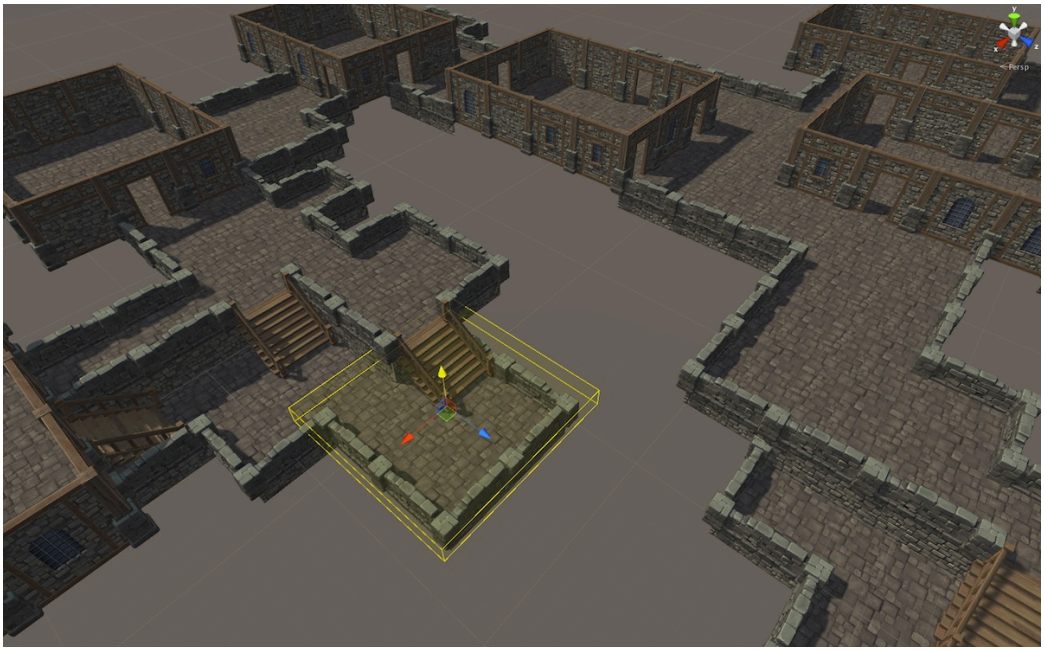


Figure 68: Volume moved down along the Y-axis

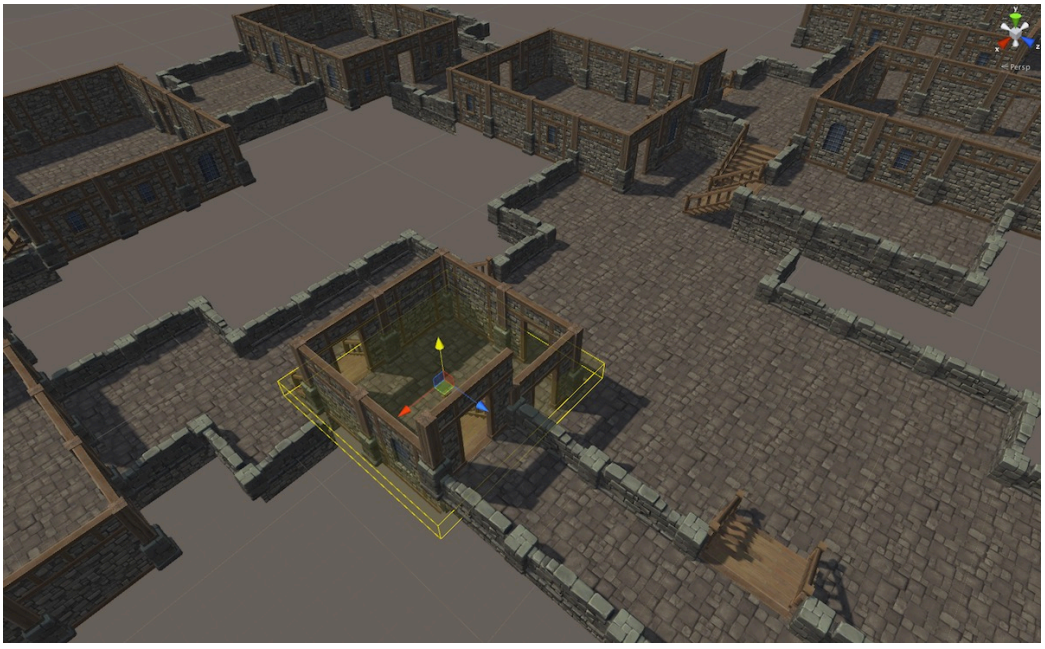


Figure 69: Room platform

This is useful for adding variations to your level

Figure 70: Sample Dungeon

Select the theme override volume and have a look at it's properties

Dungeon: Set the dungeon game object this volume should affect

Override Theme: Set the dungeon theme asset you would like to apply to the geometry within this volume

Note: When overriding, the themes needs to be designed for the same grid cell size for proper results

A button to rebuild the dungeon is provided for convenience. It rebuilds the dungeon in the scene

10.3 Negation Volume

This volume removes all procedural geometry inside of this volume. Use this to get rid of procedural geometry in areas you do not need or when it is getting in the way while manually painting your layout

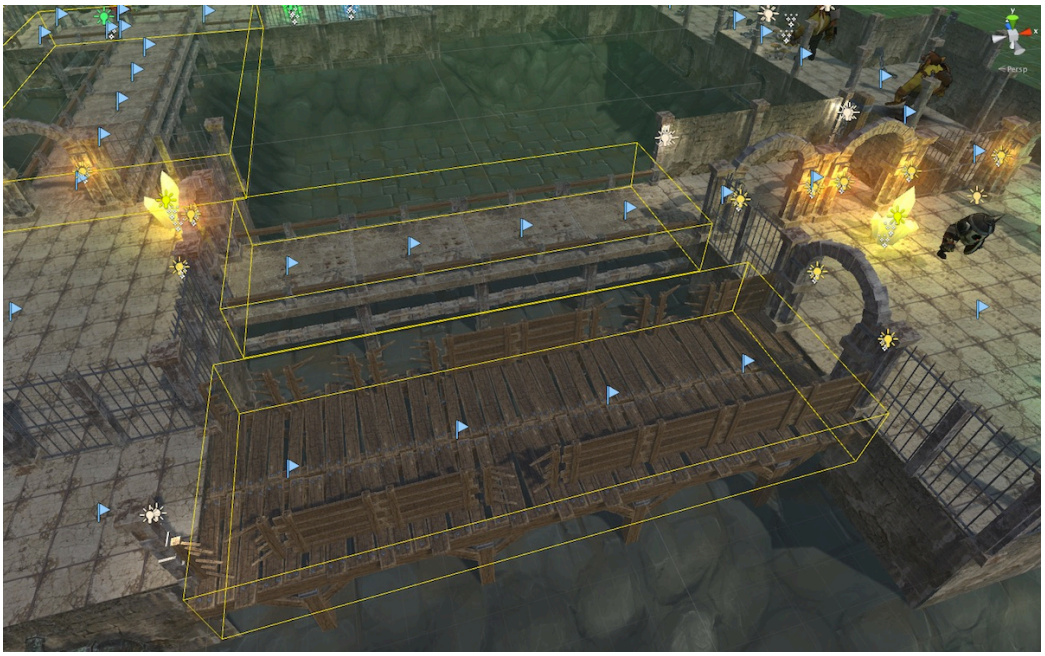


Figure 71: Selective areas overriden by Theme Override Volumes

Select the negation volume and have a look at it's properties

Dungeon: Set the dungeon game object this volume should affect

A button to rebuild the dungeon is provided for convenience. It rebuilds the dungeon in the scene

11 Landscape Transformer

Dungeon architect can also modify the landscape when it builds the dungeon. Starting with an empty terrain, it can modify its height and paint it in intresting ways.

In the above screenshot, a blank terrain was provided as input to the script. It has updated its height (based on a steepness curve provided by the user) and painted the ground, cliffs and pathways with input textures (notice the organic dirt pathway along the layout)

Here's another Example:

Figure 72: Geometry within the volume picks up the theme defined by the volume

Figure 73: Theme Override Volume Properties

Figure 74: Procedural geometry we'd like to remove

11.1 Usage

Navigate to `Assets/DungeonArchitect/Scripts/Dungeon/Landscape`

Attach the script `LandscapeTransformerGrid` to the Dungeon actor

11.1.1 Terrain Setup

Create a new terrain and center it on the dungeon (e.g. set the X and Z to -250).

Also move the terrain down along Y by around -20 (an approx lowest point your dungeon layout might reach). This is needed because the height map doesn't take negative values

Select the terrain and go to settings and set the Control Texture Resolution

Figure 75: Geometry inside the volume removed after a rebuild

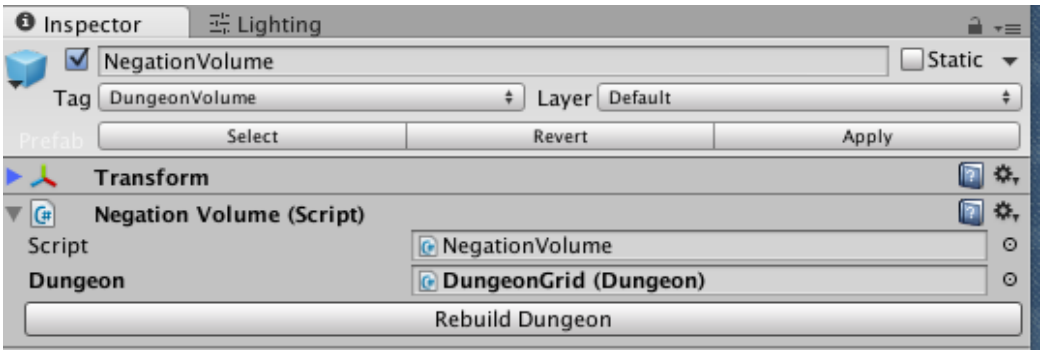


Figure 76: Geometry inside the volume removed after a rebuild

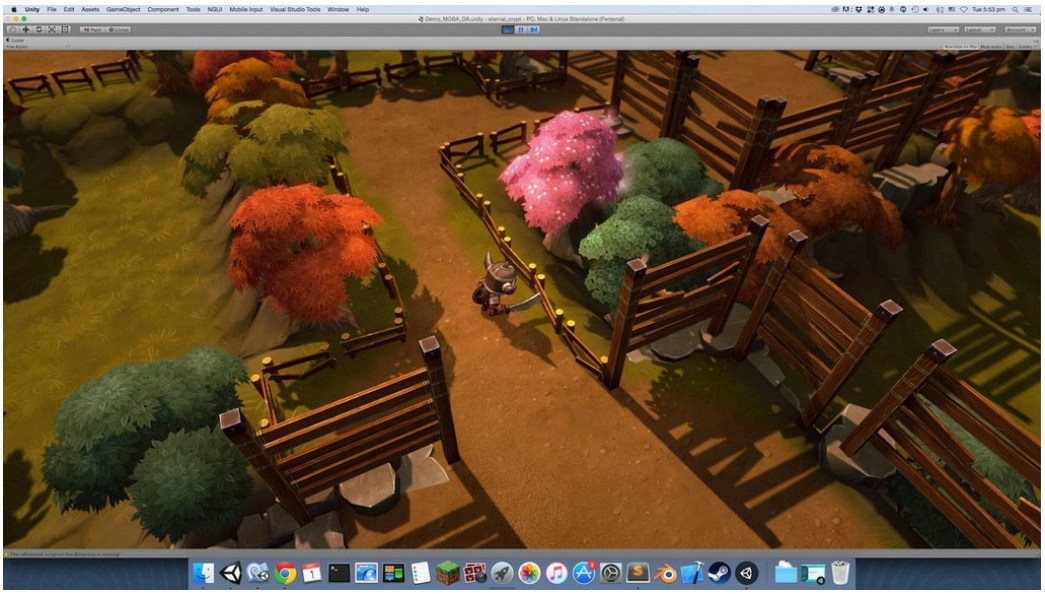


Figure 77: Terrain Transformed along the dungeon layout

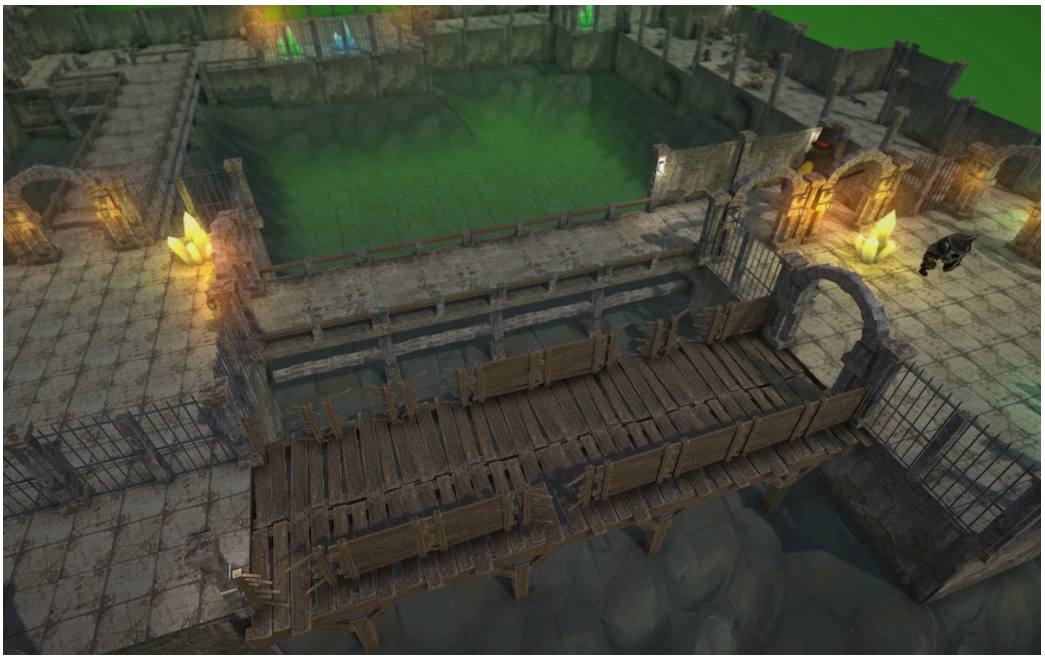


Figure 78: Terrain Transformed along the dungeon layout

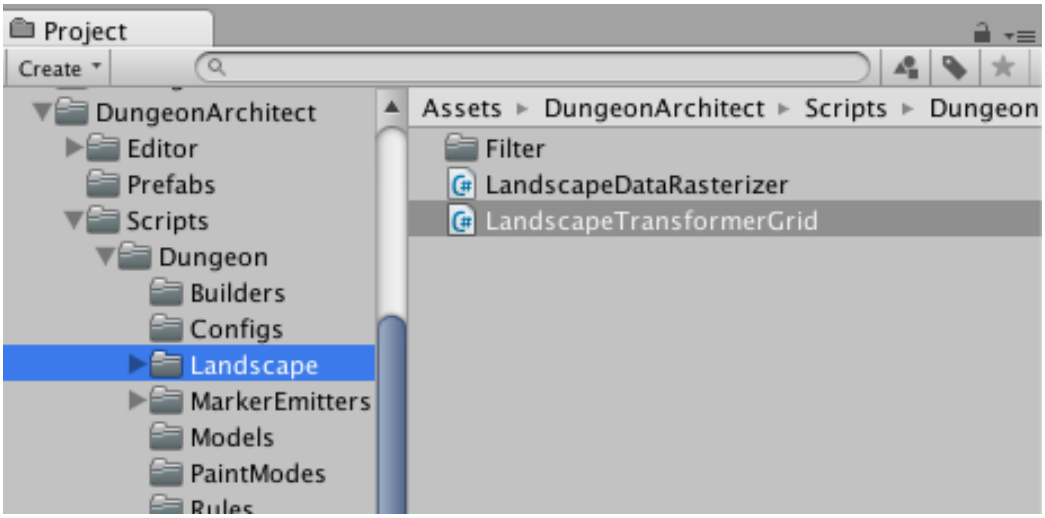


Figure 79: Landscape Transformer script

11.1.2 Properties

Terrain: Assign the this terrain reference to the **Terrain** field in the above script properties

Textures: Assign textures to paint the terrain transformer script and the terrain would be painted based on the texture type

Ground Level Height: Set the default ground level height of the terrain

Layout Level Offset: If set to 0, the terrain would raise up to touch the layout of the dungeon. Sometimes you would like this value to be lower, if you already have a ground mesh like the image below

Room Elevation Curve: The curve defines the steepness of the landscape around the rooms

Corridor Elevation Curve: The curve defines the steepness of the landscape around the corridors

Assign a preset curve if unassigned for the transformer to work properly

Smoothing Distance: The distance to perform the smoothing of the heights using the above curves

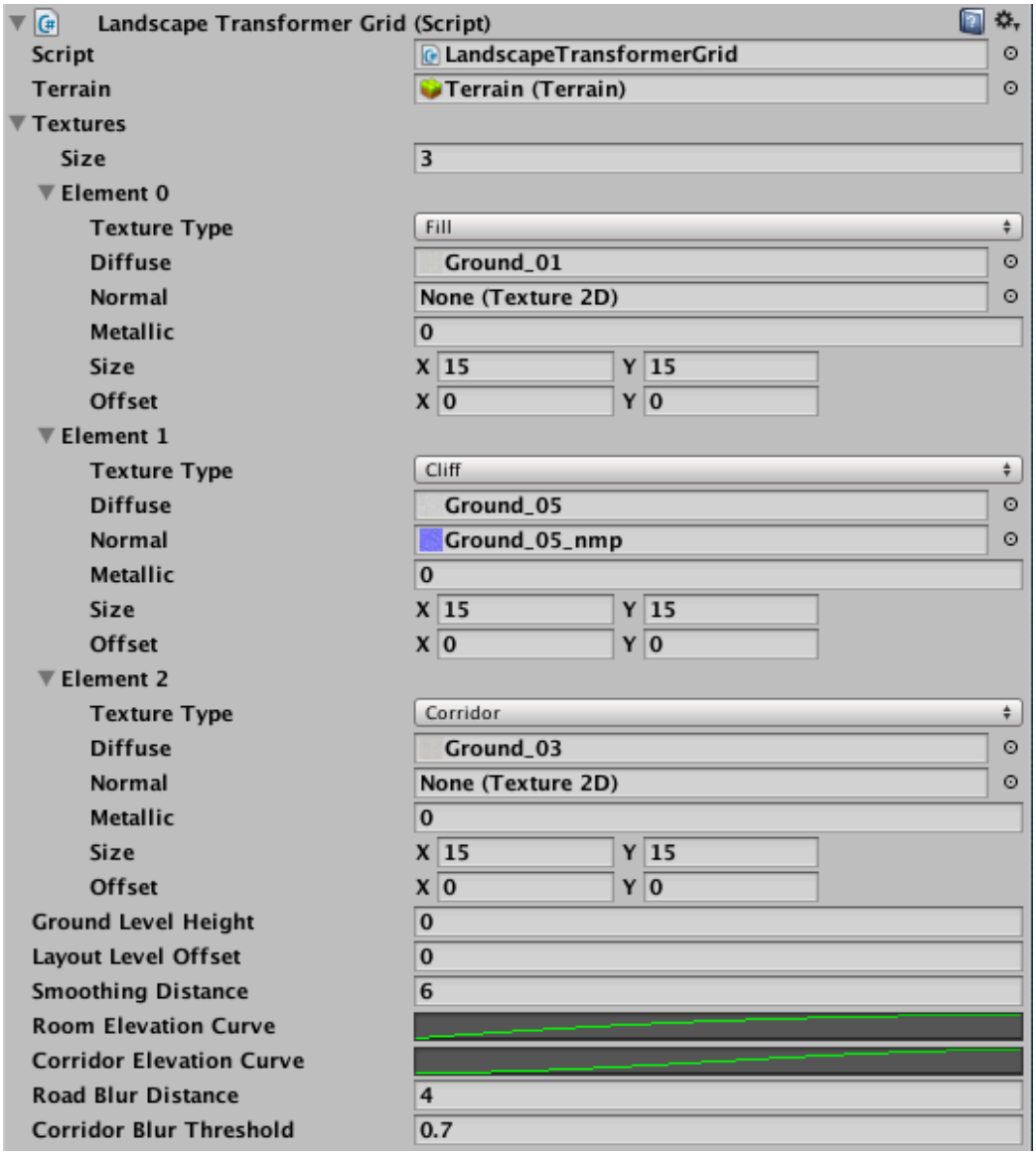


Figure 80: Landscape Transformer Properties

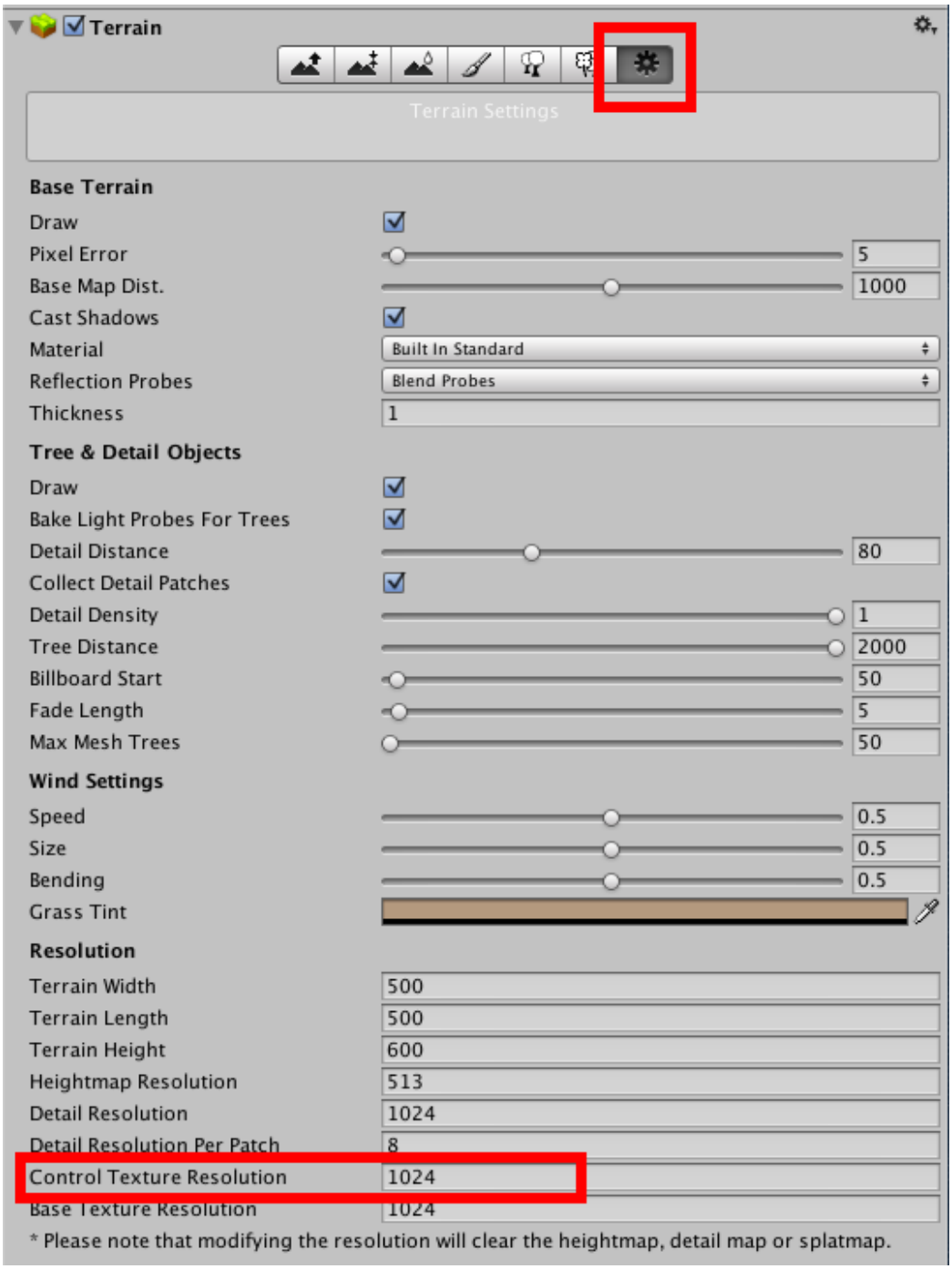


Figure 81: Increase Terrain texture resolution

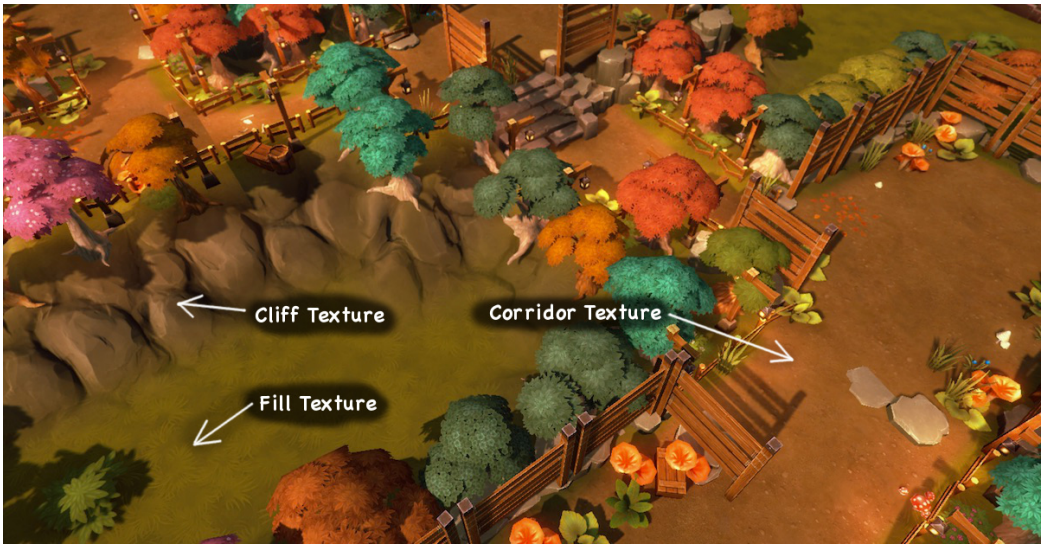


Figure 82: Landscape Transformer Properties

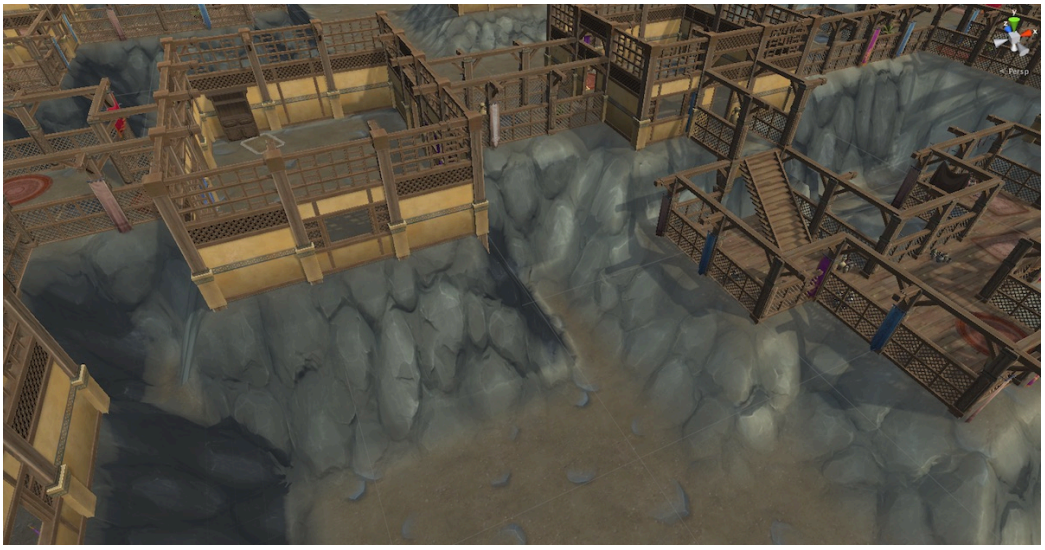


Figure 83: Landscape touches the layout ground

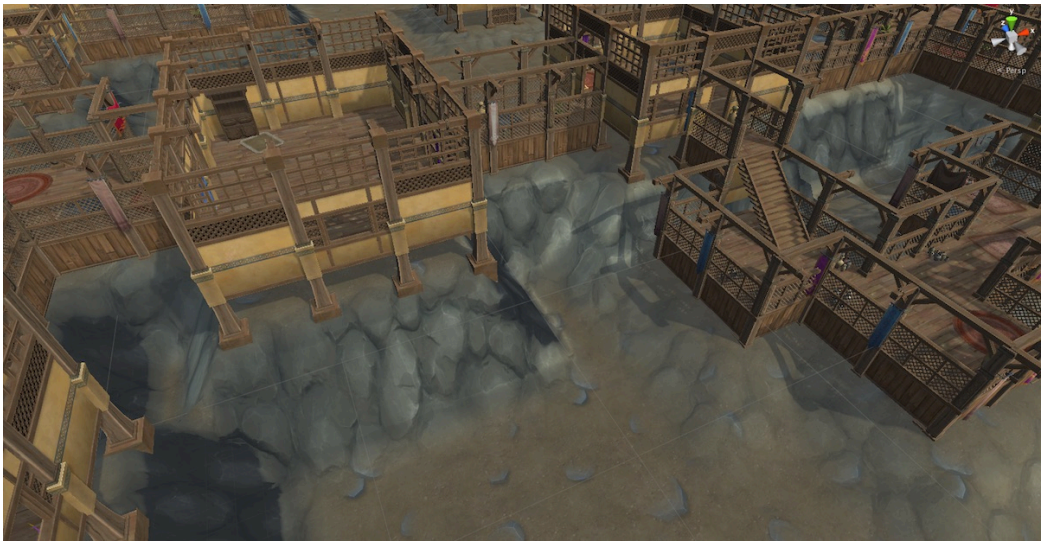


Figure 84: Offset applied to move it down using LayoutLevelOffset

RoomBlurDistance / CorridorBlurDistance: A smooth pathway is painted on the landscape using blurring algorithms. These fields affect how the smooth corridor painting is performed along the rooms and corridors

12 Marker Emitter Scripts

Marker Emitters are behavior scripts that lets you emit your own markers anywhere in the map

As seen previously, *Markers* are emitted by the Dungeon Builder class around the layout of the dungeon (e.g. Wall, Ground, Fence etc) and you can insert actors at that location from the Theme graph. You can even create your own markers emitted off of those parent markers, but without *Marker Emitters* you are restricted to the starting markers the dungeon builder has initially emitted for you

Marker Emitters gives you a lot of flexibility and you can query the dungeon model and emit markers anywhere in the map

A Marker Emitter is invoked right after the Dungeon Builder emits all the markers for the dungeon (Ground, Wall etc)

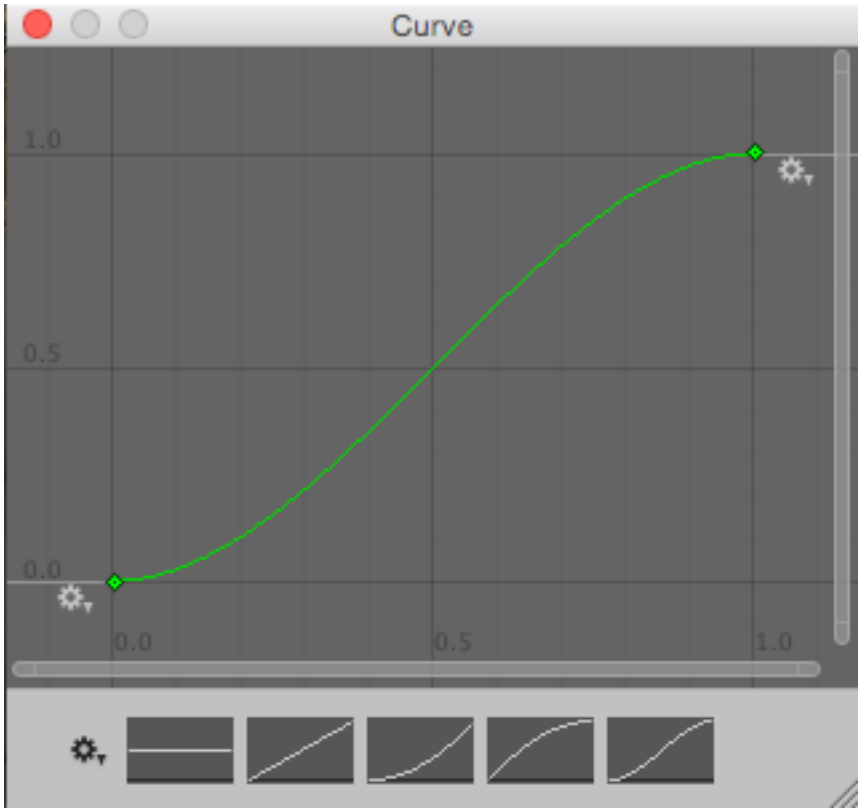


Figure 85: Offset applied to move it down using LayoutLevelOffset

Figure 86: Corridor pathway texture

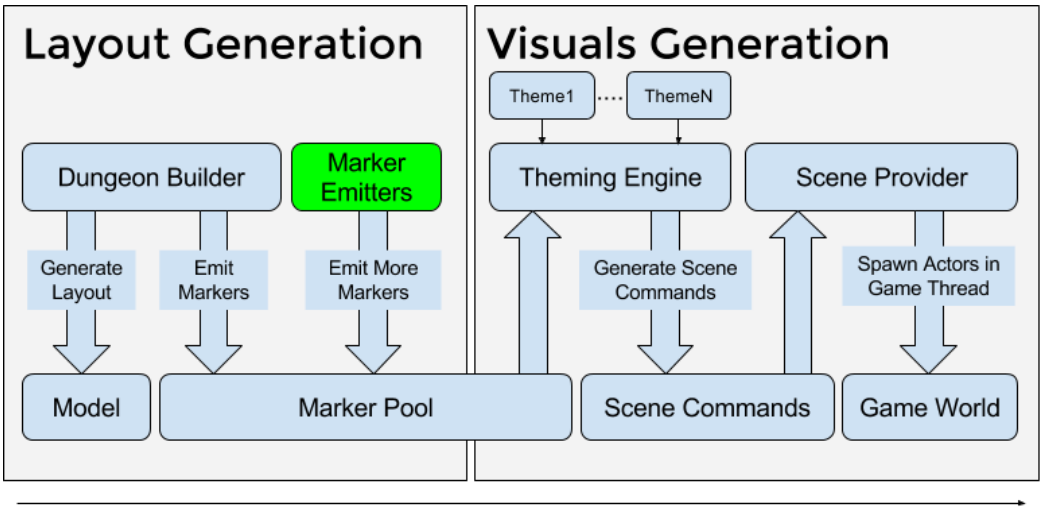


Figure 87: Architecture

12.1 Creating a Marker Emitter

To create a Marker Emitter, you need to create a script inherited from `DungeonMarkerEmitter` under the `DungeonArchitect` namespace

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

public class MyAwesomeEmitter : DungeonMarkerEmitter
{
    public override void EmitMarkers(DungeonBuilder builder)
    {
        base.EmitMarkers(builder);

        // Your emitter logic here

        // Emit as many markers as you like.
        // Emit a marker like this:
```

```
        // Fill up the marker data
        var markerName = "MyMarker";    // This name will be picked up in your theme fi
        var transform = Matrix4x4.TRS(position, rotation, scale);

        // Additionaly specify the grid based meta data (optional)
        var gridPosition = new IntVector(); // specify a position value here in grid co
        var cellId = -1;

        builder.EmitMarker(markerName, transform, gridPosition, cellId);
    }
}
```

To attach a marker emitter to your Dungeon game object, simply add this script to the game object

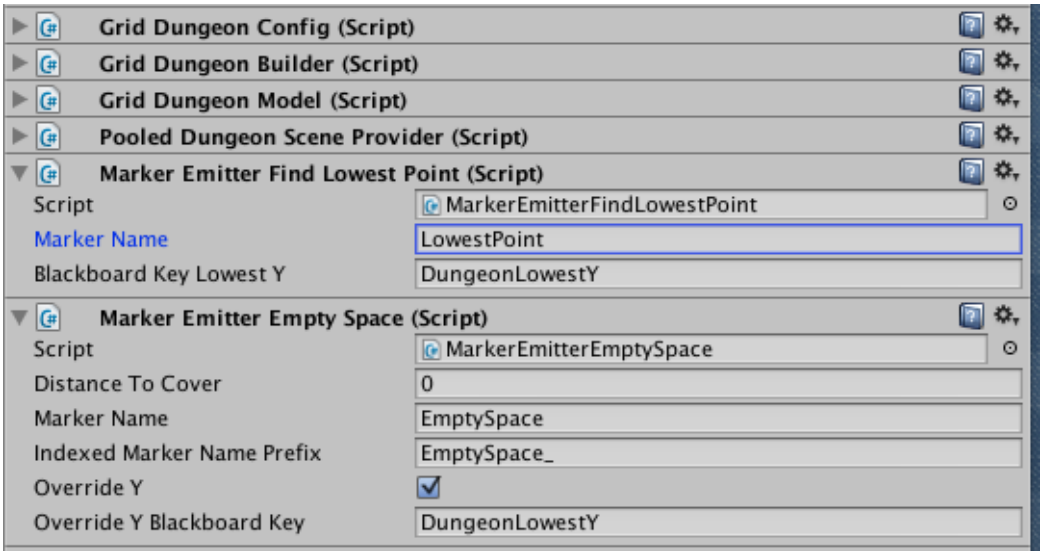


Figure 88: Marker Emitters attached to the Dungeon Game Object

Explore the existing marker emitters that come with Dungeon Architect under `Assets/DungeonArchitect/Scripts/MarkerEmitters/Grid`

Figure 89: Trees spawned outside the dungeon layout

12.2 Example #1

In this example, we'd like to decorate the area outside the dungeon layout that Dungeon Architect has created for us. Since DA creates marker points within the layout, we create a marker emitter to find nearby points in the empty space and emit markers named "EmptySpace_N" where N is 1, 2, 3, 4, 5 etc. These markers are then available to us in the thememe file

12.3 Example #2

Sometimes, it's useful to find the lowest point of the dungeon, so a large plane can be placed there (e.g. water plane, lava plane etc)

The `MarkerEmitterFindLowestPoint` lets you do just that

This emitter emits a marker named `LowestPoint`, at the lowest Y point of the dungeon with the appropriate scale, which we can decorate with any object in the theme file

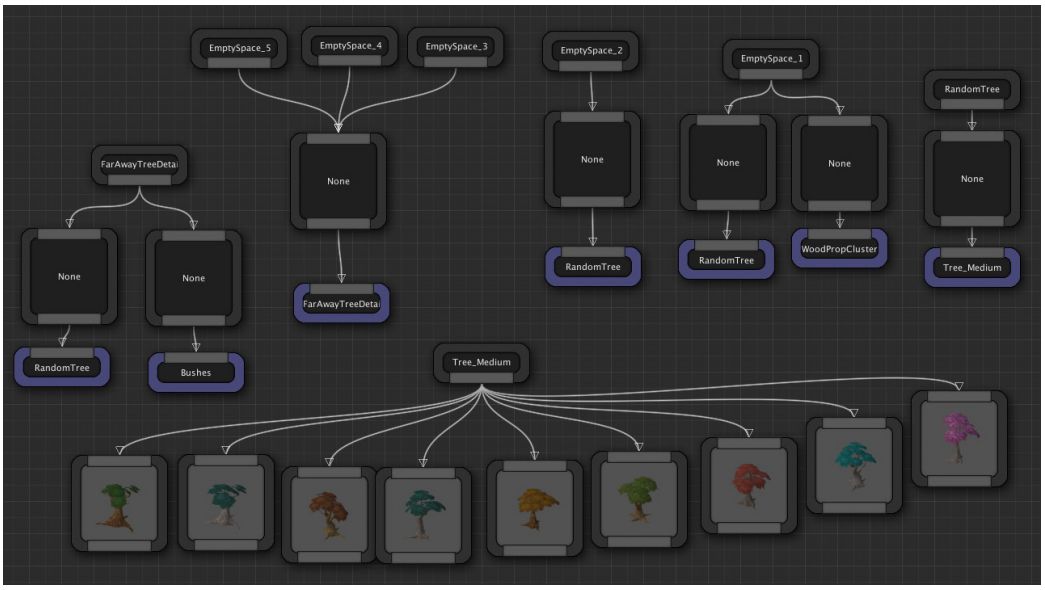


Figure 90: Theme to decorate the markers created by the emitter script

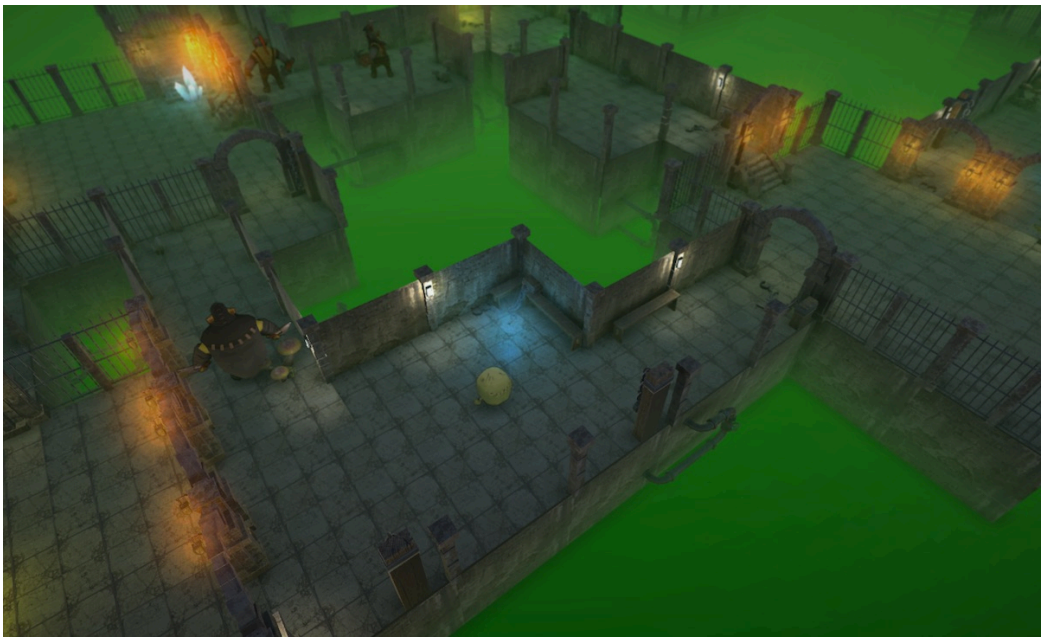


Figure 91: An acid plane created at the bottom of the dungeon

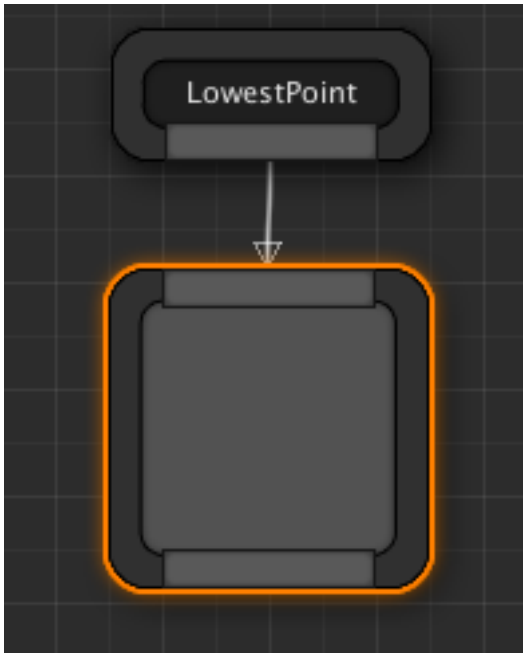


Figure 92: Node to attach a plane mesh to this marker

12.4 Example #3

A marker emitter is created to add extra contextual markers to beautify a 2D level layout (can also be used with 3D)

Check the marker named `MarkerEmitterCornerBeautifier`

All the `Corner_*` markers seen in the theme file above were emitted by the marker emitter script

13 Navigation Mesh

Dungeon Architect supports runtime navigation mesh generation, which is not supported in Unity 5 yet. This is necessary for moving your NPCs intelligently across your procedurally generated level

This is based on the Recast Navigation library, so it provides very high quality results.

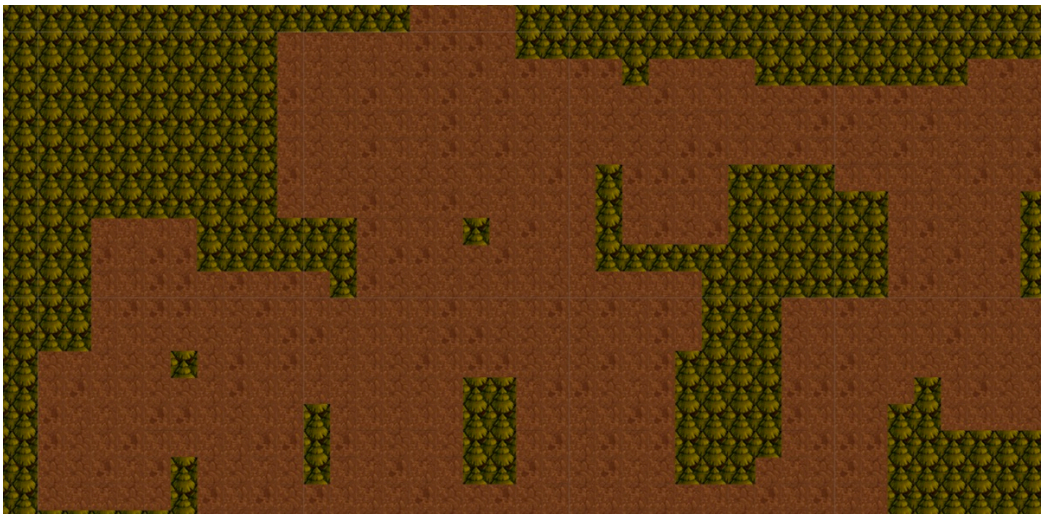


Figure 93: Simple layout with a red ground sprite

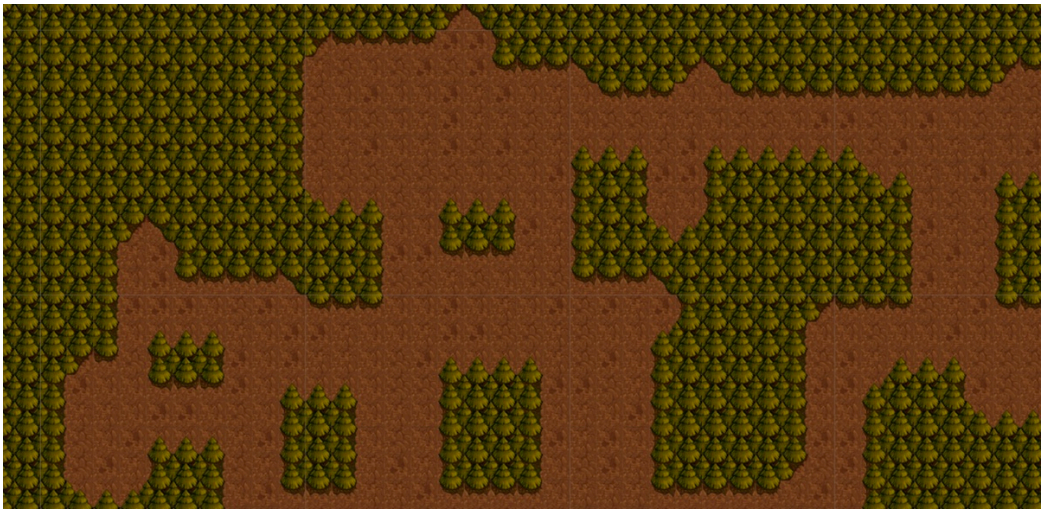


Figure 94: Added decorated sprites with spatial contextual markers

Figure 95: Theme to add the decorative sprites

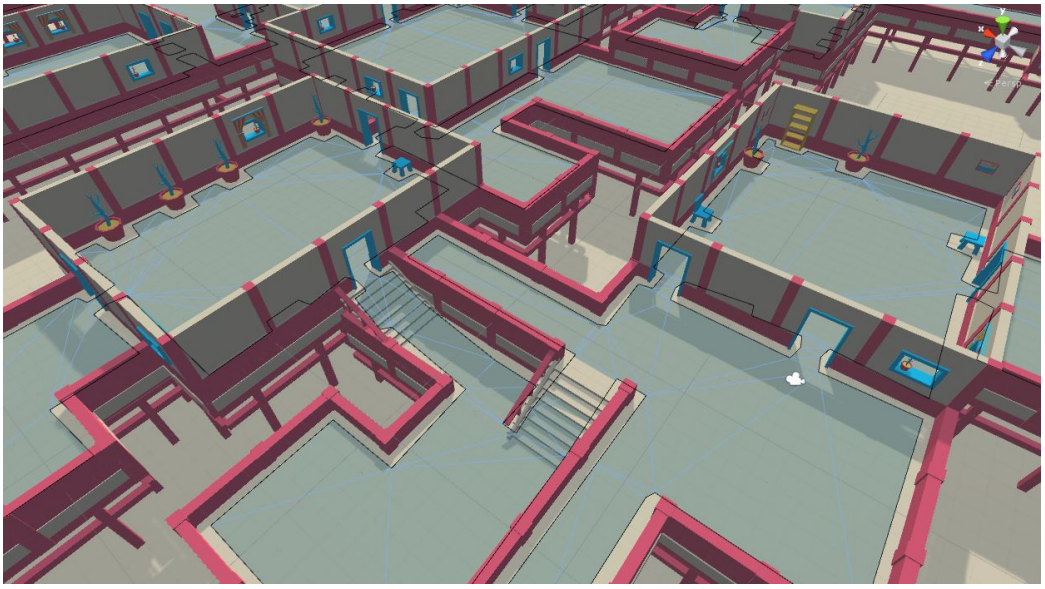


Figure 96: Navigation

13.1 Setup

To build a navigation mesh during runtime, place the DungeonNavigation prefab on to your scene

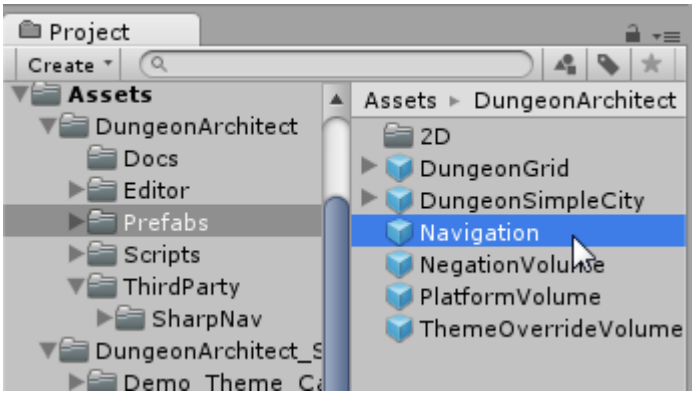


Figure 97: Navigation Prefab

Select the game object you just placed and have a look at the inspector window

To build the nav mesh, input geometry for walkable and blocked areas needs to be provided. This is done by `Triangle Providers`

There are two types of triangle providers already attached to the navigation object

- **Collision Triagle Provider**: Uses the collision mesh of the colliders present in the scene to build the navigation mesh
- **Layout Floor Triangle Provider**: This provides the floor layout of a dungeon as walkable area to the navigation mesh input

Assign the dungeon reference you would like to use

Click Build

13.2 Theme Influence

The dungeon objects (meshes, prefabs etc) do not affect the navigation by default. You need to set the **Affects Navigation** flag of the visual node to make it affect the navigation.

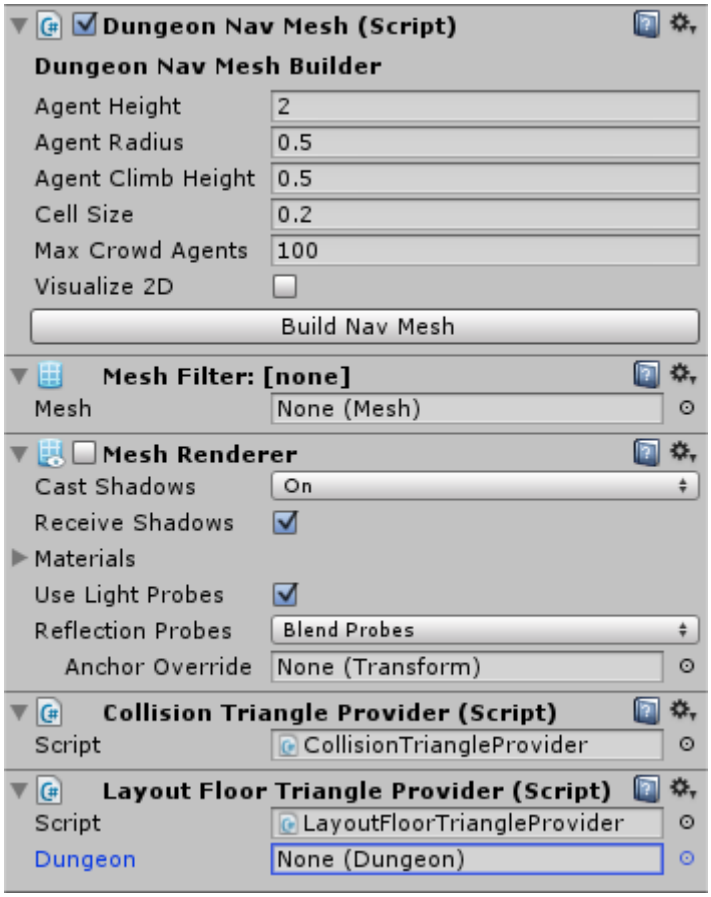


Figure 98: Navigation Properties

Figure 99: Navigation Triangle Provider

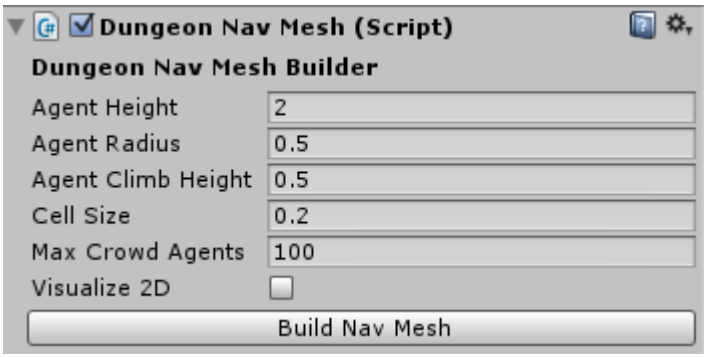


Figure 100: Navigation Properties

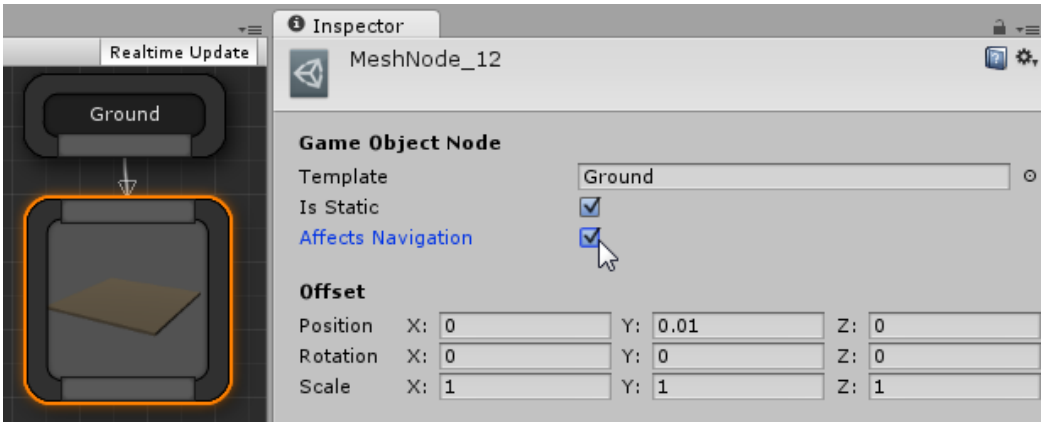


Figure 101: Influence Navigation from Theme Graph

The object also has to be static in order to affect the navigation. So the **Is Static** flag also needs to be set.

Important: You should set the *Affects Navigation* flag only when it is absolutly required to maintain a good nav mesh generation speed while building

13.3 Config

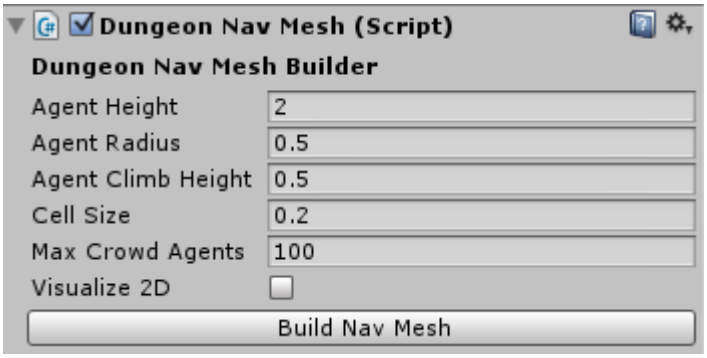


Figure 102: Navigation Generation Config

- **Cell Size**: Controls the mesh generation accuracy vs speed. It determines the resolution of the generated nav mesh. Lowering this number will generate nicer edges and a more accurate navmesh but requires more procesing power and slower. A good value is between 0.2 to 0.3
- **Agent Height**: The max height of the agents in your game
- **Agent Radius**: The max radius of the agents in your game
- **Agent Climb Height**: The max height an agent can climb on. Objects with height lower than this will not be considered obstacles, as the agent would be able to climb over them
- **Max Crowd Agents**: The max no. of agents that can be present in the game at a time.

13.4 Triangle Providers

To generate a navigation mesh, the nav mesh system requires input triangles so it can build a world and generate the nav mesh. Dungeon Architect comes with various tri-

angle providers to help you affect the nav mesh in various ways. We have seen two of them already above

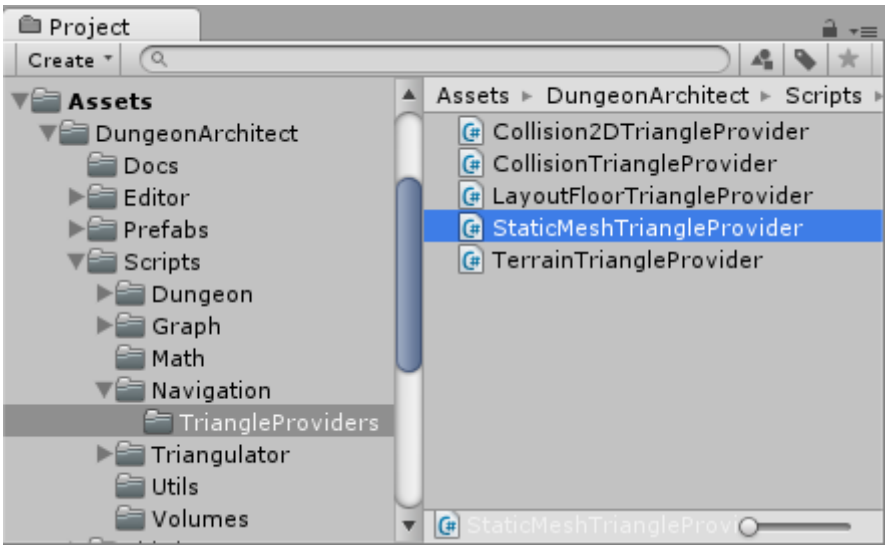


Figure 103: Navigation Mesh Triangle Providers

- **StaticMeshTriangleProvider**: Uses the mesh geometry (vertices, indices) for all the meshes defined in the prefab as a source for the input triangles to generate the nav mesh. This is usually slower but creates detailed results
- **CollisionTriangleProvider**: Uses the colliders defined in the prefab as a source for the input triangles to generate the nav mesh. This is much faster than the StaticMesh triangle provider as it works with the low poly collision geoemtry. However it requires a collider to be present in the prefab to work
- **TerrainTriangleProvider**: Feeds the terrain geometry into the nav mesh generation system. So you can have your dynamic navmesh build around terrains
- **LayoutFloorTriangleProvider**: Feeds the layout of a dungeon to the nav mesh generation system. This is usually faster than providing a ground mesh in the CollisionTriangleProvider

You can use multiple triangle providers at once. You can use only the CollisionTriangleProvider for better performance. However, you can increase the performance even further by disabling the gound mesh from affecting your navigation (because there will be lots of ground meshes) and providing that ground data from an additional LayoutFloorTriangleProvider script.

13.5 Navigation Agent

Use the NavAgent to move your NPCs in the dynamic navigation mesh

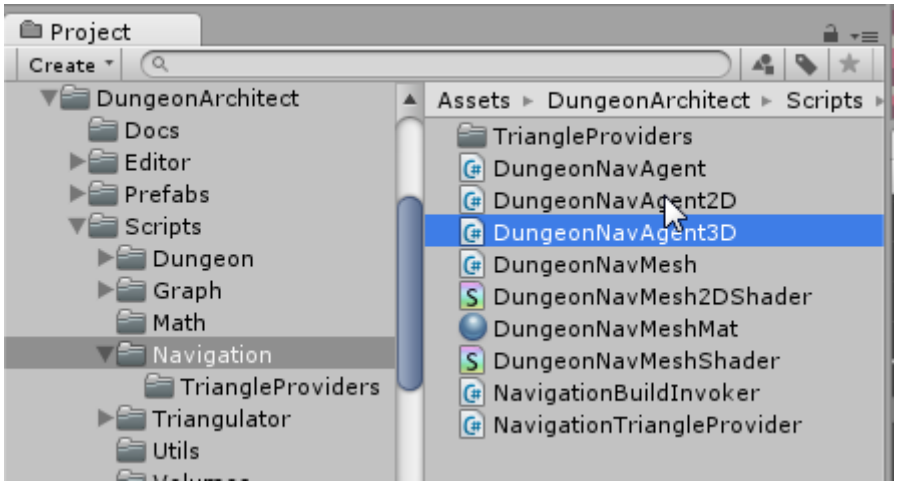


Figure 104: Navigation Mesh Triangle Providers

An fully working example of this with AI is provided in the SurvivalShooter demo game bundled with Dungeon Architect

The nav mesh agent requies a CharacterController script to be present in the game object

```
var agent = GetComponent<DungeonNavAgent>();

// Move the agent to the target position
agent.Destination = targetPosition;

var velocity = agent.Velocity;  // Agents velocity
var direction = agent.Direction;    // Agents movement direction
var distanceToDestination = agent.GetRemainingDistance();

// Stop moving the agent
agent.Stop();

// Resume movement
agent.Resume();
```

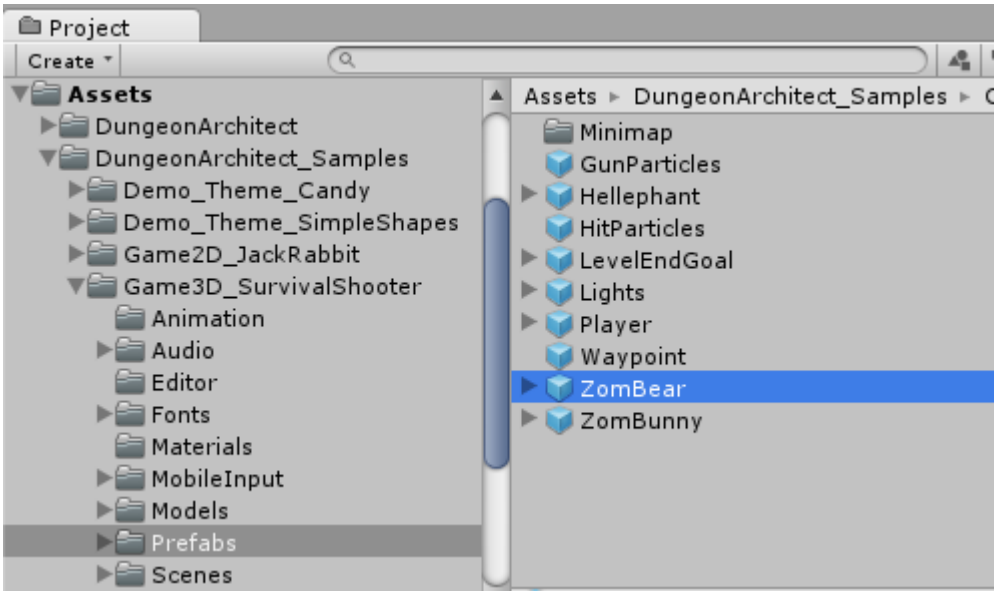


Figure 105: Sample NPCs with Navigation Agent behaviors

Have a look at the shooter game demo in the samples folder for a complete example

14 2D Support

Dungeon Architect fully supports 2D. You can use the same workflow to create beautiful 2D levels.

A sample 2D game comes along with Dungeon Architect to help you get started. It demos dynamic 2D procedural level generation, dynamic 2D navigation mesh generation, 2D AI with patrol, seek and search behaviours

2D dynamic navigation mesh generation is also support.

15 Dungeon Builders

The Default dungeon builder used to create the layout is swappable and you can provide your own implementation

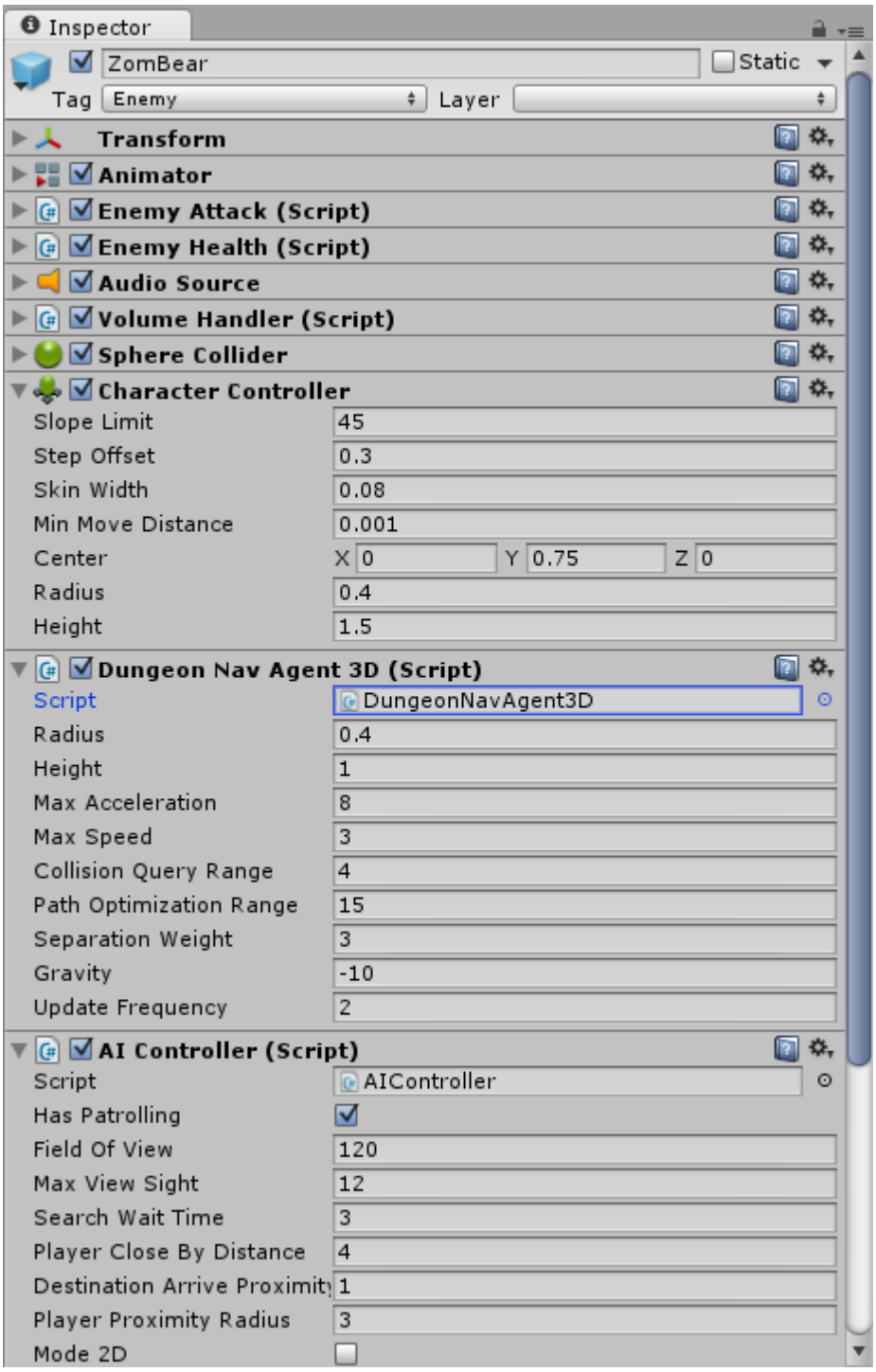


Figure 106: Sample NPCs with Navigation Agent behaviors

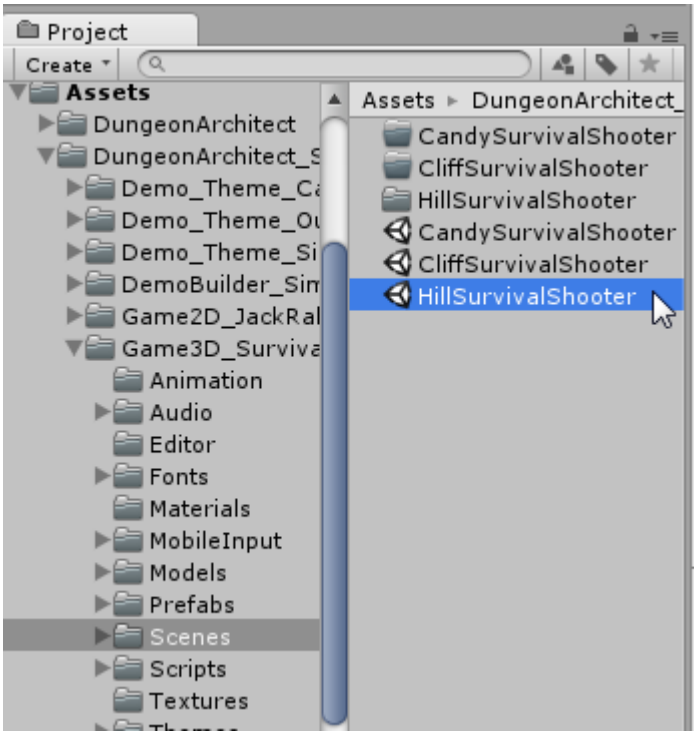


Figure 107: Shooter Game Demo in the Samples folder

Figure 108: Shooter Game Demo

Figure 109: Shooter Game Demo

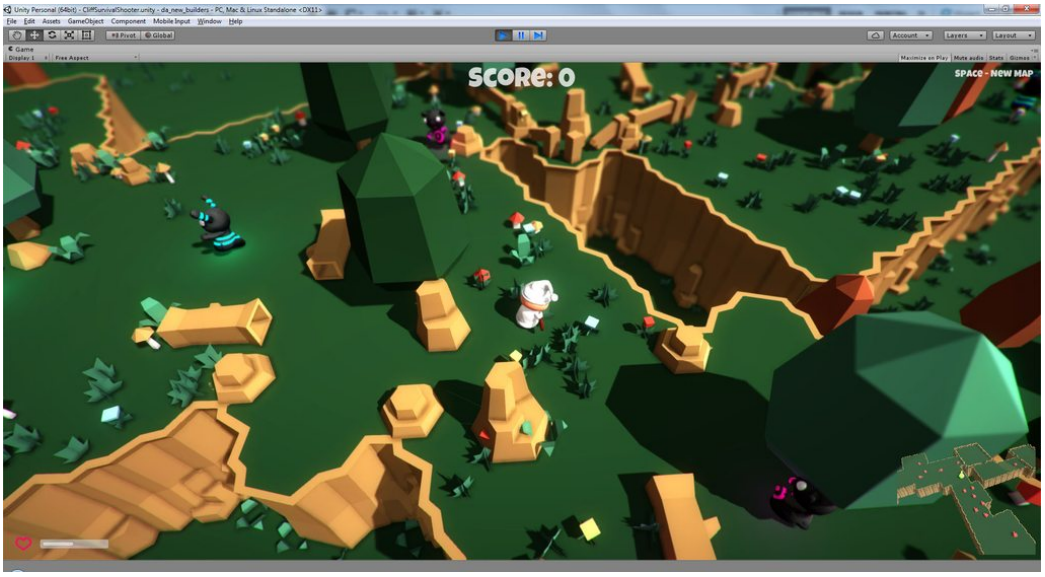


Figure 110: Shooter Game Demo

Figure 111: 2D Demo game

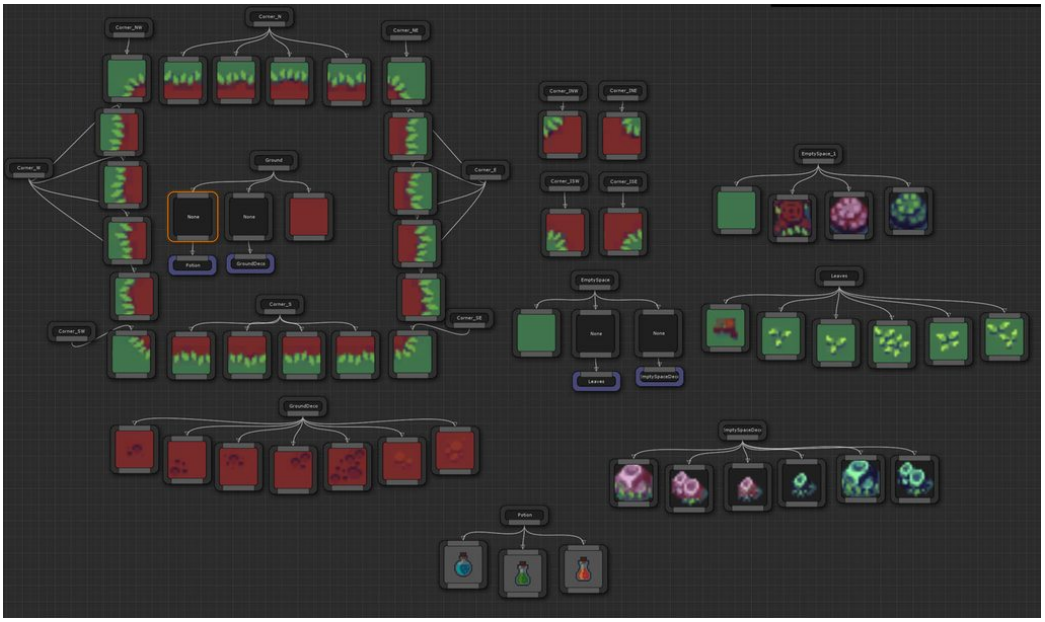


Figure 112: Theme with 2D Sprites

This is useful if you want to use your own algorithm for generating the layout of your dungeons.

You are not limited to a grid based system.

15.1 Creating a new Builder

To create a new builder, subclass `DungeonBuilder` under the `DungeonArchitect` namespace and implement the virtual methods

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using DungeonArchitect.Utils;

[ExecuteInEditMode]
public class MyDungeonBuilder : DungeonBuilder
```

Figure 113: 2D Demo game

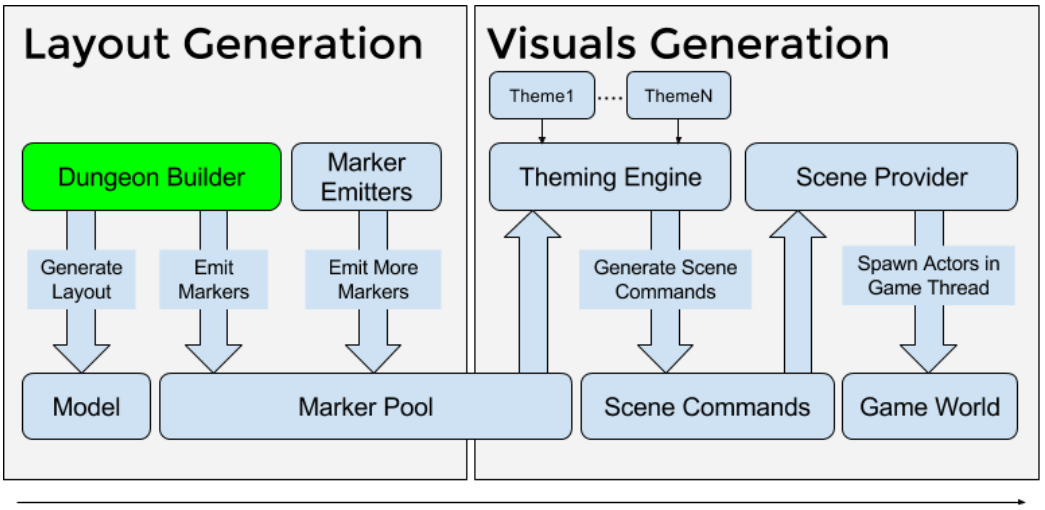


Figure 114: The Dungeon Builder can be swapped with your own implementation

```
{

    public override void BuildDungeon(DungeonConfig config, DungeonModel model) {
        base.BuildDungeon(config, model);

        // Add your builder logic here
    }


    public override void EmitMarkers() {
        base.EmitMarkers();

        // Emit markers here by calling EmitMarker()
    }
}
```

Have a look at `GridDungeonBuilder` under `Assets/DungeonArchitect/Scripts/Builders/GridDungeonB` for reference

15.2 Using a different Builder

If you've created a builder and would like to use it with your dungeon actor, drop in an existing dungeon actor, remove the existing builder script and replace it with your own

15.3 Example Builders

Dungeon Architect comes with a sample builder named SimpleCity. It could be used as a good reference for building your own builders

There are also examples on how this sample builder can be further extended by the users using Marker Emitters script. It is used emit markers around the boundary of the city, so theme files can decorate them as strongholds

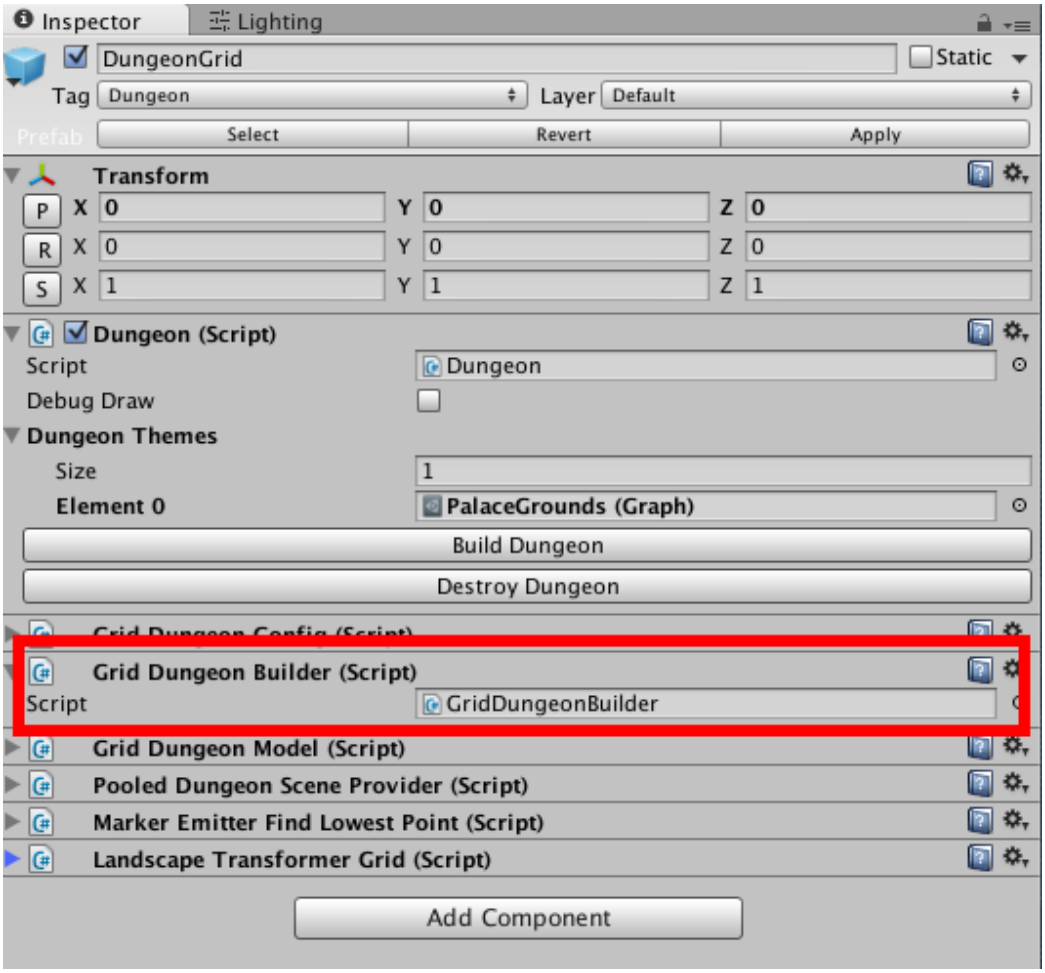


Figure 115: Dungeon builder script attached to the Dungeon game object

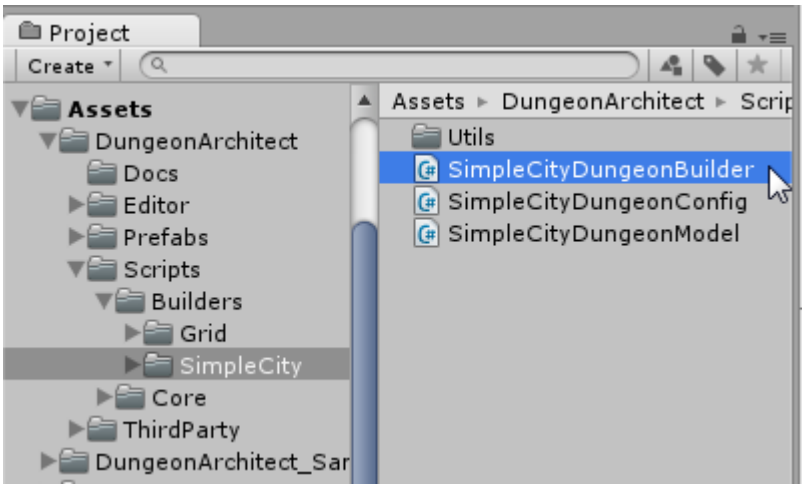


Figure 116: Builder Code Location

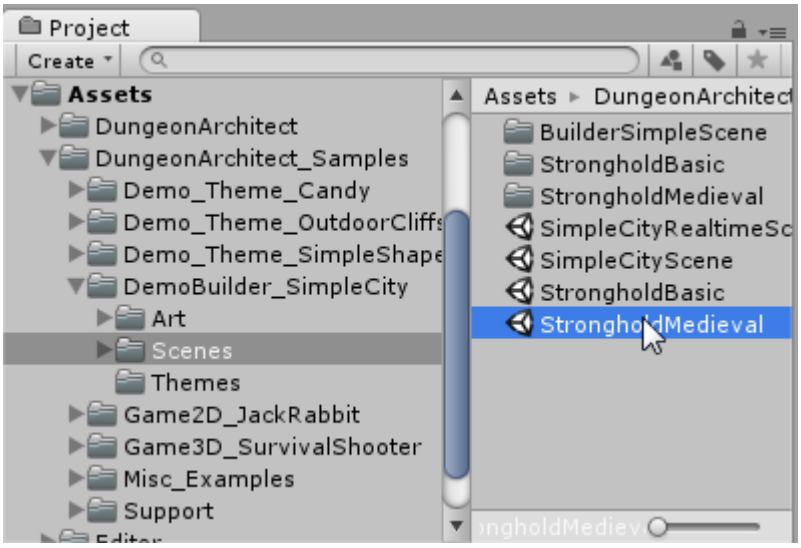


Figure 117: Builder Samples Location

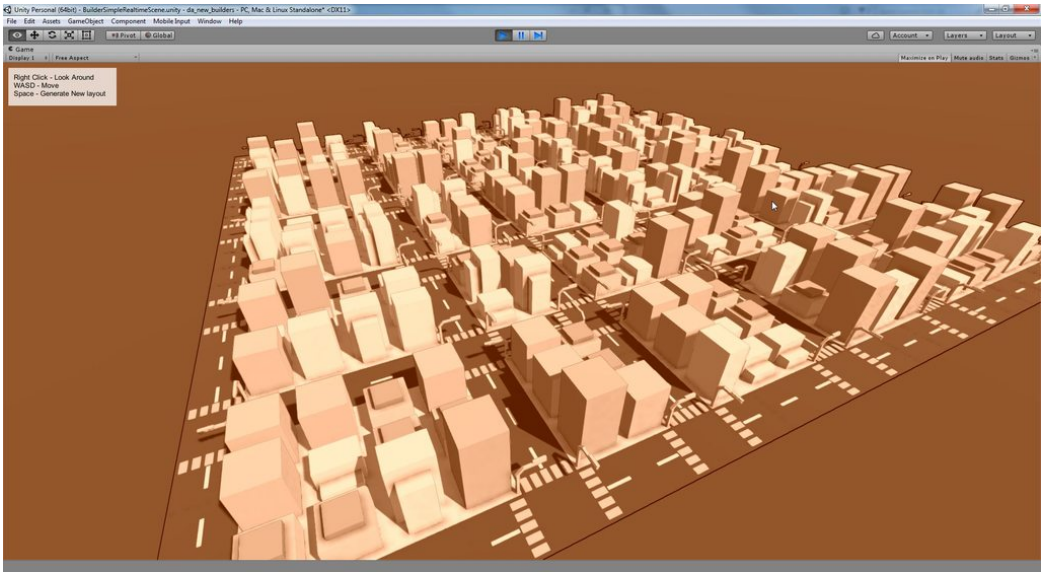


Figure 118: Sample City Builder

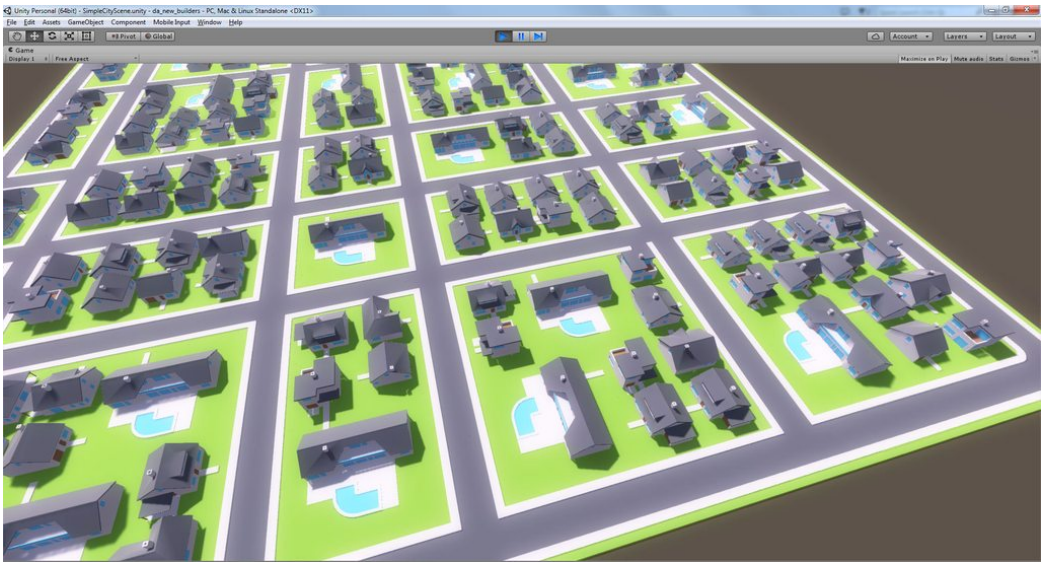


Figure 119: Sample City Builder

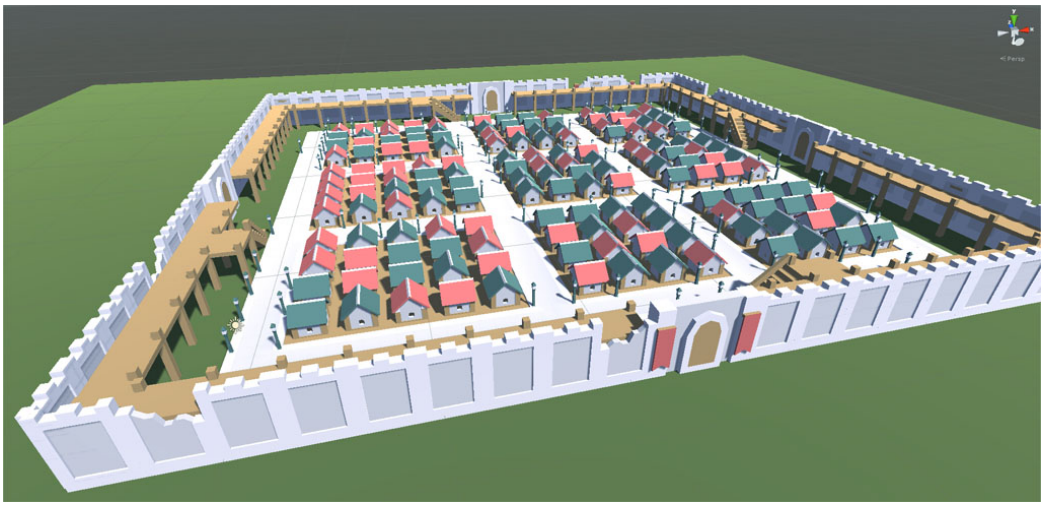


Figure 120: Stronghold Wall Emitter

Figure 121: Stronghold Wall Emitter