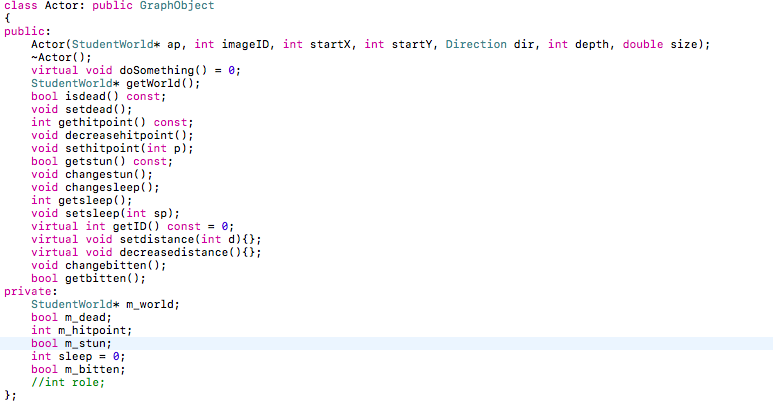
**Report for project 3**

**Weining Zou**

**Member function discussion:**

1. “Actor.h” & “Actor.cpp”

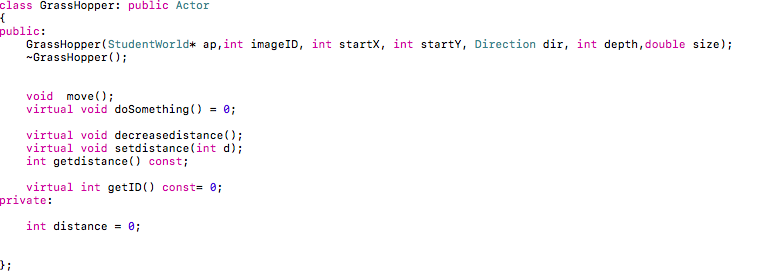
Class Actor:



In the Actor base class, I have 5 private members. The m\_world is to make sure class Student World can reach Actor class and each of its derived class by passing its own pointer. m\_dead is used to decide each members state, in order to let decide which member need to be cleared up after a tick. m\_hitpoint means different in different derived classes. In classes of insects it represent hit point, in Food class it means units of food and for Pheromone it means unit of pheromone. For Poison , Water and Pebble, the m\_hitpoint is useless, so they all hold 0 is the default number. m\_stun is for insects to decide if they are stun or not. The sleep is just for Grasshopper but I need to use it in a stun() function in StudentWorld class, which is passing Actor\* as an argument. The m\_bitten is checking a insects been bitten or not at its current square. The most public members are get and set things, which are used to make changes to its private members. The getWorld() class can let Actor and its derived class communicate with StudentWorld class. There are four virtual functions. dosomething() is the main function for every derived class to let their object to do something. An Actor itself cannot so something, so it is set to be a pure virtual function. getID() is used to distinguish each objects. An Actor will have no object so it is also a pure virtual function. The setdistance(int d) and decreasedistance are only used in Grasshopper class. So, I won’t set them as pure virtual function that will force every other derived class to implement them. The constructor of Actor will used by its derived class to define their initial location on the field and image id , the depth, size and direction.

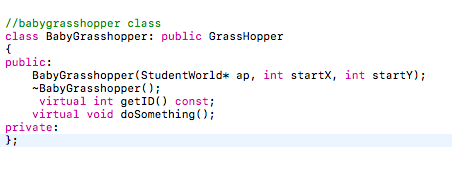
Class Grasshopper:

It is a derived class for Actor but base class for Adultgrasshopper Class and Babygrasshopper Class.



Just one private member called distance letting grasshoppers to remember how far did it go in a direction. The decreasedistance() and setdistance(int d) are actually implemented here for decreasing one step each time or assign a new distance to go. The getdistance() are not used in StudentWorld class so that I can put it here instead of in Actor class. The move() function will set a new direction and distance for Grasshopper to go for just one move step forward by using moveOneStep() function in StudyWorld class. It stays pure virtual for doSomething() and getID().

BabyGrasshopper:



It is the derive class for Grasshopper class. I put doSomething() in private because no other class will use them. At the end of the tick, the dead one will be deleted. getID() will sign a number to babygrasshopper as its ID which will be used in StudentWorld class and so as to every other derived classes.

dosomething():

Decrease hitpoint;

Check dead;

If dead add new food at the some spot by calling addnewfood(int x, in y, int unit);

Check sleep;

If not sleep

If hitpoint greater than 1600, than call turn()

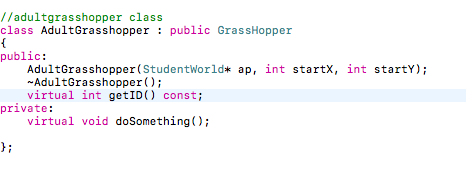
Call eat(Actor\* ap) function to eat

If have some food and eat

50% chance to move a step

change sleep state;

AdultGrasshopper:

Class AdultGrasshopper

The AdultGrasshopper class is most the same as the BabyGrasshopper.

dosomething():

Decrease hitpoint;

Check dead;

If dead add new food at the some spot by calling addnewfood(int x, in y, int unit);

Check sleep;

If not sleep

33% chance to bite

if bite

call bite(Actor\* ap) to bite

10% to jump

if jump

call jump(Actor\* ap) to jump

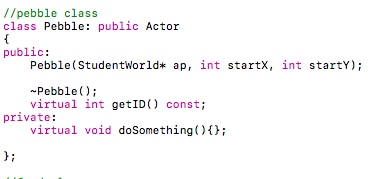
Call eat(Actor\* ap) function to eat

If have some food and eat

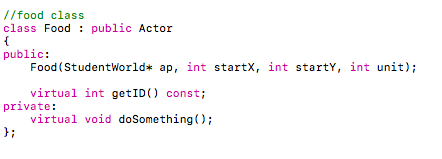
50% chance to move a step

change sleep state;

Class Pebble:



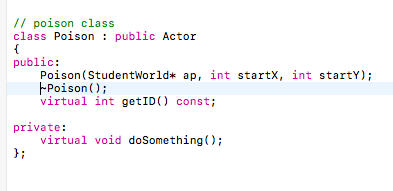
Pebble is the easiest class in the project 3. Actually except for assigning a number as ID, this class really does nothing.

Class Food:dosomething():

Decrease hitpoint;

Check dead;

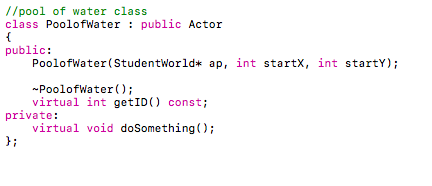
Class Poison:



doSomething()

call toxic(Actor\* ap) from StudentWorld;

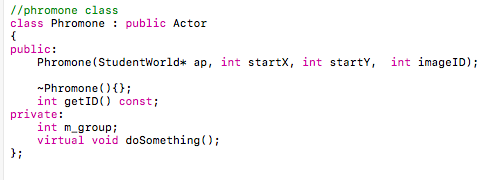
Class PoolofWater



doSomething:

call stun(Actor \*ap) from StudentWorld

Class Pheromone



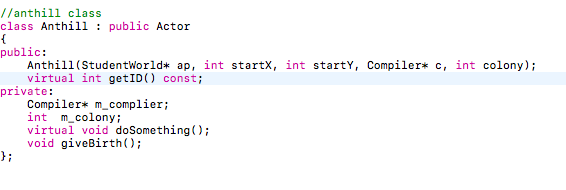
m\_group is used to decide the pheromone belong to which ant colony.

doSomething:

Decrease hitpoint;

Check dead;

Class Anthill



There are two private members for Anthill class m\_complier and m\_colony. m\_colony are used to define what colony it is and m\_complier hold a pointer for Complier class so that make the ant it produced can read commands. We need to define the m\_complier and m\_colony by calling the constructor. doSomething() and giveBirth() are both just used inside the class.

giveBirth():

check colony;

produce the right type of ant for that colony;

add ant to the map by calling addObjecttoSimulation StudentWorld;

set the tick to the last tick for producing a ant by calling settick function from StudentWorld;

doSomething():

decrease hitpoint;

check dead;

check if there is any food by calling havefood(actor\* ap)

if ture

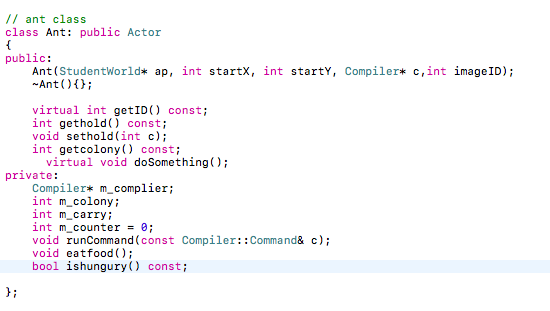
eatfood by calling absorb(actor\* ap)

check if hitpoint greater than 2000

if ture

call givebirth();

Class Ant



There are 4 private members for Ant class. m\_complier and m\_colony have the same function as Anthill. m\_carry is for the value of food hold by a ant and the m\_counter is use to move to the next command. runCommand, eatfood() and ishungury() are only used in the doSomething() function of Ant itself so they are private. There are some function in StudentWorld are passing a pointer of Ant instead of actor, so I put doSomething() in public just in case.

Gethold() and sethold (int c) are used to get and reset m\_carry. Ishugury() will return false is the hit point is smaller than 25. Eatfood() will reduce some amount of m\_carry and add the same amount of m\_carry to hit point.

runCommand will take the command as the argument and check each possible command, if the right one is found than call a proper function making the ant do what command says and increase the m\_counter. If we reach if\_command the m\_counter will be set to (c.operand2), and if we reach goto command m\_counter will be set to (c.operand1). There is parameter called time will a initial value 0 inside runCommand function to let the function return if the time is reach 10.

doSomething():

decrease hit point;

check dead;

If dead:

add new food at the some spot by calling addnewfood(int x, in y, int unit);

update number of ant by calling setNumberofAntsforAnt from StudentWorld;

check sleep

if not sleep

get command from m\_compiler then call runCommand()

check bitten

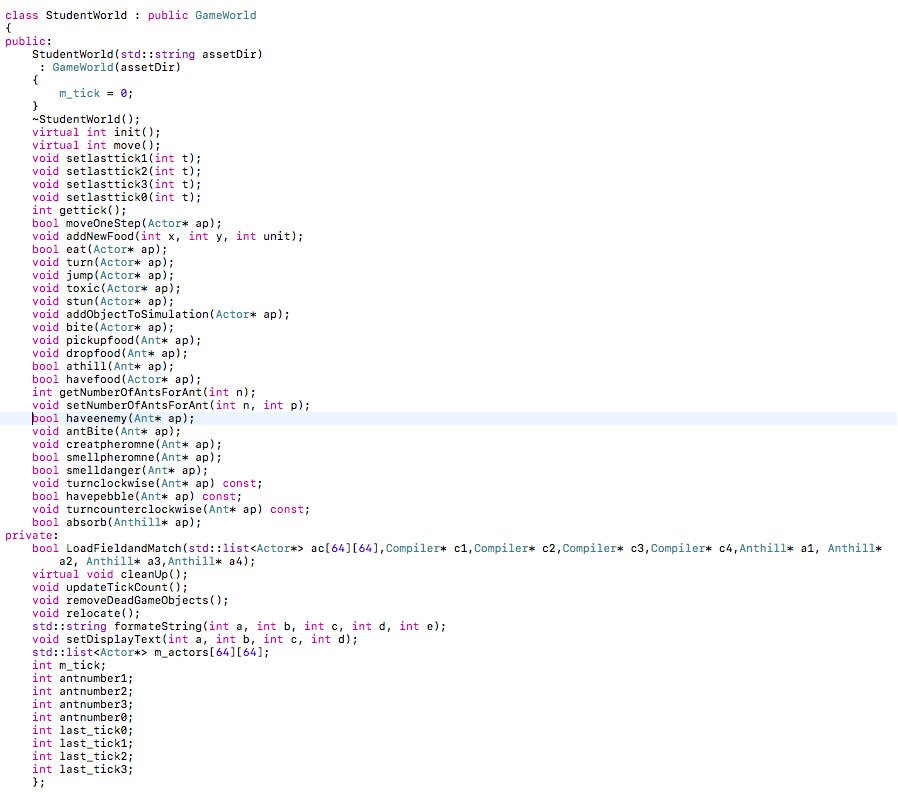
if bitten

chang bitten

Each class from Actor.h will have a destructor with a empty body.

“StudentWorld.h” and “StudentWorld.cpp”

Class StudentWorld



There are 10 private numbers and 5 private functions. antnumber0~3 and last\_tick0~3 are used to decide who is the winner. m\_tick is used to decide where to end the game and list<Actor\*>m\_actors[64][64] are used to hold all the actors in the field. I used a 2D array linked-list to hold all the actors. Each member of the array represent a square of the field and hold a list contains actors on the square. The 5 private functions are all just used in the StudentWorld class.

LoadFieldandMatch used to load the field and push every actor into the m\_actors[64][64] based on the location in the field.

Cleanup() will delete every actors in the field after simulation by using pop\_back().

updatetickCount will increase m\_tick by one.

formateString will take 5 integer and change to string, then form a nice looking string by combine with name and so on.

setDisplayText will call formateString to get the string and call setGameText to show it in the simulation.

init():

create 4 compilers and 4 Anthill;

get correct filename for the for compilers

call LoadFieldandMatch

move():

call updateTickCount;

call doSomething for every actors;

call relocate to relocate moved actors;

call removeDeadGameObjects to delete actors which are set to dead;

update ant numbers;

check if tick = 2000

check if all colonies have ants smaller than six

return no winner

find the biggest value;

check how many colonies have the biggest value by using a stack;

if just one

return the one as the winner;

check which one has the smallest last\_tick to prove the last ant;

return the one as the winner;

return game continue;

removeDeadGameObejects():

use iterator to check each list in the 2D array, if there is a object set to dead; call ~Actor() and erase the iterator.

relocate():

use iterator to check each list in the 2D array, if there is a object’s x and y are not the same as the list itself. Puch the object to the list which has the same x, y as the object and pop the object from the original one.

moveOneStep(Actor\* ap):

check the object’s direction

check if the square in front it is empty

call moveTo() function;

else

check is there is a pebble using pebble id and list.front() because if there is a pebble, the size of the list is just one.

If no pebble than call moveTo() function.

addNewfood(int x, int y, unit)

push a new food actor in the square.

eat(Actor\* ap)

check is the list is empty

if false

use iterator to check is there is a food object in the square

if ture

add some hit points to ap and decrease the same amount of hit point to food

absorb(Anthill\* ap)

use iterator to check if there is some food

increase hit point up to 10000

decrease the same amount hit point to food

The turn(Actor\* ap) function will set the current BabyGrasshopper object to dead and add a new AdultGrassHopper.

Jump(Actor\* ap)

Do:

Get a ramdom x, y has 10 squares away from ap’s position.

If x,y is out of range, set x or y at the edge

Check if square x,y has a pebble, if true, come back to the loop

Use moveTo() to move ap to the new x,y position.

toxic(Actor\* ap)

use iterator to check if there are some ants or baby grasshoppers

decrease 150 hit points to ants or grasshoppers,

if ants or grasshoppers has no more than 150 hit points

set hitpoint to 1( this is because next tick insects will decrease 1 hitpoint and set to dead).

stun(Actor\* ap)

use iterator to check if there are some ants or baby grasshoppers

if there are some baby grasshoppers

if state of stun is false

if getsleep = 2(means next tick baby suppose to move)

set sleep to 1

change stun state

else

change stun state

if there are some ants

if sleep = 0(action mode), then increase to 1

if sleep = 1, then increase to 2

if sleep =2, then set it back to 1

addObjectToSimulation(Actor\* ap)

just push it into the list at the same location

bite(Actor\* ap)

use iterator to check if there is any insects

if true

reduce hitpoint of the insects for 50

if the insects have no more than 50

decrease the hitpoint to 1

change the state of the insects

pickupfood(Ant\* ap)

use iterator to check if there is some food

if true

increase the m\_carry up to 1800 of an ant and decrease the hitpoint of food

dropfood(Ant\* ap)

check if m\_carry great than 0

use iterator to check if there is some food

if true

increase the hitpoint of the food as the values of m\_carry

set m\_carry to 0.

else

create a new food object with the hitpoint the same as m\_carry

set m\_carry to 0

athill(Ant\* ap)

use iterator to check if there is the anthill with the same colony on the squre

if ture

return true

else

return false

havefood(Actor\* ap)

use iterator to check if there is some food

if true

return true

if false

return false

haveenemy(Ant\* ap)

use iterator to check if there are other type of ants or grasshoppers

if true

return true

if false

return false

antBite(Ant\* ap)

use iterator to check if there are other type of ants or grasshoppers

if true

decrease the first one 15 hit points

if the object has no more than 15 hit points

set hit points to one

smellpheromone(Ant\* ap)

use iterator to check if there is a pheromone with the same type as the ant in front of ap

if true

return true

if false

return false

creatpheromone(Ant\* ap)

use iterator to check if there is a pheromone with the same type as the ant

if ture

increase the pheromone by 256, if pheromone will more than 768 than set it to 768.

else

push a new Pheromone object with hitpoint set to 256

smelldanger(Ant\* ap)

use iterator to check if there are other type of ants or grasshoppers in front of ap

if true

return true

if false

return false

havepebble(Ant\* ap)

use iterator to check if there is a pebble in front of ap

if true

return true

if false

return false

turnclockwise(Ant\* ap)

if direction of ap is up

change to right

if direction of ap is right

change to down

if direction of ap is down

change to left

if direction of ap is left

change to up

turncounterclockwise(Ant\* ap)

just do the opposite thing as turnclockwise(Ant\* ap)

**Failed to finish**

I don’t achieve to let an insect randomly choose one to bite if there are more than one other insects. Instead, I let the insect to bite the first one that I can use iterator to find and take it as random.

**Design decision**

The spec says if the pheromone has more than 700 units than set it to 768, I do not sure what I means. So I just set it to 768 when the units will exceed 768 when an ant try to increase the pheromone with 256 units.

The spec does not clear about what an adult grasshopper do if it try to jump out of the field area, I choose to let it jump to the edge area if it try to jump out of bound.

**Testing**

Class Pebble

I watch objects do during the simulation, For example, trying to see if they actually won’t move into a pebble, won’t stop when hit a pebble.

Class PoolofWater

First, I watch if insects will stun in the water, but I found that it is too difficult to do because the simulation is too quick. So, I use cout function in the stun(Action\* ap)

and doSomething try to see if xcode print print something which means the function is actually running.

Class BadyGrasshopper

I watch objects to see if it will in a distiance in a direaction instead of totally random. And if objects will disappear and create a food object when dead. Sometimes I will set object hit points very low to see if they dies correctly. In addition, will it became adult grasshopper at the right time. So, I also print their hitpoint out to see if the hitpoint increase and decrease as expect. Furthermore, letting xcode print something to test certain functions

Class AdultGrasshopper

I do almost same as baby grasshopper.

Class Toxic

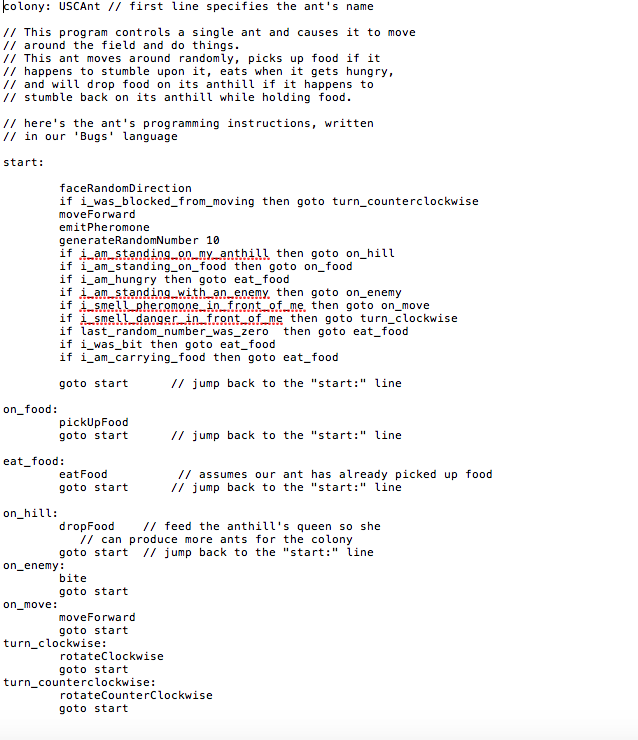
Same as poolofWater

Class Anthill

Watch if there will be 5 ants be produced at the beginning. Let Xcode print some text to test doSomething function.

Class Anthill

Besides just watch and print text, I build my own command and keep modifying to test if ants are doing what I tell them to do. It turns out the anthill can eat food and produce new ants, but it is very hard to let ants who successfully carry food to go back. I never win the game. Here is my testing command



Class StudentWorld

I get lots of function inside this class. So, I tell them one by one by letting xcode print unique texts. Sometimes I need to comment other functions for just testing one. Sometimes I try to set the simulation times less than 2000 to check some special situation. I also load different field file and bugs file to make sure they all work properly.