Jacob Karcz

Assignment 3 Design

CLASSES

```
Class Dice { from lab B }

Class Creature {
Protected:
     Dice *diceArray;
     int  attackDice,
          attackDsides,
          defenseDice,
          defenseDsides,
          armor,
          strength;
Public:
     Virtual int attack() = 0;
     Virtual int defend(int) = 0;
     damageAssess(int strengthPts);
     int rollMyDice(int numDice, int numSides);
}
Class Medusa : public Creature {
     constructor() {sets creature's member variables}
     int attack() { RollDice (
     void defend (int hits)
}
Class Gollum : public Creature {
     constructor() {sets creature's member variables}
     int attack() { RollDice
     void defend (int hits)
}
Class ReptilePeople : public Creature {
     constructor() {sets creature's member variables}
     int attack() { RollDice
     void defend (int hits)
}
Class BlueMen : public Creature {
     constructor() {sets creature's member variables}
     int attack() { RollDice
     void defend (int hits)
}
Class HarryPotter : public Creature {
     constructor() {sets creature's member variables}
     int attack() { RollDice
     void defend (int hits)
}
```

FUNCTIONS

```
int rollMyDice (int numDice, int numSides) {
     int rollValue = 0;
//create the Dice
     for (int i = 0; i < numDice; i++)
          diceArray[i] = new Dice(diceFaces)

//roll the dice
     for (int i = 0; i < numDice; i++)
          rollValue += diceArray[i]->rollDice();

//delete the dice, set array to null
     for (int i = 0; i < numDice; i++) {
          delete diceArray[i]; diceArray[i] = NULL }

     return rollValue;
}

Virtual int attack() {
     attack = rollMyDice(attackDice, AttackSides);
     return attack;
}

Virtual void defend(int attack) {
     defense = rollMyDice(defenseDice, defenseSides);
     damage = (attack – defense) - armor;
     this->strengthPts -= damage;
     }

damageAssess(int strengthPts) {
     this->strength = strength – strengthPts;
}

friend class Battle {

creature attacker;
creature defender;

int letsBattle(attacker, defender) {
     attack = attacker.attack();
     defense = defender.defense(attack);
     damage = (attack – defense) - armor;
     this->strengthPts -= damage;
          //(IF this ptr is inherited by friends…)
     defender.damageAssess(strengthPts);
}
```

MAIN

```
Creature *fighter1;
Creature *fighter2;
Int AttackPts;

Prompt user for fighter 1 and fighter 2 types
Fighter1 = new (cin type)
Fightger2 = new (cin type)

do {
     fight(player1, player2);
     cout “this happened”
     fight(player2, player1);
     cout “that happened”

} while (flag)
          // maybe have an option where 1 letter is player 1 attack
          //and another  letter is player 2 attacks?

delete Creatures.


fight(offense, defense){
     attackPts = 0;
     attackPts = offensive->attack();
     defender->defend(attackPts);
}
```

Assignment 3 Testing

Because the different creatures each have different possible attack rolls, defense rolls, armor, and strength, its not that complicated to see who would be favored in the different possible matchups (assuming no special moves are used). The various creature attributes are as follows:

Medusa		Gollum		Reptile People		Blue Men		Harry Potter	
Armor:	3	Armor:	3	Armor:	7	Armor:	3	Armor:	0
Strength:	8	Strength:	8	Strength:	18	Strength:	12	Strength:	30
Attack Rolls:	2–12*	Attack Rolls:	1–6*	Attack Rolls:	3–18	Attack Rolls:	2–20	Attack Rolls:	2–12
Defense Rolls:	1–6	Defense Rolls:	1–6	Defense Rolls:	1–6	Defense Rolls:	3–18	Defense Rolls:	2–12
						# dice decrease w/ strength	2–12 1–6		

Using the values above, it's not hard to come up with probabilities on who would win in the 25 possible match-up scenarios. The following page displays the different possible matches, the likely match winner, and the observed results.

The testing input was simple. The program prompts player 1 and player 2 to choose a character (designated A-E). Then the user is prompted to enter 1 to fight another round. To test, I simply chose the characters and pressed 1. I streamlined the procedure by taking advantage of the cin buffer, typing everything in one line after the first prompt. For example, if I am fighting the Blue Men against the Reptile People, I would have input "D C 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1" at the first prompt, all my future choices were saved in the cin buffer and I'd only occasionally have to type more 1's (ie when fighting Gollum against himself).

After testing the various combinations and having observed the outcomes and special attacks emerge during the building process, I am convinced that the program is working as specified. The various characters win as expected, though the Reptile People have a fair shot at killing the Blue Men some of the time. In addition the special powers work as specified. Medusa attacks with a power of 666 if she rolls a 12, which instantly kills everyone except Harry Potter (if he's on his first life). Harry is revived to a strength of 10 after losing his first 20 strength points (effectively giving him a strength of 30). Gollum Rolls 3 dice 5% of the time, and the Blue Men lose a dice every time they lose a third of their original strength points. Furthermore, the user is kept somewhat informed of the player's attack and defense status every round. The user is also informed when Medusa uses glare or when Harry Potter is revived, as shown bellow.

```
Round 1... fight!
**********

Harry strikes with 10 attack points!
Medusa takes on 3 damage! Her strength is now 5!

Medusa used glare! You're not supposed to look directly at the
Gorgons!
Harry takes on 662 damage! His strength is now 0!
Harry used his magic potion, he recovers some of his strength.

    [ press 1 to fight another round ]
```

Player1		Player2	Winner	Favored
Medusa	Vs	Medusa	P2 P1 P1	50/50, no ties
Medusa	Vs	Gollum	P1 √ P1 √ P1 √	Medusa
Medusa	Vs	Reptile People	P2 √ P2 √ P2 √	Reptile People
Medusa	Vs	Blue Men	P2 √ P2 √ P2 √	Blue Men
Medusa	Vs	Harry Portter	P2 √ P2 √ P2 √	Harry Potter
Gollum	Vs	Medusa	P2 √ P2 √ P2 √	Medusa
Gollum	Vs	Gollum	P2 P2* P1	50/50, no ties
Gollum	Vs	Reptile People	P2 √ P2 √ P2 √	Reptile People
Gollum	Vs	Blue Men	P2 √ P2 √ P2 √	Blue Men
Gollum	Vs	Harry Portter	P2 √ P2 √ P2 √	Harry Potter
Reptile People	Vs	Medusa	P1 √ P1 √ P1 √	Reptile People
Reptile People	Vs	Gollum	P1 √ P1 √ P1 √	Reptile People
Reptile People	Vs	Reptile People	P1 P1 P2	50/50, no ties
Reptile People	Vs	Blue Men	P2 √ P1 P2 √	BlUe Men
Reptile People	Vs	Harry Portter	P1 √ P1 √ P1 √	Reptile People

Player1		Player2	Winner	Favored
Blue Men	Vs	Medusa	P1 √ P1 √ P1 √	Blue Men
Blue Men	Vs	Gollum	P1 √ P1 √ P1 √	Blue Men
Blue Men	Vs	Reptile People	P1 √ P1 √ P2	Blue Men
Blue Men	Vs	Blue Men	P1 P2 P1	50/50, no ties
Blue Men	Vs	Harry Portter	P1 √ P1 √ P1 √	Blue Men
Harry Potter	Vs	Medusa	P1 √ P1 √ P1 √	Harry Potter
Harry Potter	Vs	Gollum	P1 √ P1 √ P1 √	Harry Potter
Harry Potter	Vs	Reptile People	P2 √ P2 √ P2 √	Reptile People
Harry Potter	Vs	Blue Men	P2 √ P2 √ P2 √	Blue Men
Harry Potter	Vs	Harry Portter	P1 P2 P1	50/50, no ties

* Special Attack used

Creature behavior testing graph

Assignment 3 Reflection

This assignment seemed very complicated at first, but after reading it closely it wasn't as difficult as I expected to come up with a design for the program. The design I came up with proved to be an excellent blueprint. I only had to do some minor changes to the classes. Namely I gave HarryPotter a counter for his reincarnation. I also modified all the creatures' attack and defense functions to account for their different special powers and to print out their own messages about their attack and defense. I knew this would come and it was a simple modification.

The made a few modifications in regards to the functions I included in my program. I wanted to have a battle class or a battle function in main to handle the actual fighting matches between the creatures, but I decided to nix the battle function. Instead I simplified it to a fight function that took pointers to the character doing the fighting and the character doing the defending. I then call the same function again with the pointers reversed. This loop continues until one of the creatures wins or the user chooses to quit. I also had a getStrengthPts() function in the base class that I realized was pointless before I even started coding.

In main I tried to have a function handle prompting the user and creating the creatures, but passing the pointer back to main proved to be more complicated than I could handle. Since the function would only save me a few lines of code I decided to simplify it into a switch statement that prompts the user and returns the choice. The program then compares the choice against the possible creatures and instantiates the proper Creature. Another change to main, is that I changed the cout statements for the creatures' attack and defense status to the attack and defense functions. This way I didn't have to create functions to print out what happened and I was able to personalize the output to the different characters.

After writing my code without the powers my program seemed to work as expected, so I created the special powers, and again this went off without a hitch. My program worked as expected right away. This was highly unexpected given the level of difficulty I expected but I welcome it, and I hope I didn't oversee something detrimental. However the behavior is as expected, it compiles on flip, and it has no memory leaks. I could honestly say I had more issues with my 10-point lab last week than with this 100-point assignment this week. Which I appreciate because it took some time to wrap my head around the implementation of a circular-linked-list for this week's lab (especially the destructor).

The one miscalculation I made was I initially thought Reptile People were the toughest of the classes, but in testing I saw that Blue Men kept winning. At first I thought something might be wrong, but in spite of the Blue Men having less strength points and armor than the Reptile People, they have higher possible attack rolls and higher base defense rolls. This accounts for the discrepancy and I revised the Creature I favored in their match-ups even though the Reptile People do occasionally secure a victory (as expected given their closely matched variables).

If I were to spend more time making this program fancier, I would try harder to implement a function that prompts the user for a creature and returns an instance of it, somehow. However, since my code worked and something like that would leave me more vulnerable to memory leaks, I leave that for the future. I would also change the fighting scenario to make it more interactive somehow. Perhaps each character gets a letter to attack or something so that the battles are less tit-for-tat. I would just want to make it more interactive for the user. I would also create more creatures to make things more interesting and I would make everyone a bit more evenly matched so that outcomes would be more of a toss-up. King Kong, Godzilla,

Aliens, and Hercules are a few of the other classes I'd implement. I would also cut Harry Potter, but I'd leave Gollum because that's just ridiculous to have him in a fighting game.

Overall I am fairly pleased with my program and I think everything went smoothly in writing it. I initially had trouble figuring out how to test it, thinking I had to write a driver function to test the outcomes and control for the different dice rolls (that would be a different program!), but once I understood that I simply had to account for the different combinations and asses the functionality of the program, I was able to finish the testing fairly quickly. I could easily say the testing was perhaps the most challenging part of the assignment, though conceptualizing it seemed daunting at first (a detailed read-through helped though).