MORE SUBROUTINES

Add these special effects subroutines to your Subroutine Library for cinematic screen clearing, protection of screen titles, and sound effects.

by P.S. Dunseath
1056 Harkness Ave.
Ottawa, Ontario
Canada K1V 6P1

Don Ravey presented some excellent suggestions for a Subroutine Library in recent issues of *Nibble*. Here are some additional subroutines for you to try.

SCREEN WIPE FROM TOP AND BOTTOM

The principle here is fairly simple: We VTAB to **Line 24**, HTAB to the first position, then clear the line using **CALL -868**. We then VTAB to the top line and clear it. The iterative loop then progressively clears the second line from the bottom, second from the top, etc., until the entire screen is cleared. We then execute a **CALL -936** (or HOME) command to put the cursor back at the top of the screen, and RETURN. The iterative loop using **J** is a small delay to make the action more dramatic.

```
150 FOR I = 24 TO 13 STEP -1:HTAB
    1:VTAB I: CALL -868: VTAB (25-I):
    CALL -868: FOR J=1 TO 50:NEXT:
    NEXT:CALL -936: RETURN:REM
    SCREEN WIPE FROM TOP AND
    BOTTOM TO MIDDLE
```

SCREEN WIPE FROM MIDDLE TO TOP AND BOTTOM

This subroutine, as you might expect, clears the middle of the screen, and then simultaneously "wipes" upward to the top and downward to the bottom. It's quite eye-catching when used on a full screen.

```
151 FOR I = 13 TO 24: HTAB 1:VTAB
    1:VTAB I:CALL -868:VTAB (25-I):
    CALL -868: FOR J = 1 TO 50: NEXT:
    NEXT: RETURN: REM SCREEN WIPE
    FROM MIDDLE TO TOP AND
    BOTTOM SIMULTANEOUSLY
```

As you can see, this positions the cursor on the middle line, clears it using **CALL -868**, and then positions the cursor alternately one line above the middle, and one line below the middle — clearing each of those lines, and progressively moving farther and farther away from the middle until the top and bottom edges are reached.

CURTAINS

This subroutine is quite attractive; it appears as if the screen were projected on a set of theater curtains which are slowly opened to reveal the blank screen behind. CURTAINS clears a vertical column down the middle of the screen, and then widens the blank area until the left and right edges of the screen are reached:

```
152 FOR Z = 1 TO 20: X = 20 - Z:POKE
    32,X: POKE 33, (2*Z): CALL -936:
    FOR I = 1 TO 50:NEXT:NEXT:RETURN:
    REM CURTAINS
```

The principle is simple, but it took me a while to get it right. We start by defining the left edge of the text window at column 19 with the **POKE 32,X** statement, and the width of the window two columns wide with the **POKE 33, (2*Z)** statement. We then clear the text window (which actually extends only from columns 19 to 21) with the **CALL -936** statement (equivalent to the command **HOME**). We then move the left edge of the window farther and farther to the left while widening it twice as fast (so that it stays centered on the middle of the screen) until we reach the edges, and there we are!

> **"... we are looking at a screen which is slowly closed to view by a pair of sliding doors..."**

SLIDING DOORS SCREEN WIPE

This is an attractive subroutine which is just the opposite of CURTAINS; it appears that we are looking at a screen which is slowly closed to view by a pair of sliding doors from the LEFT and RIGHT sides. We achieve this with a bit of screen trickery.

The width of the text window is set to one character by **POKE 33,1**. Next, we define the LEFT edge of the window as being at column 39 by **POKE 32,(40-I)**. We then clear the window — actually just the last column — using **CALL -936**. Then we redefine the LEFT edge as being at column 1, and repeat. The iterative loop progressively clears one column inward from each edge, and on completion we reset the values to normal:

```
153 POKE 33,1: FOR I=0 TO 20: POKE 32,I:
    CALL -936: POKE 32,(40 - I): CALL
    -936: FOR J=1 TO 50: NEXT:NEXT:
    POKE 33,40:POKE 32,0: RETURN:REM
    "SLIDING DOORS" SCREEN WIPE
    FROM LEFT AND RIGHT EDGES TO
    CENTER
```

VENETIAN BLINDS

Here we simultaneously clear every sixth line and then use our iteration to clear the next lines until the screen is clear. Then we **CALL -936** to go HOME.

```
154 HTAB 1: FOR I=1 TO 6: VTAB I:CALL-
    868:VTAB(2*I): CALL -868: VTAB (3*I):
    CALL -868: VTAB (4*I): CALL -868:
    FOR J=1 TO 100: NEXT:NEXT:CALL
    -936:RETURN: REM "VENETIAN
    BLINDS" SCREEN WIPE
```

"THAT'S ALL, FOLKS"

This subroutine is the most complicated screen wipe of all, both because there is so much going on, and because of the asymmetry of the text window. If we were to clear from all four edges at the same rate, we would not meet in the middle. This subroutine allows us to correct that. (There may be an easier way to do it, but I have not yet found it!)

Basically, we combine the principles used in the SCREEN WIPE FROM TOP AND BOTTOM, and the SLIDING DOORS SCREEN WIPE, except that the width of the text window is adjusted to compensate for the greater width than height of the screen. In addition, after **each** iteration, we must reset the values to normal or our clearance at the top and bottom won't work. The delay loop is shorter, since the entire operation is slower anyway.

When typing this in, I strongly advise you to do so without spaces (your Apple will parse them back in when you list in any case), or there won't be enough room on one program line for the REM statement. (For this same reason, I used **HOME** in place of a final **CALL -936**.)

```
155 FOR I=1 TO 12: POKE 35,I: CALL -936:
    POKE 34,(24-I): CALL -936: POKE 35,
    24: POKE 34,0: POKE 33,I: POKE 32,
    (I-1): CALL -936: POKE 32, (40-(2*I)):
    CALL -936: POKE 32,0:POKE 33,40:
    FOR J=1 TO 25: NEXT:NEXT:HOME:
    RETURN: REM SCREEN WIPE FROM
    4 EDGES TO MIDDLE
```

PROTECT TOP

Our final two subroutines are used to print a line on the screen, and then protect it against scrolling off the screen. This is useful if, for example, you want to keep column headings fixed in place at the top of the screen while data that is being printed scrolls.

We do this by printing our heading — here, the string ST$ — on the top line and then defining the top of the text screen as beginning below it.

```
158 HOME:PRINT ST$:PRINT:POKE
    34,2:RETURN:REM PRINTS ST$ AND
    PROTECTS UNTIL 'POKE 34,0' IS
    ENCOUNTERED
```

To prevent the heading from scrolling off the screen as soon as a **POKE 34,0** is read, precede this POKE with **VTAB 23**.

PROTECT BOTTOM

This subroutine does the same thing as PROTECT TOP, but at the bottom of the screen; and is similar in concept:

```
159 VTAB (24):PRINT ST$: POKE 35,22:
    HOME:RETURN:REM PRINTS ST$ ON
    BOTTOM LINE AND PROTECTS
    UNTIL A 'POKE 35,24' IS
    ENCOUNTERED
```

MUSIC

We all know that the Apple will play music — many games use this feature — and we all know the basic principle: Any reference to the speaker location (-16336) toggles the speaker cone. Obviously, all we need to do is toggle it rapidly enough, and it will make an audible tone; do it fast enough, and we can produce musical tones.

Did I say "**All** we need to do"? None of my Apple manuals explained how to produce music, and the fastest iterative loop I could construct in Applesoft produced nothing higher in tone than a buzz. The reason for this is that Applesoft is too slow; we must toggle the speaker at machine language speed to produce the tones we need.

As it happens, we can do this either with a machine language routine, which is BLOADED when needed; or by POKEing the equivalent values into memory from an Applesoft program. Since the POKEs don't provide much insight, let's look at the machine language routine first. As is normal practice, we will load this into memory at $0300. We will reserve the first two locations for values needed in the program for PITCH and DURATION, so the main routine begins at $0302.

Location $C030 (hex) is equivalent to -16336 (dec), which is the location of the speaker toggle. The program works as follows: Prior to starting, we place a value for PITCH in $0300 (dec 768) as 'X', and for TIME (i.e., duration) in $0301 (dec 769). We then reference the speaker, and decrement **X** and **Y** alternately.

Y, having not yet been initialized, holds a value of zero (which is the same as 256 in 8-bit binary). When **X** reaches zero, we reload it with its original value and toggle the speaker; then repeat. The speaker will therefore be toggled at a rate dependent on **X** (PITCH); the higher **X** is, the longer the interval and the lower the pitch.

When **Y** reaches zero (again), we decrement $0301 (TIME) and start over. Thus, **TIME** is multiplied by **Y** (256) so that it represents a reasonable time to us humans. (If it were not, it would be measured in milliseconds.)

When TIME reaches zero, we exit the subroutine.

```
0302-  AD 30 C0   LDA $C030
0305-  88         DEY
0306-  D0 05      BNE $030D
0308-  CE 0103    DEC $0301
030B-  F0 09      BEQ $0316
030D-  CA         DEX
030E-  D0 F5      BNE $0305
0310-  AE 00 03   LDX $0300
0313-  4C 02 03   JMP $0302
0316-  60         RTS
```

This may then be saved by typing **BSAVE MUSIC (ML), A$0302, L$0015**. In Applesoft, we can POKE these values into memory beginning at location $0302 (770, decimal) using the method in **lines 40 and 45** of the listing below:

```
 5 GOSUB 40: REM MUSIC ROUTINE
10 READ PITCH, TIME
20 IF PITCH = 0 THEN GOTO 60
30 POKE 768, PITCH: POKE 769, TIME:
   CALL 770
35 GOTO 10
40 POKE 770,173: POKE 771,48: POKE
   772, 192: POKE 773,136: POKE 774,208:
   POKE 775,5: POKE 776,206: POKE 777,1:
   POKE 778,3: POKE 779,240
45 POKE 780,9: POKE 781,202:POKE
   782,208:POKE 783,245:POKE 784,
   174:POKE 785,0:POKE 786,3:POKE
   787,76:POKE 788,2:POKE 789,3:POKE
   790,96:RETURN
50 DATA 80,80,200,80,180,80,200,80,150,
   80,160,120,200,80,200,80,180,80,200,80,
   135,80,150,180
55 DATA 0,0
60 END
```

Of course, the DATA statements in **line 50** will change with each tune you wish to play. (The ones given here play the first few bars of "Happy Birthday to You.") Also, note that the DATA statements are in pairs, with the first number in each pair obviously being PITCH and the second, TIME (or duration of the note). The pair of zeros in 55 are there to tell the program that we're finished.

A Saturday morning at the Apple keyboard led me to derive, by trial and error, the following values (if anyone can improve on these, I'd be grateful to hear about it!):

PITCH	NOTE	VALUE
	B	210
	mid C	200
	D	180
	E	160
	F	150
	G	135
	A	120
	B	107
	C	100
	D	95

TIME (duration)

Full note 180
Half note 80

Other values can also be used: a TIME of 4 gives little more than a buzz, while 16 is a very short note, 40 about a quarter note, and 255 (the maximum value) a **long** note.

Try experimenting with different octaves by using other values for PITCH.

I hope you enjoy using these subroutines and find them to be worthwhile additions to your Subroutine Library.

```
1   REM       ************************
2   REM       * MORE SUBROUTINES DEMO *
3   REM       *    BY  P.S. DUNSEATH  *
4   REM       *  COPYRIGHT (C)  1983  *
5   REM       *  BY MICROSPARC, INC.  *
6   REM       *  LINCOLN, MA.  01773  *
7   REM       ************************
9   TEXT : HOME : VTAB 11: PRINT "** COPYRIGHT 1983 BY
     MICROSPARC, INC. **": FOR I = 1 TO 3000: NEXT
10  REM  DEMONSTRATION PROGRAM
15  GOSUB 98: GOSUB 150: GOSUB 98: GOSUB 151: GOSUB 98
     : GOSUB 152: GOSUB 98: GOSUB 153
20  GOSUB 98: GOSUB 154: GOSUB 98: GOSUB 155
25  INVERSE
30  ST$ = "TOP LINE PROTECTED"
35  FOR I = 1 TO 500: NEXT : GOSUB 158
38  NORMAL
40  LIST
50  VTAB 23
60  POKE 34,0
65  INVERSE
70  ST$ = "BOTTOM LINE PROTECTED"
80  FOR I = 1 TO 500: NEXT : GOSUB 159
82  NORMAL
85  LIST
90  POKE 35,24
95  CALL  - 936
96  END
98  FOR J = 1 TO 2000: NEXT : CALL  - 1998: FOR J = 1 TO
     2000: NEXT : RETURN : REM  FILLS SCREEN FOR WIPIN
     G DEMO*****************************************
100 END
150 FOR I = 24 TO 13 STEP  - 1: HTAB I: VTAB I: CALL
     - 868: VTAB (25 - I): CALL  - 868: FOR J = 1 TO
    50: NEXT : NEXT : CALL  - 936: RETURN : REM   SCR
    EEN WIPE FROM TOP AND BOTTOM TO MIDDLE**********

151 FOR I = 13 TO 24: HTAB I: VTAB I: CALL  - 868: VTAB
    (25 - I): CALL  - 868: FOR J = 1 TO 50: NEXT : NEXT
    : RETURN : REM  SCREEN WIPE FROM MIDDLE TO TOP AN
    D BOTTOM ***************************************

152 FOR Z = 1 TO 20:X = 20 - Z: POKE 32,X: POKE 33,(2
     * Z): CALL  - 936: FOR I = 1 TO 50: NEXT : NEXT
    : RETURN : REM  CURTAINS************************
    ***************

153 POKE 33,1: FOR I = 0 TO 20: POKE 32,I: CALL  - 93
    6: POKE 32,(40 - I): CALL  - 936: FOR J = 1 TO 50
    : NEXT : NEXT : POKE 33,40: POKE 32,0: RETURN : REM
     "SLIDING DOORS" SCREEN WIPE FROM LEFT AND RIGHT
    EDGES TO CENTRE*********************************
    ******
154 HTAB 1: FOR I = 1 TO 6: VTAB I: CALL  - 868: VTAB
    (2 * I): CALL  - 868: VTAB (3 * I): CALL  - 868: VTAB
    (4 * I): CALL  - 868: FOR J = 1 TO 100: NEXT : NEXT
    : CALL  - 936: RETURN : REM  "VENETIAN BLINDS" SC
    REEN WIPE***************************************

155 FOR I = 1 TO 12: POKE 35,I: CALL  - 936: POKE 34,
    (24 - I): CALL  - 936: POKE 35,24: POKE 34,0: POKE
    33,I: POKE 32,(I - 1): CALL  - 936: POKE 32,(41 -
    (2 * I)): CALL  - 936: POKE 32,0: POKE 33,40: FOR
    J = 1 TO 25: NEXT : NEXT : HOME : RETURN : REM  S
    CREEN WIPE FROM 4 EDGES TO MIDDLE**************
158 HOME : PRINT ST$: PRINT : POKE 34,2: RETURN : REM
     PRINTS ST$ AND PROTECTS UNTIL 'POKE 34,0' IS ENC
    OUNTERED****************************************

159 VTAB (24): PRINT ST$: POKE 35,22: HOME : RETURN :
     REM  PRINTS ST$ ON BOTTOM LINE AND PROTECTS UNTI
    L A 'POKE 35,24' IS ENCOUNTERED*****************
    **********************
```

```
KEY PERFECT 4.0
RUN ON
MORE SUBROUTINES DEMO
=====================================
     CODE       LINE# - LINE#
     ------     -------------
     9325          1 -    15
     3B4F         20 -    70
     C37A         80 -   151
     01ADA8      152 -   159
TOTAL PROGRAM CHECK IS : 0680
```

```
CHECK CODE 3.0
ON: MORE SUBROUTINES DEMO
TYPE: A

LENGTH: 0605
CHECKSUM: D7
```