

# Creating worlds.

Part 2



Creating levels.

# Procedural level generation.



# Terraria

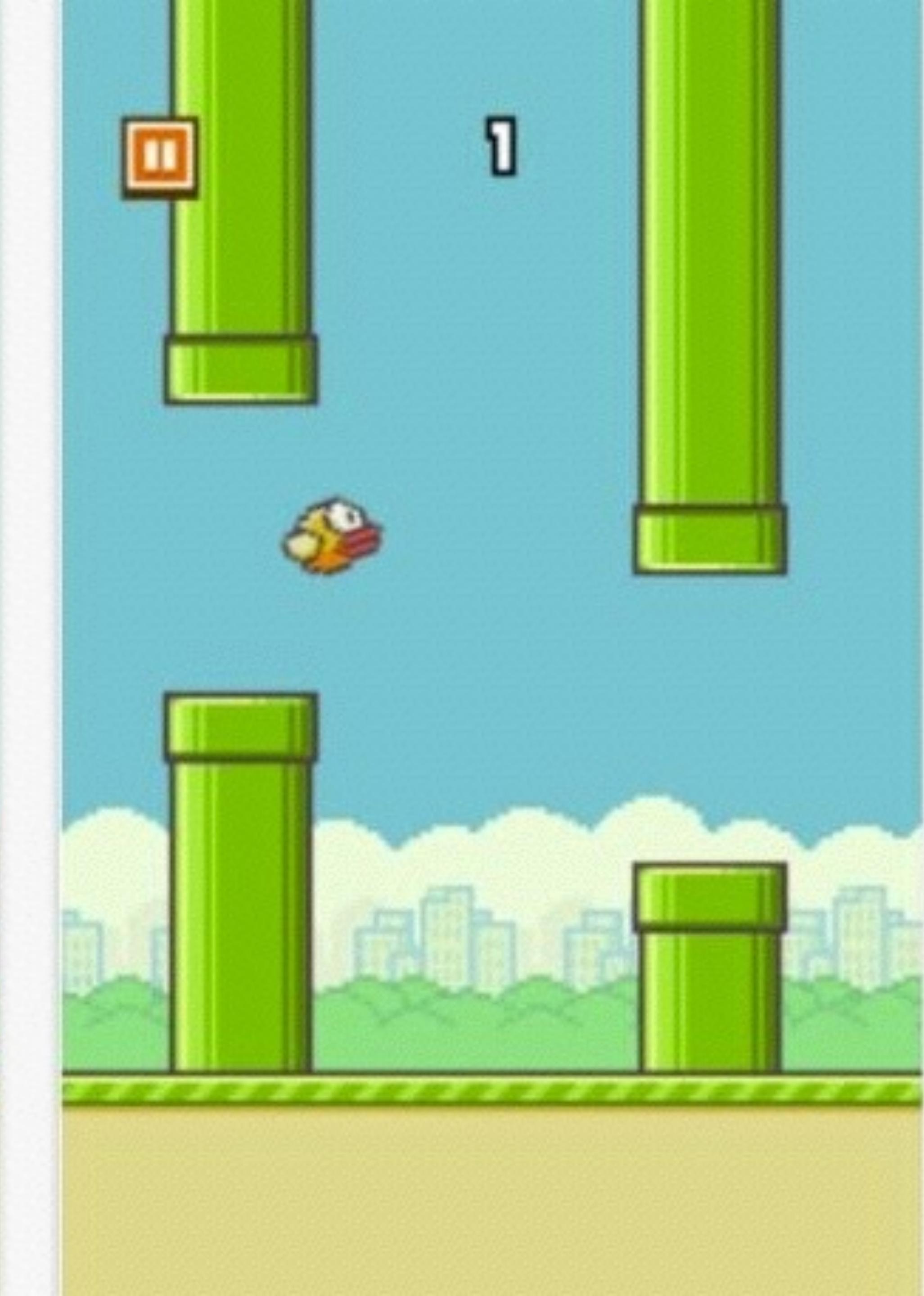
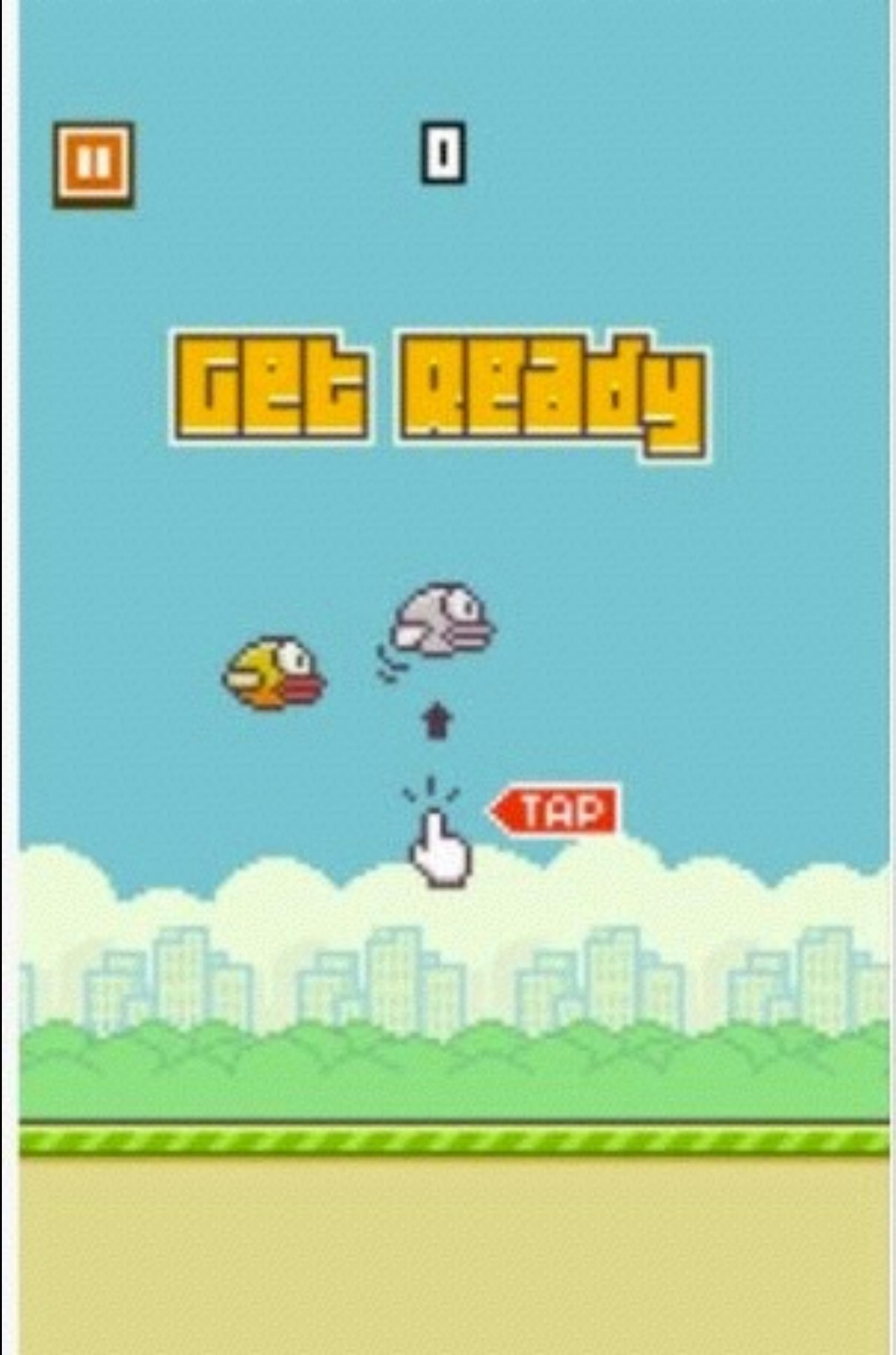




\$0

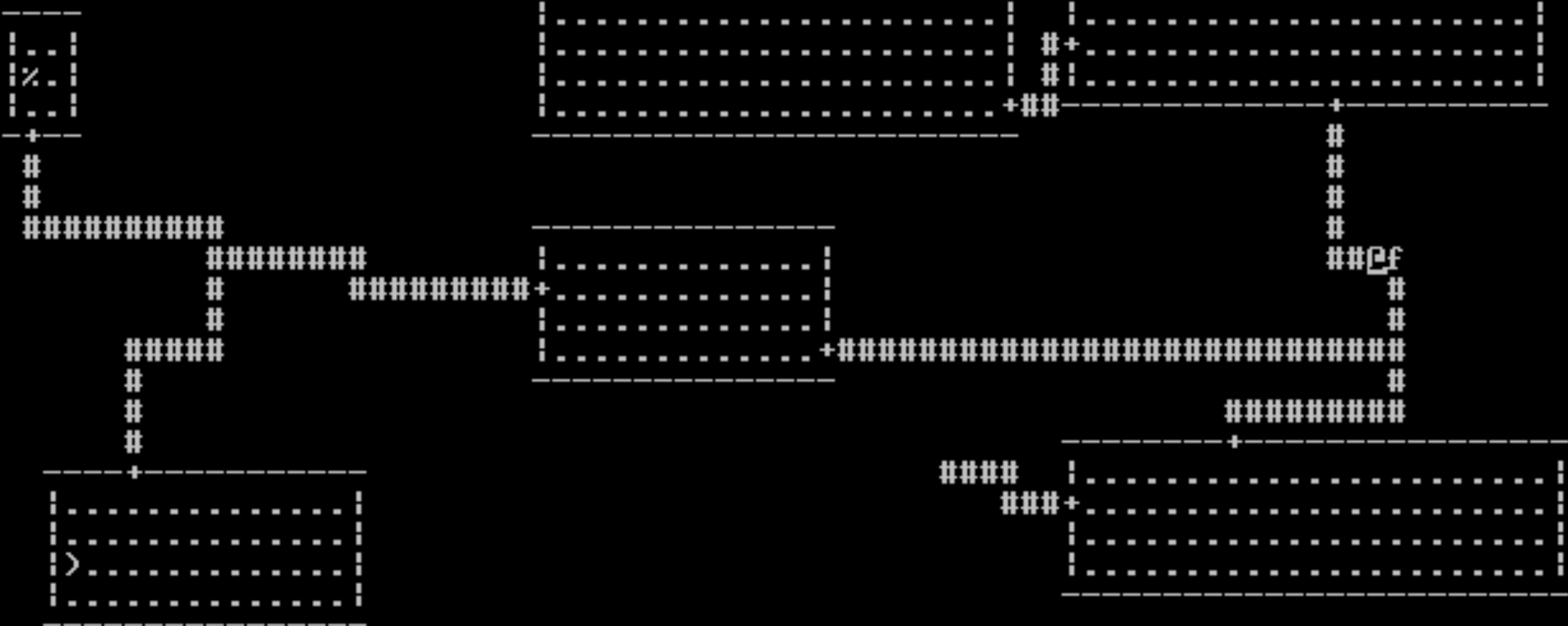








The killer frog hit.



Level: 3 Gold: 496 Hp: 32<37> Ac: 1 Exp: 4/47 Vol: 65%  
Str: 11< 11> Dex: 14< 14> Wis: 12< 12> Con: 18< 18> Carry: 54<150>





An overview of procedural generation  
examples.

Snake.

NOKIA

P

G

-

A

V

1

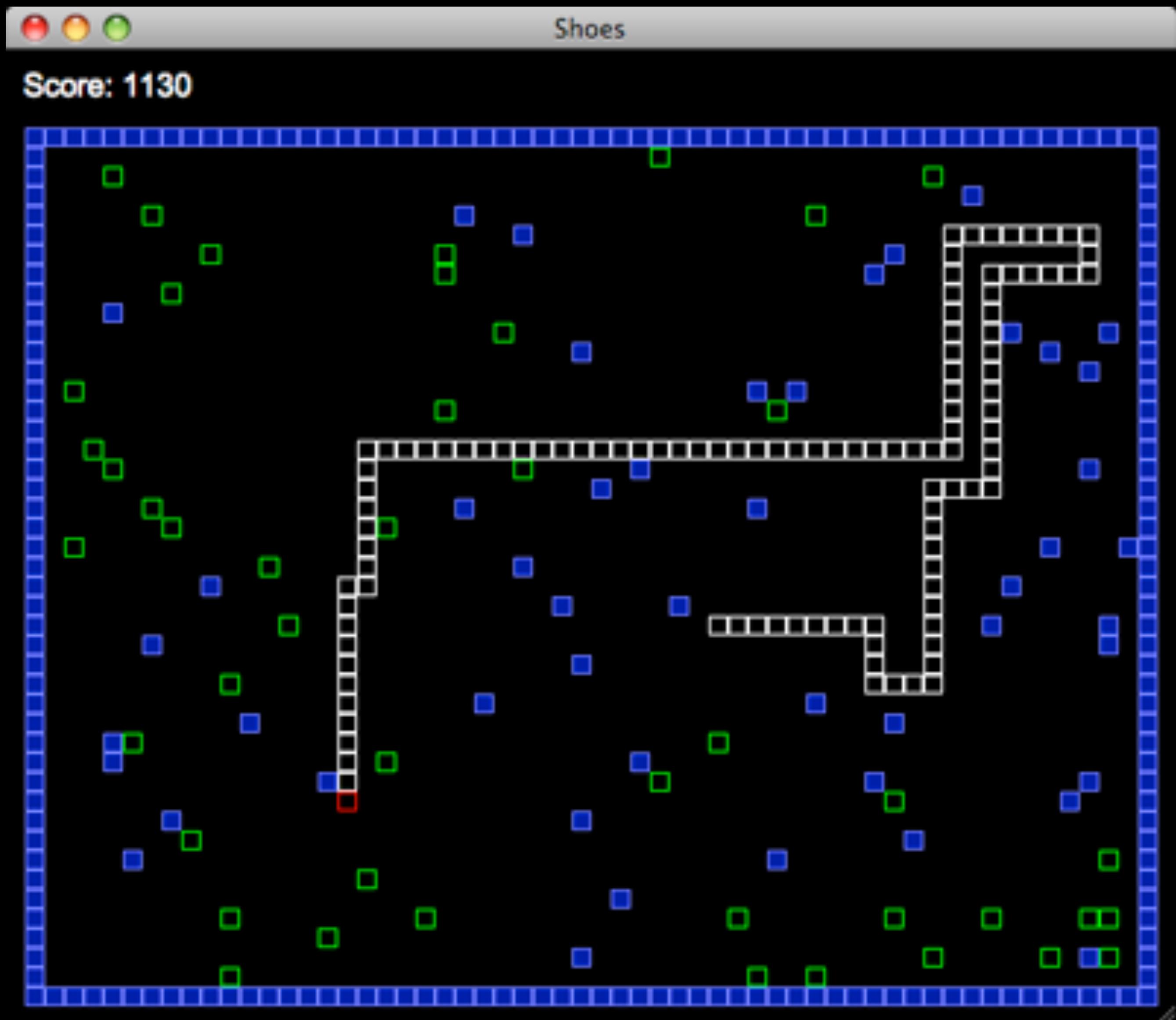
2

3

E

0

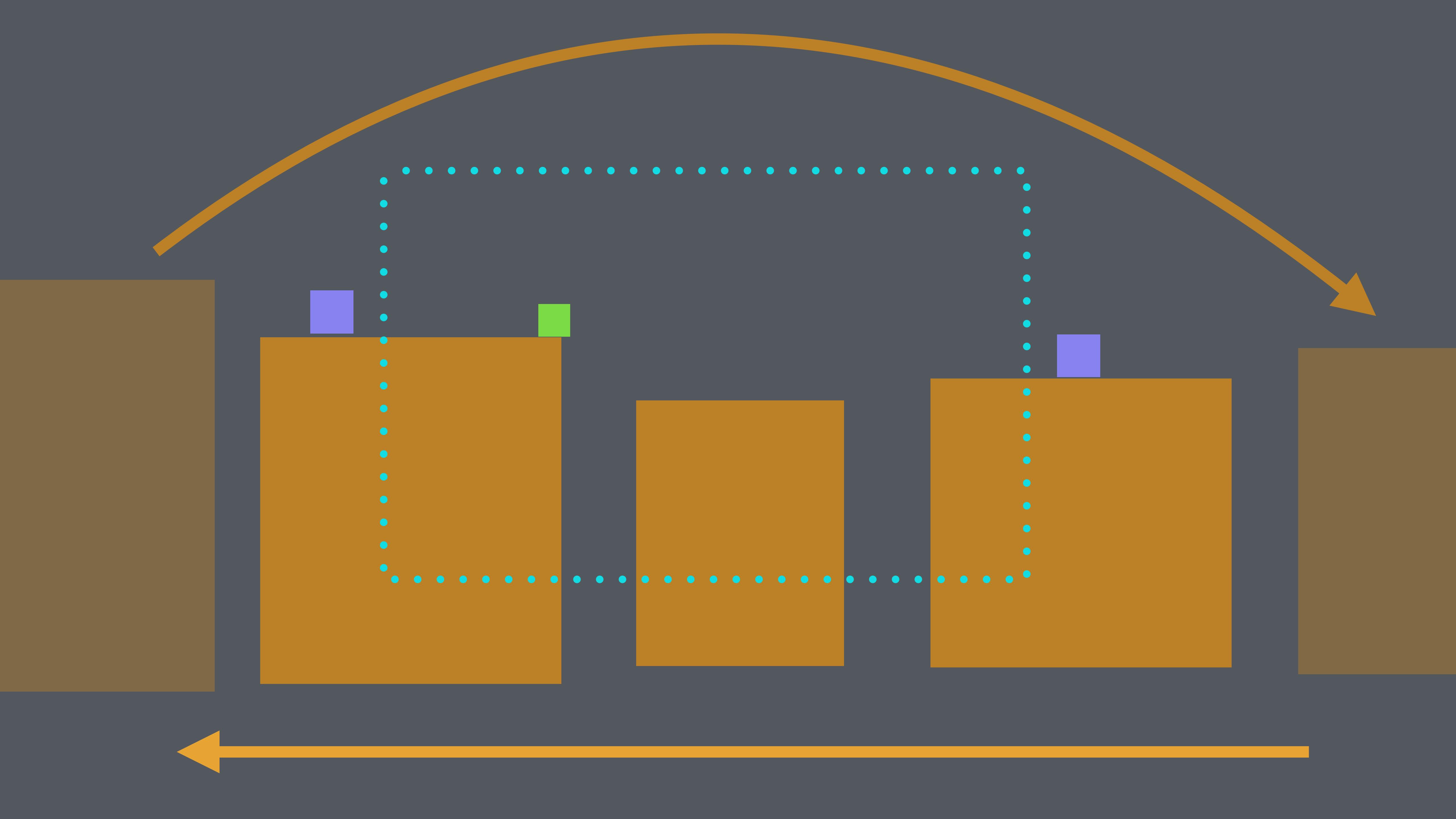




Canabalt.

151m







# Tuning Canabalt

<http://www.gamasutra.com/blogs/>

[AdamSaltsman/20100929/88155/](#)

[Tuning\\_Canabalt.php](#)

Spelunky.

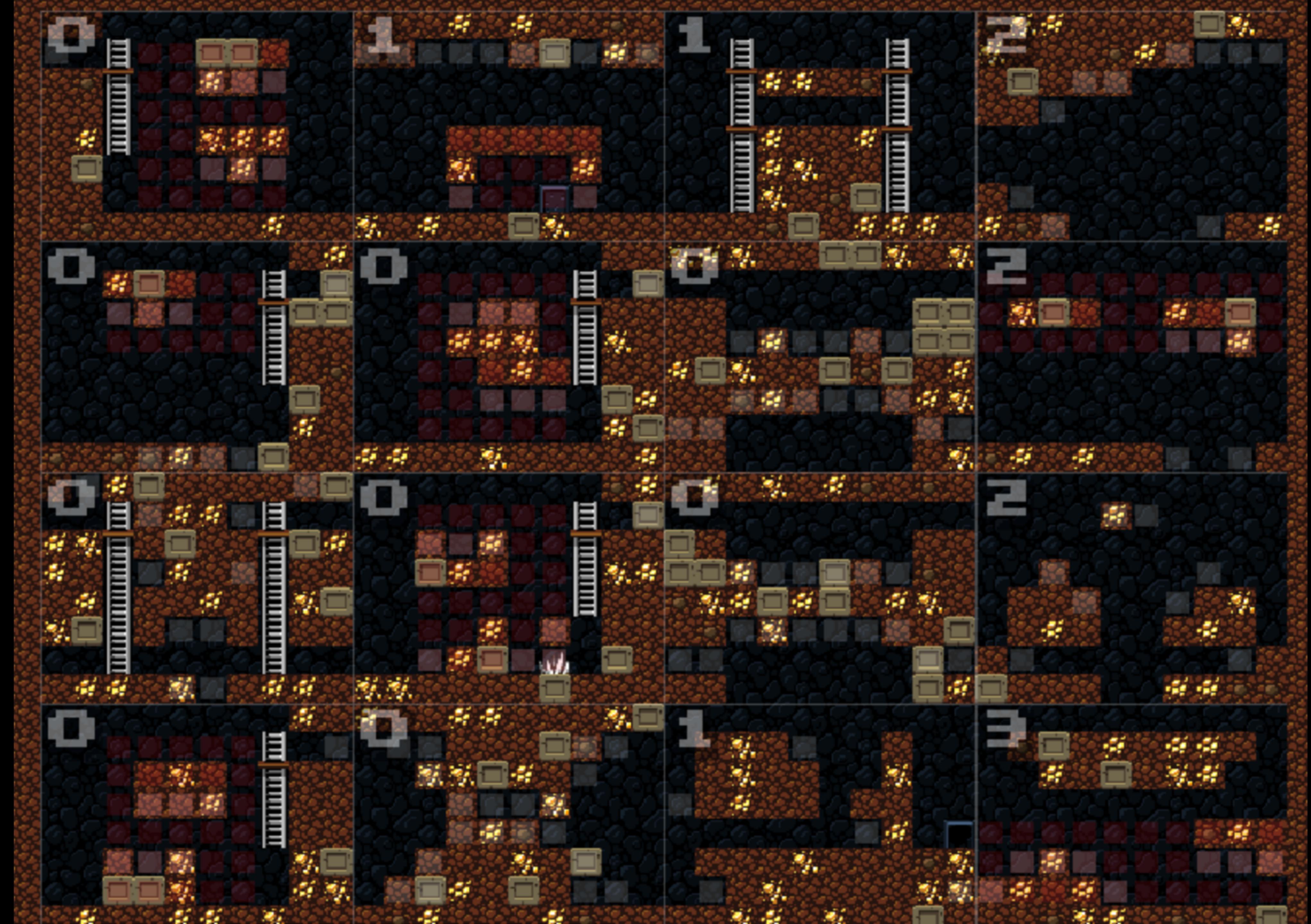


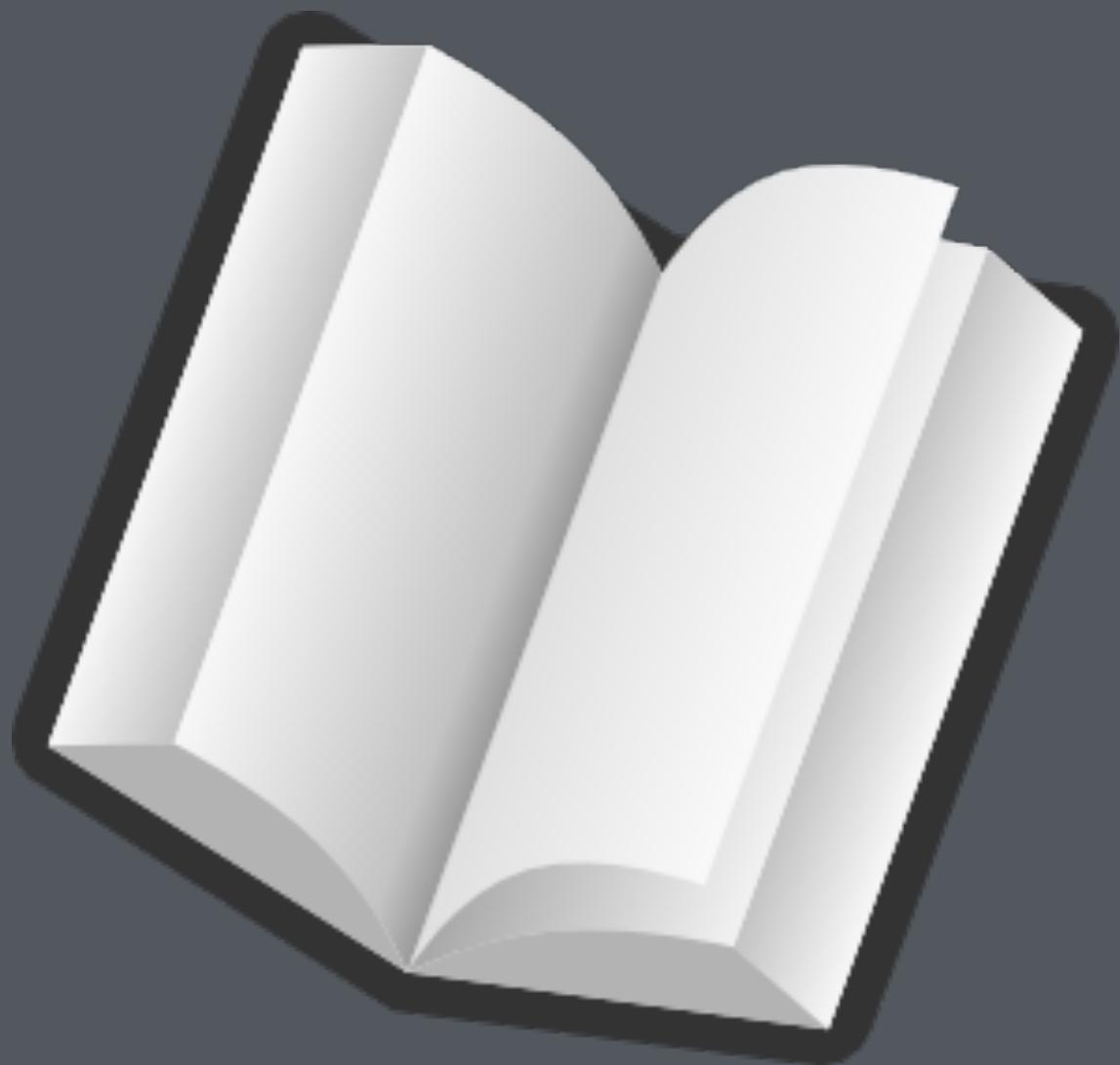
1 4 4

\$0







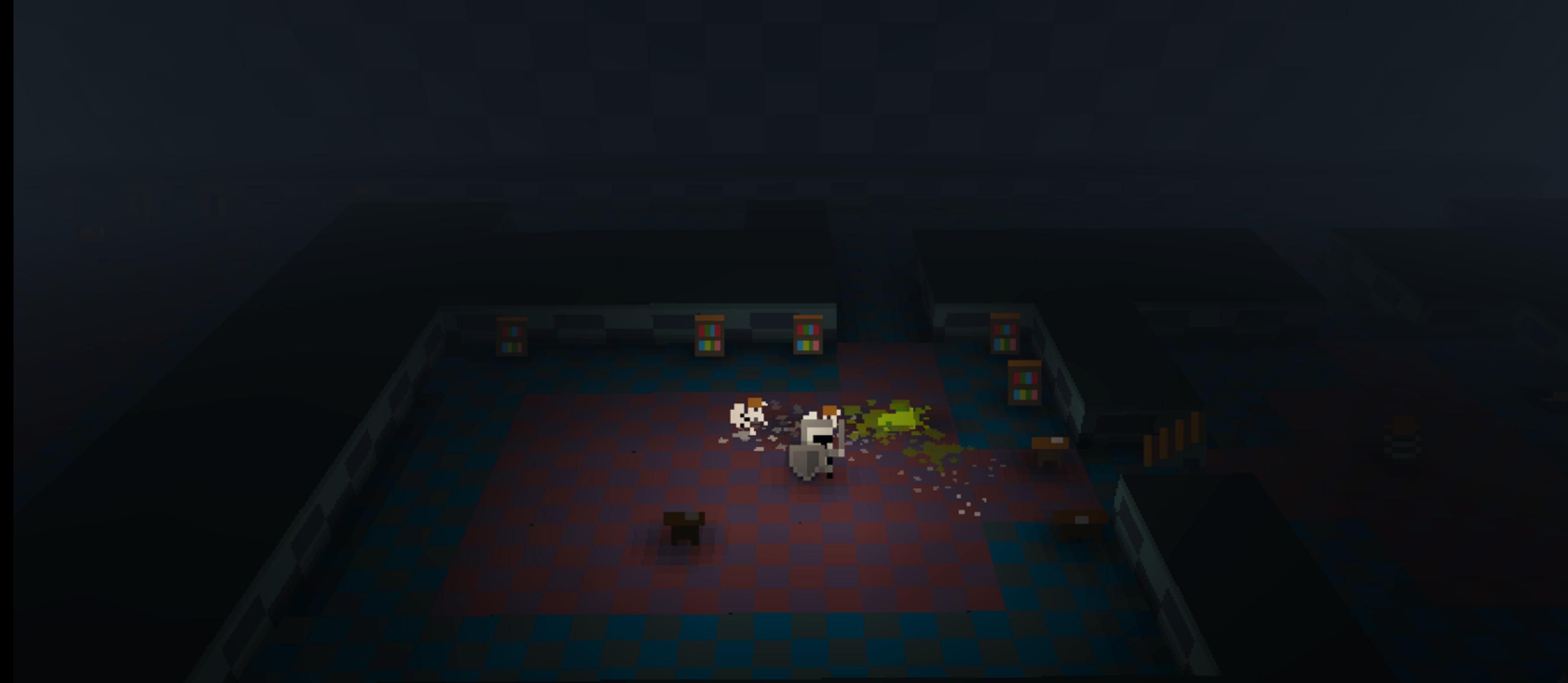


Level generation in **Spelunky**.

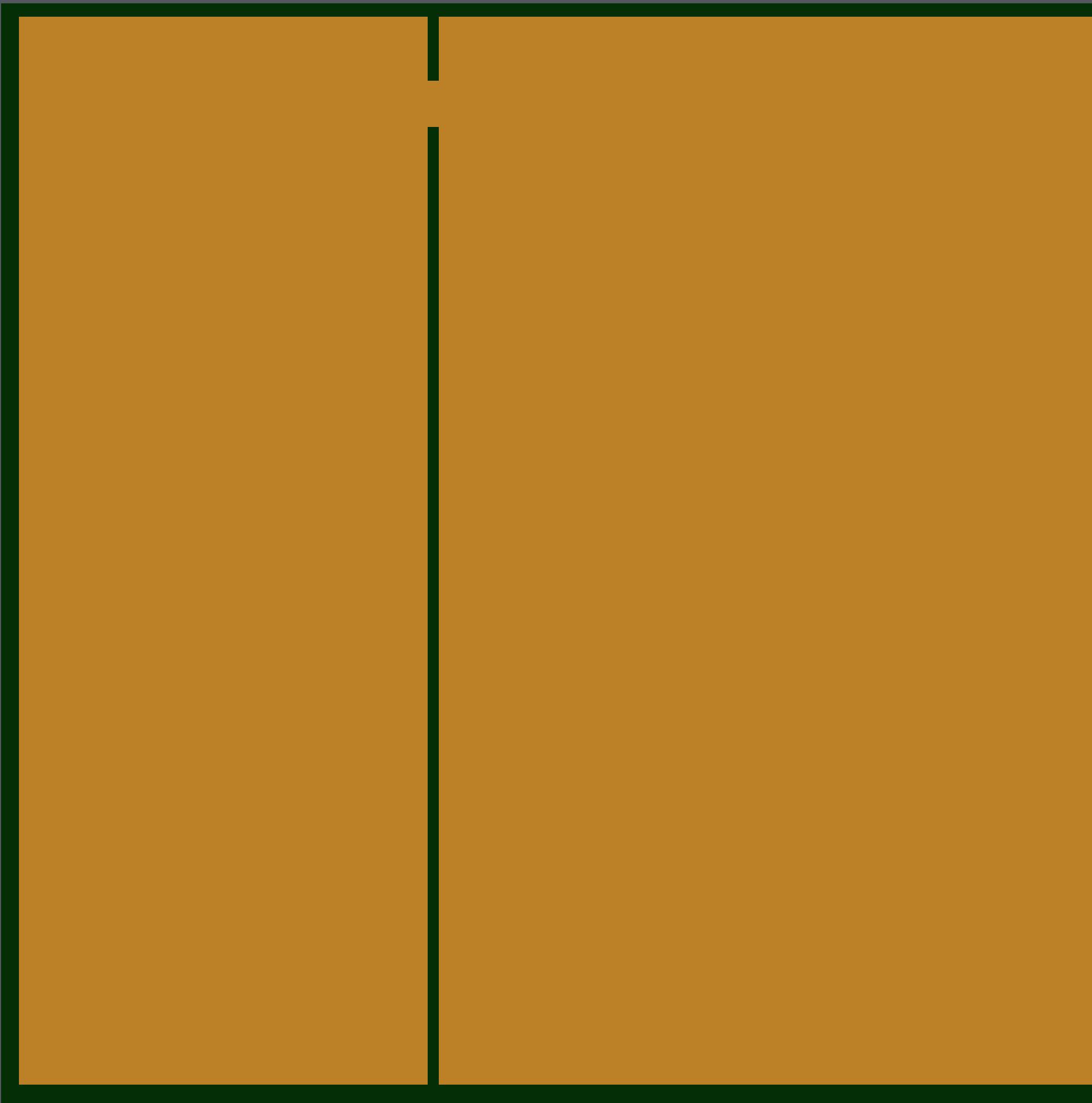
<http://tinysubversions.com/spelunkycGen/>

Bitworld.

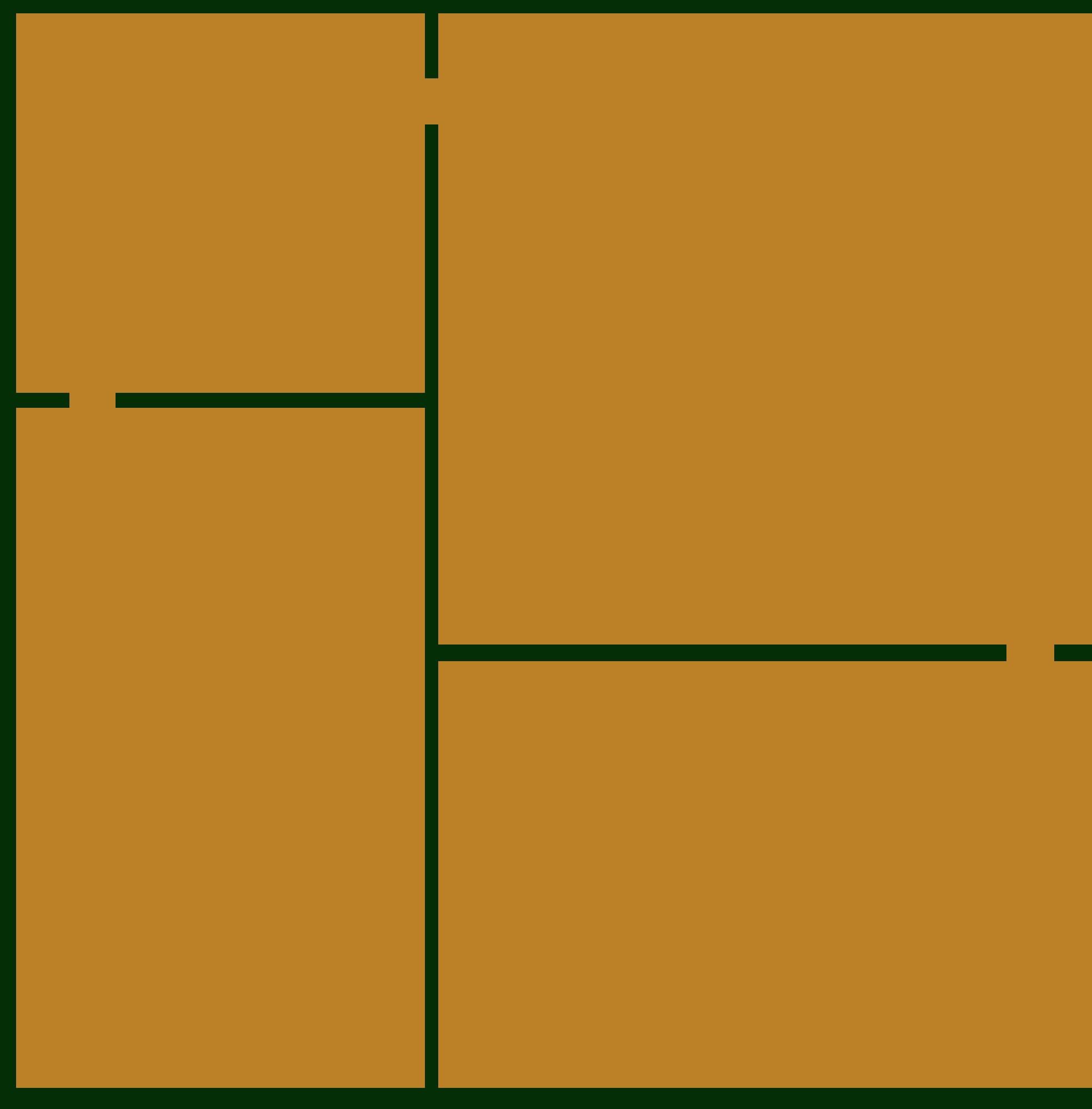
GOLD:1e



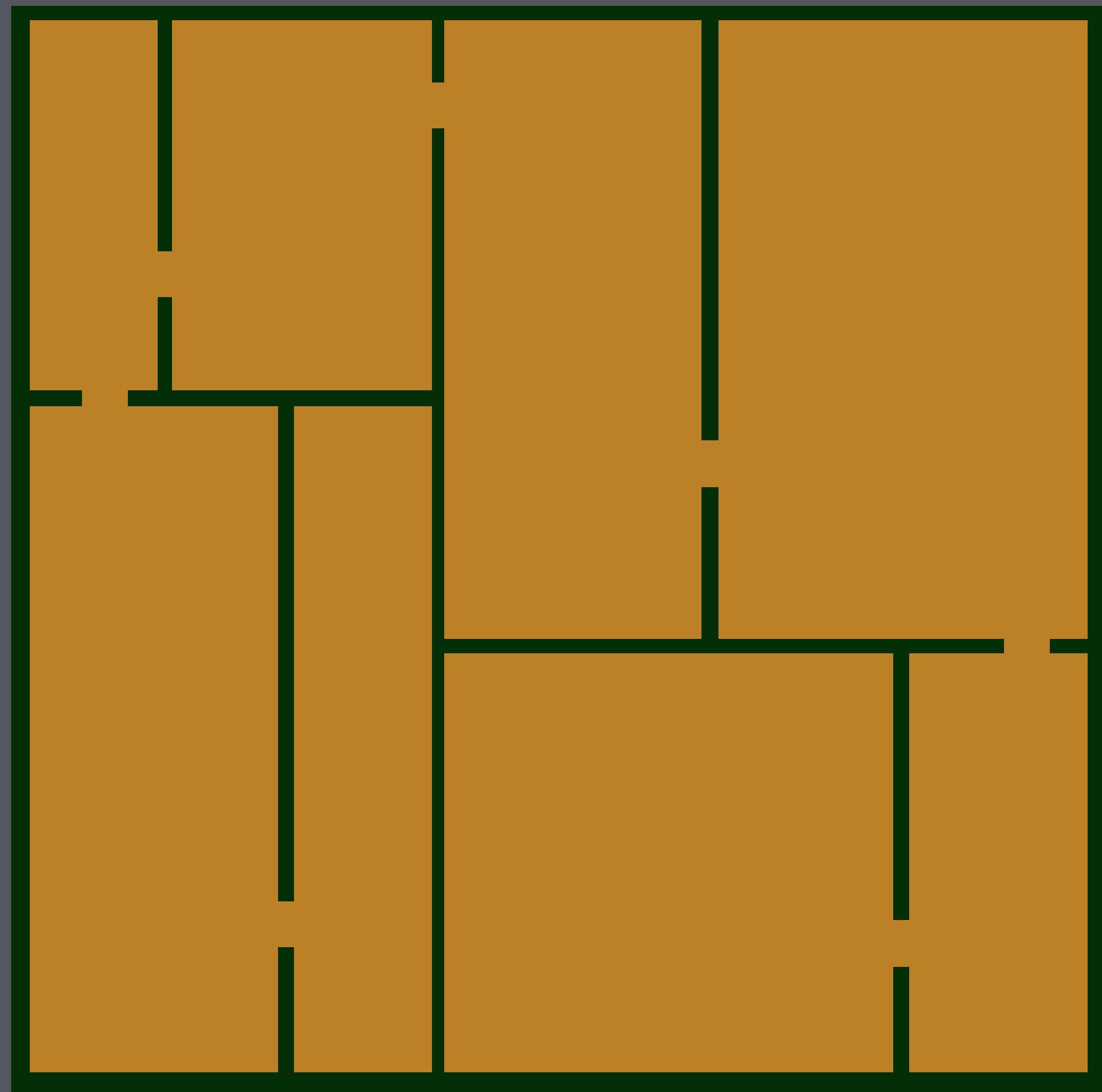




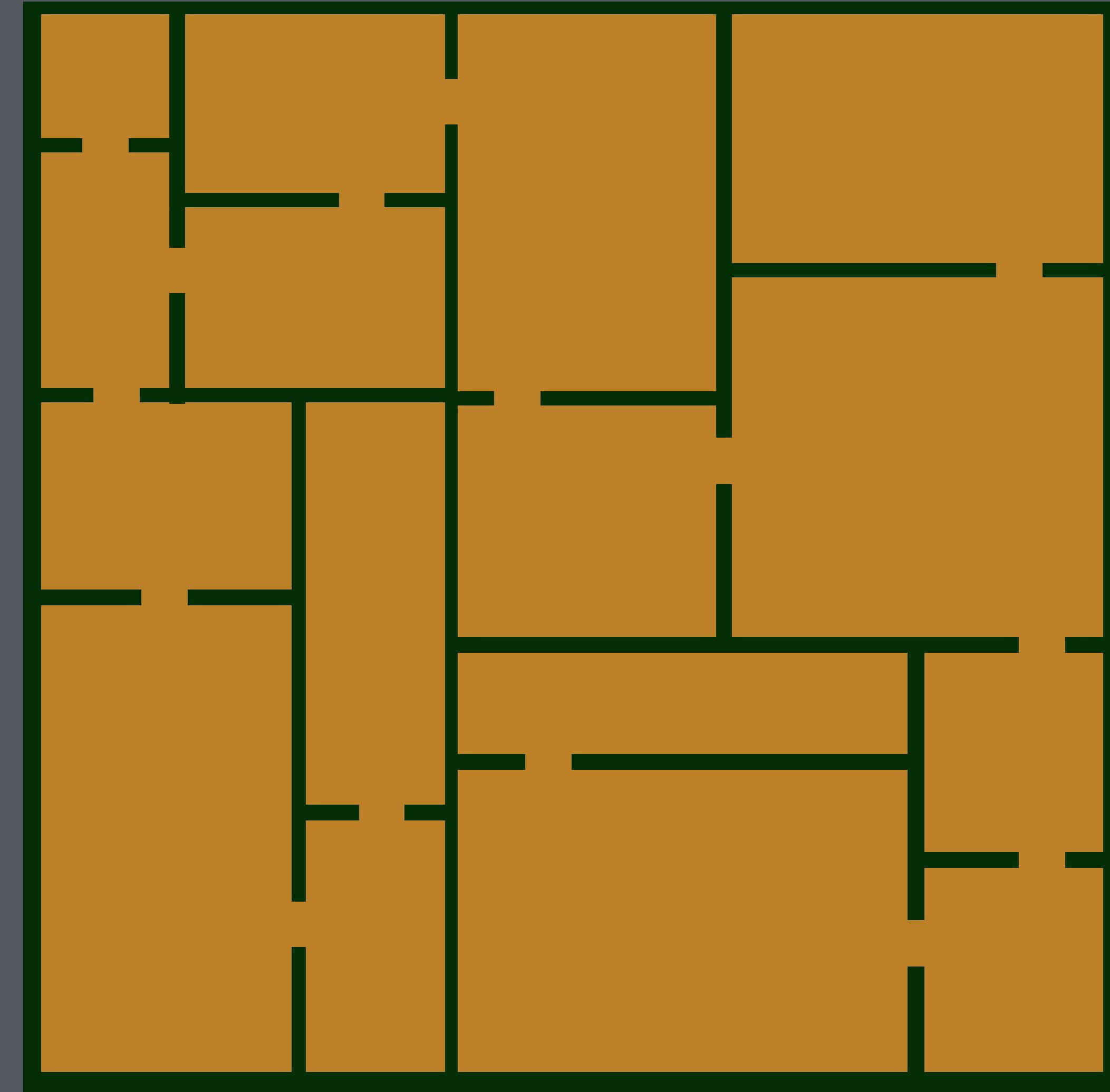
**Divide into two rooms** of random width and connect them together.



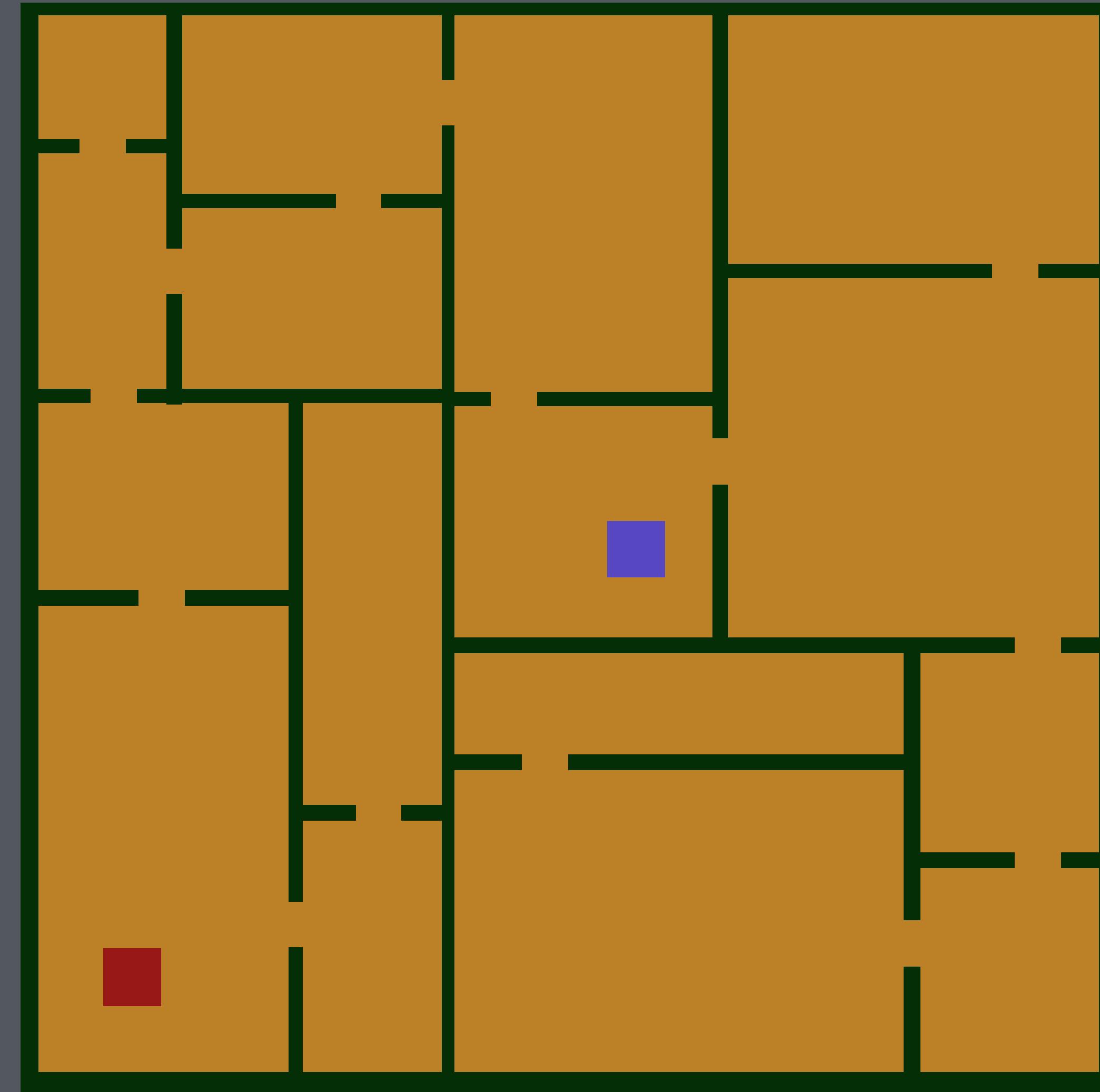
Do the same for the new rooms.



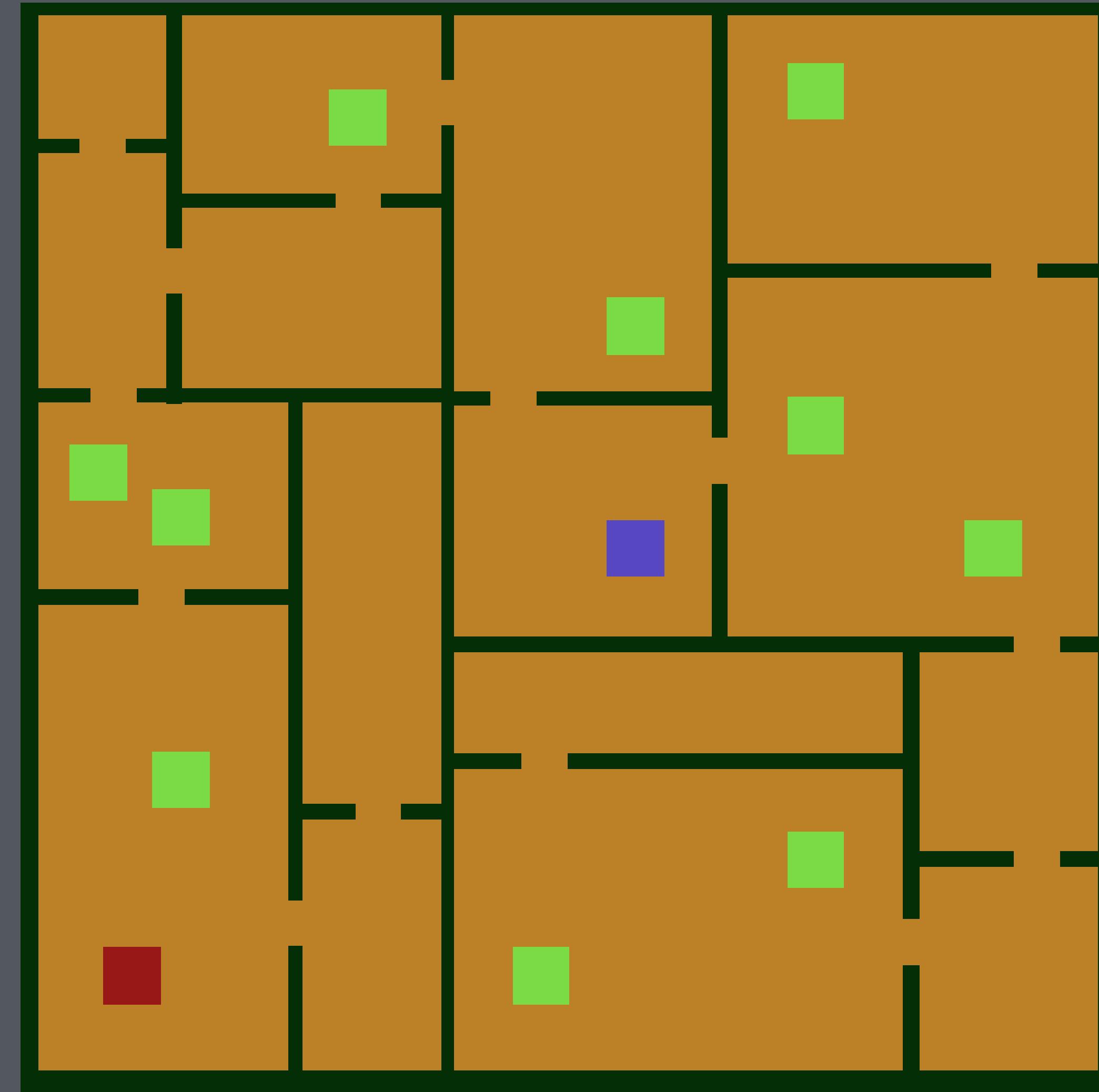
And again...



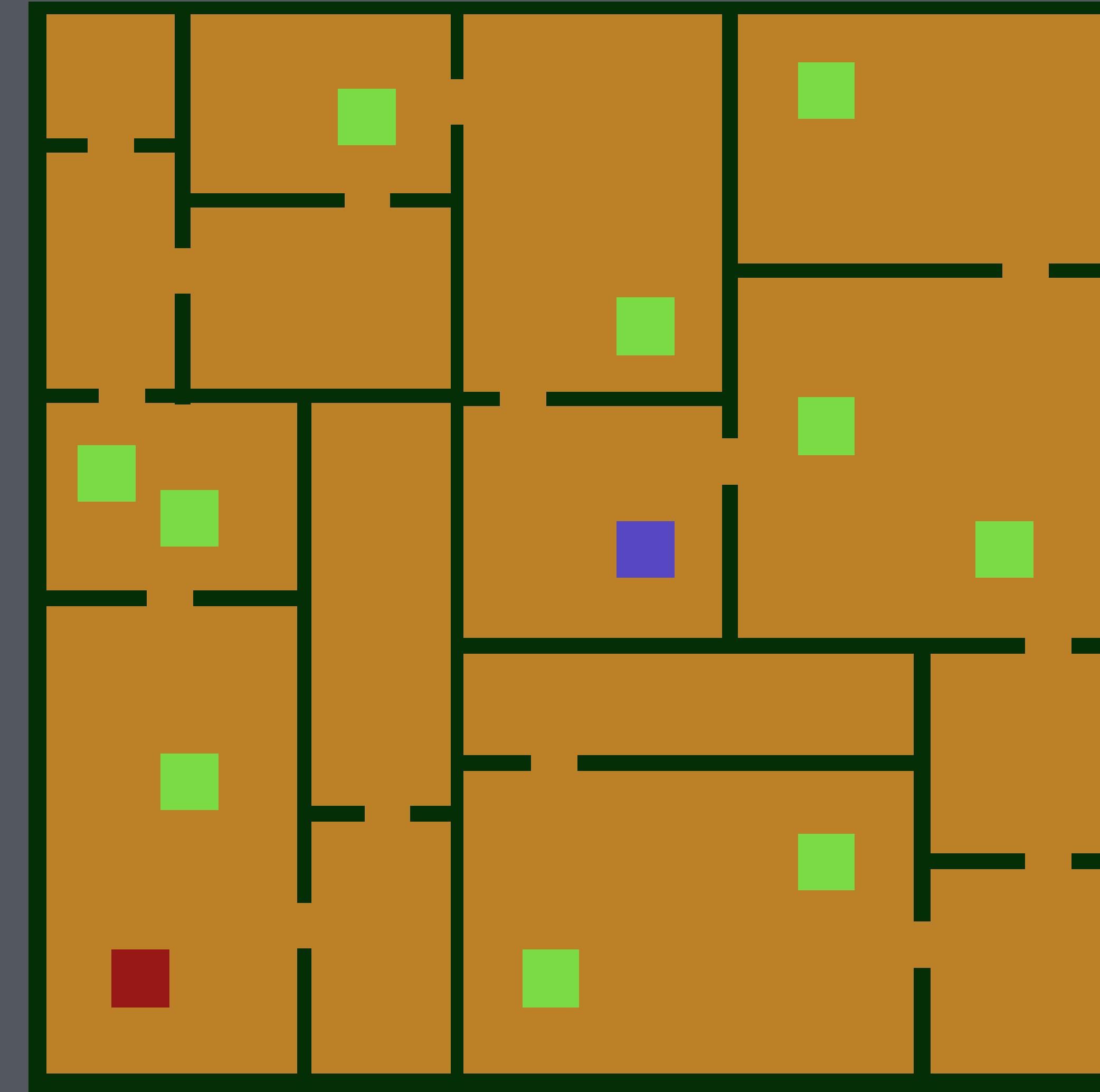
And again...



Place start and exit in a random tile.



Place enemies and other entities on random tiles.

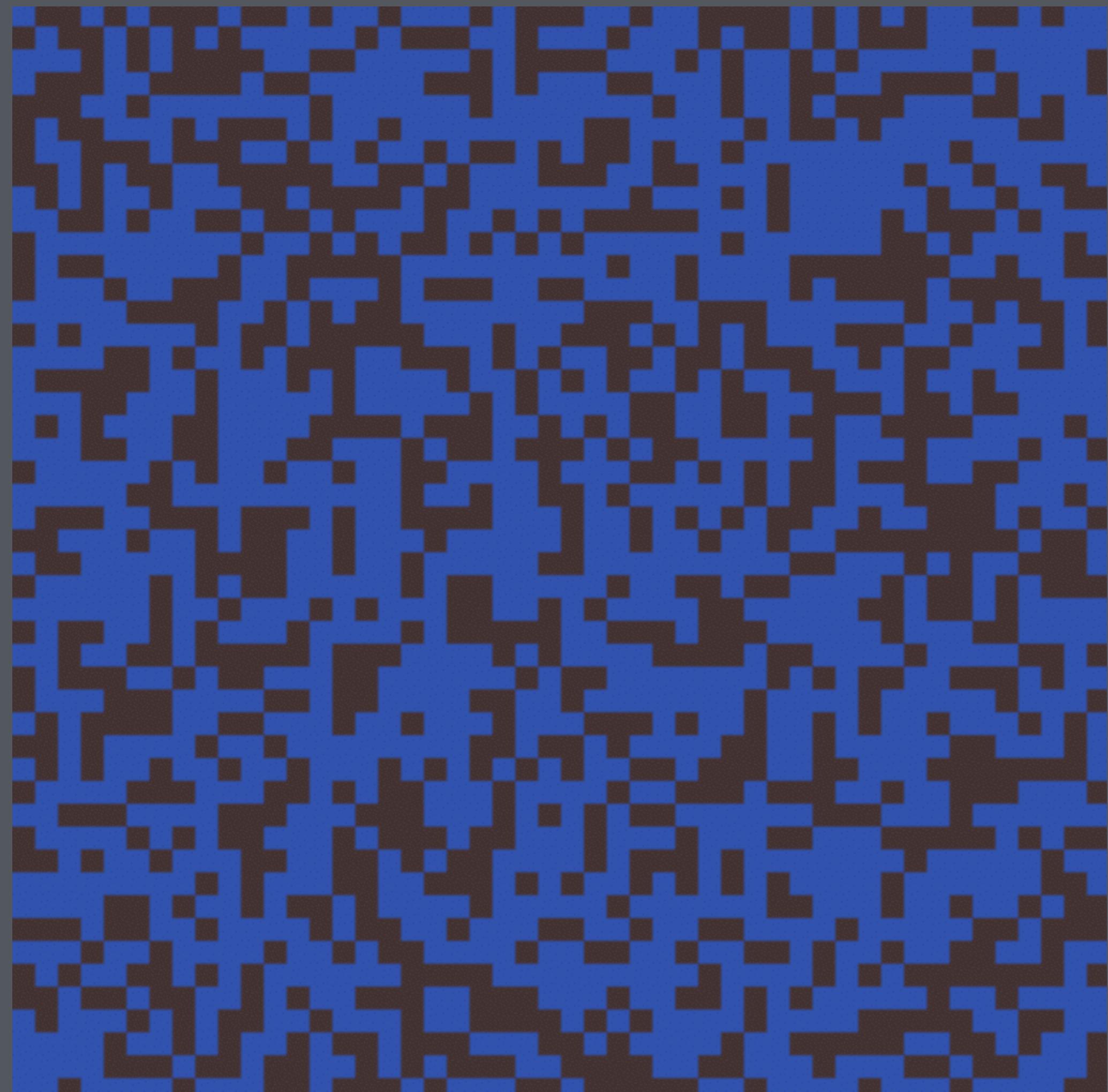


Procedural generation!

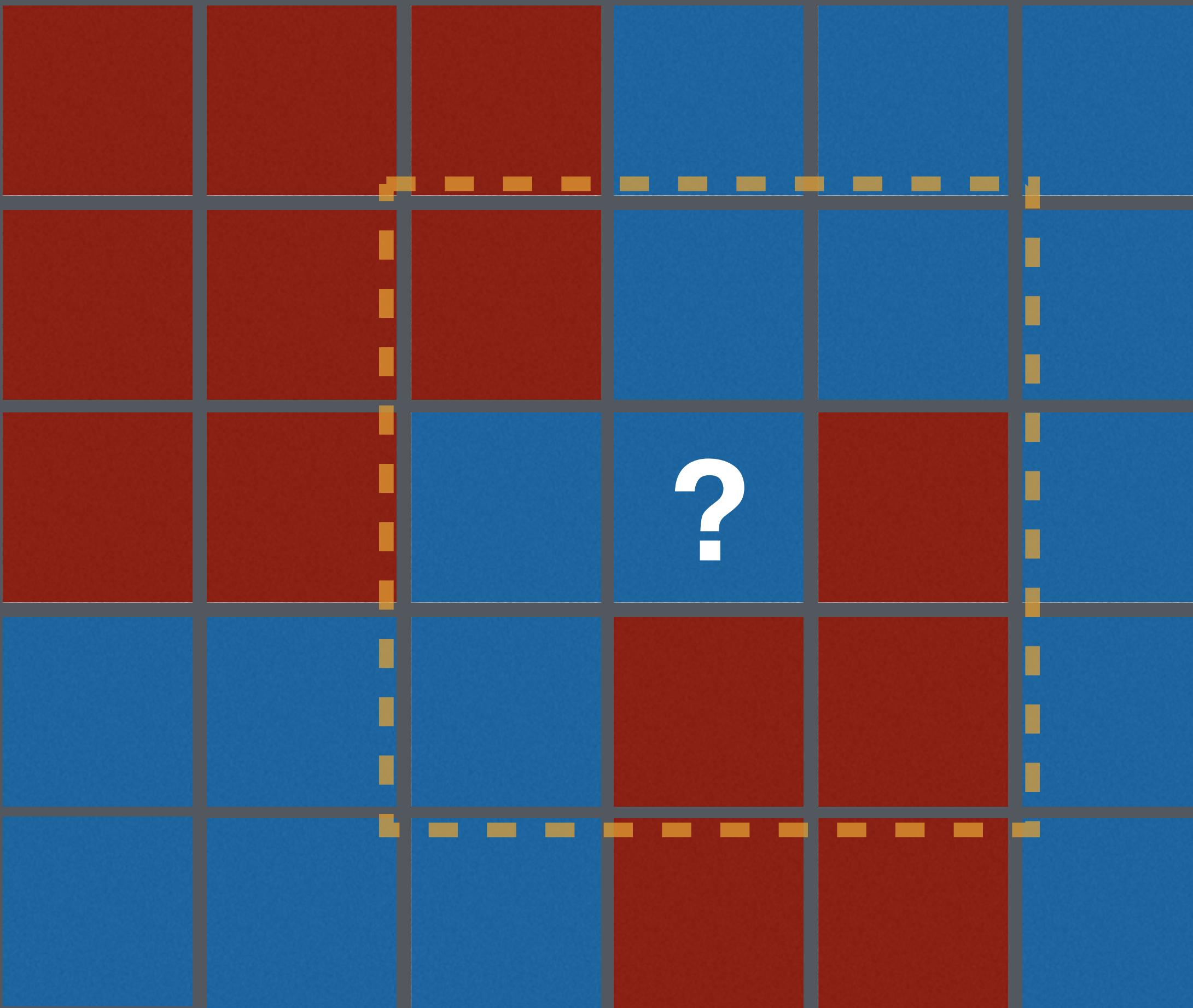
2D Caves using cellular  
automata.

Cellular automata.

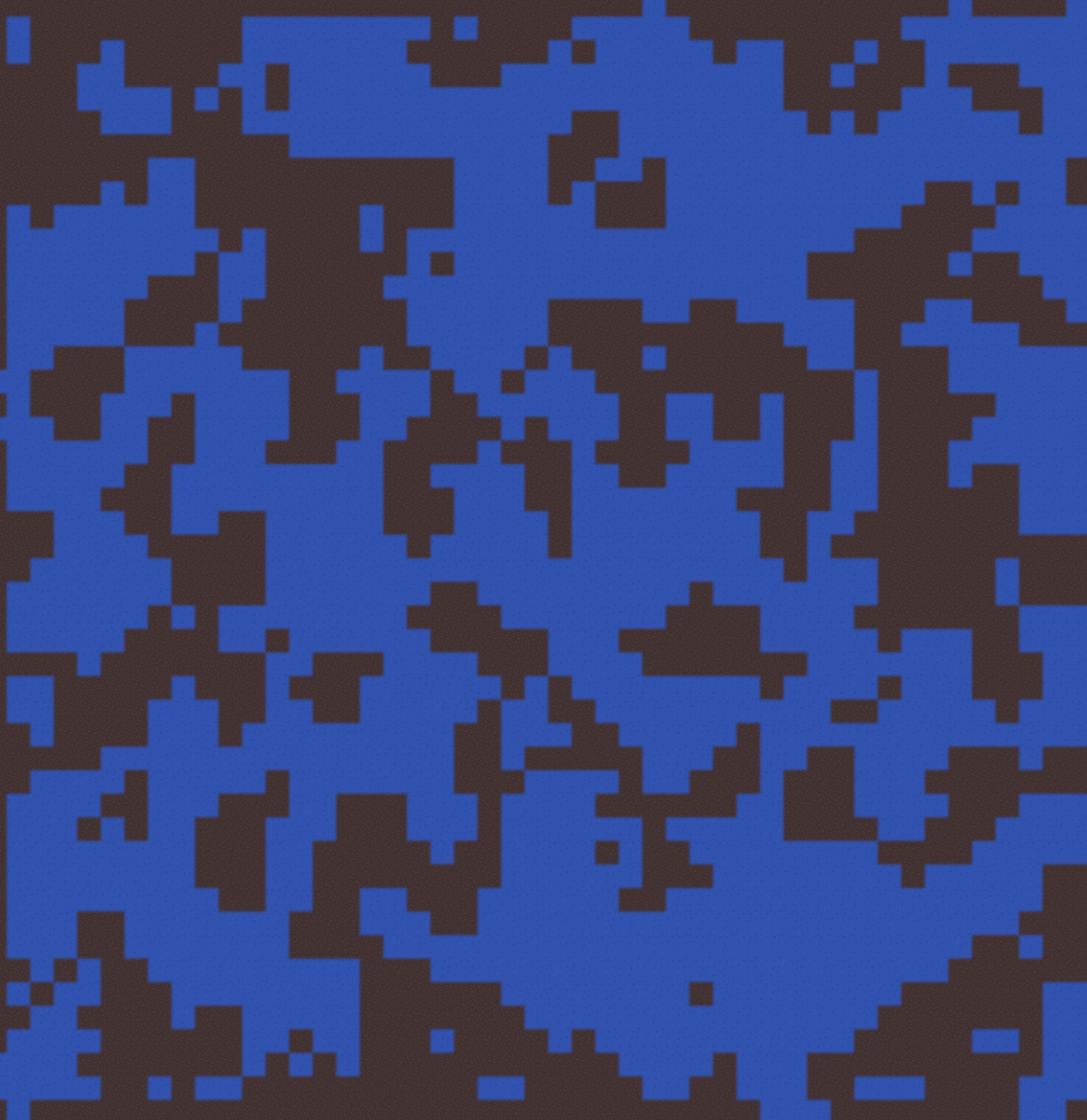




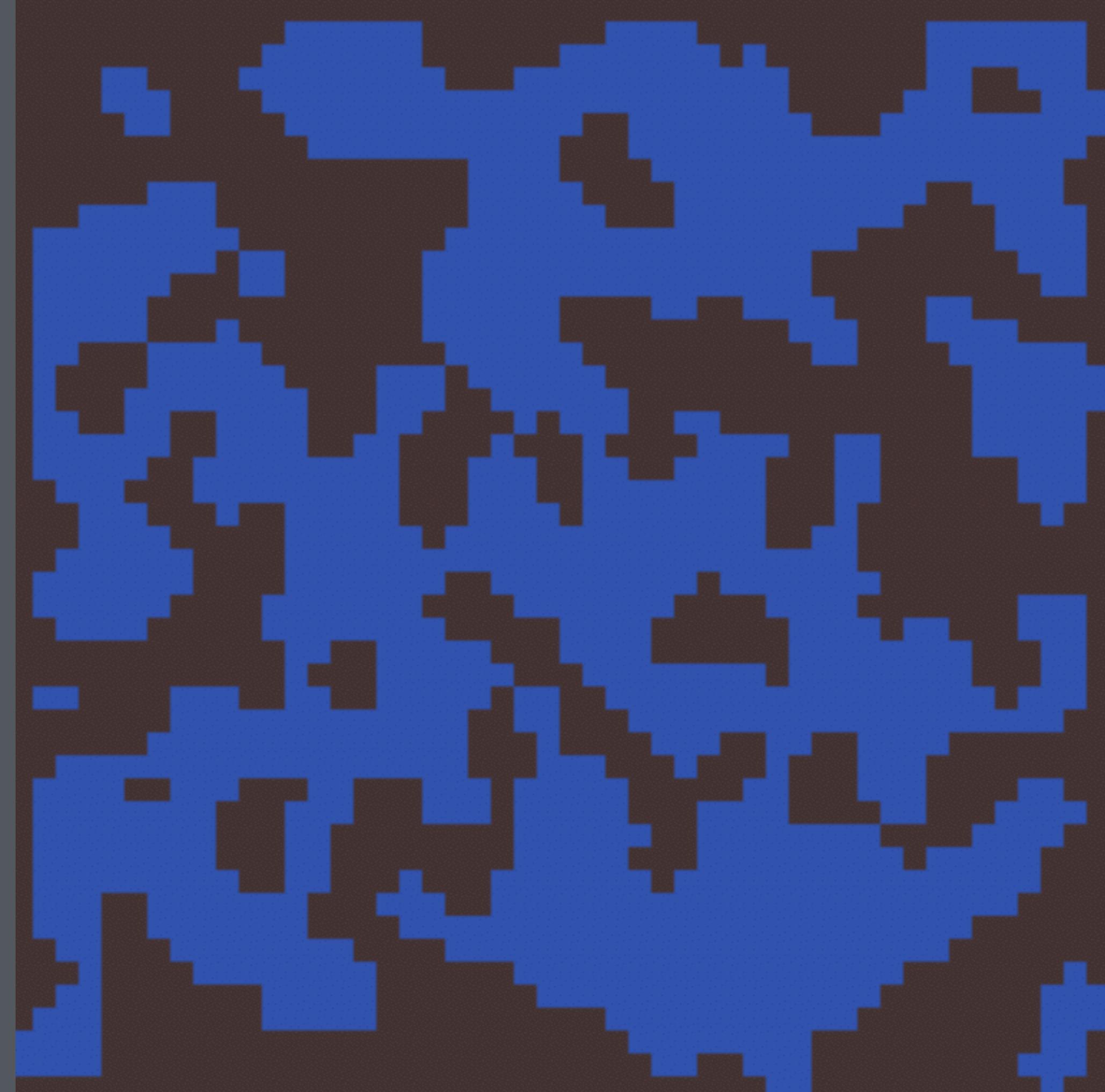
Fill tile map **randomly**.  
(50% chance solid)



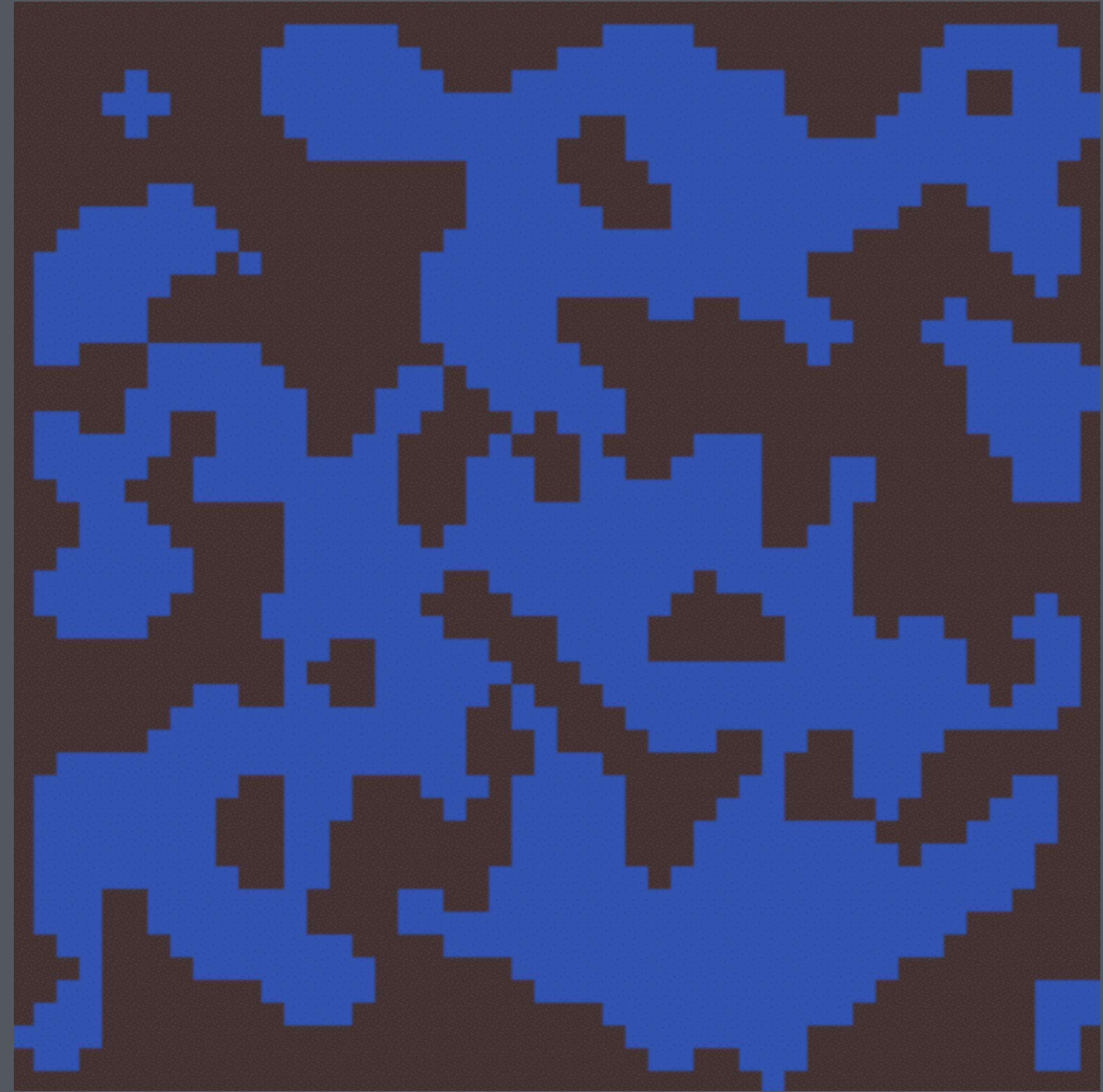
For every tile, **count all of its neighbors**, and if count is **larger or lower** than a certain **threshold**, **create a new tile or kill the existing one**.



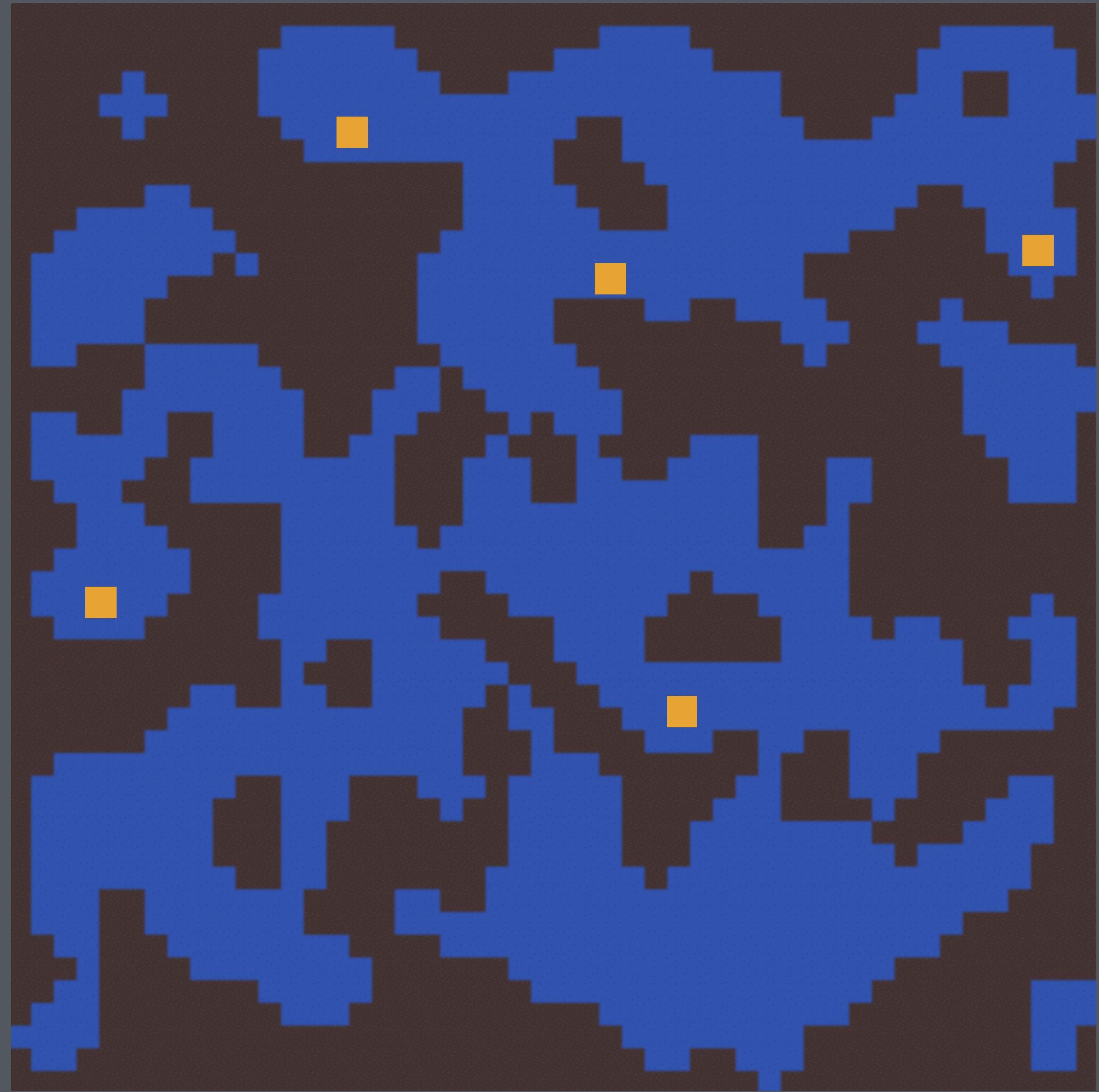
Do this for the entire tile map



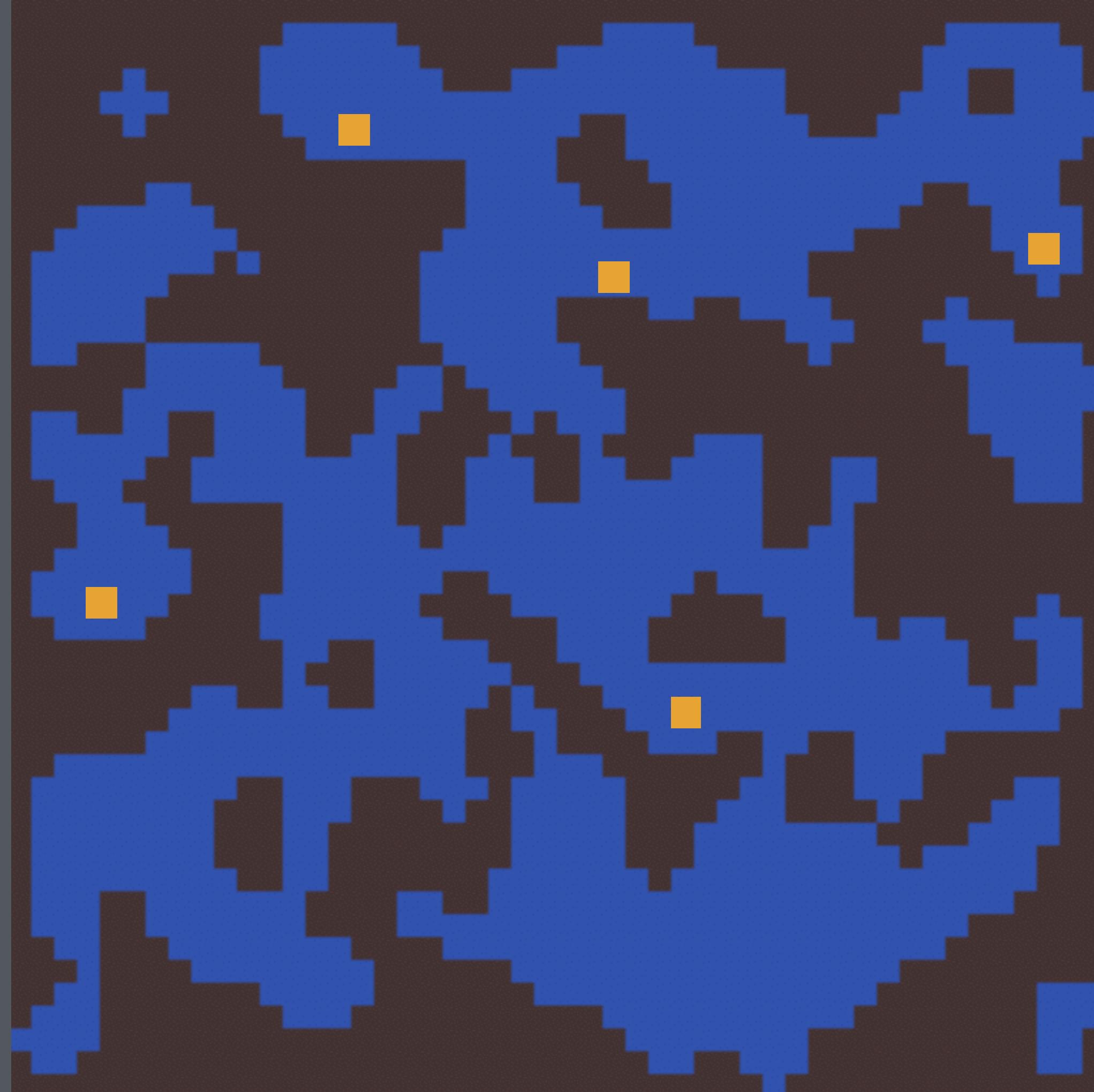
Repeat again.



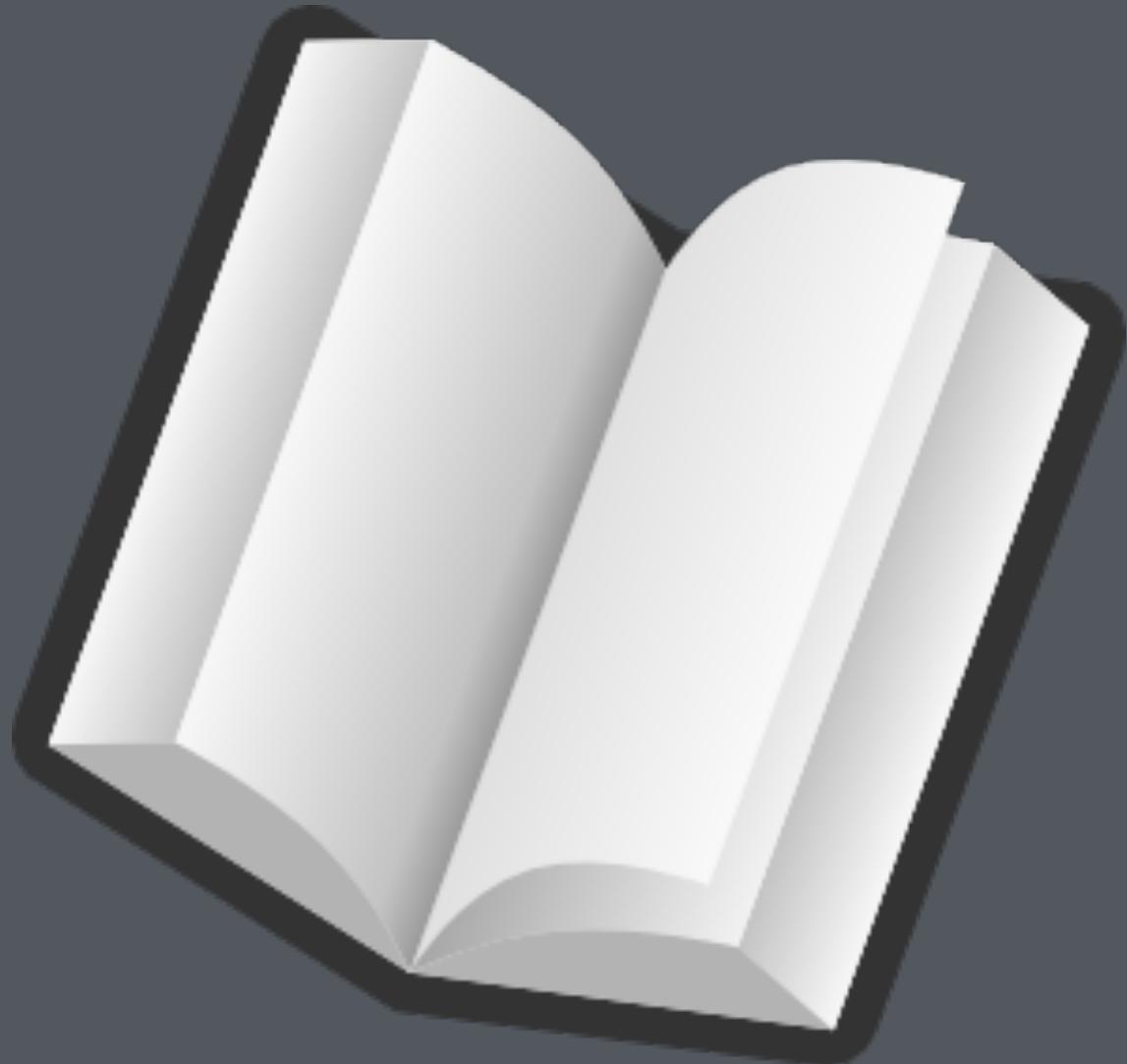
And again.



Place entities.



Procedural generation!

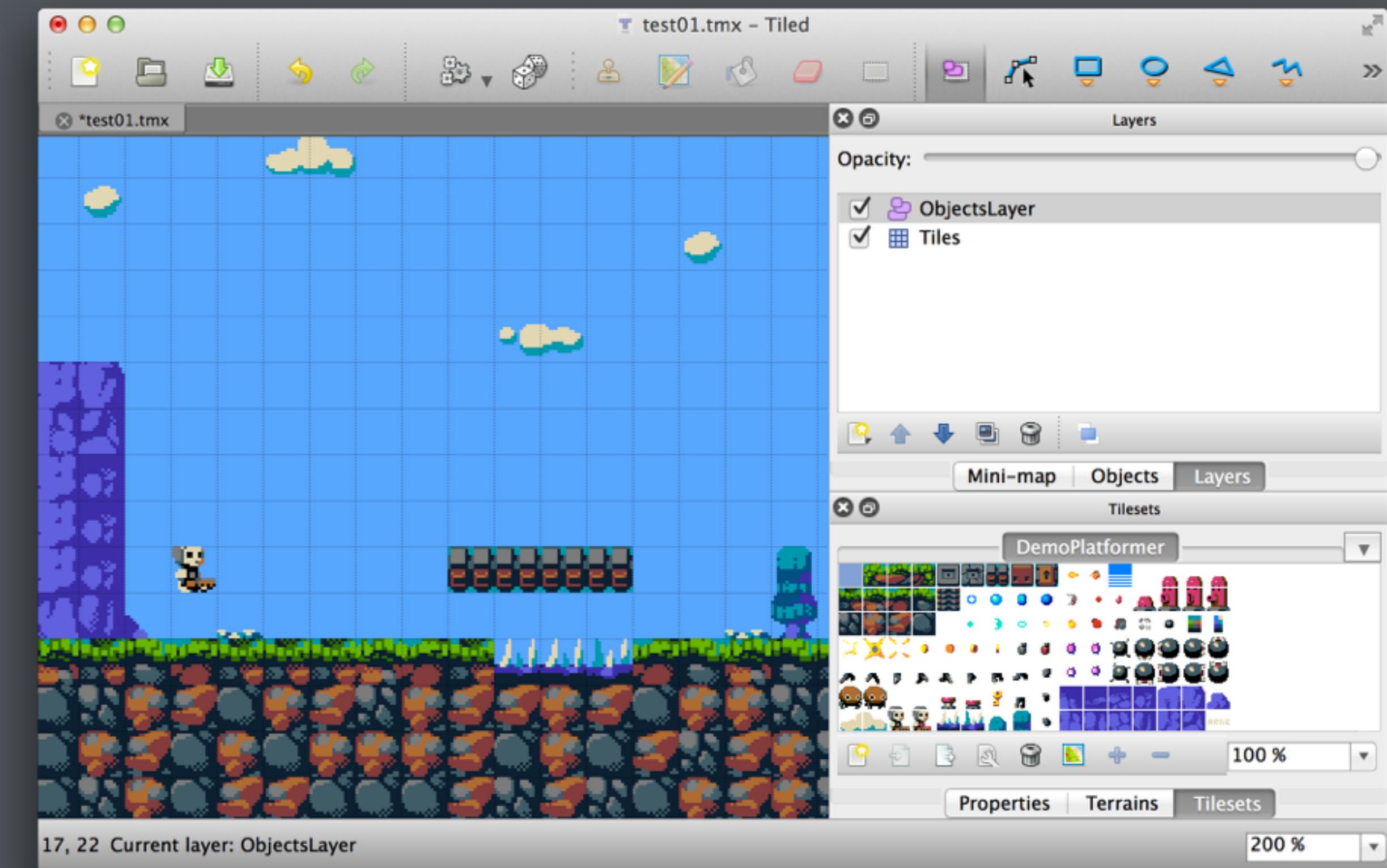


# Generate Random Cave Levels Using Cellular Automata

<http://gamedevelopment.tutsplus.com/tutorials/generate-random-cave-levels-using-cellular-automata--gamedev-9664>

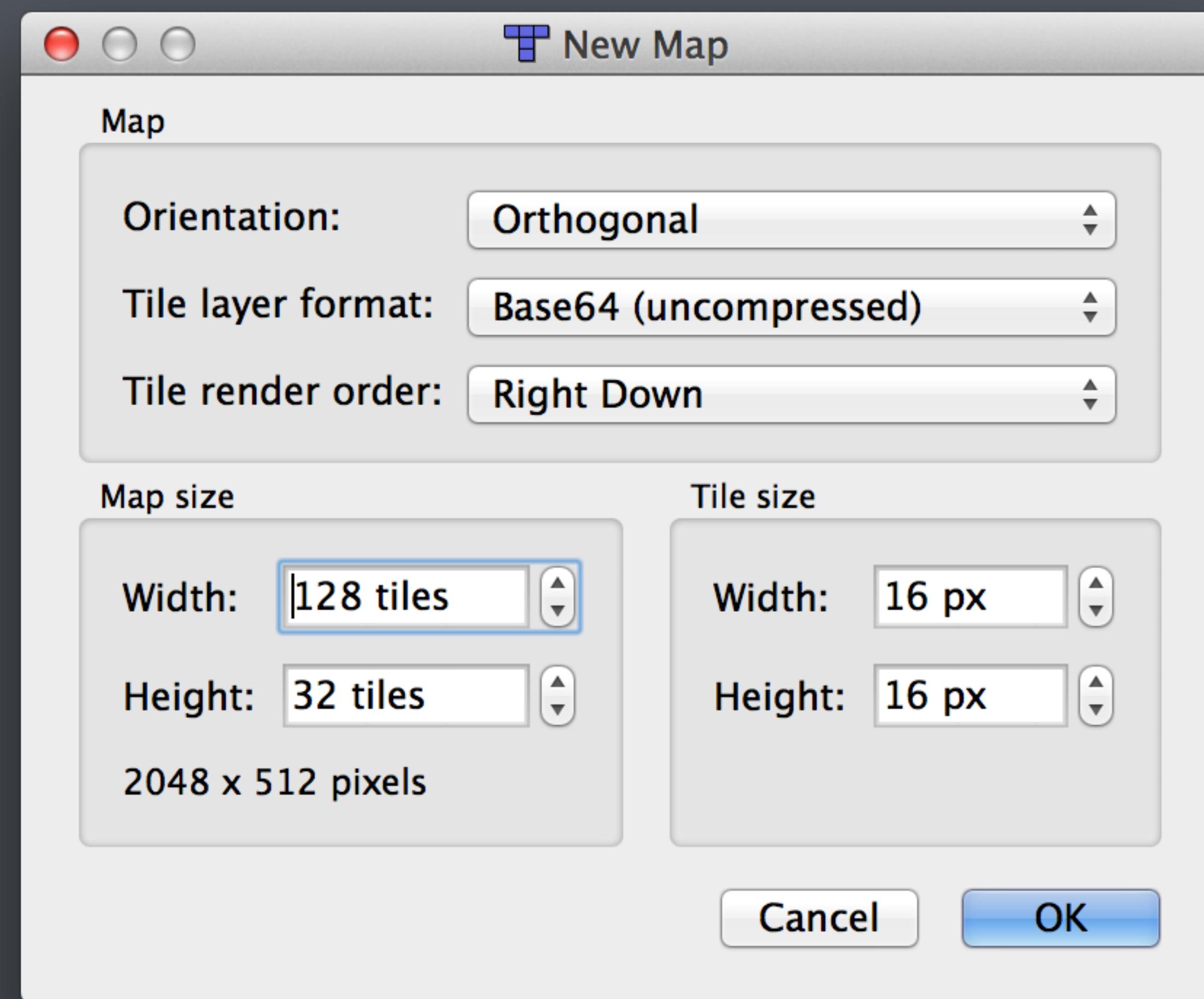
# Using a map editor.

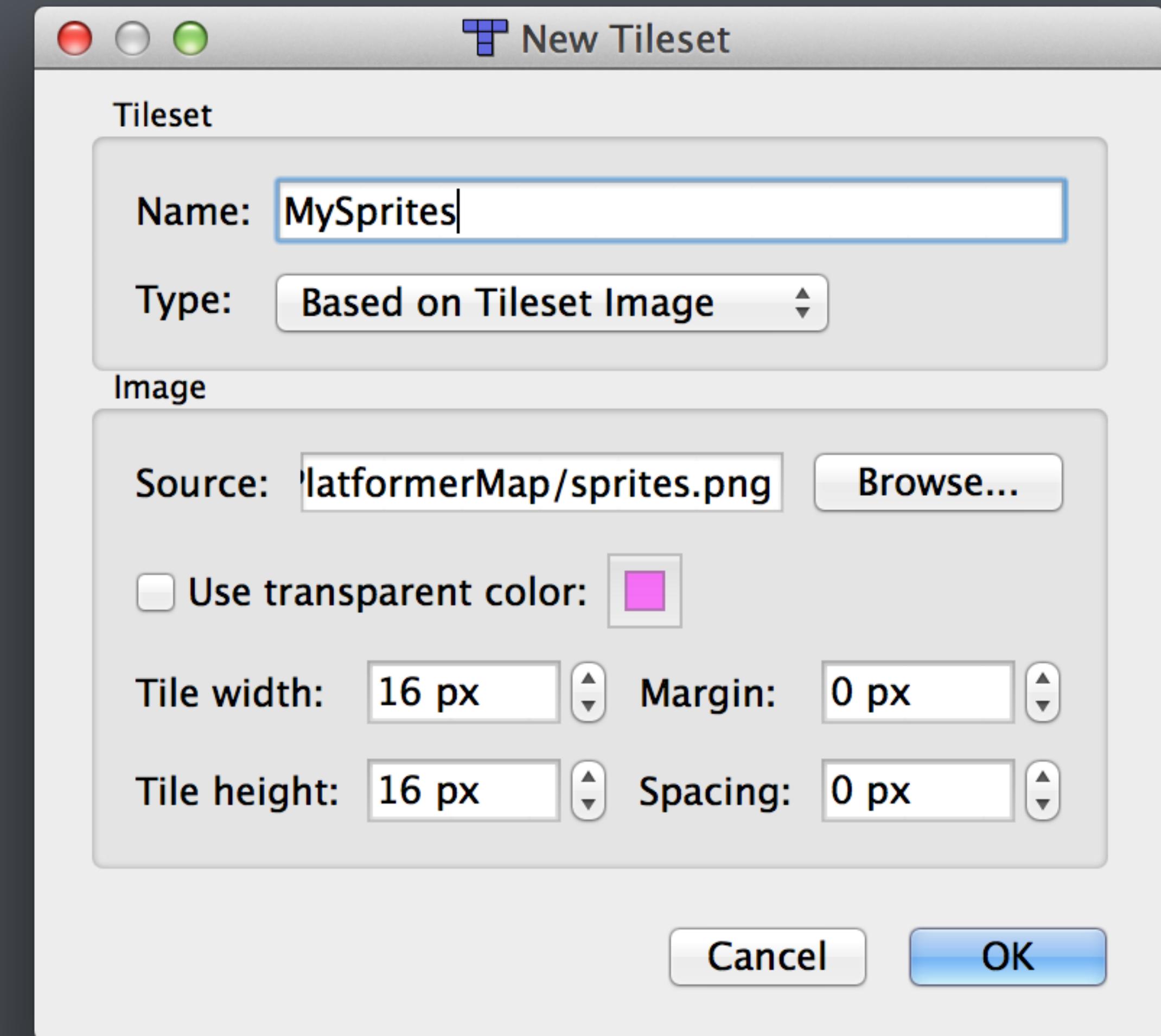
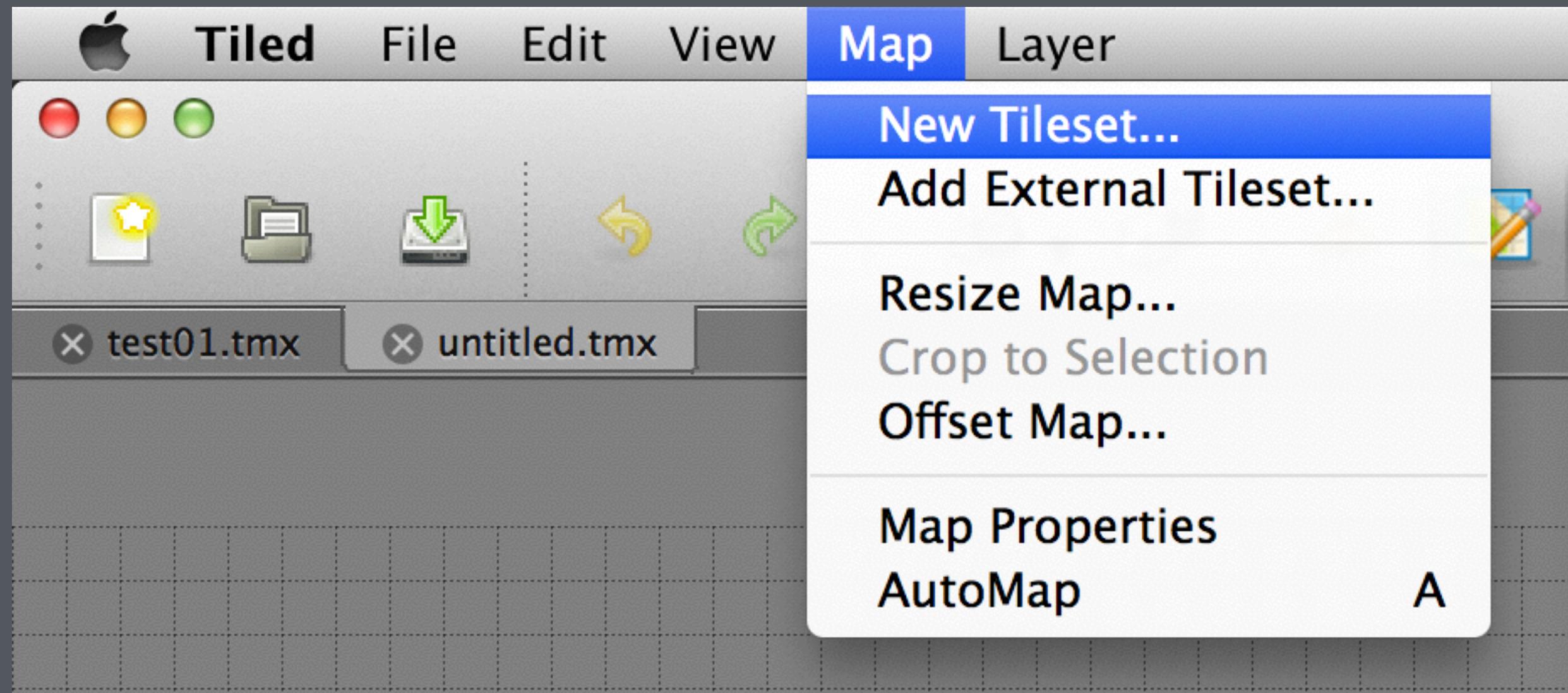
tiled



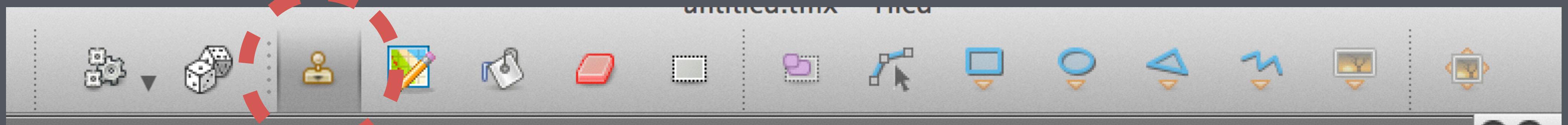
<http://www.mapeditor.org/>

# Building a level with Tiled.





# Drawing tiles



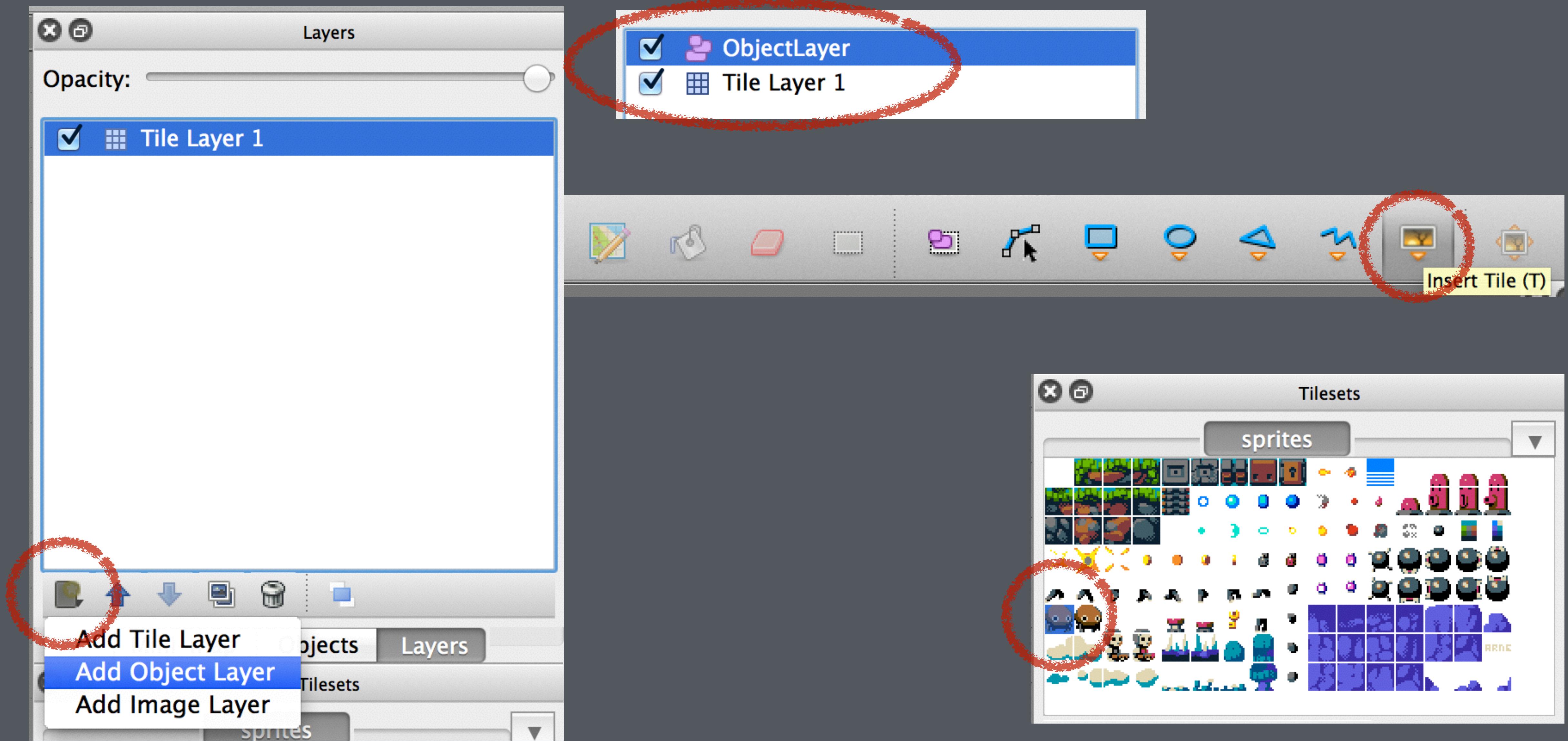
untitled.tmx

Tree

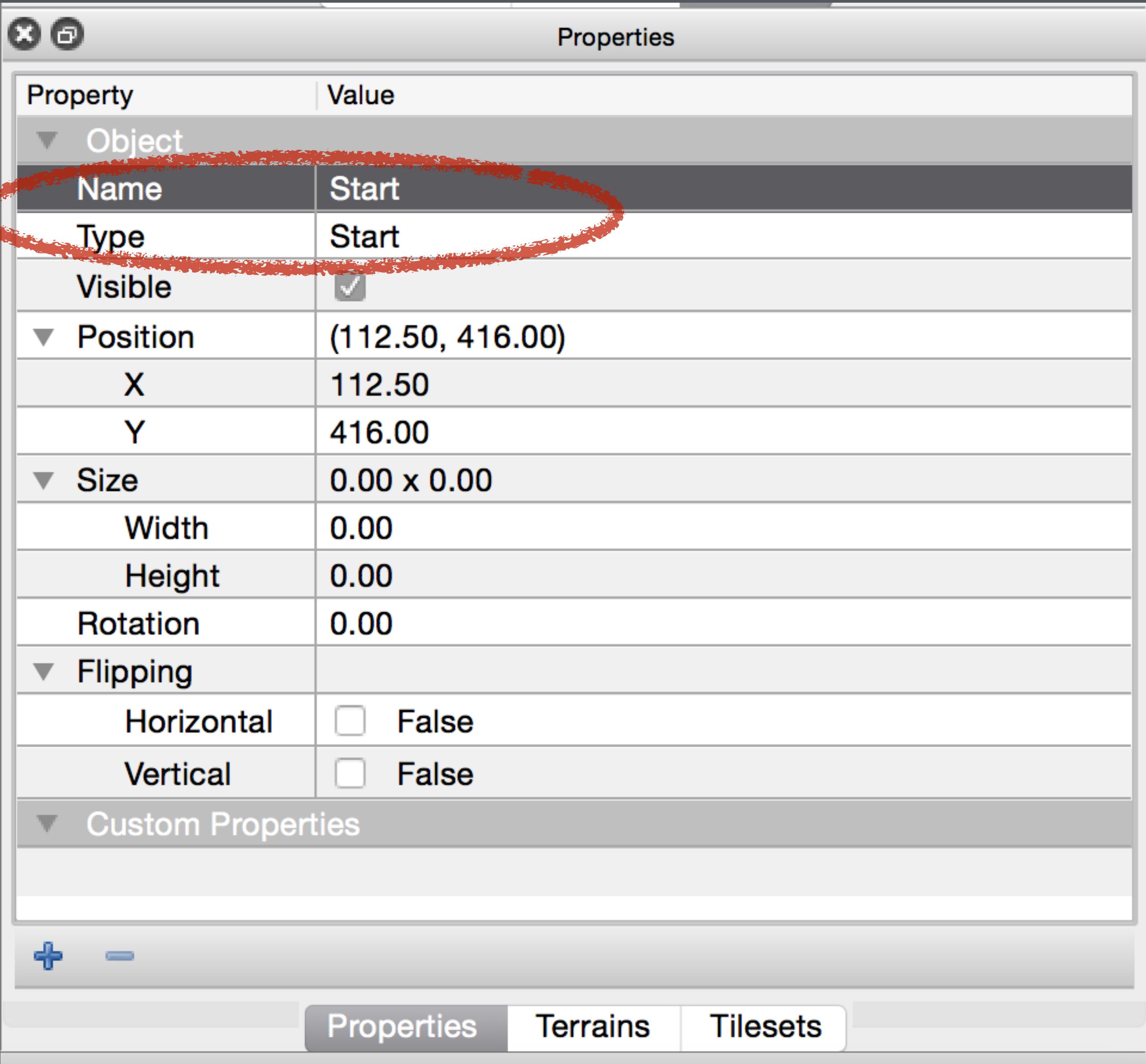
sprites

The screenshot shows the Tiled map editor's main workspace. On the left, there is a "sprites" palette containing a grid of various pixel art assets, including terrain tiles, objects, and characters. Below the palette are several tool icons and a zoom slider set to 100%. At the bottom of the palette are tabs for "Properties", "Terrains", and "Tilesets". The main workspace is a large, empty grid where a single row of green grass tiles has been placed along the bottom edge.

Placing entities.

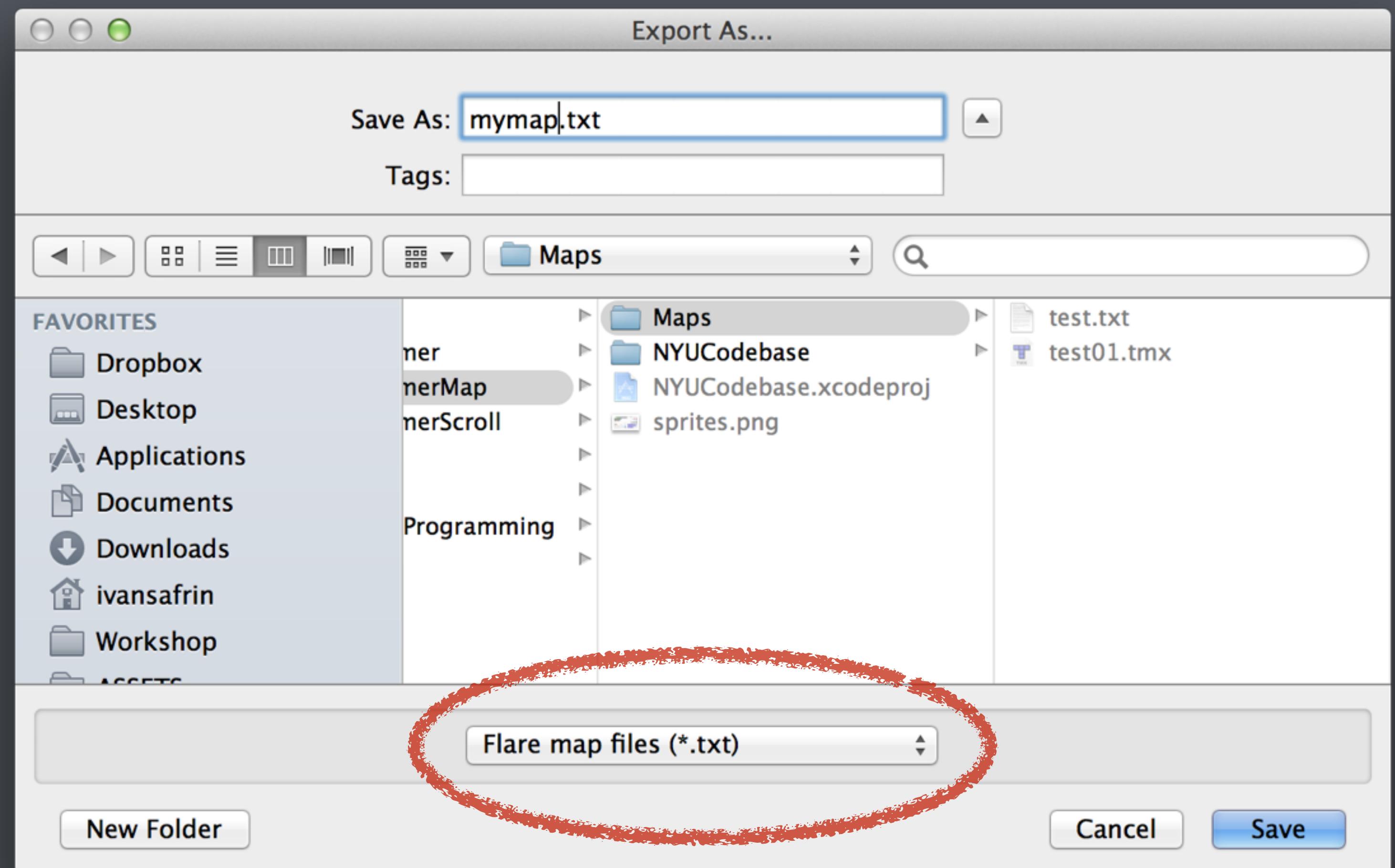
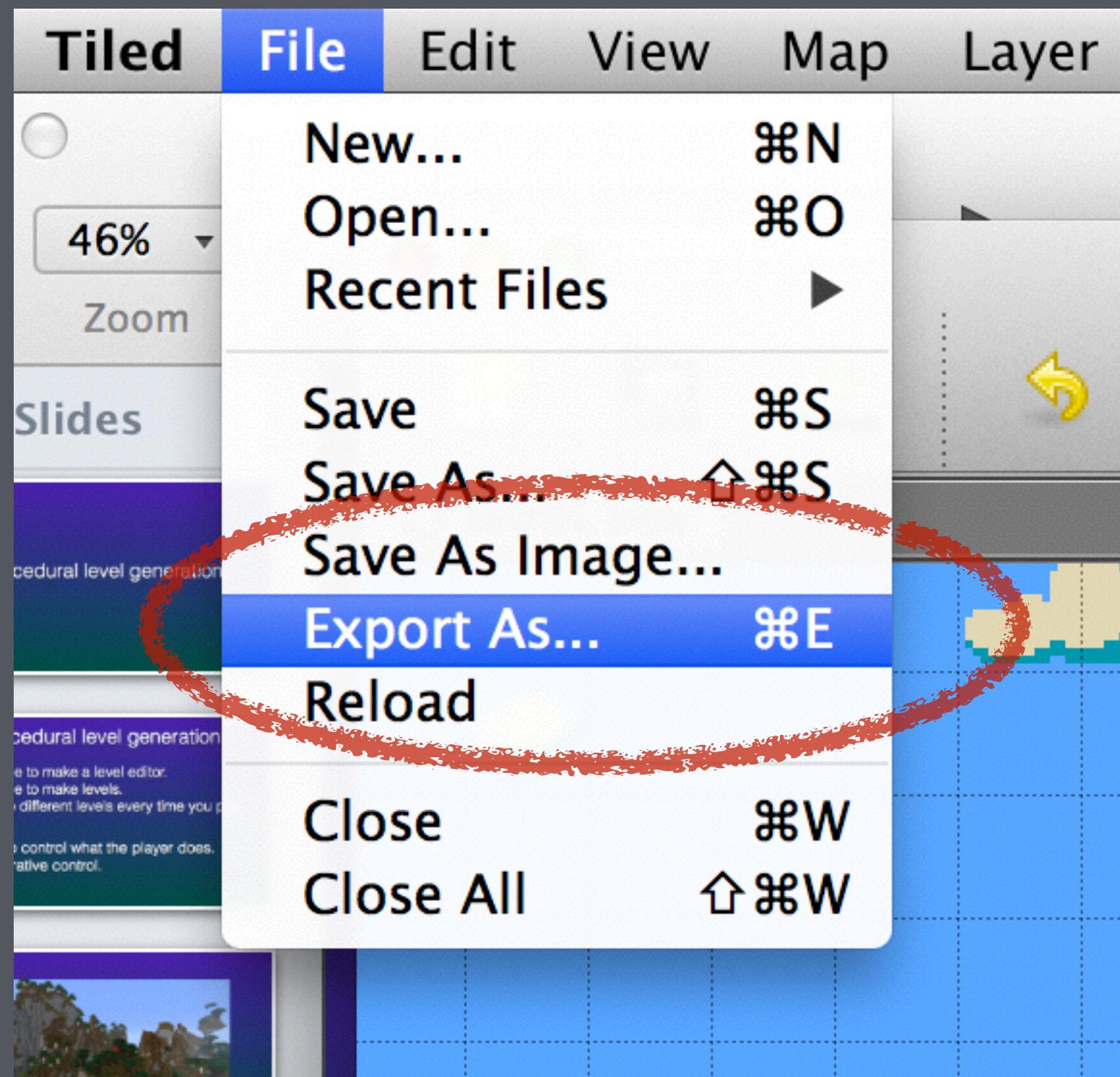


Make sure  
you fill in the  
**Name** and  
**Type** fields.



Loading Tiled levels in your game.

# Export as Flare map file.



This is what it looks like

# Parsing text from a file.

# What we'll need...

```
#include <fstream>
#include <string>
#include <iostream>
#include <sstream>

using namespace std;
```

# Reading a file **line by line**...

```
ifstream infile("file.txt");
string line;
while (getline(infile, line)) {
    // handle line
}
```

```
ifstream infile(levelFile);
string line;
while (getline(infile, line)) {

    if(line == "[header]") {
        if(!readHeader(infile)) {
            return;
        }
    } else if(line == "[layer]") {
        readLayerData(infile);
    } else if(line == "[ObjectsLayer]") {
        readEntityData(infile);
    }
}
```

```
[header]
width=128
height=32
tilewidth=16
tileheight=16

[layer]
type=Tiles
data=
17,0,0,0,0,0,0

[ObjectsLayer]
# Start
type=Start
location=112.5,416,0,0

[ObjectsLayer]
# Enemy1
type=Enemy
location=423.5,401,0,0
```

Keep in mind that **ObjectsLayer** is the name of our object layer in the editor, so you should check for whatever you named your object layer.

Reading the header.

# Reading the key=value pairs.

```
istringstream sStream(line);
string key,value;
getline(sStream, key, '=');
getline(sStream, value);
```

# Converting string to integer.

```
int intValue = atoi(value.c_str());
```

```

bool readHeader(std::ifstream &stream) {
    string line;
    mapWidth = -1;
    mapHeight = -1;
    while(getline(stream, line)) {
        if(line == "") { break; }

        istringstream sStream(line);
        string key,value;
        getline(sStream, key, '=');
        getline(sStream, value);

        if(key == "width") {
            mapWidth = atoi(value.c_str());
        } else if(key == "height"){
            mapHeight = atoi(value.c_str());
        }
    }

    if(mapWidth == -1 || mapHeight == -1) {
        return false;
    } else { // allocate our map data
        levelData = new unsigned char*[mapHeight];
        for(int i = 0; i < mapHeight; ++i) {
            levelData[i] = new unsigned char[mapWidth];
        }
    }
    return true;
}

```

[header]  
width=128  
height=32  
tilewidth=16  
tileheight=16

[layer]  
type=Tiles  
data=  
17,0,0,0,0,0,0

[ObjectsLayer]  
# Start  
type=Start  
location=112.5,416,0,0

[ObjectsLayer]  
# Enemy1  
type=Enemy  
location=423.5,401,0,0

# Reading the tile data.

```

bool readLayerData(std::ifstream &stream) {
    string line;
    while(getline(stream, line)) {
        if(line == "") { break; }
        istringstream sStream(line);
        string key,value;
        getline(sStream, key, '=');
        getline(sStream, value);
        if(key == "data") {
            for(int y=0; y < mapHeight; y++) {
                getline(stream, line);
                istringstream lineStream(line);
                string tile;

                for(int x=0; x < mapWidth; x++) {
                    getline(lineStream, tile, ',');
                    unsigned char val = (unsigned char)atoi(tile.c_str());
                    if(val > 0) {
                        // be careful, the tiles in this format are indexed from 1 not 0
                        levelData[y][x] = val-1;
                    } else {
                        levelData[y][x] = 0;
                    }
                }
            }
        }
    }
    return true;
}

```

[header]  
width=128  
height=32  
tilewidth=16  
tileheight=16

[layer]  
type=Tiles  
data=  
17,0,0,0,0,0,0  
0,0,0,0,0,0

[ObjectsLayer]  
# Start  
type=Start  
location=112.5,416,0,0

[ObjectsLayer]  
# Enemy1  
type=Enemy  
location=423.5,401,0,0

Reading the entity data.

```
bool readEntityData(std::ifstream &stream) {  
  
    string line;  
    string type;  
  
    while(getline(stream, line)) {  
        if(line == "") { break; }  
  
        istringstream sStream(line);  
        string key,value;  
        getline(sStream, key, '=');  
        getline(sStream, value);  
  
        if(key == "type") {  
            type = value;  
        } else if(key == "location") {  
  
            istringstream lineStream(value);  
            string xPosition, yPosition;  
            getline(lineStream, xPosition, ',' );  
            getline(lineStream, yPosition, ',' );  
  
            float placeX = atoi(xPosition.c_str())/(16*TILE_SIZE);  
            float placeY = atoi(yPosition.c_str())/(16*-TILE_SIZE);  
  
            placeEntity(type, placeX, placeY);  
        }  
    }  
    return true;  
}
```

```
[header]  
width=128  
height=32  
tilewidth=16  
tileheight=16  
  
[layer]  
type=Tiles  
data=  
17,0,0,0,0,0,0  
0,0,0,0,0,0  
  
[ObjectsLayer]  
# Start  
type=Start  
location=112.5,416,0,0  
  
[ObjectsLayer]  
# Enemy1  
type=Enemy  
location=423.5,401,0,0
```

# Assignment.

- Make a simple scrolling **platformer** game demo.
- It must use a **tilemap** or **static/dynamic** entities.
- It must **scroll**.
- It must be either **procedurally generated** or  
**load levels from a file**.
- You have **two weeks**.