An instructional monitor is a program which tries to detect, diagnose, and possibly help overcome a student's learning difficulties in the course of solving a problem or performing a task. In one approach to building an instructional monitor, the student uses a special task- or problem-oriented language expressly designed around some particular class of problems. Correspondingly, the diagnostic programs in this "special purpose" type of monitor system often utilize information that is specific to the kind of problem being studied. In the SIMON system a different kind of experimental approach has been taken. The student addresses SIMON in an easy and very general programming language rather than a special task language. Using SIMON, students construct programs for systems or processes which can represent vastly different situations whether from mathematics, biology, physics, engineering, or elsewhere. The student tests his program against a "true" program provided to SIMON by an instructor. At the student's request, SIMON tests his program against its "true" model to determine if it works. If not, SIMON points out where and why it does not.
SIMON - A Simple Instructional Monitor

Wallace Feurzeig and Paul M. Wexelblat
Bolt Beranek and Newman Inc.
50 Moulton Street
Cambridge, Massachusetts 02138

Ronald C. Rosenberg
Dept. of Mechanical Engineering
Michigan State University
East Lansing, Michigan 48823

August 1, 1970

Sponsored by Personnel and Training Research Programs
Psychological Sciences Division
Office of Naval Research
Arlington, Virginia 22217

Contract No. N00014-69-C-0127
NR No. 154-285

This document has been approved for public release and sale; its distribution is unlimited.

Reproduction in whole or in part is permitted for any purpose of the United States Government.
### SIMON - A Simple Instructional Monitor

An instructional monitor is a program which tries to detect, diagnose, and possibly help overcome a student's learning difficulties in the course of solving a problem or performing a task. In one approach to building an instructional monitor, the student uses a special task- or problem-oriented language expressly designed around some particular class of problems. Correspondingly, the diagnostic programs in this "special purpose" type of monitor system often utilize information that is specific to the kind of problem being studied. In the SIMON system we have taken an experimental approach of a different kind. The student addresses SIMON in an easy and very general programming language rather than a special task language. Using SIMON, students construct programs for systems or processes which can represent vastly different situations whether from mathematics, biology, physics, engineering, or elsewhere. The student tests his program against a "true" program provided to SIMON by an instructor. At the student's request, SIMON tests his program against its "true" model to determine if it works. If not, SIMON points out cases where the program fails and, if requested, informs the student which variables he has chosen that are inappropriate to the process.
<table>
<thead>
<tr>
<th>Key Words</th>
<th>LINK A</th>
<th>LINK B</th>
<th>LINK C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ROLE</td>
<td>WT</td>
<td>ROLE</td>
</tr>
</tbody>
</table>

Security Classification
SIMON - A Simple Instructional Monitor

Wallace Feurzeig  
and  
Paul M. Wexelblat  
Bolt Beranek and Newman Inc.  
50 Moulton Street  
Cambridge, Massachusetts 02138  

Ronald C. Rosenberg  
Dept. of Mechanical Engineering  
Michigan State University  
East Lansing, Michigan 48823  

August 1, 1970  

Sponsored by  
Personnel and Training Research Programs  
Psychological Sciences Division  
Office of Naval Research  
Arlington, Virginia 22217  

Contract No. N00014-69-C-0127  
NR No. 154-285  

This document has been approved for public release and sale; its distribution is unlimited.  
Reproduction in whole or in part is permitted for any purpose of the United States Government.
1. Introduction

Computers are used for instruction almost entirely in one of two distinct ways - either the student programs the machine or the machine programs the student. Certainly it is possible in principle to realize other kinds of instructional situations. Thus, for example, we would like to use the computer to monitor a student's work while he is freely programming the machine, just as an intelligent human teacher invites feedback from a student rather than always preempting him.

The image we start with, then, is the following. The student performs a task through using the machine. The machine knows what the student is trying to do and, in particular, is informed about how to tell whether an attempt is succeeding or failing. So, while the student carries out a series of operations and procedures on the machine, hopefully directed toward his goal, the machine allows the student's steps even though these had not been specifically anticipated, and attempts to diagnose his difficulties on the way.

A program of this kind is called an instructional monitor. There are many ways of designing such programs. For example, a monitor can be operating while the student is working (on-line) or just afterwards (post-mortem). Monitors can be told a lot or a little about the kinds of difficulties students might have and/or how to know them and/or how to diagnose their probable sources in his faulty conceptualization or articulation, and/or how to generate testable hypotheses about a student's conceptual "bugs" or "hang-ups." And so on.
Another way of characterizing monitors is in terms of the kind of languages used by the student and by the monitor program. These languages can be task languages highly specific to the class of problems being worked on. Thus, tasks like those of flying an airplane (in simulation) or designing an electrical circuit or solving a differential equation can be expressed in very different special languages. One can also take another line of approach in which the student (and the monitor) use a general-purpose programming language. The instructional monitor described here is of this kind.

2. An Easy Programming Language for Use in Teaching

Certain programming languages (such as Telcomp and BASIC) are easy to teach to anyone with a knowledge of school algebra and provide students with a fairly general facility for expressing a rich variety of mathematical procedures. Students have, in fact, used these languages to write programs for simulating the operations of many kinds of biological, chemical, physical, economic, linguistic, and other processes.

To illustrate how a student might use the Telcomp\textsuperscript{1} programming language in describing a simple physical process, consider the following problem: How long does it take a bead of mass M to slide down a fixed, frictionless wire from an initial point \((X_0, Y_0)\) to the origin \((0,0)\). The mathematical solution is

\begin{equation}
\end{equation}

\begin{equation}
\[ T = \frac{\sqrt{2 \cdot S}}{AC} \]

where \( S = \sqrt{(X0)^2 + (Y0)^2} \)

\( \theta = \arctan \left( \frac{Y0}{X0} \right) \)

\( ACC = \left| G \cdot \sin(\theta) \right| \)

and \( G \) is the gravitational constant.

Note that the mass \( (M) \) is not relevant.

The corresponding Telcomp program is:

1.1 \( S = \text{SQRT} \left( (X0)^2 + (Y0)^2 \right) \)
1.2 \( \text{THETA} = \text{ATN} \left( X0, Y0 \right) \)
1.3 \( ACC = 'G \times \sin(\text{THETA})' \)
1.4 \( T = \text{SQRT} \left( 2 \times S/AC \right) \text{ IF } Y0 \neq 0 \)
1.5 \( T = 10^{36} \text{ IF } Y0 = 0 \)

Note that each instruction line is given a two-part identification number, and that \( \times \) denotes multiplication, \( / \) denotes division, \( \dagger \) denotes exponentiation, and (in line 1.5) \( 10^{36} \) is used to express infinite time.

The program can be performed when the values of \( X0, Y0, \) and \( G \) are provided. A final instruction line is added to type out the result, as follows:

1.6 \( \text{TYPE T} \)

The instructions are performed in numerical order from 1.1 through 1.6. Note that the program very closely matches the mathematical statement of the solution.
Our experience is that a student is much better served by attempting to write programs of this kind than by merely using the same programs freely provided by an instructor. At the same time, students are greatly helped in their efforts to make a proper formulation of their own programs if they have a valid program available to them as a "black box." By comparing the operation of the good program with their own they can see where their errors lie and, possibly, how to fix them.

3. A Simple But Fairly General Instructional Monitor

We have developed a simple instructional monitor, SIMON, to help students carry out this process. The student tries to develop a Telcomp program to solve a problem. The student's program is accessible to the monitor. The monitor guides the student in two main ways -- by providing the student with the use of a "good" program, and by commenting on the student's errors in his choice of variables for his own program.

A typical student interaction with SIMON proceeds along the following lines:

(1) the student is given a problem to program;

(2) he chooses the variables that he thinks relevant;

(3) he runs the valid program with various inputs of his own choosing;

(4) he attempts to develop his own program, testing it with inputs of his choice along the way;

(5) when the student is satisfied with his solution, he asks the monitor to check it.
The student can call upon any of the steps (2) through (5) in any order he chooses and as often as he wishes. The interaction continues in this way under the student's control until either the student's program is correct or no further help can be given by SIMON.

Figure 1 is a schematic diagram showing the major components of the SIMON system, and the principal flows of information.

![Diagram of SIMON information flow](image)

Fig. 1. Diagram of SIMON information flow.

The figure shows that either SIMON or the student can initiate data for a problem trial, route it to either the monitor's program or the student's program, and direct the results to either
the monitor or the student. The student can say, for example, "You (SIMON) choose some trial data, try them with my program, and tell me the results." And the monitor can respond and report back "Your program appears to be right" or else "Your results do not match mine" and list some disagreements in the outputs of the two programs.

Some of the requests the student can make of SIMON are:

RUN YOU ... student asks monitor to run its program.
SELECT ... student selects relevant variables from monitor's list of allowable variable names.
COMPOSE ... student informs monitor that he wishes to create or to modify his own program.
RUN ME ... student asks monitor to run his program.
CHECK ME ... student asks monitor to check his program, listing the trial values used and results obtained.
HELP ME ... student asks monitor why his program failed the monitor's checking procedure.

4. An Example of a Student's Use of SIMON

To illustrate the use of the SIMON monitor, we again consider the sliding bead problem. The student is to develop a Stringcomp program that expresses a valid solution of the problem. The problem is given to the student along the following lines.

Bead on a Wire

A bead with a hole in it is free to slide on a piece of straight wire. If the wire is fixed in some position relative to the gravity field, how long will it take the bead to reach the end of the wire?
To discuss this problem, use the coordinates and names shown in Fig. 2 below.

Put the bead at some initial position \((X_0, Y_0)\), set the gravitational constant \((G)\), and find the time \((T)\) it takes for the bead to reach point \(O\). The mass of the bead is \(M\).

When you think you can compute the time, write a program to do so and ask SIMON to check it for you.

Figure 2. Problem statement for a bead on a wire.

An illustrative run is shown in Fig. 3 in which student typing is either preceded by an asterisk or underscored. First, the student selects the variables that he thinks are relevant for the proc. Then he chooses input values for use with SIMON's "good" program and observes the results (i.e., the three "RUN YOU" requests in Fig. 3a).

He then attempts (see Fig. 3b) to write a program of his own ("COMPOSE") which he uses twice (the two "RUN ME" requests). Though the program is wrong, the results are correct with the inputs he has chosen. He next asks SIMON to check his program.
"CHECK ME"), and his program produces erroneous results on the first set of trial values. At that point he seeks "HELP". SIMON exposes an error -- M (the mass) is not relevant to this problem.

In Fig. 3c, the student changes his choice of variables by eliminating M. He then modifies his program and runs it again. This time he tests his program on known values and results, is satisfied, and gets checked out successfully by SIMON. Notice that SIMON no longer asks for a value of M in operating either the student's program or its own.
BEAD ON A WIRE

(NOTE. XØ AND YØ MUST BE GREATER THAN OR EQUAL TO ZERO. EXPRESS INFINITY AS 10^36)

**SELECT**

SHALL I LIST THE LEGAL VARIABLE NAMES? YES

XØ INITIAL POSITION ON X AXIS
YØ INITIAL POSITION ON Y AXIS
M MASS OF BEAD
G GRAVITATIONAL ACCELERATION
T TIME

HOW MANY INPUT VALUES DO YOU THINK YOU WILL NEED? 4

LIST THEM
1 XØ
2 YØ
3 M
4 G

HOW MANY OUTPUT VARIABLE NAMES? 1

LIST IT
1 T

**RUN YOU**

XØ = 5
YØ = 5
M = 1
G = 32

T = .7905694

**RUN YOU**

XØ = 10
YØ = 10
M = 1
G = 32

T = 1.118034

**RUN YOU**

XØ = 5
YØ = 5
M = 1
G = 32

T = .8838835

Fig. 3a
**COMPOSE**

GO AHEAD

1.1 \( s = \sqrt{y_0^2 + x_0^2} \)

1.2 \( \text{acc} = g/m \)

1.3 \( t = \sqrt{\frac{2 \times s}{\text{acc}}} \) if \( y_0 \neq 0 \)

1.4 \( t = 10 \) if \( y_0 = 0 \)

1.5 TYPE T

**RUN ME**

\( x_0 = 5 \)
\( y_0 = 8 \)
\( m = 5 \)
\( g = 32 \)

YOUR PROGRAM---

\( t = 1 \times 10 + 36 \)

**RUN ME**

\( x_0 = 1 \)
\( y_0 = 5 \)
\( m = 1 \)
\( g = 32 \)

YOUR PROGRAM---

\( t = 1.118034 \)

**CHECK ME**

\( x_0 = 1 \)
\( y_0 = 1 \)
\( m = 3 \)
\( g = 32 \)

MY RESULTS: YOUR RESULTS:

\( t = 0.3535534 \) \( .5149418 \)

THESE DO NOT MATCH

**HELP ME**

YOU SELECTED A VARIABLE THAT IS NOT RELEVANT.
YOU DO NOT NEED \( m \)

Fig. 3b
SELECT

SHALL I LIST THE LEGAL VARIABLE NAMES? NO
HOW MANY INPUT VALUES DO YOU THINK YOU WILL NEED? 3
LIST THEM
1 XØ
2 YØ
3 GRAV
GRAV ISN'T ON THE LIST
DO YOU WANT TO SEE THE LIST NO
3 G

HOW MANY OUTPUT VARIABLE NAMES? 1
LIST IT
1 T

COMPOSE

GO AHEAD

\[ T = 0.8838835 \]  

CHECK ME

MY PROGRAM--- T = 0.3535534
YOUR PROGRAM--- T = 0.3535534
MY PROGRAM--- T = 0.559017
YOUR PROGRAM--- T = 0.559017
MY PROGRAM--- T = \textcolor{red}{1.104.36}
YOUR PROGRAM T = \textcolor{red}{1.104.36}
YOUR PROGRAM APPEARS TO BE RIGHT

Fig. 3c

-11-
5. How the Teacher Sets Up SIMON

To describe a problem to SIMON, the instructor needs to provide four main kinds of information:

(1) a statement of the problem for the student at the start of the session;
(2) a list of the allowable variable names, indicating those variables actually relevant to the problem at hand, and which of them are inputs or outputs;
(3) a valid solution program; and
(4) information for checking the student's program (e.g., trial values, number of trials).

The first step after the problem description is to establish an appropriate list of variables, and to designate the relevant input and output variables. If problems of a related type have previously been described to SIMON, an appropriate variable list may already exist. The instructor may add to this list, edit it, or start anew. We illustrate using the bead problem.

**WOULD YOU LIKE TO SEE THE CURRENT NAME LIST? NO**

**WILL YOU USE THIS LIST? NO**

**HOW MANY ENTRIES IN LIST OF POSSIBLE NAMES? 5**

**LIST NAME AND THEN COMMENT PLEASE:**

1. X0  INITIAL POSITION ON X AXIS
2. Y0  INITIAL POSITION ON Y AXIS
3. M   MASS OF BEAD
4. G   GRAVITATIONAL ACCELERATION
5. T   TIME

All input and output variables must be taken from this list. When the student selects his input variables, they will be compared with entries in this list. SIMON will thereby find
any extra or missing variables. Next the input and output variables are specified.

LIST THE INPUT NAMES
1 $x$
2 $y$
3 $g$

LIST THE OUTPUT NAMES
1 $t$
2

The instructor then states which test cases he will use to give a reasonable check on the validity of the student's solution program, and prov'ce, the associated trial values. The "CHECK" request causes these trials to be run according to the checking rules given below.

HOW MANY TRIES? 5

TRY NO. 1
$x=1$
$y=1$
$g=32$

TRY NO. 2
$x=\emptyset$
$y=5$
$g=32$

TRY NO. 3
$x=5$
$y=\emptyset$
$g=32$

TRY NO. 4
$x=-45$
$y=31$
$g=32$

TRY NO. 5
$x=577$
$y=466$
$g=32$

-13-
HOW MANY TIMES SHOULD I TRY FOR CHECKING? 3
SHOULD I RE-TRY THE VALUE WHICH FAILED FIRST? YES
WHAT PERCENT ERROR IS ACCEPTABLE? 1

Starting with "TRY NO. 1" SIMON will concede that the program is apparently correct when it does three trial sets correctly. If an error is found, the next check will start with that try on which the error occurred. The result-(T) in this case-must be accurate to within one percent.

The instructor then provides any desired instructions to the student. After this he writes a solution program which will be used by the monitor as the "true" program.

TYPE IN YOUR INSTRUCTIONS
TYPE DONE WHEN FINISHED

#BEAD ON A WIRE
*(NOTE: X₀ AND Y₀ MUST BE GREATER THAN OR EQUAL TO ZERO
*EXPRESS INFINITY AS 10+36)
*DONE

ENTER THE CORRECT PROGRAM NEXT USING PART 1₀

+1₀.1 \( \theta = \text{ATN}(x_0, y_0) \), \( s = \text{SQRT}(y_0^2 + x_0^2) \), \( \text{ACC} = 'g' \cdot \sin(\theta) \)
+1₀.2 \( t = \text{SQRT}(2 \cdot s / \text{ACC}) \) IF \( y_0 \neq 0 \)
+1₀.3 \( t = 10+36 \) IF \( y_0 = 0 \)
+1₀.4 TYPE T

Thus, describing a problem to SIMON is seen to be a straightforward task. This holds true for relatively complex problems.²

---

6. Some Teaching Applications

A student can use a programming language like Telcomp to investigate a mathematical or physical process experimentally. The following typescript is an example of this kind. The student interacts with a program simulating a missile trajectory problem, converging to the solution in a series of trials with successively improved estimates. (His inputs are underscored.)

YOU ARE IN COMMAND OF A GROUND-TO-AIR MISSILE WHOSE JOB IS TO BRING DOWN AN ENEMY TARGET.

AFTER YOU GIVE THE REQUIRED INFORMATION (ALL IN FEET, PLEASE) WATCH THE PROGRESS OF THE MISSILE ON THE RADAR SCREEN. THE '.1' REPRESENTS THE MISSILE. THE '': IS THE SHADOW OF THE TARGET AND '[:]' IS THE TARGET ITSELF.

WHAT IS THE DIAMETER OF THE TARGET? 14 HOW FAR AWAY IS THE TARGET? 500 WHAT IS ITS ALTITUDE? (BETWEEN 0 AND 200) 18 WHAT IS THE VELOCITY (IN FT/SEC) OF THE MISSILE? 304 AT WHAT ANGLE IS THE MISSILE TO BE SHOT? (IN DEGREES) 45

THE MISSILE WILL REACH THE TARGET ZONE IN 2.357023 SECS THE TIME INTERVAL ON THE RADAR SCREEN WILL BE .2357023 TIME (0 FEET) (200 FEET)

\[
\begin{array}{ccc}
0 & .24 & .47 \\
.71 & .94 & 1.18 \\
1.41 & 1.65 & 1.89 \\
2.12 & 2.36 & \text{[ ]} \\
\end{array}
\]

SORRY, YOU MISSED BY 267.2222 FEET.
WE'RE LOADING UP A NEW MISSILE TO USE ON THE SAME TARGET.
WHAT IS THE VELOCITY (IN FT/SEC) OF THE MISSILE? 300
AT WHAT ANGLE IS THE MISSILE TO BE SHOT? (IN DEGREES) 20

THE MISSILE WILL REACH THE TARGET ZONE IN 1.77363 SECS
THE TIME INTERVAL ON THE RADAR SCREEN WILL BE .177363

<table>
<thead>
<tr>
<th>TIME (0 FEET)</th>
<th>(200 FEET)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>.18</td>
<td></td>
</tr>
<tr>
<td>.35</td>
<td></td>
</tr>
<tr>
<td>.53</td>
<td></td>
</tr>
<tr>
<td>.71</td>
<td></td>
</tr>
<tr>
<td>.89</td>
<td></td>
</tr>
<tr>
<td>1.06</td>
<td></td>
</tr>
<tr>
<td>1.24</td>
<td></td>
</tr>
<tr>
<td>1.42</td>
<td></td>
</tr>
<tr>
<td>1.6</td>
<td></td>
</tr>
<tr>
<td>1.77</td>
<td></td>
</tr>
</tbody>
</table>

SORRY, YOU MISSED BY 26.32073 FEET.

WE'RE LOADING UP A NEW MISSILE TO USE ON THE SAME TARGET.
WHAT IS THE VELOCITY (IN FT/SEC) OF THE MISSILE? 300
AT WHAT ANGLE IS THE MISSILE TO BE SHOT? (IN DEGREES) 15

THE MISSILE WILL REACH THE TARGET ZONE IN 1.72546 SECS
THE TIME INTERVAL ON THE RADAR SCREEN WILL BE .172546

<table>
<thead>
<tr>
<th>TIME (0 FEET)</th>
<th>(200 FEET)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>.17</td>
<td></td>
</tr>
<tr>
<td>.35</td>
<td></td>
</tr>
<tr>
<td>.52</td>
<td></td>
</tr>
<tr>
<td>.69</td>
<td></td>
</tr>
<tr>
<td>.86</td>
<td></td>
</tr>
<tr>
<td>1.04</td>
<td></td>
</tr>
<tr>
<td>1.21</td>
<td></td>
</tr>
<tr>
<td>1.38</td>
<td></td>
</tr>
<tr>
<td>1.55</td>
<td></td>
</tr>
<tr>
<td>1.73</td>
<td></td>
</tr>
</tbody>
</table>

SORRY, YOU MISSED BY 6.296228 FEET.
WE'RE LOADING UP A NEW MISSILE TO USE ON THE SAME TARGET.
WHAT IS THE VELOCITY (IN FT/SEC) OF THE MISSILE? 300
AT WHAT ANGLE IS THE MISSILE TO BE SHOT? (IN DEGREES) 16.5

THE MISSILE WILL REACH THE TARGET ZONE IN 1.738248 SECS
THE TIME INTERVAL ON THE RADAR SCREEN WILL BE .1738248

<table>
<thead>
<tr>
<th>TIME (0 FEET)</th>
<th>(200 FEET)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>.17</td>
<td>x</td>
</tr>
<tr>
<td>.35</td>
<td>x</td>
</tr>
<tr>
<td>.52</td>
<td>x</td>
</tr>
<tr>
<td>.7</td>
<td>x</td>
</tr>
<tr>
<td>.87</td>
<td>x</td>
</tr>
<tr>
<td>1.04</td>
<td></td>
</tr>
<tr>
<td>1.22</td>
<td>x</td>
</tr>
<tr>
<td>1.39</td>
<td>x</td>
</tr>
<tr>
<td>1.56</td>
<td>x</td>
</tr>
<tr>
<td>1.74</td>
<td>[ . ]</td>
</tr>
</tbody>
</table>

CONGRATULATIONS! A DIRECT HIT!

In his use of this program, a student might have been guessing randomly or he might have been making estimates in a systematic way. Possibly his successive inputs were obtained from a bounding procedure independent of any thinking about the physics of the trajectory problem. Was he merely keeping one of his two variables, the velocity, fixed while systematically varying the other until the error became null? Or was he building and testing a mathematical model of the underlying physical process?

One way to find out is to teach him to write his own programs in Telcomp and then assign him the problem of writing the trajectory program himself. Since we really are more interested that he succeed in doing this than that we succeed in understanding his original estimation procedure, we have extended his capability
by embedding Telcomp in a program that allows him to check his evolving program against a "true" model. In brief, that is how SIMON was designed.

SIMON has been used experimentally with problems in mathematics, physics, engineering, and biology. In some of these applications the student's initial task has been successively developed and extended in a series of problems. Thus, for example, trajectory problems, such as the one just discussed, are complicated first by requiring an estimate of the impact point, then by making the time of release a variable under control of the student, and finally by incorporating winds.

Similarly, in a chemostat problem in biochemistry, the student's first task is to develop a model for describing the differential change in nutrient in the system and also the differential change in the bacterial density with time. In a subsequent problem, the chemostat system contains two strains of bacteria having the same maximum growth rate but requiring different amounts of nutrient to obtain this rate. The student is asked to compute the two differentials again, but for the mutant strain as well as the original strain.

The student using SIMON is ordinarily given two sources of information: a statement describing the problem or process to be programmed, and a "good" program to use for further investigating the process and for checking against his own program. This standard use of SIMON can be varied in the following three ways.

(1) The student is given the "good" program without a corresponding description of the process it represents. He tries to
determine the process by experimenting with the program, even though the intended function to be performed by the process is unspecified. The student may merely be told, for example, that the program describes an electrical circuit or a hydraulic system. His study situation is analogous to the "black box" problem in classical physics. It is similar also to some function guessing games in mathematics where one tries to determine a function by asking for its values corresponding to particular values of its variables. (Thus, if the hidden function is \( f(x,y) = x^y - y^x \), the input \( x=2, y=3 \) results in the output \(-1\).) Such mathematical games suggested the initial structure of SIMON and its first applications were of this kind.

(2) The student is given a "bad" program (a program with errors or "bugs") instead of a "good" one. Further, he is permitted to look at the instructions comprising the program (not merely to run the program as in the standard use of SIMON). His task is to debug the program, to change it into a valid description of the process. The SIMON monitor has a valid version of the program which it uses to check and critique the student's work.

(3) The student assumes the role of an instructor and inputs a problem to SIMON. Young students find particularly rewarding the task of preparing problems and tests for their peers. In so doing, they often become more aware about important problems of teaching and learning such as identifying, detecting, and diagnosing errors.

6. Summary

The SIMON monitor is an experimental system created mainly to help gain an understanding of the problems of building programs
for diagnosing learning difficulties. We had previously con-
structed an instructional monitor especially tailored for use
with more specific problems, i.e., the simulation and modeling
of certain types of dynamic physical systems. Monitors can be
expressly built around problems in a number of areas vastly
different from the ones treated there. Thus, we have recently
designed and implemented a problem-oriented instructional monitor
around a class of problems involving the task of flying an air-
plane "blindly" (i.e., by instruments), and similar monitors can
be constructed around operational tasks in areas such as naviga-
tion, maneuvering, and controlling.

The SIMON system represents a somewhat different approach to
diagnostic monitoring. SIMON is embedded in a procedure-oriented
(as distinct from a special task or problem-oriented) framework.
The SIMON monitor, and the student using it, employ a general
programming language. A fundamental limitation of this approach
is apparent from the start -- the problem of writing a program
which can "understand" other programs in some reasonable sense
is a recursively unsolvable problem. But a monitor can be useful
for teaching even though it is unable to decide whether two pro-
grams are formally equivalent. Although the prototypical
diagnostic capabilities of SIMON -- like finding out whether a
student's program works and, if not, checking for appropriate
inputs and outputs -- are rather modest, this work is just start-
ing and the early exploration has clearly shown directions in
which some further useful capabilities can be developed. For

(3) R. C. Rosenberg, W. Feurzeig, P. Wexelblat, "The ENPORT

-20-
example, the monitor can look for differences between its program and the student's in the overall behavior of the output variables as a function of the inputs. Thus, it could report to the student that activity should increase with temperature; that there was another case to consider in addition to that of a repulsive gravitation force; and so forth. Even this relatively straightforward extension of SIMON would considerably enhance its diagnostic power, and its value as a teaching instrument.

Acknowledgments

Philip Faflick and Richard Grant of BBN contributed to the design, programming, debugging, and application of SIMON.

The research was supported in part by the U. S. Office of Naval Research under Contracts NONR 4340(00) and N00014-69-C-0127.
OFFICE OF NAVAL RESEARCH
PERSONNEL AND TRAINING RESEARCH PROGRAMS (CODE 458)

DISTRIBUTION LIST

4 Chief of Naval Research
   Code 458
   Department of the Navy
   Arlington, Virginia 22217

   Director
   ONR Branch Office
   495 Summer Street
   Boston, Massachusetts 02210

3 Commanding Officer
   Naval Personnel and Training
   Research Laboratory
   San Diego, California 92151

   Director
   ONR Branch Office
   219 South Dearborn Street
   Chicago, Illinois 60604

6 Director, Naval Research
   Laboratory
   Washington, D. C. 20390
   ATTN: Library, Code 2029 (ONRL)

   Office of Naval Research
   Area Office
   207 West Summer Street
   New York, New York 10011

   Office of Naval Research
   Area Office
   1076 Mission Street
   San Francisco, California 94103

6 Director
   Naval Research Laboratory
   Washington, D. C. 20390
   ATTN: Technical Information Division

20 Defense Documentation Center
   Cameron Station, Building 5
   5010 Duke Street
   Alexandria, Virginia 22314

   Commanding Officer
   Service School Command
   U. S. Naval Training Center
   San Diego, California 92133

   Chairman
   Behavioral Science Department
   Naval Command and Management Div.
   U. S. Naval Academy
   Luce Hall
   Annapolis, Maryland 21402

   Dr. A. L. Siajkosky
   Scientific Advisor (Code AX)
   Commandant of the Marine Corps
   Washington, D. C. 20380

   Chief
   Naval Air Technical Training
   Naval Air Station
   Memphis, Tennessee 38115

   ATTENTION:
   C. S. Winiewicz
   Chairman
   Behavioral Science Department
   Naval Command and Management Div.
   U. S. Naval Academy
   Luce Hall
   Annapolis, Maryland 21402

   Dr. A. L. Siajkosky
   Scientific Advisor (Code AX)
   Commandant of the Marine Corps
   Washington, D. C. 20380

   Chief
   Naval Air Technical Training
   Naval Air Station
   Memphis, Tennessee 38115
Director
Education and Training Sciences
Department
Naval Medical Research Institute
National Naval Medical Center
Building 142
Bethesda, Maryland 20014

LCDR J. C. Meredith, USN (Ret.)
Institute of Library Research
University of California
Berkeley, California 94720

Commander
Operational Test & Evaluation Force
U. S. Naval Base
Norfolk, Virginia 23511

Office of Civilian Manpower Management
Technical Training Branch (Code 024)
Department of the Navy
Washington, D. C. 20390

Chief of Naval Operations, (Op-07TL)
Department of the Navy
Washington, D. C. 20350

Chief of Naval Material (MAT 031M)
Room 1323, Main Navy Building
Washington, D. C. 20360

Mr. George N. Graine
Naval Ship Systems Command (SHIPS 03H)
Department of the Navy
Washington, D. C. 20360

Chief
Bureau of Medicine and Surgery
Code 513
Washington, D. C. 20390

Personnel Research and Development Laboratory
Washington Navy Yard,
Building 200
Washington, D. C. 20390
ATTN: Library, Room 3307

Commander
Naval Air Systems Command
Navy Department, AIR-4132
Washington, D. C. 20360

Commandant of the Marine Corps
Headquarters, U. S. Marine Corps
Code A01B
Washington, D. C. 20380

Technical Library
Naval Ship Systems Command
Main Navy Building
Room 1532
Washington, D. C. 20360

Mr. Philip Rochlin, Head
Technical Library Branch
Naval Ordnance Station
Indian Head, Maryland 20640

Library, Code 0212
Naval Postgraduate School
Monterey, California 93940

Technical Reference Library
Naval Medical Research Institute
National Naval Medical Center
Bethesda, Maryland 20014

Technical Library
Naval Ordnance Station
Louisville, Kentucky 40214

Naval Undersea Research and Development Center
3202 East Foothill Boulevard
Pasadena, California 91107
ATTN: Code 161

- B -
Dr. Joseph W. Rigney
Behavioral Technology Laboratories
University of Southern California
University Park
Los Angeles, California 90007

Educational Testing Service
Division of Psychological Studies
Rosedale Road
Princeton, New Jersey 08540

Dr. Harold Gulliksen
Department of Psychology
Princeton University
Princeton, New Jersey 08540

Dr. George E. Rowland
Rowland and Company, Inc.
Post Office Box 61
Haddonfield, New Jersey 08033

Dr. Mats Bjorkman
University of Umea
Department of Psychology
Umea 6, SWEDEN

Dr. Howard H. Kendler
Department of Psychology
University of California
Santa Barbara, California 93106