CS2112—Spring 2012

Homework 7 : Distributed and concurrent programming

Due: ***Saturday, May 5*****, 11:59**PM

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

- April 27: removed the unsynchronized version of the ring buffer.
- May 1: Revised specs for the `Server` class. The authentication methods now return `Remote` objects instead of strings. Since this is a late change, it is okay to implement the original specs. The new specs should be easier to deal with, though.
- May 1: Due date extended until May 5.

1 Distributed Programming

For this part of the assignment, you should refactor your previous code into a client and a server. The client will be a Java applet 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.

1.1 Client

You will convert your previously-written GUI into a Java applet viewable through a web browser. Your applet will support two views: the user view, and the admin view.

In the user view, the applet will visually render the simulation and provide options for uploading and downloading critters. You will 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 applet will also provide an option for switching into the admin view.

In the admin view, the applet will provide several options for managing the simulation. The administrator should be able to perform 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 applet 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`.

2 Improved food sensing

As defined for HW6, the `food` sensor can lead to a false indication of the effort required to reach food. Effectively, this is because critters currently can smell food through a rock wall. 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) according to the original spec. But because of the rock wall, at least 12 moves are required to approach that food. Meanwhile, no obstacles stand

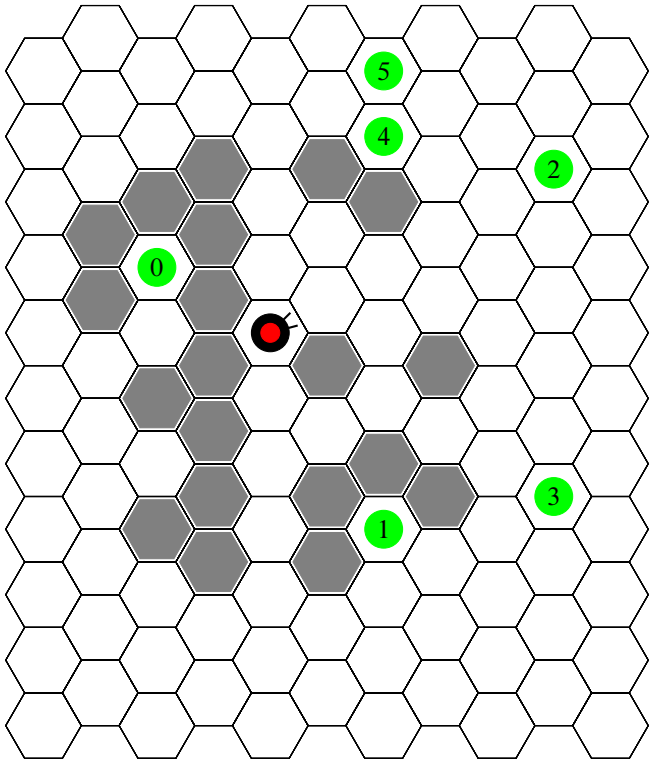

Figure 1: Finding food in a challenging environment

in the way of food #2 at distance 5. Figure 2 shows the distance from the critter to various hexes, taking into account obstacles.

For this assignment, you will make `food` return information about how to get to the closest food, taking into account obstacles in critters' way. The formula for computing the result of the `food` expression remains unchanged from the project specifications, except that *distance* and *direction* are redefined. The distance is now the fewest number of moves to the hex containing food, as long as it is no more than 10. The direction is still relative to the critter's current orientation, but is now toward a hex that decreases the minimum number of moves to the food.

In the example, 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.

Include your implementation approach and specify tiebreakers you use in the overview document.

3 Randomness

The critter rule language has acquired a new expression: `random[`n`]`. It returns a random integer in $[0, n)$.

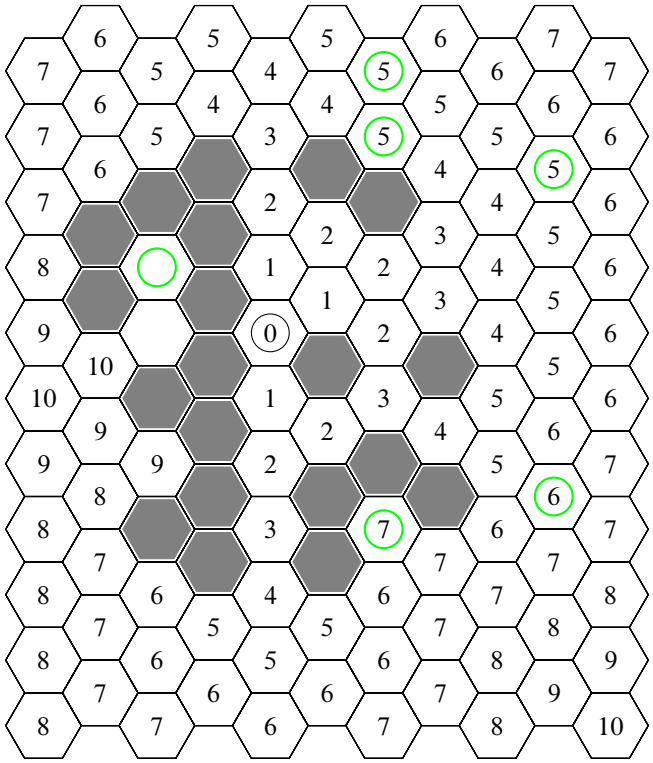

Figure 2: The fewest numbers of moves from the critter to various hexes

4 Synchronized ring buffers

Each RMI call from a client will spawn a new thread on the server, which can overwhelm the server. 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 tends to keep the number of server threads bounded.

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 anything in the `java.util`

class. Your implementation should be used in the provided `student.util.RingBufferFactory` class.

Required Methods

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

From `Queue`: all methods.

From `BlockingQueue`: all methods.

5 Tips

In order for your applet to have file-system access on the client side, the applet will need to be signed with a certificate. This can be done as follows:

- Export the applet code to a jar file.
- Run `keytool -genkey -alias YOURALIAS -keypass YOURKEYPASS` in a terminal to generate your chosen alias and password.
- Run `keytool -alias YOURALIAS -selfcert`.
- Sign the jar by running `jarsigner YOURJAR YOURALIAS`

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 also worth testing in isolation using a simple harness.

6 Edible karma

We will be identifying two projects: the one with the best GUI implementation, and the one with the most correct and fastest world simulation. These two groups will receive special mention, bragging rights, and gift certificates for dessert.

7 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.

8 Submission

As before, we are requiring you to submit an early draft of your design overview document. This is due April 26.

By the homework due date (May 3), you should submit these files 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. 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.
- `overview.txt/html/pdf`: This file should contain your overview document.