Ma yunhui

505-179-815

CS 32

Project 3 report

Class StudentWorld:

virtual int init();

initialize all actors including citizen, wall, Penelope. etc

return GWSTATUS\_CONTINUE\_GAME;

virtual int move();

for loop for all actor.

If they are alive, dosomething().

If they are dead, delete and erase them.

If no more citizen, Penelope goes to next level. return GWSTATUS\_FINISHED\_LEVEL

If Penelope is dead, return GWSTATUS\_PLAYER\_DIED

Return GWSTATUS\_CONTINUE\_GAME.

virtual void cleanUp();

delete Penelope;

For loop for each actors

Delete all actors;

bool moveToNextLevel() { return go\_next\_level; }

if go\_next\_level == true

return true; //Penelope can go to next level

bool check\_collisionForPlayer(double next\_x, double next\_y);

Loop through every actors which are blocking

Return true if any blocking actors are overlap with next\_x next\_y location

//passing the next location to this function so that the player would not go into the blocking actors

bool check\_collision(double next\_x, double next\_y, int p, int dir);

This function is used for the citizen and zombies when they are moving

Check each direction if any blocking objects at the front

Return true if it has blocking object at the front

Make sure the citizen or zombies not to block itself when running the for loop

bool check\_collision\_helper(double x1, double y1, double x2, double y2, int p);

A helper function for check\_collision to check the blocking objects

bool check\_personInFront(double next\_x, double next\_y);

This runs through all Actors, and if it’s human (citizen or penelope) overlaps with next\_x and next\_y. return true. Otherwise return fasle;

bool block\_Flame(double x, double y);

Runs through all Actors, if it’s an actor which can Block Flame and overlap with x,y return true; otherwise return fasle;

void Player\_overlapWithExit(double exit\_x, double exit\_y);

check if the player is overlap with Exit. If has no citizen at current level, set go\_next\_level to be true.

void citizen\_overlapWithExit(double exit\_x, double exit\_y);

check if the human(citizen) overlap with Exit, delete the citizen and increase 500 score;

bool Player\_overlapWith\_Goodies(double x, double y);

check if the player is overlap with goodies. Return true; otherwise return false;

void overlapWithPit(double pit\_x, double pit\_y);

Runs through all Actors, if it’s an actor which is person (citizen or zombie or penelope) overlaps with pit\_x, pit\_y set this actor dead.

If it’s citizen overlap with flame, call virtual function score\_doSomething() for citizen

If it’s Zombie overlap with flame, call virtual function score\_doSomething() for Zombie. Dumb zombie adds 1000, Smart zombie add 2000.

void overlapWithFlame(double flame\_x, double flame\_y);

Runs through all Actors, if it’s an actor which is can be damage by flame overlaps with flame \_x, flame \_y set this actor dead.

If it’s citizen overlap with flame, call virtual function score\_doSomething() for citizen

If it’s Zombie overlap with flame, call virtual function score\_doSomething() for Zombie. Dumb zombie adds 1000, Smart zombie add 2000.

void overlapWithVomit(double vomit\_x, double vomit\_y);

if Penelope is overlap with vomit\_x and vomit\_y. set Penelope infected.

Runs through all actors. If it has citizen overlap with vomit\_x and vomit\_y.

Set this citizen infected;

void fire(double x, double y, int dir);

pass Penelope location and direction (x, y, dir). Run a for loop to create 3 flames at the front of Penelope.

void compute\_vomit(double x, double y, int dir);

pass zombie location and direction (x, y, dir). Create a vomit at the front of zombie.

void searchClosestPeople(double x, double y, double& cloest\_x, double& cloest\_y, double& distance, bool &isThreat);

this function searches the closest human (Penelope and citizen), and set reference closset\_x and closest\_y as well as the Euclidean distance

void citizenDistanceToPlayer(double citizen\_x, double citizen\_y, double& player\_x, double& player\_y, double& distance);

find the Euclidean distance from citizen to player, and set the reference value player\_x and player\_y;

void searchNearestZombie(double citizen\_x, double citizen\_y, double& zombie\_x, double& zombie\_y, double& distance);

this function searches the closest Zombies (either dump or smart) with the Euclidean distance, zombie\_x and zombie\_y would be the nearest Zombie location

void createAZombie(double x, double y);

create a Zombie at x and y; 3/10 would be smart zombie, 7/10 would be dump zombie

void createAVaccineGoodie(double x, double y);

1/10 chance to create a Vaccine goodie at x, y lovation

void placeLandmine(double x, double y);

create a landmine at x, y

void landmineBoom(double landmine\_x, double landmine\_y);

landmine explore, create 8 flame around itself.

bool overlapwithLandmine(double x, double y);

check if penelope, citizen or zombie overlap with the landmine, and return true;

otherwise return false;

void setGame\_info();

set the game title;

Penelope\* getPenelope() { return m\_penelope; }

Return Penelope as a point to other so that it can call the virtual pickup function in the Goodie in Actor

int getNumCitizen() { return num\_citizen; }

return the number of getNumCitizenp;

Actor class

virtual void doSomething() = 0;

I chose to define a pure virtual version

of the doSomething() function in my base Actor class because all actors in Zombie

Dash are able to doSomething() and each type of actor doSomething() in a different way

bool getStatus() { return object\_alive; }

return the status; either alive for true or dead for false

this function is same for all actor, no need to be virtual

void setDead() { object\_alive = false; }

set dead

this function is same for all actor, no need to be virtual

void setAlive() { object\_alive = true; }

set Alive

this function is same for all actor, no need to be virtual

StudentWorld \* getWorld() { return s\_world; }

returns the studentWorld pointer so that the actors can call it for an action.

virtual bool isHuman() { return false; }

return false for all actors. Only the Penelope and citizen return true;

virtual bool isZombie() { return false; }

return false for all actors. Only the DumbZombie and SmartZombie return true;

virtual bool isGoodie() { return false; }

return false for all actors. Only the all Goodies return true

virtual bool isBlockActor() = 0;

I chose to define a pure virtual version

of the isBlockActor () function in my base Actor class because all actors in Zombie

Dash are able to isBlockActor() and each type of actor isBlockActor() in a different way

virtual bool canBeDamagedByFlame() = 0;

I chose to define a pure virtual version

of the canBeDamagedByFlame () function in my base Actor class because all actors in ZombieDash are able to canBeDamagedByFlame () and each type of actor canBeDamagedByFlame () in a different way

virtual bool canBeDamagedByVomit() { return false; }

return false for all actors. Only the citizen and Penelope return true

virtual bool canBlockFlame() { return false; }

return false for all actors. Only the Wall and Exit return true

virtual bool isAFlame() { return false; }

return false for all actors. Only the Flame return true

virtual bool person() { return false; }

return false for all actors. Only the Human (citizen, Penelope and Zombies) return true

virtual void score\_doSomething() { return; }

This function is only used for the citizen and zombies to add or reduce socre.

virtual void chanceToCreateVaccine() { return; }

call the StudentWorld createAVaccineGoodie() function

bool getInfection\_status() { return infection\_status; }

return true if the human got infect, otherwise return false;

void addInfection() { infection++; }

increment infection starts at 0;

void setInfection() { infection\_status = true; }

set infect status to be true

void cureInfection() { infection\_status = false; infection = 0; }

set infect status to be false, set infection to be 0;

int getInfectedNumber() { return infection; }

return infection number such as 0-500;

class Agent : public Actor

virtual bool isBlockActor() { return true; }

return true for all actors inheritance from Agent

virtual bool canBeDamagedByFlame() { return true; } //same

virtual bool person() { return true; } //same

virtual void follow\_player(double x, double y, double player\_x, double player\_y, int pixel);

This function allows the Zombies or citizen follow or get closer to the Penelope. X, y is the Zombies or citizen location (depends on what call this function, can be either citizen or zombies). Player\_x and player\_y is the location for Penelope. pixel is how many pixel for the Zombies (1 pixel) or citizen (2 pixel) move. The citizen or Zombies try to move or follow to the player\_x, player\_y when calling this function.

virtual void runAwayFromZombie(double x, double y, double zombie\_x, double zombie\_y, int pixel);

This function allows the citizen run away from the nearest Zombies. X, y is the citizen location. Zombie\_x and Zombie\_y is the location for Zombies. pixel is how many pixel for the citizen (2 pixel) move. The citizen try run away from the Zombie \_x, Zombie \_y when calling this function. Zombie\_x, Zombie\_y is the closest location to the citizen (x, y)

virtual void goLeft(double x, double y, int p);

move the actor p pixel to the left.

virtual void goRight(double x, double y, int p);

move the actor p pixel to the right.

virtual void goUp(double x, double y, int p);

move the actor p pixel to the up.

virtual void goDown(double x, double y, int p);

move the actor p pixel to the down.

class Human : public Agent

void doSomethingCom();

check to see if it is currently alive. If not, return immediately

check if the citizen or Penelope get infection. If it’s true, call addInfection() function to increase the infection. Set this citizen dead and create a Zombie at that location when the infection reaches 500

Don’t need to be virtual because it doesn’t need to change for citizen or player

virtual void doSomething() = 0;

I chose to define a pure virtual version

of the doSomething() function in my base Actor class because all Human in Zombie

Dash are able to doSomething() and each type of Human doSomething() in a different way

virtual bool canBeDamagedByVomit() { return true; }

virtual bool isHuman() { return true; }

class Penelope : public Human

virtual void doSomething();

call doSomethingCom from Human class.

Allow users to control the Penelope by keyboard.

void reset\_goodies() { vaccine = 0; flamethrower = 0; landmines = 0; }

void add\_vaccine() { vaccine++; }

void reduce\_vaccine() { vaccine--; }

void add\_flamethrower() { flamethrower += 5; }

void reduce\_flamethrower() { flamethrower--; }

void add\_landmines() { landmines += 2; }

void reduce\_landmines() { landmines--; }

int get\_vaccine() { return vaccine; }

int get\_flamethrower() { return flamethrower; }

int get\_landmines() { return landmines; }

Those functions are only used for Penelope. Don’t need to be virtual.

class Citizen : public Human

virtual void doSomething();

check if the penelope’s distance < Nearest Zombie’s distance and penelope’s distance is less than or equal to 80, then call the follow\_player() function.

If nearest Zombies’ distance is less than or equal to 80, then call the runAwayFromZombie () function.

virtual void score\_doSomething();

call the function increaseScore(-1000);

class Wall : public Actor

virtual void doSomething() {} //do nothing

virtual bool isBlockActor() { return true; }

virtual bool canBeDamagedByFlame() { return false; }

virtual bool canBlockFlame() { return true; }

class Exit : public Actor

virtual void doSomething();

call StudentWorld to check if the penelope or citizen are overlap with the Exit.

virtual bool isBlockActor() { return false; }

virtual bool canBeDamagedByFlame() { return false; }

virtual bool canBlockFlame() { return true;

class Pit : public Actor

virtual void doSomething();

call StudentWorld to check if any person overlap with the Pit

virtual bool isBlockActor() { return false; }

virtual bool canBeDamagedByFlame() { return false; }

class Projectile :public Actor

bool doSomethingCom();

after 2 tick, set Dead.

Don’t need to be virtual because it doesn’t need to change for the Flame and vomit

virtual void doSomething() = 0;

The Flame and Vomit have different action. I don’t need to declare this Projectile class. Therefore, I set this to be pure virtual function

virtual bool isAFlame() = 0;

Set this to be pure virtual because the Flame returns true, vomit return false;

class Flame : public Projectile

virtual void doSomething();

call StudentWorld to check if any person overlap with this Flame

virtual bool isAFlame() { return true; }

class Goodie : public Actor

bool doSomethingCom();

check to see if it is currently alive. If not, return immediately

Don’t need to be virtual because it doesn’t need to change for other goodies

virtual void pickUp(Penelope\* p) = 0;

I chose to define a pure virtual version

of the pickUp (Penelope\* p) function in my base Goodie class because all goodies in are able to pickUp (Penelope\* p) and each type of goodie pickUp (Penelope\* p) in a different way

virtual bool isBlockActor() { return false; }

virtual bool isGoodie() { return true; }

virtual bool canBeDamagedByFlame() { return true; }

class Vaccine\_goodie : public Goodie

virtual void doSomething();

call doSomethingCom from Goodie class

virtual void pickUp(Penelope\* p);

call the penelope add\_vaccine();

class Gas\_can\_goodie : public Goodie

virtual void doSomething();

call doSomethingCom from Goodie class

virtual void pickUp(Penelope\* p);

call the penelope add\_flamethrower ();

class Landmine\_goodie : public Goodie

virtual void doSomething();

call doSomethingCom from Goodie class

virtual void pickUp(Penelope\* p);

call the penelope add\_landmines ();

class Landmine : public Actor

virtual void doSomething();

check to see if it is currently alive. If not, return immediately.

After 30 tick

Call StudentWorld to check if any human or Zombie are overlap with the landmine

If it is true, call StudentWorld landboom() function

Set Dead.

virtual bool isBlockActor() { return false; }

virtual bool canBeDamagedByFlame() { return false; }

virtual bool isALandmine() { return true; }

class Zombie : public Agent

void doSomethingCom();

check to see if it is currently alive. If not, return immediately.

Zombies move randomly for each odd tick

Also call the Zombie::doSomethingVomitCom();

It’s the same for Dumb Zombie and Smart Zombie, don’t need be virtual

void doSomethingVomitCom();

call StudentWorld to check if any human in the front (check 4 direction)

if it is true, call Studentworld compute\_vomit() function to the front

It’s the same for Dumb Zombie and Smart Zombie, don’t need be virtual

virtual void doSomething() = 0;

SmartZombie and DumbZombie have different action. Don’t need to declare this Zombie class. So I set it to be pure virtual

virtual void chanceToCreateVaccine() { return; }

This virtual function return nothing for smart zombie

virtual bool isZombie() { return true; }

class DumbZombie : public Zombie

virtual void doSomething();

call doSomethingCom() from Zombie class

virtual void score\_doSomething();

increaseScore(1000);

virtual void chanceToCreateVaccine();

call StudentWorld to create a Vaccine

class SmartZombie : public Zombie

virtual void doSomething();

if any human’s distance is greater than 80, call doSomethingCom() from Zombie class.

If it’s less or equal to 80

For each odd tick.

Call follow\_player() function from Agent class

Also call the Zombie::doSomethingVomitCom();

virtual void score\_doSomething();

increaseScore(2000);

2.

Most of time it’s OK. Rarely the current level has no citizen, but the Penelope couldn’t go to the next level.

Everything looks fine except when the game is finished, or the Penelope loses all lives. It has error for the delete m\_penelope in the cleanup() function in Visual Studio.

3.

Create a Vaccine Goodie also works for the Smart Zombie because the chanceToCreateVaccine() function is in the Zombie base class for Dumb and Smart zombie.

I didn’t separate this for smart and dumb zombie

4.

To test the Penelope class, I use keyboard to control it. I move it to go to the any blocking objects and not go into the blocking objects. It can run around and shoot fire when I hit space if it has gas goodies. Hitting Tab can drop a landmine if it has landmine goodies. Hitting Enter can cure infection if it is infected and has vaccine goodies.

To test citizen, I shoot the fire to them, they die and decrease 1000 score. I go closer to citizen, and they can follow the Penelope. I take them to go to the pit. They die and decrease 1000 score. I take them to be infected by the Zombie. After 500 tick, they die and become a Zombie and decrease 500 score. I take them to the smart zombie, they would run away from the smart zombie.

To test the Wall, I check whether any other objects go into the wall.

To test the Exit, I move Penelope to the Exit. If the current level has no citizen, I can go to next level. If current level has citizen, I cannot go to next level. I shoot flame to the Exit and it blocks the flame as well as the frame from the landmine explosion. The zombie can vomit when the Penelope is inside the Exit.

To test the Pit, I move Penelope to the pit, and it dies. I take citizen and zombie to the pit, they gone. I shoot flame to the pit, and it won’t block the flame as well as the vomit.

To test the Flame, I shoot the flame to citizen, zombie, goodies, and it can destroy them. It can also trigger the landmine explosion. The flame from the landmine can also destroy citizen, zombie, goodies, and it can destroy them, and triggers another landmine.

To test the Vomit, I move Penelope to the zombie. They can vomit and the Penelope get infection as well as the citizen. Also, it has 1/3 chance for zombie vomits again to the human.

To test the All Goodies, I move Penelope to the Goodies. They can be picked up by Penelope. Penelope also increases the corresponding Goodies number.

To test the Zombie (Dumb Zombie and Smart Zombie), I move Penelope to the Zombie, and they can vomit. I go closer to the smart zombie, and they would try to get closer to Penelope. I take the citizen to the smart zombie, and I move Penelope away. The smart zombie would try to get closer to the closest citizen. The zombies can be destroy by flame and pit.