# MOUSING AROUND

Use the mouse or a joystick for fast, error-free menu selections! These simple programs demonstrate the techniques.

Keyboard input can bring out the klutz in you. Fingers turn stupid, tripping over the keys in a frenzy of typos. If your digits act like 10 sore thumbs at the Apple keyboard, why not try a mouse or joystick for menu selection? As Macintosh owners well know, the mouse guarantees fast, error-free selection. No typing, or typos.

The technique for mouse and joystick input shown in **Listings 1 and 2** can be easily implemented in your own programs, at a very small cost in added lines of code.

**Listings 1 and 2** show how to initialize the mouse, print a menu on the screen, move the cursor on the screen in response to the movement of the mouse or joystick, get the cursor position from the screen, and use that cursor position to determine the user's choice from the menu screen. The programs are not particularly useful in themselves, but show the techniques involved in using the input devices as screen pointers.

MOUSEPOINTER (**Listing 1**) uses the mouse as the input device. It works in the 40-column mode of ProDOS only. JOYPOINTER (**Listing 2**) uses the joystick as the input device, and it works in the 40-column mode under either DOS 3.3 or ProDOS.

## USING THE PROGRAMS

To use the programs, attach the mouse or joystick to the game port (or DeskTop Bus of the IIGS), and RUN the appropriate program. If you're using a mouse card, it should be in slot 4. A menu and a blinking asterisk cursor will be displayed. As you move the mouse or joystick, the cursor will match that movement on the screen.

With the cursor positioned over the menu item of your choice, press the mouse or joystick button. A message confirming which option was chosen will be displayed until you press the button again. The menu will then reappear for another choice. Notice that as the cursor moves over the letters in the menu items, the letters do not disappear, but are redisplayed after the cursor moves on. Strangely enough, this was one of the most difficult effects to program.

## ENTERING THE PROGRAMS

To enter MOUSEPOINTER, key in **Listing 1** and save it with the command:

SAVE MOUSEPOINTER

To enter JOYPOINTER, key in **Listing 2** and save it with the command:

SAVE JOYPOINTER

For help in entering *Nibble* listing, see the instructions in the Typing Tips Section of this issue.

**TABLE 1: Variables**

| Variable | Function |
|---|---|
| DU | Dummy variable used to beep the speaker |
| I | Loop index that sets the duration of the beep |
| MB | Holds the button status (1-4 for mouse; > 127 or < 128 for joystick) |
| MX | Column coordinate of the cursor position |
| MY | Line coordinate of the cursor position |
| OD | Memory location corresponding to the screen position where a character was replaced by the cursor |
| OL | ASCII code for the character that was replaced by the cursor in location OD |
| PK | Memory location corresponding to a screen position, as computed from the mouse or joystick input |

## HOW THE PROGRAMS WORK

### The Mouse Routine

**Listing 1** is made up of many subroutines. The code was written to be easily understood rather than to optimize either memory space or execution time. The main routine is in **lines 10-50**; all subroutines are called from here. The variables and ProDOS commands used in the programs are shown in **Tables 1 and 2**, respectively.

**Line 10** calls the mouse initialization subroutine at **line 500**. **Line 500** designates the mouse as the output device with PR#4, and sends a 1 to the mouse firmware. This line is necessary to prepare the mouse for use (wake it up, so to speak). The mouse is always plugged into slot 4 on the IIc, but may not be in slot 4 on the IIe or II Plus. Slot 4 is the recommended slot, but if your mouse is in another slot, simply change the number after PR# in **lines 80 and 500**.

**Line 510** resets output to the screen after the message has been sent to the mouse.

**Line 20** calls the screen menu print routine at **line 250. Lines 250-370** print the screen menu, using VTAB and HTAB. Any line or column position can be printed to, with some exceptions.

If you print to any part of line 24, or the 40th column of line 23, be sure to end that print statement with a semicolon (;). This will prevent an automatic linefeed that will scroll the top line off of the screen and ruin your menu (see **line 370**). Using the 40th column of line 24 will cause a linefeed even with the semicolon at the end of the statement. Two other column positions may also be unusable; more about that when we get to the INPUT statement.

**Line 30** calls the subroutine at **line 70** that reads the movement of the mouse and translates it into cursor movement on the screen.

**TABLE 2: ProDOS Commands**

| Command | Function |
|---|---|
| PRINT CHR$(4); "PR#4" | Sends future output to slot 4. |
| PRINT CHR$(4); "PR#0" | Sends future output to the screen. |
| PRINT CHR$(4); "IN#4" | Reads future input from slot 4. |
| PRINT CHR$(4); "IN#0" | Reads future input from keyboard. |

**Line 70** initializes a character variable to blank (ASCII 160 = blank), and initializes an integer variable to 2039, the address of a memory location that will be used later. **Line 80** sets input to be read from the mouse port, turning off the keyboard. This causes the INPUT statement in **line 90** to read the mouse's position instead of waiting for keyboard input.

The INPUT statement is the best way to get a position reading from the mouse, but it has a problem: it insists on printing a prompt on the screen. If you don't supply one inside the quotation marks, it will use its own, the question mark (?). Either way, something shows up on the screen where you don't want it.

This can't be avoided, but the effect can be minimized. Use a VTAB to move to a line where you will not use column 1 or 40. Then use HTAB 40 to move to the end of the line. Next, supply the null character as the prompt by putting the two quotation marks next to each other (""), and add a semicolon (;) to prevent the linefeed. You won't see the prompt on your menu screen, but both columns 1 and 40 will be set to blank.

The mouse input comes in three parts, and all three must be requested in the same INPUT statement. The inputs are, in order, the X-coordinate (column position), the Y-coordinate (line position), and a button status code. The coordinates each have a range of 0-1023; coordinates 0,0 are the upper-left corner of the screen, and the numbers increase as the mouse is moved to the right or down. (**Line 80** initializes the mouse to the 0,0 coordinates.)

The mouse button status code range is 1-4 as shown in **Table 3**. When you push the mouse button, the status changes from released/released to released/pushed (code 2) for one pass and thereafter, it goes to status pushed/pushed (code 1) for as long as you hold the button down. When you release the button, it becomes pushed/released (code 3) for one pass, and then settles into released/released (code 4). If at any time you press a key on the keyboard, the mouse button status code will change from positive to negative, allowing you to check for a keyboard interrupt. MOUSEPOINTER makes no use of this feature, except that **line 150** checks the status code for both a positive and a negative two. You can reset the mouse button status code to a positive value with POKE −16368,0.

Now that you have the mouse information (X-coordinate, Y-coordinate and button status code) for this pass through the loop, one more important thing must be done. **Line 100** resets input to read the keyboard, rather than the mouse. Failing to do this can create quite a colorful problem if your program later crashes.

In a crash, the operating system first displays an error message and then attempts to get new input. Since input is still set to the mouse port, however, what it gets is a three-variable mouse reading, and it can't make heads or tails of that. Confused, it prints SYNTAX ERROR and rings the bell. Then it tries again to get good input, but again it gets a mouse reading. The gist of all this is that your speaker will begin beeping its little heart out while your screen fills up with SYNTAX ERROR messages. Not a pretty sight. Worse, the original error message and line number left by the crash quickly scroll off the screen into never-never land. **Line 100** will fix this for future lines, but if **line 90** caused the crash, you'll see the SYNTAX ERROR display. (This disheartening display can be stopped by pressing Control-Reset.)

**Lines 110 and 120** handle the sensitivity of the mouse. As you move the mouse, the X- and Y-coordinates change at the rate of about 47 digits to the inch, up to a maximum of 1023, both horizontally and vertically. The range used by HTAB and VTAB to move the cursor to the limits of the screen is 0-39 for the X-coordinate and 0-23 for the Y-coordinate. The mouse moves only about one inch as its X-coordinate goes from 0-39, and about half an inch as the Y-coordinate goes from 0-23. This makes the mouse much too sensitive to use with any accuracy, and it wastes the other 984 increments of mouse output.

**Lines 110 and 120** decrease this sensitivity by dividing the initial mouse reading by a factor of 4 for the X-coordinate and 6 for the Y-coordinate. If you want the cursor to be more responsive to the mouse, just use lower factors. If you want it to be less responsive, increase them. The X-coordinate factor can reach a maximum of 25 and the Y-coordinate a maximum of 40, but at those levels, you'll need a big desk. At the settings of 4 and 6, you'll need an area 3 inches by 3½ inches in which to move the mouse, if you want to move the cursor to all corners of the screen.

**Lines 130 and 140** just prevent the Y coordinate from exceeding 24 and the X coordinate from exceeding 40 to avoid an unpleasant crash.

The next section of code has a very important purpose. As you move the cursor, it obliterates any character on the screen at that position. So before the cursor is moved there, the program must find out what character is about to disappear and save it, so that it can be replaced when the cursor moves away.

The characters you see on your 40-column by 24-line monitor are actually stored in a portion of the memory, and the image on the screen is constantly restored from that memory. When you place a character on the screen using the keyboard, the computer is actually putting the character in the memory location that corresponds to that position on the screen. For instance, the memory location with the address 1024 corresponds to the upper-left corner position. If you were to place a character into that memory location, it would immediately appear at the corresponding position on the screen. For instance, type POKE 1024,65. An A will appear in the upper-left corner of the screen (as 65 is the ASCII code for A).

Once you know the corresponding memory location for a position on the screen, you can use that information to move the cursor around, and to determine what character is at any particular position. Now the bad news: the memory locations that correspond to the character positions on the screen are not contiguous in memory. In order to find out what memory location corresponds to the screen position 5,10 (column 5, line 10), you need a formula. We'll deal with that later.

There is a lot going on in **line 150**. First, the mouse button is checked to see if it has been pressed. If it has, the speaker beeps to show that the button push was noticed. The program then jumps to the RETURN at **line 220**.

If the button has not been pressed, there is a new mouse reading. Before the cursor is moved to its new position on the screen, the character that the cursor overwrote must be restored. The old character's ASCII code is stored in the variable OL (old letter), and the address of the memory location corresponding to its screen position is stored in OD (old address). It's a simple matter to POKE OD,OL and restore that character to its original screen position.

If this is the first time through the loop, there is no old address or old letter because the cursor has not yet been placed on the screen. **Line 70** initializes OD and OL so that, the first time through the loop, a safe location is POKEd that does no harm. OD and OL place a blank at line 24, column 40. This spot cannot be used in a PRINT statement because of the automatic linefeed. There are lots of other safe POKEs, however, (e.g., POKE 49200,0). In later passes through the loop, **lines 180 and 190** set OD and OL.

**Line 170** holds the formula that translates the column and line position on the screen to the corresponding memory location. The credit for this formula, goes to the *Beagle Bros Tip Book #5*.

PK (PEEK address) is now set to the memory location corresponding to the position where the cursor will be placed next. **Line 180** sets the OL by PEEKing into this location; **line 190** sets OD equal to PK.

**Line 200** POKEs the cursor character into memory so that it appears on the screen. I've used 170, the ASCII code for the asterisk (*). You could just as easily use 171 (the plus sign) or any other character.

**Line 210** is the end of the loop. The routine now loops back to get a new mouse reading, and it continues to loop, updating the mouse position until the button is pressed, and **line 220** causes a RETURN from the subroutine.

When the mouse movement routine returns, MX holds the column in which the cursor is located and MY holds the line. These variables are used to determine what choice the user has made. **Line 140** directs flow to the subroutine that implements this choice. The choice is determined by the values of MX and MY, which show where the cursor was on the screen when the mouse button was pushed. **Line 400** is a default message that is overwritten when a legal choice is made.

**Lines 410 and 420** show how to handle menu choices that take up an entire line. The code simply checks to see if the cursor was on the appropriate line (if MY equals the line number) when the button was pushed. If there is more than one choice on a single line of the menu, then MX (the column position) will also need to be checked. **Lines 430 and 440** check for one of two choices on a line. If the menu contains more choices, this section is more complicated.

**Line 450** tests for the quit option. If that option is chosen, this line prints GOODBYE and ends the program.

The appropriate message now remains on the screen until the mouse button is pressed. **Line 470** accomplishes this wait by simply calling the mouse input subroutine again. The subroutine returns MX and MY, but the information is not used.

**Line 480** returns program flow to the main routine, where **line 50** starts the sequence all over again.

## The Joystick Routine

**Listing 2** is the same program adapted for use with a joystick. The main difference between the two programs is the input subroutine starting at **line 60**.

In **Listing 2** you don't have to initialize a joystick, or set input to the joystick device. **Lines 70 and 80** get MX and MY. With the joystick, you don't need to use the INPUT statement, and thus don't have its associated problems.

**TABLE 3: Mouse Button Status Codes**

| Previous Status | Present Status | Code |
|---|---|---|
| Pushed | Pushed | 1 |
| Released | Pushed | 2 |
| Pushed | Released | 3 |
| Released | Released | 4 |

**Line 90** retrieves the joystick button status, which is stored in memory location 49249. The joystick status is not as precise as the mouse status: it has only two possible states. When the button is pressed, the value in location 49249 is greater than 127; when the button is not being pressed, the value is less than 128. **Line 160** sounds a tone signaling that the button has been pressed. If the button is held down too long, it could be read as two presses. The tone tells you that the press was received and to release the button.

**Lines 100 and 110** adjust the sensitivity of the joystick inputs, just as **lines 110 and 120** in **Listing 1** adjusted the mouse readings. The joystick is less sensitive than the mouse, with an output range of only 0-255, about one-fourth that of the mouse. Dividing this reading by 6 horizontally and 10 vertically will give you the least sensitive setting possible. And even at that, getting the cursor to the right choice on the screen is a little like playing an arcade game.

**Lines 120-150** set the boundaries that keep the cursor on the screen.

**LISTING 1: MOUSEPOINTER**

```
1  REM  ************************
2  REM  * MOUSEPOINTER         *
3  REM  * BY KEVIN GARBELMAN   *
4  REM  * COPYRIGHT (C) 1987   *
5  REM  * BY MICROSPARC, INC.  *
6  REM  * CONCORD, MA  01742   *
7  REM  ************************
10  GOSUB 500: REM   SET UP MOUSE
20  GOSUB 250: REM      PRINT A SCREEN
30  GOSUB 70: REM    GET A READING
40  GOSUB 400: REM    ACT ON READING
50  GOTO 20
60  REM   MOUSE ROUTINE
70 OL = 160:OD = 2039
80  PRINT  CHR$ (4);"IN#4"
90  VTAB 1: HTAB 40: INPUT "";MX,MY,MB
100  PRINT  CHR$ (4);"IN#0"
110 MY =  INT (MY / 6) + 1
120 MX =  INT (MX / 4) + 1
130  IF MY > 24 THEN MY = 24
140  IF MX > 40 THEN MX = 40
150  IF MB = 2 OR MB =  - 2 THEN  FOR I = 1 TO
     5:DU =  PEEK (49200): NEXT : GOTO 220
160  POKE OD,OL
170 PK = 128 * MY + MX - (984 *  INT ((MY - 1
     ) / 8)) + 895
180 OL =  PEEK (PK)
190 OD = PK
200  POKE PK,170
```

```
210  GOTO 80
220  RETURN
230  REM   END OF GET MOUSE
240  REM    PRINT A SCREEN
250  HOME
260  VTAB 1: PRINT "MOUSEPOINTER"
270  VTAB 2: PRINT "BY KEVIN GARBELMAN": PRINT
     "COPYRIGHT 1987 BY MICROSPARC, INC."
280  VTAB 6: HTAB 5: PRINT " OPTION # 1 "
290  VTAB 8: HTAB 5: PRINT " OPTION # 2 "
300  VTAB 10: HTAB 5: PRINT " OPTION # 3
          OPTION # 4"
310  REM                                  <10SP
     ACES>
320  VTAB 18: HTAB 4: PRINT "INSTRUCTIONS:"
330  VTAB 19: HTAB 4: PRINT "MOVE THE MOUSE T
     O POSITION THE"
340  VTAB 20: HTAB 4: PRINT "CURSOR OVER AN O
     PTION AND PRESS"
350  VTAB 21: HTAB 4: PRINT "THE MOUSE BUTTON
     . THE 'FINISHED'"
360  VTAB 22: HTAB 4: PRINT "OPTION BELOW WIL
     L END THE PROGRAM."
370  VTAB 24: HTAB 5: PRINT "FINISHED";
380  RETURN
390  REM    DEAL WITH READING:
400  HOME : VTAB 12: HTAB 1: PRINT " NO OPTIO
     N CHOSEN"
410  IF MY = 6 THEN  VTAB 12: PRINT " YOU CHO
     SE OPTION # 1"
420  IF MY = 8 THEN  VTAB 12: PRINT " YOU CHO
     SE OPTION # 2"
430  IF MY = 10 AND MX < 20 THEN  VTAB 12: PRINT
     " YOU CHOSE OPTION # 3"
440  IF MY = 10 AND MX > 19 THEN  VTAB 12: PRINT
     "YOU CHOSE OPTION # 4"
450  IF MY = 24 THEN  HOME : VTAB 12: PRINT "
     GOODBYE": END
460  VTAB 18: HTAB 5: PRINT "PUSH THE BUTTON
     TO RETURN'
470  GOSUB 70
480  RETURN
490  REM    *********  MOUSE SET UP  ********
     *
500  PRINT  CHR$ (4);"PR#4": PRINT  CHR$ (1)
510  PRINT  CHR$ (4);"PR#0"
520  RETURN
```

END OF LISTING 1

```
               KEY PERFECT 5.0
                   RUN ON
                MOUSEPOINTER
     =======================================
     CODE-5.0    LINE# - LINE#    CODE-4.0
     --------    -------------    --------
     DC8D2A19        1 -    30      6CF6
     5F31908E       40 -   130      4559
     A52619A9      140 -   230      5012
     CF85BBB7      240 -   330      714F
     892C99F6      340 -   430      93C4
     FAB2F81D      440 -   520      59F8
     60230332 = PROGRAM TOTAL =     052E
```

## LISTING 2: JOYPOINTER

```
1   REM  ************************
2   REM  * JOYPOINTER           *
3   REM  * BY KEVIN GARBELMAN   *
4   REM  * COPYRIGHT (C) 1987   *
5   REM  * BY MICROSPARC, INC.  *
6   REM  * CONCORD, MA  01742   *
7   REM  ************************
10  GOSUB 250: REM    PRINT A SCREEN
20  GOSUB 60: REM   GET A READING
30  GOSUB 390: REM   ACT ON READING
40  GOTO 10
50  REM   JOYSTICK ROUTINE
60 OL = 160:OD = 2039
70 MX =  PDL (0)
80 MY =  PDL (1)
90 MB =  PEEK (49249)
100 MX =  INT (MX / 6)
110 MY =  INT (MY / 10)
120  IF MY > 24 THEN MY = 24
130  IF MX > 40 THEN MX = 40
140  IF MX < 1 THEN MX = 1
150  IF MY < 1 THEN MY = 1
160  IF MB > 127 THEN  FOR I = 1 TO 10:DU =  PEEK
     (49200): NEXT : GOTO 230
170  POKE OD,OL
180 PK = 128 * MY + MX - (984 *  INT ((MY - 1
     ) / 8)) + 895
190 OL =  PEEK (PK)
200 OD = PK
210  POKE PK,170
220  GOTO 70
230  RETURN
240  REM   PRINT A SCREEN
250  HOME
260  VTAB 1: PRINT "JOYPOINTER"
270  VTAB 2: PRINT "BY KEVIN GARBELMAN": PRINT
     "COPYRIGHT 1987 BY MICROSPARC, INC."
280  VTAB 6: HTAB 5: PRINT " OPTION # 1 "
290  VTAB 8: HTAB 5: PRINT " OPTION # 2 "
300  VTAB 10: HTAB 5: PRINT " OPTION # 3
          OPTION # 4"
310  VTAB 18: HTAB 4: PRINT "INSTRUCTIONS:"
320  VTAB 19: HTAB 4: PRINT "MOVE THE JOYSTIC
     K TO POSITION THE"
330  VTAB 20: HTAB 4: PRINT "CURSOR OVER AN O
     PTION AND PRESS"
340  VTAB 21: HTAB 4: PRINT "THE BUTTON.
      THE 'FINISHED'"
350  VTAB 22: HTAB 4: PRINT "OPTION BELOW WIL
     L END THE PROGRAM."
360  VTAB 24: HTAB 5: PRINT "FINISHED";
370  RETURN
380  REM   DEAL WITH READING:
390  HOME : VTAB 12: HTAB 1: PRINT " NO OPTIO
     N CHOSEN"
400  IF MY = 6 THEN  VTAB 12: PRINT " YOU CHO
     SE OPTION # 1"
410  IF MY = 8 THEN  VTAB 12: PRINT " YOU CHO
     SE OPTION # 2"
420  IF MY = 10 AND MX < 20 THEN  VTAB 12: PRINT
     " YOU CHOSE OPTION # 3"
430  IF MY = 10 AND MX > 21 THEN  VTAB 12: PRINT
     " YOU CHOSE OPTION # 4"
440  VTAB 14: HTAB 5: PRINT "HIT THE BUTTON T
     O RETURN."
450  IF MY = 24 THEN  HOME : VTAB 12: PRINT "
     GOODBYE": END
460  GOSUB 60
470  RETURN
```

END OF LISTING 2

```
               KEY PERFECT 5.0
                   RUN ON
                 JOYPOINTER
     =======================================
     CODE-5.0    LINE# - LINE#    CODE-4.0
     --------    -------------    --------
     02B84AB8        1 -    30      684E
     0884ADE8       40 -   130      3BD1
     225E86F3      140 -   230      48B4
     2034D65C      240 -   330      7DBD
     52BA8A76      340 -   430      92EC
     EDEE93B5      440 -   470      23F8
     88367BBD = PROGRAM TOTAL =     04AE
```