COMP 1406-Z Final Project Analysis

By: Amelie Haefele

Dungeons and Dragons Based Game

Introduction:

This program is a text and turn based game similar to Dungeons and Dragons. There are three fights and six enemies for the player and their two followers to defeat. Each NPC, non player character(Enemies and Followers), has one of three designated fight styles. Throughout the game the Player gains experience points for successfully executing an Action, that is not a rest. This involves typing in an Action and a target then rolling to see if you complete said Action (roll is greater than or equal to ten). If the player reaches one hundred experience points it will allow for the player and their followers to level up, increasing their action point or health cap by five. Then based on the Player's level they will unlock special Actions to be added in their action list and used in the next battle.

Code Explanation:

The game is broken down into different classes to allow for better written and cleaner code. Here I will go over what each class does, the hierarchy, and how they all interact with each other. All references to things like Player and Print are talking about the class.

The design was based on MVC, model view controller, so the user will only have to use and look at the gameMain. All of the logic and main game components will be hidden from view. MVC is used typically to GUI's but can be and is applied in this case.

The game first starts off in a "main" class called GameMain, which will ask the user for their name, which can be anything, even numbers work as it only takes strings. From this name it creates a PlayGame object by passing the chosen name into the constructor. From there the playGameMethod is called on the game object.

This method starts off by calling createNPCAndFights, which creates all NPC objects and puts them into fight objects, which I will go over soon. If desired the playstyle and stats of all enemies and followers can be changed for ease of testing. It then creates a Print object allowing access to the Print methods, which displays the intro and rules.

From there it enters the fight logic loop. It indexes through the fight list where each fight gets called in order. But before the fight itself the index is passed into the Print method, which according to the given index will print the proper fight intro. All of this is happening in a while loop allowing for the fight to replayed if the user loses. The main Fight method is then called, playFight which I will describe later. This will return a true or false result, true if the user wins and false if they lose. If the fight returns false keepGoing will then be called asking the user whether or not they want to keep playing. If they return yes, the playFight loop will keep going. If they input the no the game will simply end.

The Fight class is where all the fight logic happens. It starts out by calling checkDead which goes through all the Characters in the given _hero and _eneny array's checking whether or not their health is zero or below. If their health is zero or below they will be removed from their arrays and placed in the corresponding dead array so they can be added back for the next round. This is done by the method restoreAllCharacters as well as healing all Characters to full. Note when they get re-added the order of turns may be different depending on who dies first. But the Player will always be brought to the front of the hero array, for aesthetic reasons. However, the order of the array has no bearing on the game logic itself.

Then it will check if the hero array or enemy array is empty (all enemies/heros have been removed and are dead). It will then enter a for loop going through each Character in the _hero array, again checking each turn for dead Characters and win conditions. If there are no interruptions the turn method will be called on the current Character. This will set the currentCharacter local variable to the Character being passed in. It will check if the currentCharacter having its turn is alive, if not the turn method will end. If all goes well it will have the method getAction called on them passing in the allied array (an array of the current Character's allies) and the current Character itself. Note there are caps of the health and action points of all Characters, this ensures that leveling up has more meaning.

The parent class, Character's method of getAction is abstract so the subclasses Player and NPC have said methods but have their own implementation. The Player method getAction will print out a list of current actions the Player can do. This Action hashmap, like the NPC hashmap, is obtained from the ActionList class where they are stored. Along with the name of the action other stats including its action point cost, the effect it has, the action type (harmful, helpful, or neutral), and the stat the action effects is printed. This is to give the user more information about the Actions they can perform. Then it will enter a while loop asking for the chosen action which is case insensitive. It will check if said action exists if not the user will be prompted to enter a valid action. After checking whether or not the chosen action exists it will check whether or not the Player has enough action points to perform said Action and will remove the actions points needed to perform said Action if true. If not, the user will get an error message informing they do not have enough action points and will be asked to enter a valid action again.

The NPC has a different getAction implementation. An Action is chosen based on the current NPC's play styles which it will check first. From this playstyle check the appropriate method will be called for example healerAction will be called if the current NPC is healer and so on for the other play style types. From there based on AI logic I have created said playstyle action will return a chosen Action. For example the healer playstyle will first check if anyone on their team needs to be healed, if true it will return a healing Action. If not it will check if the NPC has enough action points to attack if not it will return rest.

The turn method will then check the returned Action's type. If the Action's type is neutral, a rest, the only person that it can target is the current Character themselves. As you cannot rest for others. If not the getTarget method will be called. Much like the getAction method in Character this is also abstract.

If the current Character is a Player it will be called in that class due to type casting. This will display a list of possible targets based on the type of Action previously chosen. If the chosen Action is helpful then only team mates will be displayed. If the Action type is harmful then it will display a list of the enemies the Player can choose from. The user will be promoted to enter the name of the Character they want to target, which is case insensitive. It will be in while loop and continue to re-prompt the user until a valid target is entered. However, if there is only one target in the target array the game will target that Character for the Player as no others can be inputted.

Like the getAction the NPC implementation of getTraget will be based on the current NPC's playstyle. For example if the type of Action previously chosen is helpful the healer playstyle will target the Character on their team with the lowest health. If the previously chosen Action is not helpful the healer will target the opposition Character with the largest health. From there the method, effect found in Character will be called on the current target.

This method, effect, will first roll a d20 by calling the d20 method which returns a random int between 0-20. This will determine whether or not the previously chosen Action is performed. It will check if the given Action is neutral if so it will always succeed as one can always rest. If not it will check if the role is above or equal to ten. If true what happened gets printed out and effectCharacter will be called implementing the change on the chosen target. This will also call IncreaseExperiencePoints which does nothing if the Character doing said Action is an NPC. But if the current Character doing said Action is a Player then for every successful action they will gain fifty experience points. If the roll is under ten a missed message will be printed and no effect will be added to the target.

From there the Enemies will have their turns following the same pattern. After that all Characters in the fight and their stats will be printed out. This loop will play out until one of the win conditions has been met.

If the user has won and their experience points are one hundred or above they will be able to level up. This method levelUp will be called on all Characters in the users team. This is an abstract method in Character. If levelUp is called on a Player object it will prompt the user to increase their health by five or action points. This will also case insentive and continuing re-prompting the user until they enter a valid string. After this leveledLootSystem will be called which, depending on the Players level, will add a new Action to their Action list. The levelUp system for NPCS is much simpler as they will either increase health or action points based on their level, switching between one or the other.

On the last battle a special condition of the index equaling two is met. Which calls add hope sword to the Player. This is due to the story and adds a new weapon to the Player's Action list. Once the last battle is won the ending text method is called on the Print object and the game is finished.

Important(Already mentioned):

As mentioned there is a health and action point cap and you can try to rest more than the cap but this won't do anything. When they get re-added the order of turns will be different depending on who dies first. But the Player will always be brought to the front of the hero array, for aesthetic reasons. The order of the array has no bearing on the game logic itself. If the player dies their Followers will continue fighting till they lose or win.

Improvements from the 1405 version:

Java allows for better OOP which allows for much easier creation of Characters and their stored variables and methods then trying to keep everything together using hashmaps. Unlike the python version this game works as I tested the general code instead of making condensed smaller tests on the individual methods. Having said that I include no pre-written tests as when testing. Also JUnit 4 does not allow for user input when testing which is crucial to playing the game so writing tests were not feasible.

Current version code improvements:

Instead of removing the Character from its even array next time I would create another boolean variable in Character, alive that would change if the current Character's health went below 0. When initially creating the code I removed it so I wouldn't have to constantly check whether or not a Character was dead. However, this caused some problems when, unlike in python, deep copying was not feasible. This created a bug where I wasn't able to add Character back after they had died. I was able to fix this by creating a dead array and the restore method. However, this changes up the order of attack, which isn't too much of a problem as it always switches the Player to go first.

The increaseExperincePoints method looks a bit weird but without creating a different implementation in Player and NPC this cannot be changed.

Other than those two things I believe this is a great improvement from last time and the code is well written, consinse and much neater.

Time Complexity Analysis:

This code is O(n) based on how many Characters are added in the fights and game. This due the indexing being O(n) and most actions taking O(1). This is also because there are no searching algorithms and the addingLists method is still dependent on the length of each list being added (only going through each once).

Overall Code Complexity:

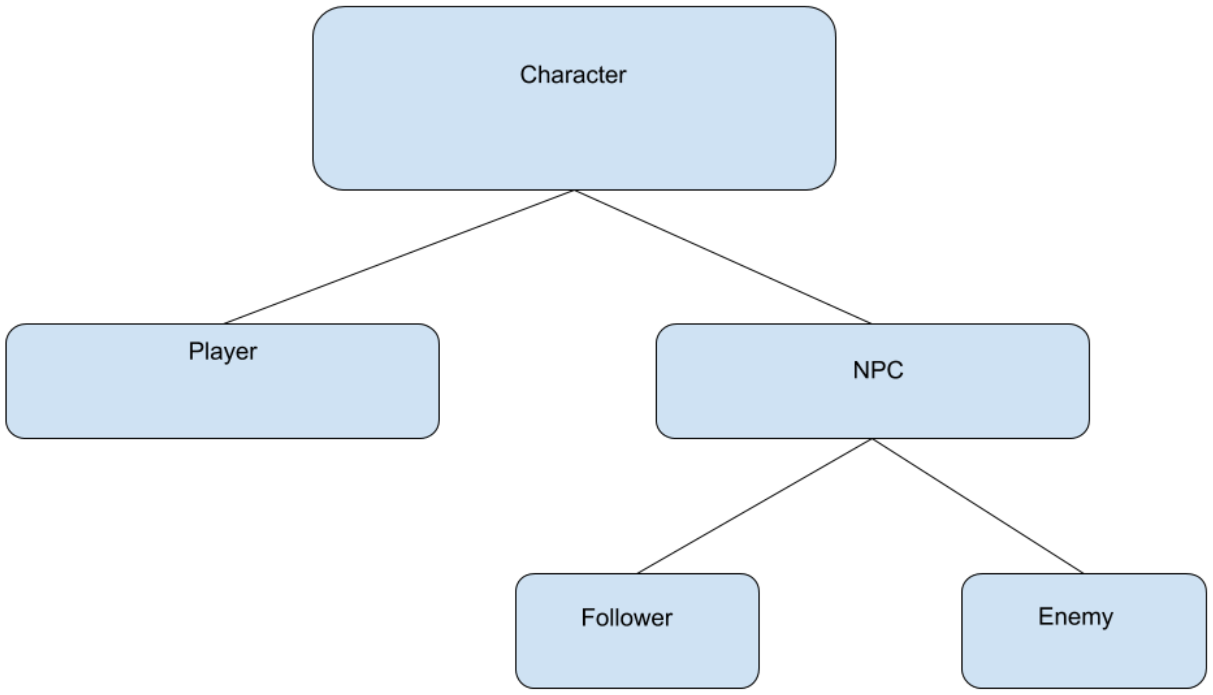
This code includes many course concepts and core principles of OOP. This complexity of this comes from the main logic and implementation of the three unique AI's.

This project shows usage of following course concepts by using:

- Polymorphism:
    - Different methods like getTarget, getAction, and levelUp are typecast to implement differently based on what object is called on them
    - Instance of is used once to set the Player to index zero in the hero in the method restoreAllCharacters
    - Overloading is used on increaseExperiencePoints on a Player object, this could also be argued is single dispatch
- Encapsulation/Abstraction:
    - Final is added to variables that so so they are not able to be changed
    - Variables are set to protected in the Character class so the Player and NPC classed below can access them
    - Variables that are used in other classes are set to public
    - Variables and methods that aren't being used are set to private so they can't be accessed when they aren't supposed to be
    - Getter methods are used to access variables directly so they aren't accidentally changed
- Abstraction Data Structures:
    - Some methods in Character are abstract so they can be implemented separately in the subclasses Player and NPC
    - The ActionList class constructor is set to private because for reason should anyone be making an ActionClass object
- Inheritance

- The Parent class Character has has two subclasses, NPC and Player which have their own implementations for certain abstract classes in Character
- NPC has two subclasses Enemy and Follower this is mostly for ease of creation of objects
- Enemy allows one to input the enemies level as they do not have the chance to level up

Below is a rough Diagram of the inheritance structure:



- Interfaces
    - Allows for Player and NPC to extend Character
    - Allows for Enemy and Follower to extend NPC
- OOP Principles
    - Actions, Players, Enemies, Followers, Fights, and PlayGames are all objects created using constructors allowing parameters to be passed in, this allows for modular programming
    - Copious methods are used to implement different things in all variables
    - The classes I have created are: Action, ActionList, Character. Enemy, Fight, Follower, GameMain, NPC, Player, PlayGame, PrintMethods

    - The code is very modular and general, you can change the Characters names, their stats, you can change your own team adding more followers to help you or you can add your own actions, allowing you to battle whoever you want with whatever you want
    - Java code etiquette:
        - Camel case is implemented for all variables and methods
        - Class variable names begin with an _ to set them apart of the non _ names passed into the constructor
        - Class methods begin with an _ to see that they are associated with a class method
        - All class names start with an uppercase letter
- JavaDoc
    - Generate a java explaining all methods and their variables

Extra Story Analysis(Same story as last time):

The story starts off with you and your two best friends as kids. As each fight progresses you get older until the boss fight. The boss's name in Latin is responsibility so in essence you're fighting growing old and being responsible. You first try to take responsibility/adult life all on your own but your friends come and stand by your side to help you. In the end you win as you and your friends have all become adults.

There are some other story elements; such as Danielle thinking the game(DnD) was lame at first but then learning to have fun without worrying about what others think, and references(Monty Python) throughout which I won't explain here.