Tip: In a hurry? Check out the Quickstart Guide at the end of this document!

Version 1.3 documentation updated Nov 24th 2018, see end of document for Release Notes.

Strata Overview

Strata is an easy to use procedural toolkit for generating 2D levels which use a combination of hand-authored and procedurally generated content that you can use to build levels. It works in the Unity editor or at runtime without additional programming. Strata works with Unity's native 2D Tilemap tools allowing you to hand-author pieces of levels and arrange and combine those pieces in interesting ways.

The design philosophy of Strata is strongly inspired by both classic roguelike dungeon crawler design approaches along with the newer school of action roguelike games like Spelunky and Nuclear Throne. To this end the project contains demos both for a classic top down roguelike dungeon generator, and a 2D platformer built from a connected chain of rooms. Both serve as good starting points for your own creations.

Programming procedural level generation tools can be complex and difficult, and with Strata I hope to open the creation of procedural content to artists, level and game designers and other non-programmer creators.

How It Works

Strata works by generating level data into a grid of ASCII characters in multiple passes using what we call Generators. Each Generator contains a procedural content generation algorithm ranging from the simple, like randomly scattering 100 instances of an object, to the complex, like generating a linked series of dungeon rooms. The Generators are run sequentially. This means that each new Generator's output is applied to the previous Generator's output. This means, for example, that we can choose to generate some organic cave shapes, carve out a series of rectilinear rooms from them and then scatter objects within the empty spaces of the result. The choice of Generators and the order that the Generators are sequenced in via the Unity Inspector interface can result in significantly different generated levels.

One of the common problems in procedural generation is that you can end up with results which while technically unique, all feel the same to the player. This is sometimes referred to as the 'Thousand Bowls of Oatmeal' problem. Yes, they're all different, but who cares? To combat this, Strata supports the inclusion of hand authored elements within it's generation process so that you as the designer can choose to have as much or as little randomness as you prefer, and insert hand authored set pieces and features for your players to discover.

Features:

- Generate complex, varied procedural levels without writing any code
- Hand author content to integrate into your procedural levels using Unity's Tilemap tools
- Multiple levels of random chance in generation: hand place items with a chance to spawn between multiple choices in authored rooms
- Generate levels at runtime for new levels on each play through, or generate at edit time in the editor, polish the output by hand and save.
- Supports deterministic seeded randomness and daily challenge style play
- Generators Included:
    - Linked top to bottom room series for platformers

    - Connected room sequence for top-down dungeon crawlers with start and end rooms
    - Middle out spreading building generation inspired by wang tiles
    - Cellular automata for organic cave shapes
    - Randomly placed rooms connected by tunnels (traditional ASCII roguelike style)
    - Random scatter X objects
    - Tunneler to connect multiple previous Generators
    - Save full tilemaps and feed them back into the Generation process (randomize existing levels)
    - Place objects within connected space (place player start and level exit)
    - Random scatter hand authored chunks
    - Mirror / Symmetry generators for making ‘man made’ feeling spaces.
    - Flood fill
    - Binary Space Partitioning for building-like subdivided series of connected rooms.
    - And more coming soon!
- Professional video tutorials, text documentation and well commented C# source code.
- Easy to extend for programmers with modular framework based on Delegation pattern and ScriptableObjects
- Rogue-lite platformer example scene
- Top down dungeon example scene
- Just does one thing, generates map data that you can use in lots of ways, not a full game project. Doesn’t include a bunch of junk you don’t need!

What Strata Is Not

Strata is not a game-kit, engine or complete game project. It's designed to support multiple types of games which use 2D tile based levels and procedural generation. The idea here is that you can start with strata as a procedural generation toolkit and build your game alongside it, whatever type of game it is.

It’s also not a cookie cutter, samey, single approach, single algorithm dungeon generator. You can think of Strata as a set of creative tools, not a replacement for your creativity in designing procedural levels and gameplay.

Architecture

The architecture of Strata is designed to be as modular as possible. To this goal, it uses a design pattern called Delegation (read more here if you like: https://en.wikipedia.org/wiki/Delegation_pattern). What this means is that we use helper asset objects to achieve our desired functionality. We can plug these in and out via the Unity Inspector. The helper objects are stored as custom Unity ScriptableObject assets. When we want to use let's say, three different Generator types to generate an outdoor area in our game, but five other Generators to generate a dungeon, we don't need to write new code to do this, we simply choose the Generator assets we want to use, plug them in via the Inspector and make sure that they each hold the settings we want for that usage. This makes it very easy to create flexible systems where we don't have to write new code when we want the system to do different things. This is helpful both for non-programmers and for programmers who want to keep their code base modular.

For programmers, this means that Strata should hopefully be very easy to extend. To create new Generators, simply inherit from the Generator base class and override the Generate function. Then you can write whatever level generation logic you like and integrate it easily in Strata's layered generation process as another pluggable option.

What's In The Box

Here are short descriptions of all Strata's classes and what they do, for full details on implementation all source code is commented line by line.

Core Classes

BoardGenerationProfile: This is a data collection that contains data about how to generate a level: what generators to run, the level size, the BoardLibrary to use, the seed and the level dimensions. We can think of this as 'how are we going to build our level'. This is a ScriptableObject Asset. It probably makes sense to use one per level or type of generated area.

BoardLibrary: This contains the mapping from ASCII character to Unity Tiles, for example: the ‘w’ character represents a wall, etc. We can think of this

as the library of pieces we are going to use to build our level. This is a ScriptableObject asset.

BoardLibraryEntry: The BoardLibrary contains a collection of these entries, and each one has information about how each tile should be spawned. These include a reference to a TileBase asset (Unity 2D Tile class), the character that it maps to in our ASCII representation of our level internally, if it should spawn a prefab instead of just a Tile in the Tilemap, whether it is the default floor tile and if it is a Chance character.

ChanceChar / **ChanceBoardLibraryEntry**: These are used to provide a second level of probabilistic randomness, you can define a tile in your BoardLibrary to have a chance to spawn other tiles instead of the one specified. This could be used to place a generic 'some monster' tile in a dungeon room and then which actual monster is chosen based on probability when the level is generated. This is very useful to make a relatively small number of rooms, features or templates more varied and unique seeming.

BoardInstantiationTechnique: This is an experimental feature which means if you want to generate something besides a Tilemap from the data Strata creates you can. An example of this would be to use the generated map data to spawn 3D objects, or a voxel mesh, or ASCII characters instead of tilemaps. If there is demand for this it may get fleshed out later, for now it's a stub. The default BoardInstantiationTechnique is TilemapInstantiationTechnique and this should be assigned by default on setup.

BoardGenerator: This takes the data from the BoardGenerationProfile and BoardLibrary and runs the Generation process. This is a MonoBehaviour that should be attached to a GameObject in your scene. By default Strata will create this for you during setup. It has the options to randomize the random seed value every time you play, to Use Daily Seeds for daily challenge type gameplay and buttons to allow level generation in the Unity Editor at edit time.

BoardGeneratorEditor: Simple editor script that just adds the Generate/Clear buttons to BoardGenerator.

RoomTemplate: The main unit of hand authored content in Strata, contains settings for x and y size and whether or not it has openings to the north, south, east and west which are used to ensure connectivity when generating chains of rooms.

RoomList: Holds a list of RoomTemplates, usually used for grouping together all rooms with north exits etc.

RoomTileSpace: used during RoomSequence and RoomTiler generation, holds information about a space in the intermediate 2D grid used during these generation processes including x,y coordinates in grid space, a list of RoomLists and a final selected RoomTemplate.

GridPosition: Simple helper classes to store an integer based 2D grid position used during level generation

GridPositionList: a holder for a list of GridPosition objects.

StrataContentEditorWindow: This is the main Editor tool for Strata which allows for the loading, saving and authoring of hand made rooms or map pieces using Unity's Tilemap tools. Access via Window > Strata Tilemap To RoomTemplate Creator

PathVisualizer component: This records information about the direction the path has moved in during generation and draws an arrow Gizmo in the scene view, along with the name of the RoomTemplate that was drawn. Works with RoomSequence and PlatformerRoomChain. Useful for debugging generation.

DrawArrow: Simple helper class that draws arrow gizmos in the Scene view. Used by Pathvisualizer component.

Generators:

Generator: Base class for creation of new Generators. Inherit from Generator and override the Generate function, adding your content generation code there, add the [CreateAssetMenu (menuName="menu/objectname")] attribute and create a ScriptableObject asset so that you can add it to your BoardGenerationProfiles.

GeneratorScatterRandom: Randomly scatter X single tile objects across the grid, choose to overwrite or not overwrite filled spaces.

GeneratorScatterShapes: Randomly scatter X instances of a single RoomTemplate or random choice across the grid.

GeneratorCellularAutomata: Generate organic cave like shapes using Cellular Automata algorithm. This is usually best used as the first layer in the generation process.

GeneratorRoomSequence: Use to start at a point on the grid (predetermined or random) and generate linked RoomTemplates until it runs out of space or reaches a maximum target number. Will retry until a minimum length is reached. Optionally can fill in the grid with random RoomTemplates as well. Best suited for top-down dungeon/building generation.

GeneratorRoomTiler:
This works similarly to GeneratorRoomSequence but instead of generating in a single direction it generates in all directions that a RoomTemplate has open exits. This means you should start with multiple traversable paths from the center and end up with at least one entrance to the structure that is traversable to the center. If using rooms with four exits it can easily fill the level in a few passes. Likes to generate squares rotated 45 degree ziggurat shapes. Also has an option to randomly not grow in a direction to create more irregular structures.

GeneratorRoomChain: (*Deprecated and removed in 1.1*) use RoomTiler or RoomSequence instead, they’re simpler, more reliable and just generally better. RoomChain turned out to have a lot of dumb bugs at release so I basically rewrote it from scratch.

GeneratorPlatformerRoomChain: Starts in top row of grid and places RoomTemplates until it hits edge or randomly drops down, then drops down to the next row and places random rooms there until it reaches the bottom edge of the grid. Modeled on the level generator used by Spelunky.

GeneratorFullMap: Very simple, just stores and loads an entire Tilemap, useful if you want to save generator output and post process it, or hand author large maps.

GeneratorTunneler: Creates lines between points, can randomly spawn rooms at tunnel end points for traditional roguelike ascii dungeon generation room/tunnel style, can also be used to connect previous Strata Layers by selecting random empty spaces from each previous layer that is tagged with 'Generates Empty Space' and tunneling between them. Can draw tunnels of any character so not just useful for empty space, could also work for rivers etc. Tunnels can have variable width.

GeneratorWanderTunnel: (*Update 1.1* Added optional noise turnNoiseValue to turning to generate more squiggly passages, also options to spawn rooms on tunnel and at tunnel turns) Similar to GeneratorTunneler but creates one, long connected tunnel that wanders randomly. Useful because all tunnels are always connected and contiguous, unless overwritten.

GeneratorTunnelBrancher: A variant of GeneratorWanderTunnel which has a chance to spawn additional tunnels at each step. It uses GeneratorWanderTunnel for branches. Nice for creating 'fingers' off a tunnel that end in a room for the player to leave the main path and explore.

GeneratorMirrorHorizontal/Vertical: Splits the map in half on the horizontal or vertical axes and mirrors the content. Great for creating maps with symmetry, including buildings, spaceships and other 'made' structures. Make sure to check for connectivity after mirroring because this can create inaccessible areas.

GeneratorFloodFill: Works similarly to a 'paint bucket tool' in an image editing program, flood a contiguous region with a character/tile. Useful for filling in floor tiles in a dungeon interior for example.

GenerateRandomContiguousItems: This is used to place entrance and exit objects that are always in connected space. The placement is random so they sometimes end up right next to each other. This is useful when not using RoomChains or other more structured approaches and you want to make sure that when you spawn the player you do so in an area which has a reachable exit.

GeneratorSquareBorder: This draws a square of a single tile, usually used to make a border around the map, if you want to make an wall at the map edge that the player can't get out of.

SimpleFill Generator: This overwrites the entire grid with a character. Useful when you want to start with a filled grid and dig out.

PerlinNoise Generator: Simple noise generator, no smoothing or post-processing, just fill the grid with noise, either overwriting what is already there or filling in empty space. Includes a variable to scale the size of the noise pattern to create rounder or more chaotic shapes.

CaveDigger Generator: Another simple generator that does a random walk clearing out a pre-defined percentage of the map. Will continue moving

randomly and creating empty space until a desired percentage of spaces have been cleared.

GeneratorSpaceDivider: Uses a binary space partitioning style algorithm to subdivide a rectangular volume repeatedly into smaller and smaller connected spaces. Options include minimum room size, number of divisions, door size and wall thickness.

RoomRect: Support class for GeneratorSpaceDivider which does most of the actual division logic.

Demo Scenes:

Penny Pixel: This demo contains a 2D platformer character, with a character controller (art and controller code from tutorials by Unity Technologies). This scene uses the PlatformerRoomChain generator and generates random platformer levels. This uses the Rule Tiles from Unity's 2D extras, included for convenience but more info and up to date code is available on github here: (https://github.com/Unity-Technologies/2d-extras)

MiniRogueDemo: This uses creative commons sprites by Arachne (www.retinaleclipse.com) to demonstrate a simple dungeon crawler / roguelike level which uses a multi-layered generation approach including cellular automata, middle out room chain, random scattering and more. No character controller provided (no movement), just level generation.

Update 1.1: Added a few more simple example scenes to this folder focusing on single Generator types in isolation.

- **MiniRogue BranchingTunnel Only Example**
- **Minirogue RoomSequence Only**
- **MiniRogue RoomTiler Only**

Update 1.2:

- **MiniRogue CaveDigger Only**
- **MiniRogue PerlinNoise Only**

Update 1.3:

- **MiniRogue SpaceDivider Only**

Compatibility:

The feature area of Unity which Strata interacts with most heavily is Tilemap, introduced in 2017.2, and the usage is pretty straightforward so Strata should work well with most versions of Unity after 2017.2.

Strata works nicely with the free version of Aron Granberg's A* pathfinding project's 2D support for pathfinding https://arongranberg.com/astar/

Strata works well with Unity's 2D Extras (a version of which are included for the Penny Pixel demo) https://github.com/Unity-Technologies/2d-extras

---

Strata Quickstart Guide

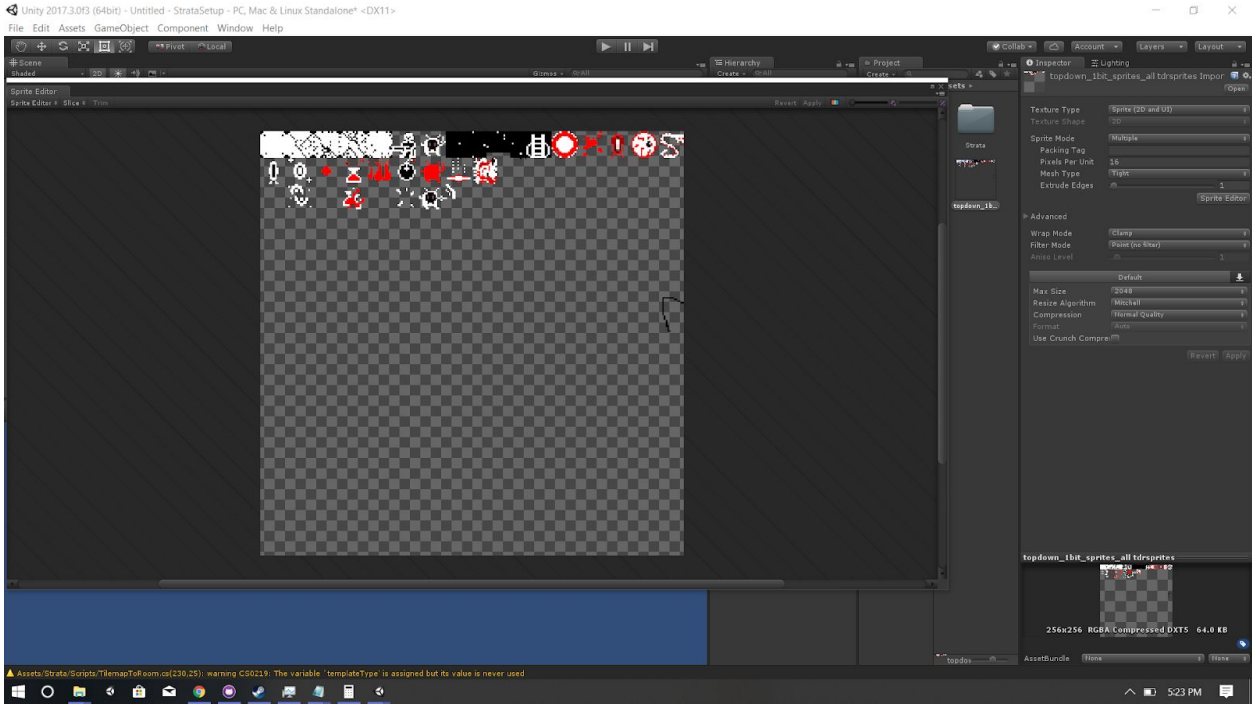
Setup

- 0: Create a new 2D project or open your existing 2D project
- 1: Import the Strata package

Import Sprites

- 2: Import your sprite tiles, making sure that they are set to the Sprite (2D and UI) Texture Type
- 2.1: Set the pixels per unit to match the resolution of your sprites (16 for 16x16 pixel sprites for example). This will cause one Unity unit to match to one sprite tile in your game.

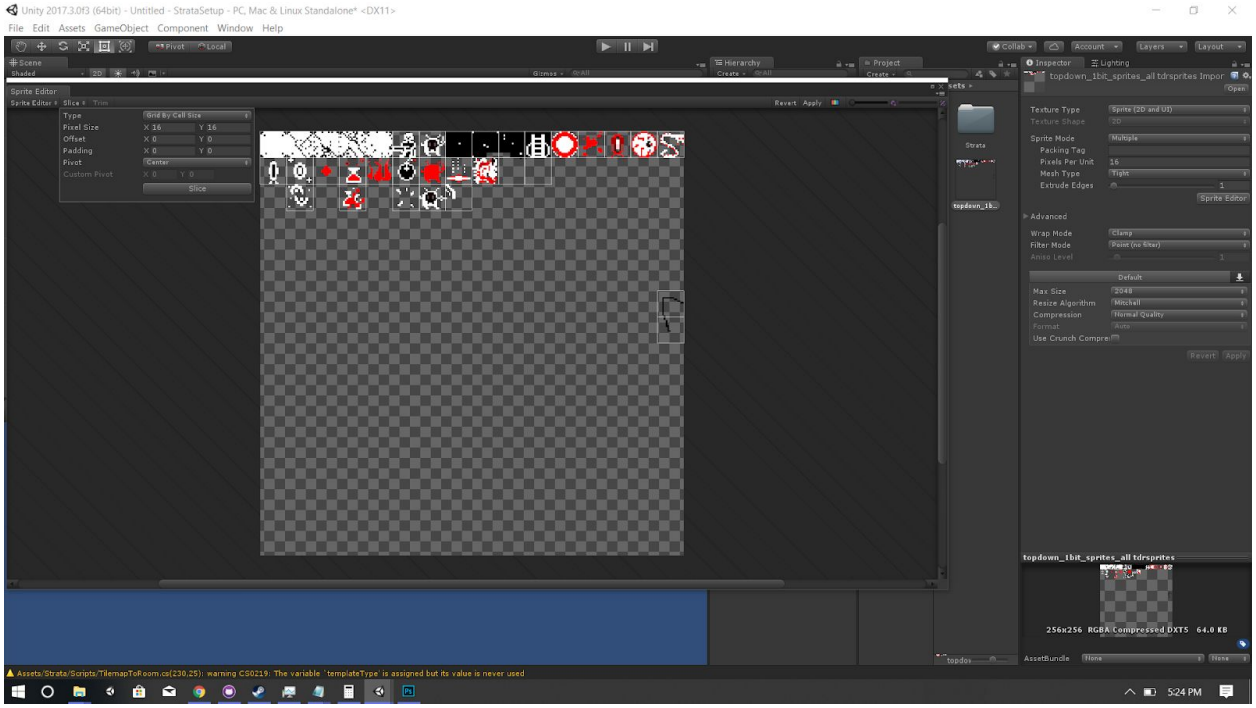
- 2.2: For pixel art, make sure your filter mode is set to Point (no filter) to keep the pixels crisp.



- 2.3: If using a sprite sheet, slice the sprites into individual tiles by setting the Sprite Mode to Multiple and using the Sprite Editor to create individual assets for each sprite in your sheet. Click Apply when opening the sheet to apply any changed settings.

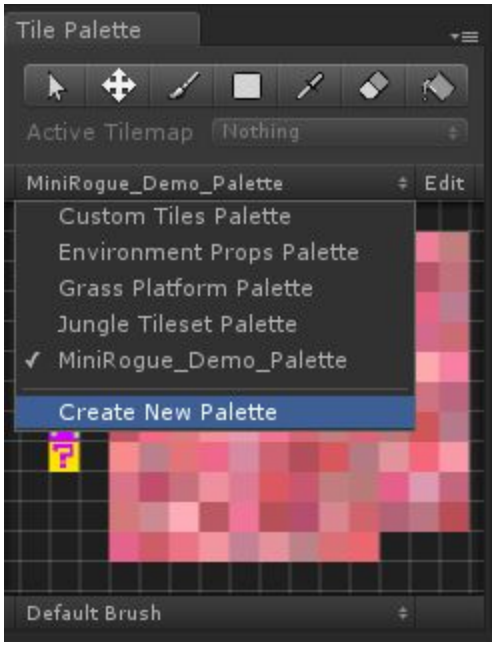
- 2.4: In the Sprite Editor click the Slice menu, choose 'Grid by Cell Size' and Input your resolution (example: 16x16). Click the Slice button.

- 2.5: Close the Sprite Editor and click Apply when prompted. You should now be able to expand your Sprite asset in the project window by clicking the triangle dropdown and see each individual sprite listed.



Creating Your Tile Palette

- 3.1 Open the Tile Palette Window and dock it somewhere handy.
- 3.2 In the Tile Palette window, create a new Tile Palette asset via the Palette dropdown.

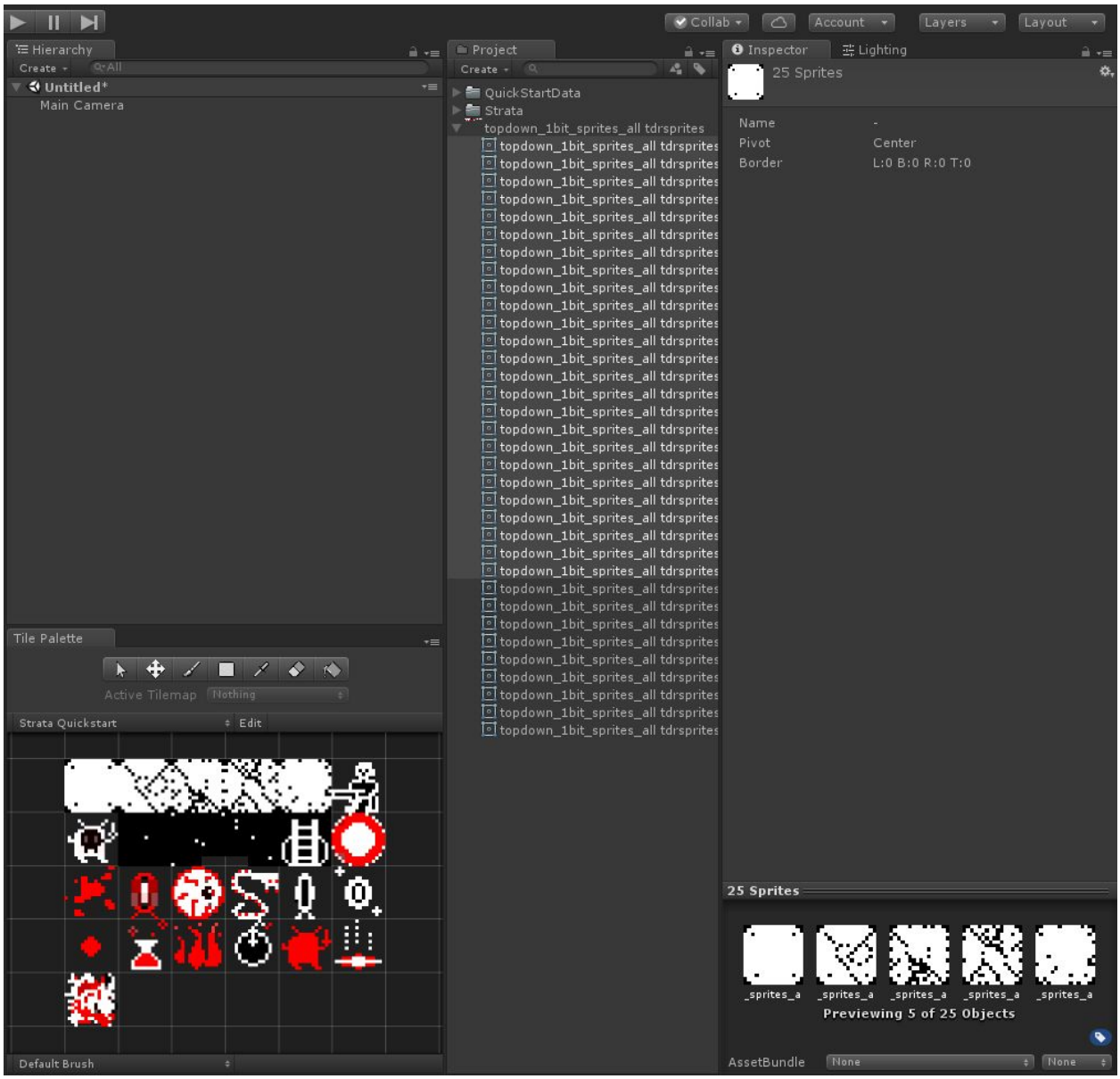


- 3.3 Name it and accept the defaults of Grid: Rectangle and Cell Size: Automatic

- 3.4 Add the desired sprites from your sprite sheet to the Tile Palette by dragging and dropping. It's possible to shift click on all your sprites and drag them all into the Tile Palette at once if desired.

- 3.5 You will be prompted for a location to save the Tile assets that will be created, create a folder for the new Tiles and save them. The tiles will be generated.

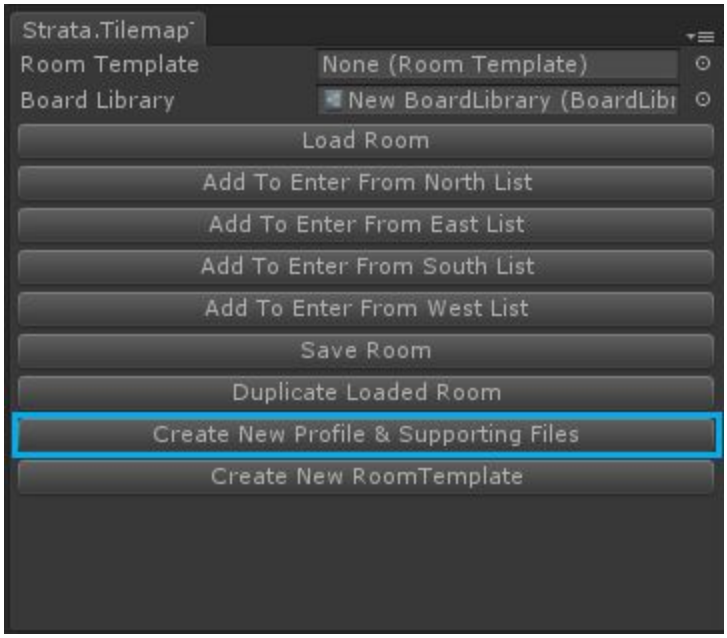
- 3.6 Now, you should have a Tile Palette ready to draw tilemaps with!



Creating Your Strata Assets

- 4.1 Open the Strata Content Editor window under Tools > Strata Content Editor

- 4.2 To set up a new Strata Profile and start creating content select the folder in which you want to store Strata's files and then click "Create New Profile & Supporting Files" in the Strata Content Editor Window.



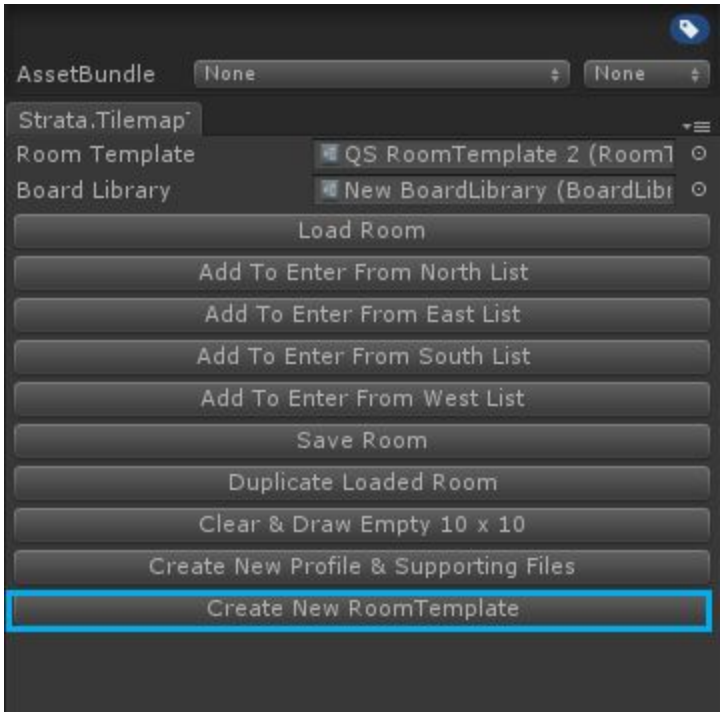
- Note: If you already have a GameObject with a Tilemap in your Scene, Strata will select it and add the BoardGenerator to the same GameObject. If you do not it will create a new Tilemap GameObject and assign the Boardgenerator component to it.

- 4.3 Rename the newly created files if desired.

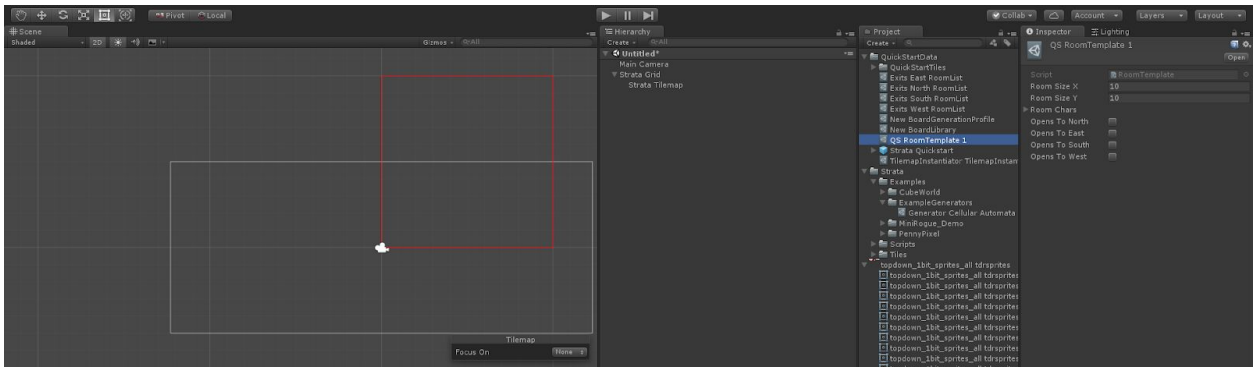
- 4.4 The next steps will vary depending on the types of content you want to generate. Let's do a quick example of spawning some connected rooms, with a few other random elements.

Creating Your First RoomTemplate

- 5.1 Let's create a new RoomTemplate. This is a hand-made room that can be spawned by Strata during the generation process. Click Create New RoomTemplate in the Strata Content Editor window. This will be created in whatever folder is currently selected in the Project view and automatically loaded as the current RoomTemplate in the Content Editor window.

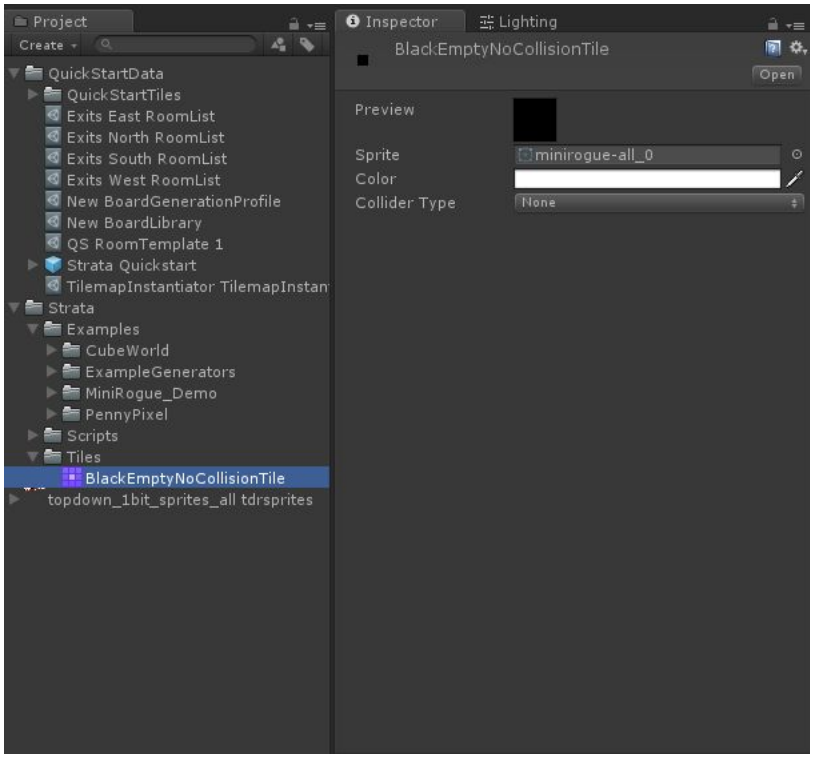


- 5.2 In the Scene View you should now see a red square drawn, this represents the current dimensions of our RoomTemplate which are 10x10 Unity units. You can modify this if you choose, it's worth noting that both of the RoomChain Generators are designed to work best with regularly sized rooms. It's possible to use irregular shapes but your mileage may vary. Note: this draws starting at 0,0 in world space if your authoring Tilemap is not at 0,0 it will misalign.

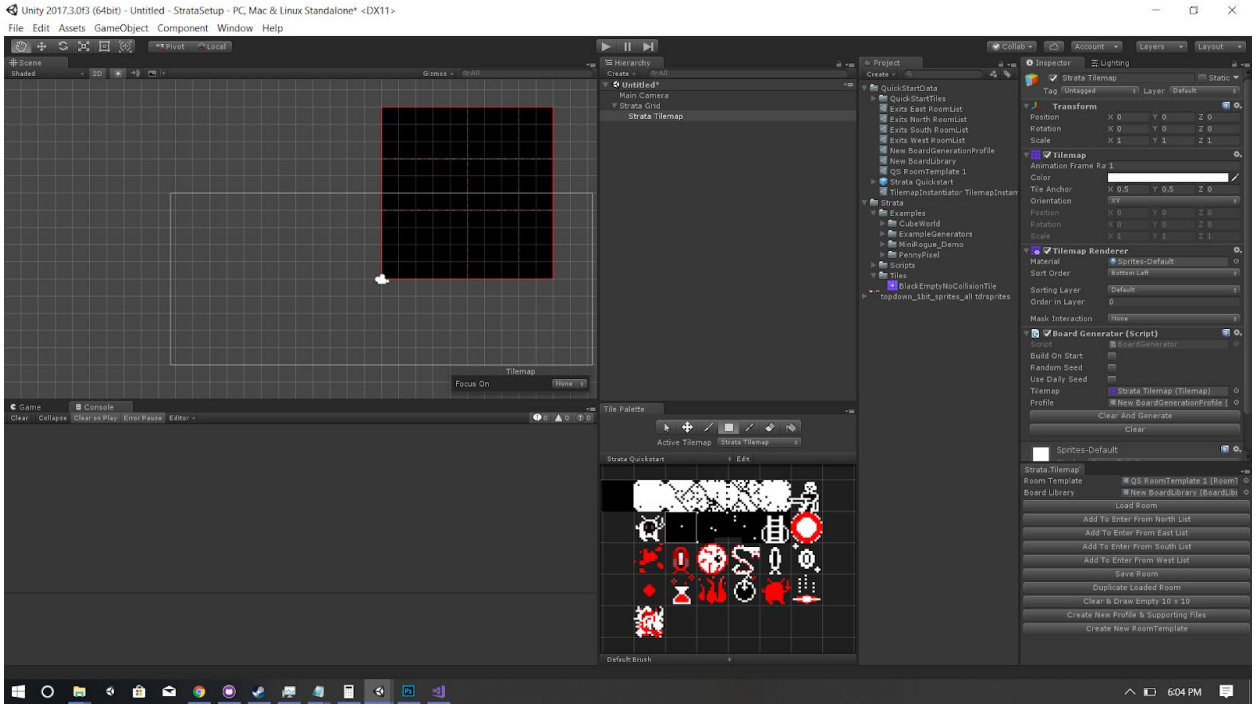


- 5.3 As a convenience I've provided a default, black empty tile asset in Strata by MirrorfishMedia Inc > Tiles > BlackEmptyNoCollisionTile. You don't have to use this if you have another tile you want to represent empty space in your maps but for now we'll use it, so let's drag it

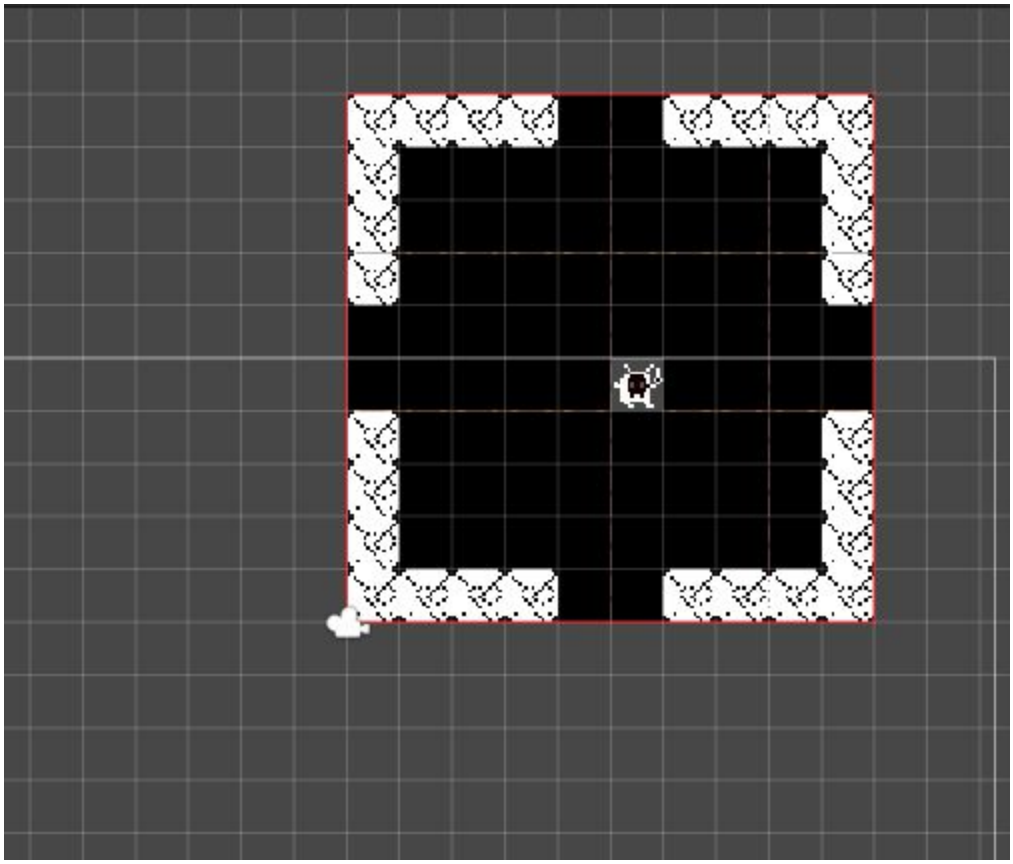
into our Tile Palette as well. Note: This should be added automatically to the BoardLibrary and flagged as ‘Use As Default Empty Space’ = true.



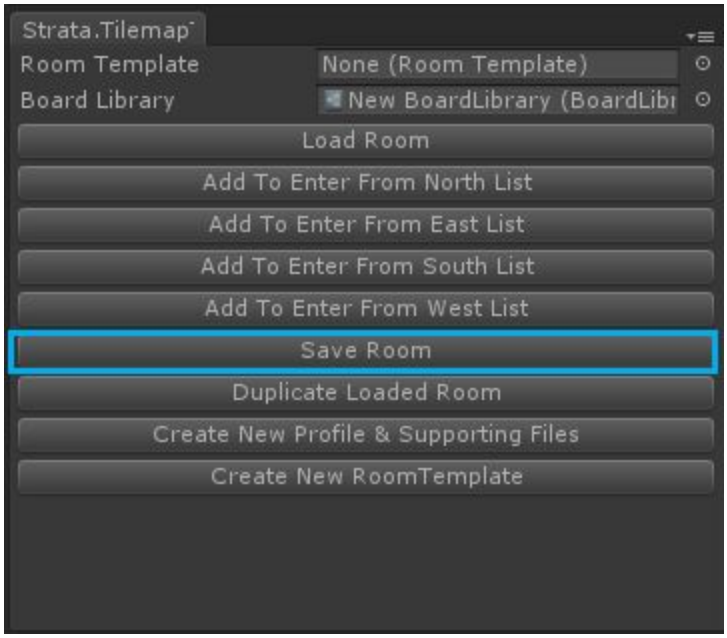
- 5.4 Click the "Clear & Draw Empty 10x10" button in the Strata Content Editor window. This will draw a black, empty room, filling the area of our RoomTemplate.



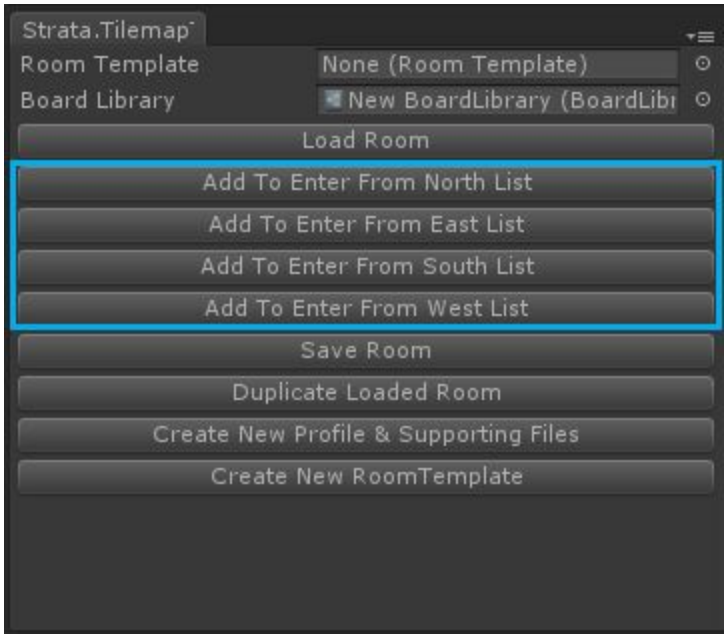
- 5.5 Use the Tilemap tools to draw a room or area in the 10x10 space provided which includes entrances to the North, South, East and West. You can decide where you want these entrances to be yourself, in our example I am using the middle two tiles of each side of the room. These will need to match up to the other rooms in order to ensure connectivity.



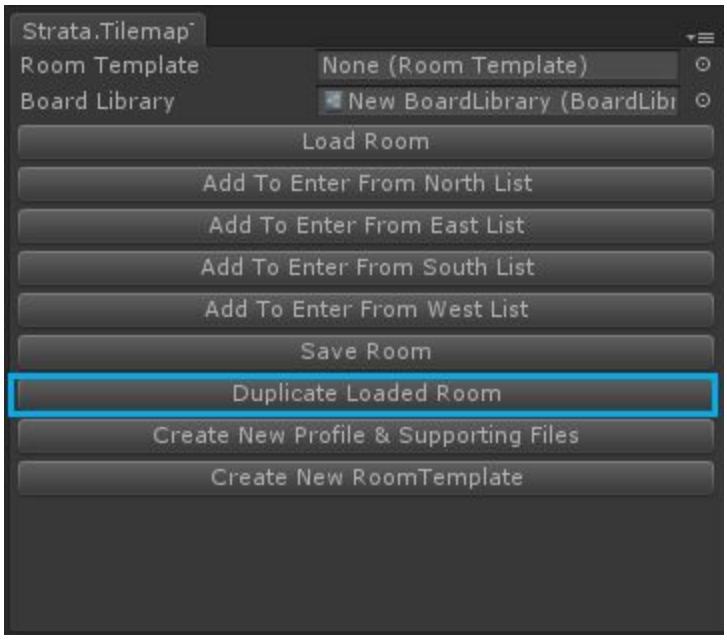
- 5.6 Click the 'Save Room' button in the Strata Content Editor window. This will write the data from our Tilemap into our RoomTemplate asset. You will see confirmation messages logged to the console that any tiles you used in your RoomTemplate have been added to the BoardLibrary asset and that the RoomTemplate was written successfully.



- 5.7 Now, since this a room that will be used in a connected series of rooms, we want to write some information to it about it's exits, and add it to our lists of rooms organized by exit. This room opens in all four cardinal directions so click all four buttons labelled 'Add To Enter From North List', south list, etc. You will see messages saying that the RoomTemplate was added in the console. This step is not required if the rooms won't be used for either the RoomSequence, RoomTiler or PlatformerRoomChain generators, they are the ones which need data about connectivity to work.

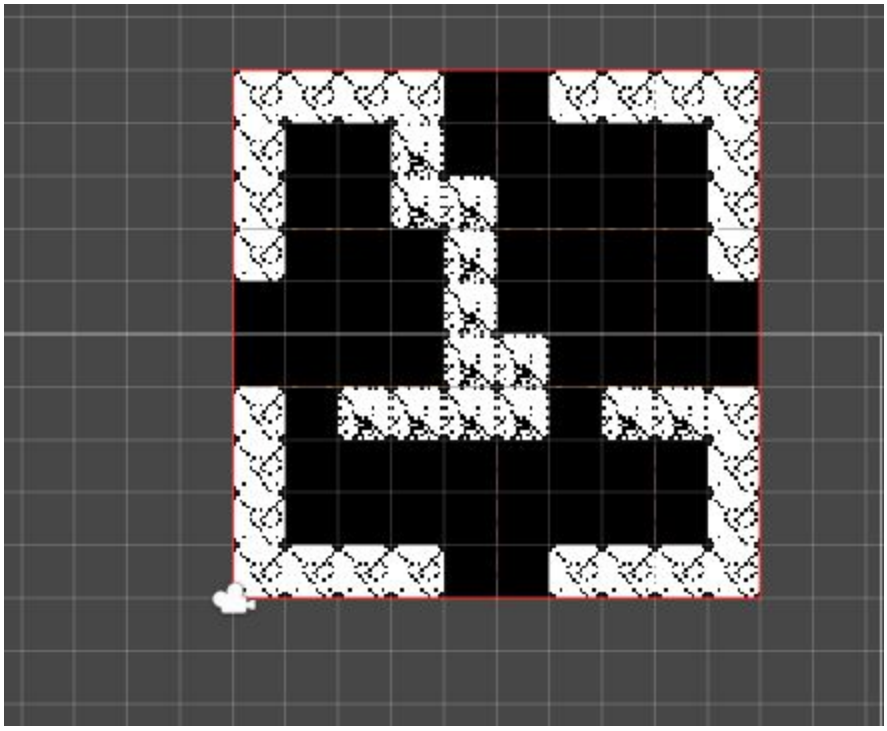


- 5.8 Let's create a variation of our RoomTemplate so we don't have only one type of room. Click the "Duplicate Loaded Room" button in the Strata Content Editor window. You'll see a message confirming this in the console. The duplicate is now assigned to the RoomTemplate variable of the Strata Content Editor window.



- 5.9 Make a change to the room using the Tilemap tools and click 'Save Room'. Add it to the relevant exit lists as well.

- 5.10 Feel free to repeat the process and make as many variations as you want, for our purposes we'll move forward with two.



Generating Content

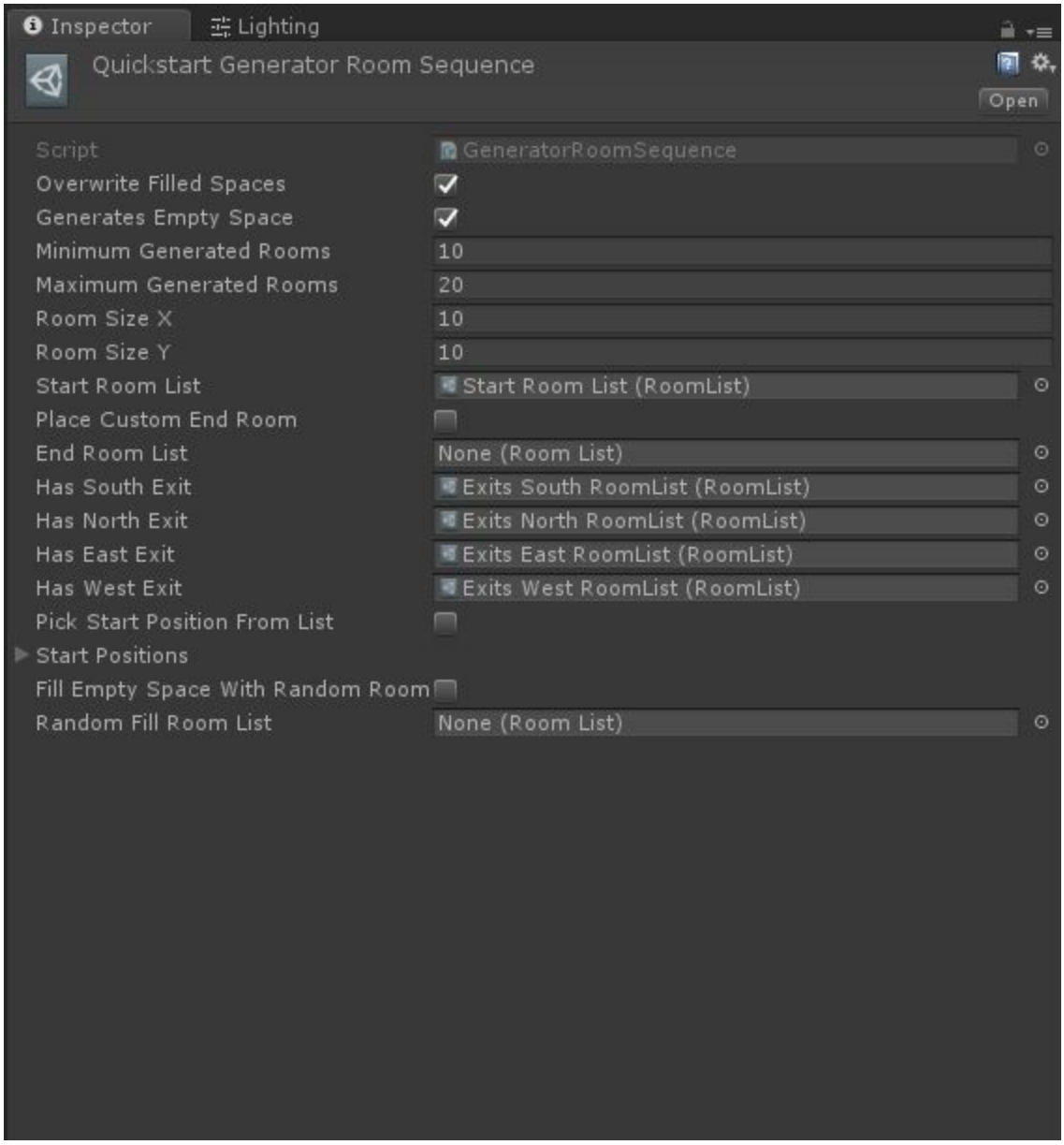
- 6.1 Let's create a new RoomSequence to generate a top down dungeon based on our two RoomTemplate assets. Select the folder where you are creating your Strata data and select Assets > Create > Strata > Generators > Generator RoomSequence
- We'll also need a list of starting rooms to pick from so choose Assets > Create > Strata > Collections > RoomList and call it Starting Room List.
- Add one (or more) of your RoomTemplates you want to start with to the RoomList property of your new Starting Room List.

- 6.2 Select the new Generator RoomSequence asset and set it's properties to the following:

  Overwrite Filled Spaces: True
  Generates Empty Space: True

  Minimum Generated Rooms: 10

Maximum Generated Rooms: 20
Room Size X: 10
Room Size Y: 10
Start Room List: (drag in your Start Room List)
Place Custom End Room: false (unchecked)
End Room List: none
Has South Exit: (drag in your list of rooms with south exits)
Has North Exit: (drag in your list of rooms with north exits)
Has East Exit: (drag in your list of rooms with east exits)
Has West Exit: (drag in your list of rooms with west exits)
Pick Start Position From List: false (unchecked)
Start Positions: (empty, leave at default)
Fill Empty Space With Random Rooms: false (unchecked)
Random Fill Room List: (none)



- 6.4 Select the BoardGenerationProfile we created and drag the newly configured RoomSequence Generator into the Generators array

- 6.5 Select the Strata Tilemap GameObject and set 'Build On Start' in the BoardGenerator component to true.
- 6.6 Enter playmode and you should see a new chain of rooms generated based on the RoomTemplates you created.

Experiment!

Now that you've got the basics, here are some other things you could try:

- Add some RoomTemplates to the RoomSequence Generator's 'Random Fill Room List' and turn on Fill Empty Space With Random Rooms to fill the rest of the board.
- try creating some of the other Generator types under Assets > Create > Strata > Generators and adding them to the BoardGenerationProfile Generators list. The order that they are added in, along with the settings in each Generator will have a large effect on what is generated.
- Take a look at the MiniRogue example scene which contains a BoardGenerationProfile that uses multiple passes.
- Experiment with Chance Tiles to add further randomness to your spawned room templates
- Play around and make cool stuff!

For further resources and examples please check out the tutorial videos on my YouTube channel: https://www.youtube.com/channel/UCxRW28Si8_Vb27-IDhFMMKQ

Release Notes:

1.0 Initial Release!
1.1:

- Deprecated RoomChain Generator and replaced with RoomSequence and a RoomChain component. RoomChain Generator proved unreliable in generating chains of desired length. New approach is much more robust and always achieves minimum chain length. Also adds support for placing start/end rooms from a RoomList at the end of the RoomSequence.
- RoomChain component is a data holding object that is created and destroyed during generation. It tracks progress of RoomSequence and RoomTiler generation so that that data is not stored in BoardGenerator, making resetting cleaner and easier.
- Added RoomTiler. RoomTiler is like RoomSequence but instead of spreading in one direction spreads in all available directions marked by exits for a number of iterations. Grows buildings with connected paths branching off from a center.

- Added TunnelBrancher: This is like WanderTunnel but has a percent chance to spawn new WanderTunnels every step. Useful for generating branching tunnel systems.
- Added turnNoiseValue parameter to WanderTunnel, this flips the generation back and forth between preferring to tunnel horizontally first or vertically first based on a random percentage roll, this makes for more irregular, squiggly tunnels. Keep at 0 for clean 90 degree turns in tunnels. Also implemented in TunnelBrancher.
- Moved some source comments into Tooltips, not all yet.

1.2

- Added SimpleFill Generator: This overwrites the entire grid with a character. Useful when you want to start with a filled grid and dig out.
- Added PerlinNoise Generator: Simple noise generator, no smoothing or post-processing, just fill the grid with noise, either overwriting what is already there or filling in empty space.
- Added CaveDigger Generator: Another simple generator that does a random walk clearing out a pre-defined portion of the map. Will continue moving randomly and creating empty space until a desired percentage of spaces have been cleared this pass (not including other generator passes).
- Added PathVisualizer component: This records information about the direction the path has moved in and draws an arrow Gizmo in the scene view, along with the name of the RoomTemplate that was drawn. Works with RoomSequence and PlatformerRoomChain. Useful for debugging generation.
- Cleaned up some small bugs in the example scenes.

1.3

- Added SpaceDivider Generator which allows for generation of a rectangular shape with subdivided internal rooms.
- Added custom icons for the different ScriptableObject types for easier identification in the Project window.
- Updated Editor script for BoardGenerator to hide daily seed option when useRandomSeed is true.
- Fixed Clear and ClearAndGenerate functionality for CubeWorld example
- Fixed PerlinNoise generator to use random seeding.
- Fixed some scripts that weren’t using Strata namespace.
- Some small documentation fixes.

Other Resources:

Disclaimer: I made a bunch of the linked videos during my ongoing work as an employee of Unity Technologies but Strata is an independent product that I have made in my free time and is not supported by or affiliated with Unity in any way. That being said some of the ideas / concepts and theory demonstrated in these videos is very applicable to Strata, and you may find useful in making games with Strata.

On working with Tilemap and 2D Tools:

Intro to 2D World building w/ Tilemap

In this live session we will explore worldbuilding in 2D using Tilemap and 2D Cinemachine. Tilemap makes it fast and easy to create and iterate level design cycles right in Unity, so artists and designers can rapidly prototype when building 2D game worlds. We will look at how you can paint levels using the new tile and brush tools which allow you to define rules for how tiles behave when placed, creating platforms with dynamic edges, animated tiles, random tiles, and more. Cinemachine, is a dynamic, procedural camera system which it easy to automate composition and tracking to enhance gameplay. Cinemachine is now available for 2D games. In this segment we introduce our project and goals.

https://unity3d.com/learn/tutorials/topics/2d-game-creation/intro-2d-world-building-w-tilemap

On Delegation Pattern and ScriptableObjects:

Pluggable AI with ScriptableObjects:

In this session we will look at creating a finite state machine based AI system which can be configured in Unity's inspector using ScriptableObjects for states, actions and transitions between those states.

https://unity3d.com/learn/tutorials/topics/navigation/intro-and-session-goals?playlist=17105

Richard Fine at Unite 2016 - Overthrowing the MonoBehaviour Tyranny in a Glorious Scriptable Object Revolution

A great talk broadly about ScriptableObjects, lots to learn here.

https://www.youtube.com/watch?v=6vmRwLYWNRo

Procedural Patterns With Unity Tilemaps:

Part I:

https://blogs.unity3d.com/2018/05/29/procedural-patterns-you-can-use-with-tilemaps-part-i/

Part II:

https://blogs.unity3d.com/2018/06/07/procedural-patterns-to-use-with-tilemaps-part-ii/