David An

Project 3: Design, Test, and Reflection

Design: The main goal for this program was to create a combat simulator using different characters with various attack/defense stats along with unique abilities. An abstract class of a Character is needed since all of the characters would need to inherit member variables such as attack, defense, armor, and strength points, and member functions such as the attack and defense functions. In order to get retrieve the attack and defense amounts, I need to utilize a Die class similar to the one I had in my previous lab. Each character rolls different number of Die and each character’s Die contain different numbers of sides so I would like to incorporate their Die usage into their own class structures. In addition, I need to write attack and defense functions within each Character class – meaning that my ‘fight’ method must be in the menu and will simply consist of a call to each of the characters’ fight and defense functions, likely within a while loop. I will also need to be overriding certain characters’ attack and defense functions to support the functionality of their abilities. One important design note is how to make sure that the battle is continuing until one of the characters is dead. To achieve this, I will need to implement a Boolean method for each character that returns true if the character is dead and false otherwise. This means that I will need getter methods for each of the character’s member variables, as I will likely need to be accessing and possibly modifying these throughout the setup and attack/defense processes.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Case | Input Values | Functions | Expected | Observed |
| Input too low | Input < 1 | Begin() [Menu] | Re-prompt for new entry | Re-prompt for new entry |
| Input in correct range | 1 <= input <= 5 | ^ | Appropriate character is chosen and created | Appropriate character is chosen and created |
| Input extreme low | Input = 1 | ^ | Vampire is chosen and created | Vampire is chosen and created |
| Input extreme high | Input = 5 | ^ | Harry Potter is chosen and created | Harry Potter is chosen and created |
| Input too high | Input > 5 | ^ | Re-prompt for new entry | Re-prompt for new entry |
| Input is incorrect type | Input = String | ^ | Re-prompt for new entry | Re-prompt for new entry |

Reflection: I began by creating the menu and abstract character class. I kept in mind the variables and methods that I needed to include in the Character class, including the virtual functions that needed to be implemented in each of the inheriting classes. One of the first roadblocks I ran into was how to incorporate the fact that each of the characters roll different numbers of die (with different numbers of sides as well) for their attack and defense functions. I decided to handle the bulk of the Die sum calculation within the Die class itself by creating different methods that return the sum of the Die depending on the number of Die that are being rolled. These methods would take in Die objects as parameters, which meant that I just needed to create member variables for the necessary numbers of attack/defense Die for each character and use these to calculate their attack and defense points. I then set the sum of these rolls as the attack and defense variables themselves which are used during the damage calculation step – after which they are overridden by the next roll methods. This made it relatively simple to write the attack and defend methods which retrieved these values and used them to calculate the inflicted damage along with displaying the information to the user. The next difficulty I had was implementing the Hogwarts ability for Harry Potter. The only way that made sense to me was to create a member variable for the Harry Potter class that could tell if the character had died already or not. I then modified his defend method to check whether the character was dead and had already died (at the end of the damage step). If the character died and had not died already, I just changed the Boolean variable to show that he has died and reset his strength points to 20. As long as I reset his strength points by the end of his defend function, the while loop should not recognize that he has died and should continue the battle as if nothing else had happened.