

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 implements 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: **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 String Array, it is named as word. It stores every String of word.

There is an override method called getAction(). The getAction() has 2 parameters which are Actor and Gamemap.

- In the getAction(), it returns Action shout() which has parameter Actor.
- The Action shout() returns ShoutAction if the probability of shouting word is achieved.

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.
- It has constructor with parameter String type.
- There are override methods called Execute() to return the description of the shout action and MenuDescription to display a descriptive string about what the actor is performing.

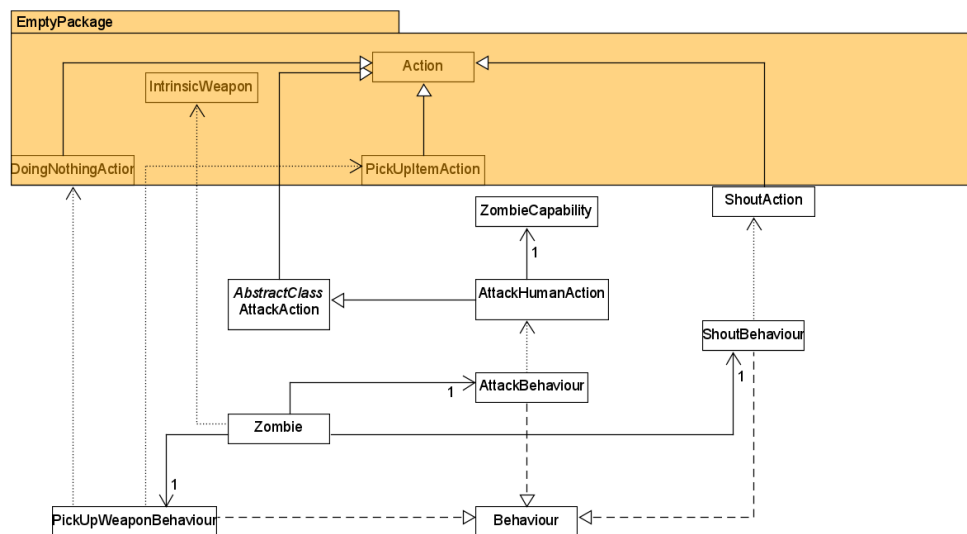
Class modified: AttackBehaviour

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 `attackableteam == UNDEAD`, it will return a new `AttackZombieAction`.

UML for Zombie Attacks



How to achieve the required functionality:

- 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` class, `Bite` is added as an intrinsic weapon into `getIntrinsicWeapon()`. So there are 2 intrinsic weapons such as `punch` and `bite` for zombie. I set the damage of `bite` as 20 per hit which is higher than `punch` damage. To make the zombie has 50% probability of using `bite` instead of their normal attack which is `punch`. An `If-else` statement has to be implemented in `getIntrinsicWeapon()`. In the `if-else` statement, `nextDouble()` 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 attack should have a lower chance of hitting than the punch attack. To achieve this functionality, 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()` can be implemented in this if-else statement to reach the aim.
- For example, if the weapon is bite and the probability of using bite is checked before the actor returns String that describe that action.
- If the zombie bites human successfully, it restores 5 health points.
- To achieve this: an if statement is added into a method `execute()` in `AttackHumanAction` class.
- Heal method from `Actor` class is being called if the human is bitten by zombie.
- 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 achieve 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 achieves those requirements, `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 "Braaaains". To achieve this functionality, new classes `ShoutBehaviour` and `ShoutAction` are created.
- Main function of `ShoutBehaviour` class is to return `ShoutAction`.
- `ShoutBehaviour` class has an override method called `getAction()` and an attribute with type String Array named as `words` that store every Word.
- Initializes this String Array with String word called "Brainnn".

- In the `getAction()`, it returns a method called `SHout()` that returns the `ShoutAction` if the requirements are fulfilled.
- One of the requirements is to check the probability of shouting word. Therefore, `random` class is imported to carry out a method called `nextInt()` within the `for` loop that iterates every word that is stored in `Array String`.
- To allow the zombie to shout a single word, not a sentence. I created an `if` statement to check if the length of the word is within 10 letters.
- If these requirements are fulfilled, then `Shout()` returns `ShoutAction`. Otherwise, it returns `null`.
- In `ShoutAction` class, in the `execute()`, it returns `menuDescription()` which is a method that prints a string descriptive about what the actor is performing.

Beating up the Zombie

New class added: **Limb**

Roles and responsibilities:

- It's an abstract class that inherits `WeaponItem` class that will have attribute and method that are shared by `Arm` and `Leg`.
- In the constructor, the limb's damage is initialised to 20, and a new `CraftWeaponAction` is created and added to the limb's allowable actions.
- Has an abstract method called `craftWeapon()` that return a new weapon (If it's an arm, player can craft it into a `ZombieClub`, if it's a leg, player can craft it into a `ZombieMace`)

New class added: **Arm**

Roles and responsibilities:

- A type of weapon that inherits `Limb`
- Has an attribute of type `Weapon` called `upgradedWeapon` that stores the upgraded weapon `Arm` will become if it's crafted by the player, added notes, only player can craft this `Arm` into another weapon and zombie can only use this weapon as it is.
- Can be used by both zombie and the player as a weapon with damage of 20 as it inherits `Limb` class

- Represented by the symbol "A" when it's on the ground
- Has method craftWeapon() will return upgradedWeapon which is an object of type ZombieClub and it can only be called by the player when the player has an Arm in their inventory.

New class added: **Leg**

Roles and responsibilities:

- A type of weapon that inherits Limb
- Has an attribute of type Weapon called upgradedWeapon that stores the upgraded weapon Leg will become if it's crafted by the player, similar to Arm, only the player can craft this Arm into another weapon and zombie can only use this weapon as it is.
- Can be used by both zombie and the player as a weapon with damage of 20 as it inherits Limb class
- Represented by the symbol "L" when it's on the ground
- Has method craftWeapon() will return upgradedWeapon which is an object of type ZombieMace and it can only be called by the player when the player has an Arm in their inventory.

Class modified: **AttackAction**

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.
- The execute method is modified into an abstract method since AttackZombieAction and AttackHumanAction has different executions.
- Has a new method called targetIsDead(Actor, GameMap) 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 (DRY).

New class added: **AttackZombieAction**

Roles and responsibilities:

- Inherits AttackAction and it represents an action that can be carried out by the Player or a human to attack the zombie at adjacent location of the player or the human.
- Has attribute of type ZombieCapability to store the attackable team UNDEAD so it only allows this action to be acted on the Zombie.
- Inherit AttackAction constructor which take in an Actor which is the target.
- In the execute method, when the player uses a weapon or intrinsic weapon which is punch, both attacks has a 70% successful hit rate. If the player successfully land a hit on the zombie, it will call a method from Zombie class called zombielsAttacked() and it will return a string that indicates whether the Zombie loses an arm, a leg or none after being attacked.
- Has a private method called dropLimb(String, Zombie, GameMap) that will get Exits of the zombie, and randomly drop the newly created limb on the adjacent location of the zombie. Then it will decides if the Zombie will drop the first weapon it currently has in its inventory if the Zombie only have one arm left after being attacked, or it will drop all the item in its inventory if the Zombie has no arm left.

Class modified: **AttackBehaviour**

Roles and responsibility:

- In the getAction method, 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: **Zombie**

Roles and responsibilities:

- Add ShoutBehaviour, PickupWeaponBehaviour and CrippledBehaviour to the private attribute Behaviours.
- Has a field of type int called arm to store the number of arms that are still attached to the zombie.

- Has a field of type int called leg to store the number of legs that are still attached to the zombie.
- Both field arm and leg are initialised to 2 to represent all zombie will have 2 arms and 2 legs upon creation.
- Has a field of type Boolean called isCrippled which is initialised to false.
- Has private loseArm(int) method that deduct the number of arm lost from its arm attribute if and only if arm – arm lost >= 0. Throw exception if the precondition is not satisfied.
- Has private loseLeg(int) method that deduct the number of leg lost from its leg attribute if and only if leg – leg lost >= 0. Throw exception if the precondition is not satisfied.
- Has public getArm() method that returns the number of arms that are still attached to the zombie.
- Has public getLeg() method that returns the number of legs that are still attached to the zombie.
- Has negateIsCrippled() method that negate the status of isCrippled when it's called.
- In the getIntrinsicWeapon() method, Bite action is implemented as an intrinsic weapon of zombie. Zombie has 50% probability of using bite instead of punch if the Zombie still have both arms. If the Zombie loses one arm and only have another one arm left, the probability of returning a punch is halved. In other words, the probability of returning bite action is increased to 75%. If the Zombie has no arm left, it will not return punch and will return bite 100% of the time.
- Has public zombieIsAttacked() method that will decides if the Zombie will lose a limb or not after being attack and deduct the Zombie's arm or leg attribute by the number of limb lost if the Zombie actually loses a limb. It will return a string that indicates the result of this action, so it will return "Arm", "Leg" or null.

Class added: **CrippledBehaviour**

Roles and responsibility:

- Inherit Behaviour class, and this behaviour will take into account the number of legs this zombie has, and the isCrippled attribute of zombie to determine if the zombie can move in that turn or not.

2. Zombie will have exactly 2 arms and 2 legs on creation.

- The number of arms and legs are hard-coded to 2 upon creation of any zombie. Although hard-coding is not a preferable practice in object-oriented design, however, this can be justified as the number of arms and legs are supposed to be a class variable and should be the same for all zombies when a new zombie object is created.

3. Any attacks on a Zombie that causes damage will have a chance to knock at least one of its limb off. If the Zombie loses one arm, the probability of punching is halved and has a 50% chance of dropping any weapon it is holding. If it loses both arms, it definitely drops any weapon it was holding. Lost limb drops onto the ground at an adjacent location to the Zombie.

- In the AttackZombieAction class, any attack on the Zombie will have a 60% successful hit rate. If the attack is successful, it will deduct the Zombie's hit points by the damage of the attack. Then, the execute function will call the zombiesAttack() method from Zombie class and determine if the attack results in the dropping of a limb from the Zombie. It will call the dropLimb(String, Zombie, GameMap) function to drop the limb at the adjacent location of the Zombie. The dropLimb(String, Zombie, GameMap) function will also determine if the Zombie should drop the weapon it's holding alongside with the limb. If the limb is indeed dropped by the Zombie, a new limb will be created and placed randomly at the adjacent location to the zombie. The dropLimb(String, Zombie, GameMap) function is created to increase the readability and avoid execute function to be too cluttered.
- The getIntrinsicWeapon in zombie class is modified so that it uses simple if-else based on the number of arms of the zombie before returning Punch or Bite, where both belong to type IntrinsicWeapon. If number of arm == 2, it's 50-50 for Punch and Bite, if number of arm == 1, its 25-75 for Punch and Bite. Lastly, if the Zombie has no arm, it will only return bite.

4. Loses one leg, movement speed is halved; loses both legs, it cannot move at all.

- The attribute Behaviours is an array of behaviours that are arranged such that the Zombie will have a 10% chance of shouting something every turn regardless of the

conditions the Zombie is in. Then it is followed by pick up weapon behavior and attack behavior, so that the zombie will pick up the weapon first before attacking a human nearby. The crippled behavior is placed right before both movement behaviors which are hunt behavior and wander behavior so that the Zombie will move in alternating turns if it has only one leg or cannot move at all if it has no leg.

- At each turn, zombie will go through its behaviours and return the first allowable action it can take. CrippledBehaviour is created to determine if the zombie can move in that turn or not based on the number of legs the zombie has. It is placed right before the two movement behaviours which are HuntBehaviour and WanderBehaviour so that the zombie can return DoNothingAction if it's not supposed to be moving during that turn, or return null if it should be able to move.
- If the number of legs is equal to 2, return null. If the number of leg is equal to 0, return DoNothingAction. Lastly, if it's equal to 1, it will use getIsCrippled() method from zombie class to determine if this zombie can move or not. Since the isCrippled attribute of a zombie is initialised to false, and the zombie shouldn't be able to move in the first turn after the zombie loses one leg, it will negate the isCrippled attribute of the zombie and return DoNothingAction. The next turn, the isCrippled attribute of the zombie is now true, and this zombie should be able to move, so it will negate the isCrippled again and return null.
- Since CrippledBehaviour is placed right before HuntBehaviour and WanderBehaviour so that it can interfere with the movement behaviour and return the appropriate action.

Crafting weapons

New class added: **CraftWeaponAction**

Roles and responsibilities:

- Inherits Action class so it can be used as an action by the player.
- Has a constructor CraftWeaponAction(Limb).
- In the execute() method, it will first remove this current weapon from the actor's inventory, then it will call this weapon's craftWeapon method which will return a

superior weapon. Finally, this superior weapon will be added into this actor's inventory.

- Has menuDescription() method that returns a string Original weapon + "is crafted into" + New weapon.

New class added: **ZombieClub**

Roles and responsibilities:

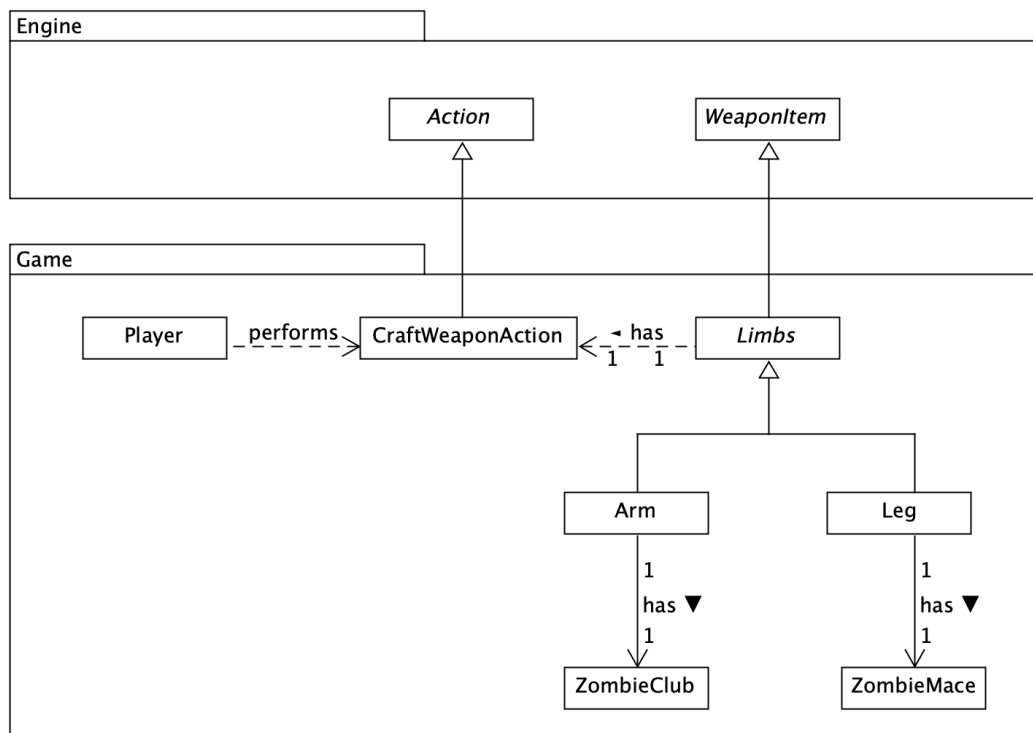
- Inherits WeaponItem class
- It is represented by the letter "B" when it's on the ground
- Has a damage of 35

New class added: **ZombieMace**

Roles and responsibilities:

- Inherits WeaponItem class
- It is represented by the letter "M" when it's on the ground
- Has a damage of 50

UML for Crafting Weapons



How to achieve the required functionality:

- Player holding an Arm or a Leg can craft it into Zombie club and Zombie mace respectively.**
 - Arm class has a private attribute of type *ZombieClub* which stores a new *ZombieClub* object. This *ZombieClub* is the successor of *Arm* and has a damage of 35.
 - Similarly, *Leg* class has a private attribute of type *ZombieMace* which stores a new *ZombieMace* object. This *ZombieMace* is the successor of *Leg* and has a damage of 50.
 - When the player has either an *Arm* or a *Leg* in the inventory, since both *Arm* and *Leg* has *CraftWeaponAction* in their allowableActions, the player will have a choice to craft them into better weapon respectively.
 - If *CraftWeaponAction* is executed, it will remove the weapon from the player's inventory, then it will call the *craftWeapon* method in *Arm* or *Leg* which will return a new weapon, either a *ZombieArm* or a *ZombieMace* depending on the weapon wished to be upgraded. Lastly, it will add this new weapon into the player's inventory.

Rising from the Dead

New class added: HumanCorpse

Roles and responsibilities:

- Extends Item abstract class to reduce code duplication as it has same functionality.
- It is used to represent dead Human which has health points ≤ 0 . If the target is Zombie type, it will create a new corpse item with displayChar='&' to represent dead zombies. when a Human is killed by Zombie, the HumanCorpse item is dropped where a new object is created of type HumanCorpse and added onto the map location.
- Has attributes String name = humanCorpse, char displayChar = 'X', portable = false.
- It has a new attribute of type int, countdown = 5-10, randomly generated between 5 to 10 turns, where this represents the number of turns before it spawns a Zombie at the original location.
- Modifies the method tick() in Item abstract class which decreases the countdown by 1 every time tick is called. When countdown reaches 0, calls spawnZombie() method.
- Has a new method spawnZombie() that create a zombie at the current location and remove the HumanCorpse object.

Farmers and Food

New class added: Farmer

Roles and responsibilities:

- Extends Human class as they share the same characteristics and abilities as human and therefore reduce the need of repeating codes.
- It has an attribute, Behaviour[] behaviours, which stores the Farmbehaviour and other behaviour that Human has.
- It uses the parent class constructor which is human to initialise 3 attributes; String name, char displaychar = 'F' , and int hitpoints= 50.
- has an @override playturn method that loops through the Behaviour[] behaviours and returns an action to be performed at that turn.

New class added: Farmbehaviour

Roles and responsibilities:

- Implements Behaviour interface as it is a new type of behaviour for Farmer.
- @Overrides the super class's getAction method in Behaviour Interface which returns an action such as SowAction, FertiliseAction, HarvestAction or Null if the Farmer cannot do these.
- It is used to create actions that Farmer can do such as sowing crops if Farmer actor is standing near a patch of dirt, which drops Crop item from inventory to the dirt ;Harvesting fully grown Wheat and drops it to the ground ;Fertilising crops to reduce the amount of turn required for it to be a fully grown Wheat.

New class added: SowAction

Roles and responsibilities:

- Extends Action abstract class as it is also a new type of action and therefore help to reduce duplication of code.
- Has an attribute of Exit exit which is the surrounding exit for Farmer to sow Crops.
- -@Overrides super class's execute method to sow crops by changing the Dirt into Crop which the Farmer performs and displays the result in the I/O.
- @Overrides super class's menuDescription method which returns a string of a description to be displayed in the menu.

New class added: FertiliseAction

Roles and responsibilities:

- Extends Action abstract class as it is also a new type of action that Farmer can do and therefore help to reduce duplication of code.
- It has attribute Crop crop, which is the crop that will be fertilised.
- -@Overrides super class's execute method to fertilise crops by decrementing the countdown of Crop so that it will ripe faster and displays the result in the I/O.
- @Overrides super class's menuDescription method which returns a string of a description to be displayed in the menu.

New class added: HarvestAction

Roles and responsibilities:

- Extends Action abstract class as it is a new type of action that Farmer can do and therefore help to reduce duplication of code.
- It has attribute Location harvestLocation, which is the location of the crop to be fertilised.
- -@Overrides super class's execute method to harvest crops by setting the location of the ground type back to dirt. If the actor is a player, it will create a Wheat item and add it into the Player's inventory. If the actor is a Farmer, it will drop the newly created Wheat item at the harvestLocation instead. This will then display the result in the I/O.
- @Overrides super class's menuDescription method which returns a string of a description to be displayed in the menu.

New class added: Food Interface

Roles and responsibilities:

- It is an interface that is used to determine whether an item can be used as a food.
- Has methods -public int hpRecover(), which is the amount of health points it will restore when being consumed by Human or Player.
- Has another method- public String verb(), which is the verb used when displaying the results after consuming this Food. For example, "chews" and "drinks".
-

New class added: FoodItem

Roles and responsibilities:

- Extends Item abstract class and implements Food interface.
- Contains 2 attributes- int hpRecover and String verb.
- Used to represent items that are used as a food that can be consumed to recover health points.
- Has 3 accessor methods heal(), verb() and name() which returns the amount of hit points recovered ,the verb to be used to display and the name of the item. For Example, "Player drinks milk and recovers for 20 hit points."

New class added: Crop

Roles and responsibilities:

- Extends Ground abstract class.
- Is an item with attributes, displaychar = 'c'.
- It also has an int countdown = 20, attribute that records the number of turns required to grow into a ripe crop which is Wheat. This countdown is updated using tick() method in Item abstract class.
- It has method getCountdown() which returns the int countdown.
- It also has method decrementCountdown(int number) which takes the number as input and decrease the countdown int variable of the crop object.
- @Overrides super class's tick() method in Item abstract class and decreases the countdown by 1 each time the tick() method is called. When the countdown attribute is 0, it will call cropRipens method
- Has a new cropRipens() method that changes the displayChar to 'C' to represent the crop is fully grown.
- @Overrides super class's allowableActions that checks if the adjacent location to the actor has a ripe crop and adds it into the Action actions and returns the new collection of actions.

New class added: Wheat

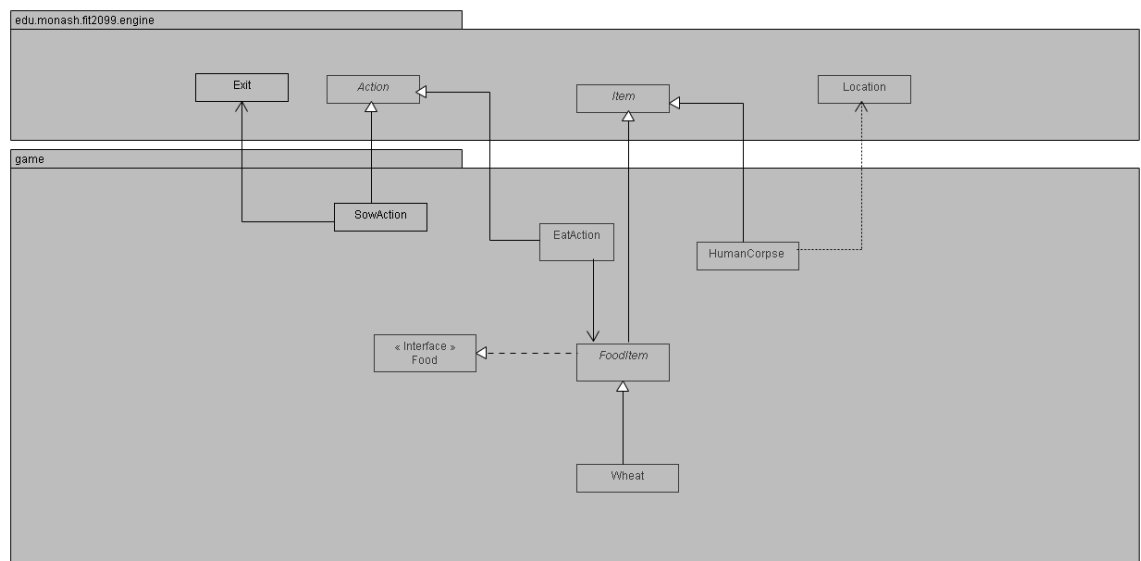
Roles and responsibilities:

- Extends fooditem abstract class.
- It is the type of food dropped when a ripe crop is harvested
- The Wheat Constructor initialise the attributes such as name = Wheat, displaychar = 'w', int hpRecover = 20, String verb= "chews".
- It also has allowableActions.add(new(eatAction(this)) so that this Wheat can be consumed by Player and Human.

New class added: EatAction

Roles and responsibilities:

- Extends Action abstract class and therefore can reduce duplication of code.
- It is an action that allows actor to consume Food.by removing it from an actor's or player's inventory and recover some health points of the actor.
- It has an attribute FoodItem fooditem which represents the FoodItem to be consumed.
- @Overrides the super class's execute method in Action to use the FoodItem by removing it from actor's inventory and heal the actor by the amount indicated by hpRecover, then returns a String to be displayed to user at the end of each turn.
- @Overrides menuDescription method which returns a String of a description to be displayed in the menu. For example, "Player consumes Wheat and recovered 20 health points."



edu.monash.ftt2099.engine

game

