```
Assignment 3 Rationale:

//////Update A2 Design//////
- All enemy characters now extend from an abstract Enemy class.
    - This is done to follow the DRY principle as many enemy characters'
classes implements the same or similar logic.
- All damaging actions now extend an abstract AttackAction class.
    - This is done to both follow the DRY principle and to follow the
Single-Responsibility principle by avoiding having one class that
      determines the type of attack generated based on players statuses.
      Instead, separate classes now handle different attacks, while
extending the properties of AttackAction.
- Tree is now an abstract class extending HighGround. The three stages of
Tree's life now extend Tree.
- The above refactoring also allows the code to better implement OCP. The
use of abstraction allow for further developments, should more enemies,
  more damaging attacks, or more high grounds be added.

//////REQ1//////
- TeleportManager class is a singleton class that is used to store a
reference to the different GameMaps and a reference to
locations that implement the interface Teleport.
- The TeleportManager class allows for the access of GameMap types outside
the World class and positions package.
- By storing instances of Teleport objects in the manager, these objects
and the Teleport methods can be accessed in sections of the
    code where only objects of the Location class are available. This
avoids instanceof and type conversion.
- The WarpPipe has a destination Location as an attribute, which is the
connected WarpPipe to teleport to.
- If no destination exists for the WarpPipe, the destination is set to be
the WarpPipe in LavaZone.
- When the player is standing on top of a WarpPipe a WarpAction is
generated for the destination of that WarpPipe.
- The destination of the WarpPipe that is being teleported to is set to the
location of the current WarpPipe. This allows the player to
    be able to travel back to the WarpPipe of origin.
- If an enemy is on top of any WarpPipe that is being teleported to, it is
removed from the map prior to the player having teleported.


//////REQ2//////
- Extending from Assignment 2, an abstract Enemy class was created, from
which all existing and new enemy characters' classes extend, except Flying
Koopa.
    - This implementation aims to follow the DRY principle and reduce as
much repetitive code as possible.
- Flying Koopa will extend from Koopa instead, as they have nearly
identical attributes other than hitpoints and an additional monologue.
    - This implementation helps fortify the LSP principle as where Koopa is
expected, Flying Koopa should be a valid parameter as well.
- Entry to a HighGround is now done by checking if the actor to enter has
the status FLY.
- Bowser will implement the same FireAttackAction as the player as their
FireAttack function is identical based on the requirements.
- The endgame is checked through the use of statuses. In Peach
allowableActions, the status of HERO is checked for the other actor.
  This status is provided through the acquisition of the Key object dropped
by Bowser. This implementation leaves room for the program to grow,
  as the win-condition is not limited by a specific object/instance.
```

```
//////REQ3//////
- The Bottle class extends the Item class. It is added to the Player's
inventory at the start of the game.
- The Bottle is a container for Water objects, storing them in a stack.
Everytime the Bottle is filled, Water is pushed to the top, everytime is
drank from
    a Water object is popped.
- Both PowerWater and HealthWater classes extend the abstract class Water.
Water implements Consumable but is not of the Item type.
    Having these subclasses extend from the Water class allows them to be
contained in the same stack, but also allows for the different logic
required for the
    different effects of each type of water.
- HealthFountain and PowerFountain extend from the abstract base class of
fountain. The allowableActions method is defined in the base class and the
abstract method
    getWater are defined in the subclasses. Each getWater method returns
the appropriate type of water for the fountain.
- When the player is standing on a fountain and has a bottle in their
possession, a FillAction is generated. FillActions allow the player to fill
their bottle with
    the Water objects associated with that specific type of fountain.
- If a player has water in their bottle, a ConsumeWaterAction is generated
that allows the player to consume the Water object at the top of the stack.
- The ConsumeWaterAction.execute() method uses the "drink" method of the
Bottle class, which handles the popping of the Water object and applies
effects to the
    actor. Each type of Water object contains the logic required to apply
effects to the actor consuming the water.
- By implementing the DrinksWater interface, baseAttack and associated
methods are added to the player class. This allows the effect of PowerWater
to be used
    for the player's intrinsic attack.
- All actors that implement the DrinksWater interface have a reference
stored in the ActorManager class.

//////REQ4//////
- The FireFlower extends Item and allow players to consume it directly from
the ground by calling its parent method addAction() in the
getAllowableActions() method.
- FireAttackAction as mentioned above extends the abstract class
AttackAction and inherits all the properties of a damaging attack.
    - This eliminates repetitive code (DRY) and opens the program up for
further modification if necessary.
- The Fire dropped by FireAttackAction is a separate class which extends
Item (non-portable) as well.
- The damage dealt by Fire to actors standing on it is done during the
object's tick() method.
    - Doing so will reduce dependencies as it takes away additional
dependencies that would have been checked in FireAttackAction instead.

//////REQ5//////
- Actors that have monologues implement the Speaks interface. Instances of
these Actors are stored in the ActorManager class using the
registerSpeech() method.
- The Speaks interface implements the setter for monologues
```