Jason Goldfine-Middleton
932675265

CS 162 Assignment 3 Reflection

Design Plan:

There will be a Creature class, five classes derived from the Creature class (Goblin, Barbarian, Reptile, BlueMen, and Shadow), and a CombatRunner class. The Creature class will be abstract so that it cannot be instantiated. The CombatRunner class will be able to execute a sequence of combat sessions between two objects derived from the Creature class, displaying each Creature's current status after each play and declaring the winner after the game is over. The main() function will contain a loop that allows the user to play repeated games and select which two types of creatures should battle each other.

Creature class:

Protected members:

- int armor
- int strength
- std::string name

Public members:

- Creature()
    - Simple constructor to seed the random number generator
- virtual int attack() = 0;
- virtual int defend() = 0;
- virtual int take_damage(int attack, int defense)
    - attack -= armor
    - damage = attack - defense
    - if (damage > 0)
        - strength -= damage
        - return damage
    - else
        - return 0
- bool is_dead()
    - return true if strength <= 0
- int sum_dice(int num_dice, int sides)
- virtual void revive() = 0;
- std::string get_name()

Goblin class:

Protected members:

- int achilles_cut
    - true if the Goblin has cut the opponent's Achilles tendon

Jason Goldfine-Middleton
932675265

Public members:

- Goblin()
    - set armor = 3, strength = 8, name = “Goblin”
- virtual int attack()
    - roll two 6-sided dice
        - if sum is 12, set achilles_cut to true
    - return sum
- virtual int defend()
    - roll one 6-sided die
    - return roll
- virtual int take_damage(int attack, int defense)
    - divide attack by 2 if achilles_cut is true
    - updated attack -= armor
    - damage = updated attack - defense
    - if (damage > 0)
        - strength -= damage
        - return damage
    - else
        - return 0
- virtual void revive()
    - restore strength to 8
    - restore achilles_cut to false
- bool get_achilles()
    - return true if has cut opponent’s achilles already

<u>Barbarian class:</u>

Public members:

- Barbarian()
    - set armor = 0, strength = 12, name = “Barbarian”
- virtual int attack()
    - roll two 6-sided dice
    - return sum
- virtual int defend()
    - roll two 6-sided dice
    - return roll
- virtual void revive()
    - restore strength to 12

<u>Reptile class:</u>

Public members:

- Reptile()
    - set armor = 7, strength = 18, name = “Reptile Person”
- virtual int attack()
    - roll three 6-sided dice
    - return sum
- virtual int defend()
    - roll one 6-sided die
    - return roll
- virtual void revive()
    - restore strength to 18

BlueMen class:

Public members:

- BlueMen()
    - set armor = 3, strength = 12, name = “Blue Men”
- virtual int attack()
    - roll two 10-sided dice
    - return sum
- virtual int defend()
    - roll three 6-sided dice
    - return roll
- virtual void revive()
    - restore strength to 12

Shadow class:

Public members:

- Shadow()
    - set armor = 0, strength = 12, name = “The Shadow”
- virtual int attack()
    - roll two 10-sided dice
    - return sum
- virtual int defend()
    - roll one 6-sided die
    - return roll
- virtual int take_damage(int attack, int defense)
    - if (rand() % 2 == 0), return 0
    - attack -= armor
    - damage = attack - defense
    - if (damage > 0)
        - strength -= damage
        - return damage

    - else
        - return 0
- virtual void revive()
    - restore strength to 12

CombatRunner class:

Private members:

- Creature *p1
- Creature *p2
- static const bool P1_WON = true
- static const bool P2_WON = false
- int total_sims
    - total number of simulations
- int cur_sim
    - current simulation

Public members:

- CombatRunner(Creature *p1, Creature *p2, int num_sims)
    - Precondition: do not pass anonymous pointers in!!
    - this->p1 = p1
    - this->p2 = p2
    - total_sims = num_sims
    - cur_sim = 1
- bool run_simulation()
    - while (true)
        - call attack() for p1
        - call defend() for p2
        - call take_damage() for p2
            - double damage if p2 is Goblin and cut opponent's achilles but opponent is also Goblin
        - if p2->is_dead()
            - revive p1 and p2 and return P1_WON
        - call attack() for p2
        - call defend() for p1
        - call take_damage() for p1
            - double damage if p1 is Goblin and cut opponent's achilles but opponent is also Goblin
        - if p1->is_dead()
            - revive p1 and p2 and return P2_WON
- int get_result()
    - int p1_num_wins = 0;
    - while (cur_sim <= total_sims)
        - if (run_simulation = P1_WON) increment p1_num_wins;

- return number of times p1 won

main():

- create two arrays (length 5) of pointers to Creature objects, setting each element to be each of the 5 derived class types
  - note: two sets of pointers are necessary to prevent an object fighting itself
- use nested loops to pair each pointer in the first array with each in the second
  - set NUM_SIMULATIONS to 5000
  - Print "matching p1->get_name() with p2->get_name()"
  - Create CombatRunner object cr with the pair of pointers and NUM_SIMULATIONS
  - set int p1_wins = cr.get_result()
  - calculate percentage of games won by p1 and print it
  - wait for user to press a key before next matchup
    - to allow for me to document the win ratio for this matchup
- free memory from pointers in the two arrays
- exit

Testing Plan:

I will use the process of incremental development, working on the most fundamental functions first and testing that they work before building any functions that use them. In order to verify that damage is correctly applied during each turn of gameplay, I will use print statements to display the values of the dice rolls, the sum of the dice, base attack, armor, defense, and current strength in the attack() and defend() functions. I will also print out the winner when the game is over. I will be using the p1_wins bool value to count the number of games the first player won in my main() function. This will help me fill out the win ratio table below. I expect that when a certain derived type is matched up with itself, the results will be about 50-50, but I expect that player 1 will win slightly over 50% of the time because each turn I first check whether player 2 is dead before allowing player 2 to attack player 1.

Testing Results:

Overall, coding went very smoothly because I spent a lot of time planning the design and thinking through possible issues. I used stubs minimally because I worked on the smaller, more fundamental functions first. I did end up creating a simple test driver main() function to demo a contest between two objects of the Goblin class, which was the first derived class I worked on. I initially had the result of player 1 always winning and with the same ratio each set of 50 games. It turned out that this was due to me running srand() with an argument of 0 rather than time(0). Another hurdle I overcame was that I received an error regarding an illegal memory access in main(), which turned out to be related to the fact that I had freed the memory for the pointers before the CombatRunner object using them had gone out of scope. I knew that I didn't want to delete any pointers from within the CombatRunner class as not new objects were created on the heap by that class. Therefore, I resolved the issue by placing the initialization of the CombatRunner object inside nested loops in main(), so that it would go out of scope before the pointers were deleted. Other than those issues, everything went smoothly.

Jason Goldfine-Middleton
932675265

In each set of simulations where a Creature object of a certain type fought its own type, the results were a little above 50% in favor of player 1. In fact, it was closer to 55%. I attribute this to the design choice of always letting player 1 attack first, and checking to see if player 2 was killed before letting player 2 attack player 1. For each matchup, I ran 5000 simulations. Please see the table below for the results of one sample run through the main() function.

		Player 2				
Player 1		Goblin	Barbarian	Reptile People	Blue Men	The Shadow
	Goblin	55.08%	34.82%	0.02%	0.00%	5.04%
	Barbarian	70.76%	53.44%	0.00%	0.00%	4.72%
	Reptile People	100.00%	100.00%	51.64%	13.44%	81.50%
	Blue Men	100.00%	100.00%	88.68%	52.86%	87.00%
	The Shadow	97.92%	97.92%	26.04%	15.56%	55.00%