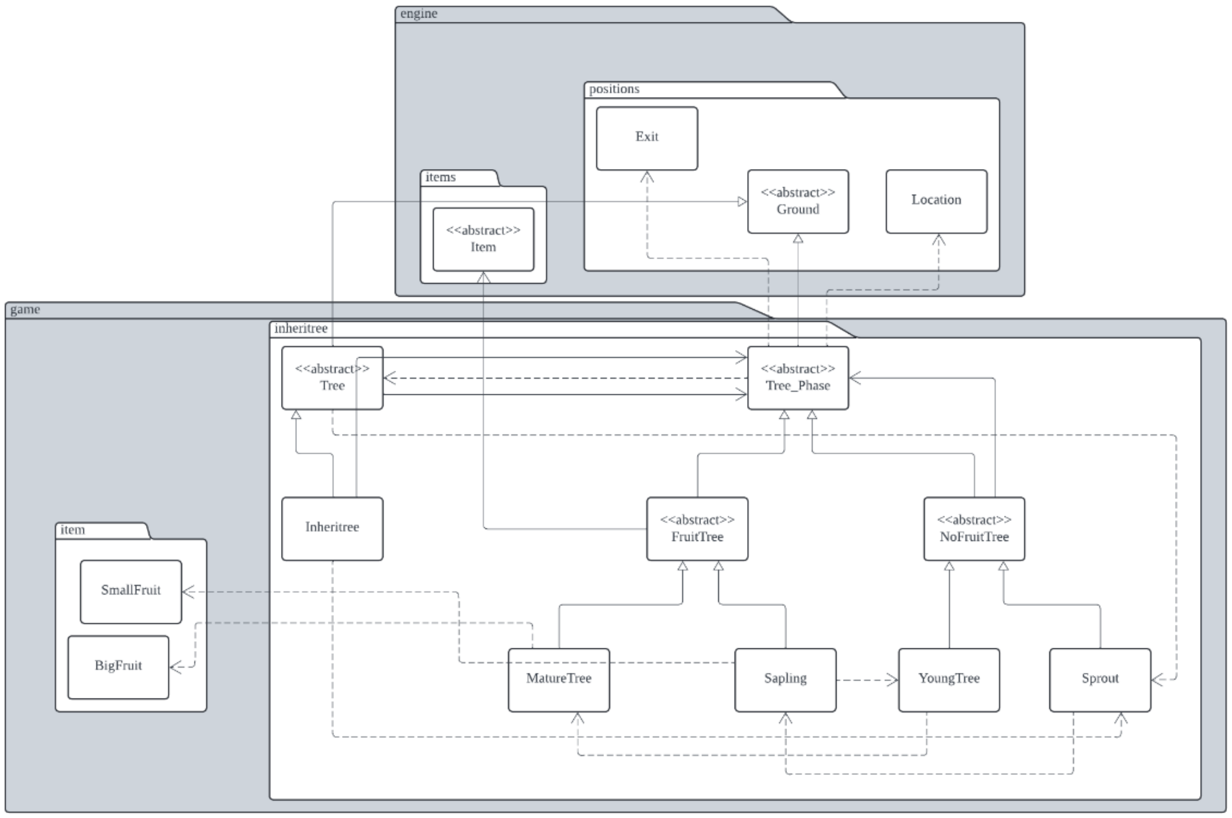
Assignment 3 Design Rationale

Requirement 4: [Survival Mode] REQ4: Refactorio, Connascence's largest moon

UML Diagram:



Design Goal:

The design goal for this assignment is to implement Inheritrees that progresses through 3 different stages before reaching maturity while adhering closely to relevant design principles and best practices. The new stages that an Inheritree can progress through once this requirement is implemented are sprout, sapling, young tree, and mature tree.

Design Decision:

In implementing the different stages of the Inheritree, the decision was made to create a new abstract class named Tree_Phase, which serves as a base for the different tree stages. Next, 2 abstract classes named FruitTree and NoFruitTree are created, which serves as the base for tree phases that produce fruits and for those that do not produce fruits. A separate Tree abstract class is also created as the base for different tree types, such as Inheritree. The approach was chosen as it provided abstraction to the different components of the requirement. The tree stages that produce fruits will extend from the FruitTree abstract class, where the fruit spawning code can be written in the parent

FruitTree class, reducing the amount of redundant code that would need to be written in the children class, adhering to the DRY principle that was taught in this unit.

Alternative Design:
One alternative approach could involve writing the stages of the tree as individual classes that do not inherit from a parent class. For example, the Mature and Sapling stages of the tree both can produce fruits, without extending from a parent class, the fruit spawning code will be repeated in the classes. If more stages of the tree were to be added that produced fruits, the code will be hard to read and more complex.

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

1. **DRY:**
    - The alternative design would require repeats of different code components that would make it harder to read and maintain the source code.
2. **SRP:**
    - In the alternative design, each stage that produces fruits will have to manage the spawning mechanic as well as the aging mechanism, which violates the single responsibility principle.

Final Design:
In our design, we closely adhere to the DRY and SOLID principles.
1. **DRY:**
    - **Application:** The code components that are frequently being reused are placed into abstract classes, from which child classes can inherit the methods.
    - **Why:** By adhering to the design method, I can reduce redundant code and improve the maintainability of the code.
    - **Pros/Cons:** Improved code readability, code is easily understood by new developers.
2. **SRP:**
    - **Application:** By separating the classes and using inheritance, each class only has to perform one function.
    - **Why:** Code becomes easy to read as there is not a single class with excessive methods that are unnecessary.
    - **Pros/Cons:** The source code is not complicated to read and is easy to maintain.

Conclusion:

Overall, our chosen design provides a robust framework for implementing the different tree stages, while also providing a platform for future tree types to be created. By carefully considering the design principles that were taught in this unit, we have developed a solution that is scalable and easily maintainable, paving the way for future extensions to the source code.