Darwin 2.0, Part II

Due 11:59 pm Mon, April 28, 2008
Tournament Submission Deadline 2:30pm Tues, April 29, 2008

1 Assignment

Download the SDK and programmer's guide from http://cs.williams.edu/~morgan/cs136/darwin2.0 (the API is the same as last week but the security model is a little tighter for this competition.)

Implement at least one Creature that reliably beats Flytrap, Rover, and SuperRover on all ns_ maps in the SDK and that wins at least 50% of the time on random maps against the Pirate. You may use any strategy that you wish.

For full credit, your solution must have the following properties:

1. A single Java source file[1] containing a public Creature subclass
   a. Name your creature whatever you want, but ensure that it is a unique name among the class. There's a running list of reserved names next to the board in lab.
   b. Override getAuthorName and getDescription so that we know who you are.
2. Additional optional data files, contained in a directory of the same name as your class
3. A detailed description of your Creature's strategy and why you think it is good in the comments of your creature.
4. Exercise good programming discipline with regard to efficiency, readability and documentation. This includes documenting why each method is threadsafe if that is an issue.

I provide nine sample maps and encourage you to create your own for testing. I will test your creature on these maps as well as new ones.

Submit your solution using the commands:

```
tar –cvf darwin3.tar files
turnin –c 136 darwin3.tar
```

where *files* are the files needed by your creature. If you have only a single Java file, just list that. If your solution contains icons or data files, include them. Do not include the starter code or any .class files.

Unlike previous labs, this one is very open-ended. Before I gave you a problem and a solution design or algorithm; you just had to implement it. Now I'm giving you a problem for which there is no perfect solution (that we know of.) You're going to have to stretch a little and design your own algorithms, and then implement them. I encourage you to discuss ideas with other students as well as the TAs and to read about various artificial intelligence algorithms. Just don't show each other your source code and be sure to cite references.

2 Optional: Tournament

Next week we will run a tournament at 2:25 pm in lab (i.e., between lab sections) to determine the fittest creature. Performance in this contest has no impact on your grade and you may opt out of the contest. The contest winner will receive a small prize.

If you do not opt out, whatever creature(s) you handed in will run in the tournament. You may also submit alternative creatures for the contest up until 24 hours before the contest. Each person may submit up to two creatures. If you only made one, it is to your advantage to make a copy of it with a different name and submit it twice. Leave a comment explaining what is going on, though, so you don't drive the grader crazy.

[1] Even if you create multiple classes. Note that in Java, multiple classes may be placed in the same file as long as only one of them is public. You can also use inner classes.

Kyle Whitson is the contest judge. Submit your updated creatures to him or opt out by email at

Kyle.A.Whitson@williams.edu

He will run the tournament and adjudicate any disputes. Kyle's rulings are final.

I encourage you to share compiled (.class) file versions of your creature and to share .txt maps you've made with other students for training purposes. Do not share source code—this counts as part of your assignment (even though it is optional).

The contest consists of two rounds. During round 1 all creatures will compete in multiple pairwise trials (and yes, your two creatures will fight each other!) In each trial, scores will be awarded as follows:

- All player creatures extinct: 0 points each
- Time elapsed with no majority population: 1 point each
- Time elapsed with a majority population: 2 points to the victor, 1 point to the minority.
- Total domination: 3 points to the victor, 0 points to the loser.

Scores from all trials will be averaged and posted. The top four creatures will advance to the second round. No person can have more than one creature advance, so if one person has two creatures in the top four, that person can choose which one advances. In the event of a tie for fourth place, Kyle will choose the based on their icons which creature advances.

In round 2, the top creatures from round 1 will compete simultaneously against each other on a new four-way map. There will be up to three trials. The victor is the creature that wins two out of three (by population or domination); if there is a tie after the third trial Kyle will chose the winner based on icons.

The walls (but not other features) of the tournament maps will be revealed Sunday night.

2.1 Disqualification

If the system crashes during a trial, the judge will determine the cause of the crash. If it is determined to be a Creature's fault, that Creature will be disqualified. Otherwise that trial will be re-run.

You will be disqualified if any of the following criteria are violated:

1. Creatures must be provided to the judge 24 hours before the contest runs as a zipfile or tar file
    a. Creatures must be supplied to the judge as Java source files.
    b. Creatures must compile without errors on Mac OS X 10.4 and OS X 10.5
    c. All classes must be within the same file.
    d. Additional data files *may* be used. These should be contained in a directory with the same name as your class. "E.g., KyleCritter/lookuptable.txt"
2. Creatures may not:
    a. Cause a stack overflow.
    b. Cause out of memory errors by consuming more than 100 MB of memory collectively.
    c. Cause a denial of service by consuming more than twice their number of Threads.

The following are specifically <u>not</u> grounds for disqualification:

1. Attempting to exploit flaws in the security of the simulator itself.
2. Throwing uncaught exceptions, or otherwise terminating your own Creature's threads.
3. Opening network connections.
4. Reading from the file system.
5. Continuing to execute after having been converted.

Keep in mind that, as with any assignment, you are bound by the CS136 policies, the Computer Science Computer Usage policy, and the Honor Code. For example, your program may not attack the host

computer, its filesystem, the network, the user account, and so on. You may not look at anyone else's source code and may not use a decompiler or any tools except those provided to you for this course.

3 Evaluation

Beats Pirates	**20**
Beats SuperRovers	**30**
Design*	**20**
Readability	**30**
Total	**100**
Custom 3D icon	+2 extra credit

* Including algorithms and thread safety

4 Advice

Make a strategy that isn't completely trivial and then explain it very clearly in your comments and through your program design. 50% of this lab is about making a good creature, 50% is about explaining your creature's design through good coding practice. Don't make out cleverness and then fail to express it clearly!

Your support code from last week is a good starting point for making a competitive creature. However, good strategies don't have to be complex or involve inter-instance communication. Try lots of different, simple strategies to see what works against the existing creatures. Then "evolve" your best strategy to see how it can be improved. Keep in mind that what works on one map might not work well on another map. For example, the Wolf is very bad at maps with lots of tiny corridors. The Rover is very bad at maps with Thorns.

You can see movies of some of the best creatures that have been created by other programmers on the Darwin website. Pirate is 90 lines of code and does not communicate between instances. Sheep is 170 lines of code and does not communicate between instances. Wolf is 370 lines of code and communicates a small amount of information between instances using a HashSet. It does not keep track of the map at all.

Whenever you find yourself thinking "I wish Creature had a method that did...X", take another look at the documentation. Observation, Direction, and Creature have lots of helper methods and fields designed specifically to aid in the kinds of tasks that you're going to encounter.

Here are some ideas for strategies. I don't know which of these are *good* ideas, and their suitability might depend on the type of map and values of the constants.

1. Model some high-level mental states for your creatures that you think might lead to useful behaviors. Sheep is based on the observation that creatures that clump together tend to do well by covering each other and thus preventing infiltration. A Sheep essentially has two parameters: lonliness and restlessness. When a Sheep can see another Sheep in front of it, it becomes less lonely and more restless. Otherwise it becomes more lonely and less restless. A Sheep that is very lonely runs towards other Sheep. A Sheep that is very restless wanders randomly. (They of course always attack an enemy that is right in front of them.)
2. Use remembered and shared information to avoid looking. Once they've built up a pretty good shared map, creatures only need to look to find enemies. They already know where the walls and other friendly creatures are, so they can navigate fine based on recalled information, which makes them much faster than something like SuperRover, which looks all of the time.
3. Allow creatures to specialize. You can only have one Creature class, but that doesn't mean that you have to have all of the instances act the same. On birth, a creature might decide to take on one of several roles, like leader, defender, scout, or attacker. In fact, that decision might be based on how many other instances are currently filling those roles.
4. Exploit map features. I've heard of creatures that try to hide in protected spots, ones that operate differently in narrow corridors than in wide spaces, and ones who try to control choke points (doorways) in the map.