CS5004 Final Synthesis Project Report

1. **Academic integrity statement**

   I understand that my learning is dependent on individual effort and struggle, and I acknowledge that this assignment is a 100% original work and that I received no other assistance other than what is listed here.

   Acknowledgements and assistance received:
   Professor Domino
   TA:
   Websites: https://docs.oracle.com/javase/8/
   https://www.baeldung.com/
   https://www.geeksforgeeks.org/

   I did not use generative AI in any form to create this content and the final content was not adapted from generative AI created content.
   I did not view content from anyone else's submission including submissions from previous semesters nor am I submitting someone else's previous work in part or in whole.
   I am the only creator for any un-cited content. All sections are my work and no one else's with the exception being any starter content provided by the instructor. If asked to explain any part of this content, I will be able to.
   By putting your name and date here you acknowledge that all of the above is true and you acknowledge that lying on this form is a violation of academic integrity and will result in no credit on this assignment and possible further repercussions as determined by the Khoury Academic Integrity Committee.

   Yongzhen "Michael" Zhang
   5.26.2024

2. Introduction:

Explain your approach and organization. What is your "main" application and what smaller applications did you have to create to fit in other concepts. Tell us anything else you want us to know as we evaluate your submission. Include a UML diagram as well.

I started with implementing the Protagonist class, and the map class. Then I was thinking about combining the protagonist class with the map, i.e. how does the movement of protagonist get reflected on the map. The functionalities of the whole RPG game, including add/remove object from the grid, set game state, interact, check if there is object nearby, was handled by the game Class, which is the model in the MVC pattern. And after successfully doing that, I implemented more game objects and was thinking about the logical abstractions that I could use to help implement the objects that will be used in my project.

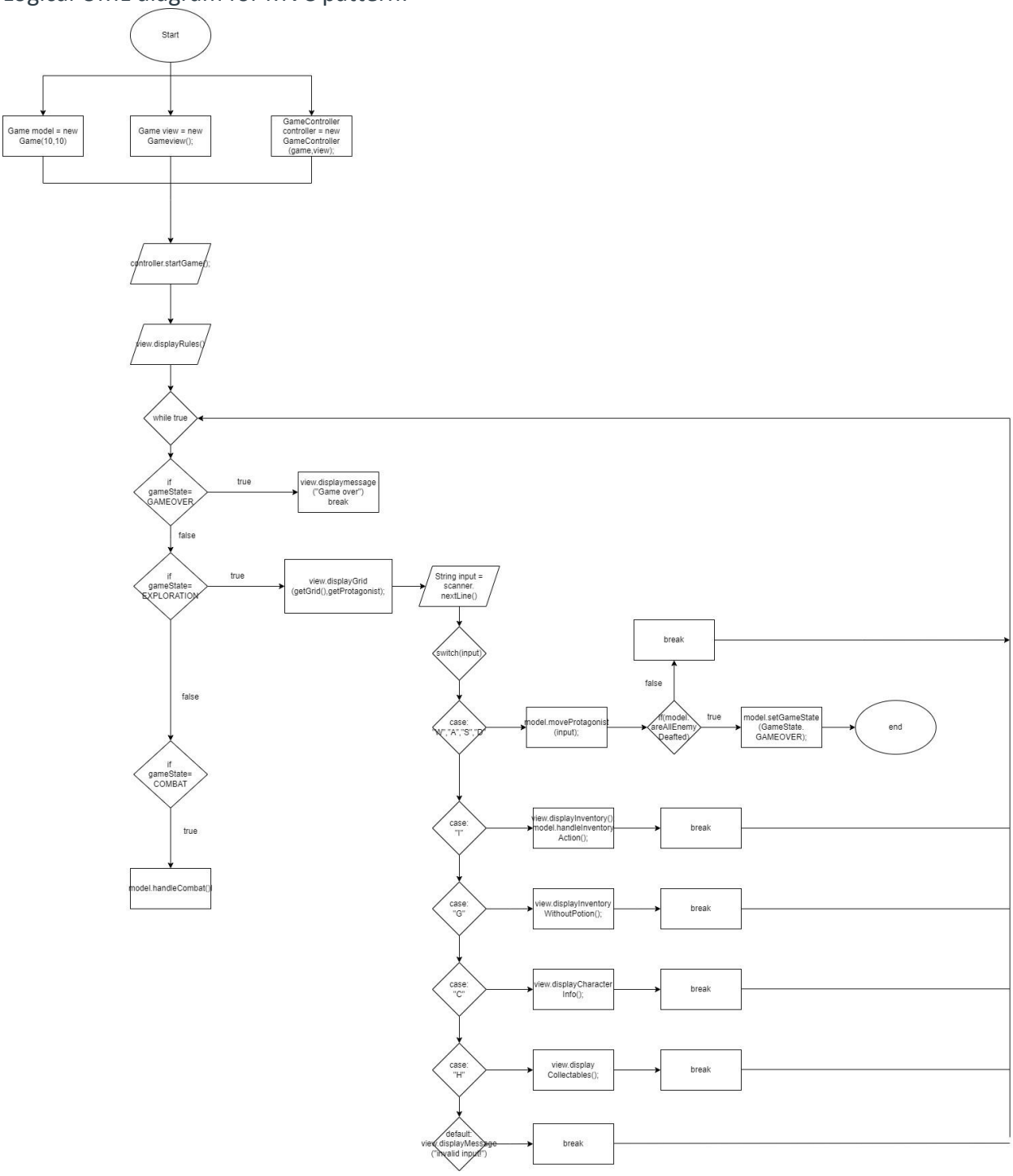
The overall structure of this game can be divided into different game states. I used an Enum to define the game state into three states: EXPLORATION, COMBAT, GAMEOVER. Exploration is when the protagonist traverses on the map before he/she meets any enemies. The grid will be shown in the exploration state.

Combat state is when the protagonist runs into an enemy, in this state, the map will not be shown, instead, the combat information will be displayed to the user with some choices given (1. attack, 2. escape). After the combat is over(either the protagonist dies or the enemy is defeated), a corresponding state will be triggered.(going back to EXPLORATION if the enemy is defeated, or GAMEOVER if the protagonist dies)

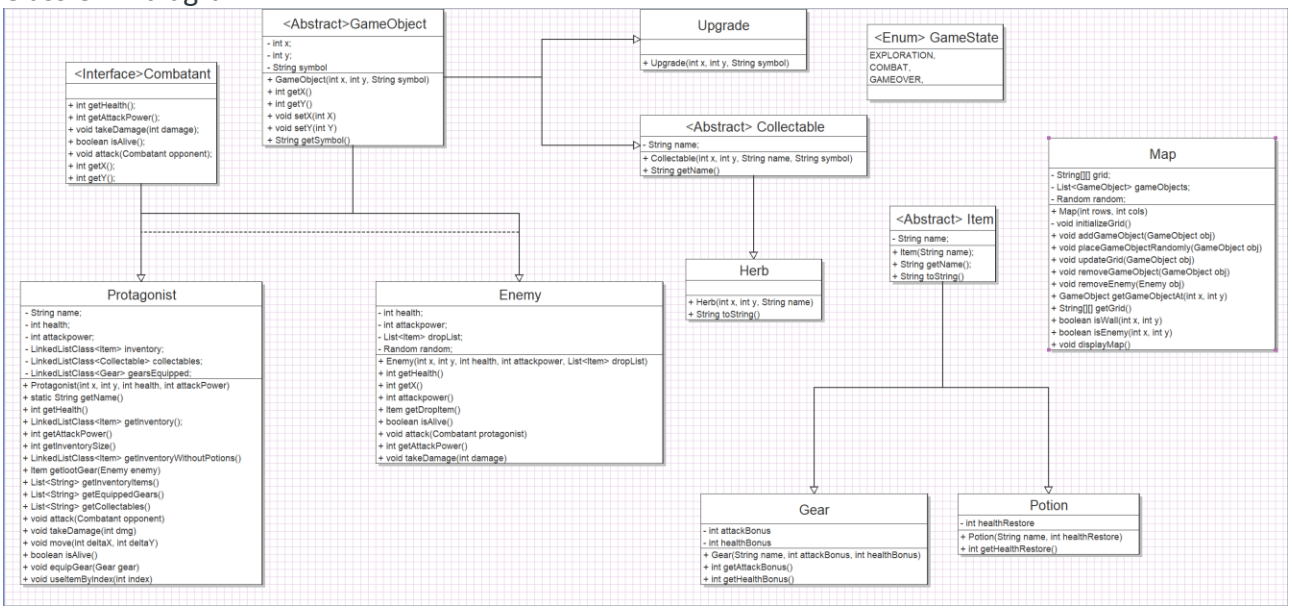
GAMEOVER state will stop the game. It is triggered by either the protagonist is defeated, or the protagonist defeating all enemies on the map (4 enemies in total).

The state transition is handled by the game controller class, as the controller part of the MVC design pattern.

Logical UML diagram for MVC pattern:



Class UML diagram:



3. **Grade yourself.**

Write me up a 1 paragraph reflection on your performance. Use the rubric below and tell me what grade you'd give yourself. I will take this into consideration when grading.

Based on the rubric, I incorporated all the required concept in my project, including: Recursion, Logical Structure and Design using Abstract Classes and Interfaces, Generics and Lambda Expressions, 2 Higher Order functions (filter and map), a linked list ADT, use the MVC design pattern, SOLID design principles. I also have turned in the initial proposal, written proposal, and submitted all check-ins earlier. I think I should get an A, if the follow up interview is finished satisfactorily.

4. **Concept Map:**

Create a table showing exactly where we can find each concept demonstrated and a short description of how it was demonstrated.

Concept 1: Recursion in Practice	I used this concept in the Node class. Look in file Node, starting on line 150, in method: map and filter. I demonstrated this concept by using recursive call in those two methods. The Node class recursively calling the map and filter function.

Concept 2: Logical Structure and Design using Abstract Classes and Interfaces	I used Abstract Class concept in the GameObject class, Item class and Collectable class. I used the Interface concept in Combatant interface and LinkedList interface. GameObject is representing the objects that will appear on the game map, so any class that extends the GameObject will be the object that has an x and y axis. The Item class is an abstract class that represents the object that will not appear on the map, the class Potion and Gear extends the Item class. Collectable class extends the GameObject class, they are the objects that will be on the map and are collectable to the Protagonist. Combatant is the interface for the object that can be in a combat, to be in combat an object must have hp, attackPower, and can attack other, can take damage. The class Protagonist and Enemy are under Combatant, these two classes implement the Combatant interface. LinkedList interface was used to help the implementation of LinkedListClass.
Concept 3: Useful and Logical Abstraction using Generics and Lambda Expressions	I used this concept of generics in the LinkedListClass class so that the linked list structure can store different types of objects, e.g. for items, collectables, gear and potions etc. I used the concept of Lambda expression in the Protagonist class, starting at the line 106, in methods getInventoryItems(), getEquippedGears() and getCollecetables(). And, at line 234, the upgradeAllGears()function, I used lambda expressions in the filter and map functions.
Concept 4: The use of 2 Higher Order Functions (Map, Filter)	I used Higher Order Functions in the Protagonist class, starting at line 106, in methods getInventoryItems(), getEquippedGears() and getCollecetables(). And, at line 234, the upgradeAllGears()function, And in the LinkedListClass class, starting at line 147, in methods filter(Predicate<T> filter) and map(Fucntion<T,R> mapper) Also in the node class, starting at line 150, in methods filter (Predicate<T> filter) and map (Fucntion<T,R> mapper)

	I used the map function in the equippedGears linkedlist of the Protagonist, so that when the Protagonist collect a gear Upgrade, all the currently equipped gears will be placed by upgraded gears. And used map function in getInventoryItems(), getEquippedGears() and getCollectables(), to form a list and return the list, and the GameView class can then iterate and display the items from those inventories. I used filter function in the Protagonist class, at line 83. In the method: LinkedListClass<Item> getInventoryWithoutPotions() to filter out the potions and so that only the gears will be displayed, as an option for the user.
Concept 5: Hierarchical Data Representation as an ADT or a Linked List ADT	I used a linked list ADT in my project, see the LinkedList class, the LinkedListClass class, and the Node class with EmptyNode. They are used to store different type of objects as an inventory.
Concept 6: Some design pattern: MVC Design	I used the MVC design pattern. See the class: Game, GameController, and GameView. Game is the model, GameController is the control and GameView is the view. Each class has its own responsibility and view would have no access to the model data. The controller is handling the model data and calling view to display information. The game is responsible for the game functionality.
Concept 7: SOLID Design Principles	1. *<u>The Single-responsibility principle:</u>* There should never be more than one reason for a class to change. Each of the classes is responsible for only one responsibility. In my project, each class are responsible for a purpose: 1) Collectable class handles the collectable items, the constructor and getters. 2) Enemy class governs the behavior of the enemy, along with the features and getters. 3) Game class works as the model in the MVC structure, it holds the functionalities of the game.

4) GameController class works as the controller in the MVC structure, it manipulates the input and invoke the corresponding methods in game without letting the view have direct access to the model data.
5) GameView class works as the view in the MVC structure, is responsible for printing out the information to the user.
6) GameObject class is an abstract class whose object has x and y axis coordinates, and thus will be on the map for display. It has getters and setters in the class.
7) GameState is an Enum that has all the possible game states.
8) The Gear class is the item class's child class. Gear class's objects are the gears that would appear in the inventory or be equipped by the Protagonist, it won't appear on the map and thus it does not have x and y axis coordinates as the features. The class has constructors and getters.
9) The Herb class is Collectable class's child class. Her class's objects are the herbs that would appear on the map, (Collectable being GameObject's child class) The class has constructor and getters.
10) The Item class is an abstract class that represents an item that will not appear on the map but can be stored in Protagonist's inventories.
11) The map class is responsible for creating a map and has other map related functionalities including placing game objects on the map or removing them.
12) The Potion class extends Item class, it represents the healing potion the protagonist can store in the inventory. The protagonist can use the potion to heal, that is why the potion class has a int feature for healing amount.
13) The Protagonist class the handling the creation, movement, interaction, inventory related action, combat related action (attack and take damage), it is extending the GameObject class and implements the Combatant interface.
14) The upgrade class extends GameObject, its objects would appear on the map and if collected by the protagonist, the currently equipped gears of the protagonist will be upgraded. The class includes constructors and getters.
15) Other classes (Node, EmptyNode,LinkedList,LinkedListClass) are used together to create the ADT linked list structure.

2. *The Open-closed principle*

I created interfaces and abstract classes (mentioned above) logically so that if there is more features needed in the class, the source code will not be modified.

3. *The Liskov substitution principle*
   Abstraction, A type can be replaced by its subtype without altering the properties of the program.
4. *The Interface segregation principle*
   Abstraction, A type can be replaced by its subtype without altering the properties of the program.
   In the GameController class, line 100, the method handleInventoryAction() checks if the Item is a potion or gear, if the item is gear, it then type casting the item to be gear, and was able to use gear's methods without causing any errors.
5. *The Dependency inversion principle (Abstractions should not depend on details. Details (concrete implementations) should depend on abstractions.)*
   The interface Combatant was used to help implement the Protagonist and Enemy class, and the details are contained in the implementations, not in the interface.