Jacob Wahbeh

CS 32 Lec 1

28 February 2019

Project 3 Report

1. Function descriptions

Class Actor: derived from GraphObject

Many trivial functions such as getWorld(), alive(), getFlammable() that return private member variables which are used to distinguish the different derived classes. Functions like setInfectable(bool b) or setTriggersLandmine(bool b) are used for derived class constructors to initiate the properties of each class

getTick() and increaseTick() are used in classes that rely on the timing of the ticks. For the flame class, we use increaseTick() for every game frame and once that number reaches 2 we set its state to dead.

Virtual void doSomething() =0; - pure virtual because all derived class have a doSomething()

Virtual ~Actor() – blank, virtual so it properly deletes all sub classes

Bool Poverlap() – returns true if Penelope is overlapping another class

Double dist(x1,y1,x2,y2) - returns the Euclidean distance of 2 points

Virtual void activate(Actor\* a) – left blank in Actor class so we are able to call this virtual function within the Actors List in Student World; used in Activator class.

Virtual void dieFromFlameorPit() – also left blank, used in Mob classes and goodies for the correct process when they come in contact with a flam or pit.

Virtual void PickupGoodie(Goodies\* g) – also left blank, used with Penelope for picking up goodies and no other class

Virtual void useExit() – used in humans when they are able to exit the level appropriately

The previous four virtual functions cannot be pure virtual because it will cause all derived classes to be abstract, and since most of them do not use every single function, the program will not run or instantiate any class

Class Activator: derived from Actor

Virtual void doSomething() – calls the activateOnActors(Actor\*) if the Activator object is alive

Virtual because I later redefine the function within its derived classes

Class Mobs: derived from Actor

Void moveDirection( Direction d, int step) – if the mob can move in that direction, it will move step pixels. Uses a switch case for each direction and uses canMove() from the studentWorld class to determine if there are any classes that block movement in front of the mob.

Void determineDirection(double x, double y, int step) – used for the following mechanism in civilians and smart zombies. Determines if the mob’s getX() matches the x value passed, which would either be Penelope’s getX() for civilians or the closest Human getX() for smart zombies. If the x values are the same, it checks if the y values are greater, for which the direction will be up, and if less than the direction chosen would be down. Similiarly we can do the opposite for determining to move right or left but with equal y values. For the diagonals. I checked if both the x and y are greater than the mob’s values then the object would choose from either the up direction or right direction. Using randInt() and canMove(), I chose one of the 2 directions randomly, then checked if it was a viable place to move to. If not, then choose the other direction, and if that doesn’t work, return. Similarly, I repeated this process for the other three quadrants.

Void randomDirection() – used for dumb zombies path plan or smart zombies when there are no humans in range. It uses randInt() and chooses one of the 4 directions randomly.

void diefromFlameorPit() – calls otherDeath() then sets the mob to dead

Virtual void otherDeath() = 0; - pure virtual because I want to add the specifics of other mobs when they die to flame or pit. Is used in conjunction with diefromFlameorPit();

Class Human: derived from mobs

Int getInfLevel() – returns the infection level of a human object

Void setInfLevel(int n ) – used for incrementing infected humans or setting it to 0 when vaccine is used

Class Penelope: derived from Human

Virtual void doSomething() – checks if infected, increments the infection level by 1 every tick. If her infection level reaches 500, sets mob to dead and plays sound player die and returns. Then we check if the user hit a key. For the direction keys, we assign the direction appropriately and then check if Penelope is able to move to a place 4 pixels in that direction. If so we change her position with the moveTo() command. If the user presses a space bar, it first checks that Penelope has more than 0 flames in her inventory. If so, we decrement flames, play fire sound, and depending on her direction, create 3 flame objects in the 3 tiles directly in front of her. For this we used StudentWorld’s addActor() function. If the user presses tab, we check if she has mines in her inventory, then we decrement the mines and add a new Landmine actor to the coordinates Penelope is currently in. If the user presses enter, we check if she has more than 0 vaccines, then we decrement the amount of vaccines, set her infected Boolean to false, and set the infection level to 0.

PickupMed(), pickUpGas(), pickUpMine() – increases her inventory by the appropriate amount of goodie.

pickupGoodie() – calls the appropriate Goodie’s receiveGoodie() function.

UseExit() – checks if there are no more civilians left on the level. Then it adds the correct amount of score, plays the finish level sound, and calls StudentWorld’s levelComplete() function.

Class Civilian: Derived from Human

Virtual void doSomething() – if alive, it first checks if there are any zombies on the level then it calculates the x, y, and Euclidean distance of the closest zombie on the level using StudentWorld’s closestZombie() function. Then it checks if it is infected, it checks if it already played the infected sound, if not it plays the sound. Then it increases its infection level by one every tick. If it reaches 500, set the civilian to dead, play zombie born sound, subtract 1000 points from score. Choose a random integer from 1 to 10. If that integer is 1,2, or 3. Summon a smart zombie where the civilian died. Else summon a dumb zombie and return. Otherwise, the zombie increases its tick count every frame, and if the count is divisible by 2, then it runs its code. If there are no zombies left, or Penelope is closer than the closest zombie and Penelope’s distance is closer than 80. It uses mob’s determine direction function with Penelope’s coordinates. Else if there are zombies left and the closest zombie is left that 80 pixels away. It uses its runaway() function.

Void runaway(double x, double y) – creates a direction d variable and a double max. it sets max equal to the current distance the civilian is from the closest zombie. For each direction, it checks if it can move 2 pixels in that direction, and it determines if the new position will be a greater distance away than max. If so it sets it direction accordingly and max becomes the new distance. it does this for each direction and picks the direction farthest away from the zombie. If it found a new direction, it will move in that direction, else it returns.

Void otherDeath() – subtracts 1000 from the score, plays citizen die sound.

Void useExit() – increases score by 500, sets citizen to dead, plays citizen is dead sound.

Void Pdist() – determines penelope’s Euclidean distance to the civilian

Class Zombie: derived from Mobs

Virtual void doSomething() – if alive, we increase the tickcount every frame. If the tick count is divisible by 2, we run the code. If there is an object that triggers vomit in front of the zombie in the direction it is facing. Using randInt(), 1 out of 3 times, the zombie will summon a new Vomit object to be added in the tile in front of the zombie with the corresponding direction. Then it calls the zombie’s plan() function. Then it determines if it can move in the direction given to it from the plan() function, and if so it updates its coordinates using the moveDirection() function and decrements its distance Plan. Else it is blocked and its distance plan is set to 0.

Virtual void plan() = 0; sets different direction based on which zombie, pure virtual because zombie is never instantiated so it will never determine its own direction to move in.

Int getPlan() – returns the zombie’s distance plans

Void setPlan(int r) – sets the distance plan to r

Void decreasePlan() – decrease distance plan by 1

Virtual void otherDeath() – increase score by 1000, play zombie die sound, call drop() function

Virtual void drop() = 0; pure virtual because dumb zombies will act differently on death and drop a vaccine

Class Dumb: derived from Zombie

Virtual void drop() – used randInt() and 1 out of 10 times calls StudentWorld’s dropVaccine() function

Virtual void plan() – if the distance plan = 0, then sets a new distance plan between 3 and 10. Then it calls Mob’s randomDirection() function.

Class Smart: derived from Zombie

Virtual void drop() – increases score by 2000

Virtual void plan() – determines the x and y coordinates of the closest human using StudentWorld’s closestInfectable() function. Then if its distance plan = 0, it creates a new distance plan between 3 and 10. If the distance to the closest human is greater than 80, it uses Mob’s randomDirection() function. Else, it runs mob’s determineDirection() function.

Class Goodies: derived from Activator

Virtual void Pickup() = 0; used to determine which goodie Penelope picked up

Virtual void activate(Actor\* a) – calls Penelope’s pickupGoodie() Function

Void receiveGoodie() – increase score by 50, set alive to false, play got goodie sound, and run the Pickup() function.

Virtual void dieFromFlameorPit() – set alive to false.

Class GasCan: derived from Goodies

Virtual void pickup() – run Penelope’s pickupGas() function

Class MedKit: derived from Goodies

Virtual void pickup() – run Penelope’s pickupMed() function

Class MineBox: derived from Goodies

Virtual void pickup() – run Penelope’s pickupMine() function

Class Projectiles: derived from Activator

Virtual void doSomething() – if the tick count = 2, set dead. Else increase the tick count and run Activator∷doSomething();

Class Flame: derived from Projectiles

Virtual void activate(Actor\* a) – calls the appropriate Actor’s diefromFlameorPite() function

Class Vomit: derived from Projectiles

Virtual void activate(Actor\* a) – sets the infection state to true for the appropriate actor

Class Wall: derived from Actor

Virtual void doSomething() – return, does nothing

Class Pit: derived from Activator

Virtual void activate(Actor\* a) – calls the appropriate Actor’s diefromFlameorPite() function

Class Exit: derived from Activator

Virtual void activate(Actor\* a) – calls the appropriate Actor’s useExit() function.

Class LandMine: derived from Activator

virtual void doSomething() – increase the tickcount, if it reaches past 30, the land mine becomes active and calls Activator∷doSomething()

Virtual void activate(Actor\* a) – if a triggersLandmines, the calls landmines setoff() function

Virtual void dieFromFlameorPit() – set alive to false and call setoff() function

Void setoff() – sets the landmine to dead, plays explosion sound, and adds a new flame object at the coordinates of the landmine. Then it checks if any actors block flames in the 8 tiles surround the landmine using student World’s blocksFlame() function. If that tile doesn’t block flames it creates a new flame object in the appropriate tiles surrounding the landmine. Then it adds a pit object at the coordinates of the landmine.

Constructors

Each object has it constructed set up as follows

*Objectname*(StudentWorld\* sw, int imageID, double startX, double startY, Direction dir = 0, int depth = 0)

: *BaseClassName*(sw, imageID, startX, startY, dir, depth)

{

Any properties specific to that class such as setFlammable(true);

}

Actor Constructor

{

m\_sw = sw;

m\_alive = true;

m\_tick = 0;

m\_infected = false;

}

Mobs Constructor

{

setBlocksVomit(false);

setFlammable(true);

setBlock(true);

setPittable(true);

setTriggersLandmine(true);

setInfectable(false);

}

Human Constructor

{

setInfectable(true);

m\_infectionLevel = 0;

}

Activator Constructor

{

setBlocksVomit(true);

setFlammable(false);

setInfectable(false);

setPittable(false);

setBlock(false);

setTriggersLandmine(false);

}

Goodies Constructor

{

setBlocksVomit(true);

setFlammable(true);

setInfectable(false);

setPittable(false);

setBlock(false);

setTriggersLandmine(false);

}

Projectiles Constructor

{

setBlocksVomit(true);

setFlammable(true);

setInfectable(false);

setPittable(false);

setBlock(false);

setTriggersLandmine(false);

}

Penelope Constructor

{

m\_mines = 0;

m\_flames = 0;

m\_vaccs = 0;

}

Wall Constructor

{

setBlocksVomit(true);

setFlammable(false);

setInfectable(false);

setPittable(false);

setBlock(true);

setTriggersLandmine(false);

}

Exit Constructor

{

setBlocksVomit(false);

}

Landmine Constructor

{

setFlammable(true);

}

Zombies Constructor

{

setInfectable(false);

m\_plan = 0;

}

All other classes constructors have nothing insides the body of the function.

Class StudentWorld: derived from Gameworld

Contains a Boolean for levelCompleted, initiated to false at the start of each level. Also has a Penelope pointer, and a list of Actor pointers

Int Init() – uses a stream string, getLevel(), and filler ‘0’ for initializing the levelFile string.

If the level reaches 100 or there are no levels found, it returns GWSTATUS\_PLAYER\_WON.

Else if the level was not loaded correctly, it cerr “improper format”. Else the level loads successfully and searches through the corresponding level data file. It passes through each class and creates a new object. The Penelope pointer is set to new Penelope and every other object is pushed onto the Actor pointer list. All objects are initiated with appropriate image ids and coordinates depending on their position on the level data file. Return continue game.

Int move() – if Penelope is alive run her doSomething(), else decrement her lives and return player died. Then, iterate through the actor list and if the actor is alive, run the do something command for each actor. If levelComplete is true, playsound level finished and return finished level. Iterate through the list again and delete any objects that are dead. For the status bar use a stream string and appropriate function getScore(), getLives(), etc to create the whole bar. Uses fill(‘0’) for the score. Return continue game.

Void cleanup() – delete Penelope, iterate through list and delete all actors. Set lvlcomplete to false.

~StudentWorld() – call cleanup()

Bool canMove(double x, double y, Actor\* a) – iterator through list, if there is an object that blocks movement, and the absolute value of the difference between its coordinates and (x,y) is less than Sprite Width and height then return false. Do the same for Penelope. Return true otherwise.

Bool overlap(double x1, double y1, double x2, double x2) – if (x1-x2)^2 + (y1-y2)^2 ≤ 100 then return true, false otherwise.

Void activateOnActors(Actor\* a) - if a and Penelope are alive and overlapping call a’s activate function with Penelope as a parameter. Iterate through list and if the actor and a are alive and overlapping then call a’s activate function with the corresponding actor from the list as a parameter.

Bool blocksFlame(double x, double y) – iterate through list, if the actor is not flammable, then the object will return true if (x,y) are within the objects hitbox. False otherwise.

Bool citizensLeft() – iterates through list and returns true if any actor is infectable, false otherwise.

Bool zombiesLeft() - iterates through list and returns true if any actor is both pittable and not infectable, false otherwise.

Bool blocksVomit(double x, double y) - iterate through list, if the actor blocks vomit, then the object will return true if (x,y) are within the objects hitbox. False otherwise.

Bool triggerVomit(double x, double y) – iterates through list and returns true if any actor is infectable and overlapping with the specified coordinates, false otherwise.

Actor\* closestInfectable(double x, double y) – Initiates an actor pointer a to Penelope and a double min to the Euclidean distance from (x,y) to penelope’s coordinates. Then it iterates through the actors list and if there is an infectable whose distance is less than min, that actor now becomes a and the min becomes its distance. Then it returns a.

Actor\* closestZombie(double x, double y) – same code as before but now were are checking for objects that are not infectable and are pittable.

Bool overlapAnything(double x, double y) – checks to see if anything is overlapping with the specified coordinates and returns true. False otherwise.

Void drop Vaccine(double x ,double y) – randomly picks one of the 4 directions using randInt(). Then it creates a destination coordinate based on the direction that is 1 tile away. For example if direction = up, then destination is (x,y+Sprite Height). Then it checks if anything overlaps at those coordinates and if not creates a new Vaccine Goodies there.

2. I finished all the functionality of the project, but for a long time I could not implement the landmine trigger function. The problem was that the landmine would not disappear and every time it would overlap with flame object it would keep spawning new flame objects on top of it, eventually leading to a crash. However, I just implemented more checks for if the Activator class was alive before each function call and that solved the problem.

3. It was not specified how the zombies or citizens were to act when they tried to follow Penelope but there was a wall in between them. For the citizens I decided that they should return and not do anything as long as they are closest to Penelope or there are no zombies nearby. And for the smart zombies, they continue to move slightly, but generally stay in the same position. Also, it was not clear how to search for zombies in the actor list so I used a combination of 2 factors: not infectable and pittable. All mobs can die from pits but only humans are infectable.

4. Test cases

For Penelope, once I implemented her movement, I created a blank level and made sure she moved the correct number of pixels in each direction. I made sure her flamethrower spew out fire in each direction according and that the landmine deployed at the correct spot below her feet. Then I checked to see if picking up goodies incremented her inventory by the correct amount and that the vaccine cures her infection and sets it to 0. Then I check that she does not run through wall or other mobs, and I compared it to the sample game to make sure that the pixels lined up. Further I had to make sure citizens and zombies did not run through Penelope. Lastly I needed to know that the dying mechanism worked, so I created a pit on the first level and made sure everything cleaned up when she died.

Walls were pretty simple, just making sure no mob clips into the wall.

Goodies – created multiple goodies on a level of all the types. Made sure they only interact with Penelope and fire. Once a goodie overlaps with flame it disappears, and when Penelope picks one it, it correctly adds to her inventory and sets the goodie to false.

Flame – made sure that each flame object is blocked by non-flammable objects. I tried using the flamethrower into a wall which didn’t output anything. Then I made sure it worked in all directions and killed anything the flame came in contact with. Also checked to make sure the score and sounds played accordingly.

Vomit – created a zombie on the first level and had a civilian right next to it. Then I waited 25 seconds to make sure that it died and turned to a zombie. Also I tried it with Penelope which worked as expected

Pit – I made sure that it did not block any mobs from moving onto it. Zombies, citizens, and Penelope all died from them. I checked that vomit and flame were blocked by pits by shooting the flamethrower at it and putting a zombie next to a pit

Landmine – I tested this by trying to walk on it with Penelope. Then I used citizens and mobs. Then I checked to make sure that fire triggers the landmine. Also, a chain of adjacent landmines should all destruct.

Exit – I made sure that Penelope can only exit when there are no remaining citizens on the current level. The exit only lets civilians and Penelope use it. Zombies can walk on it and vomit can overlap with an exit, but they do not interact. Shooting the flamethrower at an exit blocks it.

Citizen – first I checked to see if it died from flame, then I tried moving it in a random direction once Penelope got within its range. Then I implemented its following feature and spent time making sure it chose the right direction to follow me in, even if there was a wall in the way. Then I checked to see that it got infected appropriately by placing it next to zombies and I made sure that they set off landmines and did not pick up goodies. The exit mechanism worked and Penelope could not exit if there were any citizens left. Then I added the runaway from zombies code and I made sure it always chose the farthest possible distance to escape the zombies.

Dumb Zombie – I fired the flamethrower at is to make sure it died and added points. Then I implemented the random direction and that worked fine. I walk next to it to see that the vomit triggers, and I also tried that with a civilian. It died from pits and landmines, and it did not pick up goodies.

Smart Zombie – this was pretty much the same as the dumb one. To test its following mechanism I created a level with only a civilian and a smart zombie in a box and observed how they follow each other around.