COM S 327, Spring 2018
Programming Project 1.03
Path Finding

So far, we've got this lovely dungeon. And we can. . . save and restore it. And, you know. . . look at it.

And that's about it. Nice for mom's fridge, but otherwise kind of boring.

Once you have monsters (next week), they (at least the smart ones) will need to find a path to the player through the dungeon. To find that path, you'll need to implement a path-finding algorithm. We're going to have some monsters that can tunnel through walls and others that can only move through open space, so we'll actually need two slightly different pathfinding algorithms. In both cases we'll use Dijkstra's Algorithm, treating each cell in the dungeon as a node in a graph with (up to) 8-way connectivity. For the non-tunneling monsters, we'll give a weight of 1 for floor and ignore wall cells (i.e., don't try to find paths through walls; this actually degenerates to BFS, and you're welcome to use that, but we will not require you to implement two different–if very similar–algorithms for this assignment). For the tunnelers, we'll have to use weights based on the hardness; cells with a hardness of 0 have a weight of 1, and cells with hardnesses in the ranges $[1, 254]$ have weights of hardness / 85. A hardness of 255 has infinite weight. We don't have to assign a value to this. Instead, we simply do not put it in the queue.

A naïve implementation will call pathfinding for every monster in the dungeon, but in practice, every monster is trying to get to the same place, so rather than calculating paths from the monsters to the player character (PC), we can instead calculate the distance from the PC to every point in the dungeon, and this only needs to be updated when the PC moves or the dungeon changes. Each monster will choose to move to the neighboring cell with the lowest distance to PC. This is gradient descent; the monsters move "downhill". Unless the monster is already collocated with the PC, there is always at least one cell with a shorter distance than its current cell. In the case of multiple downhill cells having the same distance, the monster may choose any one of them.

Dijkstra's Algorithm is described here: http://en.wikipedia.org/wiki/Dijkstra%27s_algorithm Scroll down to find the pseudocode under "Using a priority queue". Obviously, you'll need a priority queue, one with a decrease priority operation. You may use the Fibonacci queue that I provided with my solution to 1.01, or you may implement (or use a properly-attributed third party implementation of) any other priority queue you like.

My corridor building code uses Dijkstra's algorithm, so you may start with that (it won't require much modification) or start from scratch.

To test your code, select a random floor point in the dungeon for your PC, which you will render with an '`@`'. Render your dungeon with the PC. Then render your non-tunneling monster distance map, still marking the PC with '`@`' and marking distances using the last digit of the distance from the PC as calculated by your pathfinding algorithm (see image). Repeat for the tunneling monster distance map. Note that your distance maps will be integers from zero to some max value. We are only *displaying them* using the values above, not storing them that way. It is recommended that you add a switch to place your PC at a specified (y, x) in the dungeon; this will allow you to easily compare your output with mine on the test dungeons that were released for 1.02.

Your submission, when run, should generate a dungeon, calculate all distance maps, render all three views of the dungeon, and exit.

All code is to be written in C.

```
                  6555
                  6544 10987655
                  6543 10987654
             321098765432109876543
             3210                32
665432109876543211                211
55543        3222                   0
44444        3333                   9
33333        4444                   8
32222        5555                   7                  766666
32111                               654321             765555
32100                  9876543210987654321098765432109876544
32109                  9876         654321  8765       765433
3210987654321098765432109877                8766            2
                      09888                 8777            21112
                      09888                                 21@12
                      09877                                 21112
                      09876                           987654322222
                      0987654321098765432109876543210
```

Here is an example distance map for non-tunneling monsters. The PC is near the lower right corner. Only the last digit of the distance is shown. You can get actual distances by counting the zeros along a path (sort of like reading elevations on a topographical map). Keep in mind that if you follow a non-optimal path in a circuit, you may have to count backwards! Pay attention to the gradients. Note that I don't have a solution for our dungeons, yet, so I've produced this map manually. It's highly possible I've made an error in here somewhere, but I believe it's correct.