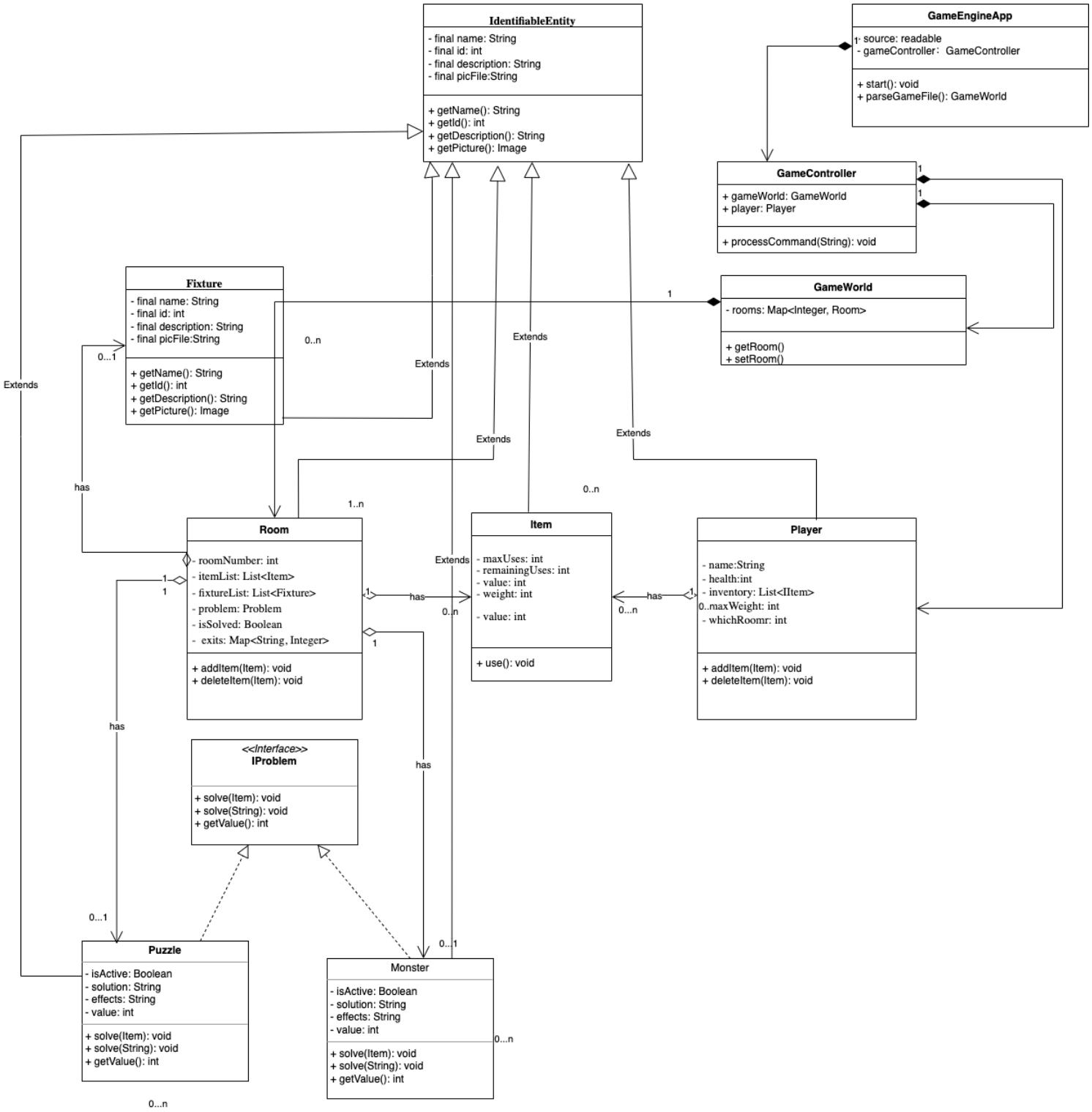
Overview

The key abstractions are still in talks, but in addition to the requirements like having a player, items, fixtures, monsters, room, and etc, we have added several other ideas, including but not limited to: IdentifiableEntity interface, GameController class, and other key ideas like having a main driver for the game, that is the GameEngineApp. Nearly all objects, both movable and non-movable, in the game will be extended from the IdentifiableEntity class, which gives them a name, Id, description, and a picture. This class encompasses the ID that differentiate all the players and monsters in the game. Specifically for players, we have a health status enum to represent four levels of health status. Players, in the game, will be able to get stronger by leveling up, gain or lose health, solvings puzzles (either monsters or word puzzles). We will probably have an interface or an abstract class for nearly all the behaviors or actions of the game. That said, many of our classes will share the same behavior. For example, both room and player have the addItem and deleteItem behavior, since they both either contain items or carry items. In addition, Items can be used in a variety of cases, and it could be used to solve a problem, something that can increase damage output, or just simply something irrelevant that the player picked up. In the beginning, we thought fixtures and items can share an interface, but we later decided on fixtures being part of the room class, since it can't be picked up or moved and is fixed in place. On the other hand, items can be picked up, dropped down, and even be used. With that being said, fixtures won't extend from IdentifiableEntity. For monsters and puzzles, we settled on two ideas. One of which is represented by the puzzles and the other is conveyed through monsters. Unlike regular puzzles, monster challenges' can affect players' health.

As mentioned above, this is a draft of our key abstractions, and we believe we will make adjustments to our design as we move forward with the project. However, we think this is a good start and, with numerous high level interfaces and classes, it will help with modularity and reusability. In this project, we will be utilizing encapsulation and polymorphism since we will be breaking down a whole idea into smaller components.

UML Class Diagram



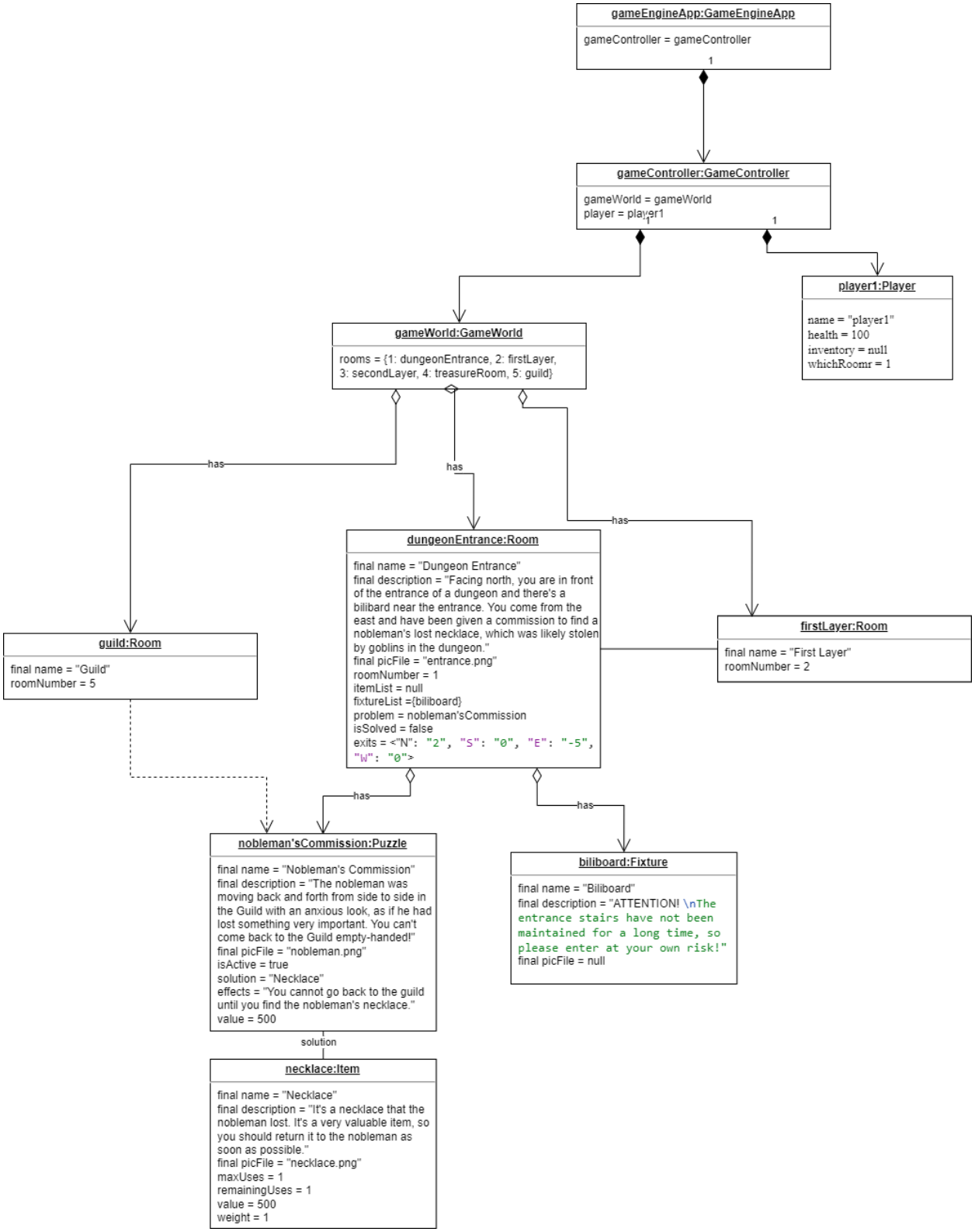
Scene 1

The adventure begins with the player arriving at the Dungeon Entrance, a dimly lit space with stone walls and a damp, musty smell. Looking around, they see two possible paths: heading north leads deeper into the dungeon, into an area called the First Layer, while going east leads to the Guild. Right in the middle of the room stands an old, weathered Biliboard, covered in faded notices. The player steps closer and reads the Billboard. It contains a warning about the dangers of the Dungeon, urging adventurers to be well-prepared before going inside. The descriptions of the lurking threats make the player uneasy—maybe heading straight into danger isn't the best idea right now. Instead, they decide to go east toward the Guild, hoping to find some help or better equipment first. But just as they reach the entrance, they're stopped. A stern-looking gatekeeper informs them that they can't enter without a Necklace—a requirement tied to something called the Nobleman's Commission puzzle. Unfortunately, the player doesn't have one. With no choice but to rethink their plan, they turn back, now needing to find this mysterious Necklace before they can proceed.

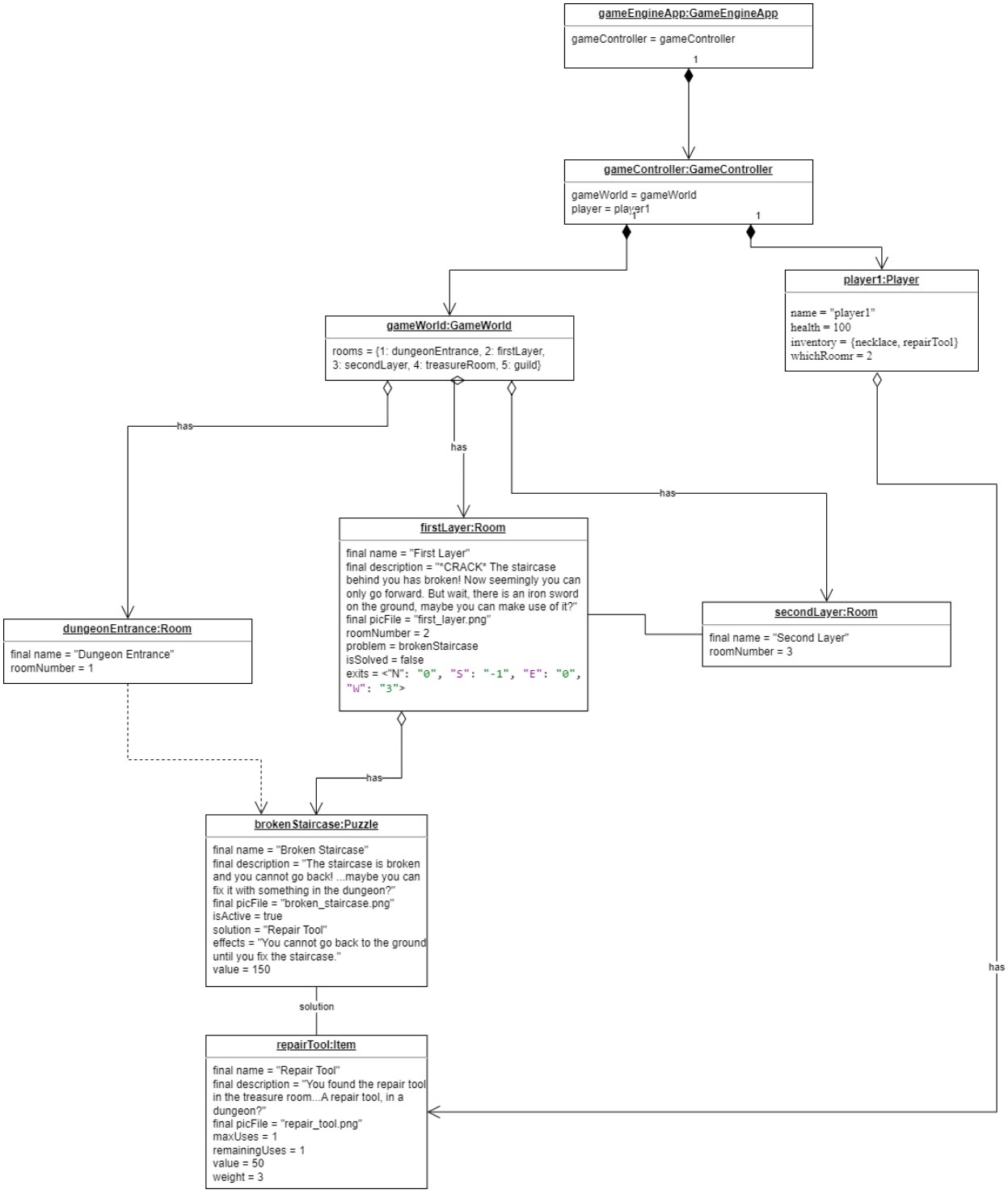
Scene 2

Determined to make progress, the player decides to explore further. First, they head north into the Second Layer of the dungeon. From there, they move east, returning to the First Layer, a place filled with the sounds of dripping water and distant, unsettling echoes. In the First Layer, they come across a major obstacle: the Broken Staircase. The stairs are completely unusable, making it impossible to move forward. But luckily, the player has a Repair Tool! They get to work, carefully fixing the damaged steps until the staircase is finally restored. Now, with a clear path ahead, they know it's time to wrap up unfinished business. With everything in place, the player backtracks south, returning to the Dungeon Entrance. This time, though, they have the Necklace—the key to solving the Nobleman's Commission puzzle. Presenting it to the gatekeeper, they're finally granted entry into the Guild. The Guildmaster, impressed by their perseverance, rewards them with a Gold Coin.

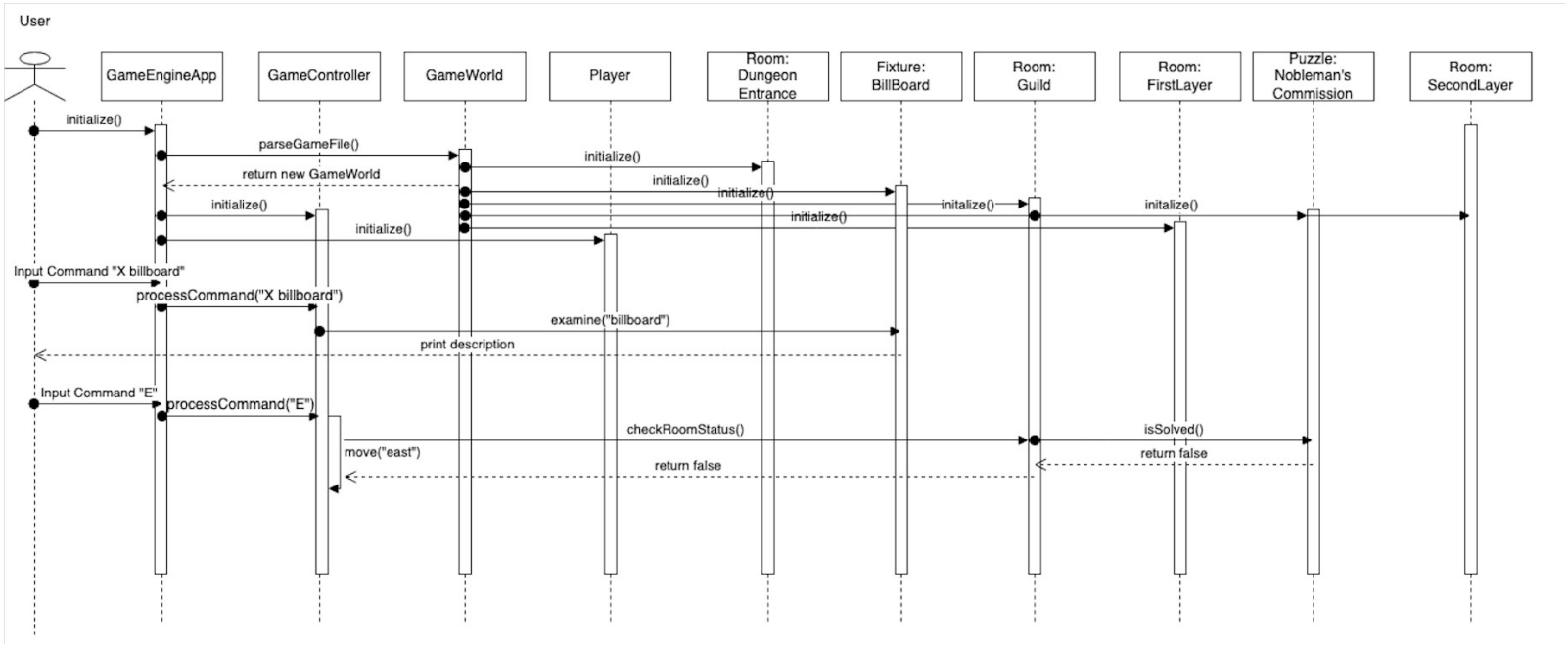
Object Diagram 1

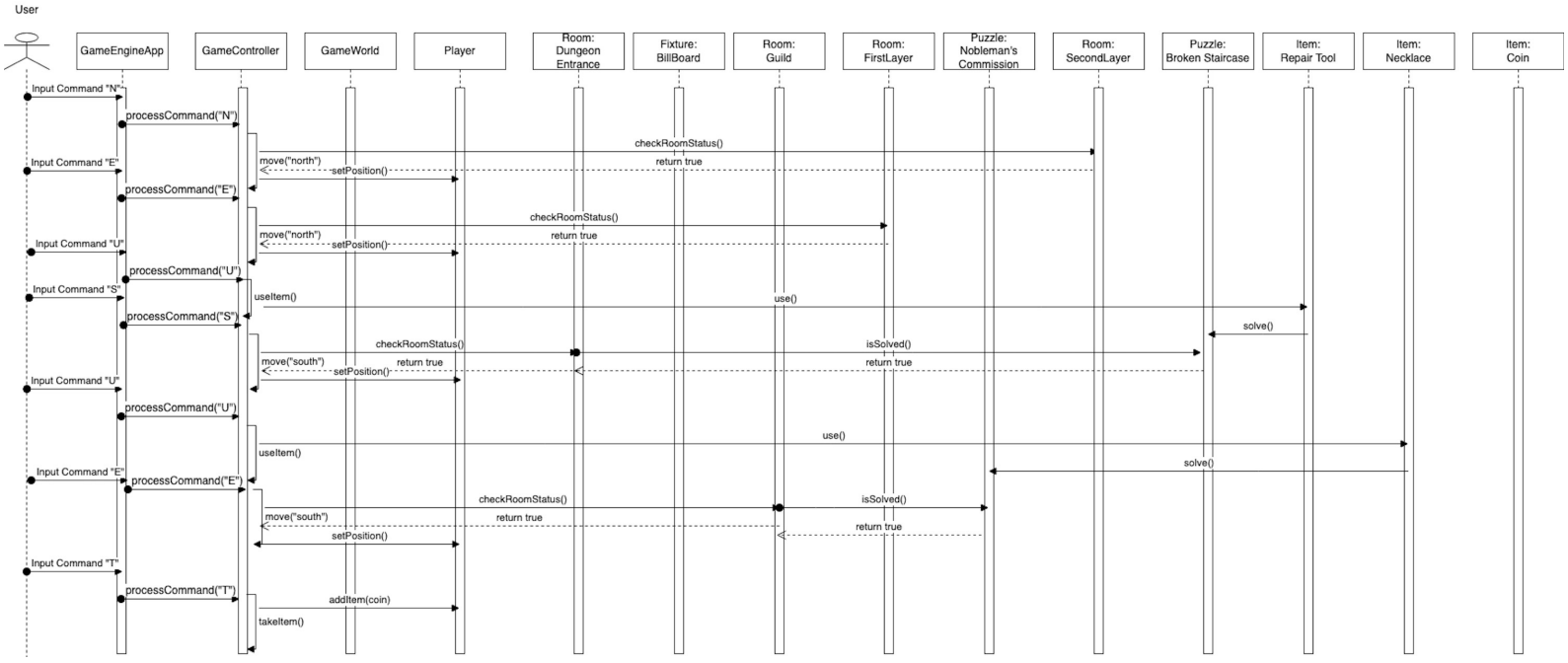


Object Diagram 2



Sequence Diagrams





Questions

Are we supposed to be utilizing singleton object mode and factory mode in this project? If so, how can we implement it?

What is the logic behind monsters and how often are they going to attack the player?

When should we construct rooms and its attributes/features? Should it be through the main function, creating them all at once, or construct them just when a player steps into a room?

Reference

Object Diagrams | Unified Modeling Language (UML) - GeeksforGeeks

Unified Modeling Language (UML) Diagrams - GeeksforGeeks

uml - How to show "if" condition on a sequence diagram? - Stack Overflow

Examples of UML diagrams - use case, class, component, package, activity, sequence diagrams, etc.

UML Class Diagram Examples of Common Scenarios | EdrawMax