**High-level description**

Inheritance hierarchy:

GraphObject

Actor

Pebble

poolOfWater

poison

energyHolder

Anthill

Food

Pheromone

Insect

Ant

Grasshopper

Babygrasshopper

adultgrasshopper

Class Actor: public GraphObject //this is a base class for all objects in the StudentWorld Simulation

Member functions:

1. Virtual void doSomething()

I make this function pure virtual because all actors are able to do something, and each type of actor would have different version of doSomething()

1. Virtual bool checkStatus() const

This function is used to check actor’s status. This function would simply return true for non-energy holders, such as pebble, poison and water. I make this function virtual because for energy holders, such as food, ant and grasshoppers, this function can be modified to check whether their energy is positive.

1. StudentWorld\* getMyworld() const

This function returns a pointer to the current StudentWorld the actor lives in. Then all the actors can use this function to access StudentWorld to do something for them. For example, when an ant dies, it can ask the StudentWorld to add a food object on its position. Since this common functionality are all the same across different actors, I didn’t make it virtual

Class poolOfWater: public Actor //this is a water class derived directly from Actor base class

1. Virtual void doSomething()

Here I implement how the water class would behave during every tick. Since in Actor class function doSomething() is virtual, by convention, I made it virtual here in poolOfWater class.

Class poison: public Actor // this is a poison class derived directly from Actor base class

1. Virtual void doSomething()

Here I implement how the poison class would behave during every tick. Since in Actor class function doSomething() is virtual, by convention, I made it virtual here in poison class.

Class energyHolder: public Actor // this is energyholder class. EnergyHolder is the base class for classes

Like ant, anthill, etc. EnergyHolder is derived from Actor class

1. Virtual bool checkStatus() const

EnergyHolders have a different way to check its status, since it might disappear due to death. Therefore, I implement this function to check for the status of energyHolder by returning m\_status, a private Boolean variable. This function is virtual because in base Actor class, checkStatus() is virtual

1. Void changeEnergy()

I implement this function to change the m\_energy variable, which can mean hitpoint for ant object, or food amount for food object, and the strength for pheromone.

1. Int getEnergy() const

I implement this function to retrieve m\_energy. This, for example, would allow me to check whether the food is enough for ant/grasshopper to eat. Since this function would be all the same for different energy holders. I didn’t make it virtual.

1. Void setTodeath()

I implement this function to set energyHolder object to death. By doing so, combining with the checkStatus() function, StudentWorld would be able to remove the dead object from its simulation. Since this function would be all the same for different energy holders. I didn’t make it virtual.

1. Bool attemptToEat(int amount)

This function checks whether there is food object on the current spot of the energyHolder. It then determine how much food the energyHolder object can eat. I use this function for grasshoppers and anthills to check whether there is food to eat. Since all energy objects behave the same, I didn't make this function virtual.

Pseudo-Code:  
bool energyHolder::attemptToEat(int amount)

{

Check whether there is food on the current spot (calling StudentWorld::checkForItem)

If there is food

Eat some food based on argument amount or the food left in the food object

set the food object to death if its energy is 0

Return true

If there is no food, return false

}

Class insect: public energyHolder //insect object is a version of energyHolder, because insects can move

around, bite other insects, and being stunned/poisoned, etc.

1. Void changePreviousWaterStunned()

I implement this function as a helper function to prevent permanent water stun by the water object. It simply changes the previouslyWaterStunned variable to indicate that it is not previously stunned by water. Since this function works the same across different classes, I didn't make it virtual.

1. Bool getPreviouslyWaterStunnedStatue() const

I implement this function as a helper function to prevent permanent water stun by the water object. When the water object tries to stun the insects on the spot, it can check whether they have already been stunned by the water on the same spot once, thus preventing the permanent stun. Since this function works the same across different classes, I didn't make it virtual.

1. Bool actionBasedOnStatus();

I implement this function as a helper function to doSomething() function of derived class of insect class. It would return true if the insect is able to actually do something in this tick. Otherwise, if the insect is dead or stunned, this function would return false. Since this function works the same across different classes, I didn't make it virtual.

Pseudo-Code:

Bool insect::actionBasedOnStatus()

{

Check whether the insect is still alive,

If not, add food object on the spot, set the insect to death and return false

Check whether the insect is stunned/sleep

If yes, then decrement the ticks it supposed to be stunned/sleep and return false

If the insect is both alive and active, return true

}

1. Void chooseRandomDirection();

I implement this function as a helper function for doSomething() function. This function simply helps to choose a random direction. For all types of insects, at some point, they will be asked to choose a random direction, so I didn’t make it virtual.

1. Virtual void poisoned()

Poisoned() will be called when the insect object is poisoned. I implement this function so that poison object can call when there is insects on same spot as the poison object. Since the adultgrasshopper cannot be poisoned, I put this function virtual so that I can implement the updated version in the adultgrasshopper class.

1. Virtual void beingBite(int amount)

beingBite() will be called when adultgrasshopper or ant attacks this insect. I implement this function so that every time the adult grasshopper or ant want to attack, they can easily call this function and use parameter amount to specify the damage they plan to exert on the insect. This function is virtual because adultgrasshopper would behave differently, they will have chances to bite back.

1. Void stunned()

I implement this function as the general stun function (except for the use of waterstun). Since all insects can be stunned/sleep in the same way, I didn't make this function virtual

1. Virtual void waterStunned()

I implement this function as the special version of stun. This function is specifically called when the water object try to stun the insect upon it. In this function, I checked whether the insect has already been stunned before, to prevent permanent stun by the water object. Since the adult grasshopper behave differently, I made this function virtual.

1. Void becomeActive()

I implement this function as a helper function. It just simply decrement m\_state, a private variable used to track the ticks insects would sleep. Since this function works the same across different classes, I didn't make it virtual.

1. Int getMyState() const

I implement this function to check whether the insect is stunned. By returning m\_state, this function can determine how many ticks the insect should wait in order to make the next move. Since this function works the same across different classes, I didn't make it virtual.

1. Bool walkingInDirection()

I implement this as a helper function. This would return true if it is possible for the insect to move in the designated direction. If it is blocked by a pebble, this function would return false. Since this function works the same across different classes, I didn't make it virtual.

Pseudo-Code:

Bool insect::walkingInDirection()

{

for four different directions(up, down, left, right)

check whether there is a pebble object on that spot to block the insect.

If there is, return false

Else move to that spot and return true;

}

Class food: public energyHolder //food class derived from energyHolder

1. Virtual void doSomething()

According to the spec, food object does not do anything during a tick. So there is no code in this function. And based on the conventionality of inheritance, I made this function virtual.

Class pheromone: public energyHolder //pheromone class derived from energyHolder

1. Virtual void doSomething()

This function is required by the spec. I implement this function according to the spec. Because of the conventionality of inheritance, I made this function virtual.

1. Int getColony() const

This function returns which anthill this pheromone belongs to. I implement this function so that the StudentWorld object can figure out whether to create a new pheromone object or add 256 on the existing one, given that some ant object emit pheromone on some spots. Since this is the unique feature of pheromone object, I didn't make it virtual.

Class ant: public insect //ant class derived from insect

1. Virtual void doSomething()

This function is required by spec. I implement this function according to the spec. And according to the conventionality of Objective-oriented programming, I made it virtual.

Pseudo-code:

void ant::doSomething()

{

changeEnergy(-1) // decrement hitPoint

check whether it is dead, if it is immediately return

check whether it is stunned/sleep, if it is immediately return

while (the ant didn’t execute more than 10 commands in this tick)

get the next command

for different commands, do different things

et the instruction counter to next command

if the ant attempt to change the outside simulation, return

}

1. Int getMyColonyNum() const

This functions returns the ant object’s colony number. I implement this so that when creating the pheromone object, it is able to check whether the pheromone on the spot has the same colony number as the ant. Since this is the unique feature of ant object, I didn't make it virtual.

1. Int convertStringToInt(string s)

This is a very useful helper function that convert an input string to an output integer. I implement it so that I can called it when I need to converting command.operand1 or command.operand2. Since this is the unique feature of ant object, I didn't make it virtual.

1. Bool dangerInFrontOfMe()

This is a helper function for ant’s doSomething. Combining with enemyOnTheSpot, it can check whether there is any ant, baby/adult grasshoppers in the direction the ant is facing. Since this is the unique feature of ant object, I didn't make it virtual.

Pseudo-Code:

Bool ant::dangerInFrontOfMe()

{

For four different directions(up, down, left, right)

Check the number of enemies in the corresponding spot(calling enemyOnTheSpot() function)

If there is enemy, return true, else return false

}

1. Bool pheromoneInFront()

This function is very similar with the last one above. It returns whether there is a pheromone in front of the ant. I implement this function as a helper function for doSomething(). Since this is the unique feature of ant object, I didn't make it virtual.

1. Virtual void beingBite(int amount)

This is an updated version of beingBite() function for insect object. In this function, besides the functionality of insect::beingBite(), the ant object also need to keep track its status, about whether being bite. According to the conventionality of OOP, I made this function virtual.

1. Vector<insect\*> enemyOnTheSpot(int posX, int posY)

This is an helper function for ant::doSomething(). I implement this function so that it returns the enemy (which include the enemy ant, baby/adult grasshopper) on spot (posX, posY) as a vector. Note that this is different from StudentWorld::actorOnTheSpot, which gives all the insect objects. Since this is the unique feature of ant object, I didn't make it virtual.

Pseudo-Code:

Vector<insect\*> ant::enemyOnTheSpot(int posX, int posY)

{

Get all insect on the spot(posX, posY) (calling StudentWorld::actorOnTheSpot function)

Iterate through the vector, eliminate ant from the same colony and keep other enemys

Return the updated vector

}

1. Bool eatFoodICarry()

This is a helper function for ant::doSomething() function. It returns true if ant do eat some food it carries. Otherwise this function would return false. Since this is the unique feature of ant object, I didn't make it virtual.

Class Anthill: public energyHolder //anthill class derived from energyHolder

1. Virtual void doSomething()

This function is required by spec. I implement this function according to the spec. And according to the conventionality of Objective-oriented programming, I made it virtual.

Pseudo-Code

Void Anthill::doSomething()

{

Decrement hit point

Check whether the anthill is dead

Try to eat the food on the spot (calling energyHolder::attemptToEat function)

if anthill have enough to produce a new ant

create an ant with same colony number and same compiler

}

1. Int getMyId() const

This is a helper function for Anthill::doSomething(). It fetches the m\_id (aka colony number) of the anthill. So when the ant object check whether it is on its own anthill, the ant object can call this function to check. Since this is a unique feature of Anthill, I didn’t make it virtual.

1. Compiler\* getMyCompiler() con.st

This is a function that fetches the pointer to Compiler object. I used this function in Anthill::doSomething() to create ant object (required by the spec). Since this is a unique feature for Anthill, I didn’t make it virtual.

Class Pebble: public Actor //pebble class derived from Actor

1. Virtual void doSomething()

According the spec, a pebble object will do nothing during a tick. So there is no code. And according to the conventionality of Objective-oriented programming, I made it virtual.

Class grasshopper: public insect // grasshopper class derived from insect

1. Int getWalkDistance()

This function return m\_walkDistance, a private variable keep tracking the walking distance. This can help me to determine whether the grasshopper has walked all the steps it supposed to walk. Since this is the same for derived classes, I didn't make it virtual

1. Void takeOneStep()

This function simply decrement m\_walkDistance. I use this as a helper function for both babyGrasshopper::doSomething() and adultGrasshopper::doSomething(). Since it is the same for both derived classes, I didn’t make it virtual .

1. Void setDistZero()

This function simply set m\_walkDistance to 0. When the grasshopper is blocked by pebble, doSomething() function can call this function to set the m\_walkDistance to 0. Since it works the same for both derived classes. I didn't make it virtual.

1. Void eatAndSleep()

This is a helper function for both babyGrasshopper::doSomething() and adultGrasshopper::doSomething(). This function would decide how much the grasshopper can eat, whether to walk, and sleep afterward. Both the baby and adultGrasshopper behave the same when it decide to eat and sleep. So I didn't make this function virtual.

Pseudo-code:

Void grasshopper::eatAndSleep()

{

Try to eat(calling energyHolder::attemptToEat(200))

If it is eat and by 50% chance it would decide to sleep, so set the tick to 2 and return

Try to walk(calling getWalkDistance() and walkingInDirection())

If the grasshopper finish this walk, reset its direction

If the grasshopper is blocked

Set the assigned walking distance to 0, and stun the grasshopper, return

Else decrease the assigned walking distance and stun the grasshopper

}

1. Void Reset()

This is a helper function that helps to get a random direction and a random walking distance for the grasshopper. Since it works the same for both baby and adult grasshopper, I didn’t make it virtual.

Class babyGrasshopper: public grasshopper // babyGrasshopper class derived from grasshopper

1. Virtual void doSomething()

This function is required by spec. I implement this function according to the spec. And according to the conventionality of Objective-oriented programming, I made it virtual.

Pseudo-Code

Void babyGrasshopper::doSomething()

{

Get hungrier (calling changeEnergy() function)

Check current status, whether it’s alive and whether it is stunned/sleeping (calling insect::actionBasedOnStatus() funtion)

//now the baby grasshopper would do something

If it is previously water stunned, set this status to false

If the hitpoint reached 1600, evolve into adult grasshopper and return

Try to eat/walk, and sleep(calling grasshopper::eatAntSleep() function)

}

Class adultGrasshopper: public grasshopper // adultGrasshopper class derived from grasshopper

1. Virtual void doSomething()

This function is required by spec. I implement this function according to the spec. And according to the conventionality of Objective-oriented programming, I made it virtual.

Pseudo-code

Void adultGrasshopper::doSomething()

{

Get hungrier (calling changeEnergy() function)

Check current status, whether it’s alive and whether it is stunned/sleeping (calling insect::actionBasedOnStatus() funtion)

Decide whether to bite, if yes, then call bite() function

Decide whether to jump, if yes, then call jumpHigh() function

Try to eat/walk and sleep (calling grasshopper::eatAndSleep() function)

}

1. bool bite()

This has to be a function because, combining with adultGrasshopper::beingBite(), there might be a recursion going on (when these two function continuously call each other until one grasshopper dies). Since this is the unique behavior for adult grasshoppers, I didn’t make it virtual.

1. bool jumpHigh()

This is a helper function of adultGrasshopper::doSomething(). If the adult grasshopper decided to jump, this function would randomly select an empty spot within the distance of 10. After moving to that spot, the grasshopper will be stunned. Since this is a unique feature for adult grasshopper, I didn't make it virtual.

Pseudo-Code:

Bool adultGrasshopper::jumpHigh()

{

For every spot on the simulation world

If the spot is within 10 distance from the adultGrasshopper and there is no pebble

Push its location into the vector

Randomly select one location in the vector and moveto that location.

}

1. Virtual void beingBite(int amount)

This is an updated version of insect::beingBite(). I implement this function to include the functionality that there is a chance the adult grasshopper would bite back. So when it is called in other doSomething() function, it can realized the functionalities described in the spec. According to the conventionality of OOP, I made it virtual.

1. Virtual void poisoned ()

This is an updated version of insect::poisoned(). According to the spec, the adult grasshopper cannot be poisoned, so when the poison call this function, it will do nothing (i.e. no code in this function). According to the conventionality of OOP, I made it virtual.

1. Virtual void waterStunned()

This is an updated version of insect::waterStunned() function. According to the spec, the adult grasshopper cannot be stunned by the water. So this function do not have any code. According to the conventionality of OOP, I made it virtual.

Class StudentWorld: public GameWorld //StudentWorld class derived from GameWorld as spec required

1. Virtual int init()

I implement this function as required by the spec

Pseudo-Code:  
int StudentWorld::init()

{

Load the field by using the Field class and its member function

For each actor in the field

Get the item property

Create a pointer based on the item property (whether it’s a pebble, ant, etc.)

Add this pointer to the data structure (I used 2D array of linked list, like the spec suggested)

Return successfully loading, and let the simulation begin

}

1. Virtual int move()

I implement this function as required by the spec. Particularly, I used set data structure to make sure that during every tick, the actors in the simulation world only call doSomething() function once.

Pseudo-Code:

Int StudentWorld::move()

{

Update tick counts

For every actor on the simulation

If they are not newly created, and if they haven’t do something

Let them doSomething()

If their location changes then update their new location

Remove newly-dead actors after each tick

Update the display text

If the game reaches its end

If we have a winner then set the winner and return we have a winter

Else return we don't have a winner

Else the game is not end then return the game will continue.

}

1. Virtual void cleanUp()

I implement this function as required by the spec:

Pseudo-code:  
virtual void cleanUp()

{

For every linked list in the 2D array

While the linked list is not empty

Get the front actor pointer and delete it

Pop the front actor pointer from the linked list

}

P.S. Since all other member functions are helper functions for init(), move(), and cleanUp(), I didn’t make them virtual.

1. Void addAnt(int posX, int posY, int colonyNum, Compiler\* comp)

This function create a new ant object on spot (posX, posY) with colony number colonyNum and compiler pointed by comp. This function will be called when anytime anthill has enough hitpoints to create a new ant.

1. Bool compileFile(int fileNum, Compiler\*& comp)

This function provide a method to use the compiler class to compile the file fileNum. This function will be called when anthill are created in the StudentWorld::init().

1. Int getCurrentTicks() const

This function return current tick of the simulation. This function will be called in the move() function to keep track of the ticks.

1. Void addFoodFromBody(int posX, int posY, int amount)

This function would add a food object with amount specified on the spot(posX, posY). This function will be called when there is an insect dies, or ant object drops food on some spots.

1. Void addAdultGrasshopper(int posX, int posY)

This function would add an adult grasshopper on the spot(posX, posY). This function would be called when the baby grasshopper evolve to the adult grasshopper.

1. Void updateTickCount()

This function simply increment m\_ticks, a private variable used to keep track of the ticks of the simulations. This function will be called in the StudentWorld::move() function as a helper function.

1. Void addPheromone(pheromone\* pp, int colNum, int posX, int posY)

This function would add a pheromone object with colony number specified by colNum at spot(posX, posY). Note that it either add strength, 256, to the existing pheromone object or create a new one with strength 256. This is decided by whether the pheromone pointer points to nullptr.

1. Void removeDead()

This function would remove all dead actors in the simulation. This would be called in move() function after every actor in the simulation doSomething()

Pseudo-Code:  
void StudentWorld::removeDead()

{

For every actor in the simulation world

If the actor is dead

Delete this object and remove the pointer from the data structure

}

1. Void changeNumOfAnt(int Id, int Num)

This function simply change the number of ant with colony number Id by Num. This function would be called every time anthill produce a new ant.

1. Vector<insect\*> actorOnTheSpot(int posX, int posY, insect\* myself)

This function return every insect object on the spot(posX, posY) except insect pointed by insect pointer myself.

1. void checkForItem2(Actor\* a, Item\*& f)

This function checks whether a pointed by an actor pointed can also be pointed by an item pointer. Here I utilized template feature and dynamic cast feature of C++. This function will be called to construct the vector in the actorOnTheSpot function.

1. void checkForItem(int posX, int posY, Item\*& f, int n = 5, int m = 7)

This function checks whether there is an Item class actor in the spot(posX, posY). Here argument n and argument m is used to identify whether “item” is pheromone or anthill. Specifically, n represent the colony number for pheromone and m represent the colony number for anthill. This function will be called to construct the vector in the actorOnTheSpot function.

Pseudo-Code

template <class Item>

void checkForItem(int posX, int posY, Item\*& f, int n = 5, int m = 7)

{

For every actor on the spot(posX, posY)

Check whether it is an Item object

If it is then for pheromone and anthill, check whether it has the same ID

If pass both case then set f to pointer point to this object

Else set f to nullptr

}

1. void setDisplayText()

This is a function helps to deal with how to write the text record of the game, like specified in the spec. I implement this function by following the pseudocode provided by the spec. This function will be called in the move() function.

1. void updateLocation(Actor\* a, int oldX, int oldY)

This function would remove actor pointer a from the spot(oldX, oldY) and put it into the new location. This function would be called in the move() function.

1. int getNumberOfAntsForAnt(int i) const

This function would return the number of ant produced in the simulation by ith anthill. This function would be called to help construct the text record, as well as determine the winner.

1. int getWinningAntNumber() const

This function would return the winning ant number. If there is a tie of the highest score, then this function would return 5

1. vector<string> getAntnames()

This function returns ant names in a vector. This function would be called to construct the text record as well as set the winner at the end of the game.

**Functionalities Failed to Implement**

None

**Assumptions and Design**

1. Each newly created actors will not doSomething() in the current stick

Example, if the baby grasshopper turns into an adult grasshopper and a food in this tick. The adult grasshopper will not act (will not eat the food, jump, or bite, etc.) during this tick.

1. If a food object has been dropped by the ant on its anthill during this tick, the Anthill will not eat the food object during this tick.
2. Left/down insects would act first (bite or move). For example, say if an ant is at (i,j), and there is an adult grasshopper at (i+1,j)/(i,j+1), and these two insects are facing each other. The ant would always have the right to act first. It might move to where the grasshopper is. And then, during this tick, the grasshopper would decide whether to bite, jump, or just move forward.
3. If there is two insects in one spot, and they are enemy, the one who come to the spot first would act first. For example, if an adult grasshopper is at spot(I,j) and then and ant comes (the grasshopper stays because it can sleep). The grasshopper would act first, deciding whether to bite the ant, or jump, or move forward.

**Test Cases**

StudentWorld

I tested this class mainly on three member functions: init(), move(), and cleanUp()

For init() function, I checked by changing the field file (e.g. field.txt), and check whether the bug.exe can correctly initialize my field. For the move() function, I mainly made sure that during every tick, actors in the simulation world only call doSomething() once. Also I checked this function with different actors, such as ant, grasshopper, poison, and food, to see whether they behave correctly. Also I monitored dead objects, to see whether they are successfully removed after each tick. Moreover, by observing bug.exe directly, I can see whether setDisplayText() function work correctly (i.e. check the number of ant, the winning ant name, etc.). For the cleanUp() function, I simply put “cout << “clean up” << endl;” command at the end of the function to see whether all the object have been correctly destroyed.

Actor

Since Actor class is the base class for all actors in the simulation. It has some member functions with no code. Testing them seems trivial. However, I did test actor’s constructor with pebble class constructor and StudentWorld::init() function, to see whether it can correctly construct derived object.

Pebble

I tested this class with grasshopper and ant. The main function for pebble class is to block insects like ant and grasshopper. So I put a lot of pebbles in my StudentWorld simulation to see whether my ant and my grasshopper will be blocked. Also, I tested pebble with adult grasshopper particularly when it decided to jump. The adult grasshopper should never jump to the spot where there is a pebble object on it.

poolOfWater

I tested poolOfWater() class to see whether it can successfully stun the insects on it (except adult grasshopper), and to prevent the permanent stun (the insects forever stays on the spot of water object).

To test whether the pool of water class can stun insect. I change the field file by putting a lot of water object into it. And by observing one single ant/baby grasshopper/adult grasshopper moving in the simulation world, I can see how they behave when they encounter the pool of water object.

To test whether the pool of water object would cause permanent stun. I follow the step described in the last paragraph to see whether the insect would all stay there forever without moving.

Also I checked several special situation like how would the insect behave when being water stun and being blocked have simultaneously.

poison

I tested poison() class by checking whether it can successfully poison the insects on it (except adult grasshopper) I tested this function by modifying the field file, putting a lot of poison in the simulation world, and monitoring the hitpoint of insects as they move around and encounter the poison object. After the poison object kills the insect, it can successfully add food object into the simulation world and remove the insect body.

energyHolder

This is the base class for insect (baby/adult grasshopper, ant), anthill, pheromone, and food. It has some important functions: changeEnergy(), getEnergy(), setToDeath(), attemptToEat(int amount), checkStatus()

Since all these five functions are closely related with the insect’s hitpoint state. I test them separately with baby grasshopper’s doSomething() function. As the baby grasshopper move around the field. I monitor the m\_energy variable (calling getEnergy() function) to see whether it correctly keep track of baby grasshopper’s hit points. As the baby grasshopper move to the spot where there is a food object, I tested the changeEnergy() function, and attemptToEat() function. When the baby grasshopper encounter the poison and died, I tested the checkStatus() function and setToDeath() function. Since this works fine with baby grasshopper, according to inheritance principle, these functions should work fine with ant, adult grasshopper, anthill, pheromone, and food objects.

Anthill

I tested this function by testing whether it can correctly produce ant and whether it can correctly eat the food object dropped by the ant. To test whether it can correctly produce the ant, I arbitrarily increase the hitpoint of anthill 100 per ticks. And see whether the new ant object would be produced every 15 ticks. To test whether it can eat the food dropped by the ant, I just arbitrarily put food object on the spot of anthill and see whether it can eat it. Both these test cases are achieved by monitoring the change of hitpoints of anthill object.

Food

To test food class, I tested the doSomething() function, and the interaction between food objects, anthill, and insect objects (ants, baby/adultGrasshoppers). In order to see insects and anthills can “consume” food, I change field file so that there is many food object in the simulation world. Since there is plenty interaction between insects, anthills, and foods, I can test the functionality of doSomething() function as well as whether the insect and anthills can actually “eat” food.

Pheromone

I tested pheromone class by testing its doSomething() function and its interaction with ant object. Pheromone’s doSomething() function is fairly simple, I just monitor its hitpoint to see whether it would decrease by one during every tick. For the interaction with ant object. I tested whether the ant could correctly emit and identify its pheromone. I tested this interaction by adding a lot of pheromone objects into the field, and command the ant to emit pheromone once it see a pheromone object in front of it. This would be quite obvious on the simulation window.

Insect

This is the base class for ant, baby/adult grasshopper. It has some important functions to test: actionBasedOnStatus(), chooseRandomDirection(), poisoned(), beingBite(), stunned/waterStunned, walkingInDirection().

I tested these function by simulating the baby grasshopper movement. actionBasedOnStatus() is the helper function. It helps to check the status of baby grasshopper. I arbitrarily set the grasshopper to death/stunned, to test the functionality of this function. Similarly, I can arbitrarily surrounded the baby grasshopper with pebbles (so that it will always reset its direction and walking distances), so that I can check the functionality of chooseRandomDirection(). To check stunned/waterStunned function and poisoned() function, I monitor the interaction between poison object, poolOfWater object and baby grasshopper object (by setting the field with a lot of water, poison and baby grasshopper to create such interaction). The interaction could test the functionality of these three functions. For beingBite() function, I tested this function with adultGrasshopper::doSomething() function. I monitor the hitpoints of two adultGrasshoppers to test the functionality of beingBite() function.

Ant

To test ant class, I mainly focus on testing doSomething() function, since other functions, such as

convertStringToInt(string s), dangerInFrontOfMe(), pheromoneInFront(), enemyOnTheSpot(int posX, int posY) are all helper functions to doSomething() function. By testing different sub functions in doSomething(), I can indirectly test helper functions listed above.

First test moveForward command, by writing only this command in the Ant.Bug file, we can see the and select a random direction and move until it was blocked by the pebble. Then I tested pickUpFood, eatFood and dropFood command, as well as several if\_command. I put “if i\_am\_carrying\_food then goto eat” , “if i\_am\_standing\_on\_food then goto on\_food”and “if i\_am\_hungry then goto eat”. By print ant’s hitpoint and observe its movement, I can check whether these commands are implemented correctly.

Also, I tested faceRandomDirection, generateRandomNumber, rotate clockwise and rotate counterclockwise commands with the moveForward commands. I put “if i\_was\_blocked\_from\_moving then goto on\_block” “if last\_random\_number\_was\_zero then on\_block”and “on\_block: faceRandomDirection/generateRandomNumber/rotateClockwise” commands into the Bug file. Then I can observe the movement of the ant to determine whether these commands are implemented correctly.

For function like bite(), dangerInFrontOfMe(), enemyOnTheSpot() and conditions like i\_smell\_danger\_in\_front\_of\_me, i\_was\_bit, i\_am\_standing\_with\_an\_enemy, I tested by putting ant from different colonies and adultgrasshoppers together into an area surrounded by pebbles. By monitoring the hitpoints of different insects, I can see whether these functions and conditions work properly.

Similarly, I tested out whether the ant can correctly recognize the anthill object, and pheromone object with the same ID by forcing them encounter each other.

Grasshopper

This is the base class for baby and adult grasshoppers. I tested this class by testing its member functions, like Reset() function and eatAndSleep() function. Since Reset() function will be called when the grasshopper is blocked by the pebble. I simply change the field file so that grasshoppers are surrounded by the pebble object. In this way, I can check whether the Reset() function is called correctly and whether it will produce the desirable result. For the eatAndSleep() function, I tested it with babyGrasshopper::doSomething() function. I changed the field file so that there is not so many pebbles and poison/water object. In this way, eatAndSleep function will be more likely to be called, and every time it is called I can check whether it is working properly.

Babygrasshopper

The baby grasshopper class is the derived class of grasshopper, I tested this class by testing the doSomething() function. I print out babygrasshopper hitpoint and state (whether it is stunned) every time I called doSomething() function. This helps me check whether this function is working properly.

Adultgrasshopper

The adult grasshopper class is the derived class of grasshopper. I tested this class by testing its member functions like jumpHigh(), doSomething(), bite(), poisoned/waterStunned()

I tested to doSomething() and jumpHigh() function by simply observing the movement of adultGrasshopper and monitoring the hit point of the adult grasshopper. To test bite() and beingBite() function, I put 2 grasshopper without jumpHigh functionality into an area surrounded by pebbles. So that there must be some instance of biting. When they bite each other, I checked whether bite() and beingBite() function are implemented correctly. For poisoned/waterStunned function, I changed the field file so that there are a lot of water/poison objects in the field to create interaction between adult grasshopper and water/poison. As poisoned/waterStunned function are called, I can test whether it works properly.