Peter Moldenhauer
moldenhp@oregonstate.edu

CS 162 Assignment 3 Reflection

Requirements:

For this assignment there are numerous key requirements that must be met. First of all, this program will focus on the use of inheritance and polymorphism. I need to create a program that demonstrates this concept and the hierarchy of base and derived classes. The "Creature" class will be the base class and then the class hierarchy will flow down to other creatures that will participate in combat. The base class (Creature) needs to be an abstract class as well. I need to develop numerous functions that will simulate each of the creature objects "fighting" each other. Certain creatures also have special abilities which I will have to account for and override their attack or defense functions if conditions are met. Lastly, I need to develop a testing suite that will allow for the user to automatically simulate all of the possible combinations of creatures fighting each other. Through this testing suite the user will be able to see that all of the functionality is working correctly.

Design Plan:

I will create a creature class (which will be the base class) and then five classes derived from the creature class. These derived classes will be the vampire, barbarian, blue men, medusa and Harry Potter classes. The creature class will be an abstract class so that it cannot be instantiated. I will also create a functions class which will "hold" various functions that I will use in my main.cpp file. I will use this function class to de-clutter the main function and to keep the main function as short as possible. The main function (main.cpp) will be able to execute a sequence of combat sessions between two objects derived from the creature class. It will also display each creature's current status after each attack/defense move and will declare the winner of the battle when the attacks are over. The main function will contain a loop that will allow the user to play repeated battles among the creatures. In the main function I will also contain the testing suite option in addition to including an option for the user to view the game rules.

Detailed look at the classes:

Creature class

- Protected members:
    - Name
    - Armor points
    - Strength points
    - Dead/not dead variable
- Public functions (pure virtual functions):
    - Attack
    - Defense
    - Get name

    - Roll dice for attack roll
    - Roll dice for defense roll
    - Return if creature is dead or not
    - Destructor

Vampire, barbarian, blue men, medusa and Harry Potter classes:

- Protected members (inherited from creature class):
    - Name
    - Armor points
    - Strength points
    - Dead/not dead variable
- Public functions (inherited from creature class):
    - Attack
    - Defense
    - Get name
    - Roll dice for attack roll
    - Roll dice for defense roll
    - Return if creature is dead or not
    - Destructor

Classes with special abilities:

It is important to note that all of the derived classes except for the barbarian class will have special abilities. I will have to take these special abilities into account as I develop the classes. The vampire class will be modified so that 50% of the time an opponent does not actually cause any damage to the vampire. With the blue men class, I need to "take away" a die for every 4 points of damage that is received. I will modify the class by using flag variables to notify if the total strength of a blue man is > 8, if the strength is <= 8 && > 4, and lastly if the strength is <= 4. With the medusa class I need to make modifications to allow for an "instant kill" to the opponent if medusa rolls a 12 between two six sided die. Lastly, I need to modify the Harry Potter class so that Harry Potter will "revive" after being killed. This second life for Harry Potter will come back with more strength points but will be his final life as well.

Testing Plan:

Throughout the process of building the program, I will use the process of incremental development. I will work on the most fundamental functions first and test each of them right away. This way, I will be sure that each function works before I build any additional functions (which will most likely use the earlier functions). In order to verify that damage is correctly applied during each turn of gameplay, I will use print statements to display the total attack/defense points (dice roll by attackers/defenders), the starting armor and strength points, and the armor and strength points throughout the course of the battle. I will also print out the winner when the game is over. I will also include various print statements throughout the course of each battle that notify when a special ability for a

given creature is encountered. For example, if a vampire is attacking Harry Potter, I will print out a line stating when the vampire "charmed its way out of an attack". Similarly, I will print out a statement that notifies the user if Harry Potter dies but then comes back to life with the "hogwarts" special ability. When I am finished coding the program I will run through a series of tests to manually collect data on the ratio of wins/losses of each creature fighting each of the other creatures in addition to fighting itself. I will then write up these results in a table for easy viewing. I expect that when a certain creature is matched against itself, the results will be about 50-50. I also expect to see that certain creatures (based on their special abilities and different die configurations) will win attacks more often than others. Throughout this series of tests with each possible creature match-up, I will also manually record the change of point values with pencil and paper. I will then compare my manual calculations of point changes in each step to the computer's calculations. In theory, the point values that the computer returns should be identical to my manual calculations if the program is running correctly.

Testing Results:

The testing process of this program was tedious and time consuming. This was mostly due to the fact that there was not only a lot of data to record by hand with each battle, but there were also a lot of battles to run. The large number of battles to run was due to the large number of creature objects and taking into account each possible creature match up. The actual data that I recorded was actually spot on with the computers returned data. The few issues/errors I had with my program were mostly not related to the calculations and numbers being returned. Running the testing suite also generally returned the same results as running by manually selecting each creature match up.

The following table lists out my recorded data of the ratio of wins/losses for 10 battles for each possible match up of the creatures:

Match up	**Player1 win**	**Player2 win**
Vampire – Vampire	50%	50%
Vampire – Barbarian	80%	20%
Vampire – Blue Men	20%	80%
Vampire – Medusa	90%	10%
Vampire – Harry Potter	10%	90%
Barbarian – Barbarian	50%	50%
Barbarian – Blue Men	0%	100%
Barbarian – Medusa	100%	0%
Barbarian – Harry Potter	0%	100%
Blue Men – Blue Men	50%	50%
Blue Men – Medusa	90%	10%
Blue Men – Harry Potter	100%	0%
Medusa – Medusa	50%	50%
Medusa – Harry Potter	0%	100%
Harry Potter – Harry Potter	50%	50%

Program changes and how I resolved problems:

Overall, the process of coding my program went fairly smoothly because I spent a lot of time planning the design and thinking through possible issues. One of the first issues that I encountered was that I was getting a warning message from the compiler about the virtual destructors. My program would compile but I would still get a warning message each time. It turns out that I did not fully understand how the destructors worked when using an abstract class and derived classes. After some research online (stack overflow) I was able to pinpoint my error (errors with how I coded the destructors in my derived classes).

A change that I made from my initial design plan was that I added "get" functions for the armor and strength points for each creature. This allowed me to print out these numbers to the user before the battle started so the user would know the starting stats on each creature. In the midst of the attacks the screen gets cluttered so it is nice to see the starting armor and strength levels for each creature and how they differ among opposing creatures.

Lastly, I had a few errors implementing the special abilities for a few of the creatures. With medusa for example, I knew that I would have to modify each class to "kill" an opponent if medusa rolls a 12. Initially I just used "if statements" which returned a value of -999 if medusa rolled a 12. I also used if statements then in each creatures' defense function to kill the creature if -999 was passed to it. However, the -999 was automatically getting changed into a different number when the opponents rolled their defense die. This made -999 never cause the opponent to die because -999 always turned into a different value. Similarly, I encountered some issues with implementing the Harry Potter special ability of coming back to life. Ultimately I fixed these errors by using a "flag" variable and if statements to notify when Harry Potter gets killed at he can come back to life one more time. I also had to make sure that Harry Potter would only come back to life one time and not every single time he died.

Reflection:

This assignment was fairly straightforward. It helped tremendously by writing out a design plan and having this design to work off of when I started to code. Having an idea of how all the classes fit together was also very helpful when I began to code the program. Overall, my design matched my final program fairly well. There were only a few relatively small changes and errors that I had to spend some extra time on. These were things that most likely could have been avoided if I took them into account at the time of my design plan. Perhaps the most annoying part of the program was typing up the code for the testing suite. This was annoying to me because I used incremental development as I coded the classes and tested each class I wrote them on their own. I literally coded and tested the entire program before I coded the testing suite. I feel like it would have complicated things if I did it any other way (ex. coded the testing suite first) so the process of coding the testing suite did not benefit me a whole lot. I assume the

testing suite is intended more for the graders of my assignment to have an easier time in testing each possible creature match-up instead of manually typing the selections in through the menu.

Overall, I thought this was a fun project and was a great way to learn about using an abstract class and pure virtual functions. I am excited to see what is next to do with these classes that I built for the next assignment.