COM S 327, Spring 2023
Programming Project 1.04
Player Character and Trainers

We've had maps for a few weeks, and now we also have path finding. We've reached a point where we can add the player character and other Pokémon trainers. User interface comes next week, so for this week, add some routines of your own devising to move the player character (*PC*) around. The PC movement doesn't have to be clever; you can move it randomly, along a chosen vector, whatever you like; even allow it to stand still (i.e., don't bother adding movement routines for the PC at all). Just make sure that you keep it inside the border (don't enter the boarder or an exit).

The PC is represented by an '`@`'. This is simply the way things are done in Roguelike games, and Roguelike gamers are traditionalists, so, sorry, you can't change this. Place your intrepid `@` somewhere on the road. Add a switch, `--numtrainers`, that takes an integer count of the number of trainers to scatter around, and hard code a default number of trainers to place on the map when the switch is not present. Something close to 10 is reasonable for the size of our maps[1]. Ensure that the code you write to generate trainers always loads at least one hiker and at least one rival (unless numtrainers is less than 2).

In roguelike games, NPCs are usually represented by letters, but occasionally numbers and punctuation are used. For the most part, there are no conventions here, beyond that each letter represents a certain class of bad guy, and that class usually begins with the letter that is used to represent it. For instance, it's common for humans (people) to be represented by `p`, humanoid non-humans by `h`, giants (big people) by `P`, and dragons by `D`, but Smaug, Ruth, Falcor, Saphira, and Norbert are all `D`s (and Rand would still be a `p`). If you've got color, you might make those `D`s gold, white, white, blue, and black, respectively.

We are going to have 7 kinds of NPCs:

- **Hikers**: These will be represented by the letter 'h'. Hikers path to the PC by following a maximum gradient on the hiker map.
- **Rivals**: These will be represented by the letter 'r'. Rivals path to the PC by following a maximum gradient on the rival map.
- **Pacers**: These will be represented by the letter 'p'. Pacers start with a direction and walk until they hit some terrain they cannot traverse, then they turn around and repeat, back and forth.
- **Wanderers**: These will be represented by the letter 'w'. Wanderers never leave the terrain region they were spawned in. They have a direction and walk strait ahead to the edge of the terrain, whereupon they turn in a random direction and repeat.
- **Sentries**: These will be represented by the letter 's'. Sentries don't move; they just wait for the action to come to them.
- **Explorers**: These will be represented by the letter 'e'. Explorers move like wanderers, but they cross terrain type boundaries, only changing to a new, random direction when they reach a bit of impassable terrain.
- **Swimmers**: These will be represented by the letter 'm' (*Merfolk?* Because I've already assigned 's' and 'w'). Swimmers move like wanderers. Any road or path which is adjacent to water in any of the four cardinal directions is a bridge, and swimmers can swim under bridges[2]. We'll render swimmers over the bridge—even though they're logically under it—so that they don't "hide". Swimmers will

[1] You don't have to hard code this. You're welcome to implement some logic that decides how many trainers to place in some dynamic way.

[2] You may want to do a little refactoring of path generation code to add a bridge terrain, which will probably be simpler than checking special cases on roads when you move swimmers.

also swim directly toward the PC whenever the PC is cardinally adjacent to water or bridge tiles. No character will move into a map cell occupied by another character. No NPC will spawn in a cell occupied by another character. No NPC will spawn in or move into an exit. No NPC may spawn or move into a cell wherein the movement cost for that NPC type is infinity.

The game is driven by a priority queue. Characters are placed in the queue and removed according to the time of their next move. The time of the first move is 0. Each subsequent move time is the current time plus the movement cost for the terrain that the character is moving to. The current time is always exactly the time of the move of the most recently dequeued character. In this way, there is no need to have an iterative timer (if you are counting time up in a loop, you're doing it wrong!). Nothing happens in the world when it's nobody's turn.

You already have at least one priority queue. You can use it again, or write or find another. Using it again is the best choice.

Redraw the map after each PC turn, pause so that an observer can see the updates (use `usleep(3)`, which sleeps for *argument* number of microseconds; something like 250000–which corresponds with 4 frames per second–is reasonable). If the PC moves, distance maps must be regenerated.

The game can run until you kill it with control-C.

All code is to be written in C.