COM S 327, Fall 2023
Programming Project 1.03
Path Finding

Next week we'll be introducing adversary trainers to our game. We'll have trainers who just stand by the road and don't move. We'll have trainers who walk back and forth within a region, and we'll have trainers that chase the PC[1]. We'll have two different kinds of trainers that pathfind: *hikers*, who can go almost anywhere except water, and *rivals*, who can't walk over rocks, trees, or water. If you've introduced other types of terrain in your maps, treat it logically.

This week we'll calculate movement paths for hikers and rivals. Both of these trainer classes can see the PC wherever it is and will take the shortest path to the PC so they can engage in a Pokémon battle and be the very best like no one ever was. Shortest distance is defined in terms of travel time. We'll define travel times through terrain in *game turn*s—where a *game turn* is the smallest length of time that can pass in the game—as follows:

Trainer↓\Terrain→	Bldr	Tree	Path	PMart	PCntr	TGras	SGras	Mtn	Forest	Water	Gate
PC	∞	∞	10	10	10	20	10	∞	∞	∞	10
Hiker	∞	∞	10	50	50	15	10	15	15	∞	∞
Rival	∞	∞	10	50	50	20	10	∞	∞	∞	∞
Swimmer	∞	∞	∞	∞	∞	∞	∞	∞	∞	7	∞
Other	∞	∞	10	50	50	20	10	∞	∞	∞	∞

This chart includes the movement costs for the PC all adversarial trainer types, but only hiker and rival costs are apropos to this assignment. The other costs will be needed in the future.

In my maps, I have mountains and forests which are separate entities from boulders and trees. There was no requirement that you do this, and there is no requirement that you add it now; however, you will want to prevent NPCs from accessing the border of a map, and by making boulders impassible (∞), and marking my border with boulders, I implicitly prevent NPCs from entering the border[2]. As a result, I also get a random terrain element that stops all characters. I use trees the same way, and could use them in my map borders if I wanted to. By having separate mountain and forest terrain (using the same symbols but different internal representation, I have a terrain that those hardy hikers can cross (at a small movement penalty) but nobody else can, including the PC. If you have boulders but not mountains and you don't want to add mountains, probably the easiest change is to create a *border* terrain which is displayed using your boulder symbol and has infinite movement cost.

Given these movement costs, we're going to calculate shortest paths from all locations to the PC. This implies that the PC will have to appear in the map as well. Place your PC in a random location anywhere on a road and not in a gate. Use an @ to represent the PC when you draw it. It would probably be wise to add a placeholder *pc* datatype at this point and put the PC's position data (the only data the PC needs at this point) inside it.

Movement can occur in all 8 directions. Use Dijkstra's Algorithm (or A*) to calculate the distance to the PC from every location in the map for hikers and rivals.

[1] I think I've used this before without defining it. In case you're not hip to all the lingo that the fly gamer kids are using, the *PC* is the *player character*, while *NPC*s and *non-player character*s.

[2] If NPCs are permitted to enter the edge cells of a map, then special-case logic is required for all NPC moves to ensure they don't attempt to move outside the bounds of the map. Filling the boarders with impassible terrain eliminates the special case, making it easier to ensure that your NPCs remain in bounds.

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

Your program should display the hiker and rival distance maps. Use two digits per distance and display distance mod 100 with a space between each distance (example output will be posted to Piazza with this PDF).

All code is to be written in C.