Algorithm for procedureral 2D map with connected paths

Problem to solve: Generate a random 2D dungeon map for a tile-based game where all rooms are connected.

I am looking for better solutions than what I currently have.

My current solution is that I run two algorithms. The first generates the dungeon with its rooms. The second make sure that all rooms are connected. I am curious what other soltions may exist. Faster and / or easier etc. Speed is not really a concern, but if speed can be gained at no real cost, well, that is a good thing. More important is that I, and others that read, may learn different ways to approach and solve the problem.

Below are my current implementation. Rooms currently have no exits or exits in any 2, 3 or 4 directions.

Generating the dungeon rooms

Setup: Set the current room to the top left room.

1. Get a valid room type for the room (where valid room type is a type with no exits out of the dungeon and that have exits that matches the exits of the room above and the room to the left. Only need to check above and to the left due to step 2 below).
2. Put down the room and advance the x-coordinate one step. If the x-coordinate exceeds the dungeon width, set the x-coordinate to 0 and advance the y-coordinate one step. If the y-coordinate exceeds the dungeon height, we are done.

3. Repeat from #1.

I then check to see if all rooms are connected If they are not all connected I run a second algorithm that, in a non-sexy but definately good enough way in terms of dungeon layout, goes through the rooms and change them so that all end up being connected.

Checking to see if all rooms are connected

Setup: Create a 2D map of integers representing paths and initialize the entries to a "unprocessed" (not yet traversed) value, -1. Set a start path index integer that keeps track of the current path to 1. Set the current room to the top left room by adding it to a stack of rooms to check.

1. If the stack contains rooms to check, pop it set the path index of the room to the current path index. If the stack does not contain any rooms, increase the path index and try to get a room by advancing column by column, row by row, until we get a room that has not been processed yet. If no room can be found, we are done.
2. Check to see if the room has an exit to the left. If it has add the left room to the stack if it is not already on there.
3. Repeat step 2 for down, right and top directions (since we are using a stack that means the rooms are traversed in clockwise order, starting with the top direction).
4. Repeat from step 1.
5. If the path indices count is greater than one, there are disconnected rooms.

If there are disconnected rooms I then group the rooms by their path index, get the index of the biggest path and connect all other rooms to those rooms. This is a work in progress, but my (current, "brutish") plan is to go through each room in a room group (except the first) check to see if there is a horizontal or vertical path to the biggeset room group, and if so, create a horizontal / vertical path there by injecting / updating the rooms inbetween. Rinse and repeat. Ugly, yes, but it is something that will not be noticable in terms of visual pattern so it works in that sense.

2d procedural-generation tilemap random

asked Aug 17 '14 at 22:10

user1323245
227 2 9

1 Have you checked out "Dungeon Generation" on PCG wiki? Does it answer your questions? – congusbongus Aug 18 '14 at 0:37

@congusbongus Useful reading for sure. That donjon generator linked on that page is awesome. Thanks. – user1323245 Aug 18 '14 at 10:37

2 Answers

One of the best, and most used, algorithms I've seen out there is generating dungeons using

Binary Space Partitioning.

The best general explanation I've read is the one found in The Chronicles of Doryen (attached at the end for backup purposes) because explains the procedure without getting into the code, thus leaving the implementation to the reader.

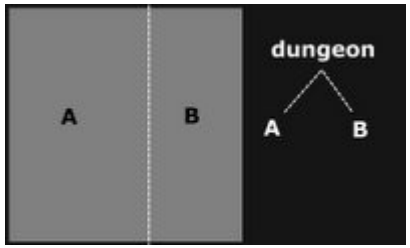
Two other tutorials on the same subject, with code, can be found at

- How to Use BSP Trees to Generate Game Maps
- Dungeon generation using BSP trees

Building the BSP tree

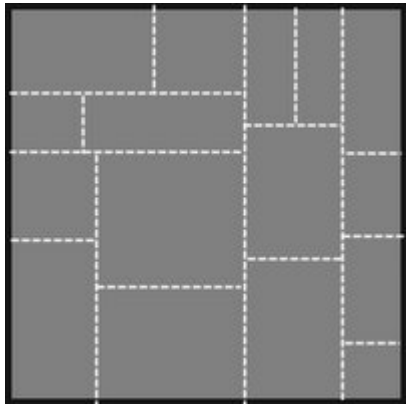
We start with a rectangular dungeon filled with wall cells. We are going to split this dungeon recursively until each sub-dungeon has approximately the size of a room. The dungeon splitting uses this operation :

- Choose a random direction : horizontal or vertical splitting
- Choose a random position (x for vertical, y for horizontal)
- Split the dungeon into two sub-dungeons



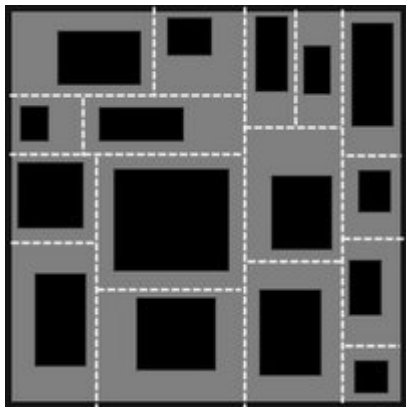
Now we have two sub-dungeons A and B. We can apply the same operation to both of them.

When choosing the splitting position, we have to take care not to be too close to the dungeon border. We must be able to place a room inside each generated sub-dungeon. We repeat until the lowest sub-dungeons have approximately the size of the rooms we want to generate.

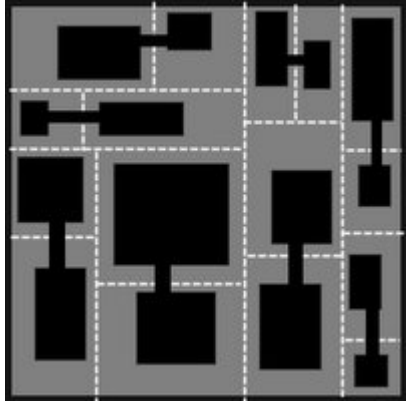


Building the dungeon

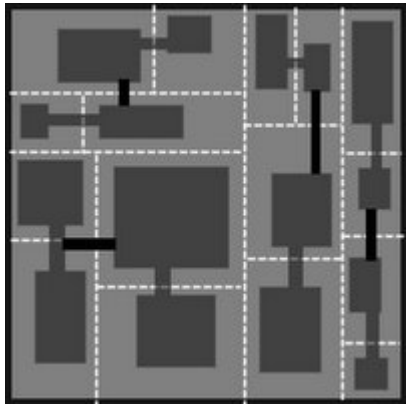
Now we create a room with random size in each leaf of the tree. Of course, the room must be contained inside the corresponding sub-dungeon. Thanks to the BSP tree, we can't have two overlapping rooms.



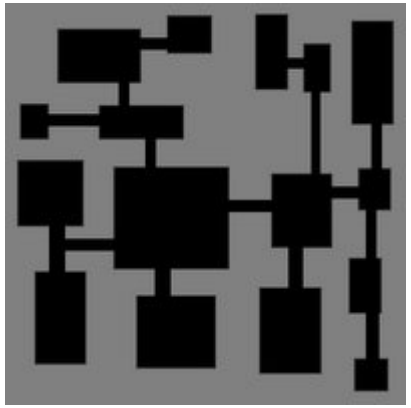
To build corridors, we loop through all the leafs of the tree, connecting each leaf to its sister. If the two rooms have face-to-face walls, we can use a straight corridor. Else we have to use a Z shaped corridor.



Now we get up one level in the tree and repeat the process for the father sub-regions. Now, we can connect two sub-regions with a link either between two rooms, or a corridor and a room or two corridors.



We repeat the process until we have connected the first two sub-dungeons A and B



answered Aug 18 '14 at 0:56

pctroll
1,893 1 14 24

Excellent answer! – Matthew Pigram Aug 18 '14 at 4:54

It may be worth nothing that this technique will never create loops, however I am not sure if there is a way around that without adding more random corridors. Still very good answer, +1 – Vality Aug 18 '14 at 8:11

This is a promising start. Just need to figure out a way to add some loops to it, but I rather work on that problem than continuing down the path I am currently at. Thanks. – user1323245 Aug 18 '14 at 10:36

2 Nice ! I was interested by the id so i made a small attempt. You need to be careful when using random otherwise too-strange results will follow. And i wonder if the corridors shouldn't be handled right during the recursive split, because i don't see easy way to build corridors out of the tree. Anyway for anyone interested fiddle is here : jsfiddle.net/gamealchemist/xt57zwb8 – GameAlchemist Aug 18 '14 at 13:16

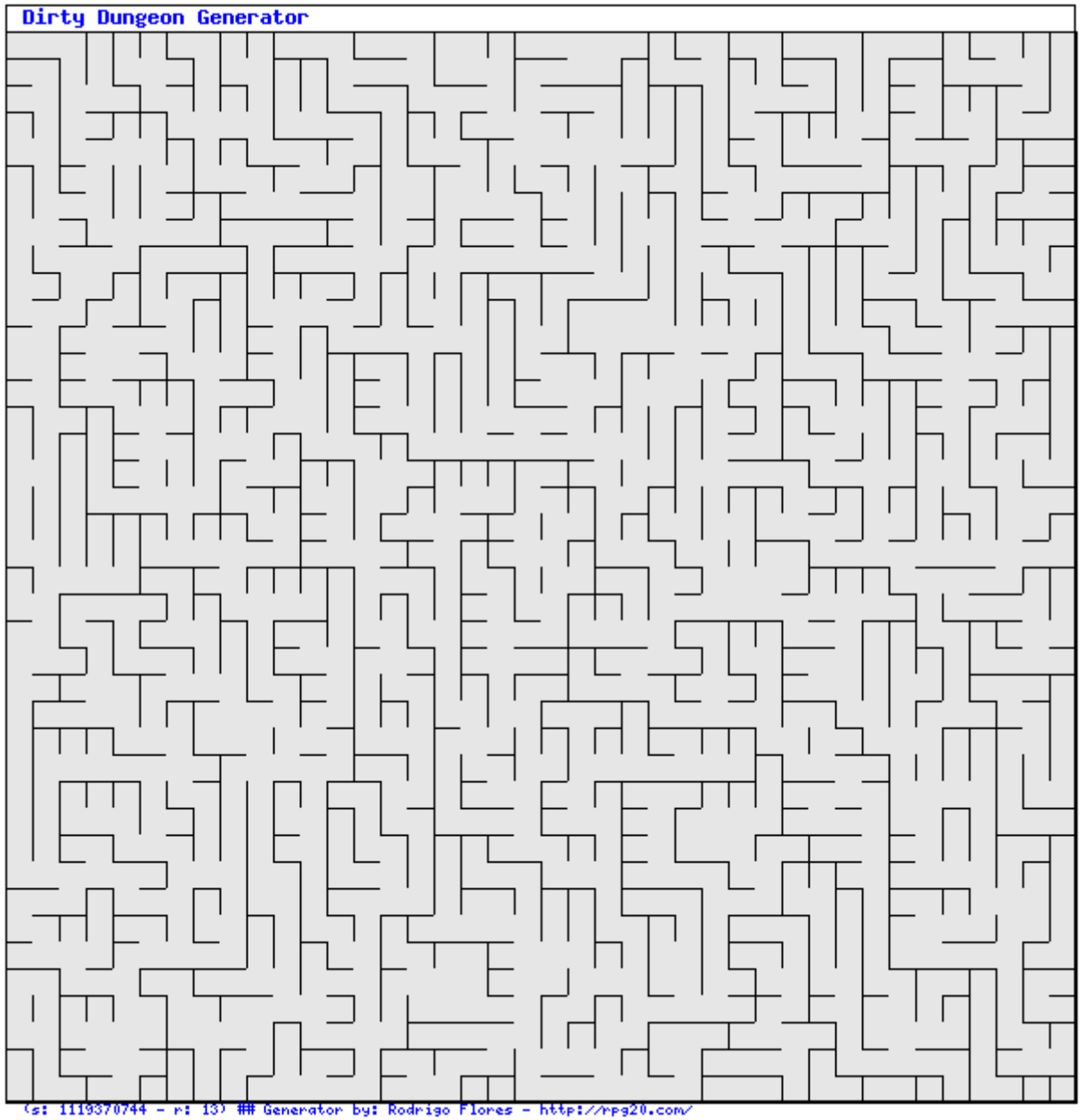
While I find this somewhat problematic in repeatable seeded proceduralization over large environments. It is probably one of the best methods I've ever seen for this kind of generation providing you are generating your entire level at once. I +1 this – That Homeless Guy Aug 18 '14 at 20:25

The **BSP method** is apparently the most popular method for generating dungeons, but it's not the only one.

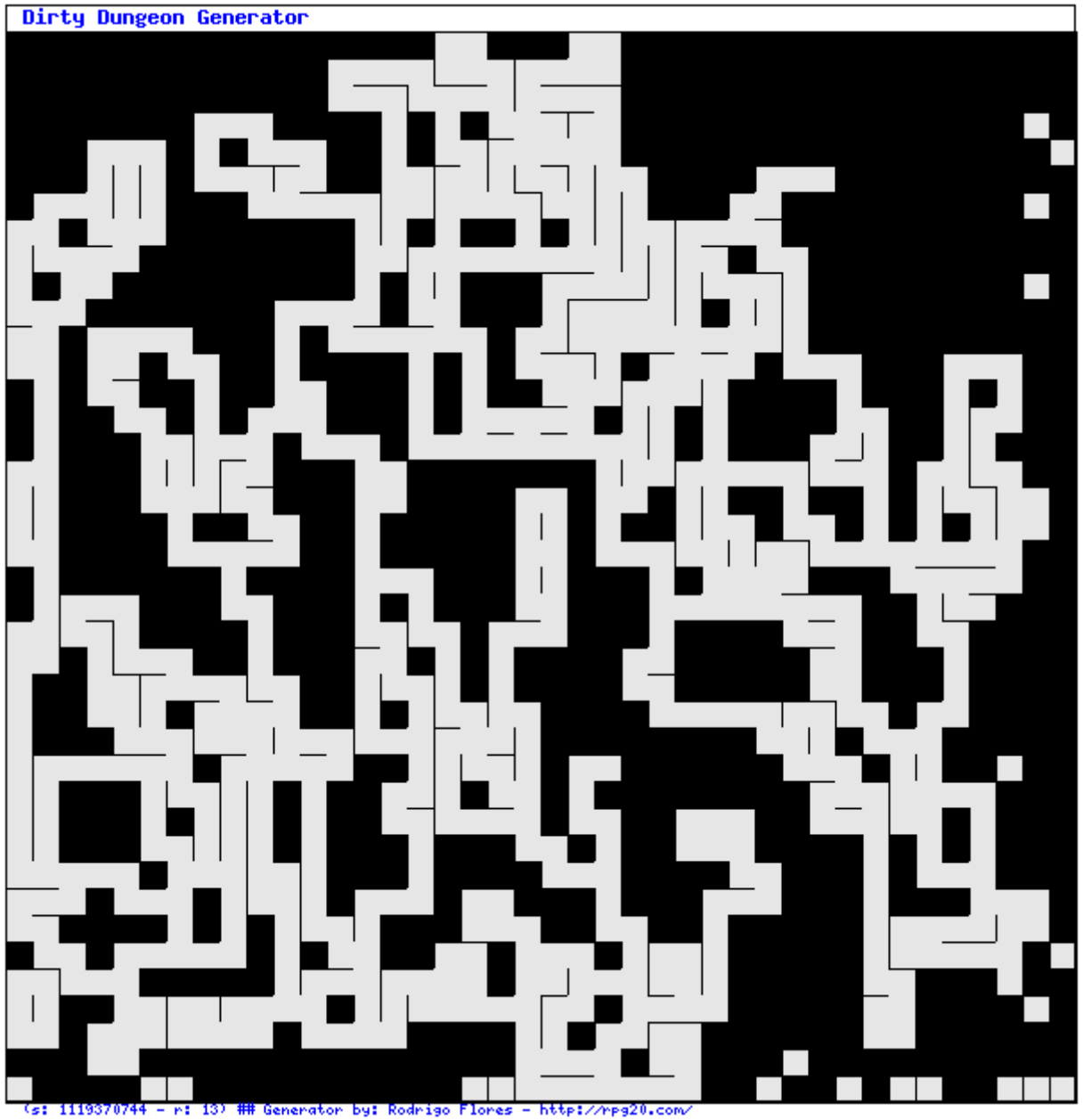
For completeness I'll explain the generator that worked for me. I have to admit that I don't recall where I read about this so I'll just say that it's not my invention (an old article by Jamis Buck sounds very familiar).

A maze with rooms

The basic idea is that a dungeon is a maze with rooms, sort of. So the first step for this algorithm, is to generate a maze:

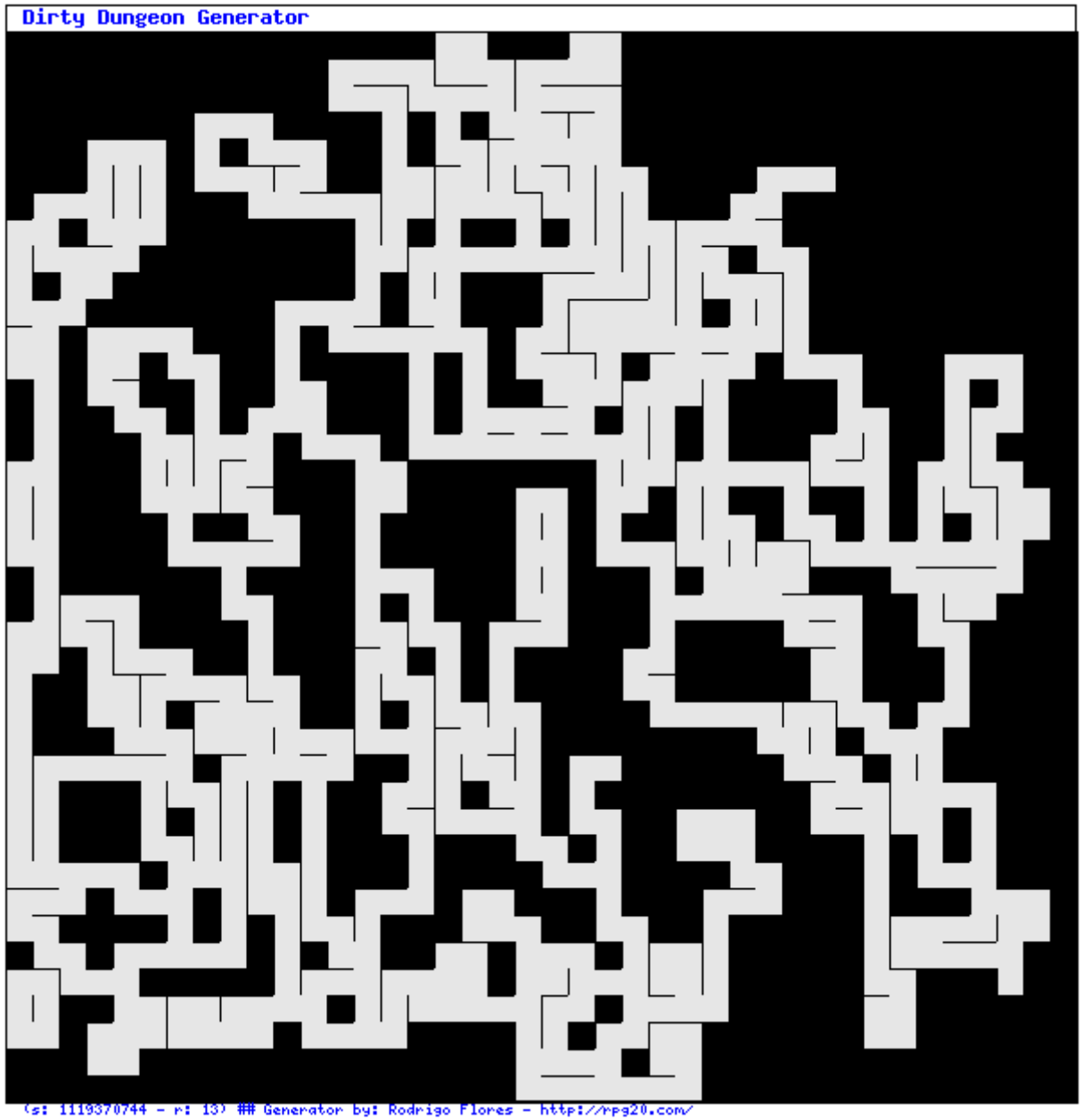


The next step is to make it sparse (remove dead ends):



Step number 3 is to add some loops (make it non-perfect) but I'll skip the image because it's barely noticeable (I didn't need a perfect maze so I took a few shortcuts on the maze generation algorithm, so it already had loops by this point).

Then, for step 4, we need to remove isolated cells:



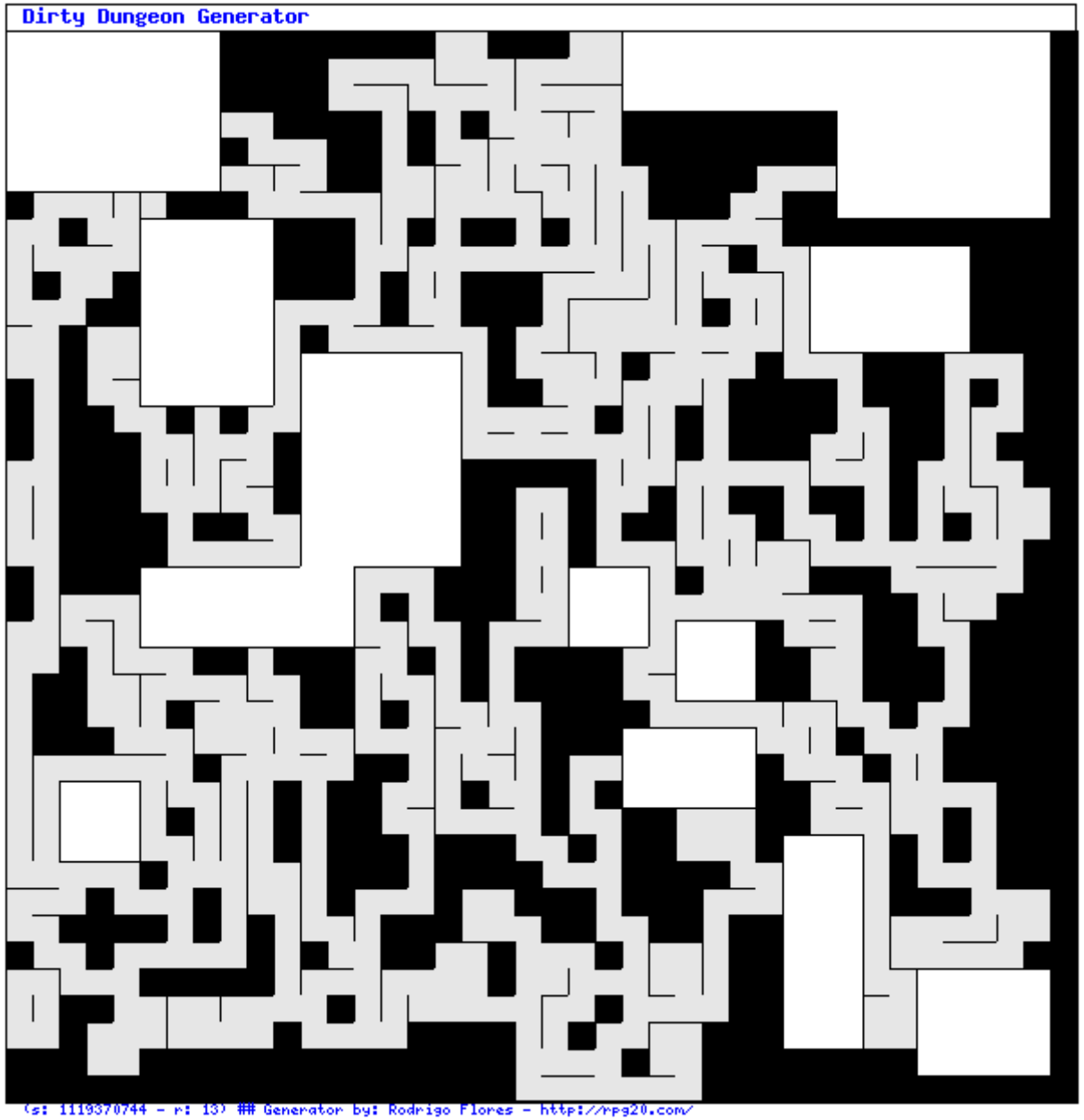
At this point we're done with the corridors and we're ready to add rooms. For that we do the following:

1. Generate a set of rooms (width and height)
2. For each room we iterate through all possible locations and decide the best location.

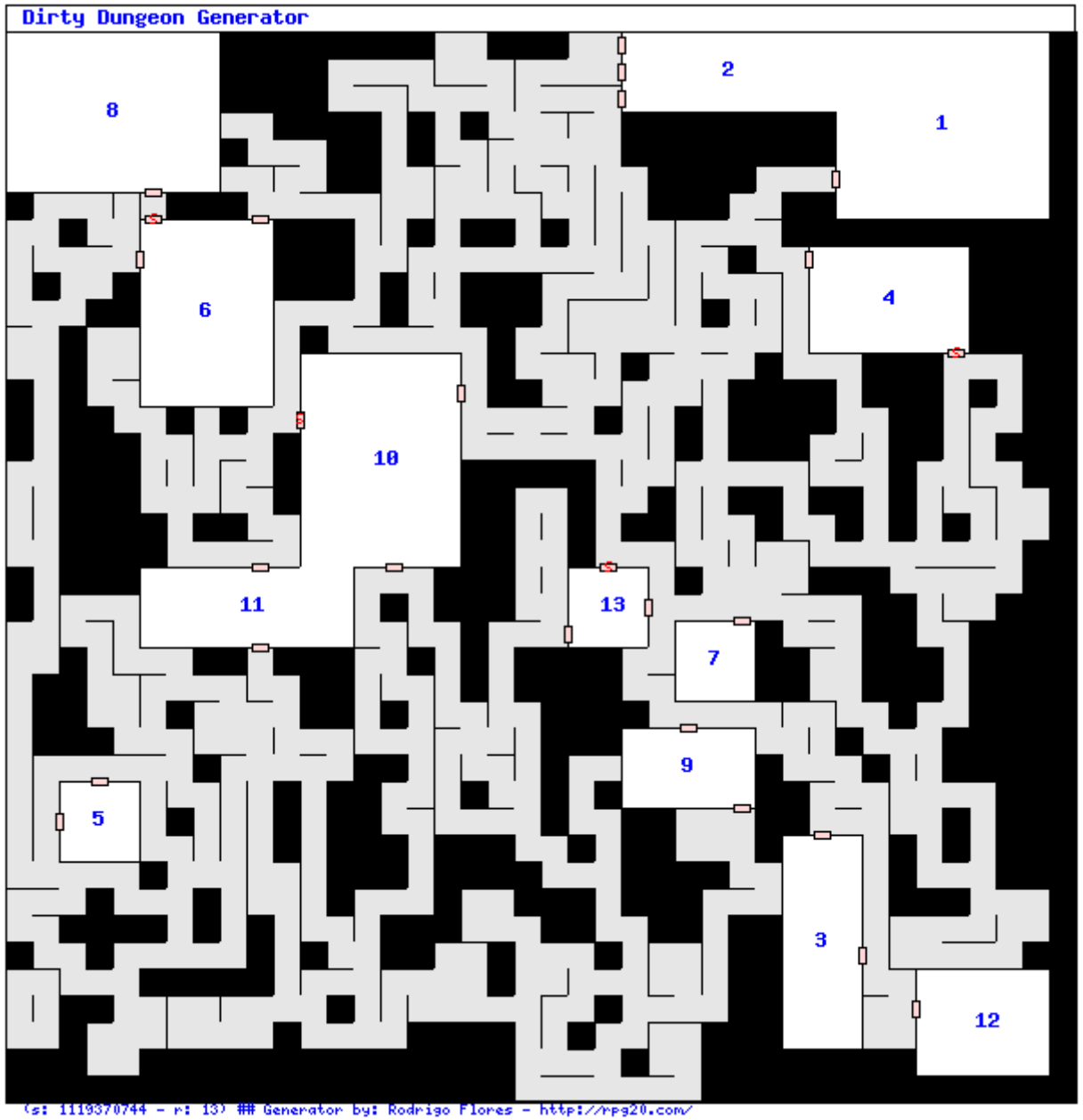
- The best location is calculated by adding a weight to conditions (such as adjacency to a corridor).

3. We place the rooms.

So far, our dungeon will look like this:



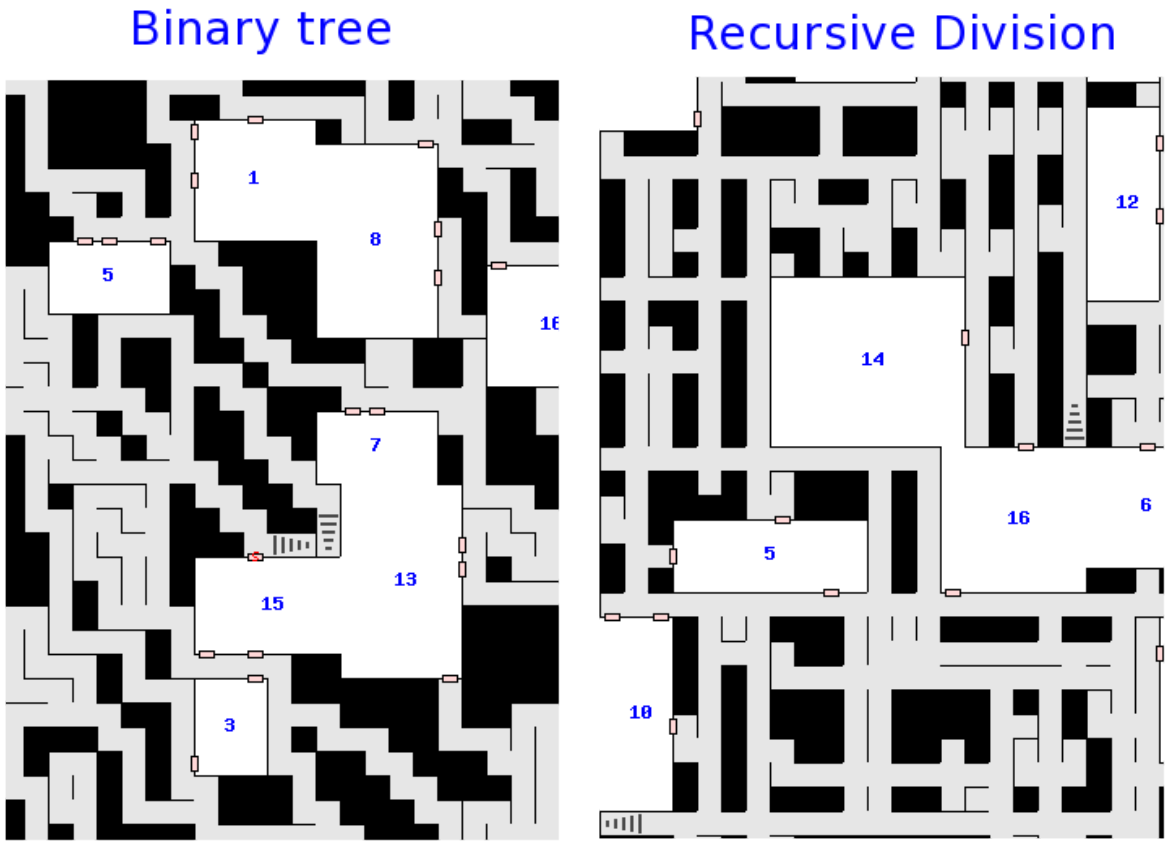
The final step is to add decorations.



Some final thoughts

- I used a stripped-down version of the Eller Algorithm.
- Different maze algorithms may result in different textures. You might prefer another algorithm. For example, the following image shows different *textures* resulting from "Binary

Tree" (diagonal bias) and a variation of "Recursive Division" (long corridors) algorithms:



edited Apr 13 at 12:18

Community ♦
1

answered Nov 20 '14 at 16:01

Roflo
153 8

2 Good stuff. I've ben looking for different ways to do it, since using different algorithms for different levels can make the game even more versatile. – user1323245 Nov 21 '14 at 2:16