CS 162 - Proj3: Fantasy Fight Club - Design + Reflection Document
Jeovani Vela
2/12/18

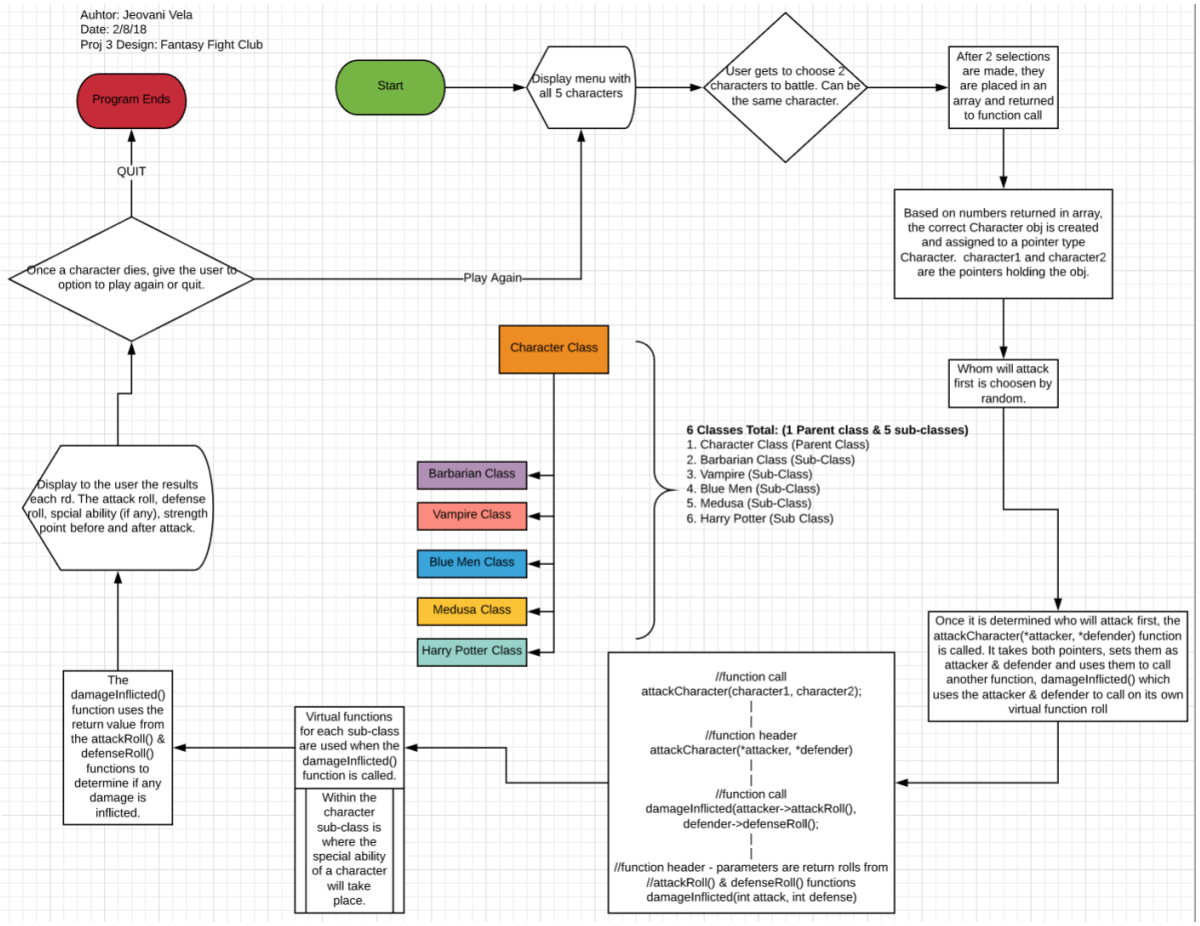
Project 3 was based on creating a fighting game amongst fantasy characters. There are 5 characters, which must inherit certain aspects from its parent class, Character. Thus, polymorphism is critical to the outcome of the game and classes created. Additionally, the use of special abilities by some characters adds a twist to the sub-class creation and the virtual functions as it must be determined in the sub-class if a special ability will be performed or not.

Requirements:

-There are 5 characters and each character needs its own class. Characters each have different number of die (for attacking and defense), armor, strength points and a special ability, except Barbarian which has no special ability.
    -Special abilities must be performed in the sub-class, not an outside function
-Develop 5 sub-classes from the parent class, Character using polymorphism.
-Gameplay consist of 2 characters battling one another until one dies. Each round consist of two attacks, one for each character. For each attack, attacked and defender both generate dice rolls.
    -The actual damage inflicted from the attacker onto the defender is calculated as follows:
        Damage = attacker's roll - defender's roll - defender's armor
-Character must serve as the base class and the actual characters the sub-classes.
-Must create a menu giving the user the option to choose 2 characters to battle.
-After battle is over, ask user to play again or quit.
    -If quit, program ends.
    -If continue, game starts over and user can re-choose which characters to fight.

Design:

When I began building my program, I started working on the base class first. As directed in the project 3 assignment sheet, it is easier to begin building the parent class followed by each sub-class.



Thus, as I built my Character class I deemed which member variables would need to be protected as they would need to be called throughout the program via getter functions. I also deemed which functions would become pure virtual functions based on rules laid out in the assignment. I had 3 pure virtual functions, the 2 functions for the rolls (attack and defense), and a bool function to determine if a character is dead or not. The constructor for my Character class consisted of only the life being set to 1 as each subclass would utilize their own constructor and only need to inherit the life from the Character class. Next I began working on my sub-classes. Each has their own constructor and destructor along with the 3 virtual functions so when called upon, the object's specific function will kick in. My Harry Potter specification file is the only sub-class with a private member variable as I had to use a bool value in the isDead() function to determine if Harry's special ability would kick in or not. All other character's special ability chances are performed in their own virtual attackRoll() or defenseRoll() function.

After my classes were built, I began working on my menu to make sure the user could select the characters. I decided to have the user inputs go into an array which would then be read out of to determine which characters the user chose. For example, if he/she chose option 1 and 2 (Barbarian & Vampire), I would read the subscripts out of the array and pair the numbers in each 2 subscripts (subscript[0] & subscript[1]) with certain if statements that would allow the correct character sub-class object to be created and then assigned to a pointer type Character. Once characters are created and assigned to the pointers, it is determined by random who attacks first. Attack character function takes the pointers as parameters and uses them in another function damageInflicted() function which uses the pointers to call on their retrospective attackRoll() & defenseRoll() functions as arguments so the damage inflicted can be determined. Characters will continue to battle one another until a character dies and a true bool value is returned to a while loop controlling the battle. I then created game functions that call on the necessary functions so that the game can be displayed and played properly before asking user to play again or quit after a character dies.

Test Plan:

Test Case	Input Values	Driver Functions	Expected Outcomes	Observed Outcomes
Start screen - menu displayed and user given 5 characters to choose from -selecting 1st character	`sgdfg, 435, 0, -1, sfd43, !@‘, a{}, 69, 04`	Input validator will only accept a number 1 thru 5	User will get message stating input is incorrect and choose a number 1-5	Program does not continue until user enters correct number 1 thru 5. Program accepts 04 and allows user to input 2nd character choice.
Start screen - menu displayed and user given 5 characters to choose from -selecting 2nd character	`45, thgfh, ‘!, [sg]l, 012, -13, af45, |l, 5`	Input validator will only accept a number 1 thru 5	User will get message stating input is incorrect and choose a number 1-5	Program does not continue until user enters correct number, 1 or 2. Program continues after user inputs 5.
Check if Vampire special ability, charm takes place and deflects attack.	`Set random values 1-4, if random value == 2 || 4, glare kicks in`	rand() % 4+ 1	User will get message stating Vampire used charm if a 2 or 4 is returned from the rand() equation	Vampire uses charm, displays message, deflects attack and the round/program continues.

Test Case	Input Values	Driver Functions	Expected Outcomes	Observed Outcomes
Check if Blue Men lose one defense die after every 4 points of damage	`No input– set Barbarian attack at 16`	Barbarian attack set at 16 should overtake any defense roll by Blue Men in order to deduct points	Blue Men will lose a defense die for every 4 points of damage and will roll a smaller defense value for every roll after losing a defense die	Barbarian imposes hard attacks that leave Barbarian with 1 defense die before winning the game and the Barbarian dying.
Check if Medusa's special ability, glare kicks in when a 12 is rolled	`Set Medusa's attack roll at 12`	With roll set at 12, Medusa will use glare and turn enemy to stone and thus win.	Message will display and show Medusa used glare and will win	Medusa battles Barbarian, uses Glare and wins the battle.
Check if Harry Potter can withstand Medusa's glare the first time and die the second time.	`Set Medusa's attack roll at 12`	With roll set at 12, Medusa will use glare and turn enemy to stone and thus win.	Message will display and show Medusa used glare and message will appear that Harry has survived and strength points has jumped to 20	Harry survives first attack with Glare, but dies in the next round as glare is used again by Medusa and he does not survive.
Check if Harry Potter's special ability Hogwarts kicks in when he dies the first time.	`No inputs. Battling Blue Men since they roll highest attack roll`	Program determines and tracks Harry's strength points before determining he will survive and have a jump in strength points to 20	Harry will survive the first time and die the second time.	Harry dies the first time, but a message appears saying he has survived and strength points is at 20. Program continues until Harry finally dies again and does not survive/no Hogwarts takes place.
After round ends, ask user to play again or quit	`fads, 121, 0. =–adsf, stop, q`	Program will validate the input before proceeding. User must choose between 'p' and 'q'	Program will not continue unless 'p' or 'q' is entered.	Program ends after a q is entered.
After round ends, ask user to play again or quit	`523, qwq34, (o]8(, a435435, p`	Program will validate the input before proceeding. User must choose between 'p' and 'q'	Program will not continue unless 'p' or 'q' is entered.	User enters p, the game starts all over again and the user can re-choose character to battle.
Check if program can handle two of the same characters battling one another	`3, 3`	User selects Blue Men to fight each other. Objects are created and assigned to pointers type Character	Program will continue and characters will battle until one dies.	Program does not crash and instead the battle plays out until one of the Blue Men dies.

Reflection:

This project was tedious to build and took quite some time to determine which functions would need to be deemed pure virtual functions and which members would need to be protected so I could call them throughout the program. For example I had my isDead() function as a virtual function until I had to use it within the Harry Potter sub-class to determine if his special ability Hogwart's would kick in or not. Thus, I had to change it (isDead()) to a pure virtual function. I also ran into issues with my random events. I made the mistake of stating srand((unsigned)time(0)); more than once in my program and it was causing issues with the die rolls. For example, the characters with the same number of die would roll the same numbers in either the attackRoll() or defenseRoll() function. Thus I had to go back and remove the srands in some areas and learned I only need to list it once in my program. I never realized this as I did not have an issue with it in my last program, however this program had some similarities within the sub-classes which caused the hiccups. I also struggled a bit determining how a character's special ability would be displayed as it could not be determined by an outside function. As I noted earlier, I had to make one function (isDead()) a pure virtual function because of this. Thus, I decided to determine each character's special ability in their virtual attackRoll() or defenseRoll() function. It made sense here as these functions were returning values and thus I could manipulate the return value based on certain circumstances performed each round. Most importantly, it was easier for me to track and build in this manner instead of creating additional virtual functions for each character's sub-class.