CS 134 Programming Exercise 9:
Termites

Objective: To gain experience working with 2 dimensional arrays.

Computers are often used to simulate real world systems. Some simulations are constructed as part of computer games. Others are created for the more serious goal of predicting or understanding the behavior of the simulated system. For example, computer programs are used extensively to simulate the behavior of the atmosphere in hopes of predicting the weather.

The field of artificial life uses computer simulations to explore the behavior of living systems. For this lab, we will have you construct a simple example of such a "living" simulation.

The scenario and model we will be using come from the text "Turtles, Termites and Traffic Jams" by Mitchel Resnick. In his book, Resnick uses simple computer simulations to illustrate the idea that the members of a large population may appear to exhibit cooperative or coordinated behavior when they are actually functioning quite independently and following very simple rules that dictate their individual behaviors.

One of the examples in the text involves the simulation of termites moving wood chips about in a simulated world. We would like you to construct a simulation similar to Resnick's termite program.

The world to be simulated starts out with a large number of wood chips and a somewhat smaller number of termites randomly spread about. The termites then begin to wander about completely aimlessly. At each step, a termite may move up, left, down or right regardless of the direction of its previous steps. If a termite happens to bump into a wood chip, it picks it up. It carries the wood chip around as it wanders until it bumps into another wood chip. At that point, it gets the urge to put the chip it is carrying down, but it doesn't do so immediately. Instead, it continues to wander around until it is no longer standing on top of another wood chip. Then, it puts its own chip down on this empty spot and wanders off randomly to repeat this process.

The surprising thing about this simulation is that although the action of the individual termites is random, the result of their combined work appears to be coordinated. As time progresses, the termites manage to gather all the wood chips into larger and larger piles.

Your program to simulate this world will be quite simple. The termites and woodchips will be represented on the screen by small (3x3?) filled squares. You should definitely use different colors for the wood chip and termite squares. You may want to also use colors to distinguish termites who are carrying chips from those who are not.

The world in which the termites and wood chips reside will be divided into a grid of rows and columns. To keep track of the chips, we suggest you use a two dimensional array of FilledRects with one entry for each cell in the world. If an entry in the array is equal to "null", then the space on the screen corresponding to that cell does not contain a wood chip. Otherwise, the array entry refers to the rectangle used to represent the chip in that cell.

The termites won't be stored in any array. Rather, their independent behavior will be realized by making each of them an active object. To do this, you will define a Termite class that extends ActiveObject. When a Termite is created, it will display a square on the screen to represent itself. Then its run method will randomly move itself around the screen and also move any wood chips it finds following the rules described above. To do this, each Termite, when created, will have to be passed the array used to keep track of the wood chips.

You can simulate a random move by picking random integers between -1 and 1. Add one of these random number to the row number and another to the termite's column number. Note that this approach allows both diagonal moves and the possibility of occasionally not moving at all. You will have to deal with the possibility that a random move will place a termite outside the boundaries of the simulated world. You can deal with this in one of two ways. You can make the world "wrap around" so that if a termite walks off the right edge it appears on the left edge (and similarly for the top and bottom). To do this, use the % operator when adding to row and column numbers. Alternately, you can treat the boundaries as obstacles through which the termites can not walk. To do this, you will need to add if statements that test each move and

reject it if it would require walking through a boundary. Make sure that your termites share the machine nicely by pausing a bit each time they move.

This program is very different from the others you have constructed in this class. It does not involve any user interaction. All that is needed to bring everything to life is a WindowController class whose begin methods randomly spreads termites and wood chips on the screen. The run methods of the termites will do the rest.

See the online version of this handout for instructions on obtaining a copy of the starter folder.

As usual, a working version of this program will be available on Cider Press so that you can see exactly what we have in mind.

Design We will again require you to bring a proposed design for this problem to lab with you. We would like your design to include headers for the methods you will define during the lab, including a brief comment describing what the method does, and declarations for instance variables, parameters and variables local to your methods.

Unlike last week's lab, this week does not involve much planning of which methods to include in which classes. The only method in the WindowController will be begin, and the only public method in your Termite class will be run. As a result, your design should focus on two other issues. Try to determine exactly what instance and local variables you will want in each class and method. Write out declarations for these variables as part of your design. Also, it may help to define some private methods in the Termite class. Such methods are appropriate when some action is required in several places within other methods of the class. For example, in the run method you will discover there are several places where you want to make your termite take a random step. A private `takeAStep` method will make writing the run method much simpler. Include this method and any other private methods you might want in your design.

Do not worry if your design is incomplete. Like an outline for a paper, your design is merely a rough plan for the final product. It is more important that what you include in your design is clear than that the design specify every detail of the program.

Extras There are a number of ways you could extend this project. As described, we intend the interface to the basic program to be simple (i.e. non-existent). You could easily add some sort of controls to specify the density of wood chips and termites before the simulation began.

Another form of extension would be to make the termite behavior a bit more complex. For example, it might seem more realistic to assume termites can somehow sense wood chips if they get close enough to them. What happens if you change your termites so that if they can see a chip within a few spaces straight ahead, they go right for it? Obviously there are many such "senses" you could add and experiment with. For those who would like to get some ideas by looking at other examples of Resnick's simulations, visit the web site:

http://education.mit.edu/starlogo/projects.html

Submitting Your Work When your work is complete you should deposit in the appropriate dropoff folder a copy of the entire folder containing all of your .java files, the project file and the other files Metrowerks requires. Before you do this, make sure the folder name includes your name and the phrase "Lab 9". Also make sure to double check your work for correctness, organization and style.