Practicum 8

Assignment Details

Assigned: March 26th, 2014. Due: March 28th, 2014 at midnight.

A complete submission will include the modified code from the practicum.

Provided Code

In your la8 directory under source control, you should find a correct solution to Programming Assignment 2.

It should be pointed out that the provided solution is, while correct[1], not necessarily *good*, and you are encouraged to use your own PA2 if it looks better.

The Practicum

So, we've got a correct solution. And what we're going to do in this practicum is error handled – which means we're going to need to break something before the error handling matters.

So, let's open up DungeonLevel.cpp, and make a change that causes it to sometimes generate invalid levels[2]. On line 111, there's a line of code that handles an edge case in dungeon level generation – if the generated room doesn't line up with the pre-generated tunnels, and can cause the level to be generated with unreachable rooms. So, let's comment it out, and rebuild. Run a few tests, and you may not see anything immediately – from my testing, commenting this line out causes only about 6.5% of the levels it generates to be invalid.

So, what we're going to do next is to modify the test program – dltest.cpp – so we use exception handling during the unit test to break out of the testing loop[3] when an error is found.

So, add a try/catch block around the actual loop, so we get something like:

---

[1] Technically speaking, it doesn't cover the requirement of being integrated into the PA1/PA1.2 makefile, but the source provided is minimal to allow you to easily integrate it into your code, if you so desire.

[2] If you're going to use this for PA3, remember to unbreak it!

[3] I should point out that this particular exercise is somewhat contrived, as there are plenty of other ways to break out of the loop, but seeing as how I needed to tie in a PA2 solution and exception handling, well, choices were limited.

```
try
{
      for( int i = 0; i < iTestCount; i++ )
      {
           testDungeonLevel(mt);
      }
}
catch(string sError)
{
        cout << "Error: " << sError << endl;
}
```

Which sets up the handling of the exceptions… Now all we have to do is actually throw them. For this part, we're just going to go through the .cpp file, and in all of the places in the code where we generate an error message, instead of writing it to the console, we'll throw an exception, so the handler will pick it up, and break us out of the big loop.

So:

```
cout << "* INVALID ROOM SIZE *" << endl;
```

becomes:

```
throw string("Invalid room size");
```

Note that:

```
throw "Invalid room size";
```

Will ***NOT*** work, because (as explained in lecture), exception handling is one case where implicit conversions do not happen – we must explicitly throw a string, not a string literal (which is of type const char *, as opposed to string).

Once this is done, you should be able to rebuilt dltest, run it, and notice that the error message is displayed and the loop stops as soon as it hits an error.