CS2112—Fall 2012

Homework 7 : Distributed and concurrent programming

Due: Friday, November 30, 11:59PM

In this assignment you will make your critter simulation a distributed web application. Additionally, your project will reflect the critter's accurate sense of smell when detecting nearest food or plant.

In addition to implementing new functionality, you are expected to correctly complete any parts of assignments 4–6 not implemented correctly earlier.

0 Changes

- (11/29) Some of the signatures have been changed. Because this is a late change, it is okay to implement the original signatures. The changes are as follows:
  - The enumeration `Server.Action` has been removed. `RemoteCritter.Action` should be used instead.
  - The `TAG` action has been added to the enumeration.
  - An accessor has been added to get the tag of a critter.
  - An accessor to read the amount of food in a cell has been added.
- Some specifications in the `Server` interface were clarified. See Piazza for details and discussion.

1 Distributed Programming

For this part of the assignment, you will refactor your previous code into a client and a server. The client will be a Java program that provides views of and allows interactions with the world. The server will simulate the world. Multiple clients will be able to connect to a single server. The client and the server will communicate using Java remote method invocation (RMI).

1.1 Client

Your client will support two views: the user view, and the admin view. These may be in separate windows or in the same window, as you choose.

The user view corresponds to the functionality you have already implemented. This view visually renders the simulation and provides options for uploading and downloading critters.

However, you do need to add support for the notion of species (critters with the same program): users should be able to view a species' attributes (length of memory, defense, offense), its program, and its lineage (the species from which it evolved). The user view will also provide an option for authenticating as an administrator and opening the admin view.

The admin view provides several options for managing the simulation. The administrator should be able to perform at least the following tasks:

- Restart the simulation.
- Load a new world.
- Alter simulation parameters.
- Take control of single critters
- Enable/disable critter uploads and downloads.
- Approve and manage user credentials.

You may add additional functionality to the admin view. For example, you might support broadcasting a message to all clients.

1.2 Server

The client will communicate with the server via remote method invocation (RMI) calls to objects on the server's JVM. The server must be able to handle multiple clients concurrently. The server should implement the released interfaces for compatibility.

The server will allow new user identities to be created through the admin interface. This is useful because different users should have different rights in general. For example, if using the simulation as a game, the users actually playing the game might be authorized to load new critters, whereas other users would only be able to view the state of the game.

There will be three tiers of user credentials: view-only (no special authorization required), users with permissions to load and control critters, and users with admin permissions.

User identities will have passwords so that users can authenticate themselves to the system when they connect with a client. These passwords will be stored, along with the user identities, in a file. It would be sensible to store the passwords in a salted-and-hashed form so they cannot be read, but this is not required. Before starting your server, you should add at least one admin user for your own use.

To run your server (which needs a main method that binds itself to the machine's RMI registry), you need to run `start rmiregistry` in a terminal first and provide this command-line argument: `-Djava.rmi.server.codebase=file:/PATH_TO_SRC_ROOT`.

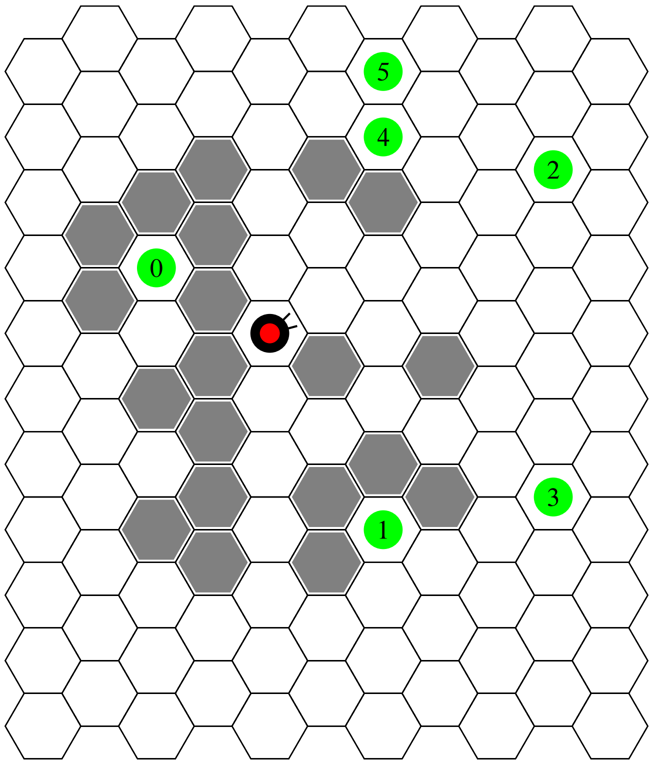

Figure 1: Finding food in a challenging environment

2 Improved food sensing

We will be giving critters a sense of smell to make it easier to find food. The new sensor expression `food` returns information about how to get to the closest food, taking into account obstacles in critters' way. For instance, Figure 1 illustrates an environment in which the critter is heading northeast. The closest food is at distance 2 to the northwest (relative direction 4). But because of the rock wall, at least 12 moves are required to approach that food. Meanwhile, no obstacles stand in the way of food #2 at distance 5. Figure 2 shows the distance from the critter to various hexes, taking into account obstacles.

The result of the `food` expression is based on two parameters: *distance* and *direction*. The distance is the fewest number of moves to the hex containing food, as long as it is no more than `MAX_SMELL_DIST` = 10. The direction is relative to the critter's current orientation and is toward a hex that decreases the minimum number of moves to food.

In Figure 1, there are three plants at distance 5. Any of these could be chosen. To reduce the distance to the closest food, the critter could move either north to get closer to foods 4 and 5, or northeast to get closer to foods 2 and 4. The corresponding valid relative directions are 0 and 5.

Given distance and direction as defined, the result of the `food` expression is:

$$distance * 1,000 + direction$$

If food is directly underneath the critter, the `food` expression evaluates to zero. If there is no

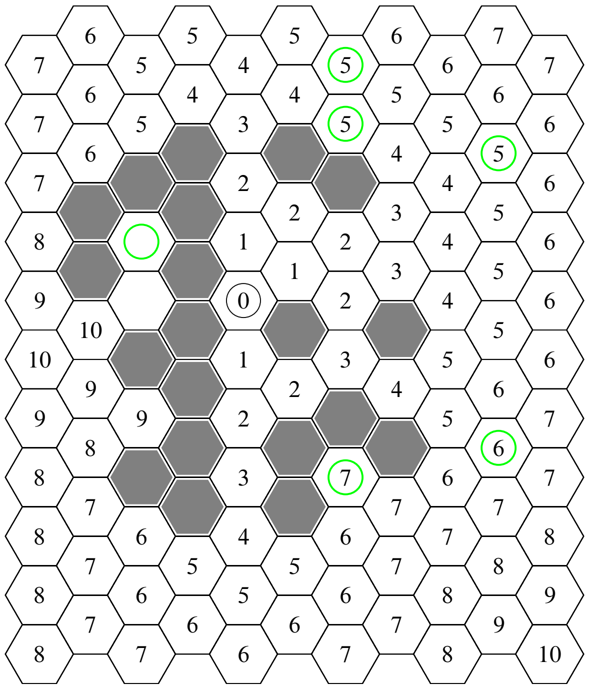

Figure 2: The least number of moves from the critter to various hexes

food within a 10-hex walk, `food` evaluates to 1,000,000.

Describe your implementation approach and any specification changes you made in the overview document.

3 Synchronized ring buffers

Each RMI call from a client spawns a new thread on the server, which could overwhelm the server if there are a lot of clients. One way to deal with this problem is for the server to provide a fixed number of threads for handling client requests. The threads spawned by RMI calls place their requests into a queue that the fixed set of threads draws its work from. This approach, which is called a *thread pool*, bounds the number of server threads doing work.

To implement this approach, you will need a thread-safe queue similar to that described in the interface `java.util.concurrent.BlockingQueue`. You are expected to build such a queue and to use it in your server implementation.

You will implement your thread-safe queue as a *ring buffer*, a data structure commonly used in distributed systems with limited memory. A ring buffer is a fixed size queue that is implemented using an array and two integer indices. Items added to the queue are inserted to the array and the tail index is incremented. If there is insufficient space in the array (as determined by comparing the two indices), then adding to the queue fails. When items are popped from the queue, the item in the slot indicated by the head index is returned and and the head is incremented.

We would like you to implement a thread-safe version of this data structure. It should implement the `java.util.concurrent.BlockingQueue` interface. You are responsible for implementing the methods listed below (all others should throw an `UnsupportedOperationException`). You must use an array to implemenent this and are not allowed to use any classes from the `java.util` package or subpackages. The provided class `student.util.RingBufferFactory` should use your implementation.

Required Methods

From `Collections`: `add`, `contains`, `equals`, `isEmpty`, `iterator`, `size`.

From `Queue`: all methods.

From `BlockingQueue`: all methods.

Tips: Testing concurrent code is very tricky because it never works the same way twice. To debug your ring buffer implementation separately from the server code that uses it, it may be helpful to plug in one of the existing thread-safe queue implementations already available in Java, then replace it with your implementation once you are confident in the client code. The ring buffer itself is certainly worth developing a test harness for. Checking invariants with assertions is highly recommended as a way to catch bugs in this implementation.

4 Restrictions

Do not include any files ending in `.class`. To make it easier for us to grade your assignments, we expect you to stick to Java 6 features and avoid features found only in Java 7. It is easy to set the project properties in Eclipse so that it warns you when Java 7 features are being used, and you should do that.

5 Evaluation

For this project, we will be grading not just the new features described above, but also the project as a whole. Most of the points on this assignment will be allocated to how well the system works as a whole. It is therefore important to make sure that problems with your earlier assignment submissions are fixed.

We also expect that you will test your project carefully and that you will document your testing procedures. Your testing methods will be a significant part of your grade.

6 Submission

As before, we are requiring you to submit an early draft of your design overview document. This is due November 20.

By the homework due date (November 30), you should submit these files on CMS:

- *Overview document*. You should include your overview document summarizing the project and your design, as in previous assignments.
- *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. You will supply a zip file containing these files.
- *Tests*: You should include code or test scripts for all your test cases. Test scripts are scripts to be followed when testing a system through its UI. A test script is preferable to just telling us that you tested the system manually. You should have explicit test cases for any algorithms and data structures that you implement.
- *Solution to the written problem.* This problem (below) is to be done with your partner.

7 Extensions

An interesting extension would be to make your client program run as a Java applet viewable through a web browser. Because the browser supports Swing, this is not inherently difficult. However, it does require that your code be signed so that the browser will let it make network connections.

8 Written problem

Consider the following algorithm:

```
// Effects: set b[0] to the minimum of all elements in a, and b[1] to
//    their maximum.
// Requires: a contains at least one element.
// Performance: given array length n, performs at most ⌈3n/2⌉
//    element comparisons.
//
void maxmin(int[] a, int[] b) {
    int max = a[0], min = a[0], n = a.length;
    for (int i = 1; i <= n-2; i += 2) {
        int j, k;
        int c = (a[i] > a[i+1]) ? 0 : 1;
        j = a[i + c];
        k = a[i + 1 - c];
        if (j > max) max = j;
        if (k < min) min = k;
    }
    if (a[n-1] > max) max = a[n-1];
    if (a[n-1] < min) min = a[n-1];
    b[0] = min;
    b[1] = max;
}
```

a) Give a loop invariant for the `for` loop that is strong enough to argue that the method satisfies its specification.

b) What is the postcondition that the loop needs to satisfy for the spec to be implemented correctly?

c) Argue that the loop satisfies the Initialization, Maintenance, and Postcondition properties.