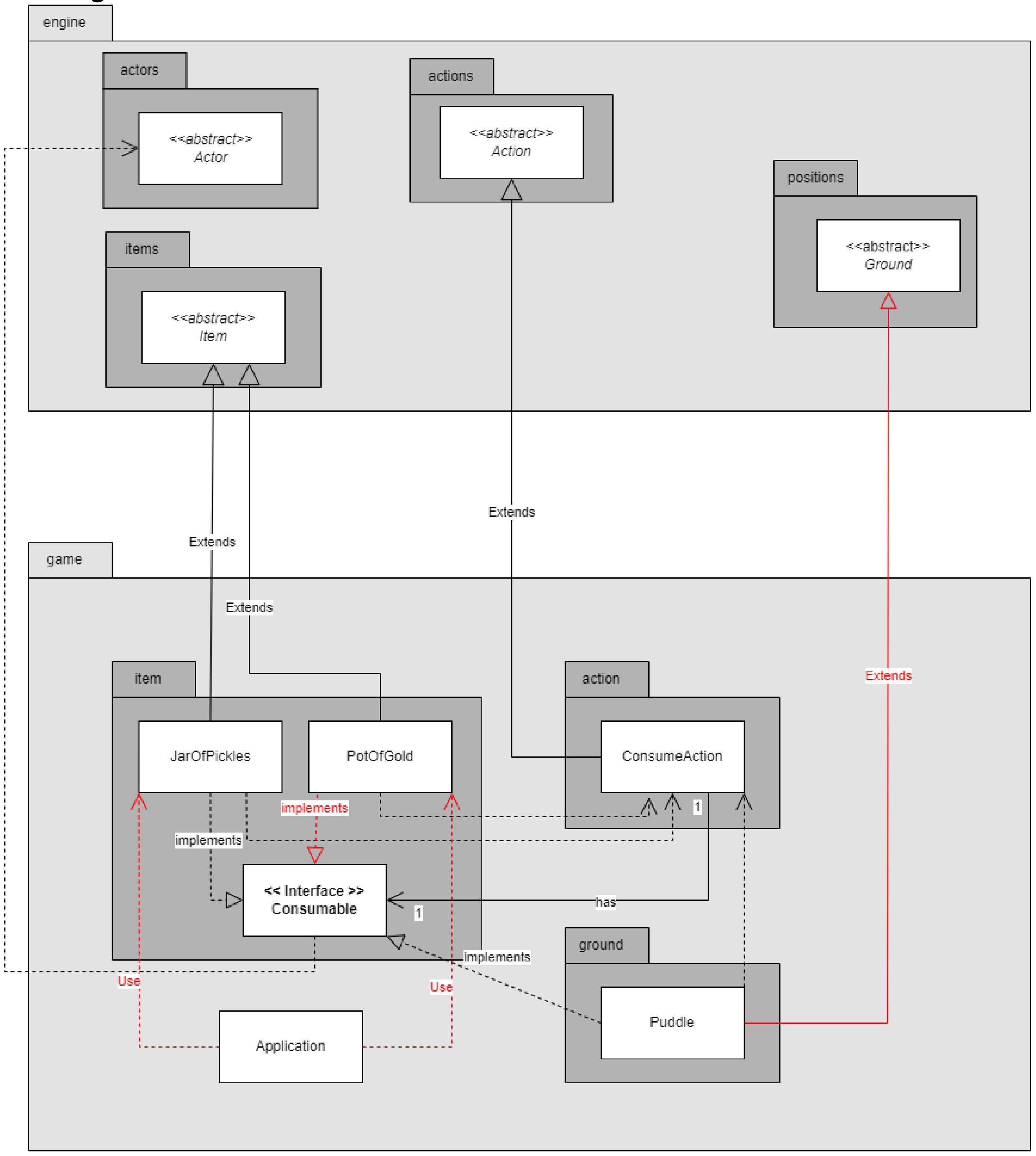
Assignment 2 Design Rationale

Requirement 3: MORE SCRAPS

UML Diagram:



Design Goal:
The design goal for this assignment is to incorporating two new items (Jar of Pickles & Pot of Gold) and a new drink water from puddle features while adhering closely to relevant design principles and best practices.

Design Decision:
In implementing two new items (Jar of Pickles & Pot of Gold) and a new drink water from puddle features, the decision was made to follow the SOLID and DRY principle as many as possible. The items and features added are implemented in different class, where each is responsible for its specific functionality. This design follows the single responsibility principle for each class, so the code is easier to understand, modify and maintain. The use of consumable interface also allows for easy extension and modification without affecting other part of code.

Alternative Design:
One alternative approach could involve implementing all item behaviour in a single class, managing different interactions through conditional statements. However, this design will violate the SOLID and DRY principle. The Single responsibility principle and Open/Closed principle will be violate, it would be hard to extend with new item behaviours.

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

1. **SRP:**
   - The SRP is broken by the alternative design, which assigns responsibility for several item behaviours to a single class. This complicates understanding, maintaining, and expanding the course.
2. **OCP:**
   - Since the alternative design cannot be extended without changing the current class, it is against the OCP. It would be necessary to change the class in order to add new item behaviours, which goes against the idea of "open for extension, closed for modification."
3. **DRY:**
   - The alternate approach may duplicate code to handle diverse item behaviours, which is against the DRY (Don't Repeat Yourself) concept.

Final Design:
In our design, we closely adhere to relevant design principles, including SOLID and DRY:

1. **SRP:**
   - **Application:** JarOfPickle, PotOfGold, Puddle
   - **Why:** There is just one duty associated with each item class (JarOfPickles, PotOfGold, Puddle) in relation to its unique item behaviour.
   - **Pros/Cons:** This encourages maintainability and clarity in the code.
2. **OCP:**
   - **Application:** Consumable interface
   - **Why:** makes it simple to extend item behaviours without changing already-existing classes.
   - **Pros/Cons:** By implementing the Consumable interface in new classes, new item behaviours can be added.
3. **LSP:**
   - **Application:** JarOfPickle, PotOfGold, Puddle.
   - **Why:** can be replaced with instances of their parent class (Item) without compromising the program's accuracy.
   - **Pros/Cons:** This promotes interoperability and extensibility.
4. **ISP**
   - **Application:** Consumable interface**.**
   - **Why:** divides the methods pertaining to item consumption from the main Item abstract class.
   - **Pros/Cons:** This lessens the effect of modifications and encourages code reusability by allowing classes to implement only the methods they require.
5. **DIP**
   - **Application:** Consumable interface.
   - **Why:** Rather than relying on specific implementations, the item classes rely on abstractions (the Consumable interface).
   - **Pros/Cons:** This promotes decoupling and testability by enabling flexibility in switching out alternative Consumable implementations without impacting the item classes.
6. **DRY**
   - **Application:** all
   - **Why:** By breaking out item behaviours into several classes, each performing a particular function, the approach prevents code duplication.
   - **Pros/Cons:** This promotes code reuse and maintainability.

Conclusion:
Overall, our chosen design provides a robust framework for implementing the Jar of Pickles, Pot of Gold, and Puddle features. By adhering to SOLID & DRY principles, we have developed a solution that is easy to understand, extend, and maintain, laying the foundation for future enhancements and optimizations.