# Documentation

# Table of Contents

# Contact

**Website:** https://pedro-rainha.com/
**Email:** pedro.rainha.dev@gmail.com
**Twitter:** @MrAbnoxDev

Feel free to give any feedback and message me for any problems or information, I will try to get back to you as quickly as possible. I hope I don't cause any inconveniences and I apologize in advance

# Generation Settings Window

To Generate a Procedural Dungeon level you need to create a Scriptable Object called Generation Settings. To create the settings, you can either press right click in the editor, press create and then at the top press Generation Settings or you can go to Window at the top and press Dungeon Generation, once the window is open you can click create new Settings.

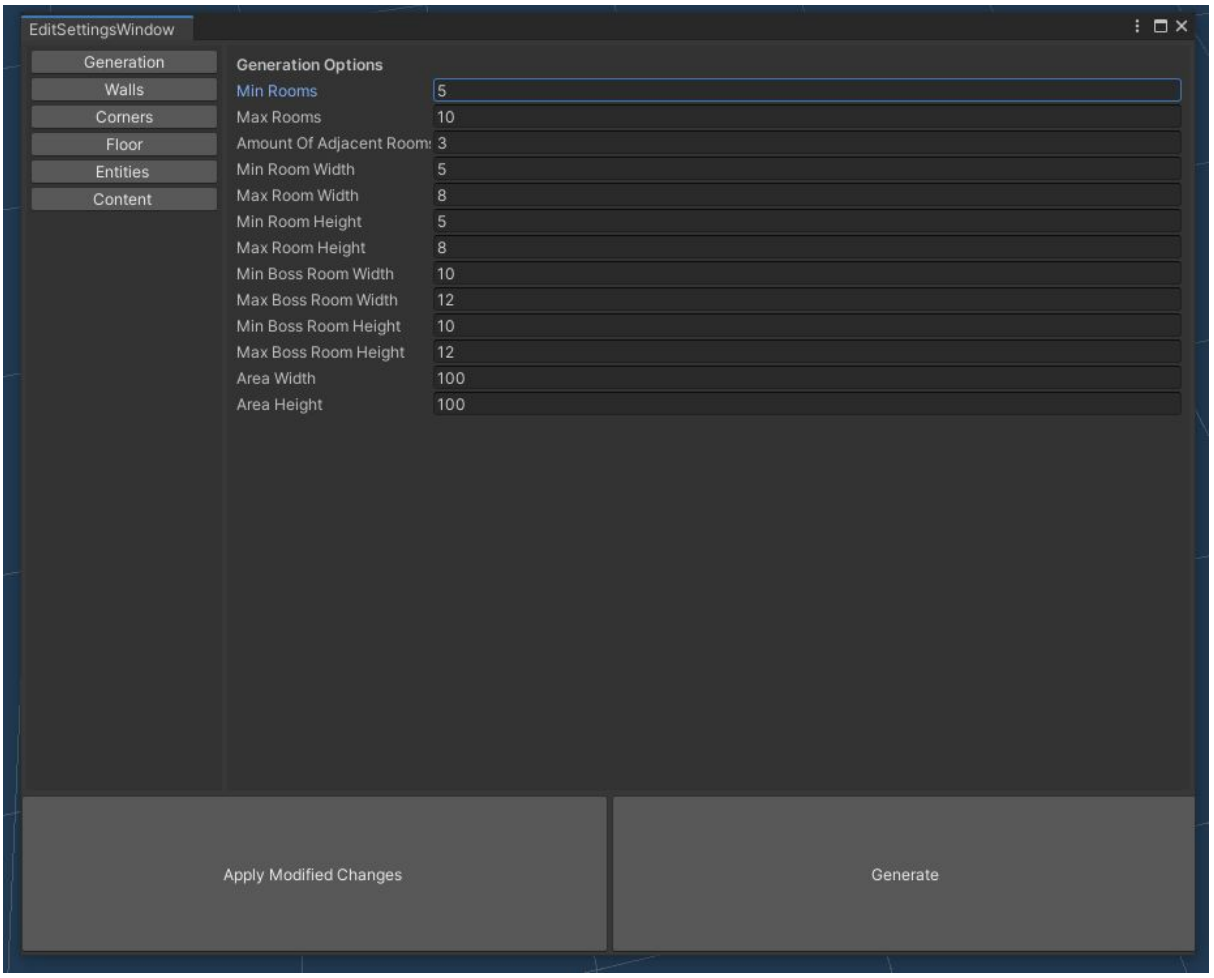
To Generate a Procedural Dungeon Generation the prefab Generator needs to be placed inside the scene, and utilise the settings scriptable object created.

The Generation Settings window is dependent on the Settings Scriptable Object, to open a window you can double click on any entity scriptable object and a Window will open up, or you can go to the inspector while the scriptable object is selected and press the button “Open, or you can also open the Generation Settings window by pressing it in the window tab, you can drag the scriptable object there and press open from there.

Once any of the values are changed make sure to press the button “Apply Modified Changes” otherwise there will be no changes. One thing to keep in mind is that unlike monobehaviour scripts where values changed in the inspector during runtime are reversed back to default after stopping the scene, changing the values for this window (any scriptable object value in matter of fact) will actually change them permanently.

## Variables

This is the window that holds all of the data for the Generation Algorithm settings. There are 6 pages in this window that can be accessed inside:




### Generation Page

| | |
|---|---|
| **UseDDA (boolean)** | This variable for the research purpose does not do anything. |
| **AmountOfScenes (int)** | This variable for the research purpose does not do anything. |
| **MinRooms / MaxRooms (int)** | These variables control the minimum/maximum number of rooms there is and will pick a random number between them. |

| | |
|---|---|
| **MinRoomWidth / MaxRoomWidth (int)** | These variables control the minimum/maximum room width and will pick a random number between them. |
| **MinRoomHeight / MaxRoomHeight (int)** | These variables control the minimum/maximum room height and will pick a random number between them. |
| **AreaWidth/AreaHeight (int)** | These variables control the width and height, and set it exactly. This area refers to the borders of where the rooms can spawn in, the bigger the area the bigger the space where the rooms can spawn. |
| **MinDistanceBetweenRooms** | This variable controls the min distance between rooms (Use so rooms don't stick completely together) |

## Walls Page

| | |
|---|---|
| **Walls (List<ObjectList>)** | List of custom class, that contains a string for the name of the object which will be replaced, and a GameObject for the object to generate. Objects will be picked randomly for generating the walls using the content in the list. *It is required that the GameObjects have the tag "Wall".* |

## Corners Page

| | |
|---|---|
| **OuterCorners (List<ObjectList>)** | List of custom class, that contains a string for the name of the object which will be replaced, and a GameObject for the object to generate. Objects will be picked randomly for generating the OuterCorners using the content in the list. *It is required that the Game Objects have the tag "OuterCorner".* |

| | |
|---|---|
| **InnerCorners (List<ObjectList>)** | List of custom class, that contains a string for the name of the object which will be replaced, and a GameObject for the object to generate. Objects will be picked randomly for generating the InnerCorners using the content in the list. *It is required that the Game Objects have the tag "OuterCorner".* |
| **UCorners (List<ObjectList>)** | List of custom class, that contains a string for the name of the object which will be replaced, and a GameObject for the object to generate. Objects will be picked randomly for generating the UCorners using the content in the list. *It is required that the Game Objects have the tag "OuterCorner".* |
| **SquareCorners (List<ObjectList>)** | List of custom class, that contains a string for the name of the object which will be replaced, and a GameObject for the object to generate. Objects will be picked randomly for generating the SquareCorners using the content in the list. *It is required that the Game Objects have the tag "OuterCorner".* |

## Floor Page

| | |
|---|---|
| **Flooring (List<ObjectList>)** | List of custom class, that contains a string for the name of the object which will be replaced, and a GameObject for the object to generate. Objects will be picked randomly for generating the Floor using the content in the list. *It is required that the Game Objects have the tag "Floor".* |

## Enemies Page

| | |
|---|---|
| **MinEnemiesPerRoom / MaxEnemiesPerRoom (int)** | These variables control the minimum/maximum number of Enemies there is and will pick a random number between them. |
| **Player (GameObject)** | This variable refers to the Player Game Object. *It is required that the GameObject has the tag "Player".* |
| **Enemies (List<ListGameObject>)** | List of custom class, that contains a string for the name of the object which will be replaced, and a GameObject for the object to generate. Objects will be picked randomly for generating the Enemies using the content in the list. *It is required that the Game Objects have the tag "Enemy".* |
| **Bosses (List<ListGameObject>)** | List of custom class, that contains a string for the name of the object which will be replaced, and a GameObject for the object to generate. Objects will be picked randomly for generating the Boss using the content in the list. *It is required that the Game Objects have the tag "Enemy".* |

## Content Page

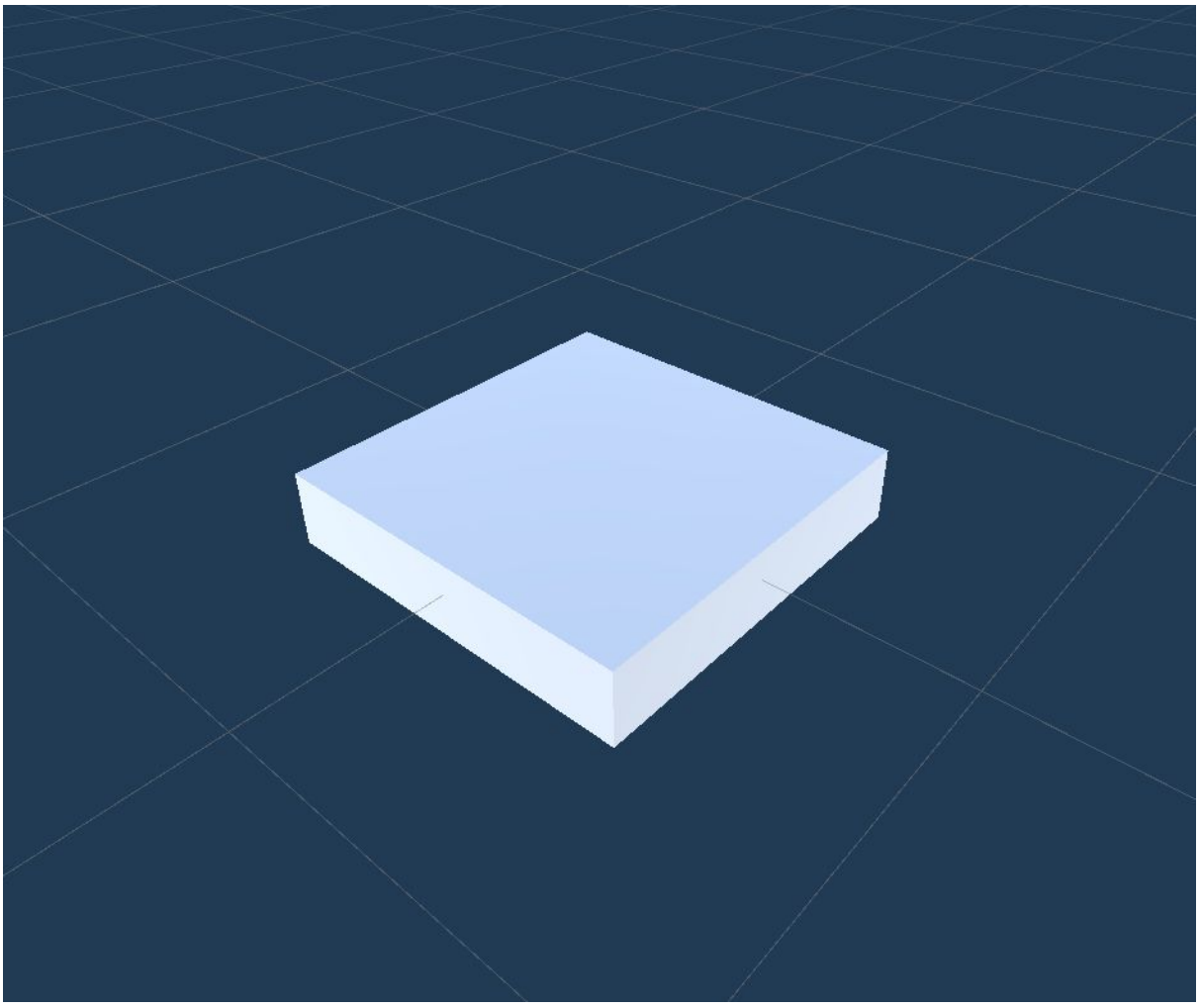
| | |
|---|---|
| **Breakables (List<ListGameObject>)** | List of custom class, that contains a string for the name of the object which will be replaced, and a GameObject for the object to generate. Objects will be picked randomly for generating the Breakables using the content in the list. |
| **Static Objects (List<ListGameObject>)** | List of custom class, that contains a string for the name of the object which will be replaced, and a GameObject for the object to generate. Objects will be picked randomly for generating the Static Objects using the content in the list. |
| **WallObjects (List<ListGameObject>)** | List of custom class, that contains a string for the name of the object which will be replaced, and a GameObject for the object to generate. Objects will be picked randomly for |

| | |
|---|---|
| | generating the objects placed on walls using the content in the list. |
| **WallObjectConsequitive Distance (int)** | Variables that control the consecutive distance between each wall Object, meaning, there will be wall objects each x. |
| **WallObjects Displacements (List<Vector3>)** | List of Vector3 that contains the displacement of the wall object, each member goes respectively with the number x of the wallObjects. |
| **CornerObjectsRate (int)** | Rate of Corner Objects (0 - 100) |
| **DestroyablesRate (int)** | Rate of Breakable Objects (0 - 100) |
| **MiddleObjects (List<ListGameObject>)** | List of custom class, that contains a string for the name of the object which will be replaced, and a GameObject for the object to generate. Objects will be picked randomly for generating the Middle Objects using the content in the list. |
| **MiddleObjectsRate (int)** | Rate of Middle Objects (0 - 100) |

# Assets

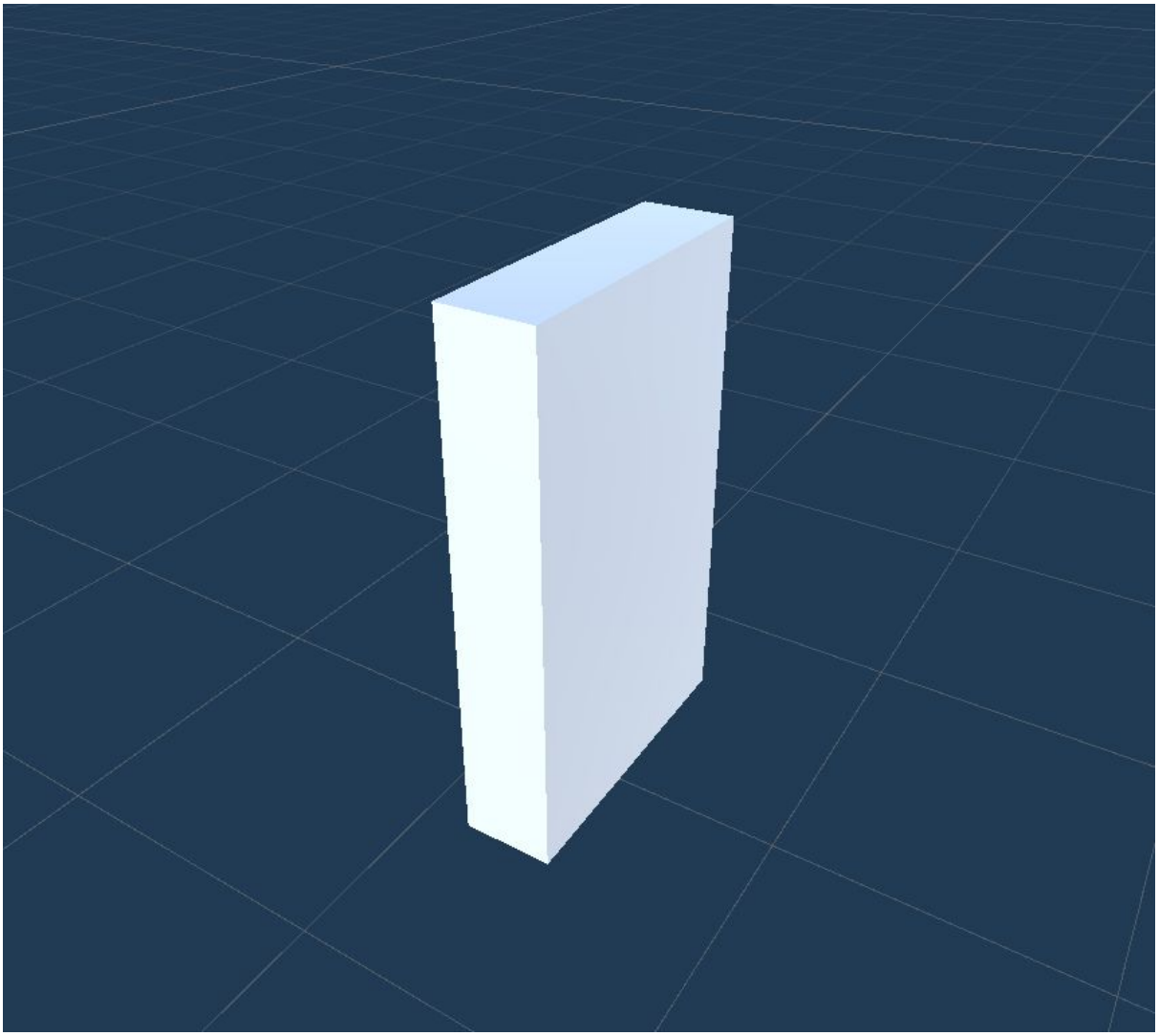
## Floor

The floor consists of a 1x1 in both x and z object and the y scale can vary for custom assets, as seen in the image below:

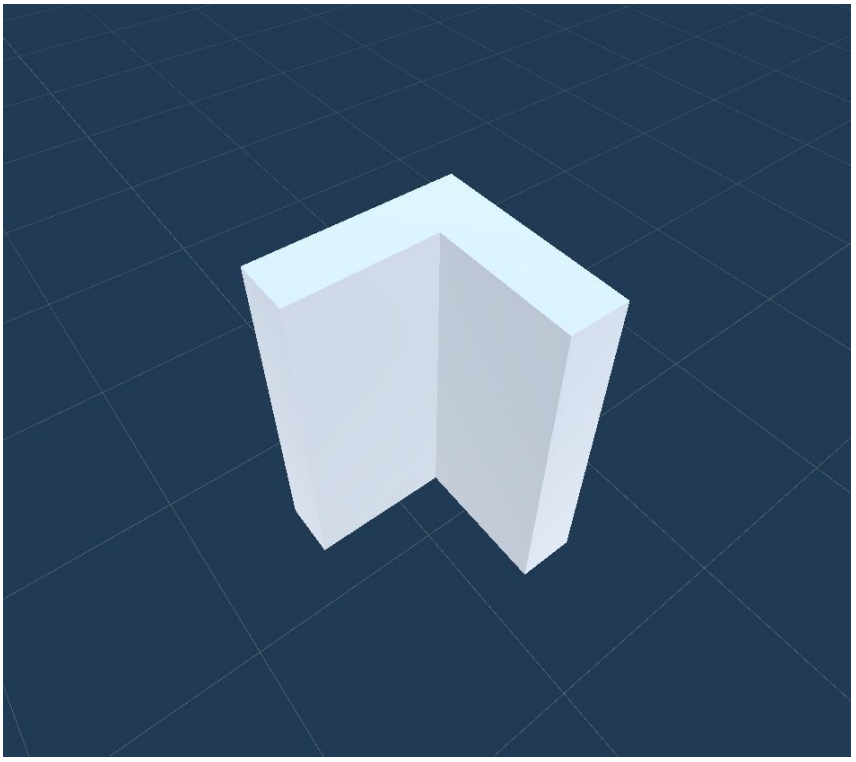


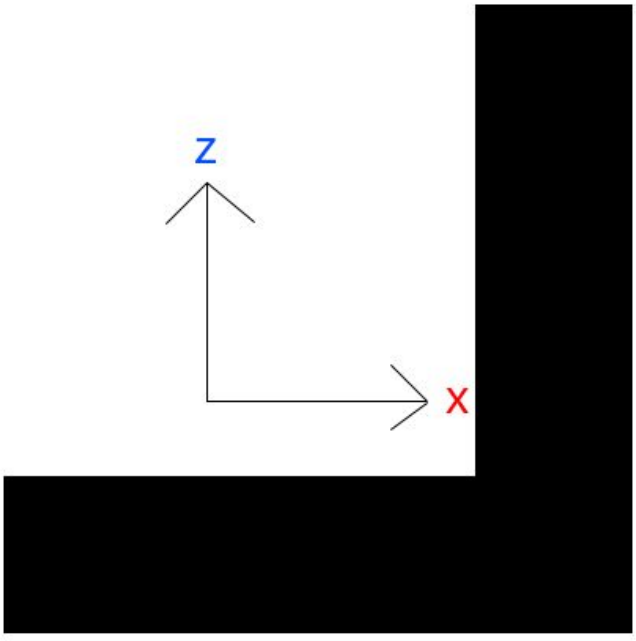
# Wall

The walls consist of 1 on the z scale, and both the x and y scale vary. The position of the prefab for the x value will need to be adjusted correctly to adjust itself perfectly with the corners, or needs to be nested with a parent prefab by using a 1x1x1 center.



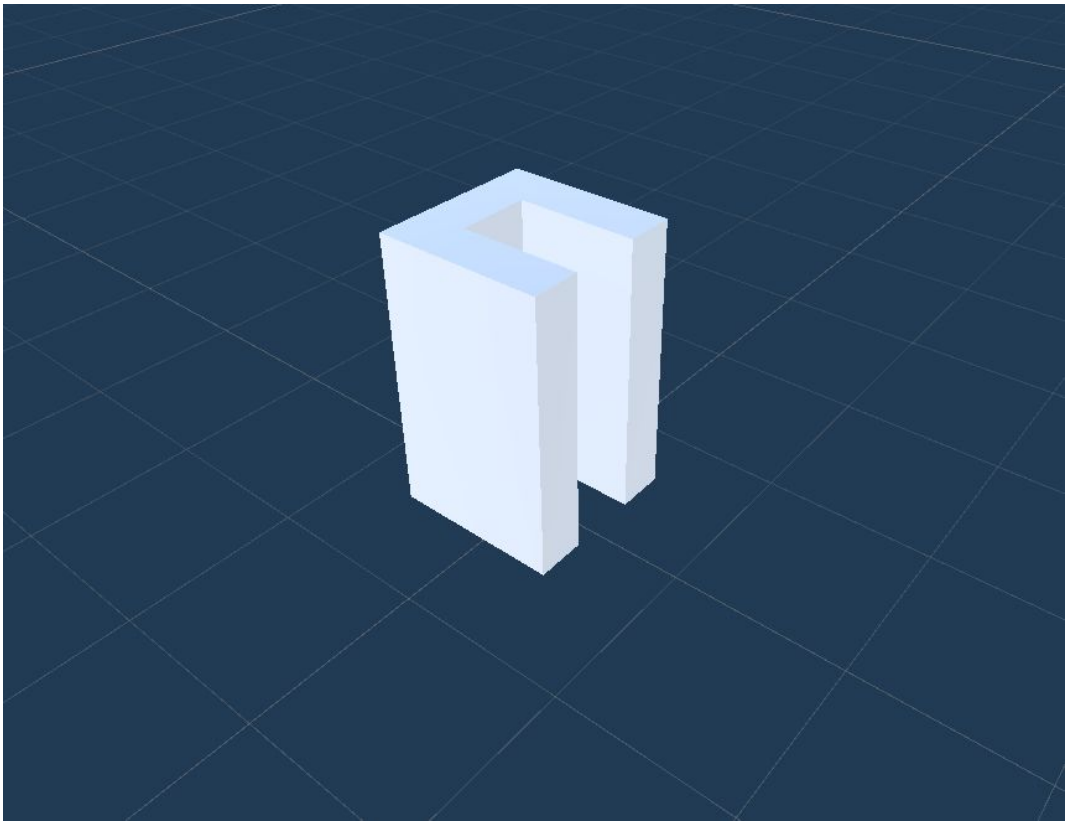
# Outer Corner

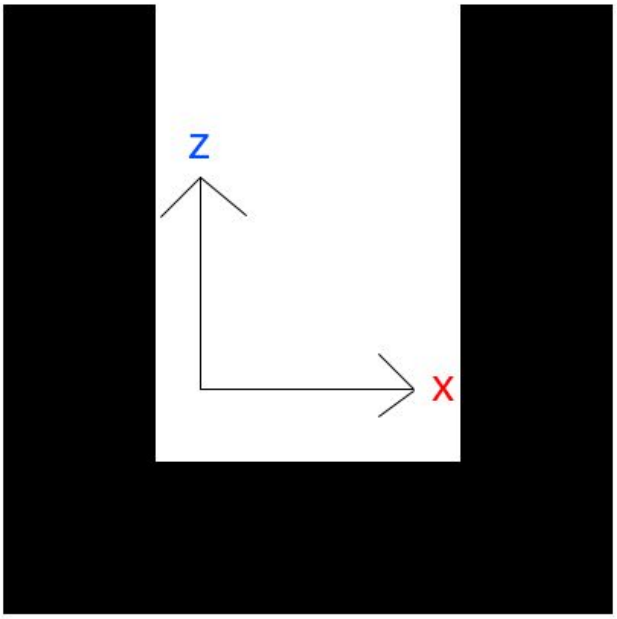





Outer Corners need to be set as seen above for the rotations to be correct.

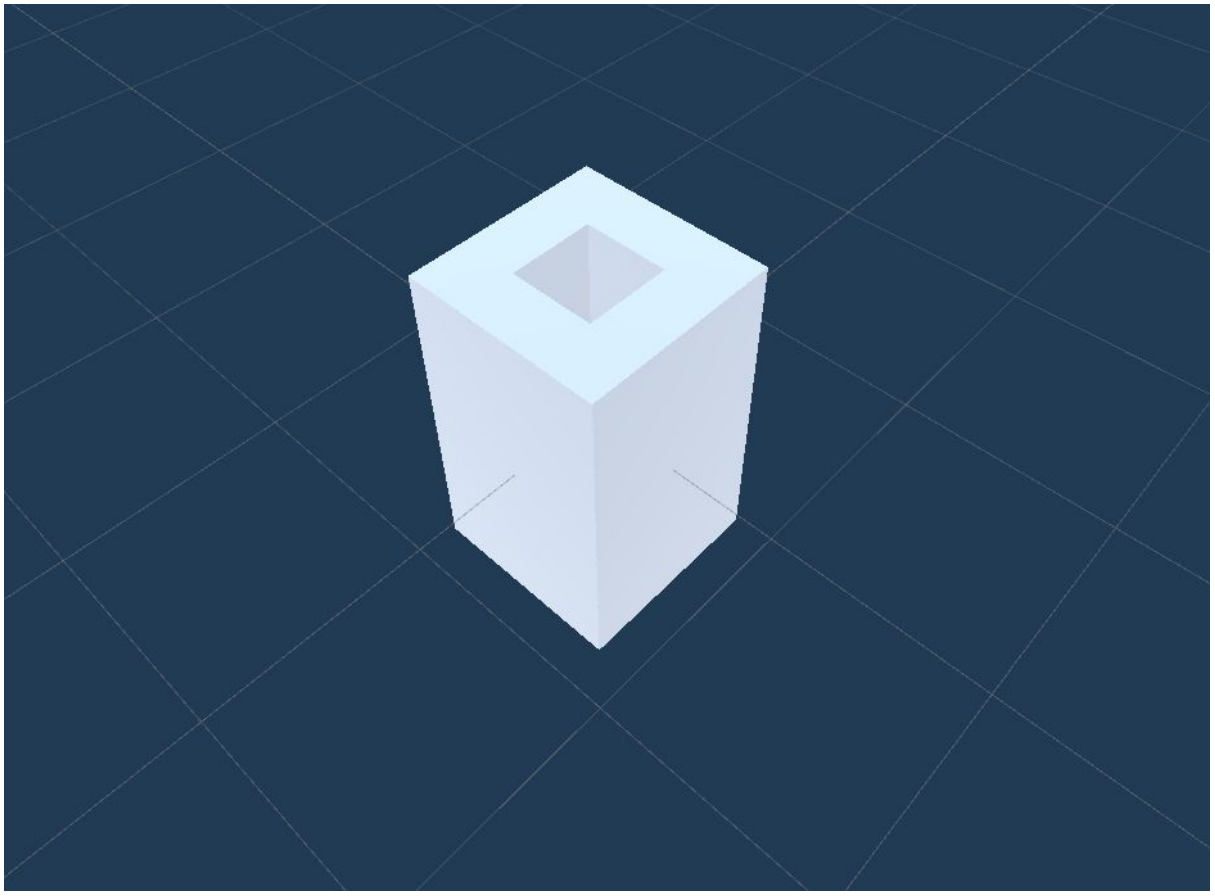
# U Corner



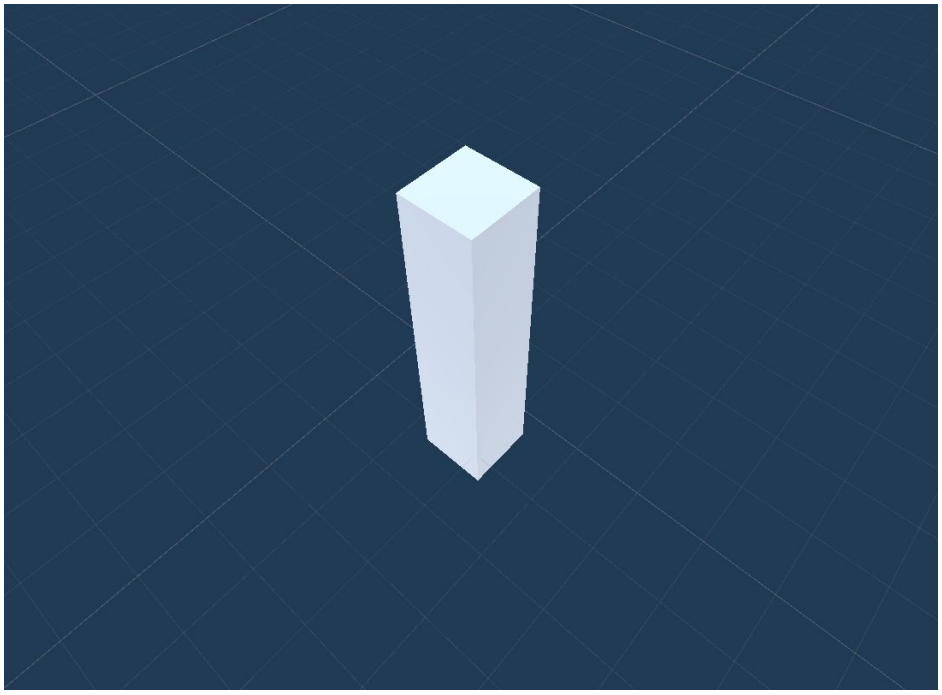



UCorners need to be set as seen above for the rotations to be correct.

# Square Corner



# Inner Corner



Inner Corners can be used as seen above for the interior walls to replace the normal walls, or using a pillar instead, to cover gaps that the walls might have when put together on a 90 degree angle due to it's model, or to detail further.

# How Does it work?

The algorithm of the Dungeon is a tile/grid system. The algorithm starts by creating off the rooms, then the flooring for the rooms, and then the corridors and its flooring, and finally the walls and corners. At the end the Content is added too.

## 1. Rooms

A random number of rooms is firstly chosen by picking a random number between the minimum amount of rooms to the max amount of rooms. After this a random number also is picked for the numbers of width and height by using both the minimum and maximum of those variables. A random position is then chosen in between the area specified, and then the algorithm checks if the room fits there without overlapping into another room.

The size of every room is randomly chosen, and the width and height apply to every room besides the boss room which has its own variables that can be changed.

## 2. Corridors

Corridors are created right after the rooms, firstly the distance between all the rooms is calculated, and the closest rooms are then chosen and the rooms connect to them, they also check if the rooms are already connected to each other.

A check is then made to see if all the rooms are connected, and there is a way to get into every room. If they are not all connected, divide all connected rooms into a group, and all the others into other groups. Then check all the groups and which two of the rooms, one from each group are closer together, then connect those two, doing this until there's a path to every room.

## 3. Walls

Once all the rooms and corridors are created, the algorithm will check for where walls are needed. It will check for walls on all sides, and walls in the same tile, in case that happens a wall is put as a child of the other. Meaning that sometimes two walls are needed in the same tile due to there being flooring on both sides, and so by adding it to the tile system would just delete the other wall, by making it a child it does not delete it.

## 4. Corners

Outer Corners prefabs are added in the right angle to the place they are needed in, they have a right angle triangle and two walls, almost in the shape of a L.

U Corners Prefabs are added too to the right places, they have three walls and two right angles in the shape of an U.

After adding these to every rotation and side needed, it is again double checked if there are any Outer Corners Prefabs and U corners needed, since the creating and outer corners for the first time run all in the same for loop, requiring a double check to ensure everything was well placed.

## 5. Content

The corners of each room around found and torches are placed there if there isn't any corridor replacing where the walls are usually. The torches are also added to corridors, every x number which can be edited.

Using the corners the adjacent sides of x and z can be placed with content including static objects or destroyables which depend on the rate, and they are chosen randomly between each other.