CS 172 - Programming Assignment 3

Mark Boady and Matthew Burlick - Drexel University

February 4, 2018

1 Overview

In this assignment, we will build a small piece of a game. In some ways this is an extension of what you performed in an earlier lab. This assignment focuses on combat between a player (which we'll refer to as the hero) and a series of monsters. Inheritance will be required to make all the monsters different without changing how battles work.

Whereas in other assignments we were quite explicit about requirements, in this assignment we'll leave a lot of the implementation details and decision up to you, albeit with some constraints.

Contents

2 The Enemy

Your first task will be to design and implement an enemy class. This will be the parent/base class of every monster our hero has to fight. It will supply the minimum functions that are needed for a monster to exist in the game. At

least one of the methods of this class must be abstract (thereby making this an abstract base class and not directly instantiable).

A good starting point for you enemy class is as follows:

- Various core attributes for any enemy. This should include (at least)
  - The enemy's health
  - the enemy's name
- A constructor
- A public interface to get and set attributes as needed
- Overloading of the string method.
- An **abstract method** that allows an enemy to be damaged by another
- An **abstract method** that allows an enemy to attack another enemy.

You may want to refer to the sample execution trace at the end of this document to help guide your decision making process.

3 Make Monsters

Create **4** monsters for the hero of your game to fight. Your monsters should be derived from the enemy base class. As per the description of the enemy class, you must at least implement methods for attacking and taking damage for each of these monsters.

Additionally, at least **1** of your monsters **MUST** have multiple attacks that it decides between at random.

4 Make a Player Character (Hero)

The hero is just a kind of enemy!

Create a hero class that derives from the enemy class.

A hero has the following properties:

- A hero has a name, chosen by the player.
- You should be able to print out a health status bar.
- A hero has 6 potions that each heals some points of damage.
- A hero has 10 fireballs, each do damage. (More damage than the sword.)
- A hero has a sword that does damage.
- A hero has a shield. When the shield move is chosen, the subsequent attack on the hero only does half the damage that it usually would. The shield is then put down prior to the hero's next decision.

When the hero's attack method is called, print out the hero's status and ask for input on which attack to use. Then do what is requested by the player (again, refer to the execution trace for further clarification).

The player may quit the game at any time by saying their action is "exit".

5 The Game

Implement a main program that plays the game. Your main file will be named **adventure.py**. You may choose your own names for any additional files.

The game should go as follows:

- Ask for the Hero's name.
- Ask how many enemies the hero is to battle.
- Create a list of that many enemies, whose specific type is chosen at random.
- For each enemy
  - Print out what monster is attacking.
  - Fight until the enemy or hero is dead.
  - If the monster died, print that out.
- Either declare the hero victorious or dead.

A single round of combat should go as follows:

- Hero attacks.
- If the monster is alive, it attacks.

Reminder: The player can type exit at any time to throw an exception. If this happens, you should tell the player "Thanks for Playing" and exit the program.

6 Testing

While you are encouraged to do your own unit testing, for this assignment we will require that you provide at least two text documents that provide the input for a run of your program for different numbers of monsters to battle.

Since some of your program's execution involves random decisions (in particular which types of monsters make up your list, and for monsters with two attacks, which attack is used), the exact execution trace will depend on the random number sequence. For debugging and testing purposes, seed your random number generate with zero at the beginning of your program. You can do this with random.seed(0).

7 Execution Trace

Here is an example of how the game could look with 2 monsters. Remember, you have a lot of freedom in the details of your execution.

```
Welcome to Adventure Battle!
What is the name of your hero?
Cloud
How many monsters will Cloud battle?
2
You have encountered a Bat!
Cloud: 100/100 health
Remaining: 10 Fireballs, 6 Potions
Enter Command: sword shield fireball potion exit
shield
Hide Behind Shield!
Bat bites you!
Cloud: 98/100 health
Remaining: 10 Fireballs, 6 Potions
Enter Command: sword shield fireball potion exit
sword
Sword Slash Attack!
Bat bites you!
Cloud: 93/100 health
Remaining: 10 Fireballs, 6 Potions
Enter Command: sword shield fireball potion exit
sword
Sword Slash Attack!
Enemy is defeated!
You have encountered a Wolf!
Cloud: 93/100 health
Remaining: 10 Fireballs, 6 Potions
Enter Command: sword shield fireball potion exit
potion
 You drank a potion.
Wolf scratches you!
Cloud: 85/100 health
Remaining: 10 Fireballs, 5 Potions
Enter Command: sword shield fireball potion exit
fireball
Fireball Attack Successful!
Wolf scratches you!
Cloud: 70/100 health
Remaining: 9 Fireballs, 5 Potions
Enter Command: sword shield fireball potion exit
fireball
```

```
Fireball Attack Successful!
Enemy is defeated!
You have defeated all enemies and saved the world. Good Job.
```

8 Grading

You will be graded on both the functionality and quality of your design.

- Enemy (10 points)
  - Abstract methods (5 points)
  - Public interface and constructor (5 points)
- Monsters (40 points) 10 per monster
  - Inherits from Enemy (1 point)
  - Monster design is distinct from all others (2 points)
  - Fully Functional Implementation (7 points)
- Player (20 points)
  - Player Chosen Name (3 points)
  - Health Bar (3 points)
  - Potions (3 points)
  - Fireballs (3 points)
  - Sword (3 points)
  - Shield (3 points)
  - Inherits from Enemy (2 points)
- Game (10 points)
  - Hero used correctly (5 points)
  - Enemies used correctly (5 points)
- Provided at least two test input files (10pts)
- Output is well formatted: (3 points)
- Name is comments of all files (2 points)
- Section in comments of all files (2 points)
- Files well commented (2 points)
- File Name Correct (1 point)

If you code has any runtime errors, a 50 point deduction will be taken. Only portions of the code that execute without errors will be graded.