Going to Town

1 new class was implemented and 1 existing classes were modified.

Vehicle class added

Vehicle class extends from Item. The purpose is this class is to move the actor from one location to the other.

Instance Variable

1. String name,
    - Name of the vehicle(It is not unique)
2. char displayChar
    - Char displayed on the map
3. Boolean portable ->False

Constructor

1. String name,
    - Name of the vehicle(It is not unique)
2. char displayChar
    - Char displayed on the map

Methods

1. public void addAction(Action action)
   It adds an action to this.allowableActions

2. public void addMoveAction(GameMap map,int x,int y,Actor actor,String direction)
   It checks whether the actor appears on the map or not. If actor is not in map, it moves actor to map based on the map, the x and y coordinate and prints the direction(String) of the player's movement.

Application class modified

A gameMap called ghostTown is added.
Two vehicle called car1 and car2 is added.

- Car1 is in charge of transporting player from map to the ghostTown. Car2 is in charge of transporting player from ghostTown to map.

New weapons: shotgun and sniper rifle

1 new class was implemented and 2 existing classes were changed.

Ammunition
Classes added

1. Ammunition
    - Currently has no methods

2. AmmunitionRifle
    - Constructor:
        Public AmmunitionRifle()
        The name is set to Rifle Ammunition
        The displaychar is set to ';'
    - @Override methods:
        1. Public PickUpItemAction getPickUpAction()
            - Return a new PickUpAmmunitionRifleAction or null
        2. Public DropItemAction getDropAction()
            - Return a new DropAmmunitionRifleAction or null

3. AmmunitionShotgun
    - Constructor:
        Public AmmunitionShotgun()
        The name is set to Shotgun Ammunition
        The displaychar is set to ':'
    - @Override methods:
        1. Public PickUpItemAction getPickUpAction()
            - Return a new PickUpAmmunitionShotgunAction or null
        2. Public DropItemAction getDropAction()
            - Return a new DropAmmunitionShotgunAction or null

4. DropAmmunitionRifleAction
    - Constructor
        Public DropAmmunitionRifleAction(AmmunitionRifle item)
        Inherits dropItemAction's constructor
    - @Override methods
        1. Public String execute(Actor actor,GameMap map)
        This method calls noAmmunitionRifle() in human which converts hasAmmunitionRifle to false.

5. DropAmmunitionShotgunAction
    - Constructor
        Public DropAmmunitionShotgunAction(AmmunitionShotgun item)
        Inherits dropItemAction's constructor
    - @Override methods

1. Public String execute(Actor actor,GameMap map)
   This method calls noAmmunitionShotgun() in human which converts hasAmmunitionShotgun to false.

6. PickUpAmmunitionRifleAction
   - Constructor
     Public PickUpAmmunitionRifleAction(AmmunitionRifle item)
     Inherits pickUpItemAction's constructor
   - @Override methods
     1. Public String execute(Actor actor,GameMap map)
        This method calls gotAmmunitionRifle() in human which converts hasAmmunitionRifle to true.

7. PickUpAmmunitionShotgunAction
   - Constructor
     Public PickUpAmmunitionShotgunAction(AmmunitionShotgun item)
     Inherits pickUpItemAction's constructor
   - @Override methods
     1. Public String execute(Actor actor,GameMap map)
        This method calls gotAmmunitionShotgun() in human which converts hasAmmunitionShotgun to true.

Class modified
Human

1. New instance variable added(Both by default set to false)
   - Private boolean hasAmmunitionRifle
   - Private boolean hasAmmunitionShotgun

This two instance variable represents whether there is ammunition for rifle/shotgun in inventory. This instance variable helps to reduce complexity by not looping through the whole inventory to check if there is ammunition or not.

2. New methods added

   i) public void gotAmmunitionRifle()
   - Sets hasAmmunitionRifle to true

   ii) public void gotAmmunitionShotgun()
   - Sets hasAmmunitionShotgun to true

   iii) public void noAmmunitionRifle()
   - Sets hasAmmunitionRifle to false

iv) public void noAmmunitionShotgun()
- Sets hasAmmunitionRifle to false

v) private boolean containAmmunitionShotgun()
- Returns hasAmmunitionShotgun

vi) private boolean containAmmunitionRifle()
- Returns hasAmmunitionRifle

Addition of Shotgun and Sniper Rifle
Classes added

1. SniperRifle
    - Constructor
      public SniperRifle(String name, char displayChar, int damage, String verb)
      Name is set to “SniperRifle”, displayChar is set to ‘R’, damage is set to 20 and verb is set to "wacks with SniperRifle")

2. Shotgun
    - Constructor
      public Shotgun(String name, char displayChar, int damage, String verb)
      Name is set to “Shotgun”, displayChar is set to ‘S’, damage is set to 20 and verb is set to "wacks with Shotgun")

Both shotgun and sniper rifle extends from weaponItem.

Classes Modified

1. **Application**
   -> Add shotgun and sniper rifle to the map
2. **Human**
    - Public weapon getWeapon()
      Calls getHighestDamageWeapon(). If getHighestDamageWeapon returns null(meaning there’s no weapon in inventory), getIntrinsicWeapon() is returned

    - private weapon getHighestDamageWeapon()
      Returns the weapon with the highest damage. If there is no weapon in inventory, return null

Shotgun special effects
SniperRifle special effects

Mambo Marie

2 new classes were implemented and 1 existing class was changed.

VoodooPriestess

Inherits from ZombieActor because the priestess is not on Human's team (should not be assigned ZombieCapability.ALIVE) but is also not a Zombie.

ATTRIBUTES

1. **private int chantCounter**
   Counter to keep track of the number of times the VoodooPriestess has chanted.
2. **private int turnsOnMap**
   A counter for how many turns has it been since Mambo Marie has appeared on the map.
3. **private Random rand**
   Random generator for when Mambo Marie appears and to generate random location.
4. **private Behaviour behaviour**
   Only WanderBehaviour right now.

METHODS

1. **VoodooPriestess(String name)**
   Constructor for a voodoo priestess. Calls super(name, '&', 200, ZombieCapability.UNDEAD). It has a parameter for the name (instead of fixing the name as 'Mambo Marie') in case more Voodoo priestesses need to be instantiated in the future. The maxHealthPoints is 200 because she should be hard to kill and the ZombieCapability is UNDEAD because she's on the same team as the Zombies.
2. **playTurn(Actions actions, Action lastAction, GameMap map, Display display)**
   Increments turnsOnMap by one. If the voodoo priestess is not on the map, then she will appear at a random location if rand is less than 0.05. Else, if she is on the map, and if turnsOnMap is divisible by 10 (every 10 turns), a ChantAction is created, otherwise she'll wander and if turnsOnMap is already 30, she will be removed from map.

ChantAction

Inherits from Action.

ATTRIBUTES

1. **private int chantCounter**
   The nth time the actor has invoked ChantAction.
2. **private Random rand**
   Random generator to generate random locations.

METHODS

1. **ChantAction(chantCounter)**
   Constructor for ChantAction the nth time the actor has chanted is passed as parameter.
2. **@Override**

execute(Actor actor, GameMap map)
Creates five new Zombie objects at random locations in the map. They would have names “Zombie Minion” + which chant it came from and what number zombie it was in that particular chant (e.g. Zombie Minion 3.2 indicates that this zombie was the second zombie to rise from the dead from the actor’s third chant). This naming convention allows the zombies to have unique names, the player to know how many times Mambo Marie has chanted, and allows zombies that have risen from the chant to be named more dynamically than picking out names from a fixed collection of names. Returns a string saying actor chants and 5 new zombies have risen from the dead.

3. **@Override**
   menuDescription(Actor actor)
   Returns a string saying actor chants.

Application
Instantiate a VoodooPriestess object with the name Mambo Marie.

Ending the game

1 new class was implemented and 1 existing class was changed.

Application

Change world to be a ZombieWorld object instead of World from the engine package.

ZombieWorld

Inherits from World.

ENUM

GameStatus

Tells if the game has been won, lost or the player has quit, or the game is running.

ATTRIBUTES

1. **private GameStatus currentStatus**
   To keep track of the currentStatus of the game.
2. **private VoodooPriestess mamboMarie**
   Mambo Marie is initialized in the world itself so that she can easily be removed and put on map while still processing her turn.

METHODS

1. **ZombieWorld(Display display)**
   Constructor. It passes display to super constructor.
2. **@Override**
   run()
3. **@Override**
   stillRunning()
4. **@Override**
   endGameMessage()

BONUS FEATURE

1 new class was implemented

ASSIGNMENT 2 MODIFICATIONS

Zombie Attack

1. Create a new behavior class call ZombieAttackBehavior
    - Returns a new ZombieAttackAction instead of an attackaction
2. Create a new attackaction called ZombieAttackAction class
    - Since zombieAttackAction is only for zombie, the if-else statement that was used to check the class for actor and target is removed from the original AttackAction's execute method. Therefore ZombieAttackAction only check if the weapon is a bite or punched and whether the target is dead or not.
3. Modification of zombie class
    - zombie class's behavior list replace AttackBehaviour with ZombieAttackBehaviour

4. Create a new attackaction called HumanAttackAction.
    - Since this guarantees that the target is a zombie, it will check whether the zombie loses any legs or hands and will add the limbs to the map.
5. Create a new behavior class call HumanAttackBehaviour
    - Returns a new HumanAttackAction
6. Modify zombie class by overriding the getAllowableAction by replacing HumanAttackAction instead of AttackAction.
    - This is because getAllowableAction is all the action that is allowed to act on the current actor(zombie) by other actor(human).

7. The old version of attackAction replaces the current AttackAction so that if a new class that has no special features was added, they will be using attackAction, with the exception of the addition of human corpse(from assignment 2 corpse) in the method isDead().
8. A new method called isDead is added to check whether the target is Dead or not.
    - If the target is human and the target is dead, it will turn into a human corpse and will turn into a zombie.

All this modifications are made to make the code more readable and extendable. The system now do not need to check if the actor and target is a zombie or not. Since the part where the target,actor and weapon that will be the one changing depending on with attackaction it has, and the part where the target is checked whether it is conscious or not, isDead reduces the repeated code in all AttackAction classes. It makes the code more extendable and readable.