Design Rationale

Class Name: FoodItem

Responsibilities:

- This class inherits from Item class and has an association relationship with ItemAbleToPickUp enum class
- Has a class attribute health that stores the amount of health point gain from the food item
- The constructor has a parameter status which indicates the enum status of the FoodItem and call the method addCapability() from Item class which adds the capability of the status
- The method getHealth() returns amount of health point of the food

Class Name: Mace

Responsibilities:

- This class inherits from WeaponItem class and has an association relationship with ItemAbleToPickUp enum class
- Mace does not have a certain damage for now but will be able to deal more damage than a zombie limb
- It is crafted by only using a zombie leg
- Mace can only be crafted by a player but if dropped can be picked up by a zombie or player
- Has a Enum status Weapons, it is to indicate that this is a weapon, which can be picked up by a zombies

Class Name: Club

Responsibilities:

- This class inherits from WeaponItem class and has an association relationship with ItemAbleToPickUp enum class
- Club does not have a certain damage for now but will be able to deal more damage than a zombie limb
- It is crafted by only using a zombie arm
- Club can only be crafted by a player but if dropped can be picked up by a zombie or player

- Has Enum status “Weapons”, it is to indicate that this is a weapon, which can be picked up by a zombies

Class Name: ZombieArm
Responsibilities:

- This class inherits from WeaponItem

- When a zombie’s arm gets knocked off, it will create an instance of this class, which is dropped on the ground

- Can be picked up by Player, but not a zombie. It does not have Enum status “Weapons”, as it is a weapon, but it should not be identified as a weapon by any zombie, thus it does not have a status

- Has an Enum Ingredient, indicating that is an ingredient used to make a potion

Class Name: ZombieLeg
Responsibilities:

- This class inherits from WeaponItem

- When a zombie’s leg gets knocked off, it will create an instance of this class, which is dropped on the ground

- Zombie leg when dropped by a zombie can be picked up by the player but not zombie and is used as a weapon

- Can be picked up by Player, but not a zombie. It does not have Enum status “Weapons”, as it is a weapon, but it should not be identified as a weapon by any zombie, thus it does not have a status

- Has an Enum Ingredient, indicating that is an ingredient used to make a potion

Class Name: Limbs
Responsibilities:

- Has a class attribute called limbCount to store the number of limbs for a zombie

- The setLimb() method is to decrease the amount of limbs of a zombie when having its limbs are knocked off

- The getLimb() method returns the limbCount, the amount of limb a zombie currently has

- It is created as the amount of limbs of a zombie need to be tracked, and it needs to be decreased depending on probability when it is attacked. It is also an abstract class as there are two kinds of limbs, both sharing the same methods (Arms and Legs)

Class Name: Arms
Responsibilities:

- This class inherits from Limbs, when arm is dropped to the ground, it becomes a ZombieLimb object, which is a weapon that can be picked up and used by the player

Class Name: Legs
Responsibilities:

- This class inherits from Limbs, when leg is dropped to the ground, it becomes a ZombieLimb object, which is a weapon that can be picked up and used by the player

ClassName: Zombie
Additional role:

- Zombie now has new attributes called arms and legs which allows movement and actions for the zombie

- Has two new Random and integer attributes used for creating and handling probability

- Has a reference to a GameMap object so that it can know its location, allowing it to drop limbs onto the location it is currently at

- Two constructors are created, one including GameMap as a parameter, one without. The one without GameMap is used in Corpse class, to create a zombie at the location which was already specified, hence it does not need the GameMap reference. Whereas the ones that require GameMap reference is for creating zombies in the Application class.

Responsibilities:

- It has an association relationship with Arms and Legs class as it requires arms to perform attack and legs to move

- The getIntrinsicWeapon() has the ability to return a new IntrinsicWeapon class but with different parameters. Depending on the probability, which is 50-50 for both, it either creates a “bite” or “punch” weapon

- The groaningSound() method allows zombie to say words like “Braaaaains” at every turn depending on probability of 10%

- In the overridden hurt method, the features of beating up zombies were implemented. There are 4 conditions and each condition performs slightly different tasks. When an arm is to be knocked off and the zombie has 2 arms left, it has a 50% chance of dropping a weapon it is holding. When a leg is to be knocked off and the zombie has 2 legs left, its movement speed is halved. When an arm is to be knocked off and the zombie has 1 arm left, it will drop all the weapons it has. When a leg is to be knocked off and the zombie has 1 leg left, it will not be able to move at all. Therefore 4 loops were created for the conditions.

- In the playTurn method. The movement speed of the zombie depends on skipTurn. skipTurn is initialized as 0. If the zombie loses one of its legs, skipTurn becomes 1. Then skipTurn is checked, if it is an odd number, the zombie will not move. If it is an even number, the zombie will move. If the zombie loses both of its legs, skipTurn becomes -1. And if skipTurn is checked to be -1, the zombie will not be able to move for the rest of the game.

Class Name: Farmer
Responsibilities:

- This class inherits from Human class as it has the same capabilities as human

- It has an association relationship with FertilizeBehaviour which allows the farmer to perform actions on fertilizing the crop, SowBehaviour which allows the farmer to perform actions on sowing a crop and HarvestBehaviour which allows the farmer to perform actions on harvesting the crop

- Farmer class now has an ArrayList of behaviours instead of using an Array. This is because in the overridden playTurn() method it will allow to clear and add the behaviours to the list and calls Collections.shuffle(behaviours) to shuffle the list so that farmer has a chance to perform every single behaviour instead of the behaviour in the first index all the time.

Class Name: Crop
Responsibilities:

- This class inherits from the class Ground

- This class has class attributes age where it stores the age of the crop and ripe where it will store boolean status for ripe, true if it is ripe, false otherwise

- Has a overridden tick() method which check the age of the crop and if age is 20 it will change the display character of the crop to ‘R’

Class Name: HarvestBehaviour
Responsibilities:

- A class has a dependency relationship with HarvestAction as it depends on HarvestAction and its method

- The overridden getAction() method checks if the current Actor is standing next to a ripe crop and if so, returns a new HarvestAction

Class Name: SowBehaviour
Responsibilities:

- A class has a dependency relationship with SowAction as it depends on SowAction and its method

- The overridden method getAction() checks if the actor is standing next to a patch of dirt and the location does not contain an Actor and if so, returns a new SowAction

Class Name: FertilizeBehaviour
Responsibilities:
- A class has a dependency relationship with FertilizeAction as it depends on FertilizeAction and its method

- The overridden getAction() method checks if the actor is standing next to an unripe crop and if so, returns a new FertilizeAction

Class Name: HarvestAction
Responsibilities:
- This class inherits from Action class and has an association relationship with Crop class

- Has a class attribute ripeCrop which indicates the location of the crop

- Action for Farmer to harvest the crop where it will remove the crop object from the ground by creating a new dirt object on the same ground and then creates a FoodItem object and drops it onto the ground at the location of the crop in the overridden execute() method

Class Name: SowAction
Responsibilities:
- This class inherits from Action class and has an association relationship with Crop class

- Has a class attribute dirt which indicates the location of the dirt

- Action for Farmer to sow the crop on a patch of dirt and in the overridden execute() method has a variable rand_num which creates a random generator to determine the probability of sowing a crop which is 33% which will then be able to create a Crop object on the location of the dirt

Class Name: FertilizeAction
Responsibilities:
- This class inherits from Action class and has an association relationship with Crop class

- Has a class attribute unripeCrop which indicates a crop object

- Action for Farmer to fertilize an unripe crop where it will increase the age of the crop by 10 so it will call the class attribute age of the crop from Crop class and add 10 to it in the overridden execute() method

Class Name: Player
Additional role:

- Allows player to perform punch attack, weapon attack, harvest crop, crafting zombie weapons, consuming food, planting seeds, reviving corpses, rangedWeapon attack, reloading ammunition for rangedWeapon

Responsibilities:

- Has a dependency relationship with PlayerCraftAction that allows player to perform action on crafting a weapon, ReviveAction that allows player to revive a corpse with a potion, OfferAction that allows player to offer ingredients to a Sorcerer to create a potion, PlantAction that allows player to plant seeds on a dirt, HarvestAction that allows player to harvest ripe crops.

- Has a dependency relationship with RangedAttackAction that allows the player to perform action on attacking with rangedWeapons such as shotgun and sniper, ReloadAction that allows the player to reload ammunition for rangedWeapons.

- In its playTurn, it is able to harvest food if it is standing on or next to a ripe crop as well as crafting limbs if it has any in its inventory into a mace or club. It can also revive a corpse that is nearby provided it has a potion, plant seeds on a dirt it is standing on.

- Has a new Actions object in method playTurn() called rangedActions that stores only RangedAttackAction and creates another menu with rangedActions as the parameter which is a submenu for when using sniper

- Has a method isInsideCircle() that returns boolean value if the zombie is within the circle area which is the sniper's range

Class Name: PlayerCraftAction
Responsibilities:

- This class inherits from Action class

- Action for player to craft weapon since only player is able to do crafting which is able to craft out a mace or a club depending on the limb that the player is holding

- If the weapon the Actor has is an instance of Zombie Arm, a Club object is created. Else if the weapon is an instance of Zombie Leg, a Mace object is created. After crafting, the limb used to craft the weapon (Zombie Arm or Leg) is removed from the inventory and the newly crafted weapon (Club or Mace) is added into the inventory

Class Name: AttackAction
Responsibilities:

- This class inherits from Action class

- In its overridden execute method, the chances of attack for zombie and Player were implemented. Player has a higher chance to attack, to ensure that Player doesn't die too quickly otherwise the game would end in a few turns. It has an 80% chance of a successful attack. Whereas a zombie has a 50-50 chance for a bite or punch attack. If the target to be attacked is a Human and it is not conscious, indicating that the Human is dead, a Corpse object is created and placed on the location of the Human. Otherwise, if the target is a zombie and it is not conscious, it is dead but a Corpse object will not be created to ensure that zombies are not immortal. Sorcerer has a lower chance to miss his attack (10%)

Class Name: PortableItem
Responsibilities:

- This class inherits from Item class

- It is a base class for any item that can be picked up and dropped.

Class Name: Corpse
Responsibilities:

- This class inherits from PortableItem class

- Has class attributes turn which indicates how many turn it has been after an actor turns into a Corpse in integer and rise which indicates the number of turns it takes to make a Corpse rise in integer

- Has a overridden tick() method which will add the turn by 1 every time it is called and if the turn matches the rise it will create a new Zombie object at the location of the Corpse

Class Name: ConsumeBehaviour
Responsibilities:

- A class has a dependency relationship with ConsumeAction as it depends on ConsumeAction and its method

- The overridden getAction() method checks whether the location of the character has a food item by checking the enum status of item and if it matches the enum keyword it will return a new ConsumeAction

Class Name: ConsumeAction
Responsibilities:

- This class inherits from Action class and has an association relationship with FoodItem class

- Has a class attribute food which indicates a FoodItem object

- Action for humans or player to be able to consume FoodItem object by checking whether the actor is conscious and if so it will heal itself by calling the getHealth() method from FoodItem class

Class Name: ItemAbleToPickUp
Responsibilities:
- This is an enumerator class and has 5 enum keywords which is Weapons, Food, Seed, Ingredient and Potion

Class Name: Plank
Responsibilities:
- This class has an association relationship with ItemAbleToPickUp enum class

- The constructor now has a parameter status which indicates the enum status of the Plank and call the method addCapability() from Item class which adds the capability of the status

Class Name: MarieSpawn
Responsibilities:
- This class acts as a location to spawn Mambo Marie

- Actors cannot enter, only Mambo Marie can be spawned on it, but it will not be able to re-enter once it walks out of the spawn location

- It is able to keep track of the status of Mambo Marie, remove it once it reaches 30 turns, or checks whether it is still alive or not. By using boolean conditions, if it is dead, then it stays dead for the rest of the game. Otherwise, it will always have a 5% chance of spawning every turn

Class Name: MamboMarie
Responsibilities:
- This class represents Mambo Marie

- Has an association relationship with AttackBehaviour, WanderBehaviour and ChantBehaviour. ChantBehaviour creates a ChantAction which allows Mambo Marie to randomly spawn 5 zombies across the map

- It is similar with a Zombie in a way where it attacks Actors with ZombieCapability Enum of Alive, the difference is that it does not hunt or pick up weapons

- To check how many turns to chant, turns start from 1. From turns 1-10 (inclusive) and 12-21 (inclusive) Mambo Marie only wanders around, until the 11th and 22nd turn, where it stops moving and chants.

Class Name: ChantBehaviour
Responsibilities:

- A class has a dependency relationship with ChantAction as it depends on ChantAction and its method

- The overridden getAction() method creates 5 random coordinates and checks whether the location is occupied by an actor, or fence. If it is occupied, it will check the exits of the location, which are the 8 locations surrounding it. It is very unlikely that the 8 locations are all occupied, therefore there will always be a location that is available. Each coordinate is stored in an ArrayList which is then stored in the bigger, main ArrayList. The main ArrayList will then be passed on into ChantAction.

Class Name: ChantAction
Responsibilities:

- This class inherits from Action class

- Has a list of names for the zombies that are to be created. There are 50 names to reduce the chances of two zombies having the same name.

- Action for Mambo Marie to spawn 5 zombies randomly across the map.

Class Name: Seed
Responsibilities:

- This abstract class inherits from Item class

- It is a base class for all types of seeds which are then inherited by Maple and Pine Seed classes

Class Name: MapleSeed
Responsibilities:

- This class inherits from Seed class, has two capabilities Seed and Ingredient, which can be planted on a dirt or made into a potion

Class Name: PineSeed
Responsibilities:

- This class inherits from Seed class, has two capabilities Seed and Ingredient, which can be planted on a dirt or made into a potion

Class Name: MapleTree
Responsibilities:

- This class inherits Ground, it is exactly the same as the original Tree class, except with new features

- Once it reaches the age of 25 or above, it has a 0.1% chance of dropping a Maple seed at every turn. It has such a low chance as there are a lot of trees in the town map.

Class Name: PineTree
Responsibilities:

- This class inherits Ground, it is exactly the same as the original Tree class, except with new features

- Once it reaches the age of 25 or above, it has a 0.1% chance of dropping a Pine seed at every turn. It has such a low chance as there are a lot of trees in the town map.

Class Name: RevivePotion
Responsibilities:

- This class inherits Item, it acts as a potion which can be used to revive corpses. It has the capability Potion, to indicate that it is a potion

Class Name: ReviveAction
Responsibilities:

- This class inherits from Action class

- It allows the Player to revive a corpse, only when the Player is around the corpse. It will not be able to revive it if the Player is standing on the location of the corpse. Once the corpse is revived and turn back into a Human, the potion is then consumed and is removed from the Player's inventory

- Action for Player to revive a corpse using a potion

Class Name: PlantAction
Responsibilities:

- This class inherits from Action class

- Action that allows Player to plant any kind of seed on dirt on either of the maps.

Class Name: OfferAction
Responsibilities:

- This class inherits from Action class

- Action that allows Player to drop ingredients on the Sorcerer's location as an offer for it to create a potion in exchange

- There are four loops to remove each ingredient separately from the Player's inventory. Since .getInventory() returns an unmodifiable list, it cannot be modified while it is being looped through using a for loop. Therefore, when an item in the inventory is found to be a type of an ingredient, break the for loop and then add the item on the Sorcerer's location and remove the item from the Player's inventory.

Class Name: Sorcerer
Responsibilities:

- This class represents a Sorcerer

- It does not inherit from Human class as it is not a Human. Also, it does not consume food and does not move. It can only pick up items with Enum Ingredient and Potion. It can attack, similarly to a Player, only towards Zombies and Mambo Marie.

- It does not have a weapon, as it only uses its Intrinsic Weapon. Since it is a Sorcerer, it uses fire blasts which deal a huge amount of damage, basically one-shotting any Zombie/Mambo Marie herself.
- In its playTurn method, there are four loops to check whether the Sorcerer's inventory has enough ingredients to create a potion. Each loop is for an ingredient. If the Sorcerer has enough ingredients, it is able to create a potion and drop it at the Player's location, else it cannot.

Class Name: DropPotionBehaviour
Responsibilities:

- A class has a dependency relationship with DropPotionAction as it depends on DropPotionAction and its method

- In its constructor, it has a parameter which is the Player's location

- The overridden getAction() method returns a new DropPotionAction with the Player's location

Class Name: DropPotionAction
Responsibilities:

- This class inherits from the Action class

- In its execute method, a new RevivePotion is created and dropped on the Player's location. The Sorcerer will only drop the potion if the Player is nearby, otherwise it will not drop anything. It has four loops to remove each ingredient separately from the Sorcerer's inventory. Exactly the same as Player's OfferAction. Since .getInventory() returns an unmodifiable list, it cannot be modified while it is being looped through using a for loop. Therefore, when an item in the inventory is found to be a type of an ingredient, break the for loop and remove the item from the Sorcerer's inventory.

Class Name: RangedAttackAction
Responsibilities:

- This class inherits from the Action class

- Has class attribute shotgun and sniper which indicates the current Shotgun object and current sniper object

- Has class attribute target which indicate the current actor that is within the range of the shotgun

- Has class attribute zombie which indicate the current zombie that is within the range of the sniper

- Has class attribute locationToShoot which is the current location where the shotgun is shooting and direction which is the direction in which the shotgun is pointing

- Has class attribute rand which creates a random probability of a given range

- Has an arrayList locationOfActors which stores every single actor that is currently on the map except for player

- Has two constructor whereby one is for shotgun and another for sniper with different parameter inputs

- Has a method areaOfTriangle() which calculates with area of the triangle which is the range of the shotgun given 3 different points on the map whereby one is the location in which the shotgun is shooting and the other two points are North-East and North-West and is 2 squares away from where the location the player is shooting.

- Has a method isInsideTriangle() which returns boolean value of whether the target actor is standing within the range of the area of the triangle that has been calculated from areaOfTriangle method().

- In the overridden execute() method. If weapon is a shotgun, it will call the method areaOfTriangle() to get the range where the player is shooting and then loop through the arrayList locationOfActors to check for any actor that is the range by calling the method isInsideTriangle(). If weapon is a sniper, if the player is holding a sniper it will read user input with respective instruction given in the submenu to either aim, shoot, change target or drop the weapon.

ClassName: ReloadAction
Responsibilities:

- This class inherits from the Action class

- Has class attribute shotgun and sniper which indicates the current Shotgun object and current sniper object

- Has class attribute ammo which indicates the current ammo item for shotgun

- Has class attribute bullet which indicates the current bullet item for sniper

- In the overridden execute method allows the player reload ammo for shotgun and bullet for sniper

ClassName: Shotgun
Responsibilities:

- This class inherits from Item class
- Has class attribute ammo which indicates the amount of ammo that the shotgun has
- Has method reduceAmmo() that reduces the ammo by 1
- Has method reloadAmmo() that increase the ammo by the amount stated in the parameter ammoAmount

ClassName: Sniper
Responsibilities:

- This class inherits from Item class
- Has class attribute bullet which indicates the amount of bullet that the sniper has
- Has method reduceBullet() that reduces bullet by 1
- Has method reloadBullet() that increases the bullet by the amount stated in the parameter bulletAmount

ClassName: ShotgunAmmunition
Responsibilities:

- This class inherits from Item class
- Has class attribute ammoAmount with initial int value of 5 which reloads a shotgun ammo by this amount

ClassName: SniperAmmunition
Responsibilities:

- This class inherits from Item class
- Has class attribute bulletAmount with initial int value of 5 which reloads a sniper bullet by this amount

ClassName: ExitGameAction
Responsibilities:

- This class inherits from Action class
- In the overridden execute() method calls System.exit(0) which terminates the program therefore ending the game

ClassName: EndGame
Responsibilities:

- This class inherits from World class

- Has class attributes humans and zombies which contains boolean values

- In the overridden method stillRunning() checks for all the maps that if all humans or farmers died then the game ends and player loses and checks if all the zombies and mambo marie dies then the game ends and player wins

- Override method for endGameMessage() to return a string depending on player wins or loses