Recommendations for change to the game engine

1. Actor class does not have an accuracy attribute or method but instead this is handled in the AttackAction class of the game package.

In the Actor class, there is a getIntrinsicWeapon() method which indicates that an Actor can attack. Since the game intends to have some form of attack accuracy for Actors to hit or miss their attacks, the accuracy of an Actor's attack should be an attribute of the Actor. However, this is implemented in the AttackAction class of the game package. As a result, when we want to implement the Zombie's bite accuracy, we are forced to either downcast within the AttackAction class, or create a new method in the ActorInterface.

As a suggestion, the Actor class should have a protected integer attribute that represents their attack accuracy so that we can easily modify their chances of missing an attack such as in the case of Zombies as well as some form of accessor method to get the value or a method that calculates whether they hit or miss their attack. This helps for encapsulating the knowledge of attack accuracies to only be within the Actor and this helps to avoid adding dependencies on AttackAction through downcasting or placing if checks with "instanceof" to modify attack accuracies of certain Actors, which is a code smell.

2. Ground class has no allowableActions list unlike the Item class.

If Items have an allowableActions list as an attribute to store Actions for Actors to perform on an Item, Ground should be the same, but strangely it does not have one. As a result, certain features such as Farmers sowing a Crop on a Dirt ground must check if the location they are standing on has a displayChar of "." to be able to perform a SowingAction. The lack of an allowableActions list attribute forces future implementations on special types of Ground to have a Connascence of Meaning (CoM) as in the case of the Farmer sowing feature, there is a connascence on the meaning of the displayChar of Dirt whereby it is assumed that "." represents a Dirt ground. This could cause issues in the future if the same displayChar represents some other kind of Ground that the Farmers should not sow a Crop on. Furthermore, we also can only initialize a new Action class in the Actor class based on the Ground they are standing on and it creates a dependency between the Actor and the Action when it could have been avoided if the Ground was the one providing the Action to the Actor.

As a suggestion, the Ground class should have a protected Actions attribute named allowableActions just like the Item class. It removes the issue of CoM from occurring as well as an Actor no longer has to keep track of the different types of Ground they are standing on to know what possible Actions they can do.

However, this only applies to the Player class. AI Actors such as Zombies and Humans go through a Behaviour array to perform specific Actions. To allow specific AI Actors to perform Actions on special types of Ground, we can create a new method in the Ground class called canActorPerformAction() which returns the respective Action to be performed only if the Actor's ZombieCapability matches the requirements accordingly. Otherwise, it returns null.

Therefore, a Behaviour subclass only needs to call that method to return an Action or a null value which goes well with Behaviour classes.

3. Instead of having 3 add methods in Actions, it's better to have 1 instead.

Problem being that all these methods have the same function name which is add() which can be really confusing to use. All 3 add() methods serve the same main purpose which is to add Action into the ArrayList of actions attribute in the Actions class.

Hence, the change I'd propose to fix this is to instead have 3 methods to add Action or multiple Action into the ArrayList of the actions class, have one method called add(Action, Actions, List<Action>). To use the method, we only have to pass in what's necessary in the chosen parameter and the other parameters can be null. The method then will run the parameter that isn't null and add Action into the ArrayList in Actions.

This will simplify the method where it is used to add an action or multiple actions into the ArrayList. The usage of adding Action or multiple Action into actions will only have to utilize one method which is easy to use compared to 3 add methods and needing to figure out which add method to use. It also will utilize the principle DRY when only using one add method.