**Design Rationale for requirement 3: Enemies**

All changes made to the existing UML diagrams to get the submitted UML diagrams are highlighted and justified in this document. Each class has its own behaviours and serves different purposes. This implements Single Responsibility Principle (SRP). The pros are the code is simpler and shorter and the program is easier to maintain. However, repetition of code may occur, and this violates DRY principle.

In the assignment, we are inheriting attributes and methods from abstract class. By extending an abstract class, DRY principle is implemented. This means codes can be reused and can be shared between classes that have common traits. As a result, codes do not need to be re-written, resulting in a simpler and shorter codebase.

* A new class Koopa was created, and several types of relationships are implemented for this class. (Includes changes made to abstract class Tree)  
    
  **Justifications:**
  + This class was created to represent the other type of enemy called ‘Koopa.’ It shares similar methods and attributes to that of existing enemy type Goomba.
  + The class Koopa is also linked to the abstract Class ‘Actor’ as it is part of the game as an actor.
  + The class Koopa is linked to the interface Behaviour as it could potentially implement similar behaviour as other actors. By implementing an interface, related methods and attributes can be grouped together. An interface will ensure the child class implements these methods and attributes by overriding them. This would avoid careless mistakes like forgetting to write an important method for a certain class.
  + The abstract class Tree has a dependency to the class Koopa because it has a chance to create an instance of Koopa based on a specified probability and condition.
  + The Koopa class uses capabilities (via the actor class) to know if it can be killed based on whether the actor (Mario) possess a weapon (a wrench). This creates a dependency from Koopa to actor, which is not shown is it is overridden by the association between the same 2 classes (Koopa is an actor).
* The class Goomba has been modified to have additional relationships  
    
  **Justification:** 
  + The abstract class Tree has a dependency to the class Goomba because it has a chance to create an instance of Goomba based on a specified probability and condition.
* Application Class now has a dependency on the abstract Enemy class instead of having individual dependency on each type of enemies.