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 game in a way that would populate the classes by reading through any given JSON file. Due to this new constraint, the data layers are managed by a new GameLogic class (called GameModel) and the original GameData class. This helps maintain a clear separation between game functionality (managed by GameModel) and game data while ensuring controlled access through getters and setters. Compared to Homework 7, most of our design ideas have remained the same and the flow of the game is very similar. There are small implementation details that have changed; for example, we focus more on composition and continue to prioritize the use of interfaces. Our implementation has a clear Model-Controller separation, where the Model manages the game state, and the Controller (along with some helper classes) manages the user input and validation. We continue the interface focused approach for encapsulation, by adding an IController and ICommand interface while keeping all the interfaces specified in Homework 7. Additionally, we have also focused on moving away from hard coding string or numeric values by utilizing a few Enums, particularly for the player's rank, score thresholds, and health status. This design decision helps us keep the game more flexible to implement new features like the game view in a future assignment. We have also left certain aspects of the game out of scope for this current implementation. We have included fields and methods to retrieve visual aspects in the game, but the implementation of any graphics is not within the scope of this current assignment and is something that will be explored in the future.

We emphasize the importance of promoting flexibility and loose coupling, which is why our approach is very interface driven. One aspect of this, connected to flexibility and modularity, is scalability. For our implementation, we haven't considered certain aspects of our game scale as we have chosen to add more features. For example, we have used the Command Pattern for our commands, and one of its downsides is that as the game becomes more complex, the number of commands can increase significantly leading to higher maintenance. These are some of the considerations that our team has made but consider to be out of scope.

UML Class Diagram

The UML class diagram represents our text-based adventure game implementations, with an emphasis on modularity, separation of concern, and maintainability, through interfaces and abstract classes. The general game flow of our initial UML diagram has remained the same, but we have now added a controller to help us manage user input and game state. From the model side, the three most important classes are the GameModel class (manages state), the GameInfo class (creates classes based on input data), and the GameData class (stores references to objects in the game). With these three key classes we can manage the whole adventure game while keeping separating responsibilities. The addition of the GameController provides a bridge between user input and the model. The GameController works with the GameInputReader and the GameCommandFinder to tell the GameModel how the game state must be changed. We follow a similar idea in the controller than we do with the model, wanting to separate responsibilities, and each method does exactly one thing. The GameInputReader validates the user input to make sure it's a valid command, the GameCommandFinder will find the method that triggers the sequence of events associated with that command, and the GameController tells the model to run it.

The UML has grown significantly since Homework 7. We were unable to fit the whole class diagram into a single page while making sure it was readable and clear. We have attached a JPEG file in the submission with the UML Class Diagram.

Written Scenarios

Written Scenario #1

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.

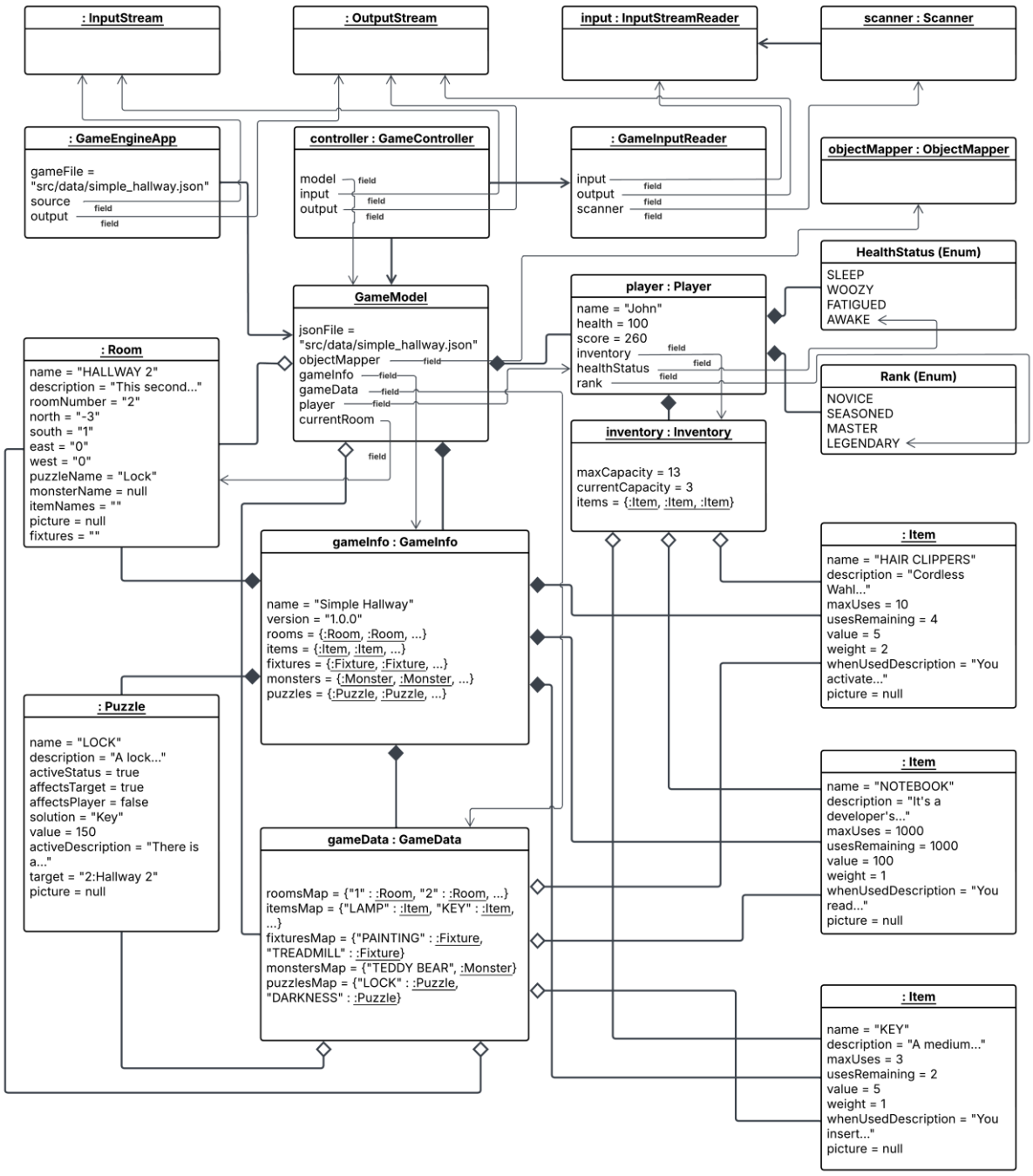
Written Scenario #2

The player is currently in Hallway 2 and decides to head north. The player enters Hallway 3 and encounters a Teddy Bear monster. The monster attacks the player and deals 5 damage points to the player. The player then checks their inventory to see what items they have to fight the monster. The items in the player's inventory are displayed and the player sees they have a notebook, key, and hair clippers. The Teddy Bear notices the movements of the player and attacks the player again and their health is reduced by 5 points. The player then uses the hair clippers to defeat the Teddy Bear and their overall score increases. The player then decides to head north and enter the Exercise Room.

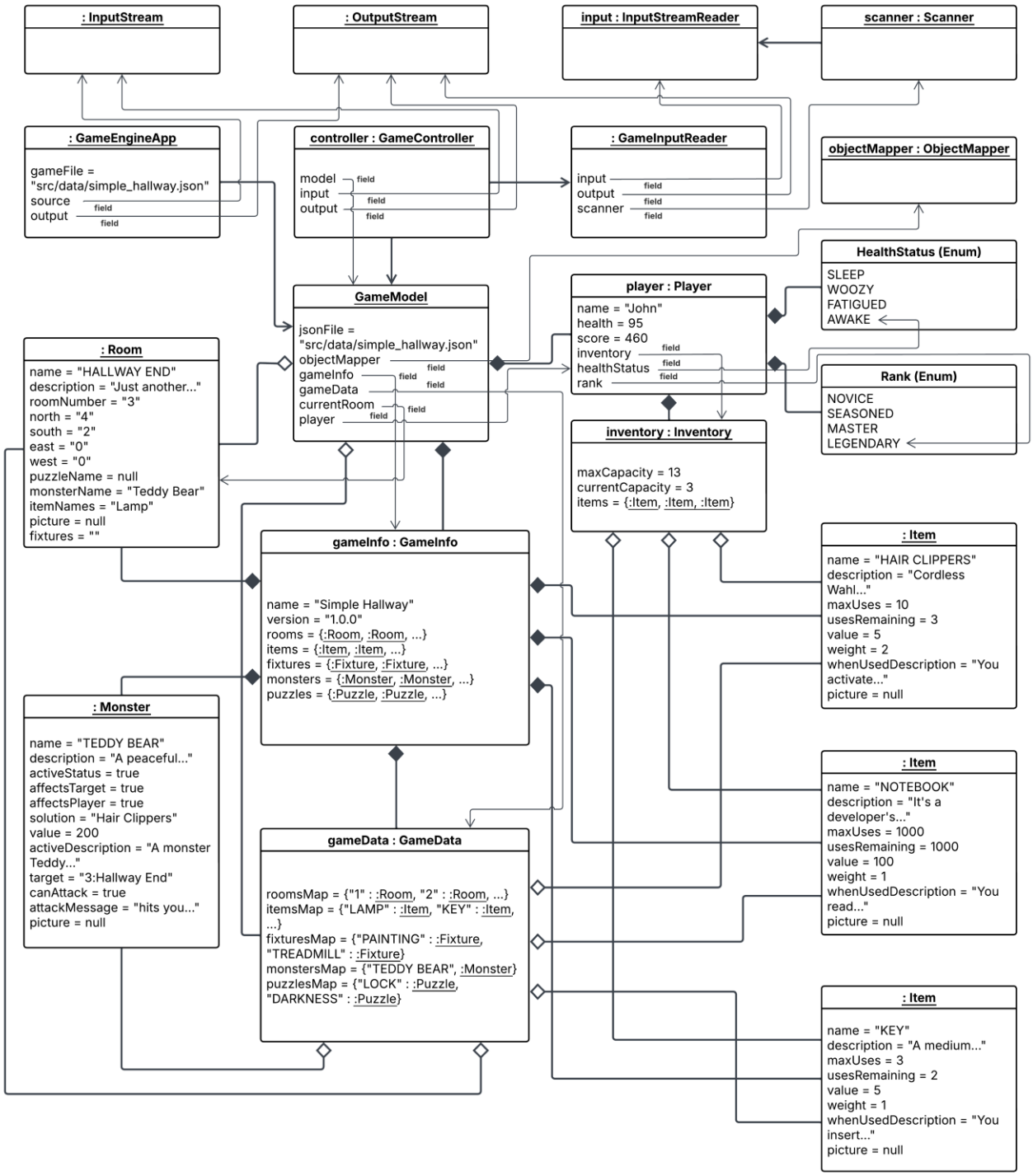
Object Diagrams

We have attached the object diagram to our submission as a PNG.

The object diagram below is based on **Written Scenario #1**. It is a concrete representation of the game's state after the player solves the “locked door” puzzle by using the “key” item and proceeds to Hallway 2. The player’s score is increased by the value of the puzzle, and the remaining uses for the “key” item is reduced by one.



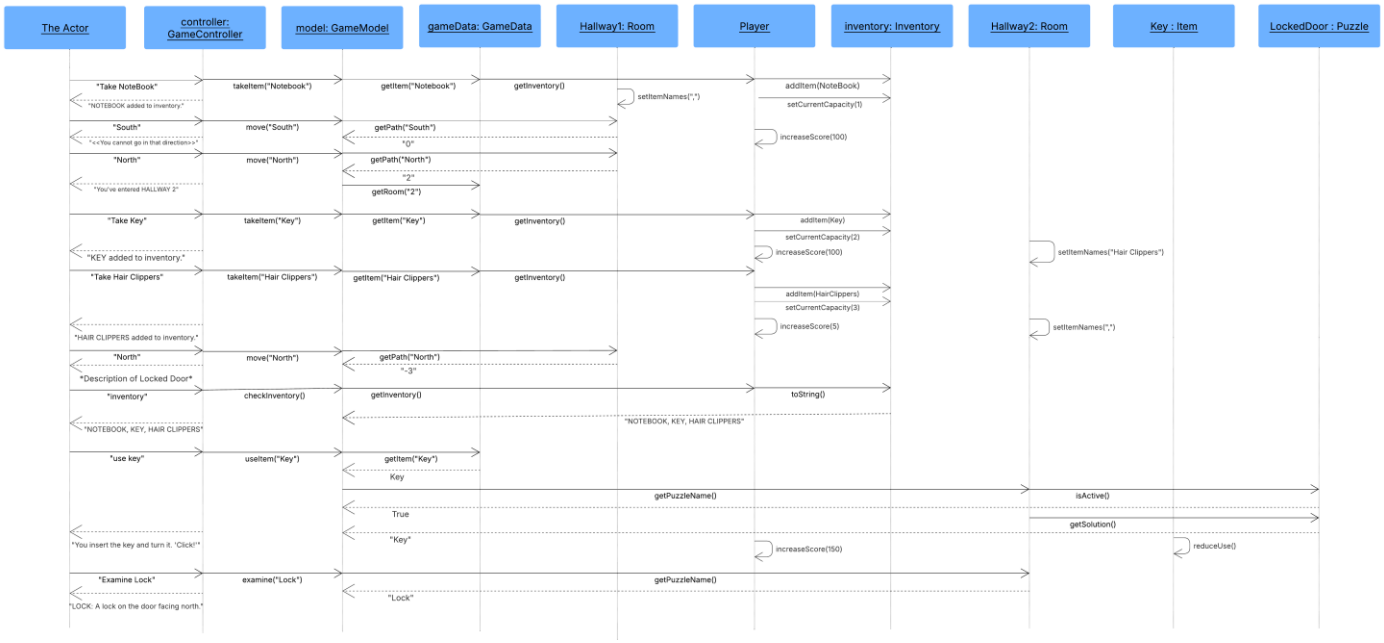
The object diagram below is based on **Written Scenario #2**. It is a concrete representation of the game's state when the player enters Hallway End, faces the "Teddy Bear" monster and defeats it with the "hair clippers" item. The player's score is increased by the value of the monster, and the remaining uses for the "hair clippers" item is reduced by 1.



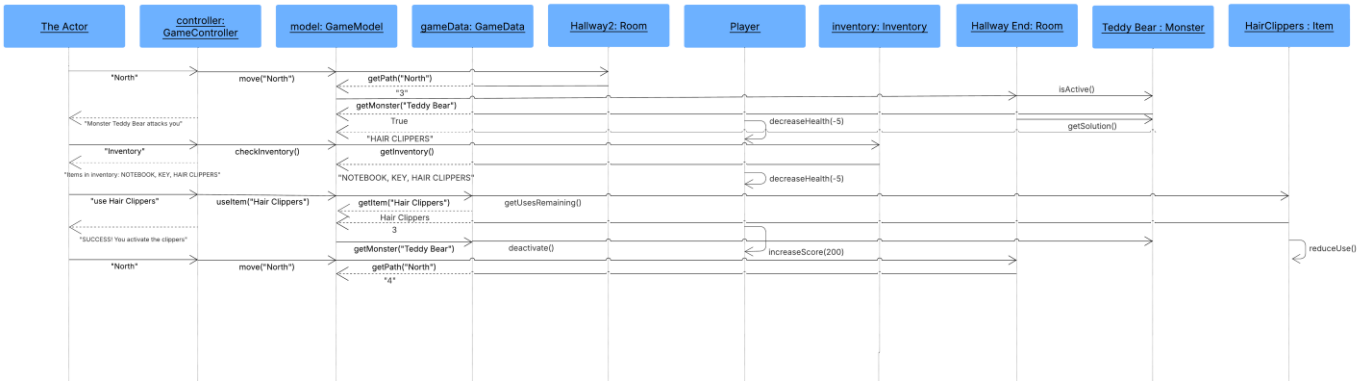
Sequence Diagrams

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

The sequence diagram below depicts **Written Scenario #1**. In this scenario, the player attempts to move until they successfully move to a hallway that contains an obstacle. The obstacle will be a "locked door" puzzle, which the player solves by using a "key" item found in their inventory.



The sequence diagram below depicts **Written Scenario #2**. In this scenario, the player moves to a new room that contains an obstacle. The obstacle is a "Teddy Bear" monster which attacks the player upon entering the room, decreasing their health. The player puts the monster to sleep by using an "hair clippers" item from their inventory.



Assumptions

Based on the information provided in Homework 8, our list of assumptions has shrunk significantly. Most of our implementation designs follow the rules specified in this assignment, but there is still room for us to explore different game behaviors. The game behaviors and assumptions that are unique to us are detailed below.

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.
- The player's rank thresholds are as follows:
    - Novice (starting rank), Seasoned (250+ points), Master (400+ points), Legendary (700+ points)

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.
- Items can be used on game entities (monsters, puzzles, etc.), but not on the player.

Movement Assumptions

- Based on how the JSON files provided are written, it seems as though a player can move from one room to another while a monster is active in one direction. The only direction they can't move in is the one with an active monster. We chose to follow this idea in our implementation.

Command Assumptions

- We assume that when dealing with a puzzle, if the solution to a puzzle is an item, you use the "USE" command, whereas if the solution is an answer, we use the "ANSWER" command.

Questions

Monster 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 a fixture just like they are in a 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
- https://cs111.wellesley.edu/~cs111/archive/cs111_fall97/public_html/lectures/ObjectDiagrams/object-diagrams.html

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 what documents had to be included in our submission along with the adventure game code.