**Design Rational for Assignment 1**

1. **ZombieActionClass**

This class is created that extends the AttackAction class to accommodate the need for zombie to decrease their probability of punch when they lose one or both their limb. Modifying the code at AttackAction class has a high risk of breaking it, because some major changes are needed in the execute method, making a new class not only shorten the code but also reduce the chance of breaking the getAllowableAction class which might affect the human class.

1. **WeaponLeg and WeaponHand**

This class is created to create a weapon obtained from fallen zombie’s limb. It inherits the WeaponItem class. We created this class, to make it easier to create an instance of this weapon every time a zombie’s limb fall off. It will be used in the in the *hurt* method in the zombie class, where every time a zombie is hurt, there will be a probability its limb will fall.

1. **Zombie**

*GetIntrinsicWeapon* method is changed to add the probability of having a bite attack. The reason it is added here instead of adding it to the weapon because biting it’s an attack from a part of the zombie (i.e. not weapon), hence its more suitable to have it in the *GetIntrinsicWeapon* method. Besides, adding it here can reduce the lines of code compared to creating a new attack class for it.

*PlayTurn* method is also modified to check for broken leg, so that it can reduce the movement speed on the zombie based on the last action of the zombie. Its implemented in this method because, it’s the only method that keeps track of the zombie last action and it’s in the zombie class, so it is also easier to access the condition of the zombie (e.g. how many legs left) which can reduce repetition compared to creating a new subclass of action for zombie with broken leg.

These designs reduce dependencies as most of the method that required access to the zombie class is implemented inside the zombie class.

1. **Zombie saying ‘Brainns’**

The probability of the zombies saying brains is implemented in the playTurn function where in every turn it takes it will have a probability of saying ‘Brainns’. This is implemented in playTurn instead of making it an action because, in every turn an actor can only take an action. We decided that we should not waste a turn for zombie just to say ‘Brainns’

1. **Limb class**

A class limb that keeps track of the number of limbs a zombie has left is created instead of putting a counter in the zombie class. This follows the design principle that ‘Classes should be responsible for their own properties’, where in this case, the purpose of this class is to keep track of a zombie’s limb. This makes future code changing easier because this code does not have relation to other method.

1. **Corpse (Rising from the dead)**

A corpse object from the Corpse class is created when an AttackAction from the zombie class is acted upon Human classes from its playTurn action. A Corpse class is extended from PortableAction class because from the current game design and forum discussions, it has been clarified that a corpse object should be portable. After 10-20 rounds based on some probability calculations, it will create a Zombie object. The reason for having a Corpse class is to allow reusability of multiple Corpse object to have its own individual ticks.

1. **CraftAction, ZombieClub, ZombieMace (Crafting weapons)**

From the current source files, we found out that many of the class objects were created as a result of actions such as corpse object being created from AttackActions class. To justify our design reasonings, we have decided to create a new class CraftAction for extensibility purposes and allowing ZombieClub and ZombieMace objects to be created through this action.

When a ZombieLeg or ZombieHand object is instantiated, CraftAction is passed in as the fourth parameter in Item constructor as an allowableActions attribute. This allows the Player to craft weapons from ZombieLeg and ZombieHand into ZombieMace and ZombieClub respectively. ZombieClub and ZombieMace are extended from WeaponItem class. With this design implementation, Non-Player Characters will not be able to utilize CraftAction

1. **Farmer, FarmBehaviour(SowAction, FertilizeAction, HarvestAction), Crop, Food, HealBehaviour(HealAction) (Farmers and food)**

We have decided to create several new actions for the Farmer and Player class. SowAction, FertilizeAction and HarvestAction are extended from the Action class and is used to interact with Location of Ground object to produce Crop objects. We have allowed only the Farmer class to have FarmBehaviour which consists of SowAction, FertilizeAction and HarvestAction while Player will only have HarvestAction as required. If there were to be multiple Farmer objects, they would reuse this behaviour and I will not have to repeat any codes.

A SowAction will have a 33% chance of creating a Crop object where the farmer will sow a seed onto nearby dirt that is unoccupied by Crop object. A SowAction will have an attribute of Location to determine whether a Crop should be sow. This allows me to utilize Fail Fast design if a Crop object should not be sown in the current location.

FertilizeAction class is created to allow Farmers to fertilize the crops and reducing the time left to ripe by 10 turns. FertlizeAction has Crop object attributes to determine if there are Crops objects.

HarvestAction class is used to act upon Crop objects that are ripe and will return a new Food object which can be used by heal method from Actor class to restore health points. This action also has Crop object as an attribute which checks if the crop is ripe before it is allowed to harvest.

The Crop class is extended from Ground class as it contains many of the required methods to determine whether the crops is ripe from the tick method. Other methods such as canActorEnter() and getDisplayChar() is useful for UI purposes.

Food Class is created from HarvestAction by Farmer and Player class and is used for healing. They are extended from PortableItem class as the Player should be able to carry the Food object in its inventory.

HealAction is a new Action stored within a new Behaviour called HealBehaviour. From the original design documentation, we found out that Humans actions are based on its behaviour. Therefore, we have decided to introduce HealBehaviour to allow HealAction to be called when a damaged Human decides to recover some health points. This design implementation improves the Human AI instead of wandering around where HealBehaviour should be prioritize over wandering. With this design implementation, Player which is an extension of Human class should be able to heal.

We have decided to include many new classes to allow further extensibility to the game engine such as having the possibility of having multiple new Crop objects implementation which produces different kinds of Food object that heals for a different amount. Attributes are assigned to the newly introduced actions to allow Fail Fast design implementation.