# ARTIFICIAL INTELLIGENCE APPLICATIONS TO COMPUTER-ASSISTED INSTRUCTION

Project Progress Report No. 4:

## AN INITIAL ATTEMPT AT BUILDING A STUDENT MODEL USING PRODUCTION RULES

**Jesse M. Heines, Ed.D.**
Staff Consultant

Systems Based Courseware
Educational Services Development & Publishing

June 2, 1983

|d|i|g|i|t|a|l|

Educational Services
Bedford, MA Ø173Ø

## 1.0 ABSTRACT

This report documents my first attempt at implementing some of the concepts described in AI/CAI Project Progress Report No. 3, specifically the ReGIS Laboratory described in Section 5.3 of that report. This report also draws heavily on the concepts presented in other sections of Report No. 3, and the reader is encouraged to familiarize himself or herself with that report before attempting to digest this one.

The organization of this report centers around a paper demonstration of software that I had running under MacLISP on a DECsystem-20 at The Open University in England. (I am just starting the process of converting this software to Common LISP on VAX, and I hope to have that done within the next two months.) The report includes reproductions of screens created on a GIGI terminal and explains how the AI/CAI program advances from one screen to the next.

## 2.Ø TABLE OF CONTENTS

## 3.Ø THE IReGIS COURSE

I have named the course described in AI/CAI Project Progress Report No. 3 "IReGIS," for "Intelligent ReGIS." The purpose of this prototype AI/CAI course is to teach the ReGIS P, V, and C commands using a rule-based tutorial strategy à la that developed by Tim O'Shea (now at The Open University in Milton Keynes, England). The course uses AI production rules to control branching between a large number of independent course modules. The full set of controlling rules defines the course's tutorial strategy.

### 3.1 IReGIS Course Components

IReGIS has four basic components: a set of teaching operations, a task analysis, a student model, and means-ends guidance rules.

**3.1.1 The Teaching Operations** - are instructional activities that the CAI program can present. These may be:

- CAI lessons of the type currently produced by Educational Services,
- exercises directed at reinforcing specific instructional objectives,
- "laboratory" sessions in which students try out graphics commands in a controlled environment, or
- tests.

**3.1.2 The Task Representation** - is a hierarchical list of the skills needed to master the material being taught. It is represented as a directed graph that defines the prerequisite relationships between each skill (see Appendices).

**3.1.3 The Student Model** - is a representation of the student's knowledge in terms of the task analysis and a history of the student's interactions.

**3.1.4 The Means-Ends Guidance Rules** - relate states defined by the student model to sets of teaching operations. These rules determine which teaching operations the AI/CAI program will present next for different student states.

These components and their interrelationships are explained in detail in AI/CAI Project Progress Report No. 3. Brief descriptions of the two components with direct bearing on this paper, the ReGIS Laboratory and the Student Model, are provided below.

## 3.2 The ReGIS Laboratory

This report is basically a paper demonstration of the ReGIS Laboratory, the third of the course's four teaching operations. This teaching operation is particularly interesting because it builds the student model by observing students as they work in a laboratory environment. Within this environment, students type in ReGIS commands and have them evaluated by the system. Correct commands yield graphic results on the screen, while incorrect commands yield detailed error messages.

## 3.3 The Student Model

Each student entry updates the student model, which is a vector describing the student's status on each of the 5Ø skills identified in the course's task representation. The value assigned to each skill is an integer between -3 and +3, inclusive, and carries the following meaning:

- -3 NON-MASTERY DEMONSTRATED on a test. The student has demonstrated that s/he does not possess this skill by failing a test that covered it. This is the strongest assertion of non-mastery that the system can make.

- -2 NON-MASTERY ASSUMED due to incorrect usage in lab. The student is assumed not to possess the skill because s/he has failed a test that covers a skill prerequisite to the one in question.

- -1 NON-MASTERY ASSUMED due to incorrect usage of a prerequisite skill. The student is assumed not to possess the skill because s/he has either demonstrated non-mastery on or used incorrectly a lower level skill for which this skill is a postrequisite. (Note that the skill in question may be more than one level removed from the lower level skill on which the student is actually working.) This is the weakest assertion of non-mastery that the system can make.

- Ø NO DATA. The student has not studied this skill, has not demonstrated mastery on any skill for which it is a prerequisite, and has not demonstrated non-mastery on any skill for which it is a postrequisite.

1 MASTERY ASSUMED due to correct usage of a postrequisite skill. The student is assumed to possess the skill because s/he has either demonstrated mastery on or used a higher level skill for which this skill is a prerequisite. This is the weakest assertion of mastery that the system can make.

2 MASTERY ASSUMED due to correct usage in lab. The student is assumed to possess the skill because s/he used the skill in either the ReGIS laboratory or the directed exercises.

3 MASTERY DEMONSTRATED on a test. The student has demonstrated mastery of this skill by passing a test that covered it. This is the strongest assertion of mastery that the system can make.

These values can be seen to change in the paper demonstration presented herein.

### 3.4 Software Disclaimer

The software described in this document ran under MacLISP on a DECsystem-2Ø at The Open University in England. It is not demonstrable on-line at this time. I am just starting the process of converting this software to Common LISP on VAX, and I hope to have that done within the next two months.

## 4.0 PAPER DEMONSTRATION

### 4.1 Loading MacLISP and IReGIS

Figure 1 shows the commands needed to load MacLISP and the IReGIS courseware under TOPS-20 on the DECsystem-20 at The Open University. (A LISP compiler was available, but I did not attempt to use it. Everything was run in interpretive mode.) Typing:

```
LISP
```

in response to the TOPS-20 monitor's standard @ prompt loaded MacLISP, and then issuing the command:

```
(load 'iregis)
```

to MacLISP initiated loading and interpretation of the commands in the file IREGIS.LSP. The results of this second operation are shown in Figure 2.

### 4.2 Running the IReGIS Course

As explained in Figure 2, the command to run IReGIS from within the LISP interpreter is:

```
(start)
```

This command actually caused interpretation of the LISP function START, which in turn called all subsequent course functions. Figure 3 shows the IReGIS title display, which is the first display generated when the course is run.

Figure 4 shows the keypad keys active during IReGIS. The layout of these keys is consistent with all other Educational Services computer-based instruction courses, but only the shaded keys are functional in the prototype AI/CAI course. Note that implementation of this keypad requires that all student input be processed in single character input mode rather than waiting for the RETURN key to initiate evaluation of student responses. This mode was accomplished by using the MacLISP construction:

```
(sstatus linmode nil)
```

which set "line mode" to NIL, effectively turning off line mode and turning on single character input mode. The construction:

```
(sstatus linmode t)
```

reset line mode, turning it back on and disabling single character input.

```
 TOPS-20 Command processor 5(712)-3
@lisp

;Loading LISP.INI 1                    FRIDAY 18 3 83 AT 9 1 47

(load 'iregis)
█
```

Figure 1

COMMANDS TO LOAD MacLISP AND IReGIS COURSEWARE

```
 --------
 IREGIS
 --------

Loading Module 1 of 10...
Loading Module 2 of 10...
Loading Module 3 of 10...
Loading Module 4 of 10...
Loading Module 5 of 10...
Loading Module 6 of 10...
Loading Module 7 of 10...
Loading Module 8 of 10...
Loading Module 9 of 10...
Loading Module 10 of 10...

Course initialized.  Type:

          (START)

(including the brackets) and then press the RETURN key to run the course.

T
█
```

Figure 2

MESSAGES PRINTED DURING LOADING OF IReGIS COURSEWARE

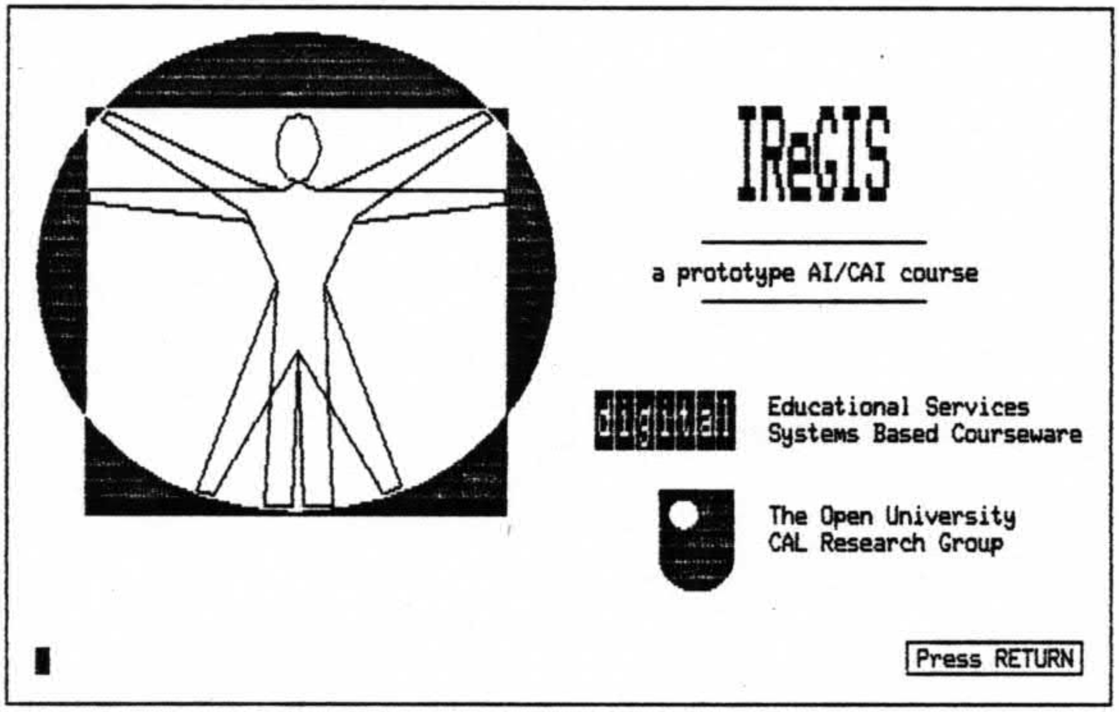



Figure 3

IReGIS COURSE TITLE DISPLAY

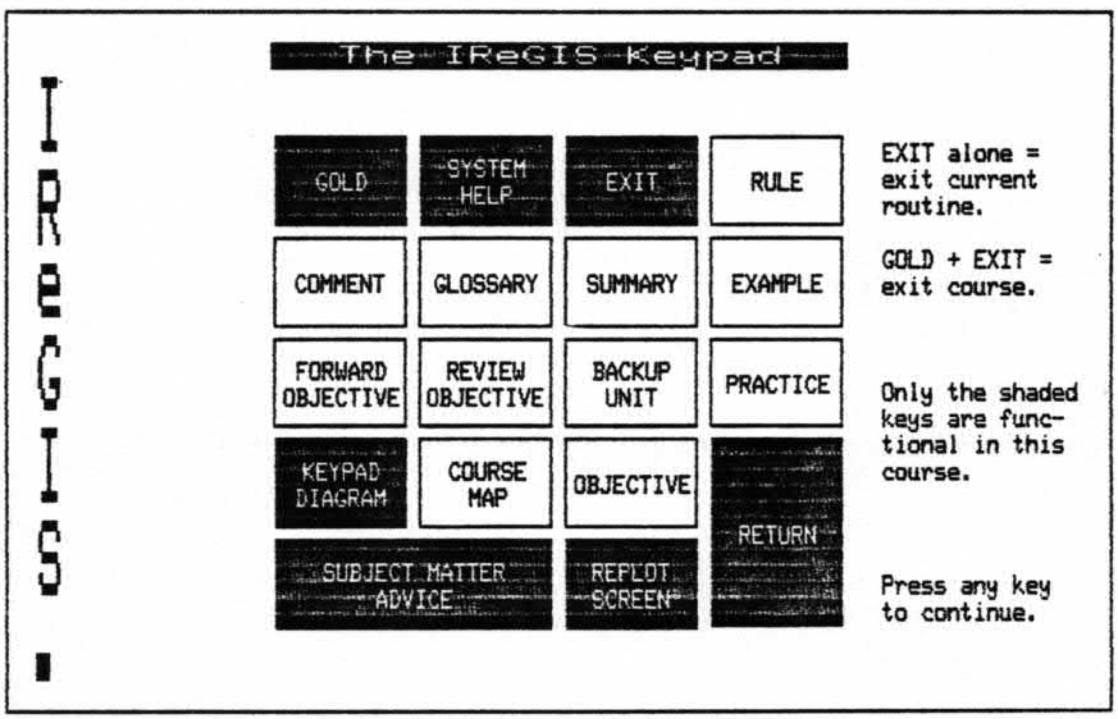



Figure 4

ACTIVE KEYPAD KEYS IN THE IReGIS COURSE

As far as I can tell, this method of input control is unique to MacLISP, and implementation of single character input might be more difficult in the versions of LISP currently available on VAX. I will be working with the Common LISP supplied by the AI Products Group in Hudson, and this will be one of the first issues that I tackle.

### 4.3 Registering a New Student

Pressing RETURN from the title display (Figure 3) causes the Student Registration form in Figure 5 to be displayed. Like the keypad layout, this form is consistent with all other Educational Services computer-based instruction courses.

Pressing the RETURN key alone causes the New Student Registration form in Figure 6 to be displayed. Here the student types his name, Russell, and a new student record is initialized.

### 4.4 Entering the IReGIS Laboratory

The only teaching operation implemented in the MacLISP software is the IReGIS Laboratory. Once the student is registered, the message in Figure 7 is displayed and the system goes directly into this teaching operation. If all four teaching operations were implemented, the system would begin by asking the student whether s/he knew anything about ReGIS rather than going directly into the ReGIS Lab. If the student answered "no" to such a query, the course would go to a menu and allow him or her to take a prerequisite test on the course's entry level skills or begin receiving instruction on basic ReGIS concepts. If the student answered "yes," s/he would be queried further on his or her experience. If s/he so desired, the system would go into the ReGIS Lab as demonstrated here to assess his or her knowledge of basic ReGIS commands.

A considerable amount of initialization occurs while the message in Figure 7 is displayed, thus accounting for the "One moment, please" message. When this initialization is complete, the screen changes automatically to display the form shown in Figure 8.

This form represents the initial state of the ReGIS Lab for new students. It includes messages informing the student of the contents of the stack and the current cursor position, and a section of the screen reserved for displaying the results of ReGIS commands entered by the student. This reserved section is of course only a portion of the screen's addressible area, but the student is requested only to enter commands that keep the cursor within this area. Commands that move the cursor outside the reserved area are not executed.

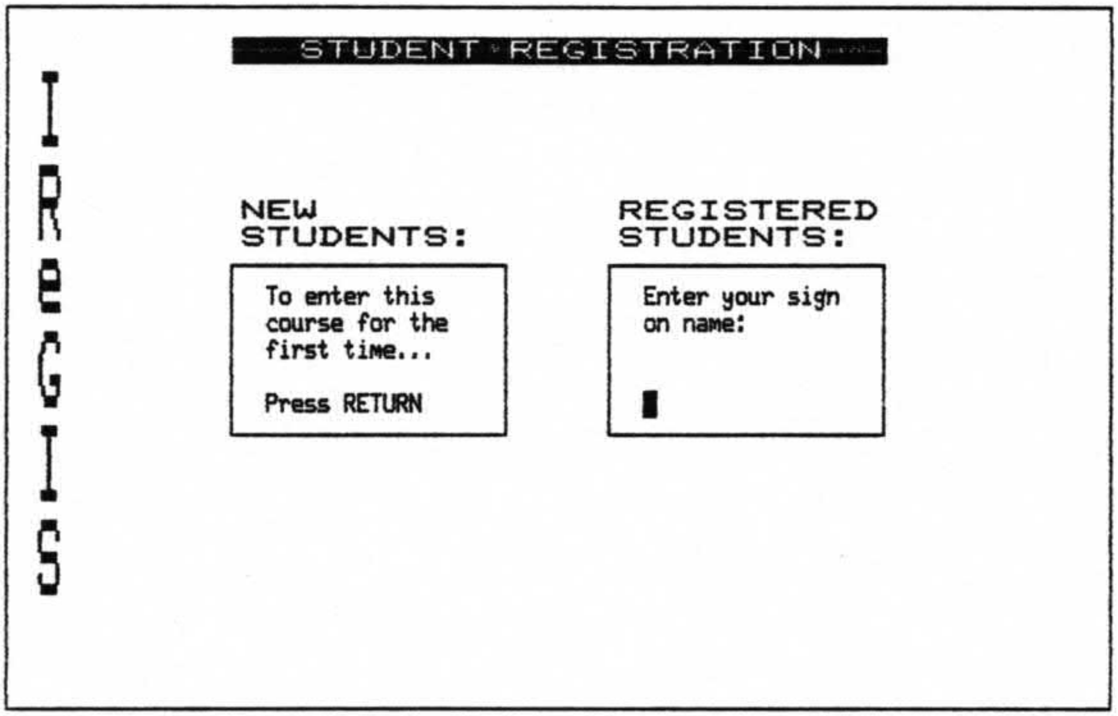



Figure 5

THE REGISTRATION FORM

The student presses the RETURN key without entering a name.

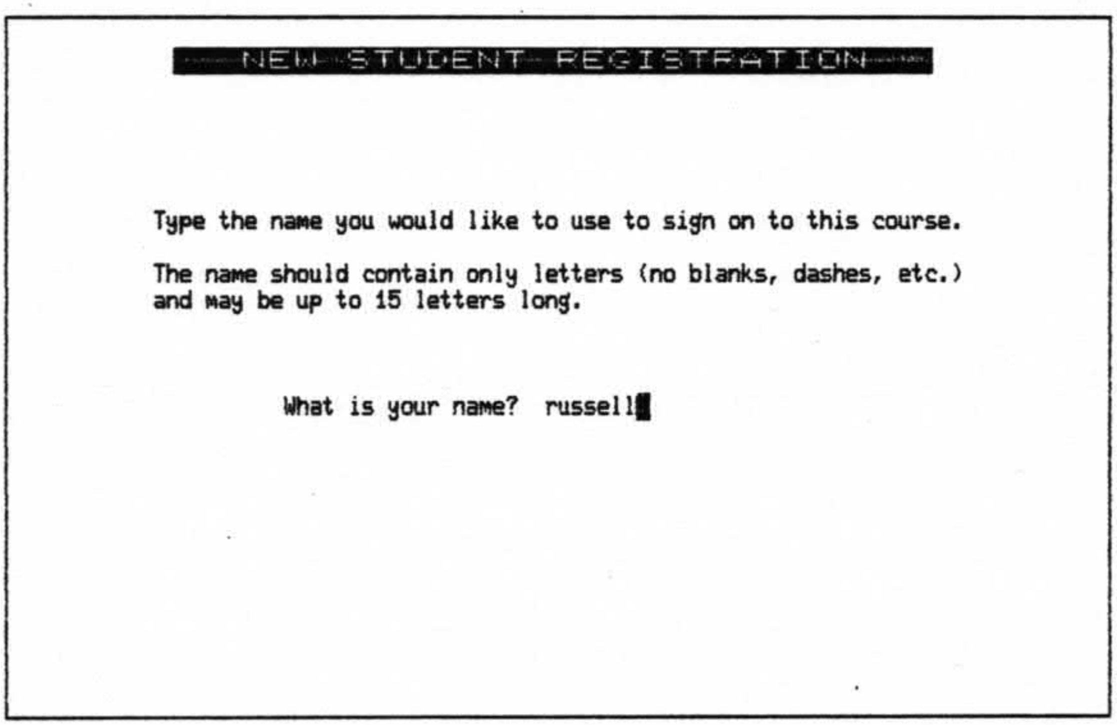



Figure 6

THE NEW REGISTRATION FORM FOR UNREGISTERED STUDENTS

The student enters his name.

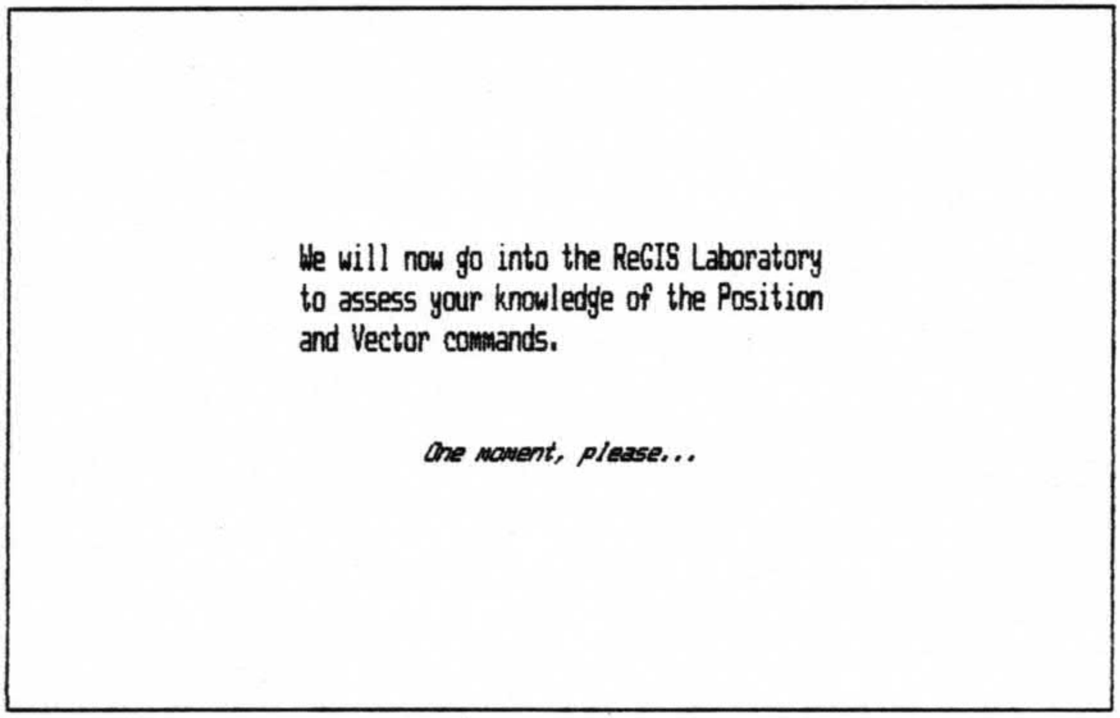



Figure 7

THE ReGIS LAB SETUP MESSAGE

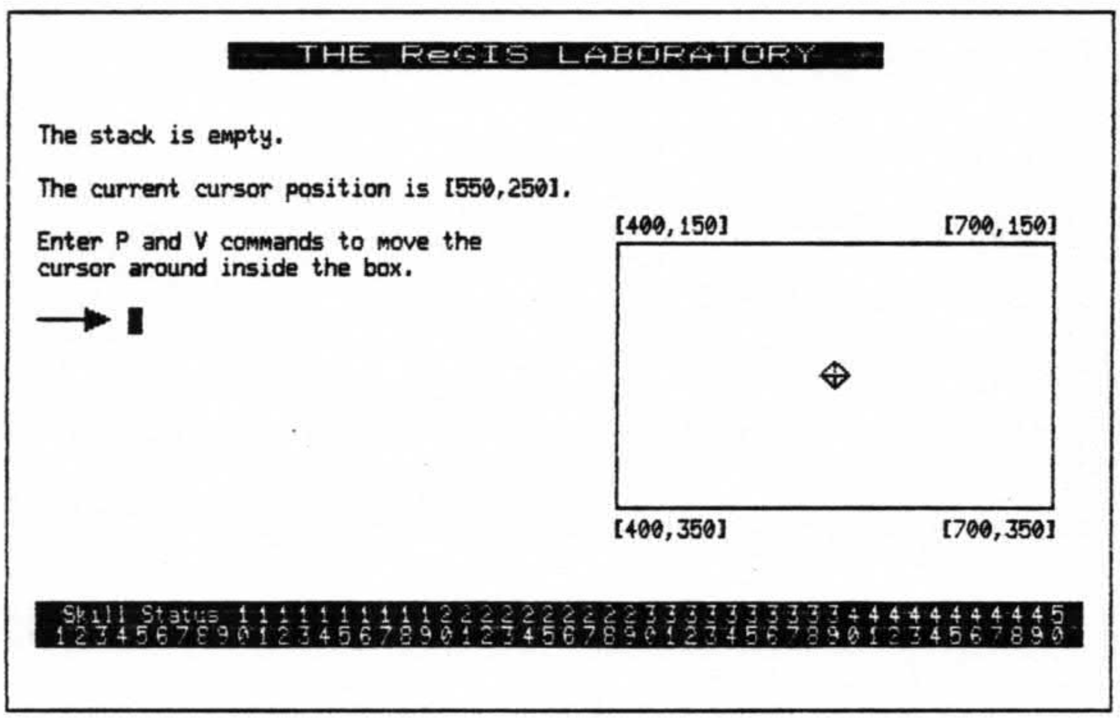



Figure 8

THE INITIAL STATE OF THE ReGIS LABORATORY FOR NEW STUDENTS

Note that no values have yet been assigned to the student model.

The Skill Status display at the bottom of the screen is for demonstration purposes only. This display would not appear in a real version of the course, although it might be advisable to make it available to students if they so request. It has little meaning, however, without a clear understanding of the course's internal task representation, so it is debatable whether it should be made available to students at all. In any event, the display as it appears here is always shown in the demonstration software so that the system's student model building processes can be observed.

The value assigned to each skill (see Section 3.3 above) is displayed below the skill number, but Ø values are suppressed. Since the student has just registered at the point shown in Figure 8, the value assigned to each skill is Ø, and therefore no values are displayed.

### 4.5 Processing a Simple Student Entry

Figure 9 shows a simple student entry to the ReGIS Lab:

```
p[6ØØ,2ØØ]
```

This is a ReGIS Position command and should move the graphics cursor to the point with an X coordinate of 6ØØ and a Y coordinate of 2ØØ.

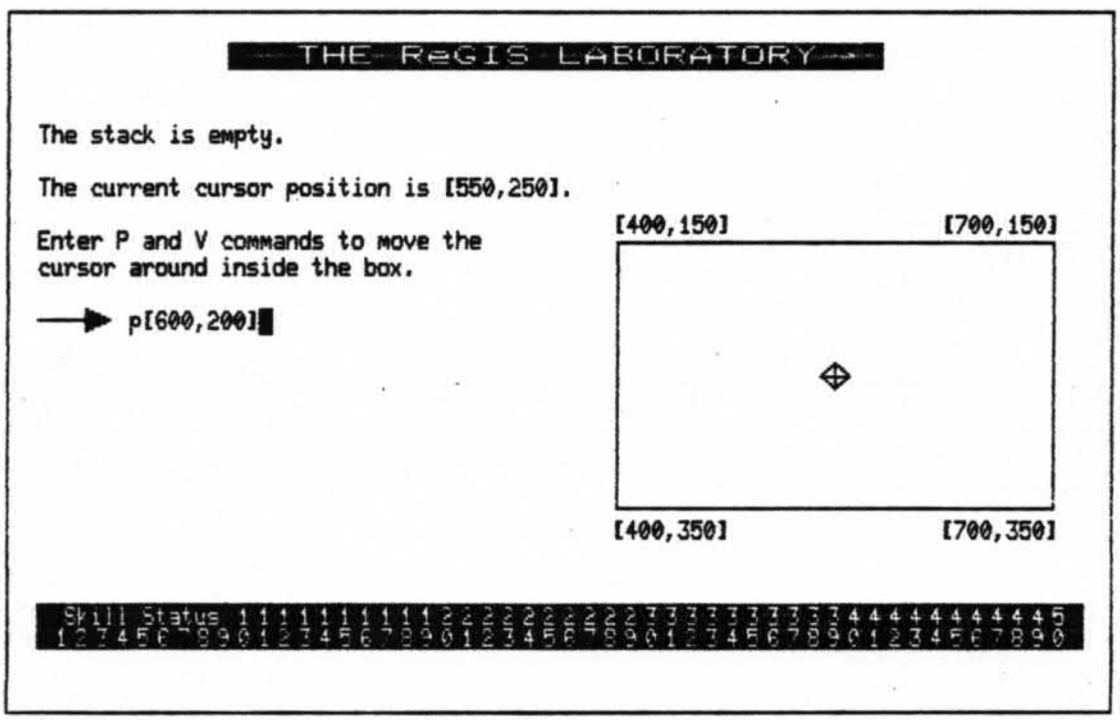



Figure 9

A SIMPLE ReGIS "POSITION" COMMAND ENTRY

The system begins parsing the student's entry in Figure 1Ø. The entered string is repeated, and the system prints "Working..." to indicate that it has begun parsing. The parsing process is rather slow, especially on a loaded system, so the need for some type of "I'm busy" message was critical.

Figure 11 shows what the state of the Laboratory after parsing of the student's entry is complete. The display has changed from that in Figure 1Ø in the following ways:

- The "Working..." message has been erased to indicate that parsing of the entire entry is complete.
- An arrowhead has been positioned under the character at which parsing stopped.
- "OK" has been printed under the arrowhead to indicate that no errors were found during parsing.
- The student's command has been executed in the graphic area at the right of the screen, moving the graphic cursor to position [6ØØ,2ØØ].
- The current cursor position message at the top left of the screen has changed from [55Ø,25Ø] to [6ØØ,2ØØ].
- The wording in the directions has changed slightly since a command has already been processed.
- The student's previous entry has been erased from after the arrow prompt to make room for a new entry.
- The student model vector at the bottom of the screen has been updated.

Look more carefully at the student model vector at the bottom of the screen. A value of +2 has been assigned to Skills 8 and 19 because the student's entry demonstrates correct use of these two skills. Specifically, these two skills are:

8. Can specify absolute screen addresses in [x,y] format.

19. Can use the basic P command to position the cursor.

(The complete list of skills is provided in Appendix A.) A value of +1 has been assigned to Skills 1, 2, 3, 7, 9, and 1Ø (listed below) because these skills are prerequisite to Skills 8 and 19:

1. Recognizes screen as a rectangular dot matrix.

2. Can translate screen positions into (x,y) pairs.

3. Can translate (x,y) pairs into screen positions.

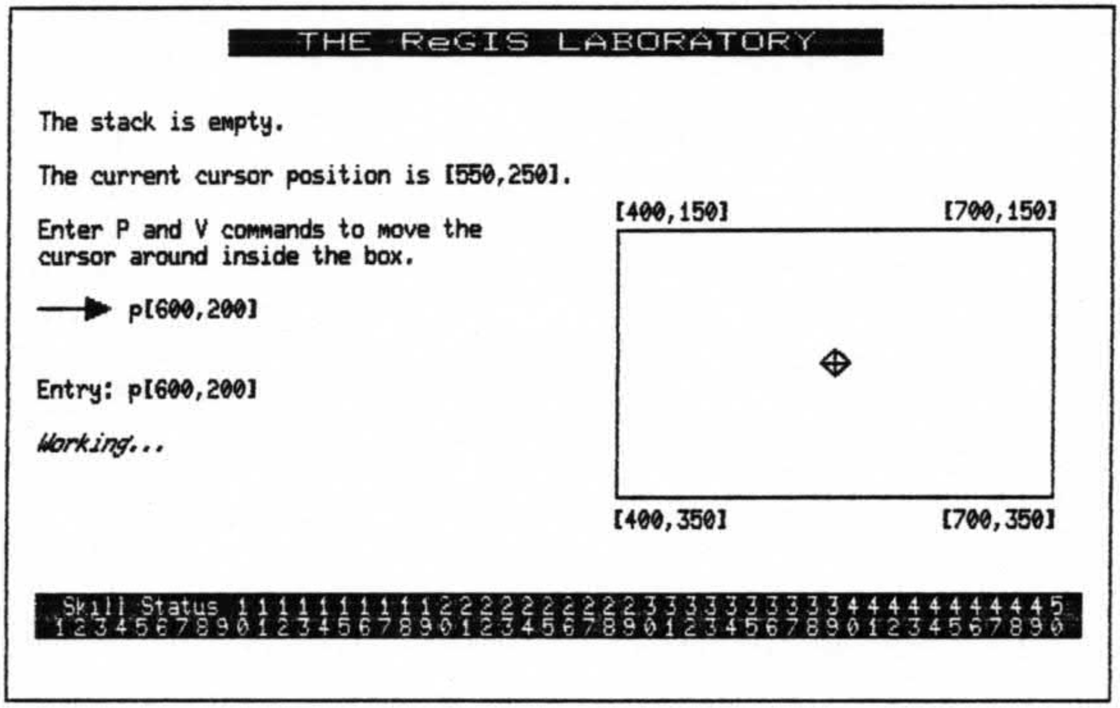



Figure 10

PARSING BEGINS

The system repeats the command and prints "Working..." to indicate that it has begun parsing.

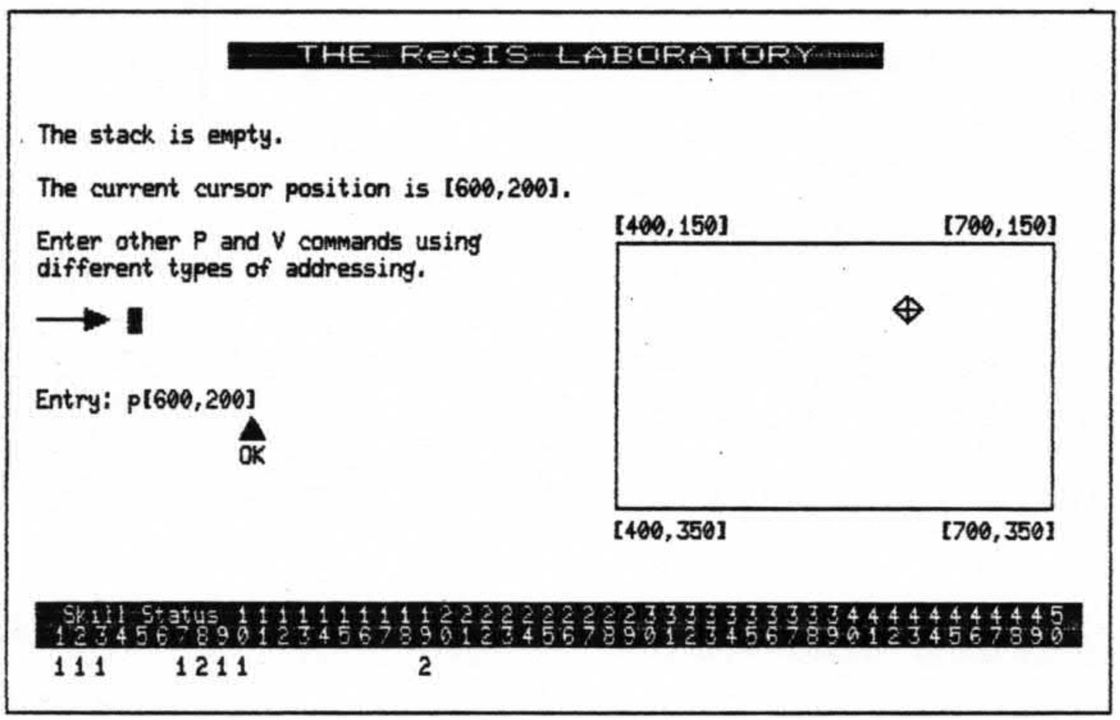



Figure 11

PARSING COMPLETE

The system displays an up arrow where parsing terminated, executes the student's command in the ReGIS window, renews the stack and position messages, updates the student model, prints "OK" to complete the problem, and displays new directions.

7. Can interpret the standard [x,y] address format.

9. Understands defaults.

1Ø. Given the current cursor position as [xc,yc], knows the meaning of [x] -> [x,yc].

These actions reflect the skill hierarchy shown in Appendix B (see Section 6.2 in AI/CAI Project Progress Report No. 2 for a full explanation of this directed graph). When viewing them in this context, it appears questionable that the system should make decisions concerning Skills 9 and 1Ø for this student entry. This situation clearly demonstrates the power and flexibility of the rule-based approached, because it indicates that the student model is in need of "fine tuning" to reflect the task analysis more accurately. Since all student model updates are governed by production rules, the fine tuning can be easily accomplished without requiring substantial reprogramming. I plan to make such an adjustment when I convert the courseware to VAX/VMS LISP.

## 4.6 Processing a Complex Student Entry

Figure 12 shows a complex student entry to the ReGIS Lab:

```
v(b)[+5Ø,+1ØØ][45Ø](e)
```

This is a four-part ReGIS Vector command:

- (b) pushes the current cursor position onto the stack.
- [+5Ø,+1ØØ] draws a vector from the current cursor position to the position 5Ø pixels to the right and 1ØØ pixels down.
- [45Ø] draws a vector from the current cursor position to the position with an X coordinate of 45Ø and a Y coordinate the same as that of the current cursor position.
- (e) pops an address off the stack and draws a vector from the current cursor position to that address.

Parsing begins when the student presses RETURN, and Figure 13 shows the screen after the first component has been parsed:

- The student's entry has been copied and the original entry and directions erased.
- The "Working..." message has been printed to indicate that parsing is in progress.

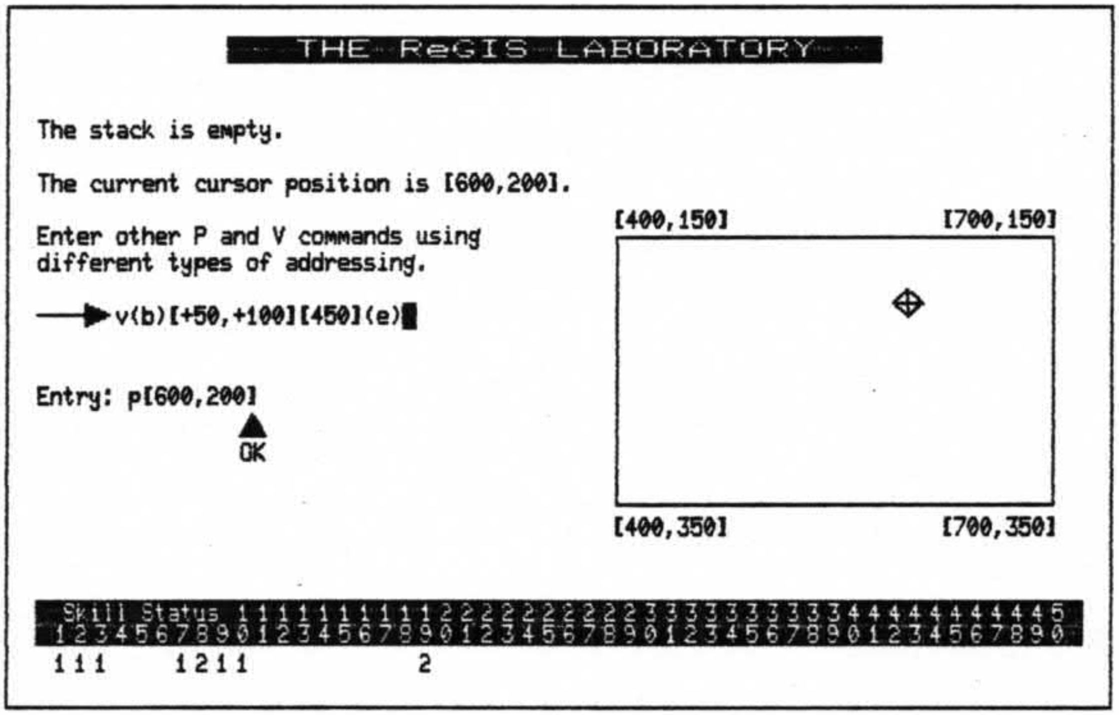



Figure 12

A COMPLEX ReGIS "VECTOR" COMMAND ENTRY

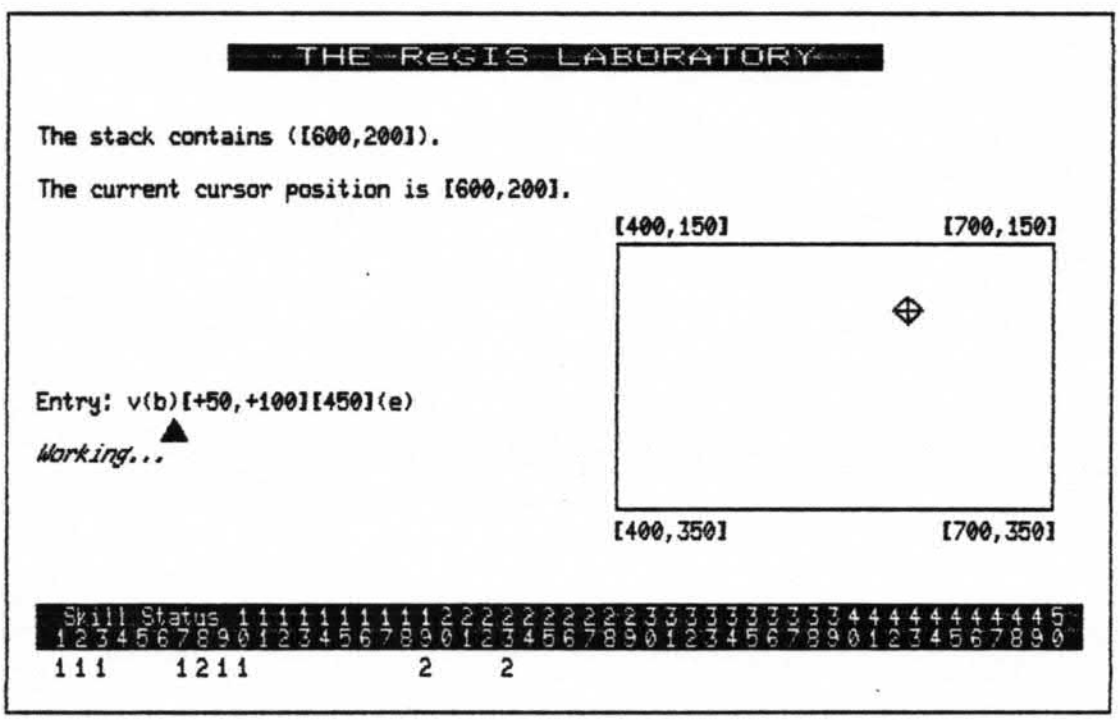



Figure 13

STACK PUSH

The system displays an up arrow where parsing terminated, renews the stack and position messages, updates the student model, and continues parsing.

- An arrowhead has been positioned under the character to which parsing proceeded thus far.

- The stack contents message at the top left of the screen has updated to indicate that [600,200] has been pushed onto the stack.

- A value of +2 has been assigned to Skill 23 ("can store addresses on the stack with (B)") at the bottom of the screen.

Parsing continues in Figure 14:

- The arrowhead has been moved over to indicate that parsing has proceeded through the second address in the command.

- The results of executing the command so far are shown in the graphic area at the right of the screen, drawing a vector from the current cursor position ([600,200]) to a position 50 pixels to the right and 100 pixels down ([650,300]).

- The current cursor position message at the top left of the screen has been updated.

- A value of +2 has been assigned to Skill 14 ("given the current cursor position as [xc,yc], knows the meaning of [+x,+y] -> [xc+x,yc+y]") and a value of +1 has been assigned to Skill 13 ("understands relative addresses").

Parsing continues in Figure 15:

- The arrowhead has been moved over to indicate that parsing has proceeded through the third address in the command.

- The results of executing the third part of the command are shown in the graphic area, drawing a vector from the current cursor position ([650,300]) to the position with an X coordinate of 450 and a Y coordinate the same as that of the current cursor position ([450,300]).

- The current cursor position message at the top left of the screen has been updated.

- A value of +2 has been assigned to Skill 10 ("given the current cursor position as [xc,yc], knows the meaning of [x] -> [x,yc]").

  Note the difference between assigning a value of +2 to Skill 10 here versus assigning a value of +1 to Skill 10 in Figure 11 as described in Section 4.5

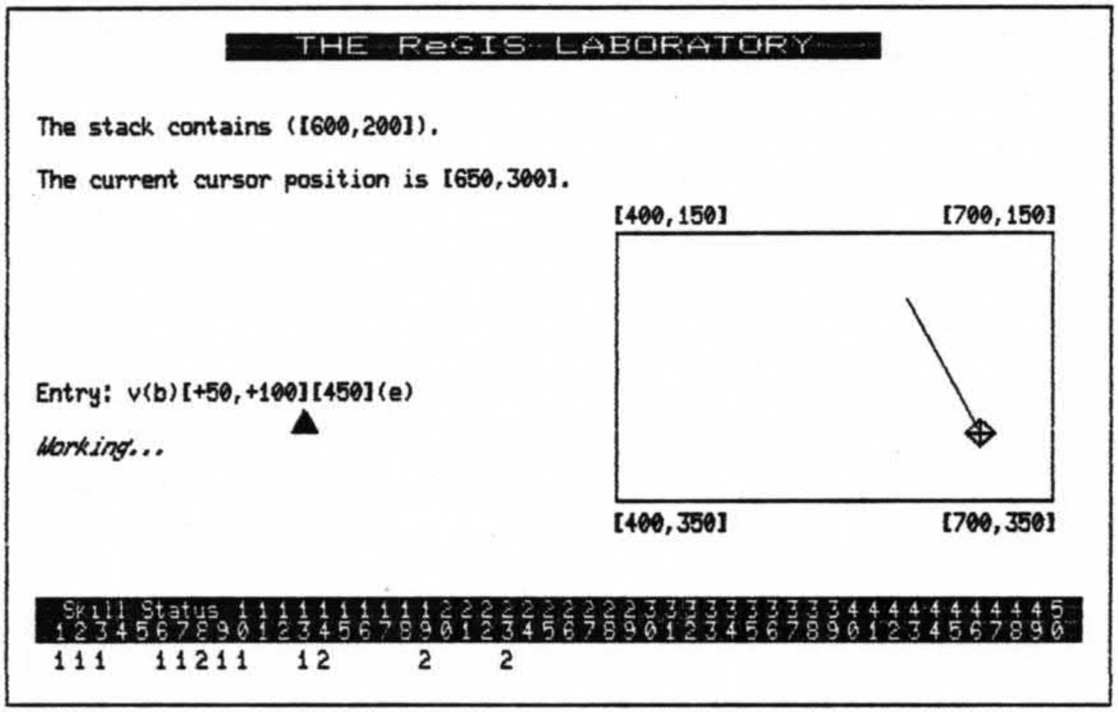



Figure 14

RELATIVE X AND RELATIVE Y ADDRESSING

The system displays an up arrow where parsing terminated, executes the student's command in the ReGIS window, renews the stack and position messages, updates the student model, and continues parsing.

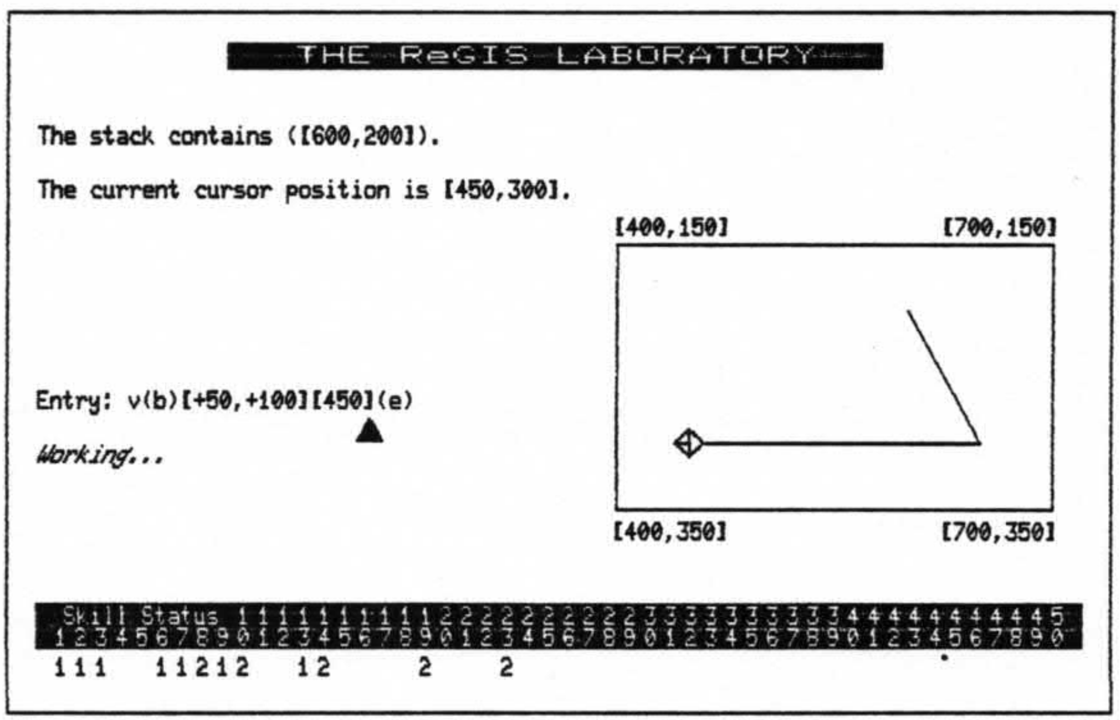



Figure 15

ABSOLUTE X AND DEFAULT Y ADDRESSING

The system displays an up arrow where parsing terminated, executes the student's command in the ReGIS window, renews the stack and position messages, updates the student model, and continues parsing.

above. Here defaults are used explicitly, yielding greater confidence that the student knows how to use defaults. In the previous example a value of +1 was assigned because the student used a higher level skill for which Skill 1Ø is a prerequisite.

Parsing continues in Figure 16:

- The arrowhead has been moved over to indicate that parsing has proceeded through the fourth address in the command.
- The results of executing the fourth part of the command are shown in the graphic area, drawing a vector from the current cursor position ([45Ø,3ØØ]) to the position popped off the top of the stack ([6ØØ,2ØØ]).
- The stack contents message at the top left of the screen has updated to indicate that [6ØØ,2ØØ] has been popped off the stack.
- The current cursor position message at the top left of the screen has been updated.
- New directions are printed to indicate that the system is ready for the next student entry.
- A value of +2 has been assigned to Skill 24 ("can pop addresses off the stack with (E)") at the bottom of the screen.

## 4.7 Storing Student Data

After Figure 16 had been displayed, the student pressed the EXIT (PF3) key. This caused the data on his work to be stored, and the message in Figure 17 to be displayed. Once the data storage operation is complete, the screen changes automatically to display the form shown in Figure 18. By pressing RETURN at this point, the student reentered the registration phase shown in Figure 19. The student entered his name again, and reentered the ReGIS Laboratory as shown in Figure 2Ø. In this case, however, the student model reflects its state when the student exited the course.

## 4.8 Processing an Incorrect Student Entry

The student now enters the incorrect entry shown in Figure 21. The problem with this entry is that the "[" in position 7 of the entry should be a "]".

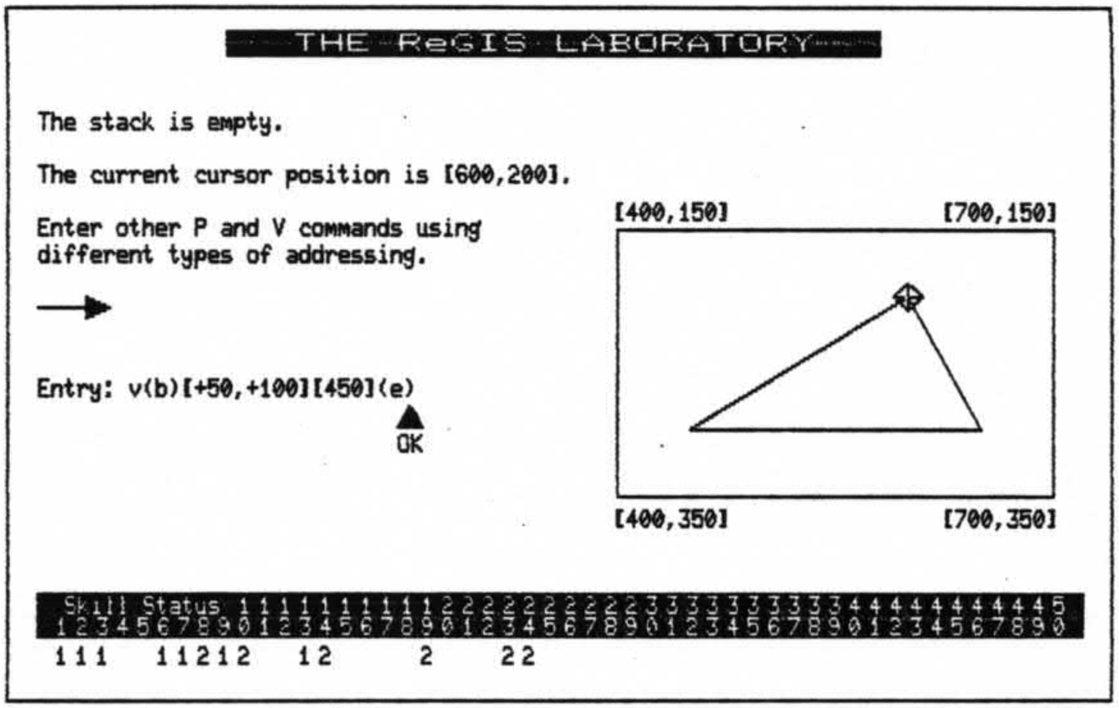



Figure 16

STACK POP

The system displays an up arrow where parsing terminated,
executes the student's command in the ReGIS window,
renews the stack and position messages, updates the student model,
prints "OK" to complete the problem, and displays new directions.

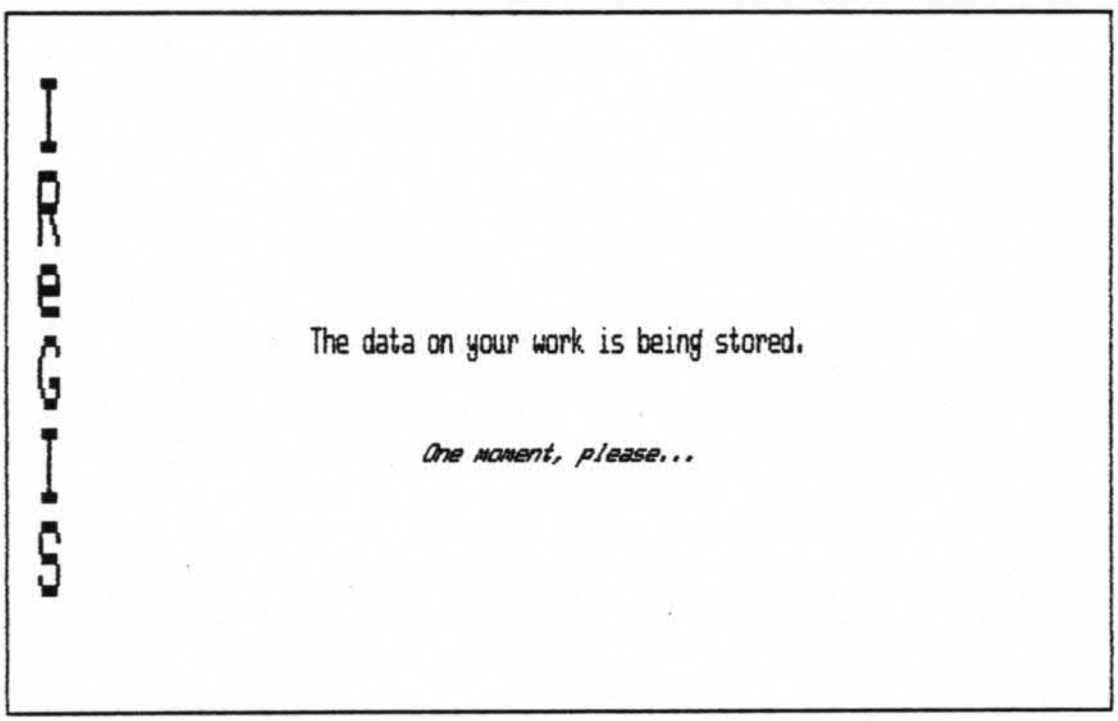



Figure 17

DATA STORING IN PROGRESS MESSAGE

The system stores the student's data
after he presses the EXIT key in the lab.


Figure 18

DATA STORING COMPLETED MESSAGE

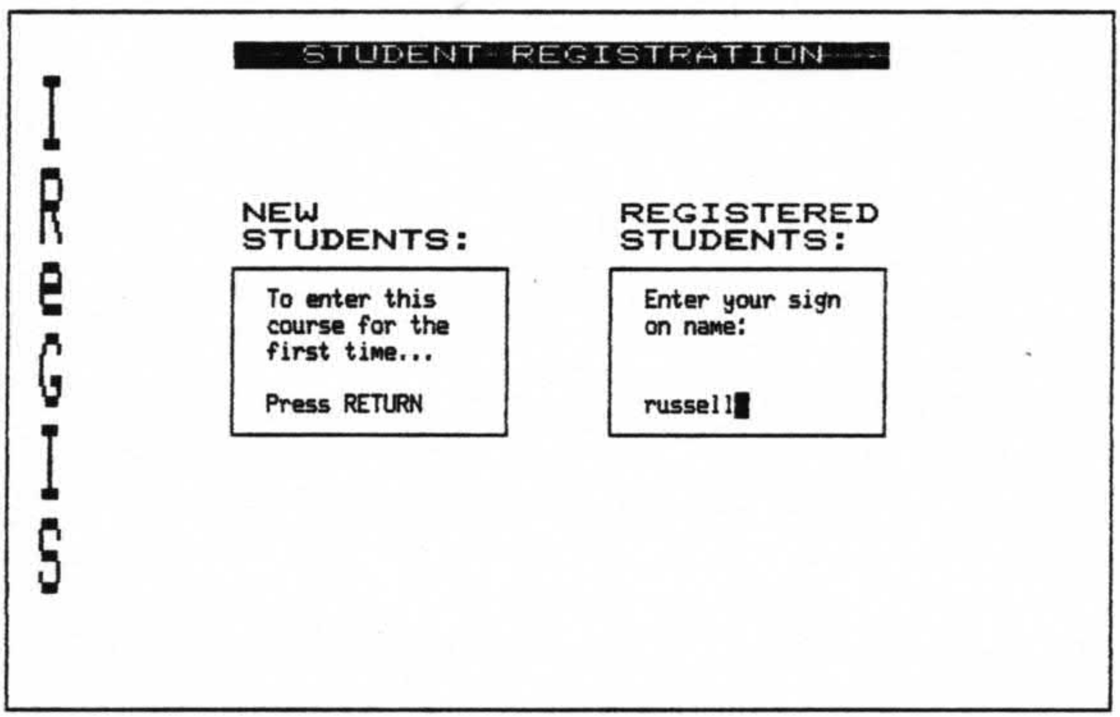



Figure 19

REDISPLAY OF THE REGISTRATION FORM

The student enters the name he registered with previously.

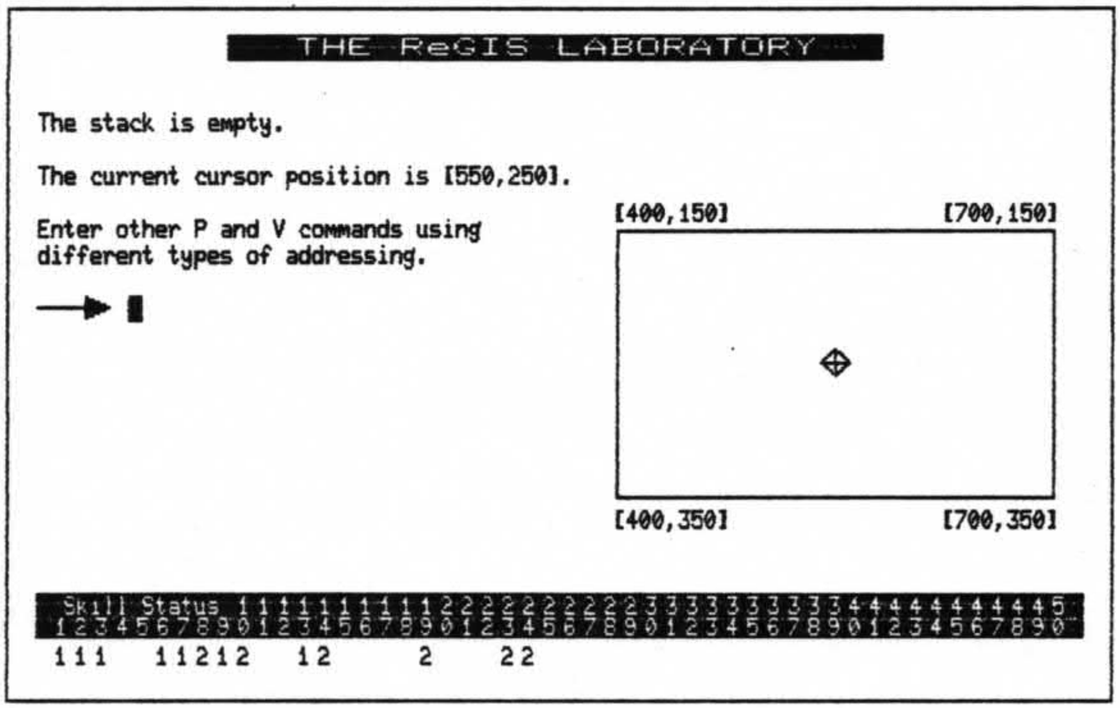



Figure 20

THE INITIAL STATE OF THE ReGIS LABORATORY
FOR PREVIOUSLY REGISTERED STUDENTS

Note that the previous student model values are retained.

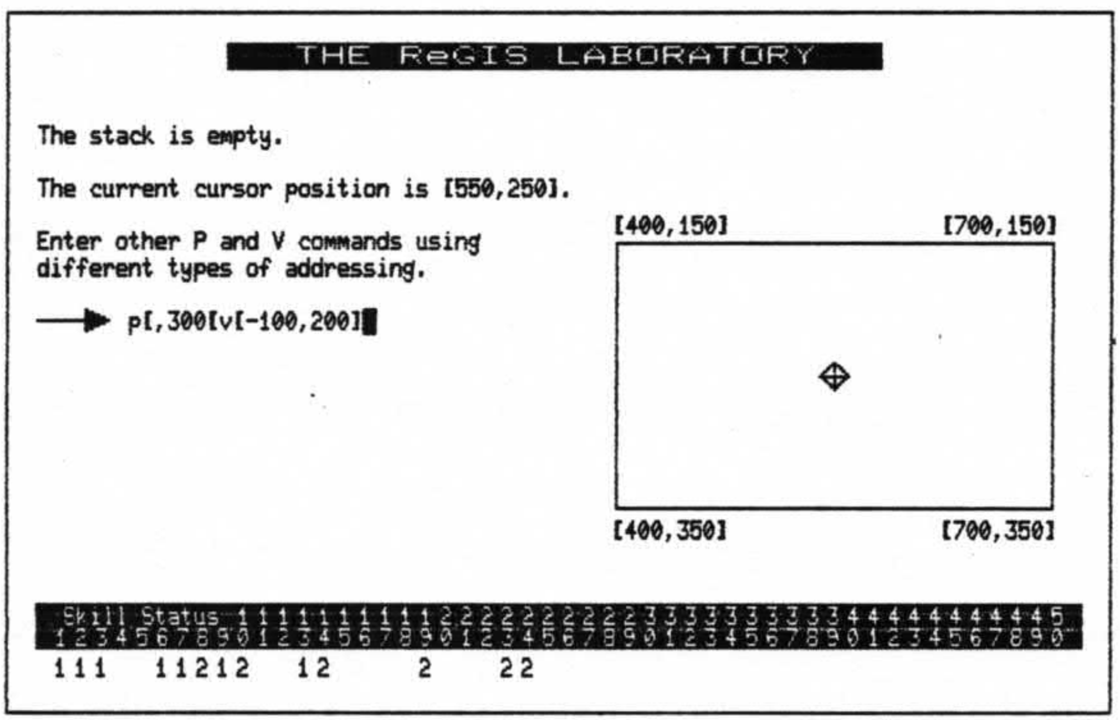



Figure 21

A COMBINATION "POSITION" AND "VECTOR" COMMAND THAT INCLUDES AN ERROR

The [ in position 7 should be a ].

Parsing begins when the student presses the RETURN key. The student's entry is copied as shown in Figure 22 and the "Working..." message is displayed.

The error is detected in Figure 23:

- The "Working..." message is erased to indicate that parsing has stopped.
- The arrowhead is displayed, indicating the point at which the error was found and parsing stopped.
- The word "ERROR:" is displayed to indicate that an error has been detected, and an explanatory error message is printed.
- The student's original entry is erased to make room for him to enter a new command.
- The student model at the bottom of the screen is updated as explained below.

In this case, a value of -2 is assigned to Skill 11 ("given the current cursor position as [xc,yc], knows the meaning of [,y] -> [xc,y]") and Skill 17 ("given the current cursor position as [xc,yc], knows the meaning of [+x,y] -> [xc+x,y]") because the student's entry indicates incorrect usage of these skills. A value of -1 is assigned to all skills that require either of these skills as prerequisites. (Negative skill values are indicated in the Skill Status display by underlining due to space limitations.)

The student corrected his mistake in Figure 24. Pressing the RETURN key initiated parsing in Figure 25. Figure 26 shows the state of the screen after the first command component is parsed. In addition to the overall screen updates identified for previous Lab displays, note that the value of Skill 11 has been changed from -2 to +2, but that the -1 values assigned to all postrequisite skills remain unchanged. Figure 27 shows the state of the screen after the second command component is parsed. Note that the value of Skill 17 has been changed from -2 to +2.

This two-part command reverses part of the damage done by the incorrect command entry in Figure 21, but all of the -1 values assigned to postrequisite skills remain. This feature allows the student model to "zero in" on the student's precise skill level. As before, all of these actions are governed by easily changed rules, so fine tuning the courseware to reflect the "real" skill hierarchy more accurately is not difficult.

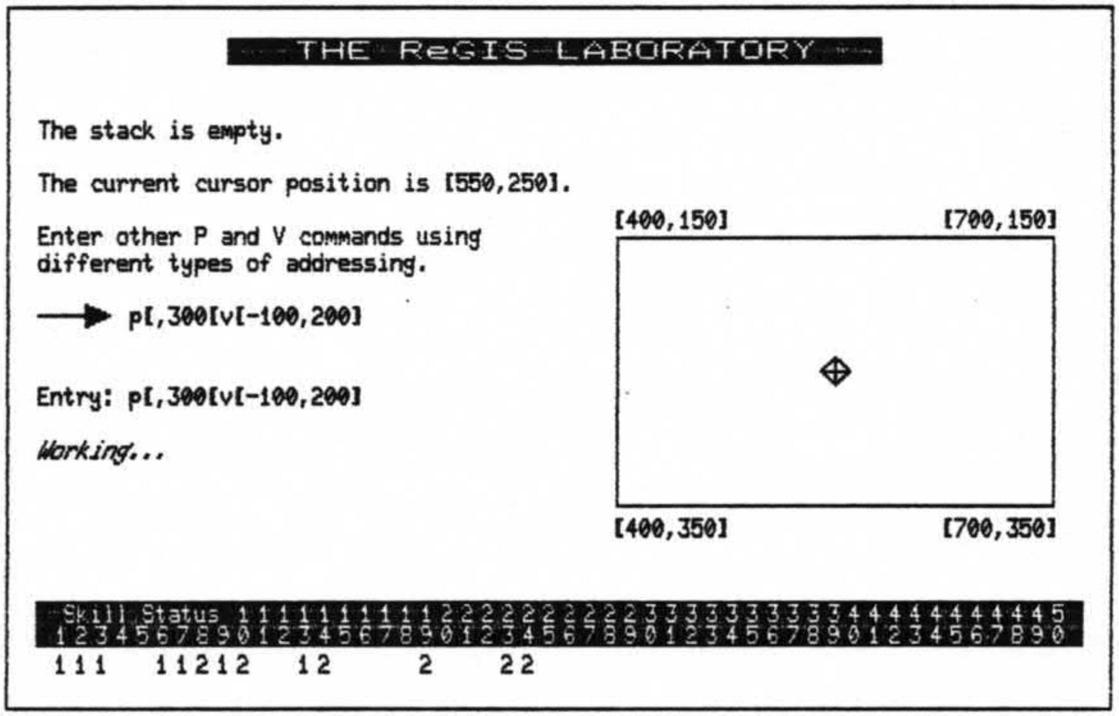



Figure 22

PARSING OF THE ERRONEOUS ENTRY BEGINS

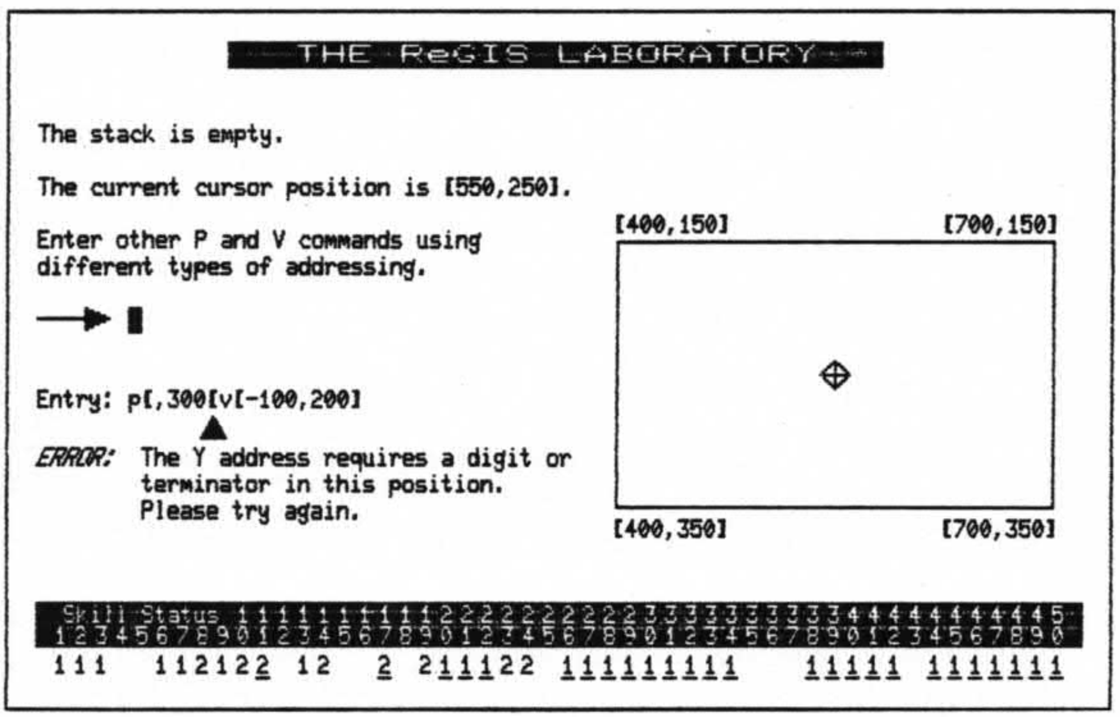



Figure 23

ERROR DETECTED

The system displays an up arrow where the error occurred, prints an error message, updates the student model, and displays new directions. (Underlined values in the student model are negative.)

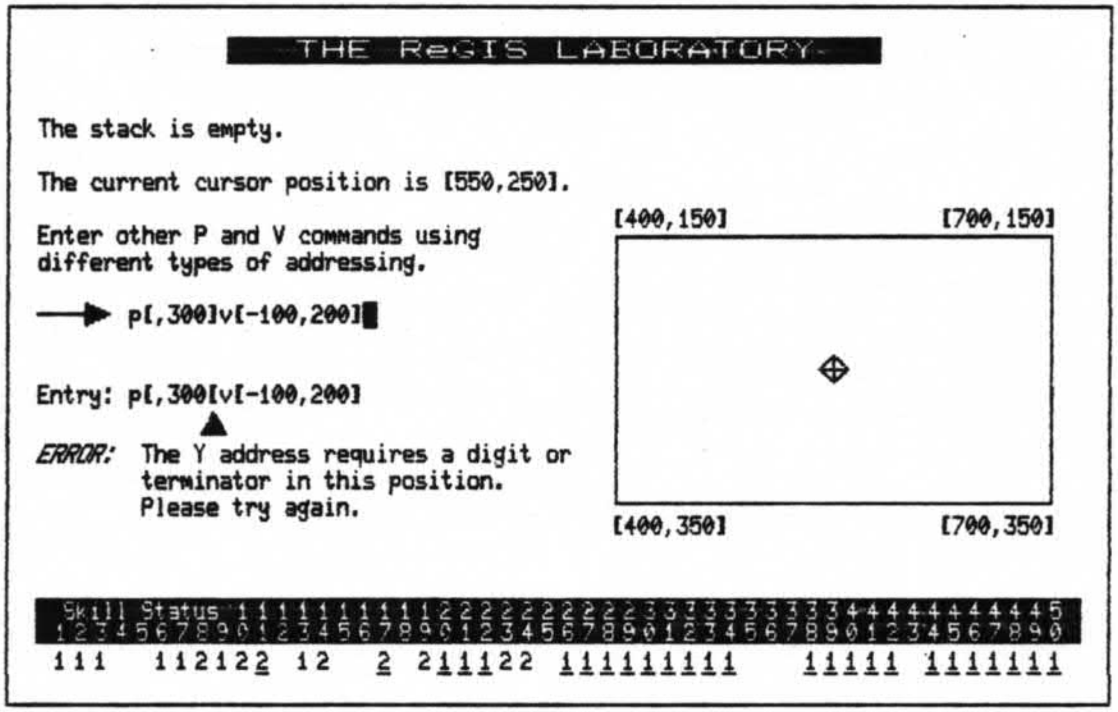



Figure 24

CORRECTED ENTRY

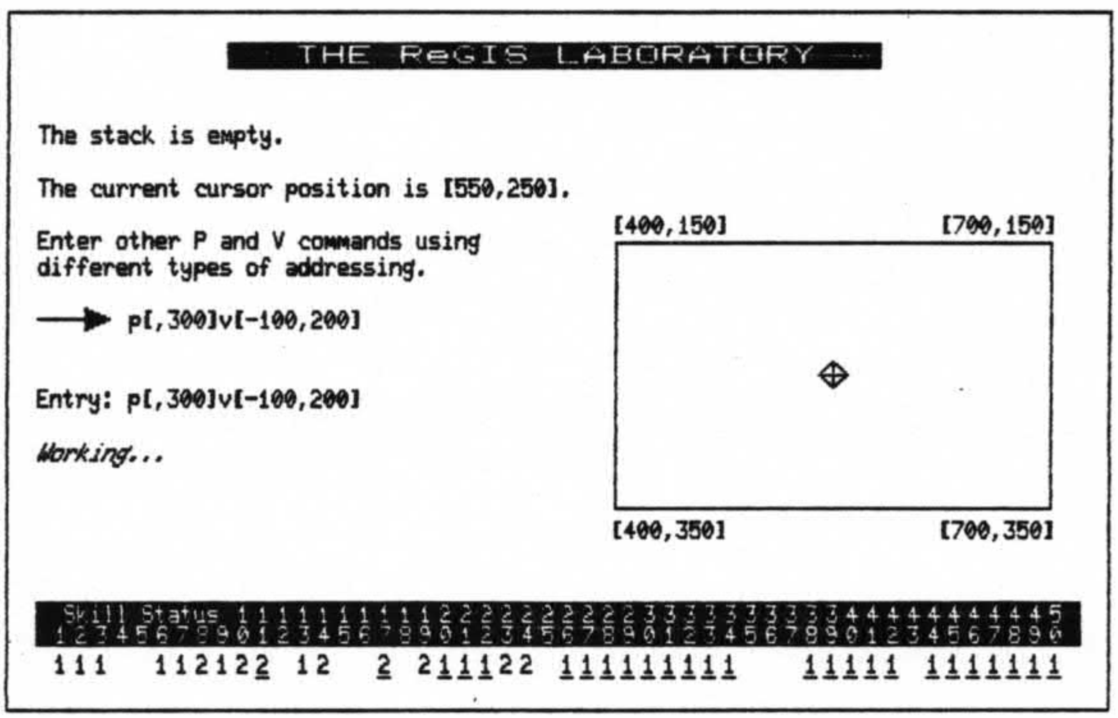



Figure 25

PARSING OF CORRECTED ENTRY BEGINS

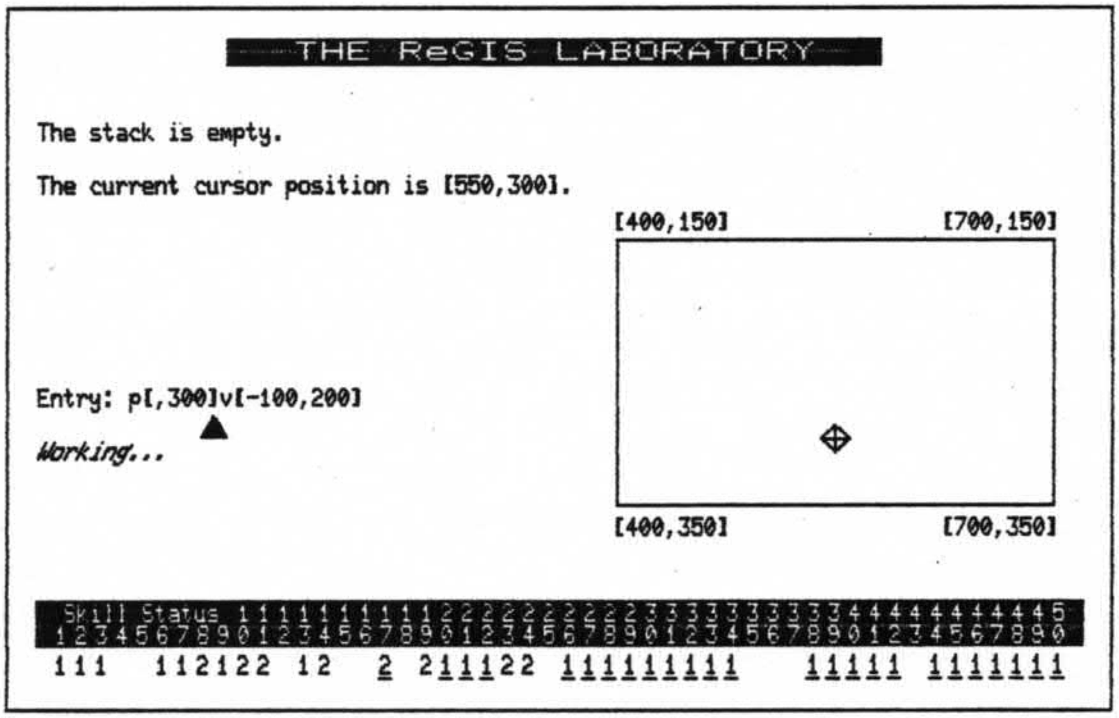



Figure 26

DEFAULT X AND ABSOLUTE Y ADDRESSING

The system displays an up arrow where parsing terminated, executes the student's command in the ReGIS window, renews the stack and position messages, updates the student model, and continues parsing. Note that the value of Skill 11 has been changed from -2 to +2.

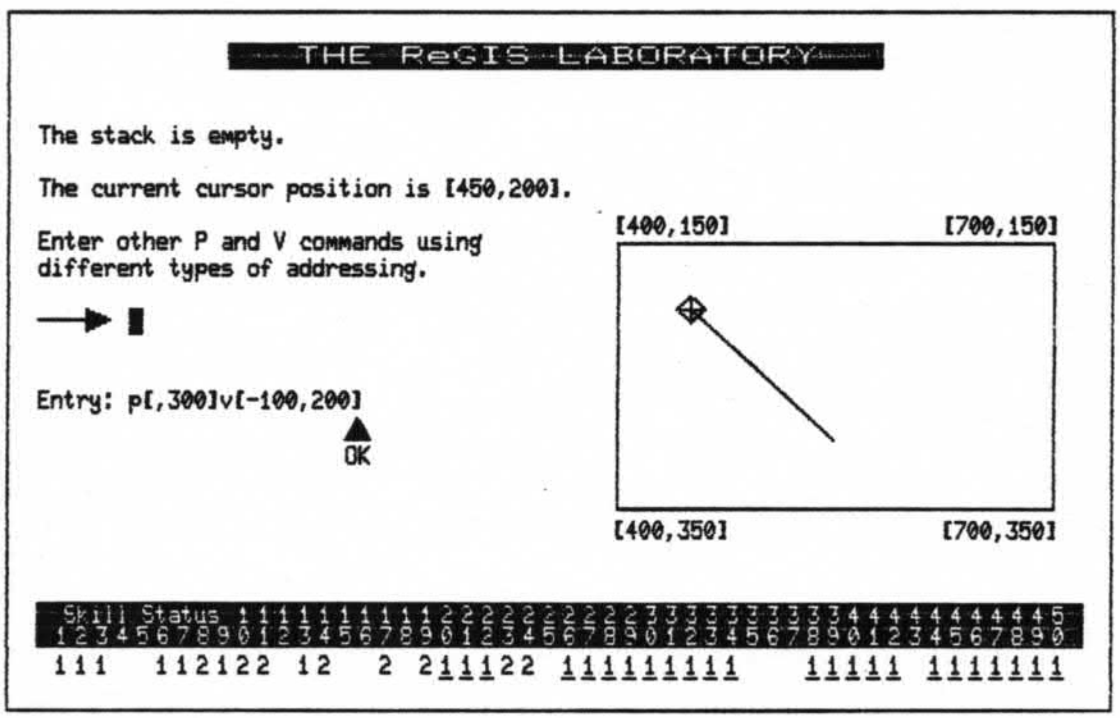



Figure 27

RELATIVE X AND DEFAULT Y ADDRESSING

The system displays an up arrow where parsing terminated, executes the student's command in the ReGIS window, renews the stack and position messages, updates the student model, prints "OK" to complete the problem, and displays new directions. Note that the value of Skill 17 has been changed from -2 to +2.

## 4.9 Miscellaneous Program Functions

**4.9.1 Pressing the HELP Key** - Figure 28 shows the result of pressing the keypad HELP key (PF2). The screen moves up to make room for an extra line at the bottom, and a short help message is printed on the newly opened line. The orientation message, "The ReGIS Laboratory," is not visible at the top of the screen while the help message is being displayed. However, pressing any key at this point causes the help message to be erased, the screen to move back down restoring the orientation message at the top, and the program to resume at the point where the HELP key was pressed. In this implementation, the help message shown in Figure 28 was printed whenever the HELP key was pressed, regardless of the context.

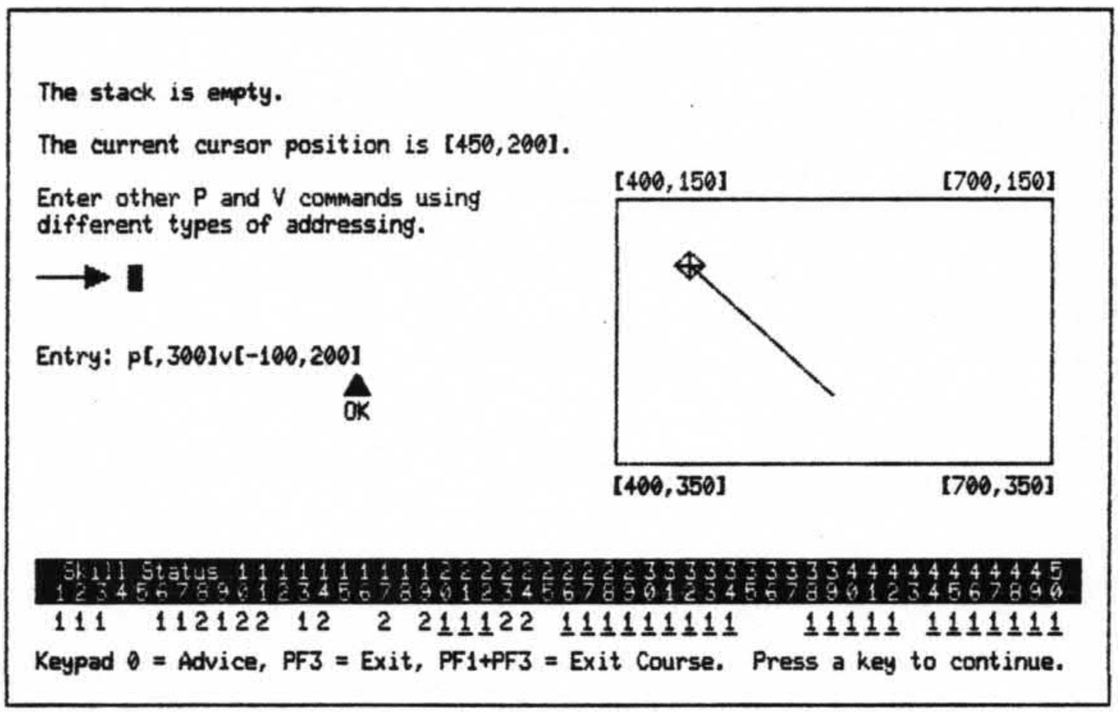



Figure 28

RESULT OF PRESSING THE HELP KEY

Note that the screen moves up to make room for the message.

**4.9.2 Pressing the ADVICE Key** - Figure 29 shows the result of pressing the keypad ADVICE key (0). As in the HELP routine, the screen moves up to make room for an extra line at the bottom, and the orientation message, "The ReGIS Laboratory," is not visible at the top of the screen while the advice message is being displayed. Unlike the HELP routine, however, advice messages in this implementation were context sensitive. The message shown in Figure 29 was the standard advice message for the ReGIS Lab, but other messages were displayed if ADVICE was pressed in, for example, the registration routines. Pressing any key at this point causes the advice message to be erased, the screen to move back down restoring the orientation message at the top, and the program to resume at the point where the ADVICE key was pressed.

**4.9.3 Pressing the GOLD + QUIT Keys** - Pressing the QUIT key alone caused the student to exit the ReGIS Lab and data on his work to be stored as discussed in Section 4.7 above. Pressing the GOLD (PF1) key followed by the QUIT (PF3) key caused the program to terminate immediately. As shown in Figure 30, this action caused the message QUIT* to be printed (by the LISP interpreter, not the IReGIS program) signifying return to LISP top level.

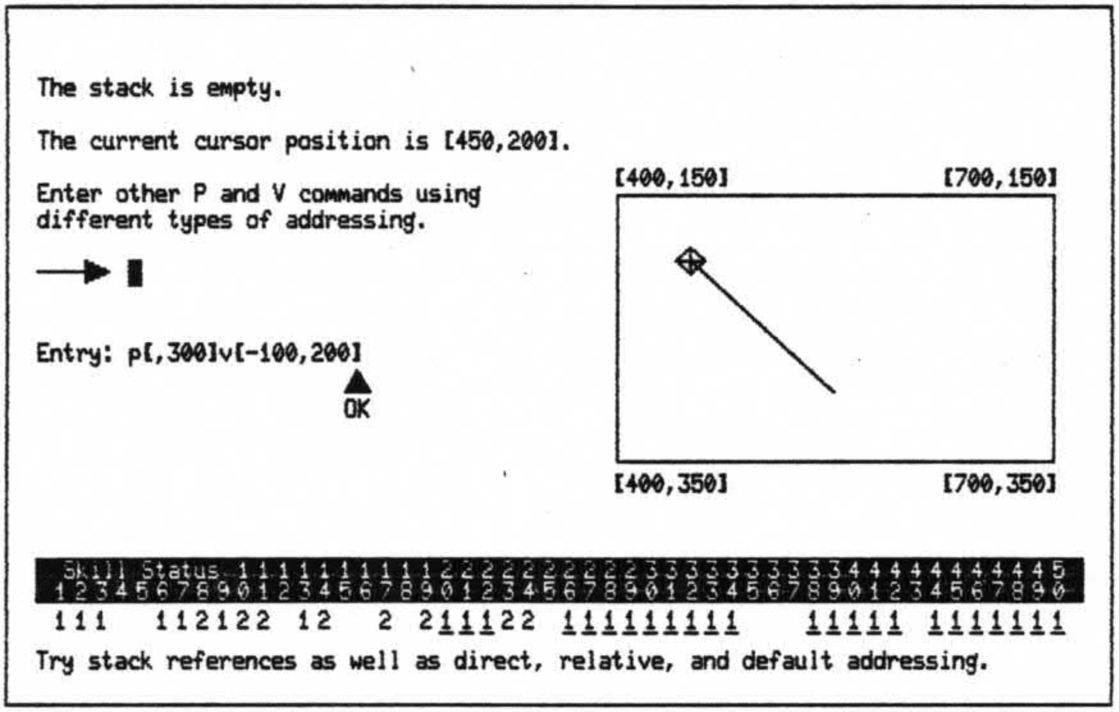



Figure 29

RESULT OF PRESSING THE ADVICE KEY

Note that the screen moves up to make room for the message.

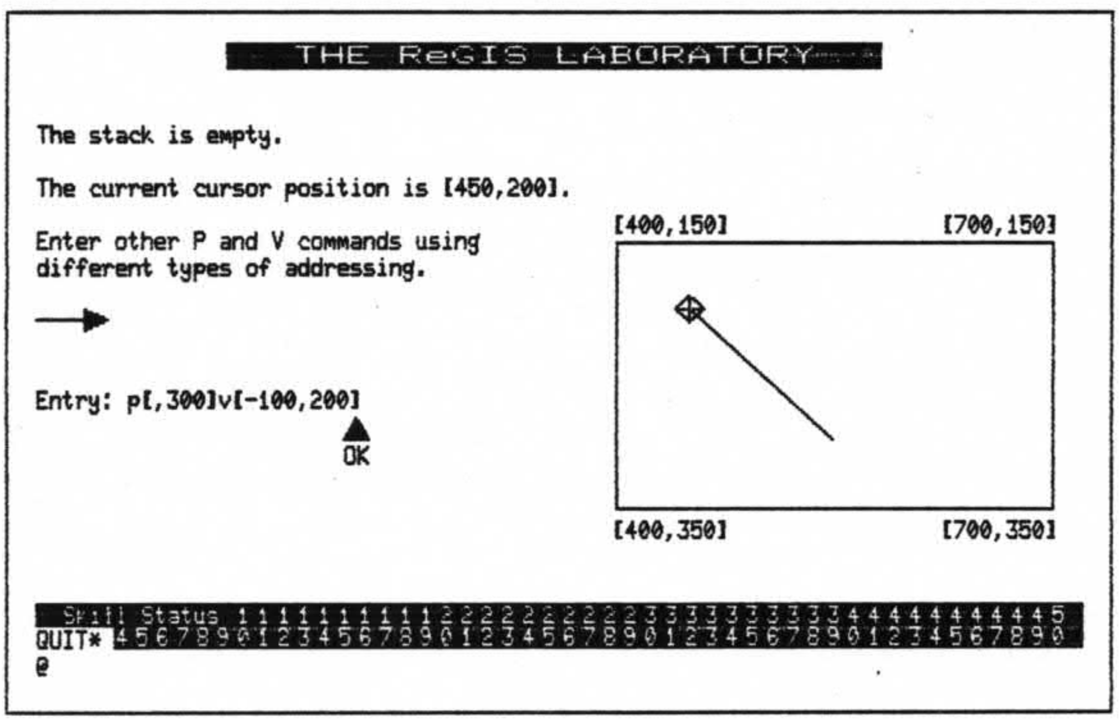



Figure 30

RESULT OF PRESSING GOLD + QUIT KEY SEQUENCE

The program terminates immediately without saving the student's data.

## 5.Ø CRITIQUE AND CONCLUSIONS

As described at the end of Section 4.5, this first implementation has already identified areas in which the student model is in need of "fine tuning". The production rule approach, however, provides all the power and flexibility needed to achieve this fine tuning quickly and easily. It therefore appears to be a very worthwhile technique to pursue for creating intelligent CAI tutors.

One conclusion from AI/CAI Project Progress Report No. 3 warrants repeating here: while the production rule formalism is highly efficient for expressing the prerequisite relationships between component skills, use of this formalism to update the student model after each response is relatively inefficient. The main inefficiency stems from processing the rules repeatedly to find all of the prerequisites or postrequisites for the skill on which the student is currently working. This problem attacked in the implementation described here by preprocessing the rules to find all of the prerequisites and postrequisites for each skill and the storing these as lists in a simple array. This approach allowed the student model to be updated much more quickly without sacrificing the elegance of the rule-based strategy.

This initial implementation has also shown that LISP can be used as an effective CAI authoring language, since I have been able to implement virtually all of Educational Services' normal CAI features without going outside the language. These features include:

- separation of lesson logic and display logic by storing screen displays in a random access file and calling them up when desired,
- implementation of a CAI keypad including HELP and ADVICE functions,
- implementation of single character input to allow full control of student input processing,
- student registration and data storage, and
- a sophisticated student response parser.

## 6.Ø RESEARCH EFFORTS TO BE PURSUED

As one would expect, my current plans have not changed substantially since my last Project Progress Report. I have, however, now made contact with Gary Brown and obtained a copy of VAX-11 LISP Version XØ.21, and will be using this LISP to try to implement the course described in AI/CAI Project Progress Report No. 3. I reiterate that I have no intention of trying to implement the entire course described in that document, but that I expect to get far enough to encounter and tackle all of the major implementation problems and to demonstrate functionality at a reasonable level of sophistication.

## 7.0 Appendix A: TASK REPRESENTATION AS A LIST OF SKILLS

See Section 6.1 of AI/CAI Project Progress Report No. 3 for a full discussion of this list.

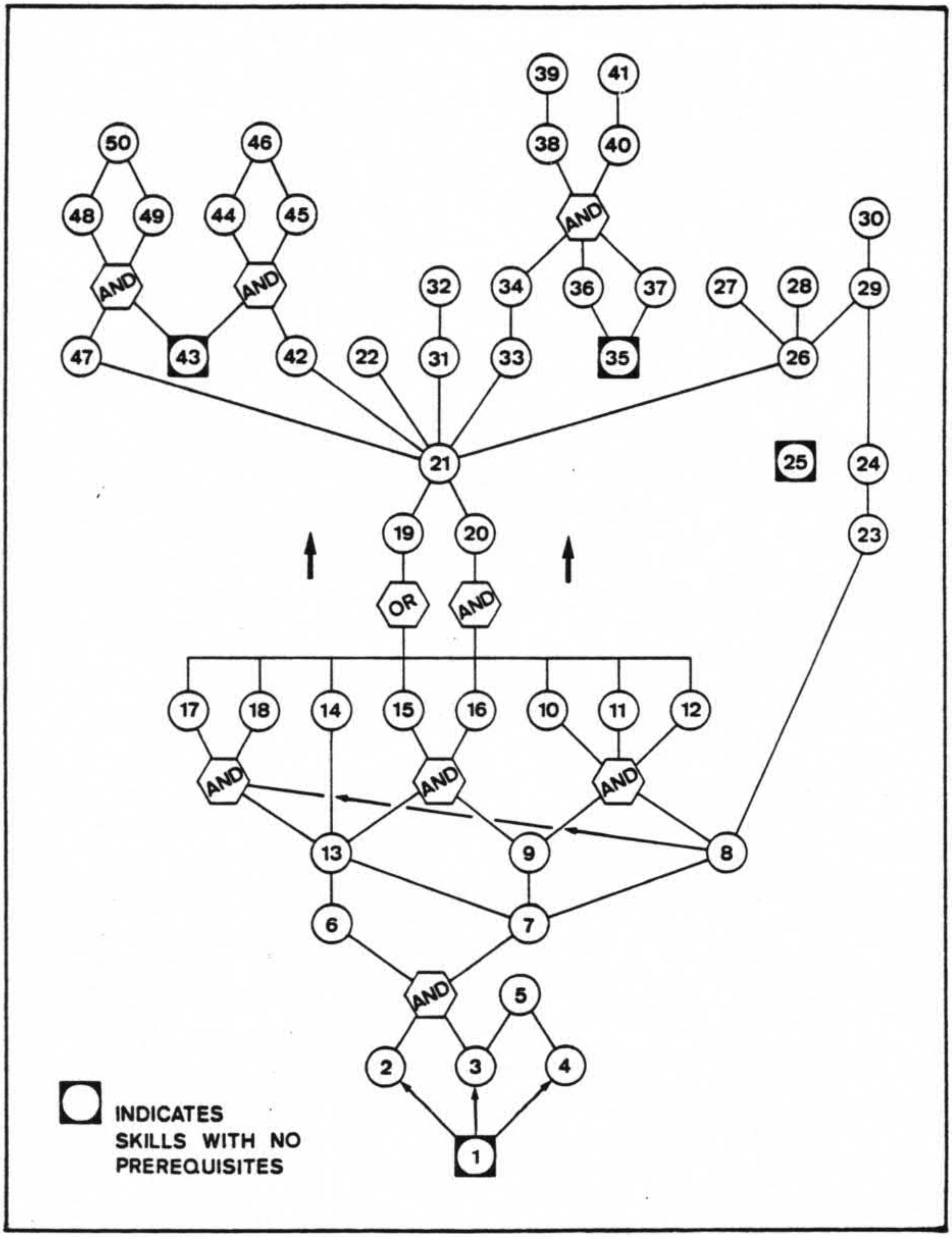
| Number | Description |
|---|---|
| 1 | Recognizes screen as a rectangular dot matrix. |
| 2 | Can translate screen positions into (x,y) pairs. |
| 3 | Can translate (x,y) pairs into screen positions. |
| 4 | Knows absolute screen limits (767,479). |
| 5 | Knows that initial cursor position is upper left-hand corner of screen. |
| 6 | Understands the concept of current cursor position. |
| 7 | Can interpret the standard [x,y] address format. |
| 8 | Can specify absolute screen addresses in [x,y] format. |
| 9 | Understands defaults. |
| | Given the current cursor position as [xc,yc], knows the meaning of: |
| 10 | [x] -> [x,yc] |
| 11 | [,y] -> [xc,y] |
| 12 | [] -> [xc,yc] |
| 13 | Understands relative addresses. |
| | Given the current cursor position as [xc,yc], knows the meaning of: |
| 14 | [±x,±y] -> [xc±x,yc±y] |
| 15 | [±x] -> [xc±x,yc] |
| 16 | [,±y] -> [xc,yc±y] |
| 17 | [±x,y] -> [xc±x,y] |
| 18 | [x,±y] -> [x,yc±y] |
| 19 | Can use the basic P command to position the cursor. |
| 20 | Knows the current cursor position after execution of a P command. |
| 21 | Can use all addressing schemes in P commands. |
| 22 | Can work with P command aberrations such as P [x1,y1] [x2,y2] ... [xn,yn]. |
| 23 | Can store addresses on the stack with (B). |
| 24 | Can pop addresses off the stack with (E). |
| 25 | Can use the S(E) command to erase the screen. |
| 26 | Can use the basic V command to draw a vector. |
| 27 | Knows the current cursor position after execution of a V command. |
| 28 | Can draw multiple vectors with a single V command. |
| 29 | Can draw closed polygons using (B) and (E) in V commands. |

(continued...)

| Number | Description |
|---|---|
| 3Ø | Knows the terms "open" and "closed" as applied to figures. |
| 31 | Can use the basic C command to draw circles. |
| 32 | Knows the current cursor position after drawing a circle with the basic C command. |
| 33 | Can use the C(C) command to draw circles. |
| 34 | Knows the current cursor position after drawing a circle with the C(C) command. |
| 35 | Understands angle measure in degrees. |
| 36 | Can translate angle measures into screen angles. |
| 37 | Can translate screen angles into angle measures. |
| 38 | Can use the C(A<degrees>)[X,Y] command to draw arcs starting a [X,Y] and using the current cursor position as the center. |
| 39 | Knows the current cursor position after drawing an arc with the C(A<degrees>)[X,Y] command. |
| 4Ø | Can use the C(A<degrees>C)[X,Y] command to draw arcs starting at the current cursor position and using [X,Y] as the center. |
| 41 | Knows the current cursor position after drawing an arc with the C(A<degrees>C)[X,Y] command. |
| 42 | Can draw open curves with the C(S) command. |
| 43 | Understands interpolation. |
| 44 | Can use [] at the front of a C(S) command. |
| 45 | Can use [] at the end of a C(S) command. |
| 46 | Knows the current cursor position after execution of a C(S) command. |
| 47 | Can draw closed curves with the C(B) command. |
| 48 | Can use [] at the front of a C(B) command. |
| 49 | Can use [] at the end of a C(B) command. |
| 5Ø | Knows the current cursor position after execution of a C(B) command. |

## 8.Ø Appendix B: TASK REPRESENTATION AS A DIRECTED GRAPH

See Section 6.2 of AI/CAI Project Progress Report No. 3 for a full discussion of this graph.



## AI/CAI PROJECT PROGRESS REPORTS IN THIS SERIES

These reports are available from Jesse Heines, Educational Services Development and Publishing, Bedford BUO/E32, DTN 249-4339, Engineering Net CLOSUS::HEINES.

These reports are intended for Digital internal distribution only. Versions approved for external publication and distribution have been prepared for some of these reports and are so identified below. If you intend to redistribute any of these reports to non-Digital personnel, please request the appropriate external versions.

1. Basic Concepts in Knowledge-Based Systems. April 20, 1982.

   External Version: Basic Concepts in Knowledge-Based Systems, published in Machine-Mediated Learning, 1(1):65-95, Spring 1983. Also available as Educational Services Technical Report No. 13.

2. Where AI Can Fit in CAI. November 4, 1982.

3. The Design of a Prototype AI/CAI Course Employing a Rule-Based Tutorial Strategy. (Coauthored with Tim O'Shea of The Open University, Milton Keynes, England.) May 3, 1983.

   External Version: The Design of a Prototype AI/CAI Course Employing a Rule-Based Tutorial Strategy. (Coauthored with Tim O'Shea.) Available as Educational Services Technical Report No. 14.

4. An Initial Attempt at Building a Student Model Using Production Rules. June 2, 1983.

Readers involved with AI languages may also be interested in "Logic and Recursion: The PROLOG Twist," coauthored with Jonathan Briggs and Richard Ennals of Imperial College, London (May, 1982). This paper is available as Educational Services Technical Report No. 15.