Artificial Ants: Simulating Ant Behaviour and Investigating Applications In Computing

BSc Computer Science Project

James Hamilton

May 2007

Goldsmiths College, University of London

Supervised by Dr Sebastian Danicic

Ants are excellent route finders, with the ability to find the shortest path between their nest and a food source, by using pheromone trail clues to organize themselves collectively. By applying behaviour observed in ant colonies to the field of computing, these ant-based techniques can provide a base for network routing algorithms, as well as algorithms to solve graph problems such as the travelling salesman problem. This report introduces ant behaviour by the implementation of an ant simulation, and finally applies lessons learned from the simulation to graph and network routing problems.

Table of Contents

Introduction

Researchers have been looking into the use of Artificial Ants to improve network communications for sometime, and the results have shown that ant foraging behaviour used as a basis for modelling network traversal algorithms are very efficient [1]. Ant based algorithms can also be used to introduce anonymity into peer-to-peer networks, by replacing the need for global routing tables with local routing clues in the form of pheromone trails [2]. Another application is ad-hoc networks [3], where hosts leave and join the network sporadically – ants can help here too: they are good at finding new routes if a route has become blocked, or reacting to a new, closer food source.

Figure 1: Meat Eater Ant Feeding on Honey (from Wikimedia Commons)

Nature has already solved many problems that these researchers are trying to solve and/or improve. Ants are very good at finding the shortest distance between their nest and a food source, and are able to work together efficiently. From the point of view of swarm intelligence ants individually are very stupid but an intelligence emerges when they start working together.

So how are ants good at working together to find short paths to food, and how can this be applied to the field of computing?

This report is divided into two parts. Part one attempts to explain the notion of ant behaviour in order to express how ants find shortest paths, and demonstrates this via the use of a graphical ant simulation. Part two uses the knowledge gained from part one and applies this to an area of computing, namely graph routing, by the introduction of shortest path problems in graph theory and applying the previously demonstrated ant behaviour to finding these paths.

Summary of Application Development

Two applications have been developed as a result of this report. Each application was developed incrementally, and every version is documented in this report. Problems and developments and changes are discussed for each version.

Ant World Simulator

The Ant World Simulator aims to simulate ant behaviour and allow users to observe the emergent intelligence of the ants.

Version 1 – Implements basic application structure, random ant movement and food.
Version 2 - Introduces a nest, ants pick up food and randomly find their way back to the nest.
Version 3 – Same as version 2 but different layout of food and obstacles.
Version 4 – Ants lay pheromone trails while they are carrying food.
Version 5 – Introduces food strengths and a threaded nest.
Version 6 – Introduces the FoodCluster class, and slider control for food strength.
Version 7 – Bug fix for version 6.
Version 8 – Removes the FoodCluster class, food strength calculated by individual grid squares.
Version 9 – Ants 'know' where their nest is. They find food and return it to the nest. Food strength calculations are changed to be more efficient.
Version 9.1 – Fixes 3 bugs from version 9.
Version 10 – Introduces an information dialog.
Version 11 – Re-introduces pheromone trails, ants follow the trails with some success.
Version 12 – Changes the way ants move, to solve problems discovered in version 11.
Version 13 – Unsuccessful attempt to create a smoother animation. Roll back to version 12.
Version 14 – Includes a re-written move algorithm, and ants follow a pheromone from the nest to the food slightly better than version 12. Includes two experiments.
Version 15 – Implements a directional pheromone trail, ignores pheromone strength. Includes experiment.
Version 15.1 – Uses directional pheromone trail, with strength. Includes experiment.
Version 16 – Takes a different approach to pheromone trails.

Ant World Graph

Ant World Graph simulates the Ant World using a graph structure. This application attempts to demonstrate how ants based algorithms can be used to solve shortest path problems on graphs, and how this can be applied to the field of computing – such as peer-to-peer networking.

Version 1 – Implements a basic graph data structure and GUI for end-user to create a graph.
Version 2 – Enhanced GUI.
Version 3 – Improved underlying GUI code.
Version 4 – Implements ant algorithms for finding the shortest distance between nest and food nodes.

Background

Ant Behaviour

Individually ants are stupid. But ants show emergent intelligence when they work together. it is this collective intelligence that is so interesting to computer scientists. Many stupid ants can find the shortest path between their nest and a food source – computer scientists are now applying these ant based techniques to computer problems.

Figure 2: Ants at Work (from Wikimedia Commons)

Each individual ant follows some simple rules: they leave the nest with a mission to find food, or a pheromone trail that leads to food. If they find food without the help of a trail, they will follow their own route back to the nest. Some species use visual clues [4], others use geometrical clues in their trails [5], but they know how to return to their nest. When ever an ant is carrying food its new mission is to return to the nest and lay a trail for other ants to follow. Ants always follow the strongest trails, resulting in positive feedback – a trail is strengthened, so more ants follow it, and then the more ants following it strengthen it more.

The double bridge experiment

The double bridge experiment [6] is a simple way to demonstrate the shortest path finding ability of ants. For this experiment we will have two ants, a and b, whose task it is to find food and return it to the nest. There are two routes for these ants to take to the food, one being double the length of the other. Ants do not like travelling a long distance to find food, they'd rather find the closest food source, lets see how they do that.

Figure 4 shows the initial set-up of the experiment, where the top route is shorter (route A) than the bottom route (route B). Ants a and b are going to set off at the same time, one will take the top and the other the bottom route.

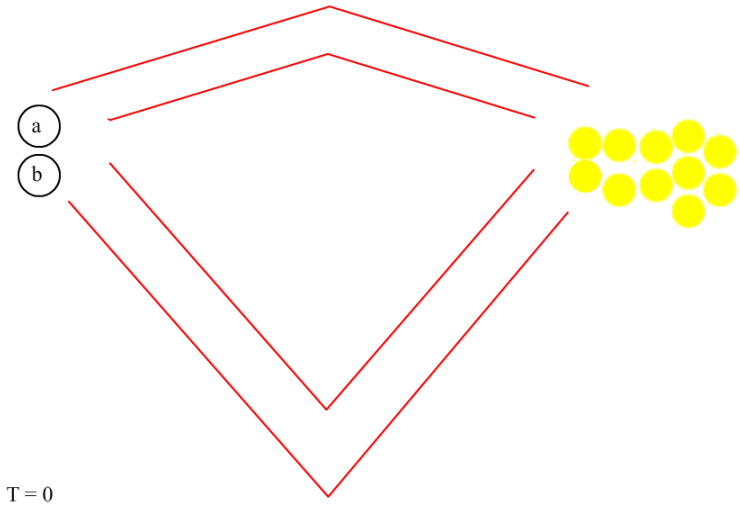


Figure 3: The Double Bridge Experiment - Initial Setup

Figure 4 shows the state of the experiment at T = 1. Ant a is already half way towards the food while ant b is only ¼ of the way there.

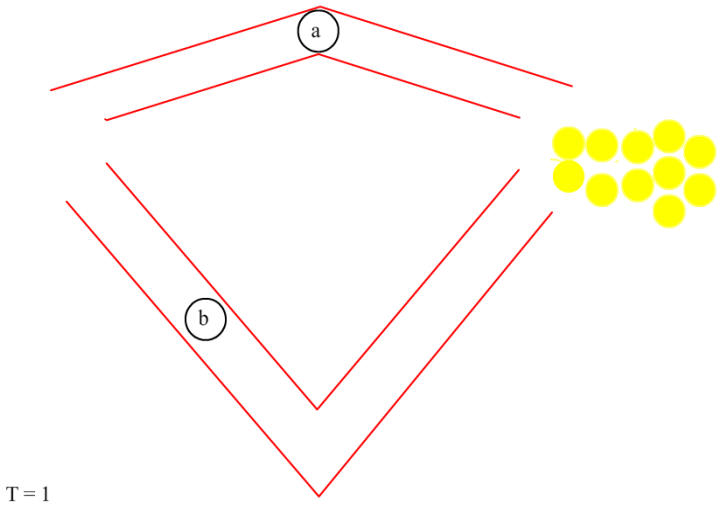


Figure 4: Double Bridge Experiment, Time = 1

At T=3 ant a has started returning to the nest via its route with food (it has 'remembered' its route), while ant b has still not reached the food being only ¾ of the way. Ant a lays a pheromone trail behind itself, to mark its path from the food to the nest.

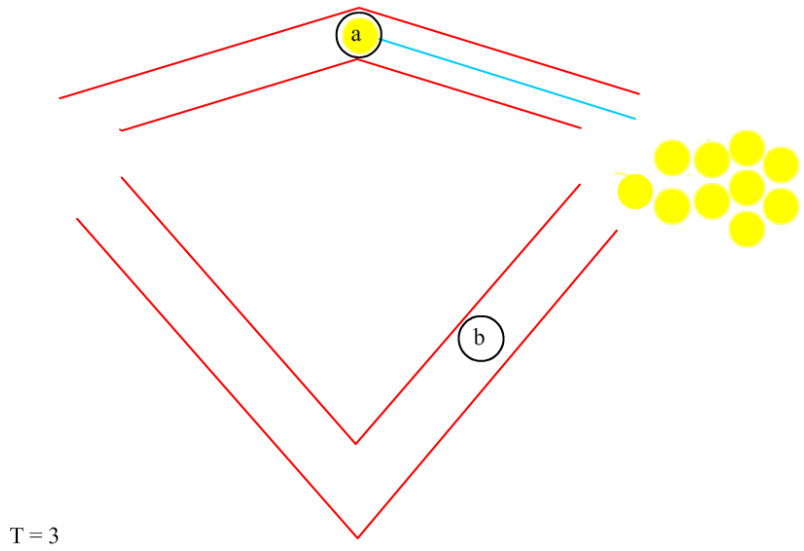


Figure 5: Double Bridge Experiment, Time = 3

Eventually ant b finds the food at T=4. When it gets to the food, it discovers the pheromone trail left by ant a. Ant b now has two choices to return to the nest, one route with pheromone and one without. An ant will always prefer the route marked with pheromone, and will therefore follow ant a's route back to the nest. Ant a will also follow the top route back to the food again for the same reason.

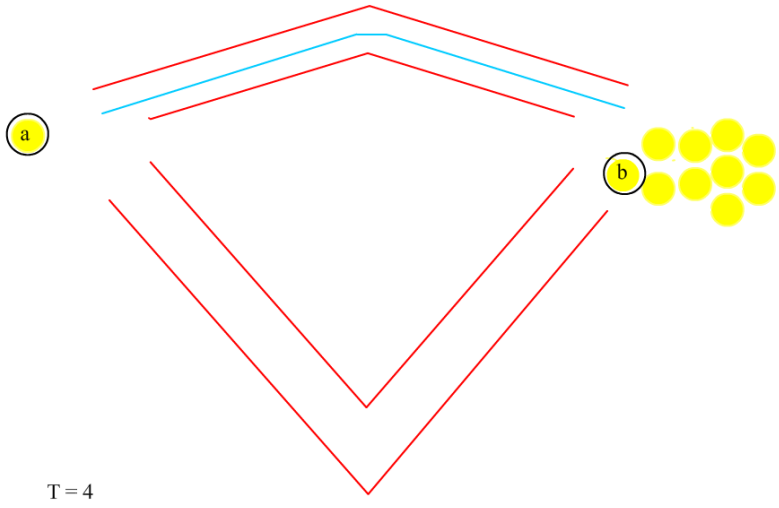


Figure 6: Double Bridge Experiment, Time = 4

Ants a and b will now follow only the top route as it is the one with pheromone. This results in positive feedback where the ants follow the stronger trail, making it stronger still. If an ant c now leaves the nest it will also follow the trail towards the food. When the food is depleted ants will no longer lay a pheromone, and the trail will eventually evaporate.

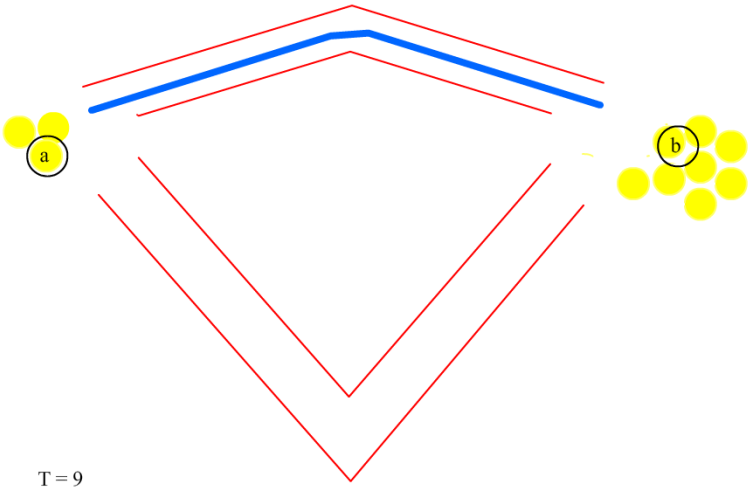


Figure 7: Double Bridge Experiment, Time = 9

The probability that an ant will choose a given path is a function of pheromone strength. If there is no pheromone on either path then the selection is made randomly.

It is entirely possible that all the ants randomly choose the longer route, but at the start both routes have equal probability of being chosen and we can assume that half the ants will choose route A and half route B.

Ants are also very good at adapting to changes in their environment [7] such as a route becoming obstructed, for example if route A suddenly became blocked.

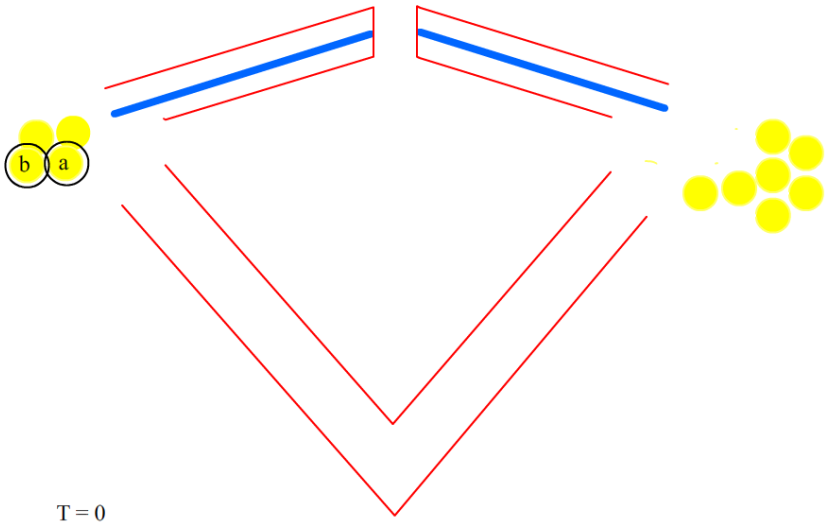


Figure 8: Double Bridge Experiment, Route Blocked

Figure 8 depicts route A being blocked by some obstacle and both ants a and b at the nest ready to set out to find more food. They will both choose route A which has been blocked due to the high concentration of pheromone. The will find themselves confronted by a dead-end (Figure 9), and will turn around to return to the nest, where they will again follow route A due to the pheromone. But the pheromone is not being re-inforced anymore since no ants carrying food at walking the route.

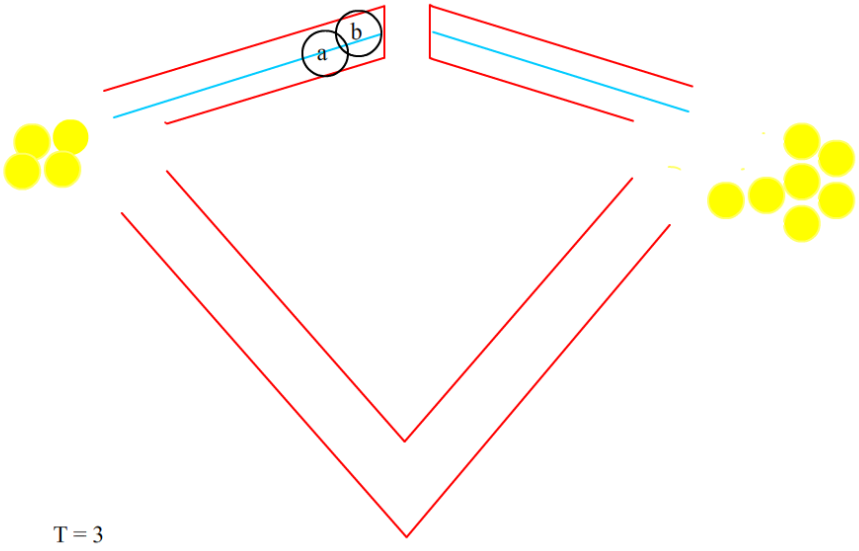


Figure 9: Double Bridge Experiment, Ants a and b at a dead-end

By T=6 the pheromone has evaporated leaving both routes with equal chance of being used to find the food. Again we can assume that one travels route A and the other travels route B. The ant travelling route A will find a dead-end whereas the ant travelling route B will find the food, and lay a trail back to the nest. Both ants will then eventually follow route B. (Ant a may also choose route B before and b returns due to the longer distance b will have to travel).

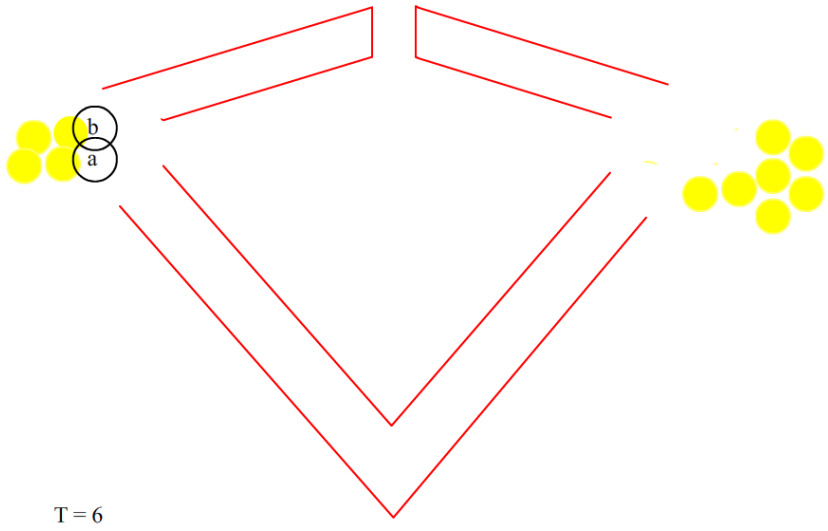


Figure 10: Pheromone has evaporated in the blocked Double Bridge Experiment.

Swarm Intelligence

> Swarm intelligence (SI) is an artificial intelligence technique based around the study of collective behaviour in decentralized, self-organized systems.
>
> Wikipedia [8]

Swarm Intelligence involves many simple separate objects working as one, each object on its own would be quite useless, but when the swarm works together they can solve seemingly complex problems. Each object, or agent, interacts with its local neighbours and environment and complex behaviours emerge. SI can be found in nature, for example ants, birds and fish.

Ant Colony Optimization

Ant Colony Optimization (ACO) is a sub-field of Swarm Intelligence that involves using behaviour observed in ants applied to optimization problems in computing. This field was created by Marco Dorigo, whose work will be referenced in this report.

Applications of ant behaviour include shortest path and travelling salesman graph problems. These can also lead to optimization of networking routing, and dynamic network routing based on network load in applications such as telecommunications.

Peer-to-Peer Networking

Ant based routing algorithms are becoming more widely spread in there uses. Peer-to-peer networking is a good example of a field where ant algorithms are becoming useful. With the growing number of users of peer-to-peer networks, many people are wanting more privacy. Mainstream peer-to-peer applications establish a direct connection between two users exchanging parts of a file, which means they must know each others IP address. An authoritative body such as the RIAA are using IP addresses to sue file uploaders, therefore more people are wanting anonymous ways of sending data via peer-to-peer applications. Applications such as MUTE are attempting to address the privacy issue by providing routing information throughout the network in the form of 'pheromone' clues. Data is routed around the network via the use of these clues, and every node only knows its immediate neighbours, and also cannot determine which node a message has come from or which node it is going to.

Ad-Hoc Networking

The same ideas used for peer-to-peer networking can be used for ad-hoc network [9], for example in a wireless network. Ants are good at finding new routes if other routes have become blocked, for example [7]. In an ad-hoc network computers can join or leave the network at any time, there routes become blocked and new routes become available.

Part 1: Simulating Ant Behaviour

Ant World

The first part of this report concerns simulating the behaviour of ants in order to show how they interact and work collectively.

The ants will live in a grid where each square will contain either the nest, an ant, a piece of food, an obstacle and/or pheromone. Figure 11 shows an example ant world which contains a nest, two ants, four obstacles, and three pieces of food.

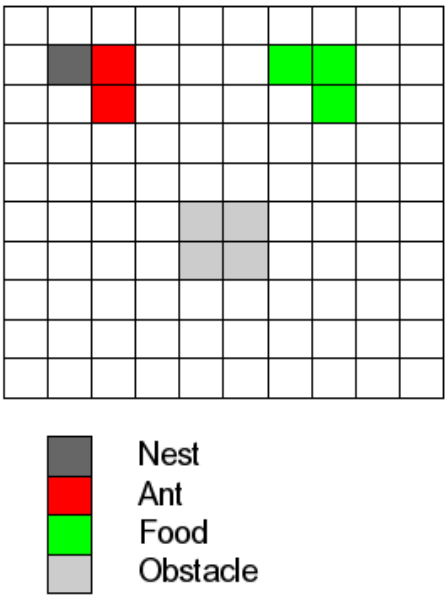


Figure 11: Example 10 x 10 Ant World

Ant Movement

Using a grid to represent the ant world means that an ant will have 8 directions to move in.

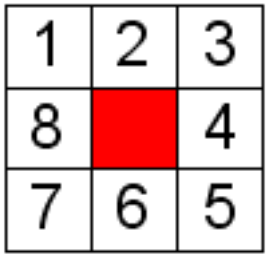


Figure 12: Directions an ant can move

Ants will select the square that is best for them based on a algorithm that will take into account food, pheromone, other ants and obstacles.

If all surrounding grid squares are empty then the ant should continue in the same direction. If the only objects within the grid squares are obstacles then these shouldn't affect the ant, unless there is an obstacle in the square in front of it. So if the ant encounters an obstacle in the direction of its next move it should turn around and move the other way (the same goes for another ant – which should be treated as an obstacle). Figures 13, 14 and 15 show examples of the the expected movement of a single ant.

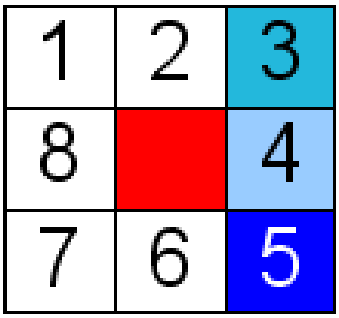


Figure 13: Example: The grid squares contain different strengths of pheromone with 5 being the strongest. The ant should choose square 5.

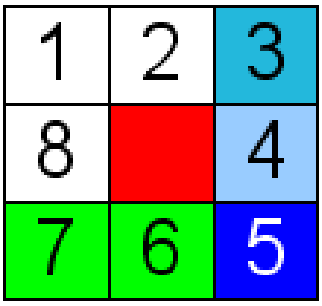


Figure 14: Example: The grid squares 3, 4 and 5 again contain pheromone, but squares 6 and 7 contain food. The ant should choose either of these.

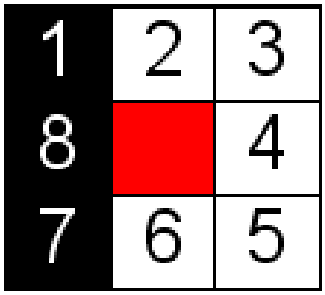


Figure 15: Example: The grid squares 1, 8 and 7 contain an obstacle. The ant should choose grid square 4 to move away from it.

Proposed Software Architecture

Using a Object-Oriented approach the Ant World will consist of GridSquare, Ant, Food, Nest, and Obstacle objects. Obstacles, Ants and Food will be subclasses of MyObject which will control the drawing of these.

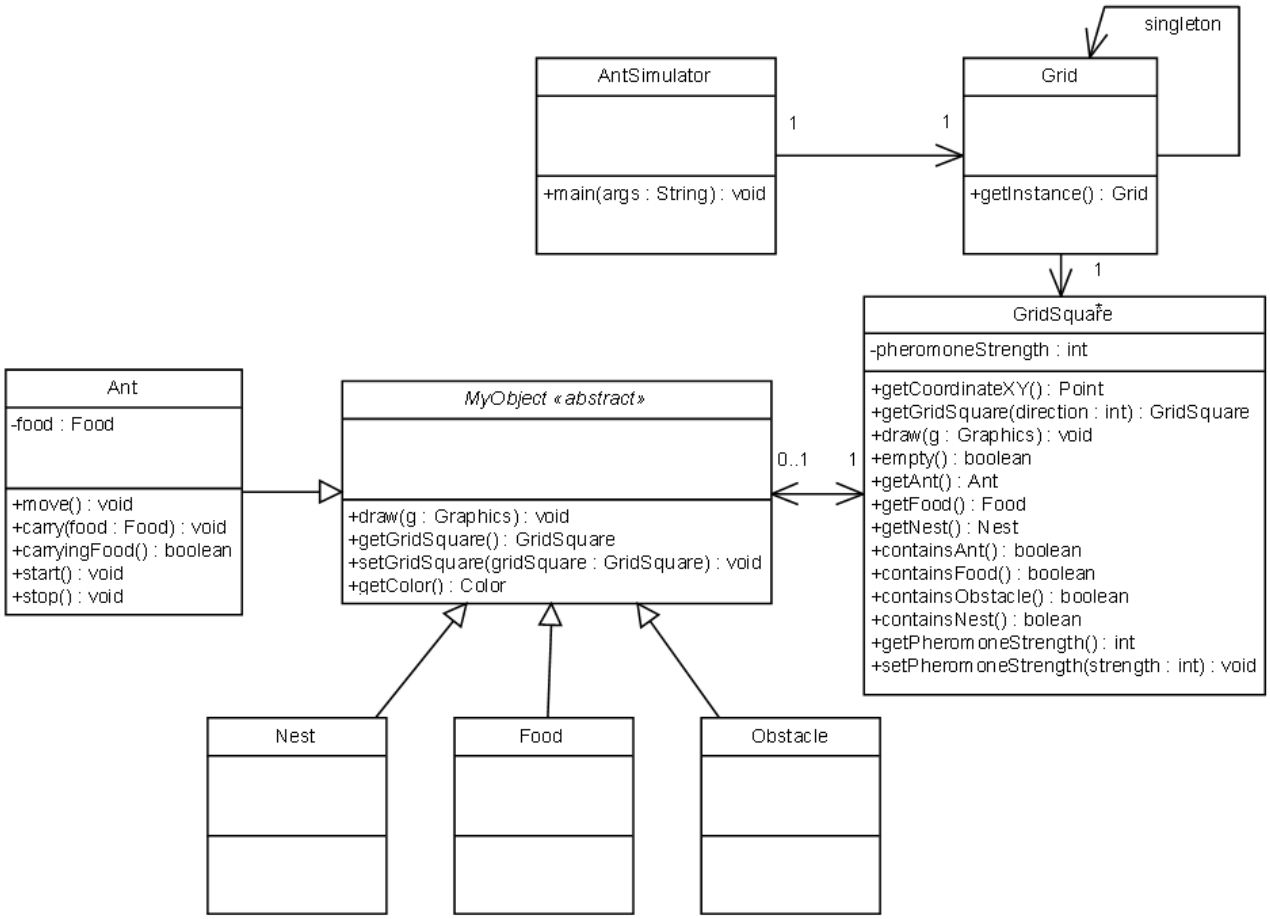


Figure 16: Ant World Simulator Proposed Class Diagram

Technical Considerations

Each ant will be threaded so that they can act individually. Each ant should follow some basic rules which govern its behaviour and should not pass messages to any other ants. This simple behaviour of each ant is expected to produce emergent behaviour, when ants interact.

A GridSquare should be capable of holding either an ant, food, or obstacle. It is proposed that each of these derive from a parent class MyObject. A GridSquare will then hold a MyObject object. Each grid square will also have a pheromone strength.

The MyObject class will be in charge of drawing, and the subclasses will take care of the specialised task of ants, food and obstacles.

A Grid object will contain a multidimensional array of GridSquare objects, that represent the Ant World. A Grid object will be a singleton class as only one of these is (and should) be needed. It will contain a multidimensional array of Grid objects.

Ant World Simulator

Version 1

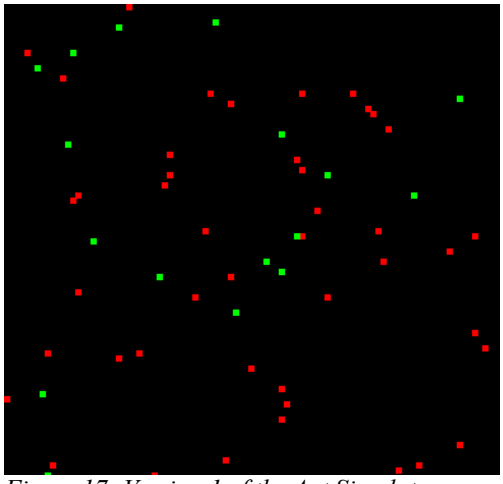


Figure 17: Version 1 of the Ant Simulator

Version 1 is a very simple starting point, which adds the basic ants,food and gridsquares.

Ants walk around moving between GridSquares. They begin at a random gridsquare facing in a random direction and move in the same direction until they hit an object (an ant is an object in this verison - food is ignored) and the turn right.

If they hit the edge of the Grid they turn in the opposite direction.

```
nextGridSquare = gridSquareInCurrentDirection

if(nextGridSquare is null) {
        change direction to opposite direction
        (wait)
}else if(nextGridSquare is Empty) {
        move to it
}else if(nextGridSquare contains Food) {
        pick up food
        move to it
}else{
        change direction clockwise
        (wait)
}
```

Listing 1: Version 1 movement Algorithm (Note: there is a bug (other than the ants!) in this version, and ants ignore the food instead of picking it up as the algorithm suggests.)

Version 2

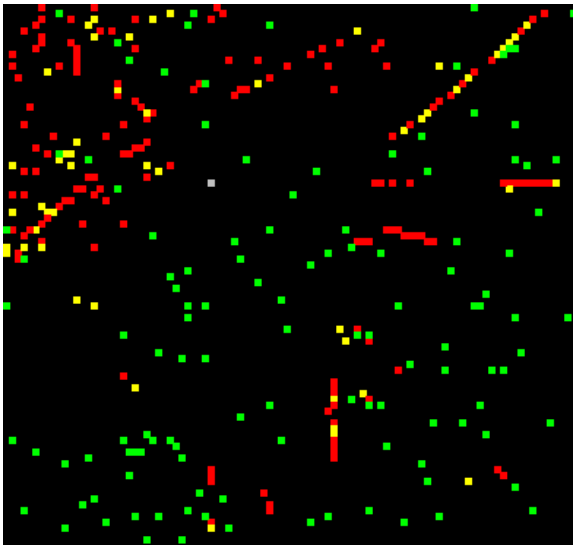


Figure 18: Version 2 of the Ant Simulator

In this version the ants start in a nest. They exit the next in a random direction. 1/8th of the ants tend to go in each direction, spreading outward.

If an ant encounters food it picks it up and continues moving, but now treats other food as an obstacle. A distinction is made between ants carrying food and those that are looking for food, by colouring the carrying ants yellow.

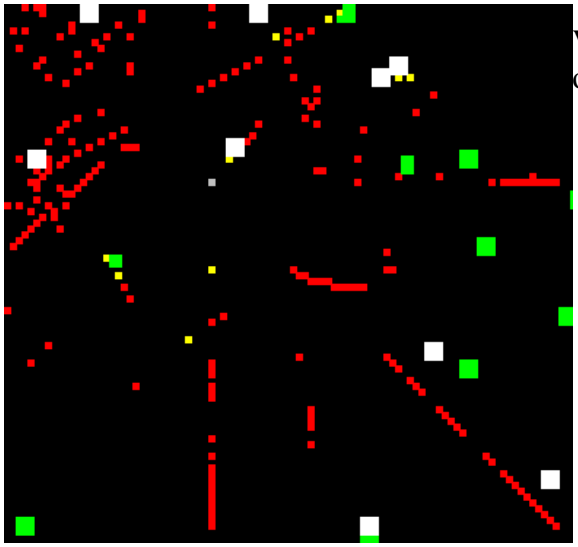
If an ant happens to find the nest it will drop its food and continue searching for more. I say 'happens' because each ant in this version has no idea where their nest is, they are walking around randomly even when carrying food.

```
nextGridSquare = gridSquareInCurrentDirection

if(nextGridSquare is null) {
        change direction to opposite direction
        (wait)
}else if(nextGridSquare contains Food  && not carrying Food) {
        pick up food
        move to it
}else if(nextGridSquare is Nest  && carrying Food) {
        move to it
        drop food
}else if(nextGridSquare is Empty) {
        move to it
}else{
        change direction clockwise
        (wait)
}
```

Listing 2: Version 2 movement algorithm

Version 3



Version 3 is exactly the same as 2, but with a different obstacle and food layout.

Figure 19: Version 3 - Different layout

Version 4

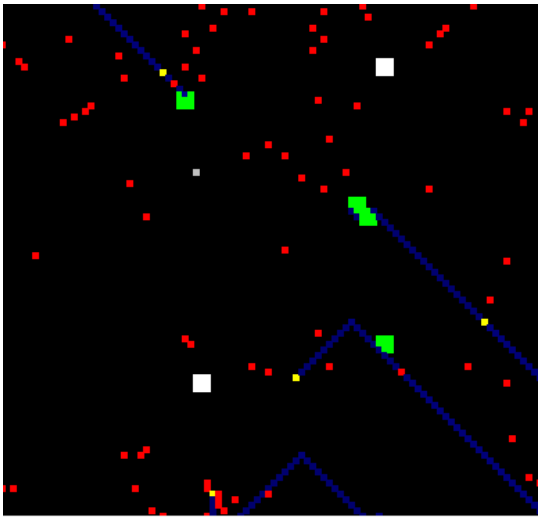


Figure 20: Version 4 - First version with Pheromone trails

Version 4 is the first version to introduce pheromone trails. Ants behave exactly as in version 3. bit when they are carrying food they lay a trail behind them.

Each time an ant enters a grid square while carrying food it increases that grid squares pheromone strength.

Each grid square has its own pheromone object, which acts as a particle of pheromone left by the ant.

The strength of a pheromone particle is an integer between 0 and 10, where 0 is no pheromone and 10 is the strongest it can be. 10 is the upper threshold so any further increases will just keep it at the same level.

Pheromone objects are threaded so they evaporate by themselves. Every 15 seconds the strength of a particle decreases by one, when it reaches zero the pheromone disappears from the grid square.

If the same path is constantly used by ants the trail won't disappear because the strength keeps getting increased. The colour of a pheromone particle is based on its strength, the stronger it becomes the lighter the colour becomes.

In this version ants do not yet follow trails that they find, they just lay a trail behind them while carrying food.

As with the previous version ants do not know where the nest is, though if they do happen to find it they will drop their food.

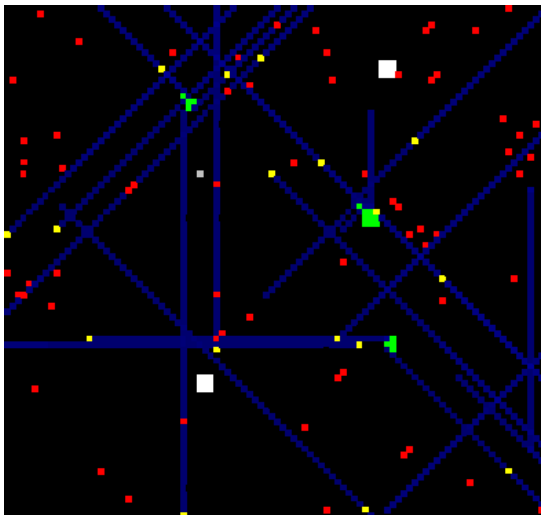


Figure 21: Version 4 - Stronger, Longer trails

```
nextGridSquare = gridSquareInCurrentDirection

if(carrying food) {
        if(current grid square has pheromone) {
                increase pheromone strength by 1
        }else{
                make new pheromone particle with strength 1
        }
}

if(nextGridSquare is null) {
        change direction to opposite direction
        (wait)
}else if(nextGridSquare contains Food  && not carrying Food) {
        pick up food
        move to it
}else if(nextGridSquare is Nest  && carrying Food) {
        move to it
        drop food
}else if(nextGridSquare is Empty) {
        move to it
}else{
        change direction clockwise
        (wait)
}
```

Listing 3: Version 4 movement algorithm

Version 5

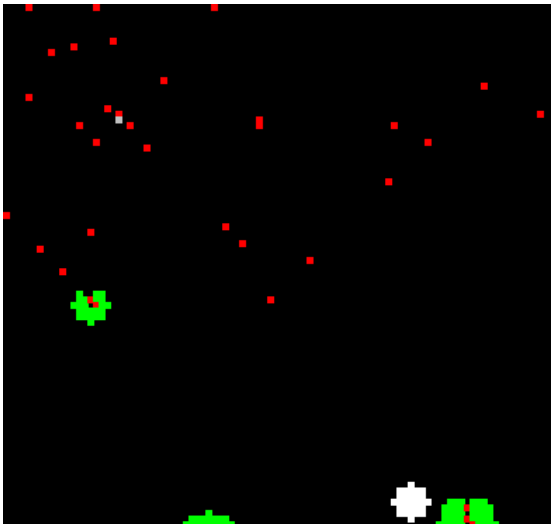


Figure 22: Version 5 - Ants moving towards food and getting trapped.

Version 5 introduces a new paradigm in the use of the grid square objects, but introducing a method Vector<GridSquare> getGridSquares(int radius). This is useful because an ant needs to look at the surrounding grid squares to make a decision about where to go next.

Other uses of this new method include creating clusters of food by picking some random gridsquare, and placing food in a certain radius of that gridsquare.

This version also introduces Food Strength. This is similar to pheromone and is designed to attract ants towards the food.

Food strength is calculated for each grid square within a given radius of a food square, and is inversely proportional to the distance from the food (see figure 23). The food strengths are initial set at the beginning of the program when each cluster of food is added – they would not yet change if an ant was to remove some food.

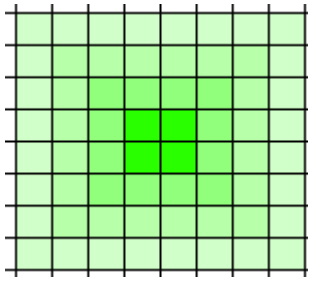


Figure 23: Food Strength - 4 Squares of Food in the middle with food strength spreading outward

This leads to the other problem with this version – the ants dont collect food, the all move toward the centres of food clusters.

This was the intended result of this version, to set up the radial gridsquare method and implement food strength. This can be seen to be working as all the ants move towards centres of the food, which the place where the food strength will be strongest.

In this version each ant looks at the squares surrounding it, and sorts them based on food strength value, so the highest is the first. The ant then moves to this grid square.

The nest is this version is now threaded and in charge of releasing ants. At the beginning of the program all the ants are in the nest, and an ant is released every second. It is remove from the nest's list of ants, and the ants thread is started so that it starts moving. When an ant returns to the nest it is re-added to the list of ants, and its food is removed from it. The ant will then be released again.

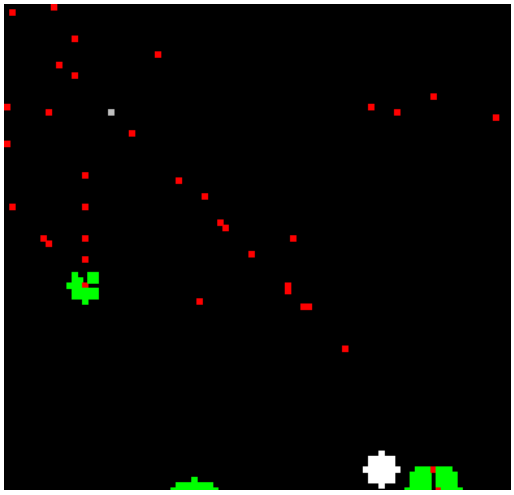


Figure 24: Version 5 - More ants getting trapped in food.

```
if(current grid square is not null) {

        if(current grid square contains an Ant) {
                (wait)
        }else if(current grid square contains food && not carrying food) {
                carry food
                current direction = opposite direction
                (wait)
        }else if(current grid square contains food) {
                current direction = opposite direction
                (wait)
        }

        sort surrounding squares
        if(no food strength) {
                move to next grid square in current direction
        }else{
                move to highest food strength grid square
        }

}else{

        current direction = opposite direction
        move back to previous grid square
}
```

Listing 4: Version 5 movement algorithm

Versions 6 and 7

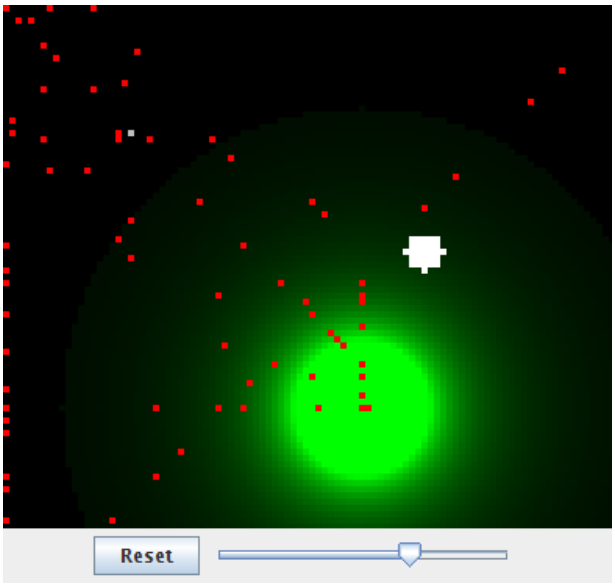


Figure 25: Version 7 - With slider control for food strength radii

Versions 6 and 7 implement a slider control for food strength radii (version 7 is just a bug fix for version 6 – where food strength radii increased by itself).

Using the slider can demonstrate how far the radius of a food cluster extends and ants will be attracted to it as soon as they 'smell' it.

This version also introduces the FoodCluster class, this class is in charge of a cluster of Food objects, recalculating their strengths every second. The FoodCluster class is threaded and every second it recalculates the strengths for all the squares in a radius of 5 from the FoodCluster.

After implementing the FoodCluster class it become apparent that this was a bad idea. A cluster of food should be made up of separate food particles, but there is no need to have a class to look after them.

The recalculations every second also use a lot of processing time and slow the whole program down. There isn't a need for the FoodCluster class because each grid square could calculate its own food strength. Food strength only needs to be updated when a piece of food is removed, rather than every second, as ants might not have taken any food.

The ant movement algorithm is the same in these versions as version 5.

Version 8

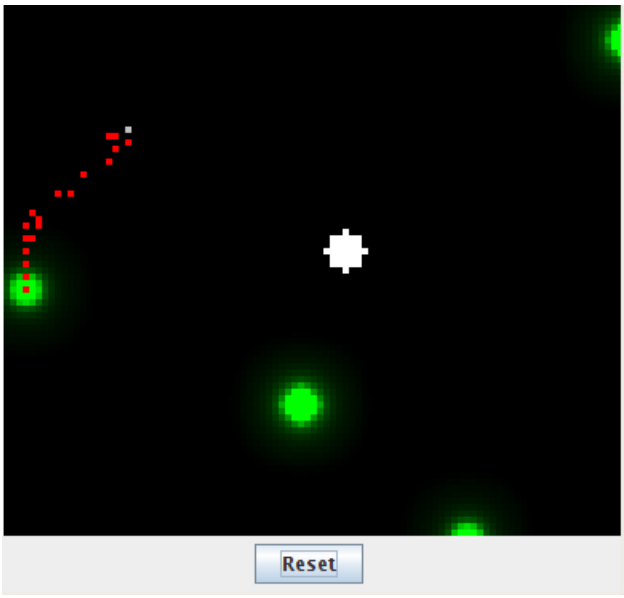


Figure 26: Version 8 - Ants move toward food

Version 8 removes the FoodCluster class and food strength is now based upon each individual food particle.

Each grid square gets checks whether it is within a radius of 10 of a piece of food, and if so updates its food strength accordingly.

Food strength does not get updated when food is taken. This is because the method for calculating food strength is very inefficient, and requires too much processing to be updated each time a piece of food is taken. Therefore food strength is only set at the beginning of the program, and does not change.

The slider was removed due to the changing of the food strength calculations.

Ants move towards the strongest food strength to the middle of the food as in the previous versions, they do not pick up food and return to the nest.

In this version two ants can occupy a grid square at the same time which is not good as it allows all the ants to move to the same square in the middle of the food.

In previous versions ants would have to check to see if it received a null value when asking for the next square to check if it had hit an edge. Obstacles are used in this version to simplify the edge detection. As ants bounce off of obstacles already, it was just a matter of placing a border of obstacles around the edge of the grid – like a fence.

Another problem encountered is 'The Dancing Ant Problem' – See section 'The Dancing Ant Problem'.

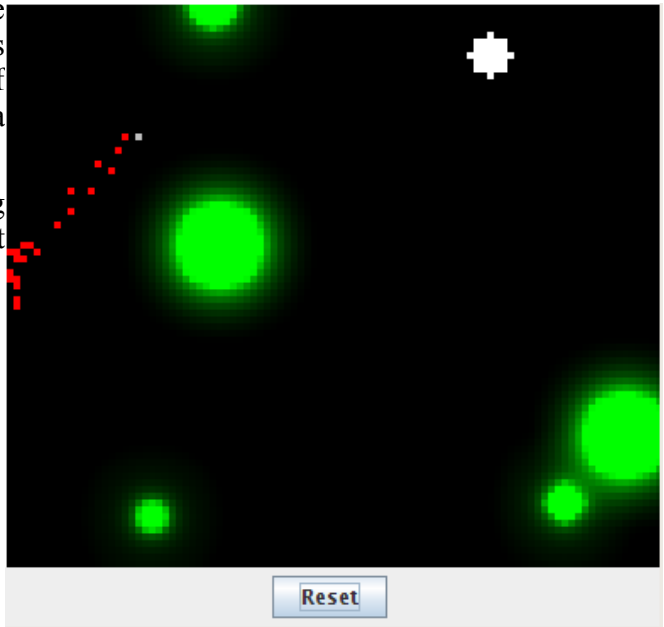


Figure 27: Version 8 - The Dancing Ant Problem

```
if(current grid square is Nest) {
        pick a random square from the surrounding squares
}else{
        Sort the surrounding squares
        foreach(square in surrounding squares) {
                if(square is obstacle) {
                        move to grid square in previous direction
                }else{
                        change direction to direction of new grid square
                        move to it
                }
        }
}
```

Listing 5: Version 8 movement algorithm

The Dancing Ant Problem

The dancing ant problem occurs when an ant's best choice for a next move is a square that it has just moved from. This can occur when it hits an obstacle, because it hits the obstacle turns around and moves back again because it is the 'best' choice.

In Figure 29 the ant's (red square) best choice is to move into position 8 because of the pheromone deposited there.

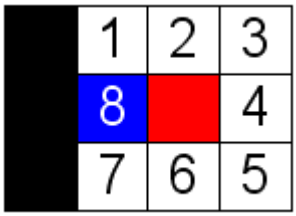


Figure 29: Dancing Ant Problem - State 1

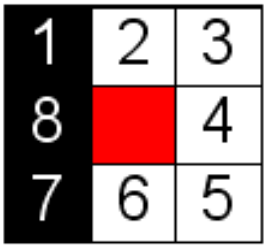


Figure 28: Dancing Ant Problem - State 2

In Figure 28 the ant has moved into the square with the pheromone but has hit an obstacle (black squares). My algorithm now tells the ant to turn in the opposite direction. The leads it to the same state as in Figure 29, and the ant therefore continues the process forever.

Version 9

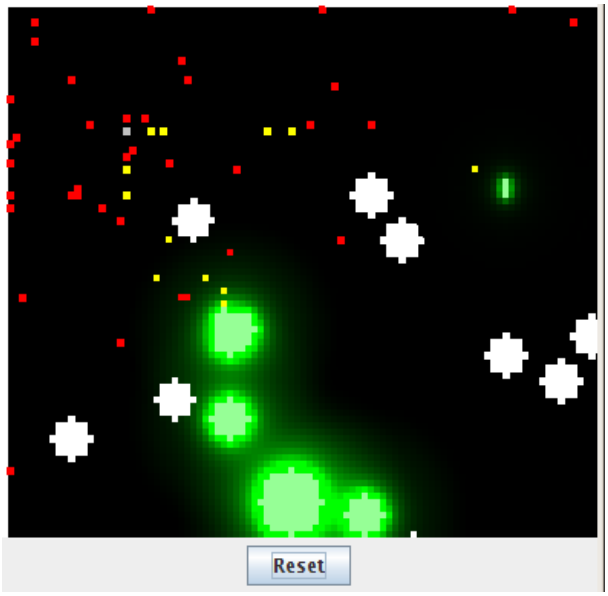


Figure 30: Version 9 - Ants collecting food.

Version 9 is a step forward in the sense that ants finally find food and return it to the nest.

Ants 'know' where their nest, some species use visual clues [10], others use geometrical clues in their trails [5], but with all ants they know how to return to their nest if they are not too far away from it. So in this version once an ant has picked up a piece of food it will head straight back towards the nest. It does this by always moving to the next grid square in the direction of the nest, until it reaches the nest.

The way food strength is calculated has also been changed, due to the in-efficient technique used in the previous version. In this version only grid squares containing food perform food strength calculations, compared to the previous version where all grid squares calculated food strengths every second. This version sets up the initial food strengths at the beginning, then when a piece of food is removed from a grid square, that grid square subtracts strength from the surrounding squares.

A grid square containing food adds food strength to all grid squares within a radius of 15. It does not recalculate the complete food strength for each surrounding square, just add strength for the one piece of food contained in the calculator grid square. This technique allows food strength to be built up by different food particles. When a food particle is removed the grid square removes the same amount of food strength that it previously added to all the surrounding squares, in order to reduce the food strength. This can be seen visually in this version, with the food strength slowly fading away as more food is taken. This technique is far more efficient than in the previous versions and does not need to be recalculated every second, only when food concentration changes.

The new food strength technique is not yet perfect. Ants dance around the outer most edge of the food strength perimeter once all food in a cluster has been taken. This is due to a very small amount of food strength being left behind.

Ants occasionally enter obstacles in this version as well, due to the obstacle square being the 'best' choice for it to take which it obviously isn't.

A third problem is that ants aren't treating food as obstacles when they are carrying food, and therefore they will walk through food. As each grid square can only contain one object (ant, food or obstacle) this removes the food from the square that the ant has walked over. Because the ant did not pick up the food, the food strength created by that piece of food are not removed.

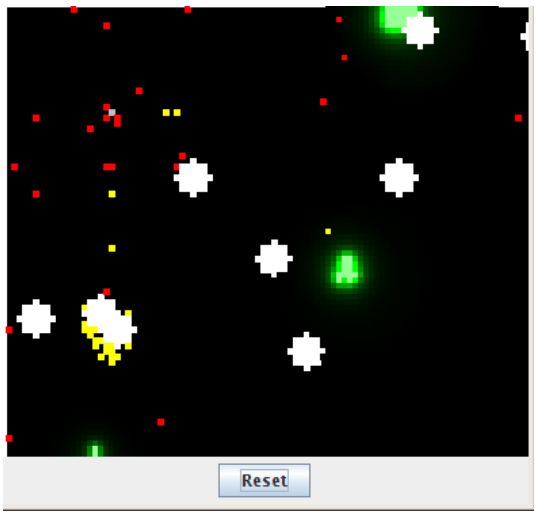


Figure 31: Version 9 - Ants get stuck at obstacles between them and the nest because they 'know' where the nest is and are going in the direction where the obstacle is.

Stop Ants Dancing

In order to stop the ants dancing when they encounter an obstacle, the movement algorithm now randomly selects one of the free squares for the ant to move to. This results in a more natural movement of the ants as before it was a pong-like movement with the ants just bouncing off.

In the actual implementation of this the ants cannot even consider the obstacle squares, they are only given the free squares to choose a new position.

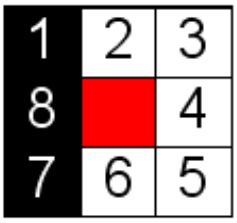


Figure 32: The ant would randomly choose either grid square 2, 3, 4, 5, or 6.

```
if(there is no grid square in current direction or it contains an obstacle or it contains an ant) {
        move to a square in a random direction
}else if(carrying food and not in nest) {
        move to next grid square in the direction of the nest
}else if((surrounding food strength is zero and surrounding squares do not contain food) or carrying food) {
        move to next grid square in current direction
}else{
        if(any free surrounding square contains food) {
                move to it
        }else{
                sort free surrounding squares by food strength
                move to the strongest
        }
}


if(current grid square is nest) {
        if(carrying food) {
                drop it
        }
}else if(next grid square contains food and not carrying food) {
        pick up food
        move to it
}
```

Listing 6: Version 9 movement algorithm

Version 9.1

Version 9.1 fixes the three main bugs in version 9: ants now treat food as obstacles if they are carrying food, ants won't randomly move into an obstacle, and most important food strength does not get left behind.

When remove food strength after removing a piece of food an extra 0.001 is removed to make sure that there is no extra remaining after all the pieces of food in a cluster are removed.

```
if(next grid square in current direction is not empty) {
        move to a square in a random direction
}else if(carrying food and not in nest) {
        move to next grid square in the direction of the nest
}else if((surrounding food strength is zero and surrounding squares do not contain food) or carrying food) {
        move to next grid square in current direction
}else{
        if(any free surrounding square contains food) {
                move to it
        }else{
                sort free surrounding squares by food strength
                move to the strongest
        }
}

if(no next grid square or (next grid square contains obstacle or ant or (food and carrying food)) {
        move to a square in a random direction
}else{
        if(current grid square is nest) {
                if(carrying food) {
                        drop it
                }
        }else if(next grid square contains food and not carrying food) {
                pick up food
                move to it
        }
}
```

Listing 7: Version 9.1 movement algorithm

Version 10

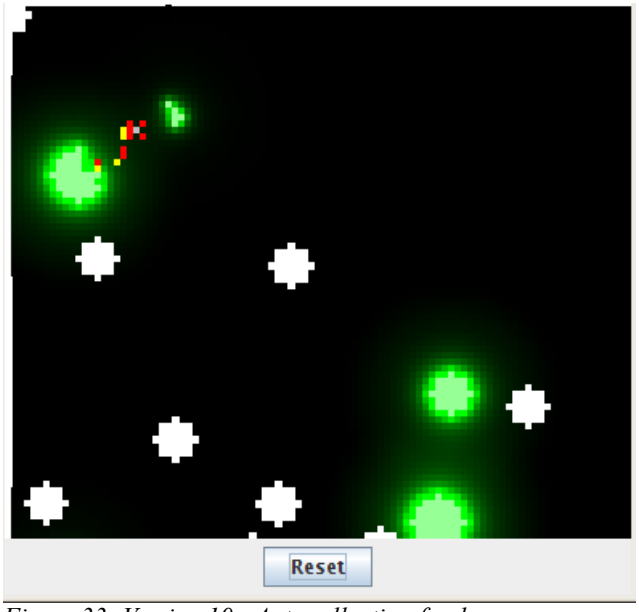


Figure 33: Version 10 - Ants collecting food.

Version 10 introduces the information dialog, the show numerically what is happening in the simulation. Figure 33 shows the graphical representation and figure 34 shows the numerical values associated with it, such as the number of ants carrying food and the number of food particles collected.

This is implemented using an observer pattern, where the information dialog observes the ants by waiting for notifications of events.

It registers itself with each at, and then when an an does something interesting the ant notifies its observers.

The rest of the application is the same version 9.1.

Figure 34: Version 10 - Information dialog

Version 11

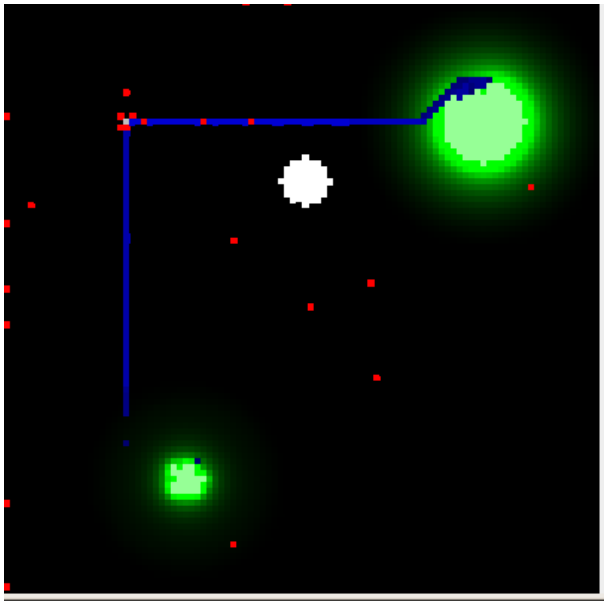


Figure 35: Version 11 - Some ants following pheromone trails

This version implements pheromone trails - although there are problems with getting ants to follow them. When a trail splits into other smaller trails the ants 'best' next grid square is the square behind them because the trial is longer (see 'The Split Trail Problem'). Also ants get knocked off trails if they bump into another ant coming in the other direction (see 'The Bouncing Ant Problem').

Ants sort the grid squares around by pheromone strength and randomly choose one, with a bias towards the highest.

A timer has been added to the information dialog in this version. It takes around the same time for ants to collect the food with or without pheromone trails. Some ants do follow the trail, but must just happen to find the food randomly - the same as with no trails.

Figure 35 shows some ants following one of the pheromone trails, showing the partial success of this version. Ants that have left the trail or left the nest before the trail was laid can also be seen moving around in other places. Some ants do not follow the pheromone as they leave the nest as they should do.

```
if(next grid square in current direction is not empty) {
        move to a square in a random direction
}else if(carrying food and not in nest) {
        move to next grid square in the direction of the nest
}else if((surrounding food strength is zero and surrounding squares do not contain food) or carrying food) {
        move to next grid square in current direction
}else{
        if(any free surrounding square contains food) {
                move to it
        }else{
                sort free surrounding squares by food strength
                move to the strongest
        }
}

if(no next grid square or (next grid square contains obstacle or ant or (food and carrying food)) {
        move to a square in a random direction
}else{
        if(current grid square is nest) {
                if(carrying food) {
                        drop it
                }
        }else if(next grid square contains food and not carrying food) {
                pick up food
                move to it
        }
}
```

Listing 8: Version 11 movement algorithm

The Split Trail Problem

The ant in Figure 36 has following pheromone trail after exiting the nest. When it reaches its current position, it has to choose the next best path to take.

To choose the next best grid square, the ant will look at the 8 surrounding squares and take the one with the highest pheromone trail.

The trails a, b and c are a third of the strength of trail X. This causes the ant to choose grid square 2, because its pheromone strength is 3 times as strong. The means the ant will move backwards.

This results in another instance of the dancing ant problem. When it moves back to square 2 its next best position is again the position in figure 1. And this continues.

Even by selecting the best grid square randomly with a bias towards the highest this problem still seems to occur to much for the ants to follow the trails properly.

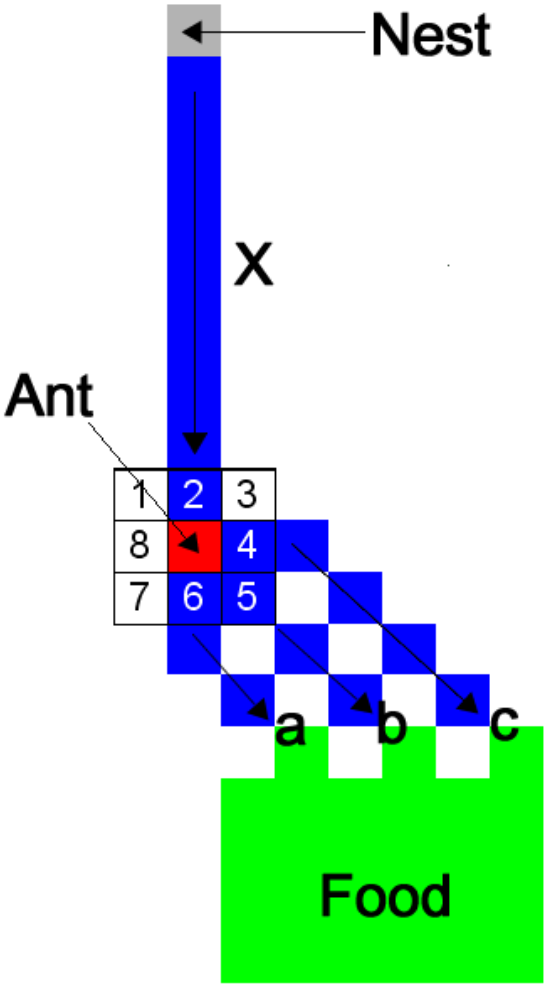


Figure 36: Pheromone Strengths: a = 1, b = 1, c = 1, X = a + b + c = 3

The Bouncing Ant Problem

When an ant is following a pheromone trail and it encounters another ant moving towards it, it randomly picks another direction to move in but this means that it moves off of the trail.

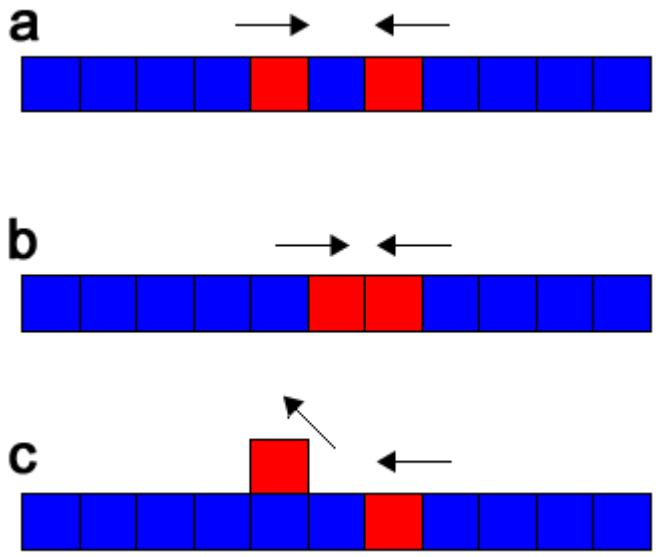


Figure 37:

a: two ants are moving toward each other.

b: ants meet each other.

c: first ant moves out of the way, in a random direction.

Version 12

Version 12 fixes 'The Bouncing Ant Problem' and 'The Split Trail Problem', introduces spreading pheromone and features an improved, if somewhat messy, movement algorithm.

Solving The Split Trail and Bouncing Ant Problems

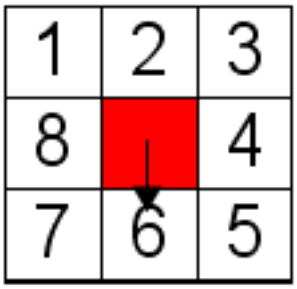


Figure 39: An ant facing south would be able to choose from 8 possible directions (assuming all grid squares around it are empty)

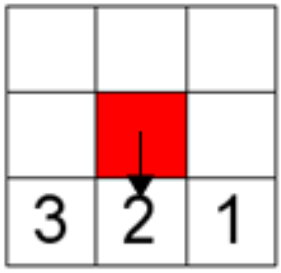


Figure 38: An ant facing south can now only choose the three squares in front of it.

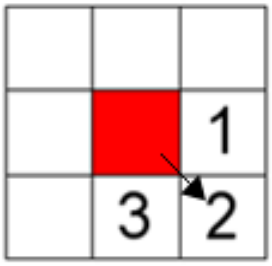


Figure 40: An ant facing south-east can only choose the three squares in front of it

Instead of ants 'seeing' in all 8 directions around them, they will now only be able to see in front of them. This is more realistic as animals usually see in the direction they are facing. This means that the ants will move in the direction they are facing and not move backward to squares that are behind them, even if the pheromone strength is stronger, because they won't be able to see them.

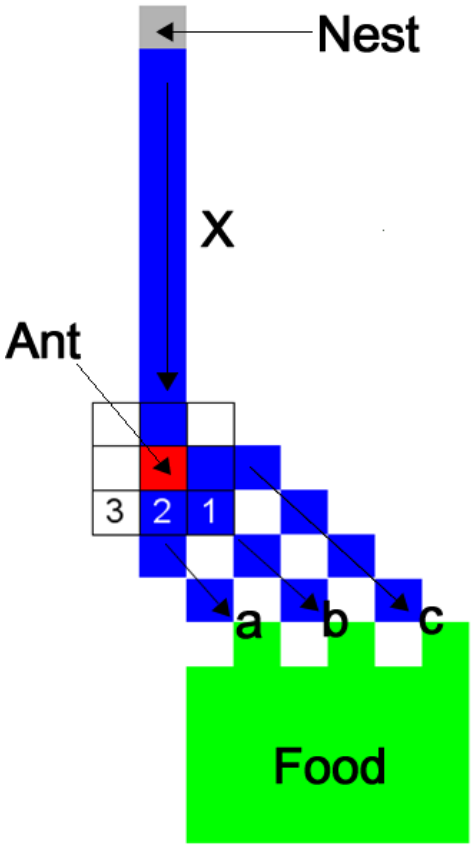


Figure 42: Demonstrates how the problem does not now occur – the ant cannot now choose the grid square behind it, so it must choose one that is in the direction of the food.

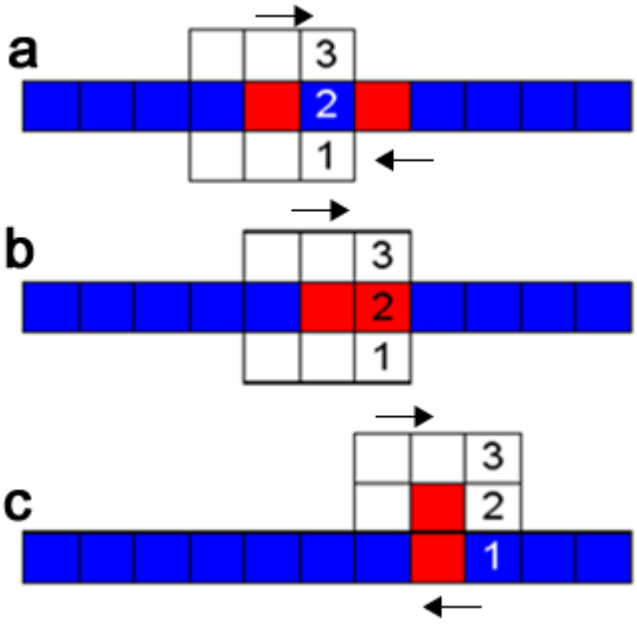


Figure 41:

a: two ants are moving toward each other.

b: ants meet each other. First ant can now only choose a random direction in front of itself. It has the choice of either 3 or 1 as 2 is occupied by the other ant.

c: first ant moves out of the way, in a random direction, it picked 3. Now it has a choice of three move squares in front of itself -it will pick the strongest and will then be back on the trail.

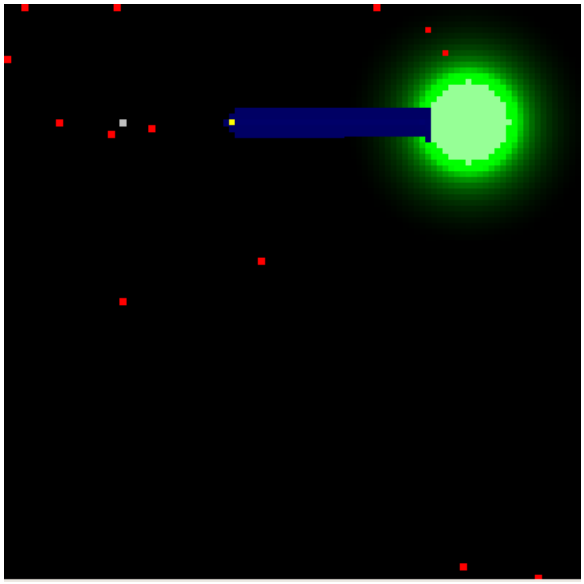


Figure 43: Version 12 - An ant laying a trail back to the nest.

While ants can only see the three grid squares in front of them, they can choose from all eight while they are in the nest, this allows them to start in the best direction.

Inspired by the ant simulator at [11] pheromone trails now spread slightly when they are laid. This will provide a wider area for the ants to find and then follow. It will also allow more ants to follow the trail at the same time.

A radius of two was found to work best, which is shown in Figure 43. The spread pheromone is the same strength as the originating pheromone particle, it is strength should really be governed by an inverse square law, because as something spreads out further it becomes weaker (like the food strength).

The movement algorithm in this version is quite messy due to many additions and changes. It needs re-writing.

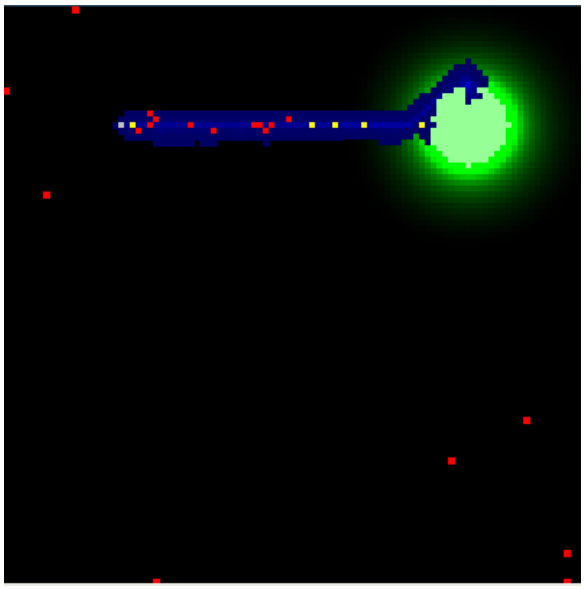


Figure 44: Ants laying and following a pheromone trail to some food

```
List surroundingSquares = free grid squares in current direction

if(in nest) {
        if(there is any pheromone in any direction)
                surroundingSquares = grid squares with pheromone only
}

if(surroundingSquares is empty) {
        surroundingSquares = free grid squares in any direction
}

if(pheromone in surroundingSquares and not carrying food) {
        surroundingSquares = grid squares with pheromone only
}

if(no next grid square or next grid square contains obstacle or (carrying food and next grid square contains food)) {
        if( surroundingSquares is empty) {
                 surroundingSquares = free grid squares in all directions
        }

        move to a random grid square in the surroundingSquares list
}else if(carrying food) {
        if(in nest) {
                drop food
        }else{
                move in current direction
                current direction = new direction of nest
        }
}else if( surroundingSquares contains food strength) {
        sort  surroundingSquares by food strength
        move to a random square biased towards to the strongest
        change direction to new square
}else if( surroundingSquares contains pheromone strength) {
        sort  surroundingSquares by pheromone strength
        if(in nest) {
                move to strongest square
        }else{
                move to a random square biased towards to the strongest
        }
        change direction to new square
}

if(not moving to previous square and carrying food) {
        lay pheromone
}

if(no where to move) {
        pick a random direction
}else if(carrying food and next grid square is nest) {
        go into the nest
}else if(nextgridsquare contains an ant or and obstacle) {
        sort squares by pheromone strength
        move to strongest
}else if(carrying food and next grid square contains food) {
        wait
}
```

Listing 9: Version 12 movement algorithm

Version 13

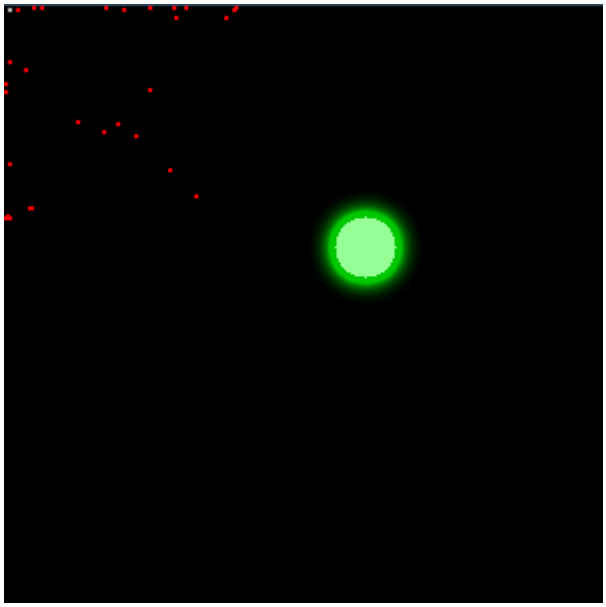


Figure 45: Version 13 - An attempt to create a smoother animation

Version 13 was an attempt to create a smoother animation to make the movements of the ants more life-like. This meant making the grid squares smaller, and having more of them. This wasn't very successful as it slowed the whole program down with the extra calculations, because of more grid squares.

A smoother animation is not the most important part of this application, since it is to demonstrate ant behaviour. It can be slow and still show how ants behave.

Therefore the next version will be built upon version 12.

Version 14

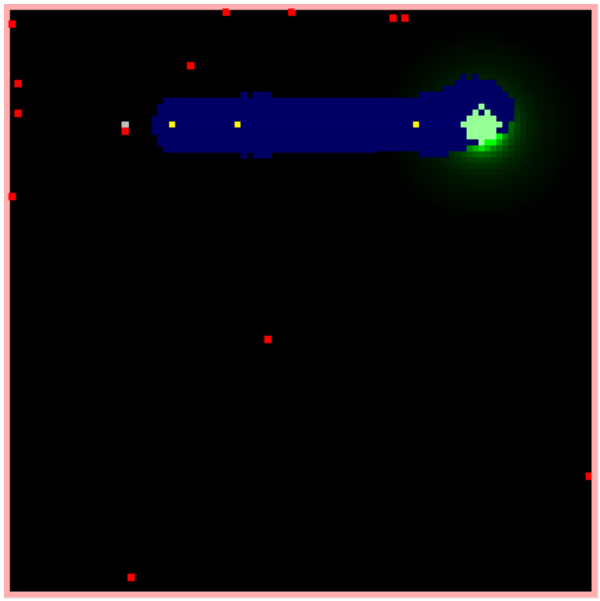


Figure 46: Version 14 - Some ants laying a trail from the food towards the nest

Version 14 includes a re-written move algorithm, and ants follow a pheromone from the nest to the food.

Ants can get stuck behind the food if it is directly between them and the nest, as they are trying to move in the direction of the nest.

Another problem is that because ants move toward the grid square with the highest pheromone strength they sometimes move backward when an ant carrying food passes them, as the ant carrying food has just increased the strength. The trail should be directional so that the ants will go the correct way, [10].

Trail following also only works if at least one ant has found the food and returned to the nest. If most of the ants have already left the nest before this happens, then they will only find the food by randomly stumbling across either the trail or the food.

Two experiments were carried out with this version, the results are shown below. Three conditions were tested: one where ants laid and followed trails but ignored the pheromone strength, one where they laid and followed trails and followed the highest pheromone strength and one where they didn't follow trails at all. The experiments showed that ants using trails and pheromone strength found the food much quicker than with no trails.

Time for ants to collect 29 pieces of food

Trails	Trails - no sort	No Trails
2:03	2:42	3:18
2:55	3:06	2:14
2:09	2:54	2:21
2:17	2:21	2:34
Averages		
2:20	2:46	2:37

Time for ants to collect 81 pieces of food

Trails	Trails - no sort	No Trails
4:09	5:26	5:54
5:01	5:56	6:41
4:17	6:35	5:43
Averages		
4:29	5:59	6:06

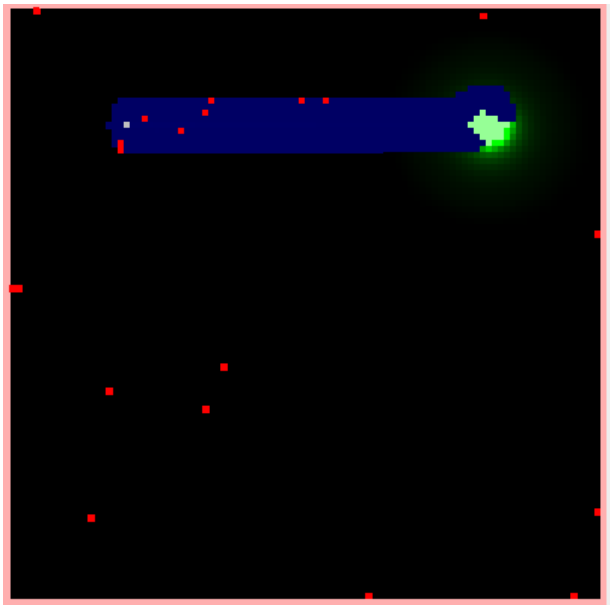


Figure 47: Version 14 - Some hungry ants follow the trail towards food

```
if(in nest) {
        look at all 8 surrounding squares
        if(carrying food) {
                drop food
        }else if(somewhere to move) {
                if(food nearby) {
                        move to it
                }else if(food strength nearby) {
                        move to strongest food strength
                }else if(pheromone nearby) {
                        move to strongest pheromone strength
                }else{
                        move to a random square
                }
                change current direction to new position
        }
}else{
        look at only the 3 squares in front
        if(carrying food) {
                current direction = direction of nest
                ignore food grid squares
        }

        if(nowhere to move in front) {
                look all around
                pick a random direction
        }else{
                if(food nearby and not carrying food) {
                        move to a food square
                        pick up food
                }else if(food strength nearby and not carrying food) {
                        move to grid square with highest food strength
                        current direction = direction of new square
                }else if(pheromone nearby and not carrying food) {
                        move to grid square with highest pheromone strength
                        current direction = direction of new square
                }else{
                        if(carrying food) {
                                move to next square
                        }else if(on trail and no trail in front) {
                                turn around
                        }else if(three free squares in front) {
                                move to middle one
                        }else{
                                move to random free grid square in front
                        }
                }
        }
}

if(carrying food) {
        lay pheromone
}
```

Listing 10: Version 14 movement algorithm

Version 15

This version implements directional pheromone trails. It does not take into account pheromone strength, just the direction of the pheromone.

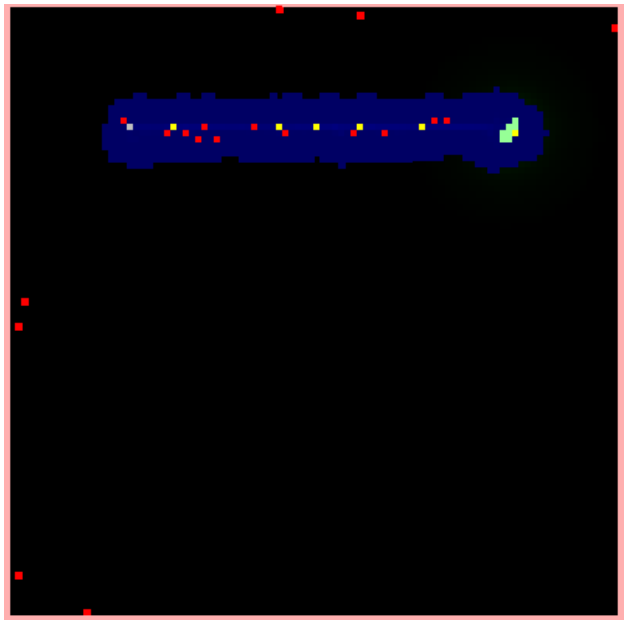


Figure 48: Version 15 - Ants laying and following a pheromone trail

When an ant carries food back to the nest it sets the direction of the trail of each square to the opposite direction to the nest, i.e. towards the food. As the trail spreads out to surrounding squares they cannot point in the opposite direction to the nest, so they point to the square where the pheromone originated. This can been seen in Figure 49 which shows the direction that the pheromone is pointing at each grid square.

An ant will exit the nest and pick the strongest square to move to, it will then pick the next grid square in the direction that its current square is pointing. If an ant has to move off the main part of the pheromone trail it will find itself on a grid square where the trail points back towards the main part of the trail.

A slight problem is found on the left side of the trail in Figure 49 as all three point into the nest.

In this version the pheromone also spreads to a radius of 5 squares from its originating square, to give the ants more room to move along the trail.

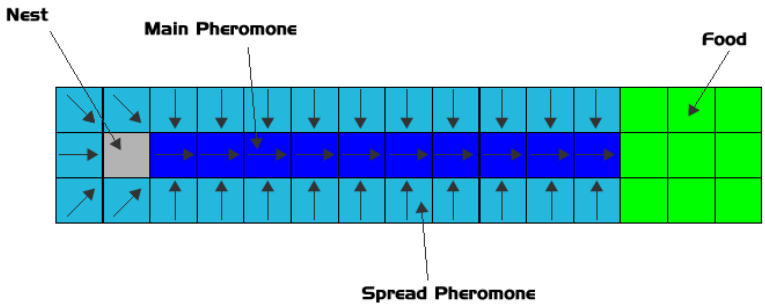


Figure 49: Directional pheromones, arrows represent the direction. There is a direct route to the food from the nest, and the spread pheromone points to the direct route. There is a slight problem with the ones to the left of the nest as they point into the nest.

Time for ants to collect 29 pieces of food

Normail Trails (Version 14)	Directed Trails
2:03	2:01
2:55	2:06
2:09	2:03
2:17	1:45
Averages	
2:20	2:00

```
if(in nest) {
        look at all 8 surrounding squares
        if(carrying food) {
                drop food
        }else if(somewhere to move) {
                if(food nearby) {
                        move to it
                }else if(food strength nearby) {
                        move to strongest food strength
                }else if(pheromone nearby) {
                        move to strongest pheromone strength
                }else{
                        move to a random square
                }
                change current direction to new position
        }
}else{
        look at only the 3 squares in front
        if(carrying food) {
                current direction = direction of nest
                ignore food grid squares
        }else if(pheromone nearby and no food strength nearby) {
                current direction = direction of pheromone
        }

        if(nowhere to move in front) {
                look all around
                pick a random direction
        }else{
                if(food nearby and not carrying food) {
                        move to a food square
                        pick up food
                }else if(food strength nearby and not carrying food) {
                        move to grid square with highest food strength
                        current direction = direction of new square
                }else{
                        if(carrying food) {
                                move to next square
                        }else if(on trail and no trail in front) {
                                turn around
                        }else if(three free squares in front) {
                                move to middle one
                        }else{
                                move to random free grid square in front
                        }
                }
        }
}

if(carrying food) {
        lay pheromone
}
```

Listing 11: Version 15 movement algorithm

Version 15.1

This version uses pheromone direction and strength. This works better as the food is found quicker than with just using directions (see experiment below).

Time for ants to collect 29 pieces of food

Directed + Strength	Directed Trails (Version 15.0)
1:49	2:01
1:49	2:06
1:50	2:03
1:41	1:45
Averages	
1:47	2:00

```
if(in nest) {
        look at all 8 surrounding squares
        if(carrying food) {
                drop food
        }else if(somewhere to move) {
                if(food nearby) {
                        move to it
                }else if(food strength nearby) {
                        move to strongest food strength
                }else if(pheromone nearby) {
                        move to strongest pheromone strength
                }else{
                        move to a random square
                }
                change current direction to new position
        }
}else{
        look at only the 3 squares in front
        if(carrying food) {
                current direction = direction of nest
                ignore food grid squares
        }else if(pheromone nearby and no food strength nearby) {
                current direction = direction of pheromone
        }

        if(nowhere to move in front) {
                look all around
                pick a random direction
        }else{
                if(food nearby and not carrying food) {
                        move to a food square
                        pick up food
                }else if(food strength nearby and not carrying food) {
                        move to grid square with highest food strength
                        current direction = direction of new square
                }else if(pheromone nearby and not carrying food) {
                        move to grid square with highest pheromone strength
                        current direction = direction of new square
                }else{
                        if(carrying food) {
                                move to next square
                        }else if(on trail and no trail in front) {
                                turn around
                        }else if(three free squares in front) {
                                move to middle one
                        }else{
                                move to random free grid square in front
                        }
                }
        }
}

if(carrying food) {
        lay pheromone
}
```

Listing 12: Version 15.1 movement algorithm

Version 16

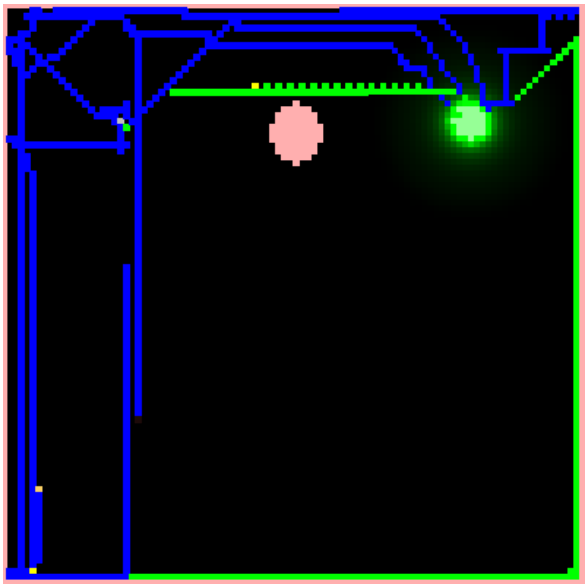


Figure 50: Version 16 - Showing two kinds of trail: blue trails lead away from the nest and green trails (hopefully) lead towards the nest. This is not the case in this example though, there is some improvement to be made.

This version takes a different approach to dealing with pheromone trails. I wanted to eliminate the need for ants to 'know' where their nest is - apart from 'cheating', this also caused problems when obstacles where placed in the way because the ants know that the nest is in the direction towards the other side of the obstacle so they move that way. Ants are also now randomly coloured so that individual ants can be followed easily.

I have based this version on some ideas from the [2] file sharing application (More about MUTE later, in Part 2).

Ants lay a trail all the time. While they are looking for food they lay a blue trail, when they have found food they lay a green trail. The idea is that when an ant finds some food it turns around and follows its own trail back to the nest.

This version introduces a new class - PheromoneParticle. Every ant lays its own pheromone particle as it moves along, over other pheromone particles.

The Pheromone contains a stack of PheromoneParticle's. The direction of the Pheromone is the highest frequency direction in the stack of PheromoneParticle's. Every 60 seconds a PheromoneParticle is removed fromt the bottom of the stack. A PheromoneParticle also knows which ant laid - so that an ant could follow his own trail.

When an ant leaves the nest it lays a blue trail, then when it finds food if there is no green trail to follow back to the nest, it follows its own trail. This works almost exactly as previous versions but whereas the ants just 'knew' where the nest was, in this version the ants use the trail they laid as clues to find the nest again.

I re-wrote the move() method again. Though its not working fully yet, which can be seen in Figure 50 where the green trail should lead back to the nest.

```
if(in nest) {
        look at all 8 surrounding squares
        if(carrying food) {
                drop food
        }else if(somewhere to move) {
                if(food nearby) {
                        move to it
                }else if(food strength nearby) {
                        move to strongest food strength
                }else if(green pheromone nearby) {
                        move to strongest pheromone strength
                }else if(blue pheromone nearby) {
                        move to strongest pheromone strength
                }else{
                        move to a random square
                }
                change current direction to new position
        }
}else{
        look at only the 3 squares in front
        if(carrying food) {
                current direction = direction of nest
                ignore food grid squares
        }else if(pheromone nearby and no food strength nearby) {
                current direction = direction of pheromone
        }

        if(nowhere to move in front) {
                look all around
                pick a random direction
        }else{
                if(food nearby and not carrying food) {
                        move to a food square
                        pick up food
                }else if(food strength nearby and not carrying food) {
                        move to grid square with highest food strength
                        current direction = direction of new square
                }else if(pheromone nearby) {
                        if(carrying food) {
                                if(trail green trail nearby) {
                                        if(own trail) {
                                                follow it
                                        }else{
                                                follow strongest green trail
                                        }
                                }else if(blue trail nearby) {
                                        follow strongest blue trail
                                }
                                current direction = direction of new square
                        }else{
                                if(blue trail nearby) {
                                        follow strongest blue trail
                                }else{
                                        move to next square in current direction
                                }
                        }
                }else{
                        if(carrying food) {
                                move to next square
                        }else if(on trail and no trail in front) {
                                turn around
                        }else if(three free squares in front) {
                                move to middle one
                        }else{
                                move to random free grid square in front
                        }
                }
        }
}

if(carrying food) {
        lay pheromone
}
```

Listing 13: Version 16 movement algorithm

Version 17.1

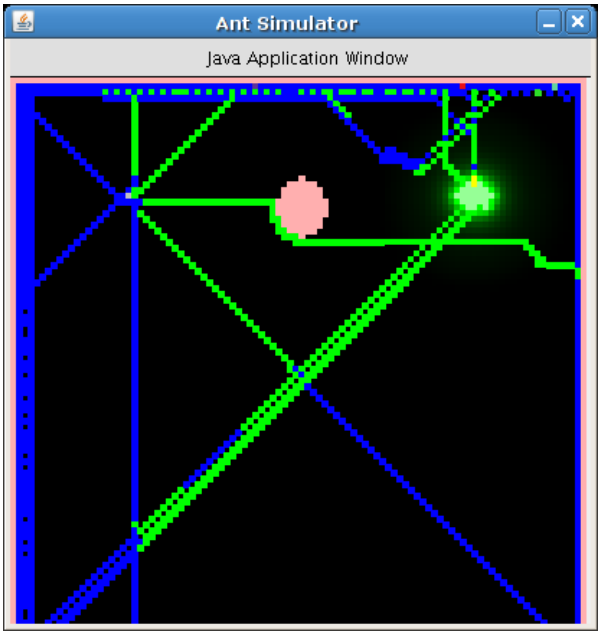


Figure 51: Version 17 - ants laying and following trails

Versions 17 and 17.1 attempt to improve the move algorithm. It has been re-written, though still does not work successfully.

An auxiliary method called getNextGridSquare() is where the main calculations for the next movement happen. It is based on version 16, with modifications.

In this version ants do not follow trails back towards the nest successfully when they find food, this leads to trails all over the grid. Even so, some ants do make it back to the nest.

```
if(in nest) {
        look at all 8 surrounding squares
        if(carrying food) {
                drop food
        }else{
                if(towards nest pheromone nearby) {
                        pick grid square with strongest towards nest pheromone.
                }

                if(no empty square) {
                        pick random square
                }
                change direction to direction of chosen grid square
        }

}else{
        if(there is nowhere to go infront) {
                look behind
        }

        if(there is nowhere to go behind or (onTrail and carryingFood)) {
                look sideways
                change direction after choice, as no squares where available in front
        }

        if(carrying food and nest nearby) {
                chose nest
        }else if(not carrying food and food nearby) {
                chose food square
        }else if(not carrying food and food strength nearby) {
                chose grid square with strongest food strength.
        }else if(carrying food and pheromone strength nearby) {
                if(pheromone towards nest nearby) {
                        choose grid square with strongest toward nest pheromone
                }else if(pheromone away from nest nearby) {
                        choose grid square with strongest away from nest pheromone
                }

                if(no free squares) pick random square biased towards highest pheromone.
        }else if(three free squares in front) {
                pick middle square
        }else if (any free squares in front) {
                pick random square
        }else{
                wait
        }


}
```

Listing 14: Version 16 movement algorithm

Ant Maze

In order to demonstrate ants shortest path finding ability, a maze could be constructed by placing obstacles on the grid, as in Figure 52. In theory it should be possible to place the obstacles in this way and for the ants to treat them as obstacles without any modification.

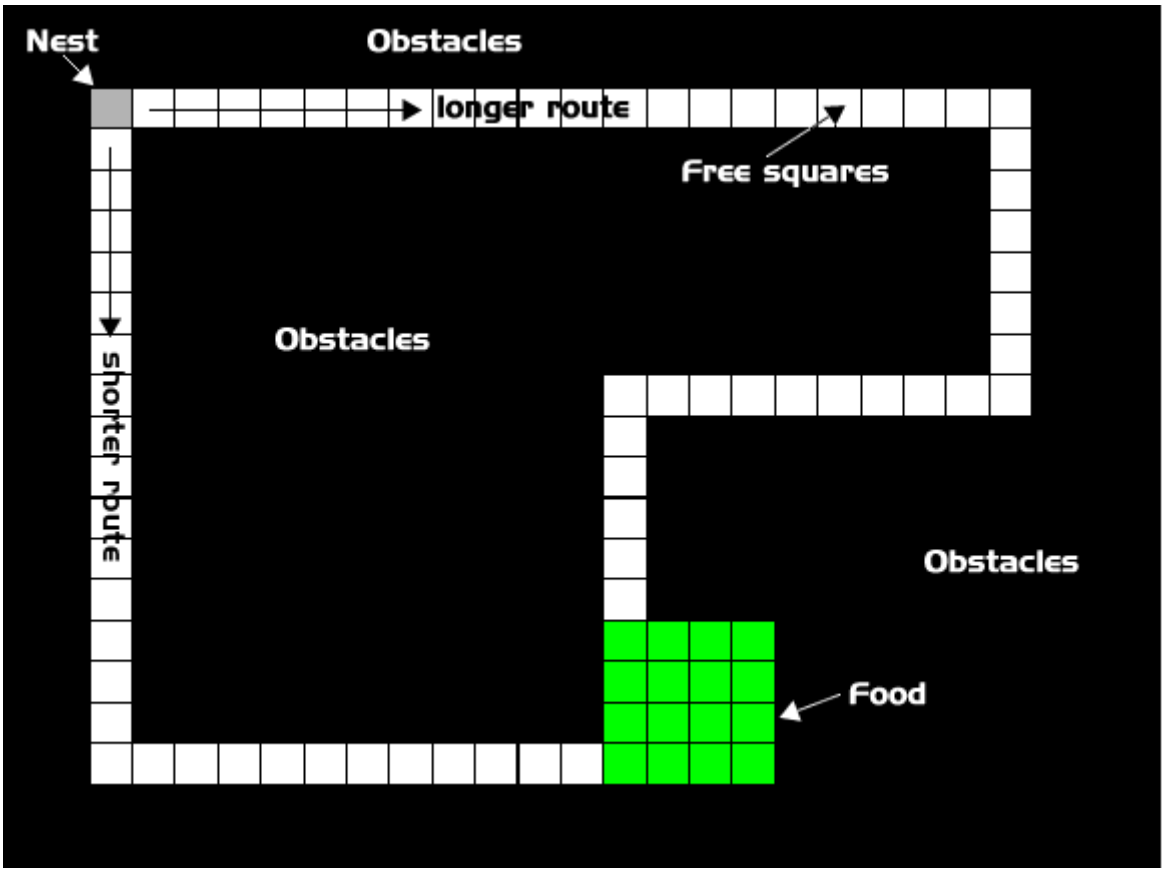


Figure 52: Graph Maze

Conclusion to Part 1

According to the original specification this section of the report aimed to simulate 'the behaviour of ants in order to show how they interact and work collectively'. Partially success was achieved. Ants could be seen to find the food randomly, and lay a trail for other ants to follow. Experiments showed that using pheromone trails meant faster food collection, and although versions 17 was not as successful as hoped due to time constraints, it demonstrates the possibility for further work on the simulator.

Several problems were encountered the main one being ants getting stuck behind obstacles – which was actually to do the ants just 'knowing' where the nest was, version 16 attempted to solve this problem but failed.

Due to the obstacle problem the maze idea will not work properly as it relies on the ants avoiding obstacles. Part 2 introduces an Ant World using a graph data structure, which aims to eliminate the obstacle problem.

Improvements and Future Work

Problems related to obstacles are the most important problem to fix in this application. If ants avoid obstacles well, then many other aspects of the application would work. The theory behind the two pheromone trails in version 16 seems to be correct the implementation is somewhat more complicated. Ants need to be restricted to travelling on trails move, they move away from a trail too easily.

The two trail routing is carried through to part 2 of this report where it will be used to help ants traverse graphs.

Part 2: Applications of Ant Behaviour

The second part of this report is concerned with the applications of ant behaviour, demonstrated in part 1, in the field of computer science. As we have already established ants are extremely good at finding the shortest distance between their nest and a food source. They also work as a collective, working individually but appearing to work as one.

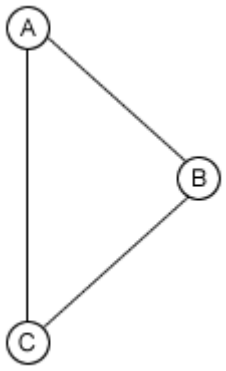


Figure 53: The Double Bridge Experiment Graph

Our Ant World can be represented as an undirected graph, which in turn could represent something such as a computer or telecommunications network. The ants task would now be to start at the nest node, and find the shortest path to the food node. By using a graph to represent the Ant World the obstacle problem of the previous Ant Simulator is instantly removed. The graph represents all the ants possible routes, and as such, obstacles are not part of the graph.

Now going back to 'The Double Bridge Experiment', we can see how the Ant World can be expressed as a graph.

Figure 53 shows a graph that represents this experiment, where A is the nest and C is the food source.

Every time unit an ant will move to another node. Route A -> B -> C is double the length of route A -> C to get from the nest to the food. So this would be the best choice for our virtual ants.

It is assumed that all edge weights are equal, though to be more useful this would need to be extended to use weighted graphs. For example in a telecommunications network weights could represent the actual distance between nodes.

Why ant algorithms?

To answer this let us consider one application of ants behaviour - network routing - more specifiably routing in peer-to-peer (from now on p2p) networks. p2p applications allow a network of users to share files and/or other data. Standard p2p applications provide a direct connection between the computers exchanging data, once it has been established where the needed data is on the network. This means that the uploader and the downloader must therefore known each others IP addresses.

Using ant algorithms to route data through a p2p network will provide anonymous communications between all parties involved. One application that has implemented this is the MUTE p2p application.

In contrast to standard p2p applications, in an application such as MUTE, messages are routed through the network by using local clues, instead of using a direct connection between the two nodes exchanging data. Each node in the network knows only its neighbours, and the strength of the edges to each of those neighbours. When an ant reaches a node it uses these local clues to decide its next move. When it reaches the nest it can use the same technique to return to the nest. When ants travel these routes they lay more pheromone and increase the strength of the trail.

The remainder of this section of the report will attempt to implement and demonstrate ant routing algorithms based on the MUTE p2p idea, to provide anonymous network routing.

Proposed Software Architecture

Using a Object-Oriented approach the Graph based Ant World will consist of Node and Edge objects, where each Node can contain Ant objects, and each edge has a pheromone strength associated with it. It should be possible for the end-user to draw a graph on-screen via mouse actions, and see the simulation played out to eventually display the shortest path between two designated nodes. The simulation should be slowed down to allow the user to see what it happening and/or allow the user to run the simulation step-by-step. The number of time units will be measured as a way of deducing the time complexity of the algorithm.

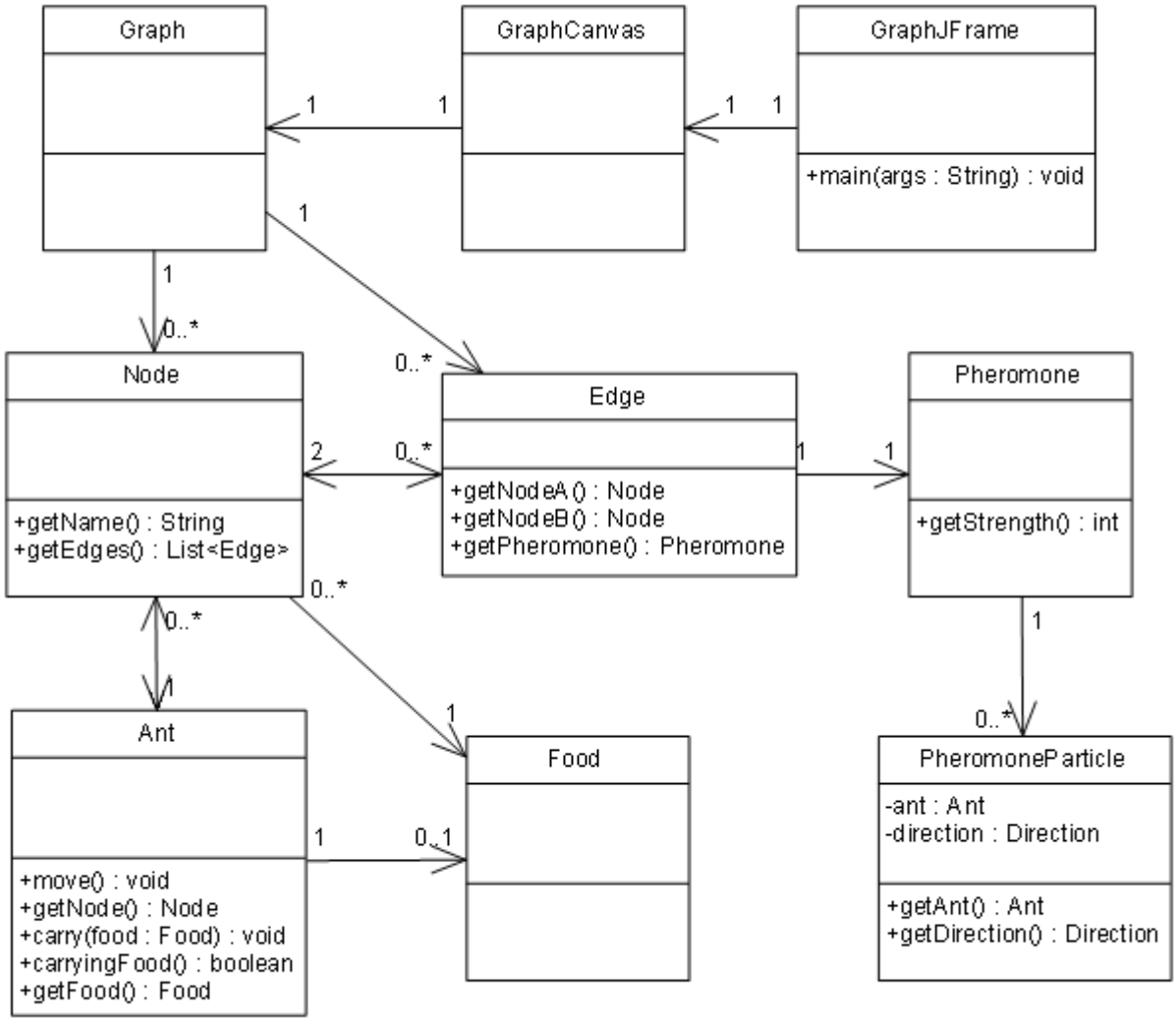


Figure 54: Ant Graph World Proposed Class Diagram

Technical Considerations

The base of the application needs to be a graph structure to be implemented using adjacency lists. A graphical user interface will allow the user to graph and edit the graph by adding, deleting and moving nodes and edges on a canvas.

Each node should be capable of holding ants and/or food. Ants will travel along the edges of the graph to find food, and return it to the nest. Like the Simulator each ant will have simple rules and must not pass messages to other ants. Each edge will have a pheromone strength, to guide the ants around the graph.

Ant Graph World

Version 1

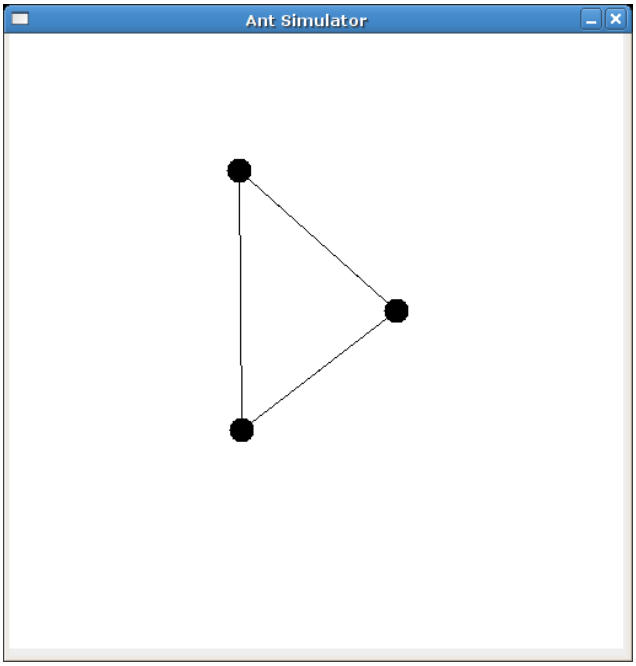


Figure 55: Version 1 - User created graph, representing 'The Double Bridge Experiment'

Version 1 implements a basic graph data structure using adjacency lists.

In this version the user can:

- click anywhere on the white canvas to add a node in that position
- click a node to select it
- click a second node to create an edge between it and the select node
- right-click to delete a node

The implementation

The node class extends from Java's built in Rectangle class, in order for it to use functions such as intersect. When the Node is drawn on the screen, an oval is created based on the dimensions of the Rectangle.

Each node contains a list of ants and food that it may contain – though this has not been fully implemented yet.

The Edge class extends from Java's Line2D.Double class, has references to its Node's and contains a pheromone object.

The Graph class contains the Node and Edge objects, and the GraphCanvas draws the Graph on the screen, and also waits for user input via a MouseListener.

This version does not do much apart from allow the user to draw a graph on the screen, subsequent versions will also implement the ant algorithms.

Version 2

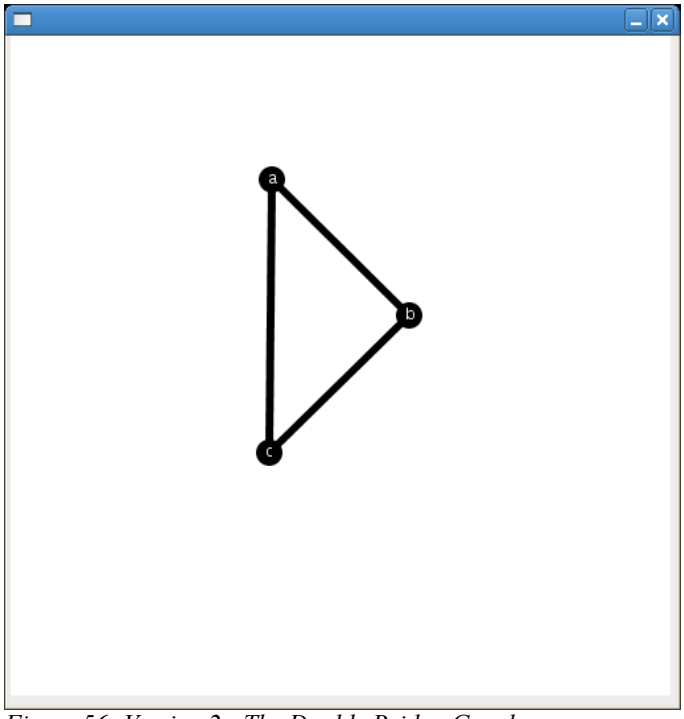


Figure 56: Version 2 - The Double Bridge Graph

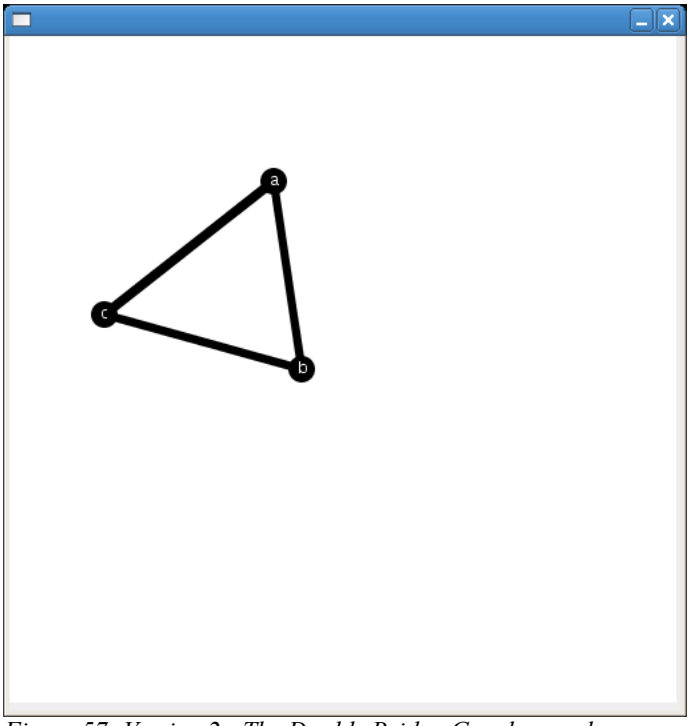


Figure 57: Version 2 - The Double Bridge Graph moved

Version 2 features an improved interface for creating graphs. It allows the user to reposition nodes by dragging them, nodes are now labelled and edges can now be removed.

Nodes are still placed as in the previous version. But if a node is already selected and another node placed an edge will automatically be created between the selected and new nodes. Right-click on an edge will remove it, the same for right-clicking on a node.

If a user drags a node that is connected to any edges then those edges will reposition automatically. This is very simply implemented – the edges end-points are the centres of the nodes at each end, therefore when the nodes position is updated the edges position is also updated.

The Edge class in this version is a subclass of Java's Polygon class. This is necessary as Line2D.Double objects only have one thickness, which was too thin to be able to select by clicking it with the mouse. With the help of some simple maths from the nice people at http://www.rgagnon.com/javadetails/java-0260.html the Edge class now draws a line by creating a polygon for its x1, y1, x2, y2, and width parameters.

Version 3

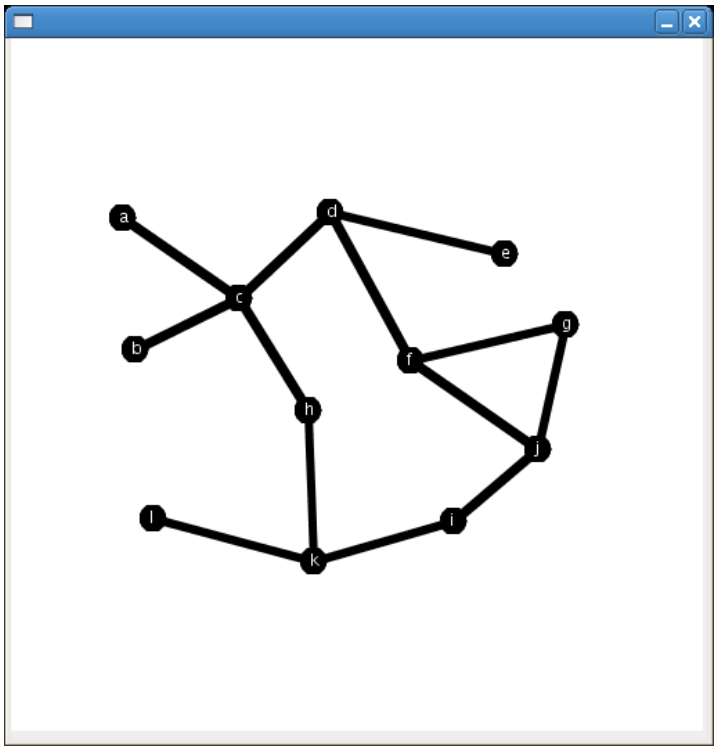


Figure 58: Version 3 - Test Graph

Version 3 improves the implementation of the graph objects, by introducing a new class GraphComponent. In this version Node and Edge are subclasses of GraphComponent.

A GraphComponent is a subclass of Java's built in Polygon class. Version 2 already used a polygon to represent an edge, and also a circle can be expressed as a polygon. The GraphComponent class is therefore in charge of the polygon's coordinates and of drawing.

The application is also now pre-loaded with a test graph.

Version 4

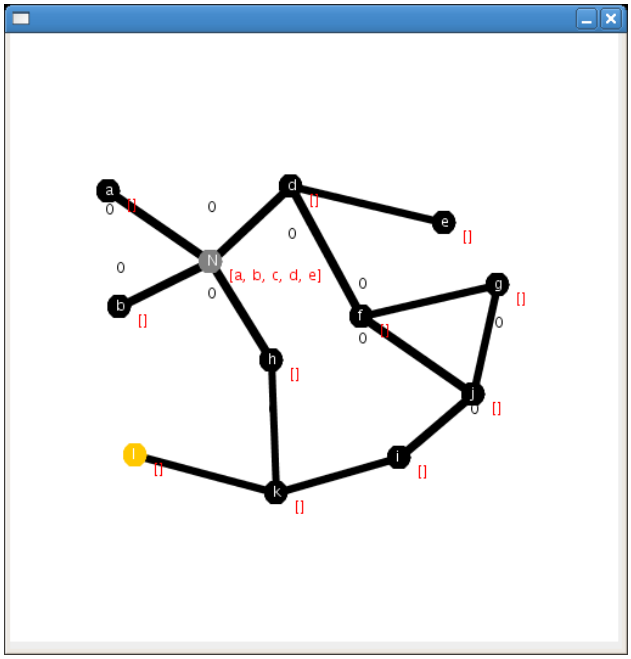


Figure 59: Version 4 - Test graph, Nest is at node 'N' and food is at node 'l'. 5 ants are in the nest.

Version 4 begins to implement ant algorithms for finding the shortest path from node N to node l. At the beginning of the simulation 5 ants (a,b,c,d and e) are in the nest at node N in figure 59.

The pheromone trails are decrease by 1 every 3 time units.

There is a problem with the positioning of the edge labels, which indicate the pheromone strength. They need to be placed near the middle of an edge.

```
if(not just picked up food and no food at current node and more than one edge) {
        dont move to previous node
}

if(carrying food) {

        if(A strength > 0) {
                travel via highest value A trail edge
        }else if(A+T strength == 0) {
                travel via random edge
        }else{
                travel via lowest value T trail edge
        }
        increase chosen edge T pheromone strength
}else{

        if(T strength > 0) {
                travel via highest value T trail edge
        }else if(A+T strength == 0) {
                travel via random edge
        }else{
                travel via lowest value A trail edge
        }
        increase chosen edge A pheromone strength
}

if(current node contains food and not carrying food and not in nest) {
        pick up food
}
```

Listing 15: Version 4 - Ant World Graph - Ant movement algorithm

A short description of classes

Ant

The Ant class represents a single ant in the application. Each ant has a name based on a character, starting at 'a'. If the ant is carrying food then the letter is capitalized. In the later versions of this application the threading was removed due to problems it caused with routing. This feature needs to be replaced, as the ants should be able to act independently.

Each ant has a memory of the previous node it has visited, this is not necessary and should be replaced with pheromone clues to decide where not to travel to.

Food

An empty class representing a piece of food.

Pheromone

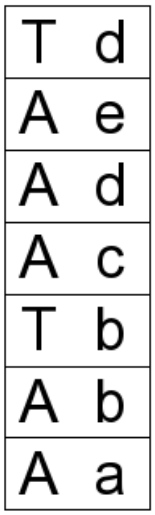


Figure 60: A Pheromone Stack

The pheromone class represents pheromone. One pheromone object is placed on each edge. Pheromone is increased as ants move over it. The pheromone contains a stack of PheromoneParticle objects.

Each PheromoneParticle object contains a direction, either Toward Nest (T) or Away From Nest (A), and a reference to the ant that 'deposited' the particle.

Figure 60 shows a graphical representation of a Pheromone object. The pheromone has been created by 5 ants, 2 of them laying pheromone twice.

The overall direction of a Pheromone object is the highest frequency of the directions of the pheromone particles. So the pheromone object in Figure 60 has an overall direction of Away From Nest.

The strength of a Pheromone object can be measured in each direction by just counting the particles of that direction.

The Pheromone object knows which ants created PheromoneParticles, therefore this could be used instead of the ants having a memory to remember the previous square they visited.

Threading also needs to be re-activated in Pheromones as well as Ants.

Node

A Node object represents one node on the graph. It can contain ants and/or food. Each node has a name, based on a character starting at 'a'. It has an x and y position corresponding to its position on the GraphCanvas. Further versions could arrange arbitrary graphs automatically, but this version nodes are placed by the user into a position.

Nest

A Nest object is a specialised Node object. It overrides the addAnt(ant : Ant) : void method so that it removes the food from the ant before it adds the ant to the node's list of ants.

Edge

An edge object represents a graph edge between two nodes. It has a reference to the node objects at

each of its ends. If the position one of its nodes is updated, its end coordinates are also updated meaning that the edge will following the movement of its end nodes. This allowed for the easy implementation of node dragging. Each edge has a Pheromone object as discussed.

The Edge class implements the Comparable interface allowing edges to be sorted. They are sorted by pheromone strength, and allows ants to sort edges of the node that they are at to make a decision about where to move next.

GraphComponent

Edge and Node classes are sub-classes of GraphComponent. A GraphComponent is a Polygon, and both edges and nodes (circles) are representation by polygons via this class. This class is responsible for drawing the polygons on screen (though both Node and Edge override the draw method, to draw extra things), and also for highlight and selection parameters.

Graph

The Graph class represents a mathematical graph consisting of Node and Edge objects, via an adjacency list. It includes methods to add/remove nodes and edges, and to highlight the shortest path via pheromone clues. The step() : void method is used to move the ants one step, and reduce the pheromone strength across the graph.

GraphJFrame

The GraphJFrame is the main window which holds the canvas. It also contains a method for creating a test graph, and handles keyboard input.

The Algorithm

The ant algorithm will allow our ants to travel from the nest node to some node X which contains food, this will hopefully be the shortest path.

The ants should do this without know where they have been or where they need to go next. They only know that they have arrived at there destination when they find the food. Then they will need to travel back to the nest, without knowing where it is. They do this by using the clues they leave behind in the form of pheromone trails.

As in the later version of the Ant World Simulator, there are two kinds of trails: toward nest trails (T) and away from nest (A) trails. While ants are looking for food they lay an A trail and look for a T trail, while they are carrying food they lay a T trail and look for a A trail.

Simple rules:

looking for food	lay A trail	follow strongest T trail	or weakest A trail
carrying food	lay T trail	follow strongest A trail	or weakest T trail

An ant looking for food without the choice of a T trail, would be best off following a trail where less ants have been looking for food, as a food source has not been discovered yet. Therefore by choosing a path that less ants look for food have chosen, it has more chance of finding a new food source, than if they all went the same way. The same holds for ants carrying food also, and following the weakest T trails.

If there are no trails at all then an ant will choose a direction randomly.

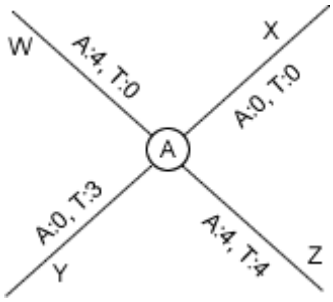


Figure 61: Hypothetical Node A

Figure 61 shows a hypothetical node, with four edges.

An ant carrying food would choose either W or Z because both of those edges have A pheromone of strength 4.

An ant looking for food would choose Z as it has a T pheromone of strength 4.

Figure 62 also shows a hypothetical node with four edges, this time there is no A trail on any of those edges.

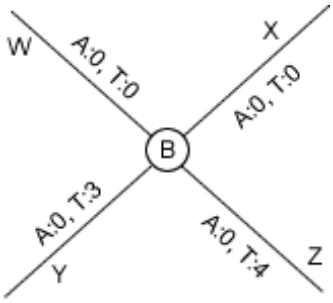


Figure 62: Hypothetical Node B

An ant carrying food would choose the edge with the weakest T trail, as less ants carrying food have recently gone that way. This would me either X or W.

An ant look for food will still choose Z as it has the highest T pheromone strength.Figure 63 shows a hypothetical node with four edges, this time there is no T trail on any of those edges.

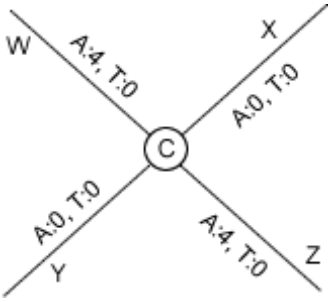


Figure 63: Hypothetical Node C

An ant looking for food would following the weakest A trail as less ants looking for food would have travelled that way.

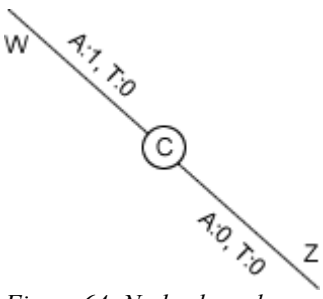


Figure 64: No backward movement

This algorithm should stop ants travelling back to a node that had just come from, in certain circumstances.

Figure 64 contains a node, C, that 1 ant looking for food has travelled to. It now has to make the decision between W and Z, the only edges connected to the node C.

As there is no T trail to help it decide, it will choose the lowest A trail. It has just laid some A pheromone on edge W, so therefore it will choose Z.

One case where this rule must be ignored is when a node only has one edge, the ant must therefore travel back on its path as there is only one option.

This case also does not hold if trails are of equal strength, for example in Figure 64 the A trail strength of Z was also one, the ant would choose randomly as both are the 'best' choice. In version 4 this is patched by having the previously travelled edge removed from its options (i.e. ants have a memory of the node they had just come from). A better implementation, that does not require the ants to have memory, would be to look at what ant laid the pheromone trail, which would allow it to 'know' its previous node.

Test Graphs

In order to demonstrate the application some test graphs are presented in this section.

Double Bridge Graph

The theory behind the double bridge experiment has been explained, now here is the application in action – finding the shortest path to the food. There are two ants (a and b) and 5 pieces of food. It takes 7 times units for the food to be collected. When the last piece of food is returned the shortest path is highlighted.

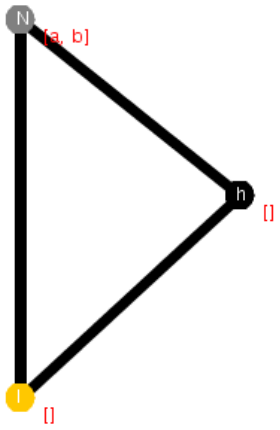


Figure 66: Double Bridge Graph, T=0

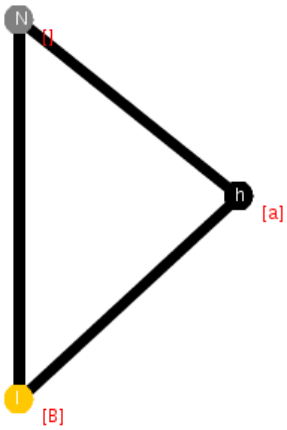


Figure 67: DBG, T=1

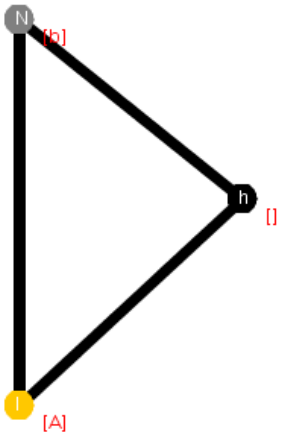


Figure 65: DBG, T=2

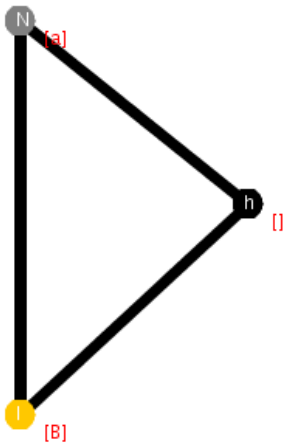


Figure 68: DBG, T=3

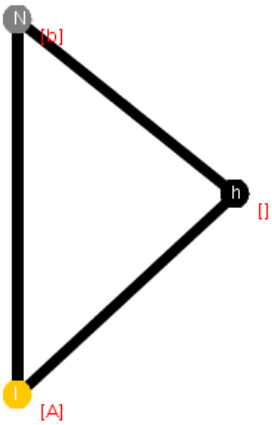


Figure 69: DBG, T=4

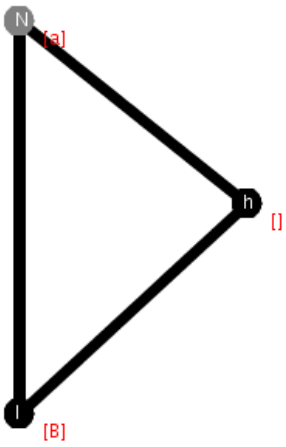


Figure 70: DBG, T=5

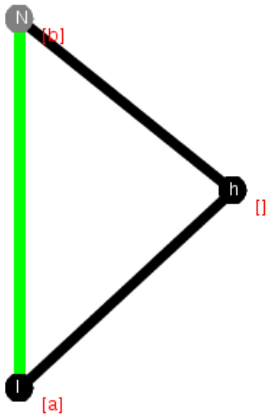


Figure 71: DBG, T=6

The Default Test Graph

When the algorithm was used on the test graph, it did not always produce the shortest route due to the initial ant finding the food via a longer route, and also due to ants being stuck in loops. Figures 73 and 72 show a successful run of the application.

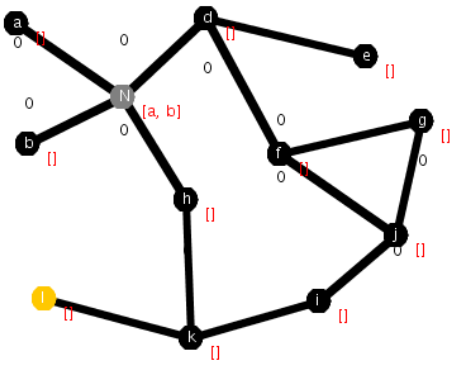


Figure 73: The Default Test Graph at T=0

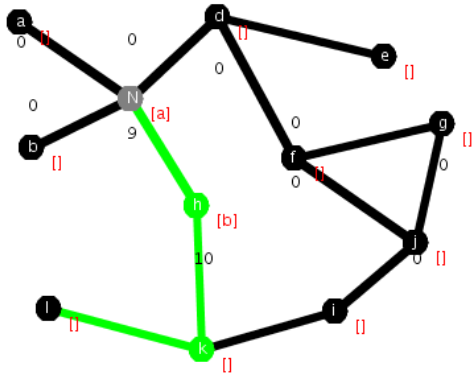


Figure 72: The Default Test Graph at T=34

Loop Problems

In reality a graph is likely to be more complicated than the double bridge graph, the test graph is an example of a slightly more complicated one. This introduces the problem of loops.

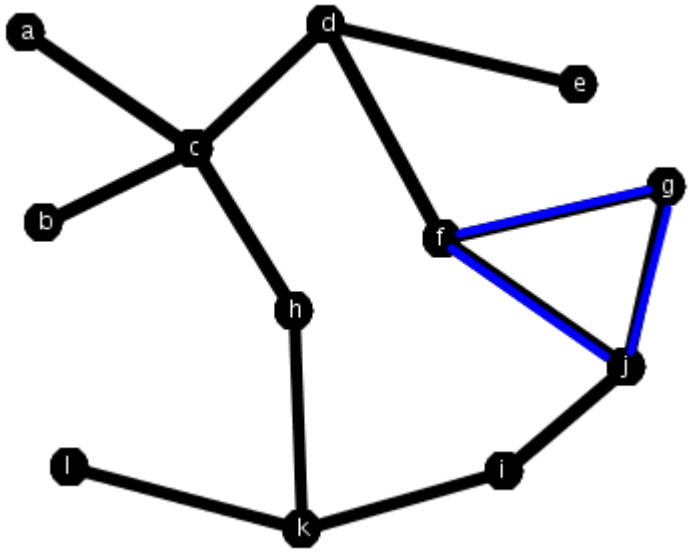


Figure 74: A possible loop problem

Figure 74 shows a possible loop problem in the test graph. Ants could find them self stuck in a loop, due to pheromone strength. As an ant travels around the loop it will reinforce the pheromone thereby wanting even more to travel around the loop.

One possible solution to this problem is to give ants a limited form of memory [1]. Ants would remember were they have been and wouldn't travel the same path twice (unless they have to).

Another solution would be for the ants to die after a certain number of time units. An an would have a Time To Live value, which would be decremented by one every move. This would mean that an ant stuck in a loop would eventually die. More ants, born in the nest, would be able to replace the dead ants. It could be that each set of newly hatched ants find themselves in the same loops, but due to randomness of their initial movements it could be assumed that not all new sets of ants will get stuck. This has the advantage that the ants do not need to remember anything.

Conclusion to Part 2

Part 2 was partly successfully as it allowed the user to create and modify a graph via a graphical user interface, and use an ant based algorithm from behaviour learned in Part 1 to find the shortest distance from the nest node to the food node.

Loop problems occurred. Also due to the randomness of the first choices, the algorithm did not always choose the shortest route between the nest and food.

Improvements and Future Work

Consider again the double bridge experiment. It is entirely possible that both ants initially follow the longest path and due to the positive feedback this creates, they will continue to follow that path. Now consider a much larger graph, such as a peer-to-peer network, where all the ants initially travel the long path. This will lead to great inefficiencies in data transfer, as all the data would be routed along a less-than adequate route.

In order to combat this problem, we can divide the message routing into two phases: route discovery and message delivery.

Two Phase Algorithm

Route Discovery

The route discovery phase involves initially finding the shortest path from the starting node (the nest) to the destination node (the food source). An ant, beginning at the starting node, must clone itself and sends copies of itself to every neighbour node. This a breadth-first-search algorithm. When an ant finds the node with the food, it will return to the nest along its own path.

If an ant finds that all edges from its current node have already been travelled i.e. they have pheromone strength > 0, then it must stop and wait to die. This will stop ants moving back if they have reached a node with only one edge and also stop nodes being travelled more than once. Again this allows the ants to rely on local clues rather than referring to a global list of visited nodes.

The first ant to reach the food, will have gotten to it in the shortest time, and therefore have taken the shortest path. Once an ant returns to the nest the route discovery phase is complete – the route between the nest and the food has been discovered.

Other ants may discover the food also, after the first ant, though they will follow the first ant back (think about the double bridge experiment again).

Giving a time to live (TTL) value for each ant will mean they will die after the TTL has expired, so that ants will not be left stuck at nodes.

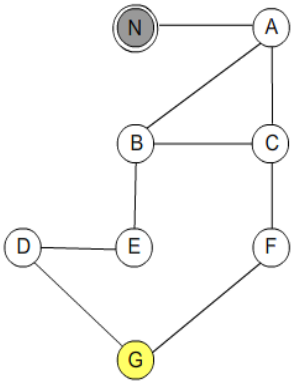


Figure 76: T=0

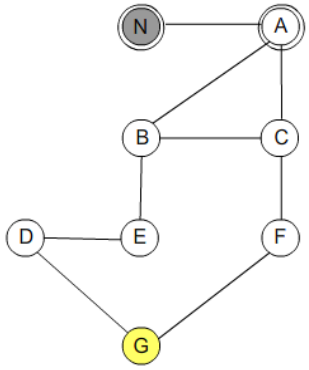


Figure 77: T=1

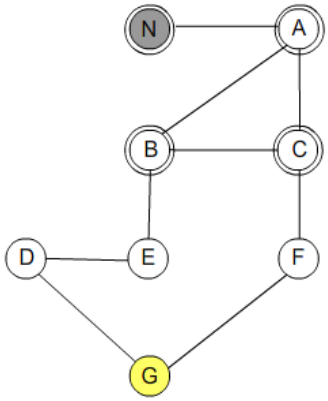


Figure 75: T=2

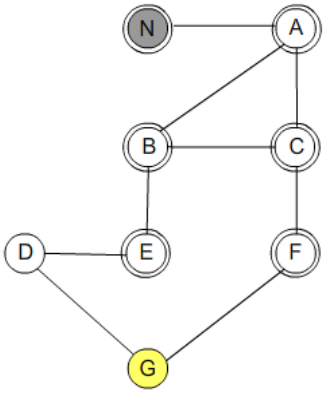


Figure 78: T=3

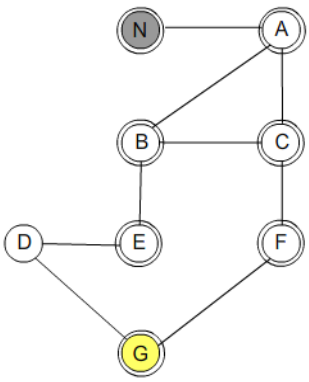


Figure 79: T=4

Message delivery

After the first ant has returned to the nest, a path now exists, marked by pheromones, for messages to be sent from the nest to the food source.

Any messages can now be routed along the pheromone trail marked path. The trail will become stronger as more ants use the path, due to the positive feedback of more ants laying more pheromone.

After the food source has been depleted (for example maybe the computer represented by the food node has left the network) ants will continue moving all the trial left by the previous ants, but the pheromone evaporates, so after a certain time there will be no trail left for the ants to follow.

When there is no trail from the starting node the route discovery phase must be started again, in order to find another food source.

MUTE [2] uses this technique of broadcasting to initially set up the trail for subsequent messages to travel.

Conclusion

Part 1 of this report intended to research and implement a simulation of ant behaviour. There were many problems encountered such as ants getting stuck at obstacles, but overall a lot was learned about ant behaviour.

Part 2 of this report intended to research and implement applications of the behaviour learned from part 1. It solved the obstacle problem by representing the Ant World as a graph, which could in turn represent something such as a computer network. Due to the randomness of the choice made by the first ant, the ant algorithm did not always find the shortest route. Further work has been suggested influenced by the MUTE peer-to-peer application, to complete this application.

Bibliography

1: Marco Dorigo and Thomas Stützle, Ant Colony Optimization, 2004
2: , , , http://mute-net.sourceforge.net/
3: Hussein, O.; Saadawi, T., Ant routing algorithm for mobile ad-hoc networks (ARAMA), 2003
4: Dr Rob Harris, Dr Natalie Hempel de Ibarra, Dr Paul Graham and and Professor Thomas Collett, Priming of visual route memories', 2005
5: Jackson, D.E., Holcombe, M., andRatnieks, Trail geometrygives polarity to ant foraging networks, 2004
6: Goss. S., Aron. S., Deneubourg J.L. and J.M. Pasteels, Self-organized shortcuts in the Argentine ant, 1989
7: Goss. S., Aron. S., Deneubourg J.L. and J.M. Pasteels, Self-organized shortcuts in the Argentine ant, 1989
8: , Swarm Intelligence,
9: Mesut Günes, Martin Khümer, and Imed Bouazizi, Ant-Routing-Algorithm (ARA) for Mobile Multi-hop Ad-hoc Networks - New Features and Results, 2003
10: Dr Rob Harris, Dr Natalie Hempel de Ibarra, Dr Paul Graham and and Professor Thomas Collett, Priming of visual route memories', 2005
11: John Montgomery, Ants foraging for food, , http://website.lineone.net/~john.montgomery/demos/ants.html

Appendix. A Accompanying Software

A CD containing application software and source code has been distributed with this report. On it you will find all versions of the two software applications, and an electronic version of this report.

Each version is distributed as a Jar file and requires that Java Runtime Environment 1.6 or greater be installed.

The CD contains a folder for Ant World Simulator, and Ant Graph World. Each of these contains a folder for each version and inside these a Netbeans project folder, in which you'll find a dist folder containing the Jar file, and a src folder containing the source code.

An accompanying website also mirrors the contents of the CD at http://whoyouknow.co.uk/ants/. On the website you will find all versions of the software available as Java Web Start applications, again you will need JRE 1.6 or greater to run the applications. Several videos of the Ant World Simulator are available to view of the application in action.

These easiest way to run the software is to use the Java Web Start versions via the website, or run the jar file from the command line: java -jar Ant.jar

Appendix. B Source Code

Included in this appendix is the source code for the latest versions of both the simulator and graph applications. Source code for all versions can be found on the accompanying CD and website.

Ant World Simulator

Ant.java

```
/*
 * Ant.java
 *
 * Created on 07 October 2006, 15:45
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */

package ants;

import ants.event.AntEvent;
import ants.event.AntListener;
import ants.event.PickedUpFoodAntEvent;
import java.awt.Color;
import java.awt.Graphics;
import java.util.Collections;
import java.util.HashSet;
import java.util.Random;
import java.util.Vector;

/**
 *
 * @author James Hamilton
 */
public class Ant extends MyObject implements Runnable {

    private Thread move;
    private int currentDirection = GridSquare.getRandomDirection();
    private Food food = null;

    private GridSquare lastGridSquare = null;
    private GridSquare previousGridSquare = null;
    private HashSet<GridSquare> dontVisit = new HashSet<GridSquare>();

    private Color color = Color.red;


    /** Creates a new instance of Ant */
    public Ant() {
    }

    public Ant(Ant ant) {
         super(ant.getGridSquare());
    }

    public Ant(GridSquare gridSquare) {
        super(gridSquare);
        Random generator = new Random();
         color = new Color(generator.nextInt( 255 ), generator.nextInt( 255 ),generator.nextInt(
255 ));

      //  setGridSquare(gridSquare);
    }

    private Vector<AntListener> listeners = new Vector<AntListener>();

    public void addAntListener(AntListener listener) {
        listeners.add(listener);
    }

    public void notifyListeners(AntEvent e) {
        for(AntListener listener : listeners)
            listener.antEventHandler(e);
    }
```

```
  65
  66    public boolean onTrail() {
  67        return getGridSquare().getPheromoneStrength() > 0;
  68    }
  69
  70    public boolean onOwnTrail() {
  71        return getGridSquare().getPheromone().laidBy(this);
  72    }
  73
  74
  75    public GridSquare getNextGridSquare() {
  76
  77        GridVector squares;// = getGridSquare().getGridSquares(1).getFreeGridSquares();
  78                                                                                    squares          =
getGridSquare().getGridSquaresInDirection(currentDirection).getFreeGridSquares();
  79
  80        if(carryingFood()) squares = squares.getEmptyGridSquares();
  81
  82        GridSquare currentGridSquare = getGridSquare();
  83        GridSquare nextGridSquare = null;
  84
  85        if(getGridSquare().isNest()) {
  86            if(carryingFood()) {
  87                Grid.getInstance().getNest().addAnt(this);
  88            }else{
  89                squares = getGridSquare().getGridSquares(1).getFreeGridSquares();
  90
  91                if(squares.getPheromoneStrength(PheromoneParticle.TOWARD_NEST) > 0) {
  92
  93                    squares = squares.getPheromoneGridSquares();
  94
  95                                                                    GridVector   towardsNest   =
squares.getPheromoneGridSquares(PheromoneParticle.TOWARD_NEST);
  96                                                                   GridVector   awayFromNest   =
squares.getPheromoneGridSquares(PheromoneParticle.AWAY_FROM_NEST);
  97
  98
  99                        if(towardsNest.size() > 0) {
 100                            Collections.sort(squares, GridSquare.PheromoneStrengthComparator);
 101
 102                            nextGridSquare = squares.firstElement();
 103                        }
 104                        /*else if(awayFromNest.size() > 0) {
 105                             if(awayFromNest.getPheromoneGridSquares(this).size() > 0) {
 106                                squares = awayFromNest.getPheromoneGridSquares(this);
 107                            }else{
 108                                 squares = awayFromNest;
 109                            }
 110
 111                            Collections.sort(squares, GridSquare.PheromoneStrengthComparator);
 112
 113                            nextGridSquare = squares.firstElement();
 114                        }*/
 115
 116                }
 117
 118                if(nextGridSquare == null) nextGridSquare = squares.getRandomGridSquare();
 119                currentDirection = getGridSquare().getDirectionOf(nextGridSquare);
 120            }
 121        }else{
 122
 123            if(squares.size() == 0) {
 124                //if there is nowhere 2 go infront, look behind.
 125                currentDirection = GridSquare.getOppositeDirection(currentDirection);
 126                                                                                    squares     =
getGridSquare().getGridSquaresInDirection(currentDirection).getFreeGridSquares();
 127            }
 128
 129            boolean changeDirection = false;
 130
 131            if(squares.size() == 0 || (onTrail() && carryingFood())) {
 132                //if there is nowhere 2 go behind, look look sidewards.
 133                squares = getGridSquare().getGridSquares(1).getFreeGridSquares();
 134                changeDirection = true;
 135            }
 136
 137            if(carryingFood() && squares.containsNest()) {
```

```
138                 nextGridSquare = Grid.getInstance().getNest().getGridSquare();
139             }else if(!carryingFood() && squares.containsFood() ) {
140                 squares = squares.getFoodGridSquares();
141 
142                 nextGridSquare = squares.getRandomGridSquare();
143             }else if(!carryingFood() && squares.getFoodStrength() > 0) {
144                 squares = squares.getFoodStrengthGridSquares();
145 
146                 Collections.sort(squares, GridSquare.FoodStrengthComparator);
147 
148                 nextGridSquare = squares.firstElement();
149             }else if(carryingFood() && squares.getPheromoneStrength() > 0) {
150                 squares = squares.getPheromoneGridSquares();
151 
152                                                                 GridVector    towardsNest    =
squares.getPheromoneGridSquares(PheromoneParticle.TOWARD_NEST);
153                                                                GridVector    awayFromNest    =
squares.getPheromoneGridSquares(PheromoneParticle.AWAY_FROM_NEST);
154 
155                 if(carryingFood()) {
156                     if(towardsNest.size() > 0) {
157                         Collections.sort(squares, GridSquare.PheromoneStrengthComparator);
158 
159                         nextGridSquare = squares.firstElement();
160                     }else if(awayFromNest.size() > 0) {
161                          if(awayFromNest.getPheromoneGridSquares(this).size() > 0) {
162                             squares = awayFromNest.getPheromoneGridSquares(this);
163                         }else{
164                              squares = awayFromNest;
165                         }
166 
167                         Collections.sort(squares, GridSquare.PheromoneStrengthComparator);
168 
169                         nextGridSquare = squares.firstElement();
170                     }
171                 }
172 
173                 Collections.sort(squares, GridSquare.PheromoneStrengthComparator);
174 
175                 nextGridSquare = squares.getRandomGridSquare(true);
176 
177 
178             }else if(squares.size() == 3) {
179                 nextGridSquare = squares.getMiddleGridSquare();
180             }else if(squares.size() > 0) {
181                 nextGridSquare = squares.getRandomGridSquare();
182             }else{
183                 //dont move
184             }
185 
186             if(changeDirection)
187                 currentDirection = getGridSquare().getDirectionOf(nextGridSquare);
188 
189         }
190 
191         return nextGridSquare;
192     }
193 
194     public synchronized void move() {
195 
196 
197           if(carryingFood()) {
198 
199                                             getGridSquare().getPheromone().increaseStrength(this,
PheromoneParticle.TOWARD_NEST);
200 
201         }else{
202 
203 
204                                             getGridSquare().getPheromone().increaseStrength(this,
PheromoneParticle.AWAY_FROM_NEST);
205 
206         }
207         GridVector squares = getGridSquare().getGridSquares(1).getFreeGridSquares();
208         GridSquare currentGridSquare = getGridSquare();
209         GridSquare nextGridSquare = getNextGridSquare();
210 
```

```
211
212
213
214
215
216
217         if(nextGridSquare != null) {
218             if(nextGridSquare.containsFood()) carry(nextGridSquare.getFood());
219             setGridSquare(nextGridSquare, false);
220         }
221
222
223
224
225     }
226
227     public void carry(Food food) {
228       //  System.out.println("Carrying...");
229        // Grid.getInstance().printStats();
230
231         try {
232             notifyListeners(new PickedUpFoodAntEvent(this));
233             this.setFood(food);
234             food.getGridSquare().recalculateFoodStrength(false);
235             food.getGridSquare().setObject(null);
236
237             turnBack();
238             //setGridSquare(getPreviousGridSquare());
239
240             food.setGridSquare(null);
241
242
243         }catch (RuntimeException e) {
244            // System.out.println("Already gone!");
245         }
246     }
247
248     public boolean carryingFood() {
249         return getFood() != null;
250     }
251
252     public void turnBack() {
253             if(getPreviousGridSquare() != null) {
254                 currentDirection = getGridSquare().getDirectionOf(getPreviousGridSquare());
255             }
256     }
257
258     public void run() {
259           //Call the move method while the Thread is running. And sleep for a time based on
iSpeed.
260         while(move != null) {
261             try {
262                 move();
263
264                 move.sleep(200);
265             }catch (InterruptedException e) {}
266         }
267     }
268
269     /**
270      * Start or stop the rectangle moving.
271      * @param start true to start, false to stop.
272      */
273     public void start(boolean start) {
274         if(start) start(); else stop();
275     }
276
277     /**
278      * Start the rectangle moving.
279      */
280     public void start() {
281         //Create and start a new Thread.
282         move = new Thread(this);
283       //  move.setDaemon(true);
284         move.start();
285     }
286
```

```
287    public void pause() {
288        if(move == null) start();
289        else stop();
290    }
291
292    /**
293     * Stop the rectangle moving.
294     */
295    public void stop() {
296        move = null;
297    }
298
299
300    public Color getColor() {
301        if(food == null)
302            return color;
303        else
304            return Color.YELLOW;
305    }
306
307    public Food getFood() {
308        return food;
309    }
310
311    public void setFood(Food food) {
312        this.food = food;
313    }
314
315    public String toString() {
316        return "Ant"
317            + getGridSquare().getGridSquares(1).getGridSquaresInDirection(
318                    getGridSquare(), currentDirection).getFreeGridSquares()
319            + ";direction: " + currentDirection;
320    }
321
322    public void draw(Graphics g) {
323        super.draw(g);
324
325
326    }
327
328    public GridSquare getPreviousGridSquare() {
329        return previousGridSquare;
330    }
331
332    public void setPreviousGridSquare(GridSquare previousGridSquare) {
333        this.previousGridSquare = previousGridSquare;
334    }
335
336    public void setGridSquare(GridSquare gridSquare, boolean setDirection) {
337        if(setDirection)
338               currentDirection = getGridSquare().getDirectionOf(gridSquare);
339
340        setPreviousGridSquare(getGridSquare());
341        super.setGridSquare(gridSquare);
342    }
343
344    public void setGridSquare(GridSquare gridSquare) {
345            setGridSquare(gridSquare, false);
346    }
347}
348
```

Food.java

```
 1/*
 2 * Food.java
 3 *
 4 * Created on 07 October 2006, 15:45
 5 *
 6 * To change this template, choose Tools | Template Manager
 7 * and open the template in the editor.
 8 */
 9
10package ants;
11
12import java.awt.Color;
13
14/**
15 *
16 * @author James Hamilton
17 */
18public class Food extends MyObject {
19
20    private static final Color color = new Color(150, 255, 150);
21
22    /** Creates a new instance of Food */
23    public Food() {
24    }
25
26    public Food(GridSquare s) {
27        super(s);
28    }
29
30    public Color getColor() {
31        return color;
32    }
33
34    public String toString() {
35        return "Food";
36    }
37
38}
39
```

Grid.java

```
/*
 * Grid.java
 *
 * Created on 07 October 2006, 01:36
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */

package ants;

import ants.control.AntControl;
import ants.event.AntEvent;
import ants.event.AntListener;
import ants.event.FoodUpdateEvent;
import java.awt.Canvas;
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Point;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.image.BufferedImage;
import java.util.Vector;

/**
 *
 * @author James Hamilton
 */
public class Grid extends Canvas implements Runnable {

    private GridSquare[][] grid;
    private GridVector gridVector;

    private BufferedImage bi;
    private Graphics big;
    private Thread animate;
    private Thread resetFoodStrength;

    private static Grid singleton = null;
    private Vector<MyObject> objects;

    private Vector<FoodCluster> foodClusters;

    private Nest nest;

    private int size = 100;

    /** Creates a new instance of Grid */
    private Grid(int size) {
        this.size = size;
        initialise();
    }

    private void initialise() {
        grid = new GridSquare[size][size];
        setGridVector(new GridVector());
        setSize(size * GridSquare.SIZE, size * GridSquare.SIZE);

        for(int x = 0; x < grid.length; x++) {
            for(int y = 0; y < grid.length; y++) {

                grid[x][y] = new GridSquare();
                grid[x][y].setLocation(x * GridSquare.SIZE, y * GridSquare.SIZE);
                getGridSquares().add(grid[x][y]);
            }

        }
        bi = new BufferedImage(getWidth(),getHeight(),BufferedImage.TYPE_INT_RGB);
        big = bi.createGraphics();

        setNest(new Nest(grid[20][20]));

        objects = new Vector<MyObject>();
        objects.add(getNest());

```

```
 76
 77
 78        setFoodClusters(new Vector<FoodCluster>());
 79
 80        animate = null;
 81        animate = new Thread(this);
 82        animate.start();
 83
 84        addMouseListener( new MouseAdapter() {
 85            public void mouseClicked(MouseEvent e) {
 86                System.out.println(getGridSquareAt(e.getX(), e.getY()));
 87            }
 88        });
 89
 90        addAntListener(AntControl.getInstance());
 91    }
 92
 93    private Vector<AntListener> listeners = new Vector<AntListener>();
 94
 95    public void addAntListener(AntListener listener) {
 96        listeners.add(listener);
 97    }
 98
 99    public void notifyListeners(AntEvent e) {
100        for(AntListener listener : listeners)
101            listener.antEventHandler(e);
102    }
103
104    public synchronized static Grid getInstance(int n) {
105        if(singleton == null)
106            singleton = new Grid(n);
107        return singleton;
108    }
109
110    public synchronized static Grid getInstance() {
111        if(singleton == null)
112            singleton = new Grid(100);
113        return singleton;
114    }
115
116    public void reset() {
117        initialise();
118    }
119
120    public void recalculateFoodStrengths() {
121        for(GridSquare gs : getGridSquares()) {
122            gs.recalculateFoodStrength();
123        }
124    }
125
126    public void setFoodStrengthRadius(int radius) {
127        for(FoodCluster cluster : foodClusters) {
128            cluster.setStrengthRadius(radius);
129        }
130    }
131
132    public void resetFoodStrengths() {
133        recalculateFoodStrengths();
134    }
135
136    public synchronized GridSquare getGridSquare(int x, int y) throws Exception {
137        try {
138            return grid[x][y];
139        }catch (ArrayIndexOutOfBoundsException e) {
140            throw new Exception("Invalid Square");
141        }
142    }
143
144    public GridSquare getGridSquare(Point p) throws Exception {
145        return getGridSquare(p.x, p.y);
146    }
147
148    public GridSquare getGridSquareAt(int x, int y) {
149        for(GridSquare gs : gridVector) {
150            if(gs.contains(x, y)) return gs;
151        }
152        return null;
```

```
153     }
154
155     public Vector<MyObject> getObjects() {
156         return objects;
157     }
158
159     public void addObject(MyObject object) {
160         objects.add(object);
161     }
162
163     public void populate(int n) {
164
165
166     }
167
168     public void populateWithFood(int n) {
169
170         for(int j = 0; j < n; j++) {
171
172             boolean stop = false;
173
174             while(!stop) {
175
176                 int x = (int)(Math.random() * grid.length);
177                 int y = (int)(Math.random() * grid.length);
178
179                 if(grid[x][y].empty()) {
180
181                     int size = (int)(Math.random() * 5) + 1;
182
183                     try {
184                         addFoodCluster(x, y, size);
185                     }catch (Exception e) {
186                         //no such gridsquare
187                     }
188
189
190                     stop = true;
191                 }
192             }
193         }
194
195         this.notifyListeners(new FoodUpdateEvent(this));
196     }
197
198     public void addFoodCluster(int x, int y, int radius) throws Exception {
199
200         getGridSquare(x, y).getGridSquares(radius, true).getFreeGridSquares().addFood();
201
202         notifyListeners(new FoodUpdateEvent(this));
203     }
204
205     public void addObstacleCluster(int x, int y, int radius) throws Exception {
206         getGridSquare(x, y).getGridSquares(radius, true).addObstacles();
207     }
208
209     public void populateWithObstacles(int n) {
210
211         for(int j = 0; j < n; j++) {
212
213             boolean stop = false;
214
215             while(!stop) {
216
217                 int x = (int)(Math.random() * grid.length);
218                 int y = (int)(Math.random() * grid.length);
219
220                 if(grid[x][y].empty()) {
221                     try {
222                         addObstacleCluster(x, y, 3);
223                     }catch (Exception e) {
224                         //no such gridsquare
225                     }
226
227
228                     stop = true;
229                 }
```

```
230             }
231         }
232 
233         for(int x = 0; x < grid.length; x++) {
234             try {
235                 Obstacle obstacle = new Obstacle(getGridSquare(x, 0));
236                 objects.add(obstacle);
237                 getGridSquare(x, 0).setObject(obstacle);
238 
239                 Obstacle obstacle1 = new Obstacle(getGridSquare(x, grid.length-1));
240                 objects.add(obstacle1);
241                 getGridSquare(x,grid.length-1).setObject(obstacle1);
242 
243             }catch (Exception e) {
244                 //
245             }
246         }
247 
248        for(int y = 0; y < grid.length; y++) {
249             try {
250                 Obstacle obstacle = new Obstacle(getGridSquare(0, y));
251                 objects.add(obstacle);
252                 getGridSquare(0, y).setObject(obstacle);
253 
254                 Obstacle obstacle1 = new Obstacle(getGridSquare(grid.length-1, y));
255                 objects.add(obstacle1);
256                 getGridSquare(grid.length-1,y).setObject(obstacle1);
257 
258             }catch (Exception e) {
259                 //
260             }
261         }
262 
263 
264     }
265 
266 
267     public int countAnts() {
268         int count = 0;
269         for(int x = 0; x < grid.length; x++) {
270             for(int y = 0; y < grid.length; y++) {
271                 if(grid[x][y].containsAnt()) count++;
272             }
273         }
274         return count;
275     }
276 
277     public int countCarriedFood() {
278         int count = 0;
279         for(int x = 0; x < grid.length; x++) {
280             for(int y = 0; y < grid.length; y++) {
281                 if(grid[x][y].containsAnt() && grid[x][y].getAnt().carryingFood()) count++;
282             }
283         }
284         return count;
285     }
286 
287     public int countNests() {
288         int count = 0;
289         for(int x = 0; x < grid.length; x++) {
290             for(int y = 0; y < grid.length; y++) {
291                 if(grid[x][y].isNest()) count++;
292             }
293         }
294         return count;
295     }
296 
297 
298 
299     public void setAntAt(Ant ant, int x, int y) {
300         grid[x][y].setAnt(ant);
301     }
302 
303 
304     public void paint(Graphics g) {
305         update(g);
306     }
```

```
307
308     public void update(Graphics g) {
309         clear();
310
311         for(GridSquare object : gridVector) {
312
313             object.draw(big);
314
315         }
316
317         g.drawImage(bi, 0, 0, this);
318     }
319
320     public void clear() {
321         big.setColor(Color.black);
322         big.fillRect(0,0,getWidth(),getHeight());
323     }
324
325     public void run() {
326         while(animate!=null) {
327
328             try {
329                 animate.sleep(300);
330
331
332                 Thread.yield();
333                 //printStats();
334
335                 repaint();
336             }catch (Exception e) {
337                 System.out.println(e);
338             }
339
340         }
341         big.dispose();
342     }
343
344
345     public void printStats() {
346         System.out.print(countAnts() + " ants. ");
347         System.out.print(countCarriedFood() + " carrying food");
348         System.out.println();
349     }
350
351     public Nest getNest() {
352         return nest;
353     }
354
355     public void setNest(Nest nest) {
356         this.nest = nest;
357     }
358
359     public Vector<FoodCluster> getFoodClusters() {
360         return foodClusters;
361     }
362
363     public void setFoodClusters(Vector<FoodCluster> foodClusters) {
364         this.foodClusters = foodClusters;
365     }
366
367     public GridVector getGridSquares() {
368         return gridVector;
369     }
370
371     public GridVector getGridSquaresWithFood() {
372         return getGridSquares().getFoodGridSquares();
373     }
374
375     public int countFood() {
376         return getGridSquaresWithFood().size();
377     }
378
379     public void setGridVector(GridVector gridVector) {
380         this.gridVector = gridVector;
381     }
382 }
383
```

AntSimulator.java

```
/*
 * AntSimulator.java
 *
 * Created on 07 October 2006, 01:43
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */

package ants;


import ants.control.AntControl;
import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import javax.swing.JApplet;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;

/**
 *
 * @author James Hamilton
 */
public class AntSimulator extends JApplet {

    private Grid grid;

    /** Creates a new instance of AntSimulator */
    public AntSimulator(int size) {

             initialise();

            int r = (int)(Math.random() * Math.sqrt(8)) * (int)(Math.random() *Math.sqrt(8));
            System.out.println(r);


        setVisible(true);
    }

    public void initialise() {
        grid = Grid.getInstance(100);
        grid.reset();
        setSize(getSize());

        getContentPane().remove(grid);
        getContentPane().add(grid, BorderLayout.CENTER);

        grid.getNest().addAnts(8);

        grid.populateWithObstacles(0);
        //grid.populateWithFood(5);
        try {
            grid.addFoodCluster(80, 20, 3);//7
           grid.addObstacleCluster(50, 21, 4);
            grid.addObstacleCluster(50, 22, 4);
           grid.addObstacleCluster(50, 23, 4);
          //  grid.addFoodCluster(30, 80, 3);
        }catch (Exception e) {
            //no such grid square.
        }

        Grid.getInstance().resetFoodStrengths();

    }

    public AntSimulator() {
        this(100);
    }
```

```
74
75    public Dimension getSize() {
76        return grid.getSize();
77    }
78
79    public static void main(String[] args) {
80        java.awt.EventQueue.invokeLater(new Runnable() {
81            public void run() {
82                JFrame f = new JFrame("Ant Simulator");
83                final AntSimulator m = new AntSimulator(100);
84                f.add(m);
85                f.setSize((int)m.getSize().getWidth() + 13, (int)m.getSize().getHeight() + 35);
86                f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
87                f.setResizable(false);
88                f.setVisible(true);
89
90                AntControl.getInstance().setVisible(true);
91            }
92        });
93
94
95
96    }
97}
98
```

GridSquare.java

```
   1/*
   2 * GridSquare.java
   3 *
   4 * Created on 07 October 2006, 01:37
   5 *
   6 * To change this template, choose Tools | Template Manager
   7 * and open the template in the editor.
   8 */
   9
  10package ants;
  11
  12import java.awt.Color;
  13import java.awt.Graphics;
  14import java.awt.Graphics2D;
  15import java.awt.Point;
  16import java.awt.Rectangle;
  17import java.util.Comparator;
  18import java.util.Vector;
  19import java.lang.Double;
  20
  21/**
  22 *
  23 * @author James Hamilton
  24 */
  25public class GridSquare extends Rectangle implements Comparable<GridSquare> {
  26
  27    public static final int SIZE = 4;
  28
  29    private MyObject object = null;
  30    private Ant ant = null;
  31    private Pheromone pheromone = null;
  32
  33    private double foodStrength = 0;
  34
  35
  36    public static final int
  37        NORTH = 1,  NORTHEAST = 2, EAST = 3, SOUTH = 4, SOUTHEAST = 5, WEST = 6, NORTHWEST = 7,
SOUTHWEST = 8;
  38
  39    public static final int LAST_DIRECTION = SOUTHWEST;
  40    public static final int FIRST_DIRECTION = NORTH;
  41
  42    /** Creates a new instance of GridSquare */
  43    public GridSquare() {
  44        setSize(SIZE, SIZE);
  45        //pheromone = new Pheromone(this);
  46    }
  47
  48    public Point getCoordinateXY() {
  49        return new Point(x / SIZE, y / SIZE);
  50    }
  51
  52    public GridSquare getGridSquare(int direction) {
  53        try {
  54            Point p = getCoordinateXY();
  55            int x = -1, y = -1;
  56
  57            switch(direction) {
  58                case EAST:  x = p.x + 1; y = p.y; break;
  59                case NORTH: x = p.x; y = p.y - 1; break;
  60                case WEST:  x = p.x - 1; y = p.y; break;
  61                case SOUTH: x = p.x; y = p.y + 1; break;
  62                case SOUTHEAST: x = p.x + 1; y = p.y + 1; break;
  63                case SOUTHWEST: x = p.x - 1; y = p.y + 1; break;
  64                case NORTHEAST: x = p.x + 1; y = p.y - 1; break;
  65                case NORTHWEST: x = p.x - 1; y = p.y - 1; break;
  66            }
  67
  68
  69            return Grid.getInstance().getGridSquare(x, y);
  70        }catch (Exception e) {
  71            System.out.println(direction +": " + getCoordinateXY() +" - " + e);
  72            return null;
```

```
73        }
74    }
75
76    public GridVector getGridSquaresInDirection(int direction) {
77        GridVector v = new GridVector();
78        v.add(getGridSquare(direction));
79        v.add(getGridSquare(getPreviousDirection(direction)));
80        v.add(getGridSquare(getNextDirection(direction)));
81        return v;
82    }
83
84    public GridSquare getGridSquare() {
85        return getGridSquare(GridSquare.getRandomDirection());
86    }
87
88    public static int getRandomDirection() {
89        return (int)(Math.random() * LAST_DIRECTION) + 1;
90    }
91
92    public static int getOppositeDirection(int direction) {
93        switch(direction) {
94            case EAST:  return WEST;
95            case NORTH: return SOUTH;
96            case WEST:  return EAST;
97            case SOUTH: return NORTH;
98            case SOUTHEAST: return NORTHWEST;
99            case SOUTHWEST: return NORTHEAST;
100           case NORTHEAST: return SOUTHWEST;
101           case NORTHWEST: return SOUTHEAST;
102       }
103       return -1;
104   }
105
106   public GridVector getGridSquares(int radius) {
107       return getGridSquares(radius, false);
108   }
109
110   public GridVector getGridSquares(int radius, boolean center) {
111      GridVector squares = new GridVector();
112       Point p = getCoordinateXY();
113
114       for(int x = -radius; x <= radius; x++) {
115
116           int z;
117
118
119               z = (int)(Math.sqrt(Math.pow(radius, 2) - Math.pow(x, 2)));
120
121           for(int y = -z; y <= z; y++) {
122
123               try {
124                   if(x == 0 && y == 0 && !center) continue;
125                       squares.add(Grid.getInstance().getGridSquare(p.x + x, p.y + y));
126
127
128               } catch (Exception ex) {
129
130               }
131
132           }
133
134       }
135
136       if(radius == 1) {
137           try {
138
139               squares.add(Grid.getInstance().getGridSquare(p.x - 1, p.y - 1));
140               squares.add(Grid.getInstance().getGridSquare(p.x + 1, p.y + 1));
141               squares.add(Grid.getInstance().getGridSquare(p.x + 1, p.y - 1));
142               squares.add(Grid.getInstance().getGridSquare(p.x - 1, p.y + 1));
143
144           } catch (Exception ex) {
145
146           }
147
148       }
149
```

```
150         return squares;
151     }
152 
153     public int getDirectionOf(GridSquare gs) {
154         Point thisP = getCoordinateXY();
155         Point otherP = gs.getCoordinateXY();
156 
157         int x = otherP.x - thisP.x;
158         int y = otherP.y - thisP.y;
159 
160         if(otherP.x == thisP.x && otherP.y < thisP.y) {
161             return NORTH;
162         }else if(otherP.x == thisP.x && otherP.y > thisP.y) {
163             return SOUTH;
164         }else if(otherP.y == thisP.y && otherP.x < thisP.x) {
165             return WEST;
166         }else if(otherP.y == thisP.y && otherP.x > thisP.x) {
167             return EAST;
168//         }else if(otherP.x == thisP.x && otherP.y > thisP.y) {
169//             return EAST;
170         }else if(otherP.x < thisP.x && otherP.y < thisP.y) {
171             return NORTHWEST;
172         }else if(otherP.x > thisP.x && otherP.y < thisP.y) {
173             return NORTHEAST;
174         }else if(otherP.x > thisP.x && otherP.y > thisP.y) {
175            return SOUTHEAST;
176         }else if(otherP.x < thisP.x && otherP.y > thisP.y) {
177             return SOUTHWEST;
178         }else{
179             System.out.println("thisP: " + thisP + ", otherP:" + otherP);
180 
181         }
182 
183         return -1;
184     }
185 
186     public Vector<GridSquare> getEmptyGridSquares(int radius) {
187         Vector<GridSquare> squares = new Vector<GridSquare>();
188         Point p = getCoordinateXY();
189 
190         for(int x = -radius; x <= radius; x++) {
191 
192             int z;
193 
194 
195                 z = (int)(Math.sqrt(Math.pow(radius, 2) - Math.pow(x, 2)));
196 
197             for(int y = -z; y <= z; y++) {
198 
199                 try {
200 
201                     if(Grid.getInstance().getGridSquare(p.x + x, p.y + y).isEmpty())
202                         squares.add(Grid.getInstance().getGridSquare(p.x + x, p.y + y));
203 
204 
205                 } catch (Exception ex) {
206 
207                 }
208 
209             }
210 
211         }
212 
213         if(radius == 1) {
214             try {
215 
216                 squares.add(Grid.getInstance().getGridSquare(p.x - 1, p.y - 1));
217                 squares.add(Grid.getInstance().getGridSquare(p.x + 1, p.y + 1));
218                 squares.add(Grid.getInstance().getGridSquare(p.x + 1, p.y - 1));
219                 squares.add(Grid.getInstance().getGridSquare(p.x - 1, p.y + 1));
220 
221             } catch (Exception ex) {
222 
223             }
224 
225         }
226 
```

```
227         return squares;
228     }
229 
230     public static int getNextDirection(int direction) {
231 
232         int newDirection = direction == LAST_DIRECTION ? FIRST_DIRECTION : direction + 1;
233 
234         return newDirection;
235     }
236 
237     public static int getPreviousDirection(int direction) {
238 
239         int newDirection = direction == FIRST_DIRECTION ? LAST_DIRECTION : direction - 1;
240 
241         return newDirection;
242     }
243 
244     public void draw(Graphics g) {
245           Graphics2D g2d = (Graphics2D)g;
246 
247          int green = (int)(foodStrength * 75) < 255 ? (int)(foodStrength * 75) : 255;
248 
249 
250          Color old = g2d.getColor();
251         if(foodStrength >0 && green >= 0) {
252 
253             g2d.setColor( new Color(0, green, 0));
254 
255             g2d.fill(this);
256 
257             g2d.draw(this);
258             g2d.setColor(old);
259         }
260 
261         if(pheromone != null && pheromone.getStrength() != 0) {
262             pheromone.draw(g);
263         }
264 
265         if(object != null) {
266             object.draw(g);
267         }
268 
269 
270          g2d.setColor(old);
271 
272     }
273 
274     public boolean empty() {
275         return object == null && getAnt() == null;
276     }
277 
278     public MyObject getObject() {
279         return object;
280     }
281 
282     public void setObject(MyObject o) {
283         if(o instanceof Pheromone) {
284             setPheromone((Pheromone)o);
285             return;
286         }
287 
288         if(!isNest() && !(object instanceof Obstacle))
289             object = o;
290 
291         if(o instanceof Obstacle) {
292             setFoodStrength(0);
293             getPheromone().setStrength(0);
294         }
295     }
296 
297     public Ant getAnt() {
298        return object instanceof Ant ? (Ant)object : null;
299     }
300 
301     public boolean containsAnt() {
302         return !empty() && object instanceof Ant;
303     }
```

```
304
305     public void setAnt(Ant ant) {
306         this.object = ant;
307         this.ant = ant;
308     }
309
310     public Food getFood() {
311         return object instanceof Food ? (Food)object : null;
312     }
313
314     public Obstacle getObstacle() {
315         return object instanceof Obstacle ? (Obstacle)object : null;
316     }
317
318     public boolean containsFood() {
319         return getFood() != null;
320     }
321
322     public boolean containsObstacle() {
323         return getObstacle() != null;
324     }
325
326     public void setFood(Food food) {
327             this.object = food;
328     }
329
330     public boolean isNest() {
331         return object != null && object instanceof Nest;
332     }
333
334     public Pheromone getPheromone() {
335         if(pheromone == null) pheromone = new Pheromone(this);
336         return pheromone;
337     }
338
339     public double getPheromoneStrength() {
340         return pheromone == null ? 0 : getPheromone().getStrength();
341     }
342
343     public void setPheromone(Pheromone pheromone) {
344         this.pheromone = pheromone;
345     }
346
347     public synchronized double getFoodStrength() {
348         return foodStrength;
349     }
350
351     public void recalculateFoodStrength() {
352
353         recalculateFoodStrength(true);
354     }
355
356     public void recalculateFoodStrength(boolean add) {
357
358         if(getFood() == null) return;
359
360         Point thisP = getCoordinateXY();
361
362         int radius = 15;
363
364         GridVector surroundingSquares = this.getGridSquares(radius);
365
366         for(GridSquare gs : surroundingSquares) {
367
368                 Point p = gs.getCoordinateXY();
369
370                 double c2 = Math.pow( thisP.x - p.x , 2) + Math.pow( thisP.y - p.y , 2);
371
372                 double newStrength = gs.getFoodStrength();
373
374                 if(add) {
375                      newStrength += 1 / c2;
376                 }else{
377                     newStrength -= (1 / c2) + 0.0000001;
378
379                 }
380
```

```
381                gs.setFoodStrength(newStrength);
382
383        }
384
385
386    }
387
388
389    public synchronized void setFoodStrength(double foodStrength) {
390
391        if(foodStrength < 0) foodStrength = 0;
392        this.foodStrength = foodStrength;
393    }
394
395    public int compareTo(GridSquare gs) {
396        return FoodStrengthComparator.compare(this, gs);
397    }
398
399    public static Comparator<GridSquare> FoodStrengthComparator = new Comparator<GridSquare>() {
400        public int compare(GridSquare gs1, GridSquare gs2) {
401            return gs1.getFoodStrength() > gs2.getFoodStrength() ? -1 : 1;
402        }
403    };
404
405              public    static    Comparator<GridSquare>    PheromoneStrengthComparator    =    new
Comparator<GridSquare>() {
406        public int compare(GridSquare gs1, GridSquare gs2) {
407                                  double   pheromoneStrength1   =   gs1.getPheromone()   !=   null   ?
gs1.getPheromone().getStrength() : 0;
408                                  double   pheromoneStrength2   =   gs2.getPheromone()   !=   null   ?
gs2.getPheromone().getStrength() : 0;
409            return java.lang.Double.compare(pheromoneStrength1, pheromoneStrength2);
410        }
411    };
412
413    public String toString() {
414        Point p = this.getCoordinateXY();
415        String s = "";
416        s += getPheromone().toString();
417        return s;
418    }
419
420
421}
422
```

MyObject.java

```
/*
 * MyObject.java
 *
 * Created on 12 October 2006, 22:50
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */

package ants;

import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Rectangle;
import java.awt.event.MouseListener;

/**
 *
 * @author James Hamilton
 */
public abstract class MyObject extends Rectangle implements MyObjectInterface {

    private GridSquare gridSquare;

    /** Creates a new instance of MyObject */
    public MyObject() {
        setSize(GridSquare.SIZE, GridSquare.SIZE);
    }

    public MyObject(GridSquare s) {
        setGridSquare(s);
        setSize(s.getSize());
        setLocation(s.getLocation());
    }

    public void draw(Graphics g) {

        Graphics2D g2d = (Graphics2D)g;

        g2d.setColor( getColor());

        g2d.fill(this);

        g2d.draw(this);
    }

    public GridSquare getGridSquare() {
        return gridSquare;
    }

    public void setGridSquare(GridSquare gridSquare) {
        if(this.gridSquare != null) this.gridSquare.setObject(null);
        this.gridSquare = gridSquare;

        if(gridSquare == null) {
            setLocation(-99, -99);

        }else{
            setLocation(gridSquare.getLocation());
//            if(this instanceof Ant)
//               gridSquare.setAnt((Ant)this);
//            else
              gridSquare.setObject(this);
        }
    }
}

```

GridVector.java

```
 1/*
 2 * GridVector.java
 3 *
 4 * Created on 24 November 2006, 17:55
 5 *
 6 * To change this template, choose Tools | Template Manager
 7 * and open the template in the editor.
 8 */
 9
10package ants;
11
12/**
13 *
14 * @author James Hamilton
15 */
16public class GridVector extends java.util.Vector<GridSquare> {
17
18    /** Creates a new instance of GridVector */
19    public GridVector() {
20        super();
21    }
22
23    public double getFoodStrength() {
24        double sum = 0;
25        for(GridSquare gs : this)
26            sum += gs.getFoodStrength();
27        return sum;
28    }
29
30    public double getPheromoneStrength() {
31        double sum = 0;
32        for(GridSquare gs : this)
33            sum += gs.getPheromoneStrength();
34        return sum;
35    }
36
37    public int getPheromoneStrength(int direction) {
38        int sum = 0;
39        for(GridSquare gs : this)
40            if(gs.getPheromone().getDirection() == direction)
41                sum += gs.getPheromoneStrength();
42        return sum;
43    }
44
45    public boolean containsFood() {
46        return getFoodGridSquares().size() != 0;
47    }
48
49    public GridVector getGridSquaresInDirection(GridSquare gs1, int direction) {
50        GridVector gridVector = new GridVector();
51        for(GridSquare gs : this)
52            if(gs1.getDirectionOf(gs) == direction ||
53                gs1.getDirectionOf(gs) == GridSquare.getPreviousDirection(direction) ||
54                gs1.getDirectionOf(gs) == GridSquare.getNextDirection(direction))
55                gridVector.add(gs);
56        return gridVector;
57    }
58
59    public GridVector getFoodGridSquares() {
60        GridVector gridVector = new GridVector();
61        for(GridSquare gs : this)
62            if(gs.containsFood())
63                gridVector.add(gs);
64        return gridVector;
65    }
66
67    public GridVector getFoodStrengthGridSquares() {
68        GridVector gridVector = new GridVector();
69        for(GridSquare gs : this)
70            if(gs.getFoodStrength() > 0)
71                gridVector.add(gs);
72        return gridVector;
73    }
```

```
74
75     public boolean containsNest() {
76         GridVector gridVector = new GridVector();
77         for(GridSquare gs : this)
78             if(gs.isNest())
79                 return true;
80         return false;
81     }
82
83     public GridVector getPheromoneGridSquares() {
84         GridVector gridVector = new GridVector();
85         for(GridSquare gs : this)
86             if(gs.getPheromoneStrength() > 0)
87                 gridVector.add(gs);
88         return gridVector;
89     }
90
91     public GridVector getPheromoneGridSquares(int direction) {
92         GridVector gridVector = new GridVector();
93         for(GridSquare gs : this)
94             if(gs.getPheromoneStrength() > 0 && gs.getPheromone().getDirection() == direction)
95                 gridVector.add(gs);
96         return gridVector;
97     }
98
99     public GridVector getPheromoneGridSquares(int direction, Ant ant) {
100        GridVector gridVector = new GridVector();
101        for(GridSquare gs : this)
102            if(gs.getPheromoneStrength() > 0 &&
103            gs.getPheromone().getDirection() == direction &&
104            gs.getPheromone().laidBy(ant))
105                    gridVector.add(gs);
106        return gridVector;
107    }
108
109    public GridSquare getGridSquare(Ant ant) {
110        for(GridSquare gs : this)
111            if(gs.getPheromoneStrength() > 0 && gs.getPheromone().laidBy(ant))
112                return gs;
113        return null;
114    }
115
116    public GridVector getPheromoneGridSquares(Ant ant, boolean own) {
117        GridVector gridVector = new GridVector();
118        if(own) {
119            for(GridSquare gs : this)
120                if(gs.getPheromoneStrength() > 0 && gs.getPheromone().laidBy(ant))
121                    gridVector.add(gs);
122        }else{
123             for(GridSquare gs : this)
124                if(gs.getPheromoneStrength() > 0 && !gs.getPheromone().laidBy(ant))
125                    gridVector.add(gs);
126
127        }
128        return gridVector;
129    }
130
131    public GridVector getPheromoneGridSquares(Ant ant) {
132        return getPheromoneGridSquares(ant, true);
133    }
134
135    public GridVector getFreeGridSquares() {
136        GridVector gridVector = new GridVector();
137
138        for(GridSquare gs : this)
139            if(!gs.containsObstacle() && !gs.containsAnt())
140                gridVector.add(gs);
141        return gridVector;
142    }
143
144    public GridVector getEmptyGridSquares() {
145        GridVector gridVector = new GridVector();
146        for(GridSquare gs : this)
147            if(gs.getObject() == null || gs.isNest())
148                gridVector.add(gs);
149        return gridVector;
150    }
```

```
151
152    public GridVector getGridSquaresWithoutNest() {
153        GridVector gridVector = new GridVector();
154        for(GridSquare gs : this)
155            if(!gs.isNest())
156                gridVector.add(gs);
157        return gridVector;
158    }
159
160    public GridSquare getRandomGridSquare() {
161        return elementAt((int)(Math.random() * size()));
162    }
163
164    public GridSquare getRandomGridSquare(boolean bias) {
165              return  elementAt((int)(Math.random()  *  Math.sqrt(size()))  *  (int)(Math.random()
*Math.sqrt(size())));
166    }
167
168    public GridSquare getMiddleGridSquare() {
169        return elementAt((int)Math.floor(size() / 2));
170    }
171
172    public GridSquare firstElementNot(GridSquare otherGS) {
173
174        for(GridSquare gs : this)
175            if(!gs.equals(otherGS))
176                 return gs;
177       return firstElement();
178    }
179
180    public void addObstacles() {
181        for(GridSquare gs : this)
182            gs.setObject(new Obstacle(gs));
183    }
184
185    public void addFood() {
186        for(GridSquare gs : this)
187            gs.setObject(new Food(gs));
188    }
189}
190
```

MyObjectInterface.java

```
 1/*
 2 * MyObject.java
 3 *
 4 * Created on 09 October 2006, 00:39
 5 *
 6 * To change this template, choose Tools | Template Manager
 7 * and open the template in the editor.
 8 */
 9
10package ants;
11
12import java.awt.Color;
13
14/**
15 *
16 * @author James Hamilton
17 */
18public interface MyObjectInterface {
19    public Color getColor();
20}
21
```

Nest.java

```
  1/*
  2 * Nest.java
  3 *
  4 * Created on 13 October 2006, 22:47
  5 *
  6 * To change this template, choose Tools | Template Manager
  7 * and open the template in the editor.
  8 */
  9
 10package ants;
 11
 12import ants.control.AntControl;
 13import ants.event.EnterNestAntEvent;
 14import ants.event.ExitNestAntEvent;
 15import ants.event.NewAntEvent;
 16import java.awt.Color;
 17import java.util.Vector;
 18
 19/**
 20 *
 21 * @author James Hamilton
 22 */
 23public class Nest extends MyObject implements Runnable {
 24
 25    private Vector<Ant> ants = new Vector<Ant>();
 26    private Vector<Food> food = new Vector<Food>();
 27
 28    private int totalAnts = 0;
 29
 30    private Thread move;
 31
 32    /** Creates a new instance of Nest */
 33    public Nest() {
 34
 35    }
 36
 37    public Nest(GridSquare gs) {
 38        setGridSquare(gs);
 39        move = new Thread(this);
 40        move.setDaemon(true);
 41        move.start();
 42    }
 43
 44    public Nest(int n, GridSquare gs) {
 45        this(gs);
 46        addAnts(n);
 47
 48        totalAnts = n;
 49    }
 50
 51    public void addAnt() {
 52        Ant ant = new Ant(getGridSquare());
 53        ants.add(ant);
 54        ant.addAntListener(AntControl.getInstance());
 55        ant.notifyListeners(new NewAntEvent(ant));
 56        totalAnts++;
 57        AntControl.getInstance().setTotalAnts(totalAnts);
 58    }
 59
 60    public void addAnts(int n) {
 61        for(int i = 0; i < n; i++)
 62            addAnt();
 63    }
 64
 65    public void releaseAnt() {
 66        if(ants.size() != 0) {
 67        //    System.out.println("Releasing Ant..");
 68
 69                                                    Vector<GridSquare>    surroundingSquares    =
getGridSquare().getGridSquares(1).getFreeGridSquares();
 70
 71           // int n = (int)(surroundingSquares.size() * Math.random());
 72
 73
 74            Ant ant = ants.remove(0);
```

```
 75          //  ant.setGridSquare(surroundingSquares.elementAt(n));
 76           // surroundingSquares.elementAt(n).setAnt(ant);
 77            ant.start();
 78            ant.notifyListeners(new ExitNestAntEvent(ant));
 79
 80        }
 81    }
 82
 83    public void addAnt(Ant ant) {
 84        food.add(ant.getFood());
 85        ant.setFood(null);
 86        ants.add(ant);
 87        System.out.println("ANT Returned to NEST");
 88        ant.stop();
 89        ant.notifyListeners(new EnterNestAntEvent(ant));
 90    }
 91
 92    public void run() {
 93
 94        while(move != null) {
 95            try {
 96
 97                releaseAnt();
 98              //  System.out.println(ants.size() + " ants in nest");
 99                move.sleep((int)(10000 * Math.random()) + 1);
100            }catch (InterruptedException e) {}
101        }
102    }
103
104
105    public Vector<Ant> getAnts() {
106        return ants;
107
108    }
109
110    public Color getColor() {
111        return Color.LIGHT_GRAY;
112    }
113
114    public String toString() {
115        return "Nest";
116    }
117
118    public int getTotalAnts() {
119        return totalAnts;
120    }
121}
122
```

Obstacle.java

```
 1/*
 2 * Obstacle.java
 3 *
 4 * Created on 15 October 2006, 18:49
 5 *
 6 * To change this template, choose Tools | Template Manager
 7 * and open the template in the editor.
 8 */
 9
10package ants;
11
12import java.awt.Color;
13
14/**
15 *
16 * @author James Hamilton
17 */
18public class Obstacle extends MyObject {
19
20    /** Creates a new instance of Obstacle */
21    public Obstacle(GridSquare gs) {
22        super(gs);
23    }
24
25    public Color getColor() {
26        return Color.PINK;
27    }
28}
29
```

Pheromone.java

```
 1/*
 2 * Pheromone.java
 3 *
 4 * Created on 15 October 2006, 23:31
 5 *
 6 * To change this template, choose Tools | Template Manager
 7 * and open the template in the editor.
 8 */
 9
10package ants;
11
12import ants.PheromoneParticle;
13import java.awt.Color;
14import java.awt.Point;
15import java.util.HashMap;
16import java.util.Vector;
17
18/**
19 *
20 * @author James Hamilton
21 */
22public class Pheromone extends MyObject implements Runnable {
23
24
25    private Vector<PheromoneParticle> particles = new Vector<PheromoneParticle>();
26
27
28    private double strength = 0;
29    private Thread move;
30    private int rate = 90000;
31    private boolean go = true;
32    private int maxStrength = 3;
33
34    private int direction = -1;
35
36    public static final int AWAY_FROM_NEST = 1;
37    public static final int TOWARD_NEST = 2;
38
39    /** Creates a new instance of Pheromone */
40    public Pheromone() {
41        setStrength(0);
42                    move = new Thread(this);
43            move.setDaemon(true);
44            move.start();
45    }
46
47    public Pheromone(GridSquare gs) {
48        super(gs);
49        move = new Thread(this);
50        move.setDaemon(true);
51        move.start();
52        setStrength(0);
53
54    }
55
56    public void run() {
57        while(direction != -1) {
58
59            try {
60
61                    move.sleep(rate);//strength * 1000);
62                    if(particles.size() > 0)  particles.remove(0);
63
64            }catch (InterruptedException e) {
65                System.out.println(e);
66            }
67
68        }
69    }
70
71
72    public void increaseStrength(Ant ant, int direction) {
73        particles.add(new PheromoneParticle(ant, direction));
74    }
75
```

```
 76
 77
 78    public void increaseStrength(int direction) {
 79        strength += 0.5;
 80        Point thisP = getGridSquare().getCoordinateXY();
 81        GridVector squares = getGridSquare().getGridSquares(5);
 82        for(GridSquare s : squares) {
 83             Point p = s.getCoordinateXY();
 84             double c2 = Math.pow( thisP.x - p.x , 2) + Math.pow( thisP.y - p.y , 2);
 85
 86             s.getPheromone().increaseStrength(1/(c2*500));
 87        }
 88
 89        if(strength > maxStrength) strength = maxStrength;
 90    }
 91
 92    public void increaseStrength() {
 93
 94        strength += 0.5;
 95        Point thisP = getGridSquare().getCoordinateXY();
 96
 97        GridVector squares = getGridSquare().getGridSquares(5);
 98        for(GridSquare s : squares) {
 99             Point p = s.getCoordinateXY();
100             double c2 = Math.pow( thisP.x - p.x , 2) + Math.pow( thisP.y - p.y , 2);
101
102             s.getPheromone().increaseStrength(1/(c2*500));
103        }
104
105         if(strength > maxStrength) strength = maxStrength;
106    }
107
108    public void increaseStrength(double n) {
109
110        strength += n;
111         if(strength > maxStrength) strength = maxStrength;
112    }
113
114    public Color getColor() {
115
116        int direction = getDirection();
117        if(direction == -1) {
118            return Color.black;
119        }else if(direction == PheromoneParticle.AWAY_FROM_NEST) {
120            return Color.blue;
121        }else{
122            return Color.green;
123        }
124    }
125
126    public double getStrength() {
127        return particles.size();
128    }
129
130    public void setStrength(int strength) {
131        if(strength > 10) strength = 10;
132        this.strength = strength;
133    }
134
135    public void start() {
136        go = true;
137        if(move == null) {
138                             move = new Thread(this);
139            move.setDaemon(true);
140            move.start();
141        }
142
143    }
144
145    public void stop() {
146        go = false;
147        strength = 0;
148    }
149
150    public int getDirection() {
151         if(particles.size() == 0) return -1;
152
```

```
153         int toward_nest = 0;
154         int away_from_nest = 0;
155 
156         for(PheromoneParticle p : particles) {
157             if(p.getDirection() == PheromoneParticle.AWAY_FROM_NEST)
158                 away_from_nest++;
159             else if(p.getDirection() == PheromoneParticle.TOWARD_NEST)
160                 toward_nest++;
161         }
162 
163                     return  toward_nest  >  away_from_nest  ?  PheromoneParticle.TOWARD_NEST  :
PheromoneParticle.AWAY_FROM_NEST;
164     }
165 
166     public boolean laidBy(Ant ant) {
167         for(PheromoneParticle p : particles) {
168             if(p.getAnt().equals(ant)) return true;
169         }
170 
171         return false;
172     }
173 
174     public int getDirection(Ant ant) {
175         if(particles.size() == 0) return -1;
176 
177         int toward_nest = 0;
178         int away_from_nest = 0;
179 
180         for(int i = particles.size(); i >= 0; i--) {
181             PheromoneParticle p = particles.elementAt(i);
182             if(!p.getAnt().equals(ant)) continue;
183 
184             if(p.getDirection() == PheromoneParticle.AWAY_FROM_NEST)
185                 away_from_nest++;
186             else if(p.getDirection() == PheromoneParticle.TOWARD_NEST)
187                 toward_nest++;
188         }
189 
190                     return  toward_nest  >  away_from_nest  ?  PheromoneParticle.TOWARD_NEST  :
PheromoneParticle.AWAY_FROM_NEST;
191     }
192 
193     public void setDirection(int direction) {
194 
195         this.direction = direction;
196     }
197 
198     public String toString() {
199         int toward_nest = 0;
200         int away_from_nest = 0;
201 
202         for(PheromoneParticle p : particles) {
203             if(p.getDirection() == PheromoneParticle.AWAY_FROM_NEST)
204                 away_from_nest++;
205             else if(p.getDirection() == PheromoneParticle.TOWARD_NEST)
206                 toward_nest++;
207         }
208              return "T: " + toward_nest  + ", A: " + away_from_nest + ", direction: " +
getDirection();
209     }
210}
211 
```

PheromonesParticle.java

```
 1/*
 2 * PheromoneParticle.java
 3 *
 4 * Created on 16 February 2007, 02:51
 5 *
 6 * To change this template, choose Tools | Template Manager
 7 * and open the template in the editor.
 8 */
 9
10package ants;
11
12/**
13 *
14 * @author James Hamilton
15 */
16public class PheromoneParticle {
17
18    private Ant ant = null;
19    private int direction = AWAY_FROM_NEST;
20
21    public static final int TOWARD_NEST = 1;
22    public static final int AWAY_FROM_NEST = 2;
23
24    /** Creates a new instance of PheromoneParticle */
25    public PheromoneParticle(Ant ant, int direction) {
26        this.setAnt(ant);
27        this.setDirection(direction);
28    }
29
30    public Ant getAnt() {
31        return ant;
32    }
33
34    public void setAnt(Ant ant) {
35        this.ant = ant;
36    }
37
38    public int getDirection() {
39        return direction;
40    }
41
42    public void setDirection(int direction) {
43        this.direction = direction;
44    }
45
46}
47
```

AntEvent.java

```
 1/*
 2 * AntEvent.java
 3 *
 4 * Created on 28 November 2006, 13:39
 5 *
 6 * To change this template, choose Tools | Template Manager
 7 * and open the template in the editor.
 8 */
 9
10package ants.event;
11
12import java.util.EventObject;
13
14/**
15 *
16 * @author James Hamilton
17 */
18public class AntEvent extends EventObject {
19
20    /** Creates a new instance of AntEvent */
21    public AntEvent(Object source) {
22        super(source);
23    }
24
25}
26
```

AliveAntEvent.java

```
 1/*
 2 * AliveAntEvent.java
 3 *
 4 * Created on 28 November 2006, 14:19
 5 *
 6 * To change this template, choose Tools | Template Manager
 7 * and open the template in the editor.
 8 */
 9
10package ants.event;
11
12/**
13 *
14 * @author James Hamilton
15 */
16public class AliveAntEvent extends AntEvent {
17    public AliveAntEvent(Object source) { super(source); }
18}
19
```

AntListener.java

```
 1/*
 2 * AntListener.java
 3 *
 4 * Created on 28 November 2006, 13:38
 5 *
 6 * To change this template, choose Tools | Template Manager
 7 * and open the template in the editor.
 8 */
 9
10package ants.event;
11
12import java.util.EventListener;
13
14/**
15 *
16 * @author James Hamilton
17 */
18public interface AntListener extends EventListener {
19    public void antEventHandler(AntEvent e);
20}
```

EnterNestAntEvent.java

```
 1/*
 2 * EnterNestAntEvent.java
 3 *
 4 * Created on 28 November 2006, 14:40
 5 *
 6 * To change this template, choose Tools | Template Manager
 7 * and open the template in the editor.
 8 */
 9
10package ants.event;
11
12/**
13 *
14 * @author James Hamilton
15 */
16public class EnterNestAntEvent extends AntEvent {
17    public EnterNestAntEvent(Object source) { super(source); }
18}
```

ExitNestAntEvent.java

```
 1/*
 2 * ExitNestAntEvent.java
 3 *
 4 * Created on 28 November 2006, 14:40
 5 *
 6 * To change this template, choose Tools | Template Manager
 7 * and open the template in the editor.
 8 */
 9
10package ants.event;
11
12/**
13 *
14 * @author James Hamilton
15 */
16public class ExitNestAntEvent extends AntEvent {
17    public ExitNestAntEvent(Object source) { super(source); }
18}
```

FoodUpdateEvent.java

```
 1/*
 2 * FoodUpdateEvent.java
 3 *
 4 * Created on 29 November 2006, 14:50
 5 *
 6 * To change this template, choose Tools | Template Manager
 7 * and open the template in the editor.
 8 */
 9
10package ants.event;
11
12/**
13 *
14 * @author James Hamilton
15 */
16public class FoodUpdateEvent extends AntEvent {
17    public FoodUpdateEvent(Object source) { super(source); }
18}
```

NewAntEvent.java

```
 1/*
 2 * NewAntEvent.java
 3 *
 4 * Created on 28 November 2006, 14:11
 5 *
 6 * To change this template, choose Tools | Template Manager
 7 * and open the template in the editor.
 8 */
 9
10package ants.event;
11
12/**
13 *
14 * @author James Hamilton
15 */
16public class NewAntEvent extends AntEvent {
17    public NewAntEvent(Object source) { super(source); }
18}
```

PickedUpFoodEvent.java

```
 1/*
 2 * PickedUpFoodAntEvent.java
 3 *
 4 * Created on 28 November 2006, 14:45
 5 *
 6 * To change this template, choose Tools | Template Manager
 7 * and open the template in the editor.
 8 */
 9
10package ants.event;
11
12/**
13 *
14 * @author James Hamilton
15 */
16public class PickedUpFoodAntEvent extends AntEvent {
17    public PickedUpFoodAntEvent(Object source) { super(source); }
18}
```

AntControl.java

Note: this class contains code generated by Netbeans for the GUI in between user code (line 142-327, lines 405-426 are fields created by Netbeans).

```
  1/*
  2 * AntControl.java
  3 *
  4 * Created on 28 November 2006, 00:49
  5 */
  6
  7package ants.control;
  8
  9import ants.Ant;
 10import ants.Grid;
 11import ants.event.AntEvent;
 12import ants.event.AntListener;
 13import ants.event.EnterNestAntEvent;
 14import ants.event.ExitNestAntEvent;
 15import ants.event.FoodUpdateEvent;
 16import ants.event.NewAntEvent;
 17import ants.event.PickedUpFoodAntEvent;
 18import java.awt.event.ActionEvent;
 19import java.awt.event.ActionListener;
 20import javax.swing.DefaultListModel;
 21
 22/**
 23 *
 24 * @author  James Hamilton
 25 */
 26public class AntControl extends javax.swing.JFrame implements AntListener, Runnable {
 27
 28    private static AntControl singleton = null;
 29
 30    private int totalAnts = 0;
 31    private int antsInNest = 0;
 32    private int antsRoaming = 0;
 33    private int antsCarryingFood = 0;
 34
 35    private int totalFood = 0;
 36    private int collectedFood = 0;
 37    private int remainingFood = 0;
 38
 39    private long startTime = System.currentTimeMillis();
 40
 41    private boolean running = true;
 42
 43    private Thread move;
 44    private DefaultListModel myAnts = new DefaultListModel();
 45    /** Creates new form AntControl */
 46    public AntControl() {
 47        initComponents();
 48
 49        this.setLocation(500, 0);
 50
 51       btnPause.addActionListener( new ActionListener() {
 52            public void actionPerformed(ActionEvent e) {
 53                for(Ant a : Grid.getInstance().getNest().getAnts()) {
 54                    a.pause();
 55                }
 56            }
 57        });
 58
 59        move = new Thread(this);
 60        move.setDaemon(true);
 61        move.start();
 62
 63
 64    }
 65
 66    public void antEventHandler(AntEvent event) {
 67        if(event instanceof NewAntEvent) {
 68            totalAnts++;
 69            antsInNest++;
 70            Ant a = (Ant)event.getSource();
 71            myAnts.addElement(a);
```

```
  72            lstAnts.setModel(myAnts);
  73
  74        }else if(event instanceof ExitNestAntEvent) {
  75            antsRoaming++;
  76            antsInNest--;
  77        }else if(event instanceof EnterNestAntEvent) {
  78            antsCarryingFood--;
  79            antsInNest++;
  80            collectedFood++;
  81                                                                                remainingFood      =
Grid.getInstance().getGridSquares().getFoodGridSquares().size();//totalFood - collectedFood;
  82        }else if(event instanceof PickedUpFoodAntEvent) {
  83            antsCarryingFood++;
  84            antsRoaming--;
  85        }else if(event instanceof FoodUpdateEvent) {
  86            totalFood = Grid.getInstance().getGridSquares().getFoodGridSquares().size();
  87            remainingFood = totalFood;
  88        }
  89
  90        txtNumberOfAnts.setText(Integer.toString(getTotalAnts()));
  91        txtNumberOfAntsInNest.setText(Integer.toString(getAntsInNest()));
  92        txtNumberOfAntsCarryingFood.setText(Integer.toString(getAntsCarryingFood()));
  93        txtNumberOfAntsRoaming.setText(Integer.toString(getAntsRoaming()));
  94
  95        txtTotalFood.setText(Integer.toString(getTotalFood()));
  96        txtCollectedFood.setText(Integer.toString(getCollectedFood()));
  97        txtRemainingFood.setText(Integer.toString(getRemainingFood()));
  98         if(remainingFood == 0) running = false;
  99         else running = true;
 100    }
 101
 102    public void run() {
 103        while(running) {
 104            long currentTime = (System.currentTimeMillis() - startTime);
 105
 106            txtTime.setText(millisecondsToString(currentTime));
 107
 108            try {
 109
 110                move.sleep(1000);
 111            }catch (InterruptedException e) {
 112
 113            }
 114        }
 115    }
 116
 117    public static String millisecondsToString(long time) {
 118        //http://www.uk-dave.com/bytes/java/long2time.php
 119        int milliseconds = (int)(time % 1000);
 120        int seconds = (int)((time/1000) % 60);
 121        int minutes = (int)((time/60000) % 60);
 122        int hours = (int)((time/3600000) % 24);
 123             String millisecondsStr = (milliseconds<10 ? "00" : (milliseconds<100 ? "0" :
""))+milliseconds;
 124        String secondsStr = (seconds<10 ? "0" : "")+seconds;
 125        String minutesStr = (minutes<10 ? "0" : "")+minutes;
 126        String hoursStr = (hours<10 ? "0" : "")+hours;
 127        return new String(hoursStr+":"+minutesStr+":"+secondsStr);//+"."+millisecondsStr);
 128    }
 129
 130    public static AntControl getInstance() {
 131        if(getSingleton() == null)
 132            setSingleton(new AntControl());
 133        return getSingleton();
 134    }
 135
 136    /** This method is called from within the constructor to
 137     * initialize the form.
 138     * WARNING: Do NOT modify this code. The content of this method is
 139     * always regenerated by the Form Editor.
 140     */
 141    // <editor-fold defaultstate="collapsed" desc=" Generated Code ">//GEN-BEGIN:initComponents
 142    private void initComponents() {
          CODE GENERATED BY NETBEANS FOR THE GUI HERE
 327    }// </editor-fold>//GEN-END:initComponents
 328
 329    /**
```

```
330     * @param args the command line arguments
331     */
332    public static void main(String...args) {
333        java.awt.EventQueue.invokeLater(new Runnable() {
334            public void run() {
335                new AntControl().setVisible(true);
336            }
337        });
338    }
339
340    public static AntControl getSingleton() {
341        return singleton;
342    }
343
344    public static void setSingleton(AntControl aSingleton) {
345        singleton = aSingleton;
346    }
347
348    public int getTotalAnts() {
349        return totalAnts;
350    }
351
352    public void setTotalAnts(int totalAnts) {
353        this.totalAnts = totalAnts;
354    }
355
356    public int getAntsInNest() {
357        return antsInNest;
358    }
359
360    public void setAntsInNest(int antsInNest) {
361        this.antsInNest = antsInNest;
362    }
363
364    public int getAntsRoaming() {
365        return antsRoaming;
366    }
367
368    public void setAntsRoaming(int antsRoaming) {
369        this.antsRoaming = antsRoaming;
370    }
371
372    public int getAntsCarryingFood() {
373        return antsCarryingFood;
374    }
375
376    public void setAntsCarryingFood(int antsCarryingFood) {
377        this.antsCarryingFood = antsCarryingFood;
378    }
379
380    public int getTotalFood() {
381        return totalFood;
382    }
383
384    public void setTotalFood(int totalFood) {
385        this.totalFood = totalFood;
386    }
387
388    public int getCollectedFood() {
389        return collectedFood;
390    }
391
392    public void setCollectedFood(int collectedFood) {
393        this.collectedFood = collectedFood;
394    }
395
396    public int getRemainingFood() {
397        return remainingFood;
398    }
399
400    public void setRemainingFood(int remainingFood) {
401        this.remainingFood = remainingFood;
402    }
403
404    // Variables declaration - do not modify//GEN-BEGIN:variables
405    private javax.swing.JButton btnPause;
406    private javax.swing.JLabel jLabel1;
```

```
407    private javax.swing.JLabel jLabel10;
408    private javax.swing.JLabel jLabel11;
409    private javax.swing.JLabel jLabel2;
410    private javax.swing.JLabel jLabel3;
411    private javax.swing.JLabel jLabel4;
412    private javax.swing.JLabel jLabel5;
413    private javax.swing.JLabel jLabel6;
414    private javax.swing.JLabel jLabel7;
415    private javax.swing.JLabel jLabel8;
416    private javax.swing.JLabel jLabel9;
417    private javax.swing.JScrollPane jScrollPane1;
418    private javax.swing.JList lstAnts;
419    private javax.swing.JTextField txtCollectedFood;
420    private javax.swing.JTextField txtNumberOfAnts;
421    private javax.swing.JTextField txtNumberOfAntsCarryingFood;
422    private javax.swing.JTextField txtNumberOfAntsInNest;
423    private javax.swing.JTextField txtNumberOfAntsRoaming;
424    private javax.swing.JTextField txtRemainingFood;
425    private javax.swing.JTextField txtTime;
426    private javax.swing.JTextField txtTotalFood;
427    // End of variables declaration//GEN-END:variables
428
429}
430
```

GraphJFrame.java

```
 1/*
 2 * GraphJFrame.java
 3 *
 4 * Created on 04 March 200dff7, 13:30
 5 * version 4
 6 * A JFrame for a GraphCanvas
 7 */
 8
 9package antgraph.gui;
10
11import antgraph.Edge;
12import antgraph.Graph;
13import antgraph.Nest;
14import antgraph.NoSuchNodeException;
15import antgraph.Node;
16import java.awt.event.KeyAdapter;
17import java.awt.event.KeyEvent;
18import java.awt.image.BufferedImage;
19import java.awt.print.PrinterJob;
20import java.io.File;
21
22import java.util.ArrayList;
23import java.util.HashMap;
24import java.util.List;
25import java.util.Map;
26import java.util.PriorityQueue;
27import java.util.Queue;
28import javax.imageio.ImageIO;
29import javax.swing.JFrame;
30
31/**
32 *
33 * @author James
34 */
35public class GraphJFrame extends JFrame {
36
37
38    private static GraphCanvas m;// = new GraphCanvas();
39
40    public static final int FOOD = 10;
41    public static final int ANTS = 4;
42
43    /** Creates a new instance of GraphJFrame */
44    public GraphJFrame() {
45        super("Ant Graph World");
46        Graph graph = getTestGraph();//new Graph();
47
48        m = new GraphCanvas(graph);
49      //  m.setGraph(graph);//.load(STANDARD_FILE)f
50
51
52        setSize((int)m.getSize().getWidth() + 13, (int)m.getSize().getHeight() + 35);
53        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
54        setResizable(false);
55        setVisible(true);
56        add(m);
57
58        m.addKeyListener( new KeyAdapter() {
59            public void keyPressed(KeyEvent e) {
60                if(e.getKeyChar() == 'p') {
61                    printGraphCanvas(m);
62                }else if(e.getKeyChar() == 'w') {
63                    for(Edge edge : m.getGraph().getEdges()) {
64                        edge.toggleWeights();
65                    }
66                }else if(e.getKeyChar() == 'h') {
67                    try {
68                        m.getGraph().highlightShortest(m.getGraph().getNestNode(), "l");
69                    }catch (NoSuchNodeException ex) {
70
71                    }
```

```
  72                }else if(e.getKeyChar() == 's') {
  73                  //  m.getGraph().getNest().start();
  74                }else if(e.getKeyChar() == 'r') {
  75                    m.setGraph(getTestGraph());
  76                }else if(e.getKeyChar() == 'n') {
  77                    try {
  78                                                              ImageIO.write(m.getBi(),   "png",   new
File("t_"+m.getGraph().getC()+".png"));
  79                    }catch(Exception ex) {
  80                        ex.printStackTrace();
  81                    }
  82                   m.getGraph().step();
  83                }
  84
  85            }
  86        });
  87       m.setFocusable(true);
  88
  89
  90
  91        initialise();
  92    }
  93
  94    private void initialise() {
  95
  96        m.getGraph().getNest().addAnts(ANTS);
  97        //   pfs(graph,  graph.getNodes().get(0),  graph.getNodes().get(graph.getNodes().size()  -
1));
  98
  99    }
 100
 101    public void printGraphCanvas(GraphCanvas gc) {
 102        PrinterJob printJob = PrinterJob.getPrinterJob ();
 103
 104        printJob.setPrintable(gc);
 105
 106        if (printJob.printDialog()) {
 107            try {
 108                printJob.print();
 109            } catch (Exception PrintException) {
 110                PrintException.printStackTrace();
 111            }
 112        }
 113
 114    }
 115
 116    public static Graph getTestGraph() {
 117
 118            Graph graph = new Graph();
 119
 120            Node a = new Node("a");
 121            a.setLocation(81, 130);
 122            Node b = new Node("b");
 123            b.setLocation(90, 225);
 124            Nest c = new Nest("N");
 125
 126            c.setLocation(165, 188);
 127            Node d = new Node("d");
 128            d.setLocation(231, 126);
 129            Node e = new Node("e");
 130            e.setLocation(357, 156);
 131            Node f = new Node("f");
 132             f.setLocation(289, 233);
 133            Node g = new Node("g");
 134             g.setLocation(401, 207);
 135            Node h = new Node("h");
 136             h.setLocation(215, 269);
 137            Node i = new Node("i");
 138             i.setLocation(320, 349);
 139            Node j = new Node("j");
 140             j.setLocation(381, 297);
 141            Node k = new Node("k");
 142             k.setLocation(219, 378);
 143            Node l = new Node("l");
 144             l.setLocation(103, 347);
 145
 146             graph.setCurrentChar('l');
```

```
147
148             graph.addNode(a);
149             graph.addNode(b);
150             graph.addNode(c);
151             graph.addNode(d);
152             graph.addNode(e);
153             graph.addNode(f);
154             graph.addNode(g);
155             graph.addNode(h);
156             graph.addNode(i);
157             graph.addNode(j);
158             graph.addNode(k);
159             graph.addNode(l);
160
161
162         l.addFood(FOOD);
163
164             try {
165                 graph.addEdge(a, c);
166                 graph.addEdge(b, c);
167                 graph.addEdge(c, d);
168                 graph.addEdge(c, h);
169                 graph.addEdge(d, e);
170                 graph.addEdge(d, f);
171                 graph.addEdge(f, g);
172                 graph.addEdge(f, j);
173                 graph.addEdge(g, j);
174                 graph.addEdge(j, i);
175                 graph.addEdge(i, k);
176                 graph.addEdge(h, k);
177                 graph.addEdge(l, k);
178             }catch (NoSuchNodeException ex) {
179                 System.out.println(ex);
180             }
181
182         return graph;
183     }
184
185     public static Graph getGraph() {
186         return m.getGraph();
187     }
188
189     public static void main(String[] args) {
190         java.awt.EventQueue.invokeLater(new Runnable() {
191             public void run() {
192                 new GraphJFrame();
193             }
194         });
195
196
197
198     }
199
200}
201
```

```
/*
 * GraphCanvas.java
 *
 * Created on 04 March 2007, 13:21
 *
 * the graph is drawn on this canvas. also listens for mouse events. the canvas is printable
 */

package antgraph.gui;

import antgraph.Ant;
import antgraph.Edge;
import antgraph.Graph;
import antgraph.GraphComponent;
import antgraph.NoSuchNodeException;
import antgraph.Node;
import java.awt.Canvas;
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.RenderingHints;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.event.MouseMotionAdapter;
import java.awt.image.BufferedImage;
import java.awt.print.PageFormat;
import java.awt.print.Printable;
import java.awt.print.PrinterException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

/**
 *
 * @author James
 */
public class GraphCanvas extends Canvas implements Runnable, Printable {

    private BufferedImage bi;
    private Graphics big;
    private Thread animate;

    public static final int WIDTH = 500, HEIGHT = 500;
    private Graph graph;
    private Node selectedNode = null;
    private Edge selectedEdge = null;

    private List<GraphComponent> selectedComponents = new ArrayList<GraphComponent>();
    private int c = 0;
    private boolean dragging = false;

    public GraphCanvas(Graph g) {
        setSize(WIDTH, HEIGHT);
        this.graph = g;

        //mouse listener

        this.addMouseListener( new MouseAdapter() {
           public void mousePressed(MouseEvent e) {

                if(e.getButton() == e.BUTTON1) {
                    if(containsNode(e.getX(), e.getY())) {
                        if(nodeSelected()) {
                            Node a = getSelectedNode();
                            Node b = nodeAt(e.getX(), e.getY());

                            try {
                                graph.addEdge(a, b);
                                deselectAll();
                            }catch (NoSuchNodeException ex) {
                                System.out.println(ex); //shouldn't happen
                            }
                        }else{
                            deselectAll();

                            setSelected(nodeAt(e.getX(), e.getY()));

```

```
 78                                                                  List<Edge>  edges  =
GraphJFrame.getGraph().getEdges(getSelectedNode());
 79
 80                              Collections.sort(edges);
 81                              System.out.println(edges);
 82                          }
 83                      }else if(containsEdge(e.getX(), e.getY())) {
 84
 85                           deselectAll();
 86
 87                           setSelected(edgeAt(e.getX(), e.getY()));
 88                      }else{
 89
 90                          if(nodeSelected()) {
 91
 92                                graph.addNode(e.getX(), e.getY());
 93                                System.out.println(nodeAt(e.getX(), e.getY()));
 94                                try {
 95                                   graph.addEdge(getSelectedNode(), nodeAt(e.getX(), e.getY()));
 96                                }catch (NoSuchNodeException ex) {
 97                                      System.out.println(ex); //shouldn't happen
 98                                }
 99
100                                 deselectAll();
101                          }else{
102
103                              graph.addNode(e.getX(), e.getY());
104                          }
105                      }
106                  }else if(e.getButton() == e.BUTTON3) {
107                      if(containsNode(e.getX(), e.getY())) {
108                          deselectAll();
109
110                          graph.removeNode(nodeAt(e.getX(), e.getY()));
111                      }else if(containsEdge(e.getX(), e.getY())) {
112                           deselectAll();
113
114                          graph.removeEdge(edgeAt(e.getX(), e.getY()));
115                      }else{
116
117
118                          //graph.highlight(graph.getNodes());
119                      }
120                  }
121
122
123              }
124
125
126              public void mouseReleased(MouseEvent e) {
127                  if(dragging) {
128                        deselectAll();
129                        dragging = false;
130                  }
131              }
132
133          });
134
135
136          addMouseMotionListener( new MouseMotionAdapter() {
137              public void mouseDragged(MouseEvent e) {
138
139                   if(componentSelected()) {
140                       dragging = true;
141                       if(nodeSelected()) {
142
143                           getSelectedNode().setLocation(e.getX(), e.getY());
144
145                       }
146                   }
147               }
148
149               public void mouseMoved(MouseEvent e) {
150                       dehighlight();
151
152                       GraphComponent gc =componentAt(e.getX(), e.getY());
153                                          if(!dragging  &&  gc  !=  null  &&  !gc.isSelected())
```

```
getGraph().highlight(gc);//gc.highlight(true);
 154
 155
 156             }
 157         });
 158
 159         //buffered image for double buffering
 160         bi = new BufferedImage(getWidth(),getHeight(),BufferedImage.TYPE_INT_RGB);
 161         big = getBi().createGraphics();
 162         animate = null;
 163         animate = new Thread(this);
 164         animate.start();
 165     }
 166
 167     public Graph getGraph() {
 168         return graph;
 169     }
 170
 171     public void setGraph(Graph graph) {
 172
 173         if(this.graph != null) {
 174             if(this.graph.getNest() != null) {
 175                 for(Ant ant : this.graph.getNest().getAnts()) {
 176
 177                     ant.stop();
 178                 }
 179                 for(Edge e : this.graph.getEdges()) {
 180                     e.getPheromone().stop();
 181                 }
 182
 183             }
 184         }
 185         this.graph = graph;
 186     }
 187
 188     public void update(Graphics g) {
 189         clear();
 190         big.setColor(Color.BLACK);
 191         if(graph.getNodes().size() > 0) {
 192
 193             if(graph.getEdges().size() > 0) {
 194
 195                 for(Edge e : graph.getEdges()) {
 196
 197                     e.draw(big);
 198
 199                 }
 200             }
 201
 202             for(Node n : graph.getNodes()) {
 203                 n.draw(big);
 204             }
 205
 206             if(graph.getNest().countFood() == GraphJFrame.FOOD) {
 207
 208             }
 209
 210         }
 211         g.drawImage(getBi(), 0, 0, this);
 212     }
 213
 214     public void paint(Graphics g) {
 215         update(g);
 216     }
 217
 218
 219     public void clear() {
 220         big.setColor(Color.WHITE);
 221         big.fillRect(0,0,getWidth(),getHeight());
 222     }
 223
 224     public void dehighlight() {
 225         for(GraphComponent gc : getGraph().getComponents())
 226                 gc.highlight(false);
 227     }
 228
 229     public void addSelected(GraphComponent gc) {
```

```
230         selectedComponents.add(gc);
231         gc.setSelected(true);
232     }
233 
234     public void setSelected(GraphComponent gc) {
235         deselectAll();
236         addSelected(gc);
237     }
238 
239     public boolean componentsSelected() {
240         return selectedComponents.size() > 0;
241     }
242 
243     public boolean componentSelected() {
244         return selectedComponents.size() == 1;
245     }
246 
247     public int countSelectedComponents() {
248         return selectedComponents.size();
249     }
250 
251     public List<GraphComponent> getSelectedComponents() {
252         return selectedComponents;
253     }
254 
255     public GraphComponent getSelectedComponent() {
256         return componentsSelected() ? selectedComponents.get(0) : null;
257     }
258 
259     public void deselectComponent(GraphComponent gc) {
260         selectedComponents.remove(gc);
261         gc.setSelected(false);
262     }
263 
264     public void run() {
265         //canvas is redrawn every 200 milliseconds.
266         while(animate!=null) {
267 
268             try {
269                 animate.sleep(200);
270 
271                 repaint();
272             }catch (Exception e) {
273                 System.out.println(e);
274             }
275 
276         }
277         big.dispose();
278     }
279 
280     public boolean containsNode(int x, int y) {
281         for(Node n : graph.getNodes())
282             if(n.contains(x, y))
283                 return true;
284         return false;
285     }
286 
287     public Node nodeAt(int x, int y) {
288         for(Node n : graph.getNodes())
289             if(n.contains(x, y))
290                 return n;
291         return null;
292     }
293 
294     public boolean containsEdge(int x, int y) {
295         for(Edge e : graph.getEdges())
296             if(e.contains(x, y))
297                 return true;
298         return false;
299     }
300 
301     public Edge edgeAt(int x, int y) {
302         for(Edge e : graph.getEdges())
303             if(e.contains(x, y))
304                 return e;
305         return null;
306     }
```

```
307
308    public GraphComponent componentAt(int x, int y) {
309        for(GraphComponent gc : getGraph().getComponents())
310            if(gc.contains(x, y))
311                return gc;
312        return null;
313    }
314
315    public void setSelectedNode(Node n) {
316
317        setSelected(n);
318        n.setSelected(true);
319
320    }
321
322    public Node getSelectedNode() {
323        return nodeSelected() ? (Node)getSelectedComponent() : null;
324    }
325
326
327    public boolean nodeSelected() {
328        return getSelectedComponent() != null ? getSelectedComponent() instanceof Node : false;
329    }
330
331    public void setSelectedEdge(Edge e) {
332        setSelected(e);
333        e.setSelected(true);
334
335    }
336    public Edge getSelectedEdge() {
337        return edgeSelected() ? (Edge)getSelectedComponent() : null;
338    }
339
340    public void deselectAll() {
341        for(GraphComponent gc : getSelectedComponents())
342            gc.setSelected(false);
343        getSelectedComponents().clear();
344    }
345
346    public boolean edgeSelected() {
347        return getSelectedComponent() != null ? getSelectedComponent() instanceof Edge : false;
348    }
349
350
351        public int print(Graphics graphics, PageFormat pageFormat, int pageIndex) throws
PrinterException {
352            /* printing */
353        if (pageIndex == 0) {
354            Graphics2D g2d = (Graphics2D) graphics;
355
356            g2d.setRenderingHint
357                    (RenderingHints.KEY_ANTIALIASING,
358                    RenderingHints.VALUE_ANTIALIAS_ON);
359
360            g2d.translate(pageFormat.getImageableX(), pageFormat.getImageableY());
361            paint(graphics);
362
363            int x = (int)pageFormat.getImageableX();
364            int y = (int)pageFormat.getHeight() / 2;
365            int h = (int)g2d.getFontMetrics().getHeight();
366
367            g2d.drawString("Adjacency List", x, y);
368            y += h + 3;
369            g2d.drawString("----------------------", x, y);
370            y += h + 3;
371            for(Node n : getGraph().getNodes()) {
372
373                g2d.drawString(n + ": " + getGraph().getAdjacencyList(n), x, y);
374
375                y += h + 3;
376
377            }
378
379
380            return PAGE_EXISTS;
381        }else{
382            return NO_SUCH_PAGE;
```

```
383         }
384     }
385 
386     public BufferedImage getBi() {
387         return bi;
388     }
389 }
390 
```

```
 1/*
 2 * GraphComponent.java
 3 *
 4 * Created on 07 March 2007, 00:20
 5 *
 6 * To change this template, choose Tools | Template Manager
 7 * and open the template in the editor.
 8 */
 9
10package antgraph;
11
12import java.awt.Color;
13import java.awt.Graphics;
14import java.awt.Graphics2D;
15import java.awt.Polygon;
16import java.awt.RenderingHints;
17
18/**
19 *
20 * @author James
21 */
22public abstract class GraphComponent extends Polygon {
23
24    public static final Color NORMAL = Color.BLACK;
25    public static final Color SELECTED = Color.GREEN;
26
27    private Color color = NORMAL;
28
29    private boolean selected = false;
30    private boolean highlighted = false;
31
32    /** Creates a new instance of GraphComponent */
33    public GraphComponent() {
34
35    }
36
37    public boolean isSelected() {
38        return selected;
39    }
40
41    public void setSelected(boolean selected) {
42        this.selected = selected;
43        color = selected ? SELECTED : NORMAL;
44    }
45
46    public void draw(Graphics g) {
47        update();
48        Graphics2D g2d = (Graphics2D)g;
49        g2d.setRenderingHint
50          (RenderingHints.KEY_ANTIALIASING,
51            RenderingHints.VALUE_ANTIALIAS_ON);
52
53        Color old = g2d.getColor();
54
55        g2d.setColor( getColor());
56
57        g2d.fill(this);
58
59        g2d.setColor(old);
60    }
61
62    public void highlight(boolean value) {
63        highlighted = value;
64    }
65
66    public boolean isHighlighted() {
67        return highlighted;
68    }
69
70    public Color getColor() {
71        if(!isSelected() && isHighlighted())
72            return Color.BLUE;
73        else
74            return color;
75    }
76
77    public abstract void update();
```

```
78
79}
80
```

```
 1/*
 2 * Ant.java
 3 *
 4 * Created on 04 March 2007, 01:34
 5
 6 * Represents a single ant
 7 *
 8 * TODO: improve move algorithm.
 9 *       remove need for previousNode (i.e. ant memory) - use pheromone clues instead.
10 */
11
12package antgraph;
13
14import antgraph.Pheromone.Direction;
15import antgraph.gui.GraphJFrame;
16import java.awt.event.KeyEvent;
17import java.awt.event.KeyListener;
18import java.awt.event.MouseEvent;
19import java.awt.event.MouseListener;
20import java.util.ArrayList;
21import java.util.Collections;
22import java.util.List;
23
24/**
25 *
26 * @author James
27 */
28public class Ant implements Runnable {
29
30    private Food food = null;
31    private Thread move;
32    private Node node = null;
33    private Node previousNode = null;
34
35    private char name;
36
37    private static int sleepRate = 10;
38    private static char nextchar = 'a';
39
40    private boolean justPickedUpFood = false;
41
42
43    /**
44     * Creates a new instance of Ant, sets is current node to Node and names it
45     * @param node The starting node for the ant
46     */
47    public Ant(Node node) {
48        this.node = node;
49        name = nextchar;
50        nextchar++;
51    }
52
53    /**
54     *
55     * @return true if the ant is carrying food
56     */
57    public boolean carryingFood() {
58        return food != null;
59    }
60
61    /**
62     *
63     * @return food that the ant is carrying or null
64     */
65    public Food getFood() {
66        return food;
67    }
68
69    /**
70     *
71     * @param food lets the ant pick up food
72     */
73    public void setFood(Food food) {
74        this.food = food;
75    }
76
77    /**
```

```
78     * moves the ant one step
79     */
80    public void move() {
81        if(GraphJFrame.getGraph().getNest().countFood() >= GraphJFrame.FOOD) stop();
82
83        //move!
84
85        Node next = null;
86        Edge chosen = null;
87
88        List<Edge> edges = GraphJFrame.getGraph().getEdges(getNode());
89
90                if(!justPickedUpFood  &&    getNode().countFood()  ==  0  &&  edges.size()  >  1)
edges.remove(GraphJFrame.getGraph().getEdge(getNode(), getPreviousNode()));
91
92        int pStrengthA = countPheromoneStrength(edges, Direction.AWAY_FROM_NEST);
93        int pStrengthT = countPheromoneStrength(edges, Direction.TOWARD_NEST);
94        int pStrength = pStrengthA + pStrengthT;
95
96
97        if(carryingFood()) {
98            //if the ant is carrying food - it wants to follow the strongest A trail back to the
nest.
99            if(pStrengthA > 0) {
100               Collections.sort(edges, Edge.AwayFromNestComparator);
101
102               chosen = edges.get(0);
103           }else if(pStrength == 0) {
104               chosen = (Edge)getRandomElement(edges);
105           }else{
106
107                 Collections.sort(edges, Collections.reverseOrder(Edge.TowardNestComparator));
108
109               chosen = edges.get(0);
110           }
111
112       }else{
113
114           if(pStrengthT > 0) {
115               Collections.sort(edges, Edge.TowardNestComparator);
116
117               chosen = edges.get(0);
118           }else if(pStrength == 0) {
119               chosen = (Edge)getRandomElement(edges);
120           }else{
121
122               Collections.sort(edges, Collections.reverseOrder(Edge.AwayFromNestComparator));
123
124               chosen = edges.get(0);
125
126           }
127
128       }
129
130                            chosen.getPheromone().increaseStrength(this,    carryingFood()    ?
Pheromone.Direction.TOWARD_NEST : Pheromone.Direction.AWAY_FROM_NEST);
131
132       setNode(chosen.other(getNode()));
133
134       if(getNode().containsFood() && !carryingFood() && !getNode().isNest()) {
135           this.setFood(getNode().removeFood());
136           justPickedUpFood = true;
137
138       }else{
139           justPickedUpFood = false;
140       }
141   }
142
143   /**
144    * should be moved somewhere else?
145    * @param list the list of edges to count pheromone strength from.
146    * @param direction the direction of the strength to count
147    * @return the pheromone strength for the direction from the specified list
148    */
149   public static int countPheromoneStrength(List<Edge> list, Direction direction) {
150       int count = 0;
151       for(Edge e : list) {
```

```
152                 count += e.getPheromone().getStrength(direction);
153         }
154         return count;
155     }
156 
157     /**
158      * returns a random element from the specified list
159      * @param l the list to choose from
160      * @return a random element from the list
161      */
162     public static Object getRandomElement(List l) {
163         return l.size() > 0 ? l.get((int)(Math.random() * l.size())) : null;
164     }
165 
166     /**
167      * moves the ant
168      */
169     public void run() {
170 
171         while(move != null) {
172             try {
173                 move();
174 
175                 move.sleep(getSleepRate());
176             }catch (InterruptedException e) {}
177         }
178     }
179 
180     /**
181      * start the ant moving
182      */
183     public void start() {
184         //Create and start a new Thread.
185         move = new Thread(this);
186         move.setDaemon(true);
187         move.start();
188     }
189 
190     /**
191      * stop the ant moving
192      */
193     public void stop() {
194         move = null;
195     }
196 
197     /**
198      *
199      * @return the rate at which the thread sleeps
200      */
201     public static int getSleepRate() {
202         return sleepRate;
203     }
204 
205     /**
206      *
207      * @param aSleepRate the rate at which the thread sleeps
208      */
209     public static void setSleepRate(int aSleepRate) {
210         sleepRate = aSleepRate;
211     }
212 
213     /**
214      *
215      * @return the ants current node
216      */
217     public Node getNode() {
218         return node;
219     }
220 
221     /**
222      *
223      * @param node set the ants current node
224      */
225     public void setNode(Node node) {
226 
227         previousNode = this.node;
228         if(this.node != null) {
```

```
229             this.node.removeAnt(this);
230         }
231         node.addAnt(this);
232         this.node = node;
233     }
234 
235     public String toString() {
236         if(carryingFood())
237             return Character.toString(name).toUpperCase();
238         else
239             return Character.toString(name);
240     }
241 
242     public Node getPreviousNode() {
243         return previousNode;
244     }
245}
246 
```

```
 1/*
 2 * Edge.java
 3 *
 4 * Created on 03 March 2007, 19:33
 5 *
 6 * To change this template, choose Tools | Template Manager
 7 * and open the template in the editor.
 8 */
 9
10package antgraph;
11
12import antgraph.Pheromone.Direction;
13import java.awt.Color;
14import java.awt.Graphics;
15import java.awt.Graphics2D;
16import java.awt.RenderingHints;
17import java.util.Comparator;
18
19/**
20 *
21 * @author James
22 */
23public class Edge extends GraphComponent implements Comparable<Edge> {
24
25    private Node a, b;
26    private Pheromone pheromone = new Pheromone();
27    public static final int THICKNESS = 7;
28    private double weight = 0;
29    private int centerX, centerY;
30    private boolean showWeights = false;
31
32    /** Creates a new instance of Edge */
33    public Edge(Node a, Node b) {
34        this.a = a;
35        this.b = b;
36    }
37
38    public Node a() {
39        return a;
40    }
41
42    public Node b() {
43        return b;
44    }
45
46    public boolean from(Node n) {
47        return n.equals(a);
48    }
49
50    public Node other(Node n) {
51        return n.equals(a) ? b : a;
52    }
53
54    public int hashCode() {
55        return a().hashCode() + b().hashCode();
56    }
57
58    public boolean equals(Object o) {
59        return (o instanceof Edge &&
60                ((Edge)o).a().equals(a()) &&
61                ((Edge)o).b().equals(b()));
62    }
63
64    public String toString() {
65        return a.toString() + " <-> " + b.toString() + ", S: " + this.getPheromone().toString();
66    }
67
68    public Pheromone getPheromone() {
69        return pheromone;
70    }
71
72    public Pheromone getPheromone(Direction direction) {
73        return pheromone.getPheromone(direction);
74    }
75
76    public void update() {
77        this.setLine(a.getCenterX(), a.getCenterY(), b.getCenterX(), b.getCenterY());
```

```
 78
 79    }
 80
 81    public void setLine(double x1, double y1, double x2, double y2) {
 82        setLine((int)x1, (int)y1, (int)x2, (int)y2);
 83    }
 84
 85    public void setLine(int x1, int y1, int x2, int y2) {
 86        setLine(x1, y1, x2, y2, THICKNESS);
 87    }
 88
 89    public void setLine(int x1, int y1, int x2, int y2, int thickness) {
 90        //http://www.rgagnon.com/javadetails/java-0260.html
 91        // The thick line is in fact a filled polygon
 92
 93        int dX = x2 - x1;
 94        int dY = y2 - y1;
 95        // line length
 96        double lineLength = Math.sqrt(dX * dX + dY * dY);
 97
 98        double scale = (double)(thickness) / (2 * lineLength);
 99
100        // The x,y increments from an endpoint needed to create a rectangle...
101        double ddx = -scale * (double)dY;
102        double ddy = scale * (double)dX;
103        ddx += (ddx > 0) ? 0.5 : -0.5;
104        ddy += (ddy > 0) ? 0.5 : -0.5;
105        int dx = (int)ddx;
106        int dy = (int)ddy;
107
108        // Now we can compute the corner points...
109
110        super.reset();
111        super.addPoint(x1 + dx, y1 + dy);
112        super.addPoint(x1 - dx, y1 - dy);
113        super.addPoint(x2 - dx, y2 - dy);
114        super.addPoint(x2 + dx, y2 + dy);
115
116        setWeight(Math.sqrt(Math.pow(x2 - x1, 2) + Math.pow(y2 - y1, 2)));
117        centerX = (x1 + (int)getWeight() / 2);
118        centerY = (y1 + (y2 - y1) / 2);
119    }
120
121    public void draw(Graphics g) {
122        super.draw(g);
123        if( showWeights()) {
124            Graphics2D g2d = (Graphics2D)g;
125            g2d.setRenderingHint
126                    (RenderingHints.KEY_ANTIALIASING,
127                    RenderingHints.VALUE_ANTIALIAS_ON);
128
129            Color old = g2d.getColor();
130
131            g2d.setColor(Color.BLACK);
132
133              g2d.drawString(Integer.toString(getPheromone().getStrength()), (int)getCenterX() -
3, (int)getCenterY() - 10);
134
135            g2d.setColor(old);
136        }
137    }
138
139    public int getCenterX() {
140        return centerX;
141    }
142
143    public int getCenterY() {
144        return centerY;
145    }
146
147    public double getWeight() {
148        return 1.0;//weight;
149    }
150
151    private void setWeight(double weight) {
152        this.weight = Math.round(weight);
153    }
```

```
154
155    public int compareTo(Edge otherEdge) {
156                        double   thisStrength   =   this.getPheromone()   ==   null   ?   -1.00   :
this.getPheromone().getStrength();
157                    double   otherStrength   =   otherEdge.getPheromone()   ==   null   ?   -1.00   :
otherEdge.getPheromone().getStrength();
158        return (int) ( otherStrength - thisStrength);
159    }
160
161    public static Comparator<Edge> TowardNestComparator = new Comparator<Edge>() {
162        public int compare(Edge e2, Edge e1) {
163                                //     int   pheromoneStrength   =   gs1.getPheromone()   !=   null   ?
gs1.getPheromone().getStrength() : 0;
164                                    return   e1.getPheromone().getStrength(Direction.TOWARD_NEST)   -
e2.getPheromone().getStrength(Direction.TOWARD_NEST);
165        }
166    };
167
168    public static Comparator<Edge> AwayFromNestComparator = new Comparator<Edge>() {
169        public int compare(Edge e2, Edge e1) {
170                                //     int   pheromoneStrength   =   gs1.getPheromone()   !=   null   ?
gs1.getPheromone().getStrength() : 0;
171                                 return   e1.getPheromone().getStrength(Direction.AWAY_FROM_NEST)   -
e2.getPheromone().getStrength(Direction.AWAY_FROM_NEST);
172        }
173    };
174
175    public boolean showWeights() {
176        return showWeights;
177    }
178
179    public void showWeights(boolean showWeights) {
180        this.showWeights = showWeights;
181    }
182
183    public void toggleWeights() {
184        this.showWeights = !this.showWeights;
185    }
186}
```

```java
/*
 * Nets1.java
 *
 * Created on March 21, 2007, 10:24 PM
 *
 * To change this tfgemplate, choose Tools | Template Manager
 * and open the template in the editor.
 */

package antgraph;

import antgraph.gui.GraphJFrame;

/**
 *
 * @author james
 */
public class Nest extends Node {

    /** Creates a new instance of Nets1 */
    public Nest(String name, int x, int y) {
        super(name, x, y);
    }

    public Nest(String name) {
        super(name);
    }

    public void addAnt() {
        Ant ant = new Ant(this);
        super.addAnt(ant);
        GraphJFrame.getGraph().addAnt(ant);
    }

    public void addAnts(int n) {
        for(int i = 0; i < n; i++)
            addAnt();
    }

    public void addAnt(Ant ant) {
        System.out.println("Ant returned to nest");
        addFood(ant.getFood());
        ant.setFood(null);
        super.addAnt(ant);
    }
}

```

```
 1/*
 2 * Pheromone.java
 3 *
 4 * Created on 04 March 2007, 02:37
 5 *
 6 * To change this template, choose Tools | Template Manager
 7 * and open the template in the editor.
 8 */
 9
10package antgraph;
11
12import antgraph.gui.GraphJFrame;
13import java.util.List;
14import java.util.Stack;
15import java.util.Vector;
16
17/**
18 *
19 * @author James
20 */
21public class Pheromone implements Runnable {
22
23    private List<PheromoneParticle> particles = new Vector<PheromoneParticle>();
24
25    private int iTowardsNest = 0;
26    private int iAwayFromNest = 0;
27
28    private Thread move;
29    private int rate = 10;
30
31    public static enum Direction { TOWARD_NEST, AWAY_FROM_NEST, NONE; }
32
33    /** Creates a new instance of Pheromone */
34    public Pheromone() {
35        this(true);
36    }
37
38    public Pheromone(boolean start) {
39        if(start) {
40            move = new Thread(this);
41            move.setDaemon(true);
42
43        }
44    }
45
46    public void run() {
47        while(move != null) {
48
49            try {
50
51                if(GraphJFrame.getGraph().getNest() != null &&
52                                GraphJFrame.getGraph().getNest().countFood() >= GraphJFrame.FOOD)
stop();
53                move.sleep(getRate());
54                decrease();
55
56            }catch (InterruptedException e) {
57                System.out.println(e);
58            }
59
60        }
61    }
62
63    public Direction getDirection(Ant ant) {
64        for(int i = particles.size() - 1; i >= 0; i++) {
65            if(particles.get(i).getAnt().equals(ant))
66                return particles.get(i).getDirection();
67        }
68        return Direction.NONE;
69    }
70
71    public void decrease() {
72
73       if(particles.size() > 0) {
74           if(particles.get(0).getDirection() == Direction.TOWARD_NEST) {
75                iTowardsNest--;
76            }else if(particles.get(0).getDirection() == Direction.AWAY_FROM_NEST) {
```

```
77                  iAwayFromNest--;
78              }
79
80              particles.remove(0);
81
82          }
83
84      }
85
86      public void increaseStrength(Ant ant, Direction direction) {
87          particles.add(new PheromoneParticle(ant, direction));
88
89          if(direction == Direction.TOWARD_NEST) {
90              iTowardsNest++;
91          }else if(direction == Direction.AWAY_FROM_NEST) {
92              iAwayFromNest++;
93          }
94      }
95
96      public synchronized void increaseStrength(PheromoneParticle p) {
97          this.increaseStrength(p.getAnt(), p.getDirection());
98      }
99
100     public Pheromone getPheromone(Direction direction) {
101         Pheromone p = new Pheromone(false);
102         for(PheromoneParticle pp : particles) {
103             if(pp.getDirection() == direction) p.increaseStrength(pp);
104         }
105
106         return p;
107     }
108
109     public int getStrength() {
110         return getStrength(Direction.TOWARD_NEST) + getStrength(Direction.AWAY_FROM_NEST);
111     }
112
113
114     public int getStrength(Direction direction) {
115         if(direction == direction.TOWARD_NEST) {
116             return iTowardsNest;
117         }else if(direction == Direction.AWAY_FROM_NEST) {
118             return iAwayFromNest;
119         }
120         return 0;
121
122     }
123
124     public void start() {
125         if(move == null) {
126             move = new Thread(this);
127             move.setDaemon(true);
128             move.start();
129         }
130
131     }
132
133     public void stop() {
134         move = null;
135     }
136
137
138     public Direction getDirection() {
139         if(particles.size() == 0) return Direction.NONE;
140
141         int toward_nest = getStrength(Direction.TOWARD_NEST);
142         int away_from_nest = getStrength(Direction.AWAY_FROM_NEST);
143
144         return toward_nest > away_from_nest ? Direction.TOWARD_NEST : Direction.AWAY_FROM_NEST;
145
146     }
147
148     public boolean laidBy(Ant ant) {
149         for(PheromoneParticle p : particles) {
150             if(p.getAnt().equals(ant)) return true;
151         }
152
153         return false;
```

```
154    }
155
156    public String toString() {
157        StringBuffer sb = new StringBuffer();
158
159        sb.append("T: " + getStrength(Direction.TOWARD_NEST));
160        sb.append(", ");
161        sb.append("A: " + getStrength(Direction.AWAY_FROM_NEST));
162        sb.append(", ");
163        sb.append("A+T: " + getStrength());
164        sb.append(", D: "  + getDirection());
165        return sb.toString();
166    }
167
168    public int getRate() {
169        return rate;
170    }
171
172    public void setRate(int rate) {
173        this.rate = rate;
174    }
175}
176
```

```
 1/*
 2 * PheromoneParticle.java
 3 *
 4 * Created on 16 February 2007, 02:51
 5 *
 6 * To change this template, choose Tools | Template Manager
 7 * and open the template in the editor.
 8 */
 9
10package antgraph;
11
12import antgraph.Pheromone.Direction;
13
14/**
15 *
16 * @author James Hamilton
17 */
18public class PheromoneParticle {
19
20    private Ant ant = null;
21    private Direction direction = Direction.AWAY_FROM_NEST;
22
23    /** Creates a new instance of PheromoneParticle */
24    public PheromoneParticle(Ant ant, Direction direction) {
25        this.setAnt(ant);
26        this.setDirection(direction);
27    }
28
29    public Ant getAnt() {
30        return ant;
31    }
32
33    public void setAnt(Ant ant) {
34        this.ant = ant;
35    }
36
37    public Direction getDirection() {
38        return direction;
39    }
40
41    public void setDirection(Direction direction) {
42        this.direction = direction;
43    }
44
45    public String toString() {
46        return getDirection() == Direction.AWAY_FROM_NEST ? "A" : "T";
47    }
48}
49
```

```
 1/*
 2 * Food.java
 3 *
 4 * Created on 04 March 2007, 01:52
 5 *
 6 * To change this template, choose Tools | Template Manager
 7 * and open the template in the editor.
 8 */
 9
10package antgraph;
11
12/**
13 *
14 * @author James
15 */
16public class Food {
17
18    /** Creates a new instance of Food */
19    public Food() {
20    }
21
22}
```

```
 1/*
 2 * Node.java
 3 *
 4 * Created on 03 March 2007, 19:33
 5 *
 6 * To change this template, choose Tools | Template Manager
 7 * and open the template in the editor.
 8 */
 9
10package antgraph;
11
12import antgraph.gui.GraphJFrame;
13import java.awt.Color;
14import java.awt.Graphics;
15import java.awt.Graphics2D;
16import java.awt.RenderingHints;
17import java.util.ArrayList;
18import java.util.List;
19
20/**
21 *
22 * @author James
23 */
24public class Node extends GraphComponent {
25
26    private String name;
27    private List<Ant> ants = new ArrayList<Ant>();
28    private List<Food> food = new ArrayList<Food>();
29
30
31    private int x, y, radius;
32
33    public Node(String name) {
34        radius = 20;
35
36        this.setName(name);
37    }
38
39    public Node(String name, int x, int y) {
40        this(name);
41        this.x = x;
42        this.y = y;
43        update();
44    }
45
46    public Node(Object o, int x, int y) {
47        this(o.toString(), x, y);
48    }
49
50    public Node(Object o) {
51        this(o.toString());
52    }
53
54    public void addAnt(Ant ant) {
55        getAnts().add(ant);
56    }
57
58    public int countAnts() {
59        return getAnts().size();
60    }
61
62    public void removeAnt(Ant ant) {
63        if(getAnts().size() > 0) getAnts().remove(ant);
64    }
65
66    public void addFood() {
67        food.add(new Food());
68    }
69
70    public void addFood(int n) {
71        for(int i = 0; i < n; i++)
72            addFood();
73    }
74
75    public void addFood(Food f) {
76        if(f != null)
77            food.add(f);
```

```
 78    }
 79
 80    public Food removeFood() {
 81        Food f = food.get(0);
 82        food.remove(0);
 83        return f;
 84    }
 85
 86    public int countFood() {
 87        return food.size();
 88    }
 89
 90    public Nest getNest() {
 91        if(isNest()) return (Nest)this;
 92        else return null;
 93    }
 94
 95    public boolean isNest() {
 96        return this instanceof Nest;
 97    }
 98
 99    public boolean containsFood() {
100        return countFood() != 0;
101    }
102
103    public boolean containsAnts() {
104        return countAnts() != 0;
105    }
106
107    public boolean empty() {
108        return !(containsFood() && containsAnts());
109    }
110
111    public String toString() {
112        return getName();
113    }
114
115    public int hashCode() {
116        return toString().hashCode();
117    }
118
119    public boolean equals(Object o) {
120        return o instanceof Node &&
121                ((Node)o).getName().equals(getName());
122    }
123
124    public String getName() {
125        return name;
126    }
127
128    private void setName(String name) {
129        this.name = name;
130    }
131
132    public List<Node> getAdjacencyList() {
133        return GraphJFrame.getGraph().getAdjacencyList(this);
134    }
135
136
137    public void draw(Graphics g) {
138        super.draw(g);
139        Graphics2D g2d = (Graphics2D)g;
140        g2d.setRenderingHint
141          (RenderingHints.KEY_ANTIALIASING,
142            RenderingHints.VALUE_ANTIALIAS_ON);
143
144        Color old = g2d.getColor();
145
146        g2d.setColor(Color.WHITE);
147
148        g2d.drawString(toString(), (int)getCenterX() - 3, (int)getCenterY() + 3);
149
150        g2d.setColor(Color.RED);
151
152              g2d.drawString(getAnts().toString(),  (int)getCenterX()  +  getRadius()  /  2  +  5,
(int)getCenterY() + getRadius() / 2 + 5);
153
```

```
154         g2d.setColor(old);
155     }
156 
157 
158     public void update() {
159         this.reset();
160         for(double t = 0; t <= (2 * Math.PI); t += 2 * Math.PI / 50)
161                addPoint((int)  (getX()+getRadius()/2 * Math.cos(t)),(int)(getY()+getRadius()/2 *
Math.sin(t)));
162     }
163 
164 
165     public void setLocation(int x, int y) {
166         this.x = x;
167         this.y = y;
168     }
169 
170     public int getRadius() { return radius; }
171     public int getX() { return x; }
172     public int getY() { return y; }
173 
174     public int getCenterX() {
175         return getX();
176     }
177 
178     public int getCenterY() {
179         return getY();
180     }
181 
182     public List<Ant> getAnts() {
183         return ants;
184     }
185 
186     public Color getColor() {
187         Color retValue;
188 
189         if(isNest()) {
190            retValue = Color.GRAY;
191         }else if(containsFood()) {
192             retValue = Color.ORANGE;
193         }else{
194              retValue = super.getColor();
195         }
196         return retValue;
197     }
198}
```

```
 1/*
 2 * Graph.java
 3 *
 4 * Created on 02 March 2007, 22:52
 5 *dfdf
 6 * To change this template, choose Tools | Template Manager
 7 * and open the template in the editor.
 8 */
 9
10package antgraph;
11
12
13import antgraph.Pheromone.Direction;
14import antgraph.gui.GraphJFrame;
15import java.awt.Image;
16import java.awt.image.BufferedImage;
17import java.util.ArrayList;
18import java.util.Collections;
19import java.util.HashMap;
20import java.util.List;
21
22/**
23 *
24 * @author James
25 */
26public class Graph {
27
28
29    private List<Node> nodes = new ArrayList<Node>();
30    private HashMap<Node, List<Node>> adjacencyList = new HashMap<Node, List<Node>>();
31    private HashMap<String, Edge> edges = new HashMap<String, Edge>();
32
33    private boolean isDirty = false;
34
35    private List<Ant> ants = new ArrayList<Ant>();
36
37    private int c =1;
38    private char currentChar = 'a' - 1;
39
40    public Graph() {
41        System.out.println(this);
42    }
43    public void addAnt(Ant listener) {
44
45        ants.add(listener);
46        System.out.println(ants);
47    }
48
49
50    public void addNode(Node n) {
51        nodes.add(n);
52    }
53
54    public void addNode(String o) {
55        this.addNode(new Node(o));
56    }
57
58    public void addNode() {
59        this.addNode(new Node(this.getNextChar()));
60    }
61
62    public void addNode(int x, int y) {
63        Node n = new Node(this.getNextChar(), x, y);
64       // n.setLocation(x, y);
65        this.addNode(n);
66    }
67
68     public void addNode(String s, int x, int y) {
69        Node n = new Node(s, x, y);
70        //n.setLocation(x, y);
71        this.addNode(n);
72    }
73
74    public void removeNode(Node n) {
75        System.out.println("Removing " + n + "...");
76
77
```

```
78          for(Node node : getNodes()) {
79              removeEdge(node, n);
80          }
81
82          nodes.remove(n);
83
84      }
85
86      public int getNumberOfVertices() {
87          return nodes.size();
88      }
89
90      public void addEdge(Node a, Node b) throws NoSuchNodeException {
91
92
93          if(hasEdge(a, b) || a.equals(b)) return;
94             System.out.println("Creating an edge between " + a + " and " + b);
95          Edge e = new Edge(a, b);
96
97          edges.put(a.toString() + b.toString(), e);
98
99          if(!nodes.contains(a))
100             throw new NoSuchNodeException("No such node: " + a);
101         if(!nodes.contains(b))
102             throw new NoSuchNodeException("No such node: " + b);
103
104         if(!adjacencyList.containsKey(a)) {
105             adjacencyList.put(a, new ArrayList());
106         }
107          if(!adjacencyList.containsKey(b)) {
108             adjacencyList.put(b, new ArrayList());
109         }
110
111         adjacencyList.get(a).add(b);
112         adjacencyList.get(b).add(a);
113
114     }
115
116     public Edge getEdge(Node a, Node b) {
117         if(hasEdge(a, b)) {
118             if(edges.containsKey(a.toString() + b.toString()))
119                 return edges.get(a.toString() + b.toString());
120             else
121                 return edges.get(b.toString() + a.toString());
122         }
123         return null;
124     }
125
126     public List<Edge> getEdges(Node node) {
127         List<Edge> edges = new ArrayList();
128
129         for(Node n : getAdjacencyList(node)) {
130             edges.add(getEdge(node, n));
131         }
132         return edges;
133     }
134
135     public List<Node> getNodes(Direction direction) {
136         List<Node> edges = new ArrayList();
137         for(Edge e : getEdges()) {
138             if(e.getPheromone().getDirection() == direction) {
139                 if(!edges.contains(e.a()))
140                     edges.add(e.a());
141                 if(!edges.contains(e.b()))
142                     edges.add(e.b());
143             }
144         }
145
146         return edges;
147     }
148
149     public List<Edge> getEdges(Direction direction) {
150         List<Edge> edges = new ArrayList();
151         for(Edge e : getEdges()) {
152             if(e.getPheromone().getDirection() == direction) edges.add(e);
153         }
154
```

```
155         return edges;
156     }
157 
158     public List<Edge> getEdges() {
159         return new ArrayList(edges.values());
160     }
161 
162     public HashMap<String, Edge> getEdgesMap() {
163         return edges;
164     }
165 
166     public boolean hasEdge(Node a, Node b) {
167         try {
168             return adjacencyList.get(a).contains(b) || adjacencyList.get(b).contains(a);
169         }catch (NullPointerException e) {
170             return false;
171         }
172     }
173 
174     public void removeEdge(Node a, Node b) {
175         edges.remove(a.toString() + b.toString());
176         edges.remove(b.toString() + a.toString());
177 
178         getAdjacencyList(a).remove(b);
179         getAdjacencyList(b).remove(a);
180     }
181 
182     public void removeEdge(Edge e) {
183         removeEdge(e.a(), e.b());
184     }
185 
186     public List<Node> getAdjacencyList(Node n) {
187         return adjacencyList.containsKey(n) ? adjacencyList.get(n) : new ArrayList();
188     }
189 
190     public HashMap<Node, List<Node>> getAdjacencyList() {
191         return adjacencyList;
192     }
193 
194     public List<Node> getNodes() {
195         return nodes;
196     }
197 
198     public List<GraphComponent> getComponents() {
199         List l = new ArrayList(nodes);
200         l.addAll(edges.values());
201         return l;
202     }
203 
204     public void highlight(GraphComponent gc) {
205         if(gc != null) {
206             if(gc instanceof Edge)
207                 for(Edge e : getEdges())
208                     e.highlight(false);
209             else
210                 for(Node e : getNodes())
211                     e.highlight(false);
212             gc.highlight(true);
213         }
214     }
215 
216     public void highlight(List<Node> nodes) {
217         try {
218             nodes.add(nodes.get(0));
219             for(int i = 0; i < nodes.size() - 1; i++) {
220                 getEdge(nodes.get(i), nodes.get(i + 1)).highlight(true);
221                 nodes.get(i).highlight(true);
222             }
223             nodes.get(nodes.size()-1).highlight(true);
224 
225         }catch (NullPointerException e) {}
226     }
227 
228     public void highlightShortest(Node start, Node end) {
229         Node pNode = null;
230         Node node = start;
231         List<Edge> edges;
```

```
232
233         Edge edge = null;
234
235
236         for(Edge edge1 : getEdges())
237             edge1.setSelected(false);
238
239         while(!node.equals(end)) {
240             edges = getEdges(node);
241             if(pNode != null) edges.remove(getEdge(pNode, node));
242
243             Collections.sort(edges);
244
245
246             edge = edges.get(0);
247
248
249             edge.setSelected(true);
250             node.setSelected(true);
251
252             pNode = node;
253             node = edge.other(node);
254
255
256         }
257     }
258
259     public void highlightShortest(String a, String b) throws NoSuchNodeException {
260         highlightShortest(getNode(a), getNode(b));
261     }
262
263     public void highlightShortest(Node a, String b) throws NoSuchNodeException {
264         highlightShortest(a, getNode(b));
265     }
266
267     public void highlightShortest(String a, Node b) throws NoSuchNodeException {
268         highlightShortest(b, getNode(a));
269     }
270
271     public void print() {
272         print(this);
273     }
274
275     public static void print(Graph g) {
276
277         System.out.println(" --- Graph --- ");
278
279         for(Node n : g.getNodes()) {
280
281             System.out.println(n + ": " + g.getAdjacencyList(n));
282
283         }
284         System.out.println(" ------------ ");
285     }
286
287     public void step() {
288
289
290         for(Ant ant : ants) {
291             ant.move();
292         }
293
294         if(getC() % 3 == 0) {
295             for(Edge edge : getEdges()) {
296                 edge.getPheromone().decrease();
297             }
298         }
299         c++;
300
301         if(getNest().countFood() == GraphJFrame.FOOD) {
302             //System.out.println(GraphJFrame.getGraph().getNest().countFood());
303             try {
304                 highlightShortest(getNest(), "1");
305             }catch (NoSuchNodeException ex) {
306                 System.out.println(ex);
307             }
308
```

```
309
310         }
311     }
312
313     public boolean isDirty() {
314         return isDirty;
315     }
316
317     public void setDirty(boolean value) {
318         isDirty = value;
319     }
320
321     public char getCurrentChar() {
322         return currentChar;
323     }
324
325     public void setCurrentChar(char c) {
326         currentChar = c;
327     }
328
329     public char getNextChar() {
330         currentChar++;
331         return getCurrentChar();
332     }
333
334     public Node getNestNode() {
335         for(Node n : getNodes()) {
336             if(n.isNest()) return n;
337         }
338         return null;
339     }
340
341     public Nest getNest() {
342         return getNestNode() != null ? getNestNode().getNest() : null;
343     }
344
345     public Node getNode(String s) throws NoSuchNodeException {
346         for(Node n : getNodes()) {
347             if(n.toString().equals(s)) return n;
348         }
349         throw new NoSuchNodeException("No Such Node: " + s);
350     }
351
352     public int getC() {
353         return c;
354     }
355}
```

```
 1/*
 2 * NoSuchNodeException.java
 3 *
 4 * Created on 03 March 2007, 21:50
 5 *
 6 * To change this template, choose Tools | Template Manager
 7 * and open the template in the editor.
 8 */
 9
10package antgraph;
11
12/**
13 *
14 * @author James
15 */
16public class NoSuchNodeException extends Exception {
17
18    /** Creates a new instance of NoSuchNodeException */
19    public NoSuchNodeException(String s) {
20        super(s);
21    }
22
23}
```

Appendix. C Weekly Diary Reports

Weekly diary reports in chronological order.

First meeting with Sebastian - Friday 13/10/2006, 02:30 PM

Discussed how to start project. We decided I should concentrate on the simulation of ants and decide whether to continue the project on as a way of solving graph path problems later.

Showed Sebastian my first version, and he suggested the next version ants should come out from a 'nest' and pick up food, then continue walking around randomly until the find the nest where they will drop off the food.

Sebastian advised to create a skeleton project report in LaTeX to fill in as I progress with the project.
See simulation(s): Version 1.0

Progress - Friday 20/10/2006, 02:00 PM

Showed Sebastian some new versions of the simulation.

Sebastian suggested that the ants are not moving realistically yet. He suggested ants could 'smell' for a certain radius and food produces a smell for a certain radius around it. This way ants will tend to move towards food.

The food smell could possible use an inverse square law to determine density as it gets further from the food.
See simulation(s): Version 2.0 Version 3.0

Meeting with Sebastian - Friday 27/10/2006, 02:00 PM

We discussed further how to make the ants seem more real.

I showed Sebastian two books that I have been using for research and a research paper about the use of ant systems for web searching: neither of us understood this - we will try to for next week!
See simulation(s): Version 4.0

Progress - Friday 3/11/2006, 02:00 PM

Showed Sebastian latest Ant Simulator: Ants tend to move towards the food due to a food strength value radiating outward from the food source.

This uses an inverse square law so ants tend to move towards the center.

I've implemented this by having the ants ask the GridSquare for all the surrounding GridSquare's with a radius of 1 GridSquare. The Ant then sorts these based on Food Strength, and moves the the GridSquare with the highest Food Strength.

Some problems include ants getting trapped between food sources, and clusters maybe too strong.

I need to implement controls to change values such as the radius of the food strength as the program is running.

We also discussed the direction the project is going to take after this section is complete. My idea is the use the Ants to solve shortest path problems. And to do this I will add Obstacles to the Grid, such that the graph is where the obstacles aren't. It should be possible to draw graphs onto the grid and

have to see the ants move around the grid and hopefully find the shortest path from their nest to the food source.
See simulation(s): Version 5.0

A Little Progress - Friday 17/11/2006, 02:00 PM

Had been distracted by a Prolog assignment, so hadn't much to show Sebastian!

The FoodClusters was not well recieved by Sebastian. I agree after he told me why.

I will change the Food Strength value to be calculated by each GridSquare individually instead of using clusters.
See simulation(s): Version 6.0 Version 7.0

Ants Get Stuck! - Friday 24/11/2006, 02:00 PM

Showed Sebastian my latest version with the new FoodStrength calculations.

Sebastian and I agreed this works much better.

I explained the problem of ants getting stuck against obstacles. Sebastian suggested introducing randomness, so that if an ant encounters and obstacle they randomly choose another direction instead of just turning around - with this version they are turning around and then going back the same way because it is the 'best' square for them to choose.

Tasks for next week

- Fix the dancing ant problem
- Get ants to return food to collect food and return it to the nest - Sebastian suggested ants know where the nest is
- Implement Pheromone trails (this was partially working before and need to get it working again

See simulation(s): Version 8.0

Meeting - Friday 1/12/2006, 02:00 PM

We discussed problems encountered with my first implementation of the pheromone trail following. See Split Trail Problem and Bouncing Ant Problem

We discussed solutions and came up with the idea that ants only see ahead of them not in all directions. See changing ant movement for more details.

Some problems solved - Thursday 7/12/2006, 02:00 PM

I showed Sebastian the implementation of new movement algorithm - he especially liked how the ants walked around each other.

There are still some problems with the algorithm and the move() method in the Ant class needs rewriting.

I showed Sebastian another ant simulation and he liked how the ants on the other simulation move a lot smoother than mine which makes them look more like ants. I will try to make the movement of my ants smoother.

We also took note that there is no use of Food Strengths in the other simulation and pheromones spread further and dissipate quicker than in mine. There version works very well.

Last Meeting of Term 1 - Friday 15/12/2006, 02:00 PM

Prolog took most of my time this week again!

We discussed what should be done over the Christmas break. The aim is to finish the main programming over this time, in order to have time to experiment and write up over the next term.

One of the main tasks is to re-write the move() method, in the Ant class. It needs some thought before actually being implemented to get it right.

Another, less important, task is to create a smoother moving animation. I believe the main problem causing this is the amount of 'work' each ant has to do every time it moves e.g. it looks at the surrounding squares, sorts them etc. The use of Vector's for this probably slows it down, and it may be needed to use arrays instead. This is not as important though as it does not effect the algorithms, it would just make the simulation look more like real ants (Am I trying to create realistic ants, or model ant behaviour? I'd say the latter - the model does not have to be super-realistic, just borrow ideas from nature).

Next will be to implement some kind of graph system; a maze like system - where the graph will be 'where the obstacles are not'. This cannot really work until the ants are actually behaving some-what like real ants.

Not much progress - Thursday 8/02/2007, 03:00 PM

Many movement algorithm rewrites, but not much progress. Discussed possible improvements to the algorithm.

Project Direction - Thursday 1/03/2007, 03:00 PM

Showed Sebastian the latest version of the ant simulator. There are many problems with it!
Sebastian was worried about the direction of the project and suggested that we move on to graph algorithms, as most of what is happening now is tweaking the existing movement algorithm to stop it getting stuck at obstacles!
We realised that implementing a graph structure and developing an ant algorithm using this should be simpler, as there are no obstacles involved.
We discussed the MUTE P2P routing algorithm description and decided to based the new ant algorithm on this idea.
Tasks for next week: implement a graph data structure, explain the new ant algorithm.

Ants 2.0 - Thursday 8/03/2007, 03:10 PM

I spent the last week working on part 2.0 of my project - implementing a graph and getting ants to find the shortest path between two points.
I've implemented a gui that allows a user to create a graph on screen by placing nodes and creating edges between those nodes. Nodes and edges can be added and removed, and nodes can be dragged around. The length of the edges are calculated and shown. Printing is also available by pressing 'p'.
Sebastian was pleased with the graph application and we discussed what to do next, in terms of getting ants to move around the graph and find the shortest path between two points.
Wed decided to base the idea on the one from the MUTE application previously described. The algorithm involves two kinds of trails - those going from nest to food, and those going from food to nest. Ants looking for food will like to follow nest to food trails, while those carrying food will like to follow food to nest trails. Trails evaporate over time and the strength is determined by how many ants have laid some trail.
We realised there could be difficulty implementing this for a general weighted graph, and decided to first implement where all weights are 1.

The difficulty is caused because ants should take longer to reach nodes that are further away, but this means that after each iteration an ant cannot instantaneously move from its current node to the next node as this would ignore the distances. An idea to implement this would be to have ants actually move along edges at a constant speed. Ants travelling along longer edges will take longer to reach their destination nodes, whereas ants travelling along shorter nodes will reach their destination quicker. To simplify this problem for the first version, we decided to have all weights equal to one so that each ant can travel the same distance in the same time.

Problems with Ants 2.0 - Friday 16/03/2007, 03:00 PM

After lengthy discussion during this meeting we came across a problem with the algorithm.
It is entirely possible that ants will go the long way round initially, and therefore lay a trail the long way to the food. The shortest path will not be found. After some different variations we came up with breadth-first search as the solution! Something wasn't quite right!
It was decided to do some more research into the MUTE application, to learn more about their implementation
Addition: I realised that MUTE uses a discovery algorithm, akin to breadth-first search to establish a route before using ant based algorithms for routing.

Last Meeting - Thursday 26/04/2007, 02:00 PM

Project has gone very well. Sebastian is pleased with the report progress so far. Almost complete. Software not completely functional yet.

Appendix. D Project Description Report

The project proposal.

Artificial Ants: Using Collective Intelligence
To Create Efficient Networks
A Final Year Project Proposal

James Hamilton
ma301jh@gold.ac.uk

March 29, 2006

Contents

1

Abstract

Computer Scientists are always looking for more efficient algorithms to solve problems in the field. To aid in the creation of new, more efficient algorithms we can take a leaf out of nature's book, where there are already many examples of efficient collective behavior. Taking ants as an example: How do ants find their food? How do they walk in a line, often the shortest distance, towards their food? Can the behavior of ants be used to find the shortest distance around a computer network efficiently? How can myrmecology1 and computer science be combined?

1.1 Introduction

Researchers have been looking into the use of Artificial Ants to improve network communications for sometime, and the results have shown that ant foraging behavior used as a basis for modeling network traversal algorithms are very efficient[2]. Nature has already solved many problems that these researchers are trying to solve and/or improve. Ants are very good at finding

the shortest distance between their nest and a food source, and are able to work together efficiently.

1.1.1 Using Ant Behavior

So ants can find the shortest path between their nest and food - how does this help computer scientists? As stated before a model based on ant foraging behaviour can be used to solve the traveling salesman problem for certain graphs. If a computer network is modeled using a graph with a computer at each node, then the ants can find their way around the computer network more efficiently than other algorithms (see [2] for actual results of other researcher's experiments).

1.2 Hypothesis

The main hypothesis to be proven is that Ant Colony Optimization algorithms will prove more efficient than other algorithms for solving the traveling salesman problem on a set of test graphs.

1.3 Objectives

Software – A simulation of ant behaviour, by using biological studies as a basis for creating simple rules that govern the behaviour of the computerised ants.
– A graph package in Java, to later be used for testing the TSP algorithms.
– A graphical interface to the graph package, which allows the user to create custom graphs and to see graphically the shortest path.

Algorithms – Heuristic algorithms such as Nearest Neighbour and Insertion algorithms.
– Ant-based algorithms.

Tests – Implement test graphs using the graph package.
– Run tests and analyse results to show difference between different algorithms.

Applications – Research applications of ant algorithms in the field of computing.
– Implement an application of the ant algorithms (e.g. network software)
– Research the use of ant algorithms in other fields.

1.4 Background

1.4.1 The Traveling Salesman Problem (TSP)

The TSP involves finding the cheapest route for a salesman traveling to

many cities given the known cost for traveling between each city. It is one of the most intensely studied problems in mathematical computing, due to its usefulness and relevance to computer networks. Even so, an effective solution still evades us[1]. Using a graph as a model for a computer network, where each node is a device on the network, this problem can be applied to find the shortest distance around the network. If an efficient algorithm can be found to traverse the entire graph, then this solution could also be applied to computer networks.

1.4.2 Ant Behaviour

Ants are stupid. But collectively ants behave intelligently.

Ants are studied due to their collective intelligence, as simple things working together to create an intelligent whole. Most ant species forage for food collectively.

How ants behave[4]:

- They randomly search the area for food.
- If they find food, they will lay a pheromone trail back to the nest.
- If an ant finds another trail it will follow it.
- Ants are attracted to the strongest trails.
- Trails evaporate at a constant rate - so the shorter the trail the stronger it will be.
- Ants will strengthen trails as they follow them - so the more ants the stronger the trail, the more ants following the trail, then it will become even stronger. This results in a positive feedback loop.
- Ants only lay a pheromone trial while they are carrying food - so as soon as the food is depleted ants will stop laying the trail and it will evaporate. Then ants will cease to follow it and search for other food sources.

Although I intend to base the ant routing algorithms on nature, the behaviour will need to be slightly modified so that ants do not get trapped in loops. This may happen when pheromone trails become strong in a loop pattern and due to positive feedback any ant finding the loop will get trapped [3].

1.5 Method

I will start my project by first conducting research into my myrmecology, followed by the implementation of this behaviour using Java.

Following this I will research graph theory the expand my knowledge in

the area, as my project is based on this. I will then implement a graph package in Java, which will allow the creation of connected graphs that will represent a computer network, for example, in later stages of the project. After implementing the graph package it will be necessary to implement several test graphs to ensure that everything is working correctly.

The implementation of a graphical user interface next will allow easier creation of graphs, and also allow the shortest path around the graph to be shown graphically. This will be more user friendly. After implementation, the previous tests can be ported to the graphical program to check that the output works. New graphs should also be created using the user interface, to check that the input works.

After the software needed to create and manipulate graphs is complete, it will be time to start researching the traveling problem and algorithms used to solve it. Then next step is to then implement these algorithms and test them on various test graphs. After the usual algorithms such as Nearest Neighbour, the time will come to research and implement ant based algorithms and compare the different algorithms.

Once the main ant algorithms have been implemented I will be able to research a use of these algorithms, for example computer networking, and implement a solution using the ant-based algorithms.

Finally I will evaluate my project and its results, and draw conclusions from mine and others works based on what I have learned during the course of the project.

1.5.1 Gantt Chart

Figure 1.1: Gantt Chart showing a summary of the project plan.

1.6 Conclusion

Artificial Ants will provide an interesting and challenging project, which will help me further my knowledge of computing and mathematics.

This will make a challenging project because it will involve a lot of background research into algorithms, graph theory, the traveling salesman problem, mymecology, computer networks and related fields that I may need to investigate.

The implementation of algorithms to solve the traveling salesman problem for the set of test graphs should provide interesting results for the different algorithms and allow me to evaluate the efficiency of those algorithms.

Furthermore the undertaking of this project will provide an interesting insight into the field of myrmecology, and the application of nature in computing.

Due to the many places that shortest paths need to be calculated (for example, calculating the shortest route for a delivery company) the project will allow for further research into these other fields and is not just restricted to routing in computer networks.

Bibliography

[1] William Cook. Travelling salesman problem. http://www.tsp.gatech.edu/.

[2] Dorigo and Gambardella. Ant colonies for the travling salesman problem. 1996.

[3] Dorigo and Stutzle. Ant Colony Optimization. The MIT Press, Massachusetts Institute of Technology (MIT), Cambridge, Massachusetts, 2004.

[4] Mitchel Resnick. Turtles, Termites, and Traffic Jams. Explorations In Massively Parallel Microworlds. The MIT Press, Massachusetts Institute of Technology (MIT), Cambridge, Massachusetts, 1994.