The Maze

Due 11:59 pm Mon, April 7, 2008
Tournament Submission Deadline 2:30pm Tues, April 8, 2008

1 Assignment

Implement a subclass of Creature within the Darwin simulator API that successfully navigates a maze in a limited amount of time. The Darwin Game is a virtual world in which Creatures race against time to complete mazes. You play the game by programming a Creature in Java. Your creature has several actions available and you are provided with both text mode and GUI simulators for debugging your creature. A full description of the API and constraints on the simulation can be found in the Darwin Programmer Guide and the generated documentation (type javadoc136 to produce it).

A subgoal of this assignment is to extend your ability to work with a specification and API. The programmer's guide, Darwin documentation, Java documentation and provided source code convey all of the information that you need. Consult all of those before asking questions.

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 try to 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. Completes **arbitrary** mazes of size 30 x 30 or smaller in fewer than 25,000 steps
    a. Mazes will contain exactly one treasure, one spawn point, and walls. They will not contain thorns, apples, and or other creatures.
4. Exercise good programming discipline with regard to efficiency, readability and documentation

I provide two sample mazes: mz_spyrus and mz_pacman. As a baseline, the simplest reliable algorithm I could write ("Tortoise") finishes mz_spyrus in about 4922 and ms_pacman in about 5241 steps. Your solution should be at least this good. I will test your creature on new mazes, so be ensure that it works on mazes with different structure!

Submit your solution using the commands:

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

2 Optional: Tournament

Next week we will run a tournament at 2:25 pm in lab (i.e., between lab sections) to determine the fastest maze 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, I will run your solution in the contest. You may submit an alternative maze creature for the contest up until 24 hours before the contest.

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

Kyle.A.Whitson@williams.edu

[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.

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 race the map multiple times and their times for each trial will be averaged. These results will be posted. In round 2, the top three creatures will run live through a second map (in reverse order of their round 1 times) with everyone watching. The creature with the lowest time on this second map wins the tournament. In the event of a tie in round 2, the top creatures will compete on a third new maze. In the event of another tie, Kyle will choose the winner by evaluating your icons and text descriptions. The mazes will not be revealed before the contest begins. They will have the same restrictions as for the assignment, however.

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 not met:

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 judges 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 change the priority of any Thread.
3. Creatures may not cause a stack overflow.
4. Creatures may not invoke System.exit or cause it to be invoked.
5. Creatures may not cause out of memory errors by consuming more than 100 MB of memory collectively.
6. Creatures may not cause a denial of service by consuming more than twice their number of Threads.

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

1. Attempting to exploit holes in the security of the simulator itself.
2. Throwing uncaught exceptions, or otherwise crashing your own Creature's threads.

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 (excepting members of an instructor-approved team) and may not use a decompiler or any tools except those provided to you for this course.

3 Evaluation

Prelab Exercise[2]	**20**
Correctness	**15**
Readability	**15**
Design	**15**
Efficiency	**15**
Total	**80**
Custom 3D icon	+2 extra credit
Custom test mazes	+2 extra credit

[2] Handed out during lab

4 Advice

Navigating a maze is a simple recursive exercise. Chapter 10.3 describes one way of doing it, where the recursion is converted to iteration using an explicit stack. My Tortoise solution takes about 65 lines of code (only about 8 of which are "the algorithm" it self) and does not use an explicit stack, although it does use a Set to keep track of previously visited locations. I did not follow chapter 10.3 very closely in my implementation but the core idea is the same. Think about a maze as a tree of decisions: Do I go left? Right? Straight? The key is that when you find your way blocked you need to backtrack. With recursion your program automatically back tracks, so just make sure that when a recursive call returns it leaves the Creature in the same position and direction that it started in. You should be able to write your solution in about 30 minutes if you think for a long time first. Debugging might take up to another half hour.

Efficiently navigating a maze is trickier. You want to avoid moving backwards for long distances, since turning around is faster if you have many squares to traverse. You want to avoid looking down the same alleys multiple times, since nothing could have changed since your last look. Finally, if you see the treasure, there's no reason to keep wandering: run to the goal!

By now you've probably learned that code you write one week may be useful in the future. Keep that in mind while working on this project. You may find that in the future you have another lab where some of the pieces of your Creature will be reusable, and you'll want to have documented and tested them…

For the required part of the assignment, I do not expect you to implement a particularly efficient creature. You should be able to beat or tie the Tortoise's times and do so in an elegant fashion, but don't need to use a better algorithm. That makes the required part of this lab easy, since by now you should be able to turn a specification into working code quickly.

For the contest, I expect some of you to be very devious. To place highly in the contest you'll probably need some sophisticated data structures and algorithms.