NPSCS-93-009

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Instructions for Use of the Metutor Means-Ends Tutoring System

Neil C. Rowe

July 1993

TECHNICAL REPORT

October 1, 1992 to July 1993

Prepared for:

Naval Postgraduate School
Monterey, California 93943

This report was prepared with research funded by the Naval Research Funds provided by the Naval Postgraduate School.

This report was prepared by:

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) NPSCS-93-009	5. MONITORING ORGANIZATION REPORT NUMBER(S) Naval Postgraduate School
6a. NAME OF PERFORMING ORGANIZATION Computer Science Dept. Naval Postgraduate School; 6b. OFFICE SYMBOL (if applicable) CS	7a. NAME OF MONITORING ORGANIZATION ONR
6c. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943	7b. ADDRESS (City, State, and ZIP Code) San Diego, CA
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Naval Postgraduate School; 8b. OFFICE SYMBOL (if applicable) NPS	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER DARPA 13 Project under AO 8939
8c. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943	10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
Instructions for use of the Metutor means-ends tutoring system

12. PERSONAL AUTHOR(S)
Neil C. Rowe

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Interim	FROM 9210 TO 9307	930719	19

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	tutoring, computer-aided instruction, means-ends analysis, virtual reality, Prolog, reactive environments, declarative specification

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

The Metutor system is a set of software engineering tools to enable instructors not especially knowledgeable about computers to implement intelligent computerized tutors.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT [✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL Neil C. Rowe	22b. TELEPHONE (Include Area Code) (408) 656-2462; 22c. OFFICE SYMBOL CSRp

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

Instructions for use of the Metutor means-ends tutoring system

Neil C. Rowe
Department of Computer Science
Code CS/Rp, U. S. Naval Postgraduate School
Monterey, CA USA 93943
rowe@cs.nps.navy.mil

1. Introduction

The Metutor system is a set of software engineering tools to enable instructors not especially knowledgeable about computers to implement intelligent computerized tutors. The tutors constructed are for skills requiring the sequential application of actions. The tools are contained at NPS in the directory ˜rowe/pro/metutor, and the main files are:

--**metutor27** etc: the core problem-independent code, written Quintus Prolog;
--**mefire**, the original demonstration application for firefighting, which can be run either with or without graphics;
--**megraph27** etc: the problem-independent graphics interface for the student, written in Quintus Prolog using the Prowindows Xwindows interface; this is loaded automatically when the program is started with **graphicsflag** asserted;
--**drawpath**, a program that helps the instructor construct line drawings with the mouse, and save them as bitmaps (this runs under Prowindows).

The directory also contain other applications that students have written, most notably:

--**kangfuel**, a aircraft emergency procedure tutor, which also has graphics;
--**galvinhostage**, a hostage-crisis management tutor;
--**seemscuba**, a scuba procedure tutor;

All these files should be unprotected; feel free to explore them. Other example applications, less thoroughly tested, also exist, including tutors for (1) replenishment at sea, (2) cardio-pulmonary resuscitation, and (3) use of an electronic mail tool.

The firefighting tutor is the best example tutor to study because it is well-debugged, has a nice graphics interface, and contains most of the ideas mentioned below. To run it, do **cd ˜rowe/pro/metutor** to Unix, then **prolog**, then **[metutor27,mefire], go.** (note the necessary period at the end).

We discuss now the necessary problem-dependent information that an instructor must specify to build a new tutor application. This will go in a separate file, like **mefire**, that will be loaded together with a **metutor** file to create the tutor.

2. Fact and operator representation

The problem definition that a teacher supplies must follow strict formats compatible with Prolog data. That means facts and actions ("operators") must be in first-order predicate calculus notation. This notation consists either of a single word or a word followed by a left parenthesis, some arguments, then a right parenthesis. Multiple arguments must be separated by commas.

Example operator representations are "yell", "test(oxygen)" meaning to test the oxygen, and "test(oxygen,tester)" meaning to test the oxygen tester. Generally the first word should be a verb, and any remaining words should constitute (when taken in order) the rest of a phrase describing the operator more precisely; otherwise the English paraphrase may sound funny.

Example fact representations are "smokey" meaning there is much smoke present, "open(door)" meaning the door is open, and "inside(fireman,compartment)" meaning the fireman is inside the compartment. Generally with arguments, the word in front of them should be a property or relationship name.

3. Operator recommendations

Please familiarize yourself with means-ends analysis (as in the Rowe book, chapter 11) before attempting a means-ends tutor. Means-ends works on the difference between a current state (a list of facts describing the current state of the world) and a partial goal description (a list of the facts which must all become true for the problem to be solved). Actions ("operators") are specified as recommendations for particular kinds of differences. For instance in firefighting, when you want the fire to be out when it isn't, it is recommended that you try the "extinguish" operator. The difference is defined as the facts in the partial goal description which are not present in the current state, not the other way around.

Most operators can have their recommendation conditions specified by two-argument predicate expressions with predicate name **recommended**. The first argument is a list (that is, items separated by commas with square brackets around the whole thing) of facts appearing in the partial goal description but not the current state. The second argument is the name of the operator recommended then. For instance:

recommended([out(fire)],extinguish).

Note the period on the end of the line: Every problem-definition assertion discussed in this manual must end with a period or the Prolog compiler will note a syntax error.

If you have more than one difference list recommending the same operator, that is like an "or" or disjunction of the two sets of conditions. Recommendation conditions can also be negative; that is, an operator may be recommended if you want something to become false. Then put the word **not** and a left parenthesis in front of the fact, and follow with a right parenthesis.

You can also use a three-argument form for recommended operators. The extra argument is the second argument, which is interpreted to be a context, a list of facts that must be present in the current state for the recommendation to be applied. Since the first argument holds facts that you desire to make true, the three-argument form permits a more precise recommendation. Example:

**recommended([not(present(casualty))], [present(medical,corpman)],
  direct_medical_corpman).**

This says that if you want a casualty to no longer be present, then if there is a medical corpman present, you should direct the corpman to handle the casualty.

When more than one operator is recommended for the same situation, the three-argument forms have priority over the two-argument forms. But among the two-argument forms and among the three-argument forms, the first listed recommendation has priority. This means that it can be important how you order recommendations in some applications. Generally speaking, you should order recommendations by decreasing seriousness of both the operator and the recommendation conditions. If two operators are really equally preferable in every situation, include a **nopref** fact with those operators as arguments. This tells the tutor not to complain if the second one is used before the first. **Nopref** can also take lists of operators as arguments, as in:

nopref([estimate(water),dewater],[desmoke]).

where estimating water and desmoke are specified to be doable in either order, while at the same time dewater and desmoke are specified to be doable in either order.

4. Preconditions

Recommendation conditions should be distinguished from preconditions, the stricter conditions that state when an operator is logically possible. Preconditions are specified by two or three arguments to predicate name **precondition**. In the two-argument form, the first argument is the operator name and the second is the list of precondition facts. For example:

```
precondition(extinguish,[location(fire),raging(fire),equipped(team),
  set(boundaries),confronted(fire)]).
```

This says that in order to extinguish a fire, your location must be at the fire, the fire must be raging, the fire team must be equipped, the boundaries of the fire must be set, and you must be facing the fire.

In the three-argument form, the additional argument is the second, the context (list of facts which must be present in the current state) for the precondition facts to apply. The difference between the first and second arguments is that the first represents subgoals that must be achieved if not already true, whereas the second argument represents facts that are just checked without any attempt to achieve them. The three-argument form is useful when two or more operators can achieve the same goal facts, and the best choice depends on the current state.

As with recommendation conditions, the three-argument forms have priority over the two-argument forms. Also note that a precondition assertion must be given for every operator in the problem; if there no preconditions on application an operator, it should have the empty list ([]) as its precondition list.

Precondition facts can also be negative, meaning they must be false before an operator can be applied. Again, use the **not** construct. Such preconditions succeed if the indicated fact is absent in the state to which you want to apply the operator. But **not** facts will never appear in state descriptions, since the absence of a fact is equivalent to the assertion of the negative of that fact.

Metutor is intended for applications that present significant numbers of feasible alternative approaches to the student (else a simpler tool is appropriate). Thus, the instructor is cautioned not overspecify preconditions so that the student is confined to only a narrow corridor of possible operator sequences. Instead, the student should be allowed to see the consequences of their suboptimal operators. For instance, if a firefighting student forgets to equip their fire team, the team members should get burnt when they try to extinguish the fire; this is more educational than putting an extra precondition on "extinguish" of "equipped(team)". In general, preconditions should be either logically necessary conditions (like you cannot put out a fire unless the fire is still burning) or critical requirements of the application.

5. Postconditions

Postconditions represents the consequences of an operator. There two kinds, facts that are false (if they ever were) after an operator is applied, "deletepostconditions", and facts that are true (if they were not already) after the operator is applied, "addpostconditions". The deletepostconditions are removed first, then the addpostconditions are added. Postconditions should represent direct consequences of the operator.

The predicate names used are **deletepostcondition** and **addpostcondition**. They have two-argument, three-argument, and four-argument forms. In the two-argument form, the first argument is operator name and second argument is the list of postconditions. In the three-argument form, the added middle argument is a context like with three-argument recommendations and preconditions. The four-argument form is like the three-argument form except that its last argument is a message string, enclosed in apostrophes, which will be printed out for the student when the postconditions are applied; this is useful for unusual postconditions. Here are some examples:

```
deletepostcondition(extinguish,[raging(fire),tested(gases),tested(oxygen),
  verified(out(fire)),watched(reflashing),debriefed(team),set(boundaries),
  safe(gases),safe(oxygen),unsafe(gases),unsafe(oxygen),confronted(fire)]).
addpostcondition(extinguish,[out(fire),watery,smokey]).
addpostcondition(extinguish,[not(deenergized(fire,area))],
   [present(casualty),dead(casualty),present(crater),raging(fire)],
   'There is a big explosion!').
```

The first definition gives all the things that become false, if not already, when a fire is extinguished: that it is raging, that oxygen and other gases are tested or safe or unsafe, that the fire is verified to be out, that reflashing of the fire is being watched, that the fire team is debriefed, that the boundaries are set, and that the fire is confronted. The second definition gives the things that usually become true in the same situation: that the fire is out, that the area is watery, and that the area is smokey. The last definition gives the things that become true in the special case in which the student tries to extinguish when the power to the fire area is not off: that a casualty is present and dead, that a crater in the floor is present, and that the fire is raging. Also, when the special case for the third rule occurs, the message "There is a big explosion!" is printed out.

Four-argument forms have priority over three-argument forms, and three-argument forms have priority over two-argument forms. This means special cases have priority over defaults.

The first argument to all these postcondition forms may also be a list of operators instead of a single operator. Then the postconditions apply to all operators in that list. Note also there cannot be negative postconditions.

6. Random changes to states

One of the must important features of the Metutor system is ability to specify random changes to states. Such changes can suggest the indeterminacy of the real world, as well as unforeseen complications that can arise. For instance in firefighting, the fire may refuse to go out after one attempt at extinguishing it, or a member of the fire team may get injured, or someone may accidentally turn the power back on.

Such events can be specified by **randchange** facts of five or six arguments. The only difference between the five-argument and the six-argument forms is in an optional sixth argument, a message that can be printed out when the random change is performed. The first argument is the operator name, the second is the context, the third is a list of facts that will be deleted (if present) by the random change, the fourth is list of facts that will be added (if not present) by the random change, and the fifth argument is the change probability (its relative frequency). For example:

```
randchange(extinguish,[],out(fire),raging(fire),0.3,'Fire is still raging.').
```

This says that 30% of the time when a student tries to extinguish a fire, the fire will continue raging.

Each random change that can apply to given state will be applied independently, with independent probabilities. However, if more than one random change is applied, the second change will operate on the state resulting from the first change. Random changes are done after the postconditions are computed.

Randchange facts can refer to a special action "init"; such changes will be applied to the start state, to derive the first state in which the student must choose an action.

As with postconditions, the first argument may also be a list of operators, in which case the random changes apply to any operators from that list.

7. Initialization

Initialization of the tutor for application requires three one-argument predicates. The **intro** predicate takes one argument, a string which printed out at the start of the session to the students. The **start_state** predicate specifies the starting state, and **goal** specifies the top-level partial goal description. The latter is partial only in the sense that you need not specify all the facts that will be true when a goal state is achieved, only the essential facts.

For example, our firefighting tutor has these specifications:

```
intro('You are the fire team leader on a U.S. Navy ship; a fire is reported.').
start_state([location(repair,locker),raging(fire),smokey]).
goal([verified(out(fire)),safe(gases),safe(oxygen),not(equipped(team)),
   not(smokey),not(watery),not(watched(reflashing)),not(present(casualty)),
   not(unreplaced(casualty)),not(treated(casualty)),not(dead(casualty)),
   debriefed(team),deenergized(fire,area)]).
```

8. Associating bitmaps with facts

If the fact **graphicsflag** of no arguments is included somewhere in the problem-defining file, the Prolog compiler will try to display bitmap graphics for facts in the current state that have associated bitmaps. Such associations are specified with the **bmap** predicate of five or six arguments. The first argument is a fact, the second is a context under which this bitmap should be displayed, the third is the name of the file containing the bitmap to be loaded, the fourth is the X-coordinate of where the upper left corner of the bitmap should go on the screen, and the fifth is the Y-coordinate of the upper left corner. For instance:

```
bmap(raging(fire), [location(fire)], fire2,308,135).
```

This says that if the fire is raging and the fire team is located at the fire, the screen should show flames (whose bitmap is in file "fire2") with upper left corner of the bitmap at (308,135). Dimensions represent integer numbers of pixels, which on a Sun terminal are approximately 0.2 mm. Coordinates are measured from left to right for the X-coordinate and top to bottom for the Y-coordinate.

An optional sixth argument to **bmap** is the color associated with the bitmap, which is only used on a color terminal, determined automatically by querying your terminal. The color argument must be one of white, black, gray, red, blue, green, yellow, purple, and cyan (do not quote the color name).

Since color graphics do not print well in a screen dump with the "xwd" command of XWindows, include the zero-argument **nocolorflag** in your file to prevent it from using color on a color terminal if you want to do screen dumps. To be able to do a screen dump just by typing "printscreen" and left-clicking the mouse in a background area of your screen, include this line in your ".cshrc" file:

```
alias printscreen 'xwd | xpr -device ps | lpr -Pps'
```

You can have more than one shape shown for the same fact, if you place them far enough apart on the screen. Shapes are stored as rectangular bitmaps; whenever two such rectangles overlap on the screen, the colored areas of both are intermingled by default, with the color chosen randomly when two different colors appear at the same place on the screen for the two rectangles. If you prefer instead that certain shapes occlude others, include facts of predicate name **draw_order** to say the first-argument bitmap file should be drawn before the second. For instance:

```
draw_order(fire1,team1).
```

says that the bitmap in file "team1" should be drawn over bitmap in "fire1". Text (see below) cannot be made to occlude unless you put it a bitmap.

The bitmap file necessary should be in Postscript black/white form, even if it is to be drawn in a color.

Two utilities, the simple **drawpath** and the more elaborate **drawgraph**, can be used to create simple line drawings and shapes and put them in bitmaps. These files are in ~rowe/pro/pw; start Prowindows, and them load them by **['~rowe/pro/pw/drawpath'].** or **['~rowe/pro/pw/drawgraph'].** (note the periods at the end).

Some facts are better represented by text labels in the drawing part of the screen, or maybe text is appropriate in addition to a shape, like labels on a picture of switch. If so, and black non-occluding labels are sufficient, use **text** facts of five arguments like **bmap** facts but where the third argument is the text string to display rather than a file to load. Text is always displayed in black. You can display several texts at different places for the same fact.

9. Nonverbal action selection

Besides clicking on the menu items, you can define left-clicks within the picture area to select operators. For instance, if you have a picture of vertical switch in the picture, left-clicking near the top of it can turn the switch up. To implement this, you must specify the coordinates of a rectangular area within the picture wherein a click means a given operator name. Predicate **opclick** needs the coordinates of the upper left corner first, then the x-width and y-height, then the context, then the operator name. For example:

```
opclick(200,20,20,50,[visible(switch,power)],turn(power,on)).
```

says that whenever the student clicks the left mouse button within the area 20 pixels wide and 50 pixels high with upper-left corner (200,20), and whenever the power switch is then visible, it means that the student is choosing the operator "turn power on".

10. Debugging flags

Two flags, **debugflag** and **studentflag**, help debug programs. The first helps debug the instructor's **recommended**, **precondition**, **deletepostcondition**, and **addpostcondition** definitions. When **debugflag** is asserted (by **assert(debugflag).**), the program will print out exhaustive information about all the hypothetical reasoning that it does.

The debugflag printout uses level numbers, which refer to depth in the means-ends problem-decomposition tree, counting the root problem as level 1. The printout is sufficient for you to build a complete means-ends tree if you need to figure out more precisely what the hypothetical reasoning is doing. The "Unsolvable problem" message is the most common indication of a bug (though it does not always mean a bug, as when two operators are recommended for a situation and the program must backtrack if it picks the wrong one). Its first occurrence is usually the most significant, because one unsolvable subproblem can make problems that depend upon it unsolvable too. It suggests a missing recommendation or addpostcondition for an operator, or an unnecessary precondition or deletepostcondition. The operator last mentioned when the unsolvable problem is detected is often the operator with the bug.

The debugflag also causes printout of other diagnostic information, including the operator that the tutor thinks is best at every point, and for a graphics program, the list of bitmaps and their parameters for the current screen display. The program caches the hypothetical reasoning, so the second time you run a problem you will get less debugging information, with indications as to when the cache is invoked. Thus, if you are swamped with information by setting the debugflag initially, first run the program awhile, type control-C and "a" for abort, and then type "go." again to restart.

When **studentflag** is asserted (by **assert(studentflag).**), the system avoids checking the instructor's code for a number of common careless errors. These include missing definitions, wrong argument types, and some spelling errors. This checking takes time, so it should be avoided when instructor is sure their code is correct.

11. English language output

The system also prints out some English paraphrases of the states encountered and objectives desired. This code simply assumes that words ending in "s" are plural, otherwise singular. If this not true, like for "gas" and "sheep", include one-argument facts **singular** or **plural** to indicate the proper forms.

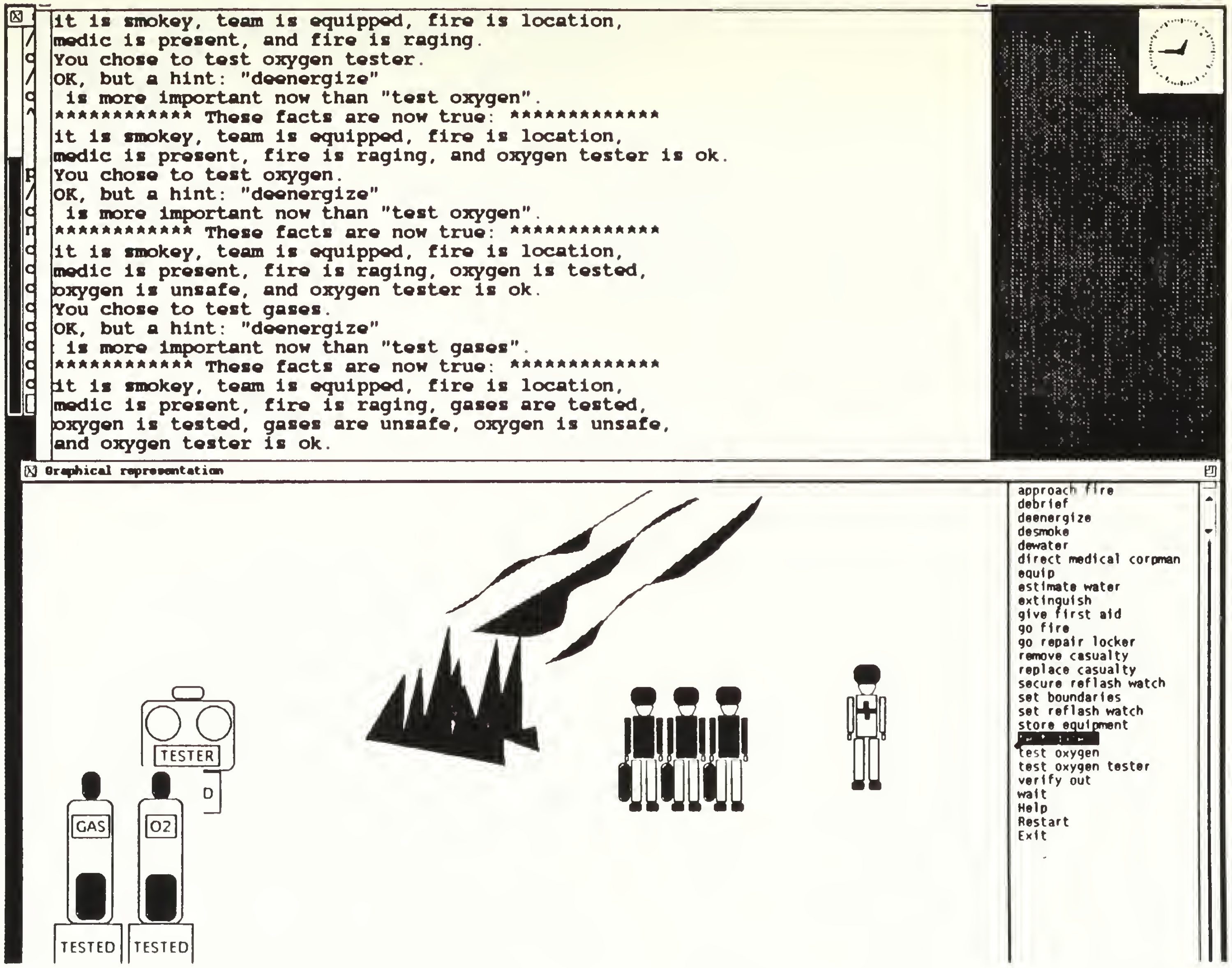
The **writelist** and **writefact** routines control the printout of English paraphrases. They are defined by specifications of many special cases. More advanced programmers may wish to modify them to fine-tune the English paraphrases for their applications.

12. Example

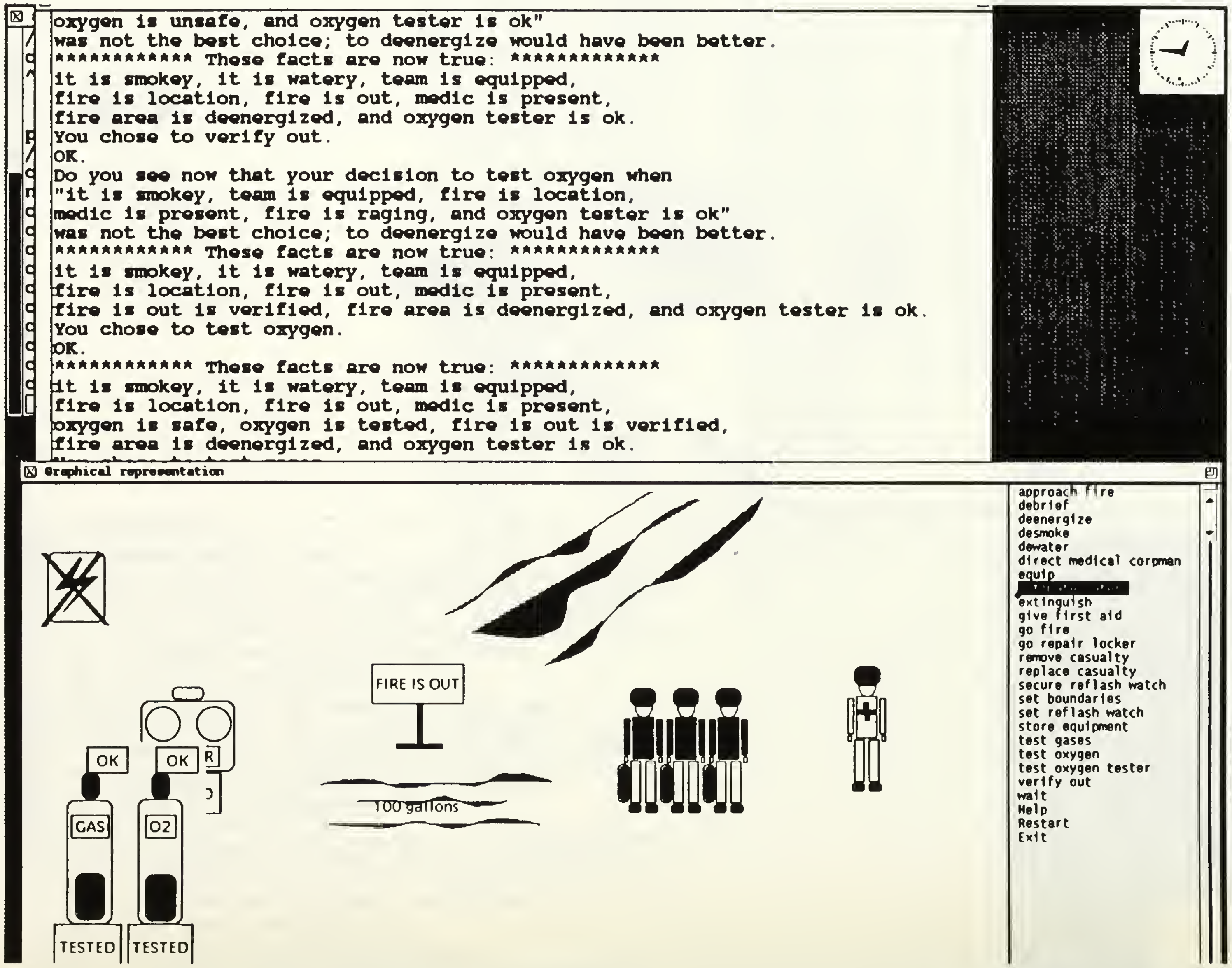
The rest of this paper shows the full definition for the firefighting tutor, illustrating most of the features described above. We also show some screen displays produced by this program.

```
/* Problem definition for the means-ends firefighting tutor */

intro('You are the fire team leader on a U.S. Navy ship.  A fire has been reported.').

recommended([treated(casualty)],[],direct(medical,corpman)).
recommended([treated(casualty)],[],give(first,aid)).
recommended([not(present(casualty))],[],remove(casualty)).
recommended([not(unreplaced(casualty))],[],replace(casualty)).
recommended([out(fire)],[deenergized(fire,area)],extinguish).
recommended([out(fire)],[not(deenergized(fire,area))],deenergize).
recommended([verified(out(fire))],[],verify(out)).
recommended([deenergized(fire,area)],[],deenergize).
recommended([set(boundaries)],[],set(boundaries)).
recommended([confronted(fire)],[deenergized(fire,area)],approach(fire)).
recommended([confronted(fire)],[not(deenergized(fire,area))],deenergize).
recommended([watched(reflashing)],[],set(reflash,watch)).
recommended([safe(gases)],[],test(gases)).
recommended([ok(oxygen,tester)],[],test(oxygen,tester)).
recommended([safe(oxygen)],[],test(oxygen)).
recommended([not(smokey)],[],desmoke).
recommended([estimated(water)],[],estimate(water)).
recommended([not(watery)],[],dewater).
recommended([debriefed(team)],[],debrief).
recommended([equipped(team)],[],equip).
recommended([not(equipped(team))],[],store(equipment)).
recommended([not(watched(reflashing))],[],secure(reflash,watch)).
recommended([location(fire)],[equipped(team)],go(fire)).
recommended([location(fire)],[not(equipped(team))],equip).
recommended([location(repair,locker)],[],go(repair,locker)).
recommended([safe(X)],[],wait).

precondition(remove(casualty),
  [present(casualty),treated(casualty),not(dead(casualty))]).
precondition(direct(medical,corpman),[present(casualty),
  present(medic),not(angry(medic)),not(dead(casualty))]).
precondition(give(first,aid),[present(casualty),not(dead(casualty))]).
precondition(replace(casualty),[unreplaced(casualty)]).
precondition(equip,[location(repair,locker),not(equipped(team))]).
precondition(deenergize,[location(fire)]).
precondition(set(boundaries),[location(fire),not(set(boundaries))]).
precondition(approach(fire),[location(fire),not(confronted(fire)),
  raging(fire),set(boundaries),equipped(team)]).
precondition(extinguish,[location(fire),raging(fire),equipped(team),
  set(boundaries),confronted(fire),not(dead(casualty))]).
precondition(set(reflash,watch),[not(watched(reflashing)),
  verified(out(fire)),safe(gases),safe(oxygen)]).
precondition(verify(out),[location(fire),out(fire)]).
precondition(test(oxygen,tester),[equipped(team)]).
precondition(test(oxygen), [ok(oxygen,tester),equipped(team),
    not(unsafe(oxygen)),not(safe(oxygen)),location(fire)]).
precondition(test(gases),
   [equipped(team),not(unsafe(gases)),not(safe(gases)),location(fire)]).
precondition(desmoke,[location(fire),out(fire),smokey]).
precondition(estimate(water),[location(fire),watery,out(fire)]).
precondition(dewater,[location(fire),watery,estimated(water)]).
precondition(store(equipment),[location(repair,locker),equipped(team)]).
precondition(debrief,[location(repair,locker),not(equipped(team)),
  watched(reflashing)]).
precondition(secure(reflash,watch),[watched(reflashing),debriefed(team)]).
precondition(go(fire),[location(repair,locker),not(dead(casualty))]).
precondition(go(repair,locker),[location(fire),not(dead(casualty))]).
precondition(wait,[]).

deletepostcondition(remove(casualty),[present(casualty),treated(casualty)]).
deletepostcondition(direct(medical,corpman),[]).
deletepostcondition(give(first,aid),[]).
deletepostcondition(replace(casualty),[unreplaced(casualty)]).
deletepostcondition(equip,[debriefed(team)]).
deletepostcondition(deenergize,[energized,not(smokey),tested(gases),
  tested(oxygen),debriefed(team),verified(out(fire))]).
deletepostcondition(set(boundaries),[tested(gases),tested(oxygen),
  debriefed(team),verified(out(fire)),confronted(fire)]).
deletepostcondition(approach(fire),[tested(gases),tested(oxygen),
  debriefed(team),verified(out(fire))]).
deletepostcondition(extinguish,[raging(fire),tested(gases),tested(oxygen),
  verified(out(fire)),watched(reflashing),debriefed(team),set(boundaries),
  safe(gases),safe(oxygen),unsafe(gases),unsafe(oxygen),confronted(fire)]).
deletepostcondition(set(reflash,watch),[]).
deletepostcondition(verify(out),[]).
deletepostcondition(test(gases),[unsafe(gases),safe(gases)]).
deletepostcondition(test(oxygen,tester),[]).
deletepostcondition(test(oxygen),[unsafe(oxygen),safe(oxygen)]).
deletepostcondition(desmoke,[smokey,debriefed(team),unsafe(gases),
  tested(gases),unsafe(oxygen),tested(oxygen)]).
deletepostcondition(estimate(water),[]).
deletepostcondition(dewater,[watery,estimated(water),debriefed(team),
  tested(gases),unsafe(gases),tested(oxygen),unsafe(oxygen)]).
deletepostcondition(store(equipment),[equipped(team)]).
deletepostcondition(debrief,[]).
deletepostcondition(secure(reflash,watch),[watched(reflashing)]).
deletepostcondition(go(fire),[location(repair,locker)]).
deletepostcondition(go(repair,locker),[location(fire),confronted(fire),
          tested(gases),tested(oxygen)]).
deletepostcondition(wait,[tested(gases),tested(oxygen),unsafe(gases),
          unsafe(oxygen)]).

addpostcondition(remove(casualty),[unreplaced(casualty)]).
addpostcondition(direct(medical,corpman),[treated(casualty)]).
addpostcondition(give(first,aid),[treated(casualty)]).
addpostcondition(replace(casualty),[]).
addpostcondition(equip,[equipped(team)]).
addpostcondition(deenergize,[deenergized(fire,area),smokey]).
addpostcondition(set(boundaries),[set(boundaries),smokey]).
addpostcondition(approach(fire),[confronted(fire),smokey]).
addpostcondition(extinguish,[out(fire),watery,smokey]).
addpostcondition(set(reflash,watch),[watched(reflashing)]).
addpostcondition(verify(out),[verified(out(fire))]).
addpostcondition(test(gases),[tested(gases),safe(gases)]).
addpostcondition(test(oxygen,tester),[ok(oxygen,tester)]).
addpostcondition(test(oxygen),[tested(oxygen),safe(oxygen)]).
addpostcondition(desmoke,[]).
addpostcondition(estimate(water),[estimated(water)]).
addpostcondition(dewater,[]).
addpostcondition(store(equipment),[]).
addpostcondition(debrief,[debriefed(team)]).
addpostcondition(secure(reflash,watch),[]).
addpostcondition(go(fire),[location(fire)]).
addpostcondition(go(repair,locker),[location(repair,locker)]).
addpostcondition(wait,[]).

deletepostcondition(deenergize,[not(equipped(team))],[tested(gases),
        tested(oxygen),verified(out(fire))]).
deletepostcondition(debrief,[location(fire)],[]).
deletepostcondition(go(fire),[raging(fire)],
   [location(repair,locker),tested(gases),tested(oxygen),
      safe(gases),safe(oxygen)]).

addpostcondition(deenergize,[not(equipped(team))],[present(casualty)]).
addpostcondition(approach(fire),[not(equipped(team))],
   [present(casualty),smokey]).
addpostcondition(test(gases),[location(fire),raging(fire)],
   [tested(gases),unsafe(gases)]).
addpostcondition(test(gases),[location(repair,locker)],
   [tested(gases),safe(gases)]).
addpostcondition(test(oxygen),[location(fire),raging(fire)],
   [tested(oxygen),unsafe(oxygen)]).
addpostcondition(test(oxygen),[location(repair,locker)],
   [tested(oxygen),safe(oxygen)]).
addpostcondition(give(first,aid),[present(medic)],
   [treated(casualty),angry(medic)]).
addpostcondition(extinguish,[not(deenergized(fire,area))],
   [present(casualty),dead(casualty),present(crater),raging(fire)],
   'There is a big explosion!').
addpostcondition(set(reflash,watch),[smokey],[present(casualty)],
   'The reflash watchperson was overcome by the smoke.').
addpostcondition(set(reflash,watch),[unsafe(oxygen)],[present(casualty)],
   'The reflash watchperson collapsed.').
addpostcondition([go(repair,locker),go(fire),equip,deenergize,
    set(boundaries),approach(fire),extinguish,desmoke,estimate(water),
    dewater,test(gases),test(oxygen,tester),test(oxygen),
    set(reflash,watch),store(equipment),debrief,secure(reflash,watch)],
  [present(casualty)],[dead(casualty)],'Your casualty died!').

randchange(init,[],[],present(medic),0.5).
randchange([approach(fire),extinguish],[],[],present(casualty),0.15,
   'A team member got burned.').
randchange(extinguish,[],out(fire),raging(fire),0.3,'Fire is still raging.').
randchange([verify(out),desmoke,dewater,store(equipment),debrief],
   [not(verified(out(fire)))],
   [out(fire),verified(out(fire)),debriefed(team),tested(gases),
      tested(oxygen),safe(gases),safe(oxygen),watched(reflashing)],
  [raging(fire),smokey],0.3,'Unfortunately the fire has flared up again.').
```

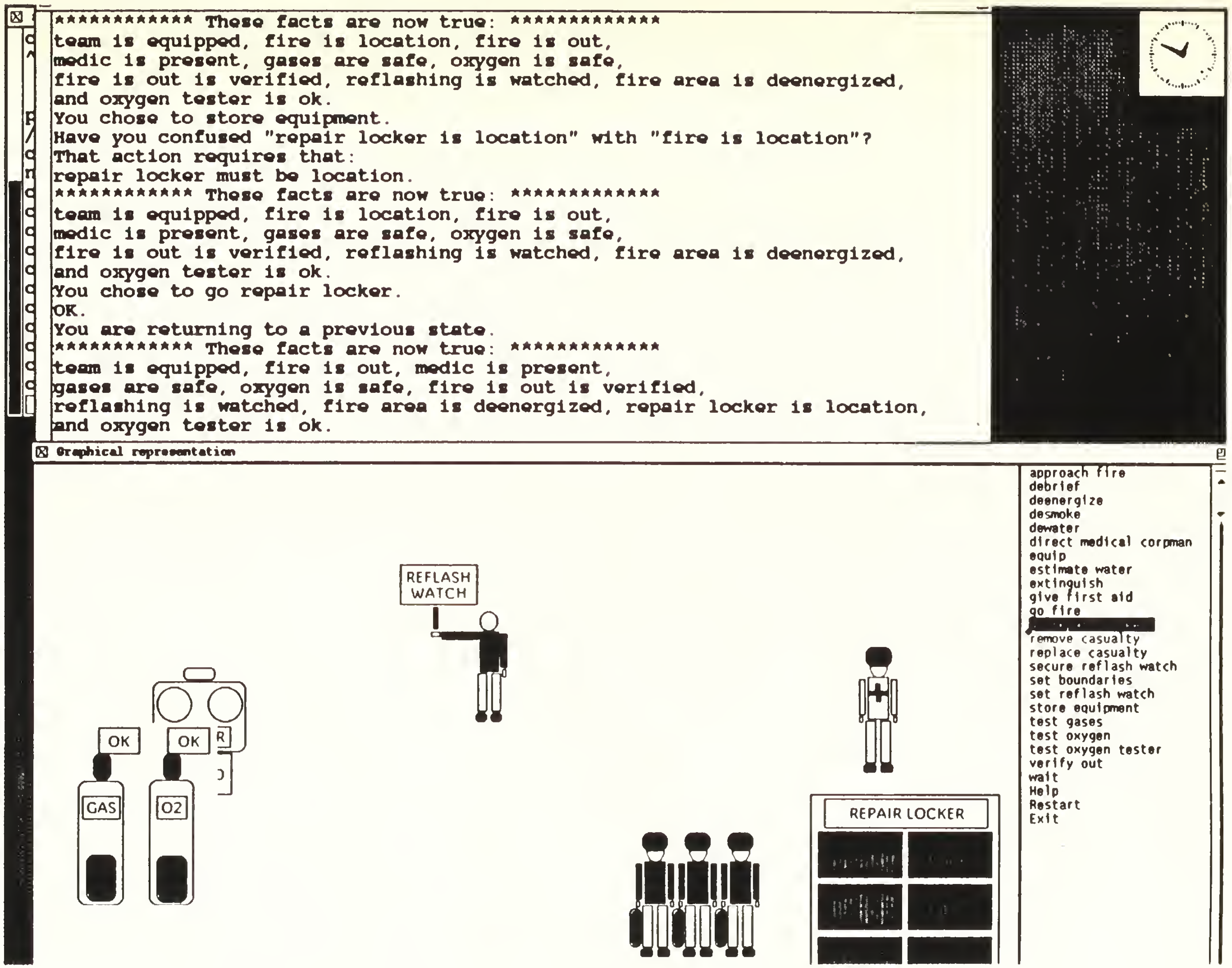
```
randchange([verify(out),desmoke,dewater,store(equipment),debrief],
  [verified(out(fire))],[out(fire),verified(out(fire)),debriefed(team),
    tested(gases),tested(oxygen),safe(gases),safe(oxygen),watched(reflashing)],
  [raging(fire),smokey],0.07,'Unfortunately the fire has flared up again.').
randchange([desmoke,dewater,extinguish],deenergized(fire,area),
   deenergized(fire,area),[],0.08,'A team member accidentally turned the power on.').
randchange([desmoke,dewater,store(equipment)],[],[],present(casualty),
   0.08,'A team member got injured.') :-
randchange(test(gases),safe(gases),safe(gases),unsafe(gases),0.3,
   'The gases are unsafe.').
randchange(test(oxygen),safe(oxygen),safe(oxygen),unsafe(oxygen),0.3,
    'The oxygen is unsafe.').

nopref(set(boundaries),deenergize).
nopref([test(oxygen,tester),test(oxygen)],[test(gases)]).
nopref([estimate(water),dewater],[desmoke]).
nopref([estimate(water),dewater],[test(gases)]).
nopref(desmoke,[test(gases)]).

/* The following are info for the graphics interface */
bmap(location(repair,locker),[equipped(team)],equipped3,550,343,blue).
bmap(location(repair,locker),[not(debriefed(team))],
   location_repair_locker,560,300,blue).
bmap(location(fire),
   [equipped(team),not(confronted(fire)),not(present(casualty)),
      not(unreplaced(casualty))],
   equipped3,550,190,blue).
bmap(location(fire), [equipped(team),not(confronted(fire)),present(casualty)],
   equipped2,550,190,blue).
bmap(location(fire),
   [equipped(team),not(confronted(fire)),unreplaced(casualty)],
   equipped2,550,190,blue).
bmap(location(fire),
   [not(equipped(team)),not(confronted(fire)),not(present(casualty)),
       not(unreplaced(casualty))],
   unequipped3,550,190,blue).
bmap(location(fire),
   [not(equipped(team)),not(confronted(fire)),present(casualty)],
   unequipped2,550,190,blue).
bmap(location(fire), [not(equipped(team)),not(confronted(fire)),
      unreplaced(casualty)],
   unequipped2,550,190,blue).
bmap(confronted(fire),[],confronted_fire,464,19,blue).
bmap(raging(fire), [location(repair, locker)], fire1,2,158,red).
bmap(raging(fire), [location(fire)], fire2,308,135,red).
bmap(smokey, [location(repair, locker)], smokey1,8,1,yellow).
bmap(smokey, [location(fire)], smokey2,380,6,yellow).
bmap(watery,[],watery,273,272,green).
bmap(present(medic),[not(angry(medic)),not(treated(casualty))],
   present_medical_corpman,775,170,red).
bmap(present(medic),[not(angry(medic)),treated(casualty)],
   present_medical_corpman,742,170,red).
bmap(present(medic),[angry(medic)],present_medical_corpman,850,50,red).
bmap(present(casualty),[],present_casualty,651,188,blue).
bmap(treated(casualty),[],treated_casualty,639,236,blue).
bmap(watched(reflashing),[],watched_reflashing,330,90,blue).
bmap(set(boundaries),[location(fire)],set_boundaries,240,227,purple).
bmap(verified(out(fire)),[location(fire)],verified_out_fire,316,163,purple).
bmap(deenergized(fire,area),[location(fire)],deenergized_fire_area,11,57).
bmap(debriefed(team),[],debriefed_team,526,200,blue).
bmap(ok(oxygen,tester),[],ok_oxygen_tester,100,190,cyan).
bmap(tested(oxygen),[],tested_oxygen,89,266).
bmap(tested(gases),[],tested_gases,15,267).
bmap(safe(oxygen),[],safe_oxygen,88,243,purple).
bmap(safe(gases),[],safe_gases,31,242,purple).
bmap(present(crater), [location(fire)], smokey2,100,190,yellow).
bmap(present(crater), [location(fire)], smokey2,5,100,yellow).
bmap(present(crater),[],present_crater,20,250).
bmap(dead(casualty),[],dead_casualty,665,190).

text(estimated(water),[],"100 gallons",330,285).

opclick(273,272,230,61,[location(fire),watery],dewater).
opclick(380,6,320,175,[location(fire),smokey],desmoke).
opclick(89,266,81,197,[location(fire),unsafe(oxygen)],test(oxygen)).
opclick(15,267,85,195,[location(fire),unsafe(gases)],test(gases)).
opclick(775,170,55,125,[location(fire),present(medic),not(angry(medic))],
   direct(medical,corpman)).

draw_order(ok_oxygen_tester,tested_oxygen).
draw_order(ok_oxygen_tester,safe_oxygen).
draw_order(present_casualty,present_medical_corpman).
draw_order(treated_casualty,present_medical_corpman).

start_state([location(repair,locker),raging(fire),smokey]).
goal([verified(out(fire)),safe(gases),safe(oxygen),not(equipped(team)),
   not(smokey),not(watery),not(watched(reflashing)),not(present(casualty)),
   not(unreplaced(casualty)),not(treated(casualty)),not(dead(casualty)),
   debriefed(team),deenergized(fire,area)]).
```



```
it is smokey, team is equipped, fire is location,
medic is present, and fire is raging.
You chose to test oxygen tester.
OK, but a hint: "deenergize"
 is more important now than "test oxygen".
************ These facts are now true: *************
it is smokey, team is equipped, fire is location,
medic is present, fire is raging, and oxygen tester is ok.
You chose to test oxygen.
OK, but a hint: "deenergize"
 is more important now than "test oxygen".
************ These facts are now true: *************
it is smokey, team is equipped, fire is location,
medic is present, fire is raging, oxygen is tested,
oxygen is unsafe, and oxygen tester is ok.
You chose to test gases.
OK, but a hint: "deenergize"
 is more important now than "test gases".
************ These facts are now true: *************
it is smokey, team is equipped, fire is location,
medic is present, fire is raging, gases are tested,
oxygen is tested, gases are unsafe, oxygen is unsafe,
and oxygen tester is ok.
```



```
oxygen is unsafe, and oxygen tester is ok"
was not the best choice; to deenergize would have been better.
************ These facts are now true: *************
it is smokey, it is watery, team is equipped,
fire is location, fire is out, medic is present,
fire area is deenergized, and oxygen tester is ok.
You chose to verify out.
OK.
Do you see now that your decision to test oxygen when
"it is smokey, team is equipped, fire is location,
medic is present, fire is raging, and oxygen tester is ok"
was not the best choice; to deenergize would have been better.
************ These facts are now true: *************
it is smokey, it is watery, team is equipped,
fire is location, fire is out, medic is present,
fire is out is verified, fire area is deenergized, and oxygen tester is ok.
You chose to test oxygen.
OK.
************ These facts are now true: *************
it is smokey, it is watery, team is equipped,
fire is location, fire is out, medic is present,
oxygen is safe, oxygen is tested, fire is out is verified,
fire area is deenergized, and oxygen tester is ok.
```



```
************ These facts are now true: *************
team is equipped, fire is location, fire is out,
medic is present, gases are safe, oxygen is safe,
fire is out is verified, reflashing is watched, fire area is deenergized,
and oxygen tester is ok.
You chose to store equipment.
Have you confused "repair locker is location" with "fire is location"?
That action requires that:
repair locker must be location.
************ These facts are now true: *************
team is equipped, fire is location, fire is out,
medic is present, gases are safe, oxygen is safe,
fire is out is verified, reflashing is watched, fire area is deenergized,
and oxygen tester is ok.
You chose to go repair locker.
OK.
You are returning to a previous state.
************ These facts are now true: *************
team is equipped, fire is out, medic is present,
gases are safe, oxygen is safe, fire is out is verified,
reflashing is watched, fire area is deenergized, repair locker is location,
and oxygen tester is ok.
```

Distribution List

Defense Technical Information Center Cameron Station Alexandria, VA 22314	2
Library, Code 52 Naval Postgraduate School Monterey, CA 93943	2
Center for Naval Analyses 2000 N. Beauregard Street Alexandria, VA 22311	1
Director of Research Administration Code 08 Naval Postgraduate School Monterey, CA 93943	1
Mr. Russell Davis HQ, USACDEC Attention: ATEC-1M Fort Ord, CA 93941	2
Dr. Neil C. Rowe, Code CSRp Naval Postgraduate School Computer Science Department Monterey, CA 93943	50
Prof. Ted Lewis, CS/Lt Naval Postgraduate School Computer Science Department Monterey, CA 93943	2