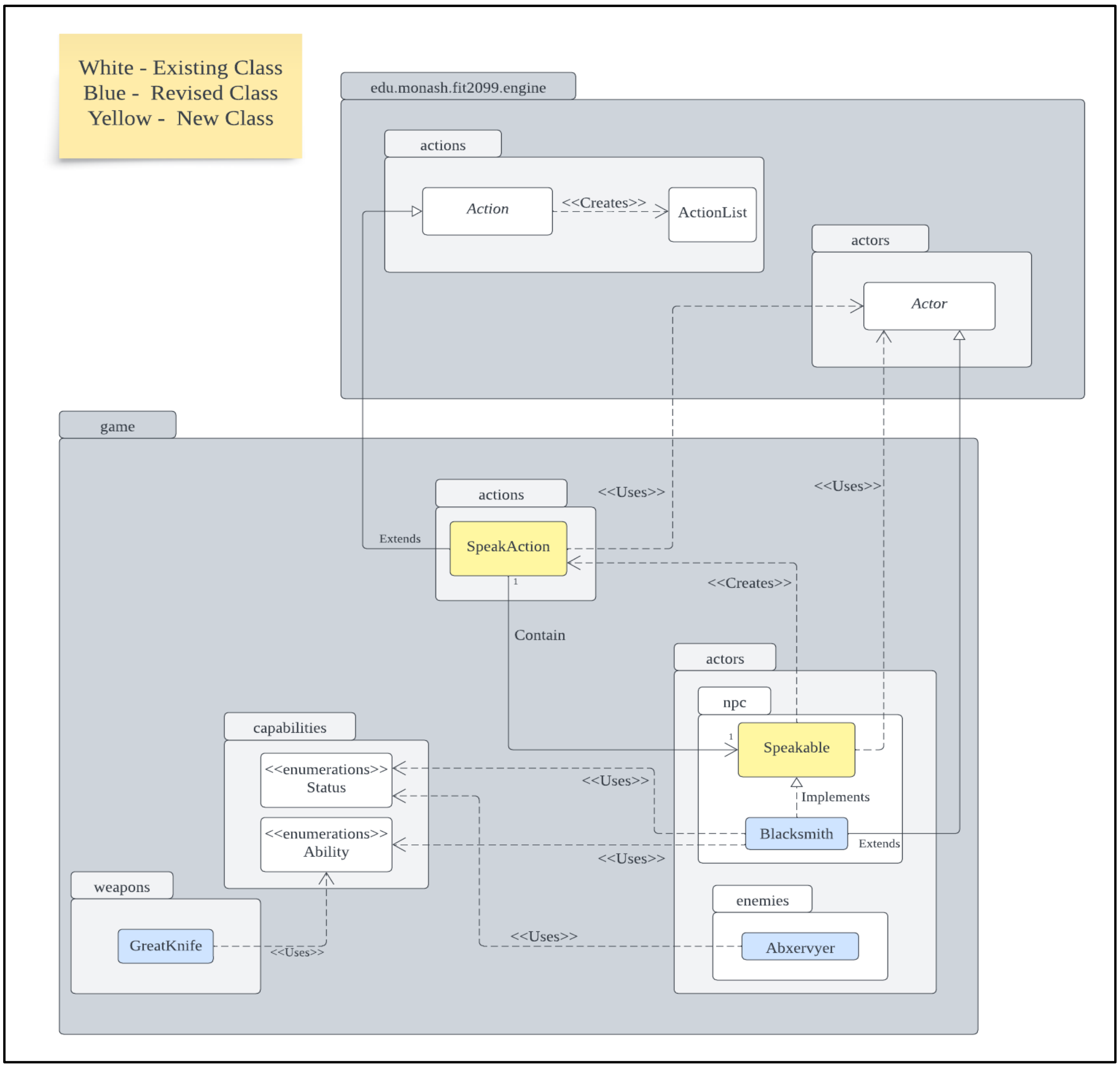
FIT2099 Assignment 3 Design Rationale

Requirement 3:



The Speakable interface was created with the ISP in mind as we wanted to prevent unnecessary method inheritance for actors that do not talk to the player. This class contains a speak method that takes in an Actor object that listens to the speaker. By creating a separate interface for the speak functionality instead of bloating all actor classes with a speak method, we ensured that only actors that need to speak will implement this interface. As a result, the possibility of there being an empty speak method in any Actor subclass (due to these actors not being able to speak to the player) is removed and this eliminates a common code smell that indicates that the

LSP has been broken: Empty, do-nothing implementations of one or more methods in subclasses. Any actor that implements the speak method of the Speakable interface can be substituted in the SpeakAction class so the current design is flexible enough to handle other NPCs that might need to speak in the future.

The SpeakAction class is a subclass tat extends the Action class. This was done to align with the SRP: the sole objective of this class is to execute the action of speaking when chosen. The SpeakAction class depends upon abstraction (Speakable interface) and not upon a concrete class (Blacksmith class) so that the DIP is met. The advantage of this is that this class can be used with any actor that is Speakable.

In the Blacksmith class, an ArrayList is used to store the possible monologues that the blacksmith might speak to the player. A monologue is chosen at random from this list and returned whenever the speak method is invoked. Certain monologues are added to this list based on certain conditions that depend on the player's state. For example, if the player has killed the Abxervyer, the monologue "Somebody once told me that a sacred tree rules the land beyond the ancient woods until this day." is added instead of "Beyond the burial ground, you'll come across the ancient woods ruled by Abxervyer. Use my creation to slay them and proceed further!". The condition checks are performed by validating the existence of certain capabilities (enums) associated with the player. Direct type checking was an alternative, but this would violate the OCP as we would have to add more type checking if conditions which a bad design in terms of extensibility due to tighter coupling. Every time we introduce a new type, we might need to revisit and possibly modify the code that does the type-checking. Therefore, using direct type checking would have bypassed the benefits of polymorphism. With all these factors taken into consideration, the current design ensures that new monologues can be added easily without modifying the existing logic (adhering to OCP).

Pros:

- Modularity: The speaking functionality can be easily integrated with other NPCs.
- Flexibility: The design is flexible enough to add more monologues, conditions for certain monologues and new speaking game Actors.
- Maintainability: The current design has clear separation of concerns which means that updating the speaking logic does not require modifying every class.

Cons:

- <u>Complexity and Overhead</u>: The current design has been implemented with the future extensibility and maintainability of the code in mind, so there exists an overhead due to the interface and classes being overkill if the monologue functionality is basic.

Future extensions support:

- More NPCs with unique monologues and conditions can be easily added using the current design.
- A more complex dialog system can be implemented where the chosen monologue changes the player’s or game state.