COM S 327, Fall 2018
Programming Project 1.08
Loading Monsters and Objects

This week we'll be loading all of the objects and monsters that we parsed last week.

Add a class for instances of objects in the dungeon; remember that last week's assignment created a class for *descriptions* of objects, not *instances* of objects. There are 8 sets of dice in our object descriptions. Of these, only *damage* should remain dice in an object instance. All others get rolled to become integers. I'll leave it up to you to figure out how to appropriately bring in the other fields from a description to an instance.

Add a method or methods to your object description class to generate dynamic instances of objects. A method that generates instances of a class is known as a *factory*, so our object description class is now an object factory. Similarly add a method or methods to generate instances of `npc`. `npc` (and probably `character`) will need to be extended to handle all the new fields. Like object instances, only *damage* will remain dice when instanced.

Upon generating instances, place them in the dungeon. In the case of characters, this means extending or replacing the current monster generation routines. Objects must go on the floor, but they can be walked over, so there's no need to, e.g., check for a monster before placing an object.

I/O routines must be updated to render the new monsters and objects appropriately with colors. In order to render with color in curses, you must take the following steps (all of which are already done in my code in the implementation of the message queue):

1. Initialize the curses color subsystem by calling `start_color()` when you initialize curses.

2. Color attributes are stored as numbered pairs in curses, each having a background color and a foreground color. Since there are 8 colors, this makes 64 color pairs; numbering starts from one, and we'll need only seven of them. Black on black cannot be read (the character is invisible), so if an object or monster is defined to be black, we'll render it white. Define a color pair with `int init_pair(short pair, short f, short b)` For example, to initialize the cyan pair, I used: `init_pair(COLOR_CYAN, COLOR_CYAN, COLOR_BLACK)`. Hint: The first parameter is the integer that curses uses to index the pair and the colors themselves are integers, so if you use the same constant for the integer as you do for the foreground color, you can always specify colors by name without an extra lookup table.

3. To render in color, call `attron(COLOR_PAIR(index))`, render your symbol, and turn the color attribute off with `attroff(COLOR_PAIR(index))`. If you take my hint above, you can, for example, render in blue by specifying the attribute `attron(COLOR_PAIR(COLOR_BLUE))`.

Some monsters have multiple colors. You may use the first color only. An implementation of multi-colored monsters will use a flickering effect to move between colors, but the actual implementation is outside the scope of this class. My solution will implement it. Yours may if you like and have time (hint: look into the `select()` system call).

When a character moves over an object, the character should be rendered, but the object should remain in place so that when the character moves the object is rendered once again.

Load at least 10 objects per dungeon level. Don't forget to clean up and properly deallocate objects and monsters when leaving a dungeon level or quitting the game.

Each artifact object may have no more than one instance in existence. Furthermore, once the PC picks up an artifact, it become ineligible for future recreation even if the PC destroys[1] the object or abandons it on the dungeon floor.

Similarly, each unique monster may have no more than one instance in existence. That instance become ineligible for generation on future dungeon levels only once it is killed. Thus, a generated but still living unique monster in a dungeon may be generated again after the PC takes a staircase.

Select which monster or object to create by:

1. uniformly select a random description from your vectors of descriptions;
2. if the item or monster is ineligible for generation, go to 1;
3. choose a random integer between 0 and 99, inclusive. If this number is greater than or equal to the selected monster's or object's rarity, go to 1;
4. Generate the object or monster and place it in the dungeon.

The following table provides a mapping of object types to symbols. Optionally, you may allow objects to stack (not required, which is what "optionally" means, but somebody will ask). If you choose not to do so, then ignore the "stack" symbol. If you do choose to allow stacks, devise a consistent way to select a stack's color, for instance, by using the color of the top object in the stack.

Type	Symbol
Not a valid type	* (might be useful for debugging)
WEAPON	|
OFFHAND	)
RANGED	}
ARMOR	[
HELMET	]
CLOAK	(
GLOVES	{
BOOTS	\
RING	=
AMULET	"
LIGHT	_ (underscore)
SCROLL	~ (tilde)
BOOK	?
FLASK	!
GOLD	$
AMMUNITION	/
FOOD	,
WAND	- (hyphen)
CONTAINER	%
STACK	&

[1] Object destruction is part of 1.09.

What follows is optional.

There are a handful of new monster abilities: the ability to pass through walls, the ability to pick up objects, and the ability to destroy objects. Telepathic pass-wall monsters will always move toward the PC as if walls were not there; there is no additional cost for these monsters when moving through walls, and they do not bore tunnels. Non-telepathic pass-wall monsters can pass through walls while moving to a location where they have seen the PC. Monsters that can pick up objects will always pick up when they pass over an object on the dungeon floor. This object will then enter the monster's inventory and be dropped when the monster dies. Monsters that destroy objects will always destroy any object that they pass over.

Of course, since this is optional, you may interpret the semantics however you like.