CMSI 186

PROGRAMMING LABORATORY

Spring 2008

Program 6: Backtracking

We get out of traditional mathematics for this penultimate assignment. Like many of the assignments in this course, it represents a specific instance of another common programming paradigm: *backtracking*. Unlike prior assignments so far, much of the program has been written for you — you just have to write the most important part.

Code to Write

Complete the program, provided on the course Web site, that simulates a mouse looking for a piece of cheese within a maze. The program is invoked via the *edu.lmu.cs.maze.app.GUISolver* class, and takes, as command-line arguments, the initial location of the mouse (*mx*, *my*) and the location of the cheese (*cx*, *cy*), respectively:

java edu.lmu.cs.maze.app.GUISolver mx my cx cy

The description of the maze to use, formatted as described in class, is read from standard input (i.e., you type it in once the program starts). Thus, the maze can also be prepared in a text file, and that text file redirected into the maze solver via the "<" directive on the command line; for example:

java edu.lmu.cs.maze.app.GUISolver 4 8 9 2 < maze.txt

The portion of the program that you are to write is *edu.lmu.cs.maze.app.MazeWalker* — this should implement the backtracking algorithm described in class. A *MazeWalker* is initialized with a *Maze* object and the desired destination. The overall program is then structured so that it repeatedly invokes *areWeThereYet()* on the *MazeWalker*:

public WalkerState areWeThereYet(currentX, currentY)

areWeThereYet() should return one of six possible values, enumerated by *WalkerState*: one value indicates that the destination has been reached, another indicates that the destination is unreachable, and the rest indicate the next move.

The provided code throws an IllegalArgumentException whenever an inconsistent operation, such as placing the mouse/rat or cheese on a non-open or out-of-bounds square, is encountered.

For full details on how this class should work, consult your class notes, the JavaDoc API on the course Web site, and the source code itself.

What You Get

The rationale behind writing much of the program for you is to provide you with a visual environment that should make it easier for you to test and debug your backtracking algorithm. In addition, the task of modifying or enhancing (or completing!) someone else's code is something that you will likely encounter frequently in almost any collaborative programming milieu.

- Understanding how your code should interact with the rest of a program may be more difficult than it sounds — if you're confused, talk to me.
- Though not necessary to complete the assignment, do feel free to read through the provided code if you're interested in how it works. You might learn a thing or two **:)**
- Some ready-made maze descriptions are included for you; these are located in the *mazes* folder. Of course, you may also create your own — the format is easy enough to figure out.

A Design Note and Extra Work

- Note how *all* of your work should be confined to a single class — *edu.lmu.cs.maze.app.MazeWalker*. *None* of the other provided classes should need to be changed. If you think you have to make a change, then talk to me about it.
- The provided code is structured so that you can implement different user interfaces "on top of" the core functionality. After you have completed *MazeWalker* successfully, you might want to try your hand at *edu.lmu.cs.maze.app.TextSolver*, which should do the same thing as *GUISolver* except in a text environment. This involves writing code for displaying the maze and its status using "ASCII art," then writing a control structure that repeatedly calls *areWeThereYet()* to direct the mouse/rat to the cheese in the maze.