Homework 8

Model and Simple Console Controller

Northeastern University

CS5004 – Object Oriented Design

Team: TBH

Chia-Hsiang Hsu Tai

Bhoomika Gupta

Harrison Pham

Implementation Overview

We implemented the main game elements based on the data provided in the JSON files. We used Jackson to map the data into a GameInput class that would then be used to create all the classes that are specified in the game file. Most of our design ideas remained the same to maintain separate responsibilities among classes and prioritizing the use of composition (with interfaces) over inheritance. We wanted to keep our decision to implement more interfaces to promote flexibility and loose coupling. The new additions to our implementation were the simple controller that managed the user input and triggered the appropriate methods in the Game Model to change the state of the game. The controller utilizes two classes, the GameInputReader (which takes an input string from the user and identifies the action word and the argument) and the GameCommandFinder (which uses the action verb identified by the reader and retrieves the appropriate method in the GameModel class). The goal behind these two classes was to maintain responsibility separately, with one focus on parsing the user input, and the other determining the correct method to process that input. The GameCommandFinder triggers the methods in the model through a ICommand interface. Each possible action in the game has its own class that implements the ICommand interface, they all have one method that executes the given action. This command design pattern was done to provide flexibility in making sure that if we wanted to add a new command or functionality, we would be able to do so by just creating a class. Additionally, we implemented three Enums to determine the player rank, the player health status and the valid commands. We wanted to stay away from hardcoding values and messages as much as possible, and using Enums allowed us that flexibility. Our implementation also has fields and methods that are used to retrieve visual images or pictures of the classes. This was data that was provided by the JSON file, but its implementation is out of scope for this current assignment. We have included the fields and methods for future implementation. It is also important to consider scalability, with each command being its own class can be challenging as the game increases in complexity. For this assignment, the idea of scalability is also out of scope.

UML Class Diagram

This UML Class Diagram represents our test-based adventure game implementation. The design continuously prioritizes modularity and separation of concern through interfaces and abstract classes. We have implemented a new controller and all its associated classes, and to remain consistent with our intent to keep the design modular, we have also chosen to use more interfaces and Enums. The UML diagram has remained consistent compared to homework 7, the new additions are mostly on the Controller side. We implemented a GameController class that manages the state of the game. The flow of the game starts with the GameInputReader that takes in an input command from the user, if the command is valid, it splits the command into an action and an argument (if applicable). It sends this information to the GameCommandFinder, to find the method in the GameModel that is associated to the command. Once it finds the associated method, it sends the method to the controller so the controller can trigger it and tell the GameModel to change the state of the game.

Each command is its own class, this is done to keep the code modular and allow us to easily delete or modify a specific command if necessary. All commands implement the ICommand interface, which makes it abide by the ICommand contract.

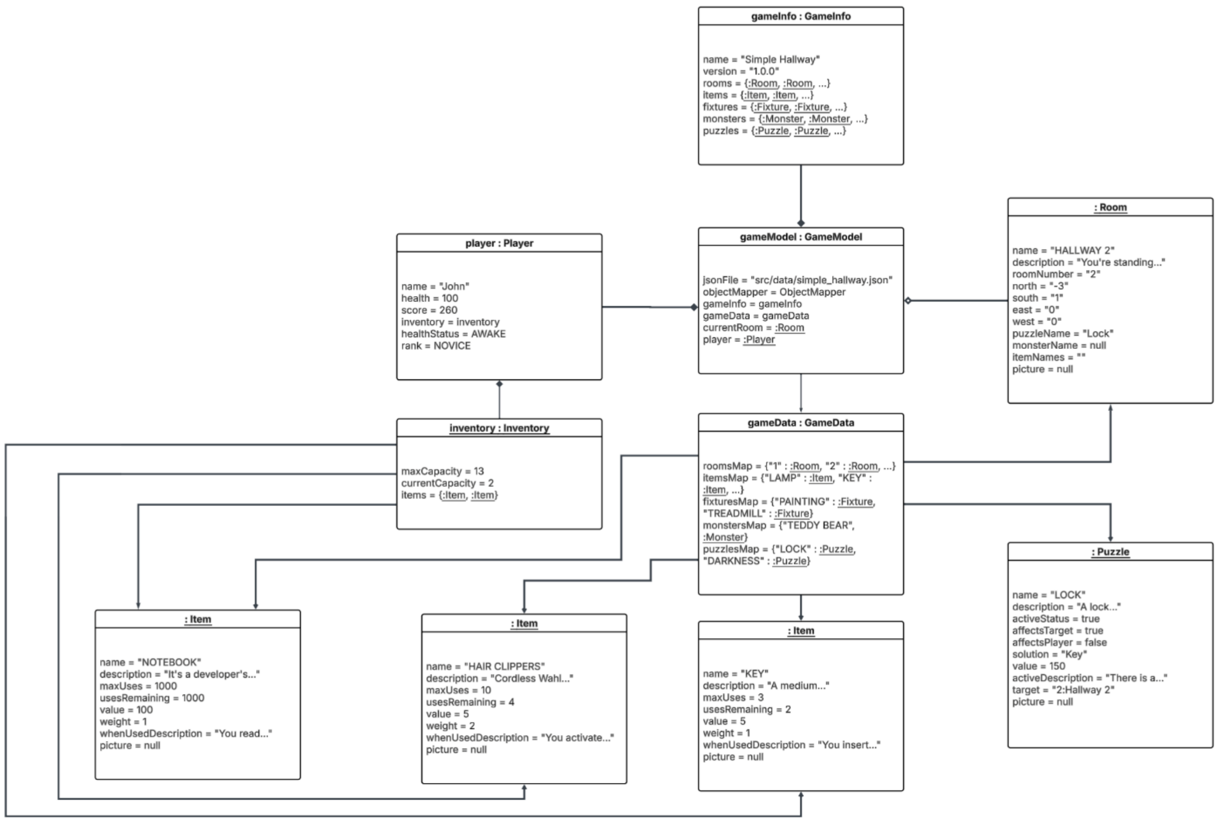
The UML has grown significantly since Homework 7. We were unable to fit it into the whole Class diagram into a single page and therefore we have attached it as a JPG in our submission.

Written Scenarios

The player starts the game in Hallway 1. The room description says that there is a notebook in this room. The player takes the notebook and places it in their inventory. The player attempts to move south but unfortunately there is no path and the player remains in Hallway 1. The player chooses to go north and enters Hallway 2, this time there is a key and hair clippers. The player takes both items. The player decides to go north again but there is a locked door. The player checks their inventory, and they see a key. They use the key to unlock the door. Success! The key turns and unlocks the door. Now that the door is unlocked, the player chooses to examine the lock.

Object Diagrams

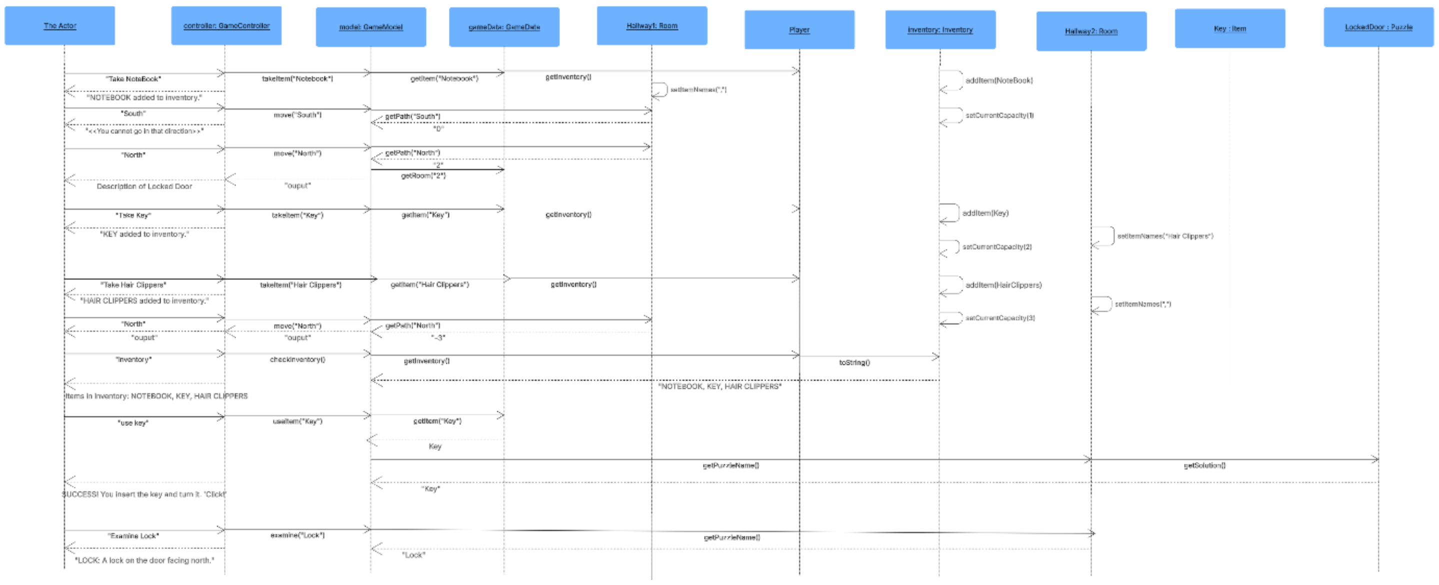
The object below diagram is based on Written Scenario. It is a concrete representation of the game state after the player solves the "locked door" puzzle by using a key item and proceeds to Hallway 2. We have attached the object diagram to our submission as a JPG.



Sequence Diagrams

The sequence diagram below depicts Written Scenario. The general idea for this written scenario is that the player attempts to move until it successfully moves to a hallway that contains an obstacle. The obstacle will be a puzzle (locked door), which the player will solve by using a key item from the inventory.

We have attached a JPG to our submission for a clearer view of the sequence diagram.



Assumptions

Based on the newly provided information, some of our assumptions are no longer true, while others have held. Below is the list of assumptions that remain after the additional information provided in Homework 8. It is significantly smaller than Homework 7, but that's mainly because the requirements and specifications have become a lot clearer with the data provided and gameplay videos.

Player Assumptions

- A player's score is determined through the sum of the following points:
  - Points from defeating a monster or solving a puzzle.
  - The sum of the item points currently in inventory.

Fixture Assumptions

- Fixtures are immovable; therefore, we assume their weight is above the maximum weight the player can put in inventory.

Item Assumptions

- Items can be USED on monsters (to defeat them) and on puzzles (to solve them).
- Items that can no longer be used remain in inventory until they are dropped by the player. Even if they can no longer be used, their presence in the player's inventory means that the item's points are counted towards the player's score.
- Items used to solve puzzles or defeat monsters also remain in the player's inventory.
- Every item can be a solution to a puzzle or to defeat a monster.

Questions

Monsters Mechanics

- When you use an item with a monster present in the room, does the item impact the monster always? Or could it be used on the player while a monster is active in the same room?

Inventory and Item Mechanics

- If an item is completely used, does it disappear from inventory or does it stay in inventory until it's dropped?

Puzzle Mechanics

- Are the solutions to puzzle clear, such that the player knows what the solution is? For example, if the puzzle is a locked door, does it tell the player they need a key or does the player have to intuitively figure out that a key is needed?
- Apparently puzzles can be in fixture just like they are in room. How does that relationship work?
- When dealing with a puzzle, is the "USE" command used to use an item on the puzzle and the "ANSWER" command used to provide an answer to a puzzle? Or is the "ANSWER" command the only command to interact with a puzzle?

Fixture Mechanics and Navigation

- We are told that puzzles can be in fixtures, what is the goal of solving a puzzle in a fixture? Can items be on fixtures?

References

- https://en.wikipedia.org/wiki/Adventureland_(video_game)
- https://iplayif.com/?story=http%3A%2F%2Fwww.ifarchive.org%2Fif-archive%2Fgames%2Fzcode%2FAdventureland.z5
- https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=c8fb2af239a3dca6c7d212a09bcf14769f12e245
- https://www.cs.cornell.edu/courses/cs5150/2014fa/slides/D2-use-cases.pdf
- https://refactoring.guru/design-patterns/command

Shout Outs

Homework 7

- We would like to thank Chirag and Karyn for providing us with more clarity surrounding the assignment and reminding us to keep in mind modularity and flexibility as we begin to think about how we are designing the game.
- We would also like to thank Kiersten for clarifying the level of granularity needed for the written scenarios.

Homework 8

- We would like to thank Melissa for clarifying some doubts regarding the requirements of the assignment. Particularly what we were supposed to submit along with the adventure game code.