Design rationale for Requirement 3:

Requirement 3 entails the creation of the Crater class, which operates similarly to the Inheritree class but spawn actors (enemies) instead of fruits. As with Requirement 2, I incorporated a CloningFactory to handle the cloning of the actors spawned by the Crater. Following the principle of DRY, I avoided the need to create separate ActorFactory and ItemFactory components by utilizing the CloningFactory for actor instantiation.

In the implementation of the Crater class, I employed a static CloningFactory to facilitate its usage with the FancyGroundFactory. This decision enables the Crater class to be initialized with constructor with parameter, thereby updating the static field and allowing the default constructor to spawn different types of actors. In anticipation of potential future requirements involving Craters spawning multiple types of actors, I will design the Crater class to potentially store an ArrayList of CloningFactory instances instead of just one in future.

For the HuntsmanSpider actors spawned by the Crater, which are NPCs tasked with attacking the player (designated with the capability Status.HOSTILE_TO_ENEMY), I opted to create an AttackBehaviour class extending from Behavior to manage their automatic attacks. This design adheres to the Single Responsibility Principle, ensuring that the Behavior class is responsible solely for defining the actions performed by NPCs.

To prevent HuntsmanSpider actors from entering the spaceship, I implemented access restrictions to the floor using the Ability.SPACESHIP_TRAVELLER ability. This approach ensures that only actors possessing the ability to travel in the spaceship can access the floor, rather than solely relying on the Status.HOSTILE_TO_PLAYER attribute to restrict their entry. This decision accounts for potential future requirements that may introduce other actor types permitted to enter the spaceship.