Design Rationale

Krishanu Donahar and Jovin Mathew

Introduction

The design rationale is split up so that each part represents a feature. It will explain how the farute will be implemented and what classes and/or methods will be called, created or modified.

Bite

Zombie Bite

New class: ZombieBiteAction
Called by: AttackBehaviour
Modifications: getAction in AttackBehaviour will be modified to call ZombieBiteAction if the attacker is a zombie and a chance variable is in the valid range.
Role/Responsibility of the class: to execute the action of a zombie biting a person
Result: if zombie bite is successful it will heal the zombie by calling actor.heal() and damage the human by calling actor.hurt(). The result will then be returned as a string.

Picking up

PickUpBehaviour

Implements: Behaviour
Role: To allow Zombies to pick up items
Reasoning: Zombies are required to pick up weapons on the ground. So we thought to include a PickupBehaviour class.
Result: upon calling the pickup Behavior, if the zombie is standing on an item which it is able to pick up, it will call addItemToInventory() from superclass Actor and thus add it to the inventory of that zombie.

Zombie limb Weapons

Added Weapons

Added the following classes: armWeapon, legWeapon, clubWeapon, maceWeapon which all implement WeaponItem.
Role: each of these classes represent the respective weapon. Arm and Leg weapons are made when a zombie loses a limb (see below in Zombie limbs and effects) and club and mace weapons are crafted from arm and leg respectively.
Description: We decided to make 4 different weapons and when one is crafted to it's upgraded version, the former is deleted and an object of the new weapon takes its place.

Zombie limbs and its effects

Modify:

- execute method in AttackAction; check if the target is a zombie and then remove limbs according to chance and limbs remaining. Also if the attacker is a zombie, check the number of arms and change the chance of bite, accordingly.
- playTurn in Zombie class to check if the zombie has one leg and movedLastTurn is true. If true then return doNothingAction() and set movedLastTurn to False.
- If Zombie has no legs remove the wanderBehaviour from the behaviours list attribute.
- Add the leg/arm weapon item on the spot the zombie fell by calling map.getLocation

Add:

- The integer attributes arms, legs and totalLimbs to track the respective limbs.
- Boolean attribute: movedLastTurn which tracks whether or not the Zombie moved last turn (this is used for when the zombie loses a leg)

Reasoning: We decided that the zombie limb should fall off in the same spot because we believe if a zombie lost a limb because it got hit, it would most likely fall off on the spot rather than fly a meter awa or so.

Say Brains

ZombieTalkBehaviour and ZombieTalkAction

Add: sayBehaviour is a Class with a chance variable that returns a ZombieTalkAction if the chance variable is satisfied. ZombieTalkAction has a list attribute of zombie sayings and a random integer attribute to choose which saying is outputted.
Modify: Zombie constructor to include sayBehaviour as the first in the behaviours list.
Reasoning: we decided that if a zombie decides on saying something, it will cost it one turn since it is not a very intelligent entity.

Crafting

CraftAction

Implements: Action.
Role: to craft an arm into a club or a leg into a mace
How: The method will create the new weapon object (club or mace) and add it to the inventory spot by calling actor.addItemToInventory(club/mace). It removes the old arm/leg item from the inventory as well.
The action itself will be added to the list of allowable actions of the arm and leg weapon so that it will be an option for the player.

Back from the dead

Corpse

New class: CorpseItem
Extends: PortableItem
Called by:
Roles/Responsibility: Corpse of humans turns into zombies in 10 turns.
Result: After a human dies, they leave a Corpse on the location of their death by calling map.locationOf(target).addItem(corpse). The corpse has an age variable which increments by 1 each turn by calling the tick methods from the item class. When the age reaches 10, the corpse turns into a zombie by removing the corpse item and creating a new zombie with the same name and planting it on the same spot or a spot nearby (if the actor is holding the zombie). The zombie is added by calling destination.addActor(newZombie).

Farming

Farmer

New class: Farmer
Extends: Human
Modify: Map in the Application class to display a set number of farmers along with the other characters on map.
Called by:
Role/Responsibility of the class: Creates a new type of actor called farmers.

Crops

Modify:

- Tree - Changing the tree class into crops.

- Application - Removing all the trees from the game map.

Role/Responsibility of the class: Stores ripe and unripe crops, and it converts crops into food when the user or a farmer encounters a ripened crop.

RipeCrop

New class: RipeCrop
Extends: Item
Called by: HarvestAction
Roles/Responsibility: A ripe crop can be harvested and turned into food.

Food

New class: Food
Extends: Item
Called by: ConsumeAction
Roles/Responsibility: Stores the information for the portable food item which can be used to regenerate a player's HP.

Sowing Crops

New class: SowAction
Extends: Action
Called by: FarmBehaviour
Modify:
Roles/Responsibility: To give farmers the ability to sow crops.
Result: When a farmer moves, it goes through the SowAction class which has a 33% probability of being executed. If the class is executed, the Farmer checks for valid dirt blocks by calling the checkForBlock() method around their location and sows an unripened Crop item on it.

Fertilizing Crops

New class: FertilizeAction
Extends: Action
Called by: FarmBehaviour
Modify:
Roles/Responsibility: To give farmers the ability to fertilise unripened crops.
Result: The farmer checks the dirt block on their location by calling the checkForCrops() method in the FarmBehaviour class. If there is an unripe crop, the farmer calls the FertilizeAction class and reduces the time to ripen by 10 turns.

Harvesting Crops

New class: HarvestAction
Extends: Action
Called by: FarmBehaviour & Player
Roles/Responsibility: To give farmers and the player the ability to harvest crops into food.
Result: The farmer/player checks for ripened crops every turn by calling checkForRipe(). If the farmer/player does have a ripened crop around them, they could harvest the crop to turn into food by converting the ripened crop to dirt by using groundfactory() at the location and then to a food item by using additem in the previously restored location. If a player harvests a crop, the food (harvested crop) goes into the players inventory. If a farmer harvests a crop, the food is dropped by the farmer on the same location.

Consuming Food

New class: ConsumeAction
Extends: Action
Called by: ConsumeBehaviour
Roles/Responsibility: Consume food to restore the player's health.
Result: While moving, humans constantly run checkForFood() to check if there is food dropped on their current location. After encountering food, healthCheck() calculates if their health is low enough to consume food or not. If true, NPC humans (every human except the player) consume the food right away. Whereas, the player has an option to either consume the food or store it in his inventory to be consumed later.

Farmer Behaviour

New class: FarmBehaviour
Extends: Behaviour
Called by: Farmer
Roles/Responsibility: Lets the farmer to either sow or fertilize a crop.
Result: Generates a FertilizeAction if the farmer is on the block or else generates a SowAction which has a 33% probability of being executed.

Consume Behaviour

New class: ConsumeBehaviour
Extends: Behaviour
Called by: Farmer, Player & Human
Roles/Responsibility: The class checks for the actor's and if the actor is damaged it calls ConsumeAction.

More Weapons

Shotgun

New class: Shotgun
Extends: WeaponItem
Called by: Application to place shotguns on the map
Roles/Responsibilities: To track ammo through the private ammo variable and public methods to check, and decrement the ammo

Sniper

Same as Shotgun but with name of Sniper

UseGunAction

New class: UseGunAction
Extends: Action
Called by: Player if there is a shotgun/sniper in inventory then add UseGunAction with mode parameter for the weapon in the inventory
Roles/Responsibilities: To create a submenu for sniper/Shotgun to select where/who to aim at. Then according to the selections, do the respective tasks and return the result as a string.
Result: If sniper then gives options of which direction to shoot in. Once selected calls Blast method which checks each location, which is shot, for an actor then uses the 75% chance to see if they are hit. If so then reduce their health and if they are unconscious, remove them and place a corpse.

ShotgunAmmo

New class: ShotgunAmmo
Extends: Portable Item
Called by: Application to place ammo boxes on the map
Roles/Responsibilities: To allow players to increase their shotgun's ammo by allowing the player to call the PickUpAmmoAction if they are standing on this item with a shotgun. Once picked up the ammo is increased by 3 and the item is removed from the inventory.

Sniper Ammo

Same as ShotgunAmmo except for sniper.

PickUpAmmoAction

New class: PickUpAmmoAction
Extends: Action
Called by: Player in playTurn, when it checks the inventory to see if there is the respective weapon for the ammo the player is standing on. If not then remove this action from the actions for the player.
Roles/Responsibilities: to allow the player to increase their gun's ammo by picking up an ammo item.

ListActors and ListItems

ListActors

New class: ListActors
Called by: Player, UseGunAction
Roles/Responsibilities: Returns a list of all the actors in the map.
Result: Runs a loop starting from the first block of the map and checks if the block has an actor on it using location.getActor(). If true, the actor is then added to an arraylist called actors, which stores all the actors present.

ListItems

New class:
Called by: Player
Roles/Responsibilities: Returns a list of all the items in the map.
Result: Runs a loop starting from the first block of the map and checks if the block has an item on it using location.item(). If true, the list of items are then added one by one to an arraylist called items, which stores all the items present.

Going to Town

Travelling

New class: TravelAction
Extends: Action
Called by: Player, Vehicle
Roles/Responsibility: Moves the player from one map to another.
Result: When the player decides to travel, the actor is moved to the other map. If the player is in the main(compound) map, they travel to the town map and vice versa.

Vehicle

New class: Vehicle
Extends: Item

Called by: Player, Application
Roles/Responsibility: Enables the player to travel/ to use TravelAction().
Result: There is a vehicle in each map and it stores an instance of the other map. Whenever a player encounters a vehicle, he has the ability to travel to the other map.

Mambo

Player Modifications

Modify: The player class has been modified to run a 5% chance every turn for Mambo Marie to spawn. The MamboMarieSpawn() method in player does this by using a random integer (rand.nextInt).

Mambo Marie

New class: Mambo Marie
Extends: ZombieActor
Roles/Responsibilities: A Voodoo priestess and the source of the local zombie epidemic. She is able to spawn 5 zombies in every 10 turns and she vanishes after 30 turns.
Result: Every turn there is a 5% chance for Mambo Marie to spawn. When spawned, Mambo Marie has a default WanderBehaviour(). The variable *turncount* keeps a track of the number turns since her arrival. But in every 10 turns the ZombieSpawnAction is called which spawns 5 zombies in random locations. On the 30th turn Mambo Marie vanishes by calling the MamboVanishAction.

Arrival

New class: MamboSpeakAction
Extends: Action
Called by: Mambo Marie
Result: When Mambo Marie is spawned, this action is called which displays a string, letting the player know that Mambo Marie has spawned.

Spawning Zombies

New class: ZombieSpawnAction
Extends: Action
Called by: Mambo Marie
Roles/Responsibilities: Adds zombies to random map locations.
Result: When called by Mambo Marie, this class executes a loop for creating 5 new zombies and adding them to random map locations.

Vanishing

New class: MamboVanishAction

Extends: Action
Called by: Mambo Marie
Result: When executed, Mambo Marie is removed from the map.

Ending the game

Quit Game

NewClass: QuitGameAction
Extends: Action
Called by: Player
Modify: Modified player class to add QuitGameAction to the menu.
Roles/Responsibilities: Gives the player an option to quite the game at any time.
Result: When executed, the player is removed from the map which makes the engine stop the game and display “Game Over”.

Possible Endings

Winning the game

Modify: Modified player to include a method called WinCheck
Result: When executed, the method calls listActors which stores all the actors in the map in a list. If the list does not have any zombie display characters ‘Z’, it goes to check if mambo marie is killed or not. The method calls listItems which stores all the items in the map and checks for the mambo marie corpse’s display character ‘$’. If it exists, the QuitGameAction is called and the player wins the game.

Losing the game

Modify: Modified player to include a method called LossCheck
Result: When executed, the method calls listActors which stores all the actors in the map in a list. If the list does not have any human display characters ‘H’ and ‘F’(farmer), the QuitGameAction is called and the player loses the game.’