|  |
| --- |
|  |
| Vacuum Cleaner Environment Programmer’s Guide |
| Version 3.0 |

|  |
| --- |
| Mark A. Cohen, Ph.D.  8/17/2010 |

Contents

[Introduction 3](#_Toc269825298)

[Requirements 3](#_Toc269825299)

[Running the Environment 3](#_Toc269825300)

[Loading an Agent 4](#_Toc269825301)

[Time Steps and Operations 4](#_Toc269825302)

[Sensors 5](#_Toc269825303)

[Agent Performance 6](#_Toc269825304)

[The Environment Options 7](#_Toc269825305)

[The Configure Run Option 7](#_Toc269825306)

[The Configure Board Option 7](#_Toc269825307)

[Agent Type Options 7](#_Toc269825308)

[Radar Sensor Option 7](#_Toc269825309)

[Penalize for Movement Option 8](#_Toc269825310)

[The Load Board Option 8](#_Toc269825311)

[Sensor Fact Representation 9](#_Toc269825312)

[Sensor Facts in Jess 9](#_Toc269825313)

[Sensor Facts in Soar 11](#_Toc269825314)

[Sensor Facts in Java 12](#_Toc269825315)

[Taking Action 13](#_Toc269825316)

[Taking Action in Jess 13](#_Toc269825317)

[Taking Action in Soar 14](#_Toc269825318)

[Taking Action in Java 14](#_Toc269825319)

[Agent Templates 15](#_Toc269825320)

[Example Agents 16](#_Toc269825321)

[Works Cited 17](#_Toc269825322)

# Introduction

The Vacuum Cleaner Environment (Cohen, 2005) is a more complicated version of an environment introduced by Russell and Norvig (2003) to explain intelligent agents and agent environments. The purpose of this environment is to provide an engaging atmosphere for learning various topics in computer science, such as procedural and declarative programming. Using this tool, students can create agents using Java, Jess (jessrules.com), or Soar (sitemaker.umich.edu/soar), and watch them interact.

# Requirements

To use the agent environment you will need the following:

1. The Java runtime environment version 1.5 or higher
2. The vacuum jar file
3. A good text editor

NOTE: If you plan to build agents using Jess or Soar, you will also need Jess version 6.x and/or Soar version 8.6.1.

# Running the Environment

Assuming you have installed all the software listed above, you should be able to run the agent environment.  To run the environment on Linux or Windows:

1. Create a new directory and download the vacuum jar file into this directory
2. Double click on the vacuum jar file or open a terminal window and type the following command: *java -jar vacuum\_3.0.jar*

If you plan to build agents using Jess or Soar, make sure the jess jar file and/or the Soar/Java bridge is in the same folder as the vacuum jar file.

If you plan to build Java agents, be sure to include the vacuum jar file in your class path when you compile your Java agent. In addition, the class file for the agent you create must be in the same folder as the vacuum jar file.

When you run the environment, you will be presented with the main agent environment window shown in .

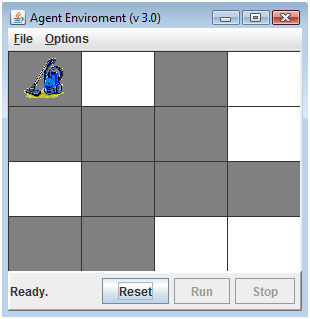


Figure : The Vacuum Cleaner Environment

# Loading an Agent

Before an agent can be executed, the file containing the rules that drive the agent must be loaded. Rules can only be coded using Java, Jess, or Soar.  To load your agent rules select the File menu and choose the type of agent you want to load.

If you are loading a Java agent, you will need to type in the fully qualified class name of the Java class you want to load. The classfile for the agent you specify must be located in the applications current classpath. If you are loading a Jess or Soar agent, a file chooser dialog will be displayed and you can browse to the Jess or Soar file that you would like to open.

Once an agent has been loaded, the Run and Reset buttons will be enabled and the name of the agent will be displayed in the title bar. You can start the agent on its journey by clicking on the Run button. You can pause the agent by clicking on the Stop button, and start it again with the Run button.  Finally, the Reset button will generate a new random board configuration and place the agent in a random location.

# Time Steps and Operations

Each time you start your agent in motion, it will be given an opportunity to perform a single operation for each time step in the run. By default, each run consists of 100 times steps, and there will be 500 milliseconds between each time step. While your agent is running, the current time step will be displayed in the status bar.

Your agent is only capable of performing one of the following five operations for each time step: left, right, up, down, and suck.

# Sensors

After each time step, your agent's sensors will evaluate the environment and place facts pertaining to the environment into the agent’s memory. The facts placed in memory are dependent on what sensors you are using. By default, the sensor your agent is equipped with can only detect the current coordinates of your agent, and the dirty or clean status of the agent’s current location.

By default, your agent is also equipped with a more advanced radar sensor. This sensor can be disabled using the Options menu. The radar sensor will report the status (clean or dirty) of the four squares surrounding your agent, as well as detect the stink of rotting trash within three squares of the agent in all four directions. The closer the trash, the more it will stink. Trash three squares away will have a stink value of 1; trash two squares away will have a stink value of 2, and trash next to the agent has a stink value of 3. The stink values for two different board configurations are shown in .

Table : Stink values for two different board configurations

|  |  |
| --- | --- |
| Board 1 | Board 2 |
|  |  |
| stink left = 0 + 0 + 0  stink right = 3 + 0 + 0  stink up = 3 + 0 + 0  stink down = 3 + 2 + 0 | stink left = 0 + 2 + 0  stink right = 0 + 0 + 0  stink up = 0 + 2 + 1  stink down = 3 + 0 + 0 |

If you ran the Vacuum Cleaner Environment using a terminal, the sensor facts that are available to your agent after each time step will be displayed in the terminal window. In addition, any information you print from your Java, Jess, or Soar code will also be displayed in the terminal window while your agent is running.

# Agent Performance

At the end of each time step, your agent will be evaluated on its performance and its total score will be updated accordingly. The score is meant to give you a general idea of how well your agent is performing its cleaning duties. You agent's score is calculated by adding one point to the total for each clean square, at the end of each time step. By default, your agent will also be penalized one point for each move your agent makes. The movement penalty can be disabled using the Options menu.

While your agent is running, its current score will be displayed in the status bar. When the run completes, the final score will be displayed in a message box.

# The Environment Options

Depending on the problem you are being asked to solve, you may need to change the default settings for the environment. Many of the environment’s properties can be configured using the Options menu, shown in Figure 2.

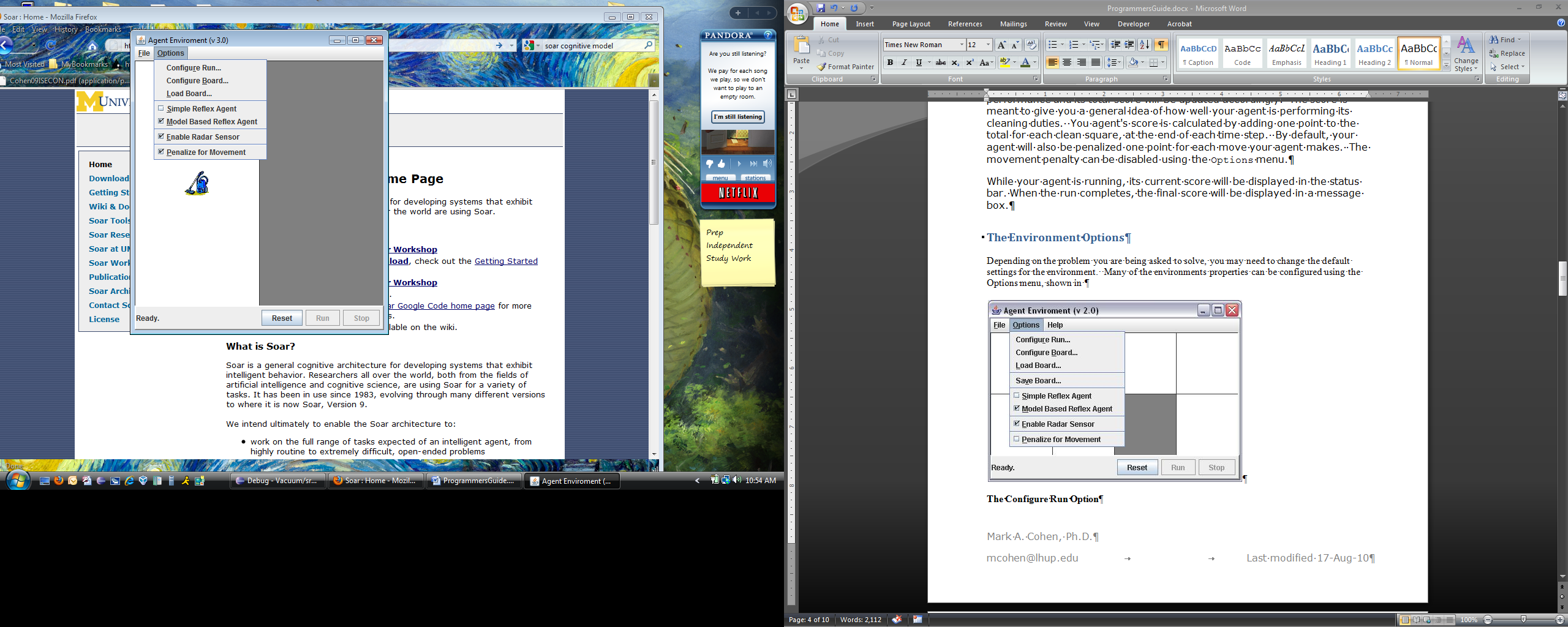


Figure : The Options Menu

## The Configure Run Option

The Configure Run menu item allows you to change the number of time steps that your agent will be allowed to run around in the environment, as well as the time between time steps (in milliseconds). You can slow down, speed up, or give your agent more time steps to play with using this menu item.

## The Configure Board Option

The Configure Board menu item allows you to change the size of the board and the number of dirty squares. This is the place to go when you want your environment to have a particular board size and dirty square density.

## Agent Type Options

There are two types of agents allowed in the environment: simple reflex agents, and model-based agents (the default). The simple reflex agent is not allowed to remember anything between time steps. The model-based agent is allowed to assert facts and remember these facts during an entire run.

All Java agents are model-based agents. As a result, this option is ignored if you are running a Java agent.

## Radar Sensor Option

By default, the agent is equipped with an advanced radar sensor that can detect the current coordinates of your agent, the dirty or clean status of the square it is located on, the status of the four squares surrounding the agent, and the stink coming from all four directions. If you want your agent to have only primitive sensor capabilities, you can disable the radar sensor using this option. If the radar sensor option is disabled, agents can only detect their current position and the clean status of the square it is located on.

## Penalize for Movement Option

Checking this option will punish your agent each time it makes a move. More specifically, your agent's score will be decreased by one point each time it moves. By default, this option is enabled.

## The Load Board Option

The Load Board menu item makes it possible to create an exact board configuration from a simple text file. If you select this option, you can browse to a board definition file and use that file to generate an environment. A board definition file is a plain text file that uses the following format: the - character represents a clean square, the \* character represents a dirty square, the v character represents a vacuum cleaner agent located on a clean square, and finally an x character represents a vacuum cleaner agent located on a dirty square. Table 2 shows some example board files along with the generated vacuum cleaner boards.

Table : Sample board files along with the generated vacuum cleaner boards

|  |  |
| --- | --- |
| Board File | Generated Board |
| \* \* \* \* - - - - - - v - \* \* \* \* |  |
| \* - \* - \* x - - |  |

# Sensor Fact Representation

Sensor facts are represented differently depending on the type of agent you are creating. The following sections describe how sensor facts are represented in Jess, Soar, and Java.

## Sensor Facts in Jess

In Jess, sensor facts are structured using the following unordered fact templates:

(deftemplate vacuum.types.position  
 (slot x (default 0))  
 (slot y (default 0))  
)

(deftemplate vacuum.types.spot  
 (slot status (default ""))  
)

(deftemplate vacuum.types.radar  
 (slot dir (default ""))  
 (slot reading (default ""))  
)

(deftemplate vacuum.types.stink  
 (slot dir (default ""))  
 (slot reading (default 0))  
)

As your agent runs, facts based on these templates will be asserted in the Jess knowledge base. Table 3 shows an example of the facts in the Jess knowledge base for a given board configuration.

Table : Sample Jess facts for a given board configuration

|  |
| --- |
| Board |
|  |
| (MAIN::initial-fact)  (MAIN::vacuum.types.spot (status "clean"))  (MAIN::vacuum.types.position (x 2) (y 1))  (MAIN::vacuum.types.radar (dir "right") (reading "clean"))  (MAIN::vacuum.types.radar (dir "left") (reading "clean"))  (MAIN::vacuum.types.radar (dir "up") (reading "clean"))  (MAIN::vacuum.types.radar (dir "down") (reading "clean"))  (MAIN::vacuum.types.stink (dir "right") (reading 0))  (MAIN::vacuum.types.stink (dir "left") (reading 0))  (MAIN::vacuum.types.stink (dir "up") (reading 0))  (MAIN::vacuum.types.stink (dir "down") (reading 2)) |

## Sensor Facts in Soar

In Soar, the sensor facts are asserted as working memory elements on the input link. As your agent runs, facts based on these templates will be asserted in the Jess knowledge base. Table 4 shows an example of the facts in Soar working memory for a given board configuration.

Table : Sample Soar facts for a given board configuration

|  |
| --- |
| Board |
|  |
| (S1 ^io I1 ^name |Testing-problemspaces-topspace| ^operator O2  ^operator O2 + ^parent S1 ^superstate nil ^top S1 ^type state)  (I1 ^input-link I2 ^output-link I3)  (I3 ^|vacuum.types.action| V11)  (V11 ^move right)  (I2 ^|vacuum.types.position| V2 ^|vacuum.types.radar| V3  ^|vacuum.types.radar| V4 ^|vacuum.types.radar| V6  ^|vacuum.types.radar| V5 ^|vacuum.types.spot| V1  ^|vacuum.types.stink| V8 ^|vacuum.types.stink| V9  ^|vacuum.types.stink| V10 ^|vacuum.types.stink| V7)  (V10 ^dir down ^reading 2)  (V9 ^dir up ^reading 0)  (V8 ^dir right ^reading 0)  (V7 ^dir left ^reading 0)  (V6 ^dir down ^reading clean)  (V5 ^dir up ^reading clean)  (V4 ^dir right ^reading clean)  (V3 ^dir left ^reading clean)  (V2 ^x 2 ^y 1)  (V1 ^status clean)  (O2 ^name |impasse\*Testing-problemspaces-pursueps|) |

## Sensor Facts in Java

All Java agents must extend the BaseJavaAgent class. As a result, Java agents have access to sensor values by calling methods. Table 5 describes the methods made available by the BaseJavaAgent.

Table : Sensor methods available to all Java agents

|  |  |
| --- | --- |
| Method | Description |
| int getX(); | Returns the current horizontal position of the agent (zero-based). |
| int gety(); | Returns the current vertical position of the agent (zero-based). |
| boolean isCurrentSquareDirty(); | Returns true if the agent is currently on a dirty square. |
| String radar(String dir); | Returns the status of a square next to the agent. The parameter dir can be “LEFT”, “RIGHT”, “UP”, or “DOWN”. This method will return either “CLEAN” or “DIRTY”. |
| int stink(String dir); | Returns the stink of rotting trash within three squares of the agent in a particular direction. The parameter dir can be “LEFT”, “RIGHT”, “UP”, or “DOWN”. This method will return an integer either “CLEAN” or “DIRTY”.  The closer the trash, the more it will stink. Trash three squares away will have a stink value of 1; trash two squares away will have a stink value of 2, and trash next to the agent has a stink value of 3.  For example, the stink emanating from the right side of an agent that has dirt in each of the three square to its right will be equal to 6. |

# Taking Action

Vacuum cleaner agents take action differently depending on the type of agent you are creating. The following sections describe how agents interact with the environment in Jess, Soar, and Java.

## Taking Action in Jess

To make a move or clean the current square from within a Jess agent you need to add an unordered fact to working memory.  This unordered fact template should be declared like this:

(deftemplate vacuum.types.action  
 (slot move (default ""))  
)

The value of the move slot can have one of five values suck, left, right, up, or down. For example, to move your agent to the right, assert the following fact:

 (assert (vacuum.types.action (move "right") ))

To command your agent to clean the current square, assert the following fact:

 (assert (vacuum.types.action (move "suck") ))

## Taking Action in Soar

In Soar, commands are added to the output link in working memory. The move working memory element can have one of five values suck, left, right, up, or down. For example, to move your agent to the right, add the following working memory element:

(<o1> ^|vacuum.types.action| <vacuum-types-action>)  
(<vacuum-types-action> ^move |right| )

To command your agent to clean the current square, add the following working memory element:

(<o1> ^|vacuum.types.action| <vacuum-types-action>)  
(<vacuum-types-action> ^move |suck| )

## Taking Action in Java

All Java agents must extend the BaseJavaAgent class. As a result, Java agents can interact with their environment by calling methods. Table 6 describes the methods made available by the BaseJavaAgent that will allow the agent to move around, clean up, and more.

Table : Methods that allow a Java agent to move, clean, and more

|  |  |
| --- | --- |
| Method | Description |
| void cleanCurrentSquare(); | Causes the agent to clean the current square. |
| void moveUp(); | Causes the agent to move up one square. |
| void moveDown(); | Causes the agent to move up down square. |
| void moveLeft(); | Causes the agent to move up left square. |
| void moveRight(); | Causes the agent to move up right square. |
| void dumpState(PrintStream out); | Prints the current sensor values to the specified PrintStream, which is typically System.out. |
| int pickRandomNumber(int from, int to); | Returns a random integer within the specified range. The range is inclusive. |

## Agent Templates

To help you get started, agent templates are provided for Jess (Table 7) and Java (Table 8) based agents. Just copy the template into your editor and add your code to the section labeled “insert your code here...”

Table : A simple template to get you started with a Jess agent

|  |
| --- |
| (reset)  (deftemplate vacuum.types.action  (slot move (default ""))  )  (deftemplate vacuum.types.spot  (slot status (default ""))  )  (deftemplate vacuum.types.position  (slot x (default 0))  (slot y (default 0))  )  (deftemplate vacuum.types.radar  (slot dir (default ""))  (slot reading (default ""))  )  (deftemplate vacuum.types.stink  (slot dir (default ""))  (slot reading (default 0))  )  ;insert your code here... |

Table : A simple template to get you started with a Java agent

|  |
| --- |
| import edu.lhup.vacuum.BaseJavaAgent;  import edu.lhup.vacuum.CheatException;  public class DefaultJavaAgent extends BaseJavaAgent  {  public DefaultJavaAgent()  {  }  protected void go()  {  // insert your code here...  }  } |

# Example Agents

Because this environment is used for class assignments, examples that are more complicated must be requested via email, and will only be provided to professors.

# Works Cited

Cohen, M. A. (2005). Teaching agent programming using custom environments and Jess. *The Newsletter of the Society for the Study of Artificial Intelligence and the Simulation of Behavior* *, 120*, p. 4.

Russell, S., & Norvig, P. (2003). *Artificial Intelligence: A modern approach.* Supper Saddle River, NJ: Prentice Hall.