Programming Assignment 2

Assignment Details

Assigned: February 28th, 2013. Due: March 16th, 2013, midnight. All code should be submitted in the program (yes, still program, as it continues to build on everything else we've done there) directory in your source control tree.

Background

So, we've got some stuff working for the overall project, but all of our critters and loot are for naught if they don't have someplace to go. For this assignment, we're going to build dungeon levels.

What makes up a dungeon level

So, let's look at nethack:

This is a typical dungeon level. Ignoring the items on the ground (which we'll work on later), the main features to note are rooms, passages, and stairways.

So, the first thing to notice is the size of the dungeon level. Ignoring the blank line at the top, the status bar at the bottom, and leaving an extra spare character on the side, we get a dungeon size of 20 tiles tall by 79 tiles wide.

Then we notice that we have two types of navigable tiles – "room" tiles (represented as **.** in game) and "tunnel" tiles (represented as **#** in game). These two tile types represent passable tiles – those that the player (and monsters!) can move through. Any other tiles (most notably, blank tiles) are considered impassable.

So, we're going to need a class to represent a dungeon level. Let's keep it simple and call it DungeonLevel.

Requirements for a dungeon level

Each level generated must meet the following requirements:

- Be 79 tiles wide by 20 tiles tall
- Must contain at least 200 "room" tiles
- All rooms must be rectangular and not overlap[1]
- All rooms must be connected by passable tiles (room or tunnel)
- All rooms must contain at least 16 tiles
- Must contain both an upstairs and a downstairs

Having distinct walls (as above) it not necessary, but will give some extra credit; if you do implement walls, they count as impassable tiles – and for full credit, the whole level has to be accessible – no walking through walls!

Requirements for the program

- You must provide an executable which generates a dungeon level and then outputs it to the console; at minimum, you should draw room tiles as "**.**", tunnels as "**#**", and up/downstairs as "**<**" or "**>**".
- You must provide an executable which implements a unit test verifying that a randomly generated dungeon level matches the specifications for a dungeon level. This program should take, as a command line argument, the number of times to run – each run should generate a new random level and test it.
- You must provide example output of both executables.
- The executable which generates a single dungeon level and displays it should be called prog2, and be generated by "make prog2"; the executable that performs the unit test should be prog2test, and generated by "make prog2test".
- The levels must be generated randomly, and use the Mersenne Twister (see below) class to generate random numbers.

[1] This may be the hardest part of the assignment to test. Note that if you choose to generate walls, you probably want to the entry tile in a room to be a tunnel tile as opposed to a room tile, as otherwise testing for the rectangularness of the rooms gets considerably more difficult.

- Everything should be integrated into the existing makefile (which will be used the rest of the semester).

Notes

- This assignment is somewhat free form[2]. It is intended to be; in most engineering situations, whoever is giving you requirements is usually going to be much more interested in what the output looks like than the process to get there.
- We will discuss strategies for setting up the rooms in class on Friday, February 28th, and likely the week after the midterm as well. The practicum on March 12th will cover PA2 stuff as well.
- How the tiles are stored in the class is largely up to you – you just need to be able to output them to the screen and test them when you put them together. Keep in mind that you will need to be able to store the location of Entity objects in the near future – be thinking about whether you want to make location part of Entity, or containment part of DungeonLevel. Each can work, but have advantages and disadvantages. You are, however, *strongly* suggested to use a two-dimensional vector of `Tile`s, where `Tile` is a new class you create to represent dungeon tiles.
- For full credit, there needs to be variation in all room locations and sizes; i.e., you can't anchor a room in each corner and randomize the rest of them. I won't be looking as closely at tunnels, though.
- You probably don't want to *fully* randomize placement of the rooms, just to make it easier on you.

Randomness

The standard random number generator – `rand()` – is terrible. Instead of using it, we're going to use STL's Mersenne Twister engine.

To use this, you will need to include the `<random>` header file. The class name is `mt19937`, and you'll need to construct an object, seed it with something (`time(NULL)`[3] is a reasonable seed), and then you get random numbers by calling the `()` operator[4] on the object, as below:

```
mt19937 mt;
mt.seed( time(NULL) );
// This generates a random number, chosen from
//all possible unsigned ints
unsigned int iRandom = mt();
// This generates a random number in the range 0-99.
int iLimitedRandom = iRandom % 100;
```

---

2 And if you think *this* is free form, just wait for PA3…

3 Found in the `<ctime>` header

4 Yes, this is unusual, but is inserted here as a follow up to function pointers: `()` is an operator which evaluates something. The `mt19937` class is set up to respond to it and generate a new random value…