***\*\*Blue text is the new implementations\*\****

**Beating up the Zombies**

**Limb**

Extends the *WeaponItem* class. and itself is an abstract class which has Leg and Arm class as subclasses. Each *Zombie* objectcontains an *ArrayList<Limb>* with a length of 4 (2 of them are from *Leg* class and the other 2 are from *Arm* class).

**Advantages:** When zombies limbs are added to an ArrayList the type of the ArrayList can be limbs instead of Item. Furthermore, since the Limb class is an abstract class it applies the rule of DRY(don’t repeat yourself). Since the player can pick up Limbs and use them as normal clubs, this class extends the weapon item and the subclasses of this class will be considered as weapon items.

**AttackAction**

When an Actor object(as a target gets attacked) type is Zombie Object with a possibility of 25% one of the zombies limbs will be dropped. Also, when a target dies using an if statement (instanceof syntax) to get the type of the Actor who died. If the actor is a human it assigns the corpse to PortableItem which means after 5 turns, that corpse will rise from the ground as a Zombie object. And if the Zombie object dies, the corpse variable will be assigned as a PortableItemZombie object which means that the zombie is dead till the end of the game. Furthermore, in this class another if statement is used when the zombie attack is successful and if the zombie object uses a bite attack, the zombie object will be healed for 5 hit points.

**Advantages**: This class is modified because it is the only class which shows the changes where an actor attacks another actor. And it can be modified to create changes in the GameMap where an actor dies.(rise of dead humans from the ground and change that type to zombies.)

**Zombie:**

1. In the getIntrinsicWeapon() method, bite attack has been added with a probability of 50%. For better implementation, since if a zombie loses its arm the chance of punching will be ⅓ and chance of biting will be ⅔, therefore the probability changes and because of that, the value of probability for punching will be assigned to a variable which makes it easier for us to change that value.
2. When a player attacks a zombie by a chance of 25% a limb of a zombie drops, with respect to that limb the zombie behaviour changes which they operate in each turn. For instance, if a leg drops, the speed halves. Or if two legs drop the zombie cannot move. For implementing these scenarios, if/elif statements can be used to state what change in behaviour can happen to the zombie (inside of playTurn() method in zombie class).

**Advantages:** Instead of creating different classes to represent the zombies with lost limbs, it's implemented in the zombie classes and those cases have been separated using if statements and since Zombies are the only actors which got limbs so creating if statements is not against DRY.

**PickUpBehaviour:**

This class is using the PickUpItemAction object to let zombies pick up an item that is in their current location. Since they use behaviour in their playTurn() method, this behaviour will be added as the first behaviour object to Zombie’s behaviour array. Because in playTurn() one of the behaviour’s actions will be executed, it's more interesting for the zombie to pickup the Weapon in one turn which gives them more damage and it gives one turn to humans to run away from zombies.

**Crafting weapons**

**Leg**

It is a class that extends the Limb class which means Leg object itself is considered as a WeaponItem since Limb has extended WeaponItem. (Limb class’ subclass) which represent the Leg of a zombie object which they can be dropped if zombie objects get attacked (with the possibility of 25%) also, this class has one allowable action which is CraftAction() which means if a player is in Leg location or if player has picked up the Leg, the player can use the Leg as a Mace() which itself is a weaponItem. and it has more damage than normal Limb object and Club weapon.

**Advantage1:** Arms and Legs got separate classes since they used as different weapons when a player craft them (Leg is crafted to the Mace and Arm is crafted to the Club)

**EscapeBehaviour:**This behavior should be added to the Human behaviour list. This behaviour contains a range attribute. The getAction() method of this class will get all of the exit direction from Exit class for a Human which from human location till the range (range attribute) does not contains a zombie and when these exit direction where collected these directions will be shuffled and it returns the move action of the location from the first direction which was collected before. The implementation is quite similar to the HuntBehaviour class though in this case the Human should find a way which does not contain zombies in some certain range.

**Arm**

It is a class that extends the Limb class which means the Arm object itself is considered as a WeaponItem since Limb has extended WeaponItem. (Limb class’ subclass) it represents the Arm of a zombie object which they can be dropped if zombie objects get attacked (with the possibility of 25%) also, this class has one allowable action which is CraftAction() which means if a player is in Arm location or if player has picked up the Arm, the player can use the Leg as a Club() which itself is a weapon Item. it has more damage than the simple Leg and Arm which is used as Weapons.

**Advantages:** Arms and Legs got separate classes since they used as different weapons when a player craft them (Leg is crafted to the Mace and Arm is crafted to the Club)

**Club**

This class has extended WeaponItem which has the same type of the Arm which the player might have in its inventory.

It is a class in which a player can craft an Arm weapon to the Club object (which is a weapon item too) which has greater damage rather than Arm.

**Mace**

This class has extended WeaponItem which has the same type of the Leg object which the player might have in its inventory.

It is a class which a player can craft an Arm weapon to the Club object (which is a weapon item too) which has greater damage rather than Leg

**CraftAction:**

CraftAction is a subclass of Action class which shows that if a player has Leg in its inventory, the Leg can be crafted as a Mace weapon and if the player has Arm in its inventory, that Arm can be crafted and used as a Club.

The way that the execute function of this action works is by removing the leg/arm weapon item from the player's inventory and adding mace/club weapon item to player’s inventory.

**Advantage1:** Instead of creating two CraftAction one for Leg and one for Arm, the craft action has two if statements which decide which Limbs are crafted to which weapon(mace or club). CraftAction is an action that has been appended to the Leg and Arm actions list. Which means if a player has Leg and Arm in their inventory the option in the menu will be written that the player can craft Leg/Arm weapon to Mace/Club.

**Advantage2:** In leg and Arm Item CraftAction objects have been added to their allowableAction field which means when the player picks up legs/arms the option will be added to the player's menu that they can craft them afterwards.

**Rising from the dead**

**PortableItem**

If a Human object dies after 5 to 10 turns the human corpse will be changed to a Zombie object with Zombie behaviour.

**Advantages:** This method is useful when it is shown that the dead human can rise from the ground as a zombie object.

**Disadvantages:** For improvement PortableItem and PortableItemZombie can be written in one class and the if condition for human death and zombie death can be justified by if statements.

**PortableItemZombie**

This class is made to differentiate the difference between Human death and Zombie death.

When the zombie Object dies, till the end of the game, the dead zombie will remain dead which means this class does not implement the Override method of tick().

**Farmers and food**

**Farmer**

Extends the *Human* class. *Farmer* objects have a *HarvestBehaviour* object that allows the *Farmer* to perform the *HarvestAction*.

**Advantages:** *Farmer* class can be extended further into different types of farmer sowing different types of *UnripeCrop* which heals for different amounts.

**HarvestBehaviour**

Implements the *Behaviour* interface. Only *Farmer* objects (*Farmer* objects are type *Human too*) have a *HarvestBehaviour* object.

**HarvestAction**

Extends the *Action* class. *HarvestAction* allows a *Farmer* object to harvest a *RipeCrop* and instantiates a *Food* object and adds it into its inventory (if the object that is performing the action is a *Player* object) or drops it on the ground (if the object that is performing the action is a *Farmer* object).

**Advantages:** When the *HarvestAction* object is added into *ripen crop*’s *allowableAction* field, the option for the *Player* to harvest a *RipeCrop* will be added into the menu.

**HarvestAction**

It is no longer added into the *Food*’s *allowableAction* field. It is now created in *addHarvestAction(Actions actions, Actor actor, GameMap map)* method in *ActorInterface* and called in the *Player* class to give the *Player* the menu option to harvest ripe crop around them.

**FertilizeBehaviour**

Implements the *Behaviour* interface. Only *Farmer* objects have a *FertilizeBehaviour* object. It allows a *Farmer* to perform *FertilizeAction* that fertilizes an Unripe Cropthat the *Farmer* is standing on.

**FertilizeAction**

Extends the *Action* class. *FertilizeAction* changes the age of the *UnripeCrop* object to decrease the time for it to ripe, given that there is an *UnripeCrop* object that the *Farmer* object is standing on.

**Advantages:** When the FertilizeAction object is added into *Player*’s *allowableAction* field, the option for the *Player* to fertilize an *UnripeCrop* will be added into the menu.

**SowBehaviour**

Implements the *Behaviour* interface. Only *Farmer* objects have a *SowBehaviour* object. It allows a *Farmer* to perform *SowAction* that sow an *UnripeCrop* on a patch of dirt, given that there is a patch of dirt next to the *Farmer*.

**SowAction**

Extends the *Action* class. *SowAction* has a 33% probability of successfully sowing a UnripeCrop on a patch of dirt given that the *Farmer* object performing the action is next to a patch of dirt.

**Advantages:** When the *SowAction* object is added into *Player*’s *allowableAction* field, the option for the *Player* to sow on a patch of dirt next to them will be added into the menu.

**EatFoodBehaviour**

Implements the *Behaviour* interface. Only *Human* objects have an *EatFoodBehaviour*. Only *Human* objects have an *EatFoodBehaviour* object. It allows *Human* objects to perform *EatFoodAction* that consumes food and heals the *Human* object.

**EatFoodAction**

Extends the *Action* class. *EatFoodAction* removes a *Food* object in a *Human* object’s inventory and heals the *Human* for a certain amount of health points.

**Advantages:** When *EatFoodAction* object is added into *Food allowableAction* field, the option for the *Player* to eat food in their inventory will be added into the menu, given that they have a *Food* object in their inventory.

**EatFoodAction**

It is no longer added into the *Food*’s *allowableAction* field. It is now created in *addEatFoodAction(Actions actions, Actor actor)* method in *ActorInterface* and called in the *Player* class to give the *Player* the menu option to eat food.

**UnripeCrop**

Extends the *Item* class. *UnripeCrop* interacts with *FertilizeAction* because *FertilizeAction* will decrease the *age* of the *UnripeCrop* object. *UnripeCrop* cannot be harvested and can only be fertilized by *Farmer*s.

**Advantages:** This class is initiated in the *SowAction* class.

**RipeCrop**

Extends the *Item* class. A *RipeCrop* object is instantiated within an *UnripeCrop* object (when *UnripeCrop* object age > 20). It replaces a *UnripeCrop* object on the map with *RipeCrop*.

**Advantages:** This class is separated from the *UnripeCrop* class because when the player and a farmer want to harvest food from a *RipeCrop*, in the code “instanceof” syntax can be used for *Farmer* and the *Player* to see if the *item* in actor’s location is a *RipeCrop* or not.

**Crop (new class implemented to replace UnripeCrop and RipeCrop classes)**

Extends the *Item* class. *Crop* class will have a boolean variable that determines whether the *Crop* is ripe or not, the boolean variable will be changed according to the *Crop*’s age variable.

**Food**

Extends the *Item* class. *Food* objects are instantiated and either added into *Player* inventory (if the object is not *Farmer* type) or dropped to the ground (if the object is not *Player* type and is *Farmer* type) when a *RipeCrop* is harvested by a *Farmer*.