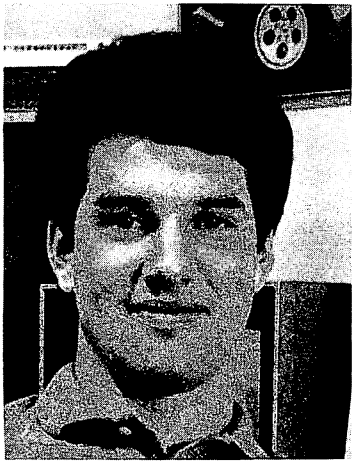
SAMS

22386

TI-99/4A™ Graphics and Sounds

Timothy Orr Knight

TI-99/4A™ Graphics and Sounds



Timothy Orr Knight is a member of the "younger generation" who started "playing around" with computers about five years ago. He developed a strong interest in "modems," the devices that allow one microcomputer to talk to another, and that interest led to the writing of his first book. While attending high school, Tim has been busy writing reviews, articles, and programs for such national magazines as *InfoWorld* and *SoftSide*, plus programming computer games, such as *EVADE*, *SPACE CHASE*, and *SPOX*, for various software companies.

Tim is a busy 17-year-old who enjoys being with friends and being involved in all areas of school activites, but most of all . . . he enjoys working. Thus, it was no surprise when Tim formed his own software company, *Knight Software, Inc.*, in the summer of 1983 to produce both business and game software for a wide variety of computers. Tim Knight's other SAMS books are *The World Connection; Megabucks from Your Microcomputer;* the Combo Pack, *Commodore 64 Graphics and Sounds;* and *Graphics and Sounds on the IBM PC*. He is coauthor of another Combo Pack, *Commodore 64 BASIC Programs*.

TI-99/4A™ Graphics and Sounds

by
Timothy Orr Knight

Adapted for the TI-99/4A™
by
Gregory L. Guntle

Howard W. Sams & Co., Inc.
4300 West 62nd St.
Indianapolis, Indiana 46268 USA

FIRST EDITION
FIRST PRINTING — 1984

International Standard Book Number: 0-672-22386-4
Library of Congress Catalog Card Number: 84-50804

Edited by *Katherine Stuart Ewing*
Printed in the United States of America.

Preface

This book is for TI-99/4A home computer users who want to learn more about the graphics and sound capabilities of their computer.

Learning to master graphics and sounds on the TI-99/4A is not very difficult. The hardest part of learning these features is understanding the BASIC commands that control graphics and sounds. Most books containing materials on this subject are very complex and hard to understand. This book provides you with the tools needed to fully use your computer's graphics, sound, and music capabilities. What sets this book apart from others like it? It provides sample programs, complete with line-by-line explanations of the program code, thus clearing up a lot of confusion.

TI-99/4A Graphics and Sounds will serve as a beginners guide for those wanting to learn, a reference book for more experienced users, and a handy collection of program routines. Anyone can master the graphics and sound features of the TI-99/4A who has the knowledge to do so. I hope this book will provide that knowledge.

The programs in this book were originally written for the Commodore 64. My thanks to Gregory L. Guntle for adapting them for the TI-99/4A computer.

TIMOTHY ORR KNIGHT

A NOTE TO THE READER

The programs in this book were not written as applications software but as educational examples of what your personal computer can do. All of the programs have been tested and work on the machine configuration for which they were designed. The programs, or subroutines, are unprotected. This means that you can modify them to better understand how they work or to fit a different machine configuration.

What is a Combo Pack?

A Combo Pack, like this package, is a step beyond your average technical book. While most books give you programming examples through printed listings (which we do here), Combo Packs provide the book and the listings recorded on magnetic media, either diskette, cassette tape, or both.

Every effort has been made to be clear, concise, and informative about how these programs and routines work. If you experience any difficulty with the software operations, the solution can be found in the book or in your computer manuals.

We are rather proud of the time and effort that went into preparing the Combo Pack. If you have purchased the Combo Pack and have enjoyed using it, let us know your thoughts. Your comments will be valuable in preparing future Combo Packs.

LOADING INSTRUCTIONS

The cassette accompanying this Combo Pack contains the subroutine listings and/or program listings printed in the book.

To load a cassette file from this tape, perform the following steps:

1. Put the cassette into the cassette recorder.
2. Position the tape at the beginning of the subroutine or program you wish to load.
3. Type **OLD CS1**
   Press <ENTER>
4. Follow the directions as they appear on your video screen.

This will cause the next program on the tape to load into the computer's memory. When the program is loaded, it is ready to be used as described in the book.

The following list shows the listing names and tape counter positions for the contents of the cassette tape. These numbers are approximate and may vary from recorder to recorder. They should, however, assist you in locating the programs you are searching for.

Tape Directory

Program Names	Counter Location
Colors	0
Colors&Gr	8
Rnd-Dots	17
Strght-Ln	23
Vert-Line	29
Diag-Line	35
Squares	41
Circles	47
Triangles	54
Variety	60
Giant-Hi	69
Rd-Pnoise	78
Rd-Wnoise	84
Up-Sound	90
Down-Snd	96
Up&Down	103
M-Scales	110
Rectangle	117
Lg-Triang	125
Cube	134
Expand-Fs	148
F-Saucer	173
Saucer-Tr	181
Sauc-Cube	191
Bombardier	210
Swanee	220
Snd-Devel	231
Mus-Trans	248
Brahms	260
Rgh-Scroll	276
Up-Scroll	285
Dn-Scroll	294
Scroll-L&R	304
Alpha-Scr	314
Character	327
Chr-Chngr	340
Data-Stat	355

Contents

CHAPTER 6

APPENDIX A

APPENDIX B

APPENDIX C

APPENDIX D

APPENDIX E

Chapter 1

An Introduction to Graphics and Sounds

The TI-99/4A home computer has the ability to produce a wide variety of graphics, sounds, and music. Yet these are often neglected because many people lack the knowledge necessary to fully utilize the graphics and sounds capabilities of the TI-99/4A.

This book contains program listings to demonstrate these powerful features. After each listing is an explanation of the programming lines. Feel free to use these programs, modify them, or attempt to create your own graphics and sounds. Try experimenting with the programs. Make changes here and there and see what happens. The main objective is to enjoy your TI-99/4A.

A few reminders before we begin:

1. When you see this statement: FCTN + 1 or FCTN + any number or key, that means to hold down the FCTN key (located just to the right of the spacebar) and press the specified key at the same time.
2. The "panic button" on the TI-99/4A is the FCTN + 4 key. By pressing these two keys simultaneously, you can stop most program exe-

cutions without losing your program. If something unusual happens on the screen, hold down on these two keys and the program will stop.

3. Use either upper or lower case letters when typing the programs.

Have fun!

Chapter 2

Colors and Basic Graphics

The TI-99/4A produces a wide variety of colors. The screen can display 16 different colors. Every character that is displayed has a *foreground* color and a *background* color, each of which can be set to any one of 16 colors. (See Appendix A for a listing of the colors with their corresponding color number.)

COLOR CHANGER

Listing 2-1 steps through the available colors. You will see how the different colors can brighten and enhance the characters as well as the screen. This listing uses the FCTN key. Press FCTN + 1 to change the foreground color. To change the background color, press FCTN + 3. Pressing FCTN + 5 produces random screen colors until you press another key. These keys are displayed on the screen as you run the program. When you run the program, your screen will look similar to Fig. 2-1.

Listing 2-1

```
10 REM   COLORS
20 T$="**  COLORS  **"
30 CALL CLEAR
```

cont. on next page

Listing 2–1—cont.

```
40 PRINT TAB(INT((29-LEN(TT$))/2));TT$
50 PRINT :::"  THIS PROGRAM ALLOWS THE"
60 PRINT :"USER TO SEE THE DIFFERENT"::"SCREEN COLORS,
   BACKGROUND"
70 PRINT :"COLORS AND FOREGROUND COLORS"::"THAT THE
   TI-99/4A CAN"::"PRODUCE.":::
80 PRINT "  PRESS ANY KEY TO BEGIN"
90 CALL KEY(0,K,S)
100 IF S<=0 THEN 90
110 CALL CLEAR
120 RANDOMIZE
130 FCOLR=1
140 BCOLR=2
150 PRINT "    TO CHANGE          PRESS"
160 PRINT "    ---------          -----"::
170 PRINT "FOREGROUND COLORS    FCTN+1"::"BACKGROUND
   COLORS    FCTN+3"::
180 PRINT "SCREEN COLORS                    RANDOMLY
   FCTN+5"::
190 PRINT "STOP RANDOM COLORS  ANY KEY"::
200 PRINT ::::::"     PRESS  X  TO EXIT"
210 CALL KEY(3,KEY,STATUS)
220 IF KEY=3 THEN 270
230 IF KEY=7 THEN 360
240 IF KEY=14 THEN 450
250 IF KEY=88 THEN 520
260 GOTO 210
270 FCOLR=FCOLR+1
280 IF FCOLR<=16 THEN 300
290 FCOLR=2
300 FOR NUMSETS=1 TO 12
310 CALL COLOR(NUMSETS,FCOLR,BCOLR)
320 CALL KEY(3,KEY,STATUS)
330 IF KEY=88 THEN 520
340 NEXT NUMSETS
350 GOTO 210
360 BCOLR=BCOLR+1
370 IF BCOLR<=16 THEN 390
380 BCOLR=1
390 FOR NUMSETS=1 TO 12
400 CALL COLOR(NUMSETS,FCOLR,BCOLR)
410 CALL KEY(3,KEY,STATUS)
420 IF KEY=88 THEN 520
430 NEXT NUMSETS
440 GOTO 210
450 REM   RANDOMLY CHANGE SCREEN COLORS
460 SCRN=INT(RND*15)+1
470 CALL SCREEN(SCRN)
480 CALL KEY(3,KEY,STATUS)
490 IF KEY=88 THEN 520
500 IF STATUS=0 THEN 460
510 GOTO 210
520 CALL CLEAR
530 CALL SCREEN(4)
```

```
540 FOR I=1 TO 12
550 CALL COLOR(I,2,1)
560 NEXT I
570 END
```

Explanation of Program

10-40	Clear screen and center title of program
50-100	Display introduction screen and wait for a key to be pressed
110-200	Clear screen, set randomizer routine, set initial foreground and background colors, display keys to press to work program
210	Wait for a key to be pressed, return ASCII value for key pressed; i.e., if key pressed was "A", KEY returns 65 as value (refer to ASCII table in Appendix B)
220-260	Wait for valid key to be pressed and act accordingly
270-290	Increment foreground color and check to make sure foreground color does not go over 16 (max number of colors). Reset color if over maximum amount
300-350	Start loop for changing each character set. Each character set's colors have to be changed. (More on character sets in next section)
360-440	Increment background color and check to see if over maximum limit of 16. If over, reset color. Loop through all the characters sets

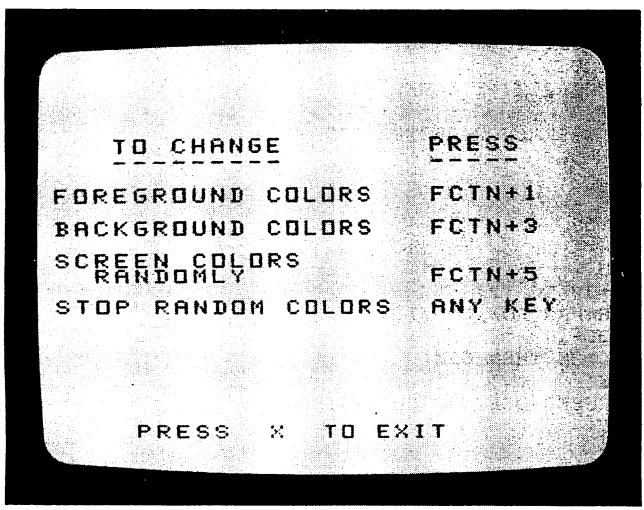


Fig. 2-1. Colors.

COLORS AND GRAPHICS

Listing 2-2 also deals with colors on the TI-99/4A. This time the program steps through the colors for you. Sit back and watch as the screen advances one color at a time. The 16 foreground colors are displayed on each of the background colors until all possible color combinations are displayed. The *characters* displayed here are the normal TI-99/4A characters (all the keys that are on the keyboard) and a few user-defined characters (more on user-defined characters in the next section). When you run the program, your screen will look similar to Fig. 2-2.

NOTE: Sometimes the screen will be blank. DON'T PANIC! The screen will only look blank when the foreground color matches the screen color; the characters blend into the screen.

Listing 2-2

```
10 REM   COLORS & GRAPHICS
20 TT$="**  COLORS & GRAPHICS  **"
30 CALL CLEAR
40 PRINT TAB(INT((29-LEN(TT$))/2));TT$
50 PRINT :::"  THIS PROGRAM DEMONSTRATES"::"THE NORMAL
    CHARACTERS AND"::"SOME USER-DEFINED CHARACTERS"
60 PRINT :"AS WELL AS THE DIFFERENT"::"COLORS THAT THE
    TI-99/4A"::"CAN PRODUCE."::
70 PRINT :::"  PRESS ANY KEY TO BEGIN"
80 CALL KEY(0,KEY,STATUS)
90 IF STATUS<=0 THEN 80
100 CALL CLEAR
110 FOR I=1 TO 10
120 READ CODE$
130 CALL CHAR(127+I,CODE$)
140 NEXT I
150 A$="* NORMAL CHARACTERS *"
160 CALL CLEAR
170 CN=4
180 RN=2
190 GOSUB 680
200 ROW=4
210 COL=10
220 FOR I=32 TO 127
230 CALL HCHAR(ROW,COL,I)
240 COL=COL+1
250 IF COL>20 THEN 650
260 NEXT I
270 A$="* USER-DEFINED CHARACTERS *"
280 CN=2
290 RN=14
300 GOSUB 680
310 FOR I=128 TO 137
320 CALL HCHAR(16,I-127+9,I)
330 NEXT I
```

```
340 A$="   AND THESE COLORS."
350 FOR DELAY=1 TO 1000
360 NEXT DELAY
370 RN=20
380 CN=2
390 GOSUB 680
400 A$="   PRESS  X  TO EXIT"
410 RN=24
420 CN=2
430 GOSUB 680
440 FOR COLR=2 TO 16
450 CALL SCREEN(COLR)
460 FOR CHCOLR=1 TO 16
470 FOR NUMSETS=1 TO 14
480 CALL COLOR(NUMSETS,CHCOLR,1)
490 CALL KEY(0,KEY,STATUS)
500 IF (KEY=88)+(KEY=120)THEN 630
510 NEXT NUMSETS
520 NEXT CHCOLR
530 NEXT COLR
540 CALL CLEAR
550 CALL SCREEN(4)
560 FOR I=1 TO 14
570 CALL COLOR(I,2,1)
580 NEXT I
590 PRINT "WANT TO SEE CHARACTERS      AND COLORS
    AGAIN? (Y/N)"
600 CALL KEY(0,KEY,STATUS)
610 IF (KEY<>89)*(KEY<>121)*(KEY<>78)*(KEY<>110)THEN
    600
620 IF (KEY=89)+(KEY=121)THEN 150
630 CALL CLEAR
640 END
650 COL=10
660 ROW=ROW+1
670 GOTO 260
680 FOR I=1 TO LEN(A$)
690 CALL HCHAR(RN,CN+I,ASC(SEG$(A$,I,1)))
700 NEXT I
710 RETURN
720 DATA 0000247E7E7E3C18
730 DATA 0010387CFE7C3810
740 DATA 0010387CFE101010
750 DATA 00000604FC7C4848
760 DATA 3C7EFFFFFFFF7E3C
770 DATA 00000000183C7EFF
780 DATA 00000000FFFFFFFF
790 DATA F0F0F0F0F0F0F0F0
800 DATA FFFFFFFF00000000
810 DATA 0F0F0F0F0F0F0F0F
```

Explanation of Program

10-90 Clear screen, center title of program, display introduction screen

100-140	Set up user-defined graphics characters
150-260	Display normal TI-99/4A characters in middle of screen
270-330	Display user-defined graphics characters in middle of screen
340-430	Short delay before changing colors. Show you how to exit program early if need arises
440-450	Start loop that changes screen color
460	Start loop that changes foreground color
470-500	Start looping through all character sets to adjust them to new color and check to see if early exit key is pressed
510-530	Increment character set until all done. Then increment foreground color until all 16 colors have been displayed. Increment screen color
540-640	Reset characters and screen color to default colors. Ask if you want to see colors again and if response is no, clear screen and exit
650-670	Advance row on which characters are printed
680-710	Print each character to specific row and column. This routine requires a row number, column number, and characters to display. Characters are contained in one string, A$
720-810	These are numbers needed to create user-defined characters

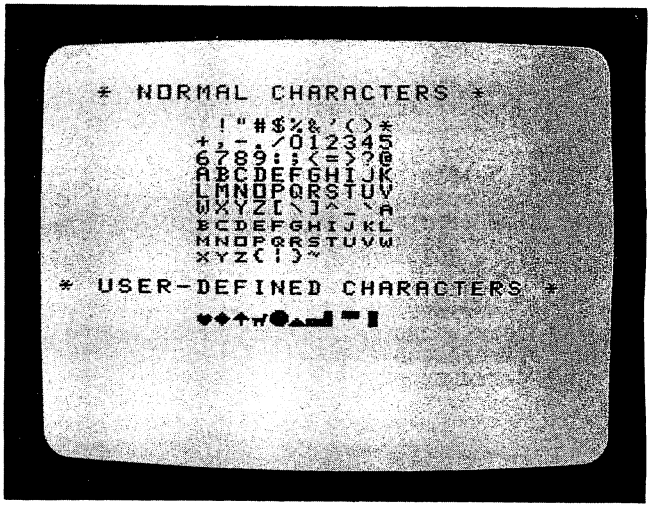


Fig. 2-2. Colors & Graphics.

BASIC GRAPHICS

The TI-99/4A has the capability to display virtually any character you can imagine. Well, any character that you can draw or design. Graphics characters are created using a *bit-map*. Fig. 2-3 shows the 8 x 8 bit-map.

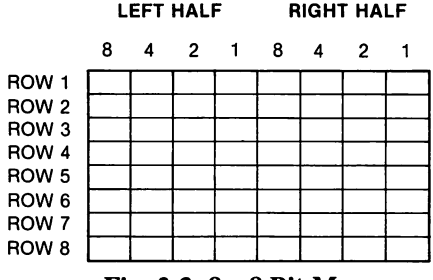


Fig. 2-3. 8 x 8 Bit-Map.

Every character has an 8 x 8 bit-map representation. Before we jump right into some examples, there is one other topic that has to be mentioned. That topic is *hexadecimals*. Hexadecimal notation is another way of counting, but there are only 16 different numbers in hexadecimal compared to the unlimited amount of numbers in the normal counting method. Here are the 16 different hexadecimal numbers and their associated numeric value:

Number	Hexadecimal Notation
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F

As you can see, there is not much difference between the two sets of numbers. The reasons for this notation: the computer can convert hexade-

cimal numbers to graphics representation faster, and it uses less memory space. You also don't have to type as much.

Let's take a closer look at the bit-map. The bit-map is divided into left and right halves. Each half is made up of eight rows consisting of four columns. The rows are numbered 1 through 8 from top to bottom. The columns are numbered 8, 4, 2 and 1 from left to right. In a moment, you'll see why they are numbered like that. Fig. 2-4 shows the two halves of the bit-map with their corresponding rows and columns marked. Now we are ready to start designing graphics characters.

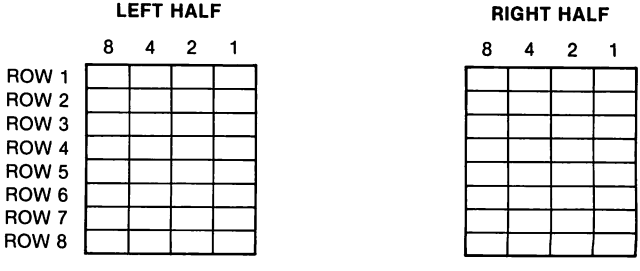


Fig. 2-4. Two Halves of a Bit-Map.

As mentioned earlier, computers convert hexadecimal notations into graphics representation. How do you get a hexadecimal notation for the graphics character you have designed? What you have to do is color squares that you want to appear on the screen for that character. Be sure to use the 8 x 8 map for this procedure. We will use the two halves to calculate the hexadecimal notation in a moment. Once you have colored the appropriate squares, simple mathematics converts the colored squares into hexadecimal notations. The best way to understand this is to look at an example. Let's say you want to generate a small dot. The bit-map would look like Fig. 2-5.

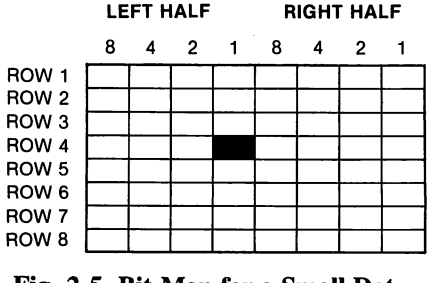


Fig. 2-5. Bit-Map for a Small Dot.

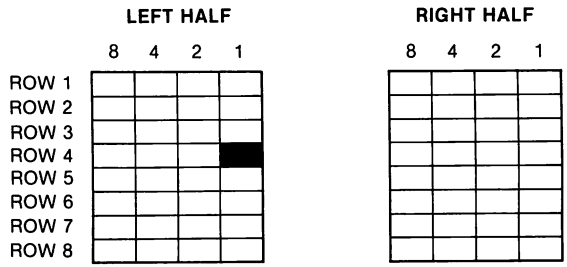


Fig. 2-6. Small Dot Bit-Map Divided into Two Halves.

Now break that bit-map into two halves (see Fig. 2-6).

Once you have done that, the rest is simple. If a block of the bit-map is not colored, the value for that block is 0. If a block of the bit-map is colored, then the value corresponds to that column number. (See Appendix D for the hexadecimal value corresponding to the colored blocks of the bit-map.) In our example, the left half, row 4, column 1 is equal to a value of 1. If the small colored dot was in row 1, column 8, the value would be 8. To complete the calculation, add all the values for each block in each row. Do this for both halves. Your results should look like Fig. 2-7.

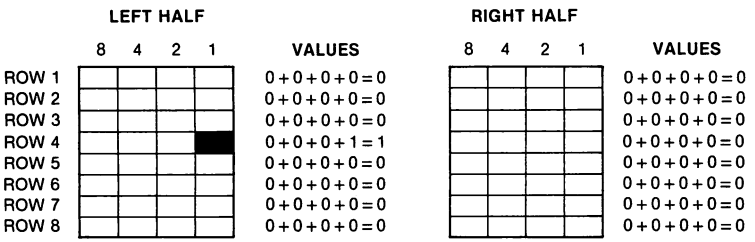


Fig. 2-7. Calculation of an 8 x 8 Bit-Map.

So far we have calculated the numeric value for each row in both halves. But we're not quite ready to enter this information into the TI-99/4A. Remember, the computer has to have the number in hexadecimal. So let's convert the totals for each row into hexadecimal. The conversion from numeric to hexadecimal is straightforward. (Look back to the beginning of this section for the conversion table.) The last step before using the computer is to combine the hexadecimal numbers into one long number. To do this, start at the Left Half, Row 1 total and write down that number. Then move over to the Right Half, Row 1 total and write that beside the Left Half, Row 1 total. Now get the Left Half, Row 2 total and write that number beside the Right Half, Row 1 total. Keep writing the totals, one after another, in this form. Congratulations! You have just completed your

first user-defined graphics character. Your hexadecimal line of numbers should look like Fig. 2-8.

Hexadecimal
0000001000000000

Fig. 2-8. Hexadecimal Notation for the Small Dot.

Even though this isn't a hard example, the steps taken to achieve the hexadecimal notation are used for every graphics character you design.

The hardest part of graphics is done. It really isn't that complicated once you step through it. Remember the hexadecimal number, because we are going to use it shortly. Before we use it, though, we have to look briefly at the command the TI-99/4A uses to generate the characters you have designed.

CALL CHAR(char-code,"hexadecimal notation") is the command needed to reproduce our graphics representation. As you can see, your hexadecimal number goes in the second part of the command. You will need to put quotation marks around the number and place it into the command. The first part, **char-code** needs a little explaining.

Every character the TI-99/4A can generate has an *ASCII value* assigned to it. An ASCII value is the number that represents that particular character. For instance, the letter "A" is represented by the number 65, the letter "B" by the number 66, and so on. (See Appendix B for a complete list of the ASCII codes for the TI-99/4A.) The **char-code** in the command is the number you want to represent the graphics character.

ASCII codes 32-127 are used by the TI-99/4A for character representation. That leaves ASCII 128-159 left for your use. (NOTE: You may use any ASCII code [32-159] that you want for your character, but, for now, let's stick to the codes 128-159. We'll use the other ASCII codes in Chapter 6.)

To give the character a certain color, use the CALL COLOR command. The format for the command is **CALL COLOR(char-set-num, foreground color,background color)**. See the previous section for a discussion of the last two parameters. The first parameter, **char-set-num** is the character set number that corresponds to the ASCII code where you have stored your graphics character. The TI-99/4A divides its standard ASCII codes (32-127) into subgroups. Each subgroup contains eight ASCII codes. For instance, the ASCII codes 32 through 39 are in character set number 1. To change the color of any one of the ASCII codes from 32-39, place a 1 in the parameter, **char-set-num.** (See Appendix C for a complete

breakdown of the ASCII codes into their corresponding character set numbers.)

To print that character, once it has been defined, use either the **CALL HCHAR(row,col,ascii code)** or **CALL VCHAR(row,col,ascii code)**. Replace **row** and **col** with the screen location (see Fig. 2-9 for a grid of the screen showing the rows and columns) at which you want your graphics representation to appear, and replace **ascii code** with the ASCII code you have chosen for your character. These two commands will cause the computer to display the character you have designed.

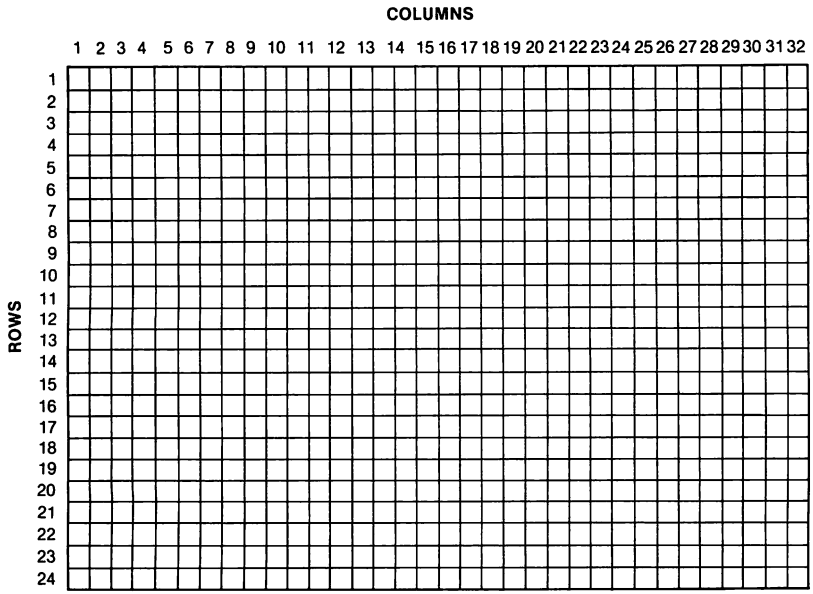


Fig. 2-9. Grid of the Screen.

RANDOM DOTS

Listing 2-3 is a program using the small dot you have just created. It stores the hexadecimal notation in ASCII code 128. This program displays 150 small dots all over the screen in a random fashion.

Listing 2-3

```
10 REM      RANDOM DOTS
20 TT$="**  RANDOM DOTS  **"
30 CALL CLEAR
40 PRINT TAB(INT((29-LEN(TT$))/2));TT$
```

cont. on next page

Listing 2–3—cont.

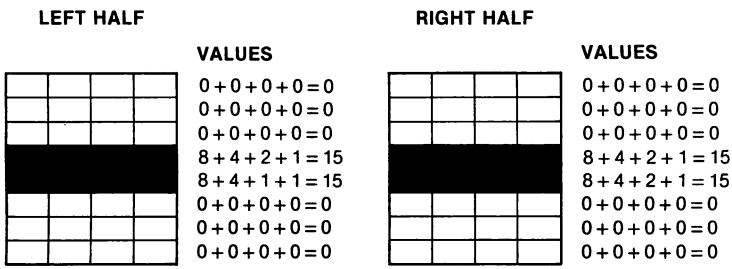
```
50 PRINT :::"  THIS PROGRAM PLOTS RANDOM":::"DOTS ON
   THE SCREEN."
60 PRINT :::::::"  PRESS ANY KEY TO BEGIN"
70 CALL KEY(3,KEY,STATUS)
80 IF STATUS<=0 THEN 70
90 CALL CHAR(128,"0000001000000000")
100 CALL COLOR(13,2,1)
110 CALL SCREEN(15)
120 CALL CLEAR
130 FOR NUMDOTS=1 TO 150
140 ROW=INT(RND*24)+1
150 COL=INT(RND*32)+1
160 CALL HCHAR(ROW,COL,128)
170 NEXT NUMDOTS
180 CALL SCREEN(4)
190 CALL CLEAR
200 PRINT "PLOT MORE RANDOM            DOTS? (Y/N)"
210 CALL KEY(3,KEY,STATUS)
220 IF (KEY<>89)*(KEY<>78)THEN 210
230 IF KEY=89 THEN 110
240 CALL CLEAR
250 END
```

Explanation of Program

10-80	Clear screen, display introduction screen, and wait for a key to be pressed
90	Define character. This is hexadecimal notation for small dot. The dot's hexadecimal representation is then stored at ASCII code 128
100-120	Set color that dot should be, set screen color, and clear screen before printing dots
130-170	Loop 150 times, printing dot in random rows and columns on screen
180-250	Clear screen and ask if you want to see more random dots. If response is no, clear screen and exit program

STRAIGHT LINE

Listing 2-4 draws a straight line across the screen. The hexadecimal notation is a little different from the small dot. Here, the hexadecimal number is "000000FFFF000000". Fig. 2-10 shows the bit-map representation of this number. When you run the program, your screen will look similar to Fig. 2-11.



TRANSLATION OF NUMERIC VALUES TO HEXADECIMAL.

0 NUMERIC = 0 HEXADECIMAL, 15 NUMERIC = F HEXADECIMAL

SO WHEN YOU COMBINE THE TWO HAVLES YOU END UP WITH THIS HEXADECIMAL NOTATION: 000000FF000000

Fig. 2-10. Hexadecimal Translation of Straight Line.

Listing 2-4

```
10 REM    STRAIGHT LINE
20 TT$="**  STRAIGHT LINE  **"
30 CALL CLEAR
40 PRINT TAB(INT((29-LEN(TT$))/2));TT$
50 PRINT :::"  THIS PROGRAM DRAWS A":::"STRAIGHT LINE
   ACROSS THE":::"SCREEN."
60 PRINT ::::::::"  PRESS ANY KEY TO BEGIN"
70 CALL KEY(3,KEY,STATUS)
80 IF STATUS<=0 THEN 70
90 CALL CHAR(128,"000000FFFF000000")
100 CALL COLOR(13,16,1)
110 CALL SCREEN(2)
120 CALL CLEAR
130 FOR COL=1 TO 32
140 CALL HCHAR(12,COL,128)
150 NEXT COL
160 FOR DELAY=1 TO 1000
170 NEXT DELAY
180 CALL SCREEN(4)
190 CALL CLEAR
200 PRINT "WANT TO DRAW STRAIGHT LINE  AGAIN? (Y/N)"
210 CALL KEY(3,KEY,STATUS)
220 IF (KEY<>89)*(KEY<>78)THEN 210
230 IF KEY=89 THEN 110
240 CALL CLEAR
250 END
```

Explanation of Program

10-80	Clear screen, center title, display introduction screen
90	Define user-defined graphics character

100-120 Set color for graphics character and set screen colors
130-150 Print straight line, one column at a time
160-250 Delay after drawing line, clear screen, and ask if you want to see straight line again. If response is no, clear screen and exit program

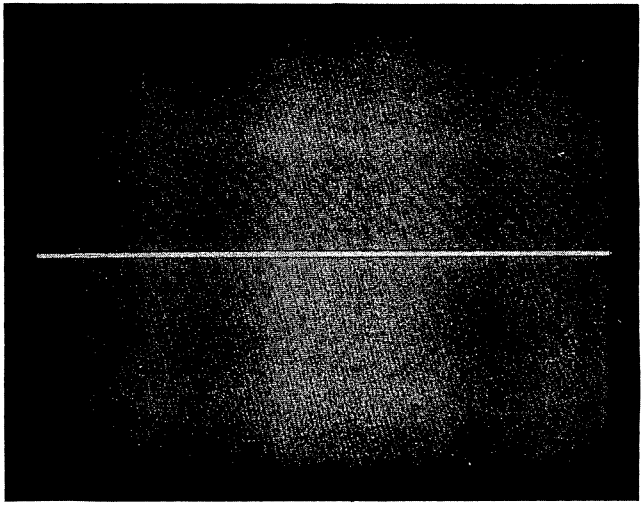


Fig. 2-11. Straight Line.

VERTICAL LINE

Listing 2-5 is similar to the last program, except this one draws a vertical line on the screen. Notice the hexadecimal number. Work out the bit-map representation as we did to produce Fig. 2-10. When you run the program, your screen will look similar to Fig. 2-12.

Listing 2-5

```
10 REM     VERTICAL LINE
20 TT$="**  VERTICAL LINE  **"
30 CALL CLEAR
40 PRINT TAB(INT((29-LEN(TT$))/2));TT$
50 PRINT :::"  THIS PROGRAM DRAWS A":::"VERTICAL LINE
   ON THE":::"SCREEN."
60 PRINT ::::::::"  PRESS ANY KEY TO BEGIN"
70 CALL KEY(3,KEY,STATUS)
80 IF STATUS<=0 THEN 70
90 CALL CHAR(128,"1818181818181818")
100 CALL COLOR(13,7,1)
110 CALL SCREEN(16)
```

```
120 CALL CLEAR
130 FOR ROW=1 TO 24
140 CALL VCHAR(ROW,16,128)
150 NEXT ROW
160 FOR DELAY=1 TO 1000
170 NEXT DELAY
180 CALL SCREEN(4)
190 CALL CLEAR
200 PRINT "WANT TO DRAW VERTICAL LINE  AGAIN? (Y/N)"
210 CALL KEY(3,KEY,STATUS)
220 IF (KEY<>89)*(KEY<>78)THEN 210
230 IF KEY=89 THEN 110
240 CALL CLEAR
250 END
```

Explanation of Program

10-80	Clear screen, center title, and display introduction screen
90	Set graphics representation for vertical line
100-150	Set color for vertical line, set screen color, and draw vertical line
160-250	Delay program while you look at vertical line, clear screen, and ask if you want to see line again. If response is no, clear screen and exit program

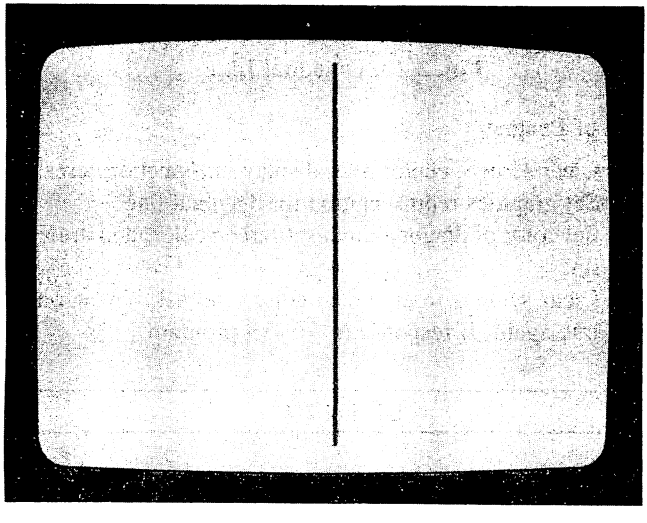


Fig. 2-12. Vertical Line.

DIAGONAL LINE

As you have seen, by coloring blocks of the bit-map, you can plan the production of many different graphics characters on your computer. (We

will discuss more detailed bit-mapping in Chapter 4.) Listing 2-6 produces another variation of the straight line. This time, the program draws a diagonal line on the screen. Notice the hexadecimal notation for the diagonal line. When you run the program, your screen will look similar to Fig. 2-13.

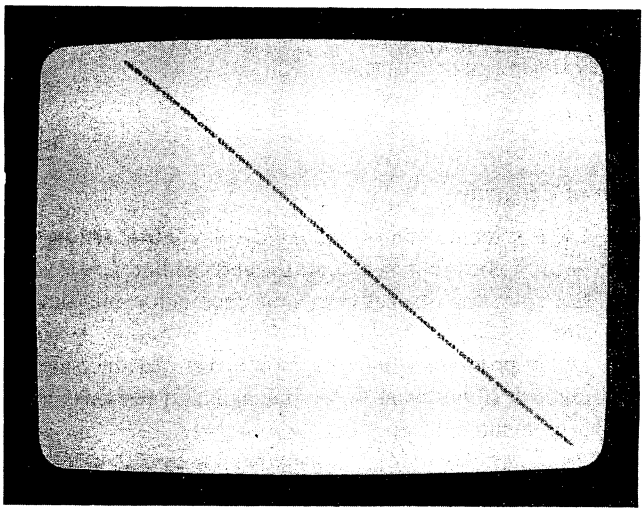


Fig. 2-13. Diagonal Line.

Explanation of Program

10-80	Clear screen, center title, display introduction screen
90	Set graphics representation for diagonal line
100-170	Set color of diagonal line, set screen color, and draw diagonal line
180-270	Clear screen, reset screen color, and ask if you want to see line again. If response is no, exit program

Listing 2-6

```
10 REM    DIAGONAL LINE
20 TT$="**  DIAGONAL LINE  **"
30 CALL CLEAR
40 PRINT TAB(INT((29-LEN(TT$))/2));TT$
50 PRINT :::"  THIS PROGRAM DRAWS A":::"DIAGONAL LINE
   ON THE":::"SCREEN."
60 PRINT :::::::"  PRESS ANY KEY TO BEGIN"
70 CALL KEY(3,KEY,STATUS)
80 IF STATUS<=0 THEN 70
```

```
90 CALL CHAR(128,"C0E070381C0E0703")
100 CALL COLOR(13,5,1)
110 CALL SCREEN(15)
120 CALL CLEAR
130 COL=4
140 FOR ROW=1 TO 24
150 CALL HCHAR(ROW,COL,128)
160 COL=COL+1
170 NEXT ROW
180 FOR DELAY=1 TO 1000
190 NEXT DELAY
200 CALL SCREEN(4)
210 CALL CLEAR
220 PRINT "DRAW DIAGONAL LINE AGAIN?   (Y/N)"
230 CALL KEY(3,KEY,STATUS)
240 IF (KEY<>89)*(KEY<>78)THEN 230
250 IF KEY=89 THEN 110
260 CALL CLEAR
270 END
```

SQUARES

Squares displays squares on random screen locations. Notice that the bit-map for a square is still simple. It requires a few more bits than the programs listed above did. The program *Squares* is given in Listing 2-7. When you run the program, your screen will look similar to Fig. 2-14.

Listing 2-7

```
10 REM   SQUARES
20 CALL CLEAR
30 TITLE$="**  SQUARES  **"
40 REM   CENTER THE TITLE
50 PRINT TAB(INT((29-LEN(TITLE$))/2));TITLE$
60 PRINT :::::"  THIS PROGRAM DEMONSTRATES HOW SQUARES
   ARE CREATED."
70 PRINT ::"  THEY ARE THEN PLACED ON   THE SCREEN IN
   RANDOM        LOCATIONS."
80 PRINT :::::"  PRESS ANY KEY TO BEGIN"
90 CALL KEY(0,KEY,STATUS)
100 IF STATUS<=0 THEN 90
110 CALL CLEAR
120 PRINT ::::::::::::::::::::::"     PRESS  X  TO
   EXIT"
130 CALL COLOR(13,2,1)
140 CALL CHAR(128,"FF8181818181FF")
150 RANDOMIZE
160 COL=INT(27*RND)+3
170 ROW=INT(18*RND)+2
180 CALL HCHAR(ROW,COL,128)
190 CALL KEY(0,KEY,STATUS)
200 IF (KEY=88)+(KEY=120)THEN 220
210 GOTO 160
220 CALL CLEAR
230 END
```

Explanation of Program

10-100	Clear screen, center title, and display introduction screen
110-130	Print way to exit program and set color of squares
140	Set character representation for square in ASCII code 128
150-210	Set randomizer routine, select random row and column numbers, and print square at that location. Check to see if X key was pressed. If so, then exit
220-230	Clear screen and exit

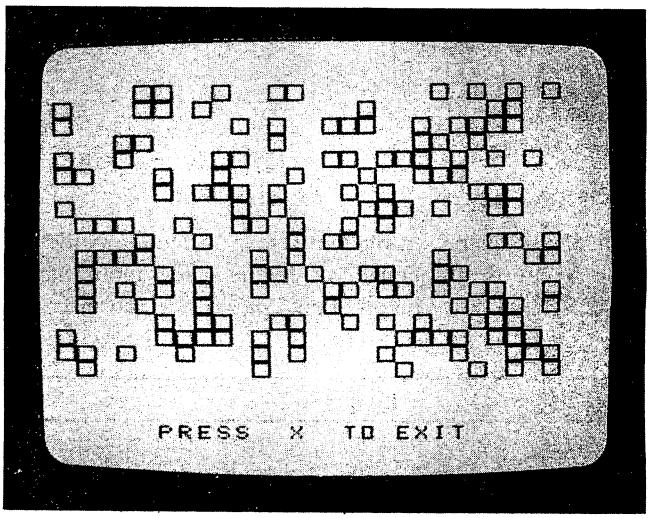


Fig. 2-14. Squares.

CIRCLES

The square was pretty easy to create. It was formed by coloring the blocks positioned along the outside of the bit-map. Generating circles is a

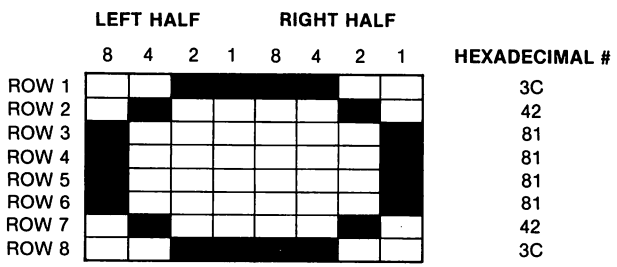


Fig. 2-15. Bit-Map and Hexadecimal Representation of a Circle.

little harder. You have to color certain blocks of the bit-map to have it look like a circle. Listing 2-8 is a program that creates a circle and then displays it randomly on the screen. See Fig. 2-15 for the bit-map and hexadecimal representation of a circle. When you run the program, your screen will look similar to Fig. 2-16.

Listing 2-8

```
10 REM  CIRCLES
20 CALL CLEAR
30 TITLE$="**  CIRCLES  **"
40 REM   CENTER THE TITLE
50 PRINT TAB(INT((29-LEN(TITLE$))/2));TITLE$
60 PRINT :::::"  THIS PROGRAM DEMONSTRATES HOW CIRCLES
   ARE CREATED."
70 PRINT ::"  THEY ARE THEN PLACED ON   THE SCREEN IN
   RANDOM        LOCATIONS."
80 PRINT :::::"  PRESS ANY KEY TO BEGIN"
90 CALL KEY(0,KEY,STATUS)
100 IF STATUS<=0 THEN 90
110 CALL CLEAR
120 CALL COLOR(5,16,1)
130 CALL COLOR(6,16,1)
140 CALL COLOR(7,16,1)
150 CALL COLOR(8,16,1)
160 CALL SCREEN(2)
170 PRINT ::::::::::::::::::::::"     PRESS  X  TO
   EXIT"
180 CALL COLOR(13,16,1)
190 CALL CHAR(128,"3C4281818181423C")
200 RANDOMIZE
210 COL=INT(27*RND)+3
220 ROW=INT(18*RND)+2
230 CALL HCHAR(ROW,COL,128)
240 CALL KEY(0,KEY,STATUS)
250 IF (KEY=88)+(KEY=120)THEN 270
260 GOTO 210
270 CALL CLEAR
280 END
```

Explanation of Program

10-100	Clear screen, center title, and display introduction screen
110-180	Set color for screen, color for characters used to display message for exiting, and color of graphics character
190	Set up and define character representation for circle
200-260	Set randomizer for retrieving random numbers. Select random row and column position for printing character. If X is pressed, then exit
270-280	Clear screen and exit program

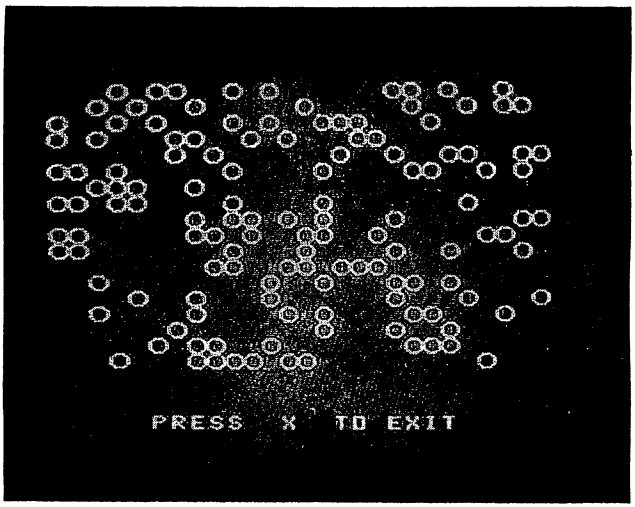


Fig. 2-16. Circles.

TRIANGLES

A triangle is no more difficult to create than a circle. Remember, you don't have to use the entire 8 x 8 bit-map. In fact, for creating a triangle, we will use the bottom half of the bit-map. This causes the triangle to be more uniform in shape. See Fig. 2-17 for the bit-map representation. The program is given in Listing 2-9. When you run the program, your screen will look similar to Fig. 2-18.

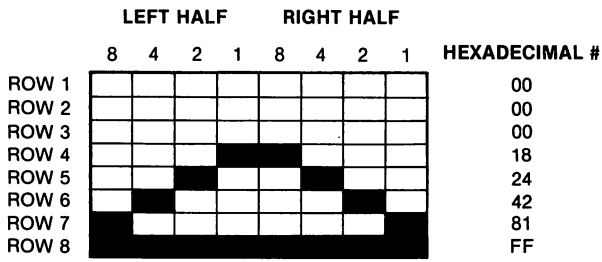


Fig. 2-17. Bit-Map for a Triangle.

Listing 2-9

```
10 REM   TRIANGLES
20 CALL CLEAR
30 TITLE$="**  TRIANGLES  **"
```

```
40 REM   CENTER THE TITLE
50 PRINT TAB(INT((29-LEN(TITLE$))/2));TITLE$
60 PRINT :::::"  THIS PROGRAM DEMONSTRATES HOW
    TRIANGLES ARE CREATED."
70 PRINT ::"  THEY ARE THEN PLACED ON   THE SCREEN IN
    RANDOM        LOCATIONS."
80 PRINT :::::"  PRESS ANY KEY TO BEGIN"
90 CALL KEY(0,KEY,STATUS)
100 IF STATUS<=0 THEN 90
110 CALL CLEAR
120 CALL COLOR(5,2,1)
130 CALL COLOR(6,2,1)
140 CALL COLOR(7,2,1)
150 CALL COLOR(8,2,1)
160 CALL SCREEN(16)
170 PRINT ::::::::::::::::::::::"     PRESS  X  TO
    EXIT"
180 CALL COLOR(13,13,1)
190 CALL CHAR(128,"0000001824428lFF")
200 RANDOMIZE
210 COL=INT(27*RND)+3
220 ROW=INT(18*RND)+2
230 CALL HCHAR(ROW,COL,128)
240 CALL KEY(3,KEY,STATUS)
250 IF (KEY=88)+(KEY=120)THEN 270
260 GOTO 210
270 CALL CLEAR
280 END
```

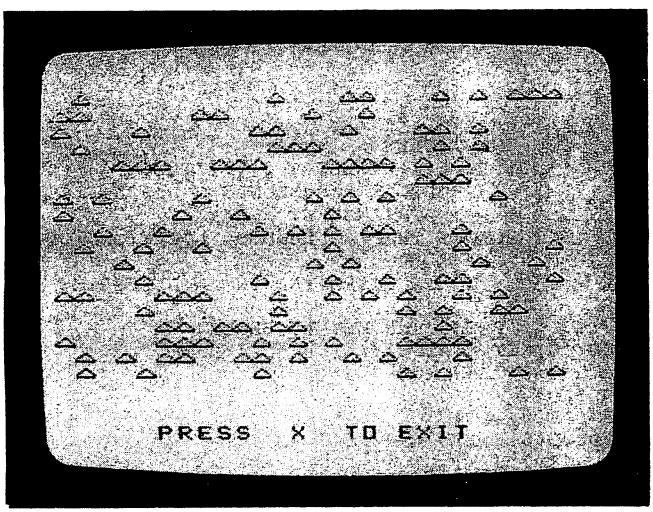


Fig. 2-18. Triangles.

Explanation of Program

10-100	Clear screen, center title, and display introduction screen
110-180	Set colors of characters for printing to screen. Set screen color and color of triangles
190	Set up and define graphics representation for triangle
200-260	Set randomizer generator. Select random row and column to display triangle. If X is pressed, then exit
270-280	Clear screen and exit program

VARIETY

By now, you should be able to develop basic graphics characters. If not, go back to the start of this section on graphics and review the examples and the sample programs more slowly. With what you have learned in this chapter, you know how to generate a single graphics character the size of a normal TI-99/4A character. It is a small character, but when you combine these small characters (squares, circles and triangles) in one program the result can be impressive.

The program in Listing 2-10 does just that. It combines the square, the circle, and the triangle and displays them on the screen in random locations. When you run the program, your screen will look similar to Fig. 2-19.

Listing 2-10

```
10 REM    VARIETY
20 CALL CLEAR
30 TITLE$="**  VARIETY  **"
40 REM   CENTER THE TITLE
50 PRINT TAB(INT((29-LEN(TITLE$))/2));TITLE$
60 PRINT :::::"  THIS PROGRAM COMBINES     SQUARES,
   CIRCLES AND        TRIANGLES.":
70 PRINT :"  IT PLACES THEM AT RANDOM  LOCATIONS ON
   THE SCREEN."
80 PRINT :"  EACH ONE IS DISPLAYED IN  A DIFFERENT
   COLOR."
90 PRINT :::::"  PRESS ANY KEY TO BEGIN"
100 CALL KEY(0,KEY,STATUS)
110 IF STATUS<=0 THEN 100
120 CALL CLEAR
130 CALL COLOR(5,2,1)
140 CALL COLOR(6,2,1)
150 CALL COLOR(7,2,1)
160 CALL COLOR(8,2,1)
170 CALL SCREEN(16)
180 PRINT ::::::::::::::::::::"     PRESS  X  TO
   EXIT"
```

```
190 CALL COLOR(14,3,1)
200 CALL CHAR(128,"0000001824428lFF")
210 CALL CHAR(136,"FF8181818181FF")
220 CALL CHAR(144,"3C4281818181423C")
230 RANDOMIZE
240 COL=INT(27*RND)+3
250 ROW=INT(18*RND)+2
260 CALL COLOR(14,5,1)
270 CALL HCHAR(ROW,COL,136)
280 CALL KEY(0,KEY,STATUS)
290 IF (KEY=88)+(KEY=120)THEN 430
300 COL=INT(27*RND)+3
310 ROW=INT(18*RND)+2
320 CALL COLOR(15,7,1)
330 CALL HCHAR(ROW,COL,144)
340 CALL KEY(0,KEY,STATUS)
350 IF (KEY=88)+(KEY=120)THEN 430
360 ROW=INT(18*RND)+2
370 COL=INT(27*RND)+3
380 CALL COLOR(13,13,1)
390 CALL HCHAR(ROW,COL,128)
400 CALL KEY(0,KEY,STATUS)
410 IF (KEY=88)+(KEY=120)THEN 430
420 GOTO 240
430 CALL CLEAR
440 END
```

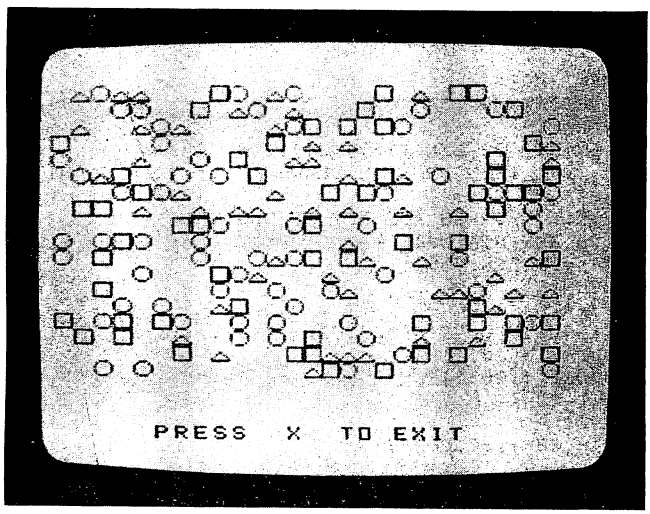


Fig. 2-19. Variety.

Explanation of Program

10-110	Clear screen, center title, and display introduction screen
120-190	Set color of characters that are in message to exit program. Set screen color

200	Set up and define triangle
210	Set up and define square
220	Set up and define circle
230-290	Display square at random location on screen
300-350	Display circle at random location on screen
360-420	Display triangle at random location on screen
430-440	Clear screen and exit program

GIANT HI

This program is an example that demonstrates combining bit-maps to generate larger graphics characters. Using 14 different bit-maps placed together on the screen, the program displays a large "HI" on your computer's screen. I hope this program will encourage you to experiment with creating larger and larger graphics characters. (You will learn more about these advanced graphics techniques in Chapter 4.) The program is given in Listing 2-11. When you run the program, your screen will look similar to Fig. 2-20.

Listing 2-11

```
10 REM   GIANT HI
20 TT$="**  GIANT HI  **"
30 CALL CLEAR
40 PRINT TAB(INT((29-LEN(TT$))/2));TT$
50 PRINT :::::"  THIS PROGRAM DRAWS A"::::"GIANT 'HI'
   ON THE SCREEN.":::
60 PRINT ::::::"  PRESS ANY KEY TO BEGIN"
70 CALL KEY(3,KEY,STATUS)
80 IF STATUS<=0 THEN 70
90 CALL SCREEN(5)
100 CALL CLEAR
110 PRINT "  SETTING UP GRAPHICS...":::::::::::
120 CALL COLOR(13,16,1)
130 CALL COLOR(14,16,1)
140 FOR CHR=1 TO 14
150 READ CODE$
160 CALL CHAR(CHR+127,CODE$)
170 NEXT CHR
180 CALL CLEAR
190 REM  DRAWS TOP OF 'H'
200 FOR ROW=6 TO 8
210 CALL HCHAR(ROW,12,128)
220 CALL HCHAR(ROW,15,129)
230 NEXT ROW
240 REM   DRAWS LINE ACROSS 'H'
250 CALL HCHAR(9,12,136)
260 CALL HCHAR(9,13,130)
270 CALL HCHAR(9,14,130)
```

```
280 CALL HCHAR(9,15,137)
290 REM  DRAWS BOTTOM OF 'H'
300 FOR ROW=10 TO 12
310 CALL HCHAR(ROW,12,128)
320 CALL HCHAR(ROW,15,129)
330 NEXT ROW
340 REM  DRAWS TOP OF 'I'
350 FOR COL=18 TO 21
360 CALL HCHAR(6,COL,COL+112)
370 NEXT COL
380 REM  DRAWS LINE DOWN 'I'
390 FOR ROW=7 TO 11
400 CALL HCHAR(ROW,19,134)
410 CALL HCHAR(ROW,20,135)
420 NEXT ROW
430 REM  DRAWS BOTTOM OF 'I'
440 FOR COL=18 TO 21
450 CALL HCHAR(12,COL,COL+120)
460 NEXT COL
470 PRINT "     PRESS  X  TO EXIT"
480 CALL KEY(3,KEY,STATUS)
490 IF KEY<>88 THEN 480
500 CALL CLEAR
510 END
520 REM  DATA FOR BUILDING THE LETTERS 'H' & 'I'
530 DATA C0C0C0C0C0C0C0C0
540 DATA 0303030303030303
550 DATA FFFF0000000000000
560 DATA FFFF0101010101010
570 DATA FFFF8080808080808
580 DATA FFFF0000000000000
590 DATA 0101010101010101
600 DATA 8080808080808080
610 DATA FFFFC0C0C0C0C0C0
620 DATA FFFF0303030303030
630 DATA 000000000000FFFF
640 DATA 0101010101010FFFF
650 DATA 808080808080FFFF
660 DATA 000000000000FFFF
```

Explanation of Program

10-80	Clear screen, center title, and display introduction screen
90-180	Set up and define necessary graphics characters and color associated with each one
190-230	Draw top section of "H"
240-280	Draw middle line of "H"
290-330	Draw bottom of "H"
340-370	Draw top of "I"
380-420	Draw line down "I"
430-460	Draw bottom part of "I"
470-510	Wait for X key to be pressed, clear screen, and exit program
520-660	Generate graphics characters for each part of "H" and "I"

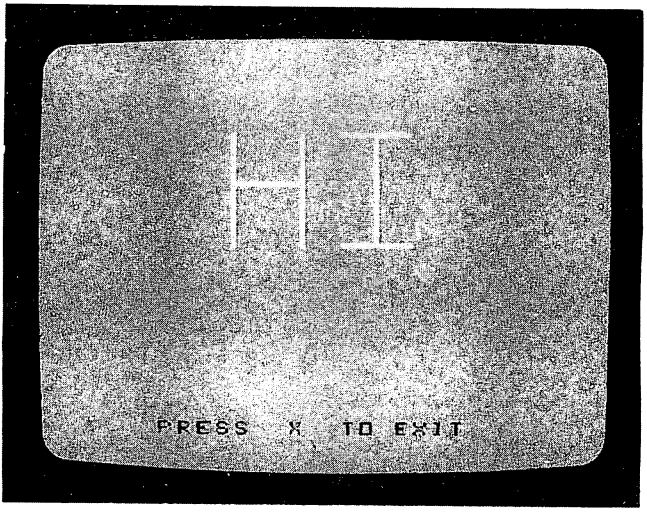


Fig. 2-20. Giant HI.

SUMMARY

At this point, you should have a very good understanding of the different colors and how they can affect the screen, the foreground, and the background characters. These small graphics characters may not seem powerful to you. Still, they are very useful in learning the more advanced graphics techniques.

Chapter 3

Special Effects with Sounds

The TI-99/4A is capable of producing music as well as noises and sounds. The reason for this is the versatility of the TI-99/4A's programming capabilities. To help you understand the sounds and music power of the TI-99/4A, though, here are a few definitions.

Sound/Music: for the purposes of this book, "sound" is defined as audio which is not music. For instance, the simulated roar of a spaceship blasting off would be a sound, while an electronic version of the Bach Minuet would be music.

Voice: a voice on the TI-99/4A refers to an individual oscillator within the computer. Each of the TI-99/4A's oscillators can produce a sound of its own, so that three different notes or sounds may be produced simultaneously. Think of *multiple voices* as *multiple musical instruments* since each voice can have characteristics all its own.

Frequency: the frequency of a tone is how high or low it sounds. The higher the frequency number, the higher the sound. A very low frequency number would designate a bass note, while a high frequency might resemble the high-pitched squeak of a mouse.

PROGRAMMING SOUNDS

The command to generate sound or music on the TI-99/4A is **CALL SOUND(duration, frequency1, volume1[, frequency2, volume2],**

[frequency3, volume3][, frequency4, volume4]). The parameters are self-explanatory. For the time being, concern yourself only with the first part of the command: **duration, frequency1,** and **volume1**.

How long a sound or musical note is played is called the *duration*. Duration can be any numeric value from 1 to 4250 or -1 to -4250, where 1 is equal to 1 millisecond and 4250 is equivalent to 4.25 seconds. If you use a negative duration, the computer produces the next tone as soon as it is encountered without waiting for the previous note or sound to end.

Frequency is defined at the beginning of this chapter. **Frequency1** can be any value from 110 through 44733. (See Appendix E for a complete listing of the frequencies and their corresponding musical notes.)

Volume1 can be any number 0 through 30, where 0 is the loudest and 30 is the quietest.

When programming a musical piece for the TI-99/4A, the data for the notes must first be entered. Usually the data is entered into DATA statements and then READ into the CALL SOUND routine one at a time. This method, while it is the most efficient, need not concern you while you read this chapter. You will use DATA statements in Chapter 5.

The TI-99/4A is capable of generating two kinds of noises. The first is *periodic noises* and the second is *white noises*.

Frequency is the only parameter that differentiates between periodic and white noises. (See *Random Periodic Noises* and *White Noises* below for the correct parameters.)

RANDOM PERIODIC NOISES

The program shown in Listing 3-1 will cause your computer to produce random periodic noises. Periodic noises are generated by frequencies from -1 through -4.

Listing 3-1

```
10 REM      RANDOM PERIODIC NOISES
20 TT$="**  PERIODIC NOISES  **"
30 CALL CLEAR
40 PRINT TAB(INT((29-LEN(TT$))/2));TT$
50 PRINT :::"  THIS PROGRAM GENERATES"::::"RANDOM
   PERIODIC NOISES."
60 PRINT :::::::"  PRESS ANY KEY TO BEGIN"
70 CALL KEY(0,KEY,STATUS)
80 IF STATUS<=0 THEN 70
90 CALL CLEAR
100 PRINT " PLAYING NOISES...":::::::::::
```

```
110 PRINT "    PRESS  X  TO EXIT"
120 NOISE=INT(RND*4)+1
130 NOISE=-NOISE
140 CALL SOUND(150,NOISE,0)
150 CALL KEY(3,KEY,STATUS)
160 IF KEY=88 THEN 180
170 GOTO 120
180 CALL CLEAR
190 END
```

Explanation of Program

10-90	Clear screen, center title, and display introductory screen
100-110	Display way to exit program
120	Select random noise from 1 to 4
130	Change positive value to negative one to generate noise
140	Play noise
150-190	Wait for key to be pressed to exit. If X is pressed, clear screen and exit. Otherwise, play another random noise

RANDOM WHITE NOISES

Listing 3-2 produces random white noises. White noises are generated with a frequency range from -5 through -8.

Listing 3-2

```
10 REM      RANDOM WHITE NOISES
20 TT$="**  WHITE NOISES  **"
30 CALL CLEAR
40 PRINT TAB(INT((29-LEN(TT$))/2));TT$
50 PRINT :::"  THIS PROGRAM GENERATES":::::"RANDOM
   WHITE NOISES."
60 PRINT ::::::::"  PRESS ANY KEY TO BEGIN"
70 CALL KEY(0,KEY,STATUS)
80 IF STATUS<=0 THEN 70
90 CALL CLEAR
100 PRINT " PLAYING NOISES...":::::::::::
110 PRINT "    PRESS  X  TO EXIT"
120 NOISE=INT(RND*4)+5
130 NOISE=-NOISE
140 CALL SOUND(150,NOISE,0)
150 CALL KEY(3,KEY,STATUS)
160 IF KEY=88 THEN 180
170 GOTO 120
180 CALL CLEAR
190 END
```

Explanation of Program

10-90	Clear screen, center title, and display introduction screen
100-110	Display notice that noises are playing and how to exit program

120	Select random noise from 5 through 8
130	Turn positive into negative number for use in frequency
140	Play noise
150-190	Check to see if you want to exit. If so, clear screen and exit. If not, play another random noise

UPWARD SOUND

Program Listing 3-3 uses a climbing frequency to produce a rapid upward sound. Notice the negative duration number. This causes the next frequency to be played as soon as it reaches the CALL SOUND statement, thus producing an upward sound.

Listing 3-3

```
10 REM  UPWARD SOUND
20 TT$="**  UPWARD SOUND  **"
30 CALL CLEAR
40 PRINT TAB(INT((29-LEN(TT$))/2));TT$::::
50 PRINT "  THIS PROGRAM GENERATES AN"::"UPWARD
    SOUND.":::::::::
60 PRINT "  PRESS ANY KEY TO BEGIN"
70 CALL KEY(0,KEY,STATUS)
80 IF STATUS<=0 THEN 70
90 CALL CLEAR
100 PRINT "   PLAYING UPWARD SOUND...":::::::::::::
110 FOR DELAY=1 TO 100
120 NEXT DELAY
130 FOR FREQ=220 TO 1662 STEP 10
140 CALL SOUND(-900,FREQ,0)
150 NEXT FREQ
160 CALL SOUND(1,FREQ+10,0)
170 CALL CLEAR
180 PRINT "PLAY UPWARD SOUND           AGAIN? (Y/N)"
190 CALL KEY(0,KEY,STATUS)
200 IF (KEY<>89)*(KEY<>121)*(KEY<>78)*(KEY<>110)THEN
    190
210 IF (KEY=89)+(KEY=121)THEN 90
220 CALL CLEAR
230 END
```

Explanation of Program

10-90	Clear screen, center title, and display introduction screen
100-120	Display notice that upward sound is playing. Delay to let you get ready
130-150	Play upward sound
160	Turn off sound
170-230	Ask if you want to hear sound again. If so, play it again. If not, clear screen and exit program

DOWNWARD SOUND

By changing a few lines, a rapid downward sound can be produced. The program is shown in Listing 3-4.

Listing 3-4

```
10 REM   DOWNWARD SOUND
20 TT$="**  DOWNWARD SOUND  **"
30 CALL CLEAR
40 PRINT TAB(INT((29-LEN(TT$))/2));TT$::::
50 PRINT "  THIS PROGRAM GENERATES A"::"DOWNWARD
   SOUND.":::::::::
60 PRINT "  PRESS ANY KEY TO BEGIN"
70 CALL KEY(0,KEY,STATUS)
80 IF STATUS<=0 THEN 70
90 CALL CLEAR
100 PRINT " PLAYING DOWNWARD SOUND...":::::::::::::
110 FOR DELAY=1 TO 100
120 NEXT DELAY
130 FOR FREQ=1662 TO 220 STEP -10
140 CALL SOUND(-400,FREQ,0)
150 NEXT FREQ
160 CALL SOUND(1,FREQ-10,0)
170 CALL CLEAR
180 PRINT "PLAY DOWNWARD SOUND          AGAIN? (Y/N)"
190 CALL KEY(0,KEY,STATUS)
200 IF (KEY<>89)*(KEY<>121)*(KEY<>78)*(KEY<>110)THEN
    190
210 IF (KEY=89)+(KEY=121)THEN 90
220 CALL CLEAR
230 END
```

Explanation of Program

10-90	Clear screen, center title, and display introduction screen
100-120	Display notice to screen that sound is being played. Delay program to let you get ready
130-170	Play downward sound and, when done, shut sound off abruptly
180-230	Clear screen and ask if you want to hear downward sound again. If so, play it again. If not, exit program

UP AND DOWN SOUND

By combining the previous two programs, a rapid upward and downward sound can be produced. See Listing 3-5.

Listing 3-5

```
10 REM    UP & DOWN SOUND
20 TT$="**  UP & DOWN SOUND  **"
30 CALL CLEAR
40 PRINT TAB(INT((29-LEN(TT$))/2));TT$::::
50 PRINT "  THIS PROGRAM GENERATES A"::"SOUND THAT
   RAPIDLY GOES UP"::"AND DOWN."
60 PRINT ::::::"  PRESS ANY KEY TO BEGIN"
70 CALL KEY(0,KEY,STATUS)
80 IF STATUS<=0 THEN 70
90 CALL CLEAR
100 PRINT "PLAYING UP & DOWN SOUND..."::::::::::::
110 PRINT "   PRESS  X  TO EXIT"
120 FOR DELAY=1 TO 100
130 NEXT DELAY
140 FOR FREQ=440 TO 660 STEP 10
150 CALL KEY(0,KEY,STATUS)
160 IF (KEY=88)+(KEY=120)THEN 250
170 CALL SOUND(-800,FREQ,0)
180 NEXT FREQ
190 FOR FREQ=660 TO 440 STEP -10
200 CALL KEY(0,KEY,STATUS)
210 IF (KEY=88)+(KEY=120)THEN 250
220 CALL SOUND(-800,FREQ,0)
230 NEXT FREQ
240 GOTO 140
250 CALL CLEAR
260 CALL SOUND(-1,110,30)
270 END
```

Explanation of Program

10-90	Clear screen, center title, and display introduction screen
100-130	Display notice that sound is being played and a notice of how to exit program
140-180	Play upward sound and check if X is pressed to exit program
190-240	Play the downward sound and check if X is pressed to exit program
250-270	Clear screen and exit program

MUSICAL SCALE

The TI-99/4A can produce songs as well as sounds and noises. Listing 3-6 causes your computer to produce a musical scale. So you can follow along, the screen changes color as each note is sounded.

Listing 3-6

```
10 REM   MUSICAL SCALE
20 TT$="**  MUSICAL SCALE  **"
30 CALL CLEAR
40 PRINT TAB(INT((29-LEN(TT$))/2));TT$
50 PRINT :::"  THIS PROGRAM PLAYS A"::
60 PRINT "MUSICAL SCALE.":::::::::
70 PRINT "  PRESS ANY KEY TO BEGIN"
80 CALL KEY(0,KEY,STATUS)
90 IF STATUS<=0 THEN 80
100 RANDOMIZE
110 CALL CLEAR
120 RESTORE
130 PRINT "   PLAYING SCALES...":::::::::::::
140 FOR NUMNOTES=1 TO 14
150 READ NOTE
160 COLR=INT(RND*15)+1
170 CALL SCREEN(COLR)
180 CALL SOUND(300,NOTE,4)
190 NEXT NUMNOTES
200 FOR DELAY=1 TO 200
210 NEXT DELAY
220 CALL CLEAR
230 CALL SCREEN(4)
240 PRINT "PLAY MUSICAL SCALES          AGAIN? (Y/N)"
250 CALL KEY(0,KEY,STATUS)
260 IF (KEY<>89)*(KEY<>121)*(KEY<>78)*(KEY<>110)THEN
    250
270 IF (KEY=89)+(KEY=121)THEN 110
280 CALL CLEAR
290 END
300 DATA 220,246,261,293,329,349,392
310 DATA 392,349,329,293,261,246,220
```

Explanation of Program

10-90	Clear screen, center title, and display introduction screen
100-130	Start randomizer generator and display notice that music is being played
140-190	Select random screen color to display on screen as note is sounded. Change screen color. Play note.
200-290	Clear screen. Reset screen to default color. Ask whether you would like to hear scales again. If not, clear screen and exit program
300-310	Play musical scale

SUMMARY

The definitions and program listings in this chapter provide basic information about the noise, sound, and music features of the TI-99/4A. Chapter 5 explores more advanced programming possibilities.

Chapter 4

Advanced Graphics Techniques and Animation

To make use of the more elaborate graphics effects on the TI-99/4A, you will need to expand on what you learned from studying Chapter 2. In that chapter, you learned the basic ingredient in creating a graphics character is the 8 x 8 bit-map. You also had a sneak preview, in the Giant HI program, of how larger graphics characters are formed. In this chapter, you will learn how to combine several bit-maps to form a larger graphics character.

There are a couple of techniques used in combining bit-maps. First, design large graphics characters on several adjacent 8 x 8 bit-maps. When you have drawn the design, all you will need to do is break the bigger bit-map into several smaller 8 x 8 bit-maps. This is similar to taking apart a jigsaw puzzle one piece at a time. Use the techniques you learned in Chapter 2 to calculate the hexadecimal notations for each 8 x 8 bit-map.

The second technique is very important: remember how you took apart the big bit-map so you'll be able to piece the puzzle back together.

It is a good idea to order the hexadecimal notations by sequential ASCII codes. (See Appendix B for the list of ASCII codes available on the TI-99/4A.) This will make it easier to remember where the bit-maps were stored in your computer. It also helps when you are displaying the puzzle's

pieces because you'll know in what order to display them. To display the characters, use the CALL HCHAR or CALL VCHAR and place the pieces side by side or on top of one another. If you forget where the hexadecimal notation for a particular bit-map goes, and you place it in the wrong row or column, some very strange results may be displayed.

RECTANGLE

The program in Listing 4-1 draws a big rectangle. The hexadecimal notation for the rectangle is given in lines 360-440. The large rectangle is drawn starting with the upper left corner. The computer draws each subsequent bit-map block beside the first. In other words, the program draws the rectangle horizontally. When you run the program, your screen will look similar to Fig. 4-1.

Listing 4-1

```
10 REM     RECTANGLE
20 TT$="**  RECTANGLE  **"
30 CALL CLEAR
40 PRINT TAB(INT((29-LEN(TT$))/2));TT$
50 PRINT :::"  THIS PROGRAM DRAWS A":::"RECTANGLE ON
    THE SCREEN."
60 PRINT :::::::"  PRESS ANY KEY TO BEGIN"
70 CALL KEY(3,KEY,STATUS)
80 IF STATUS<=0 THEN 70
90 CALL CLEAR
100 PRINT " SETTING UP GRAPHICS...":::::::::::
110 CALL COLOR(13,2,1)
120 FOR I=1 TO 8
130 READ CODE$
140 CALL CHAR(127+I,CODE$)
150 NEXT I
160 CALL SCREEN(16)
170 CALL CLEAR
180 CALL HCHAR(8,12,128)
190 CALL HCHAR(8,13,129,6)
200 CALL HCHAR(8,19,130)
210 CALL VCHAR(9,12,131,4)
220 CALL VCHAR(9,19,132,4)
230 CALL HCHAR(13,12,133)
240 CALL HCHAR(13,13,134,6)
250 CALL HCHAR(13,19,135)
260 FOR DELAY=1 TO 1000
270 NEXT DELAY
280 CALL SCREEN(4)
290 CALL CLEAR
300 PRINT "DRAW RECTANGLE AGAIN? (Y/N)"
310 CALL KEY(3,KEY,STATUS)
320 IF (KEY<>89)*(KEY<>78)THEN 310
330 IF KEY=89 THEN 160
```

```
340 CALL CLEAR
350 END
360 REM  DATA FOR THE RECTANGLE
370 DATA FF80808080808080
380 DATA FF0000000000000
390 DATA FF0101010101010101
400 DATA 8080808080808080
410 DATA 0101010101010101
420 DATA 80808080808080FF
430 DATA 00000000000000FF
440 DATA 0101010101010101FF
```

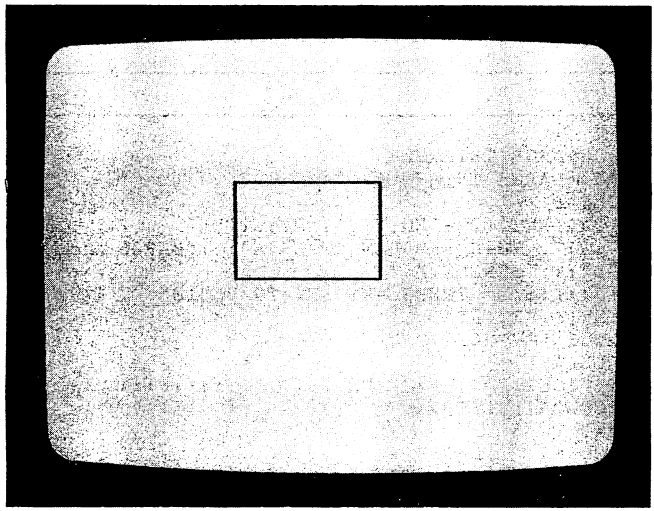


Fig. 4-1. Rectangle.

Explanation of Program

10-80	Clear screen, center title, and display introduction screen
90-170	Clear screen, set color for rectangle. (Notice that I set the color for only one of the character sets. That's because each character set contains eight ASCII codes. By setting the color for one set, I set the color for all eight ASCII code characters.) Read and store hexadecimal notation for each section of rectangle in ASCII codes 128-135
180-200	Draw top section of rectangle horizontally
210-220	Draw sides of rectangle vertically
230-250	Draw bottom section of rectangle horizontally
260-350	Delays so you can see rectangle. Ask if you want to see rectangle again. If not, exit program

360-440 Data for individual bit-maps that make up rectangle put in order to redraw rectangle

LARGE TRIANGLE

The program shown in Listing 4-2 draws a large triangle. Notice that only three graphics elements were used to draw the triangle. That's because the same elements were used more than once. In this way, the program uses less of the computer's memory to draw the picture. When you run the program, your screen will look similar to Fig. 4-2.

Listing 4-2

```
10 REM     LARGE TRIANGLE
20 TT$="**  LARGE TRIANGLE  **"
30 CALL CLEAR
40 PRINT TAB(INT((29-LEN(TT$))/2));TT$
50 PRINT :::"  THIS PROGRAM DRAWS A":::"LARGE TRIANGLE
    ON THE":::"SCREEN."
60 PRINT :::::::"  PRESS ANY KEY TO BEGIN"
70 CALL KEY(3,KEY,STATUS)
80 IF STATUS<=0 THEN 70
90 CALL CLEAR
100 PRINT " SETTING UP GRAPHICS...":::::::::::
110 CALL COLOR(13,16,1)
120 FOR I=1 TO 3
130 READ CODE$
140 CALL CHAR(127+I,CODE$)
150 NEXT I
160 CALL SCREEN(2)
170 CALL CLEAR
180 CALL HCHAR(8,14,128)
190 CALL HCHAR(8,15,129)
200 CALL HCHAR(9,13,128)
210 CALL HCHAR(9,16,129)
220 CALL HCHAR(10,12,128)
230 CALL HCHAR(10,17,129)
240 CALL HCHAR(11,11,128)
250 CALL HCHAR(11,18,129)
260 CALL HCHAR(12,10,128)
270 CALL HCHAR(12,19,129)
280 CALL HCHAR(13,10,130,10)
290 FOR DELAY=1 TO 1000
300 NEXT DELAY
310 CALL SCREEN(4)
320 CALL CLEAR
330 PRINT "DRAW TRIANGLE AGAIN? (Y/N)"
340 CALL KEY(3,KEY,STATUS)
350 IF (KEY<>89)*(KEY<>78)THEN 340
360 IF KEY=89 THEN 160
370 CALL CLEAR
380 END
```

```
390 REM   DATA FOR THE TRIANGLE
400 DATA 0102040810204080
410 DATA 8040201008040201
420 DATA FF0000000000000
```

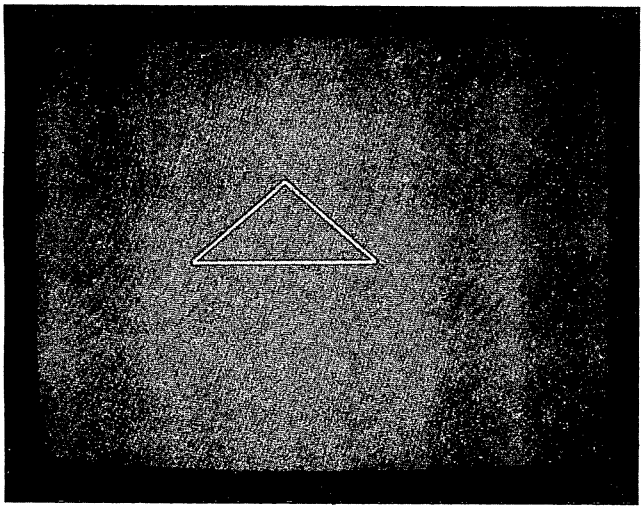


Fig. 4-2. Large Triangle.

Explanation of Program

10-80	Clear screen, center title, and display introduction screen
90-170	Set color for triangle. Define each element needed to draw triangle
180-190	Draw top of triangle (/\)
200-210	Draw next section (/ \)
220-230	Use same elements as line 200-210 but display them farther apart (/ \)
240-270	Use same elements as line 200-230 but display them farther apart (/ \) (/ \)
280	Draw bottom line of triangle (_____)
290-380	Delay program so you can see triangle. Ask if you want to see triangle again. If not, exit program
390-420	Hexadecimal notations for triangle

CUBE

We have created one rectangle. Now, we can make it smaller, then create a second smaller rectangle and place it slightly above and to the right of

the original, connecting the two figures with diagonal lines. In this manner, we can create a three-dimensional object—the cube. The program is shown in Listing 4-3. (This program isn't as efficient as it can be. Instead of using one piece many times, I used each piece once. Thus, the program has some duplicate hexadecimal definitions. It was designed this way so you could study the components of a three-dimensional object design.) When you run the program, your screen will look similar to Fig. 4-3.

Listing 4-3

```
10 REM  CUBE
20 TT$="**  CUBE  **"
30 CALL CLEAR
40 PRINT TAB(INT((29-LEN(TT$))/2));TT$
50 PRINT :::"  THIS PROGRAM DRAWS A THREE";
60 PRINT ::"DIMENSIONAL CUBE.":::
70 PRINT ::::"  PRESS ANY KEY TO BEGIN"
80 CALL KEY(0,KEY,STATUS)
90 IF STATUS<=0 THEN 80
100 CALL SCREEN(16)
110 CALL CLEAR
120 PRINT "  PREPARING THE CUBE...":::::::::::
130 FOR CH=0 TO 34
140 READ CD$
150 CALL CHAR(CH+120,CD$)
160 NEXT CH
170 FOR CHARSET=12 TO 16
180 CALL COLOR(CHARSET,2,1)
190 NEXT CHARSET
200 CALL CLEAR
210 ROW=8
220 FOR COL=14 TO 18
230 CALL HCHAR(ROW,COL,106+COL)
240 NEXT COL
250 CALL HCHAR(ROW+1,13,125)
260 CALL HCHAR(ROW+1,14,126)
270 CALL HCHAR(ROW+1,17,127)
280 CALL HCHAR(ROW+1,18,128)
290 CALL HCHAR(ROW+2,12,129)
300 CALL HCHAR(ROW+2,13,130)
310 CALL HCHAR(ROW+2,14,131)
320 CALL HCHAR(ROW+2,16,132)
330 CALL HCHAR(ROW+2,18,133)
340 CALL HCHAR(ROW+3,12,134)
350 CALL HCHAR(ROW+3,13,135)
360 CALL HCHAR(ROW+3,14,136)
370 CALL HCHAR(ROW+3,15,137)
380 CALL HCHAR(ROW+3,18,138)
390 CALL HCHAR(ROW+4,12,139)
400 CALL HCHAR(ROW+4,14,140)
410 CALL HCHAR(ROW+4,15,141)
420 CALL HCHAR(ROW+4,16,142)
430 CALL HCHAR(ROW+4,17,143)
440 CALL HCHAR(ROW+4,18,144)
```

```
450 CALL HCHAR(ROW+5,12,145)
460 CALL HCHAR(ROW+5,13,146)
470 CALL HCHAR(ROW+5,15,147)
480 CALL HCHAR(ROW+5,16,148)
490 CALL HCHAR(ROW+5,17,149)
500 CALL HCHAR(ROW+6,12,150)
510 CALL HCHAR(ROW+6,13,151)
520 CALL HCHAR(ROW+6,14,152)
530 CALL HCHAR(ROW+6,15,153)
540 CALL HCHAR(ROW+6,16,154)
550 PRINT "    PRESS  X  TO EXIT"
560 CALL KEY(0,KEY,STATUS)
570 IF (KEY<>88)*(KEY<>120)THEN 560
580 CALL CLEAR
590 END
600 DATA 000000000000007F
610 DATA 00000000000000FF
620 DATA 00000000000000FF
630 DATA 00000000000000FF
640 DATA 0000000000000080
650 DATA 0001020408102040
660 DATA C040404040404040
670 DATA 0102040810204080
680 DATA 8080808080808080
690 DATA 0001020408102040
700 DATA 8000000000000000
710 DATA 4040404040404040
720 DATA 0102040810204080
730 DATA 8080808080808080
740 DATA FF80808080808080
750 DATA FF0000000000000
760 DATA FF4040404040404040
770 DATA FF0101010101010101
780 DATA 8080808080808080
790 DATA 8080808080808080
800 DATA 4040404040407F80
810 DATA 010101010101FF01
820 DATA 000000000000FF00
830 DATA 000000000000FF01
840 DATA 8080808080808000
850 DATA 8080808080808080
860 DATA 0102040810204080
870 DATA 0101010101010101
880 DATA 0000000000000001
890 DATA 0204081020408000
900 DATA 8182848890A0C0FF
910 DATA 00000000000000FF
920 DATA 00000000000000FF
930 DATA 010101010101FF
940 DATA 0204081020408000
```

Explanation of Program

10-90	Clear screen, center title, and display introduction screen
100-200	Set screen color. Define characters. Set color for characters
210-240	Draw top row of cube

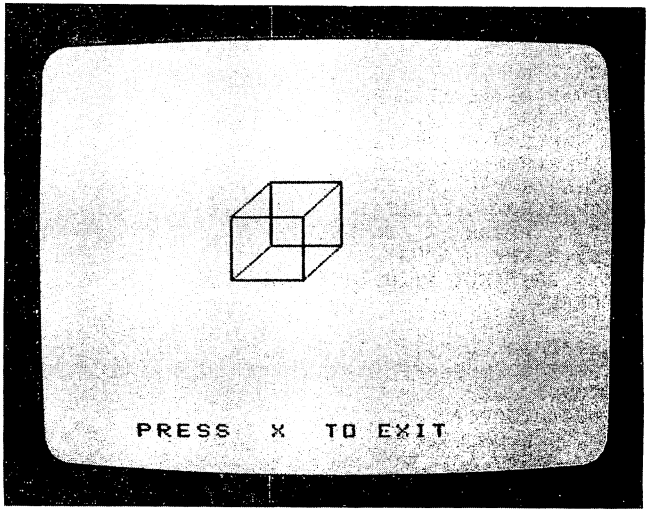


Fig. 4-3. Cube.

250-280	Draw second-from-top row
290-330	Draw third-from-top row
340-380	Draw another row
390-440	Draw another row
450-490	Draw another row
500-540	Draw last row
550-590	Wait for you to exit by pressing X key, then exit program
600-940	Characters needed to complete cube

EXPANDING FLYING SAUCERS

You now have the basic knowledge to create simple and complex graphics characters. Every character the programs have created so far was stationary on the screen. The program shown in Listing 4-4 displays three flying saucers; it also designates keys you can use to manually expand and contract the flying saucers. You will be able to expand them vertically, horizontally, or in both directions. The flying saucers are displayed in different colors when they are expanded; they return to their original color when they contract to their normal size.

In order to expand and contract graphics characters, you will need to design them twice: as they will look when they're expanded, and as they look when contracted. Place these hexadecimal notations in different locations. When you need to expand or contract the flying saucers, use

these designs. When you run the program, your screen will look similar to Fig. 4-4.

Listing 4-4

```
10 REM   EXPANDING FLYING SAUCERS
20 TT$="**EXPANDING FLYING SAUCERS**"
30 CALL CLEAR
40 PRINT TT$
50 PRINT :::"  THIS PROGRAM ALLOWS THE":::"USER TO
    EXPAND THREE FLYING":::"SAUCERS BY USING THE"
60 PRINT ::"FUNCTION KEY."
70 PRINT ::::"  PRESS ANY KEY TO BEGIN"
80 CALL KEY(3,KEY,STATUS)
90 IF STATUS<=0 THEN 80
100 CALL CLEAR
110 VEXP=-1
120 HEXP=-1
130 PRINT " SETTING UP GRAPHICS...":::::::::::
140 FOR CHSET=120 TO 122
150 READ CODE$
160 CALL CHAR(CHSET,CODE$)
170 NEXT CHSET
180 FOR CHSET=128 TO 132
190 READ CODE$
200 CALL CHAR(CHSET,CODE$)
210 NEXT CHSET
220 FOR CHSET=136 TO 141
230 READ CODE$
240 CALL CHAR(CHSET,CODE$)
250 NEXT CHSET
260 FOR CHSET=144 TO 153
270 READ CODE$
280 CALL CHAR(CHSET,CODE$)
290 NEXT CHSET
300 CALL COLOR(12,2,1)
310 CALL COLOR(13,5,1)
320 CALL COLOR(14,7,1)
330 CALL COLOR(15,13,1)
340 CALL COLOR(16,13,1)
350 CALL SCREEN(16)
360 CALL CLEAR
370 CALL HCHAR(5,8,120)
380 CALL HCHAR(10,24,121)
390 CALL HCHAR(14,11,122)
400 GOTO 1430
410 FOR I=1 TO LEN(MSG$)
420 CALL HCHAR(ROW,COL+I,ASC(SEG$(MSG$,I,1)))
430 NEXT I
440 RETURN
450 REM  EXPANDS/CONTRACTS SAUCERS VERTICALLY
460 IF (VEXP=1)*(HEXP=1)THEN 590
470 IF (VEXP=1)*(HEXP=-1)THEN 700
480 IF (VEXP=-1)*(HEXP=1)THEN 810
490 CALL HCHAR(5,8,128)
```

cont. on next page

Listing 4-4—cont.

```
500 CALL HCHAR(5,9,32)
510 CALL HCHAR(10,24,129)
520 CALL HCHAR(11,24,130)
530 CALL VCHAR(10,25,32,2)
540 CALL HCHAR(14,11,131)
550 CALL HCHAR(15,11,132)
560 CALL VCHAR(14,12,32,2)
570 VEXP=-VEXP
580 RETURN
590 REM  CONTRACT SAUCERS THAT ARE ENLARGED IN BOTH
    DIRECTION
600 CALL HCHAR(5,8,136)
610 CALL HCHAR(5,9,137)
620 CALL HCHAR(10,24,138)
630 CALL HCHAR(10,25,139)
640 CALL HCHAR(11,24,32,2)
650 CALL HCHAR(14,11,140)
660 CALL HCHAR(14,12,141)
670 CALL HCHAR(15,11,32,2)
680 VEXP=-VEXP
690 RETURN
700 REM  CONTRACT SAUCERS BACK TO NORMAL SIZE
710 CALL HCHAR(5,8,120)
720 CALL HCHAR(5,9,32)
730 CALL HCHAR(10,24,121)
740 CALL HCHAR(10,25,32)
750 CALL HCHAR(11,24,32,2)
760 CALL HCHAR(14,11,122)
770 CALL HCHAR(14,12,32)
780 CALL HCHAR(15,11,32,2)
790 VEXP=-VEXP
800 RETURN
810 REM  EXPAND SAUCERS THAT ARE HORIZ ENLARGED
820 CALL HCHAR(5,8,144)
830 CALL HCHAR(5,9,145)
840 CALL HCHAR(10,24,146)
850 CALL HCHAR(10,25,147)
860 CALL HCHAR(11,24,148)
870 CALL HCHAR(11,25,149)
880 CALL HCHAR(14,11,150)
890 CALL HCHAR(14,12,151)
900 CALL HCHAR(15,11,152)
910 CALL HCHAR(15,12,153)
920 VEXP=-VEXP
930 RETURN
940 REM  EXPANDS/CONTRACTS SAUCERS HORIZONTALLY
950 IF (VEXP=1)*(HEXP=1)THEN 1080
960 IF (VEXP=1)*(HEXP=-1)THEN 1190
970 IF (VEXP=-1)*(HEXP=1)THEN 1320
980 CALL HCHAR(5,8,136)
990 CALL HCHAR(5,9,137)
1000 CALL HCHAR(10,24,138)
1010 CALL HCHAR(10,25,139)
1020 CALL HCHAR(11,24,32,2)
```

```
1030 CALL HCHAR(14,11,140)
1040 CALL HCHAR(14,12,141)
1050 CALL HCHAR(15,11,32,2)
1060 HEXP=-HEXP
1070 RETURN
1080 REM  BOTH ARE CURRENTLY EXPANDED
1090 CALL HCHAR(5,8,128)
1100 CALL HCHAR(5,9,32)
1110 CALL HCHAR(10,24,129)
1120 CALL HCHAR(11,24,130)
1130 CALL VCHAR(10,25,32,2)
1140 CALL HCHAR(14,11,131)
1150 CALL HCHAR(15,11,132)
1160 CALL VCHAR(14,12,32,2)
1170 HEXP=-HEXP
1180 RETURN
1190 REM  SAUCERS ARE CURRENTLY EXPANDED VERT NOT
    HORIZ
1200 CALL HCHAR(5,8,144)
1210 CALL HCHAR(5,9,145)
1220 CALL HCHAR(10,24,146)
1230 CALL HCHAR(10,25,147)
1240 CALL HCHAR(11,24,148)
1250 CALL HCHAR(11,25,149)
1260 CALL HCHAR(14,11,150)
1270 CALL HCHAR(14,12,151)
1280 CALL HCHAR(15,11,152)
1290 CALL HCHAR(15,12,153)
1300 HEXP=-HEXP
1310 RETURN
1320 REM   SAUCERS ARE NORMAL IN VERT BUT ARE HORIZ
    ENLARGED
1330 CALL HCHAR(5,8,120)
1340 CALL HCHAR(5,9,32)
1350 CALL HCHAR(10,24,121)
1360 CALL HCHAR(10,25,32)
1370 CALL HCHAR(11,24,32,2)
1380 CALL HCHAR(14,11,122)
1390 CALL HCHAR(14,12,32)
1400 CALL HCHAR(15,11,32,2)
1410 HEXP=-HEXP
1420 RETURN
1430 MSG$="FCTN+1 - EXAND/CONTRACT"
1440 ROW=18
1450 COL=4
1460 GOSUB 410
1470 MSG$="VERTICALLY"
1480 ROW=19
1490 COL=13
1500 GOSUB 410
1510 MSG$="FCTN+2 - EXPAND/CONTRACT"
1520 ROW=21
1530 COL=4
1540 GOSUB 410
1550 MSG$="HORIZONTALLY"
1560 ROW=22
1570 COL=13
```

cont. on next page

Listing 4-4—cont.

```
1580 GOSUB 410
1590 MSG$="PRESS  X  TO EXIT"
1600 ROW=24
1610 COL=7
1620 GOSUB 410
1630 CALL KEY(3,KEY,STATUS)
1640 IF STATUS=0 THEN 1630
1650 IF KEY=88 THEN 1690
1660 IF (KEY<3)+(KEY>4)THEN 1630
1670 ON KEY-2 GOSUB 450,940
1680 GOTO 1630
1690 CALL CLEAR
1700 END
1710 REM   SMALL SAUCERS
1720 DATA 0000187EE77E0000
1730 DATA 003C7EE77E3C0000
1740 DATA 24183CE73C000000
1750 REM  EXPAND SAUCERS - VERTICALLY
1760 DATA 18187E7EE7E77E7E
1770 DATA 0000003C3C7E7EE7
1780 DATA E77E7E3C3C000000
1790 DATA 00422418183C3CE7
1800 DATA E73C3C0000000000
1810 REM  EXPAND SAUCERS - HORIZONTALLY
1820 DATA 0000033FFC3F0000
1830 DATA 0000C0FC3FFC0000
1840 DATA 000F3FFC3F0F0000
1850 DATA 00F0FC3FFCF00000
1860 DATA 0804030FFC0F0000
1870 DATA 1020C0F03FF00000
1880 REM  EXPAND VERTICAL SAUCER TO HORIZ AND HORIZ TO
    VERT
1890 DATA 03033F3FFCFC3F3F
1900 DATA C0C0FCFC3F3FFCFC
1910 DATA 000000F0F3F3FFFC
1920 DATA 000000F0F0FCFC3F
1930 DATA FC3F3F0F0F000000
1940 DATA 3FFCFCF0F0000000
1950 DATA 00300C03030F0FFC
1960 DATA 000C30C0C0F0F03F
1970 DATA FC0F0F0000000000
1980 DATA 3FF0F0000000000F
```

Explanation of Program

10-100	Clear screen, center title, and display introduction screen
110-120	Set vertical and horizontal expansion flags to indicate current size of saucers as normal (small saucers in both directions)
130-290	Read and set up graphics characters
300-360	Set colors for each of saucers

Fig. 4-4. Expanding Flying Saucers.

370-400	Display three original saucers at different location; go to main routine for valid key
410-440	Display message beginning at any row and column
460	Expand and contract saucers vertically. Check to see if both saucers are already expanded; if so, go to line 590
470	Check to see if saucers are expanded vertically but not horizontally; if so, go to line 700
480	Check to see if saucers are expanded horizontally but not vertically; if so, go to line 810
490-580	Expand three saucers vertically and sets flag to indicate vertical expansion
590-690	Contract expanded saucers in both directions
700-800	Contract saucers that are vertically but not horizontally expanded (saucers should be back to normal size)
810-930	Vertically expand saucers that are currently horizontally expanded
950	Expand and contract saucers horizontally. Check to see if they are expanded in both directions. If so, go to line 1080
960	Check to see if saucers are vertically but not horizontally expanded. If so, go to line 1190
970	Check to see if saucers are horizontally but not vertically expanded. If so, go to line 1320
980-1070	Horizontally expand original size saucers

1080-1180	Horizontally contract saucers (they were expanded in both directions)
1190-1310	Horizontally expand saucers (they were expanded vertically)
1320-1420	Horizontally contract saucers. Return them to normal size
1430-1620	Display keys used to expand/contract saucers. Display the key used to exit program
1630-1680	Wait for valid key to be pressed. Valid keys are FCTN + 1, FCTN + 2, and X
1690-1700	Exit program
1710-1740	Data for original size saucers. Each line contains hexadecimal notation for a saucer
1750-1800	Data for vertically expanding saucers
1810-1870	Data for horizontally expanding saucers
1880-1980	Data for expanding saucers that have been expanded in either but not both directions

BASIC ANIMATION

We can enlarge or contract graphics characters. Can we move them around on the screen? In fact, we can do just that. The effect of moving objects (graphics or normal characters) around on the screen is called *animation*. The most basic kind of animation is moving characters across the screen. It sounds easy, and it is. There is only one thing to remember: as you move something to a new screen location, you have to erase the object at the old location. You may think erasing and redisplaying objects will be slow and cumbersome, but the computer does this so quickly, you can't tell that that is what is happening.

FLYING SAUCER

In the last program, three flying saucers were stationary. Let's take one of them and make it move across the screen. The program shown in Listing 4-5 does that. To make it more interesting, I've added some noises so you'll be able to hear as well as see the flying saucer. When you run the program, your screen will look similar to Fig. 4-5.

Listing 4-5

```
10 REM  FLYING SAUCER
20 TT$="**  FLYING SAUCER  **"
30 CALL CLEAR
40 PRINT TAB(INT((29-LEN(TT$))/2));TT$
```

```
50 PRINT :::"  THIS PROGRAM DEMONSTRATES ANIMATION.":::
60 PRINT "  IT MOVES A FLYING SAUCER  ACROSS THE
    SCREEN.":::::::
70 PRINT "   PRESS ANY KEY TO BEGIN"
80 CALL KEY(0,KEY,STATUS)
90 IF STATUS<=0 THEN 80
100 CALL SCREEN(12)
110 CALL CLEAR
120 ROW=10
130 CALL COLOR(13,2,1)
140 CALL CHAR(128,"0000187EFF7E0000")
150 FOR COL=2 TO 32
160 CALL HCHAR(ROW,COL,128)
170 CALL SOUND(-200,790,10,-6,12)
180 FOR I=1 TO 50
190 NEXT I
200 CALL HCHAR(ROW,COL,32)
210 NEXT COL
220 CALL CLEAR
230 PRINT "WANT TO SEE FLYING SAUCER   AGAIN? (Y/N)"
240 CALL KEY(0,KEY,STATUS)
250 IF (KEY<>89)*(KEY<>121)*(KEY<>78)*(KEY<>110)THEN
    240
260 IF (KEY=89)+(KEY=121)THEN 100
270 CALL CLEAR
280 END
```

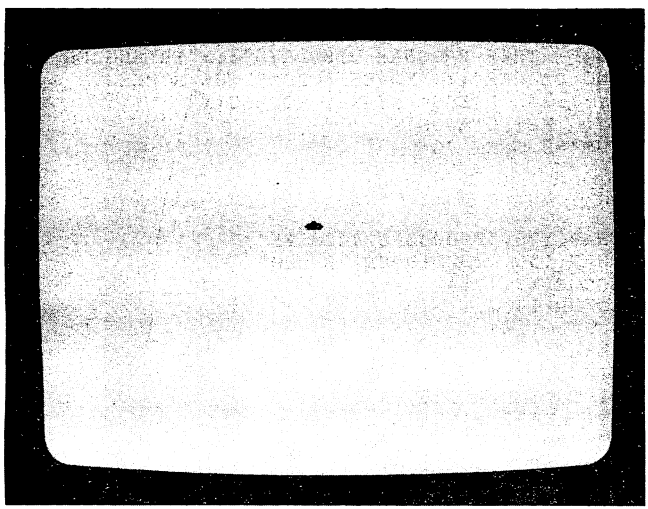


Fig. 4-5. Flying Saucer.

Explanation of Program

10-90	Clear screen, center title and display introduction screen
100-120	Set screen color and starting row number for displaying saucer

130	Set color of saucer
140	Define shape of saucer
150-190	Start loop to move saucer one column at a time, making noise as it moves
200	Erase saucer at first location
210	Repeat process
220-280	Ask if you want to see saucer again. If not, exit program

SAUCERS WITH TRIANGLES

The program shown in Listing 4-6 combines two sets of animation. It adds small triangles to the program shown in Listing 4-5. As the flying saucer moves across the screen, random triangles are displayed at the same time. When you run the program, your screen will look similar to Fig. 4-6.

Listing 4-6

```
10 REM  SAUCERS WITH TRIANGLES
20 TT$="**  SAUCERS & TRIANGLES  **"
30 CALL CLEAR
40 PRINT TAB(INT((29-LEN(TT$))/2));TT$
50 PRINT :::"  THIS PROGRAM DISPLAYS":::"TRIANGLES AS
    A FLYING SAUCER":::"MOVES ACROSS THE SCREEN."
60 PRINT ::::::"  PRESS ANY KEY TO BEGIN"
70 CALL KEY(3,KEY,STATUS)
80 IF STATUS<=0 THEN 70
90 BT=-1
100 RN=2
110 CN=1
120 CALL CHAR(128,"0000001824428lFF")
130 CALL CHAR(136,"0000187EE77E0000")
140 CALL COLOR(13,7,1)
150 CALL COLOR(14,2,1)
160 CALL SCREEN(16)
170 CALL CLEAR
180 RANDOMIZE
190 FOR I=1 TO INT(RND*5)+3
200 ROW=INT(RND*22)+1
210 COL=INT(RND*28)+3
220 CALL HCHAR(ROW,COL,128)
230 NEXT I
240 CN=CN+1
250 CALL GCHAR(RN,CN,WH)
260 IF (WH=128)*(BT=-1)THEN 370
270 IF (WH=128)THEN 410
280 CALL HCHAR(RN,CN,136)
290 IF BT<>1 THEN 330
300 CALL HCHAR(RN-1,CN-1,128)
310 BT=-1
320 GOTO 340
```

```
330 CALL HCHAR(RN-1,CN-1,32)
340 RN=RN+1
350 IF RN>24 THEN 460
360 GOTO 190
370 BT=1
380 CALL HCHAR(RN,CN,136)
390 CALL HCHAR(RN-1,CN-1,32)
400 GOTO 340
410 REM  PUT TRIANGLE IN BACK & THERE IS ONE IN FRONT
    OF SAUCER
420 CALL HCHAR(RN,CN,136)
430 CALL HCHAR(RN-1,CN-1,128)
440 BT=-1
450 GOTO 340
460 CALL SCREEN(4)
470 CALL CLEAR
480 PRINT "WANT TO SEE FLYING SAUCER  AND TRIANGLES
    AGAIN? (Y/N)"
490 CALL KEY(3,KEY,STATUS)
500 IF (KEY<>89)*(KEY<>78)THEN 490
510 IF KEY=89 THEN 90
520 CALL CLEAR
530 END
```

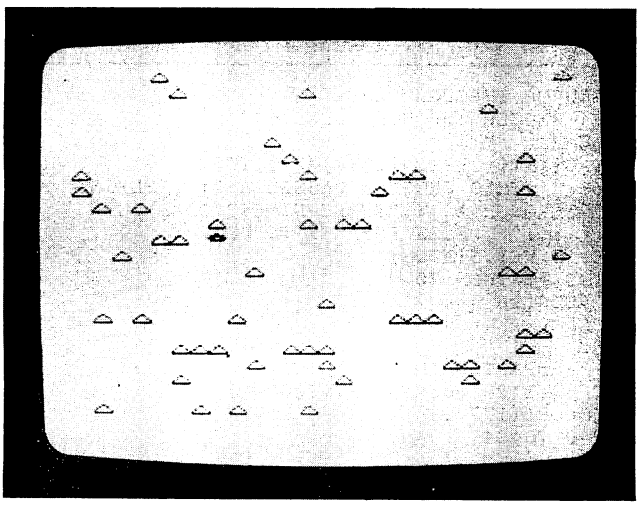


Fig. 4-6. Saucers with Triangles.

Explanation of Program

10-80	Clear screen, center title, and display introduction screen
90-170	Define shape of saucer and of triangle. Set color for characters as well as screen
180-230	Display anywhere from 3 to 8 triangles at random locations
240-270	Check to see if next saucer location already has a triangle. If

so, remember location so triangle can be re-displayed at same location after saucer moves

280-320 Display saucer with triangle behind it if necessary

330-360 If triangle need not be placed behind saucer, blank out space. Check to see if saucer has reached screen's outer edge. If not, move again

370-400 Move saucer and blank out location where it came from

410-450 Move saucer and re-display triangle when saucer is moved

460-530 Ask if you want to see saucer and triangles again. If not, exit program

SAUCER THROUGH CUBE

Listing 4-7 is similar to the previous program, but the saucer will fly through the three-dimensional cube that was drawn at the beginning of this chapter. When you run the program, your screen will look similar to Fig. 4-7.

Listing 4-7

```
10 REM   SAUCER THROUGH CUBE
20 TT$="**  SAUCER THROUGH CUBE  **"
30 CALL CLEAR
40 PRINT TAB(INT((29-LEN(TT$))/2));TT$
50 PRINT :::"  THIS PROGRAM PASSES A ":::"FLYING
   SAUCER THROUGH A":::"THREE-DIMENSIONAL CUBE."
60 PRINT ::::::"  PRESS ANY KEY TO BEGIN"
70 CALL KEY(0,KEY,STATUS)
80 IF STATUS<=0 THEN 70
90 CALL SCREEN(16)
100 CALL CLEAR
110 RESTORE
120 PRINT "PREPARING THE GRAPHICS...":::::::::::
130 CALL CHAR(112,"0000187EE77E0000")
140 CALL COLOR(11,7,1)
150 FOR CH=0 TO 34
160 READ CD$
170 CALL CHAR(CH+120,CD$)
180 NEXT CH
190 FOR CHARSET=12 TO 16
200 CALL COLOR(CHARSET,2,1)
210 NEXT CHARSET
220 CALL CLEAR
230 ROW=8
240 FOR COL=14 TO 18
250 CALL HCHAR(ROW,COL,106+COL)
260 NEXT COL
270 CALL HCHAR(ROW+1,13,125)
280 CALL HCHAR(ROW+1,14,126)
290 CALL HCHAR(ROW+1,17,127)
```

```
300 CALL HCHAR(ROW+1,18,128)
310 CALL HCHAR(ROW+2,12,129)
320 CALL HCHAR(ROW+2,13,130)
330 CALL HCHAR(ROW+2,14,131)
340 CALL HCHAR(ROW+2,16,132)
350 CALL HCHAR(ROW+2,18,133)
360 CALL HCHAR(ROW+3,12,134)
370 CALL HCHAR(ROW+3,13,135)
380 CALL HCHAR(ROW+3,14,136)
390 CALL HCHAR(ROW+3,15,137)
400 CALL HCHAR(ROW+3,18,138)
410 CALL HCHAR(ROW+4,12,139)
420 CALL HCHAR(ROW+4,14,140)
430 CALL HCHAR(ROW+4,15,141)
440 CALL HCHAR(ROW+4,16,142)
450 CALL HCHAR(ROW+4,17,143)
460 CALL HCHAR(ROW+4,18,144)
470 CALL HCHAR(ROW+5,12,145)
480 CALL HCHAR(ROW+5,13,146)
490 CALL HCHAR(ROW+5,15,147)
500 CALL HCHAR(ROW+5,16,148)
510 CALL HCHAR(ROW+5,17,149)
520 CALL HCHAR(ROW+6,12,150)
530 CALL HCHAR(ROW+6,13,151)
540 CALL HCHAR(ROW+6,14,152)
550 CALL HCHAR(ROW+6,15,153)
560 CALL HCHAR(ROW+6,16,154)
570 COL=2
580 CALL GCHAR(13,COL,WH)
590 IF WH<>32 THEN 670
600 CALL HCHAR(13,COL,112)
610 CALL HCHAR(13,COL-1,32)
620 COL=COL+1
630 IF COL>32 THEN 920
640 FOR DELAY=1 TO 75
650 NEXT DELAY
660 GOTO 580
670 CALL HCHAR(13,COL,112)
680 CALL HCHAR(13,COL-1,32)
690 FOR DELAY=1 TO 75
700 NEXT DELAY
710 COL=COL+1
720 CALL GCHAR(13,COL,WH2)
730 CALL HCHAR(13,COL,112)
740 CALL HCHAR(13,COL-1,WH)
750 FOR DELAY=1 TO 75
760 NEXT DELAY
770 COL=COL+1
780 IF WH2=32 THEN 600
790 CALL GCHAR(13,COL,WH3)
800 CALL HCHAR(13,COL,112)
810 CALL HCHAR(13,COL-1,WH2)
820 FOR DELAY=1 TO 75
830 NEXT DELAY
840 COL=COL+1
850 IF WH3=32 THEN 580
860 CALL HCHAR(13,COL,112)
```

cont. on next page

Listing 4-7—cont.

```
870 CALL HCHAR(13,COL-1,WH3)
880 COL=COL+1
890 FOR DELAY=1 TO 75
900 NEXT DELAY
910 GOTO 580
920 CALL CLEAR
930 PRINT "WANT TO SEE SAUCER THROUGH  CUBE AGAIN?
    (Y/N)"
940 CALL KEY(3,KEY,STATUS)
950 IF (KEY<>89)*(KEY<>78)THEN 940
960 IF KEY=89 THEN 220
970 CALL CLEAR
980 END
990 DATA 000000000000007F
1000 DATA 00000000000000FF
1010 DATA 00000000000000JFF
1020 DATA 00000000000000FF
1030 DATA 0000000000000080
1040 DATA 0001020408102040
1050 DATA C040404040404040
1060 DATA 0102040810204080
1070 DATA 8080808080808080
1080 DATA 0001020408102040
1090 DATA 8000000000000000
1100 DATA 4040404040404040
1110 DATA 0102040810204080
1120 DATA 8080808080808080
1130 DATA FF8080808080808080
1140 DATA FF0000000000000000
1150 DATA FF4040404040404040
1160 DATA FF0101010101010101
1170 DATA 8080808080808080
1180 DATA 8080808080808080
1190 DATA 4040404040407F80
1200 DATA 0101010101FF0101
1210 DATA 0000000000FF0000
1220 DATA 00000000000FF001
1230 DATA 8080808080808000
1240 DATA 8080808080808080
1250 DATA 0102040810204080
1260 DATA 0101010101010101
1270 DATA 0000000000000001
1280 DATA 0204081020408000
1290 DATA 8182848890A0C0FF
1300 DATA 00000000000000FF
1310 DATA 00000000000000FF
1320 DATA 0101010101010FF
1330 DATA 0204081020408000
```

Explanation of Program

10-80	Clear screen, center title, and display introduction screen
90-120	Set color of screen and display message

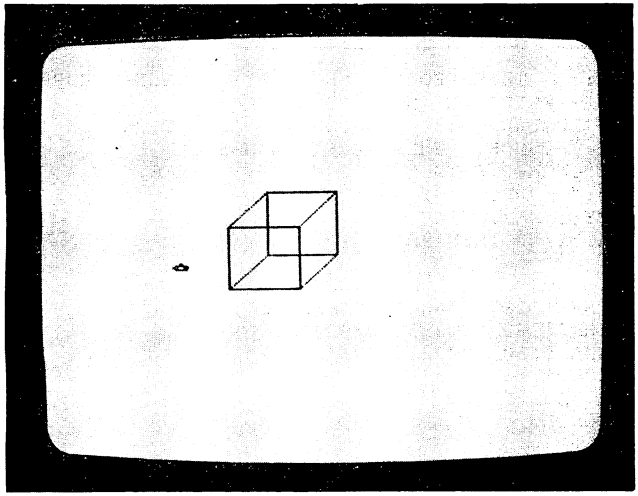


Fig. 4-7. Saucer through Cube.

130-220	Define saucer and three-dimsional cube. Set color for saucer and each section of cube
230-560	Draw cube
570-910	Move saucer across screen. If saucer crosses cube, remember to re-display pieces of cube when saucer moves on
920-980	Ask if you want to see saucer and cube again. If not, exit program
990-1330	Data for three-dimensional cube

BOMBARDIER

The program shown in Listing 4-8 combines two different kinds of animation with music to simulate a computer-type game. The program includes a bomber airplane which appears to fly across the screen and drop a bomb. Sounds effects include the engine of the plane as well as the sound of the bomb dropping, and an explosion when the bomb "hits the ground." When you run the program, your screen will look similar to Fig. 4-8.

Listing 4-8

```
10 REM     BOMBARDIER
20 TT$="**  BOMBARDIER  **"
30 CALL CLEAR
40 PRINT TAB(INT((29-LEN(TT$))/2));TT$
```

cont. on next page

Listing 4-8—cont.

```
50 PRINT :::"  THIS PROGRAM SHOWS A"::"BOMBER DROPPING
   A BOMB"::"WHILE FLYING ACROSS THE":
60 PRINT :"SCREEN.":::"  IT COMBINES ANIMATION
    WITH"::"SOUNDS.  THESE ARE THE BASIC":
70 PRINT :"INGREDIENTS FOR FORMING"::"GAMES."
80 PRINT ::"  PRESS ANY KEY TO BEGIN"
90 CALL KEY(3,KEY,STATUS)
100  IF STATUS<=0 THEN 90
110 ROW=4
120 COL=1
130 CALL CHAR(128,"000808C0FCFFFF00")
140 CALL CHAR(129,"0038383838381000")
150 CALL CHAR(130,"00925438FE385492")
160 CALL COLOR(13,2,1)
170 CALL SCREEN(6)
180 CALL CLEAR
190 COL=COL+1
200 IF COL>32 THEN 410
210 CALL HCHAR(4,COL,128)
220 CALL HCHAR(4,COL-1,32)
230 CALL SOUND(500,-7,22)
240 IF COL<5 THEN 320
250 IF ROW>24 THEN 320
260 ROW=ROW+1
270 IF ROW>24 THEN 350
280 CALL HCHAR(ROW,COL,129)
290 CALL SOUND(-150,1000-COL*10,32-COL)
300 CALL HCHAR(ROW-1,COL-1,32)
310 GOTO 180
320 FOR DELAY=1 TO 20
330 NEXT DELAY
340 GOTO 190
350 CALL HCHAR(24,COL,130)
360 FOR I=1 TO 29 STEP 7
370 CALL SOUND(I*10,110,I+1,-5,I)
380 NEXT I
390 CALL HCHAR(24,COL,32)
400 GOTO 190
410 CALL CLEAR
420 PRINT "WANT TO SEE THE BOMBARDIER  AGAIN? (Y/N)"
430 CALL KEY(3,KEY,STATUS)
440 IF (KEY<>89)*(KEY<>78)THEN 430
450 IF KEY=89 THEN 110
460 CALL CLEAR
470 END
```

Explanation of Program

10-100	Clear screen, center title, and display introduction screen
110-180	Define shape of plane, bomb, and explosion. Set screen color
190-270	Move plane across screen with engine noise
280-310	Display bomb dropping and play music
320-340	Delay routine

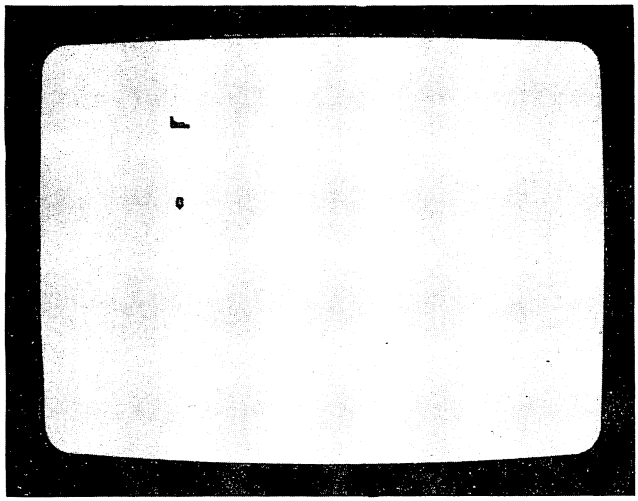


Fig. 4-8. Bombardier.

350-400 Display explosion followed by explosion noise
410-470 Ask if you want to see bombardier again. If not, exit program

SUMMARY

It is possible to combine various kinds of graphics with one another. Text may be added to make the programs more interesting, informative, and entertaining. Experiment with different combinations to find the ones most useful to you. Because the TI-99/4A allows you to define graphics characters, there is almost no limit to the number of exciting and different pictures, graphs, and games you'll be able to display on your computer.

Chapter 5

Music on the TI-99/4A

We have already explored some of the potential of sounds and noises on the TI-99/4A. We can also produce music. In Chapter 3, we presented some defintions that are necessary in working with music. You may find it helpful to refer to those definitions.

A keyword often used in programming music is the DATA statement. The DATA statement is used for storing numbers to be interpreted later as note values (frequencies). We will include, in our DATA statements, the note to play and the duration for that note. DATA statements are usually written first, though program lines written later may have lower line numbers and appear earlier in the program than the DATA statements.

You must "translate" the musical notes into numbers that the TI-99/4A can recognize as frequencies for the notes. (Appendix E gives a listing of the frequencies that can be used on the TI-99/4A with their corresponding notes.) I took the old song, "Swanee River" and converted it into the DATA statements shown in Listing 5-1.

Listing 5-1

```
1000 DATA 1000,329,250,293,250,261,250,329,250,293,
    500,261
1010 DATA 500,523,250,440,500,523,1000,392,500,329,
    500,261
```

cont. on next page

Listing 5–1—cont.

```
1020 DATA 2000,293,1000,329,250,293,250,261,250,329,
    250,293
1030 DATA 500,261,500,523,250,440,500,523,500,392,250,
    329
1040 DATA 250,261,500,293,500,293,2000,261,1000,493,
    250,523
1050 DATA 500,587,500,392,500,392,250,440,500,392,500,
    523
1060 DATA 500,523,500,440,500,349,500,440,2000,392,
    1000,329
1070 DATA 250,293,250,261,250,329,250,293,500,261,500,
    523
1080 DATA 250,440,500,523,500,392,250,329,250,261,500,
    293
1090 DATA 250,293,250,293,2000,261
1100 DATA 0,0
```

SWANEE RIVER

A DATA statement consists of two parts: duration and frequency. For instance, the first sections of the DATA statement the computer reads make the duration 1000 and the frequency 329. Duration corresponds to how long in seconds (where 1000 is equal to 1 second) the computer is to play that note.

For the computer to play the music, other statements must be added to the program. The complete "Swanee River" program is given in Listing 5-2.

Listing 5-2

```
10 REM  SWANEE RIVER
20 CALL CLEAR
30 TT$="**  SWANEE RIVER  **"
40 PRINT TAB(INT((29-LEN(TT$))/2));TT$
50 PRINT :::"  THIS PROGRAM PLAYS SWANEE"::"RIVER.":::
60 PRINT :::::"  PRESS ANY KEY TO BEGIN"
70 CALL KEY(0,KEY,STATUS)
80 IF STATUS<=0 THEN 70
90 CALL CLEAR
100 PRINT "  PLAYING SWANEE RIVER...":::::::::::
110 RESTORE
120 READ DUR,FREQ
130 IF (DUR=0)+(FREQ=0)THEN 160
140 CALL SOUND(DUR,FREQ,5)
150 GOTO 120
160 FOR I=1 TO 800
170 NEXT I
```

```
180 CALL CLEAR
190 PRINT "PLAY SWANEE RIVER            AGAIN? (Y/N)"
200 CALL KEY(0,KEY,STATUS)
210 IF (KEY<>89)*(KEY<>121)*(KEY<>78)*(KEY<>110)THEN
    200
220 IF (KEY=89)+(KEY=121)THEN 90
230 CALL CLEAR
240 END
1000 DATA 1000,329,250,293,250,261,250,329,250,293,
    500,261
1010 DATA 500,523,250,440,500,523,1000,392,500,329,
    500,261
1020 DATA 2000,293,1000,329,250,293,250,261,250,329,
    250,293
1030 DATA 500,261,500,523,250,440,500,523,500,392,250,
    329
1040 DATA 250,261,500,293,500,293,2000,261,1000,493,
    250,523
1050 DATA 500,587,500,392,500,392,250,440,500,392,500,
    523
1060 DATA 500,523,500,440,500,349,500,440,2000,392,
    1000,329
1070 DATA 250,293,250,261,250,329,250,293,500,261,500,
    523
1080 DATA 250,440,500,523,500,392,250,329,250,261,500,
    293
1090 DATA 250,293,250,293,2000,261
1100 DATA 0,0
```

Explanation of Program

10-80	Clear screen, center title, and display introduction screen
90-110	Display message that music is playing
120-150	Read duration and frequency from DATA statement and play that note
160-240	When music is done, ask if you want to hear "Swanee River" again. If not, exit program
1000-1100	The duration and frequencies for notes used in "Swanee River"

SOUND DEVELOPMENT

Sounds, as we have seen, are easy to make. The most useful knowledge, though, comes from experimenting with different frequencies and different durations. To help you develop your ability to program your computer for sounds, a sounds development program is given in Listing 5-3. This program helps you establish many parameters for the CALL SOUND routine. When you run the program, your screen will look similar to Fig. 5-1.

Listing 5-3

```
10 REM   SOUND DEVELOPMENT
20 TT$="**  SOUND DEVELOPMENT  **"
30 CALL CLEAR
40 PRINT TAB(INT((29-LEN(TT$))/2));TT$
50 PRINT :::"  THIS PROGRAM ALLOWS THE"::"USER TO
   CREATE A NOISE,"::
60 PRINT "TONE OR A MUSICAL NOTE.":::::::
70 PRINT "  PRESS ANY KEY TO BEGIN"
80 CALL KEY(0,KEY,STATUS)
90 IF STATUS<=0 THEN 80
100 CALL CLEAR
110 PRINT TAB(4);"WHAT TYPE OF SOUND?":
120 PRINT ::"      1) NOISE"::"      2) MUSICAL
   NOTE"::
130 PRINT "      3) EXIT PROGRAM":::
140 PRINT "      SELECT (1-3)":::::
150 CALL KEY(0,KEY,STATUS)
160 IF (KEY<49)+(KEY>51)THEN 150
170 ON KEY-48 GOTO 290,450,700
180 REM  ENTER DURATION AND VOLUME
190 CALL CLEAR
200 PRINT "HOW LONG SHOULD I PLAY THIS":"SOUND?
   (1-4250, 1000=1 SEC)"
210 INPUT DUR
220 IF (DUR<1)+(DUR>4250)THEN 190
230 CALL CLEAR
240 PRINT "PLAY AT WHAT VOLUME?"
250 PRINT "0 (LOUDEST), 30 (QUIETEST)"
260 INPUT VOL
270 IF (VOL<0)+(VOL>30)THEN 230
280 RETURN
290 REM  NOISE
300 CALL CLEAR
310 TYPE$="NOISE"
320 PRINT "    NOISE MENU":::
330 PRINT " 1) PERIODIC NOISE (TYPE 1)":" 2) PERIODIC
   NOISE (TYPE 2)"
340 PRINT " 3) PERIODIC NOISE (TYPE 3)":" 4) PERIODIC
   NOISE (VARIES      WITH FREQ. OF THIRD TONE
   SPECIFIED)"
350 PRINT :" 5) WHITE NOISE (TYPE 1)":" 6) WHITE NOISE
   (TYPE 2)":" 7) WHITE NOISE (TYPE 3)":
360 PRINT " 8) WHITE NOISE (VARIES WITH    FREQ. OF
   THIRD TONE         SPECIFIED)"::
370 PRINT " SELECT (1-8)"
380 CALL KEY(0,KEY,STATUS)
390 IF (KEY<49)+(KEY>56)THEN 380
400 ON KEY-48 GOSUB 740,760,780,800,820,840,860,880
410 FREQ=-(KEY-48)
420 GOSUB 180
430 GOSUB 550
440 GOTO 100
450 REM  MUSICAL NOTE
460 CALL CLEAR
470 PRINT "ENTER THE FREQUENCY"
```

```
480 PRINT :"(110 TO 44733)"
490 INPUT FREQ
500 IF (FREQ<110)+(FREQ>44733)THEN 450
510 TYPE$="MUSICAL NOTE"
520 GOSUB 180
530 GOSUB 550
540 GOTO 100
550 REM  PRINT INFORMATION
560 CALL CLEAR
570 PRINT "YOUR SOUND: ";TYPE$
580 IF SEG$(TYPE$,1,1)="M" THEN 600
590 PRINT ::"   ";TN$
600 PRINT :::"FREQUENCY     = ";FREQ
610 PRINT :"DURATION      = ";DUR
620 PRINT :"VOLUME        = ";VOL
630 PRINT ::::"PLAYING SOUND..."
640 CALL SOUND(DUR,FREQ,VOL)
650 IF DUR<=1100 THEN 670
660 DUR=INT(DUR/2)
670 FOR DELAY=1 TO DUR
680 NEXT DELAY
690 RETURN
700 REM  EXIT PROGRAM
710 CALL CLEAR
720 END
730 REM  GET TYPE OF NOISE
740 TN$="PERIODIC NOISE (TYPE 1)"
750 RETURN
760 TN$="PERIODIC NOISE (TYPE 2)"
770 RETURN
780 TN$="PERIODIC NOISE (TYPE 3)"
790 RETURN
800 TN$="PERIODIC NOISE (VARIES      WITH FREQ. OF
    THIRD TONE    SPECIFIED)"
810 RETURN
820 TN$="WHITE NOISE (TYPE 1)"
830 RETURN
840 TN$="WHITE NOISE (TYPE 2)"
850 RETURN
860 TN$="WHITE NOISE (TYPE 3)"
870 RETURN
880 TN$="WHITE NOISE (VARIES WITH    FREQ. OF THIRD
    TONE         SPECIFIED)"
890 RETURN
```

Explanation of Program

10-90	Clear screen, center title, and display introduction screen
100-170	Display main menu so you can choose sound, note, or noise
180-280	Ask for duration and volume
290-440	If selection was noise, display another menu to select type; play selected noise
450-540	Ask for frequency and play
550-630	Display information you have chosen (so you can write information to use again at later time)

640	Play note, sound, or noise
650-690	Delay program for duration of note, sound, or noise; return to main menu
700-890	Set variable equivalent to type (noise, sound, or note) computer is playing (so program can display what type is playing)

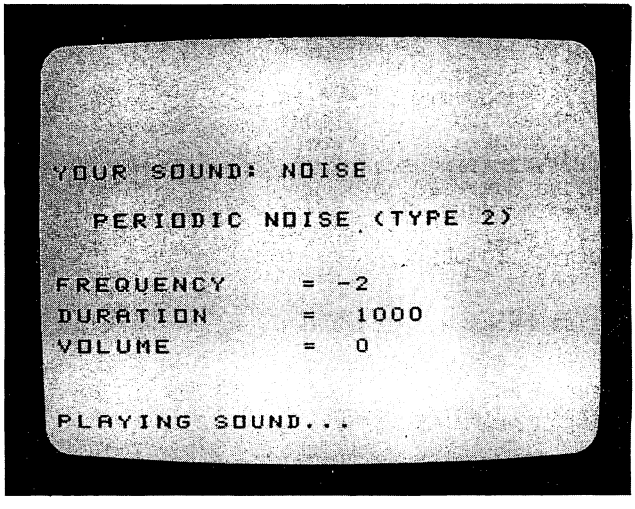


Fig. 5-1. Sound Development.

TRANSLATOR

To help you develop your ability to program musical notes, the program shown in Listing 5-4 translates any note into a corresponding DATA value. That value can then be used in any other program.

In order for the program to translate notes into DATA values, you must indicate to the computer the octave and note you want to play. The computer then displays a numeric value so you can write it down for future reference. When you run the program, your screen will look similar to Fig. 5-2.

Listing 5-4

```
10 REM   TRANSLATOR
20 TT$="**  TRANSLATOR  **"
30 DEF RD(Z)=INT(100*(Z+.005))/100
40 CALL CLEAR
50 PRINT TAB(INT((29-LEN(TT$))/2));TT$
```

```
60 PRINT :::"  THIS PROGRAM ALLOWS THE"::"USER TO
   TRANSLATE MUSICAL"::"NOTES INTO FREQUENCY DATA"
70 PRINT :"VALUES.  I WILL THEN ASK"::"YOU FOR A
   DURATION AND"::"PLAY THAT NOTE FOR YOU."
80 PRINT :::::"  PRESS ANY KEY TO BEGIN"
90 CALL KEY(0,KEY,STATUS)
100 IF STATUS<=0 THEN 90
110 CALL CLEAR
120 PRINT "ENTER THE OCTAVE"
130 INPUT "(0-LOWEST, 4-HIGHEST)":OC
140 IF (OC<0)+(OC>4)THEN 110
150 CALL CLEAR
160 PRINT "NOTES:":::"  1) A":"  2) A SHARP, B FLAT":
   "3) B":"  4) C":"  5) C SHARP, D FLAT"
170 PRINT "  6) D":"  7) D SHARP, E FLAT":"  8) E":"
   9) F":" 10) F SHARP, G FLAT"
180 PRINT " 11) G":" 12) G SHARP, A FLAT":::::
190 INPUT " SELECT (1-12) ":NOTE
200 IF (NOTE<1)+(NOTE>12)THEN 150
210 NOTE=(NOTE-1)+(OC*12)
220 FREQ=110*(2^(1/12))^NOTE
230 CALL CLEAR
240 PRINT "TO PLAY THIS NOTE, USE"::
250 PRINT TAB(8);RD(FREQ)::
260 PRINT "AS THE FREQUENCY VALUE IN":"THE CALL SOUND
   ROUTINE.":::
270 PRINT ::"PLAY NOTE? (Y/N)"
280 CALL KEY(0,KEY,STATUS)
290 IF (KEY<>89)*(KEY<>121)*(KEY<>78)*(KEY<>110)THEN
   280
300 IF (KEY=78)+(KEY=110)THEN 400
310 CALL CLEAR
320 PRINT "ENTER DURATION (1-4250)"
330 INPUT "(1 SEC = 1000) ":DUR
340 IF (DUR<1)+(DUR>4250)THEN 310
350 CALL CLEAR
360 PRINT "PLAYING NOTE FOR ";RD(FREQ)::"AT A DURATION
   OF";DUR::::::::::
370 CALL SOUND(DUR,FREQ,0)
380 FOR DELAY=1 TO INT(DUR/2)
390 NEXT DELAY
400 CALL CLEAR
410 PRINT "WANT TO ENTER MORE NOTES?   (Y/N)"
420 CALL KEY(0,KEY,STATUS)
430 IF (KEY<>89)*(KEY<>121)*(KEY<>78)*(KEY<>110)THEN
   420
440 IF (KEY=89)+(KEY=121)THEN 110
450 CALL CLEAR
460 END
```

Explanation of Program

10-100	Clear screen, center title, and display introduction screen
110-150	Ask you to enter octave from 0 to 4
160-200	Display menu of all possible notes and waits for you to select note

210-220 Calculate value needed to play selected note
230-300 Display value and ask if you want to hear frequency that is displayed
310-390 Ask for duration and play note
400-460 Ask if you want to translate more notes. If not, exit program

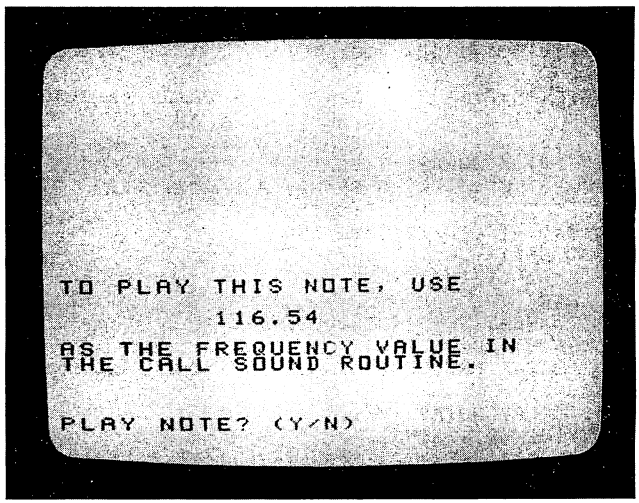


Fig. 5-2. Translator.

MULTIPLE VOICES

Certainly the most pleasing music on the TI-99/4A is produced by using three voices. These voices can be used to produce the melody and the harmony (chords) of a song. In fact, the TI-99/4A can sound like an electronic organ, complete with chords, when all three voices are utilized.

There are no tricks to getting all three voices running on the TI-99/4AA. Many people think that the task is difficult and arduous, but it's a fairly simple, mechanical process.

You need three items to program a three-voice musical piece:

1. The program used in this chapter to perform Brahms Lullaby but with your own data.
2. The translator program that was previously listed.
3. A piece of sheet music.

It also helps to have a little knowledge about music. You can usually find out what notes make up a chord by looking in a song book; often

chords are shown as letters rather than notes. In many organ books, the melody will be shown as "C" or "D minor", which doesn't give you a clue as to what notes make up that chord. That is why the *Translator* program listed above is useful for translating these codes into music.

BRAHMS

Once the chords are translated, you are ready to begin. In case you have nothing ready to play, program Listing 5-5 will play Brahm's Lullaby.

Listing 5-5

```
10 REM   BRAHMS LULLABY
20 TT$="**  BRAHMS  **"
30 CALL CLEAR
40 PRINT TAB(INT((29-LEN(TT$))/2));TT$
50 PRINT :::"  TO DEMONSTRATE THE USE OF":::"THREE
    VOICES ON THE TI-99/4A":::"I WILL PLAY BRAHMS
    LULLABY"
60 PRINT ::"FOR YOU."
70 PRINT :::::"  PRESS ANY KEY TO BEGIN"
80 CALL KEY(3,KEY,STATUS)
90 IF STATUS<=0 THEN 80
100 CALL CLEAR
110 RESTORE
120 PRINT " PLAYING BRAHMS LULLABY...":::::::::::
130 READ FV,SV,TV,NRP
140 IF (FV=0)*(SV=0)*(TV=0)*(NRP=0)THEN 360
150 IF (FV=0)+(SV=0)+(TV=0)THEN 220
160 VOL1=0
170 VOL2=0
180 VOL3=0
190 DUR=200*NRP
200 CALL SOUND(DUR,FV,VOL1,SV,VOL2,TV,VOL3)
210 GOTO 130
220 IF FV<>0 THEN 260
230 FV=110
240 VOL1=30
250 GOTO 270
260 VOL1=0
270 IF SV<>0 THEN 310
280 SV=110
290 VOL2=30
300 GOTO 320
310 VOL2=0
320 IF TV<>0 THEN 180
330 TV=110
340 VOL3=30
350 GOTO 190
360 FOR DELAY=1 TO 400
370 NEXT DELAY
380 CALL CLEAR
```

cont. on next page

Listing 5-5—cont.

```
390 PRINT "WANT TO PLAY BRAHMS LULLABY AGAIN? (Y/N)"
400 CALL KEY(3,KEY,STATUS)
410 IF (KEY<>89)*(KEY<>78)THEN 400
420 IF KEY=89 THEN 100
430 CALL CLEAR
440 END
450 DATA 0,247,0,3
460 DATA 147,247,196,1,147,294,196,6,147,247,196,6
470 DATA 147,294,196,4,147,0,196,3,147,247,196,2
480 DATA 147,294,196,2,147,392,196,4,147,370,196,6
490 DATA 147,330,196,2,147,330,185,4,147,294,185,4
500 DATA 147,220,185,2,147,247,185,2,147,262,185,4
510 DATA 147,220,185,6,147,247,185,2,147,262,185,4
520 DATA 147,262,185,3,147,220,185,2,147,262,185,2
530 DATA 147,370,185,2,147,330,185,2,147,294,185,4
540 DATA 147,370,185,5,147,370,196,1.5,147,392,196,7
550 DATA 147,196,196,4,165,392,196,7,165,330,196,2
560 DATA 165,262,196,2,147,294,196,8,147,247,196,2
570 DATA 147,196,196,2,147,262,185,4,147,294,185,4
580 DATA 147,330,185,4,147,294,196,8,147,196,196,4
590 DATA 165,392,196,8,165,330,196,2,165,262,196,2
600 DATA 147,294,196,10,147,196,196,2,147,262,185,4
610 DATA 147,247,185,4,147,220,185,4,147,196,196,13
620 DATA 0,0,0,0
```

Explanation of Program

10-100	Clear screen, center title, and display introduction screen
110-120	Display notice that Brahm's Lullaby is being played
130-350	Read data and play music
360-440	Delay program to allow you to hear final note. Ask if you want to hear Brahms again. If not, exit program
450-620	Data for three voices

SUMMARY

While running the programs shown in Chapter 3, you heard sounds and noises the TI-99/4A can produce. The programs listed in this chapter produce songs and even the effect of an electronic organ. By programming on your own, you can further explore your computer's musical "talents."

Chapter 6

Advanced Techniques

This chapter explains some techniques that can help you get the most from your TI-99/4A home computer.

ROUGH SCROLLING

One advanced technique is to improve on the computer's normal method of scrolling the screen. *Scrolling* is a term associated with how information displayed on your computer's screen seems to roll toward the top of the monitor as more information is added. The program shown in Listing 6-1 demonstrates the normal scrolling method of the TI-99/4A.

Listing 6-1

```
10 REM  ROUGH SCROLLING
20 TT$="**  ROUGH SCROLLING  **"
30 CALL CLEAR
40 PRINT TAB(INT((29-LEN(TT$))/2));TT$
50 PRINT :::"  THIS PROGRAM DEMONSTRATES"::"SCROLLING
   OF THE SCREEN IN"::
60 PRINT "A ROUGH FASHION.":::::::
70 PRINT "  PRESS ANY KEY TO BEGIN"
80 CALL KEY(0,KEY,STATUS)
90 IF STATUS<=0 THEN 80
100 CALL CLEAR
110 FOR I=1 TO 40
```

cont. on next page

Listing 6-1—cont.

```
120 PRINT " THIS IS A SCROLLING TEST"
130 NEXT I
140 CALL CLEAR
150 PRINT "WANT TO SEE SCROLLING AGAIN?(Y/N)"
160 CALL KEY(0,KEY,STATUS)
170 IF (KEY<>89)*(KEY<>121)*(KEY<>78)*(KEY<>110)THEN
    160
180 IF (KEY=89)+(KEY=121)THEN 100
190 CALL CLEAR
200 END
```

Explanation of Program

10-90	Clear screen, center title, and display introduction screen
100-130	Print message 40 times to show how screen scrolls
140-200	Ask if you want to see scrolling again. If not, exit program

UPWARD SCROLL

The program shown in Listing 6-2 demonstrates an alternate way of scrolling the screen upward. This program moves one character at a time up to the row above it, then displays a line of the same characters on the vacated line.

Listing 6-2

```
10 REM   UPWARD SCROLL
20 TT$="**  UPWARD SCROLL  **"
30 DIM CHARR(35)
40 CALL CLEAR
50 PRINT TAB(INT((29-LEN(TT$))/2));TT$
60 PRINT :::"  THIS PROGRAM GIVES YOU A"::"DIFFERENT
    WAY TO SCROLL THE"::"SCREEN UPWARD."
70 PRINT :::::::"  PRESS ANY KEY TO BEGIN"
80 CALL KEY(0,KEY,STATUS)
90 IF STATUS<=0 THEN 80
100 CALL CLEAR
110 LN$=" THIS IS A SCROLLING TEST  "
120 FOR I=1 TO LEN(LN$)
130 CHARR(I)=ASC(SEG$(LN$,I,1))
140 NEXT I
150 CALL CLEAR
160 PRINT LN$;
170 FOR ROW=23 TO 1 STEP -1
180 CTR=1
190 FOR COL=3 TO 32
200 CALL HCHAR(ROW,COL,CHARR(CTR))
210 CTR=CTR+1
220 NEXT COL
```

```
230 NEXT ROW
240 CALL CLEAR
250 PRINT "WANT TO SEE SCROLLING        AGAIN? (Y/N)"
260 CALL KEY(0,KEY,STATUS)
270 IF (KEY<>89)*(KEY<>121)*(KEY<>78)*(KEY<>110)THEN
    260
280 IF (KEY=89)+(KEY=121)THEN 150
290 CALL CLEAR
300 END
```

Explanation of Program

10-90	Clear screen, center title, and display introduction screen
100-160	Set up message to display. Convert each character to its ASCII representation
170-230	Move each character up one row; display another line
240-300	Ask if you want to see upward scroll again. If not, exit program

DOWNWARD SCROLL

The TI-99/4A does not provide an easy way for scrolling the screen downward. But the program given in Listing 6-3 does just that.

Listing 6-3

```
10 REM    DOWNWARD SCROLL
20 TT$="**  DOWNWARD SCROLL  **"
30 DIM CHARR(35)
40 CALL CLEAR
50 PRINT TAB(INT((29-LEN(TT$))/2));TT$
60 PRINT :::"  THIS PROGRAM SCROLLS THE"::"SCREEN
    DOWNWARD.":::
70 PRINT :::::::"  PRESS ANY KEY TO BEGIN"
80 CALL KEY(0,KEY,STATUS)
90 IF STATUS<=0 THEN 80
100 CALL CLEAR
110 LN$=" THIS IS A SCROLLING TEST  "
120 FOR I=1 TO LEN(LN$)
130 CHARR(I)=ASC(SEG$(LN$,I,1))
140 NEXT I
150 CALL CLEAR
160 FOR ROW=1 TO 24
170 CTR=1
180 FOR COL=3 TO 32
190 CALL HCHAR(ROW,COL,CHARR(CTR))
200 CTR=CTR+1
210 NEXT COL
220 NEXT ROW
230 CALL CLEAR
```

cont. on next page

Listing 6-3—cont.

```
240 PRINT "WANT TO SEE SCROLLING       AGAIN? (Y/N)"
250 CALL KEY(0,KEY,STATUS)
260 IF (KEY<>89)*(KEY<>121)*(KEY<>78)*(KEY<>110)THEN
    250
270 IF (KEY=89)+(KEY=121)THEN 150
280 CALL CLEAR
290 END
```

Explanation of Program

10-90	Clear screen, center title, and display introduction screen
100-150	Set up message to display; convert each character to its ASCII representation
160-220	Display message at top of screen; scroll downward until entire screen is full
230-290	Ask if you want to see downward scroll again. If not, exit program

SCROLL LEFT AND RIGHT

The program shown in Listing 6-4 scrolls the screen in a left and right direction.

Listing 6-4

```
10 REM    SCROLL LEFT & RIGHT
20 TT$="**  SCROLL LEFT & RIGHT  **"
30 DIM CHARR(35)
40 CALL CLEAR
50 PRINT TAB(INT((29-LEN(TT$))/2));TT$
60 PRINT :::"  THIS PROGRAM SCROLLS THE"::"SCREEN LEFT
    AND RIGHT.":::
70 PRINT :::::::"  PRESS ANY KEY TO BEGIN"
80 CALL KEY(0,KEY,STATUS)
90 IF STATUS<=0 THEN 80
100 CALL CLEAR
110 LN$="  THIS IS A SCROLLING TEST  "
120 FOR I=1 TO LEN(LN$)
130 CHARR(I)=ASC(SEG$(LN$,I,1))
140 NEXT I
150 CALL CLEAR
160 FOR I=1 TO 23
170 PRINT LN$
180 NEXT I
190 PRINT LN$;
200 FOR ROW=1 TO 24
210 CTR=1
220 FOR COL=1 TO 30
```

```
230 CALL HCHAR(ROW,COL,CHARR(CTR))
240 CTR=CTR+1
250 NEXT COL
260 NEXT ROW
270 FOR ROW=1 TO 24
280 CTR=1
290 FOR COL=3 TO 32
300 CALL HCHAR(ROW,COL,CHARR(CTR))
310 CTR=CTR+1
320 NEXT COL
330 NEXT ROW
340 CALL CLEAR
350 PRINT "WANT TO SEE SCROLLING       AGAIN? (Y/N)"
360 CALL KEY(0,KEY,STATUS)
370 IF (KEY<>89)*(KEY<>121)*(KEY<>78)*(KEY<>110)THEN
    360
380 IF (KEY=89)+(KEY=121)THEN 150
390 CALL CLEAR
400 END
```

Explanation of Program

10-90	Clear screen, center title, and display introduction screen
100-150	Set up message to display; convert each character to its ASCII representation
160-190	Fill screen with message
200-260	Scroll screen two characters to left
270-330	Scroll screen two characters to right
340-400	Ask if you want to see scrolling again. If not, exit program

CREATING CHARACTER SETS

This section describes a method for defining new characters rather than using the alphabet inherent to the TI-99/4A.

To define character sets, you will have to design the new characters, then store them in the old ASCII character codes on the TI-99/4A. This is exactly the same process used to develop the graphics characters in Chapters 2 and 4. In those chapters, you mostly used the ASCII codes numbered from 128 through 159. I mentioned that you could have used the lower ASCII codes when the graphics characters were stored but that we would learn more about that later. Well, this is later, and the lower ASCII codes are exactly where you are going to store the new graphics characters that are generated. By placing new definitions into ASCII codes 32-127, (see Appendix B for a list of all the ASCII codes), you'll re-define those characters. For example, if you want a heart-shaped character to appear when the **A** key is pressed, store the hexadecimal notation for a heart in the ASCII code of 65. Then, whenever the **A** key is pressed, a heart-shaped character will appear. NOTE: Once the program ends, the ASCII

codes will return to their normal (default) alphanumeric and symbol characters.

ALPHA SCRAMBLER

The program shown in Listing 6-5 demonstrates how easy it is to change the complete alphabet to nothing but scrambled characters. When you run the program, your screen will look similar to Fig. 6-1.

Listing 6-5

```
10 REM   ALPHA SCRAMBLER
20 CALL CLEAR
30 GOTO 80
40 FOR I=1 TO LEN(LN$)
50 CALL HCHAR(RN,CN+I,ASC(SEG$(LN$,I,1)))
60 NEXT I
70 RETURN
80 RANDOMIZE
90 RESTORE
100 RN=0
110 DIM ST$(11)
120 FOR I=1 TO 11
130 READ ST$(I)
140 NEXT I
150 FLG=1
160 CALL CLEAR
170 FOR I=1 TO 10
180 PRINT ST$(I)::
190 NEXT I
200 IF FLG THEN 230
210 PRINT ::
220 GOTO 260
230 PRINT :ST$(11)
240 CALL KEY(0,KEY,STATUS)
250 IF STATUS<=0 THEN 240
260 LN$="ALPHA SCRAMBLER WORKING..."
270 RN=23
280 CN=2
290 GOSUB 40
300 FOR ROW=1 TO 24
310 FOR COL=3 TO 30
320 CALL GCHAR(ROW,COL,CHR)
330 IF CHR=32 THEN 360
340 CHSET=INT(RND*22)+128
350 CALL HCHAR(ROW,COL,CHSET)
360 NEXT COL
370 NEXT ROW
380 FOR DELAY=1 TO 1000
390 NEXT DELAY
400 CTR=1
410 FOR RN=2 TO 20 STEP 2
420 CN=2
430 LN$=ST$(CTR)
```

```
440 CTR=CTR+1
450 GOSUB 40
460 NEXT RN
470 RN=23
480 CN=2
490 LN$="RUN SCRAMBLER AGAIN? (Y/N)"
500 GOSUB 40
510 CALL KEY(3,KEY,STATUS)
520 IF (KEY<>89)*(KEY<>78)THEN 510
530 IF KEY=89 THEN 560
540 CALL CLEAR
550 END
560 FLG=0
570 GOTO 160
580 DATA "  **  ALPHA SCRAMBLER  **"
590 DATA "  THIS PROGRAM SCRAMBLES","THE ALPHABET.",
    "EACH LETTER WILL HAVE A"
600 DATA "DIFFERENT CHARACTER","ASSOCIATED WITH IT.
    TO","DEMONSTRATE THIS EFFECT I"
610 DATA "WILL LEAVE THIS MESSAGE ON","THE SCREEN SO
    YOU CAN SEE","THE NEW CHARACTERS."
620 DATA "  PRESS ANY KEY TO BEGIN"
```

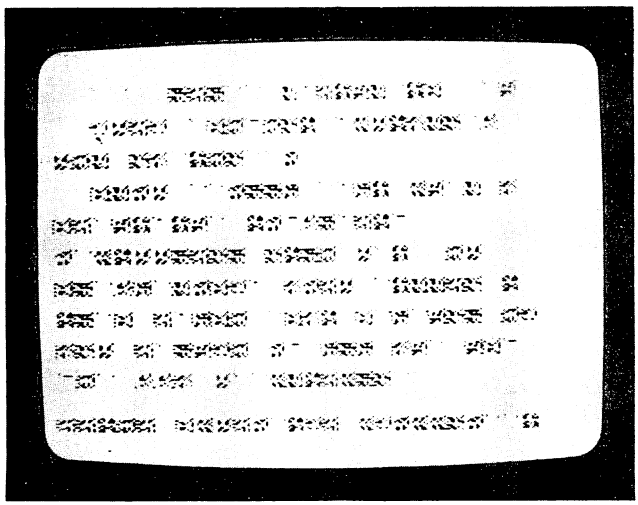


Fig. 6-1. Alpha Scrambler.

Explanation of Program

10-30	Clear screen
40-70	Display any message anywhere on screen
80-220	Fill array with messages to display on screen; display messages one by one

230-250	Wait for you to press any key
260-290	Print message that *Alpha Scrambler* is working
300-370	Start by changing only valid characters (all characters except spaces) to random character
380-500	Delay program so you can view new character set, then turn set back to normal alphanumeric and symbol characters
510-550	Ask whether you want to see *Alpha Scrambler* again. If not, exit program
560-570	If you want to see program again, reset flag and go to that routine
580-620	DATA statements containing messages to print

CHARACTERS

The program shown in Listing 6-6 changes certain letters of the alphabet into graphics characters. The program changes the onscreen letters "A," "B," and "C" into heart-, diamond-, and arrow-shaped characters, respectively. After a short delay, the characters are returned to their normal state. When you run the program, your screen will look similar to Fig. 6-2.

Listing 6-6

```
10 REM  CHARACTERS
20 TT$="**  CHARACTERS  **"
30 CALL CLEAR
40 PRINT TAB(INT((29-LEN(TT$))/2));TT$
50 PRINT :::"  THIS PROGRAM DEMONSTRATES"::"HOW TO
   CHANGE CERTAIN"::
60 PRINT "CHARACTERS.  IN THIS PROGRAM"::
70 PRINT "I WILL CHANGE THE LETTERS A,"::"B AND C TO
   THREE DIFFERENT"::
80 PRINT "CHARACTERS."::::::
90 PRINT "  PRESS ANY KEY TO BEGIN"
100 CALL KEY(0,KEY,STATUS)
110 IF STATUS<=0 THEN 100
120 CALL CLEAR
130 RESTORE
140 PRINT "  I WILL LEAVE THIS MESSAGE"::"ON THE
   SCREEN WHILE I"
150 PRINT :"CHANGE THE LETTERS A, B, AND"::"C TO THREE
   DIFFERENT"::
160 PRINT "CHARACTERS.":::
170 PRINT "  AFTER A DELAY PERIOD, I"::"WILL THEN
   CHANGE THEM BACK"::
180 PRINT "TO NORMAL.":::::
190 FOR DELAY=1 TO 1000
200 NEXT DELAY
```

```
210 FOR CH=0 TO 2
220 READ CHCODE$
230 CALL CHAR(65+CH,CHCODE$)
240 NEXT CH
250 FOR DELAY=1 TO 2500
260 NEXT DELAY
270 CALL CLEAR
280 FOR I=0 TO 2
290 READ CHCODE$
300 CALL CHAR(65+I,CHCODE$)
310 NEXT I
320 PRINT "WANT TO SEE CHARACTER":"CHANGER AGAIN?
    (Y/N)"
330 CALL KEY(0,KEY,STATUS)
340 IF (KEY<>89)*(KEY<>121)*(KEY<>78)*(KEY<>110)THEN
    330
350 IF (KEY=89)+(KEY=121)THEN 120
360 CALL CLEAR
370 END
380 DATA 6699818181422418
390 DATA 0010284482442810
400 DATA 001E060A12204080
410 DATA 003844447C444444
420 DATA 00F84444784444F8
430 DATA 0038444040404438
```

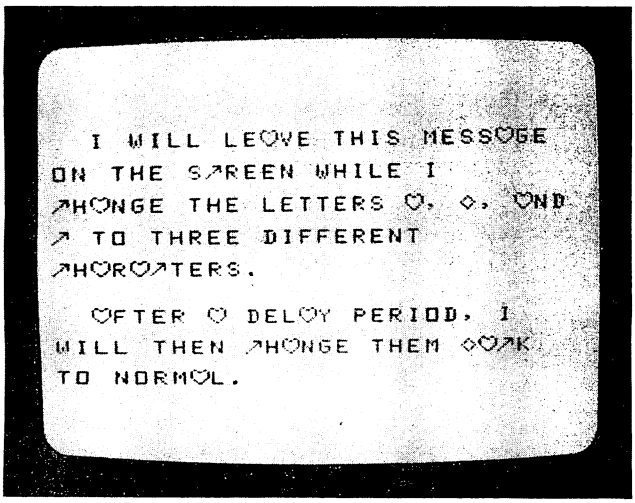


Fig. 6-2. Characters.

Explanation of Program

10-110	Clear screen, center title, and display introduction screen
120-200	Display message and delay program so you can see message
210-270	Change letters A, B, and C. Delay program so you can see new characters

280-370 Change characters back to letters; ask if you want to see program again. If not, exit

380-430 Data for new characters and for changing them back to their alphabetic form

CHARACTER CHANGER

The program shown in Listing 6-7 changes the entire alphabet and randomly replaces each letter with one of eight different characters. When you run the program, your screen will look similar to Fig. 6-3.

Listing 6-7

```
10 REM   CHARACTER CHANGER
20 CALL CLEAR
30 GOTO 80
40 FOR I=1 TO LEN(LN$)
50 CALL HCHAR(RN,CN+I,ASC(SEG$(LN$,I,1)))
60 NEXT I
70 RETURN
80 RANDOMIZE
90 CALL COLOR(11,2,1)
100 CALL COLOR(12,13,1)
110 CALL COLOR(13,16,1)
120 CALL COLOR(14,7,1)
130 CALL COLOR(15,11,1)
140 CALL COLOR(16,14,1)
150 CALL CHAR(112,"0103070F1F3F7FFF")
160 CALL CHAR(120,"FFFEFCF8F0E0C080")
170 CALL CHAR(128,"FFFFFFFF00000000")
180 CALL CHAR(136,"0F0F0F0F0F0F0F0F")
190 CALL CHAR(144,"F0F0F0F0F0F0F0F0")
200 CALL CHAR(152,"00000000FFFFFFFF")
210 RESTORE
220 RN=0
230 DIM ST$(11)
240 FOR I=1 TO 9
250 READ ST$(I)
260 NEXT I
270 FLG=1
280 CALL CLEAR
290 FOR I=1 TO 8
300 PRINT ST$(I)::
310 NEXT I
320 IF FLG THEN 350
330 PRINT :::
340 GOTO 380
350 PRINT ::ST$(9)
360 CALL KEY(0,KEY,STATUS)
370 IF STATUS<=0 THEN 360
380 LN$="CHARACTER CHANGER WORKING..."
390 RN=23
400 CN=2
```

```
410 GOSUB 40
420 FOR ROW=1 TO 24
430 FOR COL=3 TO 30
440 CALL GCHAR(ROW,COL,CHR)
450 IF CHR=32 THEN 490
460 CHSET=INT(RND*6)+1
470 CHSET=CHSET*8+104
480 CALL HCHAR(ROW,COL,CHSET)
490 NEXT COL
500 NEXT ROW
510 FOR DELAY=1 TO 1000
520 NEXT DELAY
530 CTR=1
540 FOR RN=5 TO 19 STEP 2
550 CN=2
560 LN$=ST$(CTR)
570 CTR=CTR+1
580 GOSUB 40
590 NEXT RN
600 RN=23
610 CN=2
620 LN$="  RUN CHANGER AGAIN? (Y/N)  "
630 GOSUB 40
640 CALL KEY(3,KEY,STATUS)
650 IF (KEY<>89)*(KEY<>78)THEN 640
660 IF KEY=89 THEN 690
670 CALL CLEAR
680 END
690 FLG=0
700 GOTO 280
710 DATA " **  CHARACTER CHANGER  ** "
720 DATA "  THIS PROGRAM DEMONSTRATES","HOW TO CHANGE
    A CHARACTER","SET TO A DIFFERENT SET."
730 DATA "TO DEMONSTRATE THIS EFFECT"
740 DATA  "I WILL LEAVE THIS MESSAGE","ON THE SCREEN
    SO YOU CAN","SEE THE NEW CHARACTERS."
750 DATA "  PRESS ANY KEY TO BEGIN"
```

Explanation of Program

10-30	Clear screen and go to main routine
40-70	Display message anywhere on screen
80-140	Set color for each graphics character
150-200	Define each new graphics character
210-370	Clear screen and display messages
380-500	Change each character on screen to random graphics character
510-630	Delay then change characters back to alphabetic form
640-680	Ask if you want to see changer again. If not, exit program
690-700	If you want to see program again, reset flag and go to that routine
710-750	Data for messages to display onscreen

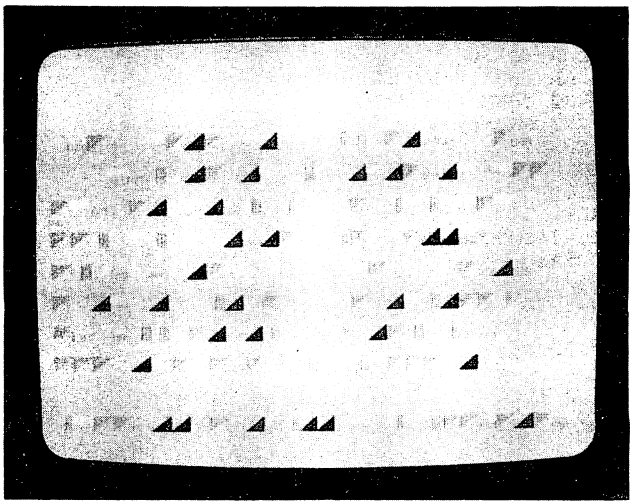


Fig. 6-3. Character Changer.

SUMMARY

There you have it: routines to help you make the most of your TI-99/4A. I hope you've learned a great deal from this book. Feel free to make changes to any of the programs in the book. Remember, the best way to learn is to take someone else's program, look through it thoroughly, then try making small changes here and there to see what effect your changes have. Also, take time to explore the TI-99/4A commands. The TI-99/4A is a powerful machine. Keep studying and working at getting the most out of your computer.

Appendix A

Color Codes for the TI-99/4A

1 = Transparent
2 = Black
3 = Medium Green
4 = Light Green
5 = Dark Blue
6 = Light Blue
7 = Dark Red
8 = Cyan
9 = Medium Red
10 = Light Red
11 = Dark Yellow
12 = Light Yellow
13 = Dark Green
14 = Magenta
15 = Gray
16 = White

Appendix B

Standard ASCII Character Codes

STANDARD ASCII CHARACTER CODES

ASCII CODE	CHARACTER	ASCII CODE	CHARACTER
32	(space)	49	1
33	! (exclamation point)	50	2
34	" (quote)	51	3
35	# (number or pound sign)	52	4
36	$ (dollar)	53	5
37	% (percent)	54	6
38	& (ampersand)	55	7
39	' (apostrophe)	56	8
40	((open parenthesis)	57	9
41	) (close parenthesis)	58	: (colon)
42	* (asterisk)	59	; (semicolon)
43	+ (plus)	60	< (less than)
44	, (comma)	61	= (equals)
45	− (minus)	62	> (greater than)
46	. (period)	63	? (question mark)
47	/ (slant)	64	@ (at sign)
48	0	65	A

ASCII CODE	CHARACTER	ASCII CODE	CHARACTER
66	B	98	b
67	C	99	c
68	D	100	d
69	E	101	e
70	F	102	f
71	G	103	g
72	H	104	h
73	I	105	i
74	J	106	j
75	K	107	k
76	L	108	l
77	M	109	m
78	N	110	n
79	O	111	o
80	P	112	p
81	Q	113	q
82	R	114	r
83	S	115	s
84	T	116	t
85	U	117	u
86	V	118	v
87	W	119	w
88	X	120	x
89	Y	121	y
90	Z	122	z
91	[(open bracket)	123	{ (left brace)
92	\ (reverse slant)	124	¦
93	] (close bracket)	125	} (right brace)
94	^ (exponentiation)	126	~ (tilde)
95	_ (line)	127	DEL(appears on screen as a blank.)
96	` (grave)		
97	a		

Appendix C

Character Sets and Their Corresponding Character Codes

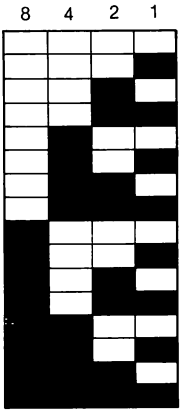
Here is a breakdown of the ASCII character codes into their corresponding character set numbers. To change the color of any character, use the character set number to reference that ASCII code.

Character Set Number	ASCII Codes
1	32-39
2	40-47
3	48-55
4	56-63
5	64-71
6	72-79
7	80-87
8	88-95
9	96-103
10	104-111
11	112-119
12	120-127
13	128-135
14	136-143
15	144-151
16	152-159

Appendix D

Pattern Identifier Conversion Table

The following is hexadecimal notation for one half of the bit-map:



HEXADECIMAL VALUES
0
1
2
3
4
5
6
7
8
9
A
B
C
D
E
F

Appendix E

Frequencies for Musical Tones

Note	Frequency
A	110
A♯,B♭	117
B	123
C (low C)	131
C♯,D♭	139
D	147
D♯,E♭	156
E	165
F	175
F♯,G♭	185
G	196
G♯,A♭	208
A (below middle C)	220

Note	Frequency
A (below middle C)	220
A♯,B♭	233
B	247
C (middle C)	262
C♯,D♭	277
D	294
D♯,E♭	311
E	330
F	349
F♯,G♭	370
G	392
G♯,A♭	415
A (above middle C)	440

Note	Frequency
A (above middle C)	440
A♯,B♭	466
B	494
C (high C)	523
C♯,D♭	554
D	587
D♯,E♭	622
E	659
F	698
F♯,G♭	740
G	784
G♯,A♭	831
A (above high C)	880

Note	Frequency
A (above high C)	880
A♯,B♭	932
B	988
C	1047
C♯,D♭	1109
D	1175
D♯,E♭	1245
E	1319
F	1397
F♯,G♭	1480
G	1568
G♯,A♭	1661
A	1760

SAMS

TI-99/4A™ Graphics and Sounds

The TI-99/4A has powerful graphics and sounds available. This book shows you these powers and how to make the most of them with 48 programs covering the following:

- Bit-mapped graphics
- Three-voice music
- Multicolored sprites
- Collisions between sprites
- Sound effects
- Multiple graphics combinations
- Low-resolution graphics
- High-resolution graphics
- Three-dimensional graphics
- Games
- Business
- Education

Howard W. Sams & Co., Inc.
4300 West 62nd Street, Indianapolis, Indiana 46268 U.S.A.

$9.95/22386

ISBN: 0-672-22386-4