

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: **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 pickUpWeaponBehaviour 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 attackableteam == 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:

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 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()` 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 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 "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.

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.
- Has an attribute of type weapon to store the weapon if this weapon is upgraded
- Has a final static field of type int called damage that stores the damage of this weapon which is equal to 20, and it's declared final static because it's a class variable and should be a constant for its child classes which is Arm and Leg
- 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 so it can be used by both zombie or the player
- Zombie should pick it up if zombie is stepping on it (has the same location as this Arm)
- 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.

- It has a possibility to be dropped (or rather created since zombie doesn't store object of Arm type as an attribute) at the adjacent location around the zombie when the zombie is attacked by the player.
- 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 so it can be used by both zombie or the player
- Zombie should pick it up if zombie is stepping on it (has the same location as this Leg)
- 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.
- It has a possibility to be dropped (or rather created since zombie doesn't store object of Arm type as an attribute) at the adjacent location around the zombie when the zombie is attacked by the player.
- 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 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 (DRY). It is currently part of the code of the execute function

New class added: **AttackZombieAction**

Roles and responsibilities:

- Inherits AttackAction and it represents an action that can be carried out by the Player to attack the zombie at adjacent location of the player.
- 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, the zombie has a 30% probability of losing a limb, and a 10% probability of losing 2 limbs (A limb can be either an arm or a leg and it's randomly chosen, so the zombie could potentially lose both arms or both legs after being hit).
- When a limb is knocked off, it will deduct the number of limbs knocked off from the zombie attribute arm or leg
- Has a private method called dropAt() that will get Exits of the zombie, and return a Location that is available for the Limb to drop.
- Has a private method called dropWeapon(int lost_limb) that take in the number of limb lost after the attack. If lost_limb == 1, it has 50% chance to return True, if lost_limb == 2, it return True.

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

Class modified: **Zombie**

Roles and responsibilities:

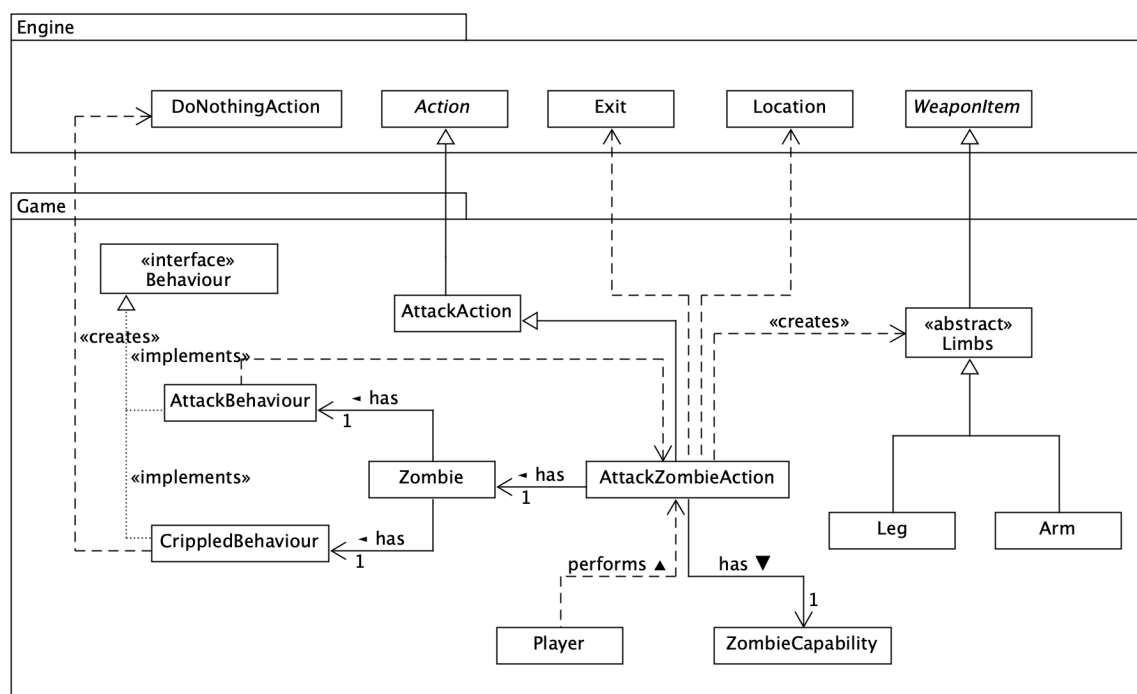
- Has a field of type `int` called `arm` to store the number of arms that are still attached to the zombie.
- Has a field of type `int` called `leg` to store the number of legs that are still attached to the zombie.
- Both field `arm` and `leg` are initialised to 2 to represent all zombie will have 2 arms and 2 legs upon creation.
- Has a field of type `Boolean` called `isCrippled` which is initialised to `false`.
- Has `loseArm(int armLost)` method that deduct `armLost` from its `arm` attribute if and only if `arm - armLost >= 0`.
- Has `loseLeg(int legLost)` method that deduct `legLost` from its `leg` attribute if and only if `leg - legLost >= 0`.
- Has `getArm()` method that returns the number of arms that are still attached to the zombie.
- Has `getLeg()` method that returns the number of legs that are still attached to the zombie.
- The `getIntrinsicWeapon()` method is modified so that it check the number of arm the zombie has before return `Bite` or `Punch`. If number of arm == 2, it has 50% of returning either `IntrinsicWeapon`, if number of arm == 1, it has 25% probability of returning `Punch` and 75% probability of returning `Bite`.
- Has `dropWeapon()` method that will drop the weapon it's currently holding, the weapon will be removed from the zombie's inventory and randomly dropped at the adjacent location around the zombie.
- Has `negateIsCrippled` method that negate the status of `isCrippled` when it's called.

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 or not.
- Has a getAction method that it will get the number of leg. of the zombie, and return either Null or DoNothingAction based on the number of legs.

UML for Beating up the Zombies



How to achieve the required functionality:

- In order to provide more details on the action of player attacking the zombie, AttackZombieAction class is created and inherits from AttackAction class, at the same time, AttackAction class has to be made abstract to increase its reusability and its child class can share some of the same functionalities without duplication of code. AttackAction is split into AttackZombieAction and AttackHumanAction so that they only handle one responsibility, hence Single Responsibility Principle (SRP) is followed.

- Limb class is an abstract class so that Arm and Leg can inherit it, and increase the code reusability since both Arm and Leg are quite similar with a key difference of what their upgraded weapon will be.
- In the AttackZombieAction class, there's a series of if-else in the execute function. Each if-else will have a predetermined probability of occurrence, for example, if the player successfully landed an attack on the zombie (70%), the zombie will have 30% probability of dropping one limb and 10% of dropping two limbs, each limb that is dropped can be an Arm (50%) or a Leg (50%).
- 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.
- 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.
- The dropWeapon method in AttackZombieAction class takes in the number of limb lost after the zombie being attack, and return a Boolean based on the number of limb lost. 50% returning true if limb lost == 1, 100% returning true if limb lost == 2. The execute function in AttackZombieAction will call this method and determine if the weapon will drop or not.
- 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 after AttackBehaviour, The zombie will return an AttackAction if it's allow to do so and the number of legs will not affect it.
- In AttackZombieAction, the method dropAt() will get the Exits of the zombie and return a Location where the limb can be placed at. Hence, this implies that the limbs will drop at the adjacent location to the zombie.
- When the player landed a successful hit on the zombie, and the zombie has dropped x number of arms and y number of legs, a matching number of x and y of Arm and Leg will be dropped on the ground. Since both Arm and Leg inherit from Limb, therefore they are a type of Weapon and has a damage of 20. It can be used by both player and zombie.

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(Actor, Item).
- 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 "C" when it's on the ground

- Has a damage of 35

New class added: **ZombieMace**

Roles and responsibilities:

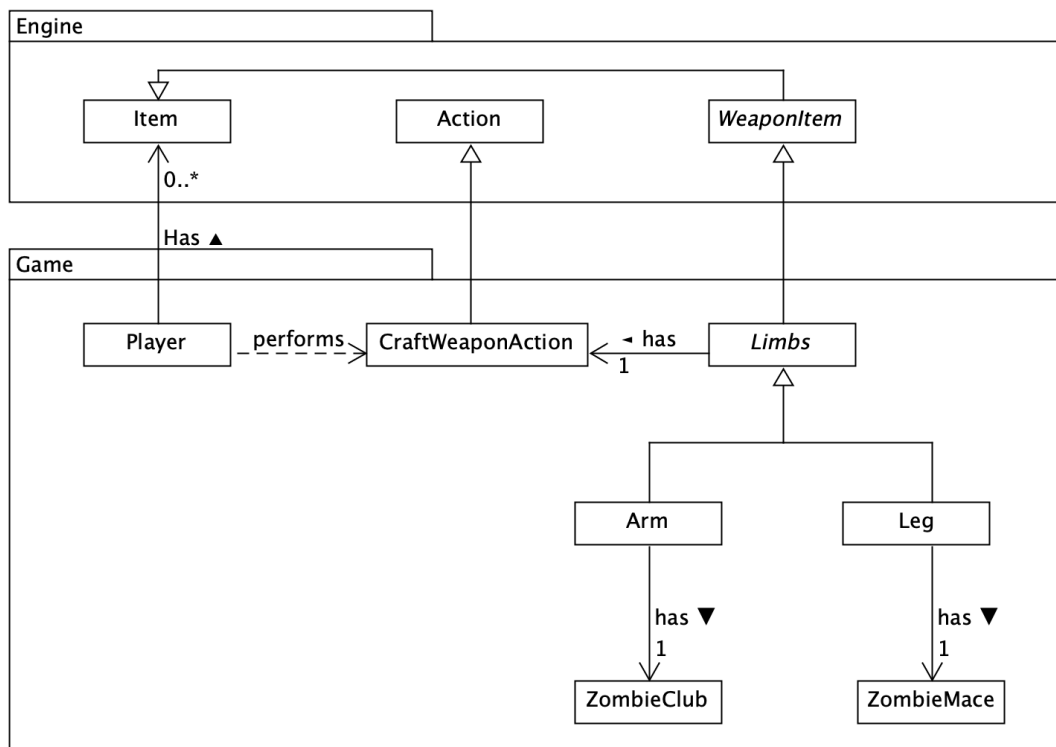
- Inherits WeaponItem class
- It is represented by the letter “M” when it’s on the ground
- Has a damage of 50

Class modified: **Limbs**

Roles and Responsibility:

- In the constructor, add CraftWeaponAction into its allowableActions since both Arm and Leg can be crafted into better weapon, putting the code here will reduce duplicity of code.

UML for Crafting Weapons



How to achieve the required functionality:

- 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 . This item is dropped when a Human is killed by Zombie by modifying the execute method in AttackAction class where the (!target.isConscious()) is false, 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 called counter of type int, 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 and removes the humanCorpse item from the map.

- Has a new method `spawnZombie()` that create a zombie at the current location using `addActor(Actor actor)` method in `Location` class, `addActor(new Zombie("ZombieName"))`

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 of `farmbehaviour` Class that allows Farmer to do their objectives in the game which is farming and crops.
- It uses a parent class constructor which is human to initialise 3 attributes; String name, char displaychar, and int hitpoints.
- Used to represent Farmer in the game and has an `@override` `playturn` method that returns an action to play from `farmbehaviour` class at a given turn.

New class added: Farmbehaviour

Roles and responsibilities:

- Implements Behaviour interface as it is a new type of behaviour for Farmer.
- 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.
- `@Overrides` `getaction` method in Behaviour Interface which returns a farmaction.

New class added: FarmAction

Roles and responsibilities:

- Extends Action abstract class as it is also a new type of action and therefore help to reduce duplication of code.

- @Override execute method which the Farmer performs an action and displays the result in the I/O.
- @Override 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 for all types of food items in the game.
- Has attributes -int heal, which is the amount of hit points it will restore when being consumed by Human or Player.
- Has an attribute- 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 attributes- int heal, String verb
- Used to represent items that are used as a food that can be consumed to recover health points.
- Has 2 accessor methods heal() and verb() which returns the amount of hit points recovered and the verb to be used to display. "Player drinks milk and recovers for 20 hit points."

New class added: Crop

Roles and responsibilities:

- Extends Item abstract class.
- Is an item with attributes name = Crop, displaychar = 'c', and portable = true.
- It also has an int countdown 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.

- @Overrides 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 spawn a Wheat in the same location.
- Has a new spawnwheat() method that drops a newly instantiated object of type Wheat on the location.
- Used by Farmer to sow crops.

New class added: Wheat

Roles and responsibilities:

- Extends fooditem abstract class
- Has attributes name = Wheat, displaychar = 'C', heal = 20, String = "consumes", portable = true -can be picked up by Player and Human to be used as food.

New class added: EatAction

Roles and responsibilities:

- Extends Action abstract class and therefore can reduce duplication of code.
- It is used to consume fooditem by removing it from an actor's or player's inventory and recover some health points of the actor.
- @Overrides the execute method in Action and 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."

