Requirement 3

Assumptions: we assume that a monologue item has at least one monologue message that they can play when the player chooses to listen to it

The design diagram represents the implementation of Astley, an AI device that has monologues that players can listen to.

Instead of having a superclass that handles all types of objects (items, grounds, and actors) that can monologue, objects that can monologue would implement the MonologueCapable Interface. This approach adheres to the Liskov Substitution Principle as it allows any object implementing the MonologueCapable interface to be used interchangeably, ensuring that they can be replaced with instances of the interface without affecting the correctness of the program. Meaning that if the game were to introduce more objects that have a monologue function, our design would accommodate for it. A disadvantage of the current design would be that if the game were to introduce more MonologueCapable objects, it can lead to a lot of code repetition since these objects share a few of the same method implementations. An alternative would be to have a parent class for each game entity (e.g., MonologueItem, MonologueGround, MonologueActor) so that methods can be abstracted. However, this would introduce a lot more complexity into the design with multi-level inheritance.

Through abstraction, the MonologueAction has an association with the MonologueCapable interface rather than individual usable objects (Dependency Inversion Principle). This promotes extensibility and reusability as different types of entities; actors, items, and ground can all use the MonologueAction without any adjustments needing to be made on the existing code.

An enumeration class was created to store all the monologue messages. This class was created so that in the future, if any of the other MonologueCapable objects share the same monologue in their monologue set, we can simply directly add the enumeration instead of rewriting the whole entire monologue (DRY).

The SubscriptionManager class contains the subscription logic. In the future, if there are objects that require subscriptions, we can instantiate the SubscriptionManager class and call its method to manage the subscription process. This approach helps to avoid code repetition (DRY). Furthermore, it also adheres to the Single-Responsibility Principle as all the subscription logic is handled by this class instead of individual objects. However, a drawback is that the current design is tailored specifically to Astley's subscription needs. If future objects with differing subscription payment requirements emerge, the SubscriptionManager would need alteration to accommodate them. We did consider making the method more flexible, but it is difficult given that we are unsure of what other subscription logic might arise in the future.