COM S 327, Spring 2024
Programming Project 1.02
Moving from map to map

In the first week we generated maps that had gates[1], theoretically to other maps. This week we'll be using those gates to connect to other terrain maps.

Later, we'll use a library called *curses* to add a proper user interface. For now, we'll simply print to the standard output and read from the standard input at the terminal using standard I/O. We're going to implement 6 commands, all of which will have to be followed with the enter key because of buffered I/O (something *curses* will eliminate for us). The commands are:

- **n**: Move to the map immediately north of the current map and display it.
- **s**: Move to the map immediately south of the current map and display it.
- **e**: Move to the map immediately east of the current map and display it.
- **w**: Move to the map immediately west of the current map and display it.
- **f** x y: x and y are integers; Fly[2] to map (x, y) and display it.
- **q**: Quit the game.

Other input should be handled gracefully; don't crash, and—ideally—print an error message.

Our world will measure 401×401 maps; we begin in the center, which we'll call map (0,0) but may be easier thought of internally as (200,200). That is, you'll probably want to represent your world as a 401^2 matrix of maps wherein the PC starts on physical map (200,200) which is displayed with the coordinate (0,0). Movement commands that would cause display to move off the edge of the world are ignored (i.e., redisplay the current map, optionally accompanied by an error message).

At all times, display the logical coordinates of the current map beneath the map itself.

Later, when we add Pokémon to the game, the levels of wild-spawned Pokémon will be a function of the distance from the center.

The frequency of Pokémarts and Pokémon Centers should decline as we move out in order to increase game difficulty. Your starting position (the center, map (0,0)) should always have both buildings. As we move out, the probability of each building should be a function of the distance. This is a value that can be tweaked for game balance later, but something like $\frac{-45d/200+50}{100}$ gives the chance of a building being created for distance less than 200, and a flat 5% for distances 200+, where d is the Manhattan distance from the center, is reasonable[3]. This gives roughly a 50% probability of each building when close to the center, falling off linearly to 5% near a center edge or farther.

When generating maps adjacent to previously-generated maps, gates should match up. For instance, if you begin your game at (0,0) and move `n n e e s s w n`, you will be in (1,-1) (assuming n is the negative y direction. All of (1,-1)'s neighbors will have been generated before (1,-1). The height position of (1,-1)'s west gate should match that of (0,-1)'s east gate, (1,-1)'s north gate should align with (1,-2)'s south gate, and so on for each direction. To implement this, you'll want your map gates to be a parameter to your map generator, which may require a small refactor of the 1.01 interface.

No map should ever be generated until it is visited. All visited maps should be saved such that they can be visited again by going to the original coordinates (revisiting a coordinate should never generate new terrain).

[1] I'd previously been referring to these as "exits". I never liked that name, but had failed to come up with anything better. A student referred to them as gates and I liked it, so I'm calling them gates now.

[2] It would be **t**eleport or **j**ump, but this is a Pokémon game

[3] In case you're wondering where this odd formulation came from, it's just the slope-intercept form of a line through the points $(0, 50)$ and $(200, 5)$.

In addition to aligning neighboring gates, maps on the edges of the world should not have gates on their world-bordering edges.

The suggested approach is to have a 401×401 array of pointers to maps. Initially all maps in the array are NULL, except for the center (internally (200,200); externally (0,0)). When generating a new map, malloc the necessary storage and assign it to the appropriate pointer in the array. Checking for the existence of a map is a simple matter of testing a pointer for NULL.

All code in C!