ProcEngine: An Open Source Procedural Map Generation Engine

Ahmed Khalifa
PhD Student at NYU
Indiegame Developer and Designer

Every time I start thinking about designing a roguelike or a game that use procedural generation for maps, I start googling to see what are the different techniques generation techniques. After selecting the best one, I start writing a code for it from scratch or copying it. This process is tiring and cumbersome especially during prototyping phase. In prototyping, I need just to test the idea as quickly as possible. The main problem when one of these ideas depend on procedural generated maps. After creating couple of roguelike prototypes, I couldn't take it anymore. I decided to write my own library that I can use it in prototyping. I called this library *ProcEngine*.

ProcEngine is an open source procedural map generation engine that allow the user to select from bunch of different generating algorithms and tune them. ProcEngine is inspired by Nicky Case (*Simulating the world (in Emoji)*) and Kate Compton (*tracy.js*). The current version of ProcEngine (v1.1.0) supports the following features:

- Different techniques to divide the map into rooms. Only two techniques are implemented: equal division and tree division. Equal division divides the map into a grid then selects room from this grid, while tree division divide the whole map along the longest dimension till reach the required number of rooms.
- Define different tiles and define their maximum count.
- Define different neighborhoods in form of 2D matrix of 1's and 0's. 1's are the places to check while 0's otherwise.
- Define any number of cellular automata that the system will apply after each other.
- Specify where to apply the cellular automata. The system support two positions either applied on the whole map regarding of the room structures (useful for smoothing the whole map or generating game objects) or on the generated rooms (useful for designing dungeons).
- Connect/delete the generated islands after applying each cellular automata.
- Cellular automata rules can have multiple conditions and replacing values.

The engine allows the users to modify the underling generator through the following functions:

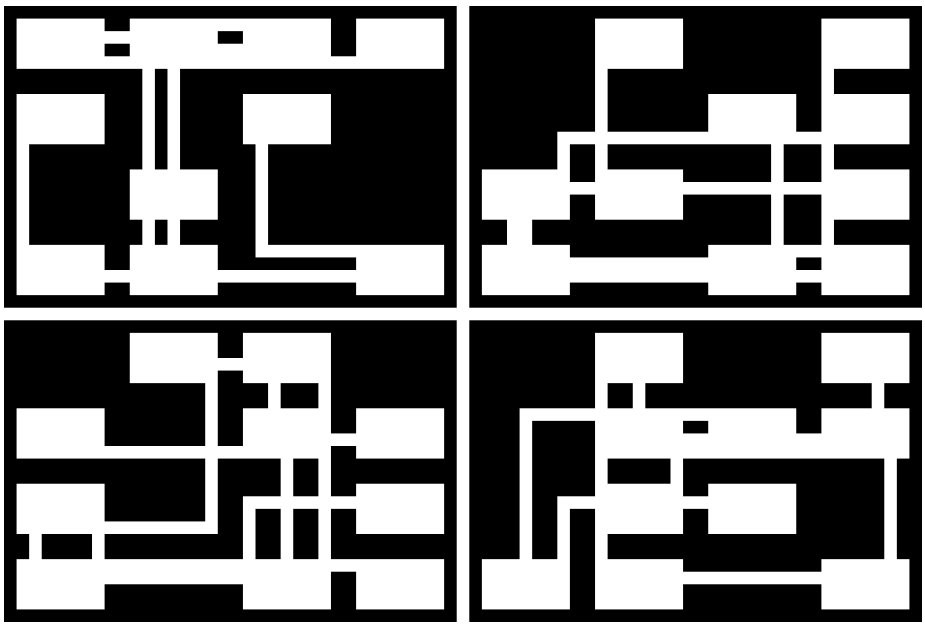
- **procengine.initialize(data):** to initialize the system with your rules.
- **procengine.generateMap():** to generate a level (you have to call initialize beforehand).
- **procengine.toString():** to get a string that shows the current data saved in the system.
- **procengine.testing.isDebug:** set to true to allow console printing after each step in the system.

In order to use the system you need to call *procengine.initialize(data)* function first then you can call *procengine.generateMap()* for as many as you want. Each time you get a new generated map. For more details about how to use the engine refer to github (https://github.com/amidos2006/procengine).

Here is a bunch of examples that shows the capabilities of the system. The first example is a very simple generator. The generator should generate a map of 36x24 with 10 rooms using equal division technique.

```
var data={
  "mapData":["36x24", "solid:empty"],
  "roomData":["equal:4x4:10", "empty:1"],
  "names":["empty:-1", "solid:-1"],
  "neighbourhoods":{"plus": "010,101,010"},
  "generationRules":[
    {"genData":["0", "map:-1", "connect:plus:1"], "rules":[]}
  ]
};
```

Here are four different generated maps from the previous data, where white is empty and black is solid:



The second example is more complicated where it generates a map of 36x24 with 5 rooms using tree division technique. Also, it uses three cellular automatas in the following order:

1. Generate the solid structure of the rooms.
2. Connect the rooms together all over the whole map.
3. Adds objects (1 player, 10 gold pieces (at most), and 15 enemies (at most)).

```
var data={
  "mapData":["36x24","solid:empty"],
  "roomData":["tree:8x8:5","empty:2|solid:1"],
  "names":["empty:-1","solid:-1","player:1","gold:10","enemy:15"],
  "neighbourhoods":{
    "plus": "010,101,010",
    "all": "111,111,111"
  },
  "generationRules":[
    {"genData":["3","room:-1","connect:plus:1"],
         "rules":["empty,all,or,solid>5","solid:4|empty:1"]},
    {"genData":["1","map:-1","connect:plus:1"],
     "rules":[]},
    {"genData":["1","room:-1","connect:plus:1"],
     "rules":["empty,plus,or,empty>2","player:1|empty:8|gold:2|enemy:2"]}
  ]
};
```

Here are four different maps generated from the previous data, where black is solid, white is empty, blue is player, red is enemy, yellow is gold: