ECE 3090 Fall Semester, 2012

PROJECT 5 - ROBOT TRASH COLLECTOR

Assigned: Nov. 2, 2012 Due: Nov 16/21, 2012 – 1:05pm

No good job goes unpunished. With the success of your past robot programs, NASA has decided to send your robot to a much more exciting planet – a far away place called Earth. They've even given your robot a name – Robby, the Soda-Can-Collecting Robot. Robby has the job of cleaning up the Earth's environment by collecting empty soda cans. Various sites (i.e. grid cells) have been littered with soda cans (but with no more than one can per site). Robby isn't too intelligent, and his eyesight isn't that great. From wherever he currently is, he can see the contents of one adjacent site in the north, south, east, and west directions, as well as the contents of the site he's currently in. A site can be empty, contain a can, or be a wall (which surrounds Robby's world).

For each cleaning session, Robby is bounded by a maximum number of actions. Each action consists of one of the following seven choices: move north, move south, move east, move west, move in random direction, stay put, or pick up can. Each action may generate a reward or punishment. If Robby is in the same site as a can and picks it up, he gets a reward of 10 points. However, if he bends down to pick up a can at a site where there is no can, he is fined 1 point. If he crashes into a wall, he is fined 5 points and bounces back into the current site. Robby is also fined 1 point for each move action. Clearly, Robby's reward is maximized when he picks up as many cans as possible in the least number of moves, without crashing into any walls or bending down to pick up a can when no can is there. Your assignment is to write code for a learning algorithm (neural networks or genetic algorithms) that learns the optimal control strategy for Robby the Robot.

Input

Robby's world map will be stored in the file "input.dat" and is defined by the following parameters:

MAP width height

Defines the width and height of the world where (0,0) represents the top, left corner of the map ACTION maximum

Defines the maximum number of actions that the robot can implement

START x_position y_position

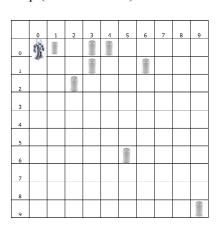
Defines the starting (x,y) position of the robot

CAN x_position y_position

Defines the (x,y) location of a can

Here is an example "input.dat" with a corresponding visual map (for illustration).

MAP 10 10
ACTION 25
START 0 0
CAN 10
CAN 3 0
CAN 4 0
CAN 3 1
CAN 6 1
CAN 2 2
CAN 5 6
CAN 9 9



Output

Given any *input.dat* consisting of a start location and set of cans, Robby should learn the necessary actions required to navigate the environment while collecting empty soda cans. For example, if Robby has learned the following control strategy:

Situation					Action
North	South	East	West	Current Site	
Empty	Empty	Empty	Empty	Empty	MoveNorth
Empty	Empty	Empty	Empty	Can	PickUpCan
Empty	Empty	Empty	Empty	Wall	MoveRandom
Empty	Empty	Empty	Can	Empty	MoveWest
1	:	:	:	:	
Wall	Empty	Can	Wall	Empty	PickUpCan
	:	:	:	:	
Wall	Wall	Wall	Wall	Wall	StayPut

Then, if Robby's current location corresponds to the situation:

North South East West Current Site
Wall Empty Can Wall Empty

the corresponding action would be: PickUpCan

Given the map environment, your program must output to "output.dat" - 1) the sequence of x, y grid points that Robby traverses, 2) the corresponding action at each of those grid points, 3) total number of actions, 4) total number of cans collected, and 5) the total score for the traversed sequence.

Grading Policy: Although there are no requirements on design of the algorithm, you must use classes for this program. The code for each class definition must be stored in a separate file (i.e. Class1.cpp, Class1.h). The project will be graded based on the following criteria.

Optimal credit (20 points for each case): Program outputs an optimal strategy based on collecting all of the possible cans in the minimum number of actions. Remember that your robot is bounded by a maximum number of actions. The program must run within 1.5 minutes.

Suboptimal credit (17 points for each case): Program outputs a sub-optimal strategy based on collecting all of the possible cans but not in the minimum number of actions. Remember that your robot is bounded by a maximum number of actions. The program must run within 1.5 minutes.

Partial credit (14 points for each case): Program outputs a sub-optimal strategy based on achieving a correct score (as discussed above) bounded by the maximum number of actions. If you believe your program will take longer than 1.5 minutes to run, you must indicate that via a message to the console.

Additional Deductions:

- Program will not compile (60 pts)
- Program crashes or does not terminate (20 pts)
- Program does not contain at least one separate class file (20 pts)
- Program does not comply with requirements or good design constraints (e.g. non-working makefile, no zip file, inadequate comments, use of global variables, etc.) (2 to 10 pts)

Submission: Your project code is to be submitted via the ECE3090 T-Square site under Assignments-Project5 on or before the due date. All relevant files (including a makefile to compile your code) should be zipped up and named by firstName_lastName_project5.zip. We will test your code on the Jinx cluster so make sure your program correctly compiles and runs on that system. Information on the cluster is located at: http://support.cc.gatech.edu/facilities/instructional-labs/jinx-cluster.

Extra Credit: Submission of your working code by the 19th of November is worth 10 points.