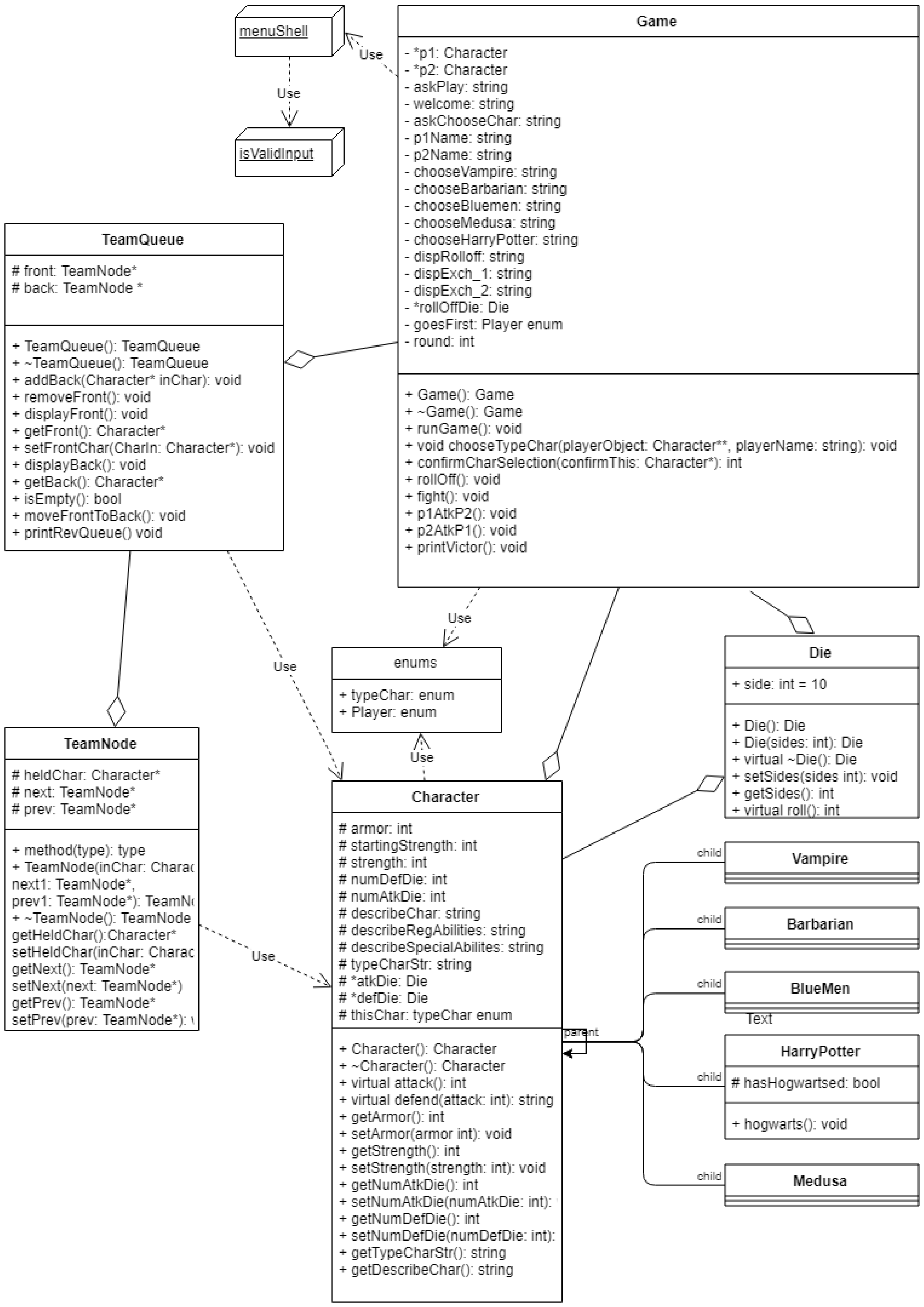
Phillip Wellheuser
CS 162-400
5/26/19
Project 4: Fantasy Combat Tournament

Planning:

This project is a further development of the previous fantasy combat game, which I got working pretty well in conjunction with the linked list designs which we've been learning the last few weeks. My linked list programs also went well, so I don't expect too much difficulty in figuring out how to make this extension of the program work. The complexity of the program is increasing, though, so I know I'll have to be careful in order to avoid getting confused or working myself into a corner with a bunch of hard to find bugs.

I feel I have a fair understanding of all of the concepts involved, so it will largely be a matter of keeping my logic sound, tracking my changes, and moving all of the characters into node structures. I intend to modify my existing queue node program from a previous assignment to make the queues I'll need for this one, and then for the stack of defeated characters, I'll simply print the queue in reverse. For the healing system after victories, I'm just going to roll a die of the initial strength of the character and heal them for that much (up to their starting strength). For the points system which determines victory, I was just going to have whichever team has members left alive at the end be the victor, but after some discussion with TAs I decided to give a point to teams each time their team wins a fight and then compare the scores at the end of the game to determine the victor.


menuShell
Use
isValidInput
Use
Game
- *p1: Character
- *p2: Character
- askPlay: string
- welcome: string
- askChooseChar: string
- p1Name: string
- p2Name: string
- chooseVampire: string
- chooseBarbarian: string
- chooseBluemen: string
- chooseMedusa: string
- chooseHarryPotter: string
- dispRolloff: string
- dispExch_1: string
- dispExch_2: string
- *rollOffDie: Die
- goesFirst: Player enum
- round: int
+ Game(): Game
+ ~Game(): Game
+ runGame(): void
+ void chooseTypeChar(playerObject: Character**, playerName: string): void
+ confirmCharSelection(confirmThis: Character*): int
+ rollOff(): void
+ fight(): void
+ p1AtkP2(): void
+ p2AtkP1(): void
+ printVictor(): void
TeamQueue
# front: TeamNode*
# back: TeamNode *
+ TeamQueue(): TeamQueue
+ ~TeamQueue(): TeamQueue
+ addBack(Character* inChar): void
+ removeFront(): void
+ displayFront(): void
+ getFront(): Character*
+ setFrontChar(CharIn: Character*): void
+ displayBack(): void
+ getBack(): Character*
+ isEmpty(): bool
+ moveFrontToBack(): void
+ printRevQueue() void
Use
Use
enums
+ typeChar: enum
+ Player: enum
Use
Die
+ side: int = 10
+ Die(): Die
+ Die(sides: int): Die
+ virtual ~Die(): Die
+ setSides(sides int): void
+ getSides(): int
+ virtual roll(): int
TeamNode
# heldChar: Character*
# next: TeamNode*
# prev: TeamNode*
+ method(type): type
+ TeamNode(inChar: Charac
next1: TeamNode*,
prev1: TeamNode*): TeamN
+ ~TeamNode(): TeamNode
getHeldChar():Character*
setHeldChar(inChar: Charac
getNext(): TeamNode*
setNext(next: TeamNode*)
getPrev(): TeamNode*
setPrev(prev: TeamNode*): v
Use
Character
# armor: int
# startingStrength: int
# strength: int
# numDefDie: int
# numAtkDie: int
# describeChar: string
# describeRegAbilities: string
# describeSpecialAbilites: string
# typeCharStr: string
# *atkDie: Die
# *defDie: Die
# thisChar: typeChar enum
+ Character(): Character
+ ~Character(): Character
+ virtual attack(): int
+ virtual defend(attack: int): string
+ getArmor(): int
+ setArmor(armor int): void
+ getStrength(): int
+ setStrength(strength: int): void
+ getNumAtkDie(): int
+ setNumAtkDie(numAtkDie: int):
+ getNumDefDie(): int
+ setNumDefDie(numDefDie: int):
+ getTypeCharStr(): string
+ getDescribeChar(): string
parent
child
Vampire
child
Barbarian
child
BlueMen
Text
child
HarryPotter
# hasHogwartsed: bool
+ hogwarts(): void
child
Medusa

New Requirements:

Basically the main task is to give each player a list of fighters set up in a queue system for their team, rather than a single combatant. So I'll need to modify several aspects of the program:

1. My main menu system must request a number of fighters for each team
2. I must modify character selection so that it creates a queue of team members
    a. For this I must create a node class to use for the queue
    b. Alter character selection to add to a queue instead of just assigning a character
    c. Add an opportunity to name characters
3. Modify checks for victory condition to look for surviving team members at all per team
4. Remove defeated characters from their current queue and put them on a stack with other defeated characters
    a. Print out the characters who died during the tournament
5. Recover some strength for victorious characters between battles
6. I may also want to modify my functions for printing battle results to make it more organized and flexible

Testing:

I think most of my testing will be a matter of just running the game and seeing if it crashes. The main parts that will break due to my misunderstandings will likely be node pointers, and not allocating memory properly, so valgrind will be valuable for this task.

<u>Test 1:</u> Does setting team sizes work?

The team size variables should reflect the choices made for both player team sizes
Result: as expected.

<u>Test 2:</u> Does creating teams work?

After choosing team sizes, characters should be selected and named for each members slot on each team.
Result: as expected.

<u>Test 3:</u> Does character selection work?

Character selection should no longer ask for confirmation of character selection and should set each team member node to the character selected as chosen
Result: as expected.

<u>Test 4:</u> Does character naming work?

When selecting each character for a team, a request for a name should be presented and the program should set the user input name to the name nameStr attribute of the character in the node currently being filled with that team member
Result: as expected.

<u>Test 5:</u> Does setting team sizes work?

The team size variables should reflect the choices made for both player team sizes
Result: as expected.

<u>Test 6:</u> Does printing queues in reverse work?

Printing a queue using the printRevQueue member function should print the names of the characters in the nodes of the queue in reverse to simulate stack format.
Result: as expected.

<u>Test 7:</u> Are defeated characters moved to the death stack queue?

The death stack queue should contain the characters who were defeated during the tournament and calling the printRevQueue member function on that queue should show this.
Result: as expected.

<u>Test 8:</u> Does the healing system work?

After winning a fight each character should heal by a dice roll of their starting strength and then heal for no more than their starting strength.
Result: as expected.

<u>Test 9:</u> Are there memory leaks?

Following every option (beyond choosing every different type of character in every combination) should result in no memory leaks when running valgrind, even when playing

through the game a second time at the end.
Result: as expected.

Reflection:

This project when surprisingly quickly. I attribute this to knowing my code base for the original Fantasy Combat game and having completed the linked list assignments for the labs already when I started this project. Pretty much everything went according to plan, and most of it was just connecting the dots between code which I'd already written for prior assignments.

The main issue I had in this project was with removing the all of the text feedback from the original Fantasy Combat assignment. I tried to pass a boolean to an overloaded constructor for the game which would turn on or off the extra text, however for some reason the boolean was not passing correctly. It seemed that constructor for Game was setting the boolean in the overloaded constructor based on whether there was a boolean parameter passed at all or not. I never did figure it out, but I switched the parameter to using an integer instead, and it worked fine.

I ended up making some of the changes I'd wanted to from the previous version, like making the attack functions for characters into a single attack function, and doing some other miscellaneous tidying up. I also fixed the miscalculation I had with the Blue Men in this version and I think I patched up the memory leaks I had in the previous version as well.

As far as I can tell, it fulfills all of the requirements of the assignment and I'm pretty confident turning it in.