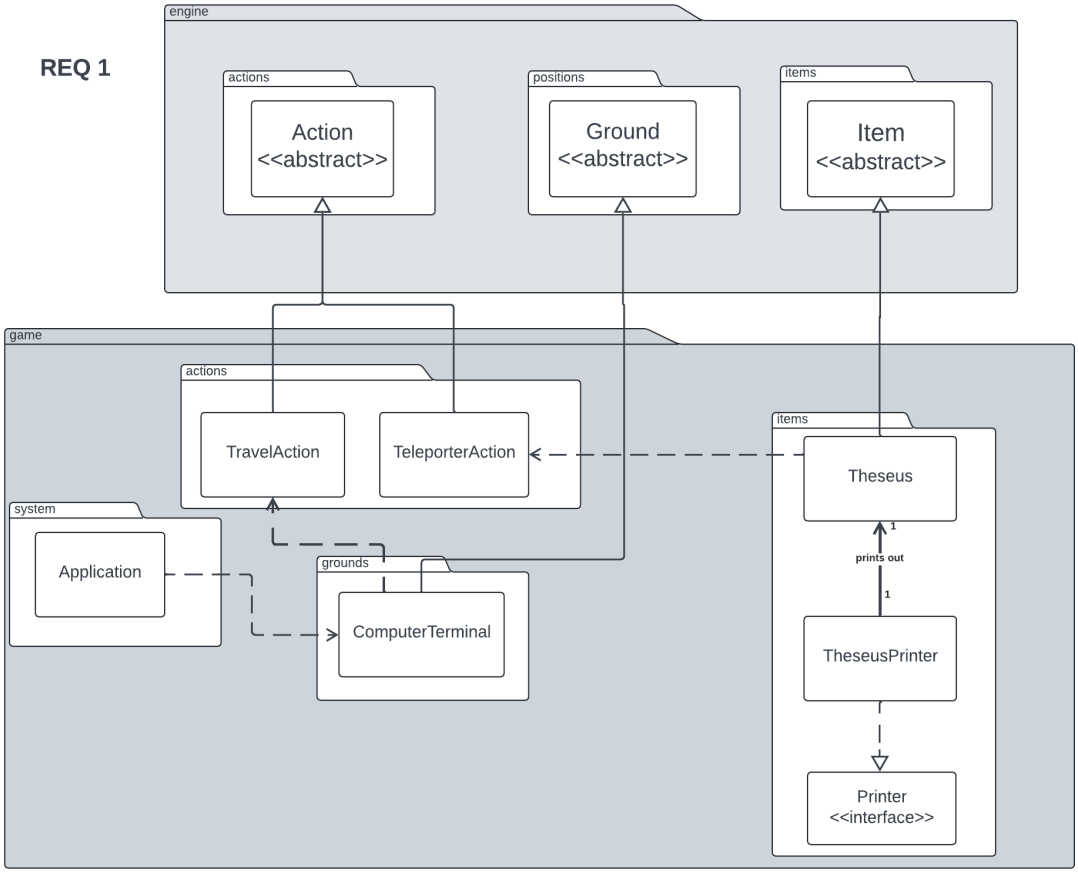
REQ1: The Ship of Theseus

Overview

This requirement aims to enhance the functionality of the ComputerTerminal in the game, allowing the intern to travel between different locations (the factory and the new moon) and purchase a new item called THESEUS, a portable teleporter. The UML diagram below illustrates the relationships and interactions between the core components necessary to fulfill this requirement:



UML Diagram Explanation

Action (abstract): Represents the base class for actions. TravelAction and TeleportAction inherit from this class.

Ground (abstract): Represents the base class for ground elements. ComputerTerminal inherits from this class.

Item (abstract): Represents the base class for items. Theseus inherits from this class.

TravelAction: Represents the action of traveling to a different location.

TeleportAction: Represents the action of teleporting within the same map using THESEUS.

Theseus: Represents the THESEUS item that allows teleportation.

TheseusPrinter: Implements the Printer interface to print THESEUS items.

Printer (interface): Defines the methods that printer classes must implement.

ComputerTerminal: Manages the actions related to the computer terminal, including purchasing items and traveling to different locations.

Application: The main class that sets up the game and initializes the necessary components.

Principles and Concepts

Single Responsibility Principle (SRP):

Each class is designed to handle one specific task. For example, the Theseus class focuses on managing the properties and actions related to the THESEUS item.

Open/Closed Principle (OCP):

The design allows for easy extension without modifying existing code. New features, such as additional items or travel destinations, can be added by creating new classes or extending existing ones. For example, adding a new item printer or travel destination can be achieved by creating new classes that implement the respective interfaces or extend the appropriate classes.

Liskov Substitution Principle (LSP):

The design ensures that derived classes can be substituted for their base classes without affecting the functionality. For instance, the Printer interface is implemented by various printer classes (e.g., TheseusPrinter), allowing them to be used interchangeably.

Interface Segregation Principle (ISP):

The design avoids large interfaces. Instead, it uses focused interfaces, such as the Printer interface, which includes only methods related to printing items.

Dependency Inversion Principle (DIP):

High-level modules depend on abstractions rather than low-level modules. For example, the ComputerTerminal interacts with the Printer interface, not specific implementations, allowing for easier extension and modification.

Connascence

Connascence of Type: By using interfaces such as Printer, the design minimizes direct dependencies on specific classes, making it easier to change implementations without affecting the system.

Connascence of Algorithm: Encapsulating logic for traveling and purchasing within their respective actions (TravelAction and PurchaseAction) ensures that changes to these algorithms do not impact other parts of the system.

Connascence of Name: Consistent naming conventions improve readability and maintainability.

Advantages of the Design

Modularity: The system is highly modular, with each class and interface having a clear responsibility. This makes the codebase easier to understand, maintain, and extend.

Flexibility: The use of interfaces and abstract classes allows for easy addition of new features without modifying existing code. For example, new items or travel destinations can be added by creating new classes that implement the respective interfaces.

Maintainability: Adherence to SOLID principles ensures that the system is easy to maintain. Each class is small and focused, making it easier to understand and modify.

Testability: The modular design and use of interfaces make the system easy to test. Each component can be tested in isolation, and mock implementations can be used to test interactions between components.

Disadvantages of the Design

Complexity: Using multiple interfaces and abstract classes can introduce complexity, especially for new developers who are not familiar with the design patterns used.

Overhead: The design may introduce some overhead due to the increased number of classes and interfaces. However, this is a trade-off for the benefits of modularity and flexibility.

Extensibility

Adding New Items: To add a new item, create a new class that implements the Printer interface and add it to the list of item printers in the ComputerTerminal. This does not require any changes to existing code.

Adding New Travel Destinations: To add a new travel destination, create a new GameMap for the destination and add a new TravelAction in the ComputerTerminal. This allows the intern to travel to the new destination without modifying existing classes.

Applying Connascence in the Design

Connascence of Type: Using the Printer interface allows us to add new printer types without changing the ComputerTerminal. This minimises dependencies and makes the code more flexible.

Connascence of Algorithm: Encapsulating travel and purchase logic within their respective action classes ensures that these algorithms can change independently without affecting other parts of the system.