Design Rationale for REQ1: Intern of the Static Factory

Objective:
In this requirement, the Intern's objective is to collect scraps (large bolts and metal sheets) on an abandoned moon.

Design Overview:
The design implements fundamental interactions of picking up and dropping off by integrating modified and existing classes to implement the Intern's tasks.

Classes and Responsibilities:

1. **Player (Intern)** - Controls player movement and interactions with items. Allows action for picking up and dropping off items.
2. **LargeBolt and MetalSheet** - Represent collectible items that are scattered in the GameMap.
3. **GameMap** - Manages the placement of interactive elements like walls, floors, and items, defining the playable area and obstacles.
4. **World** - Handles game initialization and progression, facilitating the game loop and interactions.

Interactions:
The Intern interacts with LargeBolt and MetalSheet items through actions managed by the **ActionList** class. The World and GameMap set up the game environment, enabling these interactions.

Design Justifications:

- **Modularity**: Utilizes high cohesion and low coupling to ensure that each class has a clear purpose and minimal dependencies, promoting easier future expansions.
- **Efficiency**: Adheres to the SOLID principle by leveraging existing game engine functionalities to manage item interactions and player movement.

Pros:

- **Scalability**: Designed to easily incorporate additional features in future updates.

Cons:

- **Engine Dependency**: Relies heavily on the game engine's existing features, which could restrict some future feature development.

Conclusion:

The design of both new classes is designed in a way that it is simple to implement new features in the future where perhaps the items can be used to fix the spaceship. It adheres Solid Principles when designing the classes.

Design Rationale for REQ2: The Moon's Flora

Objective:
In this requirement, the game introduces a unique flora called "Inheritree" that grows on the moon and evolves over time, producing fruits that serve as valuable resources for the player.

Design Overview:
The design focuses on creating and managing a lifecycle for the Inheritree, which undergoes growth stages and produces fruits that the Intern can collect.

Classes and Responsibilities:

1. **Inheritree** - Starts as a sapling and matures over time, capable of dropping fruits for every tick of the gameplay.
2. **SmallFruit and BigFruit** - Represents fruits produced by the sapling. These can be picked up and consumed by the player.
3. **GameMap** - Manages the placement of trees and fruits as the game progresses, facilitating the interaction between the Intern and these new items.
4. **World** - Manages the game environment, ensuring that the lifecycle of the Inheritree and the spawning of fruits are processed each tick.

Interactions:

- **Inheritree and GameMap**: The Inheritree interacts with its surrounding tiles via the GameMap to place fruits based on its growth stage.
- **Player (Intern) and Fruits**: The Intern can interact with both SmallFruit and BigFruit by picking them up or placing them, managed through the inventory system.
- **World and Inheritree**: The World class ensures that each tick influences the growth of Inheritree and the production of fruits, maintaining the game's lifecycle mechanics.

Design Justifications:

- **Dynamic Environment Interaction**: The Inheritree's ability to change over time and affect its surroundings introduces a dynamic element to the gameplay, enhancing engagement.

- **Use of Probabilities**: Implementing a chance-based mechanism for fruit drops ensures variability and unpredictability in resource availability, challenging the player's resource management strategies.
- **Clear Role Definition**: Each class has a specific role, such as the Inheritree for growth and fruit production, and the fruits for player interaction, which keeps the system modular and maintainable.

Pros:

- **Scalability**: Designed to easily incorporate additional features in future updates.

Cons:

- **Engine Dependency**: Relies heavily on the game engine's existing features, which could restrict some future feature development.

Conclusion:
The design of the three new classes is designed in a way that it is simple to implement new features in the future.

Design Rationale for REQ3: The Moon's (Hostile) Fauna

Objective:

In this requirement, the game introduces a hostile entity named Huntsman spiders that spawn from craters and attacks the player with its long legs when the player come in close with it.

Design Overview:

The design involves the integration of new game elements and behaviors that allow for the spawning, movement, and interaction of Huntsman spiders within the game environment, particularly in relation to the player character.

Classes and Responsibilities:

1. **Crater** - Acts as a spawn point for the Huntsman spiders, with a 5% chance of spawning a spider each game tick.
2. **HuntsmanSpider** - Represents the hostile creature that can attack the Intern when in range. It has specific behaviors such as wandering and attacking.
3. **Player** - The Intern character, now with enhanced interactions to manage health and react to attacks from Huntsman spiders.
4. **AttackBehaviour** - Manages the attack logic for Huntsman spiders, ensuring they only attack the Intern and within the defined accuracy.
5. **WanderBehaviour** - Controls the movement of Huntsman spiders when they are not in attack mode.
6. **GameMap** and **World** - These existing classes manage the environment and the overall game state, facilitating the interactions between the player and the spiders.

Interactions:

- **Crater and HuntsmanSpider**: Each crater checks per tick if it should spawn a spider, based on the specified probability.
- **HuntsmanSpider and Player**: Spiders will automatically attack the player if they are within one exit's distance, using the AttackBehaviour class to determine the outcome based on attack accuracy.
- **Player and Game Environment**: The player's health is constantly updated based on interactions with spiders, and health status is displayed on the screen. If health reaches zero, a message from FancyMessage is displayed.

- **World**: Executes the sequence of game actions, including spawning spiders, player and spider movements, and interactions.

Design Justifications:

- **Modular Design**: Each class has a well-defined role, enhancing maintainability and allowing for easy adjustments or expansions, such as adding different types of creatures.
- **Use of Behaviors**: Decoupling creature behaviors (attack, wander) from their class allows for flexible behavior assignment and easy modifications to creature actions without altering class code.

Pros:

- **Scalability**: The design supports the addition of more creatures and behaviors without significant redesigns, thanks to the modular and behavior-based architecture.

Cons:

- **Engine Dependency**: Relies heavily on the game engine's existing features, which could restrict some future feature development.

Conclusion:

The design can easily integrate new hostile creatures with unique behaviors into the game. The AttackBehavior class can also introduce different behavior patterns for new creatures.

Design Rationale for REQ4: Special Scraps

Objective:

In this requirement, the game introduces special scraps named metal pipe which can be considered as a weapon which the player could pick up and use. Besides, it also introduces a new usage of fruits gained from the inheritree that can heal the player when being attacked and damaged by the hostile creatures.

Design Overview:

The design is to integrate special scraps into the game's existing framework, allowing the Intern to use these items for defense and health recovery using fruits.

Classes and Responsibilities:

1. **MetalPipe** - Serves as a weapon that can be used to attack hostile creatures, with specific damage and accuracy.
2. **Player (Intern)** - Enhanced to handle new interactions like using the MetalPipe for combat and consuming fruits for health recovery.
3. **HealingAction** - Manages the healing process when the Intern consumes fruits, adjusting health accordingly.
4. **SmallFruit and BigFruit** - Act as consumable items that the Intern can use to regain health. Each type of fruit provides a different amount of healing.
5. **GameMap** - Continues to manage the placement and interaction of items and actors within the game environment.
6. **World** - Ensures that game logic concerning new item interactions is correctly processed each game turn.

Interactions:

- **Player and MetalPipe**: The Intern can pick up the MetalPipe and choose to use it as a weapon against hostile creatures when in range.
- **Player and Fruits**: When fruits are in the Intern's inventory, they can be consumed using the HealingAction, which increases the Intern's health based on the type of fruit.
- **HealingAction and Player**: Directly interacts with the Player class to apply health changes when fruits are consumed.

Design Justifications:

- **Modular Item Management**: The implementation of special scraps as items that can be picked up, dropped, used for attacks, or consumed for healing keeps the game's inventory management system flexible.
- **Integration with Existing Structures**: Utilizing existing game mechanics for item interactions ensures that new features do not disrupt the foundational gameplay loop.

Pros:

- **Scalability**: The current design supports the easy addition of more special scraps and their varied interactions, which can be extended to trading mechanisms in future assignments.

Cons:

- **Engine Dependency**: Relies heavily on the game engine's existing features, which could restrict some future feature development.

Conclusion:

The design can easily integrate new weapons into the game where the player can pick up into the inventory and use it for combat. The tracking method within the weapon helps to determine the attack direction of the weapon to the hostile creatures.