6.871 Assignment 1: Minesweeper

February 28, 2005

DUE: March 10, 2005, 9:35AM

1 The Problem

This is a knowledge engineering task, i.e., a task involving writing down the knowledge needed to be good at something. In this case the something is the Minesweeper game from Windows.

Your job is to create the best set of rules to play minesweeper that you can. We will run all the rule sets against a challenging set of examples and see if we can crown one player the class champion.

In doing this assignment you should be sensitive to what it means to specify the knowledge needed to be good at the game. That is,

- be aware of how writing rules differs from writing a traditional program, what we referred to in the lectures as "telling it what to know, not what to do."
- be aware of what goes through your mind as you figure out the rules to begin with, i.e., what the thought processes are in figuring out what you need to know and in stating it explicitly. (Simply put, try to tune in to the process of knowledge engineering.)

We have created SmartSweeper, an MSPDE – MineSweeper Player Developer Environment. It will enable you to write, test, and debug a set of rules for the game, using a syntax we call S4. You can see how well you do, and can improve your rules until you're happy with them.

2 To Begin

Play several games of minesweeper. Even if you're played it before, play it again, but try to be very aware of the patterns you are reacting to and the reasoning you are doing. If you've never played it before, you may actually have an advantage, as it's easier to be conscious of how you're thinking about the game when it's still unfamiliar.

3 S4: The SmartSweeper Specification Syntax

An S4 file contains a series of rules. A rule has a *name* (for your convenience), a *left hand side* (or condition), and a *right hand side* (the action to take if the left hand side is satisfied).

```
{name
LHS
=>
RHS
}
```

The placement of the curly brackets with respect to line breaks and the use of the "arrow" (the two character sequence `=>`) are both crucial – the parser reading this file is not robust.

A file of rules is simply one rule after another in the format shown above.

3.1 The Left-Hand Side

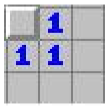
A left hand side in turn consists of a *pattern* followed by zero or more *conditions*.

```
LHS ::=  pattern
         condition*
```

A pattern is an n-by-n array of characters describing a subsection of the board. Legal characters are:

M	A labeled *mine*
0-9	Any *number*
-	Any square known *not* to be a mine
?	A square whose contents are *unknown*
*	Wildcard, matches anything
a-z	A *variable*, matches any number.
|	A spot over the edge of the board (see below for example and further explanation).

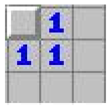
For example, in the figure below, the top left square is unknown, the three squares adjacent to it contain numbers, while the remaining five squares are known to be clear. Note that all squares except the one at top left would match a hyphen, as they are known not to be mines.



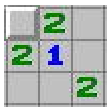
Now consider pattern1:

```
?--
-1-
---
```

This pattern matches whenever we find a 3x3 group of tiles in which the middle tile is a 1, and there's only 1 unknown tile, to the upper left. It will for example match the segment of the board shown previously:



It will also match this pattern anywhere on the board:



Any number of variables can be used, and they can be referred to in the *conditions*. A condition is any (Boolean) scheme expression that must hold for the LHS to match.

In addition to the variables you used in the LHS pattern, there are several variables whose values are maintained for you by the system:

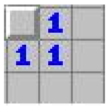
`matched-unkns`	The total number of unknown tiles matched by all wildcards in the pattern
`matched-mines`	The total number of marked mines matched by all wildcards in the pattern
`total-mines-remaining`	The number of unmarked mines remaining anywhere on the board
`total-non-mines-remaining`	The number of unmarked non-mine squares remaining on the board

Just so it's clear: at any moment, the total number of unknown squares on the board = `total-mines-remaining + total-non-mines-remaining`.

Here is an example of a more complex LHS, an example that includes a pattern and two conditions:

```
***
*n*
***
(= matched-unkns n)
(= matched-mines 0)
```

This pattern matches any time there is a numbered tile, bordering a set of wildcards that do not match any mines (because we specified `matched-mines`=0), and surrounded by a set of unknown squares equal to the number in the central tile. This applies to the example we saw before, and many others:



The | symbol matches tiles that are over the edge of the board. For example, consider LHS 3.

```
|||
|n*
|**
(= matched-unkns n)
(= matched-mines 0)
```

This pattern will match any time there is a number in the far upper-left corner of the board that matches the number of adjacent unknowns. Note that wildcards can also be used to match tiles over the edge of the board.

3.2 The Right-Hand Side

On the right-hand-side (RHS), you specify what action to take if the LHS matches. This can be expressed in the form of another pattern, and/or a pre-defined action:

```
  RHS            ::=   PATTERN        | 'no-action-pattern'
                       DEFAULT-ACTION | 'no-default-action'
  DEFAULT-ACTION ::=  'mark-unknowns' | 'click-unknowns'
```

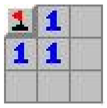
For example, consider the LHS that we saw before:

```
?--
-1-
---
```

A reasonable decision would be to mark the unknown tile as a mine, as expressed in this pattern.

```
M--
-1-
---
```

This marks the square as a mine. The outcome is shown here:



Putting together the LHS and the RHS, we have Rule 1:

```
{Rule1
?--
-1-
---
=>
M--
-1-
---
no-default-action
}
```

The final line specifies that no default action will be undertaken. The default actions refer to all tiles matched by a wildcard character * and offer a simple way to say that you want to mark all such tiles as mines, or click on them all. To illustrate a rule with default actions, consider our Rule 2:

```
{Rule2
***
*n*
***
(= matched-unkns n)
(= matched-mines 0)
=>
no-action-pattern
mark-unknowns
}
```

The rule indicates that if a number n is surrounded by n unknown squares and no mines, then all of the unknowns must be mines.

This rule has no action pattern, but executes the default action *mark unknowns*, which labels all of the unknown squares as mines. Make sure you understand why this rule works (keep in mind the definition of an “unknown” square.)

If you have a different situation in which you believe all unknowns are clear, you can use `click-unknowns` to click on all of them.

Sometimes you will want to click on one or more specific squares because you believe them to be clear; to do this place `X`s in the RHS pattern where you want to click.

3.3 Rules With Ineffective RHSs

If your rule action does not change the board in some way, the program will enter an infinite loop: It will see that the left side matches, then try to apply the action. When it scans the board again, it will once again see that the left side matches, since nothing has changed. You can use the debug flags described below to identify and correct infinite loops.

4 The SmartSweeper Rule Interpreter

SmartSweeper iterates through your list of rules and tries to match each rule everywhere on board (starting at the upper-left, but that's not particularly important). If the LHS ever matches, it applies the RHS, then returns to the top of the list of rules and the upper-left corner of the board. If none of the rules match, SmartSweeper will "guess" a square that we guarantee is not a mine. When all non-mine squares have been clicked on, the game is over. SmartSweeper always traverses the board top to bottom, then left to right.

We give you a free pass on guesses in order that your system can just keep trying. Note that as a result even if you write no rules at all, SmartSweeper will eventually guess all the non-mine tiles and you will win. As long as you don't write any rules that could click on a mine, there is no way to lose the game by hitting a mine. However, you will be scored on how few guesses your system uses (after all, the whole idea of a knowledge-based system is to know, not guess.)

4.1 Scoring

Your system's score for a given board is computed as follows:

- If you successfully complete the board in the time allotted:

$$\begin{aligned} C &= \text{Number of non-mine squares clicked} && (1)\\ G &= \text{Number of guesses} && (2)\\ T &= \text{Fraction of time allotted that remains after you finish} && (3)\\ \textit{score} &= C - G + T && (4) \end{aligned}$$

- If click on a mine, or do not successfully complete the board in the time allotted:

$$\begin{aligned} R &= \text{Number of non-mine squares remaining} && (5)\\ G &= \text{Number of guesses} && (6)\\ \textit{score} &= -R - G && (7) \end{aligned}$$

Since T is a fraction guaranteed to be less than one, the time complexity of two rulesets will only come into play when the rulesets use exactly the same number of guesses. However, you may still want to think about how to make your rulesets faster. The allotted time for each board will be thirty seconds, which should be more than enough for any reasonable ruleset on the boards that we will be using.

Here is an example of the output of the PatternEval program:

```
639.9612 pts, 3-0-0; Avg Guesses:  12.666666 Avg Time:  388.0
```

This says that a total score of 639.96 was awarded; 3-0-0 means that all three boards were completed successfully, with zero losses and zero time-outs. The average number of guesses per board is also indicated, as is the average time taken.

5 Running SmartSweeper

You will need a machine with a working Java Virtual Machine to run SmartSweeper. The JVM is available at `java.sun.com`. Then you should get smartsweeper.jar and jscheme.jar (from the assignment web page) and put them in your classpath:

- If you are on a Windows machine:

  Open a DOS window and go to the directory where you put the two files named above. Set the classpath with this command:

  ```
  set classpath=%classpath%;smartsweeper.jar;jscheme.jar
  ```

  This command will add the two new files to your existing classpath. Note that you have to type the command above exactly as given. You can check to be sure the classpath has been set correctly using this command:

  ```
  echo %classpath%
  ```

  The result should be a line containing the string

  `smartsweeper.jar;jscheme.jar`.

- If you are on the server:

  ```
  setenv CLASSPATH ‘‘smartsweeper.jar:jscheme.jar:$CLASSPATH’’
  ```

On either machine, a quick way to check that all is well is to run this command, which should print out a 15x15 board with 10% of the tiles marked as mines:

```
java edu.mit.smartsweeper.Board 15 15 .1
```

If you get a Java error message saying something like

```
Exception in thread main java.lang.NoClassDefFoundError:
edu/mit/smartsweeper/Board
```

it's likely your classpath has not been set correctly. Feel free to check with the TA.

5.1 Evaluating Your Rules

Use this command to run your rule set:

```
java edu.mit.smartsweeper.PatternEval [RULES] [DEBUG-LEVEL] [BOARD]
```

RULES specifies the file containing your rules.

DEBUG-LEVEL is an integer from 0 to 6 showing the level of debugging help you want.

- 0 Simply prints your final score
- 1 Prints the score on each board
- 2 Prints the rule usage statistics. Find out which rules are firing.
- 3 Prints the board every time none of your rules apply. You can use this to increase the coverage of your rules.
- 4 Prints each rule that fires. This will help you identify rules that cause infinite loops.
- 5 Shows the board every time a rule fires.
- 6 Shows each rule it tries, including why the rule fails to match.

BOARD is optional. It specifies a file containing a single board, or a directory containing a set of boards. If this argument is unspecified, it applies the rules to a single, randomly generated board of dimensions 30x30, with a 25% chance that each tile is a mine.

5.2 Printing Boards

You can print your own boards to evaluate your rules. PatternEval will evaluate your rules using a randomly generated board, but you can compare the values of different rulesets more accurately by using a set of standard boards. In fact, this is how the contest will work. To print a board, run the following:

```
java edu.mit.smartsweeper.Board [HEIGHT] [WIDTH] [MINE_PCT]
```

The arguments should be obvious. You can send the output to a file, and then use that file as an argument to PatternEval:

```
java edu.mit.smartsweeper.Board 15 15 .1 > testboard1.txt

java edu.mit.smartsweeper.PatternEval myruleset.txt 3 testboard1.txt
```

Note that boards larger than 60x60 may take too long for many rulesets. PatternEval times out after 30 seconds, to avoid infinite loops. You will not be evaluated on any boards larger than 30x30.

6 What To Turn In

A) Your rules: When you think you've got the best set of rules you can create, email the TA the file containing your rules. We will crown the best ruleset as champion, with appropriate fanfare and a prize for the author.

B) One page analysis. One of the points we made in an earlier lecture was that having the right language is one key to capturing knowledge. What is your evaluation of the language we gave you for writing Minesweeper rules? That is, what concepts in the language seemed powerful and useful. What if anything did you think the language (and corresponding rule interpreter) was missing, that would have made it easier to write rules?