Meredith Kiessling
CS 162-400-w16
2.14.2016
Assignment 3

Assignment 3: Dungeons and Dragons

Description

The assignment is to create a series of 'Dungeon & Dragon's' type creatures using polymorphism. The Creature base class must be pure virtual. Each subsequent character will inherit the characteristics of the base Creature Class, with each creature having unique attack or defense behaviors for combat. The Creatures duel one another using special dice specific to each character for both attack and defense.

Approach

General Class Hierarchy: Creature is base class, and all other creature classes (Barbarian, Blue Men, Harry Potter, Medusa, & Vampire) 'is-a' Creature and are polymorphic. In addition to the Creature Classes, there is also a Die class. The Die class creates a die object with sides = s. The Creature Class 'has' two Die object pointers, one for an attack die and the other for the defense die. See the attached Class Hierarchy.

The general approach to the assignment was to create the Die class, Creature class and Barbarian class (to define the basic attack() and defense() functions). Then create the additional creatures, building onto those attack()/defense() as needed.

a) Creature Class
   i. Protected Data Members : String name, int attack, int defense, int quantityAttackDie, int sidesAttackDie, int quantityDefenseDie, int sidesDefenseDie, Die *attackDie, Die *defenseDie, int armor, int strength
   ii. Public Members
      i. Creature () default constructor – initializes all int variables to 0.
      ii. Creature(Creature &beast) – copy constructor /*place holder in case needed*/
      iii. Virtual int attack(Creature *beast) = 0– (PURE VIRTUAL)
      iv. Virtual void defense(int damage) = 0 - (PURE VIRTUAL)
      v. getStrength(); /*returns strength of creature*/
      vi. getName() /*returns name of creature*/

b) Die Class – Die Class creates the attack and defense die for the Creatures and rolls those dice.
   i. Private Data Members: int quantity, int sides, int seed (for srand)
   Public Members
   i. Die() default constructor
   ii. Die(int q, int s) – overload constructor (creates a die with s sides to be rolled q times)

iii. Int roll() - rolls die with s, sides q times and keeps a running total. So if the die is 2d6, it rolls a 6-sided die twice and sums the two rolls together, then returns that total value.

c) Barbarian Class
   i. Protected Data Members : (inherited from Creature Class. Typical for all Creature sub-classes unless otherwise noted).
   ii. Public Members
      i. Barbarian() : Creature () default constructor – initializes all fields to specific values, (typical for all Creature sub-classes).
      ii. Virtual int attack(Creature *beast) – basic attack function – roll the attack die and return the damage.
      iii. Virtual void defense(int damage) – basic defense function - takes in the damage returned from the opponents attack. Roll defense die and substract (defense + armor) from the total damage (call net damage). Then subtract the net damage the from the available strength points. Set logic flags, so components can go negative!
d) BlueMen Class (**only changes from Barbarian Class are noted, typical**)
   i. Virtual void defense(int damage) –. Basic defense function - but adjust the number of defense dies rolled, with the loss of every 4 strength points. If strength -4, roll two dice. If strength -8, roll 1 die, else roll 3 dice.
e) Harry Potter Class
   i. Private Data Members: bool isFirstLife - bool flag used to determine if this is
   ii. Virtual void defense(int damage) – basic defense, until strength < =0. If this isFirstLife- set strength = 20; If strength <=0 and it's not his first life, remove from combat.
f) Medusa Class
   i. Virtual int attack(Creature *beast) – same as basic attack, except, if Medusa rolls 12, set opponent's strength to 0. Print out "Medusa Glare! Medusa Wins!"
g) Vampire Class
   i. Private: bool charm; bool flag used to indicate if charm is used
      i. Virtual void defense(int damage) – use rand()% to randomly generate 0, 1. Then use those to set the charm flag. If charm = true, "charm" attacker out of attacking (e.g. don't apply damage). If charm = false, continue with basic defense.

Meredith Kiessling
CS 162-400-w16
2.14.2016
Assignment 3

Testing

Test Case	**Input**	**Function**	**Expected Outcome**
Create 1 die with 6 sides and roll it repeatedly.	1, 6	roll() Die(1, 6);	Should return numbers between 1-6, of somewhat equal distribution
Create 2 die with 6 sides and roll them 1000 times. Keep track of who wins	1,6	roll() Die(1, 6);	Distribution of wins between both die should be almost equal.
Create 2 dice. One with 6 sides, the other with 12. Roll them 1000 times. Keep track of who wins	1,6 1, 12	roll() Die(1, 6); Die(1, 12);	12-sided die should win significantly more times than the 6-sided die.
Create 2d6 'die' and roll it 1000 times	2,6	roll() Die(2, 6);	Should return numbers between 1-12, of somewhat equal distribution
Create two 2d6 'die' and roll it 1000 times. Keep track of who wins.	2, 6 2, 6	roll() Die(2, 6); Die(2,6);	Distribution of wins between both die should be almost equal.
Create one 2d6 'die' and one 2d12 'die' and roll it 1000 times. Keep track of who wins.	2, 6 2, 12	roll() Die(2, 6); Die(2,12);	2d12 die should win significantly more times than the 2d6 die.
Create Barbarian through Vampire Objects and print strength, and armor for each	Barbarian b(); . . . Vampire v();	Barbarian() . . . Vampire v()	All objects should have characteristics specific to the object, as specified in the default constructor

Create Barbarian through Vampire using Creature *pointer and print strength, and armor for each.	Creature *p = new Barbarian(); … Creature *p = new Vampire();	 Barbarian() . . . Vampire v()	All objects should have characteristics specific to the derived object as specified in the default constructor
Create q Barbarian objects and attack 1000 times (do same for Blue Men – Vampire)	Creature *p1 = new Barbarian();	P1->attack(p2);	Should return numbers between 1-12*, of somewhat equal distribution (*results vary on object's attack die)
Create two Barabians and fight them (attack – defense – attack defense) (do same with Blue Men through Vampire)	Creature *p1 = new Barbarian(); Creature *p2 = new Barbarian();	P1->attack(p2); P2->defense(dam); P2->attack(p1); P1->defense(dam);	P1's damage should be deducted from p2's strength (less the defense roll), and vice versa. Combat continues until attack/defense functions stop since strength =0 (Check! Are special attacks/defenses being enabled?)
Create a Barabian and Blue Men and fight 100 times(attack – defense – attack defense) (fight Barbarian against all other creatures.)	Creature *p1 = new Barbarian(); Creature *p2 = new BlueMen();	P1->attack(p2); P2->defense(dam); P2->attack(p1); P1->defense(dam);	Blue Men are stronger and should win more frequently. (Barbarian should lose most frequently on average than its opponent)
Fight BlueMen and Harry Potter. (attack – defense – attack defense)	Creature *p1 = new BlueMen(); Creature *p2 = new HarryPotter();	P1->attack(p2); P2->defense(dam); P2->attack(p1); P1->defense(dam);	Blue Men are stronger and should win more frequently, (this should be true when playing against all other

			creatures)
Fight HarryPotter and Barbarian. (then Potter and everyone else)	Creature *p1 = new Barbarian(); Creature *p2 = new HarryPotter();	P1->attack(p2); P2->defense(dam); P2->attack(p1); P1->defense(dam);	HarryPotter will win most frequently against Barbarian, less frequently against Vampire. Potter will lose to BlueMen on average.
Fight Vampire and Harry Potter. 100 times(attack – defense – attack defense) (fight vampire against all other creatures)	Creature *p1 = new Harry Potter(); Creature *p2 = new Vampire();	P1->attack(p2); P2->defense(dam); P2->attack(p1); P1->defense(dam);	Harry Potter is stronger and should win more frequently on average. Vampire will win against Barbarian but rarely beat BlueMen.

Reflection:

The completed project was close to the original design. I changed a few things.

- I did have to go back and add some accessor/mutator methods as needed throughout development.
- I set up my attack function to take in a Creature *pointer to the second opponent. I was originally rolling the defense die in the attack function and then just returning the net damage (instead of total damage), which didn't work once I got to the special defense functions. I had to re-write the attack function so it only rolled the opponent's die, and return the total damage (before deductions).
- However, having the attack function already setup to take in a Creature *pointer to the opponent was helpful for the Medusa special Glare attack. If the Glare attack was initialized, I set the opponents strength to 0 (within the Medusa attack function and a setter for the strength data member).

Items that I didn't change, but would have done differently:

- It was redundant copying the basic defense() and attack() methods for each Creature (and it became cumbersome to annotate and make changes in several places). Next time try creating a different pure virtual function within the Creature class so the basic, attack() and defense(), functions can be defined.
- When I got to the Blue Men class, I REALLY wished I set up my roll class and roll() function differently. I had it set up so the Die Constructor took in the quantity and the sides (so for a 2d6 die, it took in an int parameter for the quantity = 2, and an int parameter for the number of sides = 6). The Die class then creates a six sided die and in the roll() function, rolls that die q times and keeps a running total. For the Blue Men Mob, I had to create a new Die object, every time the Blue Men reached 4 fewer strength points, so the Blue Men would have one fewer die to roll.

Areas I had troubles:

- I had a hard time with the Test Driver menu. I originally have all the menu items in its own class, but I kept on having issues with memory leaks (none of this happened during my testing without the menu). The error would always be thrown when I selected '6' from the menu, signaling the program to quit. The issue was in my switch case I used to create the specific Creature objects. I had a case1 – case 5, but my menu allowed the user to enter 6 to quit. So, when I entered 6, it was initializing the switch case instead of quitting, which somehow caused a memory leak.

Jn 3
ssling
.2.2016