Design Rationale

REQ 2:

Inheritree

To avoid repetition, the child class of Inheritree only needs to provide all the information about what characteristics/attributes it should have to the parent class. The parent class will manage all the logic of the Inheritree, since all inheritree operate similarly with minor differences such as age, which can be easily set by using the parent constructor. The downside of this implementation is that, we need to set up all the constant values for each tree, to be passed into the parent constructor.

Inheritree also implements Plant interface, so that it has all the methods a plant is expected to have.

Immature to Mature Inheritree

The logic behind the design is every tree in any phase of an Inheritree are Inheritree, so Inheritree is a parent class which trees from different phases should inherit from. To implement how a tree from one phase grows into a tree of another phase, an attribute is used to store the tree to grow into. This idea is similar to chaining trees from different phases together. This allows trees to change from one to another, and operate accordingly without using if-else statement to decide what the tree is supposed to do at each phase. But there will be additional association between concrete tree classes, to enable the chaining.

Fruit Dropping

Each Inheritree stores its own fruit that it drops, it can be large or small fruit. By doing this, there is no need for if-else conditions to check which Inheritree it is to determine which fruit to drop. This again follows the single-responsibility principle, where each child class is responsible for storing the information of what fruit to drop. The downside of this implementation is that, we still need to instantiate the concrete class of fruit inside the concrete class of Inheritree, this introduces additional association between the 2 concrete classes.

Fruit:

The 2 variations of fruit: large fruit and small fruit, extend the same abstract class, which is Fruit. By doing this, new fruit can be easily added to the game by extending Fruit. This helps to abstract away the actual implementation of each and every concrete class, and the client class that used them only needs to know on an abstract level what a Fruit

can do. Dependency Inversion is achieved, where the client class uses the abstract class instead of the concrete class, and each concrete class of fruit extends the abstract class.

Spawnable and Spawner:

In order to enable the Inheritree to spawn drop/spawn fruit, it must implement Spawner interface. This is an interface for game entities that can spawn something around them. By doing so, any future entities that are expected to be able to spawn something can just implement this interface to enable the spawning capabilities.

Spawnable entities are entities that can be spawned. These entities will implement CanSpawn interface, which provides them with a method to spawn at a given location.