Benjmain R. Olson, May 24, 2014
CS 162 – Introduction to Computer Science
Instructor: Terry Rooker
Project 4 Reflection:

Discovering Requirements

This project is about modifying and extending existing code to illustrate inheritance and polymorphism. The requirement modifications are:

(1) Making the parent class an abstract class,
(2) Virtual function to implement taking damage,
(3) Override the virtual function in at least one subclass,
(4) Redefine a function,
(5) Three substantial changes to the class hierarchy.

Designing a Solution

Here is how I fulfilled each of the above requirements (listed in same order, here):

(1) character class has the pure virtual function modifyRoll()
(2) function calcDamage() was made a virtual member of the character class
(3) calcDamage() was overriden in child classes goblin, barbarian, reptile, and descendent class dragon (but not in blueMan so as to provide a contrast that highlights inheritance)
(4) newly introduced function tailWhip() in class reptile was redefined in its child class dragon
(5) These are the changes listed in 2 through 4. In addition to this, dragon class was added as a child of reptile class.

Implementation

This is fulfilled in combat_modified.cpp, and modifications to the original, combat.cpp, are detailed below.

Implementation procedure: Modify original and compile-run to test after each modification, where I break the code and compile enough times until all error messages are gone and the code is fixed.

Testing and Debugging (refer to cs162project4testPlan_olsonbe.pdf for details):

I tested each function separately as I implemented them, instead of creating a separate file, I thought it would be more efficient to use again use gdb and set breakpoints and intervene by calling the functions within gdb. For testing the final product with all logic paths through the program, I sometimes ran the program and sometimes used gdb. In the future, if I have separate files, I'll likely want to separate out the classes into files so they can be called from a test file.

(Reflection section follows on next page)

Reflection

Changes made to the original combat.cpp:

All changes made to the original combat.cpp file are marked with the comment //modification in the new file combat_modified.cpp where it differs from the original combat.cpp file. For example, I added virtual int modifyRoll(int delta) = 0; to the parent class character declaration. In addition to lines marked with //modification, many function signatures changed slightly to accept the modified vector that stores pointers to objects of type Character (parent type).

Toward the beginning, I had to change the vector that stores the characters from type Character to Character* (in the brackets <>) and then follow the compiler error messages through all the function parameters that accept that vector, which was tedious, but I believe it was an improvement over the original code because instead of copying entire objects into the vector as before, now what I'm storing in the modified vector is pointers to dynamically allocated objects (I believe pointers are smaller than storing copies of the objects).

I ran into the issue of figuring out how to delete a character object that is dynamically created inside a function and then whose address was assigned to pointer stored in a vector. I couldn't delete the object via the pointer, so without taking the time learn smart pointers (or other advanced pointer stuff), I just added the line “delete c.at(x);” before “c.erase(c.begin() + x);” in function removePlayer().

I decided to also do this project as a single file so I could focus on the issues of polymorphism and inheritance apart from the issues of separate compilation or dealing with separate files, to make it easier to debug as I learn to modify and extend classes.

Challenges:

Slicing problem encountered: While developing and debugging, the problem was that I was wanting to make a vector of pointers where the vector is of parent type but elements added are of child types. For example, std::vector<Parent*> p did not work with p.push_back(Child pointer type). I realized I had two problems:

(1) I was not declaring the Child elements dynamically. In other words, once I started using the following type of declaration – Child* c = new Child; – things worked fine.
(2) For functions added to the Child class, I was not declaring them virtual in the Parent class.

One of the most difficult bugs to identify was when I moved calcDamage() into the parent class. It turned out the attacker object was calling it when it should have been called by the defender object, so I changed
attacker->calcDamage(attacker, attackRoll);
to
defender->calcDamage(attacker, attackRoll);

The most interesting issue I had was resolving a bug where the redefinition of tailWhip() in dragon was not getting called. The problem was that, with the class hierarchy character → reptile → dragon, reptile.modifyRoll() was public but dragon.modifyRoll() was protected, so I believe I had two versions of modifyRoll(). Also, because I was not overriding modifyRoll() in the child, it was only getting called in the parent. To fix these two issues, I simply moved dragon's modifyRoll() into the public section of its class and then overrided modifyRoll() in the dragon class.

Straying from the design document:

A slight change from the design document was the function signature for calcDamage() was virtual in the actual implementation instead of pure virtual stated in the design. I also needed to add parent class variable for rollDice() to contain the result of the roll, deciding to hold this information in the class rather than outside because it is data directly related to the character object. So, I also added an accessor getRoll() to the class.

The greatest change from the design was that I decided to simplify the redefinition of tailWhip() in the dragon class whose parent is reptile, so that the only difference is a very minor change (dragon increasing defense roll by 10 rather than by 3 in the parent, reptile) but just enough to illustrate redefining a parent function in the child class.

In the design document, “dragon.tailWhip() redefined in reptile” should have been “reptile.tailWhip() redefined in dragon.” Also, function signature changed slightly to only take a bool as an argument.

Finally, I decided to keep the function initVector() and choosePair() and just modify those so the user can choose a pair of characters to compete from the five characters instantiated at the beginning of runtime. In other words, I decided stray a bit from the design by not letting the user create new characters.

This project was a good exercise in seeing how code can be simplified by moving some functions and data into a class so the code in the function body can directly access members as opposed to passing in an object and using its accessor methods. Also, I will probably want to encapsulate more of my functions into libraries and classes in future projects.