**Design Rationale**

**Group : Team 2099**

**Members : Siang Jo Yee, Lua Shi Liang**

Rising from the dead

**Corpse** [New Class]

* An **Item** added to the **GameMap** when an **Actor** (**Human**, **Farmer** and **Zombie**) is killed.

New Changes and their Responsibilities:

* This class inherits from an abstract class called **Item**.
* It has two private attributes (**int** turn and **Actor** initialActor).
  + turn - acts as a counter
  + initialActor - the dead **Actor** who owns the **Corpse**
* It has a constructor that takes in 2 parameters (**String** name and **Actor** initialActor). It initialises initialActor and also makes use of the superclass constructor to initialise name, displayChar and portable.
* It overrides tick() method to fulfil the required functionality for the **Corpse** to “rise from the dead”
  + Parameter: **Location** currentLocation - Current location where the corpse is placed
  + This method is executed every turn in the game.
  + turn increments by 1.
  + If initialActor does not hasCapability(*ZombieCapability.UNDEAD*), in which it is not a **Zombie**, generate a random integer between 5 to 10 inclusive.
  + Then, if the turn equals the random integer and there is no **Actor** on the currentLocation, A new **Zombie** object is instantiated.
  + The **Zombie** is added onto currentLocation and the **Corpse** is removed from currentLocation using methods in **Location** called addActor() and removeItem() respectively.
  + A string is printed to notify the **Player** that a dead **Human** becomes a **Zombie**.
  + This will look as though the **Corpse** becomes a **Zombie** 5 to 10 turns later.

Design Choices / Reasons:

* The design in which the Corpse inherits from Item takes into account the Do Not Repeat Yourself principle because Corpse also uses the features in Item and inheritance helps to avoid repetition of codes.
* **Zombie** objectis instantiated in the method as a local variable instead of as an attribute to follow Declare things in the tightest possible scope principle to lessen the risk that something can depend on it, thus reduce the risk of possible failure in the future.

**AttackAction** [Existing Class]

New Changes and their Responsibilities:

* execute() method is modified
  + If an **Actor** is killed after being attacked [existing code], a new **Corpse** object is instantiated [modified] instead of a **PortableItem** [existing code], then added to the **Location** of the dead **Actor** which is removed from the **Location** [existing code].
  + This is because there is a new **Corpse** class created in which it is able to increment the counter every turn in the game to control the turn when it becomes a **Zombie** to fulfil the required functionality.

Design Choices / Reasons:

* **Corpse** is instantiated in the method as a local variable instead of as an attribute because Declare things in the tightest possible scope principle is taken into account in this design.

Farmers and food

**Farmer** [New class]

* A new kind of **Human** who shares the same characteristics and abilities as a **Human** but also able to sow, fertilize and harvest crops.

New changes and their Responsibilities:

1. This class inherits from a class called **Human**.
2. It has four private attributes stored in an array of type **Behaviour** - behaviours

* Its elements are new object instantiated from **HarvestBehaviour**, **FertilizeBehaviour**, **SowBehaviour** and **WanderBehaviour** respectively

1. It has a constructor that takes in one parameter (**String** name). It makes use of the superclass constructor to initialise name, displayChar and hitPoints. It also calls addCapability() method from **Actor** with *ZombieCapability.FARM* as the argument.
2. It overrides playTurn() method.
   1. Parameter:

* Actions actions, Action lastAction, **GameMap** map, Display display
  1. This method executes every turn in the game.
  2. In the method, it first call returnEatAction(map) method which inherits from **Human** [details discussed in the **Human** class], if it is not null, return the return value of this method,

else,

An enhanced for loop is used to loop through the elements in the array of behaviours, and if the return value of getAction(this, map) method from **Behaviour** interface is not null, the action is returned,

else,

continue looping the array based on the sequence of elements in the array

If all getAction(this, map) is null, return a new DoNothingAction() object, in which the Farmer will do nothing in that turn.

* 1. Overall, this method will return the first **Action** which is not null following the sequence which is the return value of returnEatAction method, then the return value of getAction method called by **HarvestBehaviour**, **FertilizeBehaviour**, **SowBehaviour**, **WanderBehaviour** and lastly **DoNothingAction**.

Design Choices / Reasons:

* The design where **Farmer** inherits from **Human** follows the Do Not Repeat Yourself principle because **Farmer** also uses the features in **Human** and inheritance helps to avoid repetition of codes.
* behaviours are declared as private to follow the Minimize dependencies that cross encapsulation boundaries principle because if we declare it to be more visible, i.e. public initially, it may break other systems if we change a originally more visible attribute to a lower visibility.
* *ZombieCapability* is an Enum and defines a collection of constants to indicate the capability of an **Actor**. **Farmer** is designed to have the capability *ZombieCapability.FARM* to follow the Do Not Repeat Yourself, Reduce dependencies as much as possible and Avoid excessive use of literals principle. It is designed to avoid repetition of code when checking whether an **Actor** is a **Farmer**, and also to avoid using the getDisplayChar method from **Actor** whether it is equal to the displayChar of **Farmer**, since if displayChar is changed, we would have to hunt for every place it occurs in the code and change all of them. It would help to avoid having such indirect dependencies in the system.
* returnEatAction method in Human class is called in the playTurn method instead of instantiating a new **EatBehaviour** object which calls the getAction method to return its **Action.** This is to follow the principle:
  + Do Not Repeat Yourself as to avoid repetition of codes since **Human** and **Farmer** can both eat **Food** when they are damaged [details for method discussed in Human class]
  + Reduce Dependencies as much as possible since it reduces the dependency to EatBehaviour
* An enhanced for loop is used as stated above to avoid repetition of code on checking whether the **Action** returned by the elements in behaviours is null or not and this is also where Do Not Repeat Yourself principle is taken into account in this design.