**Group Name: Team Julian  
Members: Julian Yong Hao, Khong Jun Ming**

**Zombie**Zombie is inherited from ZombieActor class. This class will represent a Zombie with Zombie behaviours.  
New Attributes: private int countLegs = 2, private int countArms = 2

Design Rationale:  
In the Zombie class, we will introduce two Integer attributes which are legs and arms to keep track the number of them each Zombie has. These two attributes are declared as private by the principle “Minimize dependencies that cross encapsulation boundaries”. We did not choose to make this public as doing this will make these two attributes exposed to the rest of the world and this will be a security breech where other classes can access its values and change them which we don’t want it to happen. We also have instantiated two list of ZombieLegs and ZombieArms objects. This is to reduce dependencies. There will be two new Behaviours added to the Behaviour array. These behaviours are: PickUpBehaviour and SayBehaviour. Both behaviours implement the Behaviour interface. In the PickUpItemBehaviour, it will check whether the location the zombie is standing exist a weapon, and if there is it will return a new PickUpItemAction. On the other hand, the SayBehaviour, will determine whether the Zombie will say “Braaaaains” which in 10% probability and if it happens it will return a new SayAction. Other than that, we will also add two new classes that inherits from the IntrinsicWeapon class which are Bite and Punch. Either one of these classes will be returned at the Overrided getIntrinsicWeapon method to let the Zombie bite or punch depending on whether the Zombie still have arms. We also Overrided the hurt method, to let the Zombie have the ability to loose its limbs after a successful attack. The lost limbs will be dropped on the adjacent location of the Zombie. The isConscious method will be Overrided and check whether it has enough limbs to continue staying alive. In the playTurn method, if the Zombie has only one leg then, we will use the lastAction to find out whether did the Zombie move on the previous action. if it moved on its last action then this turn it can’t choose to move.

**SayAction**SayAction is inherited from the DoNothingAction class. This Action will prints out the String “Braaaaains”.

Design Rationale  
This class is inherited from the DoNothingAction as both performs similar work and as the principle states “Don’t Repeat Yourself”. Therefore, there will not be repeated code.

**ZombieAttackAction**ZombieAttackAction is inherited from AttackAction. This class will only perform Zombie attacks

Design Rationale  
This class is inherited from the AttackAction class because the original AttackAction class is shared/being used by Zombie and Player but both classes have different conditions of attacking a target. Therefore, by the principle “Don’t Repeat Yourself”, we inherit ZombieAttackAction and inherit code that are essential for attacking a target while still implementing conditions of attacking for each Zombie and Player. Therefore, some of the code needed on both classes will be segregated out to another method in the AttackAction class, thus reducing duplicated code. This class will check whether a Zombie bite is successful and heal the Zombie and also halve the probability of punching if it looses one arm.

**AttackAction**We will add code to this class so that when Player attacks the Zombie.

Design RationaleTo avoid excessive repeated codes, If the target is dead, we would create an PortableItem object which inherited from Item, so it could implement the tick() method to keep track of turns. We choose not to create another class that’s specifically for this implementation, as we want to avoid excessive implementation of same class. After 5-10 turns, we create a zombie object correspond to the Corpse.

**ZombieLimb**This class is an abstract class.

**Design Rationale**This class provides some Overrided methods that helps implement the CraftWeaponAction. Subclasses which inherits this class will inherits all non-private methods, Therefore, reducing duplication of code when necessary similar methods needed to be used on different subclasses. There are overrided methods here: 1) asWeapon, this method will check whether the object still have a enum Capability of ZombieLimb, if there is, it cant be used as a weapon until it has a enum Capability of Weapon. 2) removeCapability, when this method is called by the CraftWeaponAction, this method will remove the original enum Capability of ZombieLimb and rename the this object to its respective weapon name.

**ZombieLeg and ZombieArm**Both these classes inherits the ZombieLimb class

Design Rationale  
When Zombie is hurt and falls off its limb, we take reference of weapons of ZombieLeg and ZombieArm which inherits from WeaponItem from that Zombie, this reduces the effort to recreate those 2 weapons and avoided dependency. To craft the dropped ZombieLeg and ZombieArm into a weapon, there is a CraftWeaponAction. This action will be added to the Player’s allowable action when the player picked that limb up. Therefore, the player will be able to craft the limb to a usable weapon.

**CraftWeaponAction**Allows the Player to craft a ZombieLeg or Zombie Arm to a usable weapon

Design Rationale  
This class is inherited from the Action class. Basically, this class changes the enum Capability of the ZombieLeg or ZombieArm to a enum Capability of Weapon. Therefore, by doing this, allowing the Player to use that ZombieArm or ZombieLimb as a weapon.

**HuntBehaviour**HuntBehaviour implements the Behaviour interface. The Zombie has this behaviour to chase a Human or Player.

Design RationaleThis class is not changed.

**WanderBehaviour**WanderBehaviour implements the Behaviour interface. The actor has this behaviour to move around

Design RationaleThis class is not changed.

**Human**Human is a class that is inherited from the ZombieActor class. This class will have Behaviours: WanderBehaviour, RunAwayBehaviour, HealBehaviour.

Design Rationale  
Human will have additional Behaviours like RunAwayBehaviour and HealBehaviour. To avoid excessive repeated codes , we decided to made both classes implement the Behaviour Interface. RunAwayBehaviour will detect if there’s Zombie nearby and move away from the Zombie . In this particular design, we did not implement another action for running away, we keep the dependencies minimal, so the action is done within RunAwayBehaviour just like the implementation of WanderBehaviour. Then, HealBehaviour will check whether there is harvested Food on the current location and if there is it, HealAction will be executed provided the Human is damaged. Human will only check its current location and not adjacent location. This design will allow Player has higher probability to pick up Food.

**Farmer**Farmer is a new class that is inherited from the Human class. This class will represent a Farmer with Farmer behaviours

Design Rationale  
We notice that Farmer does what a Human does, so we extend the Human class to create Farmer which not only have Human’s behaviours but also a few new Farmer’s behaviours.

**Dirt**Dirt is inherited from the Ground class. We did not change much about dirt, except adding some enum DirtCapability which allows Dirt to do certain things latter.

Design Rationale  
To make Dirt has different capability to do certain things like to be sowed, fertilized, harvested. To minimise the efforts, we use a simplest way to implement this, that’s with enum constants.

**SowBehaviour** and **SowAction**SowBehaviour will check if the adjacent Dirt of the Farmer is “Sow-able” and if it is, SowAction will sow a crop on the Dirt.

**FertilizeBehaviour** and **FertilizeAction**FertilzeBehaviour will check if the Dirt the Farmer is standing is able to be fertilized and if it is, FertilizeAction will decrease the turns needed for crop to be ripen.

**HarvestBehaviour** and **HarvestAction**HarvestBehaviour will check if the Dirt the Farmer is standing or the adjacent Dirt is able to be harvested and if it is, HarvestAction will create a PortableItem item named Food which allows actors other than Zombie to heal if use

Design Rationale  
To reduce the efforts to maintain we make the new behaviours compatible with other behaviours, we made these classes implement the Behaviour interface. As the names suggest, we made these classes responsible for its particular action. Then, since the behaviours are separated clearly, it’s less costly to change. Similar to above, the actions are implemented in these ways. Lastly, this design allows these classes be responsible for their own properties.

**Crop**  
Crop is created when a Farmer sows on a Dirt. It’s able to keep track of turns.

Design Rationale  
We simply do not want repeated codes, so we made Crop inherits from Item. To reduce dependencies, we did not create Food PortableItem immediately when Crop ripens.

**Bite & Punch**These two classes are types of IntrinsicWeapon for the Zombie

Design Rational  
These classes are inherited from the IntrinsicWeapon, therefore inherited those non-private methods which reducing duplication of code. On the other hand, we did this choice as we wanted to reduce the use of literals. As these two classes have constant input values, this will reduce the need of the developer to type in the input values whenever needed to create an instance of these classes.

**Steel**This class is a basic weapon that is placed on the map, just like the Plank

Design Rational  
By creating this class, reduces the need to type in literals as input parameters when creating an instance of it. Thus, avoiding the excessive use of literals.