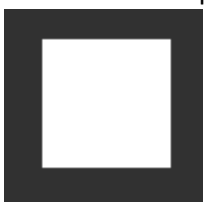
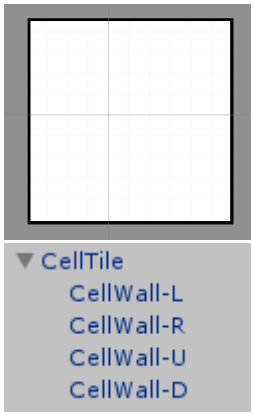
Setting up the generator:

1. Create the sprites.
   You need two Sprites:
   - The main Cell sprite. E.g.

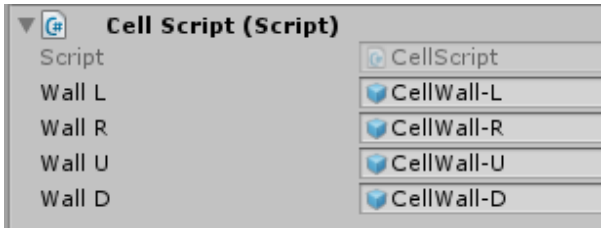


   - The Wall sprite. E.g.

2. Setup the prefab.
   - Create an object for the Cell.
   - Set the main Sprite to be the Cell sprite.
   - Create 4 child objects for the walls.
   - Set the child object sprites to the Wall sprite.
   - Position the walls on the Cell Parent and set rotation. E.g.

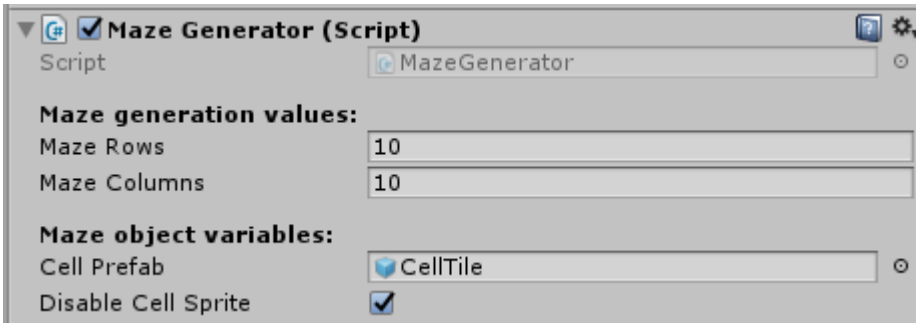


   - Save the object as a prefab.
   - Attach CellScript.cs to the prefab.
   - Drag the four walls into the script variables in the Inspector. E.g.



   !! Make sure these are set correctly! Left wall must be 'Wall L', etc.

3. Setup the Generator Object.
   - Create an empty GameObject in the scene, call it something like ‘MazeGenerator’ (or whatever you like).
   - Attach MazeGenerator.cs to this object.
   - Drag the Cell Prefab into the ‘Cell Prefab’ variable in the Inspector.
   - Set the remaining variables on this script as desired. E.g.



   !! All the variables have tooltips, hover over them for more info.

4. Generating a Maze.
   - By default, the MazeGenerator script creates a Maze on Start using the values from Maze Rows (Height) and Maze Columns (Width):

```
/* This Start run is an example, you can delete this when
 * you want to start calling the maze generator manually.
 * To generate a maze is really easy, just call the GenerateMaze() function
 * pass a rows value and columns value as parameters and the generator will
 * do the rest for you. Enjoy!
 */
private void Start()
{
    GenerateMaze(mazeRows, mazeColumns);
}
```

   - This is just an example and can be removed. You can call the ‘GenerateMaze’ function instead to create a Maze:

```
private void GenerateMaze(int rows, int columns)
{
    mazeRows = rows;
    mazeColumns = columns;
    CreateLayout();
}
```

   - This method was created to ensure you can generate more Mazes at runtime. To use, simply call it with two integer parameters, for rows and columns, respectively.
   E.g. To create a Maze with 20x20 cells:

```
GenerateMaze(20, 20);
```

5. Additional / Notes:
   - A Maze can have different values for Width or Height. E.g.

```
GenerateMaze(10, 20);
```

   - Maze rows and columns MUST be an even number. If an odd number is provided, the value will be subtracted by 1 to ensure an even number is set.
   E.g. Setting a rows or column value of 11 will cause the Generator to end up using 10.

   - Maze rows and columns have a minimum possible value of 4. This is to ensure that enough room is left for the 2x2 centre room.
   E.g. Setting a rows or column value of <= 3 will cause the Generator to set the value to 4.

   - If you don't want to have the main Cell visible (i.e. if you only want a Maze with walls and no 'background' Sprite for each Cell), you can set the 'Disable Cell Sprite' variable on the MazeGenerator script to TRUE. This will disable the SpriteRenderer on each Cell so the Cell is invisible. You do NOT need to disable this on the prefab. It's best to leave a Sprite on the prefab to ensure the generator can correctly determine its size for placement.

   - Every Maze will be generated as a single object, called 'Maze', with all Cells as child objects. The Cell objects are named according to their position on the Maze's grid. E.g. 'Cell – X1: Y1'.

   Every maze starts with Cell X:1 Y:1 as the bottom left Cell. The top right will be the rows and columns value. E.g. A 20x20 Maze's top right Cell will be X:20 Y:20. This is to make any troubleshooting in the Editor easier, as you can easily locate a Cell's position by checking its name.

   - If you wish to delete a Maze, you can do this manually by performing a 'Destroy()' on the Maze parent object and setting the 'mazeParent' variable to 'null' or alternatively, you can simply call the 'DeleteMaze()' function in the MazeGenerator script. E.g.

```
DeleteMaze();
```

   !! This function will only work if a Maze exists in the scene and is set as the 'mazeParent'. It's unlikely that this won't ever be the case but this check is in place just to prevent any possible errors.