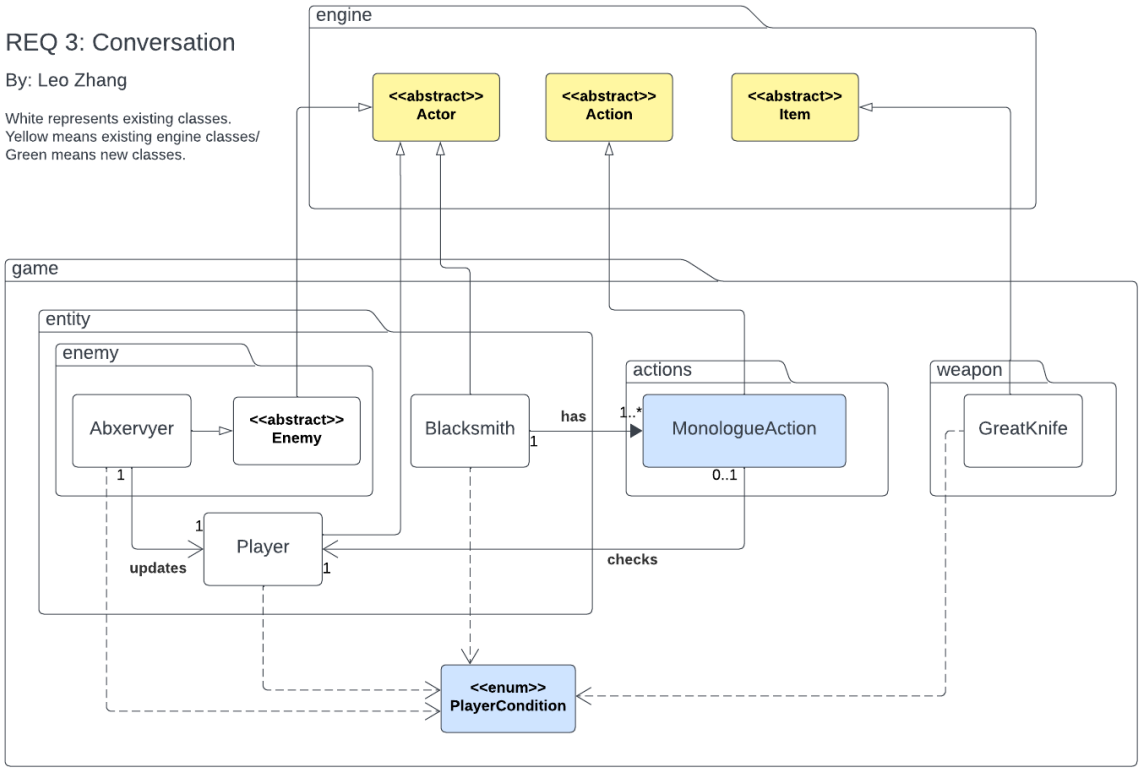
REQ 3: Conversation

By: Leo Zhang

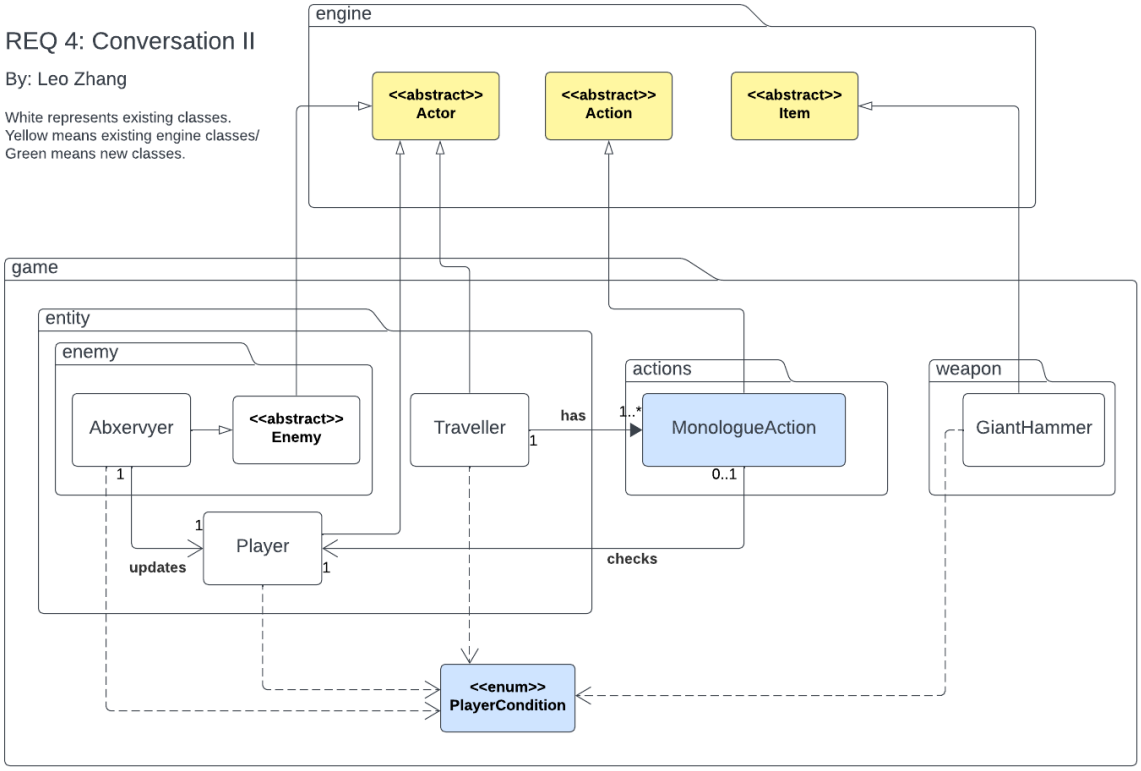
White represents existing classes.
Yellow means existing engine classes/
Green means new classes.



REQ 4: Conversation II

By: Leo Zhang

White represents existing classes.
Yellow means existing engine classes/
Green means new classes.



Overview

The above two are the UML diagrams for requirements 3 & 4, they have been combined due to similar functionality as the implementation is almost identical. New to the game with these requirements is the implementation of the MonologueAction class and the PlayerCondition enumeration, in addition to changes to the Blacksmith and Traveller classes as well as some minor tweaks to the Abxervyer and Player classes. These requirements are about implementing the functionality of letting the player listen to certain NPC monologues.

MonologueAction

The first thing to add in order to address these two requirements was to add an action handler class called MonologueAction. This extends Action and as a design choice this allows the system to be extensible in the future. Having this handle every single monologue-related action is good as it means that redundancy is reduced if this interaction is handled by the same class every time. This thus abides by the DRY principles by methods of abstraction.

MonologueList (Blacksmith & Traveller)

MonologueList is a method added to any actors who have monologue dialogue available for them. The reason this wasn't implemented in a different class was because this is only added towards certain actors and not every actor. It also is due to the fact that every actor which has monologues have varying conditions as well as varying monologue text and as such it makes more sense to put this method within the specific actor classes (in this case Blacksmith and Traveller). This puts less focus on dependencies within the classes and the MonologueAction (dependency inversion principle) and therefore suits SOLID principles.

Enums

Enums have been used (PlayerConditions) in order to act as flags for certain events happening within the game. Examples of this are conditions for if the player has a giant hammer or great knife in their inventory or if the Abxervyer boss has been defeated. This implementation is not the best, and limits extensibility if many of these conditionals need to be added in the future. However, due to the limited amount of these events it is suited here. However, in order to adhere to DRY principles, a more complex system with an overall events manager system which keeps track of these conditionals would be better for extensibility and for limiting redundancy within the system. (As is, it could end up there is a lot of copy-paste code and this would prevent that).

This implementation of the requirements generally also has strong dependencies as the actor classes and the MonologueAction classes have methods which reference each other constantly. This is good as it means that the system is not overly complex, but it does mean that changes in one could result in changes in another.