Program #1: Vacuum Cleaning Agent

Due: Monday, Feb. 14 at the beginning of class

Your assignment is to write an intelligent agent for a variation of the vacuum cleaning world described in Chapter 3 of the textbook (p. 52). Like the robot in the book, your robot is capable of sucking up dirt, moving forward and turning (although our agent has the ability to turn left as well as right). It is also able to sense if it is over any dirt. However, we will use a more complex environment, in that the agent does not know it's initial location and there are obstacles that may block the agent. Fortunately, the robot has a sensor that can detect if an obstacle is directly in front of it. Another sensor can detect if the agent has accidentally bumped into an obstacle.

The robot's task is to make sure the entire room is clean. When this task is successfully completed, it should turn itself off (this can be done with the robot facing any direction it chooses). The agent should be intelligent in that it can perform this task efficiently in new situations. In other words, the robot does not know its location, nor the location of the obstacles or dirt ahead of time, but must make sure that the room is free of dirt using as little power as possible. This means it should try to minimize its movements and only turn on its suction device when necessary. The robot should also avoid bumping into things, because this can damage valuable personal articles and/or the robot itself.

The figure below shows an example initial state for the room. The robot is indicated by the "A", dirt is indicated by "*", and obstacles are indicated by an "X". The agent will be randomly placed in any square of the room, but will always start facing north. Note that the room itself is 5x5 and is completely surrounded by obstacles (walls). For the given example, there are also obstacles (e.g., furniture) in the upper right and lower left corners of the room.

```
  0  1  2  3  4  5  6
 +--+--+--+--+--+--+--+
0|X |X |X |X |X |X |X |
 +--+--+--+--+--+--+--+
1|X |  |  |  |  |X |X |
 +--+--+--+--+--+--+--+
2|X |  |* |  |* |X |X |
 +--+--+--+--+--+--+--+
3|X |  |  |A |  |X |X |
 +--+--+--+--+--+--+--+
4|X |X |  |  |  |  |X |
 +--+--+--+--+--+--+--+
5|X |X |X |  |* |  |X |
 +--+--+--+--+--+--+--+
6|X |X |X |X |X |X |X |
 +--+--+--+--+--+--+--+
Location: (1,1)  Facing: NORTH
```

Although the robot must be able to cope with a number of different situations, we will put some constraints on the task to make it feasible. First of all, the agent will only be used in 5x5 rooms that are completely surrounded by walls. Second, it will be able to reach any dirt in the room. Otherwise, dirt and obstacles may be in any open square.

In order to help you test your agent, I have written a Java Vacuum World simulation using the design we discussed in class. This source code can be downloaded from our CourseSite page. Everything is provided for you, except you must write the **VacAgent** class yourself. This class

must be placed in a package with the same name as your Lehigh account (e.g., xxx999) and must be a subclass of **Agent.** As such, it should implement the **see(), selectAction()** and **getId()** methods.

Before each move, the agent receives a **VacPercept** by way of its **see()** method. This percept contains information about whether the robot sees dirt below it (the **seeDirt()** method), sees an obstacle directly in front of it (the **seeObstacle()** method) and if it bumped into an obstacle on its last turn (the **feelBump()** method). All of these methods return boolean values. For details, see the **VacPercept.java** file.

When the robot's **selectAction()** method is called, it must return one of five actions. Each type of action is a subclass of Action. These classes are:

- **SuckDirt** – This action will remove dirt from the square the robot is in
- **GoForward** – This will move the robot forward one square in the direction it is facing. If it is facing an obstacle then the robot will not move, but will feel a bump in its next percept.
- **TurnLeft** – This will turn the robot 90 degrees to the left.
- **TurnRight** – This will turn the robot 90 degrees to the right.
- **ShutOff** – This will cause the robot to power down. The robot should only execute this action when it is certain it has cleaned the room of all dirt.

Note, you may only write code that controls your agent. It is against the principles of the assignment to try to give your agent access to state information not available from its percept history. It is also against the principles of the assignment to try to augment the agent with additional (or *superpowered*) actions. For this reason, you may not have any non-final static variables (i.e., class variables that are not constants) in your classes. Also, any changes you make to the simulator will not be used when I test your agents. Therefore, the only appropriate changes to the simulator are those needed to test and debug your agent. On the other hand, you may create additional classes that support your **VacAgent**, and if you create such classes, they should be in the same package as it (the one named with your Lehigh account).

After you have compiled your **VacAgent** file, you can start the simulator by typing "`java vacworld.VacuumWorld xxx999`" (assuming you are in the directory directly above `vacworld`, have "." in your Java `CLASSPATH` environment variable, have put your class in a package named with your Lehigh account, and this id is xxx999). The simulator outputs a map of the state of the world to the console. To advance to the next state, simply press ENTER. The percept received by the agent and the action that it selects will be printed, along with the resulting map.

In order to help you test your agent, the simulator also takes additional command line arguments. The full syntax is "`java vacworld.VacuumWorld [-batch] [-rand` *seed*`]` *agentpack*" where *seed* is an integer. The **–batch** option turns off the pauses between each move. This is useful if you simply want to see what score your agent would get in a particular environment, and don't care about the sequence of moves. The **–rand** option will cause the simulator to generate a pseudo-random state, using the seed you provide for the random number generator. Every time you provide the same seed, you will get the same state. This is useful if you wish to test whether a change to your agent improves its behavior in a particular environment. It is also possible to create a driver class that creates a new **VacuumWorld** object

and provides it with an initial state of your own design. See the **VacuumState** class for details on how to create states. To start the simulation, call **VacuumWorld.start().**

I will test your agent with multiple room and dirt configurations, one of which will be the one shown in this assignment. Your grade will depend significantly on the performance measure your agent receives when performing the task in each of these different initial states. Therefore, it is *imperative that you test your agent under a variety of situations*. You may even want to manually create some particularly difficult initial states and test your agent on them. The performance measure depends mostly on the ability to complete the task (that is, clean the room and shut off), but also takes into account how efficiently the agent accomplishes its tasks. The performance measure of a run *x* is calculated as:

$$U(x) = 1000 - 100 * dirt(x) - 10 * bumps(x) - 2 * moves(x) - turns(x) - 2 * sucks(x)$$

where, *dirt(x)* is the number of dirt locations left at the end of the run, *bumps(x)* is the number of times during the run that the agent bumped into a wall, *moves(x)* is the total number of forwards executed in the run, *turns(x)* is the number of turns executed in the run, and *sucks(x)* is the number of times the agent tried to suck up dirt in the run. See the **getPerformanceMeasure()** method in **VacuumWorld** for details. Note that you don't have to have an optimal solution to get an A on this assignment, but a near optimal solution would help. Generally speaking, running time will not be considered, but if your agent takes more than a few seconds per step to choose its action, you will be penalized. Finally, the elegance of your design will be considered in your grade as well. However, this design need not be based on an architecture described in the book; it could be something you developed yourself. Note, when possible, make your code modular. A single method consisting of hundreds of lines with deeply nested if/then statements is inelegant, hard to debug, and hard to understand.

Submission Format:

You will submit your program via CouseSite. Your code must be placed in a package named using your Lehigh account (e.g., xxx999). Upload a ZIP file of your *userid (e.g., xxx999)* directory that includes the source code (.java files) for **VacAgent** and any other supporting classes you developed (note, all such classes must be placed in the same package as your VacAgent).

All 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 describes the basic strategy that your agent uses.