Anubha Kale

2/27/2020

CS32 W20

CS32 Project 3: Kontagion Report

1. High-level description of each public member function in each of the classes

Class Actor

`virtual bool initOverlapMatters()`
This function is called by StudentWorld's init() during the game setup process. It is virtual because certain actors (namely Food and Pit) will want to return true to indicate that the overlap does matter. This allows me to ensure that no food is initially created on top of another and that the pit is not overlapping any Food or another Pit. The Actor's implementation of the function returns false and the function is not pure virtual because most Actors will not want to redefine it.

`virtual void doSomething() = 0;`
This function is defined in Actor because every Actor is able to do something during a tick. It is pure virtual because the various types of Actors have different things to do during each tick.

`bool isDead() const`
This function is defined in Actor because every Actor is either alive or dead. The function returns true if the Actor is dead. It is const because it should not modify any of the Actor's data.

`void setDead()`
This function is defined in Actor because every Actor has the potential to be setDead at some point. The function sets the Actor to dead so that it will soon be removed from the game.

`StudentWorld* world() const`
This function returns a pointer to the world that an Actor lives in. It is const because it shouldn't modify the Actor's data. It is defined in the Actor class because every Actor belongs to a world that might need to be accessed from outside the Actor class.

`virtual bool takeDamage(int damage)`
This function is defined in Actor because all Actors can either suffer damage or cannot suffer damage. The function returns false in Actor, but is virtual so that Actors that *can* take damage can redefine the function to do so and return true. The parameter **damage** indicates how much damage the Actor is hit with.

`virtual bool blocksBacteriumMovement() const`
This function is defined in Actor because each Actor should be able to tell a user if it blocks bacterium movement or not. The function is const because it shouldn't modify any data. The function implementation in Actor returns false because most Actors do not block bacterium movement. However, it is virtual so that Actors that do block bacterium movement can redefine the function.

virtual bool isEdible() const
This function returns true if the Actor is edible and false otherwise. The implementation of this in Actor simply returns false, since most Actors are not edible. It is virtual because Actors that are edible can redefine the behavior of the function to return true.

virtual bool isDamageable()
This function returns true if the Actor is damageable and false otherwise. The implementation in Actor returns false. It is a virtual function defined in Actor so that certain Actors can redefine the function to return true. All Actors are either damageable or not damageable.

virtual bool preventsLevelCompleting() const
This function returns true if the existence of the Actor in the game prevents a level from being completed. Its implementation in the Actor class simply returns false, since the existence of most Actors do not prevent the level from being completed. The function is virtual so that Actors like bacterium can redefine the function to return true.

Class Dirt

virtual void doSomething()
This function immediately returns since Dirt should simply sit there during each tick.

virtual bool takeDamage(int damage)
This function sets the dirt to dead as long as damage is positive, then returns.

virtual bool isDamageable() const
This function returns true since Dirt is damageable.

virtual bool blocksBacteriumMovement()
This function returns true since Dirt does block bacterium movement.

Class Food

virtual bool initOverlapMatters()
This function returns true since overlapping food is does matter- it is not allowed.

virtual void doSomething()
Similar to Dirt, Food’s do something method immediately returns since Food does not do anything but exist and wait for another actor to eat it.

virtual bool isEdible() const
This function returns true since food is edible.

Class Pit

virtual void doSomething()
In this function, the Pit emits one of the bacteria remaining in its inventory. There is a 1 in 50 chance that the pit will emit a bacterium, and each bacterium remaining in the inventory has an equal chance of being emit. If all bacteria have been emitted, the function sets the Pit to the dead state.

virtual bool initOverlapMatters()
This function returns true since in the StudentWorld's init() function, Pits should not overlap one another, and Dirt should not overlap Pits.

virtual bool preventsLevelCompleting()
This function returns true since the existence of a Pit means that all bacteria have not yet been emitted, and the player is not done with the level.

Class Projectile

virtual void doSomething()
This function makes the projectile move sprite width forward in the direction it is pointed towards. It checks if it has moved its max travel distance and sets itself to dead if it has. It also checks if the projectile overlaps with any damageable actor and if so, it damages that actor by the myDamageAmt member variable and sets itself to dead.

Class Spray

This class has no member functions.

Class Flames

This class has no member functions.

Class Goodie

virtual bool takeDamage(int damage)
This function is defined in the Goodie class. It sets the Goodie to dead. It is not redefined in any Goodie class because all Goodies do the same thing when damaged.

void decrementLifetime()
This function decrements the Goodies lifetime by 1. It is defined in Goodie because all Goodie objects should be able to decrement their lifetime (in the same way).

int getLifetime()
This function returns the lifetime of the Goodie. It is defined in Goodie because all Goodies should be able to access their lifetime in the same way.

virtual void doSomething()
This function checks if the Goodie is overlapping Socrates. If so, it makes Socrates pick up the Goodie and receive the benefits/harm from the Goodie appropriately by calling the pickup function, and sets the goodie to dead. If it didn't overlap with Socrates, then it decrements the lifetime of the Goodie. If the Goodie's lifetime is over, it sets the Goodie to the dead state. The function is defined in the Goodie class because all Goodies do partly the same thing when they need to "do something." The part that is different is what happens when Socrates picks up the Goodie, and this is a separate function that is individually defined for each Goodie.

virtual bool isDamageable()
This function returns true since all Goodies are damageable.

virtual void pickUp(Socrates* s) = 0
This function is pure virtual since all Goodies must do something specific to Socrates when they're picked up by Socrates. The derived classes of Goodie are forced to implement this function. The parameter s is a pointer to the Socrates object who has picked up the Goodie.

Class RestoreHealthGoodie

virtual void pickUp(Socrates* s)
This function increases the player's score by 250 and restores Socrates health (to original health), and plays the goodie pick up sound.

Class FlamethrowerGoodie

virtual void pickUp(Socrates* s)
This function increases the player's score by 300 and gives 5 extra flame thrower charges to Socrates, and plays the goodie pick up sound.

Class ExtraLifeGoodie

virtual void pickUp(Socrates* s)
This function increases the player's score by 500 and gives Socrates an extra life, and plays the goodie pick up sound.

Class Fungus

virtual void pickUp(Socrates* s)
This function decreases the player's score by 50 and decreases Socrates' health by 50 hit points, using the takeDamage function.

Class Agent

virtual bool takeDamage(int damage)
This function implements the takeDamage function in a way that all agents can use it. Agents include Socrates and the bacteria. The function plays the hurt sound if the damage causes the player to lose hit points but remain alive. The function plays the dead sound if the damage caused the player to die. When a bacterium dies, the function increases the player's score by 100 and there is a 50% chance that the function will create a new food in the place of the bacteria that just died.

virtual bool isDamageable() const
This function returns true since all agents are damageable.

int numHitPoints() const
This function returns the Agent's number of hit points and is defined in Agent since all Agents have hit points.

void restoreHealth()
This function restores the health (number of hit points) of the agent to its original value. It is defined in Agent since all Agents should be able to have their health restored. Although in this game only Socrates ever uses restore health, features could be added to the game that require other Agents to have their health restored as well.

virtual int soundWhenHurt() const = 0

This function is pure virtual because each type of Agent has its own sound that it makes when it is hurt.

virtual int soundWhenDie() const = 0
This function is pure virtual because each type of Agent has its own sound that it makes when it dies.

Class Socrates

virtual void doSomething()
Socrates' do something function moves Socrates appropriately based on the key pressed by the game player. It also adds other Actors like Flames and Sprays to the game if that is what the user commands. It also keeps track of Socrates' remaining number of flames and sprays.

virtual int soundWhenHurt() const
This function returns the sound constant to be played when the player is hurt.

virtual int soundWhenDie() const
This function returns the sound constant to be played when the player dies.

void addFlames()
This function increases the number of Flamethrower charges Socrates has by 5.

int numFlames() const
This function returns how many Flamethrower charges Socrates has remaining.

int numSprays() const
This function returns how many Spray charges Socrates has remaining.

Class Bacterium

int getFoodEatenCount()
This function returns the number of food that the bacteria has eaten so far during its lifetime. This function is in the Bacterium class because all Bacteria have a food eaten count.

void incrementFoodEatenCount()
This function increments the Bacteria's food eaten count by 1. This function is in the Bacterium class because all types of bacteria can eat food.

void resetFoodEatenCount()
This function resets the food eaten count to 0 and resides in the Bacterium class because all types of bacteria need to reset their food eaten count in the same way.

int getMovementPlanDist()
This function returns the movement plan distance of the Bacteria and resides in the Bacterium class because all types of Bacteria have a movement plan distance (which represents the distance that the bacteria plans to keep moving in its current direction).

void decrementMovementPlanDist()
This function decrements the movement plan distance by 1 and is in the Bacterium class because all types of Bacteria make use of this function.

void resetMovementPlanDist()
This function resets the Bacterium's movement plan distance to 10 and resides in the Bacterium class because all types of Bacteria use this reset function to do the same thing.

virtual bool preventsLevelCompleting() const
This function returns true and is in the Bacterium class because the existence of any type of bacteria in the game means that the player has not completed the level (killed all the bacteria).

Class EColi

virtual void doSomething()
This function makes the EColi move in the manner outlined in the project spec. The ecoli follows Socrates around and constantly tries to move towards Socrates and kill him, ignoring the food unless they happen to stumble over it.

virtual int soundWhenHurt() const
This function returns the int constant representing the sound that the ecoli should play when it is hurt.

virtual int soundWhenDie() const
This function returns the int constant representing the sound that the ecoli should play when it dies.

Class Salmonella

virtual int soundWhenHurt() const
This function returns the sound that all salmonella should play when they are hurt. It is defined in this class because it is a common sound for all types of salmonella.

virtual int soundWhenDie() const
This function returns the sound that all salmonella should play when they die. It is defined in this class because it is a common sound for all types of salmonella.

Class RegularSalmonella

virtual void doSomething()
This function makes salmonella perform various tasks during each tick. It checks if it overlaps with a Socrates and if so, it damages Socrates. It also checks if it overlaps with (eats) food during the tick. It multiplies itself if it has eaten a total of 3 food objects and then resets its food eaten count to 0. The function implements the movement of the Regular Salmonella in such a way that it moves in a random manner, changing direction if it cannot move forward or after moving its movement plan distance.

Class AggressiveSalmonella

virtual void doSomething()
This function makes the aggressive salmonella perform various tasks during each tick. It checks if it overlaps with a Socrates and if so, it damages Socrates. It also checks if it overlaps with (eats) food during the tick. It multiplies itself if it has eaten a total of 3 food objects and then resets its food eaten count to 0. The function implements the movement of the aggressive salmonella in such a way

that it moves in a random manner if it is not nearby to Socrates. If it is near Socrates, it tries to move towards Socrates and kill him.

Class StudentWorld

virtual ~StudentWorld()
This function destructs StudentWorld. It is virtual since it is a derived class from GameWorld which also has a virtual destructor. This function calls the cleanup() function.

virtual int init()
This function initializes the Actors that start off in the Petri dish at the beginning of the game, beginning of a level, or when the user loses a life but has more lives left. The function adds a Socrates player, the level number number of pits, some food objects, and some Dirt to the petri dish.

virtual int move()
The move function asks all of the active actors in the game world to do something by calling their doSomething() function. It also checks for dead actors and deletes them from the game world. It also adds new objects to the game like goodies and fungi. Additionally, it updates the game stats text displayed at the top of the screen. It returns one of three values indicating to the game framework whether the player is dead, finished a level, or alive and continuing the current level.

virtual void cleanUp()
This function frees all actors that are currently in the game, and is called by the game framework to destroy a level/ when the game ends.

bool initOverlapIssue(int x, int y)
This function determines if there is an overlap issue with initializing a new actor in this (x,y) location. It checks to make sure that no Actor with initOverlapMatters returning true would overlap with an object placed at the x,y coordinate passed to this function.

void addActor(Actor* a)
This function adds the actor passed to the function to the list of actors held in m_actors in StudentWorld.

bool damageOneActor(Actor* a, int damage)
If actor a overlaps some live actor, this function damages that actor by the damage value passed to the function and returns true. If not, it returns false.

bool isBacteriumMovementBlockedAt(double x, double y)
This function returns true if a bacteria is blocked from moving to the x,y location passed to the function. This would be if there was a Dirt overlapping at that location or if the location would cause the bacteria to move outside the petri dish. If the bacteria is allowed to move to the x,y location, it returns false (indicating its movement is not blocked at that location).

Socrates* getOverlappingSocrates(Actor* a) const

If the actor a overlaps with Socrates, the function returns a pointer to Socrates. Otherwise, the function returns nullptr.

Actor* getOverlappingEdible(Actor* a)
If the Actor a is overlapping with an edible object, the function returns a pointer to that edible object. Otherwise, it returns nullptr.

bool getAngleToNearbySocrates(Actor* a, int dist, int& angle) const
This function returns true if this world's Socrates is within **dist** distance from the Actor a. In that case, it sets the angle argument to the direction in degrees from Actor a to Socrates. The function is constant because it shouldn't change any data in the StudentWorld.

bool getAngleToNearestNearbyEdible(Actor* a, int dist, int& angle)
This function returns true if an edible object exists within **dist** distance away from the Actor a, otherwise it returns false. If true, the angle will be set to the direction from Actor a to the nearest nearby edible object.

2. A list of all functionality that you failed to finish as well as known bugs in your classes: I did not create a functions for Bacteria that would factor out the common/repetitive code in the doSomething functions of each of the 3 bacterias.

3. I didn't adhere to 2 of the spec requirements for Spray in order to make it function like it did in the sample Kontagion game provided to us. One was that the Spray must be created at a distance SPRITE_WIDTH in front of Socrates. The sample Kontagion game provided to us did not have this behavior so I implemented my Spray to be created directly in front of Socrates (like the sample Kontagion game). This makes it look better and easier to kill bacteria closer to you. I also didn't make the Spray have a max travel distance of 112 like the spec said. This is because in the sample Kontagion game, the Spray reaches near the center of the petri dish and when I made the max travel distance 112, the Spray could not reach very far. Instead, I made the max travel distance 120 so it would reach actors (dirt and bacteria) near the center of the petri dish, like the sample game.

4. I tested the various classes by playing the game MANY times, trying out different moves, and ensuring that ALL of the spec's requirements were met for how each object should behave in the game. I played through multiple levels to make sure the levels were initialized properly as well. I also played the sample Kontagion game and compared my game with that.

   How I tested each of my classes
   Dirt:
   I tested the dirt class by shooting my spray at a Dirt and making sure that the dirt was deleted from the game. I also tested that the dirt did not overlap with any food or pits by running the game nearly 100 times to check that.

   Food:
   I tested the food by running the game many times to make sure that the food never overlapping with each other or with the pits. I made sure that the food would disappear when a bacteria went over it. I also made sure that the food would not obstruct bacteria, sprays, or flames. I made sure that the food could not be damaged by flames or sprays.

Flame:
I tested the flame by playing the game many times to make sure that the flames would:

- Run out once 5 flames was fired
- Kill bacteria in the vicinity
- Kill goodies in the vicinity
- Immediately go away if it burned something
- Not block other objects from moving near or onto it

Disinfectant Spray:

I tested to make sure that the spray would

- Damage bacteria appropriately
- Damage dirt
- Not damage food
- Immediately go away if it killed something
- Not block other objects from moving near or onto it

Pit:

I tested to make sure that

- An appropriate number of pits were put out each level
- The pits emit the appropriate number of each type of bacteria in a random manner as specified
- The pit did not get damaged by sprays or flames
- The pit did not block the movement of sprays or flames
- The pit did not block other objects from moving near or onto it

Regular Salmonella:

I tested to make sure that it

- Would move in a random manner, switching its direction when it ran out of movement plan distance
- Would damage Socrates with 1 point of damage
- Did not follow Socrates (behavior of agg salmonella)
- Multiplied upon eating 3 food
- Searched for and ate food as described in the spec
- Increased the players score by 100 when eaten
- Played sounds when hurt and dead

Aggressive Salmonella:

I tested to make sure that the aggressive salmonella would

- Move towards the Socrates when he was nearby to the Salmonella
- Search for food if Socrates was not nearby
- Move in a random direction if it was not near Socrates and could not find food nearby / was blocked from going in the direction of nearby food
- Multiplied upon eating 3 food
- Searched for and ate food as described in the spec
- Increased the players score by 100 when eaten
- Played sounds when hurt and dead

EColi:

I tested to make sure that the Ecoli:

- Would always follow Socrates no matter how far he was
- Ignored food unless it happened to eat food on the way to Socrates
- Died when hit enough by Socrates
- Damaged Socrates when it hit Socrates

Socrates:

I tested to make sure that my Socrates:

- Moved clockwise when then right button was pressed, moved counter clockwise when the left button was pressed
- Shot flames when enter was pressed
- Shot spray when space was pressed
- Could only shoot a max of 20 sprays at once

RestoreHealthGoodie:

I tested to ensure that the RestoreHealthGoodie

- Restored Socrates health when overlapped by Socrates
- Increased player's score by 250
- Appeared randomly inside the perimeter of the petri dish

FlamethrowerGoodie:

I tested to ensure that the FlamethrowerGoodie

- Increased the score by 300 points
- Increased Socrates flames count by 5
- Appeared randomly inside the perimeter of the petri dish
- Disappeared when overlapped by socrates

ExtraLifeGoodie:

I tested to ensure that the ExtraLifeGoodie

- Increased the score by 500 points
- Increased the player's lives count by 1
- Appeared randomly inside the perimeter of the petri dish
- Disappeared when overlapped with socrates

Fungus:

I tested to make sure that the Fungus

- Appeared randomly inside the perimeter of the petri dish
- Decreased the player's score by 50
- Decreased socrate's health (hit points) by 20
- Disappeared when overlapped with socrates