**CSC 385**
**Semester Project**
**Ant Colony Simulation**

## 100 points

### Introduction

For the semester project you will be designing a simulation of an ant colony. Ant colonies are interesting for a variety of reasons, but one of the most interesting features of an ant colony is that it demonstrates a property known as *emergence*. Ant colonies are composed of thousands of individuals that carry out their assigned tasks. The combined individual behaviors of all the ants bring about the emergence of a self-organized system: the colony itself. Each ant only lives for about one year, except for the queen, which can live for 20 years or more. The colony can survive as long as the queen still lives, meaning the colony can also survive for 20 years or more. Since the individual ants only live for about a year, the colony isn't able to benefit from the wisdom of old "wise" ants. The colony lives simply because each individual ant does what it has been programmed to do. A well-ordered ant society emerges from the synergistic actions of the individuals.

The colony you create will consist of a queen and her brood, which will have workers to gather food and scout the terrain surrounding the colony, and soldiers to protect the colony from enemies. The colony will start off with only the queen and a few workers and soldiers. Over time the colony will expand to function like a real ant colony, to a limited extent. Since you will be building a simulation, how your colony behaves will depend on how certain parameters have been set. The purpose of a simulation is to mimic reality as closely as possible, but in many cases there may not be enough facts on hand to build an accurate simulation. In such cases, it is beneficial to build the simulation in a way that allows the various parameters to be changed easily. I will be providing you with values for as many parameters as I can, but many aspects of the simulation involve the use of random numbers. Therefore, the precise end result is indeterminate. When I grade your project I won't be looking for a single, correct end result; instead I will be looking to see that the various components (i.e., the various ants) do their jobs correctly. If you are able to program your ants to do what they are supposed to do as individuals, your colony should emerge from their collective efforts.

### A. The Environment

1. The ants' environment should be represented using a 27 x 27 square grid.
2. Each square in the grid represents a discrete location in the environment.
3. Eight directions of movement are possible (see diagram below).
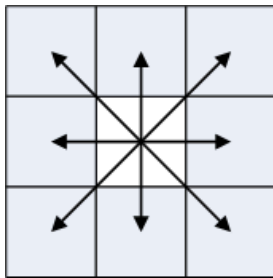


4. The entrance to the colony is represented by a single square located in the center of the grid. The queen is located in this square.
5. The remaining squares will initially be unexplored terrain.
6. Certain ants (scout ants) will be capable of revealing the areas that have not been explored.
7. All other types of ants will only be allowed to move into squares that have been revealed by scout ants.
8. Each square can contain one or more of the following, in any combination:

   a. Zero or more enemy ants
   b. Zero or more friendly ants
   c. Zero or more units of food
   d. Zero or more units of pheromone

### B. Ant Types

There are five types of ants in the simulation:

1. Queen
2. Forager
3. Scout
4. Soldier
5. Bala

Each ant type is described in more detail below.

### C. Characteristics Common to All Ant Types

1. Each ant should be identified by a unique integer ID. The queen ant should have an ID value of 0. Other ants should be numbered in ascending order as they are hatched.
2. All ant types (except for the queen) have a maximum life span of 1 year.
3. Dead ants should be removed from the simulation.
4. All ants are limited to one action per turn, *with some exceptions that will be discussed later*.
5. All ants except Bala ants may only move in squares that have been revealed by scout ants; Bala ants may also move into squares that have not been revealed by scout ants.
6. When moving, all ant types should move no more than 1 square per turn.

## D. The Queen Ant

The queen ant is responsible for hatching new ants. The specific requirements for the queen ant are:

1. The queen never moves from her square (i.e., she remains in the same square for the entire simulation).
2. The queen's maximum lifespan is 20 years.
3. The queen hatches new ants at a constant rate of 1 ant/day (i.e., 1 ant every 10 turns).
4. New ants should always be hatched on the first turn of each day.
5. The type of ant that is hatched should be determined randomly according to the initial frequencies listed below. You may change these frequencies as you see fit — these are simply suggestions for a starting point.

    a. Forager - 50%
    b. Scout - 25%
    c. Soldier - 25%
6. The queen should consume 1 unit of the food in her chamber on each turn, including the turn in which she hatches a new ant.
7. If the food level in the queen's square is zero when the queen tries to eat, the queen dies of starvation.
8. If the queen dies, either by starvation or by a Bala attack, the simulation should end immediately.

## E. Foragers

Foragers are responsible for bringing food to the queen. They have two primary modes of behavior: *forage mode* and *return-to-nest mode*. The specific requirements for the forager ant are:

1. Forage Mode

    a. Foragers are considered to be in forage mode whenever they are not carrying food.
    b. In Forage Mode, foragers should always move to the adjacent square containing the highest level of pheromone, except:
        i. If more than one adjacent square has the same level of pheromone they should randomly pick one of those squares.
        ii. When following a pheromone trail a forager should never move into the square it just came from unless it has no other choice.
        iii. Depending on how you implement your movement algorithm, it is possible for a forager to get stuck in a loop, traveling round and round the same squares without getting anywhere. Try to detect when this happens, and prevent the endless looping.
    c. Foragers should maintain a history of their movement, to be used when they need to return to the nest.
    d. When a forager enters a square containing food, it should pick up 1 unit of food, unless it is already carrying food.
    e. When a forager picks up a unit of food, it enters *return-to-nest mode*.
    f. Foragers should *never* pick up food from the square containing the queen.
    g. After a forager has picked up 1 unit of food, it should not move again until the next turn.

2. Return-to-nest Mode

    a. When a forager is carrying food, it should retrace its steps exactly back to the colony entrance; i.e., it should backtrack whatever path it took to get to the food.
    b. Foragers should ignore pheromone in this mode; i.e., they should *not* move to the adjacent square containing the highest level of pheromone.
    c. Foragers should not move randomly in this mode.
    d. Foragers should deposit 10 units of pheromone in each square along the way back to the colony entrance, *including* the square in which the food was found, but *excluding* the colony entrance (the queen's square).
    e. Foragers should only deposit pheromone in a given square if the current pheromone total in the square is < 1000.
    f. A forager may deposit pheromone in one square, and move to a new square in the same turn.
    g. When a forager reaches the colony entrance, it should add the food it is carrying to the food supply in that square, in the same turn in which it entered the colony entrance.
    h. Foragers should not move out of the colony entrance on the same turn they deliver food there.
    i. If a forager dies while carrying food, the food it was carrying should remain in the square in which the forager died.
    j. When a forager has deposited food at the nest, the forager re-enters forage mode, and its movement history should be reset.

## F. Scouts

Scouts are responsible for enlarging the foraging area available to the foragers. The specific requirements for the scout ant are:

1. Scouts should always randomly pick one of the eight possible directions of movement when it is their turn to do something.
   a. If the chosen square is open, the scout should simply move into that square.
   b. If the chosen square is closed, the scout should move into that square and the contents of that square should be revealed.

2. Whenever a closed square is revealed, there is a chance of there being food in the square, according to the following frequency:
   a. There is a 25% chance that the square will contain a random amount of food between 500 and 1000 units.
   b. The other 75% of the time the square is empty.
   c. You can predetermine the contents of all the squares at the beginning of the simulation, or you can dynamically determine the contents of each square as it is opened.

## G. Soldier Ants

Soldiers are responsible for protecting the colony by fighting the enemy Bala ants. Soldier ants have two primary modes of behavior: *scout mode* and *attack mode*. The specific requirements for the soldier ant are:

1. Scout Mode
   1. A soldier is in scout mode when it is in a square that does not contain any Bala ants.
   2. While in scout mode:
      1. If there are one or more Bala ants in one or more of the squares adjacent to the square the soldier is in, the soldier should move into any one of the squares containing a Bala ant.
      2. If there are no Bala ants in any of the adjacent squares, the soldier should move randomly.

2. Attack Mode
   1. A soldier is in attack mode when it is in a square that contains one or more Bala ants. Attack mode takes precedence over scout mode.
   2. While in attack mode, a soldier should attack any Bala ants present.
   3. If there are multiple Bala ants present, only one of them should be attacked.
   4. During an attack, there is a 50% chance the soldier kills the enemy ant; otherwise, the soldier misses and the enemy ant survives.

## H. Bala ants

Bala ants are enemies of the colony. They should enter only at the periphery of the colony (i.e., they should not simply pop up in the middle of the colony). Once in the colony they may move around freely. Assume they never leave the colony once they enter it. The specific requirements for the Bala ant are:

1. Each turn there is a 3% chance one Bala ant will appear in one of the squares at the boundary of the colony. You may choose to have Bala ants always enter at the same square (e.g., upper left corner), or you may have them enter randomly at any of the 106 squares on the edge of the colony.
2. Once a Bala appears, it should remain in the environment until it is killed, or dies of old age.
3. Bala ants should always move randomly.
4. Bala ants may move into squares that have not yet been revealed by scout ants.
5. If a Bala ant is in a square containing one or more friendly ants (scout, forager, soldier, queen), the Bala should attack one of those ants. The ant that is attacked can be selected at random, or you can pick which ant gets attacked.
6. During an attack, there is a 50% chance a Bala kills the ant it attacks; otherwise, the Bala misses and the ant that is attacked survives.

## I. Passage of Time in the Simulation

Time plays an important role in controlling what happens in the simulation. Each "day" in the simulation is divided into 10 "turns". Certain things happen at regular time intervals in the simulation:

1. The queen produces a new ant on the first turn of every day.
2. The pheromone level in each square should decrease by half (rounded down) each day (10 turns), but should never go below zero.
3. Every turn, each ant in the simulation should get a chance to perform an action, as defined above, depending on the ant type.

## J. Death

Ants may die in one of several ways:

1. They may be killed by an attack.
2. They may die of old age (they will die the day after they have reached their maximum allotted lifespan).

3. Starvation (only applies to the queen).

When an ant dies:

1. The dead ant should be removed (i.e., deleted) from the square it is in.
2. If the ant died before it had a chance to do something, it does not get to do anything posthumously on that turn.

**K. Initial State of the Simulation**

Your simulation should begin with the center square of the environment and its adjacent squares open (see figure below).
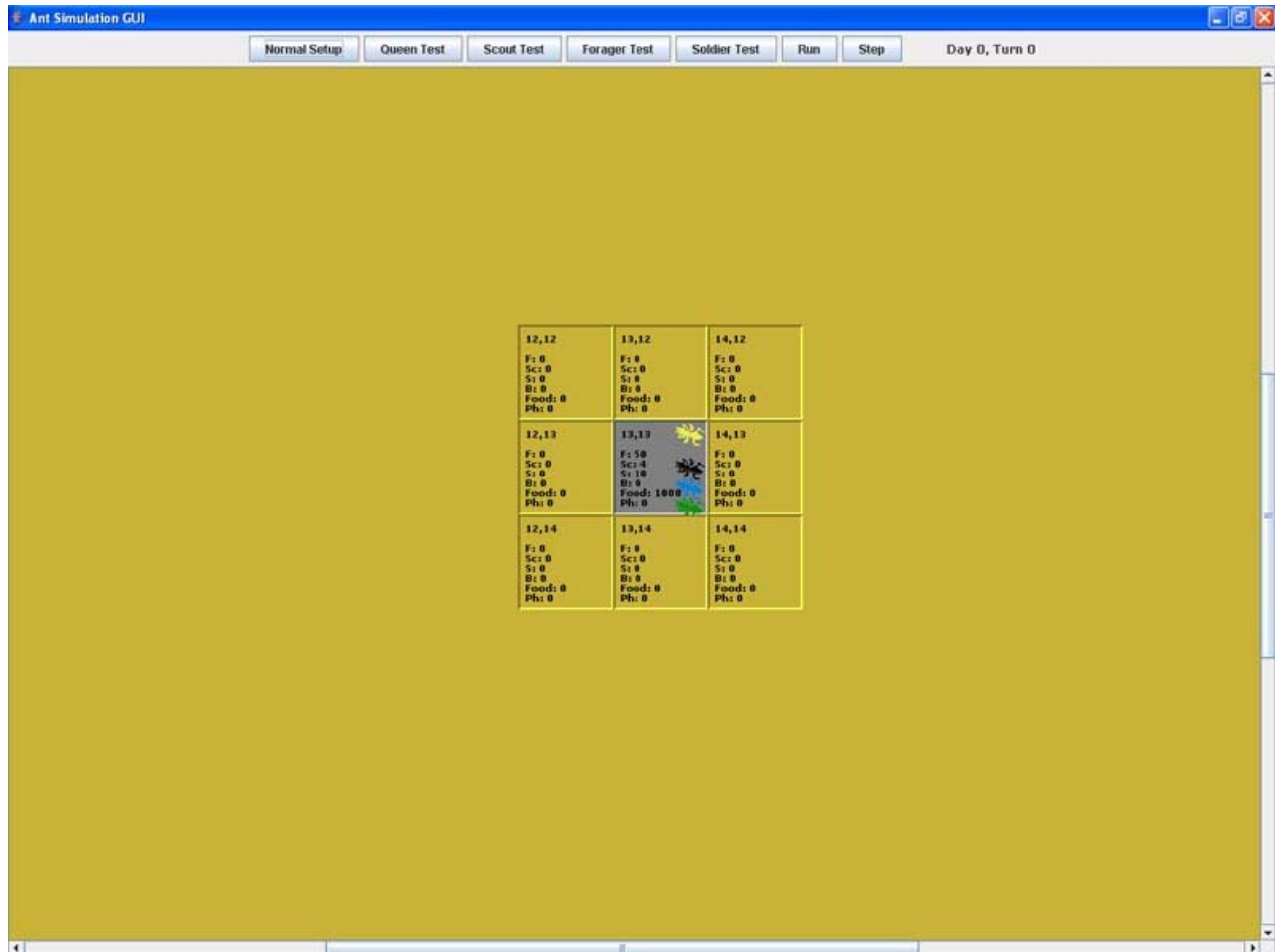


The center square represents the entrance to the colony. It should contain the following:

1. the queen ant
2. 10 soldier ants
3. 50 forager ants
4. 4 scout ants
5. 1000 units of food

**L. Ending State of the Simulation**

The simulation should end immediately after the queen dies, either from starvation, from old age, or from an attack.

**M. Controlling the Simulation**

Since this is a fairly complex project, it is useful to have two different modes of execution:

1. <u>Continuous Execution</u>

   In this mode the simulation simply runs non-stop until the ending state is reached.

2. Stepwise Execution

   In this mode the simulation can be stepped forward one turn at a time, either at the click of a button or by a key press. This is helpful for observing what happens from one turn to the next. This mode is extremely valuable for testing purposes.

**N. User Interface**

You must use a graphical user interface (GUI) for this project. A GUI will allow you to more easily interpret what is happening in the simulation as you test and debug your project. You have two options:

1. You can use the GUI I have provided. The challenge here is that you will need to design your simulation to use it. The GUI has all the necessary methods for your simulation to update it. You will need to understand how those methods work in order to connect your simulation to the interface.

2. You may build your own GUI.

Below is a screenshot from the prototype project I've built, which uses the GUI I have provided. The squares containing text have been revealed by scout ants. The gray square near the center is the colony entrance, and contains the queen ant (the gold-colored ant in the upper right corner of the square). Scout ants are blue, soldier ants are black, and forager ants are green. The violet squares represent the pheromone trail that has been laid down by the foragers that found a food supply in square 9,5. The color changes to the various colors of the spectrum to reflect the strength of the pheromone concentration (violet = lowest, red = highest).

If you decide to use the GUI I have provided, you only need to make sure the following buttons are implemented:

1. **Normal Setup** - Clicking this button should initialize the simulation (see "Initial State of the Simulation", above). This button should be clicked prior to clicking either the Run or Step buttons.
2. **Run** - After the simulation has been set up, clicking this button will run the simulation continuously.
3. **Step** - After the simulation has been set up, clicking this button will run the simulation one turn at a time.

The buttons labeled `<Ant Type>` Test are there for your own convenience for testing purposes. **You are not required to implement these buttons.** You can use these buttons to set up the simulation with special starting conditions that will help you test the operation of each ant type individually, which is very useful.

Examples of different setups you might use are:

1. Queen Test: set up with only the queen node being open, and only the queen ant present
2. Scout Test: set up with only the queen node being open, and with the queen and one scout ant present. You might want to comment out the code that allows your queen to hatch new ants for this test. You might want to comment out the code that allows your queen to hatch new ants for this test.
3. Forager Test: set up with enough open nodes to allow a reasonable food path (at least 3 node lengths), the queen ant and one forager ant present, and food present in only one node other than the queen's node.
4. Soldier Test: set up with several open nodes, the queen ant, one soldier ant, and one Bala ant present.

If you decide to build your own GUI, the minimum requirements are:

1. It should be capable of displaying the environment grid, similar to the figure below.
2. It should be capable of displaying in text format the contents of each square (# of foragers, # of scouts, # of soldiers, # of Balas, whether the queen is present, amount of food, and pheromone level).
3. It should display the simulation time.
4. It should allow the user to choose which mode of execution to use (continuous or stepwise).
5. You are NOT required to display ant icons or use color to indicate the pheromone level if you are building your own GUI, although you may find it helpful to do so.

## O. Use of Appropriate Data Structures and Algorithms

One of the primary objectives of this course is to understand how to use data structures that are more sophisticated than simple arrays. Certain facets of this project can be solved more efficiently and more appropriately using particular data structures. In order to receive full points for this project you must avoid using simple arrays. The one exception is that you may use a two-dimensional array for the grid that models the ants' environment.

To avoid conflicts with the data structures provided in the JDK library, I recommend using only the data structures that I've provided. To use them, simply copy the source files for the data structures into the same directory as the source files for your project.

You will also need to pay attention to how you design your algorithms for controlling the project's execution. For example, traversals will be common, so think about how to do them efficiently to avoid slowing down your program.

## P. Random Number Generation

For several parts of the project you will need to generate random numbers. Java has a built-in class for dealing with random numbers: `Random`. The `Random` class is in the `java.util` package, so you will need to import that package in your source files that use it. The only method you will probably need to use is the `nextInt(int n)` method. This method returns a random integer between 0, inclusive, and n, exclusive. For example, calling `nextInt(20)` will return a random number between 0 and 19. If you need a different range, for example a random number between 1 and 20, you could get that by using the following statement (`r` is a reference to a `Random` object):

```
int randNum = r.nextInt(20) + 1;
```

If you need a more complicated range, for example between 10 and 50, you would need to follow this formula:

```
int n = r.nextInt(maxValue - minValue + 1) + 10;
```

For the range 10 to 50, *maxValue* would be 50 and *minValue* would be 10, so you would write the statement:

```
int n = r.nextInt(41) + 10;
```

Keep in mind that computers don't really generate truly random numbers. Instead, they generate a finite list of pseudorandom

numbers which repeats itself when the end of the list is reached. In order to generate this list, a seed value is needed to do the random number computations. If you create an instance of the `Random` class using the `Random()` constructor, the current system time, in milliseconds, is used as the seed value. This can create a problem. If two instances of the `Random` class get created within the same millisecond, which often happens, they will both have the exact same list of pseudorandom numbers, and your application won't behave as "randomly" as it should. Your best bet is to either:

1. create a single instance of the `Random` class and use it for generating all random numbers (you will probably need to make it `static`, and use a `static` method to access it)
2. use the `Random(long seed)` constructor and specify your own seed value

## Q. Tips & Hints

1. This project may be relatively large compared to what you have done in the past, so don't just sit down and start hacking out code! Analyze the problem and spend some time designing the architecture of your program BEFORE you write any code. Figure out which classes you will need before you start writing any code. The homework assignment for Module 6 will address this issue.
2. Several of the data structures we will be discussing in the course will be very useful for certain situations. Take advantage of them. Except for the grid that represents the environment, do NOT use simple arrays for anything.
3. Good use of inheritance and polymorphism can significantly reduce the amount of code you have to write. Familiarize yourself with the `instanceof` operator and how it is used - you may find it very useful.
4. Don't try to tackle the entire problem all at once. Break the problem down into manageable sub-problems (divide & conquer).
5. *After* you have designed the project, try to code it *incrementally*. For example: first build the environment and make sure you can access the various squares. Then choose one type of ant (e.g., the queen) and add it to the project. Test it to make sure the queen can hatch the various types of ants. Then choose a second type of ant and add it to the project. Test the second type of ant alone, and in combination with the queen ant. Then add a third type of ant, etc.
6. Test your code as you go along. Be sure to unit test individual methods. Don't wait until you have the entire project built before you test, or you may find yourself in serious trouble.
7. **If you get confused or have trouble, PLEASE ASK QUESTIONS!**

## R. Academic Dishonesty

I hate having to say anything about this topic, but past experience compels me to do so. As you are probably aware, you are expected to turn in your own work for this project - it is not a group assignment. This project is complex enough that for all practical purposes it is impossible for students to come up with the same solution by coincidence.

**The penalty for cheating will be a grade of zero for the project for all parties involved.** Since the project is worth 20% of your final grade, it is conceivable that a zero can cost you the equivalent of two full letter grades for the course. Therefore, I strongly encourage you to take the following precautions:

1. Do your own work. I do award partial credit, so you are almost guaranteed to get some points if you at least turn in something.
2. Protect your source files. Do not leave your files on a public computer or any place where someone might be able to gain access to them. The same applies to the disks, flash drives, etc., on which you store your files.
3. If you print out any of your source code, don't leave the printouts where someone might be able to find them. Do not dispose of them in public trash cans or recycle bins. Your best bet is to hang on to them until the end of the semester, or shred them.
4. Do not underestimate the resourcefulness of a desperate student. Again, I hate saying this, but unfortunately these things do happen.

## S. Changes to the Requirements

In the event these requirements need to be changed, I will post a revised version of this document that clearly illustrates the change(s).

## T. What you need to turn in:

1. All of your `.java` source files. If you're worried about leaving something out, package your entire project folder into a zip file and send that to me.
2. Do **not** send me projects in the form of `.class` files or `.jar` files. I must be able to run your program from source files I have compiled on my computer.

**Make sure what you turn in compiles! I can't test your project if it doesn't compile, which means you will lose a substantial number of points.**