

Task1: Procedural Generation

Task2: Camera Following



```
1 extends Node2D
2
3 const MAP_H = 128
4 const MAP_W = 128
5 const BOUNDS = Rect2(0, 0, MAP_W, MAP_H)
6
7 var density = 46
8
9 @onready var tilemap = $TileMap
10
11 func _ready():
12     randomize()
13     generate_map()
14
15 func generate_map():
16     var grid = make_noise_grid()
17     grid = cellular_automaton(grid, 15)
18     grid = clean_edges(grid)
19     var tiles = grid_to_tiles(grid)
20     tilemap.set_cells_terrain_connect(0, tiles, 0, 0)
21     # fill_background(grid)
22
23 func fill_background(grid):
24     for x in range(MAP_W):
25         for y in range(MAP_H):
26             if grid[x][y] == 0:
27                 tilemap.set_cell(0, Vector2(x, y), 2, Vector2(0, 2), 0)
28
```

```
29 func grid_to_tiles(grid):
30     var tiles = []
31     for x in range(MAP_W):
32         for y in range(MAP_H):
33             if grid[x][y] == 1:
34                 tiles.append(Vector2(x, y))
35     return tiles
36
37 func cellular_automaton(grid, count):
38     # var tiles = []
39     for i in range(count):
40         var temp_grid = grid
41
42         for j in range(MAP_H):
43             for k in range(MAP_W):
44                 var neighbor_wall_count = 0
45                 for y in [-1, 0, 1]:
46                     for x in [-1, 0, 1]:
47                         if BOUNDS.has_point(Vector2(x + k, y + j)):
48                             if y != j or x != k:
49                                 if temp_grid[y + j][x + k] == 0:
50                                     neighbor_wall_count += 1
51                             else:
52                                 neighbor_wall_count += 1
53                 if neighbor_wall_count > 4:
54                     grid[j][k] = 0
55                 else:
56                     grid[j][k] = 1
57             # tiles.append(Vector2(j, k))
58     return grid
59
60 func clean_edges(grid):
61     var min_neighbors = 13
62     var temp_grid = grid
63     for x in range(MAP_W):
64         for y in range(MAP_H):
65             if temp_grid[x][y] == 0:
66                 var neighbor_floor_count = 0
67                 for j in [-2, -1, 0, 1, 2]:
68                     for k in [-2, -1, 0, 1, 2]:
69                         if BOUNDS.has_point(Vector2(x + k, y + j)):
70                             if y != j or x != k:
71                                 if temp_grid[y + j][x + k] == 1:
72                                     neighbor_floor_count += 1
73                 if neighbor_floor_count < min_neighbors:
74                     grid[x][y] = 0
75     return grid
76
77
78 func make_noise_grid():
79     var noise_grid = []
80     var random
81     for x in range(MAP_W):
82         random = randi() % 100
83         noise_grid.append([random])
84     for y in range(MAP_H):
85         random = randi() % 100
86         for x in range(MAP_W):
87             noise_grid[x].append(random)
88     return noise_grid
89
```

```
60 func clean_edges(grid):
61     var min_neighbors = 13
62     var temp_grid = grid
63     for j in range(MAP_H):
64         for k in range(MAP_W):
65             var neighbor_floor_count = 0
66             for y in [-2, -1, 0, 1, 2]:
67                 for x in [-2, -1, 0, 1, 2]:
68                     if BOUNDS.has_point(Vector2(x + k, y + j)):
69                         if y != j or x != k:
70                             if temp_grid[y + j][x + k] == 1:
71                                 neighbor_floor_count += 1
72             if neighbor_floor_count < min_neighbors:
73                 grid[j][k] = 0
74     return grid
75
76
77
78 func make_noise_grid():
79     var noise_grid = []
80     var random
81     for x in range(MAP_W):
82         random = randi() % 100
83         noise_grid.append([random])
84     for y in range(MAP_H):
85         random = randi() % 100
86         for x in range(MAP_W):
87             noise_grid[x].append(random)
88     return noise_grid
89
```

