Class Responsibilities

Modified Classes

Zombie

Attributes:

- *behaviours*
    - Type: Behaviour[]
    - An array of Behaviour objects where the Zombie will prioritise performing them in the following order:
        - SpeakBehaviour
        - PickUpWeaponBehaviour
        - AttackBehaviour
        - HuntBehaviour
        - WanderBehaviour
    - So firstly, the Zombie would say “Braaaaains”. If not, they will attempt to pick up a weapon at their location. If not, they will attempt to attack a nearby Human. If not, it will get closer to a nearby Human. If not, it will wander around.
- *limbs*
    - Type: ArrayList<Limb>
    - An array containing 4 Limb objects where 2 are the Zombie’s arms and the other 2 are the Zombie’s legs.
    - The limbs are stored as an attribute of Zombie and not stored into the Zombie’s inventory because Limb objects are a subclass of WeaponItem as stated in the specifications. If the limbs were in the Zombie’s inventory, the Zombie will use their limbs as weapons which is wrong.
- *numberOfArms* and *numberOfLegs*
    - Type: int
    - The number of arms and legs the Zombie has respectively.
    - Both are initialised to 2.
- *turn*
    - Type: int
    - The turn count. It is used to check if the Zombie is allowed to move at a particular turn given they lost a leg.
    - Initialised to 0.
- *rand*
    - Type: Random
    - Used for making the Zombie randomly pick a bite or punch attack.
- *biteChance*
    - Type: int
    - Used alongside *rand* to have a 50-50 chance of the Zombie either punching or biting.
    - Initialised to 50.

- *additionalEffects*
    - Type: Actions
    - A list of additional Actions to execute by the Zombie when it is attacked by the Player such as dropping their Limb on the map.

Methods:

- *getIntrinsicWeapon()*
    - *If rand.nextInt(100) < biteChance*, then Zombie bites, else it punches. (Both bite and punch are intrinsic attacks of Zombie)
- *playTurn()*
    - On every turn, for every Action, if it is not null, it also checks if the Zombie is allowed to use that Action in the case when it does not have arms or legs.
- *attackMissed()*
    - Overridden method from ActorInterface.
    - Checks if the Zombie uses a bite attack, the chance of missing the attack is 90%. For any other kind of attack, the chance of missing is 50%.
- *bonusEffect()*
    - Overridden method from ActorInterface.
    - If a Zombie successfully bites a Human, they recover 5 HP. Otherwise, it executes any Actions in the *additionalEffects* list which could be dropping a limb or a weapon.
- *hurt()*
    - When a Zombie takes damage, after depleting their HP, there is a 25% chance for them to lose a random Limb. The corresponding limb's DropItemAction is added into the *additionalEffects* list.
    - Calls *weakenZombie()* to change the Zombie's properties further.
- *weakenZombie()*
    - Private method.
    - If the Limb removed was an arm, *numberOfArms--*, *biteChance + 25%* (caps at 100%) and there is a 50% chance for the Zombie to drop a weapon. Any weapons dropped will have their DropItemAction added into the *additionalEffects* list.
    - If the Limb removed was a leg, *numberOfLegs--*.
- *canMove()*
    - Private method. Called in *playTurn()*.
    - Checks if the Action the Zombie wants to perform is a MoveActorAction. If so, it checks if the *numberOfLegs < 2* and *turn % 2 == 1* or *numberOfLegs < 1*. If so, the Zombie cannot perform the Action. Otherwise, it can perform the Action.
- *canPickUp()*
    - Private method. Called in *playTurn()*.
    - Checks if the Action the Zombie wants to perform is a PickUpItemAction. If so, it checks if the Zombie already has a weapon equipped and *numberOfArms < 1*. If so, the Zombie cannot perform the Action. Otherwise, it can perform the Action.

AttackAction
Methods:

- *execute()*
    - To check if an Actor misses their attack, it calls their *attackMissed()* method.
    - It also calls *bonusEffect()* for the attacker and the target where the respective Actors will handle any features that affect them in their own class such as healing, dropping weapons, etc. (Tell, Don't Ask)
    - When an actor dies:
        - If its a Human that died then will create a HumanCorpse class PortableItem
        - If its a Zombie that died, just place a corpse at the location

Human
Methods:

- *attackMissed()*
    - Overridden method from ActorInterface.
    - Humans have 30% chance of missing an attack unlike Zombies.

Player

- *playTurn()*
    - Checks through the Player's inventory. If there is a CraftableItem found, add a CraftAction for the Player to be able to perform.
    - Adds a QuitGameAction so that the Player can quit the game at any time.

ActorInterface

- Added two default methods: *attackMissed()* and *bonusEffect()*.
- *attackMissed()* by default returns false, meaning an Actor never misses.
- *bonusEffect()* by default returns an empty string, meaning an Actor has no bonus effects.

Application

- Instead of initializing a World object, a subclass of World, that is ZombieWorld is initialized instead. This is because ZombieWorld has additional methods to check on certain win/lose conditions such as if all Humans are dead or all Zombies are dead.
- A new town map is initialized with Humans and Zombies in it.
- Mambo Marie is initialized.
- A Vehicle is initialized for the compound map and the town map.

New Classes

SpeakBehaviour

- 1st priority on the Zombie's *behaviours* array.
- A simple Behaviour class that returns a SayBrainsAction object 10% of the time to allow the Zombie to say "Braaaaains", otherwise it returns null like all other Behaviour classes.

Constructor:

- *SpeakBehaviour(sentence, speakChance)*
- *SpeakBehaviour(sentence)*

Attributes:

- *sentence*
    - Type: String
    - The sentence that the Actor will speak.
- *speakChance*
    - Type: int
    - Defaulted to 100. It is the probability that the Actor will speak the sentence.

SpeakAction

- Inherits the Action class.
- Returns a string of the sentence that the Actor says.

Constructor:

- *SpeakAction(sentence)*

Attributes:

- *sentence*
    - Type: String
    - The sentence that the Actor will speak.

PickUpWeaponBehaviour

- 2nd priority on the Zombie's *behaviours* array right after SpeakBehaviour.
- Uses *map.locationOf(actor).getItems()* to get all the Item objects at the location and randomly picks one of them given the item is a Weapon.
- We can check whether the Actor can pick up an item on the ground if it is a weapon using the asWeapon() method.
- If there are no weapons on the ground, return null. Otherwise, return a PickUpItemAction object.

CraftableItem

- Inherits the WeaponItem class.
- Has a WeaponItem attribute which is the weapon that is to be crafted into.

Limb

- Inherits the CraftableItem class since it is an item that can be crafted into a ZombieClub or ZombieMace.
- Zombie limbs are weapons on their own according to the specifications.

ZombieClub

- Inherits the WeaponItem class.
- A weapon only obtainable after crafting a Zombie's arm.
- Code structure is the same as the Plank class (only contains a constructor).

ZombieMace

- Inherits the WeaponItem class.
- A weapon only obtainable after crafting a Zombie's leg.
- Code structure is the same as the Plank class (only contains a constructor).
- Deals more damage than the ZombieClub.

CraftAction

- Inherits the Action class.
- Has a CraftableItem attribute which is the item that is to be crafted into something else.
- An Action where the Player can perform given they are holding a Zombie arm or leg.
- The Player can craft a Zombie arm into a ZombieClub object and a Zombie leg into a ZombieMace object.

Constructor:

- Takes in 1 CraftableItem object.

Attributes:

- *item*
    - Type: CraftableItem
    - The item to be converted into a weapon.
    - After crafting the weapon, this item will be removed from the actor's inventory.

Methods:

- *execute()*
    - If the item is a Zombie arm:
        - Add a ZombieClub object into the Player's inventory.
        - Then remove the Zombie arm from the Player's inventory.
    - If the item is a Zombie leg:
        - Add a ZombieMace object into the Player's inventory.
        - Then remove the Zombie arm from the Player's inventory.

HumanCorpse

- Inherits the Item class
- Has an integer attribute "turns" to keep track of the number of turns

- After 5 turns, the corpse will turn into a Zombie using the addCorpseZombie() method.
- Idea would be to remove the item at the location or in an actor's inventory if it is picked up and spawn on the location or at a location adjacent to the human/player respectively.

Constructor:

- Takes in a String as a name, char as a display character and a Location of the corpse.

Attributes:

- *turns*
    - Type: Integer
    - Used to keep track of the number of turns after the HumanCorpse class is created

Methods:

- *tick(Location)*
    - Called every turn by default which will be used in this class to add a Zombie at the Location **if the item is not picked up**.
- *tick(Location, Actor)*
    - Called every turn by default **if the item is picked up** by the Player which will be used in this class to add a Zombie at the adjacent location of the Player.
- *addCorpseZombie*
    - To add the Zombie to a Location after 5 turns since the creation of the HumanCorpse class
    - It checks if the item has been picked up or not
        - If it has not been picked up:
            - Remove the item from the ground
            - Spawn a Zombie at the location
        - If the item has been picked up by the Player:
            - Remove the item from the Player's inventory
            - Spawn a Zombie at an adjacent location
- *checkLocation*
    - To check if the location doesn't contain an actor
    - If it does, then return an adjacent location

Farmer

- Inherits the Human class
- Has an array of behaviours available for a Farmer

Constructor:

- Takes a String as a name and a char as a displayChar
- This constructor is inherited from the Human class in order to have the same health and capability properties as a Human

Attributes:

- *behaviours*
    - Type: Behaviour[]
    - List of available behaviours for a Farmer

Methods:

- *playTurn*
    - To return the Action for each behaviour
    - If all actions are null, then return DoNothingAction()

SowingBehaviour

- Implements the Behaviour interface

Attributes:

- *random*
    - Type: Random
    - To obtain a random to be used to obtain a random value for probability

Methods:

- *getAction()*
    - Get adjacent locations, if the location has a display character '.'
        - Probability of sowing a crop is 33%
            - If decided to crop, return new SowingAction()
    - Else
        - Return null

SowingAction

- Extends the Action class

Attributes:

- *location*
    - Type: Location
    - To record the location of the ground to sow a crop
- *crop*
    - Type: Crop
    - Represent a new Crop to be sowed

Constructors:

- Takes in a Location

Methods:

- *execute()*
    - Set the ground to be a new Crop()
    - Return a menuDescription()
- *menuDescription()*
    - Return a String

Crop

- Extends Ground class

Attributes:

- *age*
    - Type: int
    - To record the number of turns after a Crop() has been created

Constructor:

- Sets the display character for a crop

Methods:

- *tick()*
    - Takes in a Location parameter
    - Increases age by 1 every time its called
    - If the age reaches 20
        - Change the display character to signify that it is a ripe crop
- *fertilize()*
    - Adds 10 to the age of the crop

FertilizeBehaviour

- Implements the Behaviour interface

Methods:

- *getAction()*
    - If the ground at the location of the actor is an unripe crop
        - Return a new FertilizeAction()
    - Else
        - Return null

FertilizeAction

- Extends the Action class

Attributes:

- *crop*
    - Type: Crop
    - To represent the Crop to be fertilized

Methods:

- *execute()*
    - Calls the Crop's fertilize() method to fertilize the crop
    - Return a menuDescription()
- *menuDescription()*

        - Returns a String

HarvestBehaviour

- Implements the Behaviour interface

Methods:

- *getAction()*
    - If the actor is a farmer
        - If the adjacent locations has a ripe crop
            - Return HarvestAction()
    - If its a Player
        - Check if player is on a ripe crop
            - Return HarvestAction()

HarvestAction

- Extends the Action class

Attributes:

- *location*
    - Type: Location
    - The get the location to turn the crop into a Food item
- *dirt*
    - Type: Dirt
    - For setting the Ground back to Dirt
- *food*
    - Type: Food
    - A food item to be added into a player's inventory or on the Ground

Constructors:

- Has 1 parameter of Location

Methods:

- *execute()*
    - If its a farmer
        - Set the ground back to dirt and add item and the location
    - If its a player
        - Set the ground back to dirt and add item into location

Food

- Extends the Item class

Attributes:

- *verb*
    - Type: String
    - To get the return string to output after Food is eaten
- *healValue*
    - Type: int

    - To record the heal value of the Food item.

Constructor:

- Takes 2 parameters.
- A name and a healValue

Methods:

- *getHealValue()*
    - Return healValue
- *verb()*
    - Return the verb String

PickUpFoodBehaviour

- Implements the interface Behaviour

Methods:

- *getAction()*
    - Get the items at the location
    - If the item at the location is a Food item, then return a PickUpItemAction()

EatBehaviour

- Implements the interface Behaviour

Attributes:

- *inventory*
    - Type: List<Item>
    - A list of Item objects

Methods:

- *getAction()*
    - Get the inventory of the actor
    - Loop through the items of the actor
        - If the inventory has the item that is a Food
            - Return new EatAction()

EatAction

- Extends the Action class

Attributes:

- *food*
    - Type: Food
    - The food Item to be eaten

Constructor:

- Takes in 1 parameter

Methods:

- *execute()*
    - Heal the actor of the healValue of the Food
    - Remove item from the inventory
    - Return menuDescription()
- *menuDescription()*
    - Returns a String output that an actor ate the food

Vehicle

- Extends the Item class

Constructor:

- No parameters
- super("vehicle", '^', false);

Methods:

- *addAction()*
    - Inspired from the code in the demo mars package.
    - Adds an Action into the allowableActions list.
    - Application will add a MoveActorAction into the Vehicle so that the Player can travel to the town and back to the compound map respectively.

PlayerGround

- Extends the Ground class.
- A Ground to place a Vehicle Item on.
- Only the Player can enter.
- Blocks thrown objects.

MamboMarie

Attributes:

- *behaviours*
    - An array of Behaviour objects which are AttackBehaviour and WanderBehaviour
- *spawnableMaps*
    - An ArrayList of GameMaps where Mambo Marie can appear in
- *hidingMap*
    - A GameMap where Mambo Marie can hide in (basically she to make her exist in the game but cannot be seen by the Player to interact)
- *appeared*
    - A boolean which tells if Mambo Marie has appeared on the map or not.
    - Initialized as false.
- *turn*
    - A turn count to keep track of when she can summon Zombies and when she has to vanish.

Constructor:

- Takes in 2 parameters:
    - A list of GameMaps where Mambo Marie can appear in.
    - A GameMap for her to hide from the game field away from the Player as well as being able to exist in the game.

Methods:

- *playTurn()*
    - If Mambo Marie has not appeared in one of the maps, there is a 5% chance for her to appear.
        - If she does appear, the *appear()* method is called that returns an AppearAction.
    - Else if she has already appeared on a map but her turn count is less than or equals to 30, she can chant and summon 5 Zombies every 10 turns, otherwise she will attempt to attack a nearby Human or wander around.
    - Else, a vanish() method is called which returns a HideAction. This is called if Mambo Marie already spent 30 turns appearing on the map or she is still in hiding.
- *appear()*
    - Randomly picks a GameMap from *spawnableMaps* and moves Mambo Marie from the *hidingMap* to that map.
    - *appeared* is set to true.
    - An AppearAction object is returned which displays a message that Mambo Marie has appeared and can be interacted in the game.
- *vanish()*
    - Private method.
    - Moves Mambo Marie back to *hidingMap*.
    - Resets her *turn* count to 0.
    - *appeared* is set back to false.
    - A HideAction is returned which displays a message that Mambo Marie is nowhere to be seen on the map to be interacted.

SummonZombieAction

Attributes:

- *numberOfZombies*
    - An int which is the number of Zombies to spawn on the map.
- *zombieName*
    - The names of these Zombies that will be spawned.
- *verb*
    - The verb used by the Actor that summons the Zombies
    - e.g. “Mambo Marie **chants** and summons 5 zombies!”

Constructor:

- Takes in 5 parameters: The number of Zombies, the Zombie name, and the verb that describes the summoning action.

Methods:

- *execute()*
    - Spawns as many Zombies as according to the *numberOfZombies* value given.
    - Returns a String description saying the Actor has summoned Zombies.

AppearAction

- Inherits the Action class.
- A simple class that returns a String that an Actor has appeared on the map.

HideAction

- Inherits the Action class.
- A simple class that returns a String that an Actor is nowhere to be seen on the map.

QuitGameAction

- Inherits the Action class.
- An Action that is added to the Player so that they are given the option to quit the game at any time.
- How it works is it removes the Actor from the game. The game ends if the Player is not found on any map.

ZombieWorld

- Inherits the World class.

Methods:

- *stillRunning()*
    - Overridden method.
    - Instead of only checking if the Player exists, it also checks if there are still Humans existing and Zombies existing on the map.
    - As such, it checks like this:
        - return playerAlive() && humansAlive() && zombiesAlive()
    - If the Player dies, the game ends where the Player loses.
    - If all Humans are dead, the game ends where the Player loses.
    - If all Zombies including Mambo Marie are dead, the game ends where the Player wins.
- *playerAlive()*
    - Returns a boolean whether *actorLocations* contains the Player.
- *humansAlive()*
    - Iterates through every Actor in *actorLocations* and checks if the Actor has an ALIVE ZombieCapability and is not the Player, return true.
    - If every Actor is iterated and none of them do not have the ALIVE ZombieCapability (aside from the Player), that means all Humans are dead, so return false.
- *zombiesAlive()*
    - Iterates through every Actor in *actorLocations* and checks if the Actor has an UNDEAD ZombieCapability, return true.

        - If every Actor is iterated and none of them do not have the UNDEAD ZombieCapability, that means all Zombies are dead, so return false.
- *endGameMessage()*
    - Change the end game message to have additional texts telling the Player whether they lost or won.
    - If all Humans are dead, the Player loses.
    - If all Zombies including Mambo Marie are dead, the Player wins.

ChooseTargetMenu

- To display a menu for choosing which target to shoot at when using the Sniper

Methods:

- execute()
    - Gets all the actors which are Zombies and Mambo Marie
    - It only gets all the actor in the map the player is currently on

ChoosingTargetAction

- To display the choice to aim at the target or to shoot straight away

Methods:

- execute()
    - Firstly it will track whether is the player is aiming at a target previously
    - If there is a target previously, it will display the line where the player has lost concentration on the previous target.
    - Then set the player's target to the new target
    - Then add the AimAction and ShootingSniperAction to be displayed on the menu of options

ShootingSniperAction

- This class to handle the shooting of the shotgun to deal the damage to an actor

Method:

- execute()
    - It will shoot the target using AttackAction
    - Then decrease number of ammo left in the player's inventory by 1 in the ShotgunAmmunition class

Sniper

- The new Sniper Weapon

Pair

- A simple data structure replicating a 2-tuple
- Stores an x and y attribute which are integers
- Accessible through getters (getX and getY)

ShootingShotgunAction

- The action class to handle the shooting of the shotgun to deal the area of damage

Methods:

- execute()
    - To implement the damage based on the coordinates in the list of pairs in the array according to the name of the direction to shoot the shotgun.
    - If the location is not out of the map and there is an actor at the location, call AttackAction to implement the damage

ShootingShotgunMenu

- The class to output a new submenu to choose which direction to fire the shotgun.
- It will get all valid locations around the player to avoid shooting out of bound outside the map.

Shotgun

- The weapon class to represent a shotgun

AimAction

- An action class that will be called to increase the aim count of the player on the target

ShotgunAmmunition

- A class that extends Item that represents an Shotgun ammo stack
- The method shotOnce will decrease the amount of ammo left
- tick() will then check if its empty at 0, if it is then delete the item from the player's inventory

SniperAmmunition

- A class that extends Item that represents an Sniper ammo stack
- The method shotOnce will decrease the amount of ammo left
- tick() will then check if its empty at 0, if it is then delete the item from the player's inventory