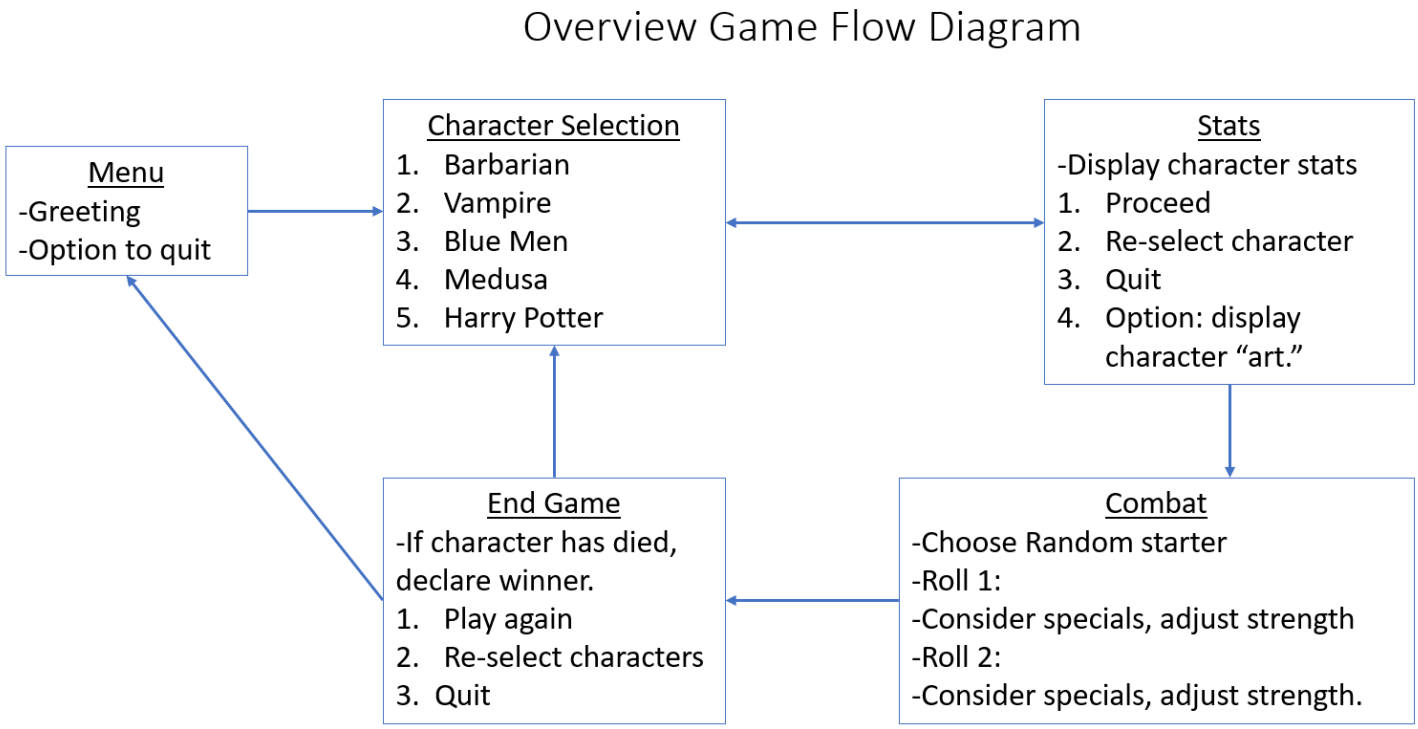
Andrew Clos
CS-162 - 400
Project 3 Design, Class Hierarchy, Test Table, and Reflection of Fantasy Combat

Initial Design:



-Menu functionality will reside in the main method, and based on the user's character choices, new characters will be created.

-Although all defense dice rolls use six sided dice, will implement this as if it could be variable in case future classes are able to roll higher dice.

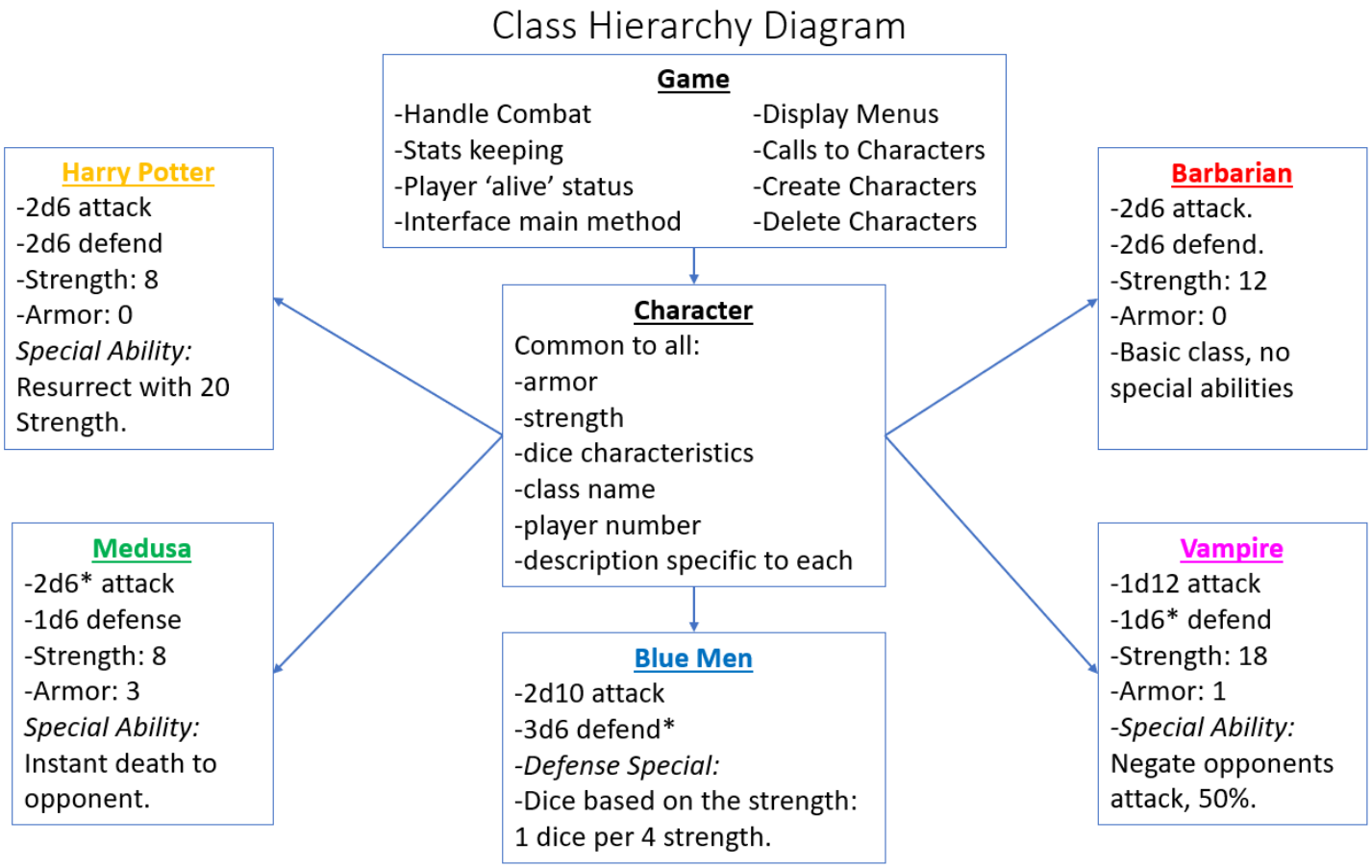
-Have Character names be colored distinctly throughout the game to add some variation and make it easier for the user to see which classes are involved in a given round of combat.

-Special ability effects could use a different color of text for damage done or mitigated.

-Collect combat statistics and display them at the end of the game. This will make debugging easier and adds some interesting data for the players/reviewers.

Andrew Clos
CS-162 - 400
Project 3 Design, Class Hierarchy, Test Table, and Reflection of Fantasy Combat

Class Hierarchy:



Changes in Design:

I had initially considered adding a menu option to simulate many attacks between different Character classes, in order to do testing and debugging. However, since this project does not have us attempting to balance the classes from a combat fairness perspective, I chose instead to implement a detailed stats collection vector that is displayed at the end of the game. During testing, it was easy enough to Addition of menu choice to simulate X number of attack rounds between chosen characters and compile statistics. This stat output proved to be extremely useful in testing the program and debugging!

I was initially going have Harry Potter automatically give himself 20 strength in the event of his first death, but it became easier to just give all Characters a "spare lives" variable. For this game, all characters have zero spare lives aside from Harry Potter, who has 1. When his spare life is triggered, his strength becomes 20. I liked this approach since it would allow for easier addition of future classes that may have spare lives, or modifications to other existing classes spare lives. For example, maybe it would be fun to give each character +3 lives to have longer combat – this is more easily done if a Character contains a "spare life" member variable.

Major Classes / Methods:

Game: Contains character selection/creation/deletion, menus, combat, 'alive' checking, and statistics collection. Keeps the game running.

Andrew Clos
CS-162 - 400
Project 3 Design, Class Hierarchy, Test Table, and Reflection of Fantasy Combat

Character: Abstract class that contains mostly get and set methods as well as private member variables that all characters must have (str, armor, lives, dice characteristics, etc.).

Barbarian/Medusa/Vampire/BlueMen/HarryPotter: These derived classes over write several virtual functions from the abstract class with information and characteristics specific to each class. One of these classes is created for each player of the game.

Main: Runs the game using while loops and making calls to the Game function for menus, combat, and stats printing.

Test Plan:

Test Category	Description of Test	Expected Results / Handling
Player dies	Make sure game ends immediately and no more attacks are carried out.	-No negative strength, no extra attack rounds. -No seg faults or memory leaks
Combat effectiveness (randomness)	Add significant number of hit points (strength) to each player in order for many rounds to run.	-Make sure random events like the Medusa Glare are happening at the correct frequency by manually counting their occurrence. -Make sure amount of damage done / defended per attack is correct based on the dice rolled (stats printing results analyzed).
Quitting	If zero or 1 characters are chosen and a quit is initiated.	-Program exits normally (immediately) -No seg faults attempting to delete a character that doesn't exist.
Input validation	1. SPACES handled (getline)	-I have finally implemented changes to input validation to use the getline function instead of cin. This should make the program properly handle spaces. I had to move/remove some of the cin.clear and cin.ignore features in the input validation functions, so this will need to be tested as well (see below)
Input validation: is an integer	1. Entering leading zeros, any other non 1-9 char in the first position or any non 0-9 in subsequent positions.	-Explain special case of leading 0 not being allowed with cout message. -Must make sure entry isn't being truncated. If user enters

	2. '2.1' float / double handling. 3. Adheres to size boundaries of integers. If limit is set too high or not at all.	'2.1,' we cannot assume they meant '2'!! -If not establishing an upper limit, warn user that their entry is above size of signed int (32,767), etc.
Random Function	1. Drastically increase hit points and run many iterations of every class combination. 2. Make sure the starting attacker is randomized properly (~50%)	-Inspecting the console combat logs as well as the end of game statistics print out will be essential here. (1/12 Medusa glare, correct damages given / mitigated, etc.)
Memory Leaks	1. Valgrind with -v options. 2. Test case of quitting right away	-Expect to see no memory leaks, since I am getting better about including proper delete statements. -Pay extra attention to the cases where a user chooses a class, but views statistics and chooses another (this should trigger two 'creates' and two 'deletes' for these temporary class instances. should -Should see it in valgrind and "vld.h" function for Visual Studio. Possible seg fault.

Reflection:

-For this assignment, I reworked input validation to use the getline function in order to handle spaces. This was long overdue, but I had previously confused myself a bit with overuse of cin.clear and cin.ignore features, so that when I initially used the getline function, I was having to press enter twice to accept the input. Since I was previously unfamiliar with getline, I thought this might have been normal (but annoying) operation, so I stayed with cin. However, after some research, I found that if I used cin.clear and cin.ignore properly (or not at all), the getline function would operate normally.

-I was also having a problem where my makefile would start the program immediately upon completion. After learning that this was not supposed to be happening, I did some investigating and commented out a command to debug the program that was causing the auto-run. I will turn this debug command on for my own testing on the FLIP server to properly test for memory leaks through various menus, but I will turn it off when it becomes time to submit projects.

-I initially thought I would have the dice rolling function be part of the Game class, but decided it was better for the character Classes to house this information. Currently, the roll function takes the number of dies to roll as well as the number of sides each dice has and returns a random integer based on those parameters. I have this function residing in each derived class instead of the base abstract class (even though the function is exactly the same for each derived class). I did this because it may be beneficial to change the roll function specific to each derived class if tweaks to the

Andrew Clos
CS-162 - 400
Project 3 Design, Class Hierarchy, Test Table, and Reflection of Fantasy Combat
game are desired. Currently the function cannot roll dice of unequal sides in a single call, but it would be easy to call the function multiple times back to back if this is desired in future versions of the game.

-The advice in the Project Specifications to start with the barbarian class was very good! This basic class was the easiest to implement and I could test most of the game features by pitting two barbarians against each other. Once I had the game working, copy and pasting the barbarian class to create the others was easy. Then, all that needed to be done was to implement their special attack or defend features.

-I decided to use console colors extensively to add some variety to the game. These console text-based games can get quite wordy as the rounds are cycled through and some people find this boring or even harder to read. So, I assigned a color to each class and used a brighter white text for attack and defend points as well as special ability text. For a console text game, I like the way this turned out and will use this in the future.