The LOGO Turtle-maze

Robert H. Seidman
Bionics Research Laboratory*
University of Edinburgh
Forrest Hill
Edinburgh EH1 2QL

*After Sept. 1972:

Systems and Information Science,
313 Link Hall,
Syracuse University,
Syracuse, New York 13210,
U.S.A.

August, 1972.

Dear Reader,

Members of the LOGO Project at Syracuse University have been teaching LOGO programming and turtle geometry for almost one year to elementary school children (ages 7-12) and are involved in a variety of research. My concern has been the growth of logical thinking and problem solving abilities in young children particularly in the transition period between what Piaget calls the stages of concrete operations (7-11) and formal operations (11-adolescence).

I have developed an investigative tool which hopefully will provide a rich experimental environment (for both user and experimenter) and whose use may shed some light on various aspects of logical thinking and problem solving. This tool together with various experiments forms a major portion of my Ph.D. research.

A series of four papers currently in various stages of development serve as a basis for this research.

A. Computer Aided Education: LOGO and the Education Milieu

The LOGO System is described and placed in what we feel is its proper perspective. The field is divided into three categories: direct instruction, games and simulation, and use as a computational device.

The LOGO System falls into the latter category but encompasses parts of the others as well.* We may think of the LOGO System as an experimental computation laboratory closely akin to experimental-laboratory simulation systems but with a difference of paramount importance: the LOGO System is in its fullest generality a meta-experimental-laboratory simulator.

* This is in direct contrast to an attitude expressed in a needlessly argumentative paper (see Seidman, R.H.: "Computer-Aided-Instruction"; Syracuse University/ Computer Aided Learning Laboratory Report No. 1: January 1972; Syracuse, New York).

We make no attempt at an in-depth survey of the entire field of computers in education. This has already been adequately done by others. Rather, a broad historical overview of the development of the three categories is presented from the following four perspectives.

- The degree of user-system control over interaction.
- The range and flexibility of user-system response.
- The system's model of the user and the user's knowledge (or model) of the workings of the system.
- The measures used to evaluate performance and achievement.

Unlike most direct instructional systems (e.g. traditional computer-aided-instruction) the LOGO System presupposes no particular model of learning and instruction. It is a convenient tool for theoretical and practical exploration and provides a broad and flexible experimental environment for both user and experimenter.

B. The LOGO Turtle-maze

We hope to gain insights into the development of logical thinking through experiments dealing with turtle-maze problem solving.

The experimenter or user creates a maze (a graph structure) and the user writes LOGO procedures which cause the turtle to traverse the maze. The range of possible experiments is large, and general and specific problem solving principles can be developed and studied. The LOGO turtle-maze will be in operation by December 31st, 1972.

C. <u>Logical Thinking and Problem Solving - Piaget's Theory</u>

Our theoretical orientation is derived from the work of Jean Piaget and since LOGO is not tied to any one cognitive theory we have chosen this one as the basis for our investigations. Other researchers may wish to use the LOGO turtle-maze tool to investigate other theories.

Piaget's theory of cognitive development and logical thinking is presented and focused on areas pertinent to our research. The specific research areas are described and we show how turtle-maze problem solving may shed light on certain aspects of logical thinking. Our hypotheses are developed.

D. <u>Experimental Design</u>

The various turtle-maze experiments are described.

A, C and D are in various stages of thinking and writing while B is complete (attached) and awaits implementation.

A tentative dissertation outline follows.

<u>Children, Computers and Cognitive Development: investigations of logical thinking and problem solving in young children</u>

The LOGO Turtle-maze

I. Introduction

LOGO is a fully recursive LISP-like language (developed at M.I.T. and Bolt Beraneck and Newman) used in conjunction with a time-sharing digital computer via teletypewriters. The language through the computer drives various peripheral devices and together make up the LOGO System. LOGO has a simple English-like syntax and researchers at M.I.T., Syracuse University and elsewhere use it to teach young children (ages 7-12) computer programming.

A major peripheral in the LOGO System is an electromechanical device called a turtle. The user can direct the turtle to move forward and backwards a specified number of turtle-steps, rotate clockwise or counterclockwise a specified number of degrees, and toot a high horn, low horn or ring a bell a specified number of times.

In addition, the user can direct a pen to drop or rise from the turtle's mid-center thus enabling the device to leave a turtle-trace.

The electromechanical turtle has a graphical counterpart represented by an arrow on a CRT screen and is capable of leaving a turtle-trace for a period of time specified by the user.

Proc. 1 is a LOGO procedure (program) for a turtle-drawing of an expandable diamond.

```
> TO DIAMOND  :X :Y        [procedure name, side length
                            input, recursion control index
                            input]*
> 10 PD                    [pen down, see Fig. 1]
> 20 FD  :X                [forward side length, Fig. 2]
> 30 RT  120               [rotate clockwise 120°, Fig. 3]
> 40 FD  :X                [Fig. 4]
> 50 PU                    [pen up]
> 60 :Y←(:Y-1)             [decrement recursion control
                            index by one]
> 70 IF  :Y=0 STOP         [if recursion control index = 0,
                            STOP, otherwise line 80]
> 80 RT  120               [Fig. 5]
> 90 DIAMOND  :X :Y        [recursive call to DIAMOND with
                            new value for recursion control
                            index, Figs. 6-8]
> 100 STOP
> END
```

<u>Procedure 1</u> Expandable Diamond

Proc. 1 is executed with inputs :X=20 (turtle-steps) and :Y=2. Varying the value of :X varies the size of the diamond.

```
> DIAMOND 20 2             [execute DIAMOND]
```

* Bracketed sentences are explanatory text notes.

Figs. 1-8 illustrate the turtle's actions.

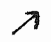
Fig. 1 Initial turtle position.

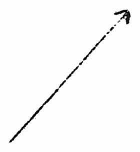
Fig. 2 Turtle moves 20 steps.

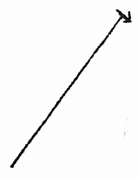
Fig. 3 Turtle rotates 120°.

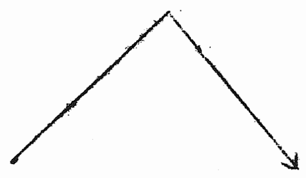
Fig. 4 Turtle moves 20 steps.

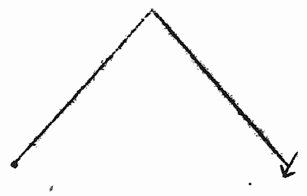
Fig. 5 Turtle rotates 120°.

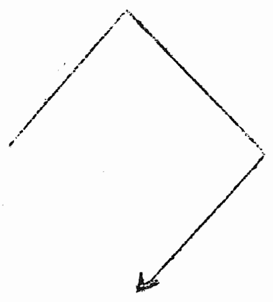
Fig. 6 Turtle moves 20 steps.

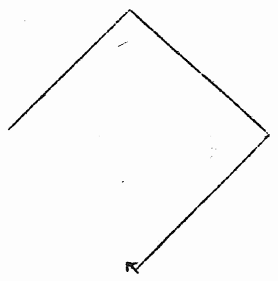
Fig. 7 Turtle rotates 120°.

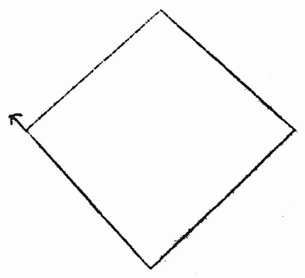
Fig. 8 Turtle moves 20 steps.

The user has the freedom to correspond spatial distances with turtle-steps.

```
> STEPSIZE 1                [1 inch=1 turtle-step]
```

With the above setting the electromechanical turtle will draw a diamond with cross diagonals of 34.6 and 20 inches.

Seymour Papert, one of the system's creators, foresees the placement of touch sensors on the front, sides and rear of the turtle as well as a light source and sensor thus providing the system and user with real-time environmental feedback.

We propose an alternative to this scheme which provides at once a flexible and extremely rich real-time environmental feedback interface.

II. The Basic Software Turtle-maze

A. The following scenarios will serve to partially illustrate this alternative scheme.

We would like to provide the user with the turtle-maze shown in Fig. 9.* Proc. 2 constructs a software model. Underlined items signify LOGO System responses. 0^{o} indicates due north and 180^{o} due south.

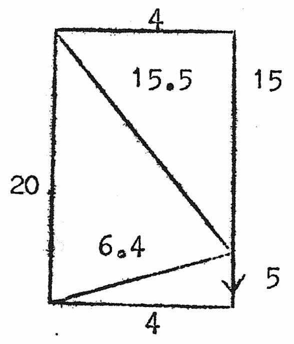

Fig. 9 MAZE1. Numbers correspond to feet, $\rightarrow$ indicates a one-way path, otherwise two-way.

```
> MAZEGEN                              [call software maze generator]
* NAME:   MAZE1                        [name of maze to be created]
* N1→N2:   0  20                       [see Fig. 10]
* N1→N3:   45 6.4                      [Fig. 11]
* N1→N4:   90 4 OK                     [Fig. 12]
* N2→N5:   90 4                        [Fig. 13]
* N2→N6:   N3    OK                    [Fig. 14]
* N3→N4:   N4    ONEWAY                [Fig. 15]
* N3→N5:   N5    OK   FIN   SUP        [Fig. 16]
* MAZE1 FINISHED
```

Procedure 2 Creation of software model of MAZE1. Confirmation printout has been suppressed by SUP in the next to last line.

An automatic maze inconsistency-checker warns the maze creator when necessary and provides suitable alternatives.

* Figures are not drawn to scale.

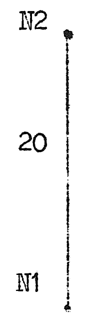

Fig. 10 0° from N1 and 20 units.

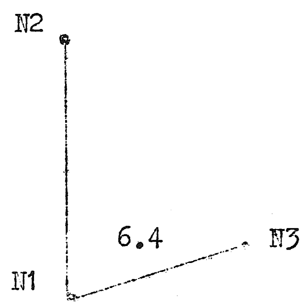

Fig. 11 45° from N1 and 6.4 units.

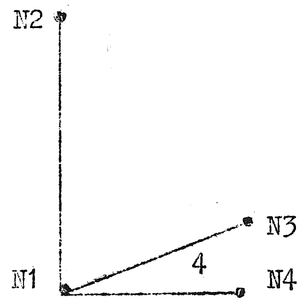

Fig. 12 90° from N1 and 4 units.

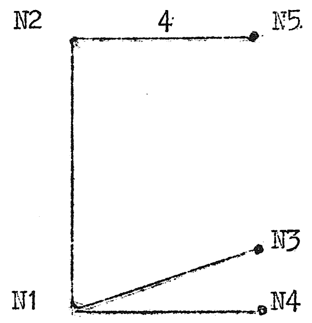

Fig. 13 90° from N2 and 4 units.

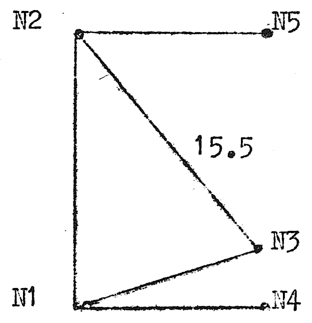

Fig. 14 (140° from N2) connect to N3.

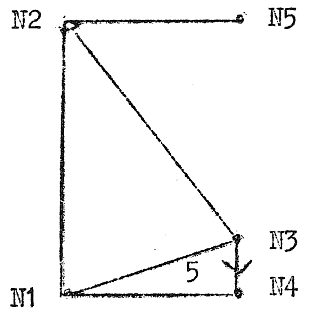

Fig. 15 (180° from N3) connect to N4.

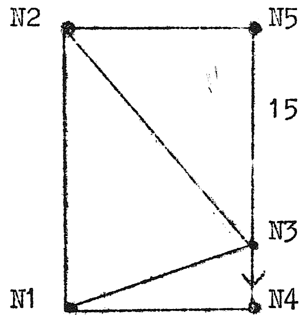

Fig. 16 (0° from N3) connect to N5.

Figs. 10-16 show the development of the software model for MAZE1. The system now has an internal model of the maze and needs only to know the turtle-step spatial correspondence, start point and orientation, and goal point(s) for total information.

Mazes, like procedures, can be stored, called and copied. The maze may be chalked or laid out using tape on the floor or turtle surface.

```
> STEPSIZE MAZE1  1      [1 inch=1 turtle-step for MAZE1]
```

With the above information the software model for MAZE1 is now represented by Fig. 17.

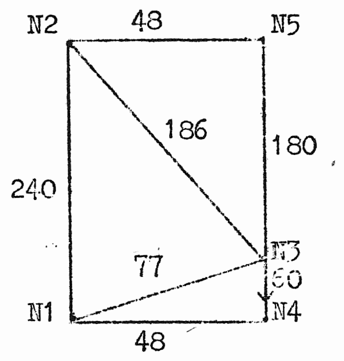

Fig. 17 Software maze model for MAZE1. Numbers represent turtle-steps.

The following command causes the software model of MAZE1 to be printed in table form (Fig. 18).

```
> DISPLAY MAZE1
```

PATH	ANGLE	DISTANCE (STEPS)
N1→N2	0	240
N1→N3	45	77
N1→N4	90	48
N2→N5	90	48
N2→N3	140	186
N3→N5	0	180
N3→N4	180	60 ONEWAY IN N3→N4 DIRECTION

Fig. 18 Table for software representation of MAZE1.

We wish to write procedures (Procs, 3,4) to cause the turtle to search MAZE1 for a goal. The turtle may start at any node with any heading and we may designate any node as the goal-node. Another possible version would allow the turtle to start at any point on any path in the maze. The goal may be similarly placed.

```
> TO SEARCH1
> 10 FD 480
> 20 RT 45
> 30 BK 720
> 40 STOP
> 50 PI: INTER1        [local nonexecutable instruction,
                        designates procedure to be called
                        when turtle goes off-course]
> 60 GOAL: STOP        [local nonexecutable instruction;
                        when goal is found STOP (a procedure
                        name could replace STOP causing
                        control to be transferred when goal
                        is found)]
> END
```

Procedure 3   Main search procedure with program interrupt (PI).

```
> TO INTER1
> 10 RT 1
> 20 IF BLOCK INTER1      [if truth value of BLOCK is still
                           TRUE (means turtle is off-course)
                           recursively call INTER1, otherwise
                           line 30]
> 30 SEARCH1 COMPLETE     [return to SEARCH1 and try to complete
                           the instruction interrupted by the BLOCK]
> 40 STOP
> END
```

Procedure 4   The interrupt procedure designated in SEARCH1.

The following instructions provide the system with total maze information.

```
> INITIALIZE MAZE1
* START:   N1  0          [turtle starts at N1 heading 0°]
* GOAL:    N3             [goal at N3]
```

The turtle is placed at N1 on the turtle surface headed 0^o (Fig. 18) and procedure SEARCH1 is executed.

> SEARCH1

The turtle moves to N2 and halts (Fig. 19).

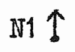

Fig. 18 Initial turtle position

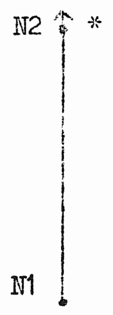

Fig. 19 Turtle halts at N2. * signifies BLOCK set at TRUE.

Before instruction 10 in Proc. 3 is executed the projected resulting action is compared against the N1→N2 path length in the software model. It is noted that execution of this instruction would cause the turtle to overshoot N2. The turtle is allowed to proceed to N2 where it is halted and TRUE assigned to BLOCK. This situation is analogous to a sensor being triggered on the turtle. A main procedureinterrupt occurs at line 10 and control is transferred to the designated interrupt procedure, INTER1.

INTER1 is executed recursively ninety times causing the turtle to turn right 90^o thus changing the value of BLOCK to FALSE (Fig. 20).

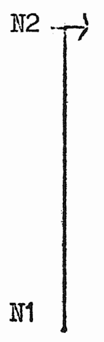

Fig. 20 Turtle at N2 headed 90^o with BLOCK=FALSE.

With the BLOCK condition now satisfied line 30 in INTER1 causes control to revert to SEARCH1.

The LOGO control-keyword COMPLETE causes the attempted completion of the interrupted instruction in SEARCH1 (line 10). Recall that the turtle has already successfully moved forward 240 steps. Therefore, the pseudo-instruction now attempting execution at line 10 is 'FD 240'.

If CONTINUE had been the control-keyword control would have passed to line 20 in SEARCH1. In the absence of any control-keyword a new call to SEARCH1 would have been initiated.

Pseudo-instruction 'FD 240' is now executed and Figs.21-23 illustrate the resulting actions.

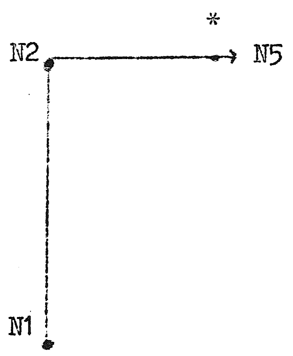

Fig. 21 Turtle is halted at N5 and control reverts to INTER1.

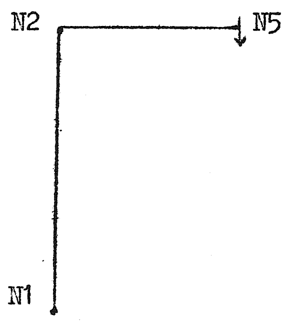

Fig. 22 Turtle rotates to a position of 180° causing BLOCK=FALSE and reverting control to SEARCH1.

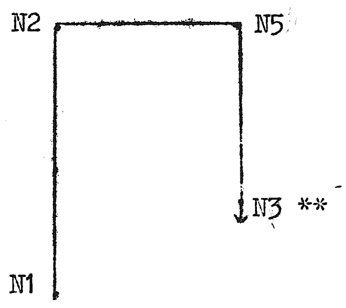

Fig. 23 Turtle moves to N3. ** signifies goal attained. SEARCH1 terminates on line 60 which halts the turtle.

What actions would result if paths N2→N5 and N1→N4 were 6 feet (72 steps) instead of 4 given the same starting and goal conditions?

Figs. 18-22 would remain the same but Figs. 24-26 would replace Fig. 23.

The control-keyword COMPLETE in line 30 of INTER1 causes the instruction of line 10 of SEARCH1 to be completed leaving the turtle 12 steps (one foot) short of the goal at N3. As a result line 20 in SEARCH1 is executed causing the turtle to turn 45° clockwise giving it a heading of 225°. The instruction in line 30 of SEARCH1 causes the turtle to attempt to back up 720 steps which precipitates an interrupt causing a halt and BLOCK=TRUE (Fig. 24).

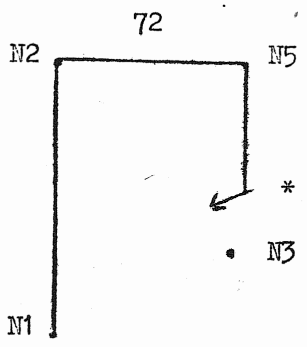

Fig. 24 Turtle is halted 12 steps short of goal at N3 headed 225°.

Control transfers to INTER1 which calls itself recursively 135 times resulting in Fig. 25 and BLOCK=FALSE.

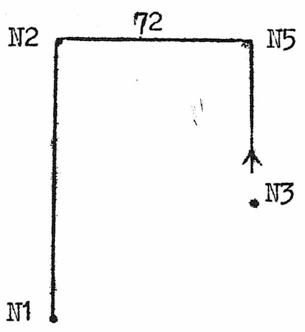

Fig. 25 Turtle heading 0° with BLOCK=FALSE.

Control reverts to the suspended instruction on line 30 in SEARCH1 whose pseudo-instruction is identical to its real instruction. Execution results in goal attainment in 12 turtle-steps (Fig. 26).

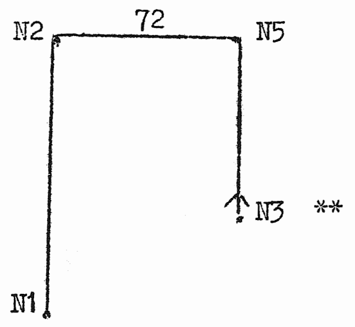

Fig. 26 Goal attained in line 30 of SEARCH1
Note the difference between the terminal heading in this figure and Fig. 23.

B. After creating a maze we may wish to alter it by adding, deleting and changing the directed nature of paths (Proc. 5, Fig. 27).

```
> ALTER MAZE1                     [calls ALTER program and applies
                                   it to MAZE1]
* DEL N2→N3                       [delete path N2→N3]
* ADD N3 90 20 ONEWAY N3→N6       [path is added]
* DIR N3→N1 ONEWAY FIN SUP        [make N3→N1 one-way]
* MAZE1 ALTERED
```

Procedure 5 A path in MAZE1 is deleted, one is added and one has its directed nature changed. The new maze confirmation print-out has been SUPpressed.

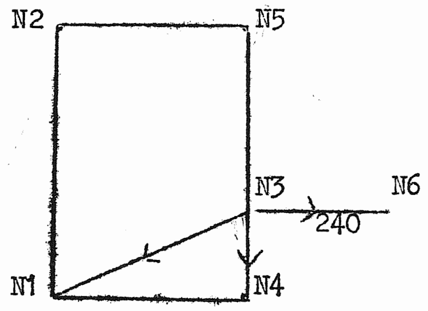

Fig. 27 New software model of MAZE1. A one-way path N3→N6 of length 240 turtle-steps and heading 90^{o} has been added. N2→N3 has been deleted and N1→N3 is now one-way in the N3→N1 direction.

Note that venturing along path N3→N6 in search of a goal could be disastrous for any self-respecting turtle. We shall see in a later section how a turtle might avoid this type of turtle-trap.

We may insert nodes into existing paths and grow paths from these nodes if we wish (Proc. 6, Fig. 28).

```
> ALTER MAZE1
* INSERT
* N7:  N1→N2 100 FIN SUP     [insert a new node N7 100
                              steps from N1 on path N1→N2]
* MAZE1 ALTERED
```

Procedure 6   A new node has been inserted 100 steps from N1 on path N1→N2.

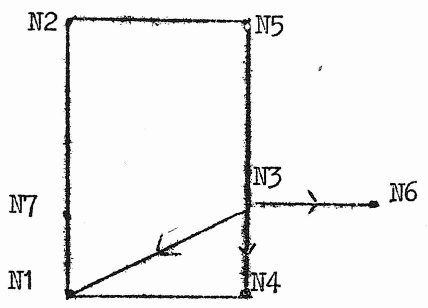

Figure 28   MAZE1 with new node N7.

C. We may wish to designate certain nodes as red-light nodes (NOGO nodes) for certain of their connecting paths (Proc. 7, Fig. 29).

```
> NOGO MAZE1                 [calls NOGO program and
                              applies it to MAZE1]
* N2→N5  N5                  [N5 is blocked in the N2→N5
                              direction.  It is still
                              possible to go from N5 to N2]
* N4→N1  N4                  [N4 is blocked in the N4→N1
                              direction]
* FIN SUP
* NOGO MAZE1 COMPLETED
```

Procedure 7   Blocking certain paths into certain nodes.

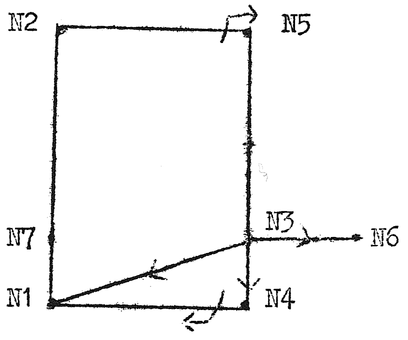

Fig. 29 N5 is blocked in the N2→N5 direction signified by ↱ and N4 is blocked in the N4→N1 direction by ↲. ↲↱ signifies a bi-directional block (barrier).

We may wish to place a barrier on a path (Proc. 8, Fig. 30).

```
> NOGO MAZE1
* N1→N7  N7 }
* N2→N7  N7 }  or * BARR N7
* FIN SUP
* NOGO MAZE1 COMPLETED
```

Procedure 8 Barrier placed at N7

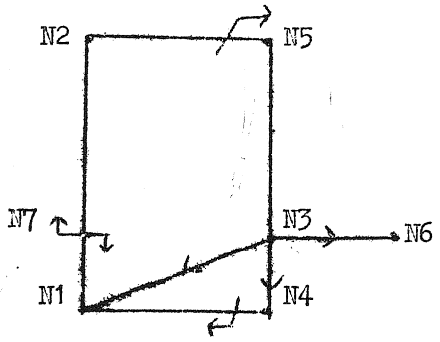

Figure 30 Barrier placed at N7. Equivalently: block at N1 in the N1→N7 direction and a block at N2 in the N2→N7 direction.

So far our alternative software turtle-maze poses few advantages over real world sensor devices. Notably one is the system's ability to keep records of the various mazes, procedures and paths traversed. The restriction of the turtle to established paths is a drawback but this is overcome in a later section.

We may of course have quite complicated mazes with multiple goals perhaps having some sort of priority ordering. But the environmental feedback and turtle responses are essentially the same in the software maze scheme as in the physical sensor scheme.

The first major advantages of our scheme can be seen in its facility for decision-point traversals.

III. Decision-point Traversal Maze

A. We may consider each node of the maze a decision-point. Instead of traversing the maze by successive turtle-steps the addition of four new instructions allows the turtle to move directly from node to node.

Procedure 4 applied to the graph represented by Fig. 31 would cause three nodes and four paths (including the path containing the goal) to be by-passed during the execution of the first instruction alone.

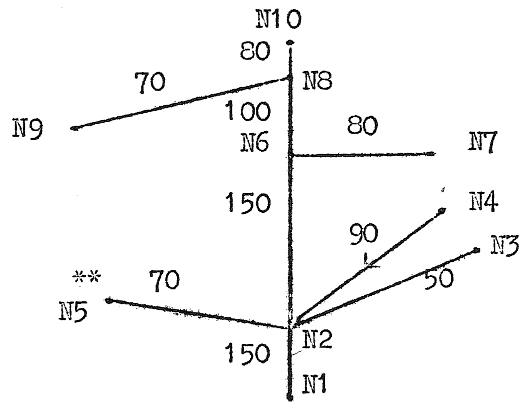

Fig. 31 MAZE2. The turtle starts at N1 headed 0^{o}. The goal is at N5 and the distance N1→N10 is 480 steps.

The addition of the following four turtle instructions (Proc. 9) to the instruction repertoire increases environmental feedback information.

FD DN	[forward to the next decision node]
BK DN	[backwards to the next decision node]
RT DN	[turn right until headed toward the next decision node]
LT DN	[turn left until headed toward the next decision node]

Procedure 9 Decision-point turtle instructions.

Path-overshoot cannot occur and control is transferred from the main procedure to the designated interrupt procedure only for the following reasons.

1. The turtle attempts to traverse a one-way path in the wrong direction. BLOCK set to ONEWAY.

2. The turtle attempts to traverse a one-way path in the correct direction. BLOCK set to ONEWAY→. This warns of a possible turtle-trap.

3. The turtle attempts to traverse a path blocked by a barrier, a NOGO red light or another turtle. BLOCK is set to BARR, NOGO, TURTLE, respectively. In the latter case a LOGO key-word TURTLENAME is set to the identifier of the encountered turtle.

Procedures 10 and 11 utilize MAZE2 to illustrate the use of these new instructions. Figs. 32-40 show the turtle's actions. Assume that a software model of MAZE2 has been created.

```
> TO TRAVERSE
> 10 FD DN
> 20 RT DN
> 30 TRAVERSE
> 40 STOP
> 50 PI:  INTER2
> 60 GOAL:  STOP
> END
```

Procedure 10 Main procedure.

```
> TO INTER2
> 10 IF BLOCK ONEWAY GO TO 40     } these instructions
> 20 IF BLOCK ONEWAY→ GO TO 40    } could be combined:
> 30 IF BLOCK TURTLE GO TO 70     } IF BLOCK GO TO 40 40 70
> 40 RT DN
> 50 TRAVERSE
> 60 STOP
> 70 TURTLE-ENCOUNTER    [calls procedure to deal with
                          a turtle encounter]
> 80 STOP
> END
```

Procedure 11 INTERrupt procedure.

We provide the software model with total information and then execute the main procedure.

```
> INITIALIZE MAZE2
* START:  N1 0
* GOAL:   N5


> TRAVERSE              [execute main procedure]
```

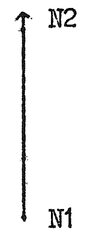

Fig. 32 Turtle travels to N2.

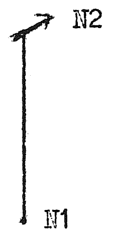

Fig. 33 Turtle turns right to path N2→N4.

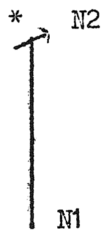

Fig. 34 Turtle blocked by one-way path. BLOCK=ONEWAY and control passes to INTER2.

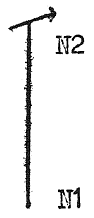

Fig. 35 Control passes to line 40 in INTER2 and turtle turns right to N2→N3. Control is passed back to TRAVERSE.

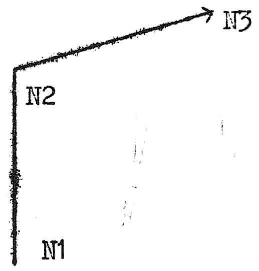

Fig. 36

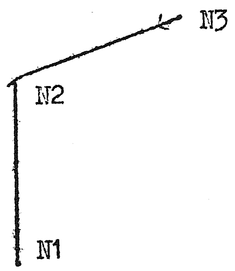

Fig. 37

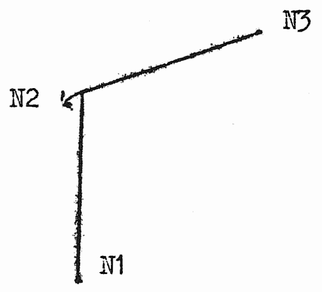

Fig. 38

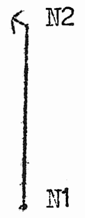

Fig. 39 Turtle turns right to N2→N5.

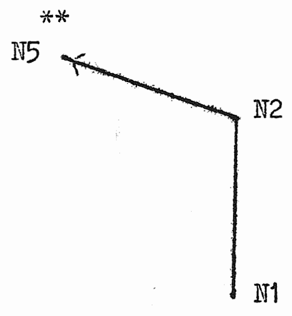

Fig. 40 Goal found.

The use of decision-point turtle instructions increases the turtle's sensitivity to its environment thus giving the user an additional margin of freedom. The turtle may be thought of as possessing path-sensitive eyes.

We may mix decision and non-decision-point turtle instructions.

B. We could allow the turtle an additional freedom at decision-point nodes by providing it with a list of path possibilities.

For example, when the turtle arrives at N2 (Fig. 32) the LOGO key-list VECTOR could be consulted.

VECTOR: (N4←45 N3↔80 N1↔180 N5↔240)

where ← signifies one-way in the N4→N2 direction and ↔ signifies two-way.

Additional instructions could be provided for choice and action.

IV. <u>Continuous Path-feedback and Correction</u>

In II and III (mixed mode) interrupts occur when the turtle strays from its path.

We allow path deviations by permitting user-set limits on and remedies for these deviations.

The user may set LOGO key-words PATHSTRAY, ANGLESTRAY and DISTANCESTRAY which cause procedure interrupts resulting in transfer of control to user created analyzer and corrective procedures.

For experimental reasons deviations may be introduced at various path regions by the experimenter.

With this feature, fixed turtle-paths are no longer restrictions on movement. In the extreme case (no STRAY-interrupts) the maze is a collection of unconnected nodes. These nodes represent obstacles (barriers) in the turtle's world.

V. Path Recognition

Consider the maze shown in Fig. 40 and Procedure 12.

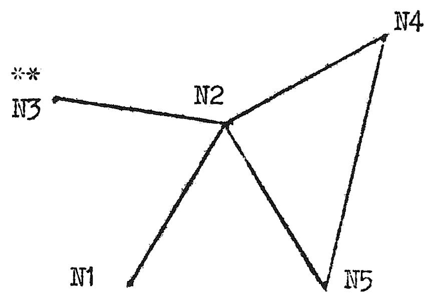

Fig. 40 MAZE3. Goal at N3 and turtle starts at N1.

```
> TO LOOP
> 10 RT DN
> 20 FD DN
> 30 LOOP
> 40 STOP
> 50 IP:  INTER2          [Proc. 11]
> 60 GOAL:  STOP
> END
```

Procedure 12 Procedure to search MAZE3 for goal.

We provide the software model of MAZE3 with total information and execute the main procedure.

```
> INITIALIZE MAZE3
* START:  N1              [no specific direction]
* GOAL:   N3

> LOOP                    [execute LOOP]
```

Procedure execution places the turtle into an endless loop: N2→N4→N5→N2→ ....

We may avoid such a catastrophy by providing the turtle with a path memory. In effect, the turtle would have the ability to mark each path traversed and to recognize such marks.

A LOGO key-list containing all paths previously traversed (and frequency) by the turtle in chronological order is provided. A LOGO key-word contains the name of the path about to be traversed. Key-list and key-word are MEMPATH and CURPATH, respectively, and can be accessed at any place in a procedure.

The following four instructions augment the repertoire of turtle instructions.

```
RT DNM          [after execution, control transfers
                 to designated MEMORY-Interrupt procedure]
LF DNM          [same as RT DNM]
FD DNM          [before execution, control transfers to
                 designated MEMORY-Interrupt procedure]
BK DNM          [same as FD DNM]
```

The designated MEMORY-Interrupt procedure would interogate MEMPATH, determine which path to pursue and appropriately transfer control to the main procedure or to some part of itself.

We may wish to provide a LOGO key-word, say PATH, which would be automatically set to a truth value depending on whether or not the current proposed path has been travelled previously.

Procedures 13 and 14 search MAZE 3.

```
>TO RECOGNIZE
>10 RT DNM
>20 FD DNM
>30 RECOGNIZE
>40 STOP
>50 MI:  TEST1    [local nonexecutable MEMORY-Interrupt
                   designation instruction]
>60 IP:  INTER2   [Proc. 11]
>70 GOAL:  STOP
>END
```

<u>Procedure 13</u>   Procedure to search MAZE3.

```
> TO TEST1
> 10 IF PATH GO TO 40    [if PATH=TRUE, line 40;
> 20 RECOGNIZE CONTINUE   if PATH=FALSE, continue
                          execution of RECOGNIZE]
> 30 STOP
> 40 RECOGNIZE
> 50 STOP
> END
```

Procedure 13   Procedure designated by MEMORY-Interrupt in RECOGNIZE.

```
> RECOGNIZE                [execute main procedure]
```

Figs. 41-48 illustrate the turtle's actions.

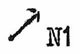

Fig. 41 Turtle is headed in N1→N2 direction after first instruction in RECOGNIZE.

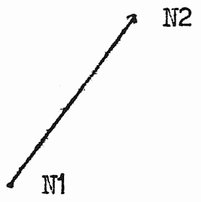

Fig. 42 Turtle moves to N2.

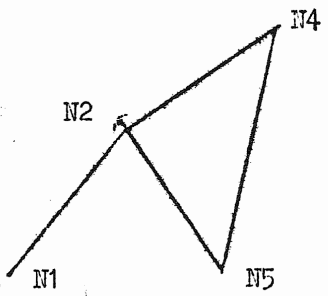

Fig. 43 Turtle traverses N2→N4→N5→N2 and is about to execute line 10 in RECOGNIZE again.

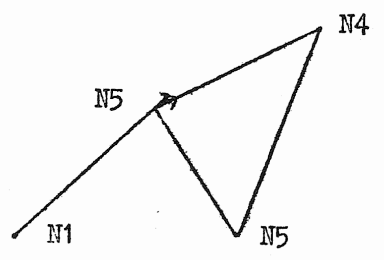

Fig. 44 Turtle turns toward N2→N4 path, PATH is set to TRUE and control transfers to line 10 in TEST1.

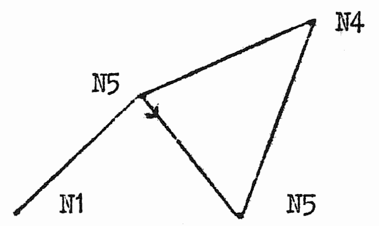

Fig. 45 Line 10 in RECOGNIZE is executed, then TEST1 is called.

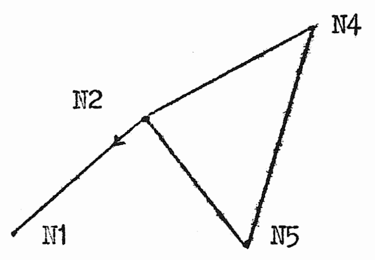

Fig. 46 Line 10 in TEST1 called line 10 in RECOGNIZE

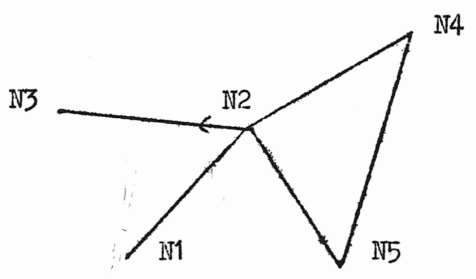

Fig. 47 Line 10 in RECOGNIZE causes this figure, however, line 10 in TEST1 causes control to go to line 20 in TEST1 which in turn causes control to be transferred to line 20 in RECOGNIZE which results in Fig. 48.

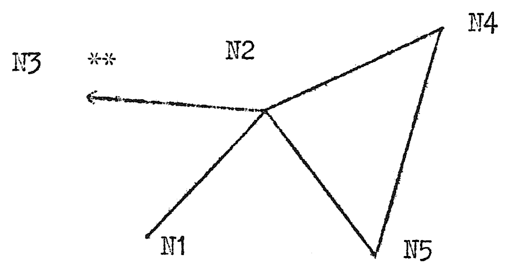

Fig. 48 Goal discovered.

Using the key-list and key-words provided the user may write pattern recognition procedures analyzing paths taken in order to develop strategies.

We would like the turtle to be aware of paths taken by mobile goals to enable the user to organize pursuit and intercept strategies.

VI. Mobile Goals and Pursuit Strategies

In our examples goals have been singular and stationary although provision is made for multiple stationary goals.

We would like to provide the pursuit turtle with knowledge (complete or partial) of the paths (pattern) traversed by mobile goals, i.e., a game of turtle-tag. For simplicity we consider only one mobile goal and one pursuit turtle. An alternative scheme would provide for multiple pursuit and goal turtles.

The pursuit turtle would have access to the goal turtle's key-list path chronology and current path key-word as well as a knowledge of the structure of the maze itself. If both turtles move at the same rate straightforward pursuit would be futile although we could provide for variations in turtle speed.

If the pursuit turtle could block a path travelled by the goal turtle or intercept it at a node a capture would result. The margin of capture could be varied in an analogous fashion to the range of light source and sensor receptivity.

In addition, communication between turtles could be established and counterstrategies employed.

VIII. Graphical Turtle-maze

Using a light-pen the user creates a maze on a CRT screen and designates starting and goal points.

Procedures written for the electromechanical turtle are interchangeable with the graphical turtle.

IX. Interactive-feedback Interrupts

So far interrupts in procedures merely passed control to other procedures.

Another form of interrupt transfers control to the user thus providing a dynamic interactive human-machine system.

X. Book-keeping

All interactions between user, LOGO System, turtle and environment are stored. This includes mazes, procedures, paths traversed and blocked. All actions are recorded in the user's dribble-file and are available for analysis.

XI. Computer Aided LOGO Analysis

A computer program may assist the experimenter in the analysis of user-system interactions.

Robert H. Seidman
August, 1972.