Project 3 – Fantasy Combat Game

CS162 – Intro to Computer Science – Section 400

Winter, 2018

Elizabeth Donato

*Introduction*

For the purposes of this assignment, I have implemented the first phase of the development of a fantasy combat game. This game uses polymorphism to allow characters to have different skills and abilities that can be used during combat. An abstract base Character class establishes characteristics common to both predators and prey, including attack and defense capabilities, armor bonuses, and strength points. Additionally, the base class contains virtual member functions for fighting and defending. From the base Critter class, classes are derived to represent the following types of combatants: Vampire, Barbarian, Blue Men, Medusa, and Harry Potter. Each of the derived classes establishes characteristics unique to the character type as specified by the assignment requirements. Combat is conducted in rounds in which each character is given the opportunity to attack. For each attack, both the attacker and defender generate random dice rolls to indicate the outcome of the attack. Combat is managed through a Game class that contains information about objects and standard utility functions. My initial design pseudocode, class diagrams, and testing plan are included below.

*Initial Design Pseudocode*

*Establish Combatants*

* Display Menu of options
  + If ! Exit
    - Get fighter 1 choice & instantiate fighter
    - Get fighter 2 choice & instantiate fighter

*Conduct Combat*

* While both fighters have strength remaining
  + Display heading for round stats
  + Fighter 1 attacks
  + Fighter 2 defends
  + Calculate damage to fighter 2
    - If fighter 2 has no strength left, fighter 1 wins
    - Else
      * Fighter 2 attacks
      * Fighter 1 defends
      * Calculate damage to fighter 1
        + If fighter 1 has no strength left, fighter 2 wins
  + Increment round counter

*Play again or exit*

* Prompt user to play again or exit exit
  + If they enter 0, exit the program
  + If !0, play game again..

*Class Diagrams*

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | **Creature Class** | | | | | | ***Member Variables*** | | ***Member Functions*** | | | | int strength  int armor  int attackRoll | int defenseRoll  int totalRoll  int newLives | Creature()  void setStrength(int)  int getStrength() | void setArmor(int)  int roll(int, int)  void newLife() | virtual int attack() = 0  virtual int defend() = 0  ~Creature | | | | | |
| |  |  |  | | --- | --- | --- | | **Barbarian Class**  **Derived from Creature Class** | | | | ***Member Variables*** | ***Member***  ***Functions*** | | |  | Barbarian()  int attack() | int defend()  ~Barbarian() | | | |  |  |  | | --- | --- | --- | | **BlueMen Class**  **Derived from Creature Class** | | | | ***Member Variables*** | ***Member***  ***Functions*** | | |  | BlueMen()  int attack() | int defend()  ~BlueMen() | | | |
| |  |  |  | | --- | --- | --- | | **HarryPotter Class**  **Derived from Creature Class** | | | | ***Member Variables*** | ***Member***  ***Functions*** | | |  | HarryPotter()  int attack() | int defend()  ~HarryPotter() | | | |  |  |  | | --- | --- | --- | | **Medusa Class**  **Derived from Creature Class** | | | | ***Member Variables*** | ***Member***  ***Functions*** | | |  | Medusa()  int attack() | int defend()  ~Medusa() | | | |
| |  |  |  | | --- | --- | --- | | **Vampire Class**  **Derived from Creature Class** | | | | ***Member Variables*** | ***Member***  ***Functions*** | | |  | Vampire()  int attack() | int defend()  ~Vampire() | | |  | | |
| |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | **Game Class**  **Includes Menu class and all derived Creature classes** | | | | | | | ***Member Variables*** | | | | ***Member Functions*** | | | int choice1  int choice2 | Creature \*fighter1  Creature \*fighter2 | int fighterAttack  int fighterDefend  int totalDamage | int fighter1Strength  int fighter2Strength  int round | Game()  ~Game() | bool playGame()  void displayRound()  int damage(int, int) | | | | | | |

*Initial Testing Plan*

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Description** | **Expected Outcome** | **Observed Outcome** | **Changes Needed** |
| User inputs zero for menu choice | Input is accepted.  Exit message is displayed.  Program terminates. | As expected | None |
| User inputs 1-5 for menu choice | Input is accepted.  New fighter is created with appropriate type. | As expected | None |
| User inputs letter or character for menu choice. | Input is rejected.  Error message is displayed.  User is prompted to re-enter choice. | As expected | None |
| User inputs floating point number for menu choice. | Input is rejected.  Error message is displayed.  User is prompted to re-enter choice. | As expected | None |
| User inputs space for menu choice. | Input is rejected.  Error message is displayed.  User is prompted to re-enter choice. | As expected | None |
| User inputs newline for menu choice. | Input is rejected.  Error message is displayed.  User is prompted to re-enter choice. | As expected | None |
| User inputs negative value for menu choice. | Input is rejected.  Error message is displayed.  User is prompted to re-enter choice. | As expected | None |
| Fighter 1 attacks. | Attack value is within range for fighter. | As expected | None |
| Fighter 2 attacks. | Attack value is within range for fighter. | As expected | None |
| Fighter 1 defends. | Defense value is within range for fighter. | As expected | None |
| Fighter 2 defends. | Defense value is within range for fighter. | As expected | None |
| Medusa’s glare is effective | Opponent strength becomes 0.  Medusa wins fight. | Medusa’s glare was depleting strength, but defender was still given the opportunity to defend. | Establish Boolean variable isStone in Game class to determine if opponent has been turned to stone. |
| Vampire’s charm is effective. | Opponent does not attack. | As expected | None |
| HarryPotter’s strength is 0 and HarryPotter has not used his extra life. | Harry Potter regenerates with full strength. | Message is not printing in appropriate place. The program is executing correctly, but because the message is being displayed in the wrong place it appears that Harry Potter is regenerating before his strength reaches 0. | Move the function to check for a new life so that it displays after current round information has displayed. Now the user sees that HP’s strength is 0 before seeing the message about a new life being used. |
| HarryPotter’s strength is 0 and HarryPotter has used his extra life. | Harry Potter does not regenerate and his opponent wins. | As expected | None |
| Defense roll is large enough to negate all damage. | Defendant is not affected by attack. | As expected | None |
| Defense roll is not large enough to negate all damage. | Defendant loses appropriate amount of strength points. | As expected | None |
| Fighter 1 has a strength of 0. | Game ends.  Fighter 2 wins.  Menu is displayed and user is prompted for choice to exit or play again. | As expected | None |
| Fighter 2 has a strength of 0. | Game ends.  Fighter 1 wins.  Menu is displayed and user is prompted for choice to exit or play again. | As expected | None |

*Reflection*

I spent a considerable amount of time designing this project before coding and as such, I had worked out the flow of the program and the logic behind the conditional statements prior to writing any code. The polymorphic nature of the various combatants made creating these Creatures relatively easy, and I felt that the most challenging part of the project was implementing the combat sequences. I added numerous statements into my program about what was happening during each phase of combat, and that helped me to make sure that things were running as I wanted them to and also to pinpoint problem areas. I did encounter two minor errors and I was able to resolve these issues fairly quickly.

First, I had designed the program to check and see if Harry Potter had a new life available prior to printing round information; however, by doing so my program was generating messages that indicated Harry Potter had regenerated when it looked like he had not yet lost all of his strength. By simply rearranging the sequence of these events so that the program checked for a new life after printing round information, the program would first show that Harry Potter’s strength was depleted, and would then generate Harry Potter’s new life.

Secondly, Medusa’s Glare was not working as I wanted it to. It was becoming effective upon a roll of 12, and it was depleting the opponent’s strength, but the program then allowed the defender to defend against the attack even though the defender had supposedly already been turned to stone. I resolved this issue by creating a Boolean variable in the Game class that indicated if a fighter had been turned to stone. Once a fighter was turned to stone, defending was no longer possible. Additionally, if Harry Potter was turned to stone with a new life remaining, the Boolean variable would be reset to indicate the Harry Potter was no longer stone.