CS2112—Fall 2012

Assignment 6

Interpretation

Due: Sunday, November 11, 11:59PM

(overview draft due November 6)

In this assignment you will get the critter world fully working by developing an interpreter for the critter rule programming language. Your Assignment 5 implementation had critters choosing actions randomly; now they will choose actions according to their rules. The graphical user interface developed in the last programming assignment should continue to be supported by your implementation, with some small additions. Your program will also support loading the initial state of the world from a file, and will support adding new creatures whose rules are defined by a file loaded from the file system.

In addition to implementing new functionality, you are expected to correctly complete any parts of assignments 4 and 5 not implemented correctly as part of earlier assignments.

0 Changes

- The input file format was fixed to specify posture rather than appearance. (11/9)

1 Requirements

1.1 Graphical user interface

The graphical user interface will be extended to support the new functionality of the application.

- Information displayed about the currently selected critter will include more information about its state will include a display of the currently executed rule, as well as at least the first few memory locations. Exactly what is displayed is a design choice.
- The user interface will provide controls for loading a world definition from a file. A world definition will specify the locations of rocks and plants using the syntax defined below.
- The user interface will provide controls for loading a new critter definition from a file. Critter definitions will consist of rules in the syntax described in Assignment 4, plus attribute definitions as described below.
- You may choose to go beyond the minimal UI requirements specified here. Useful extensions to the user interface will give additional points on this part of the assignment. (However, the UI is only a fraction of the total points, so don't go overboard.)

You already implemented a graphical user interface as part of the previous assignment. For this assignment, you will need to integrate the interpreter that you will write for the critter rules with

your previous code so that the critters' actions are dictated by their programs (although it would certainly be useful to retain the ability to manually control a critter). You will also need to make some extensions to your GUI, which are described below.

1.2 Simulating critter rules

The core of this assignment is implementing an *interpreter* for critter programs. An interpreter is a program that emulates the execution of programs written in some programming language. For example, the Java runtime system includes a *bytecode interpreter* that executes code from Java class files.

Your interpreter will work directly on the AST generated by the HW4 parser. It will interpret the rules by recursively evaluating the AST nodes representing conditions and expressions, in the context of the current state of the critter and the world state. It will execute rules until the action to use is decided and return that command. It will also update the critter's memory as described by the rules applied.

You will need to implement the recursive algorithm described in the project specification to decide which action to take using the evaluated AST. You will also need to call your AST mutation code for mating and budding.

We are also providing an example AST implementation with this assignment, including code for mutation. You are not required to use it, and if you do choose to use it, you are allowed to make any changes to it. However, keep in mind that if you decide to use our AST implementation, you will need to write a new parser that generates our AST in order to be able to read rules files. Even if you choose not to use our AST implementation, it will probably be helpful to look at the stubs for `eval` to determine how to interpret your own AST.

1.3 Loading new critters

Your graphical user interface should be extended with a way to load a rules file and add new critters to the world using those rules. This requires integration with your parser from HW4. In addition to specifying a rules file to load, you should also allow the user to specify how many critters to create using those rules; that number of critters are then randomly placed in the world. It is encouraged but not required to also implement an alternate method of adding critters where instead of placing them randomly, the user can click on an empty hex to place a critter there.

The syntax of a critter file is as follows. It should begin with a specification of some of the first few memory locations in the following format:

```
memsize: <memory size>
defense: <defensive ability>
offense: <offensive ability>
size: <size>
energy: <energy>
posture: <posture>
```

Each of the values specified is an integer. You can assume the attributes appear in this order, but you may choose to be more flexible. You may choose to support additional attributes too.

Following this section of the input file, the critter rules should appear, in the syntax described in HW4.

Your critter file parser should ignore blank lines and lines that start with a double slash (`//`).

You should also provide a way to display the program of a critter in the world. This shouldn't be too difficult—you already implemented the ability to select a critter in the previous assignment, and your AST has a pretty-print method that generates appropriate text.

1.4 Loading world definitions

Your graphical user interface should be extended with a way to load a world from a text file. Each line in the text file will have one of the following forms:

1. `plant` *`<row> <column>`*
2. `rock` *`<row> <column>`*
3. `critter` *`<critter_file> <row> <column> <direction>`*

You are not required to check for objects being placed on the same hex or on hexes outside of the game world, although it is encouraged to do so.

See `world.txt` for an example world specification.

1.5 Running your program

We require your program to support the following command-line interface:

- `java -jar <your_jar>`
  Start the simulation with a world populated by randomly placed plants and rocks. Your program should automatically read the file `constants.txt` to determine the world parameters.
- `java -jar <your_jar> <world_file>`
  Start the simulation with the world specified in file `world_file`. If there are any critters specified in `world_file`, your program will open and parse their associated rules files. Your program should automatically read the file `constants.txt` to determine the world parameters.

1.6 Plant growth probability

We mentioned in the previous assignment that while each hex adjacent to a plant has probability `PLANT_GROW_PROB` of growing a plant each step, this probability should actually be scaled and evaluated between critter actions, or $(1 +$ number of critters$)$ per time step. If $p =$ `PLANT_GROW_PROB`

and we want to evaluate this probability in n pieces (i.e. we have $n-1$ critters), then the scaled probability q is found as follows:

$$\begin{aligned}(1-q)^n &= 1-p \\ 1-q &= \sqrt[n]{1-p} \\ q &= 1-\sqrt[n]{1-p}\end{aligned}$$

A study of probability theory is outside the scope of this course, but if you're interested, you can look up "exponential distribution" on Wikipedia; this is the distribution that we are using to model continuous plant growth.

In addition to plant propagation, on each time step `PLANTS_CREATED_PER_TURN` plants are spontaneously generated. We want this generation to be fair to the critters regardless of when in the turn order the critter gets to move, so you should generate each plant at a random time between critter turns during the time step, rather than all at the beginning or all at the end.

All of these probabilities can be slightly thrown off if new critters are added to the turn order during the time step. Don't worry about these inaccuracies; we only require your probabilities to be mostly correct.

We are providing a data structure for efficiently generating low-probability events, in the `eventgen` package. This package will enable you to significantly speed up the simulation of plant growth, which otherwise tends to dominate run time. Use of this code is optional.

2 Programming tasks

You will want to figure out with your partner how to break up the work involved in this assignment. To get you started thinking about this, here are some of the major tasks involved:

- Designing a good suite of test cases to ensure that the whole critter language is correctly implemented. You may even want to add more diagnostic features to the user interface to help you evaluate whether the program is working correctly.
- Extending your user interface to support additional program features such as loading new world layouts and critters.
- Implementing a new component or components to display the state of the critter world. This will involves graphically rendering hexes and critters. It should be possible to at least distinguish critters of different species, and to see the size and direction of a critter.
- Implementing the state of the critter world and all the critters in a way that is decoupled from the graphical display, according to the model-view-controller design pattern.

Please get started early and plan with your partner how you're going to do this assignment.

3 Restrictions

You may use any standard Java libraries from the Java SDK. However, you may not use a parser generator.

4 Overview Draft

We are requiring you to submit an early draft of your design overview document by November 6. As usual, you may not be able to predict what your design and testing strategy will look like in full at that point, but we want to see how far you have gotten. We will aim to get you quick feedback on this draft.

5 Submission

You should compress exactly these files into a zip file that you will then submit on CMS:

- *Source code*: You should include all source code required to compile and run the project.
- *Other files*: It is possible to use other files as part of your UI. For example, you might read in image files or other data files that control appearance. Don't forget to include these.
- *Tests*: You should include code for all your test cases.
- `overview.txt/html/pdf`: This file should contain your overview document.

Do not include any files ending in `.class`. We expect you to stick to Java 6 features and avoid features found only in Java 7. You can set project properties in Eclipse so that it warns you when Java 7 features are being used.

6 Written problem

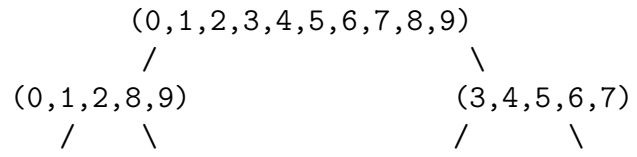
The following problem is to be done jointly with your partner.

Suppose you are sorting arrays of integers on a massively multicore processor. You decide to use a *concurrent merge sort* in which different threads work on the different subarrays. Your first cut looks like this:

```
/** Place elements x[lo..hi-1] in sorted order.
    Requires: ... */
static void sort(final int[] x, final int lo, final int hi, final int[] y) {
   if (hi <= lo + 1) return;
   final int mid = (lo + hi) / 2;
   final Barrier barrier = new Barrier();
   final Thread t = new Thread() {
       public void run() {
         sort(...);
         synchronized(barrier) {
             barrier.notifyAll();
         }
       }
   };
   t.start();
   sort(...);
   synchronized(barrier) { barrier.wait(); }
   merge(x, lo, mid, hi, y);
 }
 class Barrier {} // really just an Object so far...
```

1. Fill in the three missing parts marked "...", including the requires clause, to correctly implement the mergesort algorithm, modulo any synchronization issues.

2. Suppose you use your algorithm to sort the following array: (0,9,1,8,2,7,3,6,4,5). Complete the following call tree to show how the calls to `merge()` arrive at the final result. Put a star on each call that occurs in the original thread. (You do not need to show the contents of `y`).



3. After filling in the missing recursive call arguments in the code, you discover that the `sort()` function often fails to return. Briefly explain the sequence of events that can cause this to happen.

4. Fix the problem identified in the previous part by changing the class `Barrier` and the uses of the variable `barrier`. With the exception of `Barrier`, you should be able to make the above code look simpler. (Hint: what is the *condition* the main thread is waiting for?)