Christopher Merrill
CS162
10/31/2016

Assignment 3 Reflection Document
(Design, Test Plan, Test Results and Reflection)

DESIGN OVERVIEW

I plan on creating an abstract base class called Hero that contains virtual attack and defend functions that will be used in the following subclasses: Vampire, Barbarian, Blue Men, Medusa and Harry Potter. The actual game logic (die rolls) will take place in a Combat class.

The special powers (charm, glare, etc.) will be executed in their respective subclass, either in the attack or defense function, and there will be a bool value that keeps track of whether or not a power is activated on a given turn.

DESIRED DISPLAY

The output after each round should look something like the following:

ROUND 1 RESULTS:

Player One's Vampire rolled an attack of: 5
Player Two's Medusa defended with a roll of: 3
Damage to Player Two: 2

Player Two's Vampire rolled an attack of: 6
Player One's Vampire defended with a roll of: 4
Charm Enacted Player One's Vampire charmed the opponent
Damage to Player One: 0

Player One Strength Remaining: 18
Player Two Strength Remaining: 6

ROUND 2 RESULTS:

Player Two's Medusa rolled an attack of: 12
Glare Enacted Player Two's medusa turned the opponent to stone
Player One has been defeated...
WINNER: Player Two

PSEUDOCODE

MAIN.CPP:

Display the main menu with options to play or exit
If the user decides to play
    Ask the user for player one and player two's hero type
    Create a new combat object
    Run the play loop
    Ask the user if they want to play again

COMBAT CLASS:

Contains two pointers to hero's, which represent player one and player two.

Play Function:

Start the game loop:
    Player one attacks and player two defends
    If player one is medusa and glare is activated
        Notify the user and declare the winner
    Otherwise continue the game loop
    If a player is a vampire and charm is activated
        Notify the user
    Display player one's attack results
    If Hogwarts or Mob was activated during the turn
        Notify the user
    If player one won the game
        Notify the user and declare the winner
    Otherwise display each player's remaining strength
    Reset any powers that were activated
    Player two attacks and player one defends
    If player two is medusa and glare is activated
        Notify the user and declare the winner
    Otherwise continue the game loop
    If a player is a vampire and charm is activated
        Notify the user
    Display player two's attack results
    If Hogwarts or Mob was activated during the turn
        Notify the user
    If player one two the game
        Notify the user and declare the winner
    Otherwise display each player's remaining strength
    Reset any powers that were activated
Repeat the game loop until a winner is declared
Clean up memory by deleting both player's hero objects

HERO CLASS (abstract base class):
This class will contain the following data members pertaining to each hero's stats:
The hero's name
The hero's armor
The hero's strength
The amount of sides on the hero's attack die
The amount of sides on the hero's defense die
Whether or not a hero's power has been activated (bool)
Whether or not hogwarts has already been used (bool, Harry Potter only)

This class will also contain two pure virtual functions, attack and defend, along with a few getter/setter functions for some of the data types (to be used in the main combat loop)

VAMPIRE CLASS:
Attack Function:
Roll the attack die using rand() and return the result

Defend Function:
Take the opponent's attack value as input
Generate a random number, 1 or 2 that represents the odds of activating Charm
If the number is one, don't activate charm
    Roll the defense die
    Calculate the damage done
    Update the vampire's strength
    Return the damage done
Otherwise if the random number is two
    Activate charm
    No damage is done (return 0)

BARBARIAN CLASS:
Attack Function:
Roll two attack die using and return the sum of both rolls

Defend Function:
Take the opponent's attack value as input
Roll two defense die and sum the results
Calculate the damage done
Update the barbarian's strength
Return the damage done

BLUE MEN CLASS:
Attack Function:
Roll two attack die using and return the sum of both rolls

Defend Function:
Take the opponent's attack value as input
If the Blue Men's strength is higher than 8
    Roll three defense die and return the sum all three rolls
If the Blue Men's strength is between 4 and 8
    Roll two defense die and return the sum of both rolls
Otherwise roll one defense die and return the result
Calculate the damage done
Update the Blue Men's strength

If the Blue Men's strength dropped below 12 this turn
    Activate Mob power
If the Blue Men's strength dropped below 8 this turn
    Activate Mob power
If the Blue Men's strength dropped below 4 this turn
    Activate Mob power

Return the damage done

MEDUSA CLASS:
Attack Function:
Roll two attack die and return the sum of both rolls
If the sum of both rolls equals 12
    Activate glare

Defend Function:
Take the opponent's attack value as input
Roll one defense die
Calculate the damage done
Update Medusa's strength
Return the damage done

HARRY POTTER CLASS:

Attack Function:

Roll two attack die and return the sum of both rolls

Defend Function:

Take the opponent's attack value as input
Roll two defense die
Calculate the damage done
Update Harry Potter's strength
If Harry Potter's strength is equal to or less than zero after the attack, and hogwarts hasn't been activated yet:

- Set strength to 20
- Activate Hogwarts
- Update the bool that says Hogwarts was used this game

Return the damage done

TEST PLAN and TEST RESULTS:

Test	Results	Comments
Input Validation: Check that the input validation works correctly in main.cpp by entering the following each time the program asks for an int: **-1, 0, 100, F, @**	**-1:** The program asks for a nonnegative value. **0:** The program asks for an integer greater than zero **100:** The program asks for an integer within the range of the particular menu function **F:** The program asks for an integer value **@:** The program asks for an integer value	No changes necessary. (I have been using the same input validation function since the beginning of the class, so this was expected to work)
Math Test: Play a round and assure that the damage and strength points are being calculated correctly at each step	The damage for each roll and the updated strength points were accurate in each round. And the correct winner was displayed.	No changes necessary. I also continued to verify the math in subsequent tests.

Charm Test: Play 5 matches with two vampires and check if Charm is activated on roughly 50% of the rolls.	**Number of rolls:** 103 **Number of times Charm was activated:** 45 **Percentage:** 44%	No changes necessary. The percentage is slightly below 50, but it's close enough to where I don't have any doubts in the rand() function.
Glare Test: Play through matches using two Medusa's until one of them rolls an attack of 12, to assure that the program displays the Glare notification and that the game ends	The game accurately displayed the Glare notification and the Medusa that used Glare was named the winner	No changes necessary.
Hogwarts Test: Play a match with two Harry Potter's and assure that each player uses Hogwart's only once, and that a player's strength is set to 20 after Hogwarts is activated	Each player used Hogwart's only once before the game ended, and on both occasions the player's strength was restored to 20.	No changes necessary.
Mob Test: Play a match with two Blue Men and assure that the Mob notification appears both when a Blue Men dips below 8 health and when they dip below 4 health	Mob notifications successfully appear when a Blue men dips below 8 or 4 health. (Only one notification if it dips from above 8 to below 4 in one turn)	No changes necessary. I specifically chose to only have one Mob notification occur in the case where a player loses two Blue Men in one roll.

REFLECTION:

I didn't have to make many changes to the logic of my initial design, the main game loop and the class specific attack/defend functions remain virtually unchanged in my program. However I did forget to to account for negative damage in my defense function, so I had to add two additional lines of code to each of my defense functions that changed the "damage done" to zero if it had a negative value.

I also had to add a decent amount of code to the combat class to assure that the display was formatted correctly. I ended up adding phrases to the hero class “powerText” and “powerTextTwo” that held class-specific text regarding the special powers (charm, mob, etc.). That way I could call generic getPowerText and getPowerTextTwo functions in my Combat play function to display the correct text for a given hero type.

I began my design (and most of my implementation) before receiving the email from the professor that we should randomly decide which player goes first. Therefore I had to add a rand() to the beginning of the game loop that decides which player goes first. This meant that I also had to update the game loop cout statements to contain generic “firstText” and “secondText” phrases rather than just hardcoding “Player One” and “Player Two” like I had originally planned. This made the code a little bit harder to decipher in the game loop, but the results are displayed accurately regardless of which player goes first!