Report

CS 32 project 3

Xiaoshu Luo

904971914

1. Description of public member functions:

**Class Actor:**

Actor(int ID, double level\_x, double level\_y, int direction, int depth, StudentWorld\* p);

This is the constructor. It is used to construct an actor in a given position.

virtual void doSomething() = 0;

Each actor must be given a chance to do something during one tick. Since each kind of actor has its own behavior, this function is made virtual. Since actor is an abstract concept, this function is made pure virtual.

virtual bool blocksMovement() const {};

This function is used to check whether an actor blocks the way for other moving actors. It is made virtual because most of the actors do not block movements while a few do. Therefore it is made false by default and made true for some particular cases.

virtual bool canBeInfected()const {}

This function is used to check whether the actor is a human. It is made virtual because only humans can be infected while other actors can’t. Therefore it is made false by default and made true for some particular cases.

virtual bool canBeDamaged() const {}

This function is used to check whether the actor can be destroyed or killed. It is made virtual because only humans, zombies and goodies can be destroyed or killed. Therefore it is made false by default and made true for some particular cases.

virtual bool triggersZombieVomit() const {}

This function is used to check the actor is a human. It is virtual because only human trigger vomits. Therefore it is made false by default and made true for some particular cases.

virtual bool blocksFlame() const {}

This function is used to check whether the actor blocks flames. It is virtual because only walls and exits blocks flame. Therefore it is made false by default and made true for some particular cases.

virtual void beVomitedOnIfAppropriate() {}

This function is used to check whether a human gets infected by a vomit. It is virtual because only humans can be infected. Therefore it is made false by default and made true for some particular cases.

virtual bool beSmartZombie() {}

This function is used to check whether the actor is a smart zombie. It is virtual so that smart zombies can be distinguished.

virtual bool beDumbZombie() {}

This function is used to check whether the actor is a dumb zombie. It is virtual so that dumb zombies can be distinguished.

virtual bool canLeadCitizens() {}

This function is used to check whether the actor is a Penelope. It is virtual so that Penelope can be distinguished.

virtual bool triggersOnlyActiveLandmines() {}

This function is used to check if the actors trigger active landmines. It is virtual as only humans and zombies can trigger landmines.

virtual bool beLandmine() {}

This function is used to check whether the actor is a landmine. It is virtual so that landmines can be distinguished.

virtual bool moveTowards(int ch);

This function is used for a certain actor to move in a certain direction by 4 pixel. it returns true if the movement is actually made. It is virtual as it is made specifically for Penelope.

virtual bool isAlive() const;

This function is used to check if the actor is still alive. if not, it will be deleted at the end of this tick. It doesn’t need to be virtual.

virtual void setDead();

This function is used to set the actor’s to be dead. It is virtual as landmine actors have some additional actions after they are dead.

virtual StudentWorld\* getWorld()const;

This function is used to get the pointer to the Studentworld the actor is related to. It is the same for all objects, and need not be virtual.

**Class Agent:**

Agent(int ID, double level\_x, double level\_y, int direction, int depth, StudentWorld\* p);

This is the constructor.

virtual bool canBeDamaged() const {}

This function returns true as all agents can be killed.

virtual bool blocksMovement() const {};

This function returns true as all agents block movements.

virtual bool paralyzed() {}

This function checks if a citizen is paralyzed during a certain tick. it is virtual as only Penelope will never be paralyed.

virtual bool triggersOnlyActiveLandmines() {}

This functions returns true as all agents can trigger active landmines.

virtual void setRandomDirection();

This function sets a random direction for the actor. It is the same for all actors, so need not be virtual.

virtual bool tryMoveBy(int d);

This function lets the actor move in its current direction by d pixels. It is same for all actors and need not be virtual.

**Class Human:**

Human(int ID, double level\_x, double level\_y, int direction, int depth, StudentWorld\* p);

This is the constructor.

virtual bool isInfected() const {}

This function returns true if the human is infected. it doesn’t need to be virtual.

virtual bool triggersZombieVomit() const {}

This function returns true as humans can trigger vomits.

virtual bool canBeVomited()const {}

This function returns true as humans can trigger vomits.

virtual bool canBeInfected()const {}

This function returns as humans can be infected.

virtual void beVomitedOnIfAppropriate() {}

This function sets the human actor to be infected.

virtual void heal() {}

This function sets the human to be uninfected and its infection count to be zero. It needs not be virtual.

virtual bool paralyzed() {}

This function sets the human to be paralyzed.

virtual void doSomethingSpecific() = 0;

This function lets the human do something specific (for citizens or Penelope). It is virtual as citizens and Penelope have different actions.

virtual void doSomething();

This function lets any human actor do something in common.

int infectionCount() {}

this function returns the infection count of human.

void incrementInfection() {}

this function increments the infection count by one.

**Class Citizen:**

Citizen(double level\_x, double level\_y, StudentWorld\* p);

This is the constructor.

virtual bool paralyzed();

This functions checks if the citizen is paralyed during each even-numbered tick.

virtual void doSomethingSpecific();

This function lets a citizen behave as mentioned in the spec.

virtual bool citizenTryMove(Direction d);

This function enables a citizen to try to move in a direction by 2 pixels. It needs not be virtual.

**Class Penelope:**

Penelope(double level\_x, double level\_y, StudentWorld\* p);

This is the constructor.

virtual void doSomethingSpecific();

This function enables Penelope to receive commands from the player and behave accordingly.

void adjustLandmine(int n);

This function changes the amount of landmines Penelope has by n.

void adjustVaccine(int n);

This function changes the amount of vaccines Penelope has by n.

void adjustFlame(int n);

This function changes the amount of flames Penelope has by n.

virtual bool canLeadCitizens() {}

Penelope can lead citizens, so it returns true.

int getFlame() {}

This function gives the number of flames Penelope has.

int getVaccine(){}

This function gives the number of vaccines Penelope has.

int getLandmine() {}

This function gives the number of landmines Penelope has.

bool useVaccine();

This function lets Penelope use one vaccine. The number of vaccines decreases by 1.

bool useFlame();

This function lets Penelope use one flame. The number of flames decreases by 1.

bool useLandmine();

This function lets Penelope use one landmine. The number of landmines decreases by 1.

**Class Zombie:**

Zombie(double level\_x, double level\_y, StudentWorld\* p);

This is the constructor.

virtual int getPlan()const;

This function returns the planned distance. It needs not be virtual.

void getNewPlan();

This function gives a zombie a new planned distance.

virtual bool paralyzed();

This function returns if the zombie is paralyzed during even-numbered ticks.

virtual void doSomething();

This function lets zombies do something in common as mentioned in the spec.

virtual void doSomethingSpecific() = 0;

This function lets zombies do something specific (for smart ones and dumb ones). It is virtual as smart zombies and dumb zombies are different. It is pure virtual as there are no actors that are only zombies.

virtual bool checkfront();

This function lets zombies check if they can create a vomit in their front and do so if possible.

**Class Smart zombie:**

Smart\_Zombie(double level\_x, double level\_y, StudentWorld\* p);

This is the constructor.

virtual void doSomethingSpecific();

This function lets one smart zombie to check if it can follow the closest person and do so if possible. If not, it will choose a random direction.

virtual bool beSmartZombie() {}

This function confirms the actor is a smart zombie.

**Class Dumb zombie:**

Dumb\_Zombie(double level\_x, double level\_y, StudentWorld\* p);

This is the constructor.

virtual void doSomethingSpecific();

This function lets one smart zombie to choose a random direction.

virtual bool beDumbZombie() {}

This function confirms the actor is a smart zombie.

**Class Wall**

Wall(double level\_x, double level\_y, StudentWorld\* p);

This is the constructor.

void doSomething();

This function returns immediately as walls do nothing.

virtual bool blocksMovement() const {};

This function returns true as walls do block movement.

virtual bool blocksFlame() const {}

This function returns true as walls do block flames.

**Class Activating object:**

ActivatingObject(int ID, double level\_x, double level\_y, int direction, int depth, StudentWorld\* p);

This is the constructor.

**Class Exit:**

Exit(double level\_x, double level\_y, StudentWorld\* p);

This is the constructor.

virtual bool blocksFlame() const {}

It returns true as exits do block flames.

virtual void doSomething();

This function checks if any human overlaps with the exits.

**Class Pit:**

Pit(double level\_x, double level\_y, StudentWorld\* p);

This is the constructor.

virtual void doSomething();

This function checks if any agents overlaps with the pits.

**Class Flame:**

Flame(double level\_x, double level\_y, int dir, StudentWorld\* p);

This is the constructor.

virtual void doSomething();

This function checks if any agents overlaps with the flame and sets them dead if so.

**Class Vomit:**

Vomit(double level\_x, double level\_y, int dir, StudentWorld\* p);

This is the constructor.

virtual void doSomething();

This function checks if any human overlaps with the vomit and sets them infected if so.

**Class LandMine:**

Landmine(double level\_x, double level\_y, StudentWorld\* p);

This is the constructor.

virtual bool canBeDamaged() const {}

It returns true as landmine can be destroyed.

virtual bool beLandmine() {}

It returns true to identify landmines.

int getTicks() {}

It returns the ticks of the landmine.

void decrementTicks() {}

It decreases the ticks of the landmine by 1.

bool isActive() {}

It returns true if the ticks of the landmine reaches 0.

void setActive() {}

It sets the state of the landmine to be active.

virtual void setDead();

It sets the landmine dead, creates flames around it, and leaves a pit.

virtual void doSomething()

It checks if any agents trigger the landmine in this tick.

**Class Goodie:**

Goodie(int ID, double level\_x, double level\_y, StudentWorld\* p);

This is the constructor.

virtual void doSomething();

It checks if the player picks up the goodie.

void bePickedUp();

It checks if the player overlaps with the goodie.

virtual void bePickedSpecific() = 0;

It gives players specific response.

virtual bool canBeDamaged() const {}

It returns true as all goodies can be destroyed.

**Class Vaccine\_Goodie:**

Vaccine\_Goodie(double level\_x, double level\_y, StudentWorld\* p);

This is the constructor.

virtual void bePickedSpecific();

It gives the player 1 vaccine.

**Class Flame\_Goodie:**

Flame\_Goodie(double level\_x, double level\_y, StudentWorld\* p);

This is the constructor.

virtual void bePickedSpecific();

It gives the player 5 flames.

**Class Landmine\_Goodie:**

Landmine\_Goodie(double level\_x, double level\_y, StudentWorld\* p);

This is the constructor.

virtual void bePickedSpecific();

It gives the player 2 landmines.

1. Functionality I fail to finish:

According to my testing results, I have realized all the functionalities mentioned in the spec.

However, I created some useless functions, and wrote some repetitive codes for some particular functions that could have been integrated. Also, I set most of my functions as virtual as I am not confident that each of them will only need to be implemented once.

1. Other assumptions:

There are no non-trivial assumptions in my code.

1. Testing classes:

**For citizens:**

Check if they do nothing when no zombies or Penelope is around.

Check if they try to run away when some zombies approach them.

Check if they try to follow Penelope when she is close enough.

Check if they do nothing when some zombie approaches but they get stuck in some corner.

Check if they can trigger a vomit.

Check if they can attract zombies.

Check if they can become a zombie after 500 ticks.

Check if they can use the exit.

Check if they can be killed by flames or pits.

Check if they can be blocked by walls.

Check if they

**For Penelope:**

Check if she moves as the player commands.

Check if she is blocked by other agents and walls.

Check if she can trigger a vomit.

Check if she can attract zombies.

Check if she can use goodies as instructed.

Check if she can be killed by flames or pits.

Check if she can be infected and become a zombie.

Check if she can use the exits before and after all the citizens are gone.

**For dumb zombies:**

Check if they can randomly move around.

Check if they can be blocked by other agents or walls.

Check if they create a vomit when some human is in front of them.

Check if they can be killed by flames or pits.

**For smart zombies**:

Check if they can randomly move around when no human is nearby.

Check if they can follow the closest person if distance is short enough.

Check if they can create a vomit when some human is in front of them.

Check if they can be killed by flames or pits.

**For walls:**

Check if they can block flames.

Check if they can block movements.

**For pits.**

Check if they can kill any agents overlapping with them.

**For exits:**

Check if citizens and Penelope can leave the scene when overlapping with them.

Check if they block flames.

**For flames:**

Check if they kill any agents overlapping with them.

Check if they can destroy goodies.

Check if they can detonate a landmine.

Check if they can be blocked by exits or walls.

**For vomits:**

Check if they can infect any human in contact.

**For landmines:**

Check if they can be triggered by agents before and after 30 ticks.

Check if they create 9 flames around it.

Check if they leave a pit in the original position.

Check if it can be triggered by flames.

**For the three goodies:**

Check if they can be picked up by Penelope.

Check if they can be destroyed by flames.