Project 4 Design + Reflection

Program Design

Problem

In this project, we will develop the game from project 3 to run a tournament between 2 teams of fighters.

Input Validation Design:

I will recycle my InputValidation class used in Project 1. It has an overloaded function called getInt(). I chose to make InputValidation its own class so that it could be used in future programs. getInt() receives input from the user, converts it into a stringstream, which then checks to see if there is anything beyond an integer that is left over. If there is, it makes the user try to input an integer again.
getInt(int min, int max) serves the same function as getInt() but also runs another validation checking input to see if it lies between min and max. getInt(min, max) will be useful for specifying minimum and maximum values for the following inputs:

- Menu options for starting or exiting the game at the end of each game
- Menu options for selecting each character for Team A and Team B
- Menu options for continuing or exiting the game at the end of each game

Menu Requirements:

At the start of the game, the menu should display 2 choices for user:

1. Play
2. Exit

If user chooses to play the menu should prompt user for number of fighters for both team, and ask for the type of character each fighter is, and the name for each fighter. From this point on the game does not require any more user interactions until the tournament is over.

During the tournament, the game will output the information about each battle in the tournament. For each fight between 2 teams, the game should display the type of character and the name of 2 fighters, and which won the combat.

Ex: Round 1: Team A Blue Man No.1 vs. Team B Harry Potter No.1, Harry Potter No.1 Won!

After the tournament is finished, the game should display the final score for each team, and then, depending on the final score, display the winner of the tournament, or a tie. The scoring system is your design decision, for example, winner +2pts, loser -1pts. Afterwards, provide the user a choice to whether display the content of the loser pile. If the user chooses yes, print them out from top to bottom (fighters who is defeated in order of last to first).

After the results are displayed, the menu should provide 2 options for user to choose:

1. Play again

2. Exit

If the user chooses to play again, the who process of choosing fighters and the tournament restarts. Otherwise, the game exists.

Menu Design:

I will recycle my Menu class used in Project 1 to display menu options, get input from the user, and store that input as a user choice variable. I chose to make Menu its own class so that it could be used in future programs. It references my Input Validation class to ensure the user is entering a valid input from the menu. There are functions to add an option to the menu, print the menu options, ask the user for his/her choice, and return the user choice.

The main menu will present 5 options for each character: Vampire, Barbarian, Blue Men, Medusa, and Harry Potter.

There will be a 2-option menu as specified to give the options to start or exit the game at the beginning of the game.

There will also be a 2-option menu as specified to give the options to continue or exit the game after each combat.

Die Design:

Die class is recycled from Lab 3. Die class will have a member integer variable called numSides in order to keep track of the number of sides of each die object. It will also have 2 member functions called getSides() and getRandomInt(). getSides() will return numSides for use in the getRandomInt() function. getRandomInt() returns a random integer between 1 and the number of sides that the die has in order to represent the roll of a die.

This class will be used to determine attack and defense values for the below Character types.

Character Classes Requirements:

The requirement should be the same to project 3 (requirements listed below), where you should use polymorphism, and all characters should inherit from the Character class. Characters class should be an abstract class.

To hold all types of characters in the containers, each object should be instantiated and their pointers should be put into the lineup, so the containers should contain only Character pointers. If you are uncertain how to implement the lineup with Character pointers and using polymorphisms, ask questions on Piazza or go to one of the office hours.

Kevin Harvell
Project 4 - Design + Reflection

Characters

Our game universe contains Vampire, Barbarian, Blue Men, Medusa, and Harry Potter. Each character has attributes of **attack, defense, armor, and strength points**.

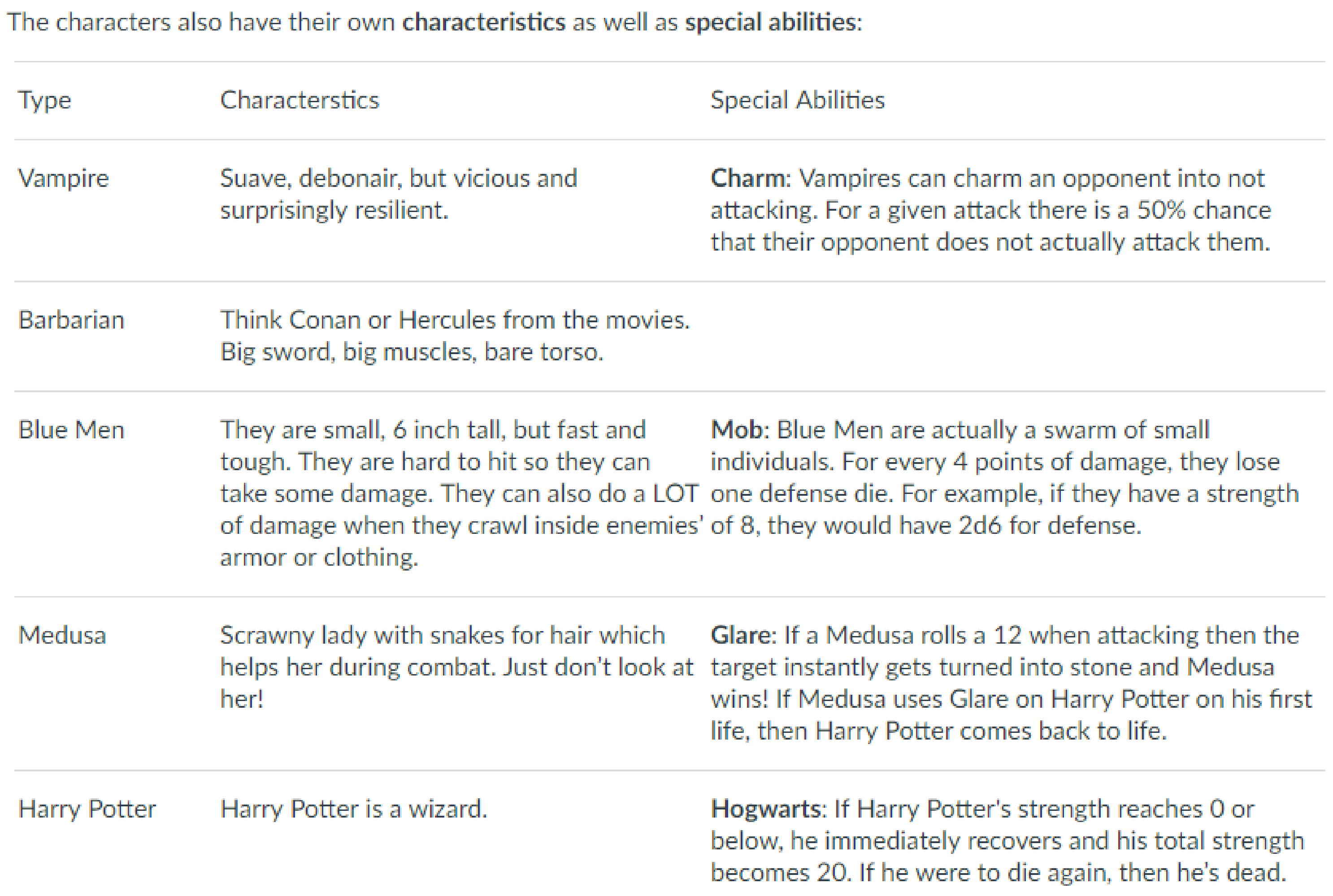
The table containing the attributes data is shown below.

Note: “1d12” means rolling one 12-sided die, and “2d6” means rolling 2 6-sided dice, etc.

Type	Attack	Defense	Armor	Strength Points
Vampire[1]	1d12	1d6* Charm	1	18
Barbarian[2]	2d6	2d6	0	12
Blue Men[3]	2d10	3d6	3	12 *Mob
Medusa[4]	2d6* Glare	1d6	3	8
Harry Potter[5]	2d6	2d6	0	10/20*Hogwarts

Note:

1. If Medusa uses “glare” on Harry Potter on his first life, then Harry Potter comes back to life after using “hogwarts”.
2. If the Vampire’s “charm” ability activates when Medusa uses “glare”, the Vampire’s charm trumps Medusa’s glare.
3. The sample characters are unbalanced intentionally. This will help you in debugging your program! Some will win a lot, while others won’t.



The characters also have their own **characteristics** as well as **special abilities**:

Type	Characterstics	Special Abilities
Vampire	Suave, debonair, but vicious and surprisingly resilient.	**Charm**: Vampires can charm an opponent into not attacking. For a given attack there is a 50% chance that their opponent does not actually attack them.
Barbarian	Think Conan or Hercules from the movies. Big sword, big muscles, bare torso.	
Blue Men	They are small, 6 inch tall, but fast and tough. They are hard to hit so they can take some damage. They can also do a LOT of damage when they crawl inside enemies' armor or clothing.	**Mob**: Blue Men are actually a swarm of small individuals. For every 4 points of damage, they lose one defense die. For example, if they have a strength of 8, they would have 2d6 for defense.
Medusa	Scrawny lady with snakes for hair which helps her during combat. Just don't look at her!	**Glare**: If a Medusa rolls a 12 when attacking then the target instantly gets turned into stone and Medusa wins! If Medusa uses Glare on Harry Potter on his first life, then Harry Potter comes back to life.
Harry Potter	Harry Potter is a wizard.	**Hogwarts**: If Harry Potter's strength reaches 0 or below, he immediately recovers and his total strength becomes 20. If he were to die again, then he's dead.

Character Design:

The Character is an abstract base class. All the characters have their own subclass that inherits from the Character class. The Character class will have variables common to all of the sub-classes: m_type, m_strength, m_attackVal, m_defenseVal, and m_damage. It will also have a public enum CharacterType variable representing each sub-class for increased readability in the menu code for selecting a character for player 1 and player 2. The Character class will have functions common to all the sub-classes: getType, getArmor, getStrength, getAttack, getDefense, takeDamageFrom, attack, and defense. The attack and defense functions are pure virtual functions that will be defined by the sub-classes because they all differ in the possibilities of attack (types and number of dice). takeDamageFrom will also be a virtual function to account for Harry Potter's revival.

Barbarian, Blue Men, Harry Potter, Medusa, and Vampire Design:

Each of these classes will work roughly the same. Each class will define its member variables via the constructor (i.e. m_type, m_strength, m_attackVal, m_defenseVal, and m_damage). Each class' attack and defend functions will specify the type of dice and number of rolls as specified for the Character type.

The major differences in the sub-classes will be found in Vampire, Blue Men, Medusa, and Harry Potter. In Medusa's attack function, it will check if the rolls add up to 12, and then return an attack value of -1. This value of -1 can be checked for via the takeDamageFrom function in order to provide lethal damage to whatever adversary Medusa is up against.
In Vampire's class, there will be a function called isCharmed that flips a 2-sided die to return a true or false value. This true/false value indicates whether the "Charm" ability is used or not. In Vampire's defense function, it will check to see if the Vampire is using "Charm" via the isCharmed function. If it is charmed, it will set Vampire's defense to 100,000, thus nullifying any damage received. If it is not using "Charm", it will roll the defense die like normal.
In Blue Men's defense function, it will check to see how many rolls Blue Men get based on Blue Men's strength points. Strength points will be divided by 4.0 and the result will be rounded up no matter what. This results in a defense die being lost for every 4 points of damage dealt to Blue Men.
In Harry Potter class, the takeDamageFrom function will be overridden to check if Harry Potter has an extra life. If Harry Potter dies but has an extra life, Harry Potter's strength will be set to 20 and lives will be reduced by 1.

Containers Requirements:

For this game, you need to make your own containers for the team lineups and the loser containers. It is required to use stack-like or queue-like linked-list structures for the container.

Important: Please make your own containers instead of the STL containers.

Containers Design:

Containers for this program will be simple to implement as much of the work has been accomplished from Lab 6. I plan to use Lab 6's doubly linked lists that can work as either a stack or queue due to the removeFront, removeTail, addToHead, and addToTail functions. The main alterations to Lab 6 will be to change the information stored from integers to Character pointers. I will also borrow the Lab 7 isEmpty function in order to determine if the lists are empty.

Gameplay Requirements:

This game is a one-user-two-player game, so one user should be able to play setup fighters for both teams. For simplicity, we will call the order of fighters in each team "lineup", like a batting order in baseball or softball.

The flow of gameplay is as follows:

At the start of the tournament, it should prompt the user to enter the number of fighters for both team. After that, for each fighter, the game should ask the user to choose the type of character, and enter a name for the fighter. For example: assume the user chooses Harry Potter for player No.1's team lineup, the user can name the Harry Potter character "Larry Snotter" or just "Harry Potter No.1". The game should allow more than 1 of the same character for team lineups. For example, Team A has 3 members: 2 barbarians and 1 Harry Potter. The order of the lineup should be the same as the order user entered.

After the user supplied the lineup of fighters in order, for Team No.1, and Team No.2, the tournament starts. In the tournament, the fighter at the head/front of each lineup will battle, in the same way they fight in project 3. The winner of each fight gets put at the back of their lineup, while loser goes to the top of the loser container (There should only be one containers to hold defeated fighters on both team).

If a fighter won, it may have taken damage, so the game should restore some percentage of the damage they took when they get back in line.

The lineup order cannot be changed, meaning that the game cannot modify the order of the lineup except when putting winner at the back of lineup and loser at the top of the loser container.

The tournament finishes when one of the team does not have any more fighters to fight. When the game ends, it should print the result of the game, and allow user to have the choice to display the contents of the loser pile. More information is shown below in the menu section

Game Class Design:

The game class will be the engine for most of the gameplay functionality. Its central function runGame will organize the processes. Starting with a setFighters function, which borrows from Project 3's code to create fighters for each team. setFighters will be very similar to Project 3, except each created character goes into a linked list for each respective team.

runGame will then display the lineups for each team. It will call a fight function that has the fighters faceoff against each other one pair at a time. A lot of the fighting code can be recycled from Project 3. Based on each fight, the winner will go to the end of the linked list, and the loser will go to the top of the loser stack. In addition, there will be a scoring system where the winner gains 2 points, and the loser loses 1 point. I plan to keep some of my code from Project 3, but only make it accessible with a debug variable set to true. This code displays many details of each actual combat which are not required to show for this project, but will be useful to look at when troubleshooting.

Finally, runGame will call a displayResults function which determines which team is the winner based on the above scoring system.

Main Design:

main runs the start menu. If the user wants to play, it creates a game object and runs the game using runGame. After the game is finished, there is another menu called continueMenu which allows the user to play again.

Reflection

I approached this program methodically starting with importing the easy stuff: my already written Character, List, Die, Menu and InputValidation classes. I find that starting out with these small victories gives me confidence when the project looks daunting at first.

There was not much new code written for this assignment. It was more a matter of taking existing code and modifying it to suit my needs. Everything went easily. I just had to tweak the data type in my List class from Lab 6 to Character pointers. I tested out whether this worked, and it did without a hitch.

My only struggle during this project was deallocating the Character pointers stored in the lists. I spent a good hour thinking through the logic of what was happening as I continued to get segmentation faults. My mistake was thinking that the List class deallocated the Character pointers, but it only deallocated the node pointers for the list. I simply added a delete character line to the node deallocation code, and it solved the problem.

Because this project utilized so many aspects from projects we have already completed, it was seemingly easy as it was familiar territory. I really appreciated the break in difficulty because I have been completely stressed throughout this class thus far. I actually managed to finish this project 6 days early.

Test Tables

For InputValidation Class – This class was already tested in Project 1

Test Case	Input Values	Driver Functions	Expected Outcomes	Observed Outcomes
input too low or high	input < 2 input > 200	getInt(2, 200)	You must select an integer equal to or between 2 and 200.	You must select an integer equal to or between 2 and 200.
input in correct range	2 < input < 200	getInt(2, 200)	Returns input	Returns input
input extreme low	input = 2	getInt(2, 200)	Returns 2	Returns 2
input extreme high	input = 200	getInt(2, 200)	Returns 200	Returns 200
input not an integer	input = a	getInt(2, 200)	You must enter an integer value.	You must enter an integer value.
input not an integer	input = 123abc	getInt(2, 200)	You must enter an integer value.	You must enter an integer value.
input not an integer	input = enter key	getInt(2, 200)	You must enter an integer value.	You must enter an integer value.
input not an integer	input = asdf 123	getInt(2, 200)	You must enter an integer value.	You must enter an integer value.

For Menu Class - This class was already tested in Project 1

Test Case	Input Values	Driver Functions	Expected Outcomes	Observed Outcomes
input option string	"Start Langton\'s Ant simulation"	addOption(string) printOptions() main()	1. Start Langton's Ant simulation	1. Start Langton's Ant simulation
input option string	"Quit"	addOption(string) printOptions() main()	1. Start Langton's Ant simulation 2. Quit	1. Start Langton's Ant simulation 2. Quit
input too low or high	input < 1 input > 2	getInt(1, 2) promptUserChoice() main()	You must select an integer equal to or between 1 and 2.	You must select an integer equal to or between 1 and 2.
input not an integer	input = a	getInt(1, 2) promptUserChoice() main()	You must enter an integer value.	You must enter an integer value.
input valid	input = 1 input = 2	getInt(1, 2) promptUserChoice() getUserChoice();	returns 1 returns 2	returns 1 returns 2

Die Class – This class was already tested in Lab 3

Test Case	Input Values	Driver Functions	Expected Outcomes	Observed Outcomes
roll random number in corresponding sided die	numSides = 2 numSides = 6 numSides = 20	getRandomInt()	Outputs values from 1 to 2 Outputs values from 1 to 6 Outputs values from 1 to 20	Outputs values from 1 to 2 Outputs values from 1 to 6 Outputs values from 1 to 20

Character/Vampire/Barbarian/Blue Men/Medusa/Harry Potter Classes

Test Case	Input Values	Driver Functions	Expected Outcomes	Observed Outcomes
initiates each Character/ Vampire/ Barbarian/ Blue Men/ Medusa/ Harry Potter object with correct values	See above chart for exact values on the following variables: m_type m_armor m_strength	Character() Vampire() Barbarian() Blue Men() Medusa() Harry Potter()	Correct values are stored determined by specifications of class type	Correct values are stored determined by specifications of class type
each Character/ Vampire/ Barbarian/ Blue Men/ Medusa/ Harry Potter object attacks with the correct values	See above chart for exact values of each class's attack dice rolls	attack() getRandomInt()	Correct attack outputs based on specifications class type attack dice rolls.	Correct attack outputs based on specifications class type attack dice rolls.
each Character/ Vampire/ Barbarian/ Blue Men/ Medusa/ Harry Potter object defends with the correct values	See above chart for exact values of each class's defense dice rolls	defense() getRandomInt()	Correct defense outputs based on specifications class type defense dice rolls.	Correct defense outputs based on specifications class type defense dice rolls.
each character takes the appropriate amount of damage during attack/defense combat	See above chart for exact values on the following variables/ functions: m_type m_armor m_strength attack() defense()	Vampire() Barbarian() Blue Men() Medusa() Harry Potter() attack() defense() takeDamageFrom()	damage = attack() – defense() – m_armor	damage = attack() – defense() – m_armor
if damage taken is	See above chart for exact values	Vampire() Barbarian()	if damage taken is negative, the	if damage taken is negative, the

negative, the character does not gain health	on the following variables/ functions: m_type m_armor m_strength attack() defense()	Blue Men() Medusa() Harry Potter() attack() defense()	character does not gain health	character does not gain health
Vampire's Charm ability works about 50% of the time and successfully blocks damage	See above chart for exact values on the following variables/ functions: m_type m_armor m_strength attack() defense()	Vampire() Barbarian() Blue Men() Medusa() Harry Potter() attack() defense() isCharmed()	Vampire's Charm ability works about 50% of the time and successfully blocks damage	Vampire's Charm ability works about 50% of the time and successfully blocks damage
Vampire's Charm ability trumps Medusa's Glare ability	See above chart for exact values on the following variables/ functions: m_type m_armor m_strength attack() defense()	Vampire() Medusa() attack() defense() isCharmed()	Vampire's Charm ability trumps Medusa's Glare ability	Vampire's Charm ability trumps Medusa's Glare ability
Blue Men lose a defense die for every 4 points of damage	See above chart for exact values on the following variables/ functions: m_type m_armor m_strength attack() defense()	Vampire() Barbarian() Blue Men() Medusa() Harry Potter() attack() defense()	Blue Men lose a defense die for every 4 points of damage. Defense outputs only show within an acceptable range.	Blue Men lose a defense die for every 4 points of damage. Defense outputs only show within an acceptable range.
Medusa's Glare ability activates when she rolls a 12. Glare instantly kills opponent.	See above chart for exact values on the following variables/ functions: m_type m_armor m_strength	Vampire() Barbarian() Blue Men() Medusa() Harry Potter() attack() defense()	Medusa's Glare ability activates when she rolls a 12. Glare instantly kills opponent.	Medusa's Glare ability activates when she rolls a 12. Glare instantly kills opponent.

	attack() defense()			
Harry Potter's Hogwarts ability allows him to revive after death with a new strength of 20	See above chart for exact values on the following variables/ functions: m_type m_armor m_strength attack() defense()	Vampire() Barbarian() Blue Men() Medusa() Harry Potter() attack() defense()	Harry Potter's Hogwarts ability allows him to revive after death with a new strength of 20	Harry Potter's Hogwarts ability allows him to revive after death with a new strength of 20
Harry Potter dies for real after being revived once.	See above chart for exact values on the following variables/ functions: m_type m_armor m_strength attack() defense()	Vampire() Barbarian() Blue Men() Medusa() Harry Potter() attack() defense()	Harry Potter dies for real after being revived once.	Harry Potter dies for real after being revived once.

For Game.cpp

Test Case	Input Values	Driver Functions	Expected Outcomes	Observed Outcomes
Correct Character type created based on character selection input	1 <= p1Character <= 5 1 <= p2Character <= 5	Character() Vampire() Barbarian() Blue Men() Medusa() Harry Potter() Menu() getUserChoice() setFighters() createCharacter()	Correct Character type created based on character selection input	Correct Character type created based on character selection input
Combat displays mathematically correct output	Varies	getStrength() getType() getArmor() attack() defense() takeDamageFrom() fight()	Combat displays mathematically correct output over several rounds	Combat displays mathematically correct output over several rounds

Combat continues until one character's strength reaches 0 (except Harry Potter)	Varies	getStrength() getType() getArmor() attack() defense() takeDamageFrom() fight()	Combat continues until one character's strength reaches 0 (except Harry Potter)	Combat continues until one character's strength reaches 0 (except Harry Potter)
Game gives option to continue after combat	Varies	Menu() getUserChoice() promptUserChoice()	Game gives option to continue after combat	Game gives option to continue after combat
Game displays correct results after each combat	Varies	fight()	Correct winner and loser are displayed	Correct winner and loser are displayed
After each combat, lists update correctly	Varies	fight()	Winner goes to end of team lineup list, loser goes to losers list	Winner goes to end of team lineup list, loser goes to losers list
Final tournament results displayed	Varies	fight() displayResults()	Based on previous round winners/losers, final scores are accurate and correct winner is determined.	Based on previous round winners/losers, final scores are accurate and correct winner is determined.
Can replay game after tournament (and new game works)	continueMenu choice = 1	continueMenu	Can replay game after tournament (and new game works)	Can replay game after tournament (and new game works)