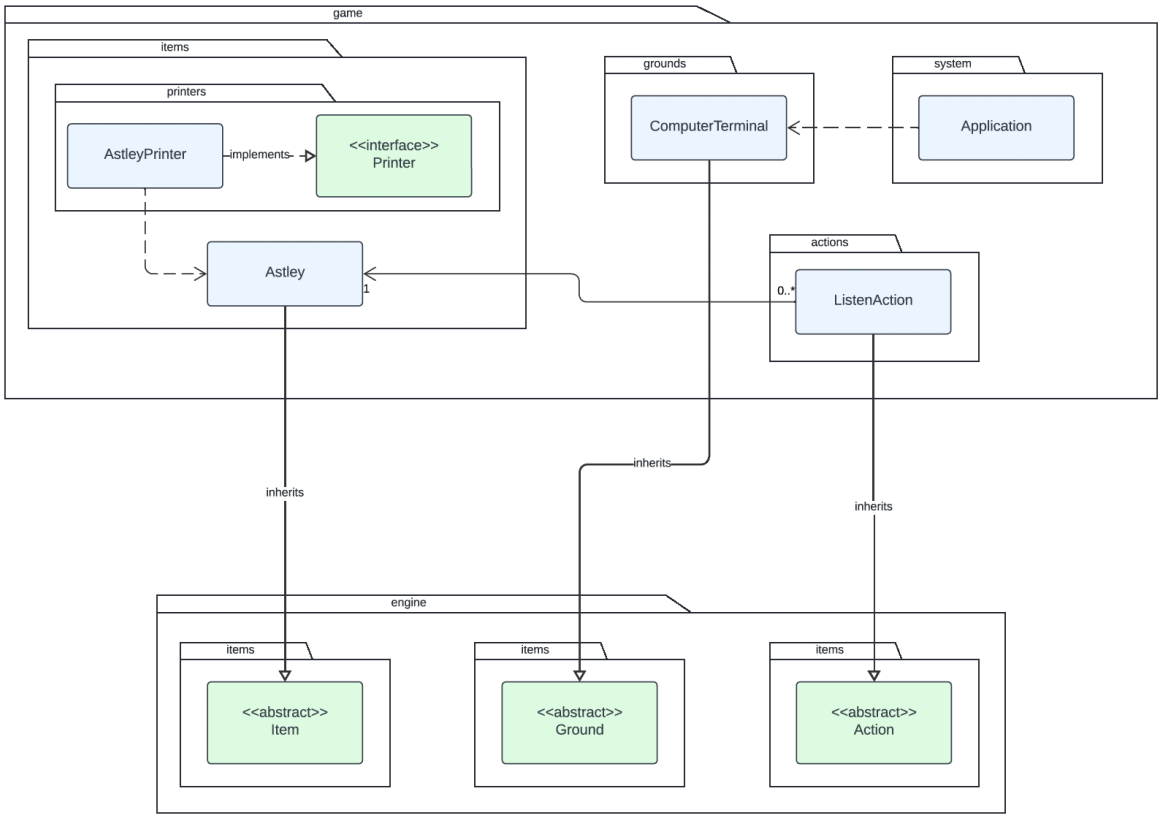
FIT2099 Assignment 2 Design Rationale
Requirement 3: dQw4w9WgxcQ

UML Diagram:



Design Goal:
The design goal for this requirement is to enhance the gameplay experience by allowing an intern to purchase and interact with an AI device, Astley, which provides a subscription service for the Player in exchange for the ability to play its monologue. This implementation aims to promote clean code practices such as simplicity, abstraction, and modularity, ensuring maintainability and extensibility by adhering to best coding practices.

Design Decision:
To implement the AI device, Astley, the decision was made to create a new class ‘Astley’ bby extending it from the existing ‘Item’ class within the game engine. This class handles the subscription fee logic and provides randomised monologues based on the specific conditions of the actor's state. An additional ‘AstleyPrinter’ class is introduced to manage the creation and pricing of Astley devices to increase abstraction, so that the ‘ComputerTerminal’ class does not need to be directly dependent on the ‘Astley’ class . Furthermore, a new ‘ListenAction’ class allows actors, such as the Player to interact with Astley and listen to its monologues. This design utilises the advantages of inheritance and composition to integrate the new functionality seamlessly into the existing game structure.

This approach allows for the abstraction of the Astley AI device's functionality, promoting extensibility and maintainability. This makes it so that future enhancements or new AI devices can be easily integrated into the existing code structure by extending the ‘Item’ class and implementing similar actions.

Alternative Design:
An alternative approach could involve handling the listening mechanism within the Astley class, eradicating the need for a separate ListenAction class. This may lead to a significant decrease in the maintainability, readability and extensibility of code.

Analysis of Alternative Design:
The alternative design is not ideal because it violates various Design and SOLID principles:

1. **Single-Responsibility Principle (SRP):**
    - This would violate the SRP as the Astley class would be responsible for multiple behaviours, including managing it's subscription lifecycle, storing its state (such as dropped/in inventory) as well has handling the listening mechanism. Instead of focusing solely on the AI device's properties and state, the Astley class would also need to manage the actions related to listening, leading to a class with more than one reason to change.
2. **Open-Closed Principle (OCP):**
    - Astley class would need to be modified to add new listening behaviours or change existing ones. For example, if you wanted to add new types of interactions with Astley (e.g., an 'AnalyseAction'), you would have to alter the Astley class each time, which contradicts the principle of extending behaviour via new classes rather than modifying existing ones.
3. **Dependency Inversion Principle (DIP):**
    - Higher-level modules (e.g., the game's action handling mechanism) would depend on the concrete implementation of the Astley class for the listening behaviour. Rather than depending on an abstraction (like an Action interface), the game would directly depend on the Astley class for the listening action, which would make it harder to swap out or change implementations without modifying the dependent classes.

Final Design:
In our design, we closely adhere to SOLID principles to ensure code quality and future extensibility:

1. **Single-Responsibility Principle (SRP):**
    - **Application:** Each class has a clear and specific role. The 'Astley' class handles the AI device's behavior, 'AstleyPrinter' manages the creation and pricing, and 'ListenAction' facilitates the listening interaction between the Player and the device.
    - **Why:** This allows for the abstraction of code, which promotes maintainability and extensibility by isolating responsibilities.
    - **Pros:** Improves maintainability and requires careful design to avoid overlap in class responsibilities.
    - **Cons:** May result in many classes if they don't share exact features, necessitating specific methods for each class.
2. **Open-Closed Principle (OCP):**
    - **Application:** New features and additional AI devices can be implemented through extensions of the 'Item' class and similar patterns without modifying existing code.
    - **Why:** Enhances flexibility and makes it easier to accommodate future changes.
    - **Pros:** Improves flexibility and extensibility, making it easier to include new features.
    - **Cons:** May increase complexity if not managed well.
3. **Liskov-Substitution Principle (LSP):**
    - **Application:** Objects of a superclass are replaceable with objects of its subclasses. For example, 'Astley' can be substituted with other items extending 'Item'.

- **Why:** Allows for seamless substitution of different AI devices without changing existing code.
- **Pros:** Renders the system more modular and extensible, simplifying maintenance.
- **Cons:** Can lead to overgeneralization if not careful.

Conclusion:
Our chosen design provides a robust framework for enhancing gameplay while adhering to best coding practices. By following SOLID principles and good design practices such as simplicity, modularity, and abstraction, we have developed a solution that is both efficient and scalable, paving the way for future game enhancements and feature integrations.