**Design Rationale**

**Crafting Weapons: ZombieClub and ZombieMace**

A Player now has the capability to craft weapons when holding a zombie limb. To detect if a Player is in possession of a limb and thus able to craft a weapon, we added an additional behaviour to the Player class titled CraftBehaviour. This class implements the Behaviour interface.

CraftBehaviour can determine a Player’s inveotry using the getInvetory method. If the Player is holding a ZombieLimb, the Player can execute CraftAction. CraftAction is inherited from Action. This removes the instance of ZombieLimb and creates an instance of the ZombieWeapon class and puts it in their possession.

CraftAction can create two different instances of ZombieWeapon depending on the instance of ZombieLimb the Player is holding.

ZombieClub is crafted when the Player is holding ZombieArm and ZombieMace is crafted when the Player is holding ZombieLeg. They are inherited from WeaponItem. ZombieClub has damage points of 30, ZombieMace has damage points of 40.

This implementation was deemed most appropriate. ZombieClub and ZombieMace share the same attributes so it is appropriate that they are instantiated from a single class we have titled ZombieWeapon. Since only the Player has the capability to craft weapons it is appropriate that only the Player receives an additional Behaviour and not Humans or Zombies.

**Creating the Farmer**

The Farmer character is an extension of a Human with the additional capability to sow crops, fertilize crops and harvest crops for food.

To achieve this, we’ve:

* Created a class Farmer that extends Human
* Created the FarmerBehaviour class which the Farmer has access to
* Created classes FertilizeAction, HarvestAction and SowingAction allowing Farmers to execute these capabilities

To action the Farmer’s capabilities, we need to know if they are in range of dirt, crops, or food. The FarmerBehaviour class implements the Behaviour interface and is dependent on the Location class. This allows the Farmer to detect if it can execute the appropriate actions based on its Location.

To interact with crops and food we needed to create classes that the Farmer can interact with.

**Creating the Crop and Food Class**

The Crop class is instantiated when a Farmer successfully sows a crop on a patch of dirt. Left alone it will ripen in 20 turns or a Farmer can fertilize it to speed up the aging progress by 10 turns. Crop has an age attribute to track how many turns since it was instantiated, a tick method to increment its age and a displayChar attribute that is dependent on if the Crop is deemed ripe or unripe.

Crop is an extension of the Ground class, just like Dirt and Tree. This is appropriate as it cannot be picked up like an Item, it needs to be displayed on the map and interacted with by Farmers.

When a Crop is considered ripe it can be harvested by a Farmer, creating an instance of the Food class. Food can be picked up by Players and thus is an extension of the Item class. It will have an attribute displayChar ‘F’ so that Players can recognise it on the map.

Now that these classes exist, we can create actions so that the Farmers and Players can interact with the.

**SowingAction**

When standing next to a patch of dirt, a Farmer has a 33% chance of sowing a crop in it. The FarmerBehaviour class detects if the Player is in range of an instance of Dirt, and if this is true executes SowingAction.

SowingAction is inherited from the Action class. This occurs on every Farmer’s turn. When successful a Crop class is instantiated, and the Location of the Dirt is instead set to Crop. To achieve this SowingAction has a setGround(Crop) method

*\*\*\* Assumption: Farmer has a 33% chance of sowing a crop onto every dirt instance it is next to per turn*

SowingAction was decided not to be used as the Farmer’s action in playTurn(). This is because the majority of the GroundMap’s Ground is Dirt and it is likely that the Farmer will be surrounded by up to 8 instances of Dirt for most of the game. This would mean their movement around the map and interaction with Zombies would be limited as it is likely every turn, they would sow a Crop. Instead FarmerBehaviour loops through all the locations of Dirt in range and executes SowingAction on each. The Farmer is then able to use their playTurn() to move around, attack Zombies and/or pick up Items.

**FertilizeAction**

To fertilize an unripe crop the class FertilizeAction was created. Now when the Farmer is standing on an unripe crop (Crop with age attribute < 20) it can fertilize it, decreasing the time left to ripen by 10 turns (increase Crop age by 10).

To achieve this FertilizeAcion has a fertilize method that increase the Crop’s age attribute by 10.

FertilizeAction is an extension of the Action class in the Engine package. To check if a Farmer can execute FertilizeAction the FarmerBehaviour class loops through the surrounded locations and if it is method setGround returns Crop, FertilizeAction can be executed.

**HarvestAction**

To harvest a ripe crop the class HarvestAction was created. Now when standing on or next to a ripe crop (Crop age => 20, a Farmer can harvest it for food. If a Farmer harvests the food, it is dropped to the ground.

Similarly, to FertilizeAction, the FarmerBehaviour class loops through the surrounding Farmers location using the getGround method. If a Crop with age => 20 returns, then a Farmer can execute HarvestAction.

HarvestAction is an extension of the Action class. To achieve this there is method to create a Food instance at the Crops location. HarvestAction also uses the setGround method to remove the instance of Crop and reset the ground to Dirt.

*\*\*\* Assumption: We have interpreted this feature as:*

* *If a Farmer harvests a ripe crop, the ripe crop is replaced with food and is dropped to the ground*
* *A player can harvest food, placing it in the player’s inventory*
* *A player cannot harvest a ripe crop*
* *We were unsure if ripe crop = food, so we have said they are not the same and implemented the above*

*\*\*\* Assumption: After crop is harvest, ground returns to dirt*

**FarmerBehaviour**

In summary FarmerBehaviour loops through the Farmers location and surrounding locations, checking for instances of Dirt and Crops. Dependent on the results the Farmer can execute certain actions as explained above in SowingAction, FertilizeAction and HarvestAction.

FarmerBehaviour inherits the Behaviour interface and is dependent on Location.

**Players, Humans and Food**

*\*\*\* Assumption: Features state that only a Player can harvest food and place it inventory, but next feature mentions food can be eaten by damaged humans. We have assumed Humans are thus able to pick up food and store in inventory (therefore also farmers and players).*

*\*\*\* Assumption: A Human can harvest food (pick up) if standing on or next to food.*

Humans can harvest food and as a result store the item in their inventory. No additional classes, associations or dependencies are needed to include this. Humans can use existing methods including but not limited to getPickUpAction, addItemToInventory and removeItem.

To eat food, the human must be damaged and have food in their inventory. To determine if this criterion is met the EatFoodBehaviour class was created. EatFoodBehaviour retrieves the Humans hitPoints and checks if it is below what Humans are initialsed as (50). It also retrieves the list of Items in the Humans inventory using the getInventory method.

If the criteria are met, the Human can eat the food by executing the EatFoodAction, a class inherited from Action. Eating food recovers the humans health by 20 points and thus EatFoodAction has a method that increases the Humans hitPoints by 20. EatingFoodAction removes the Food instance using the removeItem method.

EatFoodBehaviour implements the Behaviour interface.

Why

Talk about new assoiciations, dependencies, pros and cons

**Rising from the Dead**

A dead Human can now be reincarnated as a Zombie after 5-10 turns. To achieve this, we’ve added some attributes to the Human class so we can track how many turns have passed after their death. These attributes are corpseAge which tracks its age since death (initialised as 0) and reincarnationAge which is set to a random number between 5 and 10.

When a Human is killed its ZombieCapability is set to UNDEAD. When this happens the only Behaviour they can implement on playTurn() is DeadBehaviour. DeadBehaviour implements the Behavour interface and its role is to retrieve the Humans corpseAge and reincarnationAge and dependant on these numbers use the getAction method to execute either RotCorpseAction or RiseFromDeadAction.

If corpseAge < reincarnationAge the Human is not ready to be reincarnated yet. RotCorpseAction is executed and a method is used to increment the Humans corpseAge.

If corpseAge = reincarnationAge the Human is ready to rise. RiseFromDeadAction is executed and removes the current Human and instantiates a new Zombie at the same location.

RotCorpseAction and RiseFromDeadAction are both inherited from the Action class.

Why