Design Rationale
Newly created/modified classes with new relationships

PrincessPeach
Responsible for representing PrincessPeach.

Key
Responsible for representing a Key item, in order to win the game.

EndGameAction
Responsible for ending the game.

Bowser
Responsible for representing the Bowser enemy. Extends enemy class to not violate DRY.

PiranhaPlant
Responsible for representing the PiranhaPlant enemy. Extends enemy class to not violate DRY.

WarpPipe
Modified to ensure PiranhaPlant spawns on the second tick of the game, and when resetted spawn another PiranhaPlant if there is no currently no actor on it.

Mature
Modified to have a chance to spawn FlyingKoopa

FlyingKoopa
Responsible for representing the FlyingKoopa enemy. Extends Koopa class to not violate DRY. Further explanation regarding Liskov Substitution Principle will be down below.

Features of PrincessPeach
"Once you have defeated Bowser and obtained a key, you can interact with her to end the game with a victory message!"
In PrincessPeach's allowableActions method, we can loop through otherActor(player)'s inventory and check whether he has a key or not, if so then we can use EndGameAction to end the game. In EndGameAction, it terminates the game by removing our main player from the GameMap.

Features of Bowser
"Whenever Bowser attacks, a fire will be dropped on the ground that lasts for three turns."
Bowser will be given the capability of fire breathing when instantiated. As to how the fire will be dropped, it will be explained in REQ4's design rationale.

"When the Bowser is killed, it will drop a key to unlock Princess Peach's handcuffs."

Bowser will be instantiated with a Key in his inventory.

"Bowser punches with 50% hit rate and 80 damage"
We can override the getIntrinsicWeapon method and change the damage value and verb when creating a new IntrinsicWeapon. Hence, the dependency from Bowser to IntrinsicWeapon.

"Resetting the game (`r` command) will move Bowser back to the original position, heal it to maximum, and it will stand there until Mario is within Bowser's attack range."
In Bowser's resetUpdate method, we can check if Bowser needs to be reset. If so, check if Bowser's original spawn point has any actors, if not just move Bowser back to his original spawn point and heal him to maximum. If so, remove the actor currently at Bowser's original spawn point and move Bowser there, and heal him to maximum.

Features of PiranhaPlant
"Piranha Plant will spawn at the second turn of the game"
In WarpPipe's tick method, we can check if we have already spawned a PiranhaPlant, if not then spawn it.

"PiranhaPlant chomps with 50% hit rate and 90 damage"
We can override the getIntrinsicWeapon method and change the damage value and verb when creating a new IntrinsicWeapon. Hence, the dependency from PiranhaPlant to IntrinsicWeapon

"Once the player kills it, the corresponding `WarpPipe` will not spawn Piranha Plant again until the player resets the game (`r` command)."
In WarpPipe's tick method, we can check if WarpPipe needs to be reset. If so, if there is no actor on the WarpPipe, spawn a new PiranhaPlant.

"Resetting will increase alive/existing Piranha Plants hit points by an additional 50 hit points and heal to the maximum."
In PiranhaPlant's resetUpdate method, we can check if PiranhaPlant needs to be reset. If so, use increaseMaxHp to increase max HP and heal to maximum.

Features of FlyingKoopa
"Furthermore, it can walk (fly) over the trees and walls when it wanders around (incl. other high grounds)."
Every FlyingKoopa will be given the capability of being able to fly when instantiated. In HighGround's tick method, if the actor on it has capability to fly, then return true.

Explanation on FlyingKoopa extending Koopa
The only different feature a FlyingKoopa has from Koopa is that it can fly. So, every other feature of FlyingKoopa is the same as Koopa. Hence, FlyingKoopa extends Koopa class as mentioned earlier to adhere to DRY. The reason why this still adheres to Liskov Substitution Principle is because FlyingKoopa does not overwrite any of Koopa's methods, hence it will not result in any unexpected behavior. We can replace FlyingKoopa with Koopa and it still will not result in any unexpected behavior. The only method that will return a different output depending on if it's FlyingKoopa or Koopa is HighGround's tick method. But, replacing FlyingKoopa with Koopa will just return false instead of true since Koopa can't fly. Of course, down the line if there were more

different features of FlyingKoopa compared to Koopa, Liskov Substitution Principle might be broken, but given the assignment instructions so far, this is not the case.