**Artificial Intelligence**

**Programming Exercise: Vacuum cleaner world**

This activity considers a variation of the vacuum cleaner world problem. The textbook considers only two "cells" in the vacuum cleaner world, we will consider variable sized worlds. You will develop an agent(s) for this world. A performance measure is computed for your agent. The score is a penalty score (so the lower the score, the better the agent did). Two files are provided for your use:

* EnvironmentInterface.java
* Environment.java

The Environment.java class implements the methods defined in EnvironmentInterface, shown below:

public interface EnvironmentInterface {

public int moveLeft();

public int moveRight();

public int moveUp();

public int moveDown();

public void suck(); // clean current cell

public boolean isDirty(); // is current cell dirty?

public int getScore(); // returns penalty score

}

There are two constructors to create an Environment object. The signatures are as follows:

* public Environment(int numRows, int numCols, int nDirty)
* public Environment(int envNum)

The constructor with three parameters can be used to create a vacuum cleaner world environment with numRows rows, numCols columns, and nDirty dirty cells (if nDirty is more than the number of "cells" in the world, all cells are set to dirty). The agent is always placed in this world at row 0, column 0. Note that valid rows are 0..numRows-1 and valid columns are 0..numCols-1.

The constructor with a single parameter can be used to construct a predefined (rectangular) vacuum cleaner world with "hidden" dimension. Also, the number of dirty cells is unknown (hidden). One difference here is that the agent begins at a "random" location in the world. The sample Environment class comes with five predefined vacuum cleaner worlds (parameter value 0, 1, 2, 3, or 4, default is 0). If you look at the code, you can see the default worlds. Do **NOT** assume these values will be used to grade your program/agent. They are provided for testing convenience.

The Environment methods provided (as defined in the interface) include methods to move the agent around in the world, moveLeft, moveRight, moveUp and moveDown. If the move is invalid (bump against edge of world), a 1 is returned; otherwise, a 0 is returned and the agent moves to the intended, neighboring cell. Every move, whether valid or not, incurs a (small) penalty.

The method isDirty can be used to determine whether or not the cell currently occupied by the agent is dirty or clean. There is no penalty for checking the status of the "dirt sensor". The suck method can be used to clean the cell currently occupied by the agent. This operation has a fairly significant penalty, so you should always check the status of the location before using the suck method. Once the task is complete you should obtain your agent's score using the getScore method and display it. Use the getScore method only once, since using it repeatedly incurs a very large penalty.

You are to build a model-based agent to clean vacuum cleaner worlds created using the three-parameter constructor. Your rule set should be ordered and implemented as a "nested if" statement. You can keep track of other local information to help your agent, e.g. direction it is heading. Test your agent on a variety of different sized worlds with a range of dirty cells. Be sure to use both odd and even number of rows and columns. You can hardcode the worlds you define, similar to the definition of worlds used in the one-parameter constructor of the Environment class.

Do not make changes to the Environment class (other than possibly making more predefined worlds to test with).

**Due date: November 10, 2013 Monday**