**Design Rationale for requirement 6: Monologue**

* An abstract class ToadAction was created in order to regroup all the actions a toad can perform in a single class. Codes can be reused and shared. By using abstraction, methods that are static and necessary are created for all instances of the class.

**Justifications:**

* + The class Speak allows the toad to speak different dialogues to the player and the class Trade allow a trade to be done.
  + These two above-mentioned classes have an inheritance to the abstract class ToadAction as both are relevant actions that any instances of Toad should/can perform.
  + The abstract class ToadAction has an inheritance to the abstract class Action because it can inherit and make use of several common actions that any actors can perform.
* A new class ‘Toad’ is created. The class contains all the attributes/methods that are to be used by instances of class Toad. Various types of relationships are implemented to fit the class in the system.  
   **Justifications:**
  + The class Toad is also linked to the abstract Class ‘Actor’ as it is part of the game as an actor.
  + The class Toad 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 class player has a dependency to the class Toad as the player has to speak to the Toad to initiate the conversation and the class Player does not have any attributes of type Toad, hence dependency.
  + Toad class is linked to the abstract class Item via a dependency as the Toad instance’s action changes depending on the items in the actor’s inventory. (whether the player has Power Star effect active in his inventory)
  + Toad class is linked to the interface weapon as Toad instance need to what to say based on whether Mario has a wrench.
  + Toad class has an association to the class Location as the former uses an instance of the class Location to determine a location appropriate to place the toad. The location class further has a dependency on the class Wall as the toad needs to be placed within walls.
  + The class Toad has an association on the class Display as it needs to output whatever he is saying to the output window. As a result, it will need to have an instance of the class Display as its attributes in order to access the relevant methods to display the output.
  + The class Toad has an association to the abstract class ToadAction. Using an abstract class means that DRY principle is implemented, and codes can be reused. This is done as both actions of a toad are similar and relevant to an instance of toad.