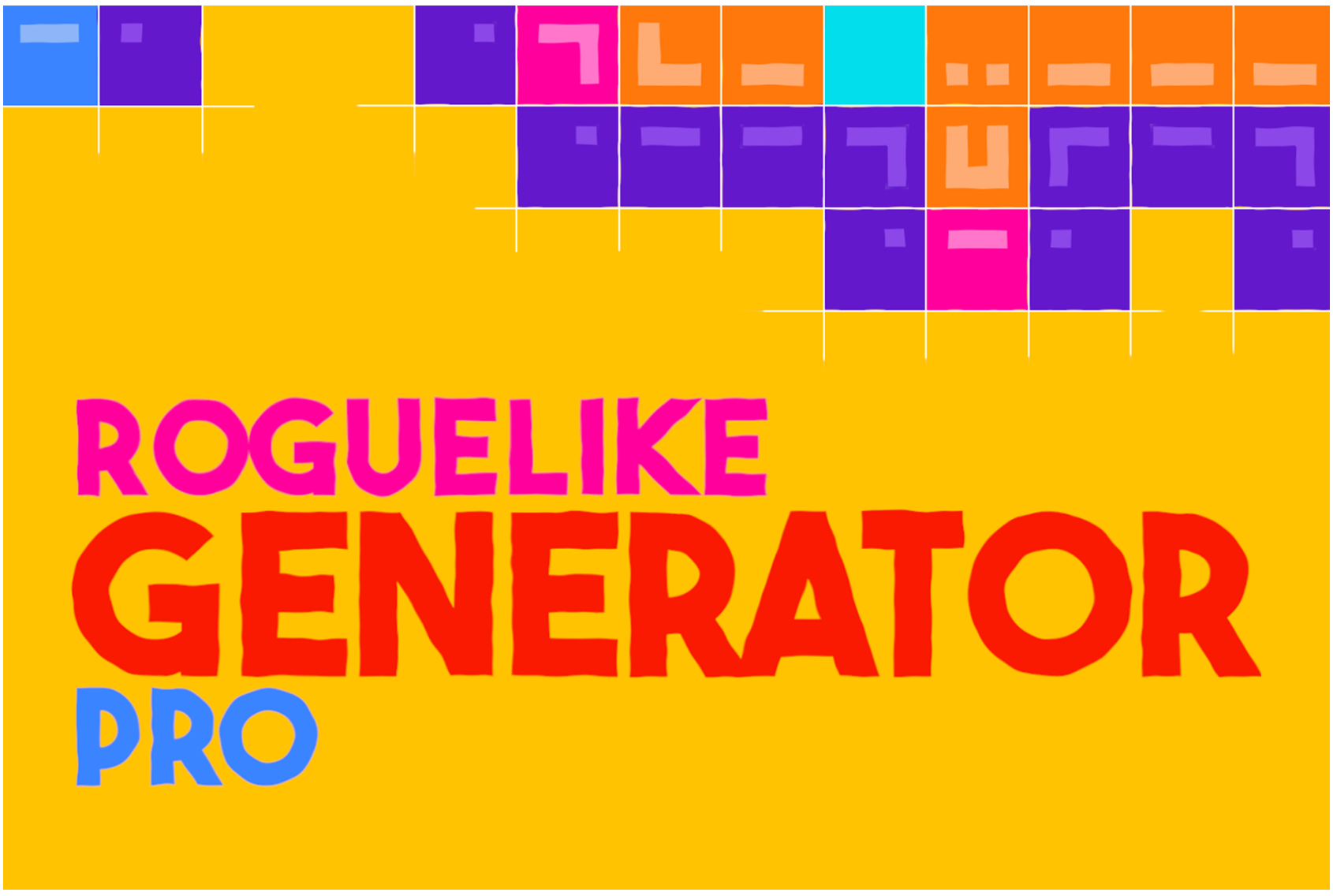


Roguelike Generator Pro

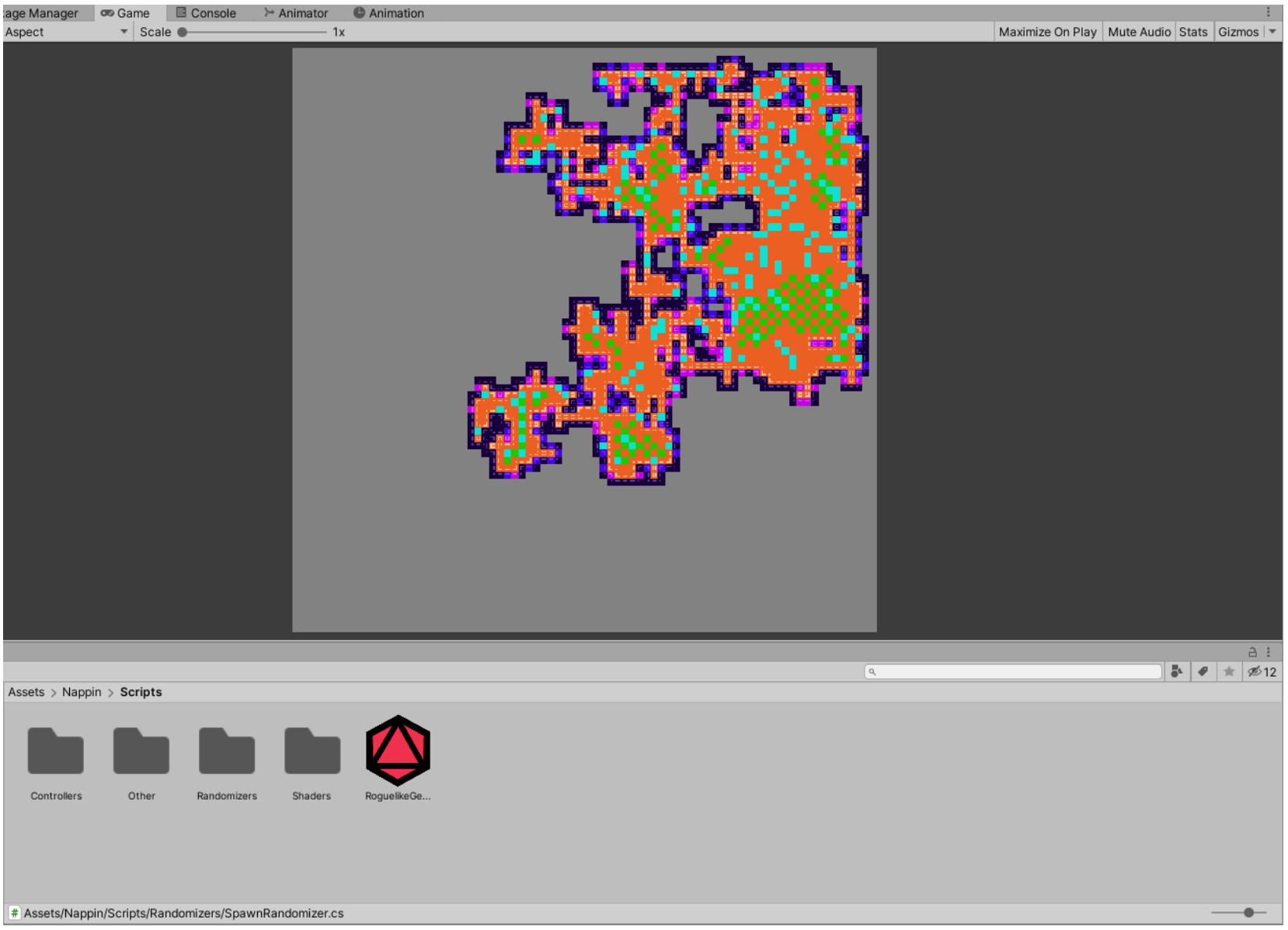
*Support email: **nappin.1bit@gmail.com***

Table of Contents

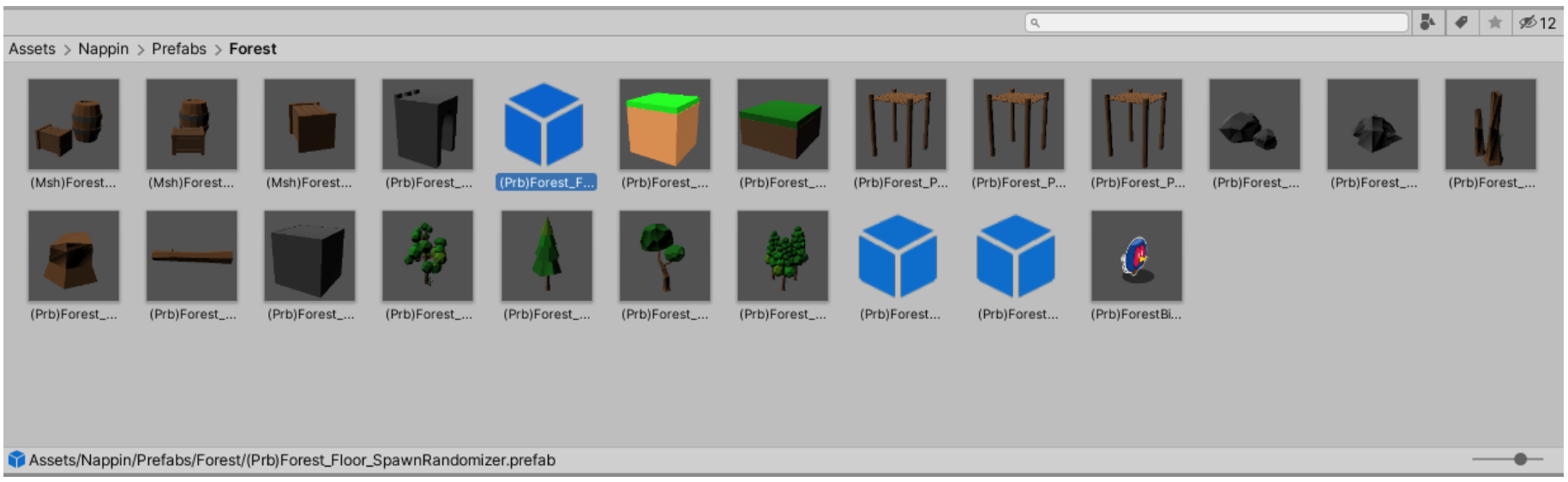
Asset Content

The asset contains multiple assets, prefabs and textures used in the demo scenes and easy to use to kickstart your project. The core content of the asset can be found in the *Prefabs* folder and in the *Scripts* folder.

In the *Scripts* folder you can find the **RoguelikeGeneratorPro** script used to generate the dungeon / levels in all the demo scenes. Here you can also find additional scripts to sub-randomize the tiles that appear in the demos (**Randomizers folder**, RotationRandomizer and SpawnRandomizer) and multiple character controllers inside the **Controller folder**. You can also find a **GameManager** script used to call the dungeon generation in the various demos.



In the *Prefab* folder you can find assets, models and characters used in all the demo scenes. In here you can also find *Prefab* randomizers that spawn GameObjects when enabled.

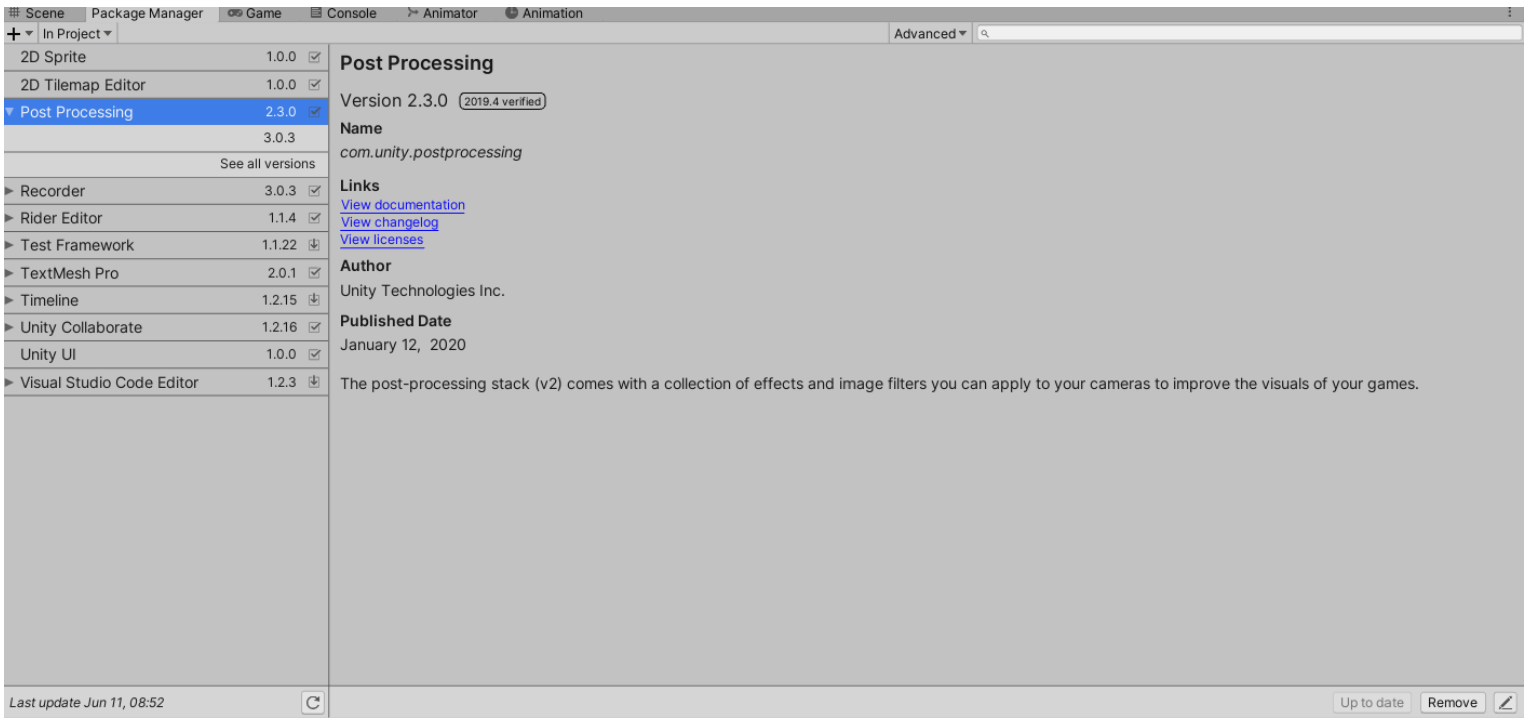


The only 2 necessary elements to run the asset and generate content are a GameObject with the script **RoguelikeGeneratorPro** and a **GameManager or custom script** that calls the function RigenerateLevel() in the RoguelikeGeneratorPro script.

> The RoguelikeGeneratorPro GameObject(s) present in a scene can also generate levels IN EDIT MODE without a GameManager by pressing RegenerateLevel in the script component or by using the shortcut CTRL + G

External Packages

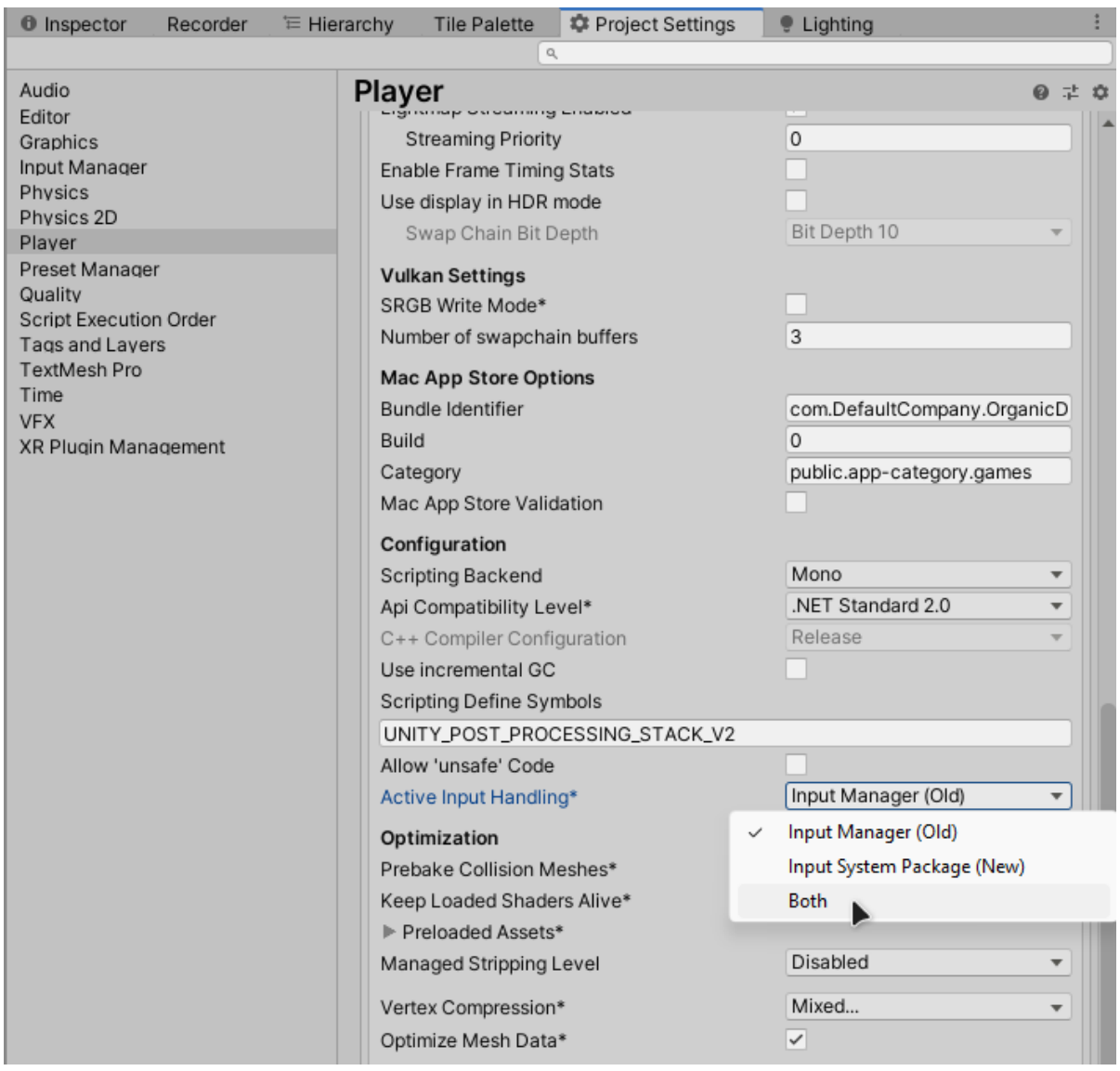
The asset doesn't require any additional package. In the demo scenes some PostProcessing has been used so if you are interested in keeping the visual flair it's recommended to import unity PostProcess package.



N.B. If you are having troubles with PostProcess compatibility or you intend to use the asset in HDRP, LWRP or URP it's recommended to just remove the PostProcess.

Setup the Input

The asset demo scenes use the **OldUnityInput** so no setup is required: the default settings used for the old input are enough to operate the character controllers without edits. If you have imported the asset and are interested in testing out the demo scenes but want to use the new **InputSystem** in your project is recommended to enable both in the *Player* tab of the *ProjectSettings*



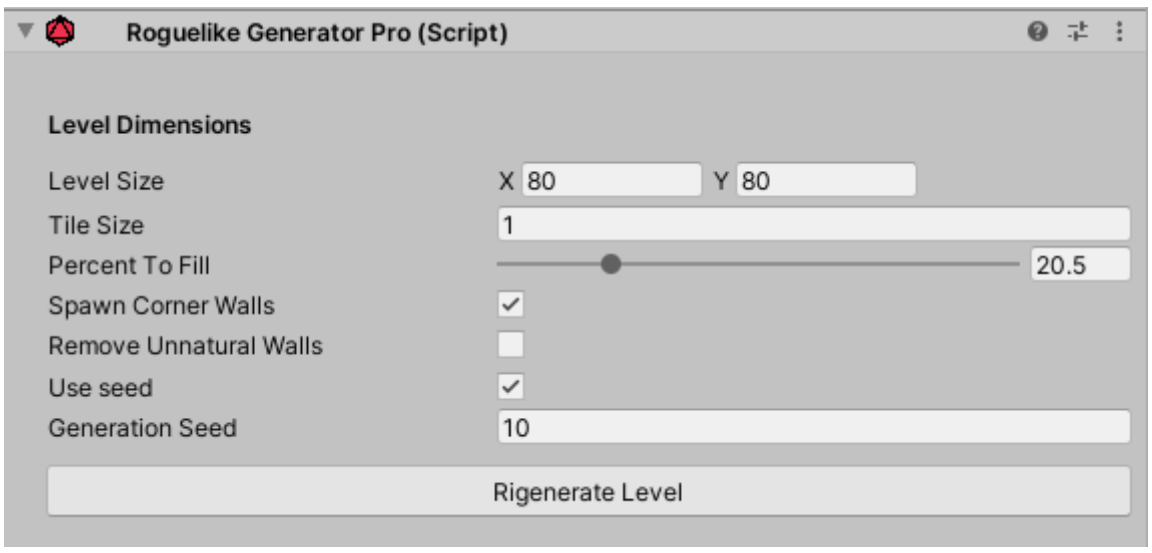
Generator Setup

Let's dive into how to setup a **RoguelikeGeneratorPro** component and how to handle its different generation types.

All the variables in the **CharacterManager** script have a description, you can read it just by hovering over the variable.

Level Dimensions

Here you can setup the dimension specifics as well as customize the generation rules.



Its variables are:

- ***Level Size***: set the level size of the level / dungeon, by default X and Y can't be smaller than 4. Regardless of the value you or your player set a check is performed to make sure that the generation is possible.
- ***Tile Size:*** if GameObjects are used for the generation the TileSize is each tile localScale. If a Tilemap is used for the generation the TileSize is the Grid cell size.
- ***Percent To Fill***: percentage of the overall area to fill with tiles. The asset has a fallback safety feature to avoid infinite loops and if more than 1000 generation iterations have been done the generation stops. The 1000 value is editable via script.
- ***Spawn Corner Walls***: if enabled walls are spawned in the corners (*highlighted in red in the picture below*).

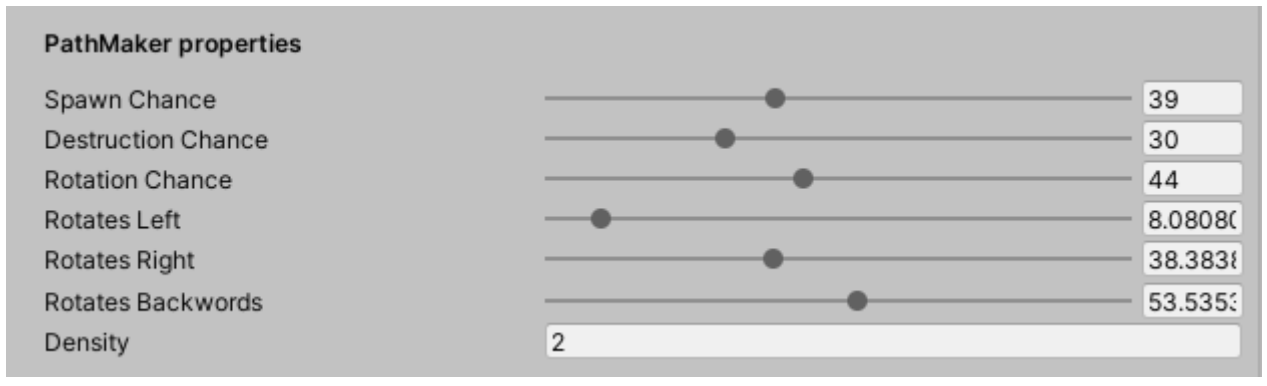
- ***Remove Unnatural Walls***: if disabled doesn't remove "unnatural" walls from appearing in the middle of the floors. Ideal for horror corridors or labyrinths. If the option is enabled more open and wide spaces are created (*highlighted in red in the picture below*).
- ***UseSeed + GenerationSeed***: allows you to use a generation seed to obtain always the same level / dungeon

Rigenerate Level

You can regenerate your dungeon both in game and in EDIT MODE by pressing the **RegenerateLevel** button shown in the screen below. This can be done for every dungeon present in the scene and when done the previously generated dungeon content is automatically cleared. If you press CTRL + G all the **RoguelikeGeneratorPro** components available in the scene regenerate their respective level / dungeon.

PathMaker Properties

The dungeon generation algorithm works like this: at the beginning of the generation a PathMaker is created and moves up. The tile that the pathmaker just walked on is now a dungeon tile. After the pathmaker moves forward it can rotate left / right / backward, spawn another PathMaker (until a maximum value) or destroy itself (unless it's the only pathmaker alive). When the level is filled according to the ***PercentToFill*** value all the pathmakers are killed.



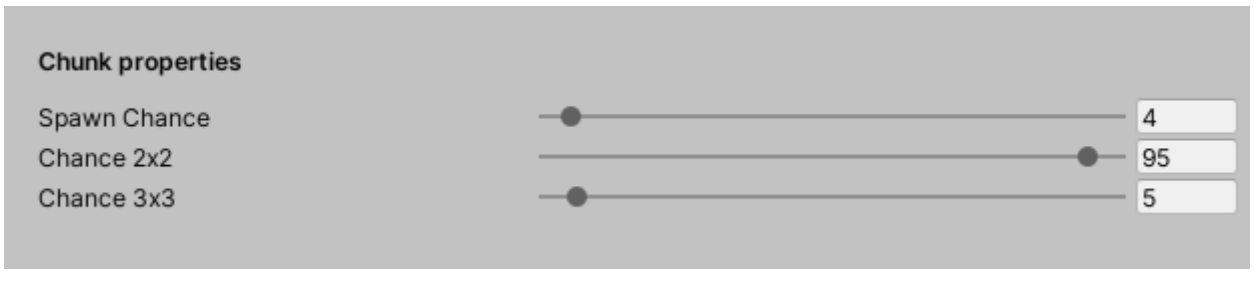
Its variables are:

- ***Spawn Chance***: the chance of spawning another PathMaker.
- ***Destruction Chance***: the chance of destroying the current pathmaker (not calculated if there is only a single pathmaker).
- ***Rotation Chance***: chance that the PathMaker will rotate.

- ***Rotation Left***: chance that the PathMaker will rotate left.
- ***Rotation Right***: chance that the PathMaker will rotate right.
- ***Rotation Back***: chance that the PathMaker will rotate back.
- ***Density***: the maximum number of PathMakers that can be alive at any time.

Chunk Properties

A **chunk** is a set of tiles (2x2 or 3x3) that can be spawned by a PathMaker during generation. When a lot of chunks are spawned the level / dungeon feels generally bigger and with wider corridors.

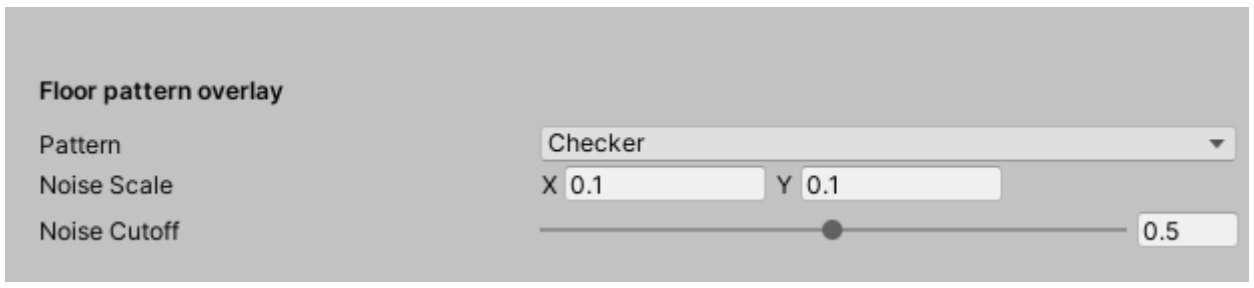


Its variables are:

- ***Spawn Chance***: the chance that a chunk will be spawned.
- ***Chance 2x2***: the chance that a 2x2 chunk will be generated.
- ***Chance 3x3***: the chance that a 3x3 chunk will be generated.

Floor Pattern Overlay

Overlay pattern tiles can be added to the floor.



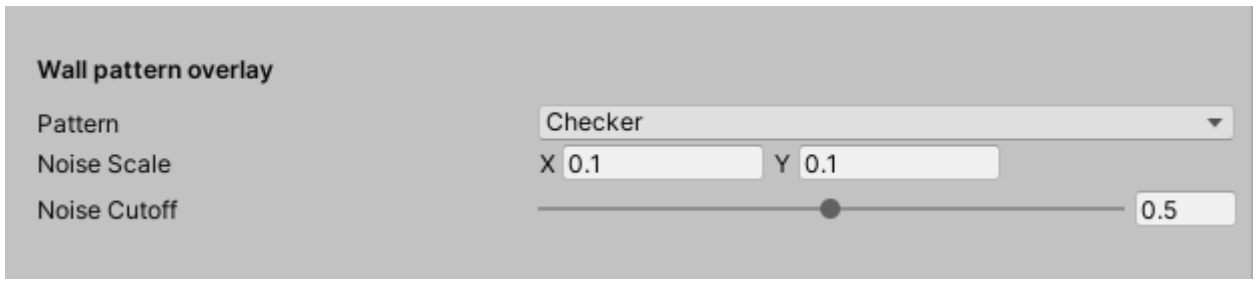
Its variables are:

- ***Pattern:*** the pattern used for the generation, the pattern can be:
  - ***Perlin:*** simple perlin noise.
  - ***Checker:*** using a checker pattern.
  - ***Wide Checker:*** using a checker pattern with a distance of 2 between the pattern tiles.
  - ***LineLeft:*** pixelated line tilted to the left (-45°).
  - ***LineRight:*** pixelated line tilted to the right (45°).
- ***Noise Scale***: the scale of the noise used for the pattern generation.
- ***Noise Cutoff***: the cutoff used for the generated noise.

> The spawn of patterns can be distabled but not in this section of the component. Check the Generation section below

Wall Pattern Overlay

Overlay pattern tiles can be added to the walls.



Its variables are:

- ***Pattern:*** the pattern used for the generation, the pattern can be:
    - ***Perlin:*** simple perlin noise.
    - ***Checker:*** using a checker pattern.
    - ***Wide Checker:*** using a checker pattern with a distance of 2 between the pattern tiles.
    - ***LineLeft:*** pixelated line tilted to the left (-45°).
    - ***LineRight:*** pixelated line tilted to the right (45°).
- ***Noise Scale***: the scale of the noise used for the pattern generation.
- ***Noise Cutoff***: the cutoff used for the generated noise.

> The spawn of patterns can be distabled but not in this section of the component. Check the Generation section below

Floor Random Overlay

Overlay pattern tiles can be added to the floor.

Spawn Chance: the chance to spawn a random tile.

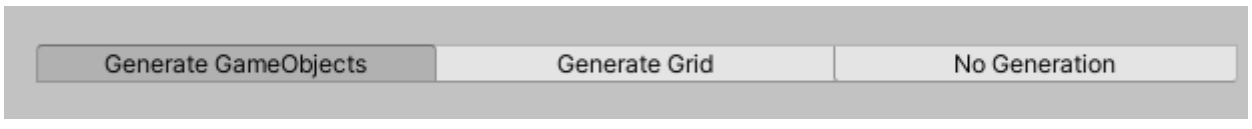
Wall Random Overlay

Overlay pattern tiles can be added to the walls.

Spawn Chance: the chance to spawn a random tile.

Generator Spawn

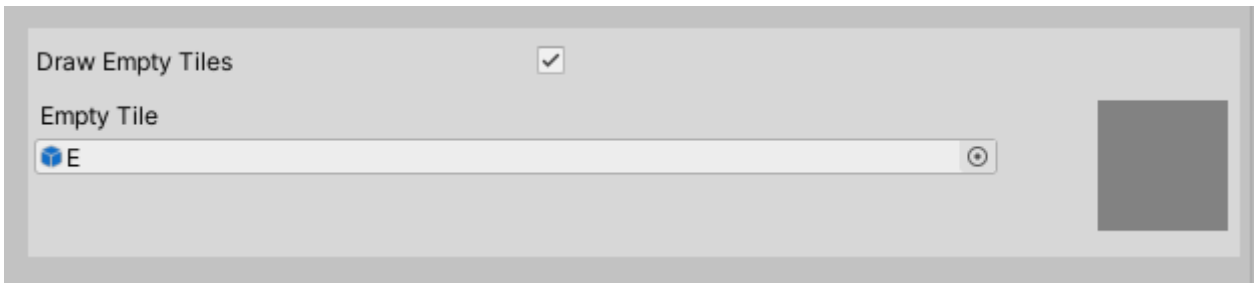
After the content of the generation has been setup the script proceeds with the content generation itself.



> To add flare and randomness to Tilemap generations is recommended to use TileRules

Draw Empty

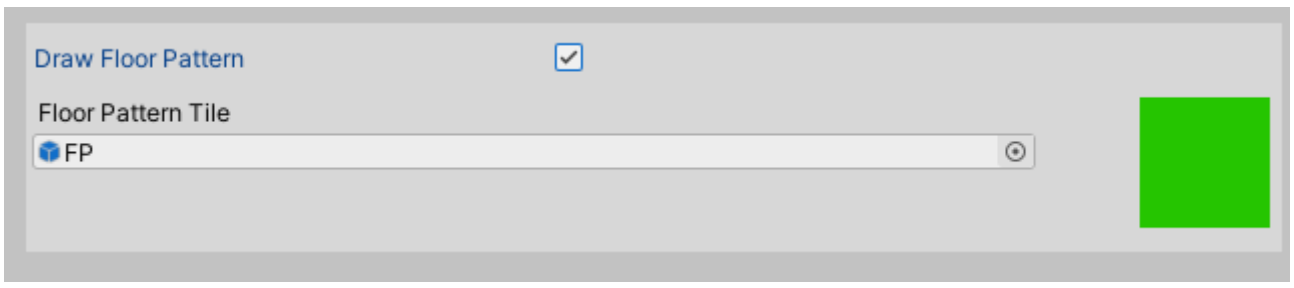
Here you can customize even further your generation, in particular you can draw / not ***Draw Empty Tiles.*** Meaning you can fill all the tiles that are not wall / floor within the levelsize.



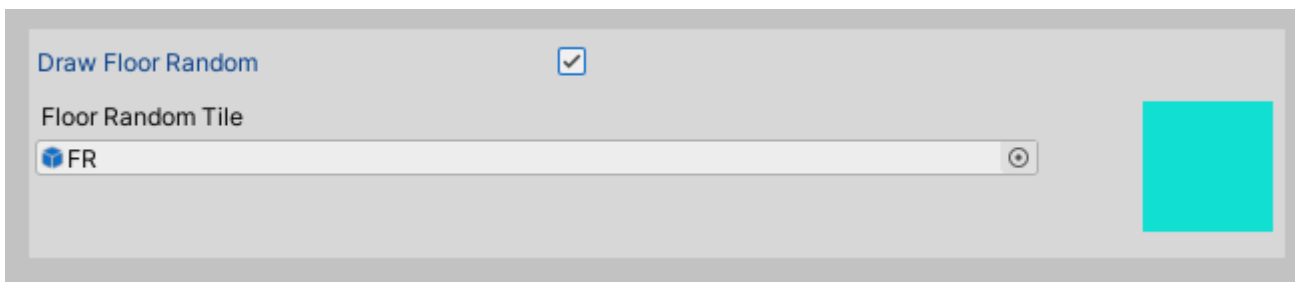
Draw Floor Overlays

Here you can customize even further your generation, in particular you can:

- ***Draw Floor Pattern Tiles***: draw the pattern tiles on the floor.



- ***Draw Floor Random Tiles***: draw the random tiles on the floor.

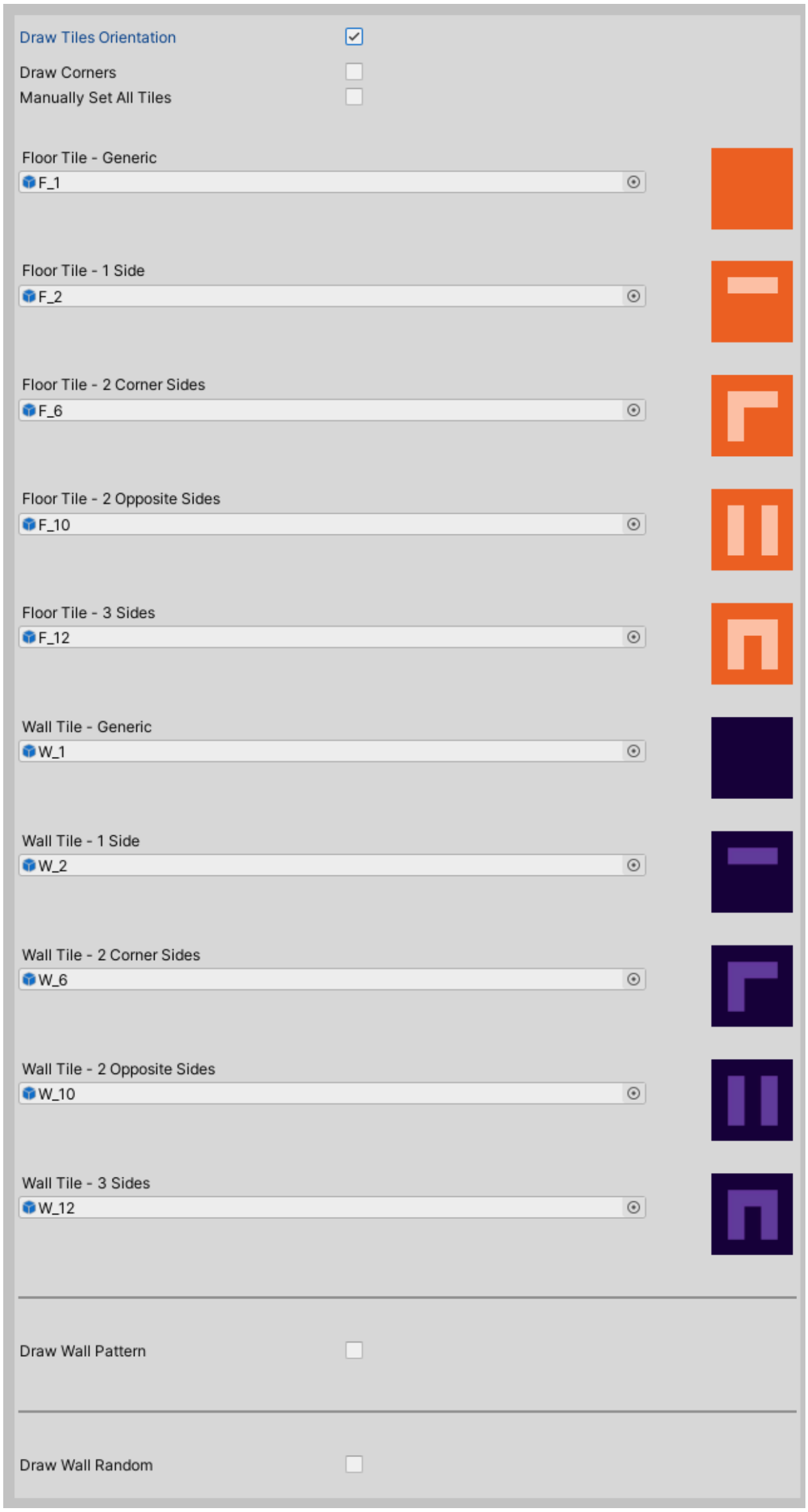


Draw Floor / Walls

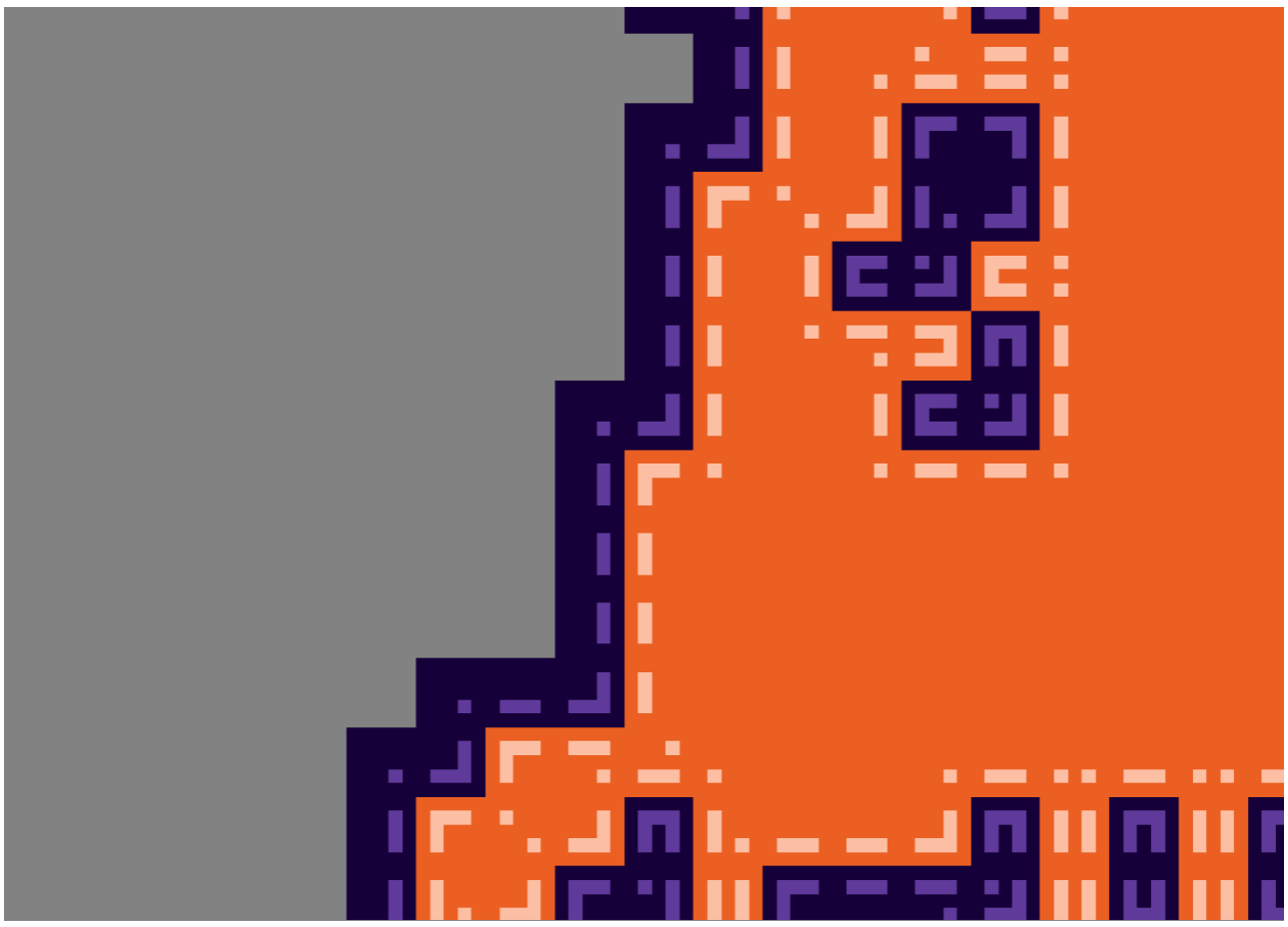
Here you can customize even further your generation, in particular you can ***Draw Tiles Orientation:***

- If the option is disabled the orientation isn't drawn.

- If the option is enabled the orientation is drawn.



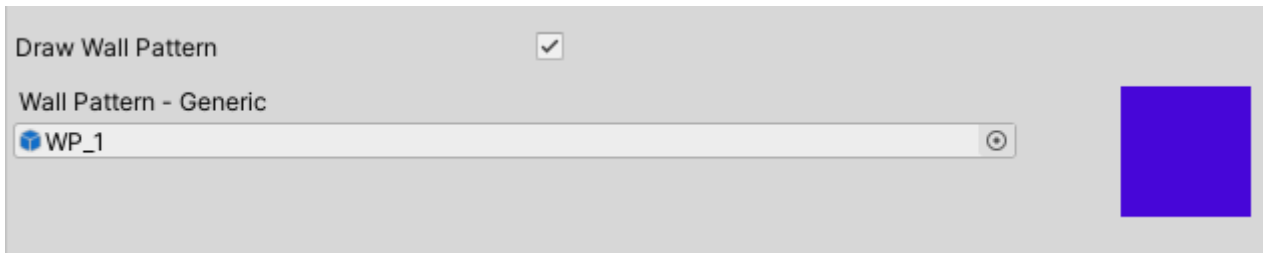
When the orientation is drawn additional options are available like the possibility to ***Draw Corners Tiles*** on the floor and wall and the possibility to ***Manually Set All Tiles*** (useful for example to fake perspective in a 2D game).



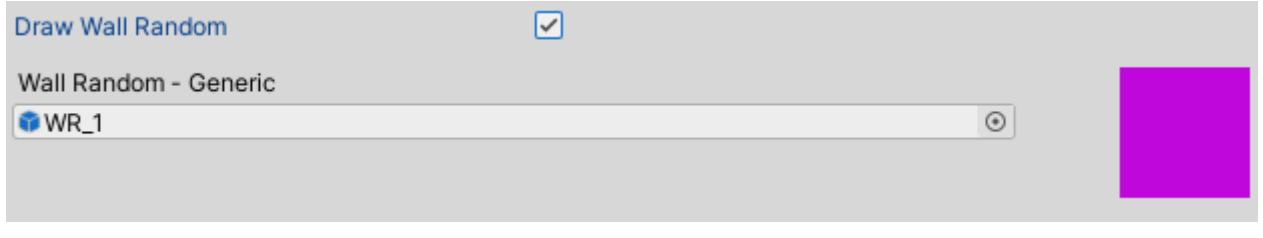
Draw Wall Overlays

Here you can customize even further your generation, in particular you can:

- ***Draw Floor Pattern Tiles:*** draw the pattern tiles on the floor.



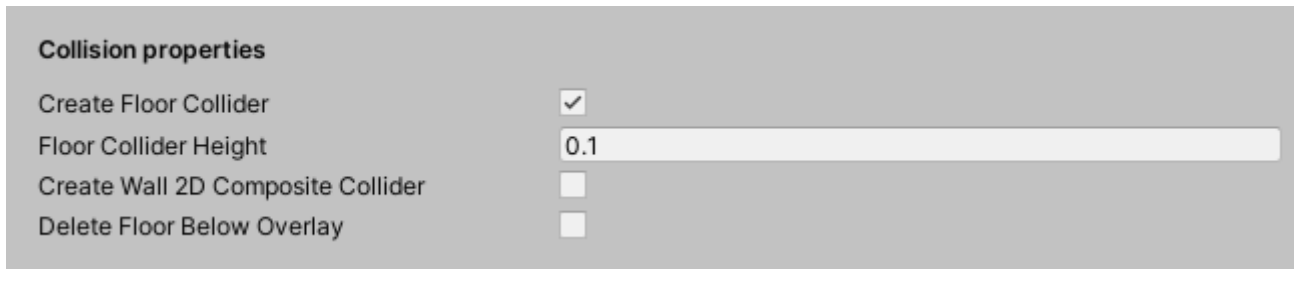
- ***Draw Floor Random Tiles:*** draw the random tiles on the floor.



N.B. The wall overlays inherit the orientation from the wall / floor tiles. This means that if ***Draw Tile Orientation*** is enabled for the wall / floor tiles the wall overlay tiles will have orientation as well

Collision Properties

Here you can set you level collision regardless if you are developing a 3D / 2D or 2.5D game.

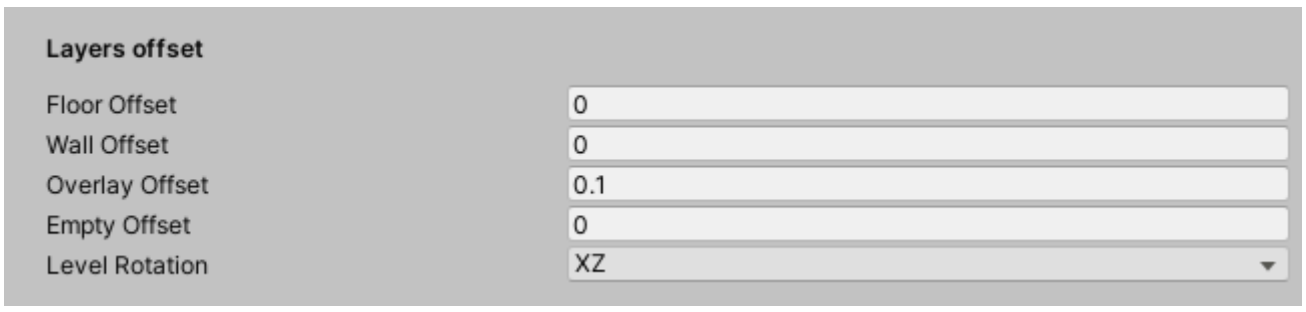


Its variables are:

- ***Create Floor Collider:*** create a single giant box collider at the floor level. Ideal for big 3D worlds. When the option is enabled you can set the box collider height:
    - ***Floor Collider Height:*** height of the box collider.
- ***Create Wall 2D Composite Collider***: create a composite collider on the "Wall" parent gameobject (generated with the level / dungeon). The composite collider allows the game to have a easier time to calculate collisions, ideal for dense 2D worlds.
- ***Delete Floor Below Overlay***: delete the floor tile below an overlay. Might be useful for pits or animated traps in both 2D and 3D.

Layer offsets

Allows you to add an offset to the different layer parents (generated dynamically with the dungeon). The offset is of course rotation dependent.



Its variables are:

- ***Floor Offset:*** offset of the floor parent.
- ***Wall Offset:*** offset of the wall parent.
- ***Overlay Offset:*** offset of the overlay parent.
- ***Empty Offset:*** offset of the empty parent.
- ***Level Rotation:*** allows you to rotate the level / dungeon on whatever axis you want (the option is not available for tilemap generation as it's not needed).

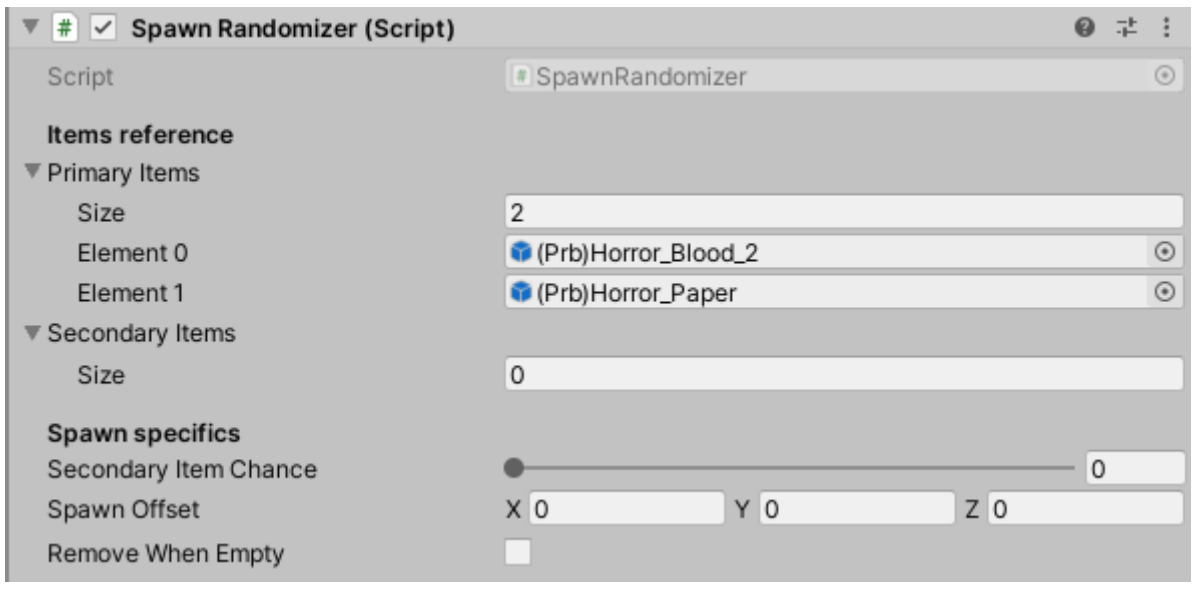
Randomizers

If you are interested in sub-randomizing the outcome of the generation you can do it by spawning an empty gameobject and instantiating a random inside a list on Awake(). The asset already includes 2 useful randomizing solutions.

Both the scripts are available in the **Scripts/Randomizer** folder. Prefabs are also available inside each scene folder.

Spawn Randomizer

This randomizer allows you to spawn a random in a set of primary items and/or to spawn secondary items to replace the primary ones.



Its variables are:

- ***Primary Items***: is an array of N items. On awake the script spawns randomly one of the list
- ***Secondary Items:*** is an array of N items. If the ***Secondary Item Chance*** value is greater than 0, a random secondary item will take the place of a primary one.

- ***Secondary Item Chance*** the probability that a secondary item will override a primary item.
- ***Spawn offset*** the offset position of the asset spawned.
- ***Remove When Empty*** if the primary and secondary item list is left empty the parent is removed. Why make it an option and not the default? So that you can potentially still use the tile and enable it and call the Awake function with a delay.

Rotation Randomizer

Allows you to randomize the rotation of a gameobject when spawned. The script is quite simple, you can edit the ***Rotation*** option and rotate your gameobject a random angle on a specific axis

No Spawn - Getters

If you want to spawn the content yourself you can do it by simply recovering the information you need through code, in particular using the available **Getters**. The following are shown below but you can of course edit the code and add new ones according to your needs:

```
#region Getters

1 reference
public tileType[,] GetTiles()...


0 references
public overlayType[,] GetOverlayTiles()...


1 reference
public Vector2Int GetLevelSize()...


1 reference
public float GetTilesSize()...


1 reference
public levelRotation GetLevelRotation()...


1 reference
public genType GetGenerationType()...

#endregion
```

The available getters are:

- ***GetTiles():*** returns all the tiles and their tileType (wall, floor, detail, empty).
- ***GetOverlayTiles():*** returns all the overlay tiles and their overlayType (empty, wallPattern, wallRandom, floorPattern, floorRandom).
- ***GetLevelSize():*** returns the level size.
- ***GetTilesSize():*** returns the tiles size.
- ***GetLevelRotation():*** returns the level rotation.
- ***GetGenerationType():*** returns the genType (generateObj, generateTile, noGeneration).

Contact

If you found this guide useful but need further help feel free to contact me at the email ***nappin.1bit@gmail.com***
P.S. A positive review of the asset would help a lot!

Cheers