Design Rationale: Zombie Attacks
New Class Added: AttackHumanAction

Roles and responsibilities:
- Extends the AttackAction class because they share similar functionality, therefore duplication of code can be reduced.
-Random class has to be imported so that an instance of Random class can be created to generate the probability of hitting from the bite attack at 25%.
-Method nextBoolean() or nextDouble() from Random class can be implemented to achieve the probability of hitting.
- Represents an action carry out by the Zombie to attack the Human or Player at adjacent location of the player.
-Inherit AttackAction constructor which take in an Actor which is the target.
-Has attribute of type ZombieCapability to store the attackable team ALIVE so it only allows this action to be acted on the Human or Player.
-The execute() method will be override to let the zombie has 50% of probability of using bite instead of their normal attack. The bite attack is one of the intrinsic weapon of zombie, but it has a lower chance of hitting than the punch attack but higher damage. So, I will set the probability of hitting from the bite attack at 25% and the probabilities of the hitting from weapon or punch at 50%.
-Once the bite attack is not missed, heal() from Actor class should be called to restore 5 health points of zombie.

New Class Added: PickUpWeaponBehaviour
Roles and responsibilities:
-It implemets the Behaviour Class to override the method getAction() from Behaviour class.
-In the override getAction(), there will be a for loop that iterates every item in that specific location.(for (Item item: map.locationOf(actor).getItems()))
-Inside the for loop, portability and weaponization of the item will be checked.
-Method asWeapon() in Item class will be called to check the weaponization of the item.
-Method getPickUpAction() in Item class will be called to check the portability of the item.
- If the item can be weaponized and it is portable, then it returns PickUpItemAction().Otherwise, return null.

New Class Added: Word
Roles and responsibilities:
-It is a class that represent every word that can be shouted by zombie or human.For example, “Brainnn”.
-It has an attribute with type String, word. It indicates the name of the word.
-It also has a constructor with a parameter String name.
-It has a method called getWord(), it returns the String **word**.

New Class Added: ShoutBehaviour
Roles and responsibilities:
-It is a class that returns ShoutAction that will undergo execute() to print a String.
-It implements Behaviour class to inherit the method getAction().
-It has an attribute with type ArrayList<Word>, it is named as **Words**.
-There is an override method called getAction(). The getAction() has 2 parameters which are Actor and Gamemap.
-In the getAction() a Word instance is created and **Words** adds the new instance. For example,
(Word word = new Word("Brainnn"))
-There is an override method called getAction() that returns ShoutAction.

New Class Added: ShoutAction
Roles and responsibilities:
-It extends Action class because they share similar functionality, therefore duplication of code can be reduced, hence it improves reusability in java.
-There are override methods called Execute() to perform the shout action and MenuDescription() to display a descriptive string about what the actor is performing on the menu.
-It imports Random class from java.util.package. After importing Random class, Random instance can be created to perform its method to carry out the probability of shouting brain from each zombie in every turn.
-The probability of zombie shouting "brain" is set to 10%.

Class modified:Zombie
Modified part
- An override method called getIntrinsicWeapon().
- Add an override method called heal() from actor.
- Add pickUpWeaponBehavoiur into the attribute, Array of Behaviour type.
- Add ShoutBehaviour into the attribute, Array Behaviour type.

Roles and responsibilities:
-Bite is added as an intrinsic weapon of zombie.
-Zombie has 50% of probability using bite instead of punch. To achieve this requirement, Random class has to be imported to Zombie Class.
-An instance of Random class can be created so that the method nextBoolean() can be used to generate the 50% probability of using bite for each zombie.

Class modified: AttackBehaviour
Modified part:
- An override method called getAction().

Roles and responsibility:
- In the getAction() , instead of returning a new AttackAction, it will first determine the attackableteam, if attackableteam == ALIVE, it will return a new

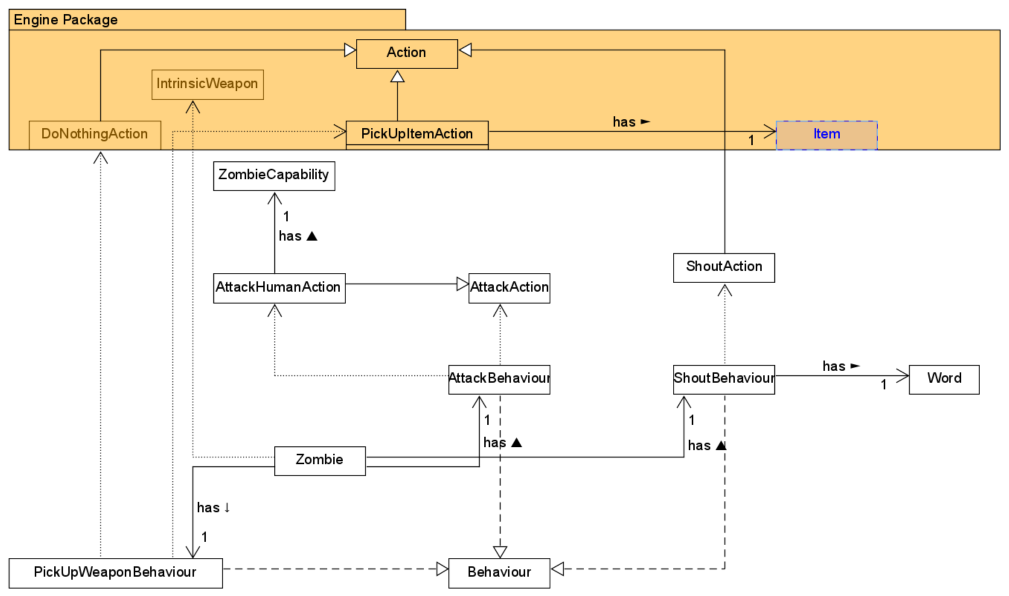
AttackHumanAction. On the other hand, if attackableteaem == UNDEAD, it will return a new AttackZombieAction.

Class modified: AttackAction
Modified part:

- Become abstract class.
- A method called execute() is modified into an abstract method
- Add a new method called targetdDied()

Roles and responsibility:
-This class is modified into an abstract class that provide a base code for all the attack actions in the game, which are AttackZombieAction class and AttackHumanAction class where they share some similar functionality such as the execution part when the target dies.
- The execute method is modified into an abstract method since AttackZombieAction and AttackHumanAction has different execution.
- Has a new method called targetDied() that is responsible for the action when the target dies and this functionality is shared among AttackZombieAction and AttackHumanAction, therefore duplication of code can be reduced and it improves the reusability of code.

UML Zombie Attacks



Zombies should be able to bite. Give the Zombie a bite attack as well, with a 50% probability of using this instead of their normal attack. The bite attack should have a lower chance of hitting than the punch attack, but do more damage.
To achieve this functionality:
-In the Zombie classBite is added as an intrinsic weapon into getIntrinsicWeapon(). So there are 2 intrinsic weapons such as punch and bite for zombie. I set the damge of bite as 15 per hit which is higher than punch damage. To make the zombie has 50% probability of using bite instead of theirnormal damage which is punch. An If-else statement has to be implemented in getIntrinsicWeapon(). In the if-else statement, nextBoolean() method that is imported from Random class has to be used to check the 50% probability of using bite for every zombie in each turn.
-Bite atack should have a lower chance of hitting than the punch attack. To achieve this functionaility, A new class, AttackHumanAction class is created and it inherits AttackAction class. In AttackHumanAction class, there is an override method, execute(). The execute() has an if-else statement to check the probability of bite and punch getting missed. nextDouble() or nextBoolean() can be implemented in this if-else statement to reach the aim.

If there is a weapon at the Zombie's location when its turn starts, the Zombie should pick it up. This means that Zombie will use that weapon instead of its intrinsic weapon like bite or punch.
-To achive this functionality, a new class, PickUpWeaponBehaviour is created. It implements Behaviour class to improve the reusability of code. There is an override method called getAction(). In this method, there will be a for loop that iterates every item in that specific location. To pick up the item from zombie, the item must have the ability to be weaponised and it is portable. To achieve these, getPickUpAction() and asWeapon() from item class will be called to check the requirement before pick up the weapon.
-If the item acheives those requirementys, getAction() will return PickUpItemAction.
-If the item cannot be weaponised or it is not portable, getAction() will return null.

Every turn, each Zombie should have a 10% chance of saying "Braaains".
-To achieve this functionality, a new class, Word , ShoutBehaviour and ShoutAction are created. Word represents every words that can be shouted by zombie or player. It has attribute with type String to store name of word like "Brain".
-Main function of ShoutBehaviour class is to return ShoutAction.
-ShoutBehaviour class has an override method called getAction() and an attribute with type Arraylist<Word> that store every Word.
- In this case, an instance of Word is created in getAction(). For example, Word word = new Word (" Brain").
-After all the Word instances are created, the attribute with type ArrayList<Word> will be called to add all the Word instance that are just created.
-After that there is a for loop to iterates the ArrayList<Word>. In this case, an instance of Word is created which is "Brain". And if else statement is undergo to check that whether there is this actor in this location. If it is , ShoutAction is returned with just 1 word which is " Brain".
- In the ShoutAction class, it inherits Action class. Therefore there is an override method called execute () to carry out the shout "Brain". In this execute() , Random instance has to be created to call method nextDouble() in order to achievie the setting of probability to shout word.