COM S 327, Spring 2024
Programming Project 1.06
Moving to Neighboring Maps and Porting to C++

Remember way back in 1.02 when we generated a whole world full of maps and added the ability to move between them? Let's add that functionality back in. But now, we'll put it behind our new user interface that we built using Curses.

The PC can now use the movement keys to move into a gate. Doing this does not leave the PC standing in the gate; instead, it places the PC in the in the road square directly abeam of the gate in the neighboring map. For example, if the west gate is at (0,10) in map (0,0), moving west into (0,10) will place the PC at (78,10) in map (-1,0).

NPCs cannot move into gates.

NPCs and their respective turn queues should be maintained across movement through gates. To achieve this, you'll probably want a turn queue per map. Time does not (need to[1]) pass in a map where the PC is not present.

Additionally, implement fly, as per 1.02, but using the curses interface you built last week. The command 'f' indicates that the player wants to fly. The interface will then need to read X and Y coordinates. Placement within the destination map is at your discretion (on the road is probably a good idea). Turn queues and NPC positions should also be maintained while flying.

We'll be converting (*porting*) our entire game into C++ this week, as well. You'll need to rename all of your C files with a `.cpp` extension (if using my makefile, do a `make clobber` before renaming). You'll also need to update your makefile. If you're using one of my code drops (or if you've taken my makefile into your project), there is already automatic support to compile C++ code; however, you will need to change the link line to link with the C++ compiler (try to figure this out on your own).

The major structs in your game should be changed to classes; this includes the map and the character structures. There are no requirements about access to members (public, private, etc.); you decide what you think is best. You also don't need to implement methods on these classes if you don't want to. You may simply take what was called `struct` in 1.05, call it instead `class`, make everything public, and make it compile.

My code uses a pseudo-object-oriented design of the `character_t`, with sub-types `pc_t` and `npc_t`. If you are using my code, you should change these so that they use C++ class inheritance (`pc` and `npc` should inherit from `character`[2]). If you're not using my code, but you have a similar design, you should also use inheritance[3].

You do not have to do anything with `heap.h` and `heap.c`. These already include the necessary syntax to make them play nice with C++ and may remain as C files.

---

[1] It can if you really want, but that will be more work for you

[2] This is a requirement, not an optional change.

[3] Requiring this would be harder to enforce—a lot of manual effort for the TAs—so we won't do that, but you should make an effort to use inheritance somewhere in your code if the organization allows it (your notions of character, pc and npc are are very sensible place for this!), just so that you get the experience.