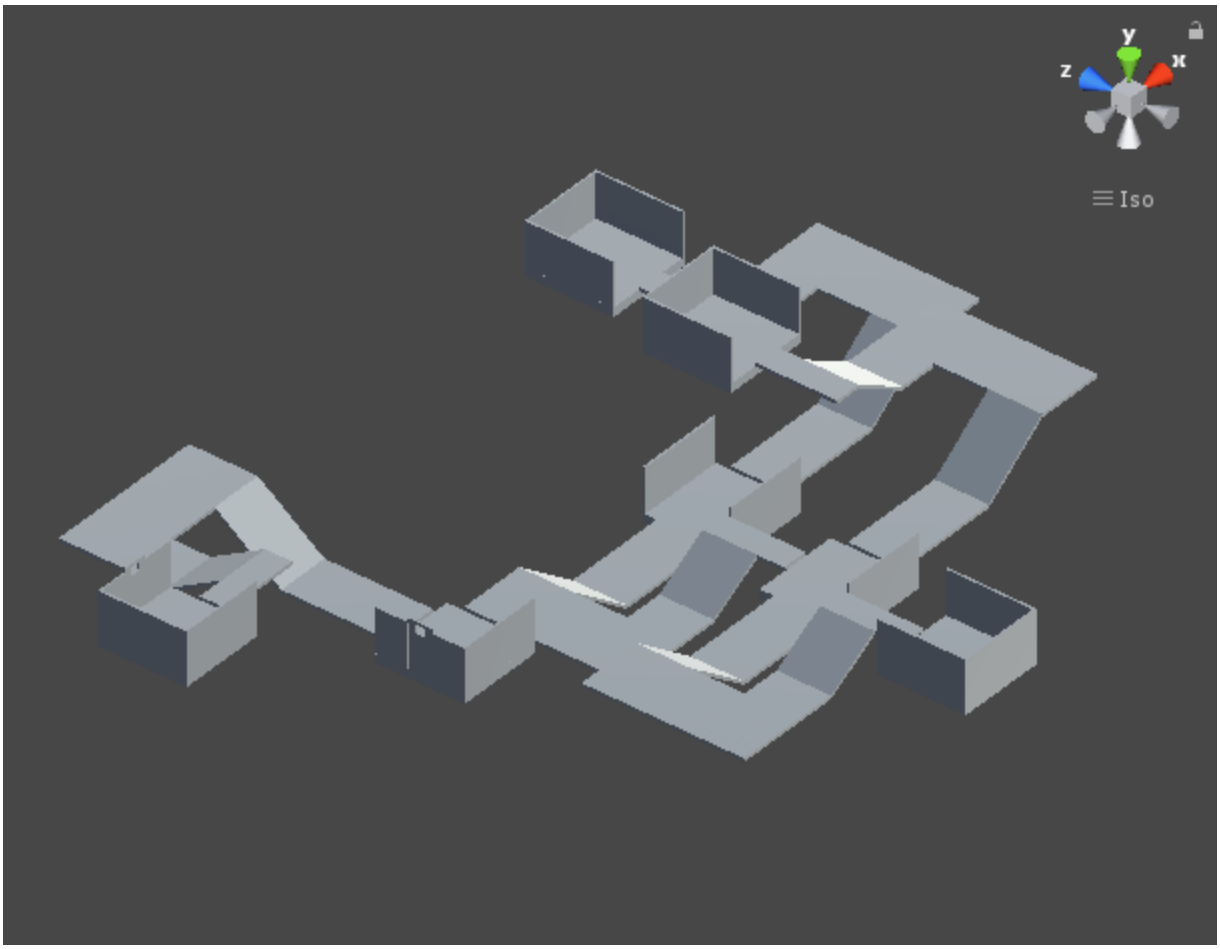
PROCEDURAL TOOLKIT

A procedural generation toolkit for sprites and 3D meshes in Unity3D.



By: Flamboyance - A Unity3D Asset Store Developer

Contents

Introduction: What is Procedural Toolkit?

Procedural Toolkit is a procedural generation toolkit/randomizer that will aid you in the construction of your next procedurally generated game. The product aims to make customizing modular procedural assets as easy as possible.

This package allows you to turn a set of assets (modelled by yourself, or bought) into small instances/cells/rooms/parts that will be randomized based upon the settings you provide. In addition the toolkit has been made to be extremely flexible so that you do not need to design and set the objects up in Unity, you may template the objects in your own modelling software (where you have access to hierarchies and parenting) – Procedural Toolkit makes use of a strict naming convention to allow the toolkit to process and setup the necessary data.

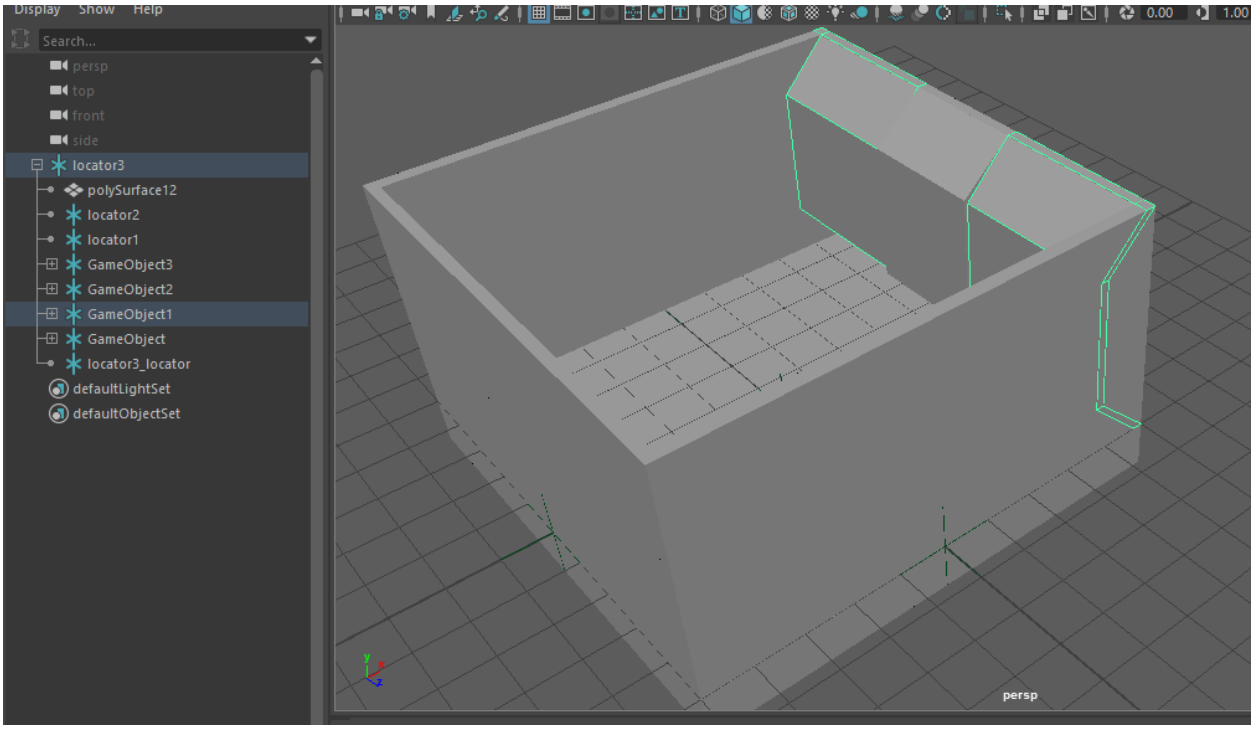


Fig 1. An example of an asset designed in Autodesk Maya – ready for Procedural Toolkit to process into a modular cell.

Beginner Tutorial: Get Started with the Procedural Toolkit:

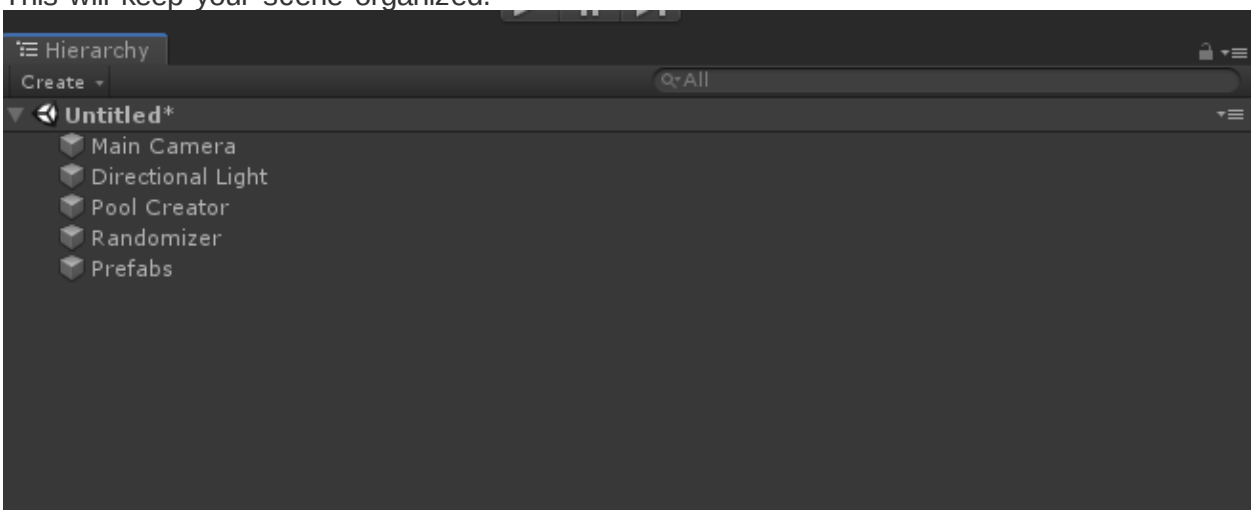
Custom procedurally generated objects

Step 1:

Create a new scene, name it "Random Generator".
Create 3 empty Game Objects
Name them as follows:

- Pool Creator
- Randomizer
- Prefabs

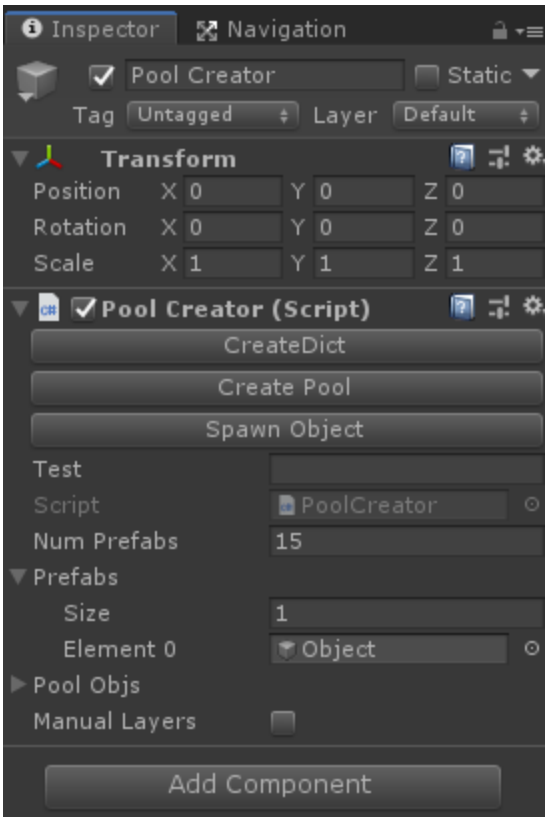
This will keep your scene organized.



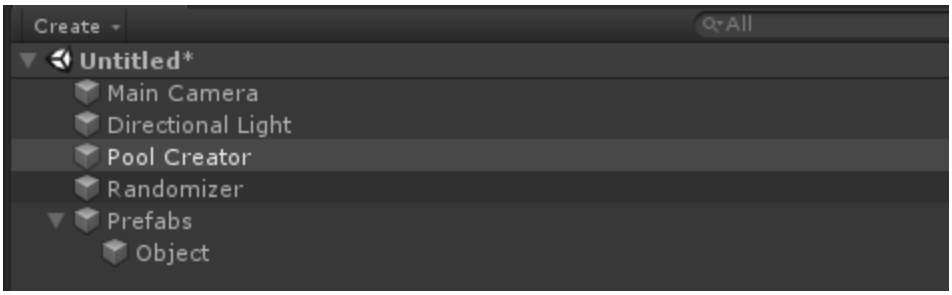
Step 2:

Add a Pool Creator script to the Pool Creator object
Add a Randomizer script to the randomizer object

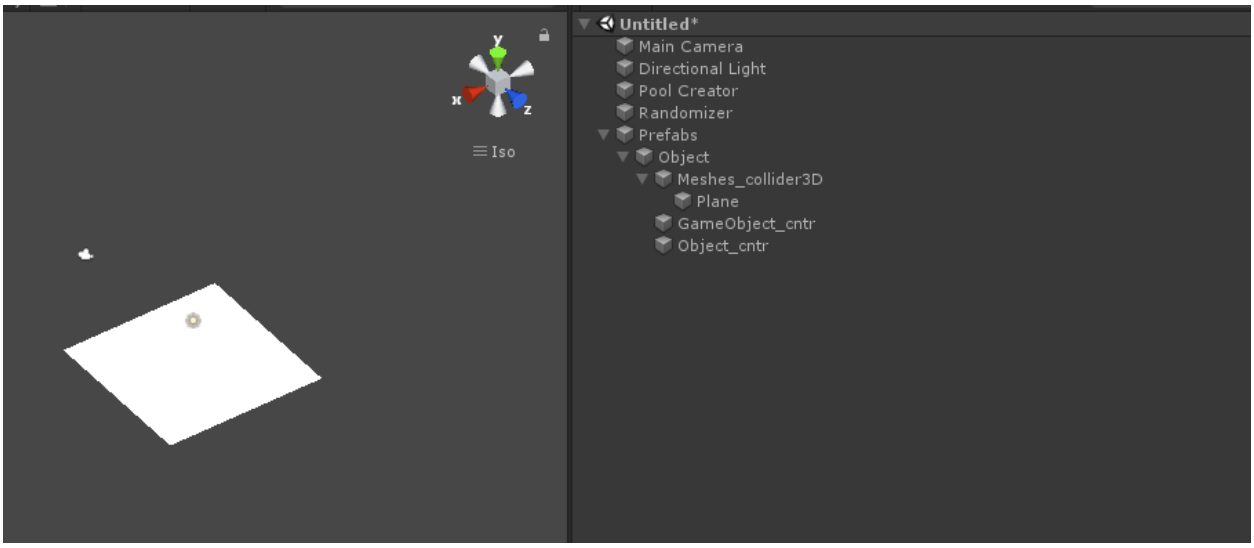
Under Prefabs add another GameObject – rename this to “Object”.
In your object you can add the meshes and details that you want to randomize. Add it to the list of prefabs on the pool creator



[Grab your reader’s attention with a great quote from the document or use this space to emphasize a key point. To place this text box anywhere on the page, just drag it.]

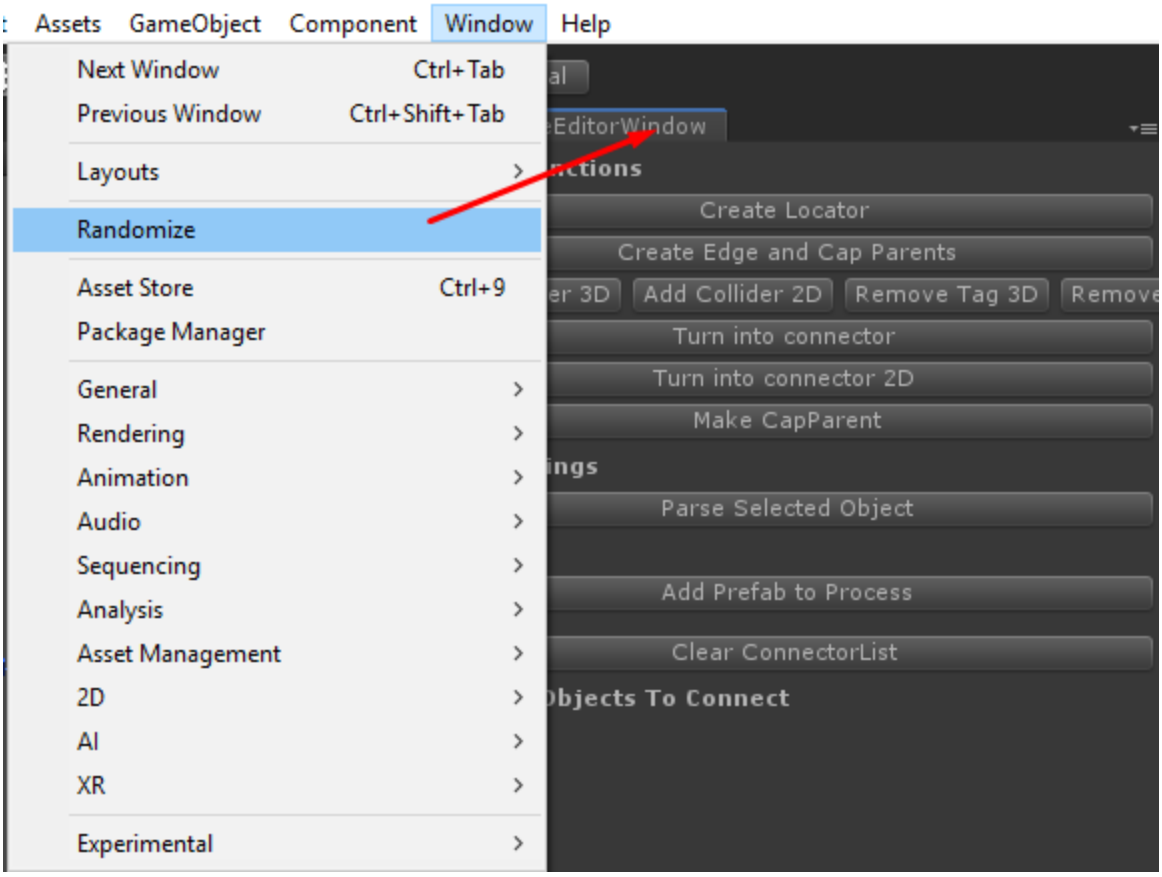


For tutorial purposes I am using the default plane from unity



Step 3:

Open the Randomize Editor Window -> Window Randomize



Colliders and Connectors

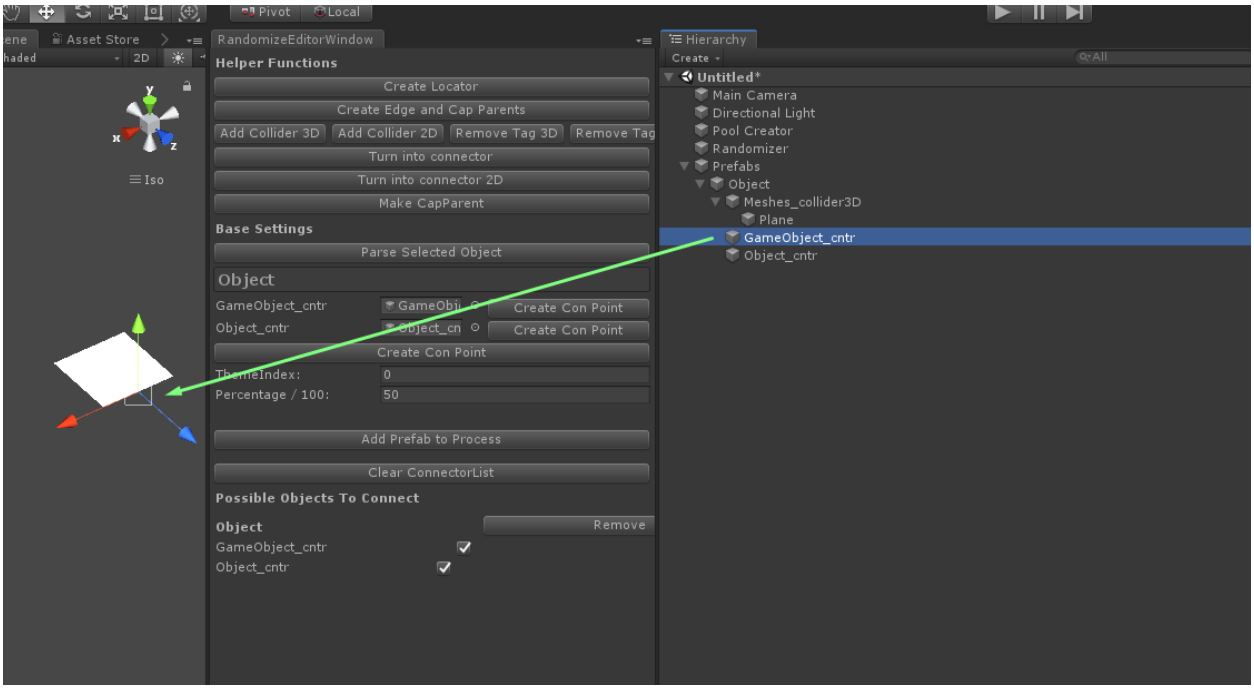
Create a box around your mesh that you want to check collisions on and add _collider3D to the box.
*Alternatively on your mesh, add a _collider3D tag on the topmost parent

For each connector location you want to add, place a GameObject with the Z axis facing outwards (blue axis). Add a _cntr tag to the GameObject *These gameobjects should have unique names so that they can be found by the Pool Creator.

For 2D objects, instead of 3D, add 2D instead (_collider3D/_cntr2D)

**Shortcuts are available in the randomize editor window to add these tags quickly to selections.

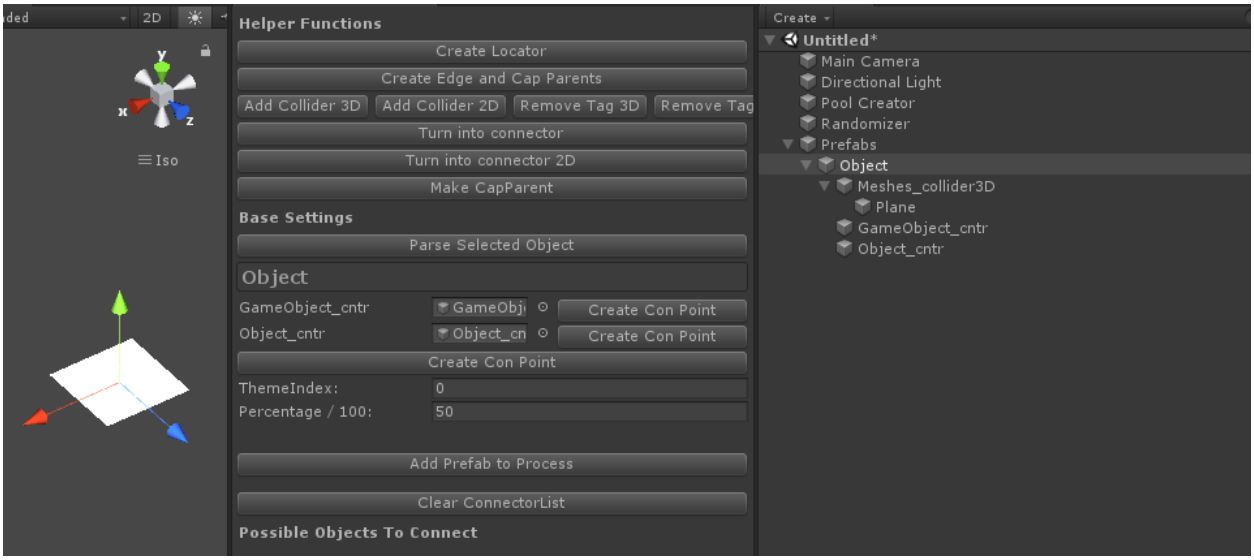
**Note in the troubleshooting section, there is some info on how to create vertical or stacked connectors.
Vertical connector points in the troubleshooting section explains how to use the system further



Add as many connectors as you want to randomize on this object.

Step 4:

Lets add some meta data! Select the top level of the object and click on the “Parse Selected Object” button. You will see the connectors have been populated in the list.



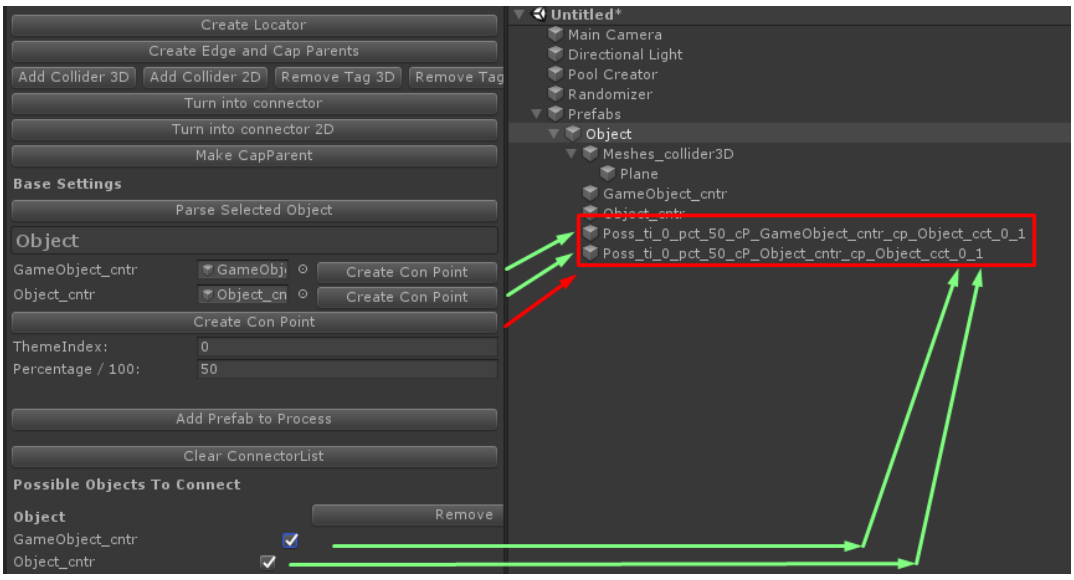
Now to add the connection possibilities:

We want this room to randomize itself so click on “Add Prefab to Process” while selecting the top of the prefab. You will see the list of connectors appear with a Boolean checkbox next to it. This is the possible connection points that can be tested by a connection. (For example if you want to rotate an object to fit each connector, or customize the connectors so they can only have specific rotations.) For now select all of them.

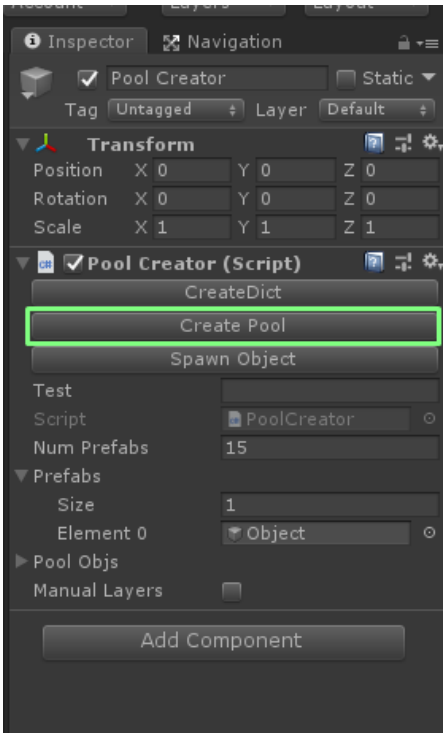
Click on CreateConPoint – you can create a con point for each individual connector, or for all of them (big button). Click on the big button for now.

You will notice that there are some new locators that have been created with the data you inputted.

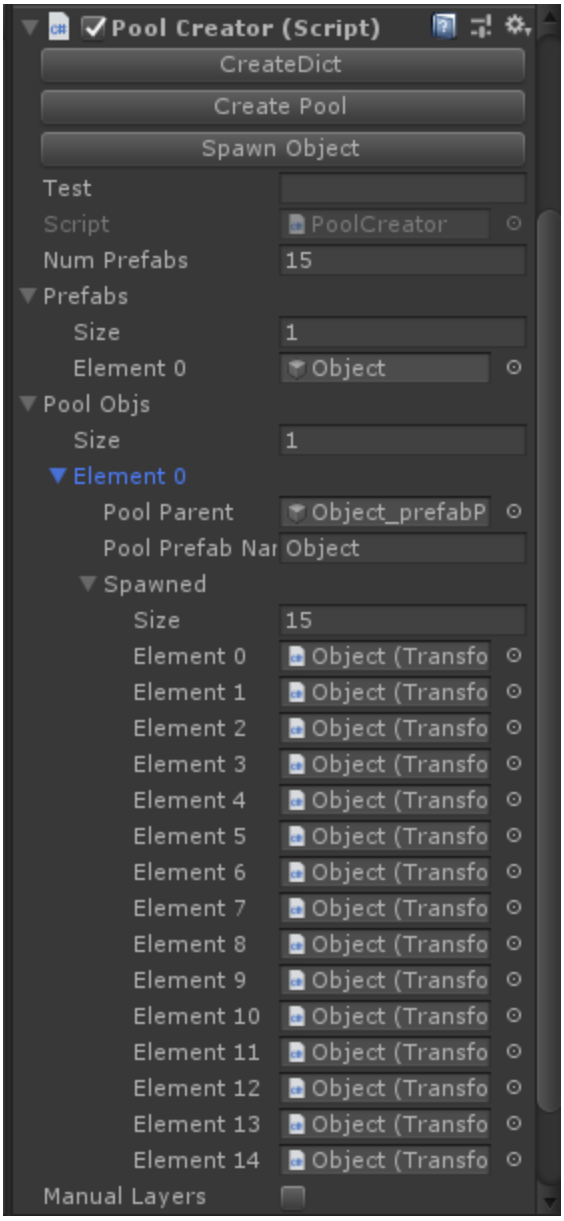
Congratulations – you have just created your first randomized object. It seems like a lot of work to go through for procedural generation, however this method allows for a lot of customization once you understand how it works and why it works this way. (Explained below further)

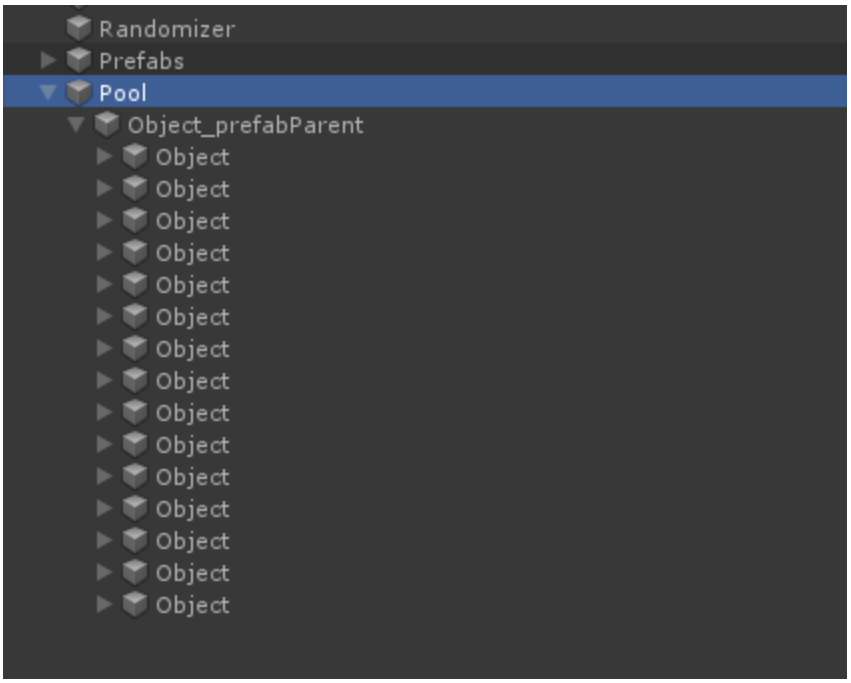


For now we can test it out – Select your Pool Creator and hit “Create Pool”.



If done correctly the pool objs list should now have been populated with a list of objects.





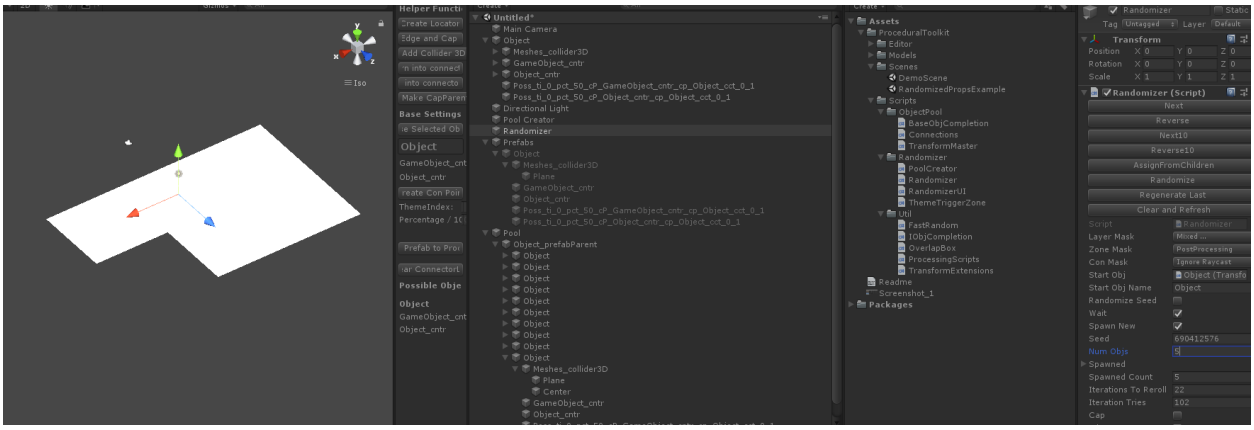
You will also notice that a pool object has been created with all the pooled prefabs that you have just created

Step 5: Randomize!

Select your randomizer.
Change the "StartObj" value from GameObject to Object

Hit play.

Now click randomize and you should see your randomizer in action!



**Note as this test object I created in the examples only had 2 connectors there are not many permutations that are possible, the demo scenes have a few examples of what can be possible using some of the techniques included in this kit. The more connectors you have, the higher the chance of something random happening.

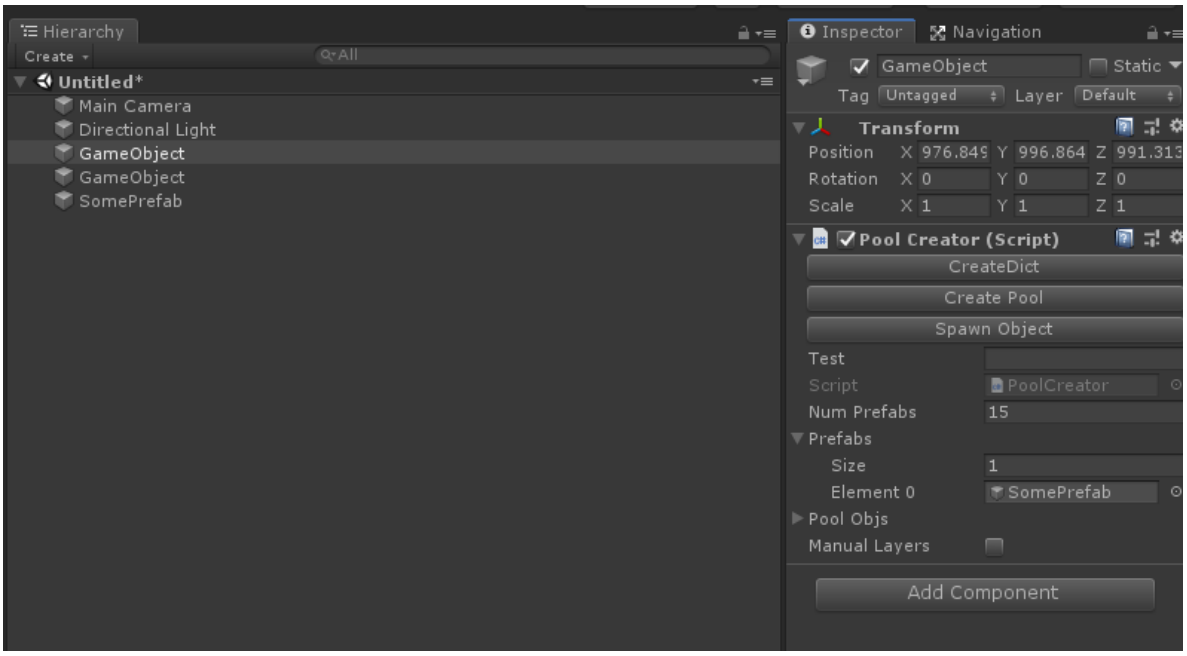
A nice number of connectors to have to actually see more randomizing happening is 3 (on different facing axis').

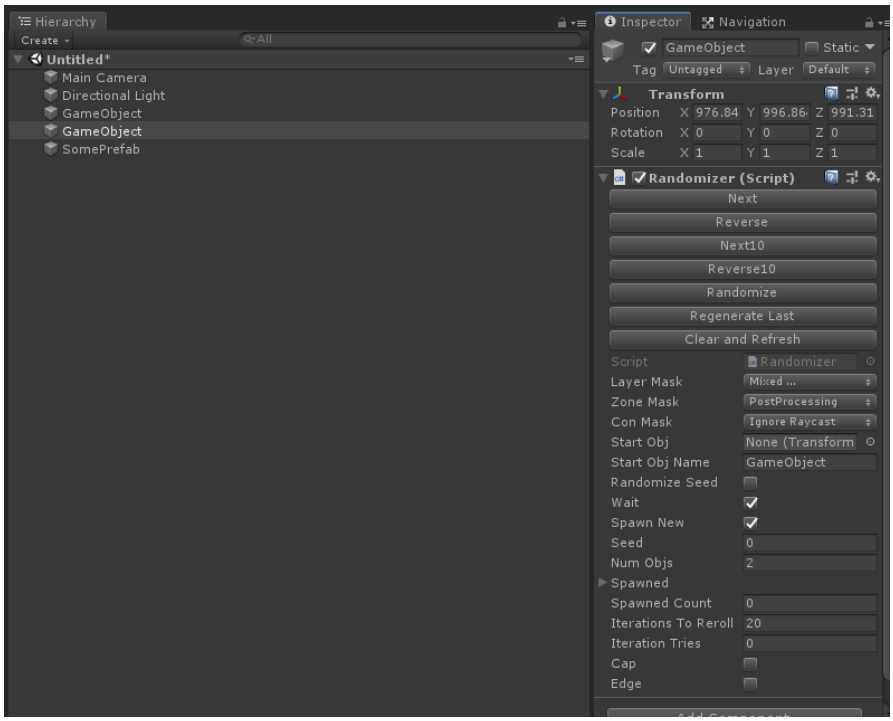
You may see an example of this scene completed – Basic Start Complete in the demo scenes folder.

How Procedural Toolkit Works – a primer

There are 3 main components to procedural toolkit:

The Pool Creator/Manager
The prefabs
The Randomizer script





Once these three objects are in your scene you may start producing prefabs that are ready to be randomized in your project.

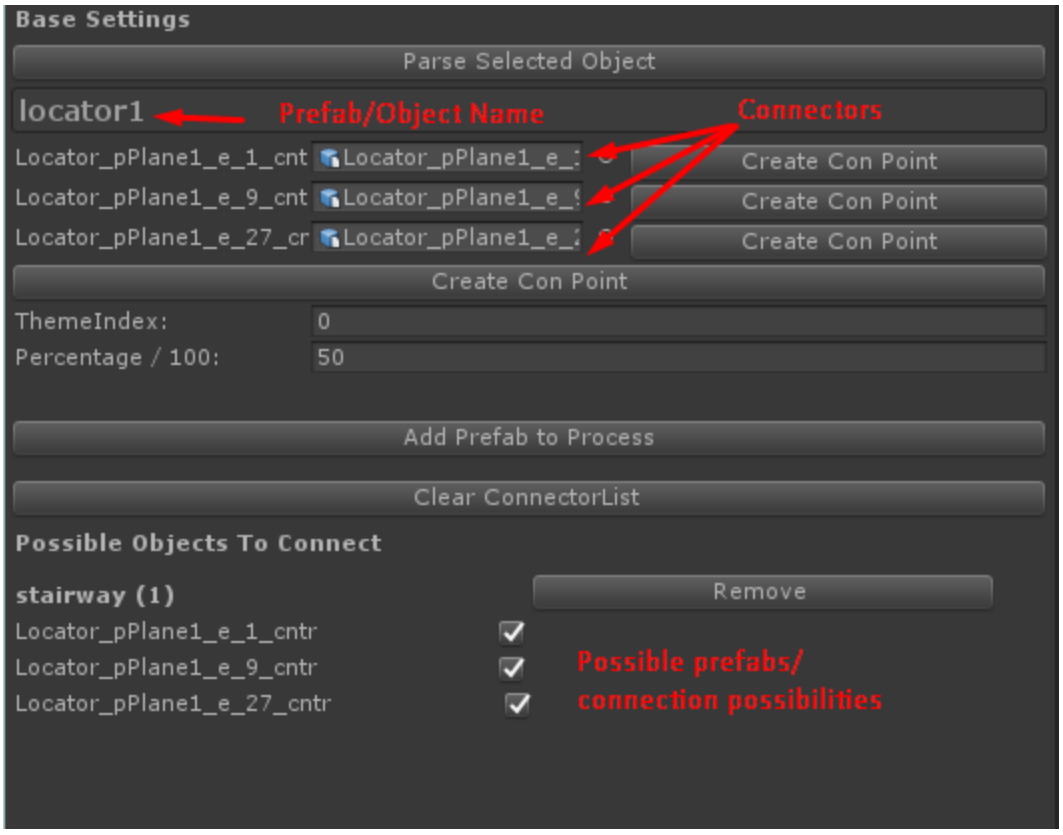
The basics of procedural toolkit prefabs are very simple. Create your mesh and create a few locators where you want the objects/cells to align with each other. You can then define as many connector links as possible that are related to this room.

A typical dungeon roomcell definition

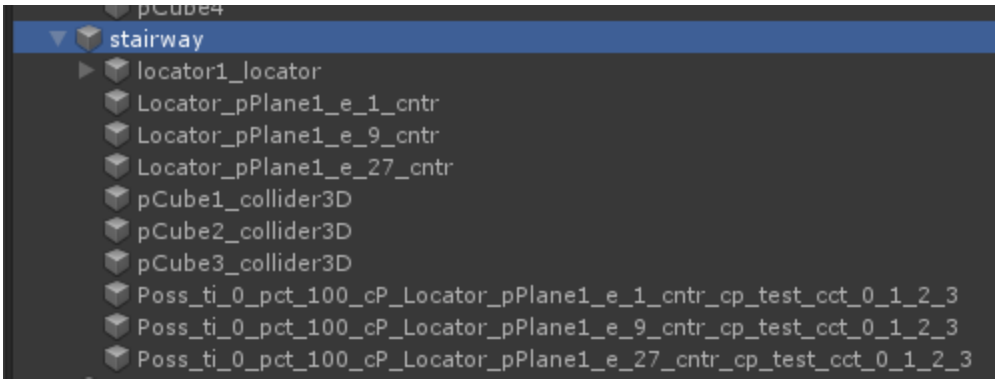
A start room may have 4 connectors (NESW)

A connector can have links to the same dungeon cell, or a different dungeon cell. And rooms can be rotated in specific orders per connecter per connector possibility.

After defining your object requirements you can use the editor window to quickly and accurately create the required meta locators that are used when the pool is created. Locators with naming were used as sometimes a designer may not want to do this step inside of Unity – thus by using the basic locator/game object one can simply create them in their program of choice as long as there is the required hierarchy and naming conventions.



You must have at the bare minimum some connectors on your object (_cntr / _cntr2D) prefix with the z axis (blue) facing outwards for 3D and the Y axis facing outwards for 2D.



By creating connection points the editor will know on pool connection which objects can be connected to which connectors.

What do these strings mean?

A brief explanation of each separator:

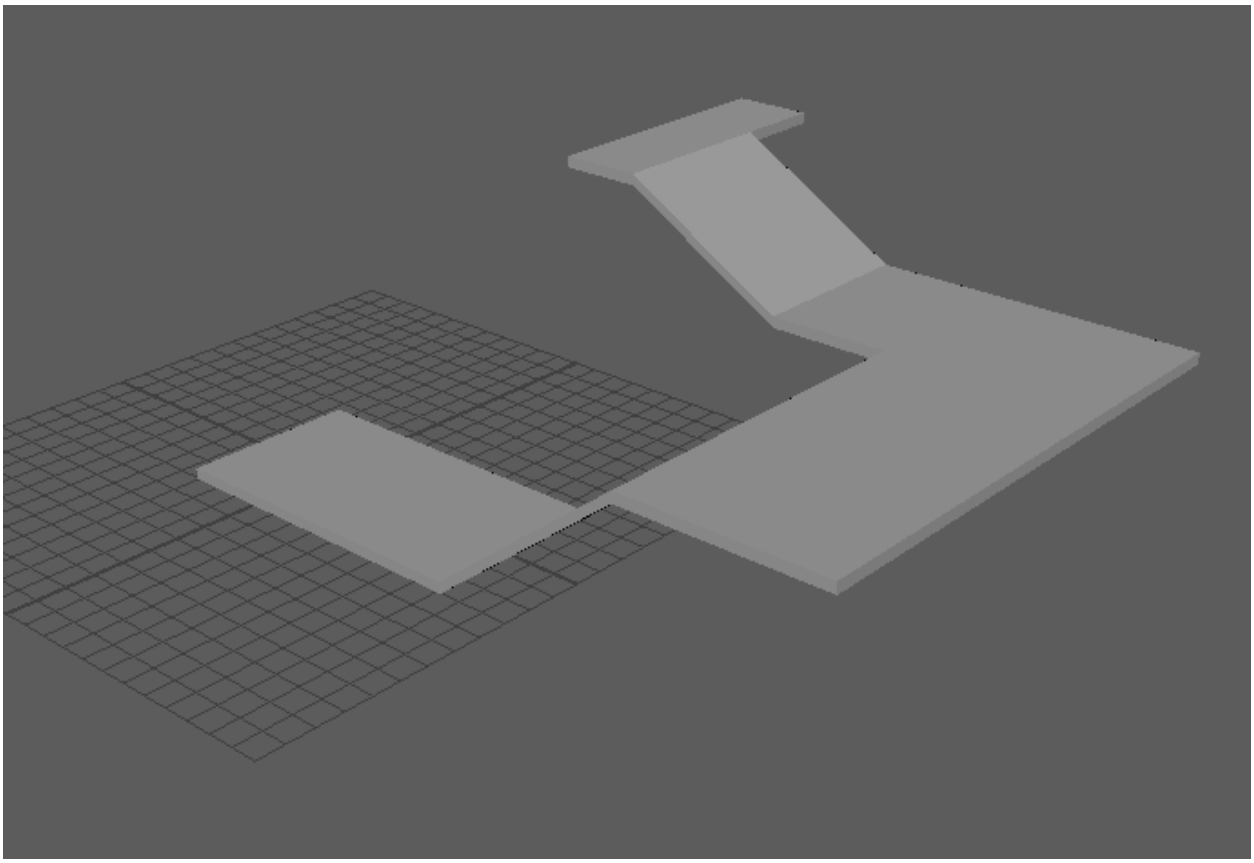
_ti_ = 0 ~ ∞
Theme Index, to be used with theme trigger zones (more advanced concepts)
pct = 0 ~ 100
A percentage value (dice roll) – each connector possibility will be rolled against a value, if that value is within the percentage this room will be tested against.
_cP_ = connection point name
The connection point of the prefab that you are spawning from
_cp_
The connection prefab that you are spawning
cct
The possibilities on the prefab connection points that you can connect to (index of the possible connections.)

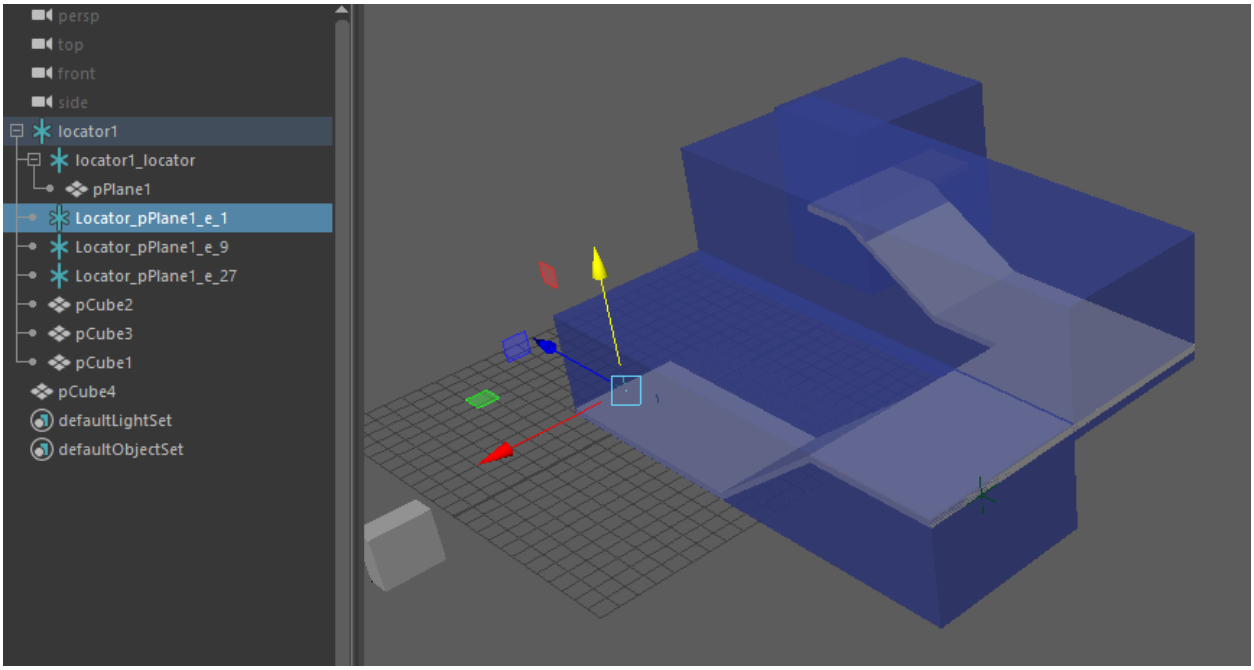
Collisions

Procedural toolkit handles collisions by using box colliders. (Not AABB) but Physics Overlap. This creates the intersecting logic of the random generation process. By using Physics Overlap instead of Intersects it allows us two important things: a non-allocating test against all objects in the scene, and a separation from the AABB bounds of the object. Which means that we can arbitrarily rotate and create multiple colliders per mesh. This allows for many complex and creative object/prefab designs that are not otherwise possible with the usual bounds.intersects method.

As per above screenshot –

To create colliders that will be tested against create some cuboids that will form around the object in the best fitting form as possible.
A 2D object will have a collider2D tag
The 3D object will have a collider3D tag
E.g. this stairway model:





I have created 3 cubes (pCube1~3) that surround the form of the mesh. You may create more to have a more accurate test, however the generator will take a lot longer the more colliders you have.

You will need to decide if you require the extra accuracy or not.

These box meshes will then be appended with the suffix
_collider3D for 3D meshes
And _collider2D for 2D meshes
pCube1_collider3D eg
or sprite_collider2D for sprites.

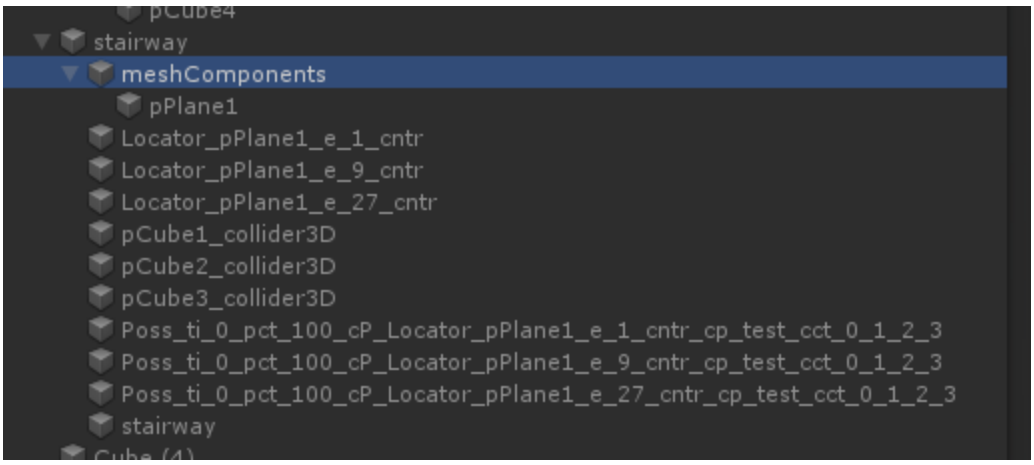
As can be seen from the example images, there is a locator that represents the connector we are going to connect the stairway to. This locator has its Z axis (blue) facing outwards. Always ensure that each model you are turning into a prefab has at least 1 axis facing outwards or some of the features will not work.

The full structure of the Procedural Toolkit Prefab:

By default, the layers used are Ignore Raycast (for the connector testers) a small box collider that is created on each connector to test for edge/cap creation, and PostProcessing for Trigger Zones. The default layermask that is used to test the object collision is the Everything but the two of these layers.

If you would like to use your own layers just follow the same structure. Have 1 layer for Trigger Zones, 1 Layer for Connectors and then invert the layermask for the default layer mask. Inside the pool creator you can select to use default masks, or your own custom ones – however you will need to manually assign layers to your connections if you opt to use custom layers before creating the pool.

Below is a full example of a prefab that is ready for procedural toolkit to process:



The root of the object for sanity checks is just a basic locator, this generally should be best kept at default scale and rotation. I have included extra checks for scale just in case but child scaling and rotations can be inaccurate – so use wisely.

meshComponents (or whatever you want to call it) is just a locator/gameObject that keeps all the actual game meshes in the scene.

Locators with then _cntr or _cntr2D tag –

These are the locators that the randomizer will cycle through and test to see if there are any possible prefabs that can be connected to these connectors.

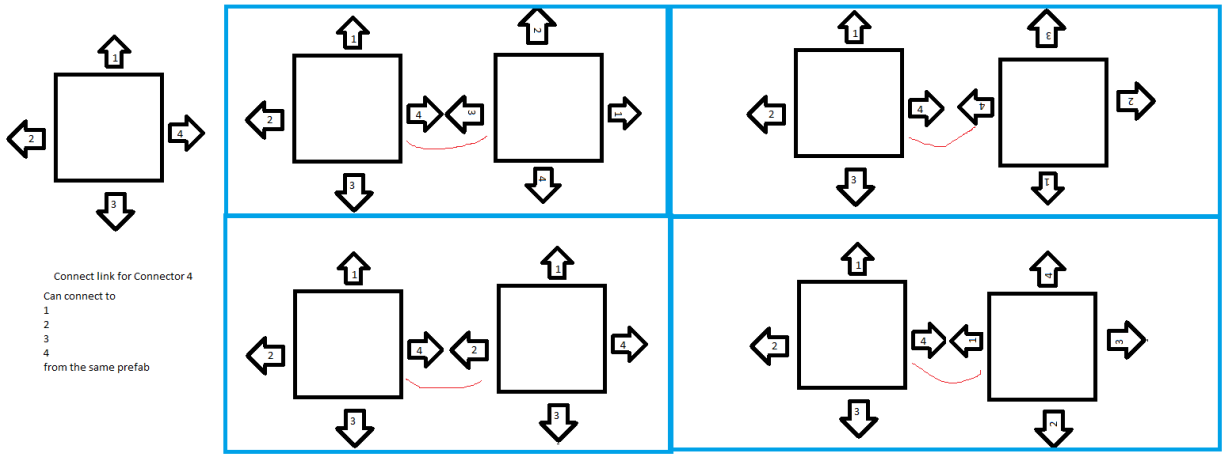
Objects with _collider3D / _collider2D tags

These are the objects where there will be collider information (boxes or whatever) under them. When processing the pool manager will attempt to create the best possible collider based upon all the children under this object that have either a renderer or sprite renderer component attached.

Poss + (suffix combinations)

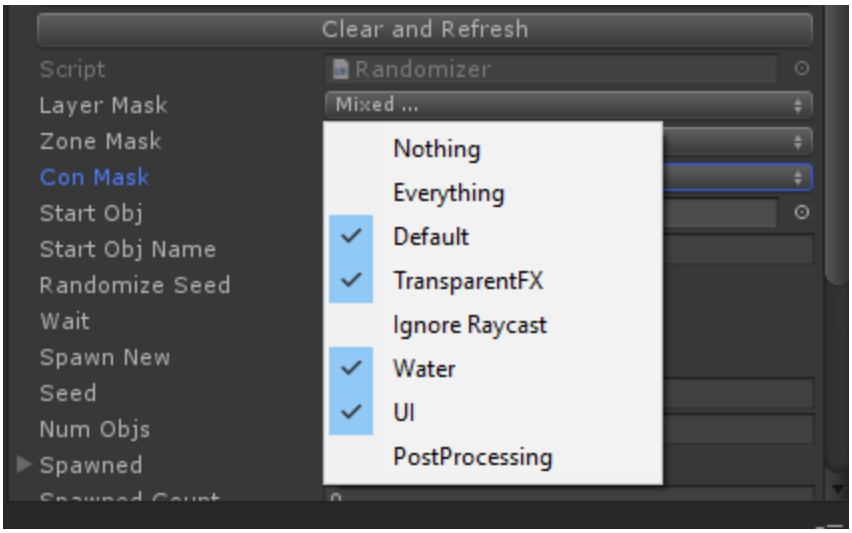
The meta locators that contain data about which prefabs can be spawned at which connectors.

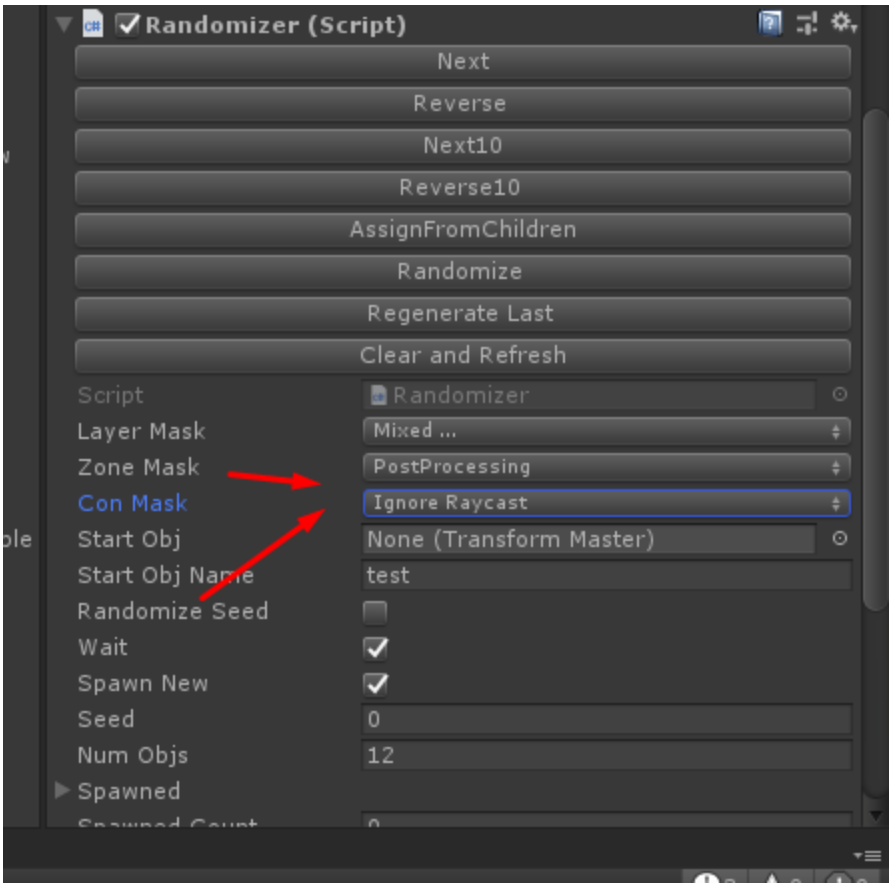
Connector Example while randomizing:



This is an example of how the randomizer works – when it has found a possible connector (Connector 4) it will loop through each possibility and try align the links until it reaches a fit. If it does not reach a fit it will continue iterating until either the iteration tries reduce to 0 or the number of objects required has been reached.

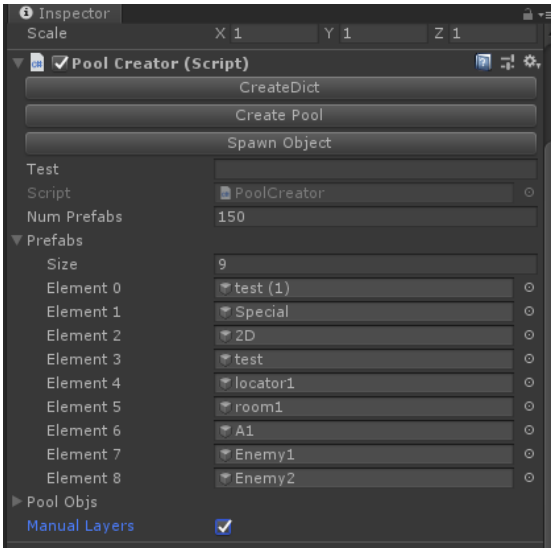
Advanced Concepts: Layers

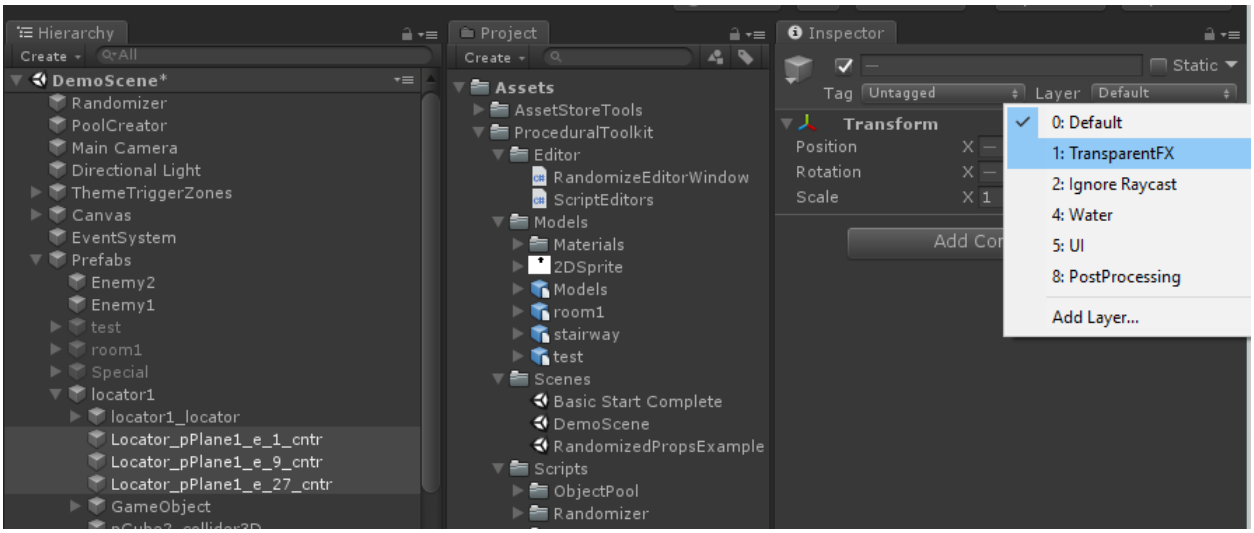




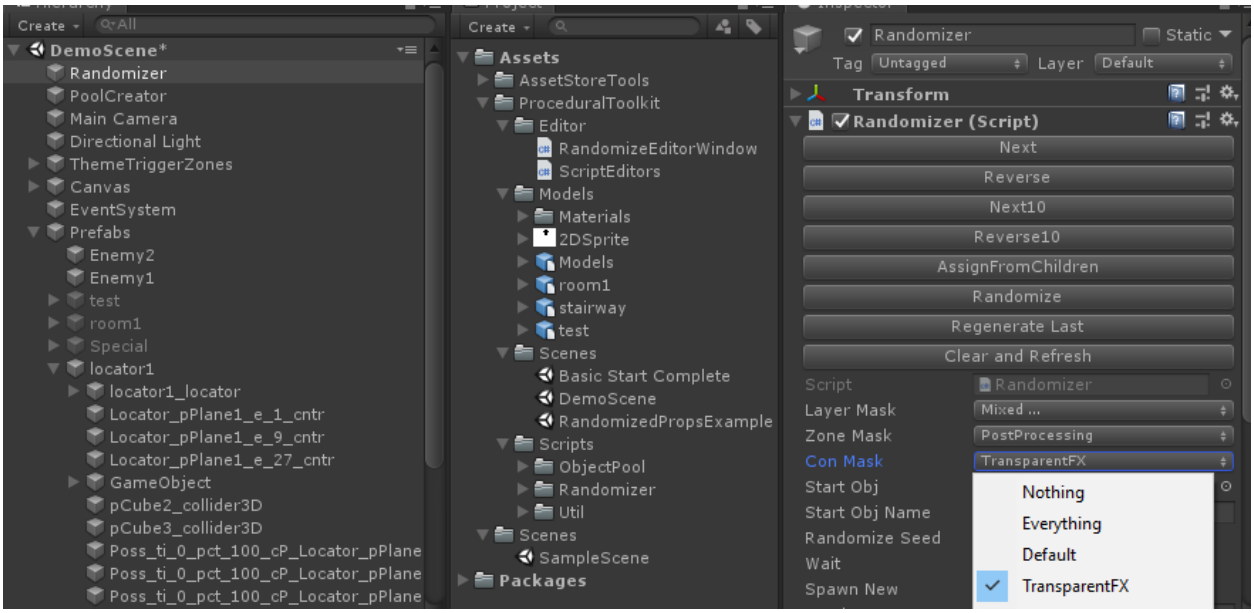
Procedural Toolkit uses layer masks to test for various intersections – meshes, colliders and edges/caps during the randomizing process. This ensures that you do not get an object colliding with itself or any other mesh in the scene.

If you check Manual Layers (off by default) – ensuer you go through all your prefabs and set the collider of the connector (only connector not children) from default to whatever layer you are using on your randomizer.





*If I use TransparentFX as my layer for testing connectors, I want to check this off



And select it as a connector mask.

Additional Features: Spawnpoints

Spawnpoints

A meta locator can also contain a spawnpoint. This is denoted by the name “Spawnpoint_” and the tag “_sp_” followed by the name of the object that you want to spawn. These objects are also seeded through the randomizer. The randomizer will pick from the list of spawnpoints a random object to spawn every time.

Eg.

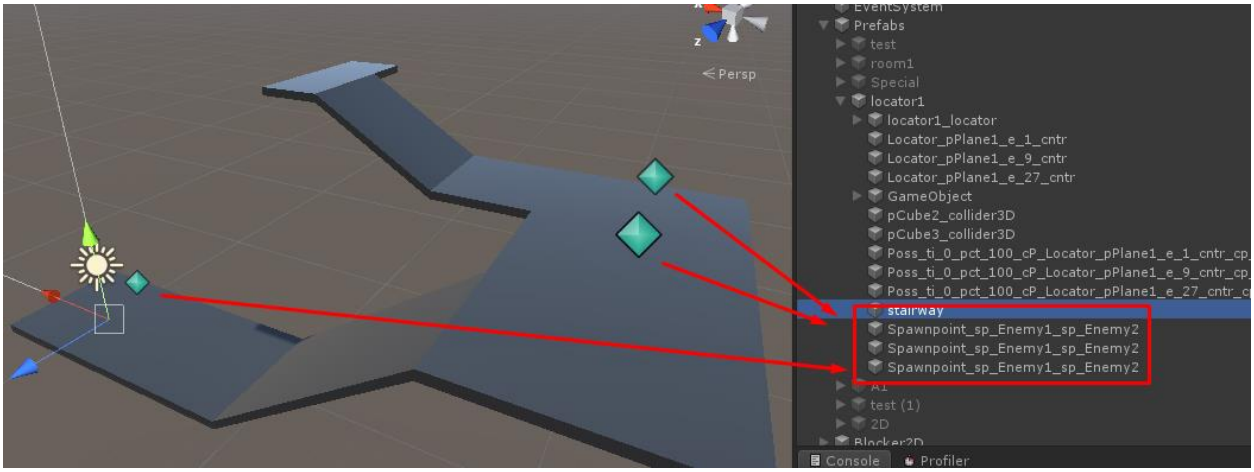
Spawnpoint_sp_Object1_sp_object2_sp_object3

Means that

The spawnpoint can spawn object1/2 or 3

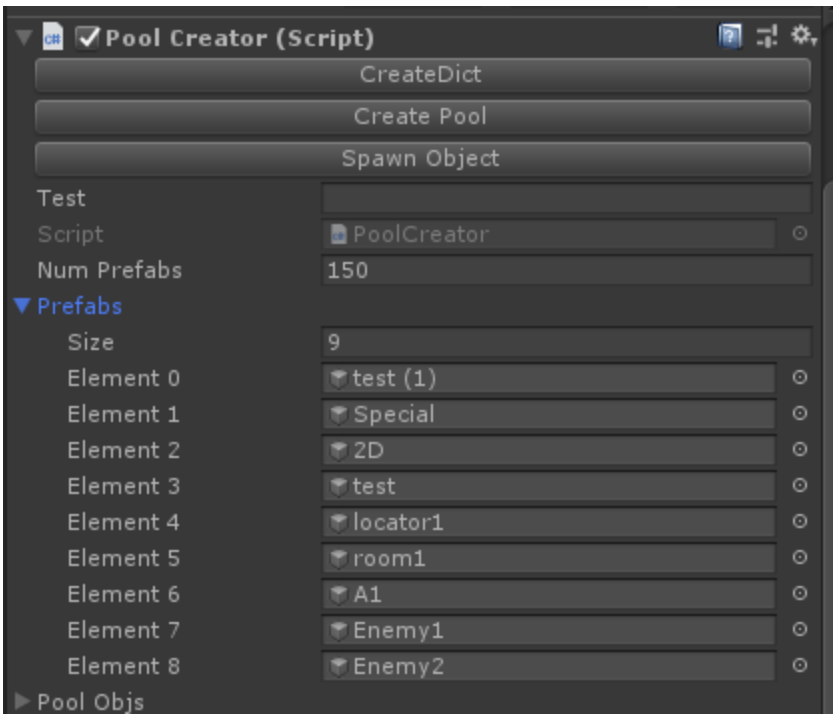
In the Finalizing step, the randomizer will roll a dice based on the seed/spawncount and attempt to spawn an object for this spawnpoint.

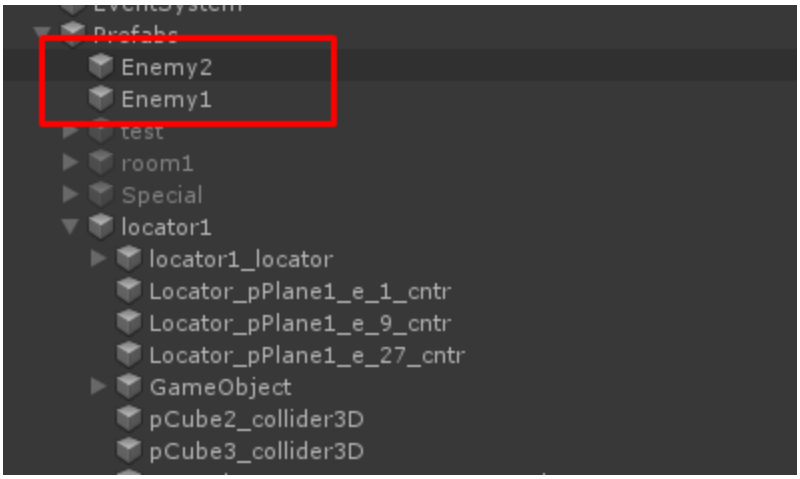
**Remember that spawned objects are also prefabs that need to be added to the pool creator.



In the above figure the object has a chance of spawning Enemy1 and Enemy2 on these 3 prefabs.

Additional Features: Spawnpoints

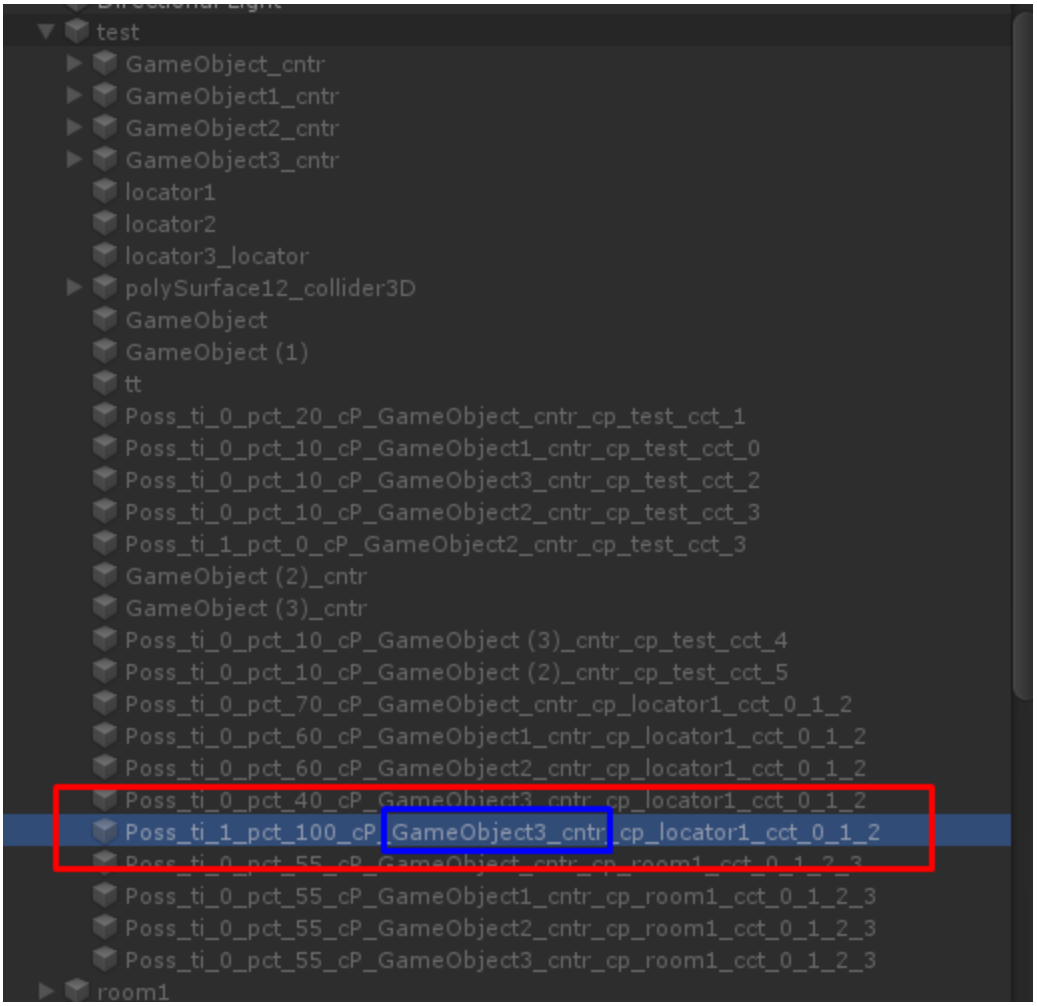




*Note the PoolCreator does not need to have the meta data tags – if there are none it will just not add any connection possibilities for the randomizer and act as a generic fixed pooling system.

Advanced Concepts – Trigger Zones

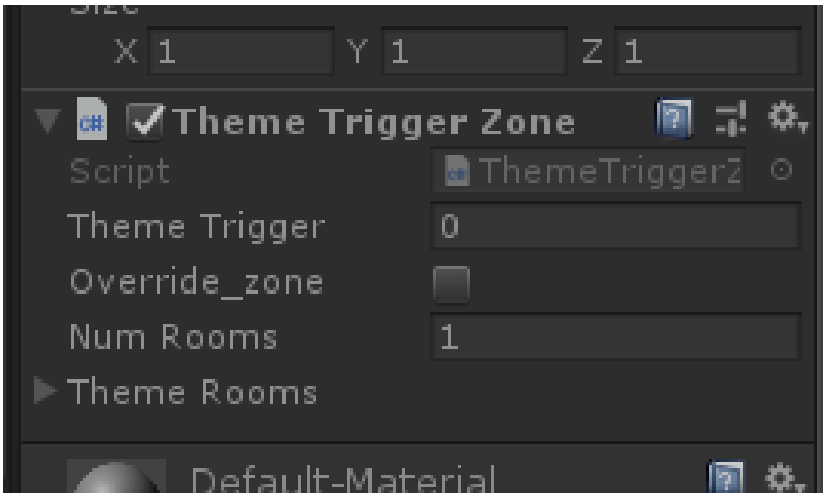
Sometimes you will want to control the flow of the object you are randomizing, Procedural toolkit also includes a way to do this that is intuitive and easy to understand. Simply add a ThemeTriggerZone script to the object and define the theme index of the objects you want to fall into that zone. In addition you will need to add more meta data to an object that includes the information about the objects that you want to spawn in that theme index zone.



In the example above we can see that the connector GameObject3 has a possibility of 100 to spawn the prefab locator1 if there is a TriggerZone with the theme index of 1.

In addition there are more things TriggerZones can do – you can set a maximum number of rooms for trigger zones, or you can override the triggerzone completely and have it randomize as many rooms as possible.

Theme triggers can be defined inside the Theme Trigger field in the randomize editor window (or you can just write one yourself by renaming the object)

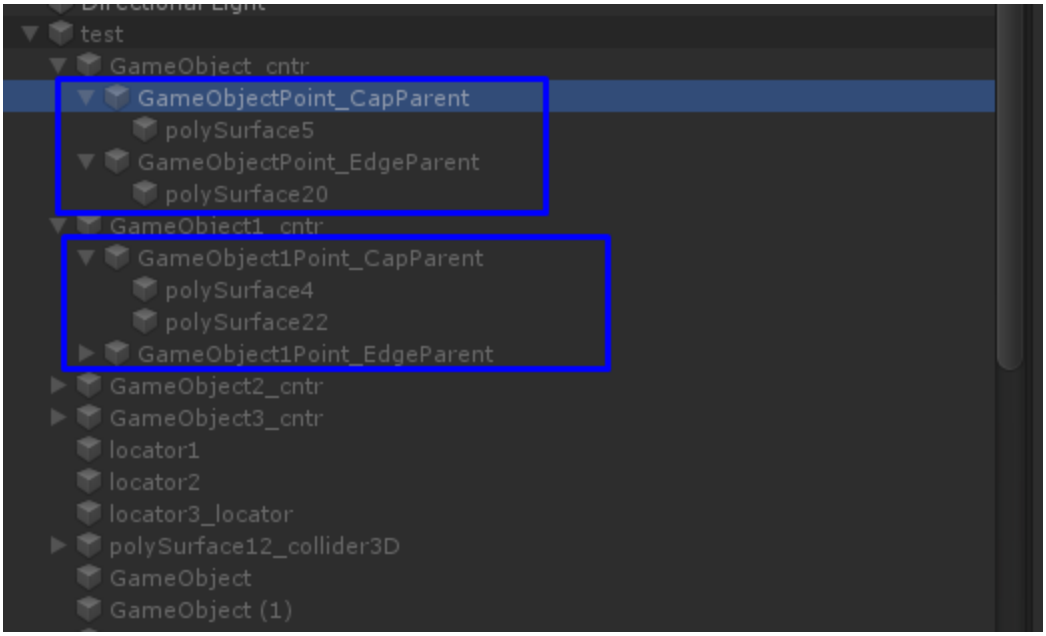


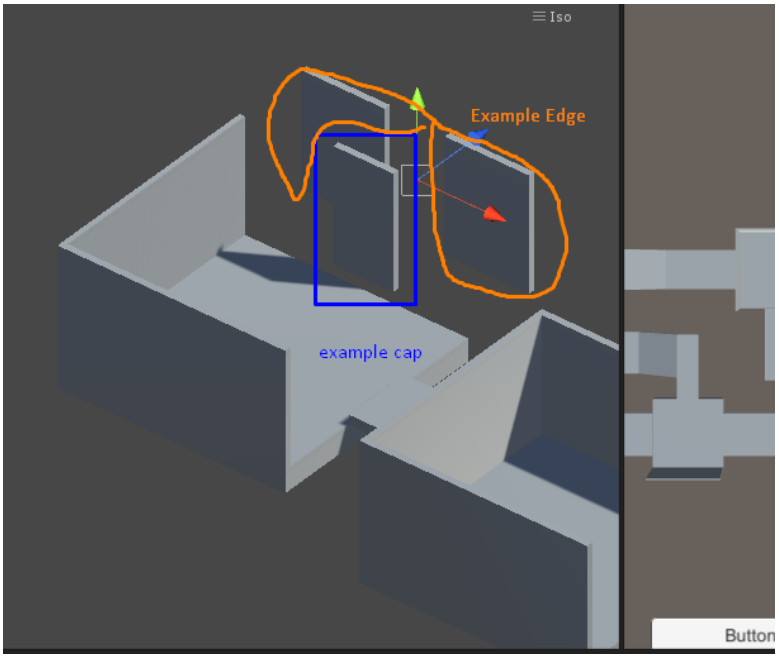
The following parameters are available for trigger zones, the number of rooms shows the max rooms that the trigger zone (under this index) can spawn globally – (eg if there is more than 1 trigger zone in the scene with a trigger zone that has a theme trigger of 1). The theme trigger is a grouping value that allows you to specifically set types of prefabs that can be connected for this value.

Theme rooms is used internally to count how many rooms are available (for capping the number of rooms)

Advanced Concepts – Edge and Cap Parents

Sometimes you may want to introduce a form of walling off a cell, such as doors and debris, this is also included as a feature in PTK, you can create a child object under each of your connectors labled with suffix _CapParent or _EdgeParent and the pool manager will process these items accordingly. An example of this is shown in the demo scene with a dungeon cell that has walls as edges and doors as caps, you can also add as many numbers of objects under these locators and the randomizer will pick a random one after the finalizing stage.





Controlling the randomizer through code: (ScriptEditor.cs/Randomizer)

The script editor has some examples of how to use the randomizer via code. The main functions are to find a **TransformMaster** script to try and call the Randomize(bool) method. RandomizeFromStart will generate a new startObject (if checked) and clear the spawned objects.

Undo and Redo

Undo and redo can be used to create the feeling of a growing object – the undo and redo methods (Reverse/Randomize) will create the same object based upon seed no matter what values you input.

Useful Methods:

Method Name (Randomizer.cs)	What it does
RandomizeFromStart(false);	Creates just the start object
RandomizeFromStart(true);	Creates the start object and loops until numObj is complete. If wait(bool) on randomizer is selected it will use a coroutine, otherwise a normal method will be used
Randomize(false);	Tries to iterate using the next seeded object until we reach numObjs+1
Randomize(true);	Tries to iterate using the next seeded object until we reach the max numObjs
InitSeed();	Initializes the seed again to a given value +spawncount (step value for restoring seed)
Clear and Refresh();	Moves the object that has been created to one side and spawns another – if you want to repeatedly spawn the same seeded object.
Reverse	Goes back one cell in the randomized process, stops at the first iteration (startObj) ***Loopable with for loop/while loop***
Move Forward	Moves forward one cell in the randomized process. Keeps going until no more iterationTries or no more objects to spawn. ***Loopable with for loop/while loop***

**RandomizerUI has a few options that are available for use with the Unity UI system. Look at DemoScene for examples on usage.

Method Name (BaseObjectCompletion.cs)	What it does
This is an interfaced script that has some hooks for spawning objects. You can add some custom logic to this script so you can do some actions after each item is created. For basics I have added in spawnpoints as starter examples	
`SpawnObject` (false);	This object is called on object creation (after RandomizeOnce step by default – does spawning
SpawnObject(true);	This object is called on object creation (after RandomizeOnce) – no implementation yet, but if set to true the default interface is to assume that the object is an "exit" object.

Method Name (TransformExtensions.cs)	What it does
This is an static method that has the methods for aligning objects. There are some methods you can use to create your own linked objects manually (without the need for a transform master)	
`Alignment(aCon, bCon, aObj, bObj, parent, is2D)`	This object aligns two objects, (A object to B object) (B object is the parent object) by their connector axes. If parent is selected – the a Obj will be parented to the b connector. By default (is2D off) the object uses the transforms Z forward and inverts it to create the connection. If IS2D is selected the object will use the inverse of the Y axis instead as the forward.

Troubleshooting your meshes

These are some of the common things that may happen when you create your prefabs for randomizing

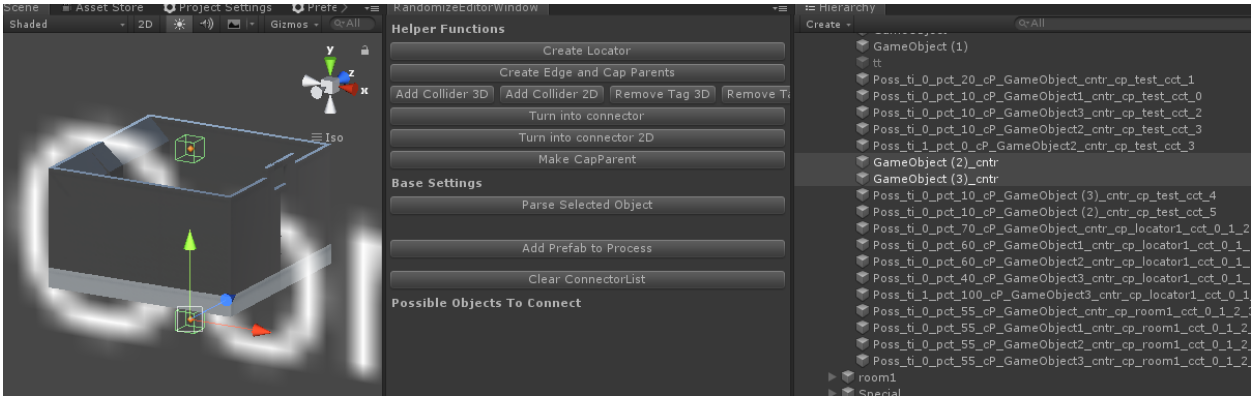
Try have exits that will not loop over each other on long randomized paths (eg if you have a 90 degree turn with 2 connectors, the randomizer will likely end up failing after a few iterations as it will create a O shape most of the time.

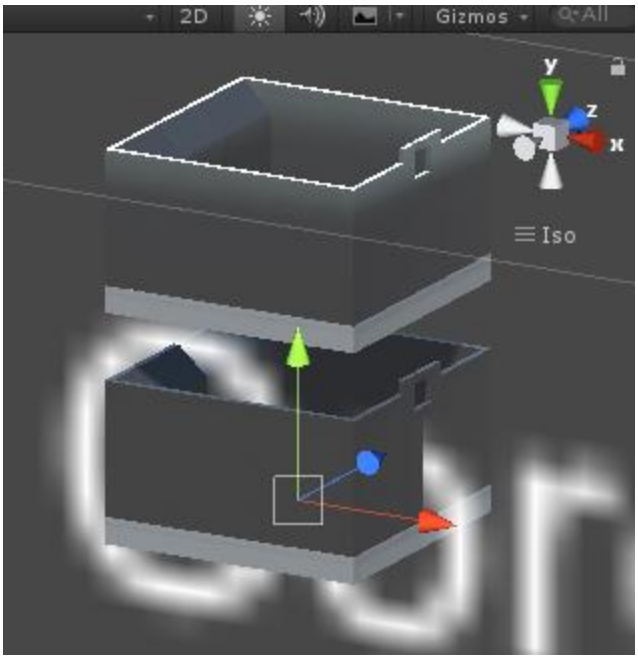
Some tips – the meshes will need colliders if you are to use them with the randomizer. Unless *** you know that the objects are never going to overlap regardless of connectors.

***Ensure your prefabs have at least one connector correctly facing outwards otherwise there may be some inconsistencies.

Ensure that your walls are inside your box colliders that you are going to use for testing. If you have colliders on your walls* if not it is ok – this is because in the wall creation step the objects connector points will test and see if there are any other objects that are colliding with the mesh. (Essentially the reverse of the randomize step). If there are none it will create a wall. Colliders that are outside of the default colliders will sometimes trigger the edge/cap intersecting method depending on how you have your objects set up.

Vertical connector points – Unless you want your **vertical connectors** rotated in the z axis, just create them as if they were facing forward. That way they will emulate an object that is stacked on top and aligned properly.





As can be seen from the example these locators are facing z forward(not rotated)

If you rotate the Z upwards it will "reverse" the axis of the connector. Which is ok if you need that feature. However for most cases this is the expected behavior for vertical or stacked cells.