Wenjie Mo

Professor Smallberg

CS 32 Project 3

Feb 26 2020

Project 3 Report

Actor inheritance hierarchy:

GraphObject

Actor

Creature

Socrates

Bacteria

RegularSalmonella

AggressiveSalmonella

Ecoli

StaticObject

Dirt

Food

Projectile

Spray

Flame

Goodies

HealthGoodie

FlameGoodie

LifeGoodie

Fungus

Pit

I added a virtual destructor for every class, which does nothing.

**class** Actor: **public** GraphObject{

**public**:

Actor(StudentWorld\* sw, **int** imageID, **double** startX, **double** startY, Direction dir, **int** depth, **double** size);

Constructor: take in several parameters and pass them to GraphObject (Student world pointer will be passed for local data member)

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

Every derived class from Actor has their own doSomething function, so I decided to make it pure virtual.

**bool** isAlive() **const**;

Return the data member m\_alive (indicate whether this actor is alive)

**void** setDead();

Set m\_alive data member to false (dead)

StudentWorld\* getWorld() **const**;

Return the studentWorld pointer from private data member to indicate the StudentWorld this actor is at

**virtual** **void** receiveDamage(**int** damage);

I decide to make it virtual because, for some actors who do not have hit points (their status is either dead or alive), this function will turn them dead by using setDead() function. Otherwise, for some actors who have hit points, their own class will override this function

The following four functions return the specific feature to a type of actor in order to identify the type of the actor during the iteration. They will return true or false according to the chart. I made them all virtual since a type of identifier may be common for some class and others may override this function

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Identifier/class | Creature | Projectile | StaticObject | Pit | Goodies |
| isProjectile | N | Y | N | N | N |
| isStatic | N | N | Y | N | N |
| isCreature | Y | N | N | N | N |
| isDamagable | Y | N | N | N | Y |

**virtual** **bool** isProjectile() **const**;

**virtual** **bool** isdamageable() **const**;

**virtual** **bool** isStatic() **const**;

**virtual** **bool** isCreature() **const**;

**virtual** **int** damage() **const**;

I made this function virtual because more than one derived class have damages (Projectile and Creature(bacterias)), and they will override this function. So for other objects, this function will just return zero.

**private**:

**bool** m\_alive;

StudentWorld\* m\_world;

};

/\* No test for an abstract base class \*/

**class** Creature : **public** Actor{

**public**:

Creature(StudentWorld\* sw, **int** imageID, **double** startX, **double** startY, Direction dir, **int** health);

Constructor: take in several parameters and pass them to Actor (health will be passed for local data member). Depth and size are the same for all derived class, which will directly be passed to Actor class as numbers.

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

Every derived class from Creature has their own doSomething function, so I decided to make it pure virtual

**int** getHealth() **const**;

Return the health from m\_health data member

**void** setHealth(**int** health);

Increase the health by modifying the m\_health data member

**virtual** **void** receiveDamage(**int** damage);

Decrease the health by certain amount (Although basically the same as setHealth, but I seperate them to avoid ambiguity). I made it virtual since it override its base function

**virtual** **bool** isCreature() **const**;

Return true (they are all creatures). I made it virtual since it override its base function

**virtual** **bool** isBacteria() **const**;

I made it virtual since for Bacterias, it will return true and for Socrates. its class will override this function to return false

**virtual** **void** soundHurt() **const** = 0;  
  
Play the sound when being hurt. Every derived class from Creature has their own soundHurt function, so I decided to make it pure virtual

**virtual** **void** soundDie() **const** = 0;

Play the sound when died. Every derived class from Creature has their own soundDie function, so I decided to make it pure virtual

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

Return the points awarded. Every derived class from Creature has their own points awarded, so I decided to make it pure virtual

**private**:

**int** m\_health;

};

**class** Socrates : **public** Creature{

**public**:

Socrates(StudentWorld\* sw, **double** startX = 0, **double** startY = 128, Direction dir = 0, **int** health = 100, **int** angle = 180, **int** imageID = IID\_PLAYER);

Constructor: since the start position is always the same, so this constructor have all default argument except the StudentWorld pointer which will be passed in during the construction. All paramerters except angle will be passed to Creature class.

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

If player is already died. Return.

Get key pressed from student world

case enter

release flame if flame() count is larger than 0

decrease getFlame count

case space

release spray if Spray() count is larger than 0

decrease getSpray count

case left

move counterclockwise by setAngle(5) function

case right

move clockwise by setAngle(-5) function

if no key pressed

add spray count if getSpray() count is less than 20

return

**int** getSpray()**const**;

Return the Spray count from m\_spray data mamber

**int** getFlame()**const**;

Return the Flame count from m\_flame data member\

**void** restoreHealth();

Set the m\_health to full (100)

**void** addFlame();

Add five to m\_flame count

**int** getAngle() **constconst**;

Get the positional angle of Socrates’ position from m\_angle data member (the start position have 180 positional angle)

**void** setAngle(**int** angle);

Change the positional angle by certain degrees by modifying m\_angle data member.

Calculate the x, y position Socrates will move to use trigrometry.

move Socrates use moveTo function

Set the angle Socrates will face to.

**virtual** **void** soundHurt()**const**;

Play SOUND\_PLAYER\_HURT. I made it virtual so it could override Creature class’ function

**virtual** **void** soundDie()**const**;

Play SOUND\_PLAYER\_DIE. I made it virtual so it could override Creature class’ function

**double** sX()**const**;

Return Socrates’ X coordinate

**double** sY()**const**;

Return Socrates’ Y coordinate

**private**:

**int** m\_spray;

**int** m\_flame;

**int** m\_angle;

};

**class** Bacteria : **public** Creature{

**public**:

Bacteria(**int** startX, **int** startY, StudentWorld\* sw, **int** imageID, **int** damage, **int** health);

Constructor: since the start position will vary, so this constructor will take in the position a class object is at. All parameters except angle will be passed to Creature class except health for local data member.

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

**void** Bacteria::doSomething(){

**if** (! isAlive())

**return**;

**if** (generalStep())

specialization();

**return**;

}

I made this function virtual because it is the sign of overriding this function from base class.

This doSomething function firstly check whether the current actor is alive. The general step function is a private helper function in doSomething which denote some general step every bacteria will have during doSomething (step 1234 for Ecoli and regular Salmonella, step 12345 for aggressive Salmonella).

The general step function also contains a helper private function findSocrates72() which execute code specific for aggressive salmonella class(step 2)

Steps in findSocrates72() function:

If Socrates could be found in 72 pixels, try to move towards Socrates

if blocked by dirt or may move outside the boundary, do not move, and return false

otherwise, move towards Socrates, and return true.

Steps in generalStep() function:

If overlap with Socrates, attack with damage value

If have eaten three foods, split and clear the food count

If overlap with food, eat it.

Use the return value from ! findSocrates72() to decide whether Aggressive can do following steps // class other than aggressive Salmonella will always return false, and they will execute following codes. Specialization().

Then for the specialization, for Regular Salmonella and Aggressive Salmonella, their specializations are the same (random movement or move towards food).

If movement plan is not zero,

Try to move by current direction

If blocked by dirt or will go outside the disk

Change a random direction and reset movement plan to 10

Else

Move by moveAngle() function

Else

Find the nearest Food use findFood(x, y) function

If food can be found within 128 pixels

Set its direction to food

If blocked by dirt or will go outside the disk

Change a random direction and reset movement plan to 10

Else

Move by moveAngle() function

If food cannot be found

Choose a random direction and reset movement plan to 10

But for Ecoli, the specialization is different, so I made this function virtual and Ecoli class will override this function. (Step 5 in Ecoli).

If Socrates can be found within 256 pixels (nearly all of the disk) with oversocrates() function

Set the direction towards Socrates and try to move,

If blocked by dirt

Change direction by 10 degrees, until a direction without blocking is available

If it is still blocked after 10 tries,

stay still.

Else

Move in this direction

Else

Move in the direction of Socrates by moveAngle() function

My implementation for finding and move towards Socrates is exactly the same as the spec, I will not post pseudocode here

**virtual** **bool** isBacteria()**const**;

Return true

**virtual** **void** split(Bacteria\* b) = 0;

All bacteria will split and their split function is different according to different types of Bacteria, so I made it pure virtual

**virtual** **void** soundHurt()**const**;

It will play sound when bacteria is hurt. For Regular Salmonella and Aggressive Salmonella, their soundHurt is the same (SOUND\_SALMONELLA\_HURT). But for Ecoli, the sound is different(SOUND\_SALMONELLA\_DIE), so I made this function virtual and Ecoli class will override this function.

**virtual** **void** soundDie()**const**;

It will play sound when bacteria is hurt. For Regular Salmonella and Aggressive Salmonella, their soundHurt is the same (SOUND\_SALMONELLA\_HURT). But for Ecoli, the sound is different(SOUND\_SALMONELLA\_DIE), so I made this function virtual and Ecoli class will override this function.

**virtual** **void** addPoint();

All bacteria reward the same points(100) when being killed.

I made this function virtual because it is the sign of overriding this function from base class.

**void** splitCoordinates(Bacteria\* b, **double**& x, **double**& y);

This function will calculate the spawn coordinate for a newly splitted bacteria according to the spec’s rule and pass the coordinate back.

**void** setAngle(Bacteria\* b);

Use trigonometry to calculate and set the angle bacteria should face to trace Socrates from delta x and delta y

**private**:

**virtual** **bool** findSocrates72();

**virtual** **void** specialization();

**bool** generalStep();

**double** m\_plan;

**int** m\_damage;

**int** m\_food;

};

**class** RegularSalmonella : **public** Bacteria{

**public**:

RegularSalmonella(**int** startX, **int** startY, StudentWorld\* sw, **int** imageID = IID\_SALMONELLA, **int** damage = 1, **int** health = 4);

Constructor: the XY coordinates and the StudentWorld this bacteria is in will be passed in during construction. All other parameters are unique to this bacteria, so I kept them default argument. All parameters will be passed to Creature’s constructor.

**virtual** **void** split(Bacteria\* b);

Calculate the new split coordinate from this splitCoordinates function and generate a new Regular Salmonella then push it to StudentWorld’s list. It will then notify the SwudentWorld’s bacteriaPitCount to increase by 1.m

};

**class** AggressiveSalmonella : **public** Bacteria{

**public**:

AggressiveSalmonella(**int** startX, **int** startY, StudentWorld\* sw, **int** imageID = IID\_SALMONELLA, **int** damage = 2, **int** health = 10);

Constructor: the XY coordinates and the StudentWorld this bacteria is in will be passed in during construction. All other parameters are unique to this bacteria, so I kept them default argument. All parameters will be passed to Creature’s constructor.

**virtual** **void** split(Bacteria\* b);

Calculate the new split coordinate from this splitCoordinates function and generate a new Aggressive Salmonella then push it to StudentWorld’s list. It will then notify the SwudentWorld’s bacteriaPitCount to increase by 1.

**private**:

**virtual** **bool** findSocrates72();

};

**class** Ecoli : **public** Bacteria{

**public**:

Ecoli(**int** startX, **int** startY, StudentWorld\* sw, **int** imageID = IID\_ECOLI, **int** damage = 4, **int** health = 5);

Constructor: the XY coordinates and the StudentWorld this bacteria is in will be passed in during construction. All other parameters are unique to this bacteria, so I kept them default argument. All parameters will be passed to Creature’s constructor.

**virtual** **void** split(Bacteria\* b);

Calculate the new split coordinate from this splitCoordinates function and generate a new Ecoli then push it to StudentWorld’s list. It will then notify the SwudentWorld’s bacteriaPitCount to increase by 1.

**virtual** **void** soundHurt()**const**;

Ecoli has its own soundHurt to be played(SOUND\_ECOLI\_HURT). I made this function virtual because it is the sign of overriding this function from base class.

**virtual** **void** soundDie()**const**;

Ecoli has its own soundHurt to be played(SOUND\_ECOLI\_DIE). I made this function virtual because it is the sign of overriding this function from base class.

**private**:

**virtual** **void** specialization();

};

**class** StaticObject : **public** Actor{

**public**:

StaticObject(**int** startX, **int** startY, StudentWorld\* sw, **int** imageID, **int** angle);

Constructor: take in several parameters and pass them to base class’s constructor. Depth and size are the same for all derived class, which will directly be passed to Actor class as numbers.

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

Dirt and food will do nothing. I made this function virtual because it is the sign of overriding this function from base class.

**virtual** **bool** isStatic()**const**;

This function will return true for food and dirt. I made this function virtual because it is the sign of overriding this function from base class.

};

**class** Dirt : **public** StaticObject{

**public**:

Dirt(**int** startX, **int** startY, StudentWorld\* sw, **int** imageID = IID\_DIRT, **int** angle = 0);

Constructor: take in several parameters during construction and pass them to StaticObject constructor. The imageID and angle are unique and same for the Dirt class, so they are passed by default argument.

};

**class** Food : **public** StaticObject{

**public**:

Food(**int** startX, **int** startY, StudentWorld\* sw, **int** imageID = IID\_FOOD, **int** angle = 90);

Constructor: take in several parameters during construction and pass them to StaticObject constructor. The imageID and angle are unique and same for the Food class, so they are passed by default argument.

**virtual** **bool** isdamageable()**const**;

Food is not damageable. So this function will return false. I made this function virtual because it is the sign of overriding this function from base class.

};

**class** Projectile : **public** Actor{

**public**:

Projectile (**int** startX, **int** startY, **int** angle, **int** imageID, StudentWorld\* sw, **int** distance);

Constructor: take in several parameters during construction and pass them to Actor constructor. distance is the maximum travel distance each derived class will have. Depth and size are the same for all derived class, which will directly be passed to Actor class as numbers.

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

If the projectile is not alive, return

If this projectile is overlap with a damageable object, give amount of damage from damage () function and set itself to dead and return

Travel several pixels (from maxDistance) by moveAngle() function, and decrease m\_distanceLeft by this amount

If the distance travelled has already exceed the m\_distanceLeft, set itself to dead.

I made this function virtual because it is the sign of overriding this function from base class.

**virtual** **bool** isProjectile()**const**;

Projectiles are projectile, so this function will return true. I made this function virtual because it is the sign of overriding this function from base class.

**virtual** **bool** isdamageable()**const**;

Projectiles are projectile, so this function will return true. I made this function virtual because it is the sign of overriding this function from base class.

**virtual** **int** damage() = 0;

Return damage a projectile could cause. I made it pure virtual since each different type of projectile has different value of damage.

**private**:

**int** m\_distanceLeft;

**int** m\_angle;

**virtual** **int** maxDistance();

};

**class** Flame : **public** Projectile{

**public**:

Flame (**int** startX, **int** startY, **int** angle, StudentWorld\* sw, **int** distance = 32, **int** imageID = IID\_FLAME);

Constructor: take in several parameters during construction and pass them to Projectile class constructor. The imageID and max travel distance are unique and same for the Flame class, so they are passed by default argument.

**virtual** **int** damage();

Return damage a Flame could cause. I made this function virtual because it is the sign of overriding this function from base class.

**private**:

};

**class** Spray : **public** Projectile{

**public**:

Spray (**int** startX, **int** startY, **int** angle, StudentWorld\* sw, **int** distance = 112, **int** imageID = IID\_SPRAY);

Constructor: take in several parameters during construction and pass them to Projectile class constructor. The imageID and max travel distance are unique and same for the Spray class, so they are passed by default argument.

**virtual** **int** damage()**const**;

Return damage a Spray could cause. I made this function virtual because it is the sign of overriding this function from base class.

**private**:

};

**class** Goodie : **public** Actor{

**public**:

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

If the Goodie is not alive, return.

if Socrates overlap with this (StudentWorld’s oversocrates() function)

Do special ability from specialization() function. (For goodies, add health/add life, or add flame count and then play the sound of being picked up. For fungus, let Socrates receive the damage by studentWorld’s Socratesreceivedamage() function)

Set itself to dead (setDead()) and return

If no overlap, decrease lifetime by 1.

If the lifetime is lower or equal to zero, setDead().

I made this function virtual because it is the sign of overriding this function from base class.

Goodie (**int** startX, **int** startY, **int** life, StudentWorld\* sw, **int** imageID);

Constructor: take in several parameters during construction and pass them to Actor class constructor. life will be set for m\_life data member to indicate the lifetime. Depth and size are the same for all derived class, which will directly be passed to Actor class as numbers.

**virtual** **int** returnPoint() **const** = 0;

Return the point rewarded when being picked up. Every derived class from Goodie has their own points awarded, so I decided to make it pure virtual

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

Do the specialization during doSomething. Every derived class from Goodie has their own specialization, so I decided to make it pure virtual.

**private**:

**int** m\_lifetime;

**int** getLifetime();

**void** setLifetime();

};

**class** HealthGoodie : **public** Goodie{

**public**:

HealthGoodie (**int** startX, **int** startY, **int** life, StudentWorld\* sw, **int** imageID = IID\_RESTORE\_HEALTH\_GOODIE)

);

Constructor: take in several parameters during construction and pass them to Goodie class constructor. The image ID is unique and same for the HealthGoodie class, so it is passed by default argument.

**virtual** **int** returnPoint()**const**;

Return the point rewarded when being picked up (250). I made this function virtual because it is the sign of overriding this function from base class.

**private**:

**virtual** **void** specialization();

};

**class** FlameGoodie : **public** Goodie{

**public**:

FlameGoodie (**int** startX, **int** startY, **int** life, StudentWorld\* sw, **int** imageID = IID\_FLAME\_THROWER\_GOODIE)

Constructor: take in several parameters during construction and pass them to Goodie class constructor. The image ID is unique and same for the FlameGoodie class, so it is passed by default argument.

);

**virtual** **int** returnPoint()**const**;

Return the point rewarded when being picked up (300). I made this function virtual because it is the sign of overriding this function from base class.

**private**:

**virtual** **void** specialization();

};

**class** LifeGoodie : **public** Goodie{

**public**:

LifeGoodie (**int** startX, **int** startY, **int** life, StudentWorld\* sw, **int** imageID = IID\_EXTRA\_LIFE\_GOODIE)

);

Constructor: take in several parameters during construction and pass them to Goodie class constructor. The image ID is unique and same for the Life Goodie class, so it is passed by default argument.

**virtual** **int** returnPoint()**const**;

Return the point rewarded when being picked up (500). I made this function virtual because it is the sign of overriding this function from base class.

**private**:

**virtual** **void** specialization();

};

**class** Fungus : **public** Goodie{

**public**:

Fungus (**int** startX, **int** startY, **int** life, StudentWorld\* sw, **int** imageID = IID\_FUNGUS);

Constructor: take in several parameters during construction and pass them to Goodie class constructor. The image ID is unique and same for the Fungus class, so it is passed by default argument.

**virtual** **int** returnPoint()**const**;

Return the point rewarded when being picked up (-50). I made this function virtual because it is the

sign of overriding this function from base class.

**private**:

**virtual** **void** specialization();

};

**class** Pit : **public** Actor{

**public**:

Pit (**int** startX, **int** startY, StudentWorld\* sw);

Constructor: take in several parameters during construction and pass them to Actor constructor. Depth, size, and imageID are the same for this class, which will directly be passed to Actor class as numbers/IDs.

Initialize bacteria count

m\_regular = 5

m\_aggressive = 3

m\_coli = 2

m\_empty is false

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

If the Pit is not alive, return.

If the Pit is empty, call StudentWorld’s bacteriaPitCount() function to decrease the count by 1. Return.

generate a chance a bacteria will be released

If the chance is met

Continuously loop through three types of bacteria until a bacterium is released

Generate a chance a certain type of bacteria will be released is not zero

If the chance is met and the bacteria count is not zero, generate a new bacterium at Pit’s position

decrease count of this type of bacteria

play the SOUND\_BACTERIUM\_BORN sound

if all three bacteria count is zero, change the m\_empty to true

I made this function virtual because it is the sign of overriding this function from base class.

**virtual** **bool** isdamageable()**const**;

It is not damageable, return false

**private**:

**int** m\_regular;

**int** m\_aggressive;

**int** m\_coli;

**bool** m\_empty;

**bool** isEmpty()**const**;

};

#endif // ACTOR\_H\_

**class** StudentWorld : **public** GameWorld

{

**public**:

StudentWorld(std::string assetPath);

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

This function is used for initializing the whole studentWorld. It is virtual because it shows it is a overridden function from its base gameWorld class.

Steps in init

1. create new Socrates into Gameworld
2. set m\_bacteriaPitCount to 0 (the count for number of pits and bacteria)
3. generate the number of pit will be formed, loop this number times:
   1. generate x, y coordinates for each pit (randomCoord(double& x, double& y)),
   2. if the coordinates is not shown overlapped with StaticObject members (Food or Pit) in the list (overlap(double x, double y)), (if overlap, generate another set of coordinates)
      1. Construct the pit and push it into the list.
      2. Increase the BacteriaPitCount.
4. generate the number of food will be formed, loop this number times:
   1. generate x, y coordinates for each food (randomCoord(double& x, double& y)),
   2. if the coordinates is not shown overlapped with StaticObject members (Food or Pit) in the list by overlap(double x, double y), (if overlap, generate another set of coordinates)
      1. Construct the Food and push it into the list.
5. generate the number of dirt will be formed, loop this number times:
   1. generate x, y coordinates for each dirt (randomCoord(double& x, double& y)),
   2. if the coordinates is not shown overlapped with StaticObject members (Food or Pit) in the list by overlap(double x, double y), (if overlap, generate another set of coordinates)
      1. Construct the dirt and push it into the list.
6. Continue game

The coordinates are calculated by random number in radius and angle and then use polar coordinates to calculate the x, y coordinates

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

This function is used to give every game actor a chance to move during a tick. It is virtual because it shows it is a overridden function from its base gameWorld class.

steps in move

1. Ask Socrates to doSometing()
2. setup a iterator to iterate every actor in the list until the end
   1. Give the actor this iterator pointed to a chance to doSomething()
   2. If the player is dead, decrease his life by decLives() and return GWSTATUS\_PLAYER\_DIED
   3. if the m\_BacteriaPitCount is zero, play the sound and return GWSTATUS\_FINISHED\_LEVEL.
3. if the actor this iterator pointed to is dead, remove it from the list. (deleteActor() function)
4. Add goodies by addGoodies() function, which calculate the goodie spawn chance (and then each specific type of chances) fungus spawn chance, if the chance is met then it will calculate a random coordinate around the edge of circle and add the object into the Gameworld and list
5. Setup the stat text by setGameStatText(combine()). combine() is a private function to use sstream to add all game stats together and pass it to setGameStatText() function
6. Continue game

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

This function will delete every actor in the list and the Socrates when being called. It is virtual because it shows it is a overridden function from its base gameWorld class.

steps:

1. setup an iterator to iterate every actor in the list until the end
   1. delete actor this iterator pointed to
2. delete Socrates and set its pointer to nullptr

**virtual** ~StudentWorld();

Destructor: Simply call claenUp(). It is virtual because it shows it is a sign of overridden function from its base gameWorld class (virtual destructor).

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

This function is used during the init to calculate whether the passed in coordinates are overlapped with current StaticObject's coordinates

**bool** oversocrates(Actor\* a, **int** distance);

This function is used to determine the overlap between Socrates and a actor, and also used in Aggressive Salmonella and Ecoli class to find Socrates within 72 or 256 pixels.

**bool** overlap(Actor\* b, **int** index);

This function is used to determine overlap in other circumstances. The parameter index is used for the following switch statement

1. Food with bacteria
2. Projectile with bacteria/goodies/dirt
3. Bacteria with boundaries
4. Bacteria with dirt

It will set up a iterator to iterate every actor in the list until the end. The iterator will try to find the right hand side objects by combination of isDamagable, isStatic, isCreature, and isProjectile functions described in actor class.

**void** findFood(**double** &x, **double** &y);

This function is used to find the nearest food in the Gameworld. It will set up a iterator to iterate every actor in the list until the end. The iterator will try to find food objects by combination of ! isDamagable(), isStatic() functions described in actor class. If the food is found, record the distance and compare or update it with the temporary shortest distance. This function return the x, y coordinates of the shortest distance.

**void** receiveGoodies(**int** index);

It is called by Goodie Class during the doSomething(). The index is used for switch statement for specific behavior (add health, add life, add flames)

**void** socratesReceiveDamage (**int** damage);

This function is used during Bacteria’s doSomething. When Socrates is damaged, this function will tell how much damage a Socrates will receive.

**double** sX()**const**;

It will return Socrates’ X coordinates by retrieving from Socrates pointer

**double** sY()**const**;

It will return Socrates’ Y coordinates by retrieving from Socrates pointer

**void** addActor(Actor\* a);

It will push a Actor into list (mainly used during split of bacteria)

**void** BacteriaPitCount(**int** i);

It will change the m\_BacteriaPitCount by i amount

**void** addFlame();

It will construct 16 pieces of flame (First one will be spawned 2\*SPRITE\_RADIUS in front of Socrates and following flames will be 22 degrees more) into list of actors and play SOUND\_PLAYER\_FIRE sound.

**void** addSpray();

It will construct a Spray object 2\*SPRITE\_RADIUS in front of Socrates and add it into list and play SOUND\_PLAYER\_SPRAY sound.

**private**:

Socrates\* m\_s;

std::list<Actor\*> m\_a;

**int** m\_BacteriaPitCount;

std::string combine()**const**;

**int** isEmpty()**const**;

**void** addGoodies();

**void** deleteActors();

};

**Conclusion:**

I have finished all of functionality specified by the spec and I basically followed the spec and found my game behave pretty similar to sample.

There are several points that I followed the spec but still found the behavior slightly different to the sample program.

The sample game will still show the picture of spray and flame just as they have met the movement maximum, but I think the spec said that as long as the maximum is met, we should set the Projectile to dead. So, I think the Flame and Spray object should not be displayed at the last tick movement. The result of this confusion is that, my spray may never be able to hit the center of disk and some of dirt may not be able to be cleaned.

Then, I found that in the sample, when Bacteria overlaps with a goodie, the goodie will immediately die. I did not find this part in spec and I just follow the spec’s instruction to not deal with it.

The spec did not mention a lot about the game status stat bar, so I mainly followed the instruction given by the sample game based on spec’s information.

When my Ecoli are overlapping with Socrates, it will stay at the top of Socrates, which is opposite than spec. I think it is the depth issue, and I choose to follow the spec.

Thank you for looking at my long report. Have a nice day : )

**Test:**

**Socrates class:**

I test this class by press different keys when Socrates is alive. If I press left, Socrates should move counterclockwise 5 degrees while still face the center of the circle. If I press right, Socrates should move clockwise 5 degrees while still face the center of the circle. If I press enter, I should see flames pop out and the flame count from display should decrease by 1. If I press space, I should see Spray pop out and the Spray count from display should decrease by 1. If I did not press any key, the Spray count should automatically change to 20.

For health, I only introduce the Ecoli into the gameWorld and see whether the health will decrease by certain amount (Health stat -1) every tick and Play SOUND\_PLAYER\_HURT sound (I use F to lock the frame to test the performance every tick). Then, if the health goes below 0, the game should immediate stop and continue next round (life stat -1) or stop the game (if life stat will reach 0). After this, the sound SOUND\_PLAYER\_DIE will be played.

The process of picking up goodies will be discussed in Goodie class.

**Bacteria:**

There are still some general test for all types of Bacteria

General testing guideline:

For all Bacteria’s movement, it should not go beyond the central disk, move onto/under dirt. Unless being attacked by projectiles, it should never suddenly disappear from the disk.

Initialization of Bacteria:

Bacteria should start with right amount of hit points (4 for Regular, 10 for Aggressive, 5 for Ecoli), food count (0), and movement plan (0). I used breakpoints to check these details. When it is constructed, the sound SOUND\_BACTERIUM\_BORN should be played.

Test for attack:

If a Bacteria overlaps with Socrates (I used keyPress F and breakpoints to check the overlap distance), it should give Socrates damage (1 for Regular Salmonella, 2 for Aggressive Salmonella, 4 for Ecoli).

Test for being attack:

I used keyPress F and breakpoints to check the overlap distance, and the decrement of Salmonella’s hitpoint by certain amount when overlapping with Projectiles (hitpoint -2 for spray and -5 for flame). the sound SOUND\_SALMONELLA\_HURT should be played. A salmonella should never be attacked by the same Project object twice (will be discussed in Projectile’s class). If the hitpoints go below 0, the m\_alive member should be false, and it should disappear from the GameWorld. Then the sound SOUND\_SALMONELLA\_DIE should be played. The score stat should increase by 100 (I tested it by killing several types of Bacteria)

**Regular Salmonella Class:**

Test of find food:

Create a gameWorld only with 40 foods and one regular Salmonella, after it has borned, it should constantly move towards the nearest foods (there are many foods in the disk, so very high probability of being detected by salmonellas). Whenever a Salmonella has eaten three foods, it will split, and I used breakpoints to confirm the right split coordinates. Then, I will create some foods around the edge of the disk and put the pit at the center of the disk. Initially, Salmonellas should not go directly for foods (beyond their detection range)

Test for movement:

Create a Gameworld with no dirt and one Salmonella, it should move around randomly and I used keyPress F to check it does change the moving direction after 10 ticks.

Create a Gameworld with regular amount of dirt and one Salmonella, it should move around randomly and I used keyPress F to check changing of direction after 10 ticks. Then, whenever the Salmonella has met the dirt, it changed direction. I used keyPress F and breakpoints to check the overlap distance. A Regular Salmonella should not move towards Socrates (unless instructed by random movement). All of movement should follow the General testing guideline.

Tests for initialization, attack and being attack have been posted in Bacteria Class.

**Aggressive Salmonella:**

Test of find food: Same as Regular Salmonella.

Test for movement:

The Aggressive Salmonella’s primary goal should be finding and moving towards Socrates within 72 pixels. I put the pit near Socrates and only let it release one Aggressive Salmonella, it should move towards Socrates even if food is closer to it. Then after it has overlapped with Socrates, it should go under Socrates when Socrates stands still and follow Socrates when Socrates tries to move (there is a high possibility that Salmonella could overlap wit Socrates again). I used keyPress F to check these behaviors. If it is far away from Socrates, it should behave like a Regular Salmonella.

**Ecoli Class:**

Test of find food:

It should not move towards food (unless move under random direction). But it could still eat (overlap) food and split when it has eaten three foods which just behaves like other types of bacteria.

Test for movement:

Ecoli could always find the position of Socrates and move towards him. I tested this property by create a GameWorld without Dirts, so the movement is clear. If the movement of Ecoli is blocked (overlapped) by Dirt, it should change direction ten degrees ten times until it has found a new direction without blocking of the Dirt. After ten tries, it will stop finding new direction and stop moving until the next tick. I could validate this property by use keyPress F and break points. Then after it has overlapped with Socrates, it should go under Socrates when Socrates stands still and follow Socrates when Socrates tries to move (it may not overlap with Socrates again, but it will still face towards Socrates).

**Static Object class:**

General properties of derived class of this base class: no moving

**Dirt class:**

Initialization:

Dirt should randomly appear in the disk and could overlap with each other. Dirt is the last object constructed in StudentWorld’s init() function, it should not overlap with pit or foods.

It should be generated in the disk and never appear in the moving sub circle of Socrates. It could be destroyed by projectiles (let Socrates attack it). Dirt does not have hitpoint and should immediately. It should blocked the movement of Bacteria (let Bacteria randomly move and see the behavior).

**Food Class:**

Initialization:

Food is the second last object in the StudentWorld’s init() function. The number of foods constructed should follow the max function and it should never overlap with each other or with pits.

Food should not be damaged by Projectiles (let Socrates attack it). When food is being eaten (overlap), it’s m\_alive should be set as false and it should disappear from the StudentWorld. I could validate this property by use keyPress F and break points.

**Projectile Class:**

There are still some common tests for Projectile Classes derived class.

Initialization: all Projectile objects should only be created SPRITE\_WIDTH from Socrates instead of firing directly from body of Socrates and its m\_alive status should be true when being created. I used breakpoints to check the status and position of Projectile objectives upon the creation.

Attack: if they overlap with a isDamagable() object (Bacteria, dirt, goodies) it should give certain amount of damage (Spray 2, Flame 5), which should be effective on these damageable objects (I will talk about being attack in specific classes). Then, a Projectile’s m\_alive status should be false.

Movement: they will move SPRITE\_WIDTH distance each tick without changing direction. After they have moved their maximum distance (32 for Flame and 112 for Spray) without overlapping with damageable objects, their m\_alive status should be false, and they should disappear from GameWorld. I used keyPress F to check the movement trail and the maximum movement distance for Projectiles and breakpoints for behaviors after exceeding maximum movement distances.

**Flame Class:**

Creation: Flames are always created by a group of 16 (I used keyPress F to check this property).

Other properties are the same as description in Projectile class.

**Spray Class:**

Properties are the same as description in Projectile class.

**Goodie Class:**

There are still some general properties for this class:

Creation:

The creation of fungus and Goodies are controlled by two separate probabilities, then the specific type of a Goodie being created is controlled by another probability. As a result, a goodie could be created accompanied by a fungus within one tick, but no two types of Goodies could be created with in one tick.

Every object in the Goodie class should be created on the circle of movement of Socrates and nowhere else.

I checked above properties by creating an empty GameWorld with only goodies could be created and used keyPress F to check for each tick.

Movement:

Every object in the Goodie should not move. But they should disappear by certain amount of ticks (m\_lifetime). At this time, their m\_alive status should be false. I checked by created four kinds of Goodie objects with constant lifetime to see whether they will die after the lifetime turned to zero.

Being attack:

They do not have hitpoints, so whenever they overlap with Projectiles (Flame and Spray), they should immediately die (m\_alive status should be false) and disappear from the Gameworld. I checked this property by firing a Flame around Goodies and immediately press F to check the attacking scene.

**Health Goodie Class:**

It should recover player’s health to full, play sound SOUND\_GOT\_GOODIE, and add Socrates’ score by 500 when being eaten. (I checked by setting breakpoint on Socrates to check the m\_life status or check the game stat bar). After being eaten by Socrates, they should immediately die (m\_alive status should be false) and disappear from the Gameworld.

Other properties should follow testing of Goodie base class.

**Flame Goodie Class:**

It should add player’s Flame by 5, play sound SOUND\_GOT\_GOODIE, and add Socrates’ score by 500 when being eaten. (I checked by setting breakpoint on Socrates to check the m\_flame status or check the game stat bar). After being eaten by Socrates, they should immediately die (m\_alive status should be false) and disappear from the Gameworld.

Other properties should follow testing of Goodie base class.

**Spray Class:**

It should add player’s Life by 1, play sound SOUND\_GOT\_GOODIE, and add Socrates’ score by 500 when being eaten. (I checked by checking the game stat bar). After being eaten by Socrates, they should immediately die (m\_alive status should be false) and disappear from the Gameworld.

Other properties should follow testing of Goodie base class.

**Fungus Class:**

It should decrease player’s hit point by 20, decrease Socrates’ score by 50 when being eaten. (I checked by setting breakpoint on Socrates to check the m\_health status, or check the game stat bar). After being eaten by Socrates, they should immediately die (m\_alive status should be false) and disappear from the Gameworld.

Other properties should follow testing of Goodie base class.

**Pit class:**

Creation:

This class of object should be created during StudentWorld’s init() process, so when I entered the game, the pit should already be placed in the disk. Pits should appear in the disk without overlapping with other pits, foods, or dirt and it should not appear in the circle the Socrates is moving. I checked this property by add a lot of pits into disk to see whether there were any exceptions. Pit should not be damaged by Projectiles. I checked this property by letting Spray and Flame attack Pits.

Movement:

A Pit should never move and has chance to release a Bacteria each tick (The test for Bacteria creation is posted in Bacteria part). Only after it has released all of its Bacteria should it immediately die (m\_alive status should be false) and disappear from the Gameworld. I checked this property by setting up breakpoints to check the Bacteria counts upon death of Pits or counting Bacteria in the GameWorld.

**Student World Class:**

Initialization:

The StudentWorld should allocate a Socrates, right amount of foods, pits and dirt (specified in each class) in the disk and begin the game. I checked this property by changing the amount of Objects upon creation and see whether StudentWorld could follow changes.

Move:

Each tick, StudentWorld should ask every alive actor to do something, let Socrates to do something. I checked this property by setting up breakpoints and check whether the list of Actors has been properly iterated. Then, the game status stat should also be updated, the specific content of changes has been specified in each class. I checked the property by checking whether the status bar is following the following rules:

Score change: Socrates to eat a Goodie, kill a Bacteria or, overlap with a fungus. (See corresponding class for details)

Health change: Socrates is overlapping with/attacking by a Bacteria or eats a fungus. (See corresponding class for details)

Life change: Socrates has a health lower than 0, after another round begins (-1). Socrates has eaten/overlapped with a life Goodie (+1)

Level change: Socrates has finished the current level, after another round begins (+1).

Flame change: Socrates has fired a flame (-1). Socrates has eaten/overlapped a flameGoodie (+5).

Spray change: Socrates has fired a Spray (-1). There is no key press received at this moment (+1).

Four status after moving:

If the Socrates is still alive -> next tick.

If all bacteria have been removed -> next level (clear the current level, display the text for next level, and prepare for another higher-level round).

If Socrates’ health has gone below 0 -> Socrates died (clear the current level, display the text for Socrates is died, and prepare for the same level round).

If Socrates’ health had gone below 0 and he has 0 live now. Display the message “You lose” and exit the game.

I checked these properties by playing the game.

Clearing up:

All current level game Objects should be removed and no memory leak or access of null pointer or delete an already deleted object should appear. (XCode will warn this)