Kristen Harrison
CS 162
Project 3: Creature Battle

Program requirements:

Create a polymorphic class hierarchy as a basis for a fantasy combat game. Creature must be an abstract base class, inherited by the individual character classes. Characteristics are as follows, where attack and defend indicates a number of dice with a certain number of sides.

Type	Attack	Defense	Armor	Strength
Vampire	1D12	1D6	1	18
Barbarian	2D6	2D6	0	12
Blue Men	2D10	3D6	3	12
Medusa	2D6	1D6	3	8
Harry Potter	2D6	2D6	0	10 → 20

Special powers or incidentals:
Vampire has a 50% chance of defusing an attack before it happens (Charm)
Blue Men lose one defensive die for every 4 strength points lost
Medusa, when rolling an attack value of 12 (Glare), kills her opponent unless the Vampire rolled charm
Harry Potter comes back to life after his first death, with a strength value of 20

Print out the results of each round, as well as the final winner to the screen. Create a test driver class to demonstrate the class through an interactive game.

Design:

This is the only assignment that I've needed to greatly restructure after writing it. I misunderstood some of the instructions the first time; it sounded to me like the creature classes were supposed to be largely standalone and able to handle processes internally like working out the damage and loss within defense and updating their own strength members, so that all that would be needed was an outside testing program to call attack and defense functions for the creatures to be able to interact with each other and implement the effects of the battle. ("Medusa must use Glare from the Medusa class", "add whatever you want to the parent class", and the rubric implying that only Vampire and Medusa class should have overridden functions).
But then TA clarification on piazza indicated that the classes should not be directly interacting with one another and that we needed to have at least one function be purely virtual and preferably not the destructor, so I rewrote everything and it was much simpler and easier to just have each creature designated with its own attributes and powers and no awareness of other classes, with an outside class

handling the processing. It also makes more sense to do it this way considering that this assignment is about learning and practicing polymorphism.

I originally had all functions written out in Creature, because the only ones that needed modification were Medusa’s attack, Vampire’s defense and setStrength, BlueMen’s defense, and Harry Potter’s setStrength. Each class had a type of Die for attack and defense and int numbers of attack and defense die. So the default function was just a for-loop that cycled through the dice and added the roll to the total. Medusa’s attack set the other Creature’s strength to 0, BlueMen rolled fewer dice depending on strength, HarryPotter had int lives set to 2 and if he lost one life, his strength was set to 20. The Vampire had some extra functions and variables so that if Medusa’s glare was activated it affected tempStrength, not strength itself. There were a set of conditional statements in Vampire’s defense that checked for charm and whether tempstrength != strength (which indicated Glare had happened), and output the corresponding messages and updated strength or not.

This approach worked, but it felt convoluted, decentralized, and messy, and I generally avoid having cout messages come directly from the derived classes. The other drawbacks were that it was difficult to figure out how to avoid the attack printing out if Vampire defended with charm, there were no functions that clearly should have been purely virtual, because all of them were inherited, and it required more information being passed back and forth. After seeing the requirements clarified on piazza I rewrote it all to have the Game class handle the processing, and this solved all the previous issues I was having, and made the entire project more cohesive and easier to understand. I do think the assignment could have been more clear; it was worded in a way that made it sound as if the functions should be mostly inherited from base and that the derived classes should handle their own processing internally.

New design:

Creature has attributes for name, armor, and strength. The attack() and defend() functions are purely virtual. Besides these there are also accessor and mutator functions and empty constructor and destructors. Creature’s data members are protected to give the derived classes access to them.

Creature – abstract base class
protected: string name int armor int strength

public:
Creature();
virtual int attack() = 0;
virtual int defend() = 0;
virtual string getName();
virtual void setName(string newName);
virtual int getArmor();
virtual int getStrength();
virtual void setStrength(int newStrength);
virtual ~Creature();

The Creature base class is inherited by Barbarian, BlueMen, HarryPotter, Medusa, and Vampire. Each of these classes include as private members the individual dice needed for attack and defense, as well as name, armor, and strength defined. Because attack and defend are purely virtual, they are defined in every one of the derived classes, and HarryPotter also has a unique setStrength function in order to handle his two lives.

All these classes are very similar, so I'll just include the most detailed one here:

class HarryPotter : public Creature

Die attackDie1;
Die attackDie2;
Die defenseDie1;
Die defenseDie2;
int lives;

HarryPotter();
int attack();
int defend();
void setStrength(int newStrength);
~HarryPotter();

In most cases, the attack and defend functions just make an int variable attackPts or defensePts and add the value of each die roll to this before returning it at the end. Special cases are as follows:
– A series of if statements in BlueMen::defend() determine how many dice are rolled. Statements check if strength is > 0, > 4, and > 8. In this way, progressively more dice are rolled depending on the strength threshold.
– HarryPotter has int lives set to 2 at the outset. If his lives are still at 2 and his strength has fallen to zero or lower, his lives are decremented and his strength set to 20.
– If Medusa rolls a 12 (Glare), the attack function returns 1000.
– If Vampire activates Charm (a rand statement that checks for an even number), function returns 2000.

I also have a Game class that controls the flow of the game, and provides the processing for the return values of attack and defense and the implementation of special powers. Game has 3 Creature* pointers: creature1 and creature2, to handle the battling foes, and winner, which is assigned to the winning creature. The game constructor sets all these to null and seeds srand. The playGame function handles subsequent program flow by introducing menus for user input, and routing control to chooseCreatures (which calls creatureMenu and assignCreature to assign the pointers to new creatures), then calls turn repeatedly until a player wins, then calls printWinner to print the final stats.

The Game class is responsible for creating the new instances of Creatures, and therefore contains two pointers to Creature objects (Creature* creature1 and Creature* creature2). The winner pointer is assigned to the winning creature.

class Game

Creature* creature1;
Creature* creature2;

Creature* winner;

Game();
void playGame();
void chooseCreatures();
int creatureMenu();
void assignCreature(Creature* &cr_ptr, int creatureNum);
void turn(Creature* attacker, Creature* defender);
void printWinner();
~Game();

This project reinforced for me that pointers have to be passed by reference in order to be used in another function. I didn’t realize or didn’t remember that just a copy of the pointer is passed otherwise. That was really the only problem I had with the Game class; everything else was straightforward.

A loop in playGame runs until a winner emerges. For each round, two turns are called, one for each creature to attack the other. In each turn, an attack and a defense are called, and several if/else statements assess the results and make the appropriate print-out messages and strength updates.

Testing:

Type of test	Expected output	Actual Output	Comments
menu works	Play again/quit. And the 5 creature options	menu validates input properly	
characters can be created	creature objects function through pointers	core dumps (creature pointer was still null because I had passed in a copy)	in pointer assignment function, had to send by reference
Barbarian vs. Barbarian	normal dice rolls until death	yes	normal dice roll ranges are tested further down
Barbarian vs. BlueMen	BlueMen win	yes	
Barbarian vs. Vampire	tossup and Vampire uses charm half the time	yes	
Barbarian vs Medusa	tossup; chance of being turned to stone	yes	I ran it enough time to see that Glare wins
Barbarian vs HarryPotter	HP comes back to life if killed	yes	
BlueMen vs BlueMen	defense rolls decrease over time	yes, noticeably	the rolls stay within the ranges in other tests
BlueMen vs Vampire	BlueMen win, though often charmed	yes	

BlueMen vs Medusa	BlueMen win but Medusa may Glare	yes	
BlueMen vs HarryPotter	HP resurrects if killed	yes	
Vampire vs Vampire	lots of charming (50%)	yes	
Vampire vs Medusa	Vampire charm stops glare	yes	when I hardcoded charm and glare to occur, charm won out
Vampire vs HarryPotter	half charm defenses from V and HP has 2nd life	yes	
Medusa vs Medusa	even match unless Glare	yes	
Medusa vs HarryPotter	Harry Potter comes back to life if Glared at	yes	
HarryPotter vs HarryPotter	comes back to life once if killed, but only once	yes	
Charm	comes half the time, and results in no loss of strength	yes	
Glare overrules everything except Charm and HP 1st life	program will only show the Charm if coincide; glare ends game unless HP has not died yet	yes	
memory management	no leaks	there were problems at first with how I had freed memory	it corrected when I set pointers back to null in playGame() rather than the destructor
random numbers	are random and within specified range	yes	see tests below for specific ranges
game ends if defender dies in first round	i.e. no counterattack in that round	yes	
starting attacker is random	50% chance creature1 will start	yes	
name updates if two of same creatures fighting	BlueMen and BlueMen_2	yes	
winner correctly displayed	final stats and accurate winner name	yes	
math is correct	strength deducted as indicated	yes	
HP attack rolls	2 to 12	yes	
HP defense rolls	2 to 12	yes	
Vampire attack rolls	1 to 12	yes	

Vampire defense rolls	1 to 6, possible charm	yes	
Barbarian attacks	2 to 12	yes	
Barbarian defense	2 to 12	yes	
BlueMen attacks	2 to 20	yes	
BlueMen defense	3 to 18 at strength 9 - 12 2 to 12 at strength 5 - 8 1 to 6 at strength 1 - 4	yes	
Medusa attacks	2 to 12 (Glare)	yes	
Medusa defense	1 to 6	yes	

Overall the tests mostly worked because by the time I tested individual creatures, any kinks were worked out. The issues I ran into while coding were: thinking that passing a pointer in as a parameter would work (but it only passes a copy), so trying to use the attack() function gave me core dumps because the pointer didn't point to anything. As well as where to nullify the pointers at the end, and getting undefined behavior when I tried to make the destructor virtual to satisfy the abstract class requirement in my first design.