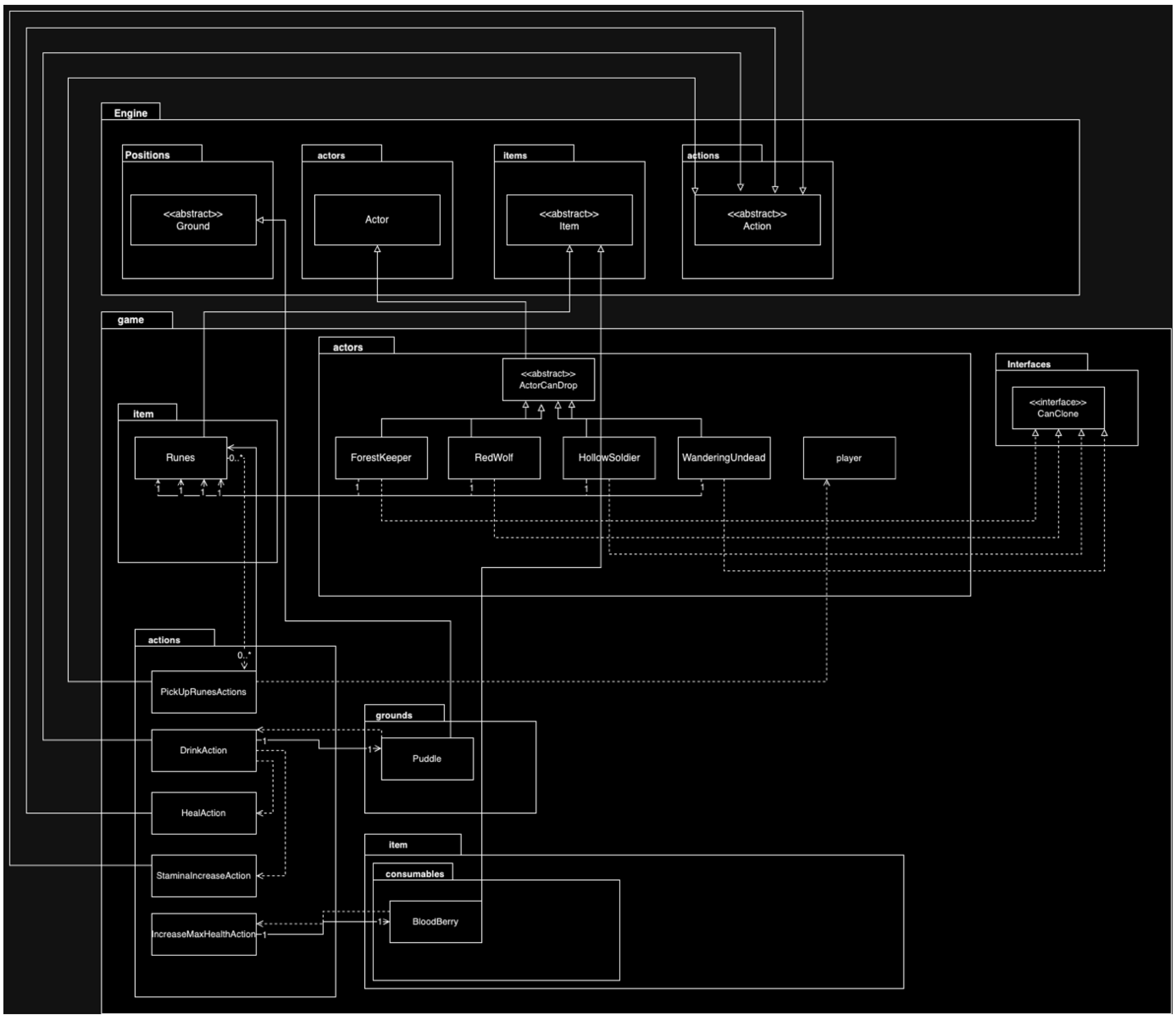
Requirement 2 Design Rationale



The code adheres to the DRY principle by encapsulating common functionality into reusable classes and actions. For example, the "DrinkAction" and "IncreaseMaxHealthAction" both encapsulate reusable behavior for different items.

SOLID Principles:

Each class adheres to the Single Responsibility Principle by focusing on a specific responsibility. For instance, the "Runes" class represents an item, the "DrinkAction" represents an action for drinking from a puddle, and the "Puddle" class represents a type of ground.

The design supports Open/Closed Principle by allowing the addition of new items, actions, or ground types without modifying existing code. For example, adding a new consumable item only requires implementing an "AllowableAction" for it.

Extensibility:

- **Advantage**: The design supports easy extensibility. To add new items, actions, or ground types, developers can create new classes implementing the relevant interfaces and follow established patterns. For instance, adding a new consumable item only requires implementing an "AllowableAction" for it.

Modularity:

- **Advantage**: The code is modular, with distinct classes for different game entities (e.g., items, actions, grounds). This modularization enhances code organization and maintainability.

Dependency Inversion:

- The design doesn't include complex dependencies, making it relatively simple. However, dependency injection could be applied more explicitly in some places to enhance flexibility further.

Advantages and Disadvantages:

- **Advantages**:
  - Encapsulation and reusability improve code maintainability.
  - Support for the addition of new items, actions, and ground types without modifying existing code.
  - Clear separation of responsibilities in classes.
- **Disadvantages**:
  - The code could benefit from additional documentation and comments to explain the design choices and the purpose of specific classes and methods.
  - The design doesn't include comprehensive error handling or exception management, which is crucial for robust game development.