# Design Changes

## Zombie Attacks/Beating up the Zombies

## General

1. Instead of integrating the bite attack into **AttackAction**, a **BiteAttack** class was created instead. The bite attack requires a different method to retrieve the **Weapon** to execute a bite attack and it also has a different probability to miss compared to a normal attack. This disrupts the flow of the code through many conditional checks to check if the **actor** is a **Zombie** and whether **weapon** is for a bite attack would make the code messy and hard to follow. It would also be difficult to integrate other types of attacks (if needed) if this design is followed again.

At first, it was decided to integrate the bite attack into **AttackAction** to avoid duplicated code that would be in the **execute(..)** method of **AttackAction** and **BiteAttack**. This would exist if **BiteAttack** was created to inherit **AttackAction** without any refactoring. In the new design, this is avoided by refactoring common blocks of code in the **AttackAction.execute(...)** method into package-private methods. **BiteAttack** inherits these methods and uses them in its own execute method which avoids duplicated code and follows the DRY principle.

1. Because of change #1, the bite attack health restore is now in **BiteAttack.**
2. Because of change #1, **AttackBehaviour** is now modified to randomly return an **AttackAction** or **BiteAction** action.
3. **DropAdjacentItemAction** was changed to inherit **DropItemAction** instead of **Action** as it will contain the same attribute and most methods. The **execute(...)** method is different and is overridden to drop the item in an adjacent cell, rather than at the **actor’s** location. Since of the change in inheritance, **DropAdjacentItemAction** will now only have a minimum and maximum of 1 **ZombieLimb**.
4. **AttackAction** is no longer designed to be responsible for dropping zombie limbs when they are knocked off. This is because when a **Zombie** takes damage, it may have its limbs knocked off which modifies attributes in the **Zombie** class. To follow the “Classes should be responsible for their own properties”, the loss of limbs should be handled in the **Zombie** class for encapsulation and avoids down casting.
5. Because of #5, **executeReaction(...)**, **getArmCount()**, **getLegCount()** and **getLimbCount()** are added to the **ActorInterface** interface. The **executeReaction(...)** allows **Actions** actors have as a reaction to an action done to them. For example, damage is inflicted on a **Zombie** and it knocks some limbs off. The action to drop the limbs are stored in the **actions** attribute in the actor. After it gets back to the caller of the damage infliction, **executeReaction(...)** is called on the **Zombie** and it executes the actions to drop its limbs on the map.

**getArmCount()**, **getLegCount()** and **getLimbCount()** returns the actor’s arm, leg and limb count respectively. This is used to check the conditions of the actor before returning an action. For example, **AttackBehaviour** checks how many arms the **Zombie** has, to determine the probability of a normal attack or bite attack.

1. It was originally thought that a **Zombie** speaking wouldn’t turn up a turn, however this is incorrect, and it should take up a turn. So, a **SpeechBehaviour** and **SpeakAction** class have been added to fit the requirement. The **SpeechBehaviour** is the second behaviour in the **Zombie** behaviour list. This is because a **Zombie** is required to pick a weapon at the start of its turn if it is available which is checked in the **ScanvengeBehaviour**. Then there is a 10% chance that the **Zombie** will speak before checking for anything other possible actions.

## Class diagram layout

1. Dependencies of subclasses that are shared with its superclass are no longer shown in the class diagram. A subclass inherits the dependencies of its superclasses so also look at the dependencies of the superclasses to know all the dependencies of a subclass.

## Relationships

1. The solid line from **AttackBehaviour** to **<<interface>>Behaviour** is changed to a broken line as it should be a realisation relationship. This is the same for **ScavengeBehaviour** to **<<interface>>Behaviour**. The multiplicity from **Zombie** to **AttackBehvaiour** is changed to 1 to 1 as a **ScavengeBehaviour** instance should only belong to one **Zombie** instance. This is the same for **Zombie** to **ScavengeBehvaiour**

## Crafting Weapons/Rising from the dead/ Farmers and food

## General

1. **CraftWeaponAction** is dependent on **ZombieLimb** instead of **Item** as **ZombieLimb** is the item that **CraftWeaponAction** stores as one of its available actions that the item can perform.
2. Instead of a method in **Human** creating a zombie a new class called **HumanCorpse** is created which extends **PortableItem**. By doing this extension allows us to track the ticks that the game goes through so we can turn the corpse item into a **Zombie** actor later.
3. Because of change #2, the dependency between **Human** and **Zombie** is removed.
4. When a **Zombie** actor is created from the **HumanCorpse** item we either remove the item from the players inventory or remove it from the map and replace it with a **Zombie** actor at its location or at an adjacent location without an **Actor**.
5. A **Farmer** needs to exhibit **WanderBehaviour** like the other **Humans** or else they will do nothing.
6. Humans and Farmers returns a by default **DoNothingAction** if its **WanderBehaviour** does not return an action. This shouldn’t happen but is required due to the try-catch block on the **WanderBehaviour**.
7. **SowAction** and **FertilizeAction** were added so a **Farmer** can sow or fertilize for its turn. Although Farmers are currently the only **Actor** that can sow and fertilize, having these classes allows other **Actors** to execute these actions if the game design were to change in the future, similar to the unique **CraftWeaponAction**  for **Player**.
8. **HarvestAction** was added so both **Player** and **Farmer** can harvest a ripe crop for its turn. The harvest move must be an **Action** a **Player** can choose from, so it should be a class that stores an appropriate **Location** object so the **Player** can easily execute the action.
9. **EatAction** was added so **Player, Human** and **Farmer** can eat food that is on the ground. Similar to **HarvestAction**, the eat move must be an **Action** a **Player** can choose from, so it should be a class that stores an appropriate **Item** object so the **Player** can easily execute the action.
10. **Food** now inherits **PortableItem** as all **Food** objects are can be carried by an **Actor** and placed on the ground. This inheritance fits better than inheriting **Item**.

## Relationships

1. Between **PickUpItemAction** and **Item**, the multiplicity was changed to 1 to 1 as an **Actor** needs to be at the **Item**’s location to pick it up and a **Location** can only have up to one **Actor**, therefore an **Item** can only have one **PickUpItemAction** instance.
2. Between **CraftWeaponAction** and **ZombieLimb**, the multiplicity was changed to 1 to 1 as a **ZombieLimb** instance may only have one crafting path it can take. For example, a **ZombieLimb** that is a zombie arm can only be crafted to a **ZombieClub**, nothing else. Therefore, an **Item** instance can only have one **CraftWeaponAction** instance.