Program #2: Package World Agents

Due: Thursday, Oct. 27

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-2016/). Everything is provided for you, except you must write a **PacAgent** class that extends agent.Agent and place it in a package named using your Lehigh user id. This class must have a one-arg constructor that takes an integer id and uses it to create a unique String id for the agent, i.e., the constructor signature is:

```
public PacAgent(int id);
```

Additionally, you must implement the **getId()**, **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 each **PacAgent** object will be created by invoking the one-arg constructor with a unique integer id for each agent.

Each agent can only see a limited view of the world. In particular, it can see all packages and agents inside a 11x11 square centered on itself. The percept includes the (x,y) coordinate of each seen object as well as the delivery locations for each visible package. Unfortunately, the environment is noisy, and the agent's percepts are not always accurate. 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 in sight range. 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 package in sight range. 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. Thus, every agent gets exactly one opportunity to hear everything that was said. This array only has as many elements as messages sent; if no agent sends a message, then the array will have length 0. It is up to you to decide the format and content of any messages sent.

- **public boolean feelBump()** – Returns true if the agent (or the package it was carrying) bumped into something on the last turn. The agent does not feel a bump if another agent bumps into it.
- **public VisiblePackage getHeldPackage() –** Return an object describing the package held by the agent. If the agent is not holding a package, returns null.

The percept is subject to the following three types of noise:

- There is a 5% chance per package seen that the agent will perceive the package's location to be in a square adjacent to its actual location. The agent will always perceive the locations of other agents, as well as any package it is holding, correctly.
- There is a 20% chance per package seen that the agent will perceive its destination incorrectly. If this occurs, the perceived destination will be a valid destination for other packages. The agent will always correctly perceive the destination of any package it is holding.
- There is a 2% per percept that agent will perceive a package that is not there. This package will be randomly located inside the agent's radius of visibility, but could be thought to be outside the bounds of the map or on the same location as another package or an agent.

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

- **Dropoff** – Causes the agent to drop the package that it is holding in an adjacent square (the agent must specify a compass direction; see the **Direction** class for predefined constants). 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 package, *even if this location is the package's destination*.
- **Idle** – Causes the agent to skip its turn.
- **Move** – Causes the agent to move one step north, south, east or west (see the **Direction** class for predefined constants). 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, so this will silently fail if the agent is already holding a package. The agent cannot pickup a package that is held by another agent.
- **Say** – Agent broadcasts a message to all other agents. The message is a string and will be heard by all agents up until the sender's next action. The hearers will not be provided the identity of the speaker, so if this is important, it must be encoded in the message. The exact format and content of the message is up to you, but beware that the performance measure penalizes messages based on their lengths, so communication should be used judiciously.

Note, most actions require a parameter.

In order to start the simulator with *n* agents defined in a package *userid*, *p* packages, and *d* destinations, type **java pacworld.PackageWorld *userid n p d*.** Generally, the simulator is run as follows:

```
java pacworld.PackageWorld [-rand seed] agentPackage [numAgents]
      [numPackages] [numDestinations] [worldSize]
```

The only mandatory parameter is the name of the package that contains your **PacAgent** class. The optional –rand switch allows you to specify a seed to generate a different random layout of packages and destinations. The other four optional command line parameters are 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 to use this parameter). If fewer parameters are provided, then defaults are used (i.e., numAgents = 4, numPackages = 20, numDestinations = 3). 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 **getTeamPerformanceMeasure()** 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 sharing information to build a global view of the environment and 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. You should also think about how the agent can have confidence in the location of packages, given that its percepts are subject to some noise. 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 Format:

You must submit a ZIP file of your *userid (e.g., xxx999)* directory that includes the source code (.java files) and compiled (.class) files for **PacAgent** and any other supporting classes you developed. Do not put the .java and .class files in separate subdirectories. Recall, all of your classes must be in the *userid* package or subpackage of it. I will unzip the contents of your submission into the same directory where my **agent** and **pacworld** directories are, and execute **java pacworld.PackageWorld *userid*** . If I have to do any additional configuration to make your program run, you will lose points! Upload your ZIP file to the Project #2 assignment on Course Site

Any source code you submit should be reasonably commented, including an initial comment that identifies you as the author, a descriptive comment for each class and method, and comments to explain any complicated logic you might have. In particular, you should have a comment that provides a clear and detailed description of the strategy used by your agents. There is no hardcopy submission required for this project.