Practicum 9 – Class Factory

Assignment Details

Assigned: April 1st, 2014. Due: April 4th, 2014 at midnight.

Background

This programming assignment introduces the factory design pattern, and provides a step of PA3.

Design patterns

It has been mentioned in class that design patterns are not features of a language, but instead common strategies for solving problems. In this lab we're going to both use a pattern we've already talked about – the singleton – and add a new one – the factory.

So what exactly are we doing?

For PA3, we're going to need to be able to randomly generate enemies to populate our dungeon with. In this lab, we're going to write a class – EnemyFactory – which does that for us, and uses the factory pattern to do so.

Since this procedure requires some data that we don't want to reacquire each time we need to generate a enemy, we're going to use a singleton object to acquire and store this data, and then generate enemies for us on demand.

Update your source control tree – you should see a la9 directory which has a framework to get started. This framework also includes a correct solution for PA1.2 which you can integrate into your PA3 if you so decide…

The singleton

First things first, the singleton has to be set up. This should be done the same way described in Practicum 6 – note that the header file has everything you need for this practicum, we should only need to edit EnemyFactory.cpp (and then main_la9.cpp) to get everything working.

The factory

Once the singleton is set up, you will need to actually acquire the data. In the EnemyFactory constructor (which should only ever be called once by virtue of the singleton pattern), open up a file for the provided file critters.xml, and call the parseXML function (declaration in parser.h) on it to populate a vector of XMLSerializable pointers. Once that's complete, you can then populate the Enemy vector member of EnemyFactory by converting those pointers to Enemy pointers. This conversion (from base class pointer to derived class pointer is done as follows):

```
Enemy * pEnemy = dynamic_cast<Enemy*>(pObject);
```

Where pObject is an XMLSerializable pointer. Note that pEnemy in this case can be NULL if the underlying object is not an actual Enemy pointer!

Generating the enemy

So, once that's done, we're going to need to generate a random enemy from the list that is of a specified level or lower – after all, we don't want to drop dragons on our players in the first room[1]…

While there are a number of ways to do it, the simplest is going to be to create a new vector of enemies, iterate through the stored vector, and push_back every element that is the specified health or below. Once you're done with that, pick a random number from 0 to the size of this new vector minus 1, and then you've got your pointer.

But – we're not done yet. Since we'll need to be able to have multiple enemies of the same type, we can't just return the pointer, we need to copy it. So for the return value, we'll call the copy constructor[2] to get a new object that's a copy of the original – that we can modify elsewhere without changing the original. The syntax for doing this via new is:

```
Enemy * pReturnValue = new Enemy(*pEnemy);
```

Where pEnemy is a pointer to a enemy we picked out randomly.

And then?

Once that's done, edit the main file for the lab assignment (main_la9.cpp), create five random enemies of level 3 or lower, and dump them to the console (via the dumpObject method that's already there).

[1] Second room, though? That's debatable.
[2] Yes, this is fairly evil, but don't worry, actually implementing a copy constructor is part of the homework this week.