

Your dungeon generator should be written with an eye toward extensibility. Remember that you will be adding functionality on top of this next week, and for the rest of the semester (see the roguelike roadmap on the assignments page). For this step, you generate a dungeon, draw it on the standard output, and exit. Here are the requirements:

- All code is in C.
- Dungeon measures 80 units in the x (horizontal) direction and 21 units in the y (vertical) direction. A standard terminal is 80×24 , and limiting the dungeon to 21 rows leaves three rows for text, things like gameplay messages and player status, which come later.
- Require at least 6 rooms per dungeon
- Each room measures at least 4 units in the x direction and at least 3 units in the y direction.
- Rooms need not be rectangular, but neither may they contact one another. There must be at least 1 cell of non-room between any two different rooms.
- The outermost cells of the dungeon are immutable, thus they must remain rock and cannot be part of any room or corridor.
- There should be at least one up and one down staircase. Staircases work like floor (for now) and should be placed in a location where floor would otherwise be. A character on the floor somewhere in the dungeon should be able to walk to the stairs.
- Room cells should be drawn with periods, corridor cells with hashes, rock with spaces, up staircases with less-than signs, and down staircases with greater-than signs.
- The dungeon should be fully connected, meaning that from any position on the floor, your adventurer should be able to walk to any other position on the floor without passing through rock.
- Corridors should not extend into rooms, e.g., no hashes should be rendered inside rooms.

Here is an informal description of a dungeon generator that I wrote to produce the figure; I've played with dungeon generation enough to know that much simpler methods can do the job, but this method does a nice job of balancing aesthetically-pleasing (to me) dungeons and having a small, straightforward implementation. There are certain tunable parameters, for example, determining the size of a new room, that I leave out of the description. You may use this algorithm, something you find online, or something of your own devising.

To implement my algorithm, you will need an array of rooms and an 80×21 matrix of cells representing the dungeon. I initialize the dungeon by setting an immutable flag on the outermost cells and assigning a hardness to the material in every cell. I then attempt to randomly place random rooms in the available space, checking that the room can be placed legally each time, until some termination criterion is reached. Example criteria: the dungeon is at least 7% open (not rock); there were 2000 failed placement attempts in a row; a `create_new_room()` predicate failed; etc.

After placing rooms, move through the room array of n rooms, connecting room 1 with room 2, then room 3 with rooms 1-2, ... until you've connected room n with rooms 1-($n - 1$). Okay, so how do we make that connection? Find the closest room in the already connected set using Euclidean distance to its centroid then carve a path to it by changing rock to open space; this can always be done with zero or one change of direction. If you get that working, then add some random changes of direction in there to make it look a little more exciting.

The hardness field isn't strictly required now, but it will be required beginning in 1.02, so you should at least be thinking about how you'll add it. From a design perspective, I think that now is the time to put it in, even if you don't use it.