Program #2: Package World Agents

Due: Thursday, Oct. 21

For this assignment, you will design a team of homogeneous agents that will cooperate in order to achieve a simple package delivery task. The agents are situated in a 50x50 grid. The grid will have some number *p* of randomly located packages, each of which must be delivered to one of *d* locations. Note, there is a specific destination for each package. Your agents must be able to handle any number of packages, destinations, and team size. I have provided a simulator which you can use to test your agent teams.

The Agent Simulator

The simulator is similar to the one we used for the last project. This source code can be downloaded from our course web page (http://www.cse.lehigh.edu/~heflin/courses/agents-2004/). Everything is provided for you, except you must write the **PacAgent** class yourself. There is a skeleton of this class in the **pacworld** package. In particular you must implement the **see()** and **selectAction()** methods. You are not allowed to modify any simulator code other than **PacAgent**. In this assignment, the only direct communication that should occur between agents is through the use of the **Say** action. In particular, static variables are prohibited unless they are constants. When the simulation is started, it will place *n* copies of your agent at random locations in the environment. Note, that the **PacAgent** constructor provides a unique integer id to the agent.

The environment is essentially fully observable. That is, the agent knows the locations of all packages and agents. Furthermore, it knows where each package must be delivered. On each turn, each agent receives a **PacPercept** which has the following methods:

- **public VisibleAgent[] getVisAgents()** – Returns an array which contains the id and location information for each agent. See the **VisibleAgent** class for details on how to access this information.
- **public VisiblePackage[] getVisPackages()** – Returns an array which contains the id, current location, delivery destination, and status (is it held by an agent or not) for each remaining package. See the **VisiblePackage** class for details on how to access this information.
- **public String[] getMessages()** – Returns an array of messages broadcast by the agents since the perceiving agent's last turn. It is up to you to decide the format and content of the message.
- **public boolean feelBump()** – Returns true if the agent (or the package it was carrying) bumped into something on the last turn.

There are separate classes for each action available to the agents. The actions available to each agent are:

- **Drop** – Causes the agent to drop the package that it is holding in an adjacent square (the agent must specify a compass direction). If the package is dropped at its destination then it is successfully delivered and disappears. The package cannot be dropped on a location where there is an agent or another packages.
- **Idle** – Causes the agent to skip its turn.

- **Move** – Causes the agent to move one step north, south, east or west. If the agent is holding a package, the package will move in the same direction. If the agent or the package is blocked by an obstacle (another package or agent) then the agent will feel a bump and not move.
- **Pickup** – Causes the agent to pick up an adjacent package at the direction specified. The package will be held to that side of the agent until the agent drops the package. The agent can only pick up one package at a time.
- **Say** – Agent broadcasts a message to all other agents. The message is a string.

Note, some actions require a parameter for use.

The simulator takes four optional command line parameters: the number of agents, the number of packages, the number of destinations, and the size of the world (note, since we are fixing the world size at 50 for this environment, you do not need this parameter). If fewer parameters are provided, then defaults are used. In order to start the simulator with *n* agents, *p* packages, and *d* destinations, type **java pacword.PackageWorld *n p d*.** The simulator will launch a simple graphical user interface that allows you to step through each turn in the environment or to run it to completion in real time. In this user interface, agents are represented by black squares, packages by colored squares, and destinations by colored circles. Each package will be the same color as the destination it is to be delivered to.

Evaluation

The performance of the team will depend on the number of packages successfully delivered, the number of turns required to deliver these packages, the amount of communication needed, and the processor time used by your agents to make their decisions. See the **getPerformanceMeasure()** method in **PackageWorld** for details. Note, that communication and processor time are relatively cheap compared to the number of turns taken, so you should consider how thinking more intelligently and communicating can improve the coherence of your agents. In particular, you should consider the kinds of conflicts that can occur when two agents get in each other's way or choose to go after the same package. Note, that coordination may be more important in worlds that have more packages and agents. You should test your agents with a range of configurations (i.e., different team sizes, number of packages, and number of destinations). You will be graded both on the performance of your agents and on the quality of your design.

Submission

You must submit your assignment to me by the beginning of class on Thursday, October 21. There are hardcopy and electronic components to your submission. Your electronic submission must consist of the source code (.java files) and compiled (.class) files for **PacAgent** and any other supporting classes you developed. There should be a comment at the beginning of the file that identifies you and that provides a clear and detailed description of the strategy used by your agents. I also expect you to have a descriptive comment for each class and method, as well as comments to explain any complicated logic you might have. You may send this by e-mail to heflin@cse.lehigh.edu (please put "CSE 497-012 Project #2 Submission" in the subject line) or you can submit it on a 3.5" floppy disk. You must also hand in a hard copy of your source code.