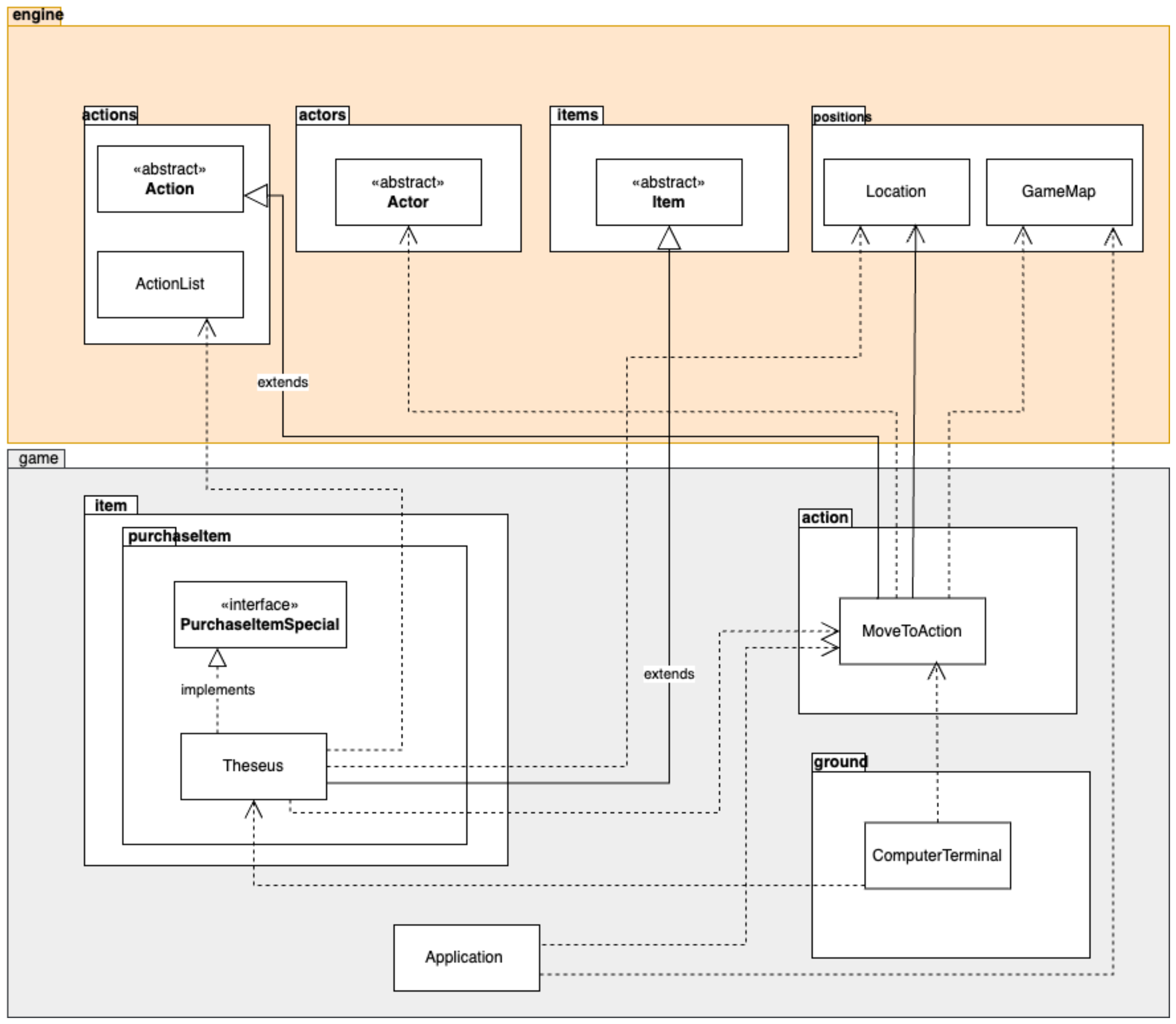
Assignment 3 Design Rationale

Requirement 1: **The Ship of Theseus**

UML Diagram:



Design Goal:

The design goal for this assignment was to enhance the gameplay experience by introducing new interactive elements and functionalities. Specifically, we aim to:

1. Implement the ability for the intern to travel between different locations (Factory, Polymorphia, and a new moon) using a computer terminal.
2. Introduce a new portable teleporter item, THESEUS, which allows for instantaneous movement within a single map.
3. Ensure that these new features integrate seamlessly with the existing game mechanics and adhere to relevant design principles and best practices.

Design Decision:

In implementing the Ship of Theseus feature, the decision was made to create a Theseus class to represent the portable teleporter and a MoveToAction class to handle the action of moving between different locations. Additionally, the existing ComputerTerminal class was updated to include the option for actors to purchase the Theseus item and to provide travel options between locations.

1. **Modularity and Encapsulation:** The ComputerTerminal class handles travel between locations and item purchases, encapsulating related logic. This promotes organization and reduces complexity, ensuring changes are localized and minimizing broader impact. While this design improves maintainability and readability, it may require refactoring if the class becomes too large.
2. **Scalability:** An arrayList of MoveToAction objects allows for easy addition of new travel destinations. This approach supports dynamic updates and future expansions without major code changes. The design is flexible and future-proof, though managing a growing list may necessitate performance optimization.

Alternative Design:
One alternative approach could involve integrating both travel and purchasing functionalities directly within the ComputerTerminal class itself. Instead of using separate MoveToAction and PurchaseAction classes, the ComputerTerminal would handle all actions related to traveling and purchasing. This would simplify the overall class structure but could lead to a more monolithic design.

Integrating all functionalities into the ComputerTerminal would simplify initial implementation but compromise modularity, scalability, maintainability, and flexibility, making it less suitable for a robust and extensible game design.

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

1. **Single Responsibility Principle (SRP):**
   - The alternative design violates the SRP by combining multiple responsibilities—travel and purchasing—into the ComputerTerminal class. This makes the class more complex and harder to manage.
2. **Open/Closed Principle (OCP):**
   - This design is not open for extension but closed for modification. Adding new actions or functionalities would require modifying the ComputerTerminal class, rather than simply extending the system with new action classes.
3. **Liskov Substitution Principle (LSP):**

- The proposed design may not adhere to the LSP, as it might introduce unexpected behavior or violate contracts when subclasses are used. For example, if there are variations in travel or purchasing actions for different types of terminals, the integrated design might not handle these variations appropriately.

Final Design:
In our design, we closely adhere to:

1. **Single Responsibility Principle (SRP):**
    - **Application:** Each class in our design has a single responsibility. For example, the ComputerTerminal class handles travel between locations and item purchases, while the Theseus class represents the portable teleporter item.
    - **Why:** Adhering to SRP promotes better code organization, readability, and maintainability. It ensures that classes are focused and cohesive, making them easier to understand and modify.
    - **Pros/Cons:** The benefit of this design choice is that it reduces complexity and makes it easier to identify and isolate changes. However, it may require more classes, potentially increasing initial development time.
2. **Open/Closed Principle (OCP):**
    - **Application:** Our design is open for extension but closed for modification. New functionalities, such as additional travel destinations or item types, can be added without altering existing classes. For example, new destinations can be added by creating new instances of the MoveToAction class.
    - **Why:** Adhering to OCP ensures that the system can be easily extended without risking the stability of existing code. It encourages modular design and supports future updates and expansions.
    - **Pros/Cons:** The advantage of this approach is its flexibility and adaptability to changing requirements. However, it requires careful planning and design upfront to define extension points effectively.
3. **Liskov Substitution Principle (LSP):**
    - **Application:** Subclasses adhere to the contracts established by their base classes. For instance, subclasses of the MoveToAction class must implement the required behavior for moving actors to different locations.
    - **Why:** LSP ensures that derived classes can be used interchangeably with their base classes without altering the correctness of the program. It promotes polymorphism and facilitates code reuse.
    - **Pros/Cons:** By adhering to LSP, our design promotes robustness and maintainability. However, it requires careful consideration of class hierarchies and contracts to prevent unexpected behavior.

Conclusion:

In conclusion, our design offers a robust framework for the game system, prioritizing flexibility and extensibility. By adhering to SOLID principles, especially the Single Responsibility Principle, we've ensured each class has a distinct purpose, fostering modularity and ease of future modifications. Separating concerns between the ComputerTerminal class and specialized action like MoveToAction enhances maintainability and scalability. Overall, our design lays a strong foundation for the game's development, capable of meeting current needs and adaptable for future enhancements and optimizations.