Komal Prakash

304980756

Report

1. Actor class: I chose to make my actor’s do something pure virtual because an actor type alone has no functionality. It only does something if it is of a specific subclass of actor. I includes getters for the dead status and hit points because other sub classes need that information. Also, I added a setDead function as virtual so that various Actors can be set to a dead status. I made it virtual so specific subclasses like Bacteria and Socrates can make the necessary sounds or turn into food (in the case of bacteria) and add to the setDead function. I also added bool functions like damageable, blocksBacteria, isEdible, and preventsLevelMoveOn (these all need to be altered by other classes so they’re virtual, but some actors follow the same pattern and just use the Actor version, so they do not need to be pure). I added these functions so that when the Actors of my Student World’s myMembers vector iterates through each actor, it has access to this various information about the type of Actor it is coming across. I added the withinDish in order to check that various actors are within the dish and do not leave the dish. I also added an imHurt function as virtual so that other classes can play a different sound when they’re hurt. I also added a function to access a student world pointer so that every actor can access the functions of the student world.

**virtual** **void** doSomething()=0;

**int** getHitPoints();

**void** setHitPoints(**int** myPts);

    StudentWorld\* getWorld();

**virtual** **void** setDead();

**bool** isAlive();

**bool** damageable();

**virtual** **bool** blocksBacteria();

**virtual** **bool** isEdible() **const**;

**bool** withinDish(**double** x, **double** y);

**virtual** **void** imHurt();

**virtual** **bool** preventsLevelMoveOn();

1. Socrates class: I included getters for the flames and sprays because only socrates has these and also an add flames because only he needs to add more flames. Also, a damageAmount function that reduces his hit points from the various bacteria. I also added the setDead that was altered to play the player sound dead when he dies.

**void** addFlames(**int** flameNumber);

**void** damageAmount(**int** damage);

**virtual** **void** imHurt();

**int** getSprayCharges();

**int** getFlameCharges();

**virtual** **void** setDead();

1. Bacteria class: The doSomething is altered by each bacteria type. I added a getter for the food eaten which is common to all bacteria and an increase function for food eaten along with a function that checks if the bacteria overlap with Socrates and then damage him by each of their respective impacts. I also included a function that checks if the bacteria has consumed 3 foods and then make another bacteria of that same type using “hitTotalFood” function. I also have an overlapFood function to check if any bacteria overlaps a food and then removes the bacteria. I also altered the preventsLevelMoveOn function to be true for all bacteria (while bacteria still exist, the level must continue).

**virtual** **void** doSomething();

**void** increaseFoodEaten();

**int** getFoodEaten();

**void** emptyFoodEaten();

**bool** socOverlapAndDamage(**int** howMuch);

**bool** hitTotalFood(string whichBac);

**void** overlapFood();

**virtual** **bool** preventsLevelMoveOn();

1. EColi class: The Ecoli class has its own doSomething and includes a function to check if it is 256 pixels from Socrates (pixelsFromSoc) and also an altered setDead function that plays the appropriate sound. I also have an imHurt function that plays the appropriate sound when Ecoli dies.

**virtual** **void** doSomething();

**bool** pixelsFromSoc();

**virtual** **void** setDead();

**virtual** **void** imHurt();

1. Salmonella class: The Salmonella class has its own doSomething and includes a function to check if it is within 128 pixels from food (within128Pixels) and also an altered setDead function that plays the appropriate sound. I also have an imHurt function that plays the appropriate sound when Salmonella dies. I also have the function getMovePlan to decide whether or not a salmonella wants to follow its movement plan. I also have a getter and setter for the movement plan and then a MOVE() function that combines the movementPlan and within128Pixels functions in an if-else capacity.

**virtual** **void** doSomething();

**int** getMovePlan();

**void** setMovePlan(**int** newPlan);

**void** movementPlan();

**void** within128Pixels();

**virtual** **void** setDead();

**virtual** **void** imHurt();

**void** MOVE();

1. Aggressive class: This class just has a nearbySocrates to see if it is near Socrates by 72 pixels and then has its own doSomething with all the prior functions.

**virtual** **void** doSomething();

**bool** nearbySocrates();

**7. Regular Sal Class: Doesn’t need any new functions, just a doSomething using the previously defined functions.**

**virtual** **void** doSomething();

8.Projectile class: moves each type of projectile until it hits socrates and then dies and then also travels until it hits its max travel length.

**virtual** **void** doSomething();

**9.** FlameProj class: does not need any additional classes

10. SprayProj class: does not need any additional classes

11. GoodieOrFung class: The doSomething will check if a goodie overlaps with Socrates and take approatrate action. It also checks if goodie has exceeded its lifetime and if it should stay.

The pick up function is used by all goodies and returns Socrates if he has picked up the goodie

**virtual** **void** doSomething();

**virtual** **void** pickUp(Socrates\* s) = 0;

12. RestoreHealth class: alters the pickup function with correct score and points

**virtual** **void** pickUp(Socrates\* s);

13. FlameGoodie class: alters the pickup function with correct score and points

**virtual** **void** pickUp(Socrates\* s);

14. ExtraLife class: alters the pickup function with correct score and points

**virtual** **void** pickUp(Socrates\* s);

15. Fungus class: alters the pickup function with correct score and points

**virtual** **void** pickUp(Socrates\* s);

**16.**Pit class:

**virtual** **void** doSomething(); does nothing

**virtual** **bool** preventsLevelMoveOn(); It also returns true for preventing the level to move on.

**17.** Food class:

**virtual** **void** doSomething(); does nothing

**virtual** **bool** isEdible() **const**; Food also alters the isEdible function to be true.

18. Dirt class:

**virtual** **void** doSomething(); does nothing

**virtual** **bool** blocksBacteria(); returns true for blockingBacteria

StudentWorld Functions:

Student world has a createBacteria function that can be used to add new bacteria’s to the myMembers list. It also has an addSpray and addFlame method to add sprays and flames to the world. It also has overLapAndDamagable to check if any sprays or flames overlap any damageable object and reduce them indicated hit points. My makeFood also adds foods to my myMembers list. I also created a getAngleToNearbySocrates that returns the angle to Socrates from the Actor inputted. I also created a getAngleToNearbyEdible to return the angle towards the closest food from the actor. I used isBacteriaMovementBlockedAt to find out if an actor is blocked by dirt at a certain location. I used getOverlappingEdible to find return the food if the actor overlaps the food.

**bool** overlapAndDamageable(**double** posX, **double** posY,**int** hitPts);

    Socrates\* socOverlap(Actor\* a);

**void** createBacteria(string typeBac, **double** posX, **double** posY);

**void** addSpray(**double** posX, **double** posY, **int** dir);

**void** addFlame(**double** posX, **double** posY,**int** dir);

**virtual** **int** init();

**virtual** **int** move();

**virtual** **void** cleanUp();

**void** makeFood(**double** x, **double** y);

**bool** getAngleToNearbySocrates(Actor\* a, **int** dist, **double**& angle) **const**;

      // Return true if there is a living edible object within the indicated

      // distance from actor a; otherwise false.  If true, angle will be set

      // to the direction from actor a to the edible object nearest to it.

**bool** getAngleToNearestNearbyEdible(Actor\* a, **int** dist, **double**& angle) **const**;

      // Is bacterium a blocked from moving to the indicated location?

**bool** isBacteriaMovementBlockedAt(Actor\* a, **double** x, **double** y) **const**;

    Actor\* getOverlappingEdible(Actor\* a) **const**;

2. I did not make the salmonella move fully normally. There is some glitchy-ness in their movement. Also did not account for overlap of food and pits when multiple are populated.

3. I decided to add a check for the E.Coli to make sure they do not leave the petri dish using my withinDish function.

4. A description of how you tested each of your classes (1-2 paragraphs per class).

As I was debugging I would create print statements at various locations to check what functions were being entered and which statements were being skipped. I played the game over and over and also used breakpoints to check what my values were at different moments. For example, when testing my getAngleToNearestNearbyEdible and such functions, I used the breakpoint feature to make sure that the x and y coordinates were pointing to the correct object and that the correct angle was being found. I also made various modifications to the code and played the game over and over to see how each component was functioning. For example, when testing whether my movement functions were working for the various bacteria, I reduced the number of aggressive and regular salmonella to zero in order to just test the functionality of the Ecoli. I did the same for other bacteria types. Printing out various statements also allowed me to see whether I was entering the correct functions and whether certain if statements were returning true. For example, when testing the functions for whether the bacteria was a certain number of pixels from socrates, I made sure it entering the if statement if the object was in the vicinity of Socrates, by adding a print statement after the if statement.