Daxuan Shu

204853061

CS 32

Project 3 Report

Report

The Actor class design that I use is the same as the one post on the website:

GraphObject

Actor

Pebble

EnergyHolder

Food

AntHill

Pheromone

TriggerableActor

WaterPool

Poison

Insect

Ant

Grasshopper

BabyGrasshopper

AdultGrasshopper

class Actor derived from Graphobject

Member Functions:

1.

Constructor Actor(StudentWorld\* world, int startX, int startY, Direction startDir, int imageID, int depth);

Since Actor is a derived class from GraphObject, we must use initializer list to pass in the parameters to the GraphObject’s constructor like this:

Actor::Actor(StudentWorld\* world, int startX, int startY, Direction startDir, int imageID, int depth):GraphObject(imageID, startX, startY, startDir, depth)

inside the implement of the Actor’s C’tor, I initialize all the private variables of Actor class.

The private variables that I used In Actor class are :

int m\_hit\_points;

        bool Alive;

        bool isPoisoned

        bool isStunned;

        int m\_colony;

        StudentWorld\* m\_world;

2.

Pure Virtual function virtual void doSomething() = 0;

Since every Actors in student World will do Something during a tick.Thus, I use the pure virtual function to implement and I must redefine this function in every derived class of Actor.

3.

virtual bool function isDead() const

return if the actor is alive. Since this function does not change any private variable in our function. we use const keyworld to implement it.

4.virtual bool blocksMovement() const

return if the actor is pebble since pebble will block the insects to keep moving in current direction.

5.Virtual void setDead();

Change the Alive private variable to false in order to be delete In the move()function in studentWorld class.

6. Virtual void getPoisoned()

return the Actor’s private variable isPoisoned. Since the Insect derived class will redefine this function , we use virtual.

7. Virtual void getStunned()

return the Actor’s private variable isStunned. Since the Insect derived class will redefine this function, we ues virtual.

8. virtual bool isEdible() const

return true if the Actor can be eaten. For example, we use this function to search the food in current square in Insect class. Since we will reuse this function in our derived class Insect, we use virtual before the function. Since it does not change any variable in our Actor class, I use const keyworld.

9. virtual bool isPheromone() const

return true if the Actor is Pheromone. I use virtual because I will redefine the function in Pheromone class. Since it does not change any variable in our Actor class, I use const keyworld.

10.virtual bool isEnemy(int colony) const

return true if the Actor is an enemy of colony. I use virtual since I will reuse this function in derived class ant.

11. virtual bool isDangerous(int colony) const

return true if the Actor is dangerous to colony. Since I will redefine this function in the derived class Insect, I use virtual. Since it does not change any variable in our Actor class, I use const keyworld.

12. virtual bool isAnthill(int colony) const

return true if the Actor is an anthill of the indicated colony. Since I will redefine this function in the derived class Ant, I use virtual. Since it does not change any variable in our Actor class, I use const keyworld.

13. Static Ditrction random\_direction()

return a random direction from(up, down, left and right). I use static because it allow us to call this function without from an Actor object. It is convenient to call this function in the parameter of the derived class’s constructor.

14 Virtual bool isCC()

return true if the actor has a debuff like stun and sleeping I use virtual since I will redefine this function in the derived class Insect.

15. Virtual int getEnergy() const

return m\_hit\_points, use virtual since redefine in derived class TriggerableActor like water pool or Pheromone.

16. Virtual void getbitten(int amt)

adjust the private member variable m-hit\_points to the amt; like m\_hit\_poins += amt.

I use Virtual since I will use this function in the derived class Insect and Energyholder.

**Class Pebble derived from class Actor**

**Member functions:**

the pebble class have all the public member functions in class actor.

1. I define the pure virtual function doSomething() in actor class by simply return since the pebble won’t do anything during a tick.

2. Virtual bool blocksmovement() const

redefine this function to return true since the pebble will block any insects attempted to across it.

3. Constructor: simply pass in the parameter into the base class Actor using the initialize lists.

**class Energyholder derived from class Actor**

**Member functions:**

1. Constructor : simply pass in the parameter into the base class Actor using the initialize lists.

2. Virtual bool isDead() const

virtual since redefine this function by checking m\_hit\_points to see if it is less than 0.

If less than or equal 0, return true, otherwise, return false.

3. Virtual int getEnergy() cosnt

reuse the function for the EnergyHolder derived class like food.

4. void updateEnergy(int amt)

I did not use virtual since the way for all EnergyHolder derived class to adjust their energy is the same.

5. void addFood(int amt)

I did not use virtual since the way for all EnergyHolder derived class to hold the food is the same

6. void pickupFood (int amt)

I did not use virtual since the way for all EnergyHolder derived class to pick up food is the same

7. void pickupAndEatFood(int amt)

I did not use virtual since the way for all EnergyHolder derived class to pick up and eat food is the same

8. Virtual bool becomeFoodUponDeath()

I used virtual since I would redefine this function in the derived class like class Insect

9 Virtual void getBitten(int amt)

I used virtual since I would redefine this function in the derived class like class ant ot class grssshooper since they have different reaction for getting bitten.

**class Food derived from EnergyHolder**

**Memberfunctions**

1. constructor

2. virtual void doSomething()

specialize the reaction for the food during a tick

3. Virtual bool isEdible()

I use virtual since I want to override the function to return true.

**class Anthill derived from EnergyHolder**

**Memberfunctions**

1. constructor

2. virtual void doSomething()

specialize the reaction for the Anthill during a tick

3. virtual isMyhill (int colony) const

redefine the function to specific insect

**class Pheromone derived from EnergyHolder**

**Member functions**

1. constructor

2. Virtual void do something()

tell a Pheromone object to do something during a tick

3.Virtual bool isPheromone(int colony) const

redefine , return true, so virtual.

4. void increaseStrenth()

I did not use virtual since this character is only for Pheromone, like += 256 and max 768.

**class TriggerrableActor derived from Actor**

**Member functions**

1. constructor

2. virtual bool isDangerous (int colony)

I use virtual since I redefine the function from base class Actor to return true

**class WaterPool derived from TriggerrableActor**

**Member functions**

1. constructor

2. virtual void doSomething()

to tell water pool to do something during a tick

**class Poison derived from TriggerrableActor**

**Member functions**

1. constructor

2. virtual void doSomething()

to tell water poison actor to do something during a tick

**class Insect derived form EnergyHolder**

**Member funtions:**

1. constructor

2.virtual void doSomething()

doSomething during a tick

3. virtual void getBitten(int amt)

I use virtual to redefine the function from base class actor

4. virtual void getPoisoned();

I use virtual to make the actor to do the correct thing when getting poisoned. For example, the Adult grasshopper is immune to poison.

5.virtual void getStunned()

I use virtual since the implement to change the status of stunned is different for the insects. thus I need to override this function in the derived class.

6 virtual bool isEnemy(int colony)

I use virtual to redefine function from base class Actor.

7. virtual bool becomesFoodUponDeath() const

Since all the insects will become food upon death, redefine this function from our base class to return true

8. void getXYInFrontofMe(int &x, int&y) const

to get the coordinate of all the insects’s front square are all the same. Thus, we do not need virtual in this function.

9.virtual bool moveForwardIfPossible();

Since the adult grasshopper can jump across pebble, I use virtual to redefine this function.

10 void increaseSleepTicks(int amt)

Since all the Insects’ adding sleep ticks are in the same way by increasing the stun\_index private member variable, I did not use virtual

11. void setSleepTicks(int amt)

Since all the Insects’ setting sleep ticks are in the same way by setting the stun\_index private member variable, I did not use virtual

private member variable stun\_index

**class Ant derived from insect**

**Member funtions:**

1. constructor

2. virtual void doSomething()

tell an Ant do something during a tick

3.virtual void getBitten(int amt)

Specialize the function from base class

4. virtual bool is Enemy(int colony) const

to redefine this function base on the colony

5.virtual bool moveForwardIfPossible()

to specialize the function from base class Insect

**class Grasshopper derived from insect**

**Member functions:**

1. constructor

2.virtual void doSomething()

tell an grasshopper to do something during a tick

3. int getDistance()

Since both baby or adult grasshopper has the same random distance from 2 to 10, I did not use virtual

**class BabyGrasshopper derived from class Grasshopper**

**Member functions**

1. constructor

2.virtual bool isEnemy(int colony) const

redefine the function from our base class

**class AdultGrasshopper derived from class Grasshopper**

**Member functions:**

1. constructor

2. virtual void getBitten(int amt)

Since adult grasshopper might bit back, so use virtual to redefine function.