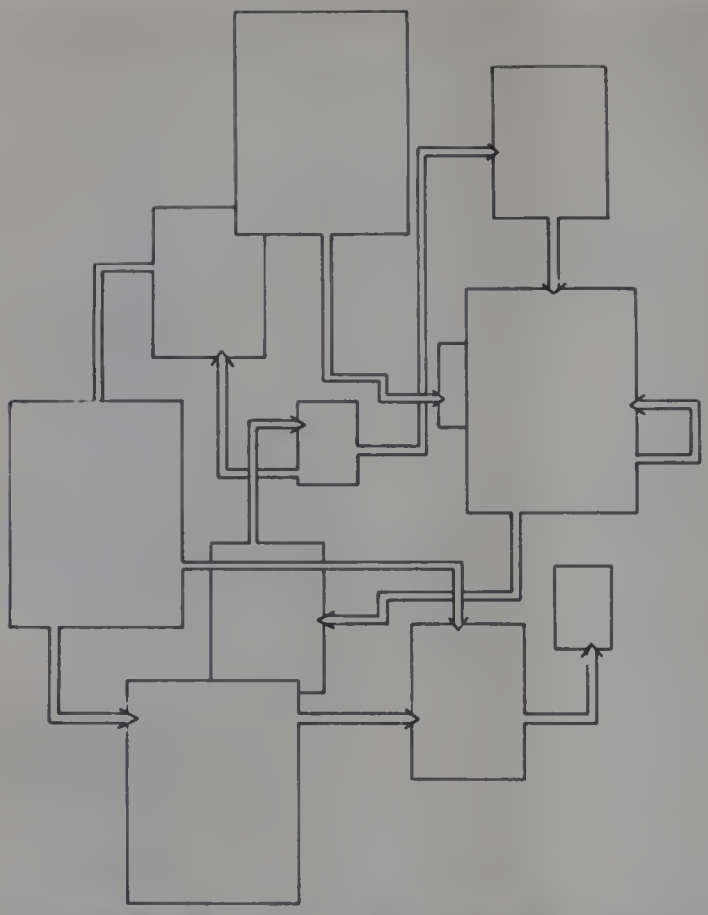

GROW:

A Program That "Learns"

Jeff Levinsky

One of the educational programs on the CHAOS II system (Ref 1 and 2) is a special extendable "game" named GROW. Based upon the well-known games of ADVENTURE and ANIMAL, GROW allows students to travel about in a vast world, learning about various topics while solving problems related to those topics. At any time, the students, as well as the teachers, may add new topics and questions to GROW, thus helping (or hindering) other who later explore the same route. In this context, GROW is as much a tool used to build CAI programs as it is a game.

One of the advantages of GROW is the speed and ease by which the things the program knowns about can be explored and extended. The basic framework can be built very quickly by a teacher and the task of filling in details (such as hints or paths to take to skip ahead to new lessons) can be added by students. A program which permits changes to itself is far more interesting than a standard interactive instructional system especially since the user's interactions may be preserved. This, of course, inspires and amplifies natural creativity.

GROW does not possess any artificial intelligence 'understanding' of either the subject matter or the user's interactions. Instead, simple pattern matching is used - but as the following sample session shows, this can be quite powerful in itself. The session begins with the user (whose input is in italics) invoking GROW from CHAOS II and trying some simple addition problems:

```
% GROW
WELCOME TO GROW
LEARN ADD
DO YOU WANT A DIFFICULT
 LESSON?
NO
SUPPOSE WE ADD 13 and -12.
 WHAT DO WE GET?
+1
RIGHT. SUPPOSE WE ADD -13
 and 12. WHAT DO WE GET?
+1
NO
-1
RIGHT
```

This standard pattern of GROW describing something (perhaps stating a problem) and the user then responding with a command or answer can be used far more imaginatively:

```
YOU ARE IN A DARK EERIE
 CAVE.
LIGHTS ON
THE CAVE IS NOW FILLED WITH
 GLARE- FREE FLORESCENT
 LIGHT. THERE IS A ROCK ON
 THE GROUND WITH THE MYS-
 TIC QUESTION: "WHAT IS
 THREE CUBED THRICE?"
19682
HUH?
19683
THRICE - NOT TWICE.
7625597484987
GOOD. THE ROCK VANISHES
 AND A GLISTENING METAL
 CYLINDER APPEARS IN ITS
 PLACE. THE CYLINDER IS
 ROUGHLY 6 INCHES HIGH
 AND TWO FEET ACROSS.
KICK IT
THAT MIGHT HARM IT.
SIT ON IT
THAT MIGHT HARM IT.
STAND ON IT
YOU HAVE JUST BEEN TRANS-
 PORTED TO THE STAR SHIP
 ENTERPRISE. YOU ARE ON
 THE BRIDGE. THE COMPUTER
 IS MALFUNCTIONING! IT
 DOES NOT KNOW WHAT THE
 SQUARE ROOT OF 3865156 IS.
ASK KIRK
HUH?
```

The scene the user is currently in can always be extended. In brief, the word EXTEND causes GROW to ask for

Jeff Levinsky, 2240 California St. #2, Mt. View, CA 94040.

Grow, con't...

keywords or phrases that will cause certain actions:

```
EXTEND
KEY WORDS / PHRASES
: SPOCK
: KIRK
: MCCOY
: —
ACTIONS
: PHE'S AWAY ON A MISSION!
: —
EXTEND
KEY WORDS / PHRASES
: 1966
: —
ACTIONS
: PTHAT'S RIGHT — YOU've
 SAVED THE FLEET.
: +10
: GSICKBAY
: —
```

As the extensions given are entered immediately, the question "ASK KIRK" no longer fails; instead, a message is printed, as specified above.

```
ASK KIRK
HE'S AWAY ON A MISSION!
WHAT ABOUT SPOCK
HE'S AWAY ON A MISSION!
IT'S 1966
THAT'S RIGHT — YOU'VE
 SAVED THE FLEET.
DESCRIBE SICKBAY
```

One of the actions to be taken when "1966" is given is "GSICKBAY," that is, "go to the SICKBAY scene." GROW discovers that this scene has never before been entered, so the user is asked to describe it. The user then extends that scene.

```
: YOU ARE IN THE SICKBAY -
 AN ALIEN HAS A HEARTBEAT
 OF 57 BEATS
: PER HOUR. HOW MANY
 BEATS A YEAR IS THAT?
: (A STANDARD GALACTIC
 YEAR IS 365 EARTH DAYS)
: —
EXTEND
KEY WORDS / PHRASES
: 499320
: —
ACTIONS
: PTHAT'S IT
: +5
: GCASTLE2
: —
EXTEND
KEY WORDS / PHRASES
: MCCOY
: DOCTOR
: —
ACTIONS
:PTHE GOOD DOCTOR CAN'T
 HELP.
: —
```

Now the SICKBAY scene can be immediately used:

```
WHERE'S MCCOY
THE GOOD DOCTOR CAN'T
 HELP.
TELL MCCOY IT'S 499320
THAT'S IT
YOU ARE IN AN ANCIENT
 CASTLE TOWER, AND MUST
 TELL THE EVIL WIZARD THE
 FULL ORIGINAL NAME OF THE
 EMPEROR AUGUSTUS.
GAIUS OCTAVIUS
VERY GOOD.
```

And, finally:

```
HELP
TO QUIT, TYPE 'QUIT'
TO START OVER AGAIN, TYPE
 'RESTART'
TO ADD TO THIS* TYPE 'EX-
 TEND'
QUIT
QUIT WITH 70 POINTS
%
```

Using GROW

As the examples suggest, users of GROW always travel from scene to scene, or more precisely, from node to node. Correct answers typically cause a new node to be entered, but incorrect answers might also. The term "Node" is preferred over "scene" as sometimes a new node might still be refering to the same scene, as was the case in the cave example above.

Each node consists of an initial description of the node and a number of keywords/phrases matched with various actions. Whenever a user enters a new node, the description is printed (e.g., "YOU ARE IN A DARK EERIE CAVE"). However, if the node has never before been entered, then the user must instead provide an initial description for the node, which GROW records permanently (as in the SICKBAY node above). The keywords/phrases and actions can be added to a node via "EXTEND." In all cases, when node data is entered a blank line is used to indicate the end of the data. When the user is in a node, the actions will be performed if the command typed in by the user contains one of the associated keywords or phrases. Once one match is found, GROW will look no further. As an example, in the SICKBAY node the keyword "499320" has the actions "PTHAT'S IT," "+5," and "GCASTLE-2" associated with it while the keywords "MCCOY" and "DOCTOR" are associated with the action "PTHE GOOD DOCTOR CAN'T HELP." Due to the order in which these two extensions were made, the command "TELL MCCOY IT'S 499320" is interpreted correctly: GROW first tries to find "499320" and then, if that fails, it tries to find "MCCOY." Care is needed in cases like this, for if the node had been extended in the other order, "MCCOY" would be matched before "499320" and the user would not have been credited for the correct answer.

There are no limits on the number of nodes, extensions, keyword/phrases, or actions, or on the length of the initial descriptions of nodes other than the overall file space of CHAOS II. The set of primitive actions (those defined by GROW itself) is small but quite powerful. In addition to nodes, GROW also keeps track of the user's score and two of the primitive actions allow this to be modified. Two other primitive actions allow nodes to be entered and extended. The primitive actions are:

+number	- Adds points to the user's score, as in "+20."
-number	- Subtracts points from the user's score, as in "-15."
Ptext	- Prints out the text, as in "PTHAT MIGHT HARM IT."
Gnode	- Goes to another node, as in "GSICKBAY."
X	- Allows the user to extend the current node.
Q	- Causes GROW to print the user's score and quit.

These primitive actions are used much like statements in a programming language, and there is one restriction: actions following a "G," "X" or "Q" within the list of actions of a single extension are ignored. This is because GROW immediately goes to another node, modifies the current node, or quits, respectively, upon performing these actions and thus cannot come back to do the next.

When the user types in a command, GROW often searches the current node for a matching keyword or phrase unsuccessfully. If this happens, GROW will then search a node named DEFAULT. Some keywords in DEFAULT are "QUIT" and "EXTEND" which cause the "Q" and "X" actions, respectively. DEFAULT then becomes the current node and it can also be extended. If GROW cannot find a match in DEFAULT, then a random message is printed out and no action is taken. GROW also knows about a node named INIT, which is the current node whenever GROW is invoked from CHAOS II initially. INIT can also be extended in the normal fashion.

All of the effects illustrated in GROW can be obtained by clever uses of nodes, keyword/phrases and actions. For instance, GROW knows nothing of arithmetic but numerical

Grow, con't...

answers can be stored as keywords. Placing spaces around numerical keywords can be used to insure correct matching. Experimentation is the best way to learn how to use GROW to set up lessons, tutorials, quizzes and games.

Inside GROW.

GROW is written in CHAOS II BASIC which is similar to most microcomputer versions. The program uses mainly strings (but no string arrays) and sequential files, and should be easy to transport onto most disk or cassette systems.

Nodes are stored in files with the name of the file being the name of the node. The internal format uses null strings to separate the components of the node. The format is:

1. The initial description of the node (zero or more strings),
2. A null string and
3. Zero or more of:
   a. a list of keywords/phrases (one to a line),
   b. a null string,
   c. a list of actions (one to a line) and
   d. a null string.

Nodes are extended by adding another list of keywords/phrases and list of actions at the end of the file (separated and terminated with null strings). Since files are totally dynamic in CHAOS II, GROW need not worry about the size of the node.

The main variables in GROW are:

- N$ the name of the node the user currently is in,
- P the user's score,
- I$ a line input from the user and
- S$ a line input from a file.

GROW is constructed modularly with the main program consisting of a call to the initialization subroutine and an infinite loop in which the subroutine to get and process commands from the user are called. The subroutines in GROW are:

INITIALIZE line 2000
Clears space for the strings and files, zeroes the score and causes the introductory message in the INIT node to be printed.

INPUT LINE line 3000
Gets a (command) line from the user. The prompt is a space.

PROCESS LINE line 4000
Tries to find a match for the (command) line by first looking in the current node (N$) and then in the DEFAULT node. The FIND LINE subroutine does the actual searching and returns with F set to 1 if, and only if, it has found a match. If a match is found, then the DO ACTIONS subroutine is used, otherwise a random response is given.

FIND LINE line 5000
Searches the currently opened file for a match to I$. Subroutine SKIP BLOCK is used to skip over the initial description of the node and later over the lists of actions. Subroutine GET LINE is used to obtain a line S$ which contains a keyword/phrase. FIND LINE returns with F = 0 unless a match is found - in which case F = 1.

DO ACTIONS line 5400
Finds and performs the actions associated with a keyword/phrase that has been matched. SKIP BLOCK is used to get to the list of actions and GET LINE is used to obtain the actions one at a time. Null and undecipherable actions are ignored. Subroutine CONVERT is used to get the numerical value for actions with "+" or "-." Subroutine GO TO NEW NODE is used for action "G;" subroutine EXTEND NODE is used for action "X."

RANDOM RESPONSE line 5800
Prints a random response.

SKIP BLOCK line 6000
Reads through a file (#1) until a null line or the end-of-file is encountered. For instance, SKIP BLOCK is used to advance through files containing nodes skipping over lists of actions.

GET LINE line 6200
Reads the next line from a file and returns this in S$. If there are no more lines, S$ is set to the null string.

CONVERT line 6400
Converts the second through the last characters of S$ into an integer and returns this in N.

INPUT NEW DATA line 6600
Obtains lines from the user and writes them into a node. The prompt is a colon. The user indicates the end of the list of lines by an empty line.

COPY NODE line 6800
Copies the current node into a node called TEMP. TEMP can later be expanded by subroutine INPUT NEW DATA.

GO TO NEW NODE line 7000
Sends a user to another node if the name appears legal. N$ is updated. If the node exists, subroutine GET LINE is used to help print out the description in the node. If the node does not exist, the ERROR trap is used and the user must describe the new node. The description is entered via subroutine INPUT NEW DATA. The ON ERROR GO TO 0 resets the ERROR trap.

EXTEND NODE line 8000
Uses INPUT NEW DATA to add the list of keywords/phrases and the corresponding list of actions to the current node. The COPY routine is required due to CHAOS II file usage.

```
BASIC
MITS BASIC VER. 4.1
OK
LOAD"GROW1
OK
LIST

10 REM                  ***  EXTENSIBLE ADVENTURE  ***
20 REM                  *    COPYRIGHT (C) 1979     *
30 REM                  *      BY JEFF LEVINSKY     *
40 REM                  *    ALL RIGHTS RESERVED    *
50 REM                  * WRITTEN IN CHAOS II BASIC  *
60 REM
1000 REM                                    MAIN LOOP
1010 GOSUB 2000
1020 GOSUB 3000
1030 GOSUB 4000
1040 GOTO 1020
2000 REM                                    INITIALIZE
2010 CLEAR 300,2
2020 P=0
2030 N$="INIT"
2040 I$=" INTRO "
2060 GOSUB 4000
2070 GOTO 1020
3000 REM                                    INPUT LINE
3010 LINE INPUT " ";I$
3020 I$=" "+I$+" "
3030 GOTO 1030
4000 REM                                    PROCESS LINE
4010 OPEN "I",1,N$
4020 GOSUB 5000
4040 IF F=1 THEN 4110
4045 CLOSE
4050 OPEN "I",1,"DEFAULT"
```

Grow, con't...

```
4060 GOSUB 5000
4080 IF F=1 THEN 4110
4090 GOSUB 5800
4100 GOTO 4120
4110 GOSUB 5400
4120 CLOSE
4130 RETURN
5000 REM                                    FIND LINE
5010 F=0
5020 GOSUB 6000
5030 IF EOF(1) THEN RETURN
5040 GOSUB 6200
5050 IF S$=""THEN 5020
5060 S$=" "+S$+" "
5070 IF INSTR(I$,S$)=0 THEN 5020
5080 F=1
5090 RETURN
5400 REM                                    DO ACTIONS
5410 GOSUB 6000
5420 GOSUB 6200
5430 IF S$=""THEN RETURN
5440 T$=LEFT$(S$,1)
5450 IF T$="P"THEN PRINT MID$(S$,2):GOTO 5420
5460 IF T$="Q"THEN PRINT"QUIT WITH "IP;"POINTS":END
5470 IF T$="+" THEN GOSUB 6400:P=P+N:GOTO 5420
5480 IF T$="-"THEN GOSUB6400: P=P-N:GOTO5420
5490 IF T$="G"THEN GOSUB 7000:RETURN
5500 IF T$="X"THEN GOSUB 8000:RETURN
5510  GOTO5420
5800 REM                                     RANDOM RESPONSE
5810 R=INT(RND(0)*3)
5820 IF R=0 THEN PRINT"HUH?"ELSE IF R=1 THEN PRINT"WHAT"ELSE PRINT"I DON
 T UNDERSTAND"
5830 RETURN
6000 REM                                    SKIP BLOCK
6010 IF EOF(1)THEN S$="":RETURN
6020 INPUT#1,S$
6030 IFS$<>""THEN 6010
6040 RETURN
6200 REM                                    GET LINE
6210 IF EOF(1) THEN S$=""ELSE INPUT#1,S$
6220 RETURN
6400 REM                                    CONVERT
6410 N=0
6420 I=2
6430 IF I>LEN(S$)THEN RETURN
6440 N=N*10+ASC(MID$(S$,I,1))-ASC("0")
6450 I=I+1
6460 GOTO 6430
6600 REM                                    INPUT NEW DATA
6610 LINE INPUT ":";I$
6620 PRINT#1,I$
6630 IF I$<>""THEN 6610
6640 RETURN
6800 REM                                    COPY MODE
6810 OPEN "I",2,N$
6820 OPEN "O",1,"TEMP"
6830 IF EOF(2) THEN RETURN
6840 INPUT #2,I$
6850 PRINT #1,I$
6860 GOTO6830
7000 REM                                    GOTO NEW MODE
7010 IF(LEN(S$)<2)OR(LEN(S$)>9)THEN RETURN
7020 CLOSE
7030 N$=MID$(S$,2)
7040 ON ERROR GOTO 7100
7050 OPEN "I",1,N$
7060 GOSUB 6200
7070 IFS$=""THEN7150
7080 PRINTS$
7090 GOTO 7060
7100 RESUME 7110
7110 CLOSE
7120  OPEN "O",1,N$
7130 PRINT"DESCRIBE "IN$
7140 GOSUB 6600
7150 ON ERROR GOTO 0
7160 RETURN
8000 REM                                    EXTEND MODE
8010 CLOSE
8020 GOSUB 6800
8030 PRINT"KEY WORDS PHRASES"
8040 GOSUB 6600
8050 PRINT"ACTION"
8060 GOSUB 6600
8068 CLOSE
8069 KILL$
8070 NAME "TEMP"AS N$
8080 RETURN
```

Figure 1

To run GROW on a new system, the nodes INIT and DEFAULT must also be provided. On CHAOS II, these nodes can be created using the text editor. On other systems a short program can be written to read lines from the terminal and place them in a file of strings. The listings of GROW, INIT and DEFAULT are given in Figure 1.

Beyond GROW

Some of the features absent in GROW are actually provided by the CHAOS II system. These include the ability to have several GROW subsystems and the ability to protect specific nodes from extension. CHAOS II also enforces a quota on files thus preventing GROW from overgrowing the entire computer. Users of other systems might obtain these features by modifying GROW itself. For instance, GROW might automatically prefix each node name with an ASCII character indicating to which subsystem the node belongs. This provides up to 256 subsystems and prevents errors arising from similarly-named nodes in different subsystems.

Those accustomed to programming languages such as BASIC or PILOT will probably feel uncomfortable with the limited number of primitive actions that GROW provides. A language such as BASIC or PILOT could be used in place of the primitive actions, but the implementation can be difficult. One technique is to encode nodes as subroutines rather than as data files and to merge the current node into the program workspace. Certain BASICs appear to have this capability.1 Another technique, which can be used with BASICs that provide an immediate mode, is to encode GROW in assembly language and have it call upon the BASIC to execute the actions in the node. Using a high level language for actions does require that all users who add extensions learn that language (or at least a subset of it), which may be impractical, especially with elementary school students.

7 POINT FOOTNOTE GOES HERE

On larger computers, GROW can very easily be implemented in interactive languages such as APL and LISP.

In connection with a computer science curriculum, GROW can be used effectively as a base for a series of programming exercises. For example, a program can be written which will find all nodes and all paths between nodes. This can then be modified to print all of the descriptions of nodes or all of the keywords and phrases. If the GROW program has been in use for some time, the

Grow, con't...

listings of what has been added can be quite amusing. An advanced project is to find the shortest way (in terms of nodes visited) to score as many points as possible (without scoring at a particular node more than once). This is a variant of the Traveling Salesman problem.

Finally, imagine a single GROW system on a network of personal computers, with thousands of users playing simultaneously. Even with only a dozen or so users providing extensions at a time, one could be certain to be able to explore new nodes forever. □

References

1. Kroening, Mary E. **CHAOS User's Manual.** San Diego, California: 1978.

2. Levinsky, Jeff L. "CHAOS: An Interactive Timeshared Operating System for the 8080," **Dr. Dobb's Journal.** XXXI (January, 1979), 6-13.

CHAOS II

CHAOS II is a multi-user, multi-tasking 8080-based system developed at Clairemont High School in San Diego, California as an economical alternative to large timeshare or networked microprocessor installations. CHAOS II permits at least half a dozen students to simultaneously develop and/or run programs written in BASIC, 8080 machine language and SHELL (a powerful command language). Students and teachers may each have private directories; both files and directories may have access rights specified to restrict or allow access from other directories. CHAOS II currently uses floppy disks for mass storage, and directories are assigned to a particular disk, thereby permitting one disk to be used per class. This in turn results in greater economies, as the disks for one class can be copied and dismounted and the disk for another class mounted on the same drive within a five to ten minute class break.

For further information about CHAOS II, please write to:

Computer Systems Design Group
3632 Governor Dr.
San Diego, CA 92122

"You don't have a mind of your own and can be programmed to do anything, but on the other hand, you have a great memory and..."

©Creative Computing