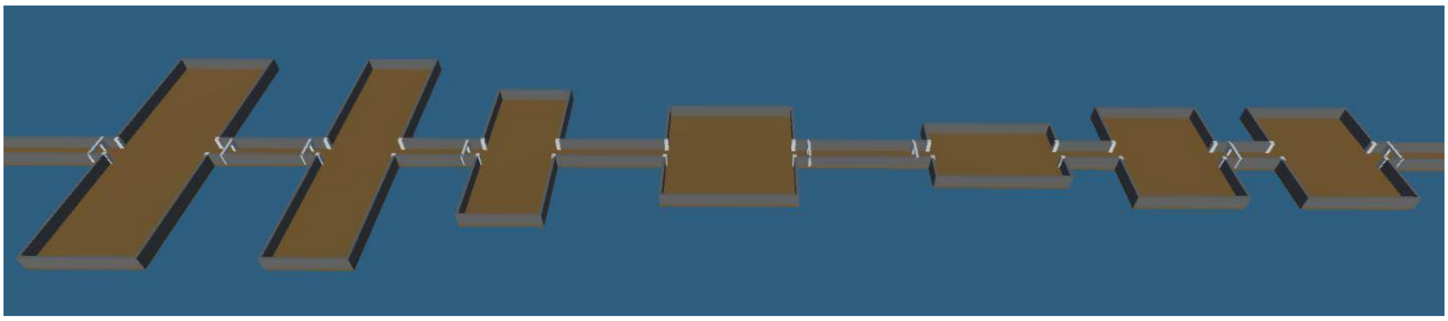


EASY DG

USER MANUAL AND SETUP GUIDE

RANDOMISED, POWERFUL AND EASY TO USE DUNGEON GENERATOR

FOR 3D BASED UNITY PROJECTS.

c00pala on Unity Connect

c00p.b.coding@gmail.com

CONTENTS

INTRO

GREETINGS

Greetings, Friend.

I'd like to firstly thank you for taking the time to check out my Dungeon Generator system, Easy DG.

I wrote this system because my love of randomisation means I'm constantly needing to write new level generators to prototype and play with and also because I'm no stranger to the asset store, where I've invested in quite a few different dungeon generators over the years. While I've never found a generator asset that I regret purchasing - they're usually always great assets / systems that achieve their purpose well - they always get to a point where they need to be customised to achieve the 'as perfect as possible' level layout. I knew I needed my own solution, something that could easily fill whatever gap I needed it to and ideally save me hours adjusting and re-writing systems to suit whatever new purpose I had in mind.

I knew what I was looking for in a dungeon / level generator and the many different roles they needed to fill so I set out to try and write a flexible, yet easy to use and implement solution that could be quickly added to any project. It needed to retain its simplicity and randomisation but most importantly, still offer a greater level of control over the end result. Thus, Easy DG was born.

Easy DG has gone through many iterations before arriving at its current point. Although it's not an uncommon idea or method for generating dungeons, I believe I've refined it and built in a wide range of customisation and automation to make Easy DG one of the easiest to use and most flexible dungeon generators available. With many different generation options available to customise how you'd like your dungeons to look and flow, as well as many automated functions; Easy DG is designed to give you complete control over the randomised layout with minimal setup and fuss.

Don't like the end result of a generation, want to adjust it or build your own from scratch easily? Perhaps you like the end result but also want to build more refined, smaller layouts, or personalised rooms and levels? With the Manual Edit Mode option, Easy DG allows you to generate a blank dungeon for editing, or you can go into your dungeon after generation and adjust it as you see fit; deleting and adding areas of your choosing before finally regenerating the level prefabs. This simple and powerful built-in editor will allow you a significant amount of customisation over your dungeons and will help achieve a quick and personalised workflow for your level design.

Once again, regardless of whether you have purchased this asset or not, thank you for taking the time to give it a look. I hope that if you do or have decided to purchase Easy DG, you find it easy to use and that it meets your requirements nicely. Any feedback, comments, complaints or bug reports can be directed to the support email at the end of this document - I'm only one person but I'll endeavour to reply to all emails as quickly as I can.

I'd also like to take a moment to thank my wonderful fiancé. Without her never ending support and assistance, in the good times and the hard, projects like these could never happen.

Thank you again and the best of luck in all of your gaming and developing adventures!

- c00pala.

USING THIS MANUAL

If you're just hoping to jump right in without any of the nitty gritty details, then scroll down to the Quick Setup section to get started. This section also contains some easy to copy example settings to demonstrate different level layouts.

For more detailed information on the system and its operations, such as how it works, the scripts, control variables and functions, etc. see the User Manual itself in the next section.

If you're experiencing trouble getting the generator system to work, or think you've found a bug, please see Troubleshooting and Support for known issues or bugs, FAQ and support contact details.

Outside of this document, there's the Easy DG Videos playlist and Unity Connect project page for any further information posted regarding Easy DG.

Please also see the Change Log text document included with the Easy DG asset files for any recent changes or updates.

USER MANUAL

HOW THE GENERATOR WORKS

Easy DG’s primary purpose is for 3D, Unity projects to randomly generate Dungeons (parent objects containing collections of spawned prefabs) using the values provided by the user. You can do this in the Editor or at runtime / during the game via script. You can also utilise Manual Edit Mode to further adjust any Dungeon prefabs, or create personalised dungeons from scratch using the same generation parameters to populate them with doors, walls, etc.

Be aware! Play mode can interrupt the generator and will cause Manual Edit Mode to break. Please read the relevant sections under Known Bugs / Potential Issues.

THE GENERATION PROCESS

At its most basic level, the generation process can be explained as follows:

- Starting from the centre of the world the generator will run lines of tiles (a prefab) in a random direction that will run for a set number of tiles.
- Using a random chance variable, the script will occasionally change the direction of these lines.
- While generating these lines, the generator will create rooms by getting a random value for length and width and increasing the width of the line in tiles for the determined length – e.g. a room might be 4 tiles wide for 5 tiles long.
- When the end of a line is reached (i.e. it reaches its max number of tiles) it will either return to world centre or continue on from the end of the last line.
- Once the maximum number of tiles has been reached, the generator performs a series of checks over the world to determine which tiles have neighbouring tiles and where they are.
- It then uses the results of these checks to generate walls, doors and corners, prefabs chosen by the user, randomly from arrays.
- *Walls will only spawn on the edge of tiles that don’t neighbour another tile, and will spawn on the edge with no neighbour. Each tile can have 4 walls.*
- *Doors will spawn based on chance and only spawn on tiles that neighbour two other tiles. Those neighbours must sit either above and below, or left and right. Doors will not spawn in open rooms, intersecting or corner hallways. Doors can be set to only spawn at the ends of hallways, to prevent a long hallway having multiple needless doors.*
- *Corners will spawn in the diagonal corner of tiles that don’t have a diagonal neighbour (e.g. the top left corner of a tile would be -X, +Z, any tile in this direction would be the top-left diagonal neighbour) but only when that ‘corner’ actually has tiles on either side. To continue with the top-left example, if a tile has no top-left diagonal neighbour, it would also need to have a neighbour both above and to its left before a corner will spawn, this prevents corner objects spawning constantly in hallways. Corners are to give a sense of depth to the Dungeon and help the doorways and entrance points in a room stand out. They, like walls and doors above, can be prevented from spawning.*
- Once generation is complete, you will have a parent object, named according to the name chosen in the inspector pane and numbered according to an auto incrementing, resettable counter, that contains your dungeon. This object can be saved as a prefab, or adjusted using the Manual Edit Mode.

THE CORE SCRIPTS

- DUNGEONGENERATOR.CS

Usage: Place script on an empty GameObject called ‘DungeonGenerator’ that’s present only once within your scene.

Info: This script does all the work. It runs the generation and the manual editor, both of which you’ll generally control by selecting the DungeonGenerator object in your scene to view the control variables in the Inspector.

The editor, when active, is controlled using the keyboard in the scene view but is toggled and controlled from this script. See Using the Manual Edit Mode for more information.

- DUNGEONGENERATOREDITOR.CS

Usage: Can be present in any folder in the project, does not need to be added to an object.

Info: Responsible for presenting the buttons for the generator, for both the generator itself and the manual editor. Controls the inspector pane when the DungeonGenerator object is selected.

- TILESCRIPT.CS

Usage: Place script on your base tile prefab.

Info: Stores tile info while generation runs. You shouldn’t need to edit or interact with this script but it’s addition to your tile prefab is essential for the system to function.

- TILEEDITOR.CS

Usage: Can be present in any folder in the project, does not need to be added to an object.

Info: Provides the ‘Copy to Editor’ functionality in the inspector pane when tiles are selected in the editor.

- SETUPWIZARD.CS

Usage: Must be present in a folder titled ‘Editor’ for the menu item to work.

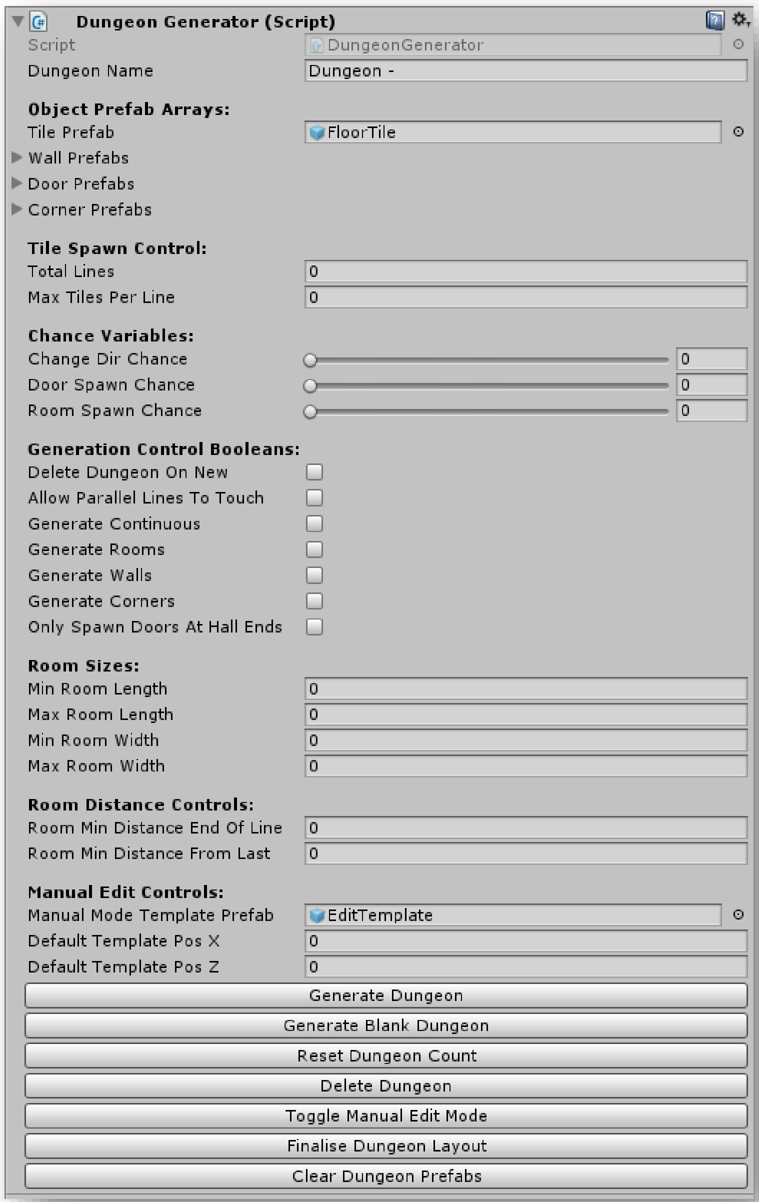
Info: Provides the Setup Wizard functionality.

DUNGEON GENERATOR - CONTROL (PUBLIC) VARIABLES

For the purposes of this explanation, the variables will be documented in the format they appear in the inspector. The variables covered here are the ones required for the system to function and are generally set by the user in the inspector.

THE INSPECTOR WINDOW.

When using Easy DG, the options (variables) will be presented in the inspector as follows:



Dungeon Generator (Script)

Script	DungeonGenerator
Dungeon Name	Dungeon -
Object Prefab Arrays:	
Tile Prefab	FloorTile
Wall Prefabs	
Door Prefabs	
Corner Prefabs	
Tile Spawn Control:	
Total Lines	0
Max Tiles Per Line	0
Chance Variables:	
Change Dir Chance	0
Door Spawn Chance	0
Room Spawn Chance	0
Generation Control Booleans:	
Delete Dungeon On New	☐
Allow Parallel Lines To Touch	☐
Generate Continuous	☐
Generate Rooms	☐
Generate Walls	☐
Generate Corners	☐
Only Spawn Doors At Hall Ends	☐
Room Sizes:	
Min Room Length	0
Max Room Length	0
Min Room Width	0
Max Room Width	0
Room Distance Controls:	
Room Min Distance End Of Line	0
Room Min Distance From Last	0
Manual Edit Controls:	
Manual Mode Template Prefab	EditTemplate
Default Template Pos X	0
Default Template Pos Z	0

The following pages will list these variables in the order they appear above.

DUNGEON NAME

- **Dungeon Name:**
  Type: String
  This variable determines how your Dungeons are named when they're generated. They will have an auto incremented value attached to the end of them that represents how many Dungeons have been generated.

  E.g. Setting this value to 'Dungeon #' will give the Dungeons the name 'Dungeon #1', 'Dungeon #2' and so on. The auto incrementing counter can be reset using the 'Reset Dungeon Count' button on the inspector.

OBJECT PREFAB ARRAYS

- **Tile Prefab:**
  Type: GameObject (Prefab)
  This is the assigned tile prefab. This is not an array, like the other prefab variables, due to the way the generator functions. If greater flexibility or randomisation is required for tiles, you can always use a spawning prefab that generates a new tile on game start. This prefab is your base, floor tile and represents 1 segment in the dungeon. It needs to be square, as in the X and Z size/scale of the prefab need to be identical. The 'TileScript' script is attached to this object.

- **Wall/Door/Corner Prefabs:**
  Type: GameObject (array, prefabs)
  These prefabs are your walls, doors and corners. They will spawn randomly from the arrays during generation. They can generally be any X and Z scale you like, their sizes will be automatically calculated and adjusted for when the generator runs. You need to have at least 1 prefab in these arrays for the generator to function.

TILE SPAWN CONTROL

- **Total Lines:**
  Type: Integer
  This integer controls how many lines are generated when the generator runs. Each line will contain many tiles and rooms and will 'snake' around the scene, changing direction randomly via a chance variable. Setting this variable to higher amounts will drastically increase the number of tiles in your level and, depending on the 'Generate Continuous' variable, can cause the dungeon to 'cluster / clump' up, or stretch out for long distances. This variable must be set to at least 1 for the editor to function.

- **Max Tiles Per Lines:**
  Type: Integer
  This variable controls how many lines will be generated in each line. Setting this value to higher amounts will increase the number of tiles in your level but less drastically than adjusting the number of lines would. Combining this with the number of lines variable will greatly impact the overall layout of your level.

CHANCE VARIABLES

- **Change Dir Chance:**
  Type: Integer (range: 0 - 100)
  This integer controls how often the lines will change direction when generation is running. Setting this variable to lower values will cause a more grid like and long reaching dungeon. Alternatively, higher values will cause a more compact and maze-like dungeon.

- **Door Spawn Chance:**
  Type: Integer (range: 0 - 100)
  This integer will control the frequency at which doors will appear in hallways. Setting this variable to 0 will prevent any doors from spawning. Alternatively, setting it to 100 will cause doors to appear in every location identified as a possible location for a door.

- **Room Spawn Chance:**
  Type: Integer (range: 0 - 100)
  This integer will control the frequency at which rooms will be generated while lines are being run. Rooms are simply wider sections of hallways. Setting this to lower values will create more hallways. Alternatively, higher values will create a more blocky, clustered dungeon.

GENERATION CONTROL BOOLEANS

- **Delete Dungeon On New:**
  If true, when you click the 'Generate' button to create a new Dungeon, this bool will ensure the current Dungeon is deleted from the scene. Setting this to false will cause a new Dungeon to be generated while retaining the current one.

- **Allow Parallel Lines To Touch:**
  If this bool is set to true, when the generator runs, lines will be allowed to place tiles directly alongside neighbouring, parallel tiles causing hallways to expand in width. Alternatively, setting this to false will cause the generator to try and enforce a 1 tile gap between hallways – it isn't always possible, also rooms and intersecting (as in not-parallel) lines are allowed to touch. This variable, combined with 'Generate Continuous' and 'Change Dir Chance' will drastically change the end layout of your Dungeon. Setting it to true will cause rooms to have far less of an impact on the level layout, leaving you with a dungeon containing large, non-square rooms and very few hallways.

- **Generate Rooms:**
  If this bool is set to true, rooms will be generated when lines are run. False will prevent any rooms from being generated. You can also achieve this by setting 'Room Spawn Chance' to 0. Having no rooms, depending on the setting of 'Allow Parallel Lines To Touch', will generally create a more sprawling, hallway heavy Dungeon.

- **Generate Walls / Corners:**
  If this bool is false, no walls or corners will be generated in your Dungeon. Doors can be controlled via 'Door Spawn Chance'. This bool is useful when making use of the Manual Edit Mode, as it allows you to only generate the layout without finalising the Dungeon.

- **Only Spawn Doors At Hall Ends:**
  If this bool is set to true, doors will only spawn on the first and last tile of a hallway and nowhere in between. This bool is useful for when you might not want multiple doorways to appear in one hall. When true, it still abides by the 'Door Spawn Chance' value, so you'll usually use a value of 100 when this bool is true. Adversely, setting this bool to false will cause a heavy reliance on the chance variable to control the number of doors placed within a Dungeon as, for example, if the door spawn chance is 100 and this variable is false, hallways will be completely filled with doors.

ROOM SIZES

- **Min Room Length:**
  Type: Integer
  This variable will determine the minimum possible 'length' for a room when generation decides to start one. For the purposes of this variable, the length of a room is how many tiles long it is within a line. It is recommended to set this to a value > 1 if you wish to have rooms spawning.

- **Max Room Length:**
  Type: Integer
  Same as above but controls the maximum possible length.

- **Min Room Width:**
  Type: Integer
  This variable will determine this minimum possible 'width' for a room when generation decides to start one. The width of a room is the number of tiles spawned on either side of a line when a room is being generated. If a room's width is 1, it is actually 3 tiles wide (1 tile per side + the centre tile).

- **Max Room Width:**
  Type: Integer
  Same as above but controls the maximum possible width.

ROOM DISTANCE CONTROLS

- **Room Min Distance End Of Line:**
  Type: Integer
  This variable will determine how far from the start and end of each line a room is allowed to start spawning. Setting this to higher values will cause a hallway or dead-end to appear often in the Dungeon. Alternatively, lower values will cause T-Intersection type ends, or dead-end rooms.

- **Room Min Distance From Last:**
  Type: Integer
  This variable controls how far from the end of the last room a new one is allowed to start. Higher values will cause longer hallways and fewer rooms per line.

MANUAL EDIT CONTROLS

- **Manual Mode Template Prefab:**
  Type: GameObject (prefab)
  This variable is for storing the object that's used to create the template grid in Manual Edit Mode. This is usually a copy of the main tile prefab with a transparent material but it can be anything. It needs to have any collider removed for the system to function.

- **Default Template Pos X / Z:**
  Type: Float
  These variables will determine the placement of the first template when Manual Edit Mode is toggled on. This can be copied from existing tiles by selecting them in the editor and clicking the 'Copy this pos to Edit Mode' button.

DUNGEON GENERATOR - CONTROL BUTTONS

- **Generate Dungeon:**
  Will generate a new dungeon, using the values provided to DungeonGenerator in the inspector.

- **Generate Blank Dungeon:**
  Will generate a new, empty, dungeon. This will be the parent object only and can be edited using Manual Edit Mode.

- **Reset Dungeon Count:**
  Will reset the auto incrementing counter added to the names of Dungeons.

- **Delete Dungeon:**
  Deletes from the scene, the most recently generated Dungeon, including any tiles added using Manual Edit Mode.

- **Toggle Manual Edit Mode:**
  Will turn on or off Manual Edit Mode and place the starting template object in the scene at the location specified in the 'Default Template Pos' variables.

- **Finalise Dungeon Layout:**
  Will turn off Manual Edit Mode and populate the Dungeon with walls, doors and corners, using the values specified in the inspector. It still abides by the above variables and if 'Generate Walls' is off, for example, then this button won't spawn any walls.

- **Clear Dungeon Prefabs:**
  Will remove all wall, door and corner prefabs from the most recently generated Dungeon.

- **Import Dungeon**
  Imports and old Dungeon and sets it as current, so it can be edited and regenerated as needed.
  See Importing an old Dungeon for further information.

DUNGEON GENERATOR - CONTROL (PUBLIC) FUNCTIONS

These functions are the public functions contained within the DungeonGenerator script. You should never need to call the majority of these but they've been listed to assist with extending or customising the script.

Please see Advanced Functions for more information on running the generator via script.

- `public void DeleteDungeon()`
  Does as it says, deletes the most recently generated Dungeon. This function is run when clicking the 'Delete Dungeon' button.
- `public void GenerateBlankDungeon()`
  Will generate a blank Dungeon. Same as clicking the 'Generate Blank Dungeon' button.
- `public void ResetDungeonCount()`
  Resets the Dungeon counter used in naming. Same as clicking the 'Reset Dungeon Counter' button.
- `public void SpawnTile(Vector3 spawnPos)`
  Will instantiate a new tile at the given position. If generating tiles from another script, use this function to create it, as it adds it to the required Lists and sets the TileItem properties. Needs a current Dungeon in the scene set as dungeonParent or function will fail.
- `public bool CheckPosClear(Vector3 checkPos)`
  Will check the provided position to determine if any object with a collider is present at it using a Raycast. Returns true if the position is clear.
- `public void PluginCheck()`
  Checks for and runs Plugins prior to generation. Same as clicking the 'Generate Dungeon' button.
- `public void GenDungeon()`
  Generates a new Dungeon using the currently set control variables. Will not include Plugins.
- `public void ChangeManualTemplate(int length, int width, Vector3 direction)`
  Used to update the position or size of the Manual Build Mode template grid. You shouldn't ever have a need to call this from a script. This function, as well as the following 2 (SpawnAtTemplate and DeleteAtTemplates), are included only to assist with extending the editor functionality, should a user decide to do so.
- `public void SpawnAtTemplates()`
  Will cause tiles to be spawned at the position of any Manual Edit Mode template grid tiles currently in the scene.
- `public void DeleteAtTemplates()`
  The opposite of above, will delete any tiles in the same position as a template grid tile.
- `public void FinaliseDungeon()`
  Will finalise the current Dungeon. Same as clicking the 'Finalise Dungeon Layout' button.
- `public void ClearDungeon()`
  Will delete all wall, door and corner prefabs attached to the current Dungeon. Same as clicking the 'Clear Dungeon Prefabs' button.
- `public void ImportDungeon()`
  Function responsible for importing old / non-current Dungeons. See Importing an old Dungeon for more information about this process.

-
  ```
  public void EasyGen(int lines, int tiles, int dirChance, int doorChance, int
  roomChance, bool allowPar, bool genCon, bool genRooms, bool genWalls, bool
  genCorners, bool hallEnds, int minRoomL, int maxRoomL, int minRoomW, int
  maxRoomW, int distEOL, int distFromLast)
  ```
  Function to call generation of a new Dungeon at runtime. See Controlling the Generator via Script for more detailed info on this function. Will include any Plugins.

USING MANUAL EDIT MODE

BEFORE USING EDIT MODE

Before you use Manual Edit Mode, it's worth noting that it can be a bit fiddly and there's a few things to be aware of. Also, please read this section in the FAQ!

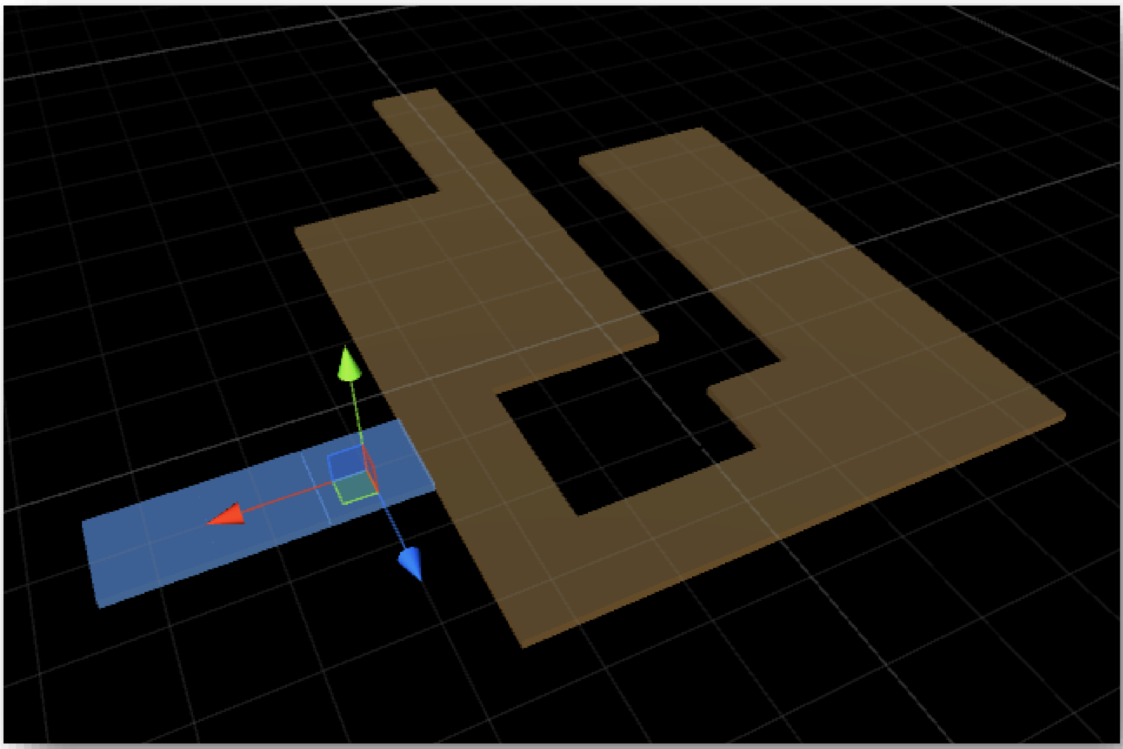
- When using edit mode, moving and rotating the scene view camera via the mouse, as well as interaction with objects/windows in the scene view and project (such as selecting objects), is disabled. To move the view, the DungeonGenerator object will follow the template grid around the game world and you should be able select it in the hierarchy, reselect the scene view and focus on it (default key 'F') to move the camera to the current build point, where you can zoom the camera (which does still work) to a more comfortable viewpoint. Toggling edit mode off should resume all functionality.

- When you toggle edit mode on, you need to select the scene view by clicking on it before the controls will work.

TURNING EDIT MODE ON

Before you can use edit mode, you must have a current Dungeon generated and present in the scene, this can be a layout with or without prefabs or it can be a blank Dungeon with no contents.

To turn Manual Edit Mode on or off, select the DungeonGenerator object and click the 'Toggle Manual Edit Mode' button in the inspector. A 1x1 grid of template tiles (template prefab), will appear in your game world, using the default X and Z values defined in 'Default Template Pos X/Z'. Once toggled on, click on the scene view to start controlling edit mode.



CONTROLLING THE TEMPLATE GRID

By default, the grid can be controlled using the following keys on the keyboard:

- **Move grid:**
  Left/Right/Up/Down arrow keys.

- **Increase/Decrease grid length:**
  1 / 2 keys (Not the numpad).

- **Increase/Decrease grid width:**
  - / + keys (Not the numpad).

- **Add tiles:**
  Space bar.

- **Delete tiles:**
  Delete key.

These controls can be redefined by editing the KeyCode values in the ‘DungeonGeneratorEditor’ script.

ONCE EDITING IS COMPLETE

Turn off Manual Edit Mode. You can then set your wall, door and corner variables and click the ‘Finalise Dungeon Layout’ button to populate the Dungeon with the required prefabs. If further editing is required, you can simply re-enter edit mode and continue placing tiles, the next time you finalise the layout, these changes will be factored in. If required, you can also clear the prefabs via the ‘Clear Dungeon Prefabs’ button.

Tip: Pressing the ‘Finalise Dungeon Layout’ button multiple times will adjust the positioning and randomisation of your doors, allowing you to find an ideal layout.

IMPORTING AN OLD DUNGEON

All the editing and generation functions require that the Dungeon being edited is set as the current, or most recently generated, Dungeon. Once you generate a new Dungeon, this variable is cleared and replaced with the new one and if you happen to generate another Dungeon, or save one as a prefab and wish to return to editing it later, you first need to import it so the system knows its layout and can alter it accordingly. Thankfully, this process has been automated and can be achieved with minimal hassle.

To import an old Dungeon:

- Ensure the scene is free of any ‘current’, or generated, Dungeons.
- Add the old Dungeon to the scene (if not present already) at position 0,0,0 and rename the parent object to ‘Import’ (without quotes, capitalise the ‘i’).
- Click the ‘Import’ button in the DungeonGenerator’s inspector.

The old Dungeon’s layout will then be rebuilt as a new Dungeon that you can edit and use normally.

Be aware that this function requires the tiles to be unchanged from how they were when the dungeon was generated, if they’ve been replaced with other prefabs, or had their TileScript component removed, this function will fail.

ADVANCED FUNCTIONS

CONTROLLING THE GENERATOR VIA SCRIPT

You can generate levels at runtime by calling the EasyGen() function within the DungeonGenerator script.

This function has a significant number of required parameters, these are as follows:

`int lines` – Sets the 'Total Lines' variable pre-spawn.
`int tiles` – Sets the 'Max Tiles Per Line' variable pre-spawn.
`int dirChance` – Sets the 'Change Dir Chance' variable pre-spawn.
`int doorChance` – Sets the 'Door Spawn Chance' variable pre-spawn.
`int roomChance` – Sets the 'Room Spawn Chance' variable pre-spawn.
`bool allowPar` – Sets the 'Allow Parallel Lines to Touch' variable pre-spawn.
`bool genCon` – Sets the 'Generate Continuous' variable pre-spawn.
`bool genRooms` – Sets the 'Generate Rooms' variable pre-spawn.
`bool genWalls` – Sets the 'Generate Walls' variable pre-spawn.
`bool genCorners` – Sets the 'Generate Corners' variable pre-spawn.
`bool hallEnds` – Sets the 'Only Spawn Doors At Hall Ends' variable pre-spawn.
`int minRoomL` – Sets the 'Min Room Length' variable pre-spawn.
`int maxRoomL` – Sets the 'Max Room Length' variable pre-spawn.
`int minRoomW` – Sets the 'Min Room Width' variable pre-spawn.
`int maxRoomW` – Sets the 'Max Room Width' variable pre-spawn.
`int distEOL` – Sets the 'Room Min Distance End Of Line' variable pre-spawn.
`int distFromLast` – Sets the 'Room Min Distance From Last' variable pre-spawn.

For detailed info on these variables see Control Variables.

An example of this function used in a script would be:

```
    void Start()
    {
        DungeonGenerator dunGenerator = GetComponent<DungeonGenerator>();

       dunGenerator.EasyGen(2, 50, 15, 20, 25, false, false, true, true,
        true, true, 3, 6, 2, 4, 3, 5);
    }
```

This would generate a new Dungeon at game start.
For some examples of settings, see Example Generation Values.

USING THE GENERATOR IN A 2D PROJECT

Currently, this isn't possible using the script as-is.

Depending on demand / time, this will hopefully be a future addition to this system, possibly via the use of another version of the DungeonGenerator script that uses 2D instead.

Users are welcome to attempt to re-write the DungeonGenerator script to use Vector2 positions instead of Vector3 and to update the move functions and directions (such as Vector2.up instead of Vector3.forward) to try and achieve this.

USING PLUGINS

Plugins are easy to add / remove scripts that extend the functionality of the base generator.

As each plugin contains different functions, documentation for each plugin as well as a change log detailing any changes made to them can be found separate to this manual, in the Documentation folder included with the Plugins themselves.

To use a Plugin, simply add the desired script to the DungeonGenerator object before generating your Dungeon. Plugin functions are controlled separately on each script and integrate in with the core generator, so even when using Plugins, you can still use the same buttons and interface to generate Dungeons as you would normally.

QUICK SETUP

There are three methods you can use to set up the DungeonGenerator object.

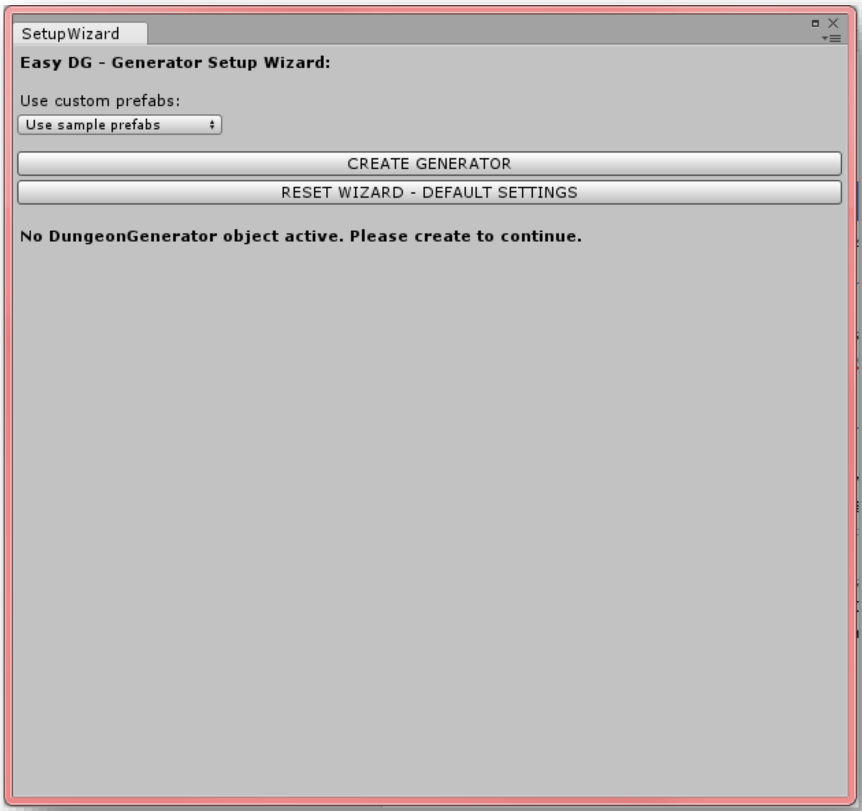
METHOD 1 – DRAG AND DROP THE PREFAB

If you have imported the sample scene and assets, the easiest way to get started is to simply load the sample scene, or drag and drop the DungeonGenerator prefab into your scene. By default, it's found under: Easy DG\Sample Scene\Sample Prefabs\Resources.

METHOD 2 – THE SETUP WIZARD

As of version 1.1, Easy DG now includes an easy setup wizard to create the DungeonGenerator object – no more drag and dropping, or manually assigning variables! You can use this to create the generator and get it ready, or even and create your first dungeon using a series of pre-made template options.

To open the setup wizard, import the Easy DG asset, with or without the sample prefabs (you'll need them only if you want to use the pre-made assets). Once imported, a menu item should appear in the Unity toolbar for 'Easy DG'. To open the Wizard, simply open this menu and select the 'Easy DG Setup Wizard', you'll be given the following window:



Easy DG - Generator Setup Wizard:

Use custom prefabs: Use sample prefabs

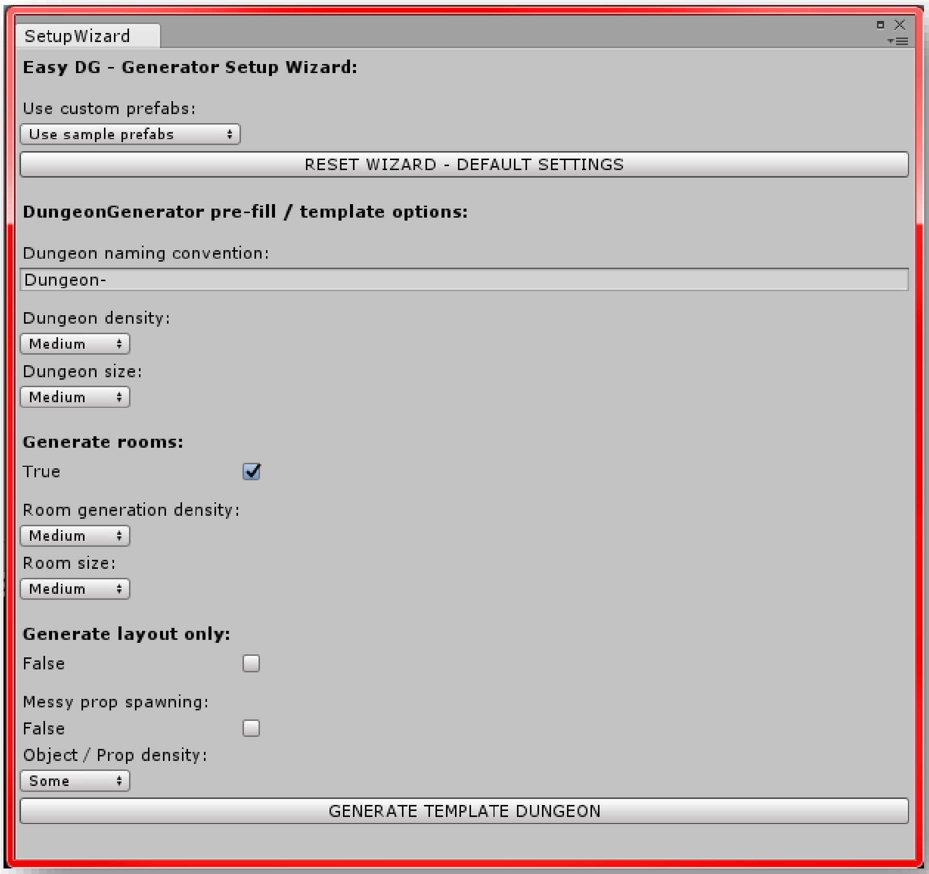
No DungeonGenerator object active. Please create to continue.

By default, the wizard will attempt to setup and use generator with the sample prefabs and settings. To change this, simply change the 'Use custom prefabs' field from 'Use sample prefabs' to 'Use custom prefabs'.

If the sample prefabs are not present and you try and use the wizard, you'll receive an error in the console.

To set up a new DungeonGenerator object, simply click 'Create Generator'. If one is present already, the system will alert you and prompt before continuing, as creating a new DungeonGenerator will overwrite any currently in the scene.

Creating a new DungeonGenerator will update the Wizard with some new options:



SetupWizard

Easy DG - Generator Setup Wizard:

Use custom prefabs:
Use sample prefabs

RESET WIZARD - DEFAULT SETTINGS

DungeonGenerator pre-fill / template options:

Dungeon naming convention:
Dungeon-

Dungeon density:
Medium

Dungeon size:
Medium

Generate rooms:
True ☑

Room generation density:
Medium

Room size:
Medium

Generate layout only:
False ☐

Messy prop spawning:
False ☐

Object / Prop density:
Some

GENERATE TEMPLATE DUNGEON

These options are template settings you can use to streamline the creation of your first Dungeon. Tooltips have been provided for each setting to explain their usage, hovering over the field name will cause these to appear.

Set these as desired and click the 'Generate Template Dungeon' button to generate a dungeon with the chosen setting. To reset all settings and delete the current DungeonGenerator, click the 'Reset Wizard' button.

If any plugins are present, these will need to be added to the object and configured before clicking the 'Generate Template Dungeon' button. When the template Dungeon is generated, these plugin settings will be applied as normal.

METHOD 3 – SET UP THE GENERATOR MANUALLY

It is assumed that you have some prefabs already for the floor, walls, doors and corners. If needed, sample prefabs / assets are included with the sample scene.

The system is designed to be implemented with minimal hassle and it shouldn't take more than 10 - 15 minutes to get it working but depending on your prefabs, some setup or changes may be required. Please pay careful attention to 'Step 2 – Configuring the Prefabs' to ensure your Dungeons are generated correctly.

STEP 1 - SETTING UP THE SCENE

- Starting with a new 3D Unity project, import the Easy DG asset.
- Place a copy of the **DungeonGenerator** prefab (under Core | Prefabs) into the scene.
  OR Create a new Empty GameObject in the scene, name it 'DungeonGenerator' and add the **DungenGenerator.cs** script to it as a component.

STEP 2 – CONFIGURING THE PREFABS

- The system has 5 main prefab 'types'. See the prefabs in the Sample Scene folder for examples.

- **Tiles:** The base floor prefab.
  Setup: Attach TileScript.cs to this prefab as a component.
  Requirements:
  - You can only have 1 prefab for this object, as the system does not choose this prefab from an array.
  - This prefab must be square: X and Z scale / size need to be identical.
  - This prefab must have a Collider and Mesh Renderer attached to the parent object.

- **Walls:** Prefab used for the Dungeon walls.
  Requirements:
  - Can have multiple, will be chosen randomly from an array.
  - While it can be any size, this prefab must be aligned horizontally when at a default rotation of (0,0,0). To help determine the correct rotation, see How do I determine the best rotation for my prefabs? In the FAQ.

- **Doors:** Prefab used for the Dungeon doors.
  Requirements:
  - Can have multiple, will be chosen randomly from an array.
  - While it can be any size, this prefab must be aligned horizontally when at a default rotation of (0,0,0). To help determine the correct rotation, see How do I determine the best rotation for my prefabs? In the FAQ.

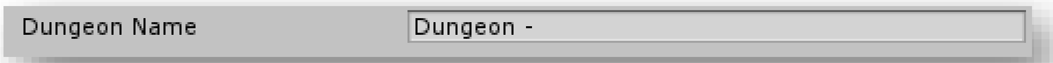
- **Corners:** Prefab used for the Dungeon's corners.
  Usually a 'Pillar' or 'Column', some type of edging for your walls when they turn a corner.
  No requirements.

  NOTE: All three of the Wall, Door and Corner prefab types MUST have a MeshRender attached to either the parent, or at least one child object. If no MeshRenderers are found, the generator will fail.
- **Manual Mode Template:** Prefab used to build the Manual Build Mode template grid.
  Requirements:
  - Must NOT have a collider attached.
  - A recommendation is to use a prefab similar in size to the base tile prefab. Recommend also using a

different, transparent material to make it easy to identify when in use. Otherwise, this prefab can be any size or shape.
- A premade template prefab has been provided under Core | Prefabs, called EditorTemplate. It uses the EditorTemplate material in the Materials subfolder.

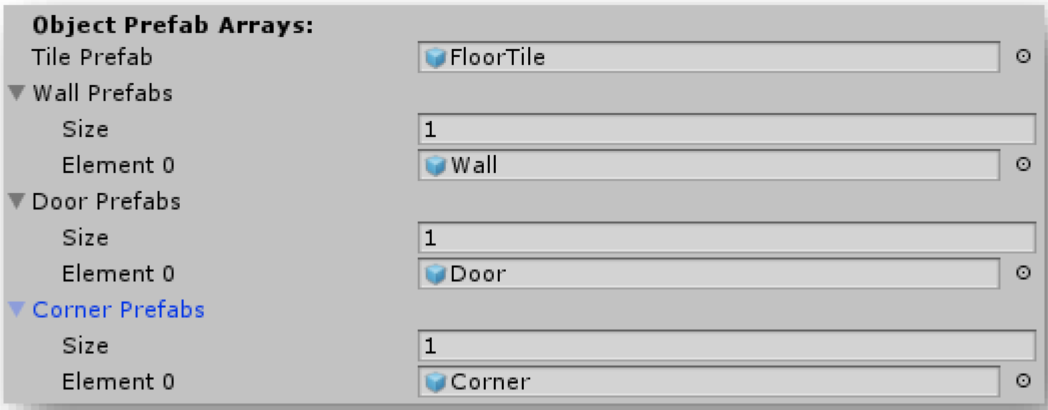
STEP 3 - ASSIGNING VALUES.

- Select the DungeonGenerator object in the scene hierarchy.
- In the inspector pane for the object, you should see a list of the control variables (see below under Example Generation Values for some good values to start with).
- Set a Dungeon Name:



Dungeon Name	Dungeon -

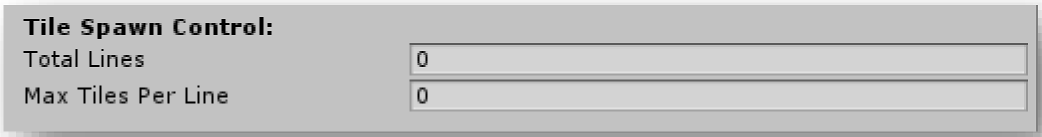
- Add your prefabs to the Object Prefab Arrays:



Object Prefab Arrays:

Tile Prefab	FloorTile
▼ Wall Prefabs	
Size	1
Element 0	Wall
▼ Door Prefabs	
Size	1
Element 0	Door
▼ Corner Prefabs	
Size	1
Element 0	Corner

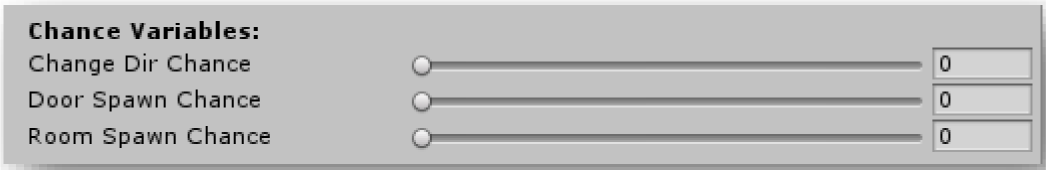
- Provide values to Tile Spawn Control:



Tile Spawn Control:

Total Lines	0
Max Tiles Per Line	0

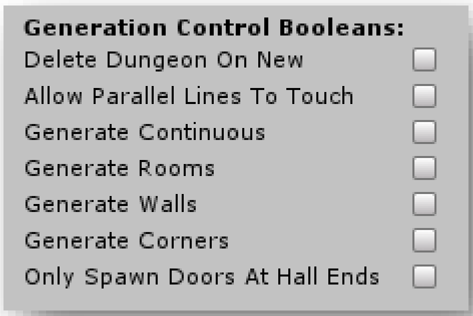
- Set the Chance Variables:



Chance Variables:

Change Dir Chance	0
Door Spawn Chance	0
Room Spawn Chance	0

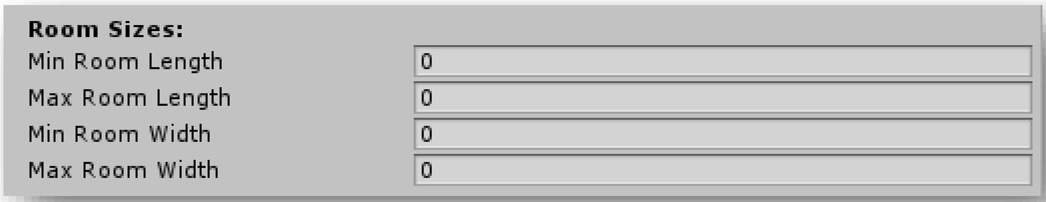
- Set the desired Generation Control Booleans:



Generation Control Booleans:

Delete Dungeon On New	☐
Allow Parallel Lines To Touch	☐
Generate Continuous	☐
Generate Rooms	☐
Generate Walls	☐
Generate Corners	☐
Only Spawn Doors At Hall Ends	☐

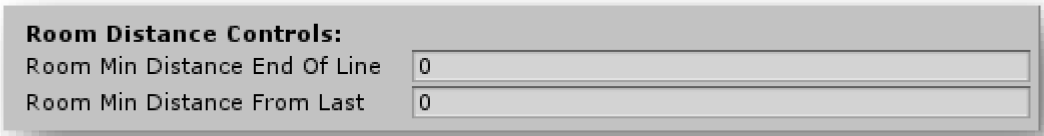
- Set the desired Room Sizes:



Room Sizes:

Min Room Length	0
Max Room Length	0
Min Room Width	0
Max Room Width	0

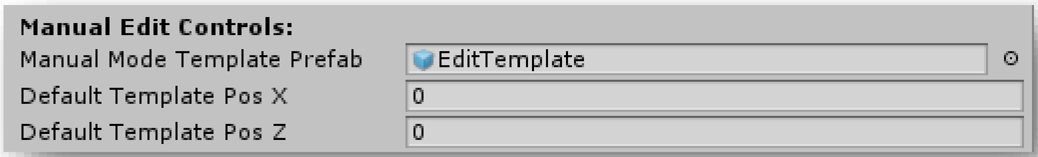
- Set the desired Room Distance Controls:



Room Distance Controls:

Room Min Distance End Of Line	0
Room Min Distance From Last	0

- Provide a template prefab and set the default start point of edit mode to Manual Edit Controls:

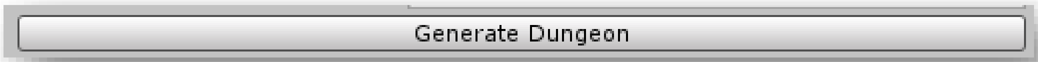


Manual Edit Controls:

Manual Mode Template Prefab	EditTemplate
Default Template Pos X	0
Default Template Pos Z	0

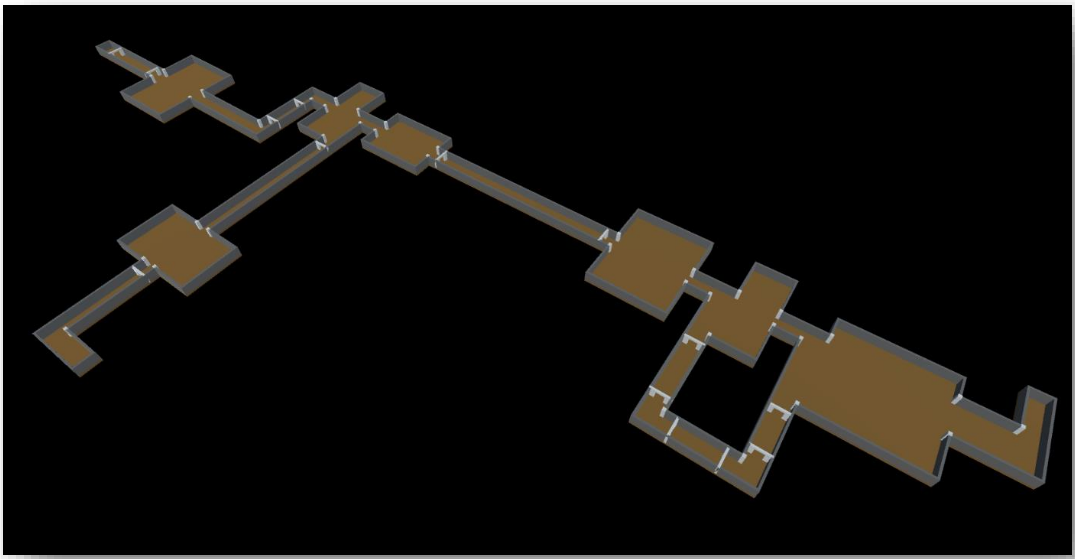
STEP 4 – RUN THE GENERATOR

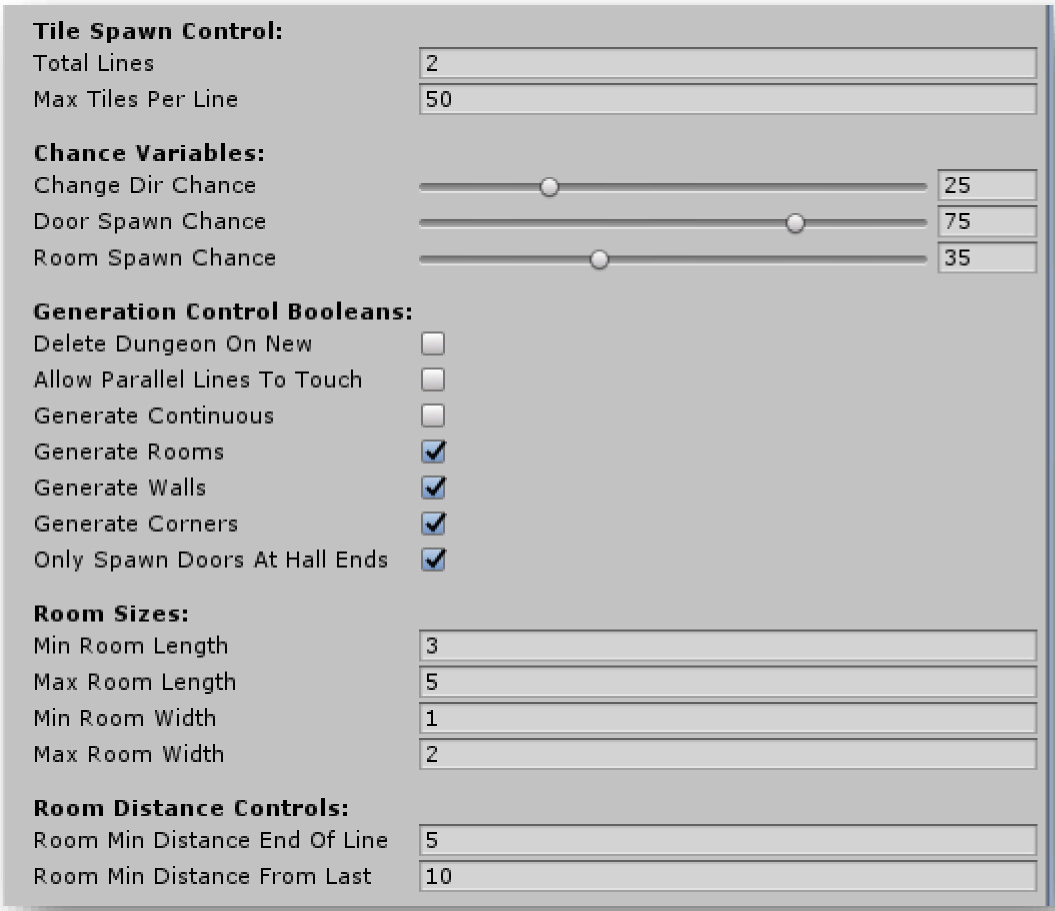
- Now that the above values are set, click the Generate Dungeon button in the inspector to generate your first Dungeon:



EXAMPLE GENERATION VALUES

LONG HALLS, GROUPED ROOMS





Tile Spawn Control:

Total Lines	2
Max Tiles Per Line	50

Chance Variables:

Change Dir Chance	25
Door Spawn Chance	75
Room Spawn Chance	35

Generation Control Booleans:

Delete Dungeon On New	☐
Allow Parallel Lines To Touch	☐
Generate Continuous	☐
Generate Rooms	✓
Generate Walls	✓
Generate Corners	✓
Only Spawn Doors At Hall Ends	✓

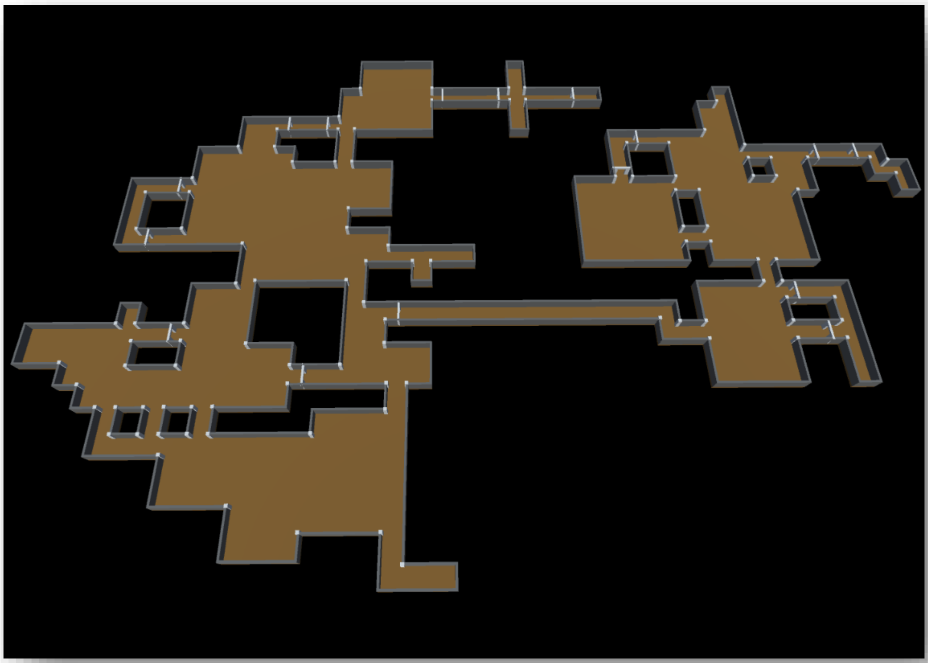
Room Sizes:

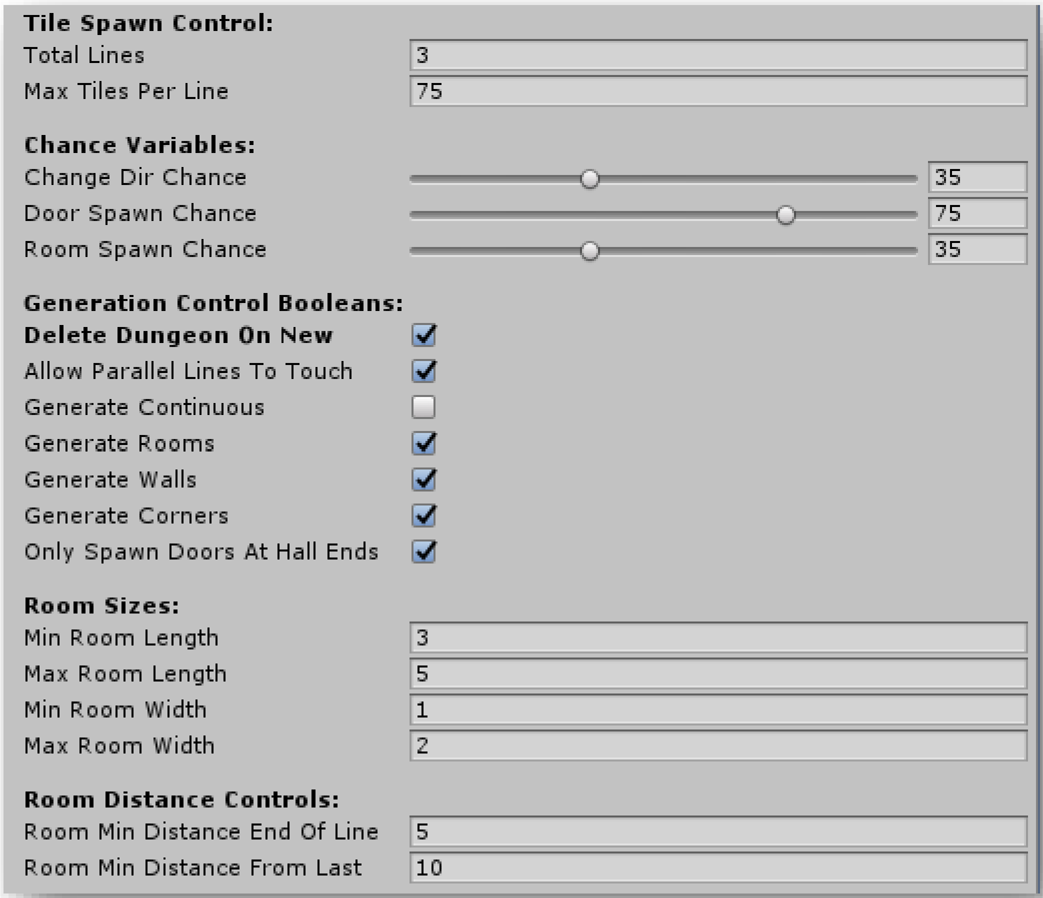
Min Room Length	3
Max Room Length	5
Min Room Width	1
Max Room Width	2

Room Distance Controls:

Room Min Distance End Of Line	5
Room Min Distance From Last	10

COMPACT, ERRATIC ROOMS





Tile Spawn Control:

Total Lines	3
Max Tiles Per Line	75

Chance Variables:

Change Dir Chance	35
Door Spawn Chance	75
Room Spawn Chance	35

Generation Control Booleans:

Delete Dungeon On New	✓
Allow Parallel Lines To Touch	✓
Generate Continuous	
Generate Rooms	✓
Generate Walls	✓
Generate Corners	✓
Only Spawn Doors At Hall Ends	✓

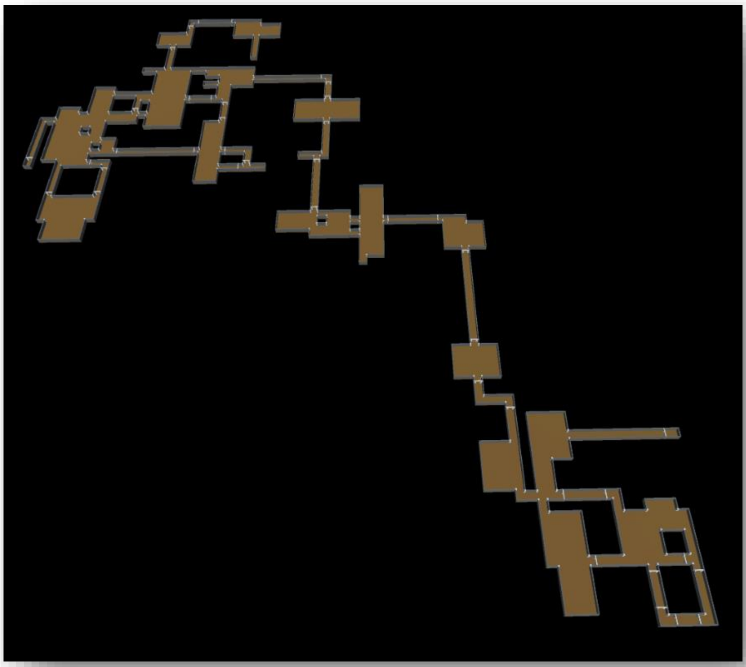
Room Sizes:

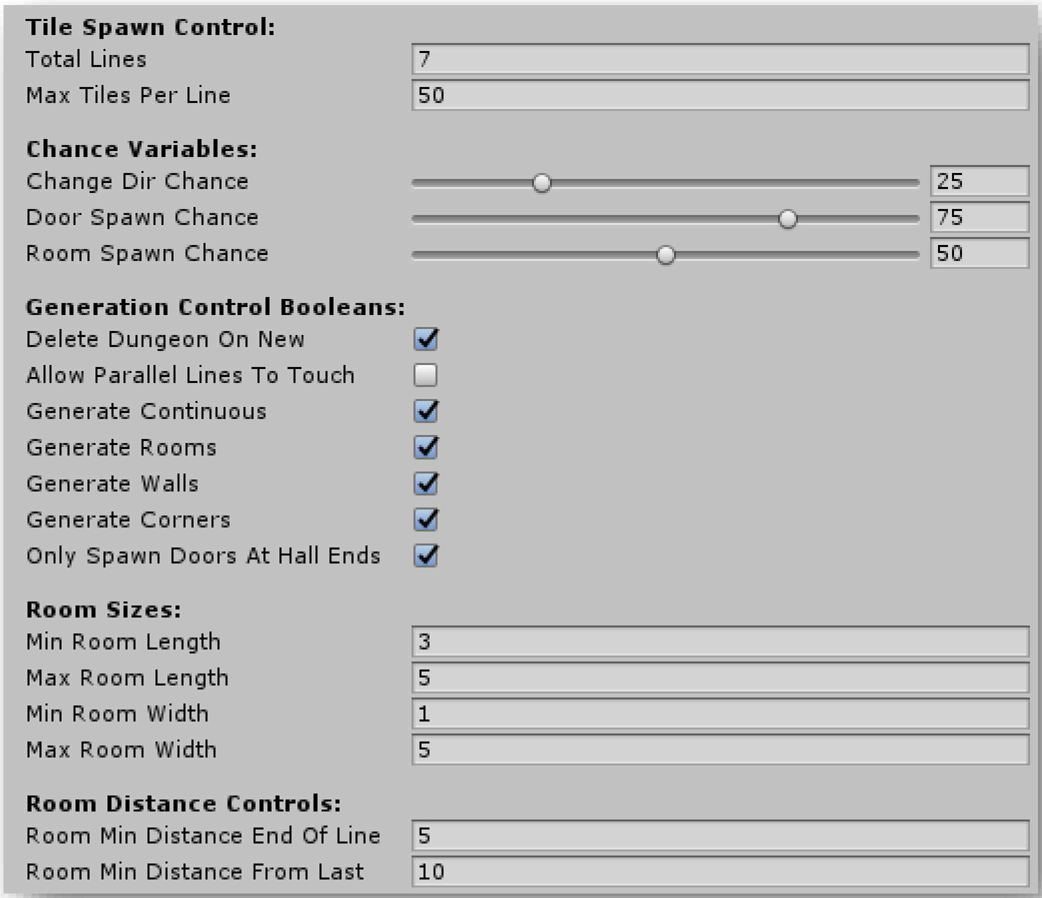
Min Room Length	3
Max Room Length	5
Min Room Width	1
Max Room Width	2

Room Distance Controls:

Room Min Distance End Of Line	5
Room Min Distance From Last	10

SPRAWLING, LONG HALLS, MAZE LIKE





Tile Spawn Control:

Total Lines	7
Max Tiles Per Line	50

Chance Variables:

Change Dir Chance	25
Door Spawn Chance	75
Room Spawn Chance	50

Generation Control Booleans:

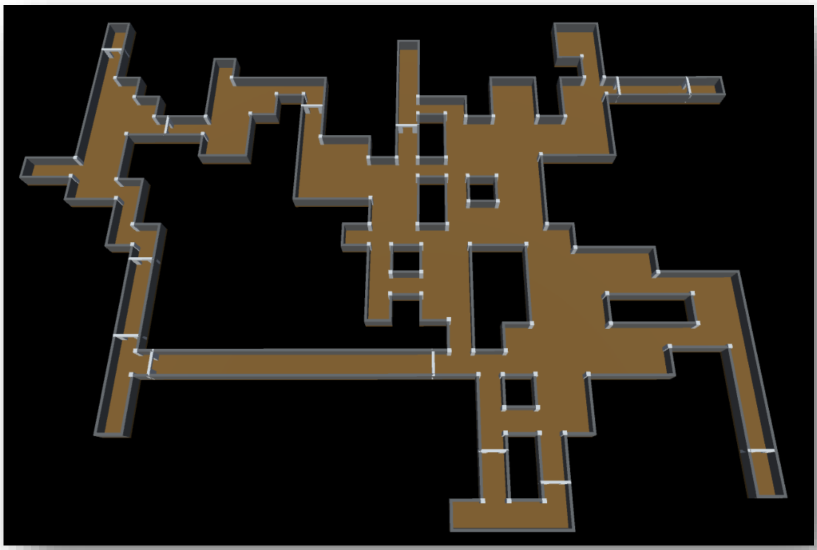
Delete Dungeon On New	✓
Allow Parallel Lines To Touch	☐
Generate Continuous	✓
Generate Rooms	✓
Generate Walls	✓
Generate Corners	✓
Only Spawn Doors At Hall Ends	✓

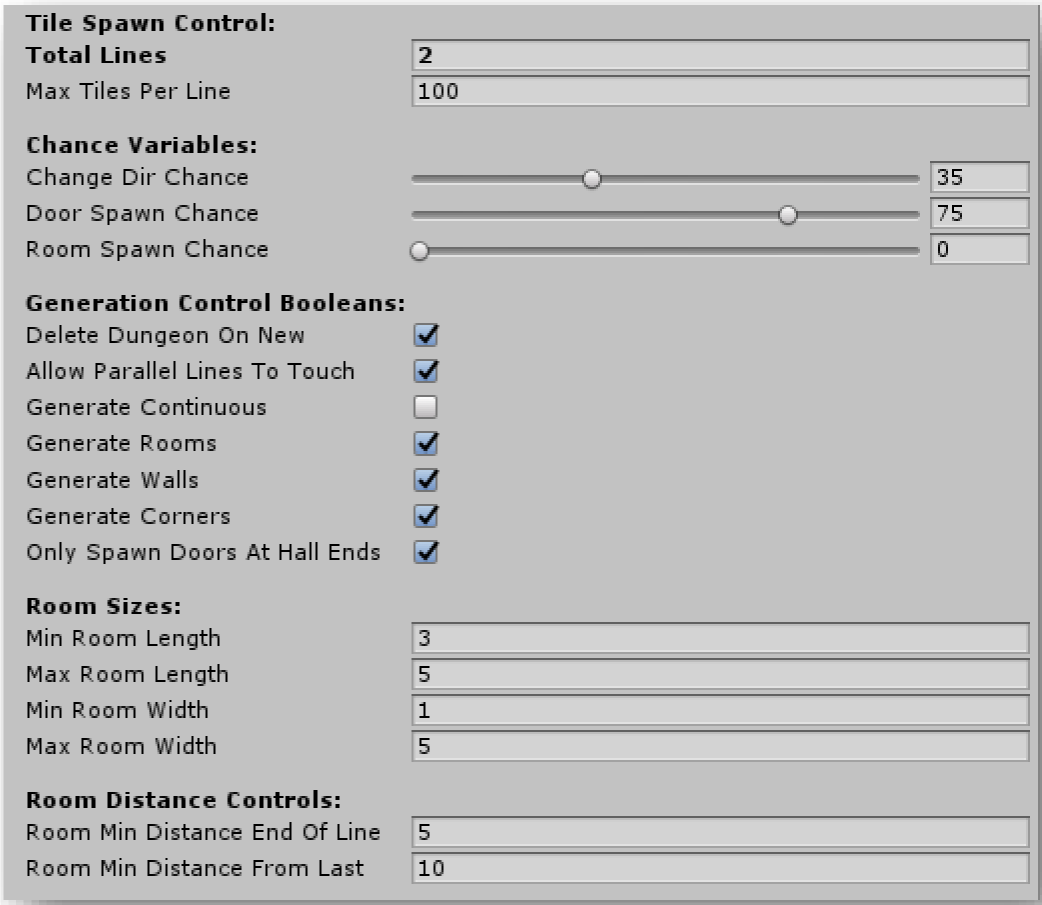
Room Sizes:

Min Room Length	3
Max Room Length	5
Min Room Width	1
Max Room Width	5

Room Distance Controls:

Room Min Distance End Of Line	5
Room Min Distance From Last	10



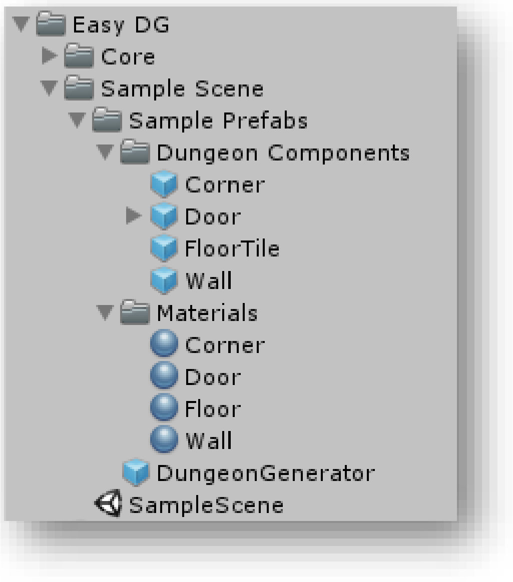


Tile Spawn Control:	
Total Lines	2
Max Tiles Per Line	100
Chance Variables:	
Change Dir Chance	35
Door Spawn Chance	75
Room Spawn Chance	0
Generation Control Booleans:	
Delete Dungeon On New	✓
Allow Parallel Lines To Touch	✓
Generate Continuous	☐
Generate Rooms	✓
Generate Walls	✓
Generate Corners	✓
Only Spawn Doors At Hall Ends	✓
Room Sizes:	
Min Room Length	3
Max Room Length	5
Min Room Width	1
Max Room Width	5
Room Distance Controls:	
Room Min Distance End Of Line	5
Room Min Distance From Last	10

THE INCLUDED SAMPLE SCENE AND PREFABS

CONTENTS

Included with the Easy DG Core system, under the ‘Sample Scene’ folder are some prefabs, materials and a sample scene:



- Easy DG
  - Core
  - Sample Scene
    - Sample Prefabs
      - Dungeon Components
        - Corner
        - Door
        - FloorTile
        - Wall
      - Materials
        - Corner
        - Door
        - Floor
        - Wall
      - DungeonGenerator
    - SampleScene

THE SAMPLE SCENE

If you load the included scene file ‘SampleScene.unity’ you’ll be presented with a scene containing a main camera / light and a pre-filled DungeonGenerator object.

To use this sample scene:

- Click on the DungeonGenerator object in the Hierarchy window to select it.
- In the Inspector window for the Dungeon Generator object, scroll down and click the ‘Generate Dungeon’ button.
- Navigate and view the Dungeon via the Scene view camera
- Alternatively, position the main camera, or import a character controller and play.

THE DUNGEONGENERATOR PREFAB

Included with the sample scene is a prefab named ‘DungeonGenerator’.

This is simply an empty GameObject with the DungeonGenerator script attached; the variables are pre-filled with example values and the sample prefabs so it can be used instantly. It’s the same generator as used in the sample scene and functions exactly the same as creating this object yourself manually as detailed in Quick Setup.

THE PREFABS AND MATERIALS

These are simply some basic prefabs to demonstrate the functionality of the generator. They aren’t required for the system to run but they can be used as a reference and allow you to easily test values.

TROUBLESHOOTING AND SUPPORT

FAQ

CAN I SAVE A DUNGEON FOR EDITING LATER?

Yes. Generate the Dungeon and save the whole Dungeon object as a prefab. When you want to edit it later, you can import it using the process detailed in Importing an old Dungeon.

CAN I IMPORT AND ATTACH DUNGEON PREFABS TO ANOTHER DUNGEON?

You can achieve this if you position the old Dungeon prefab you wish to attach in the game world, move / attach its child objects to the new Dungeon, then import the new Dungeon again using the Import Dungeon method, it should correctly register the 'piece' as part of the new Dungeon layout.

USING MY OWN / CUSTOM PREFABS IS NOT WORKING.

Aside from the requirements detailed in Configuring Prefabs there are several requirements for your custom models (meshes) that must be met before they can be used with Easy DG.

The meshes used by your prefabs MUST have a centred Pivot Point and a default rotation of (0, 0, 0) when imported into Unity. Should you encounter any issues with Dungeon generation when using custom prefabs, please consult the 'Cutom Asset Troubleshooting - Pivot Point Fix' document located in the Documentation folder within the Easy DG asset folder.

HOW DO I DETERMINE THE BEST ROTATION FOR MY PREFABS?

As specified in Configuring Prefabs your walls, doors and to some extent, corners, need to be aligned horizontally in your game world when at a default rotation of (0,0,0). If unsure, the easiest way to tell what this should look like is to create a new Cube object in the scene and scale it to (x:10, y:5, z:1). For the purposes of the generator, the cube's (assume it's a wall) X value is the 'length' of the wall and the Z value the 'width'. For positioning, it's usually the Z value that matters.

The cube, as it's rotated currently, is considered to be horizontal. Ensure your walls, doors and corners match the rotation of the example cube when they're at (0,0,0).

WILL THIS SCRIPT WORK IN A 2D PROJECT?

Unfortunately, not at this point. As detailed in Advanced Functions this will hopefully make it in one day but as it is currently, using this system in a 2D Unity project is not supported.

DO I USE THIS SYSTEM IN THE EDITOR, OR AT RUNTIME?

Either. The default editor functions will allow you to test generation values and see results more easily. Using the editor, you can create, edit and save Dungeons for use in-game. You also use the editor to customise Dungeons and create smaller prefabs.

If you need to, you can call the EasyGen() function and provide the values as parameters to have the generator run in-game and generate a Dungeon based on the values provided. You won't be able to edit this dungeon further with Easy DG but you can still interact with it via scripting. The Generator was deliberately designed to avoid using Layers and Tags so you can use these freely to perform further checks on the Dungeon's child objects. See Controlling the Generator via Script for details.

WHAT IS THE END RESULT OF GENERATION? WHAT WILL I BE WORKING WITH?

Once the generator completes, you're left with a parent GameObject in the scene containing child objects for all the tiles, walls, doors and corners created during generation.

WHAT IF I CAN'T ACHIEVE THE PERFECT DUNGEON? CAN I PERSONALISE PARTS OR ROOMS?

Yes. Once a Dungeon is generated, either blank or with layout / prefabs, you can edit it by using Manual Edit Mode, see Using Manual Edit Mode for more information.

WHY DOES MY DUNGEON TAKE A WHILE TO GENERATE? / IS EASY DG FREEZING?

The system should never freeze, all possible infinite loops contain break point counters that prevent bad values from leading to a crash. If something is wrong with the generator values however, the generator / Unity can go unresponsive for a moment, until the built-in break points are hit or it finishes generating.

If you try to generate a Dungeon with bad values or values that are too high, it can sometimes take a while for the generator to find new locations to place tiles – e.g. if the start position of the new line is heavily covered with tiles and it has to search long distances to find new spawn locations – and Unity will appear to freeze. The system is designed to cope with bad values and if any problems occur, errors should be logged to the console to assist with troubleshooting. Note that plugins that add additional tiles will also cause the generation time to increase.

If the generator / Unity doesn't appear to be responding, try waiting to see if the generation exits (at default this should happen after about 20 seconds). If it doesn't resume after a few minutes, please try restarting Unity and adjusting the generator values (start by lowering the lines, max tiles or direction change chance variables), you can also try re-creating the DungeonGenerator object. If all this fails, please log a bug report.

If you're generating very large dungeons and need to adjust the break point to prevent the generator exiting, alter the 'preventInfLoopMax' variable declared at the start of GenDungeon() in the DungeonGenerator script – default value: 5000. Higher values will cause the system to try for longer before exiting. As a rough estimate, for every 5000 you add to 'preventInfLoopMax', expect it to take about 20 seconds longer before the generator gives up. Ideally, level generation shouldn't ever take more than 5 to 10 seconds for large dungeons.

KNOWN BUGS / POTENTIAL ISSUES

- **Entering Play mode stops some functions from working:** If you generate a Dungeon and then enter Play mode from the Unity Editor, some functions (e.g. 'Finalise Dungeon') will stop working for the current Dungeon. This is due to the way the tiles are tracked and I haven't been able to work around this. To avoid, ensure you save any generated Dungeons as a prefab before you launch play mode. If you need to continue editing a Dungeon it will need to be reimported.

- **Entering Play mode while Manual Edit Mode is turned on causes it to break:** Similar to above, if Manual Edit Mode is turned on (i.e. template tiles are present in the scene) when you enter Play mode from the Unity Editor it will stop functioning correctly. To avoid this, ensure Manual Edit Mode is off before entering Play mode. The Dungeon will need to be reimported to resume editing.

- **Further issues:** Any exceptions generated by the scripts should be caught and records of any errors or issues logged to the debug console. If something isn't working, check for any logs that might indicate why the generator is failing.

BUG REPORTS / SUPPORT

If something isn't working as expected, please contact / log a bug report to: c00p.b.coding@gmail.com.

As much as possible, bug reports should contain:

- A description of the issue, what you were doing or trying to do when it occurred / was first noticed.
- Screenshots of your hierarchy and DungeonGenerator inspector, copies of any errors or logs.
- Snippets of any altered code or code not included in the original system that might be related.
- Can the problem be reproduced? If so, how?
- Contact details.

CREDITS / ACKNOWLEDGEMENTS

I’d like to take a moment to thank the following people for their support:

- As mentioned already in the greeting, my fiancé, for everything. For putting up with my constant barrage of ideas and opinions and for keeping me focused - no small task. :CakeEmoji:

- All my wonderful friends from Discord with particular thanks to, Bilz, Gabe, Ahmadred and Nat. You guys are all awesome and I’ll never tire of our coding sessions and chats.

- My talented teammate, Jason. Sorry dude, totally still working on our projects in between all this, I swear!

- Thomas, for all the advice, suggestions and feedback. Seriously, thanks for letting me pester you, without your help, a lot of the new stuff in 1.1 wouldn’t even be a thing.

- Hasnain, for feedback and bug reporting that led to some significant fixes in the latest release (1.2).

- Yilmaz Kiymaz (@VoxelBoy), for his SetPivot script from the Unify Wiki that I’ve bundled with this asset.
  (http://wiki.unity3d.com/index.php?title=SetPivot)

- All the Ladz for putting up with my rants and lectures while my brain is buzzing. Zetz, Donut, Luckzor, Eminence, DLord, thanks. Don’t ever stop being you guys. <3

- And last but not least, all my viewers, supporters and customers.

  To anyone that’s bought an asset from me, watched one of my videos, sent me an email, said hi to me in a game, checked me out on Connect, stopped past the Discord, anything, whatever! Everything I do, I do for you.