Assignment 8

Part 1: CarpenterBot

A robot named karel has been hired to carpet some rooms along a section of its world. A room is one block wide and may be of any height, but must have continuous walls on its west and east side and at its northern end. If any walls are missing, the area must not be carpeted. Also, karel must not reuse beepers. This means that once a beeper has been put down, it must not be picked up. (Start your robot off with an infinite number of beepers.) When the robot encounters a beeper on street 1, then the carpeting is

![Start and End Diagrams]

There are two sample worlds on the class web site to use in testing your program.

Part 2: SteeplechaseBot
Program a robot to run a steeplechase; that is, a hurdle race where the hurdles can be of differing heights. The race is over when the robot encounters a beeper.

(The robot’s course is shown in red in the example below.)

Test your solution on both steeplechase worlds provided on the class web page.
Part 3: SuperSteeplechaser

In a super-duper steeplechase, the hurdles can be arbitrarily high and arbitrarily wide. As in part 2, the end of the course is marked by a beeper. The best way to solve this problem, of course, is to create a subclass of the SteeplechaseBot class from part 2 (i.e., extend the SteeplechaseBot class).

One possible race course is shown below. There are others on the class web page.
Part 4: Harvester (Revisited)

The harvesting task has become more difficult. Now a field can be of any size and is bounded by walls (with an opening as shown below). It also has any finite number of beepers at each corner. Rewrite your HarvesterBot so that it extends SuperBot and can harvest any field bounded by walls.

Try to accomplish this as efficiently as possible! You may want to override one or more of SuperBot's methods.

There are test worlds such as this one on the web site:
Part 5: MazeWalker

Program a robot to escape from a maze that contains no islands. The exit of the maze is marked by a beeper on the first corner that is outside the maze, next to the right wall. This task can be accomplished by commanding the robot to move through the maze such that there is always a wall to his right side. Here is one such maze:

Before tackling this problem, read the next page!
Think about the following situations and what move(s) the robot should make to keep a wall on his right side:

Is it possible to write a move() method that takes all these possible situations into account and performs the correct action? If you had one, writing the MazeWalker would be trivial.