

The Last Whole MouseLab Manual

This interim documentation is produced for very limited circulation to those requiring the capabilities of MouseLab 6.0. The documentation may not correspond to the features of the provided program. Use at your own risk.

Produced: November 11, 1996, from file D:\DATA\MSS\MANUAL\MLMAST.WPD.

1 Background

Much current decision research attempts to identify the cognitive processes underlying judgment and choice (see Einhorn & Hogarth, 1981; Pitz & Sachs, 1984). To better understand those processes, investigators are increasingly adopting process tracing techniques to complement many traditional final output measures such as choice proportions, rankings, or ratings. This paper explains and documents a methodology which facilitates process tracing studies of information acquisition and decision making.

1.1 Alternative Process Tracing Methods.

The two process tracing methods of greatest interest to decision researchers have been: (1) verbal protocol analysis, and (2) the analysis of information acquisition behavior. Verbal protocol analysis takes as data the continuous verbal reports provided by a subject who is asked to "think aloud" while performing a decision task. An utterance at a particular point in time is taken to indicate knowledge or an operation at that time (Newell & Simon, 1972). A model of how people respond to instructions to think aloud is developed by Ericsson and Simon (1980; 1984). Ericsson and Simon argue forcefully for the validity of verbal protocols in providing a high density of observations of the intermediate stages of processing, and they provide numerous examples of the successful use of verbal protocols in the study of cognitive processes.

Information acquisition analysis is based on data concerning what information the subject seeks, the sequence of acquisition, how much information is acquired, and for what duration information is examined. Typically these data have been obtained from tasks where subjects acquire data from a matrix of several alternatives, each described on several attributes. Data on information acquisitions is important for several reasons. First, the evaluation strategies that have been proposed in the decision literature imply certain patterns of search (Payne, 1976). Second, information acquisition relates to the role of attention and memory in decision making, a relatively neglected area of study (Einhorn & Hogarth, 1981). Finally, understanding the amount, types, and pattern of information acquisition is important in the design of decision aids ranging from labels on packages to computer-based decision support systems. Further arguments for adopting verbal protocol analysis and the monitoring of information acquisition behavior in decision research are provided by Payne, Braunstein, and Carroll (1978) and Carroll and Johnson (1990).

In principle, verbal protocols can provide data on information search behavior as well as additional process data. Obviously, a person might report each item of information examined. An example of additional process data would be reports of recall from long-term memory. However, there are a number of reasons why monitoring information acquisition behavior might

complement or supplement verbal protocols. First, in some decision tasks, subjects will fail to provide a complete verbal report of all the information acquired. Complementing verbal protocols with a decision task structured so that subjects' information selections can be easily monitored ensures a more complete record of the information processing. Monitoring information acquisition is also a much easier form of process tracing. While the coding of information acquisitions is not always as straightforward as it might seem, it is generally easier than the coding of verbal reports. Russo (1978), in his critical evaluation and comparison of process tracing techniques, states that "the most important point about verbal protocols is that they are difficult to analyze formally" (p. 564). Finally, in spite of the arguments of Ericsson and Simon (1984), there is concern that generating a verbal report is a secondary task that must be performed along with the primary decision task (Russo, Johnson, & Stephens, 1989). Such a secondary task will certainly slow down the decision process substantially, and perhaps fundamentally alter the process. For these reasons, the monitoring of information acquisition behavior has seen widespread acceptance in decision research as a process tracing methodology.

If one wants to monitor information acquisition, what is the best technique to use? At one extreme are simple "information board" procedures. For example, in Payne (1976) the information board consisted of a matrix of envelopes attached to a sheet of cardboard. To obtain the value on a particular dimension for a particular alternative, the subject had to pull a card out of the appropriate envelope, turn it around, read the card, and place it back into the envelope. The value was printed on the back of the card. Similar types of information boards have been used successfully in a number of studies. However, a major criticism of information boards is their reactivity, due to the time and effort required to acquire a piece of information (Arch, Bettman, and Kakkar 1978). At the other extreme is the monitoring of information acquisition using sophisticated techniques for the recording of eye fixations. Russo and his associates have been the primary users of eye movement recording to study decision behavior. Russo and Doshier (1983) used a photoelectric sensing device and a computer for recording and analyzing the fixations. One disturbing feature of this methodology is that the eye-position sensor required that the subject's head be immobilized by a bite bar. However, Russo and Doshier state that subjects adjusted quickly to the apparatus, and they believed that it had no influence on the choice processes. Another factor that has limited the use of eye movement recording is the complexity and expense of the equipment. As Russo (1978) acknowledges, to record eye movements precisely requires a "quite expensive" system. Just and Carpenter (1984), who have used eye fixations in a variety of cognitive tasks from reading to problem solving, also view the cost of instrumentation and cost of training personnel as major limitations of the eye movement methodology. Yet, despite these problems, monitoring eye fixations has proved informative.

In one of the few efforts to directly compare various information monitoring techniques, Russo (1978) reports that eye movements typically required a minimum of 200 - 300 milliseconds. On the other hand, a physical retrieval of information from an information display board could involve 3 - 4 seconds. One consequence of this difference in acquisition time (effort) was that subjects seldom reacquired information in information display board studies. Russo (1978) reviews several information display board studies and his own work using eye-movement recording and reports that the reacquisition rate for information boards never exceeded 7% of all acquisitions. On the other hand, the reacquisition rate with eye-movements was as high as 56%. As Russo notes, differences in stimulus materials make comparisons across

studies somewhat imprecise. Nonetheless, the reacquisition rate is likely to be much higher with eye movements. More generally, information display boards yield a much sparser pattern of observations than eye-movement recording. Consequently, Russo argues that there is a danger that the less dense data set may yield misleading conclusions about a decision maker's strategies.

MOUSELAB is an alternative to simple information boards and to expensive and difficult to use eye movement recording. This methodology for monitoring information acquisition behavior uses computer graphics to display information which is accessed using a computer-based pointing device called a "mouse." The methodology comes close to the recording of eye movements in terms of speed and ease of acquisitions, while minimizing instrumentation cost and the difficulty of use for both subject and experimenter. The rest of this manual documents the methodology, with sections on using the mouse, the computer programs, the display options and features, and the output file.

1.2 The Mouse as Pointing Device.

There are a number of computer-based pointing or position entry devices such as a lightpen, joystick, directional cursor keys, and mouse. Of these devices, the mouse has three marked advantages (Card, Moran & Newell, 1983).

The first advantage of a mouse as a pointing device is ease of learning. Card, Moran, and Newell (1983) compared the mouse, joystick, and two key operated devices for selecting text on a screen. Although each of the devices yields improved performance with practice, research indicates that the mouse and joystick have a significantly faster rate of learning than using cursor keys. In our experience, even people unfamiliar with computers are relatively comfortable with the mouse after a brief training period of five to ten minutes.

A second major advantage of a mouse is its rapid movement. Card, Moran, and Newell found the mouse to be significantly faster than the other devices they tested. An earlier study by English, Englebart, and Berman (1967) found the mouse faster than a lightpen and several other devices. An analysis of the time to move the mouse from point to point suggests that this follows Fitts Law (Card, Moran, & Newell, 1983). They suggest that the time to move a mouse is primarily limited by the central information-processing capacities of the eye-hand guidance system. In other words, the major limitation in speed is due to the time it takes to think where to point, not in the movement of the mouse. Further analysis of the performance of the mouse indicated that it was within 5% of this optimal pointing device described by Fitts Law. We have done an analysis of one of our decision tasks using Fitts Law (see Appendix B), which indicates that subjects could move between information cells in less than 100 milliseconds, a figure we have occasionally observed with practiced subjects. Surprisingly, these times are of similar magnitude to eye-movements, or about 160 - 200 milliseconds per bit of information (Card, Moran, & Newell, 1983). This suggests that reactivity due to effort might be minimal.

The third advantage of the mouse is its error rate, the percentage of times that the device is used to select an incorrect item of information. Card, Moran, and Newell report that the mice have a significantly lower error rate than other devices.

Despite these advantages, the use of the mouse does have limitations. In order to precisely monitor what information is acquired at particular point in time, it is necessary to

structure the decision task so that only one item of information is visible at a time. This was done by setting up the decision tasks so that the relevant information is hidden in boxes on the screen until the subject moves the mouse to point at a box. At that time, the box opens and the information is revealed. This procedure eliminates the possibility of the subject's acquiring information from peripheral vision, as might be the case in a more normal visual information environment. Our task is more similar to those used in research in reading, which manipulates the display to remove any peripheral information from around the current eye fixation (Rayner, 1975; McConkie & Rayner, 1976). Apparently, such manipulations have little effect upon the overall ability of an individual to understand text. The manipulations, however, do seem to increase the average time of each fixation, and to diminish the length of the distance covered by each fixation. By analogy, the mouse may have the same effect on search patterns. Specifically, we might see fewer transitions between distant areas of the display. However, reacquisition of information is often seen with the mouse, suggesting that much of the reactivity discussed by Russo is not present.

Ultimately, of course, the comparison of eye-movement recording to a mouse-based information acquisition system will require a series of empirical studies. Nonetheless, the existing evidence suggests that with a mouse-based system the time and effort to acquire a piece of information from a computer display are relatively small. This should reduce the reactivity of studies that examine decision behavior through the monitoring of information acquisition behavior.

The next sections of this paper describe the software that utilizes a mouse to perform a variety of decision experiments.

2 Overview of the MOUSELAB Programs

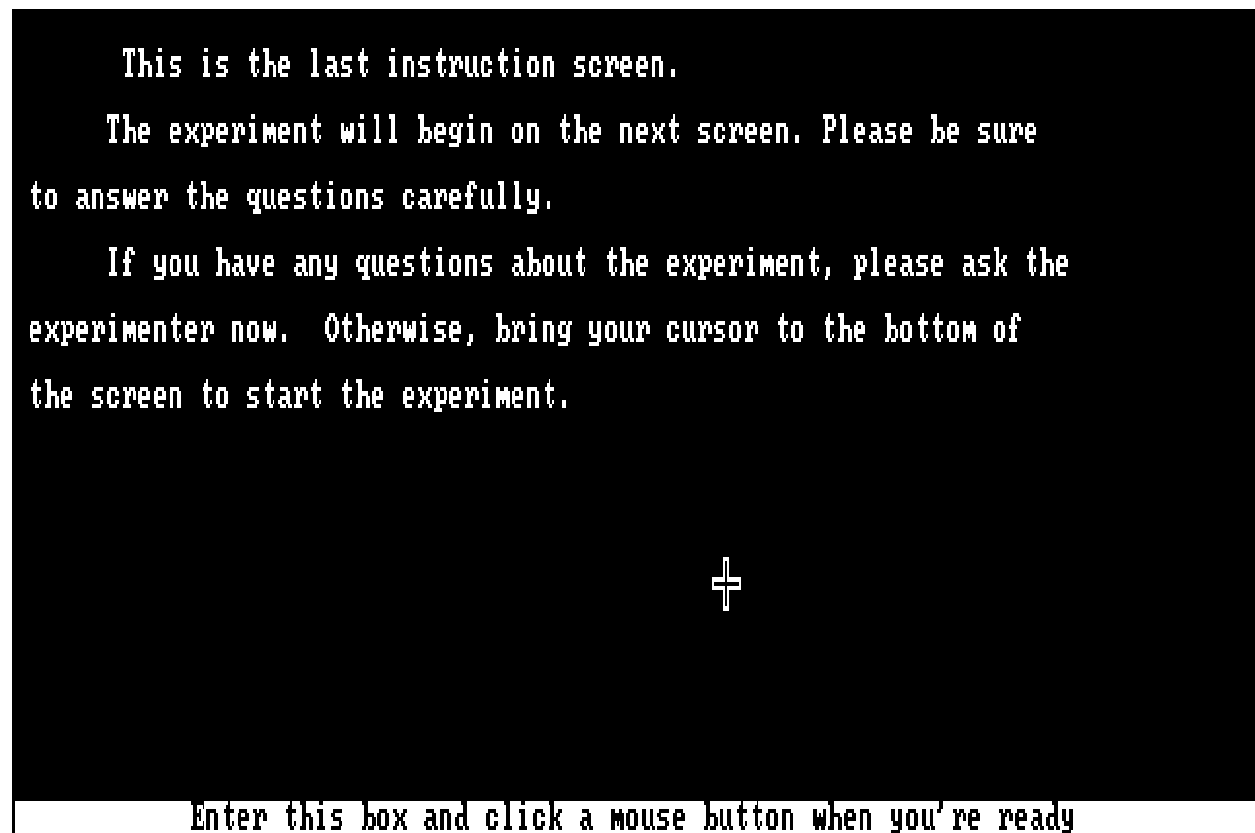
The core of the mouse-based system for doing decision studies is a program called MOUSELAB. The program can be used to present the instructions for an experiment, present decision problems using one of four general types of screens or "schemas," and automatically record what information was acquired, the duration of the acquisition, search order, and the final judgment or choice. Response times are recorded to an accuracy of 1/60th of a second. In addition to MOUSELAB, the decision laboratory software includes a program called BISECT that can be used in the analysis and reduction of the data generated from the process tracing studies. A final set of programs allow the randomization and counterbalancing of MOUSELAB input files. The programs that make up the mouse decision laboratory software are coded in Microsoft Pascal, Version 4.0, with some of the auxiliary programs for data analysis written in Borland's Turbo Pascal.

The MOUSELAB program is designed to run on an IBM-PC XT, AT, PS/2 or equivalent. Because of the real time processing load, the original IBM PC's system requires an Intel 8087 coprocessor to effectively run MOUSELAB. Faster processors do not require a coprocessor. A recommended system configuration would include an IBM-PC with 384K of free memory, a hard disk with at least 2 free megabytes of disk, and a Microsoft mouse. MOUSELAB will also work with other mice which emulate the Microsoft mouse. The computer monitor can be monochrome

or color. A graphics card is also required. Currently Hercules, IBM CGA, EGA, and VGA displays and compatibles are supported.

2.1 Display Options.

Each MOUSELAB screen or display has two major components. The SCHEMA type determines the manner in which information is displayed on the screen. The RESPONSE MODE determines the manner in which the subject is to respond to the information presented in the schema. Any schema type may be used in conjunction with any response mode (Tversky, Sattath, & Slovic, 1988). MOUSELAB 5.0 supports four different schemas and three different response modes. Each of these is now briefly described.



Display Example 1: Sample Text Schema

```

@BEGIN<TEXT>
@BEGIN<SCREENTEXT>
  This is the last instruction screen.

  The experiment will begin on the next screen. Please be sure
to answer the questions carefully.

  If you have any questions about the experiment, please ask the
experimenter now. Otherwise, bring your cursor to the bottom of
the screen to start the experiment.
@END<SCREENTEXT>
@END<TEXT>

```

Input Example 1: Input Format for Text Schema

2.2 Schemas

TEXT: A text schema simply presents lines of text that the researcher wishes to present to the subject (e.g., a set of instructions). Up to 24 lines of text can be presented on a screen. To present additional information on additional screens, the text schema is just evoked repeatedly. See Display Example 1 for sample text schema; Input Example 1 contains the MOUSELAB code used to create the text schema.

```

@BEGIN<FILE>
@BEGIN<MATRIX>
@SET<RESPONSELINE="Which house would you buy?">
@SET<ALTERNATIVES=3;ATTRIBUTES=3>
@SET<ALT[1]="House A";ALT[2]="House B";ALT[3]="House C">
@SET<ATTRIBUTE[1]="Cost";ATTRIBUTE[2]="Size";ATTRIBUTE[3]="Neighborhood">
@SET<BOX[1,1]="$700,000";BOX[1,2]="Small";BOX[1,3]="Downtown">
@SET<BOX[2,1]="$400,000";BOX[2,2]="Medium";BOX[2,3]="Suburbs">
@SET<BOX[3,1]="$150,000";BOX[3,2]="Large";BOX[3,3]="Country">
@END<MATRIX>
@END<FILE>

```

Input Example 2: Input Format for Matrix Schema

MATRIX: A matrix schema presents an M rows \times N columns matrix of boxes which can be opened to display information. Labels can be specified for the rows and columns. Boxes used to choose one of the alternatives appear at the bottom of the screen. The maximum size matrix

we have

| | Cost | Size | Neighborhood |
|---------|------|------|--------------|
| House A | | | |
| House B | | | Suburbs + |
| House C | | | |

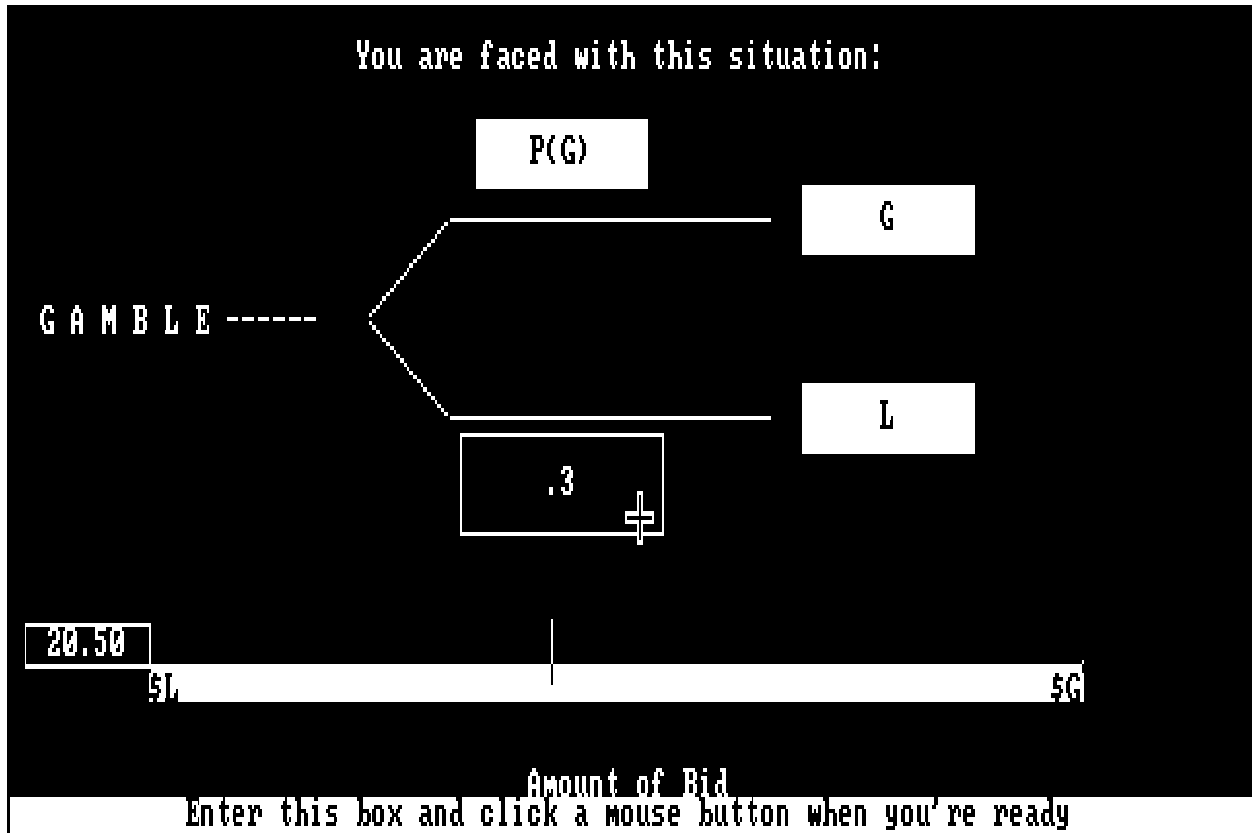
Which house would you buy?

Choose one: House A House B House C

House B was chosen. Enter this box and click once to continue.

Display Example 2: Sample Matrix Schema

used corresponded to an 8 alternative \times 8 attribute decision problem. This schema is most useful for non-risky, multiattribute decision problems. See Display Example 2. Input Example 2 contains the code used to create Display Example 2.



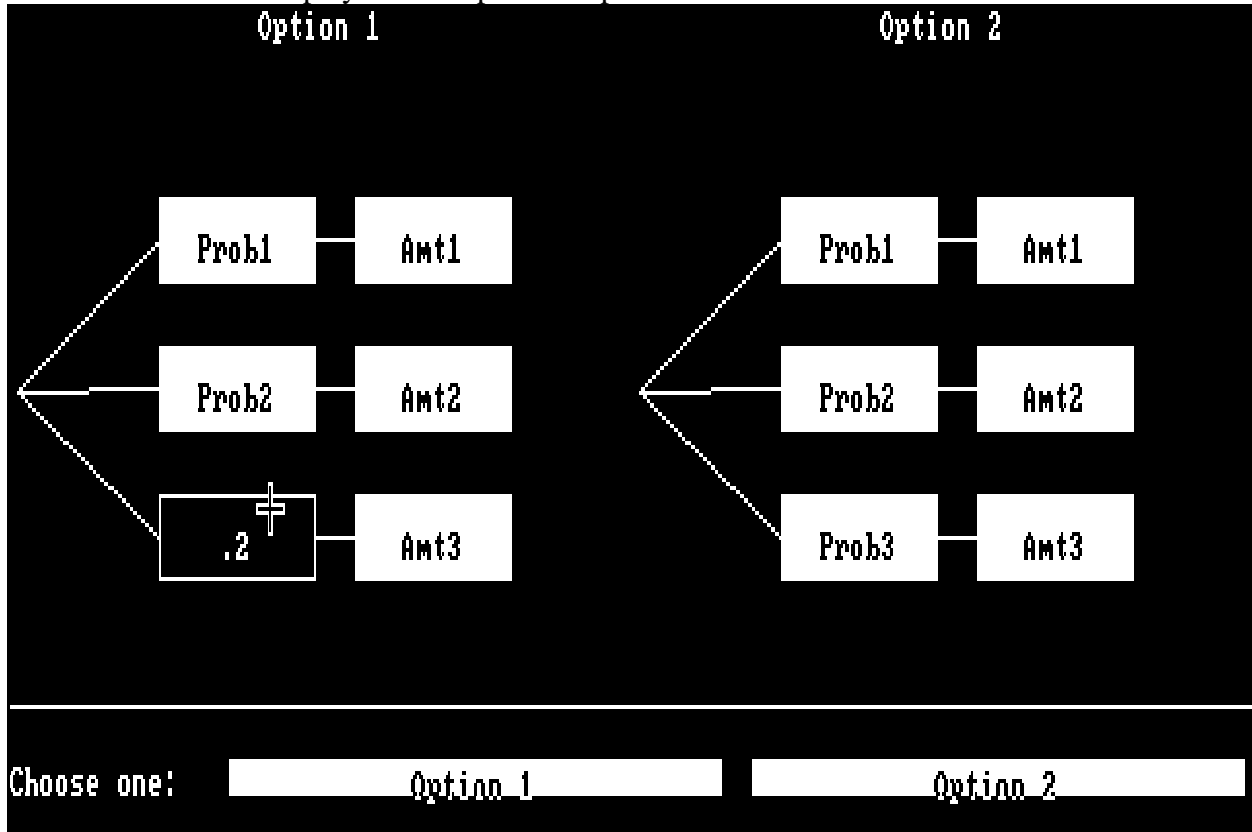
Display Example 3: Sample Gamble Schema

GAMBLE: A gamble schema presents a two-outcome gamble and a certain option in a format appropriate for either a probability equivalence or a certainty equivalence judgment. See Display Example 3. Input Example 3 contains the code used to create the gamble schema in Display Example 3. A study of preference reversals using this schema is reported in Johnson, Payne, and Bettman (1988).

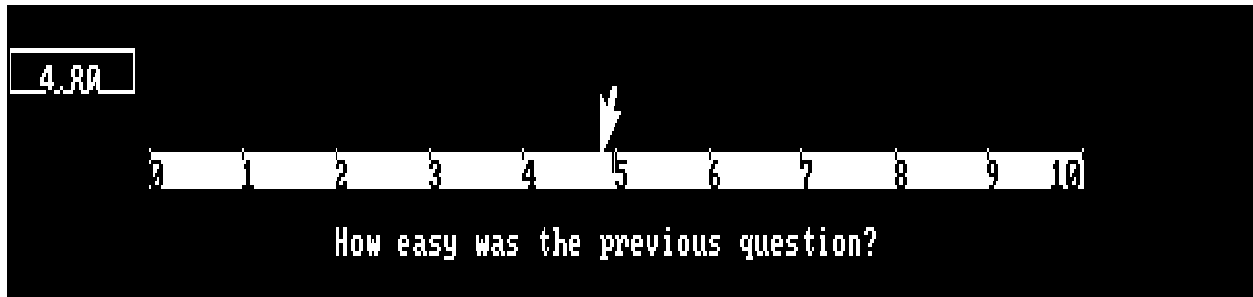
```
@BEGIN <FILE>
@BEGIN <GAMBLE>
@TITLE <"Demo Amount of Bid">
@SET <GAMBLEMODE = AOB; ANCHORMODE = OFF; ANCHORVALUE = 0>
@SET <HIGHPAY = "$33"; HIGHPAYPROB = ".7";
    LOWPAY = "$11"; LOWPAYPROB = ".3">
@SET <LABELS = 2>
@SET <PUTLABEL[1] = 0.0; LABEL [1] = "$L";
    PUTLABEL[2] = 1.0; LABEL [2] = "$G">
@SET <SCALELEFT=10;SCALEBOTTOM=20;SCALESIZE=60>
@END <GAMBLE>
@END <FILE>
```

Input Example 3: Input Format for Gamble Schema

MRC (Multiple Risky Choice): An MRC schema can be used to present any number of different gambles in a decision-tree format. The gambles in a set can have different numbers of outcomes and attributes, although each gamble in a set must have the same structure. This schema can also be used to present non-risky multiattribute problems by setting the number of outcomes equal to one. An example of a MRC display is shown in Display Example 4. The code used to create Display 4 is in Input Example 4.



Display Example 4: Sample MRC Schema



Display Example 5: Sample Response Scale

2.3 Response Modes

In any schema, an experimenter can choose between three different response modes: (i) response boxes, (ii) response scales, and (iii) keyboard input responses. Choices are made using the response boxes mode, while judgments are made using the response scale mode. The keyboard input response model can be used for choices or judgments.

BOXES: The boxes response mode presents a variable number of labeled choice boxes at the bottom of the screen. To choose a response, the subject simply moves the cursor into the desired response box and clicks to register a response.

```
@BEGIN<FILE>
@BEGIN <MRC>
@TITLE <"DEMO: MRC SCREEN">
@SET <GAMBLES = 2; OUTCOMES = 3; ATTRIBUTES = 2>
@SET <ORIENTATION = HORIZONTAL>
@SET <GAMBLE[1] = "Option 1"; GAMBLE[2] = "Option 2">
@SET <BOXLABEL[1,1,1] = "Prob1"; BOXLABEL[1,2,1] = "Amt1";
    BOXLABEL[2,1,1] = "Prob2"; BOXLABEL[2,2,1] = "Amt2";
    BOXLABEL[3,1,1] = "Prob3"; BOXLABEL[3,2,1] = "Amt3">
@SET <BOXLABEL[1,1,2] = "Prob1"; BOXLABEL[1,2,2] = "Amt1";
    BOXLABEL[2,1,2] = "Prob2"; BOXLABEL[2,2,2] = "Amt2";
    BOXLABEL[3,1,2] = "Prob3"; BOXLABEL[3,2,2] = "Amt3">
@SET <BOX[1,1,1] = ".2"; BOX[1,1,2] = "$65";
    BOX[1,2,1] = ".6"; BOX[1,2,2] = "$5";
    BOX[1,3,1] = ".2"; BOX[1,3,2] = "-$50";
    BOX[2,1,1] = ".5"; BOX[2,1,2] = "$40";
    BOX[2,2,1] = ".3"; BOX[2,2,2] = "$0";
    BOX[2,3,1] = ".2"; BOX[2,3,2] = "-20">
@END <MRC>
@END<FILE>
```

Input Example 4: Input Format for MRC Schema

```

@BEGIN<FILE>
@BEGIN<TEXT>
@SET <RESPONSEMODE = SCALE>
@SET <SCALELEFT = 10; SCALEBOTTOM = 12; SCALESIZE = 60>
@SET <LABELS = 11; ANCHORMODE = ON; ANCHORVALUE = 5>
@SET <PUTLABEL[1] = 0.0; LABEL[1] = "0";
    PUTLABEL[2] = 0.1; LABEL[2] = "1";
    PUTLABEL[3] = 0.2; LABEL[3] = "2";
    PUTLABEL[4] = 0.3; LABEL[4] = "3";
    PUTLABEL[5] = 0.4; LABEL[5] = "4";
    PUTLABEL[6] = 0.5; LABEL[6] = "5";
    PUTLABEL[7] = 0.6; LABEL[7] = "6";
    PUTLABEL[8] = 0.7; LABEL[8] = "7";
    PUTLABEL[9] = 0.8; LABEL[9] = "8";
    PUTLABEL[10] = 0.9; LABEL[10] = "9";
    PUTLABEL[11] = 1.0; LABEL[11] = "10">
@SET<RESPONSELINE="How easy was the previous question?">
@END<TEXT>
@END <FILE>

```

Input Example 5: Input Format for Response Scale

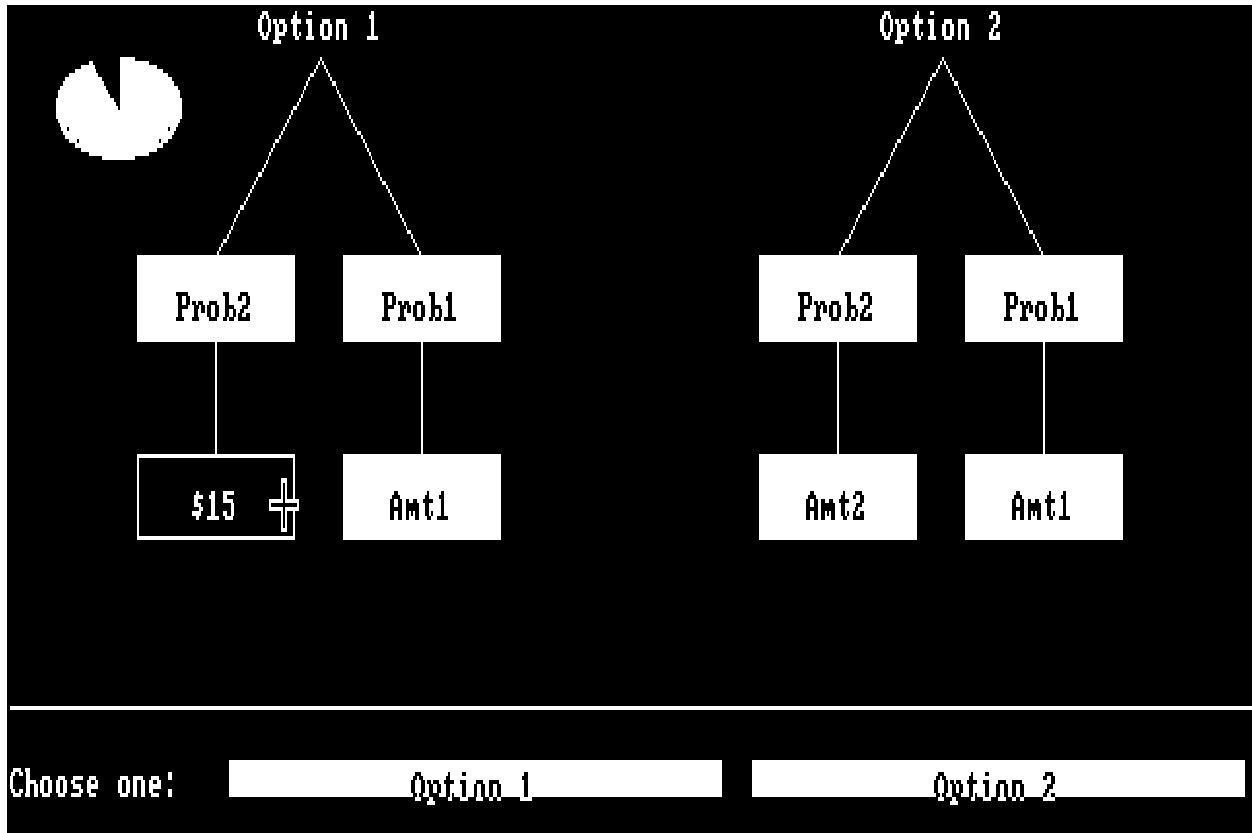
SCALE: The scale response mode presents a horizontal (Likert-type) scale for responses. Scale labels and the numbered divisions of the scale can be specified. See Display Example 5 and Input Example 5.

KEYBOARD INPUT: The keyboard input response mode allows subjects to type in a real number as their response to a screen. The number must be a real number less than approximately 32000 in absolute value. The keyboard input response mode may be used with matrix or MRC schema types.

All response modes provide for a one line message either immediately above or below the response area. This is sometimes useful for clarifying the meaning of possible responses.

2.4 Features

MOUSELAB also supports some general features that can be used with the various displays. They are: (i) time pressure, (ii) move checking, (iii) open boxes, and (iv) graphics.



Display Example 6: Sample MRC Schema with Time Pressure

```

@BEGIN<FILE>
@BEGIN <MRC>
@TITLE <"DEMO: MRC SCREEN">
@SET <CLOCK=ON;TIME=60.0>
@SET <GAMBLES = 2; OUTCOMES = 2; ATTRIBUTES = 2>
@SET <ORIENTATION = VERTICAL>
@SET <BEEPSTATUS = ON>
@set <ENDSTATUS = CLOSEDCHECK>
@SET <GAMBLE[1] = "Option 1"; GAMBLE[2] = "Option 2">
@SET <BOXLABEL[1,1,1] = "Prob1"; BOXLABEL[1,2,1] = "Amt1";
  BOXLABEL[2,1,1] = "Prob2"; BOXLABEL[2,2,1] = "Amt2">
@SET <BOXLABEL[1,1,2] = "Prob1"; BOXLABEL[1,2,2] = "Amt1";
  BOXLABEL[2,1,2] = "Prob2"; BOXLABEL[2,2,2] = "Amt2">
@SET <BOX[1,1,1] = ".4"; BOX[1,1,2] = "$25";
  BOX[1,2,1] = ".6"; BOX[1,2,2] = "$15";
  BOX[2,1,1] = ".7"; BOX[2,1,2] = "$10";
  BOX[2,2,1] = ".3"; BOX[2,2,2] = "$30">
@END <MRC>
@END<FILE>

```

Input Example 6: Input Format for MRC Schema with Time Pressure

TIME PRESSURE: Time pressure is implemented by depicting an analog clock that counts down for a specified number of seconds. The clock is displayed on the screen along with the information contained in a schema. When the allotted time has expired, the request "Please make a choice or indicate a value" is displayed at the top of the screen. Several other things can happen to the screen, depending on what is specified. For example, after the clock has finished, the information boxes can be closed so that no additional information can be acquired, and the subject must make a decision based on only the information acquired up to that point. See Display Example 6 and Input Example 6 for a sample MRC schema with a clock. The time pressure feature was used in a study of adaptive decision behavior by Payne, Bettman, and Johnson (1988).

MOVE CHECKING: On occasion, an experimenter will want to make sure that subjects acquire information in a certain order, perhaps to ensure that a certain decision strategy is followed. One could, for example, use the move-monitoring feature to ensure that a subject used an elimination-by-aspects strategy (Tversky, 1972), which suggests search by attribute. To implement this feature, the experimenter must specify the desired sequence of boxes. If the subject enters any box out of sequence, the box will not open and the PC will emit a beep and record in the output file the time and location of the cursor. Bettman, Johnson, and Payne (1990) provide an example of how this feature might be used to estimate the cognitive effort associated with various decision strategies.

| | Cost | Size | Neighborhood |
|---------|-----------|--------|--------------|
| House A | \$700,000 | Small | Downtown |
| House B | \$400,000 | Medium | Suburbs |
| House C | \$150,000 | Large | Country |

Which house would you buy?

Choose one House A House B House C

House C was chosen. Enter this box and click once to continue.

Display Example 7: Matrix Schema with Open Boxes

```

@BEGIN<FILE>
@BEGIN<MATRIX>
@SET<BOXES=ON>
@SET<RESPONSELINE="Which house would you buy?">
@SET<ALTERNATIVES=3;ATTRIBUTES=3>
@SET<ALT[1]="House A";ALT[2]="House B";ALT[3]="House C">
@SET<ATTR[1]="Cost";ATTR[2]="Size";ATTR[3]="Neighborhood">
@SET<BOX[1,1]="$700,000";BOX[1,2]="Small";BOX[1,3]="Downtown">
@SET<BOX[2,1]="$400,000";BOX[2,2]="Medium";BOX[2,3]="Suburbs">
@SET<BOX[3,1]="$150,000";BOX[3,2]="Large";BOX[3,3]="Country">
@END<MATRIX>
@END<FILE>

```

Input Example 7: Input Format for Matrix Schema with Open Boxes

OPEN BOXES: This option uncovers all boxes within a given screen, displaying their normally hidden contents. This feature is useful for experimental validation and for instructional purposes, e.g. to familiarize subjects with the experimental stimuli before introducing the task of acquiring stimuli. Display Example 7 and Input Example 7 contains a sample of a matrix with open boxes.

GRAPHICS: In addition to typing in values for boxes in a matrix of MRC schema, the

experimenter can "call in" graphics, such as figures and graphs, from external files. These files which are stored in a PCX format can serve as the basis for subjects' choices and judgments.

Other features are available which modify the look of schemas:

DATA TRACING: The DATA TRACING option allows the researcher to suspend tracing and recording of subject activity within a display. This option is also useful for instructional screens, where there may be no need to monitor subject's behavior.

TRANSPOSE: The transpose option allows a matrix schema to be rotated so that rows become columns and vice-versa. For example, the standard form of display for the matrix schema is to consider the rows as alternatives and the matrix columns as attributes. With the transpose feature, the same input format is used, but the columns become the choice alternatives while the rows become attributes. The transposition feature can be used to make the display look better or to counterbalance the possible impact of the normal left-right reading order on search behavior.

ORIENTATION: The orientation feature does for MRC schemas what TRANSPOSE does for matrix schemas, displaying the gambles in either horizontal or vertical orientation.

SCALE TRACKING: Normally, when a subject is manipulating a scale, MOUSELAB only records the value of the scale each time the subject changes its value. Scale tracking allows the researcher to monitor the scale adjustment, recording the time and scale value each time the subject changes the direction of adjustment.

SUPPRESSED ALTERNATIVES: This feature works with the matrix and MRC schema to prevent row labels from appearing as choice selections. For instance, if the first row of a matrix was needed to reflect attribute weights, then this option would be used to make sure that "weights" was not one of the choice boxes at the bottom of the screen.

2.5 The MOUSELAB Programming Language

New to Version 5.0 of MOUSELAB is a fairly complete programming language. This allows the experimenter to alter the display according to the responses of the subject. The language both provides certain predeclared variables which contain information about how the subject behaved on each trail, and allows experimenters to declare their own variables as well.

2.5.1 Pre-declared variables

Pre-declared variables allow the experimenter to access data from a previous screen. These variables are either non-array-type variables or array-type variables. For example, the non-array-type variable %LRESPONSE retrieves the subject's choice from the previous screen. This information can be used to determine which alternatives will be presented in the current decision matrix.

2.5.2 Experimenter variables

The experimenter may now create his/her own variables in the input file. For example, the experimenter could create non-MOUSELAB variables like NAME and AGE. When the subject is prompted to type in his/her name and age, the MOUSELAB program records the responses in the output file.

The variables can also be assigned values, used in SET statements and aid in the use of flow-of-control. An example of this use would be the creation of a new variable, CLOCK, which determines the amount of time a subject can spend in a given matrix screen as a function of time spent in earlier screens. The experimenter can "set" the clock time to any desired length of time by assigning the variable a time value based on an earlier decision.

2.5.3 Flow of Control

Flow-of-control is a set of MOUSELAB commands that control the order in which MOUSELAB commands are executed. In previous versions of MOUSELAB, the statements were executed in sequential order. These statements can be used to execute the same set of MOUSELAB statements several times or to conditionally execute a set of MOUSELAB statements. Version 5.0 provides a fairly complete set of control constructs similar to any block oriented programming language.

In addition, the MOUSELAB language provides a fairly complete set of arithmetic operations and an assignment statement.

3 Running MOUSELAB

The MOUSELAB program runs under DOS. To run MOUSELAB, the researcher must first make sure that all the files on the MOUSELAB distribution diskette are present in the current DOS directory. The user then types MOUSELAB and hits the enter key.¹ The MOUSELAB program will provide prompts for four items of information. First, it displays a prompt COMM: which requires the communications port is used for the mouse. For most systems, the response would be either 1 or 2, depending on the serial port to which the mouse is connected. With a Microsoft bus-type mouse or PS/2 mouse port, the response would be 32. If you use a PS/2 system and wish to use a serial mouse, then the mouse must emulate a Microsoft mouse and you must use a COMM code of 32. See your mouse manual for details of Microsoft Mouse emulation. The program will next ask for the type of graphics card installed in the PC. In response to the CARD: prompt, you can specify any of the following video standards supported by your machine: Hercules (specified as HER), CGA, EGA, VGS, or VGA. Third, the program will ask for the name of the main input file (e.g., demo.inp). Finally, the fourth prompt requests the name of the data output file to contain the data which MOUSELAB creates while monitoring subject behavior (e.g., demo.out). For the files, any nonreserved file name or extension can be specified: ".inp," and ".out" are just conventions we have found useful. One can also specify device filenames such as "prn" (see the IBM DOS manual for further details) or "nul". WARNING: "Nul" should be used as a response to the fourth prompt ONLY when you do not want to record data, because no output file will be created. **Table I** is an example of starting MOUSELAB.

¹. MOUSELAB.BAT is actually a batch file which installs METASHEL graphics shell, starts the actual MOUSELAB program, and exits the graphics shell once MOUSELAB is finished.

Table I: Invoking MOUSELAB (User input in bold).

```
MetaSHEL - MetaWINDOW runtime graphics driver
Copyright (c) 1984-1989, Metagraphics Software Corp.
MetaWINDOW Ver 3.5A for DOS 86/REAL
DOS Version: 3.30
```

```
Invoking application . . . . true
COMM: 32
CARD: vga
BOXFILE: demo.inp
DATAFILE: demo.out
```

The MOUSELAB program then reads the specified input file to make the next screen. When the subject has finished with a screen, the program writes to the output file and reads the input file to generate the next screen of information. This process continues until all the screens defined in the input file have been displayed.

MOUSELAB creates a file called "debug.mli." The MLI extensions stand for MOUSELAB Internal. The "debug.mli" file will contain useful information when an output file contains an error.

Once MOUSELAB has started, CONTROL-BKSP (backspace) will abort MOUSELAB, and the message "voluntary abort of MOUSELAB" will appear. If CTRL-BKSP is hit in mid-screen, the abort will not take place until the end of the screen. This is useful to exit MOUSELAB programs while debugging, yet sufficiently obscure not to be found by subjects.

4 Making MOUSELAB Displays: Schemas and Features

The preceding section of the paper provided an overview of what the MOUSELAB program can do by demonstrating some of the displays that can be constructed using combinations of schemas, response modes, and options. In the next two sections we provide the details of the format and syntax of the input files used by MOUSELAB, along with a discussion of some of the features of MOUSELAB's programming language.

For simplicity, we organize our discussion in two parts. This section presents the basics of MOUSELAB, concentrating on how to construct displays to be presented to subjects. With the commands described in this section, the experimenter can run experiments where a static set of screens are presented to subjects in a fixed order. The next section will describe the MOUSELAB programming language in more depth, allowing the experimenter to change MOUSELAB's behavior in response to the subject's input.

4.1 General Input Information

MOUSELAB input files are written using a special high-level language which specifies the characteristics of each display. This input language consists of four principal elements. These are **commands**, **schemas**, **variables**, and **values**. **Commands** are always an action to be performed and are always preceded by an @ symbol. These include assignment statements, and commands that control the flow of a MOUSELAB program. **Schemas** as described above are the basic units of a display, for example a gamble or matrix display. **Variables** can be thought of as slots that hold the different types of information that describe a display, including whether various features are on or off. **Values** are the actual information. This information, depending on the variable (or command in those cases in which a command operates directly on a value), can be in the form of numbers, text strings, or special keywords. In more formal terms, the MOUSELAB language is a typed language: The programmer must specify the type of a variable when it is declared, and be aware of the type of values required by a command.

The variables and/or values on which the command is operating (its arguments) are always enclosed in angle brackets < >. Text strings are usually enclosed in quotation marks. A typical input line might look like this:

@SET<LABELS=3>

"@SET" is the command, "LABELS" is the variable, and 3 is the value assigned to the variable. Let's look at six basic commands in the MOUSELAB input language which will be used in almost any input file.

This language is free format (with one exception which will be discussed later) and is case insensitive. In practice, this means that spaces and line breaks are ignored and any combination of upper and lower case letters may be used. For the sake of clarity, however, this document will use upper case for all commands in its examples. In all descriptions of the syntax, lower case will stand for variable names, and alternative arguments are separated by a vertical bar "|" which can be translated as "or." The input files can be created by using any text editor or word processor which is capable of producing normal ASCII output files.

When MOUSELAB detects a syntax error in an input file, it will halt execution and display a message to that effect, identifying the guilty term and indicating its location in the input file. While MOUSELAB checks for proper syntax, it does not detect errors of omission. Failing to provide information that is needed by a schema will produce an unusual screen to say the least.

4.1.1 The BEGIN Command

Syntax:

@BEGIN<*schema_name*|*FILE*|*SCREENTEXT*>

The BEGIN command indicates the beginning of a new input file, schema, or display, depending on the argument used.

Example:

@BEGIN<*FILE*>.

4.1.2 The END Command

Syntax:

@END<*schema_name*|*FILE*|*SCREENTEXT*>

The END command indicates the end of an input file, a display specification, or a body of screen text. Each END statement however, must have a corresponding BEGIN command (and vice versa).

Example:

@END<*MATRIX*>

4.1.3 The COMMENT Command

Syntax:

@COMMENT<*Any text string*>

The COMMENT command indicates that the enclosed text is a comment and not part of a display. This comment is useful for documenting input files.

Example:

@COMMENT<**A comment does not have to be in quotes.**>

4.1.4 The TITLE Command

Syntax:

@TITLE<"*Any string which is output to the output file*">

The TITLE command sends the enclosed text to the output file as the title of the screen in which it is used. This is useful for labeling the output from experiments. Any text within a TITLE command must be in quotation marks.

Example:

```
@TITLE<"Part 2, Trial 5">
```

4.1.5 The SET Command**Syntax:**

```
@SET<variable=value/expression>
```

The SET command assigns values to variables and is therefore the most frequently-used command. Any number of assignments can be made with a single SET command as long as they are separated by semicolons.

Example:

```
@SET<LABELS=5>
```

```
@SET<SCALELENGTH=50> which is the same as
```

```
@SET<LABELS=5;SCALELENGTH=50>
```

4.1.6 The RESET Command**Syntax:**

```
@RESET
```

RESET command resets a particular group of variables, referred to as local variables, to their original default values. Local variables are explained in greater detail below.

Variables in MOUSELAB can be divided into two classes depending whether or not they apply just in this schema (a local variable) or across schemas (a **global** variable). Local variables control certain frequently used MOUSELAB features. When MOUSELAB starts, these variables are automatically assigned default values. If these values are changed within a display, the change is local to that display. After MOUSELAB has finished processing that display, the values of these variables are automatically restored to their original default values. At some point, the user may wish to change the default value of a local variable. To do this, the local variable must be reassigned by a SET command in between displays instead of within a display. The variable will then retain this new default value until the end of the input file, or until the RESET command is used. The RESET command restores all local variables to their original default values. This capability allows the user to avoid constantly reassigning values to these variables. Note also that the RESET command cannot be used for individual local variables -- once the RESET command is used, all local variables return to their original default value.

Global variables, on the other hand, will retain their assigned values until the user reassigns new values to them. Retained global values are most useful when identical schemas occur consecutively. For example, if the user specifies a matrix schema followed by another matrix schema, he/she can assume that all of the global variables from the previous schema retain their previous values. Thus, the user does not have to respecify the matrix. But, if identical

schemas do not occur back to back, global values will not be retained. For example, if the user specifies a matrix schema followed by a gamble schema followed by another matrix schema, the values specified for the first matrix schema will not be retained for the second matrix schema. (The user will have to fully specify the second matrix schema.) Note that this problem does not occur if the same schema is used with different response modes. As we proceed, we will indicate which variables are local variables. Any variables not explicitly identified as local variables are global variables.

4.1.7 Program Structure

The first command in an input file is always the `@BEGIN<FILE>` command. Likewise, the last command in an input file is always the `@END<FILE>` command. Between these two commands, any number of displays may be defined.

Two commands may only be used at certain points in the input file. The `RESET` command may only be used between display definitions. The `TITLE` command may only be used within display definitions and only once within each definition. The `COMMENT` and `SET` commands may be used at any time.

4.2 Schema Types

4.2.1 Text Schemas

A text schema display definition would start with the `@BEGIN<TEXT>` command. To specify a screen of text to be displayed, the user would insert an `@BEGIN<SCREENTEXT>` command. Starting on the following line, the user can specify up to 24 lines of text to be displayed. This text will be displayed exactly as it appears in the input file. On the line immediately following the last line of text to be displayed, the user places an `@END<SCREENTEXT>` command. This must be the only text on this line, otherwise MOUSELAB will interpret it as another line of display text. Following any additional commands (specifying a response mode, for example), the display definition would end with an `@END<TEXT>` command.

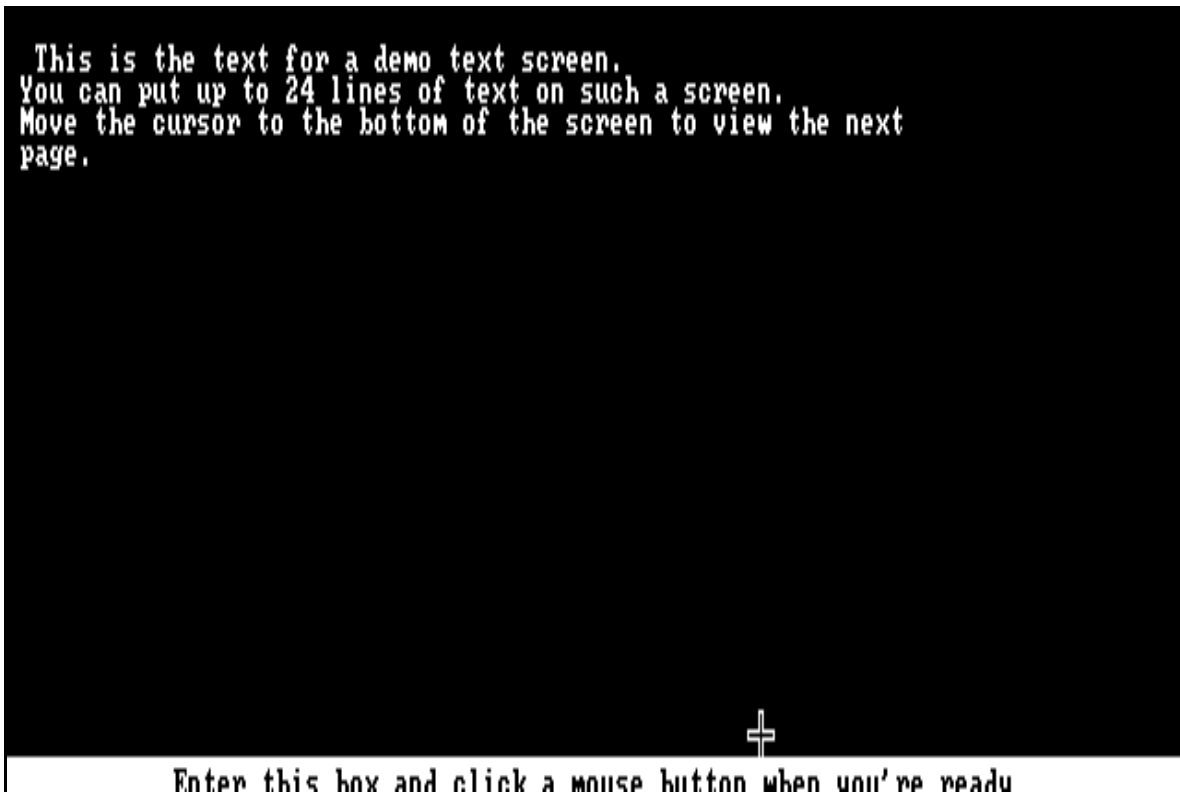
Screen text is always specified as described above and is the one exception to the free format rule. In addition, there is no default response mode for a text schema. This is the only schema which can be used without a response mode. If no response mode is designated, the message "Enter this box and click a mouse button when you're ready" is placed at the bottom of the screen automatically. This enables subjects to continue on to the next display after having read the text contained in the screen. If an experimenter desires a response mode within a text schema, the response mode must be specified between the "`@END<SCREENTEXT>`" and the "`@END<TEXT>`" commands. See Display Example 8 and Input Example 8 for a sample text schema.

```

@BEGIN <FILE>
@BEGIN <TEXT>
@TITLE <"DEMO TEXT SCREEN">
@COMMENT <THIS IS THE TEXT FOR A DEMO TEXT SCREEN.>
@BEGIN <SCREENTEXT>
This is the text for a demo text screen.
You can put up to 24 lines of text on such a screen.
Move the cursor to the bottom of the screen to view the next
page.
@END<SCREENTEXT>
@END <TEXT>
@END <FILE>

```

Input Example 8: Input Format for Text Schema



Display Example 8: Sample Text Schema

4.2.2 Matrix Schemas

The matrix schema is one of the most powerful schemas supported by the MOUSELAB program. Consequently, the matrix input specification is a bit more complex. However, a matrix display definition still conforms to the same general pattern as all other schemas.

Like other schemas, a matrix display definition begins and ends with BEGIN and END

commands with the appropriate argument. In this case, the commands are @BEGIN<MATRIX> and @END<MATRIX>. Since MOUSELAB schemas are nonprocedural, the information describing the schema can be provided in any order **within** the schema in any order using the SET command.

In order to process a matrix display definition, the program must know the size of the matrix. The alternatives of the matrix correspond to its rows. The command @SET<ALTERNATIVES = 4> tells MOUSELAB that there are to be four rows in this matrix. The number 4 in the command could be replaced with any positive integer. The attributes of the matrix correspond to its columns. The command @SET<ATTRIBUTES = 3> indicates there will be three columns. Again, any positive integer may be substituted for the number used here. There are no formal limits on the size of matrices. Practical limits, however, result from the size and resolution of your screen. We suggest you experiment to get a feel for these limits. For many displays a 6 X 8 matrix is a practical maximum.

Of course, MOUSELAB also needs labels for the alternatives and attributes. The command @SET<ALTERNATIVE[1] = "Newport"> indicates that alternative number 1 is Newport. Likewise, the second alternative could be specified by @SET<ALTERNATIVE[2] = "Chapel Hill">. As previously discussed, indices for variables must be integers and must be enclosed by square brackets. The labels for the attributes can be specified in similar fashion. For example, @SET<ATTRIBUTE[1] = "Costs"> indicates that the first attribute is Costs. Moreover, because of MOUSELAB's ability to recognize abbreviations, the labels can be specified more conveniently by commands such as @SET<ALT[1] = "Newport"> and @SET<ATTR[1] = "Costs">.

MOUSELAB also needs specification of the contents of the boxes of the matrix. The command @SET<BOX[1,2] = "Excellent"> indicates that the contents of the box for the first alternative and its second attribute should be Excellent. For each BOX, the first index refers to the alternative or row number while the second index refers to the attribute or column number. Multiple indices are always separated by commas.

Note, however, that in the output file, the matrix boxes are referenced by a single number. A 4 x 3 matrix, for example, is numbered this way in terms of program output:

```

1 2 3
4 5 6
7 8 9
10 11 12.
```

If the BOXES response mode was used, the choice boxes would be numbered starting with 13 and proceeding left to right. The multiple index method of referencing matrix boxes is used only in the input file.

As noted above, the alternative labels appear on the left side of the matrix and the attribute labels at the top in this, the normal screen orientation. If the TRANSPOSE option is switched on, the alternative labels would appear at the top of the matrix and the attribute labels on the left side. The input specifications, however, would remain the same, as would the box references in the output file. To transpose a matrix, the command would be @SET<TRANSPOSE = ON> or @SET<TRANSPOSE = YES>. Thus, the box references in the output file will look like this:

```

1 4 7 10
```

2 5 8 11
3 6 9 12.

| | Costs | Climate | Family |
|-------------|-------------|---------|--------|
| Newport | Expensive + | | |
| Chapel Hill | | | |
| Pittsburgh | | | |
| Chicago | | | |

Choose one: Newport Chapel Hill Pittsburgh Chicago
Pittsburgh was chosen. Enter this box and click once to continue.


Display Example 9: Sample Matrix Schema

The default response mode for a matrix schema is BOXES. Moreover, when the response mode is specified as DEFAULT, the labels of the alternatives (regardless of whether the matrix is transposed) are automatically used as the labels of the response boxes. In the DEFAULT mode, all response made details are handled automatically so that no other specifications are required. See Display Example 9 and Input Example 9.

```
@BEGIN <FILE>
@BEGIN <MATRIX>
@TITLE <"DEMO MATRIX SCREEN">
@SET <ALTERNATIVES = 4;ATTRIBUTES = 3>
@SET <ALT[1]="Newport";ALT[2]="Chapel Hill";
    ALT[3]="Pittsburgh";ALT[4] = "Chicago">
@SET <ATTR[1]="Costs";ATTR[2]="Climate";ATTR[3]="Family">
@SET <BOX[1,1]="Expensive";BOX[1,2]="Excellent";BOX[1,3]="YES";
    BOX[2,1]="Moderate";BOX[2,2]="Good";BOX[2,3]="Yes";
    BOX[3,1]="Moderate";BOX[3,2]="Fair";BOX[3,3]="No";
    BOX[4,1]="High";BOX[4,2]="Poor";BOX[4,3]="No">
@END <MATRIX>
@END <FILE>
```

Input Example 9: Input Format for Matrix Schema

If, however, the researcher does not want all of the alternatives to appear as response choices, he/she can use the suppressed alternatives feature. This feature prevents the first n alternatives from appearing as possible responses. This is accomplished with a command such as @SET<SUPPRESSED = 2>. It would suppress the first two alternatives specified; the first response box would therefore correspond to alternative number 3. This feature is useful if one wishes to include such things as attribute weights in the matrix.

| | Weights | Chapel Hill | Pittsburgh | Chicago |
|---------|---|-------------|------------|---------|
| Costs | | | | |
| Climate | | | | |
| Family | 7  | | | |

Choose one Chapel Hill Pittsburgh Chicago
 Chapel Hill was chosen. Enter this box and click once to continue.

Display Example 10: Transposed Matrix with Suppressed Alternative

```

@BEGIN <FILE>
@BEGIN <MATRIX>
@SET <TRANSPOSE = on>
@TITLE <"Demo matrix screen with weights">
@SET <ALTERNATIVES = 4; ATTRIBUTES = 3; SUPPRESSED = 1>
@SET <ALT[1] = "Weights"; ALT[2] = "Chapel Hill";
    ALT[3] = "Pittsburgh"; ALT[4] = "Chicago">
@SET <ATTR[1] = "Costs"; ATTR[2] = "Climate"; ATTR[3] = "Family">
@SET <BOX[1,1] = "10"; BOX[1,2] = "5"; BOX[1,3] = "7";
    BOX[2,1] = "Moderate"; BOX[2,2] = "Good"; BOX[2,3] = "Yes";
    BOX[3,1] = "Moderate"; BOX[3,2] = "Fair"; BOX[3,3] = "No";
    BOX[4,1] = "High"; BOX[4,2] = "Poor"; BOX[4,3] = "No">
@END <MATRIX>
@END <FILE>

```

Input Example 10: Input format for Transposed Matrix with Suppressed Alternative

Matrix schemas may also be used in conjunction with the move checking feature. The next example adds a row of decision weights to a 3 x 3 choice matrix. The weight row is now suppressed. See Display Example 10 and Input Example 10 for a transposed matrix with one attribute column (no longer row) suppressed.

4.2.3 Gamble Schemas

There are four variations on the gamble schema, each providing a different way of expressing preferences. The sub-schemas are (1) Probability Equivalence, (2) Certainty Equivalence, (3) Attractiveness Ratings, and (4) Bidding. Each sub-schema presents a two-outcome gamble in a decision tree format. The probability equivalence sub-schema also presents a certain outcome alternative. The names of the four sub-schemas listed above are descriptive of the different responses that are requested of the subject.

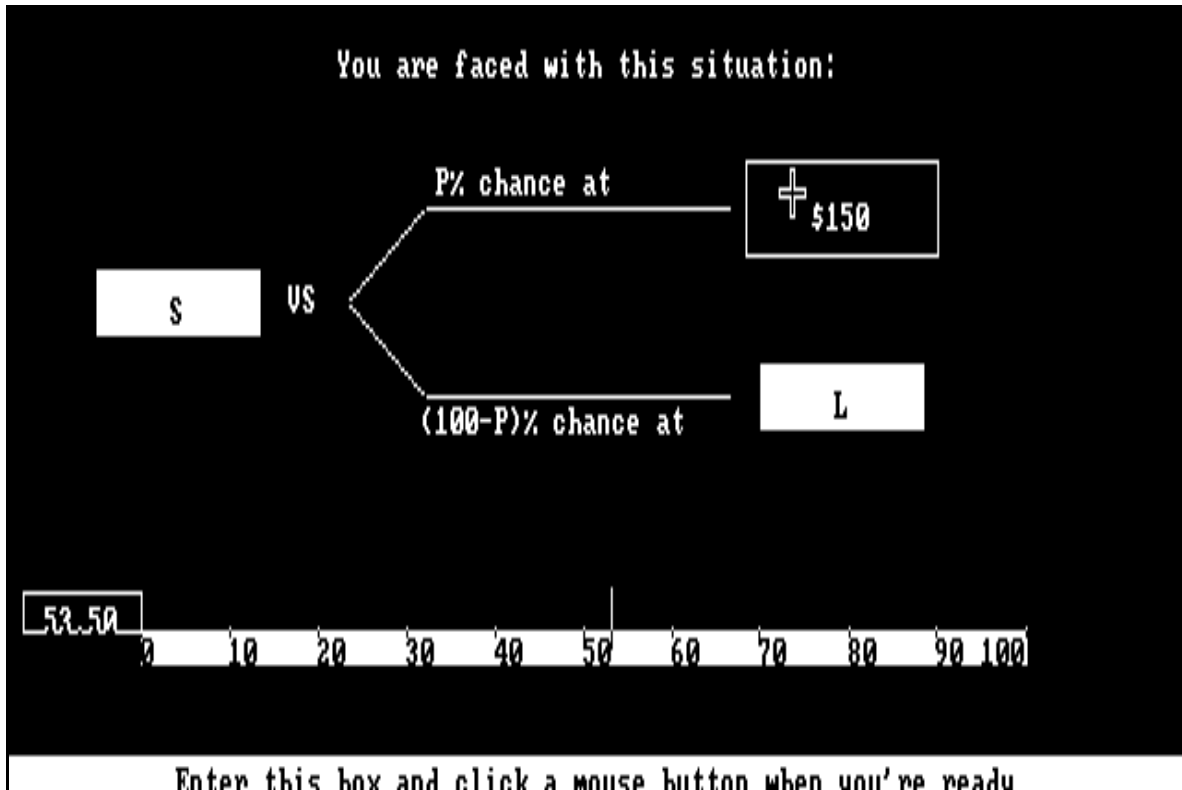
These are reflected in the default response modes. This feature is useful since it has been shown that different response modes can generate different preference orders (Lichtenstein & Slovic, 1971). Johnson, Payne, and Bettman (1988), and Schkade & Johnson (1989) have used MOUSELAB to examine preference reversals.

The relevant variables are slightly different across the four sub-schemas. Regardless of sub-schema, a gamble schema specification begins and ends in the usual fashion:

```

@BEGIN<GAMBLE> and @END<GAMBLE>. The sub-schema is selected with a command
such as @SET<GAMBLEMODE = PE>. The value PE stands for the Probability Equivalence
sub-schema. The other possible values for GAMBLEMODE are CE (Certainty Equivalence),
ATT (Attractiveness Rating), and AOB (Amount of Bid). GAMBLEMODE is a local variable
with a program default value of PE.

```



Display Example 11: Sample PE Gamble Schema

```

@BEGIN <FILE>
@BEGIN <GAMBLE>
@TITLE <"Demo PE Screen">
@SET <GAMBLEMODE = PE>
@SET <SUREAMOUNT = "$75"; HIGHPAY = "$150"; LOWPAY = "$0">
@END <GAMBLE>
@END <FILE>

```

Input Example 11: Input Format for PE Gamble Schema

For the PE sub-schema, MOUSELAB must be given a sure amount, the greater payoff of the gamble, and the lesser payoff of the gamble. The command `@SET<SUREAMOUNT = "$75"; HIGHPAY = "150"; LOWPAY = "$0">` assigns values to the appropriate variables. Note that the amounts are actually text strings. See Display Example 11 and Input Example 11 for a sample PE Gamble schema.

The other three sub-schemas are slightly more complex in terms of the information that must be supplied to the program. For the CE, Attractiveness, and Amount of Bid sub-schemas you will need to assign values to four variables specifying (1) the greater payoff, (2) the lesser payoff, (3) the probability of the greater payoff, and (4) the probability of the lesser payoff. The command `@SET <HIGHPAY = "$100"; HIGHPAYPROB = "2/10"; LOWPAY = "$0"; LOWPAYPROB = "8/10">` assigns values to the appropriate variables. Again, note that the

values are text strings which will actually appear within their respective boxes on the screen.

Moreover, when using the CE sub-schema, the TITLE command must be used to indicate the type of certainty equivalence problem being specified. The specified title must be either "Gain" or "Loss". Use the phrase which best describes the type of problem being presented to the subject.

The default response mode for gambles is SCALE. Depending on the sub-schema invoked, the user must specify some aspects of the scale. The PE sub-schema requires nothing in the way of scale specifications when using the default response mode. It automatically presents the subject with a probability scale ranging from 0 to 100. The other three sub-schemas require a full scale specification as described in Section 4.3.2. A special feature of the AOB sub-schema, however, is that a scale is automatically anchored by the actual greater payoff and lesser payoff values of the gamble. This ensures that the response is within the values of the gamble, i.e. the logical range defined by the payoffs. If the lesser payoff is a negative amount, then the lower endpoint of the response scale is set at \$0.

```
@BEGIN <GAMBLE>
@TITLE <"Demo Amount of Bid">
@SET <GAMBLEMODE = AOB; ANCHORMODE = OFF; ANCHORVALUE = 0>
@SET <HIGHPAY = "$33"; HIGHPAYPROB = ".7";
    LOWPAY = "11"; LOWPAYPROB = ".3">
@SET <LABELS = 2>
@SET <PUTLABEL[1] = 0.0; LABEL [1] = "$L";
    PUTLABEL[2] = 1.0; LABEL [2] = "$G">
@END <GAMBLE>
```

Input Example 13: Input Format for AOB Gamble Schema

Also note that if no RESPONSELINE is specified for the scale in the Attractiveness and AOB sub-schemas, MOUSELAB inserts "Attractiveness" and "Amount of Bid" respectively in the

```
@BEGIN <GAMBLE>
@TITLE <"Gain">
@SET <GAMBLEMODE = CE; ANCHORMODE = ON; ANCHORVALUE = 25>
@SET <HIGHPAY = "$100"; HIGHPAYPROB = "2/10";
    LOWPAY = "$0"; LOWPAYPROB = "8/10">
@SET <LABELS = 5>
@SET <PUTLABEL[1] = 0.0; LABEL[1] = "0";
    PUTLABEL[2] = 0.25; LABEL[2] = "25";
    PUTLABEL[3] = 0.5; LABEL[3] = "50";
    PUTLABEL[4] = 0.75; LABEL[4] = "75";
    PUTLABEL[5] = 1.0; LABEL[5] = "100">
@END <GAMBLE>
```

Input Example 12: Input format for CE Gamble Schema

RESPONSELINE location. Input Examples 12 and 13 contain input specifications for CE and AOB sub-schemas, respectively, with SCALE response modes.

4.2.4 MRC (Multiple Risky Choice) Schema

The MRC schema is the most complex of the schemas available with the MOUSELAB program. With the MRC schema, any number of sets of gambles can be specified, each set of gambles having any number of outcomes, and with each set of outcomes defined on any number of attributes (the first attribute value per outcome would be used to specify the probability of that outcome). The only limit on the number of sets, gambles, or outcomes used is the size of the screen. The gambles can be displayed in either a horizontal or vertical orientation. Finally, the MRC schema works with the move checking feature. This schema was designed with risky decision problems in mind. However, it can also be used to present nonrisky alternatives as well.

As before, an MRC display definition is bracketed by @BEGIN<MRC> and @END<MRC> commands. A display title may be sent to the output file in the usual fashion.

In order to process an MRC schema, MOUSELAB must be given the number of gambles (alternatives) in the set, the number of outcomes of a gamble, and the number of attributes per gamble. All values must be positive integers. To specify the number of gambles in the set, a command such as @SET<GAMBLES =3> is used. The command @SET<OUTCOMES = 2> indicates the number of outcomes every gamble in the set will have. The number of attributes per outcome is specified by @SET<ATTRIBUTES = 3>. The last three commands describe an MRC schema with three gambles, each with two outcomes, each outcome with three attributes. Once again, it is sensible to consolidate commands, yielding @SET<GAMBLES = 3; OUTCOMES = 2; ATTRIBUTES = 3>.

Again, there are no limitations in terms of program logic on the number of gambles, outcomes, and attributes that can be specified for an MRC display. The graphic capabilities of the machine, however, do limit the quality of display of very complex schemas. The appearance of complex choice problems should be judged on a case-by-case basis.

Each gamble should be labeled. The command @SET<GAMBLE[1] = "Gamble 1"> indicates that the first gamble will be labeled Gamble 1. Labels should be specified for the remaining gambles in similar fashion. The gamble names will appear at the top of the screen.

Each box should also be labeled. The command @SET<BOXLABEL[1,2,1] = "Box 2"> informs the program that the box corresponding to the second attribute of the first outcome in Gamble 1 should be labeled Box 2. The first index of a BOXLABEL variable refers to the outcome, the second index refers to the attribute, and the third to the gamble, i.e. BOXLABEL [outcome,attribute,gamble]. The box labels appear in reverse video on all "closed" boxes.

Finally, the contents of each box must be specified. The command @SET<BOX[2,1,3] = "\$250"> indicates that the contents of the box corresponding to the third attribute of the first outcome in the second gamble should be \$250. The first index in an MRC BOX specification refers to the gamble, the second index refers to the outcome, while the third index refers to the attribute, i.e. BOXLABEL [gamble,outcome,attribute].

As noted earlier, box references in the output file are in the form of single numbers. It is therefore important to understand this single index numbering scheme. The box numbers for a 3

X 2 X 3 display are given below:

| ONE | TWO | THREE |
|-------------|--------------|--------------|
| - 1 - 2 - 3 | 7 - 8 - 9 | 13 - 14 - 15 |
| X | X | X |
| - 4 - 5 - 6 | 10 - 11 - 12 | 16 - 17 - 18 |

For consistency, the boxes are numbered in the same order regardless of display orientation. To specify vertical orientation, the command `@SET<ORIENTATION = VERTICAL>` would be used. To produce the orientation above, one would specify `@SET<ORIENTATION = HORIZONTAL>`. `ORIENTATION` is a local variable with a program default value of `VERTICAL`. With a vertical orientation, the box numbers above would look like this:

| ONE | TWO | THREE |
|-----|------|-------|
| X | X | X |
| 4 1 | 10 7 | 16 13 |
| 5 2 | 11 8 | 17 14 |
| 6 3 | 12 9 | 18 15 |

The single index box numbering schema is also used by the move checking feature for specifying the legal move sequences.

The default response mode for MRC schemas is `BOXES`. The gamble names are automatically used as the labels for the choice boxes. As with the matrix schema, there is no need for the user to specify anything when using the MRC default response mode.

Again, with a vertical orientation, the inputs would be the same; only the actual display would change. The single index box numbering schema is also used by the move checking feature for specifying the legal move sequences.

The `MRCLINES` variable can be either on or off. When `MRCLINES` is on (the default), lines are drawn between boxes in the MRC screens. When `MRCLINES` is off, lines are not drawn between boxes. This allows for a reasonable amount of flexibility in designing custom made schemas. For example to present only two items of information to the subject one could set `MRCLINES` to `OFF`, and use a single gamble with two outcomes.

4.3 Response Modes

This section describes the two basic response modes available with MOUSELAB: choice and judgment. Choices are made using the response boxes mode, while judgments are made using the response scale mode. Choices and judgments are made using the keyboard input response mode.

4.3.1 Response Boxes

Response boxes appear at the bottom of a display and offer the subject a choice of responses. They are essentially answers to multiple choice questions. To select a response, the subject simply moves the cursor into the chosen box and presses a button on the mouse. Response box labels are visible regardless of whether the box is open or closed. The subject may change his/her mind simply by moving the cursor into a different response box and pressing a mouse button again. A choice is not final until the subject is satisfied with his/her own response and indicates that he/she has completed that display.

To specify the BOXES response mode, one would use the command `@SET<RESPONSEMODE = BOXES>`. The number and contents of the response boxes must also be specified. The command `@SET<RESPONSES = 2>` indicates that there should be two response boxes at the bottom of the screen. (RESPONSES can take on any positive integer value.) Each response box now needs a text string as a label. Assigning RESPONSES a value of 2 automatically creates two label variables, `RESPONSE[1]` and `RESPONSE[2]`. The number of label variables will always be equal to the number of response boxes. The command `@SET<RESPONSE[1] = "Response 1";RESPONSE[2] = "Response 2">` would label the first response box (the response box furthest to the left) as Response 1 and the second response box as Response 2. Note that the indices after RESPONSE must be integers and are always enclosed in the square brackets.

If necessary, a line of text may be placed just above the response boxes for clarification. The text may be up to 80 characters in length and is specified with a command of the form `@SET<RESPONSELINE = "Which response do you choose?">`. The contents of the RESPONSELINE variable are cleared after each display.

4.3.2 Response Scales

The scale response mode is selected with the command `@SET<RESPONSEMODE = SCALE>`. To understand the commands that describe the scale, it is necessary to understand how screen locations are specified. The screen coordinates are divided into areas roughly corresponding to the text positions on a normal IBM PC screen: 80 columns wide and 25 rows from top to bottom. For details on the coordinate system see Section 4.4.7

The command `@SET<SCALELEFT = 10>` places the left end of the scale 10 (out of 80) columns from the left side of the screen. The command `@SET<SCALEBOTTOM = 20>` places the bottom of the scale 20 lines from the top of the screen. Finally, `@SET<SCALESIZE = 60>`

indicates the scale is to be 60 columns long. These commands describe a scale that starts 10 positions from the left (x-coordinate), on row 20 from the top (y-coordinate), and extends 60 positions to the right. We have found that values of 10, 20, and 60 work well for SCALELEFT, SCALEBOTTOM, and SCALESIZE respectively and suggest you try these values as a starting point.

The scale must also be labeled. The command @SET<LABELS = 3> informs the program that the scale will have three labels. Each label must be defined as a text string. The position of each label must be explicitly defined from 0.0 to 1.0, 0.0 corresponding to the far left end of the scale and 1.0 to the far right end. For example, @SET<PUTLABEL[2] = 0.5; LABEL[2] = "5"> instructs the program to place the second label (5) at the midpoint of the scale.

Position along the scale is indicated by a pointer, an upside-down arrow. The mouse must first be positioned anywhere on the top half of the scale. A button on the mouse is then pushed, transforming the cursor into a pointer on the scale. The mouse is then used to move the pointer along the scale. (The scale mode also provides a digital readout of the pointer position on the scale.) When the desired scale value has been found, the final response is recorded by pushing the button once more. The cursor reappears, and a small "hairline" cursor marks where the subject exited the scale. Also, a subject may wish to reenter the scale to change his/her response. To reenter, a subject must touch the cursor to the hairline marker and press a button. The pointer will reappear, and the subject may readjust his response.

The MOUSELAB variable, DIGITALBOX, enables the experimenter to decide whether the subject will see the current digital value of the response scale. When DIGITALBOX is on (the default), a box containing the digital value of the response scale is displayed. When DIGITALBOX is off, no box is shown.

Normally, MOUSELAB automatically uses the end labels to calculate scale positions. This necessitates the use of numeric end labels. (Note that MOUSELAB can automatically handle dollar signs, though.) However, non-numeric end labels (e.g., "Min" and "Max") may be specified by using the SCALERANGE option. The command @SET<SCALERANGE = ON> activates this feature. The user specifies the scale range with a command of the form @SET<SCALEMIN = 0; SCALEMAX = 100>. This command informs the program that the scale position will range from a minimum value of 0 to a maximum value of 100. The values assigned to SCALEMIN and SCALEMAX must be positive integers. SCALERANGE is a local variable with a default value of OFF.

An initial position of the scale's hairline cursor may be specified by the command @SET<ANCHORVALUE = 50>. The initial position is given as an integer from 0 to 100, with 0 corresponding to the far left end of the scale and 100 to the far right end. In this case, the scale cursor would appear at the midpoint of the scale. The anchoring option is turned on by the command @SET<ANCHORMODE = ON>. Note that the values ON and YES, OFF and NO are always interchangeable. ANCHORMODE is a local variable with a default value of OFF or NO. This feature is useful for investigating aspects of the anchoring and adjustment heuristic (Slovic & Lichtenstein, 1972; Tversky & Kahneman, 1974).

A RESPONSELINE may also be specified in the same manner as for the BOXES response mode. In the SCALE mode, the line of text appears at the bottom of the screen. For an example of a response scale, please turn back to Display Example 5 and Input Example 5.

4.3.3 Keyboard Input Responses

| | Amt 1 QT | Amt 2 QT | Amt 3 QT | Amt 4 QT |
|---|----------------------|----------------------|----------------------|----------------------|
| Probs. | <input type="text"/> | <input type="text"/> | <input type="text"/> | <input type="text"/> |
| Gamble A | <input type="text"/> | <input type="text"/> | <input type="text"/> | <input type="text"/> |
| Gamble B | <input type="text"/> | <input type="text"/> | <input type="text"/> | <input type="text"/> |
| Gamble C | <input type="text"/> | <input type="text"/> | <input type="text"/> | <input type="text"/> |
| Gamble D | <input type="text"/> | <input type="text"/> | <input type="text"/> | <input type="text"/> |
| okay here is a response line | | | | |
| Type Response: | <input type="text"/> | Current Response = | 8.00 | |
| Enter this box and click a mouse button when you are ready. | | | | |

Display Example 14: Numeric input

The keyboard input response mode allows subjects to type in a real number as their response to a screen. The number must be a real number less than approximately 32000 in absolute value. This response mode is indicated by the following command:

Syntax:

@SET<RESPONSEMODE=KEYBOARD>

The subject indicates that he is ready to make a selection by clicking in the box containing the text "Keyboard Input."

4.4 MOUSELAB Features

4.4.1 Time Pressure

The time pressure feature can be used with any of the schemas. To turn on the feature, the command `@SET<CLOCK = ON>` is used. `CLOCK` is a local variable with a program default of `OFF`. As always, the values `YES` and `NO` may be substituted for `ON` and `OFF` respectively.

The user may specify where the clock should appear on the screen. The location of the clock image is specified in terms of an (x,y) coordinate pair indicating the location of the center of the clock face. As before, the x coordinate refers to number of columns to the right and ranges from 1.0 to 80.0; the y coordinate refers to number of rows from the top and ranges from 1.0 to 24.0. These coordinates may also be given in fractional units (i.e., expressed in real numbers).

The command `@SET<XCENTER = 7.0; YCENTER = 3.0>` places the center of the clock on the third line down and 7.0 character positions to the right. We suggest this location when using the time pressure feature with the gamble schema; it places the clock in the upper left corner of the screen. With large matrices one might try (6.0,2.0). These are only suggestions, however. The clock can be placed anywhere it looks good on the screen. (See Display Example 6). `XCENTER` and `YCENTER` are also local variables. Their default values are 7.0 and 3.0 respectively.

The researcher chooses what will happen when time expires. The command `@SET<ENDSTATUS = NOCHANGE>` instructs the program to print the message "Please make a choice or indicate a value" and record the fact that the countdown ended in the output file. The subject is allowed to continue normal search through information boxes and normal response. On the other hand, the command `@SET<ENDSTATUS = CLOSEDBOXES>`, records the same messages as above but does not allow the subject to search for additional information. All boxes

```
@BEGIN<MATRIX>
@TITLE<"keyboard response test; matrix 5 4">
@SET< ALTERNATIVES= 5; ATTRIBUTES= 4 >
@SET< RESPONSEMODE = KEYBOARD; SUPPRESSED = 1 >
@COMMENT<Labels for alterntives and attributes go here...>
@COMMENT<set box commands go here....>
@SET<RESPONSELINE = "okay here is a response line">
@END<MATRIX>
```

Input Example 12: Keyboard Input

are forced closed (even if the subject is inside one) when time expires. The subject is still allowed to make a response. The command `@SET<ENDSTATUS = CLOSEDCHECK>` does the same things as the `CLOSEDBOXES` option with the additional features of beeping when the subject enters a box that is not a response box and recording that fact in the output file. In other words, the behavior of the subject after the countdown ends is monitored. `ENDSTATUS` is a local variable with a program default value of `NOCHANGE`.

The `BEEPSTATUS` variable indicates whether a beep should sound when time expires. The command `@SET<BEEPSTATUS = ON>` instructs the program to beep at that time. `BEEPSTATUS` is also a local variable. Its default value is `OFF`.

Finally, one selects the amount of time to count down in seconds. The command `@SET<TIME = 60.0>` indicates a countdown of 60 seconds. The `TIME` variable accepts fractions of seconds, i.e. real numbers. As is the case for all other time pressure variables, `TIME` is local. Its program default is 5. Note that since all time pressure variables are local, one can avoid constant respecification of their values by assigning their values with a `SET` command in between schemas.

The clock closes all boxes at time 0. If the subject is in a box and the `NOCHANGE` option was selected, the box will re-open immediately. This was done to aid the analysis of the time pressure output using the `BISECT` program. The output file will always show the line like:

```
97 30.00 12
```

where 30.00 is the elapsed time and 12 is the current box containing the mouse cursor. No matter which time pressure option has been selected, the position of the cursor when the countdown ended is recorded.

The clock starts timing just after it has been drawn. This is when the boxes can be opened. There will be a slight discrepancy between the time you set in the schema input file and the actual time recorded in the output file. This reflects the fact that the program takes a few milliseconds to make sure that all the boxes are closed, print the ending message, beep (if selected), and record to the output file. However, the clock itself ends on time and the search boxes are closed immediately.

4.4.2 Move Monitoring

As noted earlier, this feature can be used to make sure a subject enters boxes only in a certain order, for example that of comparison of all alternatives for each attribute. If the subject tries to enter a box that is not in the specified sequence, the program will beep and the box will remain closed. The command `@SET<MOVEMONITOR = ON>` engages move checking. `MOVEMONITOR` is a local variable with a default value of `OFF`.

The move statements are embedded in the input file. They are contained within a pair of `@BEGIN<MOVE>` AND `@END<MOVE>` statements. The move-monitoring commands are the final statements inside the `@BEGIN<MATRIX>` AND `@END<MATRIX>` statements. Within the `@BEGIN<MOVE>` AND `@END<MOVE>` statements, the move file consists of one line for each box to be entered, in the order they are to be entered. Note that box references are in terms of a single box number. An example of move statements within an input file is given below.

```

@BEGIN<MOVE>
@COMMENT<An example of specifying move monitoring>
1 (Probability, Outcome 1)
2 (Attribute 1, Outcome 1)
3 (Attribute 2, Outcome 1)
6:4 (Box 6, 4 second limit)
3:5 (Box 3, 5 second limit)
@END<MOVE>

```

Input Example 15: An example of specifying legal moves.

The ":4" sets the time before a beep to 4 seconds. This feature allows you to set different times for different moves. The comments placed in an input file also appear in the output file when that move has been made. This can aid in the analysis of your results.

The sequence of legal moves is updated by the order of moves and not by the times or time in a legal box. For example, in the demo move file above, a subject can enter box 1 as often as he/she likes. This is perfectly legal. But, once the subject enters box 2, he/she cannot reenter box 1.

In most cases the move checking option treats the response (choice) boxes as always legal moves. That is, a subject can enter and leave a response box at any time. However, if you want to define only one legal response box then you can add the following line to the move file:

```
@RESPSEVERE=TRUE
```

This line can appear anywhere in the move file. However, you would have to follow that command at some point with the number of the legal response box. At that point, a subject entering any response box other than the legal one will find that the box does not open and there will be a beep. This event will also be recorded in the subject's output file. @RESPSEVERE is useful when training subjects to follow certain rules.

4.4.3 Open Boxes

The command @SET<BOXES = ON> displays the contents of all boxes within a given screen. This feature is useful for instructional purposes. BOXES is a local variable, and its default value is OFF.

4.4.4 Data Trace Option

The Data Trace option allows the researcher to suspend tracing of a subject's actions within a display. The output file will simply show the normal screen header along with beginning and ending times for that display. The command @SET<DATATRACE = OFF> suspends data tracing. DATATRACE is a local variable with a program default value of ON. In the case of text schemas with no response mode, however, MOUSELAB automatically suspends data tracing since there can by definition be no subject activity to trace. Again, the data trace option is most useful during instructional parts of an experiment, where collection of data is

usually unnecessary.

4.4.5 Scale Tracking

The scale tracking feature gives the researcher a record of changes in the direction of scale adjustment. Each change in direction is recorded in the output file with a box number of 96 along with the time at which the change occurred and the value of the scale. Obviously, scale tracking can only be used in conjunction with the SCALE response mode. The command @SET<SCALETRACK = YES> activates the scale tracking feature. SCALETRACK is a local variable with a default value of NO.

4.4.6 Click-to-Open

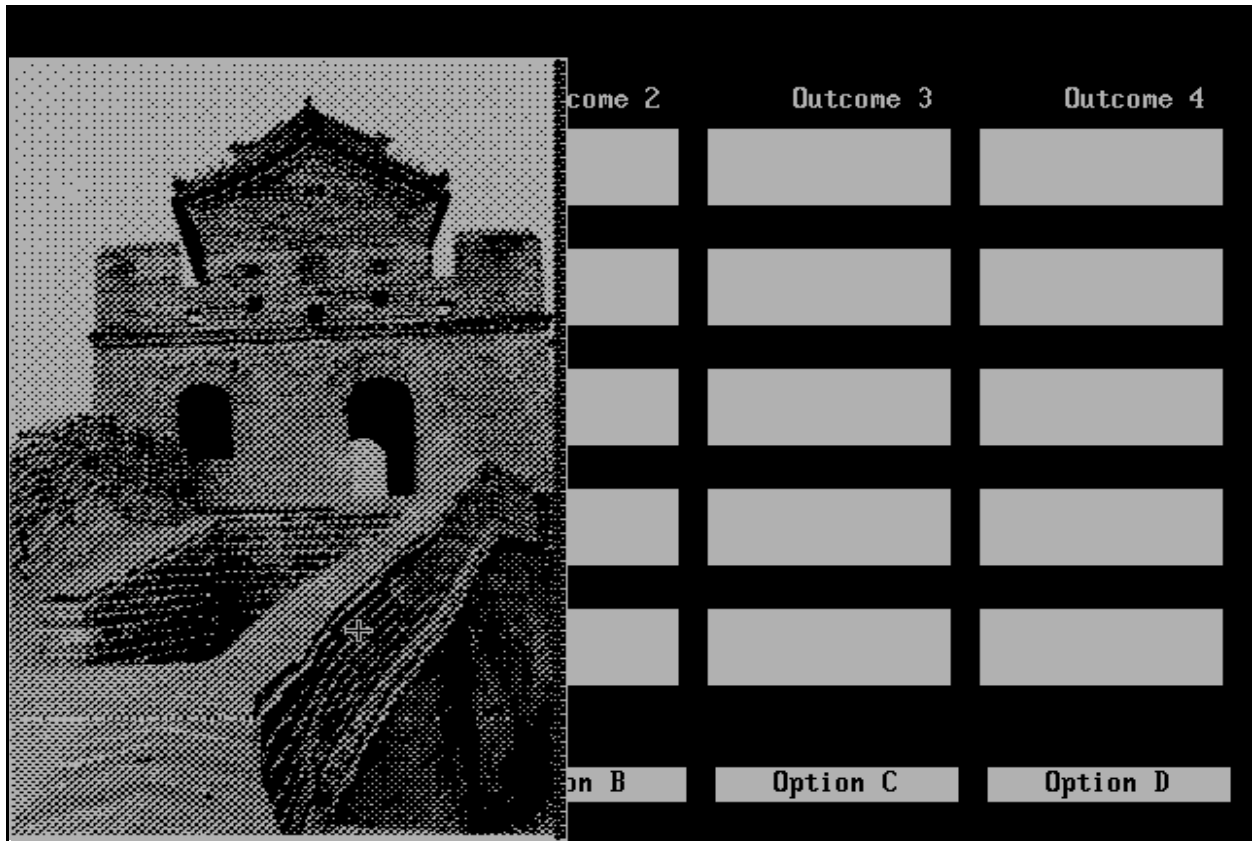
The click-to-open feature is a MOUSELAB variable called CLICKON that can be either on or off. For it to have any effect, the MOUSELAB variable BOXES must be off. When CLICKON is off, MOUSELAB operates as usual. When CLICKON is on, the subject must enter a box AND click the left mouse button to open a box. To close a box, the subject must click the right button (no other boxes may be opened when one box is open). This option is especially useful when large graphics boxes are being used.

4.4.7 Locating the Scale and Clock

Most of the screen elements are located in positions determined by the schema. However, explicit specification of screen location is needed for the positions of the response scale and time-pressure clock. The position of these objects on a MOUSELAB screen is described by an ordered pair, indicating its x and y coordinates. The screen is divided from top to bottom into 24 rows, and from left to right into 80 columns. The coordinates of the top left corner of the screen are (1,1). The bottom right corner is (24,80). All location specifications are expressed in terms of these units.

For example, to tell MOUSELAB where to put a response scale, the user would have to specify appropriate x,y coordinates.

4.5 Graphics Boxes



Display Example 16 MOUSELAB graphics: The Great Wall

A new feature of MOUSELAB 5.0 is the ability to display simple black and white images on the screen. These are held in boxes, just like text information, and are opened by entering (or clicking on, see below) any box containing the location of a picture. MOUSELAB currently displays the PCX format, in two colors, black and white. These pictures can be created by a Scanner, a program like PC-Paint (and, in theory, any other programs which generate a PCX format graphic file). The graphics file should be created on the same adaptor type as it will be displayed on. A graphics box can be displayed by using the following general command:

Syntax:

`@SET<PBOX[1,1] = "filename" (XSIZE,YSIZE)>`

where *filename* is the name of a ".pcx" file containing the graphics to be displayed and *XSIZE* and *YSIZE* are the size that the graphics are to be scaled to for display. Size is specified here in the standard MOUSELAB coordinate system. If *XSIZE* or *YSIZE* is zero, then the picture will be displayed at the size it was drawn. In almost all cases, this is the preferred option.

Example:

```
@COMMENT<display the graphics in hello.pcx>  
@SET<PBOX[1,1] = "hello.pcx" (10,10)>
```

Note: it often takes a very long time for MOUSELAB to scale a picture to size on a standard IBM-PC, Our best advice is to scale the picture to the size you want before entering it to MOUSELAB. PC Paint is one program that we find useful for doing this.

A couple of other caveats. Currently only two-color (i. e. black and white) images are supported. This limits reasonable displays to line-art, simple figures like probability wheels, etc. Also remember that even VGA displays only show 640x480 dots. This is less than a 3" by 2" image on most laser printers. Finally, we find the clickon feature quite useful with large pictures. You might want to look at this feature in the next section. Even though it loses something in the printing, Display Example 16 shows that you can get your last vacation pictures into MOUSELAB. Of course, we anticipate that more use will be made of consumer products, statistical displays, etc.

5 Changing MOUSELAB's Behavior: The MOUSELAB Programming Language

5.1 Overview.

New with Version 5.0 is the capability of writing programs with a full-fledged MOUSELAB programming language. This allows the experimenter to change the behavior of the program in reaction to subjects' responses. Our goal is to provide the flexibility that comes with most blocked structured languages, such as PASCAL or C, to control the presentation of MOUSELAB schemas. To use this language, you will need to understand how MOUSELAB handles variables, expressions, and the flow of control among statements.

5.2 Variables

Two types of variables help make MOUSELAB a flexible tool for designing customized experiments.

5.2.1 Pre-declared Variables

Scalar Variables

In order to allow the experimenter to access data from the previous screen, three variables have been declared. These variables are treated the same as experimenter variables, except they already exist and do not need to be declared. The variables and their contents are:

1. %LRESPONSE: a real variable that is the subject's last response
2. %TTIME: a real variable that is the total time spent in the last screen, and
3. %TBOXES: the total number of boxes entered *and* left.

Array Type

Two array type variables that are used to access information from the previous screen are also declared. The variables are preceded by an "&" rather than a "%" to indicate that they are arrays. To access a specific element in the array, a subscript must be used inside square brackets. The variables:

1. &PELAPSED: an array of real values that indicate at what time a box is entered or left, and
2. &PBOXES: an array of integer values that indicate the number of each box entered and left.

Example:

```
@COMMENT<get the last box left and store the value>
@ASSIGN<%mylast = $pboxes[%tboxes]>
```

5.2.2 Experimenter-variables

The experimenter can customize a study by creating unique variables in the input file. For example, the experimenter could create non-MOUSELAB variables like NAME and AGE. When the subject is prompted to type in his name and age, the MOUSELAB program records the responses in the output file.

The variables can also be assigned values, used in SET statements and aid in the use of flow-of-control. An example of this use would be the creation of a new variable, CLOCK, which determines the amount of time a subject can spend in a given matrix screen as a function of time spent in earlier screens. The experimenter can "set" the clock time to any desired length of time by assigning the variable a time value based on an earlier decision. If the experimenter wants the subject to redo the problem until she no longer runs out of time, the flow-of-control features can be used to return the program to the same problem, resetting the clock for each attempt.

5.3 Using Variables

To use an experimenter-declared variable, the experimenter first declares the variable. Then, if desired, he can assign a value to the variable. Assignment can be accomplished in two

ways:

1. by simply specifying a value for the variable, or
2. by using an "expression" that evaluates a response and returns a variable value.

Expressions are discussed in Section 5.5.

Experimenter variables can be used in regular MOUSELAB SET statements. They can also be printed out to the output file. All of these options are detailed below.

5.3.1 Declaring Variables

Before an experimenter variable may be used in the input file, it must be declared to be one of four types:

1. **string**, a collection of characters,
2. **real**, a real number,
3. **int**, an integer value, or
4. **boolean**, true/false.

An experimenter variable is declared in a declare statement and its name must always be preceded by a "%".

Declaration statements are of the general form:

Syntax:

@DECLARE<*type* : %*var_name*>

where *type* is one of the four types described above and *var_name* is the name that a user assigns to a variable.

Example:

@DECLARE<string : %name>

@DECLARE<int : %count; real : %score; boolean: %okay>

5.3.2 Assign

An experimenter variable may be assigned a value in an assign statement. The value is an expression (as defined in the Section 5.5 to follow). Assignment statements take the general form:

Syntax:

@ASSIGN<%*var_name* = *expression*>

where *var_name* is the name of a previously declared variable and *expression* will be described in the section on expressions.

Example:

```
@ASSIGN<%name = "Mark">  
@ASSIGN<%count = %count + 1>
```

5.3.3 Using variables in a SET statement.

MOUSELAB variables may be set to experimenter variables of the appropriate types (i.e., a MOUSELAB variable that is a string may not be set to an integer experimenter variable). For example, if the experimenter wants to set a box to be the value in a experimenter variable called "%instring" (where %instring has been declared a string), he could use the following command:

```
@SET<BOX[1,1] = %instring>
```

This means that when the box numbered [1,1] is opened by the subject, the value inside the box will be the last value of the experimenter variable "%instring."

5.3.4 Printing Variables

The experimenter may print the value inside an experimenter variable by using the print command. The print command writes out the variable's values in the output file. The print command takes the general form:

Syntax:

```
@PRINT<%var_name>
```

where *var_name* is any previously declared experimenter variable.

Example:

```
@PRINT<%instring>
```

An example of the use of several experimenter-declared variables is given in the following input file, which will print out all the boxes entered in the last trial, along with the time of entry and exit. This, of course, duplicates the actual MOUSELAB output file, but is an illustration of how the @PRINT statment can work.

Example:

```
@DECLARE<int : %i>
@ASSIGN<%i = 1>
@WHILE<%i #lt# (%tboxes+1)>
    @PRINT<"BOX">
    @PRINT<&pboxes[%i]>
    @PRINT<"TIME">
    @PRINT<&pelapsed[%i]>
@ASSIGN<%i = %i + 1>
@END_WHILE
```

5.4 Other MouseLab Variables and the RESET Command.

Fifteen MOUSELAB variables are local variables. These variables are automatically given default values at the outset of the program. A list of these variables and their defaults is given in **Table II**.

Table II: Local Variables and Their Defaults.

| Variable | Default Value |
|--------------|---------------|
| ANCHORMODE | OFF |
| BEEPSTATUS | OFF |
| BOXES | OFF |
| CLOCK | OFF |
| ENDSTATUS | NOCHANGE |
| GAMBLEMODE | PE |
| MOVEMONITOR | OFF |
| ORIENTATION | VERTICAL |
| RESPONSEMODE | DEFAULT |
| SCALERANGE | OFF |
| SCALETRACK | OFF |
| TIME | 5.0 |
| TRANSPOSE | NO |
| XCENTER | 7.0 |
| YCENTER | 3.0 |

When MOUSELAB creates a new display, it makes copies of all local variables. It is these copies along with their preassigned values which are referenced within individual display specifications. As a result of this procedure, all changes made to the value of a local variable within a display specification are local and affect only that display. The copies disappear at the end of that display.

By using the SET command between display definitions, the user can alter the default values of any or all local variables. Once this is done, the copies of the variables used in subsequent display definitions will automatically contain the new default values.

The command @RESET is used between display definitions to return all local variables to the original program defaults listed above. Note that RESET cannot be used on a single variable; it operates on all of them.

5.5 Expressions

An expression is a collection of binary operators, conversion functions, and experimenter variables. Binary operators take two values and evaluate them to a single value. Binary operators take the general form:

Syntax:

$X \text{ op } Y,$

where X and Y are expressions and *op* is a binary operator. The binary operators are:

- $+$: adds 1 real (integer) expression to another real (integer) expression to yield a real (integer) value
- $-$: subtracts 1 real (integer) expression from another real (integer) expression to yield a real (integer) value
- $*$: multiplies 1 real (integer) expression by another real (integer) expression to yield a real (integer) value
- $/$: divides 1 real (integer) expression by another real (integer) expression to yield a real (integer) value
- $|$: concatenates 1 string expression to another string expression
- $\#eq\#$: compares 1 real (integer) expression to another real (integer) expression to yield a boolean value (True if equal; False otherwise)
- $\#lt\#$: compares 1 real (integer) expression to another real (integer) expression to yield a boolean value (True if the first expression is less than the second; False otherwise)
- $\#gt\#$: compares 1 real (integer) expression to another real (integer) expression to yield a boolean value (True if the first expression is greater than the second; False otherwise)

Functions that convert one type to another type can also be used in expressions. These functions take the general form:

Syntax:

FUNC_NAME(expression)

where *FUNC_NAME* is the name of a function. The conversion functions are:

| | |
|------------|--|
| INTTOSTR: | converts an integer expression to a string value |
| REALTOSTR: | converts a real expression to a string value |
| STRTOINT: | converts a string expression to an integer value |
| FLOAT: | converts an integer expression to a real value |
| ROUND: | rounds a real expression to an integer value |
| TRUNC: | truncates a real expression to an integer value. |

Parentheses can (and should) be used to specify the order of evaluation in complicated expressions. The only guarantee about the order of evaluation of expressions is that expressions inside parentheses will be evaluated first.

Example:

```
@COMMENT<set the integer variable %count to 5>
@ASSIGN<%count = 5>
@COMMENT<increment the value in %count by 1>
@ASSIGN<%count = %count + 1>
@COMMENT<calculate a percentage>
@ASSIGN<%percent = %count / 100>
@COMMENT<now change this percentage (a real)>
@COMMENT<into a string value for use in boxes>
@ASSIGN<%boxstring = realtostr(%percent)>
@COMMENT<a nonsensical, but complicated expression>
@ASSIGN<%percent = round((%count + 1) / %percent)>
```

5.6 Flow-of-Control

Flow-of-control is a set of MOUSELAB commands that control the order in which MOUSELAB commands are executed. These statements can be used to execute the same set of MOUSELAB statements several times or to conditionally execute a set of MOUSELAB statements. There are six different types of flow-of-control commands.

5.6.1 The WHILE Statement

The WHILE statement executes a block of MOUSELAB commands repeatedly WHILE a given boolean expression is true. The While statement takes the general form:

Syntax:

```
@WHILE<boolean expression>
  any number of MOUSELAB commands
@END_WHILE
```

In more formal terms this is called a pretest loop, since it is only executed if the condition at the beginning of the loop is true.

Example:

```
@COMMENT<The following example will print the string variable 5 times>
@ASSIGN<%count = 0>
@ASSIGN<%mystring = "hi">
@WHILE>%count#lt#5>
@ASSIGN<%count = %count + 1>
@ASSIGN<%mystring = %mystring | "hello">
@PRINT<%mystring>
@END_WHILE
```

5.6.2 The REPEAT Statement

The REPEAT statement will execute a block of MOUSELAB commands until a given boolean expression is true. The REPEAT statement takes the general form:

Syntax:

```
@REPEAT
  any number of MOUSELAB commands
@UNTIL<boolean expression>
```

Example:

```
@COMMENT<same thing as in previous example>
@ASSIGN<%count = 0>
@ASSIGN<%mystring = "hi">
@REPEAT
@ASSIGN<%count = %count + 1>
@ASSIGN<%mystring = %mystring | "hello">
@UNTIL<%count#eq#5>
```

5.6.3 The IF Statement

The IF statement causes a block of MOUSELAB commands to be executed once if a given boolean expression is true. The IF statement takes the general form:

Syntax:

```
@IF<boolean expression>
Any number of MOUSELAB commands
@END_IF
```

Example:

```
@COMMENT<if %account is 3, then do the commands>
@IF<%count#eq#3>
@ASSIGN<%mystring = "hi there">
@END_IF
```

5.6.4 The ELSE IF Statement

The ELSEIF commands executes one block of MOUSELAB commands if a given boolean expression is true and another if the boolean expression is false. The ELSEIF takes the general form:

Syntax:

```
@ELSEIF<boolean expression>
any number of MOUSELAB commands
@THENELSE
any number of MOUSELAB commands
@END_ELSEIF
```

Example:

```
@COMMENT<Set %count to 4 if it is 10>
@COMMENT<otherwise set %count to 3>
@ELSEIF<%count#eq#10>
@ASSIGN<%count = 4>
@THENELSE
@ASSIGN<%count = 3>
@END_ELSEIF
```

An example of the ELSEIF feature is the following. Suppose you want a subject to repeat a problem if he does not give the right answer. You can use the "repeat" option to make the screen appear again. If, however, you want different text screens to appear following the first

presentation of the problem, you can use the ELSEIF option to specify when MOUSELAB should continue to a new problem.

Example:

```

@COMMENT<subject has seen problem once>
@ELSEIF<%lresponse#eq#10>
@ASSIGN<%okay = true>
@COMMENT<text screen for correct response>
@BEGIN<TEXT>
@BEGIN<SCREENTEXT>
.
.
.
@END<SCREENTEXT>
@END<TEXT>
@THENELSE
@ASSIGN<%okay = false>
@COMMENT<problem screen is repeated if answer was wrong>
@END_ELSEIF

```

5.6.5 The CASE Statement

The CASE statement causes one of several blocks of MOUSELAB statements to be executed depending on the value of a particular variable. The CASE statement takes the general form:

Syntax:

```

@CASE<variable>
@OPTION<value1>
@any number of MOUSELAB statements
@END_OPTION
@OPTION<value2>
Any number of MOUSELAB statements
@END_OPTION
@OPTION<value3>
Any number of MOUSELAB statements
@END_OPTION
.
.
@END_CASE

```

If the variable in the CASE statement has the value in a particular OPTION clause, then the MOUSELAB statements inside that OPTION clause are executed. The variable in the CASE statement must be of the same type as the variable in the OPTION clauses.

Example:

```

@COMMENT<Execute one of several blocks of code>
@COMMENT<depending on the value of %count>
@CASE<%count>
@OPTION<1>
@PRINT<"Count is One">
@END_OPTION
@OPTION<5>
@PRINT<"Count is Five">
@END_OPTION
@OPTION<10>
@PRINT<"Count is Ten">
@END_OPTION
@END_CASE

```

5.6.6 The GOTO Statement

The GOTO statement is an excellent debugging tool for the experimenter. It causes the execution of statements to skip to the label specified in the GOTO statement. The general form is:

Syntax:

```

@GOTO<%label_name>
@LABEL<%label_name>

```

The *label_name* must not be the same as any declared variable and is preceded by a "%".

Example:

```

@GOTO<%checkscreen>
Any number of MOUSELAB statements
@LABEL<%checkscreen>

```

The LABEL statement can be before or after the GOTO. GOTO's should not jump into or out of other flow-of-control statements. They can be used by the experimenter to skip over several screens in an input file that he knows are correct to check a screen later in the program.

5.7 Programming Language Examples

5.7.1 Accuracy feedback

The experimenter may wish to provide a subject feedback regarding his choice on a previous screen. The following example shows how the predeclared variable, %LRESPONSE, and the ELSEIF statement can be utilized to provide feedback.

Example:

```

@BEGIN<MATRIX>
definition of a matrix screen
@END<MATRIX>
@COMMENT<If the subject did not select the>
@COMMENT<1st alt., then he is incorrect>
@ELSEIF<%lresponse#eq#1>
@BEGIN<TEXT>
@BEGIN<SCREENTEXT>
You made the correct choice
@END<SCREENTEXT>
@END<TEXT>
@THENELSE
@BEGIN<TEXT>
@BEGIN<SCREENTEXT>
Sorry, the correct choice was number 1.
@END<SCREENTEXT>
@END<TEXT>
@END_ELSEIF

```

5.7.2 Force a correct choice

It may be desirable in some situations, particularly practice screens, to cause a subject to encounter the same screen over and over until the subject makes the correct choice. A combination of the REPEAT statement and the %LRESPONSE variable can achieve this.

Example:

```

@COMMENT<Force the subject to select the>
@COMMENT<first alternative>
@REPEAT
@BEGIN<MATRIX>
definition of a matrix screen
@END<MATRIX>
@UNTIL<%lresponse#eq#1>

```

5.7.3 To enforce an effort level

In order to force a subject to put at least some semblance of effort into making a choice, it may be desirable to force him to enter at least a minimum number of boxes or spend at least a minimum amount of time in a screen. If he does not expend the required amount of effort, he can be forced to repeat the screen. The following example forces the subject to enter at least five boxes.

Example:

```
@REPEAT  
@BEGIN<MATRIX>  
definition of a matrix screen  
@END<MATRIX>  
@UNTIL<(%tboxes/2)#gt#4>
```

5.8 Command Abbreviations

Abbreviations of input language keywords are permitted. If the abbreviation is unique and could not match another keyword, there is no problem. If the abbreviation could apply to more than one keyword, however, caution must be exercised. When MOUSELAB attempts to identify a keyword, it does so by checking an internal list of legal keywords alphabetically. If an abbreviation could refer to more than one keyword, the one selected as a match will be the first encountered in the alphabetic search. It is important to keep this in mind when using abbreviations, particularly since MOUSELAB does not give warnings in multiple match situations; it simply chooses the first legal keyword that matches the abbreviation. Obviously, we find that using complete keywords greatly facilitates understanding MOUSELAB programs.

Table III: MOUSELAB Reserved Words

| | | | |
|---|---|---|--|
| <u>ALTERNATIVE</u> <u>ALTERNATIVES</u> <u>ANCHORMODE</u> <u>ANCHORVALU</u> E <u>AOB</u> ASSIGN <u>ATT</u> <u>ATTRIBUTE</u> <u>ATTRIBUTES</u> <u>BEEPSTATUS</u> <u>BEGIN</u> <u>BOX</u> <u>BOXES</u> <u>BOXLABEL</u> <u>CE</u> CASE CLICKON <u>CLOCK</u> <u>CLOSEDBOXES</u> | <u>CLOSEDCHEC</u> K COMMENT <u>DATATRACE</u> DECLARE DEFAULT ELSEIF <u>END</u> ENDIF <u>ENDSTATUS</u> <u>FILE</u> <u>GAMBLE</u> <u>GAMBLEMOD</u> E <u>GAMBLES</u> <u>GOTO</u> <u>HIGHPAY</u> <u>HIGHPAYPRO</u> B <u>HORIZONTAL</u> IF <u>LABEL</u> | <u>LABELS</u> <u>LOWPAY</u> <u>LOWPAYPROB</u> <u>MATRIX</u> <u>MOVEMONITOR</u> <u>MRC</u> NO <u>NOCHANGE</u> <u>OFF</u> <u>ON</u> <u>ORIENTATION</u> <u>OUTCOMES</u> <u>PE</u> <u>PUTLABEL</u> <u>REPEAT</u> <u>RESET</u> <u>RESPONSE</u> <u>RESPONSELINE</u> <u>RESPONSEMOD</u> E | <u>RESPONSES</u> <u>SCALE</u> <u>SCALEBOTTOM</u> <u>SCALELEFT</u> <u>SCALEMAX</u> <u>SCALEMIN</u> <u>SCALERANGE</u> <u>SCALESIZE</u> <u>SCALETRACK</u> <u>SCREENTEXT</u> <u>SET</u> <u>SUPPRESSED</u> <u>SUREAMOUNT</u> <u>TEXT</u> <u>TIME</u> <u>TITLE</u> <u>TRANSPOSE</u> <u>VERTICAL</u> <u>WHILE</u> <u>XCENTER</u> <u>YCENTER</u> <u>YES</u> |
|---|---|---|--|

6 The MOUSELAB Output File

6.1 Output file format.

A sample of the MOUSELAB output format is contained in **Table IV** (the headings do not occur in a real output file). The first line in the example is a header that identifies the screen (schema) that generated the responses that follow. The next line is numbered 100. It marks the beginning of the output from a particular screen. Notice that the time and response are 0.000 and 0.00, respectively. The next lines are a record of the mouse cursor movements. The Boxnumber is the number of the box that the cursor was in. The boxnumbers always come in pairs. For example, the pair of 3s above means that the subject entered and left box number 3. If a subject had entered and left and then entered and left the same box you would see something like 3....; 3....; 3....; 3.... on four separate lines. The next number on a line is the time at which the cursor entered or left the box. Time is counted from the beginning of the display. The third number indicates a possible response. That is, if the associated box is a choice or response box rather than an information box, then a number other than 0 would be listed. For a choice box, the number listed corresponds to the alternative being considered, as listed from left to right on the bottom of the screen. For a scale response, the number listed corresponds to the scale value. The value 9999.00 signals that the subject has not yet made a response on the scale. The last item on an output line could be a comment (not shown in the example). This is a text string taken from a move file, if that feature had been selected. The last line of the output from a particular screen is also numbered 100. It marks the end of that particular set of responses and gives a record of the

Table IV: A Sample MOUSELAB output file.

| BOXNUMBER | TIME | RESPONSE | COMMENT |
|-----------|-------|----------|----------------|
| 99 | | scr # 1 | Demo Screen #1 |
| 100 | 0.000 | | 0.00 |
| 1 | 0.879 | | 9999.00 |
| 1 | 0.988 | | 9999.00 |
| 3 | 0.988 | | 9999.00 |
| 3 | 1.784 | | 9999.00 |
| 5 | 2.202 | | 9999.00 |
| 5 | 4.598 | | 9999.00 |
| 8 | 4.598 | | 2.00 |
| 8 | 4.954 | | 2.00 |
| 100 | 5.230 | | 2.00 |

total time viewing the screen and the final response (choice) indicated by the subject.

In addition to the standard output discussed above, five other lines will be written to the output file, depending on the MOUSELAB program features you have selected. Each of these are noted by a special box number ranging from 94 to 100.

The first marks the end of the countdown under the time pressure condition.

Example:

97 10.051 0.0 (The countdown ended here)

The first number on the line just identifies it. The second number is the time. The third number is the current response (choice) if any. Finally, there is the comment that the countdown has ended.

The second is a variation of this line and is used by the text schema to indicate elapsed reading time. The output from a typical text screen is as follows:

Example:

| 99 | scr #1 | Demo | | |
|------------|---------------|--------------|--------------|----------------------------|
| 100 | | 0.000 | 0.000 | |
| 100 | | 3.725 | 0.000 | (text reading time) |

The third special feature output line is for the movechecking condition. It uses essentially the same format.

Example:

98 2.470 0.0 (buzzed in box 10)

The comment at the end lets you know that the subject was in a wrong box for more than the specified period of time and which box he or she was in.

The fourth special output line results from each @print statement. Every time that statement is executed, a line is added to the output file which specifies the type of variable being output, and its value. These records are set off by the box number 94.

Example:

| | | |
|-----------|---------------------|----------|
| 94 | INTEGERVAR = | 1 |
| 94 | INTEGERVAR = | 3 |

The final special output line indicates a change in the direction of response scale adjustment when the SCALETRACK feature is engaged.

Example:

96 22.746 36.21 (change in scale adjust direction)

The line specifies, from left to right, the special identifying number (96), the time, the scale value when direction of adjustment changed, and a descriptive comment. Also, a line starting with a 0

occurs when the subject enters or leaves the scale.

Example:

```

0 25.785 38.24
96 26.394 35.91
0 27.132 45.36

```

These lines specify that the subject entered the scale at a value of 38.24, adjusted down to 35.91, and then left the scale at 45.36.

An output file can be analyzed directly. However, it is often useful to use the BISECT program that is part of the MOUSELAB package to make the task of analysis easier. BISECT determines the total time the box was open, the current response, and comments for each search (fixation) of a box at a particular point in the sequence of acquisitions. In other words, it compresses the standard output file from the MOUSELAB program.

6.2 BISECT

BISECT is a utility program that accepts as its input any output file produced by MOUSELAB v5.0. As its output, BISECT produces a "fix" file which, using as input the MOUSELAB output file on page 40, would have the general format illustrated in Table 3 (again, the headings are not part of the BISECT output file):

Table V: A BISECT Output File.

| Subject | Trial | Box-number | Time | Plustime | Sequence | Response |
|---------|-------|------------|-------|----------|----------|----------|
| 1 | 1 | 1 | 0.109 | 0.109 | 1 | . |
| 1 | 1 | 3 | .796 | 1.214 | 2 | . |
| 1 | 1 | 5 | 2.396 | 2.396 | 3 | . |
| 1 | 1 | 8 | 0.356 | 0.632 | 4 | . |
| 1 | 1 | 100 | 5.230 | 0.000 | 5 | 2.00 |

SUBJECT is the subject number which, as will be explained later, is determined by the user. TRIAL is the screen number. BOXNUMBER and BOXTIME are the same as described in section 8.0. If the same box is reentered 1 or more times, the BOXNUMBER will only be listed once and BOXTIME will be the sum of the individual boxtimes. PLUSTIME is calculated by

subtracting the entry time of the current box from the entry time of the next box. Plustime is the amount of time until the next box acquisition. If the next box acquisition does not exist, plustime is 0.0. SEQUENCE is the order of acquisition, while RESPONSE is the same as described in section 8.0.

The last line of a screen will always have boxnumber '100', indicating when the subject left the screen. In addition, the contents of any @TITLE command will always be placed at the end of the first line of BISECT output for that display. This can be useful since data concerning experimental stimuli can be included within the BISECT output file and can be used for data analysis programs.

Other programs written in SAS or some other suitable language can be used to analyze the results of BISECT. Here is one method, often used in our experiments. Typically, the BISECT output file is uploaded from the microcomputer to a mainframe computer. A SAS program inputs this file and produces a temporary data set generating appropriate process measures. This data set can then be summed across a trial using the PROC MEANS procedure using the 'sum' option. The results of this are merged with a data set describing the stimuli to create any other necessary variables such as average acquisition time, alternative variance, attribute variance, etc. Final data analysis is then produced by using SAS procedures such as Proc GLM.

Like MOUSELAB, the BISECT program runs under IBM PC/DOS. Place the disk containing 'BISECT.EXE' in drive A. The user then types BISECT and presses the return key. BISECT will then prompt the user for three types of information. First, it will ask for the name of the input file. This can be any MOUSELAB v5.0 output file. Second, it will ask for the name of the output file. This can be any valid PC/DOS file name. Finally, it will ask for the subject number.

7 Randomizing and Counterbalancing in MOUSELAB

The MOUSELAB program itself does not do the kinds of randomization and counterbalancing usually used in psychological experimentation. This can be done two ways. The first is to use the MOUSELAB programming language to do this. This requires real programming skills and care. The central problem to using the MOUSELAB language to randomize and counterbalance stimuli is making sure that enough information is written to the output file to analyze order effects.

The other alternative is done externally to the program, either by hand or by writing a special purpose program. The input would be a "master" experimental file which would produce many rearranged output files, one per subject, which would serve as input files to MOUSELAB.

With the previous release, we made available a generalized solution, consisting of a preprocessor which can produce randomized and counter-balanced files for input to MOUSELAB.

As described below, a post processor is also available to unscramble the files.

The preprocessor adds three command pairs (BLOCK, RANDOMIZE, and COUNTERBALANCE) to the MOUSELAB input language. The experimenter simply needs to add these commands within a normal MOUSELAB input file, creating a "master" input file. The preprocessor then takes as input this master input file and a unique subject number, and produces a MOUSELAB input file, removing the new "ordering" commands, but leaving the appropriate sections randomized and counterbalanced. In addition, the preprocessor also produces a file showing the nested nature of the input file -- helpful in debugging -- and an auxiliary file showing the new order of the schemas. The preprocessor is case insensitive, as the other MOUSELAB routines are.

7.1 The BLOCK Command

A line beginning with "@BEGIN<BLOCK>" will signify the beginning of a block or section. A line beginning with "@END<BLOCK>" will signify the end of that block. Anything in between these two lines will be treated as an indivisible unit and will remain as a block in the produced MOUSELAB input file. Note, also, that any global command, schema, or preprocessor commands may be nested within a block. For example, assume an experiment contains three different groups of schemas: MRC, Matrix, and PE Gamble. The experimenter would like to randomize the order of the groups while keeping the different schema types segregated. The master input file, abbreviated, would have the following structure (indentation is used here only to denote nesting and is not necessary)L

Example:

```

@BEGIN<RANDOMIZE>
  @BEGIN<BLOCK>
    MRC schema #1
    .
    .
    MRC schema #x
  @END<BLOCK>
  @BEGIN<BLOCK>
    Matrix schema #1
    .
    .

```

```

        Matrix schema #y
    @END<BLOCK>
    @BEGIN<BLOCK>
        PE gamble schema #1
        .
        .
        PE gamble schema #z
    @END<BLOCK>
@END<RANDOMIZE>

```

7.2 The RANDOMIZE Command

A line beginning with "@BEGIN<RANDOMIZE>" signifies the beginning of a randomized pair, and a line beginning with "@END<RANDOMIZE>" signifies the end of the pair. Any schemas, blocks, and preprocessor command pairs contained between these two lines will be randomized. Note that schemas and preprocessor command pairs are treated as indivisible units by the randomize command. For example, if an experimenter wished to randomize three blocks of schemas, each block containing different types of schemas, the master input file would contain the following structure:

Example:

```

@BEGIN<RANDOMIZE>
    @BEGIN<BLOCK>
        SCHEMA #1-#x
    @END<BLOCK>
    @BEGIN<BLOCK>
        SCHEMAS #1-#y
    @END<BLOCK>
    @BEGIN<BLOCK>
        SCHEMAS #1-#Z
    @END<BLOCK>
@END<RANDOMIZE>

```

In randomization, the preprocessor uses the subject number as the random number generator seed.

7.3 The COUNTERBALANCE Command

"@BEGIN<COUNTERBALANCE>" and "@END<COUNTERBALANCE>" are the last pair of preprocessor commands. Any schema or preprocessor command pair between these two lines will again be treated as an indivisible unit. The preprocessor will automatically count the total number of indivisible units contained between the "counterbalance" commands. If the master input file contains N units within the pair, the preprocessor will write to the new input file the Mth through Nth units, and then the 1st through M-1th units. M is equivalent to the remainder of N (number of counterbalanced units) divided by the subject number. Again, any schemas or other preprocessor command pairs can be nested within a Counterbalance pair.

7.4 Using RANDOM and UNRANDOM

Usage of the preprocessor is straightforward. Again, before using the preprocessor, any master input file must be tailored using the three commands explained above. Then, after starting the preprocessor, RANDOM.EXE, the user will be asked to type in the name of the master input file. Then, the preprocessor will prompt the user for a subject number. The preprocessor will then proceed to create three new files ending in ".inp", ".aux", and ".dbg". The master input file's base name, with the subject number behind it, will be used for the base name of the three new files. Thus, if the master input file was named "test.doc", and the entered subject number was 23, "test23.inp", "test23.aux", and "test23.dbg" would be created. The ".inp" file contains the new MOUSELAB input file. The ".aux" file contains the order of the scramble screens -- this file is used by the postprocessor to unscramble the MOUSELAB data output and must not be deleted. If a mistake has been made using the preprocessor commands, the program will terminate. To aid in locating the mistake, the ".dbg" file contains a copy of the master input file, but with indentation and indexing to illustrate any nested structures. After a file has been successfully created by the preprocessor, the ".dbg" file is no longer needed by the pre/post processors and may be deleted.

The postprocessor, simply rearranges the output from a MOUSELAB run. The postprocessor prompts the user for the name of the output file, and then searches for an ".aux" file with the same base name. It will proceed to unscramble the output file, writing to a new output file. The new output file is given the same base name, but will have a ".rat" extension (an acronym for raw data). Output from all screens and schemas will be in their original master file order (i.e. the order before the preprocessor was used). In addition, the postprocessor alters the first line of each schema, denoted by a 99. The 99 line might look like this:

```
99      scr# 23          Original #4.
```

This was the 23rd screen the subject viewed, but was the fourth screen in the master input file.

Usually, to perform data analysis, rearranging the data files into a standard order (that of the master file) is necessary.

The postprocessor is named UNRANDOM.EXE. After starting it, the post processor will simply prompt the user for the output file name. Execution will terminate if an ".aux" file of the same name is not found.

8 Summary

As decision researchers have become more interested in explaining the processes of judgment and choice, there has been a concomitant search for new methods of studying decision behavior. One technique that has proven valuable is the monitoring of information acquisition behavior. This technique involves setting up the decision task so that the decision maker must view or select information in a way that can be easily monitored. The procedures used to study information acquisitions and decisions have ranged from simple information boards constructed of cardboard and 3 x 5 cards to sophisticated eye-movement apparatus. Unfortunately, the simple techniques may impose an unspecified effort and time cost on acquiring information. Eye-movement recording, which reduces the effort and time costs, has the disadvantage of involving expensive equipment that is difficult to use.

This report has documented a new procedure for monitoring information acquisition behavior. The procedure is based on a micro computer-controlled pointing device called a mouse. The procedure also employs a number of flexible graphics and data recording routines. There are several thousand different display and experimental condition combinations that can be used. Included are relatively new capabilities, such as checking the order of information acquisitions to insure that the subject is following a specified decision strategy. Using the MOUSELAB Software offers a decision researcher a viable way to obtain the high temporal density of observations regarding a subject's pre-decisional behavior that appears necessary to develop and test process models of decision behavior.

Ultimately, the usefulness of any technique is determined by the insight that it generates. We have included a MOUSELAB bibliography which lists some of the research that has employed this technique.

REFERENCES

- Arch, D.C., Bettman, J.R., and Kakkar, P. (1978). Subjects' information processing in information display board studies, in Advances in Consumer Research, Volume V, Hunt, H.K. (Ed.), 550-560.
- Bettman, J.R., Johnson, E.J., & Payne, J.W. (in Press). A componential analysis of cognitive effort in choice. Organizational Behavior and Human Decision Processes.
- Card, S.K., Moran, T.P., & Newell, A. (1983). The Psychology of Human-Computer Interaction. Hillsdale N.J.: Lawrence Erlbaum.
- Einhorn, H.J., & Hogarth, R.M. (1981). Behavioral decision theory: Processes of judgment and choice. Annual Review of Psychology, 32, 53-88.
- English, W.K., Englebart, D.C., & Berman, M.L. (1967). Display-selection techniques for text manipulation. IEEE Transactions on Human Factors in Electronics, 8, 5-15.
- Ericsson, K.A., & Simon, H.A. (1980). Verbal reports as data. Psychological Review, 87, 215-251.
- Ericsson, K.A., & Simon, H.A. (1984). Verbal Reports as Data. Cambridge, Mass: MIT Press.
- Johnson, E. J., Payne, J.W., Bettman, J. R. (1988). Information displays and preference reversals. Organizational Behavior and Human Decision Processes, 42, 1-21.
- Just, M.A., & Carpenter, P.A. (1984). Using eye fixations to study reading comprehension. In D.E. Kieras and M.A. Just (Eds.) Reading Comprehension Research, Hillsdale, N.J.: Lawrence Erlbaum, 151-182.
- Kahneman, D., Slovic, P., & Tversky, A. (Eds.) (1982). Judgment Under Uncertainty: Heuristics and Biases. New York: Cambridge University Press.
- Lichtenstein, S. & Slovic, P. (1971). Reversals of preference between bids and choices in gambling decisions. Journal of Experimental Psychology, 89, 46-55.
- McConkie, G.W., & Rayner, K. (1976). Asymmetry of the perceptual span in reading. Bulletin of the Psychonomic Society, 8, 365-368.
- Newell, A., & Simon, H.A. (1972). Human Problem Solving. Englewood Cliffs, N.J.: Prentice-Hall.

- Payne, J.W. (1976). Task complexity and contingent processing in decision making: An information search and protocol analysis. Organizational Behavior and Human Performance, 16, 366-387.
- Payne, J.W., Bettman, J.R., & Johnson, E.J. (1988). Adaptive strategy selection in decision making. Journal of Experimental Psychology: Learning, Memory, and Cognition, 14, 534-552.
- Payne, J.W., Braunstein, M.L., & Carroll, J.S. (1978). Exploring predecisional behavior: An alternative approach to decision research. Organizational Behavior and Human Performance, 22, 17-44.
- Pitz, G.F., & Sachs, N.J. (1984). Judgment and decision: Theory and application. Annual Review of Psychology. 35, 139-163.
- Rayner, K. (1975). The perceptual span and peripheral cues in reading. Cognitive Psychology, 7, 65-81.
- Russo, J.E. (1978). Eye fixations can save the world: A critical evaluation and a comparison between eye fixations and other information processing methodologies. Advances in Consumer Research, Volume V, Hunt, H.K. (Ed.), 561-570.
- Russo, J.E., & Doshier, B.A. (1983). Strategies for multiattribute binary choice. Journal of Experimental Psychology: Learning, Memory, and Cognition, 9, 676-696.
- Russo, J.E., Johnson, E.J., & Stephens, D.L. (1985). When are verbal protocols valid? Unpublished Working Paper. Cornell University.
- Slovic, P. & Lichtenstein, S. (1971). Comparison of Bayesian and regression approaches to the study of information processing in judgment. Organizational Behavior and Human Performance, 6, 649-744.
- Tversky, A. (1972). Elimination-by-aspects: A theory of choice. Psychological Review, 79, 281-299.
- Tversky, A. & Kahneman, D. (1974). Judgment under uncertainty: Heuristics and biases. Science, 185, 1124-1131.

9 Appendices

Appendix A: Licensing Agreement

We request a \$25.00 fee for MOUSELAB disks and manual. This should be payable to Trustees of the University of Pennsylvania. This covers only our reproduction costs and labor. In return, permission to use the MOUSELAB software is given for not-for-profit research and educational uses only. Please contact one of the authors of this document if you intend to use this software in any other application.

While we distribute this software on an at-cost basis for research and educational use, we would appreciate if you do not copy the software or documentation for others. Please ask them to contact us for additional copies. This allows us to provide updates to all users and allows us to monitor usage. If you did not receive this copy from us, please contact us so that we may register you as a user. Thank you.

Appendix B: MOUSELAB files and their functions

The following is a list of MOUSELAB files included on the distribution diskettes and their functions:

MOUSELAB.BAT: Batch file which starts MOUSELAB.

METASHEL.EXE: Command which installs the graphics driver. Called in MOUSELAB.BAT

MOUSE50.EXE: The MOUSELAB program itself.

RANDOM.EXE: Contains the preprocessor used to scramble and counterbalance MOUSELAB input files.

UNRANDOM.EXE: Contains the postprocessor used to unscramble subjects' MOUSELAB raw data output. (Can only be used if original MOUSELAB input files were preprocessed with RANDOM.EXE.)

BISECT.EXE: Program which takes as input MOUSELAB raw output and organizes the data into a form which can then be used for statistical analysis. If RANDOM.EXE was used on a MOUSELAB input file, BISECT should be used only AFTER URANDOM.EXE has been used to unscramble the file.

*.MLI: MOUSELAB input file. DO NOT ALTER BY ANY MEANS.

*.FNT: All files with the .FNT extension are graphics files used by the graphics driver. These

files should not be altered.

DEMO5.INP and DEMO5.MOV: MOUSELAB demonstration files used to illustrate the different capabilities of MOUSELAB.

Appendix C: Fitt's Law and the time to point the mouse

The time required to point to a box on a screen using a mouse can be described by Fitts Law, a basic equation in the human factors literature.

Fitts Law:

Time to position = $K_o + I_m (D/S + .5)$ seconds.

where D is the distance to the target box, S is the size of the target, $I_m = .100$ sec/bit, and K_o is a constant.

The size of the target S and the distance between targets D varies across the different displays possible with the MOUSELAB program. Consequently, time to position will differ across the displays. However, it is useful to consider a few displays that will involve large differences in the time to position the mouse cursor in a box.

Gamble Schema. The gamble schema involves the smallest number of boxes spread across the greatest width of the screen. The boxes are 1.2 cm high and 3.0 cm wide. They have a maximum distance of 7 cm. For such a display, Fitts Law yields a value of $K_o + 245$ milliseconds.

Matrix Schema. A matrix display involving just two alternatives and two attributes is one that is likely to yield the fastest times to position the cursor. The boxes are 8 cm high and 8 cm wide. They have a maximum distance of 8 cm. For such a display, Fitts Law yields a value of $K_o + 82$ milliseconds.

Of course, the time to position values above depend on a value for K_o . Card, Moran, and Newell (1983) present values for K_o that range from 600 to 1100 milliseconds. However, those values are likely to be overestimates of K_o 's for our task. Unlike some of the pointing devices and tasks studied by Card, Moran, and Newell, in our task the subject's hand is always present on the mouse.

Appendix D Other Matters

Errors

To err is human, but computer programs are even worse. Therefore, a few words about errors.

Errors you may encounter when using MOUSELAB will be of two basic types: program generated errors and machine generated errors. Program generated errors are errors which are detected and handled by MOUSELAB itself. These usually have to do with program start-up parameters and input file formatting. A program generated error message is distinguished by the presence of the word "MOUSELAB" prior to the actual message. These messages refer to problems specific to the MOUSELAB program; they usually result from user mistakes. For example, one such message is "MOUSELAB: unrecognized graphics card." This indicates that the user has specified a graphics card in the start-up parameter list which MOUSELAB is not able to handle. This may result either from a typo or from attempting to use MOUSELAB with a graphics card it cannot use. To correct the problem, the user must fix the typo in the former case or use a different graphics card in the latter.

Machine generated errors are errors which are detected and handled by the operating system. These will usually be the result of a previously undetected bug in the MOUSELAB program. They may, however, also result from creative user mistakes which we failed to anticipate. Machine generated error messages will not have the "MOUSELAB" label prior to the message text. In this case, try to determine if the error message refers to something under your (the user's) control. If it doesn't or if it's just so much gobbledegook, please follow the directions in this appendix under Reporting Bugs.

Reporting Bugs

If you discover a bug in the program, either fatal or nonfatal to program execution, please write to

Professor Eric Johnson/MOUSELAB
Marketing Department, SH/DH 1457
The Wharton School
University of Pennsylvania
Philadelphia, PA 19104-6327

Be sure to send COMPLETE information, including a full description of the problem, the full text of the error message (if any), the system configuration, and any other pertinent information regarding the circumstances (e.g., if MOUSELAB consistently bombs when it reaches a certain screen, send a listing of the input file specifications for that screen and the one preceding it).

We also can be reached by computer mail. Any mail should be sent to the following address:

JOHNSON00@WHARTON.UPENN.EDU

Time constraints prevent us from responding to telephone inquiries.

Other Inquiries

Please direct all other inquiries to Professor Johnson at the above address.

We Beg Your Indulgence

Please remember that we are not professional software developers. We will do our best to help you, but we make no promises of fast and/or efficient service.

Coming Attractions

As time marches on, we will be working to enhance MOUSELAB. We are currently working on a version that transmits messages across networks. Be forewarned, however, that this may be at the expense of upward compatibility. In other words, files which work with version level 5.0 of MOUSELAB may not work with subsequent versions (version 3.0 files, for example, do not work with version 5.0). Constraints on program size and on the amount of time we can devote to it will probably force us to sacrifice upward compatibility for the sake of user friendliness and increased capabilities. We feel this is a worthwhile tradeoff and apologize in advance for any hassles this may cause in the future. Therefore, always keep a copy of this version of the program if you want to be able to use its input files after the next version is released.