Juhwan Lee
Professor Karla Fant
CS202
November 7th 2020

Program 3 Design Write-up

Operator overloading is best implemented when we create new abstract data types. This time, we want to create our own abstract data types implemented with a full set of operators and have them used by our OO programs. For this programming assignment, we will be creating four breeds of our own monsters. We will need to specify when they are active, how they relate to other monsters and their special abilities. We have to push up the common elements and derive the differences. To make it fun, we will have monster bashes where they will battle each other. We will need two different types of data structures for this program. First, there should be a data structure that supports the monsters available to battle. We will have a Binary Search Tree (BST) of all of the monsters available. We have an option to implement a Balanced Tree (of our choice) for extra credit. Second, there should be another data structure that supports the list of available attacks and defenses. This will be an array of doubly linked lists. The primary purpose is to support operators with our classes: =, +, +=, ==, !=, the relational operators, and the ability to input/output the data. First of all, the overall design of this program I thought is as follows. First, when running the program, the user has the following options: create monsters, display available monsters, and select monsters. When the user chooses to create a monster, they can choose what kind of monster to create. There are 4 types and each has its own characteristics. As soon as the monster is created, the monster object is stored in the binary search tree. And the user comes back to the main screen. When the user selects Display Available Monsters, the monsters stored in the binary search tree are displayed. Finally, when the user selects the select monster, the user can select one of the monsters stored in the binary search tree. When the user completes the selection, the screen moves to the screen where the attack option and the defense option can be selected. Attack options and defense options are already stored in the Array of Doubly Linked List, and the user is given different options depending on the type of monster. For example, type 1 monsters have different attack and defense options than type 2 monsters. The user can select two options, one attack option and one defense option. When the user finishes selecting the attack option and the defense option, the screen goes to the screen where the opponent monster can be selected. Opponent monster's attack option and defense option are determined randomly (once it is conceived in this way, it can be done in different ways depending on the programming situation). Then, the battle proceeds and the result screen is displayed. The outcome of the battle is determined by the monster's characteristics, attack and defense options (operation overloading plays an important role in this area). When the battle is over, the user returns to the first screen and can create a monster again, and battle again with different attack options and defense options with other monsters. The class design is as follows. First, create a base class called monsters, create four different monster type classes, and inherit them from the monster class. Then, create an attack class and a defense class, and form a "has a" relationship with the monster class. As a result, my overall program design is in this way, and there seems to be some changes while programming, but the overall design will not deviate greatly from the method described so far.