Justin Tromp
Project 3 – Fantasy Combat Game
CS 162 – Winter 2019

Fantasy Combat Game
Character Class (Parent class to all characters) – Abstract Class (Has at least 1 pure virtual function) - .cpp & .hpp
Variables:
armor – Integer variable for armor rating for character
strength – Integer variable for strength rating for character
Functions:
Default Constructor – Set all character variables to 0 with set functions
Constructor (int inputArmor, int inputStrength) – Set all character variables to input passed as parameters with set functions.
virtual int attack() – Rolls die or dice for character to perform attack based on characters attack die type values.
virtual void defend() - Rolls die or dice for character to perform defense from attack based on characters defend die type values.
int randomInt(int, int) – returns random integer variable within two ranges passed as parameters to function. For example: minInt through maxInt parameters.
int getArmor() – Returns armor rating for character object
void setArmor() – Sets armor rating for character object
int getStrength() – Returns strength rating for character object
void setStrength() - Sets strength rating for character object

Vampire Class (Child class to character class) - .cpp & .hpp
Variables:
(variables from base class)
Functions:
Default Constructor – Set all character variables to -1 through inheritance
Constructor (int inputArmor, int inputStrength) – Set all character variables through inheritence
Overridden defense function - Rolls die or dice for character to perform defense from attack based on characters defend die type values. Includes charm ability.

Barbarian Class (Child class to character class) - .cpp & .hpp
Variables:
(variables from base class)
Functions:
Constructor – Passes values for class to parent class Character constructor for implementation to respective variables.

Blue Men Class (Child class to character class) - .cpp & .hpp
Variables:
(variables from base class)
Functions:
Constructor – Passes values for class to parent class Character constructor for implementation to respective variables.
Overridden Defense – Override defense function from Character class to implement mob ability.

Medusa Class (Child class to character class) - .cpp & .hpp
Variables:
(variables from base class)
Functions:
Constructor – Passes values for class to parent class Character constructor for implementation to respective variables.
Overridden attack function from Character class – Implements same functionality as Character class, but with the added ability of stare ability. Stare will set damage return to 99 in order to trigger a reaction in a defending character's defense function.

Harry Potter Class (Child class to character class) - .cpp & .hpp
Variables:
(variables from base class)
harryPotterLives – Integer variable set to 2 at object creation allows Hogwarts ability to be implemented effectively giving Harry Potter 2 lives.
Functions:
Constructor – Passes values for class to parent class Character constructor for implementation to respective variables.
Overridden Defense – Override defense function from Character class to implement Hogwarts ability.

Game Class - .cpp & .hpp
Variables:
Functions:
Default Constructor – No values to set, leave empty.
Destructor – Deallocate memory assigned to vector during character selection.
gameRun() – Runs the actual fantasy game. Starts by bringing up menu and brings together all functionality of program.
Void printTurn() – Display turn/round results to user by printing to screen.
Results to print:
- Attacker type (strength point)
- Defender type (armor/strength point)
- Attackers attack dice roll value
- Defenders defend dice roll value
- Total damage inflicted (calculated)

- Defenders updated strength point amount after subtracting damage

void addCharacter(Character*) – Add character to vector based on selection from user. Takes Character pointer as parameter.

Menu Functions - .cpp & .hpp

Overview: Menu functions will prompt user for input and use input validation function to validate for proper selection. Once a proper selection is made, return value to calling function. In my program design, the calling function will be the gamePlay() function in the game class.

Functions:

int startMenu() – See if user wants to start the game. If user selects 1 to start game, return integer 1. If user selects 2 to not start game, return integer 2.

void playMenu() – Displays five characters by name to user and prompt the user to select two of the characters to fight. The selection can be two different characters or two of the same character types.

std::string characterMenu() – After play menu is displayed to user, display characterMenu for each selection. Character menu will prompt user for a name selection and validate selection. Once validated, function will return string of validated name. I will be comparing all strings in capitalized form for greater accessibility in order to make the program less finicky/strict while still ensuring proper selections.

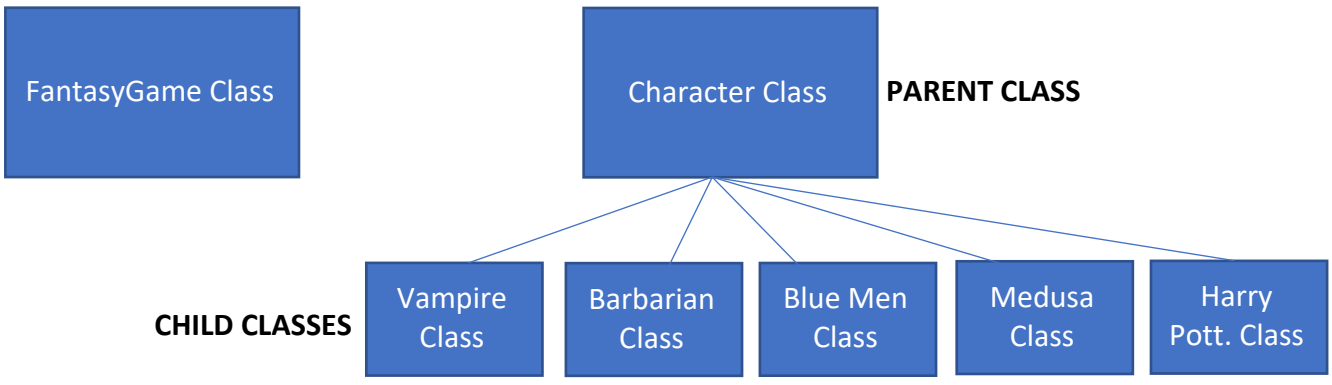
int endMenu() – See if user wants to keep playing the game. If user selects 1 to play again, return integer 1. If user selects 2 to exit, return integer 2. Integer selection is validated by validateRangeInt function.

Input Validation Functions - .cpp & .hpp

Functions:

int validateRangeInt (int min, int max) – Requests user input and places in string value for validation. If values are entered outside of a positive integer range, re-prompt user. Integer range is provided by parameter to the function. Once integer is validated, return integer to calling function.

Class Hierarchy



MEMORY LEAK TESTS
Select 2 (exit) at start: No memory leaks
Select 1 and play 1 round (Barb v Barb): No memory leaks
Select 1 and play 2+ rounds (Barb v Barb): No memory leaks
Select 1 and play 10+ rounds (Various characters): No memory leaks

INPUT VALIDATION TEST

Test Case	Input Value	Driver Functions	Expected Outcomes	Observed Outcome Initial
Start Menu	0	validateIntRange	Re-prompt/err	Re-prompt/err
	3		Re-prompt/err	Re-prompt/err
	12		Re-prompt/err	Re-prompt/err
	21		Re-prompt/err	Re-prompt/err
	Enter		Re-prompt/err	Re-prompt/err
	Space		Re-prompt/err	Re-prompt/err
	1a		Re-prompt/err	Re-prompt/err
	a1		Re-prompt/err	Re-prompt/err
	1.5		Re-prompt/err	Re-prompt/err
	1 1		Re-prompt/err	Re-prompt/err
	1		Start Game	Start Game – Char. Selection
	2		Quit Program	Quit Program
End Menu	0	validateIntRange	Re-prompt/err	Re-prompt/err
	3		Re-prompt/err	Re-prompt/err
	12		Re-prompt/err	Re-prompt/err
	21		Re-prompt/err	Re-prompt/err
	Enter		Re-prompt/err	Re-prompt/err

	Space		Re-prompt/err	Re-prompt/err
	1a		Re-prompt/err	Re-prompt/err
	a1		Re-prompt/err	Re-prompt/err
	1.5		Re-prompt/err	Re-prompt/err
	1 1		Re-prompt/err	Re-prompt/err
	1		Start Game	Start Game Again – Char. Selection
	2		Quit Game	Quit Program

CHARACTER SELECTION TEST

- All characters can be chosen by typing name.
- Character types can be selected in varying caps due to "to_upper" function allowing for easier input.
- Anything other than a character's name is rejected and requests new input from user.

GAMEPLAY TEST (ONLY TEST EACH FUNCTION, NO NEED TO TEST EVERY CHARACTER COMBINATION) – TOP ROW OF EACH TABLE IS ROUND 1 INCREMENTS ACCORDINGLY AS ATTACKER/DEFENDER ALTERNATE. LAST ROW IS FINAL ROUND IF SPECIFIED TO TEST END.

Barbarian vs. Barbarian # 1 –

Attacker	Attack Roll	Damage Inflicted	Defender	Defense Roll	Armor	Strength (Start/After)	Event Info
Barbarian	6	Expected: 0 Actual: 0	Barbarian	11	0	12/12	
Barbarian	8	Expected: 0 Actual: 0	Barbarian	9	0	12/12	
Barbarian	5	Expected: 0 Actual: 0	Barbarian	11	0	12/12	
Barbarian	10	Expected: 7 Actual:	Barbarian	3	0	12/12	**ERROR**

Changes Made: Strength was being displayed before subtracting value.

Barbarian vs. Barbarian # 2 – Check!

Attacker	Attack Roll	Damage Inflicted	Defender	Defense Roll	Armor	Strength (Start/After)	Event Info
Barbarian1	10	Expected: 2 Actual: 2	Barbarian2	8	0	12/10	
Barbarian2	8	Expected: 0 Actual: 0	Barbarian1	12	0	12/12	
Barbarian1	6	Expected: 4 Actual: 4	Barbarian2	2	0	10/6	
Barbarian2	7	Expected: 0 Actual: 0	Barbarian1	10	0	12/12	
FINAL ROUND 9: Barbarian2	10	Expected: 7 Actual: 7	Barbarian1	3	0	3/-4	**Barbarian Died**

Changes Made: Made design decision to change strength to 0 at death instead of displaying a negative strength value.

Barbarian vs. Blue Men – Check!

Attacker	Attack Roll	Damage Inflicted	Defender	Defense Roll	Armor	Strength (Start/After)	Event Info
Barbarian	12	Expected: 1 Actual: 1	Blue Men	8	3	12/11	
Blue Men	14	Expected: 6 Actual: 6	Barbarian	8	0	12/6	
Barbarian	10	Expected: 0 Actual: 0	Blue Men	13	0	11/11	
Blue Men	2	Expected: 0 Actual: 0	Barbarian	8	0	6/6	

Changes Made: Made design decision to output number of dice rolled for attack and defense.

Harry Potter vs. Blue Men (Test for Hogwarts Ability) – Check!

Attacker	Attack Roll	Damage Inflicted	Defender	Defense Roll	Armor	Strength (Start/After)	Event Info
Harry	8	Expected: 0 Actual:	Blue Men	7	3	12/12	
Blue Men	10	Expected: 6 Actual:	Harry	4	0	10/4	
Harry	9	Expected: 0 Actual:	Blue Men	12	3	12/12	
Blue Men	15	Expected: 8 Actual:	Harry	7	0	4/20	**Used Hogwarts Ability**
Final Round 20: Blue Men	19	Expected: 13 Actual: 13	Harry	6	0	1/0	**Died second time.**

Blue Men vs. Blue Men (Testing for Mob) – Initial test showed some poor wording during output. Stated that 2 blue men died on same round as death of all blue men.
Changes Made: Added if statement in mob ability function to check strength level. If strength is 0, do not state reduction of defense dice.
Blue Men vs. Blue Men (Testing for Mob) – Mob worked correctly and removed defense dice.

Vampire vs. Blue Men (Test for Vampire Charm – Modified Defense) – Check!

Attacker	Attack Roll	Damage Inflicted	Defender	Defense Roll	Armor	Strength (Start/After)	Event Info
Vampire	4	Expected: 0 Actual:	Blue Men	9	3	12/12	
Blue Men	2	Expected: 0 Actual: 0	Vampire				**Vampire Charmed**
Vampire	9	Expected: 0 Actual: 0	Blue Men	14	3	12/12	
Blue Men	18	Expected: 11 Actual: 11	Vampire	6	1	18/7	
Final Round 10: Blue Men	10	Expected: 5 Actual: 5	Vampire	4	1	1/0	**Vampire Died**

Vampire Charm – Charm worked correctly and allowed vampire to avoid attack approximately 50% of the time.

Vampire vs. Medusa (Test for Medusa Stare – Modified Attack) – Check!

Attacker	Attack Roll	Damage Inflicted	Defender	Defense Roll	Armor	Strength (Start/After)	Event Info
Vampire	7	Expected: 0 Actual: 0	Medusa	6	3	8/8	
Medusa	6	Expected: 0 Actual: 0	Vampire	6	1	18/18	
Vampire	4	Expected: 0 Actual: 0	Medusa	5	3	8/8	
Medusa	3	Expected: Charmed Actual: Charmed	Vampire	0	1	18/18	
Final Round 8: Medusa	12	Expected: Stone Actual: Stone	Vampire	N/A	1	14/0	**Vampire Died – Turned to stone**

Vampire Charm – Overrides Medusa's glare as expected in a second test.
Medusa Glare – Works on vampire. Must test on a base character defense type.
Medusa Glare vs. Harry Potter – First time does not kill Harry Potter.

```
Medusa rolled 2 dice for an attack of 12.
Medusa attempted to use glare on opponent.

Harry Potter is turned to stone.
Harry Potter took a deadly blow but he is back to life at double
strength!
```

Strength is back to 20. Second time glare is used on Harry Potter, he dies for good.
Base character defense function works with medusa as well as Harry Potter Hogwarts ability.

Harry Potter vs Harry Potter – Hogwarts ability works as expected.
Harry Potter vs Blue Men – Hogwarts ability and dice reduction works as expected.

REFLECTION
CHANGES MADE
Game Class
playMenu() moved to game class and is a combination of playMenu and characterMenu in design.

Character Class
Added variables for: charType, attackDieSize, defenseDieSize, numAttackDice, and numDefenseDice.
Character constructor takes additional parameters of attack die size, defense die size, number of attack dice, and number of defense dice.
Added additional functions to get and set dice values such as number of sides.

Added a get function for character type to return string value for character type.
Added function to print defender information for user.
Added virtual take damage function that takes integer value as parameter for damage inflicted and reduced strength based on that value.
Realized I was not using pure virtual functions in parent class design, meaning it was technically not an abstract class. This required deviation from design. Made attack and defense abstract with default functions that can be called by child classes that do not have special abilities.

All Child Classes
Added static constant values for the attributes for each character type. These values are all passed directly to the Character constructor through their constructor and can be changed at any time easily. Character type is set to the class type name (such as "Vampire").

Blue Men Class
Added mobAbility function to implement mob ability through an overridden virtual take damage function. This allowed me to implement mob without overriding the much larger defense function for the entire Blue Men class. I used this same method for the Harry Potter Hogwarts ability. The overridden takeDamage function in this class implements the mobAbility function.

Harry Potter Class
Added hogwartsAbility function to implement Hogwarts ability through an overridden virtual take damage function. This allowed me to implement mob without overriding the much larger defense function for the entire Harry Potter class. If Harry Potter has 2 lives left, reset strength to 20. If Harry does not have 2 lives, then do nothing. Overridden takeDamage function implements the hogwartsAbility function.

Design/Gameplay Decisions:
Initially my game was setup with the following output, which made the most sense to me:

Medusa rolled 2 dice for an attack of 12.
Medusa attempted to use glare on opponent.

Harry Potter is turned to stone.
Harry Potter took a deadly blow but he is back to life at double strength!"

Start next round

Attacker: Harry Potter

Etc…

This made the most sense since there really are no defense rolls for glare (except charm ability, which is not a defense roll). After talking with a TA on Piazza however, I opted to include ending

strength values for every attack round regardless of the presence of defense rolls. I did not include defense rolls if there was no roll, but I thought for grading and debugging purposes it would be nice to at show that strength did not change when an ability such as charm occurred.

I chose to keep armor values constant for character types that do have armor. This was a design decision based on the idea that armor does not get destroyed in a short period of "brawl" between two characters.
My final design decision was to create static constant values for each class type. These constants were then passed as parameters to class constructors to create the class object with the values applicable to the class type. This design decision was made to allow for increased code readability and allow for easy changes to character type values if necessary, in future designs.

What went wrong?
Towards the end of development, I realized I was not using pure virtual functions in my parent class design, meaning it was technically not an abstract class. This required deviation from my original design. I chose to keep a design similar to my original, as I still felt it was the most efficient way of doing things. By adding pure virtual functions for both attack and defense in Character class, I needed to maintain the abstract class template throughout all of the child classes instead of simply using or overriding the two functions when necessary. This meant that I had to create an attack and defense function in every child class branching from the abstract parent class Character. In my Character class I chose to maintain a default function procedure that could be called by classes which do not need different defense or attack function procedures. By doing so, I was able to save a ton of time on coding for every character class type while still maintaining abstraction. In child classes that required an overridden attack or defense function such as Vampire or Medusa, I created a modified attack or defense for those character types. For any child classes that could use the base class default functionality, I created a virtual attack and/or virtual defense function that called the Character classes attack and/or defense function for use.

In addition to the previous issue/modification, I also ran into issues with restarting the actual combat simulation without deleting the game object. Although I could have deleted the object by loop in main, I decided to maintain all operations within the gameRun function of FantasyGame class. In order to reset character between battles, I added a custom reset function called removeCharacters that acted as a sort of destructor for all dynamically allocated objects in the game vector.

What would I do next time?
If I was to make the same program again, I would probably split the defense and attack functions into smaller procedures. Towards the end the functions were getting a little bit on the larger end and I feel like some of the printouts could have been modularized into smaller pieces. Looking back, I was worried about some of the statements in the assignment that stated it was unacceptable to create outside functions to assist in certain aspects of attack and defense. However, I think that was less about what was inside attack and defense and more

related to what was inside each character class type themselves. Overall though, I am happy with the way the program turned out and I feel like it was created efficiently while allowing room for future modifications and additions as we enter into the next programming assignment.