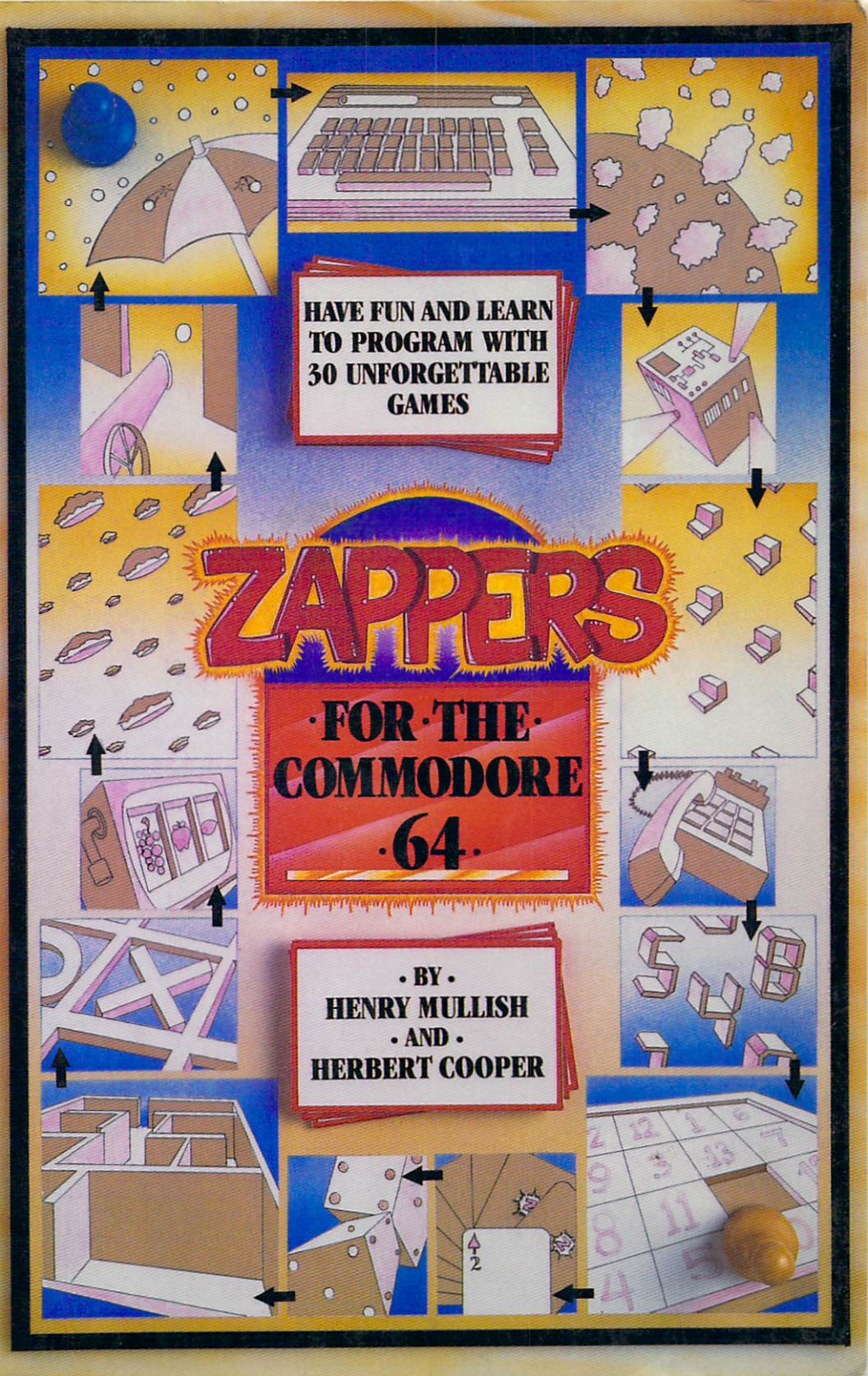

HAVE FUN AND LEARN TO PROGRAM WITH 30 UNFORGETTABLE GAMES
ZAPPERS
·FOR·THE·
COMMODORE
·64·
·BY·
HENRY MULLISH
·AND·
HERBERT COOPER

OTHER BOOKS BY HENRY MULLISH:

A BASIC APPROACH TO STRUCTURED BASIC

CRUNCHERS (*with Yin Chiu*)

AN INTRODUCTION TO PROGRAMMING IN BASIC ON THE TIMEX-SINCLAIR

STRUCTURED COBOL: A MODERN APPROACH

FINANCIAL ANALYSIS BY CALCULATOR

ZAPPERS: HAVING FUN PROGRAMMING AND PLAYING 23 GAMES FOR THE TI-99/4A (*with Dov Kruger*)

ZAPPERS

FOR THE

COMMODORE 64

30 Great Games to Program and Play on Your Commodore 64

by HENRY MULLISH
and HERBERT COOPER

Computer Book Division
Simon & Schuster, Inc.
NEW YORK

Published by the Computer Book Division
Simon and Schuster, Inc.
Simon & Schuster Building
Rockefeller Center
1230 Avenue of the Americas
New York, New York 10020
SIMON AND SCHUSTER and colophon are registered trademarks of
Simon & Schuster, Inc.
Designed by Irving Perkins Associates
Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

Library of Congress Cataloging in Publication Data
Mullish, Henry.
Zappers for the Commodore 64.

1. Commodore 64 (Computer)—Programming. 2. Basic
(Computer program language) 3. Computer games.
I. Cooper, Herbert. II. Title.
QA76.8.C64M86 1984 001.64'2 84-20241
ISBN: 0-671-50714-1

CONTENTS

INTRODUCTION

The Commodore 64 computer is, without doubt, one of the best computer buys you can make for the money. Not only does it come with 64K of memory but it also supports advanced color and sound features not found even in more expensive computers.

All thirty games found in this book were written specifically for the Commodore 64 computer and probably will not run on any other computer without substantial changes being made. Each program is accompanied by a detailed description of those features which are unique to the Commodore 64. They have all been written in the C64 version of the BASIC language.

In general, anyone with a knowledge of BASIC (regardless of the dialect) will be able to follow the logic of most of the programs. However, in many of them we have taken advantage of the BASIC-supported PEEK and POKE commands. These powerful commands are used to pass information to and from specific locations in memory. Whenever the sound feature is used, there is no alternative but to resort to these PEEKs and POKEs, since the sound chip is controlled by specific locations in memory which can be accessed only through these BASIC commands. Sometimes these are used merely to improve the efficiency of the program, to speed it up, to enhance the display, or to shorten the programs. Whenever they are used, we describe their effects in detail so that the stranger to PEEKs and POKEs will not feel completely at bay but will, on the contrary, expand his knowledge of the computer.

Whenever a new technique is introduced, we explain it in detail so that when the same concept occurs in subsequent programs, the reader will be familiar with the concept and we avoid having to repeat it. For this reason it would be advisable to initially try out the programs in the order in which they appear in the book. After a while, the techniques which are repeatedly used will become part of your own repertoire of tricks.

Not all of the characters obtainable from the keyboard are printable. For example, to clear the screen in immediate mode, all one has to do is to press the shifted CLR key. However, if within a program we wish to clear the screen, it would be impossible to denote this in our listing unless provision were made to incorporate the CLR key symbolically. This is made possible through the special characteristics of the *literal* within the PRINT statement (i.e., the string of characters, between the quotation marks, that is to be printed). If in this statement we type an open quote and then hit the CLR key, we shall see displayed on the screen a reversed graphics character—in this case, a reversed heart-shaped figure. In fact, each special function key, such as those that set the character color, is represented by a reversed graphics symbol when it is typed within the literal. These symbols act as control characters; that is, when the program is run, the symbols do not appear on the screen, but rather the desired function is performed, such as clearing the screen. Table 1 shows these control characters as they appear in program listings.

Since the program listings as shown in this book are exactly as produced by the Commodore 64, you will need this table to figure out which key to press in order to obtain the reversed graphics character that is shown in any given listing. (When examining the listings, be careful not to confuse these reversed graphics characters with "real" graphics characters.) For the reader who wishes to extend his understanding of this and any other of the sophisticated features such as sound generation, the official *Commodore 64 Programmer's Reference Guide* (Indianapolis: Howard W. Sams & Co.) is a valuable source of information.

In a recent article on computers published in *Time* magazine (May 23, 1983) it was reported that in the pioneering days there

TABLE 1

Function	Appearance Within Quotes
BLK	
WHT	
RED	
CYN	
PUR	
GRN	
BLU	
YEL	
Commodore Key-1 (ORANGE)	
Commodore Key-2 (BROWN)	
Commodore Key-3 (LT RED)	
Commodore Key-4 (GRAY 1)	
Commodore Key-5 (GRAY 2)	
Commodore Key-6 (LT GRN)	
Commodore Key-7 (LT BLU)	
Commodore Key-8 (GRAY 3)	
RVS ON	
RVS OFF	
HOME	
CLR	
CRSR down	
CRSR up	
CRSR right	
CRSR left	
F1	
F2	
F3	
F4	
F5	
F6	
F7	
F8	

were only 70,000 personal computers in households across the U.S., as compared with today's 4 million; There were only a

handful of programs available, and game playing was the principal pursuit of the home-computer owner. Today, according to the report, game playing is still the main use of home computers. According to a survey conducted by the Gallup Organization, 51 percent of personal-computer owners use their computers at home for video games, 46 percent for business purposes and as a child's learning tool, 42 percent as an adult's learning tool, 37 percent for balancing checkbooks and budgeting, 27 percent for business at home, and 18 percent for word processing. It is pointed out that this totals well above 100 percent because computers are often employed for more than one purpose. According to Future Computer Inc., a Texas-based marketing research firm, 52 percent of all home software sold in 1983, totaling some 12 million packages, were entertainment programs.

The popularity of television prepared the way for millions of children to take immediately to the arcade video machines that have achieved such popularity in recent years. Unfortunately, the video arcade does not provide the player with the ability to change a game should that be desired. What it is good at is demanding quarters without end.

In this book you will be introduced to a wide variety of games ranging from those demanding intellectual skills to action games, entertaining competitive games, and some interesting utilitarian programs. The programs are there for you to exploit to your heart's content. You are encouraged to type them into your Commodore 64 computer and store them on tape or disk for future use, thereby avoiding the necessity to type them in again in the future. You will find that game programs present a considerable intellectual challenge. The easiest way to discover this for yourself, of course, is to try and write one of your own. Not everybody, however, is suitably equipped for this chore, and toward that end we provide numerous suggestions at the end of each program to make them even more interesting or slightly different. Each program can be customized to suit the level of each member of the family, and thus the computer can assume the role of a magnet around which the family may gather. Although the microcomputer is an extremely versatile machine, it happens to be uniquely suited to game playing. What is so convenient about it

is that you can participate in the privacy of your own home.

If you have seen listings of game programs written for commercial use, you will have noticed that they are invariably *not* written in BASIC but rather in a much more arcane and cryptic language known as machine language. The practitioners of the art of machine-language programming are few and far between. Programs written in this manner are extremely difficult to follow, especially for the uninitiated. For this reason, we have taken very special care to be sure that each of the programs included in this book is written in the standard BASIC supported by the Commodore 64.

The colors chosen for these games were selected based upon what we considered to look good on our television set plus what was recommended by the manufacturer. On your own television or color monitor, however, these colors may not look as attractive and may even render some of the characters illegible. If adjusting the color controls does not rectify the situation, you may opt to change the color selections to suit your own equipment.

THE PROGRAM LISTINGS

In order to make the program listings in this book more readable, we have printed them in a format that is slightly different from the way they will appear on the Commodore screen. For example, graphics characters are designed as they appear on the keyboard and in the *Commodore 64 User's Guide*. You will not see the outlining boxes on the screen. Also, all graphics characters (including control characters, arrows, and the pound sign) are given the space of two regular characters in these listings. When you type in a line containing any of these characters, therefore, the line will appear shorter on your screen.

Some lines in the programs in this book are more than forty characters long. When you type such a line into the Commodore 64, the cursor on the screen will "wrap around" to the next line and will continue displaying typed characters there. This continuation line is recognized as part of the first line. Again, for readability we have chosen to split lines (when necessary) in the listings at reasonable points, such as between words instead of

in the middle of a word. These line breaks will have no correspondence to the line as it appears on your screen. When you are typing a line that is continued on a new line, simply continue typing, inserting a space between words as you would normally. Do not press the RETURN key until you have typed the entire statement and are ready to begin with the next line number.

On occasion you will notice that a literal in a PRINT statement contains a number of blank spaces. More often than not, the number of spaces (inserted with a stroke of the space bar for each space) is of paramount importance. Blank spaces are sometimes used in the programs to delete previous information that is displayed on the screen. If the number of spaces included is less than the required number, unwanted information will remain on the screen, possibly leading to confusion. Whenever the number of spaces required may be unclear in the listings (whether due to sheer length or to adjacent graphics characters), we specify the number in the Lines of Special Interest section of each chapter.

Finally, some cautionary notes concerning the typing in of programs from any book. One of the most frequent errors in typing programs is mistaking the letter *I* for the number *1* and vice versa. Unfortunately, many typefaces indicate the letter *I* with a straight vertical line regardless of the fact that this might be confusing to a novice. Also, the letter *O* is easily confused with the digit *0*, and for that matter even the number *6* with the letter *G*. So when typing in a program, be absolutely sure of each character you are typing, since if it is incorrect, the results are unpredictable. And be sure to include every character, however minor it may appear. If a PRINT statement terminates with a semicolon, be sure to include it, because it affects the manner in which output is displayed on the screen. We mention this only because it can be so easily overlooked. Be especially careful if the semicolon is followed by a colon, indicating that the line contains more than one instruction.

CHAPTER 1

NOW YOU SEE IT...

This game is a test of your ability to remember numbers that are flashed on the screen in a sequence. You don't have to have a prodigious memory, because there are five levels (or speeds) on which you can play. Level (or speed) 1 is the easiest and level 5 is the hardest. The numbers 0 through 9 are listed at the top of the screen, and beneath them, below the corresponding number, one of the digits is flashed momentarily. This is repeated in a random sequence; by the end of the sequence each digit has been flashed four times except for one digit—a randomly selected one—which was flashed a fifth time. The idea is to recognize, without resorting to pencil and paper, which digit was flashed five times. You may go through another round on the same level as many times as you wish. When you are done, the program displays your performance rating—how many you scored right and how many wrong, together with the corresponding percentages. After each game you may restart the program to play on a different level.

```
10 REM NOW YOU SEE IT...
20 DIM NUM(40),PIT(40)
30 CNT=0:YES=0
40 PRINT "WHAT'S YOUR SPEED (1-5)? ";
```

```
50 GET A$:IF A$="" THEN 50
60 IF A$<"1" OR A$>"5" THEN 50
70 PRINT A$
80 S=100+(53-ASC(A$))*275
90 PRINT "▼ Q Q Q Q Q Q Q Q Q"
100 PRINT TAB(10);"0 1 2 3 4 5 6 7 8 9"
110 CNT=CNT+1
120 PRINT "◥"
130 GOSUB 400
140 FOR I=0 TO 40
150 PRINT TAB(9+2*NUM(I));NUM(I);
160 FOR J=1 TO S:NEXT J:PRINT "▮▮ ▯"
170 FOR J=1 TO S:NEXT J
180 NEXT I
190 PRINT:PRINT
200 PRINT TAB(10);"◧WHAT'S THE DIGIT? ";
210 GET A$:IF A$="" THEN 210
220 IF A$<"0" OR A$>"9" THEN 210
230 PRINT A$
240 IF ASC(A$)-48=X THEN 320
250 PRINT:PRINT "NO, IT WAS";X
260 PRINT:PRINT "AGAIN (Y OR N)? ";
270 GET A$:IF A$="" THEN 270
280 IF A$<>"Y" AND A$<>"N" THEN 270
290 PRINT A$
300 IF A$="Y" THEN 90
310 GOTO 340
320 PRINT:PRINT "RIGHT!!":YES=YES+1
330 GOTO 260
340 GOSUB 600
350 END
400 REM LOAD THE SEQUENCE
410 FOR J=0 TO 40:PIT(J)=J:NEXT J
420 TP=41
430 FOR TMES=1 TO 4
440 FOR DIG=0 TO 9
450 Q=INT(RND(0)*TP)
460 NUM(PIT(Q))=DIG
```

```
470 TP=TP-1
480 PIT(Q)=PIT(TP)
490 NEXT DIG
500 NEXT TMES
510 X=INT(RND(0)*10)
520 NUM(PIT(0))=X
530 RETURN
600 REM PERFORMANCE RATING
610 PRINT "▼ ■ ■ ■ ■ ■";TAB(14);"YOUR SCORES:"
620 PRINT:PRINT
630 PRINT TAB(9);"CORRECT      INCORRECT"
640 PRINT TAB(9);"--------------------"
650 PRINT:PRINT:PRINT TAB(10);YES;
660 PRINT TAB(23);CNT-YES
670 FINE=INT(YES/CNT*100)
680 PRINT:PRINT TAB(9);"(";FINE;"■■% )";
690 PRINT TAB(22);"(";100-FINE;"■■% )"
700 RETURN
```

GLOSSARY OF VARIABLES

NUM An array of 41 elements numbered 0 through 40. (On the Commodore 64 all arrays have a 0th element, so that if a variable is DIMensioned 40, the computer reserves not 40 locations but 41. In this book we sometimes take advantage of this feature and at other times we ignore it.)
This array contains the sequence of randomly selected digits that are flashed on the screen. The sequence is generated at the beginning of each round and is saved in the array NUM.

PIT This array of 41 elements is also used to generate the sequence. It is used to ensure that whenever a digit is stored in a randomly selected location in the array NUM, that location is not used twice.

CNT This variable counts the number of rounds the user has played for the purpose of generating the performance rating at the conclusion of the game.

YES This variable, also used in generating the performance rating, counts how many times the player answers correctly.

S This holds a value which is proportional to the delay between flashes of the digits and is based on the speed value that was entered.

X Holds the digit that is flashed 5 times instead of 4.

LINES OF SPECIAL INTEREST

50–70: Get the level number. Here we introduce the interesting command GET. When keys are pressed on the keyboard, their corresponding ASCII values are placed in a buffer in memory. A GET command checks to see if there are any values resident in the buffer. If there are, it removes the first value from the buffer (the one that was there the longest time) and, assuming that the GET command is followed by a string variable (as it is here), the value from the buffer is converted to a character and is placed in the variable. If there are no values residing in the buffer, the null string is returned. When the GET command is used to receive input, no carriage return is necessary. All you have to do is to depress the key. In many game situations, this is a distinct advantage. Thus line 50 has the effect of waiting until any key is pressed on the keyboard. Line 60 ensures that the pressed key is a numeric key between 1 and 5. If it is, the value is printed in line 70. If not, the value is rejected and another value is waited for. The net effect is that the program totally ignores any key other than 1, 2, 3, 4, or 5.

80: The variable S is assigned a delay value proportional to the speed or level selected. Note that the ASC function is used to return the ASCII value of the keyed digit. The ASCII value for 0 is 48, the ASCII value for 1 is 49, that for 2 is 50, and so forth. Therefore, it is always possible to convert any digit from its character form to its numeric form by subtracting 48 from the value returned from the ASC function. In line 80 we are not so direct, but examination will show that the values assigned to S can range from 1200 for level 1 to 100 for level 5.

110: The variable CNT is incremented by 1, indicating that a new round is about to commence.

130: The subroutine at line 400 is called to set up the flashing sequence.

140–180: The sequence flashes on the screen underneath its corresponding digits. The correct positioning is accomplished by the TAB function in line 150. The PRINT command in line 160 backspaces over the digit and replaces it with a blank. The FOR/NEXT loops in lines 160 and 170 are what are called "delay" loops. Their

function is merely to keep the computer busy while the digit is displayed on the screen and to create a pause between the displaying of the digits. Note that here is where the variable S is used.

210–230: We again wait for a digit to be input from the keyboard, but this time it may be any digit from 0 to 9. This represents the player's guess as to which digit was flashed one extra time.

240: Here we convert the character representation of the digit to its numerical value and compare it with the correct answer. This is an example of the most common use of the ASC function.

270–290: The GET command is again used to check the keyboard, but this time it accepts only the characters Y or N.

320: The variable YES is incremented by 1 if the player guessed correctly.

340: The player has requested that the game be ended. The subroutine at line 600 is therefore called to display the performance rating.

Subroutine

400–530: This subroutine loads the digit sequence into the array NUM.

410–420: Line 410 initializes the elements of the array PIT to the numbers 1 through 40. These numbers correspond to subscripts of available locations in array NUM. In line 420 the variable TP (for TOP, but TOP cannot be used because it contains the keyword TO) is set to the number of available locations in array NUM, which is initially 41.

430–500: The FOR/NEXT loop indexed by variable TMES (we couldn't use TIMES because it contains the reserved keyword TI) ensures that each digit is inserted 4 times.

440–490: The FOR/NEXT loop with the index DIG (for DIGit) takes on the values of each of the digits for insertion into the array NUM.

450–460: The variable Q is assigned a randomly selected value from 0 to TP-1. This number is used to select an element of the array PIT. The value of this element is, in turn, used to subscript the array NUM, and this subscripted element of NUM is assigned the value residing in DIG.

470–480: TP is decremented by 1, indicating that an element has been "removed" from PIT. This element is replaced with the topmost element in PIT.

510–520: A random digit between 0 and 9 is selected and placed in the variable X. This digit is also placed in the last available element of NUM. (Note that at this point the subscript of this last remaining element ends up in PIT(0)).

Subroutine

600–700: Subroutine to display performance rating.

660: CNT-YES gives the number of wrong guesses that the player made.

670: FINE is assigned the percentage of correct answers.

680–690: The backspace character is printed before the percent sign because the PRINT command always tabs forward one space after printing a number.

690: 100 – FINE gives the percentage of wrong answers.

SUGGESTED ENHANCEMENTS

1. The game may be modified so that two players may compete, with separate scores being kept for them.
2. Rather than use numbers, letters of the alphabet may be substituted. You might want the whole range A through Z, although this would undoubtedly prove to be quite taxing for the average mind. This would require substantial changes in several lines. For example, lines 150, 240, 440, and 510 would have to be modified. You would have to represent the letters by numbers (perhaps by their ASCII equivalents).
3. The digits could be flashed randomly on the screen rather than under the corresponding digit.
4. Once the sound features have been covered, they could also be used to enhance the program.

CHAPTER 2

FIND-A-WORD

Word games have fascinated people of all ages for centuries. This game is a Commodore 64 version of a word game that is often published in the national newspapers and magazines. The program asks the player how many words (out of a pool of fifty) he would like the computer to embed in a mass of letters that are displayed on the screen in a 40 × 24 matrix. These words are randomly selected from the pool and are displayed going in one of eight directions on the screen (up or down, left-to-right or right-to-left, diagonally to the right and diagonally to the left in both directions). Words may share letters if they cross. The vacant space is filled in with random letters of the alphabet so as to make finding the hidden words that much more difficult. As the words are being placed in the matrix, sequential numbers are displayed on the screen to tell you of the progress the program is making. If you set the value of N at 50 (or some such large number), you should expect the computer to encounter some difficulty in placing the last of the words. Despite hours of testing with this particular vocabulary (stored in DATA statements), we have not experienced the situation where the computer was unable to place all of the words.

In the event that, at some point, the game proves too challenging, the player may press the F1 key. This will cause the hidden words to be highlighted in reversed graphics. Once the player is finished examining the screen, the F7 key is pressed and the player is asked whether another round is required.

```
10 REM FIND-A-WORD
20 DIM WURD$(50),MAT$(23,39)
30 RESTORE
40 FOR I=1 TO 50
50 READ WURD$(I)
60 NEXT I
70 FOR I=0 TO 23
80 FOR J=0 TO 39
90 MAT$(I,J)=""
100 NEXT J
110 NEXT I
120 INPUT"HOW MANY WORDS (1-50)";N
130 IF N<>INT(N) OR N<1 OR N>50 THEN 120
140 FOR I=1 TO N
150 X=INT(RND(0)*(51-I))+1
160 X$=WURD$(X)
170 WURD$(X)=WURD$(51-I)
180 GOSUB 400
190 PRINT I;
200 NEXT I
210 GOSUB 800
220 GET A$:IF A$="" THEN 220
230 IF A$="■" THEN 270
240 IF A$<>"■" THEN 220
250 GOSUB 900
260 GET A$:IF A$<>"■" THEN 260
270 PRINT:PRINT "ANOTHER (Y OR N) ?";
280 GET A$:IF A$="" THEN 280
290 IF A$<>"Y" AND A$<>"N" THEN 280
300 PRINT A$
310 IF A$="Y" THEN 30
320 PRINT "■FINISHED"
```

```
330 END
400 REM PLACE A WORD IN THE MATRIX
410 L=LEN(X$)
420 ROW=INT(RND(0)*23)
430 COL=INT(RND(0)*39)
440 VERT=INT(RND(0)*3)-1
450 IF VERT=0 THEN 480
460 HIZ=INT(RND(0)*3)-1
470 GOTO 490
480 HIZ=INT(RND(0)*2)*2-1
490 GOSUB 600
500 IF S=0 THEN 420
510 FOR Q=1 TO L
520 MAT$(ROW,COL)=MID$(X$,Q,1)
530 ROW=ROW+VERT:COL=COL+HIZ
540 NEXT Q
550 RETURN
600 REM CHECK FOR FIT IN MATRIX
610 S=0
620 IF HIZ<0 AND COL<L-1 THEN RETURN
630 IF HIZ>0 AND COL>40-L THEN RETURN
640 IF VERT<0 AND ROW<L-1 THEN RETURN
650 IF VERT>0 AND ROW>24-L THEN RETURN
660 FOR Q=ROW TO ROW+VERT*(L-1) STEP VERT
670 FOR R=COL TO COL+HIZ*(L-1) STEP HIZ
680 IF MAT$(Q,R)="" THEN 700
690 IF MAT$(Q,R)<>MID$(X$,ABS(Q-ROW)
    +ABS(R-COL)+1,1) THEN RETURN
700 NEXT R
710 NEXT Q
720 S=1
730 RETURN
800 REM DRAW THE MATRIX
810 PRINT "■";
820 FOR ROW=0 TO 23
830 FOR COL=0 TO 39
840 IF MAT$(ROW,COL)<>"" THEN 870
850 A$=CHR$(INT(RND(0)*26)+65)
```

```
860 PRINT A$;:GOTO 880
870 PRINT MAT$(ROW,COL);
880 NEXT COL,ROW
890 RETURN
900 REM COLOR IN THE WORDS
910 PRINT "■■■■";
920 FOR ROW=0 TO 23
930 FOR COL=0 TO 39
940 IF MAT$(ROW,COL)<>"" THEN 960
950 PRINT "■";:GOTO 970
960 PRINT MAT$(ROW,COL);
970 NEXT COL
980 NEXT ROW
990 PRINT "■■";:RETURN
1000 REM WORDS
1010 DATA"BENEFIT","INSIST","PLEASURE"
1020 DATA"STORM","ACRONYM","TRAPEZE"
1030 DATA"WORLDLY","FUNGUS","MARKET"
1040 DATA"YESTERDAY","GUIDANCE","YEARN"
1050 DATA"ENGAGE","RECLAIM","KNOWLEDGE"
1060 DATA"COLONY","UTILITY","JOKE"
1070 DATA"DRIVER","REMAINDER","HYDROGEN"
1080 DATA"OCTOPUS","TECHNICAL","LAUGH"
1090 DATA"QUIVER","VOLTAGE","NIMBLE"
1100 DATA"FOLIAGE","AMBITION","VALVE"
1110 DATA"GEOMETRY","LOCATION","GLACIER"
1120 DATA"OUTRAGEOUS","WARRANT","KICK"
1130 DATA"CIVILIZE","MATERNITY","HEART"
1140 DATA"DISARRAY","ZOMBIE","BLUEPRINT"
1150 DATA"MILDEW","IMPRESS","JINGLE"
1160 DATA"PERFECT","EXCESSIVE","QUOTA"
1170 DATA"NAVIGATE","SEPARATE"
```

GLOSSARY OF VARIABLES

**WURD$** This array of 50 string elements is loaded at the beginning of each round with the full set of words. In case you haven't already guessed, the reason we spell this variable WURD$

is that the more natural spelling WORD$ contains the BASIC keyword OR.

**MAT$** This 2-dimensional string matrix is used to represent the positions on the screen. The selected words are fitted into this matrix. Each element holds only one character. Of the two numbers used to subscript MAT$, the first represents the row on the screen (0–23) while the second represents the column (0–39). Row 24 is not used because this would cause scrolling problems.

N This variable holds the number of words the player wants placed on the screen.

**X$** This variable holds a word randomly selected from the array WURD$ and is used to pass this word to the subroutine in line 400.

L The length of the word in X$ is held in this variable. This is done mainly for conciseness and a slight savings in time.

ROW This variable first occurs in the subroutine at line 400. It represents the row in which the first character of X$ is tentatively placed.

COL This variable also first occurs in the subroutine at line 400. It represents the column in which the first character of X$ is tentatively placed.

VERT First used in the subroutine at line 400, this variable determines the vertical direction in which the program tries to place a word in the matrix. It can take on the values −1, 0, or 1, where −1 means the word goes upward, 1 means the word goes downward, and 0 means there is no vertical change in position.

HIZ This stands for "horizontal" (the OR proved to be a problem!). It is first used in the subroutine at line 400. It determines the horizontal direction in which the program tries to place a word in the matrix. It also takes on the values −1, 0, and 1, but this time −1 means the word goes left to right, 0 means no change, and 1 means right to left. Note that the variables VERT and HIZ are used together to indicate the eight possible directions mentioned in the program description above. The one combination not used is when VERT=0 and HIZ=0, which would mean that the word wasn't going in any direction.

S This variable is used as a flag returned by the subroutine at line 600. It ends up with the value 1 if the word has been successfully placed in the matrix and the value 0 if it does not fit in the position and direction tentatively selected by the computer.

LINES OF SPECIAL INTEREST

30–60: Read the words from DATA statements into the array WURD$. The data must be restored and the process repeated at the beginning of each round, because as words are selected for the matrix they are removed from WURD$.

70–110: Clear out the matrix MAT$.

150: A number is randomly selected and stored in the variable X (not to be confused with X$). This number represents the subscript in array WURD$ of a word that is yet to be used. When the index I has the value 1, the number selected can range from 1 to 50. When I is equal to 2, this number can range from 1 to 49, and so on.

170: The selected word is replaced by the word with the highest subscript among those not yet selected. (This is similar to the method we used in "Now You See It..." to fill the array used in that program.)

190: Prints the sequential number of the word just placed in the matrix.

220–240: Wait for either the F1 or the F7 key to be pressed. If F7 is pressed, control passes to line 270. If F1 is pressed, control is passed to line 250. (Don't confuse the control character for F7 with that for the color light green, which is never used in this text.)

260: Since at this point the F1 key has already been pressed and the hidden words have already been uncovered, the program waits only for the F7 key to be pressed.

Subroutine

400–550: This subroutine tries to place a word in the matrix until it succeeds. It does this by repeatedly selecting starting positions and directions and calling the subroutine at line 600. When the subroutine at line 600 finally returns an affirmative code (S = 1), the word is actually placed in the matrix.

420–430: Select a row from 0 to 23 and a column from 0 to 39 as a starting position for the word.

450–480: If the randomly selected value for VERT is 0, we want to avoid selecting 0 for HIZ so we select only the numbers −1 or 1 (this is done in line 480). If VERT is not equal to 0, we may choose any of the three possible values for HIZ (line 460).

530: Here we see the significance of the values assigned to VERT and HIZ. Depending upon the direction in which we are going, we increment or decrement ROW and/or COL by 1 each time we place a character in the matrix.

Subroutine

600–730: Checks to see if the word in X$ fits in the selected position and direction of the matrix MAT$. It first makes sure that the word will not "fall off" the edge of the matrix. Then, if two words are found to overlap, the subroutine checks to see if their common letter is indeed the same.

620–650: These lines figure out whether the word will be contained within the confines of the matrix. If it cannot, then the subroutine immediately returns. (Because of line 610, S returns with a value of 0, indicating that the word did not fit.)

680: If the current space in the matrix is unoccupied, everything is OK and the next space is checked.

690: If this line is reached, it means that the current space is occupied. In that case, the program checks to see whether the character in this space is the same as the one that would be placed there from the current word (that is, a legal overlap). If not, the subroutine returns (with S = 0).

720–730: If these lines are reached, no mishaps have occurred. Therefore, S is set to 1 and the subroutine returns.

Subroutine

800–890: This subroutine places the letters on the screen. For any location in the matrix that is empty, the corresponding space on the screen is filled with a randomly selected character.

850: If this line is reached, the current location in the matrix is empty. A letter is then randomly selected by choosing a number between 0 and 25, adding the value 65 to that number to generate the ASCII value of a letter, and applying this number to the CHR$ function to produce the corresponding letter.

Subroutine

900–990: This subroutine highlights the words that are hidden on the screen, in the event that F1 was pressed.

950: If this line is reached, the current character on the screen is not part of a word and so the cursor is moved 1 space to the right, pointing to the next character.

960: The current character is printed in reversed graphics because it is part of a word. Because line 910 turns on reversed graphics, all that is necessary to do here is to print the character.

990: The first control character in the literal is RVS OFF. Be care-

ful, it looks a lot like the symbol for F3. F3 is seldom used in these programs, and we note it when it is.

1000–1170: These are the DATA statements containing the available words. They are placed at the end of the program so that more words may be added without having to renumber the statements.

SUGGESTED ENHANCEMENTS

1. If the player thinks he has spotted a word on the screen, he should be allowed to type it in; if it is indeed one of the words, it alone should be printed in reversed graphics.
2. Points may be awarded based on how many correct words the player is able to detect. This can be further improved by basing the points on the length of the successfully spotted words.
3. The words in the DATA statements may be changed and may be selected based upon the difficulty level desired.
4. The number of words used may be larger than fifty. However, if this is done, be sure to make the necessary changes in lines 20, 40, 120, 130, 150, and 170. Also, be aware that if the program is asked to place a large number of long words in the matrix, it may get caught in a situation where there simply isn't enough room to contain the words. What will happen is that the subroutine at line 400 will go into an infinite loop in which it keeps trying random starting positions and directions, none of which will succeed. One solution to this dilemma is to modify this subroutine so that it "gives up" after a certain number of tries and prints a message telling the user that his request cannot be fulfilled.
5. The game can be made more difficult simply by including a larger number of short words.

CHAPTER 3

ELAPSED DAYS

Although not strictly a game, this program has a considerable amount of utilitarian value. It permits the user to input two dates, the computer immediately calculating the number of elapsed days between those two dates. This information is often useful for investment analysis where interest is paid based on the number of days the principal is invested. The program expects the two dates to be typed in the standard American form: MM/DD/YYYY, where MM refers to the month (1–12), DD to the day of the month (1–31) and YYYY to the year in question (all four digits).

The program checks to make sure that the two dates are valid—that is to say, that no month has a value less than 1 or greater than 12, no day falls outside of the range 1 through 31, or no year consists of fewer than four digits. Furthermore, since the current Gregorian calendar starts at the year 1583, a test is made to ensure that the year in question does not precede 1583. Should the first date be later in time than the second, the elapsed days are still computed but a cautionary message is displayed to bring the user's attention to this fact. All leading zeroes may be omitted when inputting data. However, each date must contain two slashes in the appropriate locations since these are tested for as well. Adjustment is automatically made for any intervening leap years.

```
10 REM ELAPSED DAYS
20 DIM DAYS(12)
30 FOR I=1 TO 12:READ DAYS(I):NEXT I
40 DATA 31,28,31,30,31,30
50 DATA 31,31,30,31,30,31
60 PRINT:INPUT "ENTER 1ST DATE
   (MM/DD/YYYY)";A$
70 S$=A$:GOSUB 300
80 IF S>0 THEN 100
90 PRINT "FIRST DATE INVALID":GOTO 60
100 A=S
110 PRINT:INPUT "ENTER 2ND DATE
    (MM/DD/YYYY)";B$
120 S$=B$:GOSUB 300
130 IF S>0 THEN 150
140 PRINT "SECOND DATE INVALID":GOTO 110
150 B=S
160 IF A<=B THEN 190
170 PRINT "WARNING: DATES REVERSED"
180 S=A: A=B: B=S
190 PRINT:PRINT:PRINT "THERE ARE";B-A;"DAYS"
200 PRINT "BETWEEN ";A$;" AND ";B$
210 PRINT:PRINT
220 PRINT "DO YOU HAVE MORE DATES (Y OR N)?";
230 GET R$:IF R$="" THEN 230
240 IF R$<>"Y" AND R$<>"N" THEN 230
250 PRINT R$:PRINT:PRINT
260 IF R$="Y" THEN 60
270 END
300 REM CHANGE DATE TO ABSOLUTE DAYS
310 S=0:I=1
320 IF LEN(S$)>10 OR LEN(S$)<8 THEN 590
330 IF MID$(S$,I,1)<"0" OR MID$(S$,I,1)>"9"
    THEN 590
340 M=ASC(MID$(S$,I,1))-48
350 I=I+1
360 IF MID$(S$,I,1)="/" THEN 410
370 IF MID$(S$,I,1)<"0" OR MID$(S$,I,1)>"9"
    THEN 590
```

```
380 M=M*10+ASC(MID$(S$,I,1))-48
390 I=I+1
400 IF MID$(S$,I,1)<>"/" THEN 590
410 I=I+1
420 IF MID$(S$,I,1)<"0" OR MID$(S$,I,1)>"9"
    THEN 590
430 D=ASC(MID$(S$,I,1))-48
440 I=I+1
450 IF MID$(S$,I,1)="/" THEN 500
460 IF MID$(S$,I,1)<"0" OR MID$(S$,I,1)>"9"
    THEN 590
470 D=D*10+ASC(MID$(S$,I,1))-48
480 I=I+1
490 IF MID$(S$,I,1)<>"/" THEN 590
500 I=I+1
510 IF LEN(S$)-I<>3 THEN 590
520 Y=0
530 FOR J=I TO I+3
540 IF MID$(S$,J,1)<"0" OR MID$(S$,J,1)>"9"
    THEN 590
550 Y=Y*10+ASC(MID$(S$,J,1))-48
560 NEXT J
570 IF M<1 OR M>12 OR D<1 OR Y<1583 THEN 590
580 GOSUB 600
590 RETURN
600 REM CALCULATE DAYS (CHECK LEAP YR.)
610 LEAP=(INT(Y/4)*4=Y AND (INT(Y/100)*100<>Y
    OR INT(Y/400)*400=Y))
620 DAYS(2)=28
630 IF LEAP THEN DAYS(2)=29
640 IF D>DAYS(M) THEN 700
650 S=D+365*Y+INT((Y-1)/4)-INT((Y-1)/100)
    +INT((Y-1)/400)
660 IF M=1 THEN 700
670 FOR I=1 TO M-1
680 S=S+DAYS(I)
690 NEXT I
700 RETURN
```

GLOSSARY OF VARIABLES

DAYS: This array of 12 elements holds the number of days contained in each month.

S$ String variable used to pass a date to the subroutine at line 300.

S Used by the subroutine at line 300 to return the numeric equivalent of the date passed in S$.

A$ Holds the inputted first date in string form.

A Holds the numeric value of the first date.

B$ Holds the inputted second date in string form.

B Holds the numeric value of the second date.

M Holds the numeric month value of the date being processed by the subroutine at line 300.

D Holds the numeric day of the date being processed by the subroutine in line 300.

Y Holds the numeric year of the date being processed by the subroutine in line 300.

I In the subroutine at line 300 this variable points to the character in S$ currently being processed.

LINES OF SPECIAL INTEREST

20–50: Establish array DAYS, which, for each month, contains the number of days contained by that month.

60–70: Input the first date as a string and pass it to the subroutine at line 300 for conversion to a number. The reason for this approach is to provide for the inclusion of the 2 slashes, which are, of course, nonnumeric.

110–150: Perform for the second date what lines 60–100 do for the first date, except that the numerical equivalent of the date is assigned to the variable B.

180: In case the two dates were entered in the wrong order, the values of A and B are exchanged. Note that here the variable S is being used only as a temporary storage area.

190–210: Print the number of elapsed days by merely subtracting A from B.

Subroutine

300–590: This subroutine converts the date passed in the variable S$ to a numeric value representing the absolute number of days from the beginning of the common calendar, namely the equivalent of

January 1 of the year 0001. It also does a character-by-character analysis of the string to ensure that the format is correct in all respects.

320: This is an attempt to save time by testing immediately whether the length of S$ is within the correct bounds.

330–350: If the first character of S$ is a digit, then its numeric equivalent is assigned to variable M. The variable I is then incremented to point to the next character in S$.

360–390: If the current character of S$ is a slash, control skips to line 410 with the variable M containing the value representing the month. Otherwise, this character should be a digit; if it is, it is also converted to a numerical value and the value of variable M is adjusted so that it now contains the value of the month.

400–410: Test that the current character of S$ is a slash.

420–500: These lines do for the variable D what lines 330–410 do for the variable M.

510: Ensures that there are four characters remaining to be checked in S$.

520–560: Examines the four remaining characters and, if they are digits, converts them to their corresponding numbers and adds them to the adjusted value of variable Y, so that Y winds up containing the value of the year.

570: Checks to determine that the month, date, and year are within their proper ranges. (The upper limit for the date is not yet checked; this is done later in subroutine 600.)

580: Goes to subroutine 600 to compute the actual number of elapsed days and to determine if the year in question is a leap year.

Subroutine

600–700: This checks for a leap year and calculates the final value for S based on M, D, and Y.

610: The variable LEAP is assigned the Boolean value "true" if Y is a leap year. A year is a leap year if it is evenly divisible by 4; the only exceptions to this rule are century years (years that are evenly divisible by 100). In order for century years to be leap years they have to be evenly divisible by 400.

620–630: Set the number of days in February, depending upon whether the year in question is a leap year or not. Notice that since LEAP already has a value of "true" or "false," it is sufficient to say "IF LEAP THEN..." If, indeed, the year in question is a leap year, the value of LEAP is "true," and the THEN clause is executed. If LEAP is "false," the THEN clause is ignored, and control resumes on the next line.

640: Checks to ensure that the variable D is not greater than the number of days in the month.

650–690: Assign to S the absolute number of days that have elapsed up to the given date. Notice that, once again, leap years are taken into account. Lines 660–690 add in the number of days in the months preceding the given date.

SUGGESTED ENHANCEMENTS

1. It might be desired to print out if either of the two dates in question fall on leap years. This information is available in line 610, where the variable LEAP is assigned the value "true" if the year is a leap year.
2. The days of the week on which the dates fall can also be printed out. See "Print-a-Month" (Chapter 21) for details on how this may be done.

CHAPTER 4

HAILSTONES

Even if you're not a mathematician, you will appreciate that to get to a given destination by going three steps forward and two steps back is not the most efficient way to make progress. It seems fairly clear, however, that you will eventually reach the destination. Oddly enough, there is a mathematical problem that seems to suggest that this may not always be the case.

What we are talking about is a simple but nevertheless extremely interesting problem of mathematics which may be stated as follows. Given any positive integer (let it be called N), operate repeatedly on N in the following manner: If N is odd, multiply it by 3 and add 1, using the result as the new value of N (in BASIC this may be written N=N * 3+1); if N is even, replace it with N / 2 (in other words, it may be written as N=N/2). The question that arises is, given any value of N, when it is treated in the manner described above, what happens to N? If you were to take the number 7, for example, and try it out for yourself, you will find that it eventually reaches the number 1 because 7 is odd, and multiplying 7 by 3 and adding 1, N becomes 22. Since 22 is even, it is divided by 2 to yield 11. Now 11, being odd, is also multiplied by 3 with 1 added, giving 34. This in turn

becomes 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, and finally 1. If we were to start with 11, clearly this would also become 1, since 11 is one of the numbers in the above series.

It has been conjectured that all positive numbers when treated in this fashion eventually converge to 1. The fact of the matter is that nobody in the world has yet proved that this is the case, nor has it been disproved. All that we can be sure of is that nobody has yet found a number that does not converge to 1. Armed with your trusty Commodore 64 computer and the program below, you can try to find a number that does not transform to 1. Should you be successful, you will become world famous, and perhaps even rich. When you collect your prize, don't forget, at least, to acknowledge your Commodore 64 computer.

To run the program, simply type in any positive integer of your choice. As soon as it passes the validation test, it is operated upon in the manner described above, the result of each operation being displayed on the screen for your personal observation, together with the sequential operation count. Whenever the number decreases (is divided by 2), the computer emits a low tone. When the number increases (is multiplied by 3 and has 1 added), a high tone is sounded. If the number you have selected behaves in the manner suggested by the conjecture—that is to say, it eventually converges to 1—you are so notified and told exactly how many operations were required to do so. This announcement is accompanied by a melodic overture and you are asked if you would like to try another number. A number that behaves in a most curious and interesting fashion is the number 27. Why don't you try it yourself and see exactly what happens. You might be in for a surprise.

Incidentally, this is the first of our programs that uses the PEEK and POKE commands. They are used here to produce the sounds.

In case you are wondering why we called this program "Hailstones," it is a name suggested in an article by Brian Hayes, published in the prestigious journal *Scientific American* dated January 1984. If you want to know more about the history of this mathematical problem, and all the hypotheses relating to it, you might be interested in reading this excellent article for yourself.

```
10 REM HAILSTONES
20 POKE 54296,15
30 PRINT
40 INPUT "PICK A NUMBER, ANY NUMBER";N
50 IF N=INT(N) AND N>0 THEN 80
60 PRINT:PRINT "NO, NOT THAT NUMBER!"
70 GOTO 30
80 Q=N:IT=0
90 IF Q=1 THEN 180
100 IT=IT+1:EVEN=(Q/2=INT(Q/2))
110 IF EVEN THEN 140
120 Q=Q*3+1
130 GOTO 150
140 Q=Q/2
150 PRINT "OPERATION #";IT;TAB(26);Q
160 GOSUB 300
170 GOTO 90
180 PRINT:PRINT "YES,";N;
190 PRINT "GOES TO 1 IN";IT;"OPERATIONS!"
200 GOSUB 400
210 PRINT:PRINT "ANOTHER (Y OR N)? ";
220 GET A$:IF A$<>"Y" AND A$<>"N" THEN 220
230 PRINT A$
240 IF A$="Y" THEN 30
250 POKE 54296,0
260 PRINT:PRINT "FINISHED"
270 END
300 REM SOUND
310 FR=17167:IF EVEN THEN FR=4291
320 POKE 54272,FR AND 255
330 POKE 54273,INT(FR/256)
340 POKE 54277,34:POKE 54278,34
350 POKE 54276,17
360 FOR I=1 TO 200:NEXT I
370 POKE 54276,16
380 RETURN
400 REM SUCCESS
410 HI=16:LO=195
```

```
420 POKE 54277,34:POKE 54278,34
430 POKE 54272,LO:POKE 54273,HI
440 POKE 54276,17
450 FOR I=1 TO 6
460 POKE 54276,16
470 POKE 54276,17
480 LO=LO*9
490 IF LO<256 THEN 510
500 LO=LO-256:HI=HI+1:GOTO 490
510 POKE 54272,LO:POKE 54273,HI
520 NEXT I
530 POKE 54276,16
540 FOR I=1 TO 200:NEXT I
560 POKE 54272,15:POKE 54273,67
570 POKE 54276,17
580 FOR I=1 TO 20:NEXT I
590 POKE 54276,16
600 RETURN
```

GLOSSARY OF VARIABLES

N	The inputted number to be operated upon.
Q	The original number N is stored in Q and the copy in Q is actually operated upon.
IT	Counts the number of iterations (or operations) that the value in Q undergoes.
EVEN	A Boolean value that is "true" (nonzero) if Q is currently an even number.
FR	Used in the subroutine at line 300 to hold the frequency of the note to be produced by the sound chip. We could not use either the name FREQ or NOTE because both of these names contain a keyword. Do you know what they are?
HI	Used in the subroutine at line 400, this variable holds the upper 8 bits of the frequencies to be played by the sound chip.
LO	Used in the subroutine at line 400, this variable holds the lower 8 bits of the frequencies to be played by the sound chip.

LINES OF SPECIAL INTEREST

20: Turns on the sound chip.
100: Calculates the value of EVEN; if Q/2 is an integer, then Q is even. Note how the equal sign within the parentheses is used as a logical operator rather than for assignment or for comparison.
110–140: These are the lines that perform the crucial operations on Q.
250: Turns off the sound chip.

Subroutine
300–380: Plays a note of the appropriate pitch for each new value of Q. All the POKE statements in this subroutine send information to the sound chip. For a detailed explanation on the roles played by each of these locations, you are respectfully referred to the official Commodore *Programmer's Reference Guide*.

310: Selects the correct frequency based on the value of EVEN.
320–330: Split the variable FR into its upper and lower halves.

Subroutine
400–600: Plays a little melody upon reaching 1.

450–520: Increment the frequency being played.
560: Plays the final "sting" note.

SUGGESTED ENHANCEMENTS

1. Instead of displaying all the intermediate results, confine the output to the final line, specifying how many calculations were required to converge to 1. Perhaps you might want to print out every tenth line instead. Both these modifications will speed up the program.
2. For any value of N inputted, print out the highest value reached before convergence to 1 is achieved. For the example quoted above, where N was set to 7, this would be 52.
3. Insert a FOR/NEXT loop going, say, from 1 to 10,000,000, to determine the number of operations that are required for convergence to 1 for each of those numbers.
4. Calculate for any given value of N the percentage of odd and even numbers encountered along the way to convergence to 1.

CHAPTER 5

TYPING TEST

This is a game intended to increase your skills at typing. The game starts out by asking you how many words a minute you would like. Any response equal to or greater than 1 is accepted, so long as it is an integer. It is recommended that until you have had an opportunity to practice a little you select a slow speed, some number around 10 or 20. As your prowess increases you can attempt higher speeds.

Once the speed has been selected, a random word from a pool of fifty is displayed in the middle of the screen. You are then required to type in the word correctly within a time limit that is established by the computer based on the length of the word and your selected speed. If you type it in correctly within the allotted time limit, 1 is added to your score and a saluting beep is sounded. An error in typing or a time-limit violation adds 1 to the number of attempts that failed and produces an appropriate sound. In the event that you wish to correct a mistyped character, all you need do is press the left-pointing arrow key, the one located at the extreme left of the top row of keys. At any time that you wish to terminate the session, press the F1 key. At that point the screen clears and displays a summary of your efforts, indicating how many you got right, how many wrong, and their respective percentages.

```
10 REM TYPING TEST
20 POKE 53281,8:CNT=0:YES=0
30 INPUT"■HOW MANY WORDS PER MINUTE";N
40 IF N<>INT(N) OR N<1 THEN 30
50 RESTORE:W=INT(RND(0)*50)+1
60 FOR I=1 TO W
70 READ WURD$
80 NEXT I
90 L=0:ANSWER$="":CNT=CNT+1
100 PRINT "■■■■■■■■■■■";
    TAB(15);WURD$
110 POKE 198,0
120 POKE 209,159:POKE 210,5
130 LIM=TIME+660*LEN(WURD$)/N+20
140 PRINT "■ ■■";
150 GET A$:IF A$<>"" THEN 210
160 IF TIME<LIM THEN 150
170 GOSUB 400
180 PRINT:PRINT:PRINT"SORRY, TIME'S UP!"
190 FOR I=1 TO 400:NEXT I
200 POKE 54276,32:GOTO 50
210 IF A$="■" THEN 390
220 IF A$=CHR$(13) THEN 300
230 IF A$<>"←" THEN 270
240 IF L=0 THEN 150
250 L=L-1:ANSWER$=LEFT$(ANSWER$,L)
260 PRINT " ■■■";:GOTO 140
270 IF A$<"A" OR A$>"Z" THEN 150
280 PRINT A$;:L=L+1:ANSWER$=ANSWER$+A$
290 GOTO 140
300 PRINT " "
310 IF ANSWER$<>WURD$ THEN 350
320 GOSUB 500:PRINT"CORRECT!":YES=YES+1
330 FOR I=1 TO 400:NEXT I
340 POKE 54276,32:GOTO 50
350 GOSUB 400
360 PRINT:PRINT"SORRY, YOU MISTYPED THE WORD"
370 FOR I=1 TO 400:NEXT I:POKE 54276,32
```

```
380 FOR I=1 TO 2000:NEXT I:GOTO 50
390 POKE 54296,0:GOSUB 600:END
400 REM BADBEEP
410 POKE 54272,132
420 POKE 54273,3
430 POKE 54277,160
440 POKE 54278,250
450 POKE 54296,15
460 POKE 54276,33
470 RETURN
500 REM GOODBEEP
510 POKE 54272,32
520 POKE 54273,78
530 POKE 54277,34
540 POKE 54278,138
550 POKE 54296,15
560 POKE 54276,33
570 RETURN
600 REM PERFORMANCE RATING
610 POKE 53281,6
620 PRINT "▼■■■■■■";TAB(14);"◆YOUR
    SCORES:"
630 PRINT:PRINT
640 PRINT TAB(9);"CORRECT      INCORRECT"
650 PRINT TAB(9);"--------------------"
660 PRINT:PRINT:PRINT TAB(10);YES;
670 PRINT TAB(23);(CNT-1)-YES
680 FINE=INT(YES/(CNT-1)*100)
690 PRINT:PRINT TAB(9);"(";FINE;"■■% )";
700 PRINT TAB(22);"(";100-FINE;"■■% )"
710 RETURN
800 REM WORDS
810 DATA"BENEFIT","INSIST","PLEASURE"
820 DATA"STORM","ACRONYM","TRAPEZE"
830 DATA"WORLDLY","FUNGUS","MARKET"
840 DATA"YESTERDAY","GUIDANCE","YEARN"
850 DATA"ENGAGE","RECLAIM","KNOWLEDGE"
860 DATA"COLONY","UTILITY","JOKE"
870 DATA"DRIVER","REMAINDER","HYDROGEN"
```

```
880 DATA"OCTOPUS","TECHNICAL","LAUGH"
890 DATA"QUIVER","VOLTAGE","NIMBLE"
900 DATA"FOLIAGE","AMBITION","VALVE"
910 DATA"GEOMETRY","LOCATION","GLACIER"
920 DATA"OUTRAGEOUS","WARRANT","KICK"
930 DATA"CIVILIZE","MATERNITY","HEART"
940 DATA"DISARRAY","ZOMBIE","BLUEPRINT"
950 DATA"MILDEW","IMPRESS","JINGLE"
960 DATA"PERFECT","EXCESSIVE","QUOTA"
970 DATA"NAVIGATE","SEPARATE"
```

GLOSSARY OF VARIABLES

WURD$ Holds the word that has been randomly selected by the computer from the pool of 50 words.

N The inputted typing speed.

CNT Holds the total number of words that the user tried before quitting.

YES The number of words that were correctly typed and in time.

ANSWER$ Holds the word typed in by the player.

L Holds the length of the word as typed in by the player.

LIM This is the time limit that the player has to input the word. (It is measured in increments of 1/60 of a second, since these are the units used by the built-in TIME function.)

LINES OF SPECIAL INTEREST

20: Here we encounter a new POKE statement. The memory location referenced, 53281, controls the background color. This is accomplished by poking a number between 0 and 15 into that location. We have elected to use the number 8, which sets the screen color to orange.

30: Note that here the first character of the literal in the INPUT statement changes the character color to brown.

50–80: A word is randomly selected from the pool of DATA statements by continually reading a word into WURD$ for some random number of times between 1 and 50. When the reading stops, the word that is currently in WURD$ is used as the selected word.

Notice that the RND function is not used directly in the FOR statement but rather is used outside of the loop.

110: Another POKE; this time location 198 is set to 0. Location 198 indicates the number of characters in the keyboard buffer. If the player takes too long to type in a word, he is interrupted, but any keys pressed after that point will still enter the buffer. These characters would be erroneously accessed by subsequent GET statements if the buffer were not cleared at the beginning of every round.

120: Still more POKEs; locations 209–210 together form a 16-bit number specifying the address of the current cursor position on the screen. What we mean by "address" is this: The character positions on the screen are represented by consecutive locations in the Commodore's memory. These locations normally start at address 1024. The characters on the screen are represented in screen memory in what is called screen display code. These values differ from ASCII values and both can be found in the *Programmer's Reference Guide*. Given any row (0–24) and column (0–39) on the screen, we can determine the corresponding memory location by multiplying the row by 40, adding to this the column number, and then adding 1024.

Since the resulting address must be divided into two memory locations, the Commodore 64 divides it as follows: The lower 8 bits are placed in the lower-numbered location (209) and the upper 8 bits are stored in the higher-numbered location (210). So, to convert the numbers in line 120 back into an address, we take the contents of 210 (which is 5), multiply it by 256, and add the contents of location 209, which is 159. The resulting number is 1439. To convert this into row and column notation, we first subtract 1024, leaving us with 415. To find the row number, we divide this by 40 and take the integer portion, giving a result of 10. To get the column number, we take the remainder of the previous division, which is 15. The net result of all this is that the POKEs in line 120 reposition the cursor at row 10, column 15.

One final note: This method of positioning the cursor affects only the next line printed. All subsequent lines will be printed where the computer would normally have printed them had we not poked. So the normal practice, if repositioning of the cursor is used at all, is to do so before each PRINT statement.

130: Here we determine the time limit for entering the word. It is based on a formula which takes into account the length of the word and the typing speed requested by the user. Obviously, the longer the word, the more time is allowed; the higher the speed requested, the shorter the time allowed.

140–160: These lines accept the letters typed in by the user. Since the computer does not display a cursor when the program is running, line 140 remedies this by printing a cursor image at the beginning of the field allotted for the user's response. Each time a GET is attempted, the current TIME is compared to the value of LIM. As long as the former is less than the latter, input may continue.

190: This delay loop ensures that the message in line 180 stays on the screen long enough to be read.

200: This POKE ends the tone which was initiated by the subroutine in line 400. Turning it off at this point allows the sound to persist for a short time after returning from the subroutine.

210: This line tests for the pressing of the F1 key, which terminates the session.

220: CHR$(13) is the equivalent of the RETURN key. The user's response is not considered to be over until this key is pressed.

230: If the key pressed is not the backspace, it is assumed for the time being that it is a letter. Otherwise, control passes to line 240.

240–260: These lines remove the last character from ANSWER$. Line 240 makes sure that there is at least one character in ANSWER$. Line 250 decrements the length and removes the last character of ANSWER$. Line 260 blanks out the cursor character and backspaces over to the last character printed on the screen (the one that is to be deleted). Control then returns to line 140, where a cursor is printed in this space.

270–290: Accept the letter just entered. Line 270 ensures that the character is indeed a letter. Line 280 prints the character, adds it to ANSWER$ and increments the length of ANSWER$.

300: Blanks out the last cursor character.

380: An extra long delay is allowed here so the user may see where the word was misspelled.

390: The POKE in this line turns off the sound chip entirely.

Subroutine

400–470: This subroutine produces a raspberry sound to accompany the message stating that the user either misspelled a word or took too long.

Subroutine

500–570: Produces a beep indicating that the user has typed in the word successfully.

Subroutine

600–710: This subroutine displays the player's performance rating at the end of a game. It is identical to the one used in "Now You See It...," with the following three exceptions:

610: This POKE statement restores the screen color to its default value of dark blue.

620: The first character of the second literal restores the character color to light blue.

670–680: We use the value CNT-1 rather than CNT because the word which was interrupted by the pressing of the F1 key has been included in the count.

800–970: These are the DATA statements containing the pool of 50 words.

SUGGESTED ENHANCEMENTS

1. The pool of words can always be changed or its length increased or decreased. (Don't forget to change line 50 if the number is modified.)
2. For those who are not only enterprising but also are blessed with unlimited patience, the program may be modified to determine the player's typing speed, based on how quickly he enters a series of words. A good formula to do this might require considerable thought.

CHAPTER 6

LANDER

In this game the player controls a space vehicle which is trying to land on the planet Commodon. As the game opens up the landing vehicle is seen floating downward in the middle of the screen. In a short while the landing vehicle starts to meet oncoming and randomly located meteors. The ship must steer its way through this barrage without striking any of these meteors, in order to reach the planet's surface. For each row of meteors successfully dodged, the player scores points which are displayed at the top of the screen. An extra bonus is awarded for reaching the planet's surface unscathed.

The game may be played at five levels of difficulty. The higher the level the greater the density of the meteor cloud and the more points are scored. Level 1 is rather easy while level 5 may well prove to be impossible. At the beginning of the game, the player is asked at which level he wishes to start. As each level is successfully completed, the player automatically moves on to the next level. Once level 5 is reached, he remains there for as long as he can last. Each level consists of five "trips" down to the planet's surface. Each trip is longer than the previous one. The trip lengths are 40, 50, 60, 70, and 80, these numbers referring to the number of rows of meteors the spaceship must pass through to reach the planet. When the trip of length 80 is successfully completed, the player is automatically moved up to the next level.

At the beginning of the game, the player has a supply of three ships. Each time a ship collides with a meteor, it is lost with a crashing sound and the player must repeat the trip from the beginning. The game ends when the player loses all the ships. However, each time a level is successfully completed, an extra ship is added to the supply. At the beginning of every trip the program displays the level number, the trip length, and graphically shows how many ships the player has left.

The ship is capable only of movement to the left and to the right. Its movement is controlled by the two cursor keys, located at the extreme right of the bottom row of keys. Instead of performing their normal functions, the up/down cursor key controls the ship's movement to the left, and the left/right key controls movement to the right.

```
10 REM LANDER
20 SHIPS=3:PTS=0:POKE 54296,15
30 PRINT "STARTING LEVEL (1-5)? ";
40 GET A$:IF A$<"1" OR A$>"5" THEN 40
50 PRINT A$
60 LEVEL=ASC(A$)-48
70 FOR TRIP=40 TO 80 STEP 10
80 GOSUB 700
90 COL=20:D=0:GOSUB 500
100 FOR Q=1 TO TRIP
110 GOSUB 300:GOSUB 800
120 D=0
130 A=PEEK(197):IF A<>2 AND A<>7 THEN 160
140 D=2:IF A=7 THEN D=-2
150 IF COL+D<3 OR COL+D>36 THEN D=0
160 S=0:GOSUB 500:IF S<0 THEN 200
170 IF Q>10 THEN PTS=PTS+LEVEL
180 NEXT Q
190 GOTO 230
200 GOSUB 1000
210 SHIPS=SHIPS-1:IF SHIPS=0 THEN 270
220 GOTO 80
230 PTS=PTS+20*LEVEL:GOSUB 800
240 GOSUB 900:NEXT TRIP
```

```
250 IF LEVEL<5 THEN LEVEL=LEVEL+1
260 SHIPS=SHIPS+1:GOTO 70
270 PRINT "▼":GOSUB 800:POKE 214,11
280 PRINT:PRINT TAB(15);"◆GAME OVER"
290 POKE 54296,0:POKE 198,0:END
300 REM DRAW METEORS
310 POKE 214,23:PRINT
320 IF Q>TRIP-11 THEN 410
330 PRINT " ";
340 FOR I=1 TO 38
350 IF INT(RND(0)*(85-LEVEL*15)) THEN 380
360 PRINT "R ▄";
370 GOTO 390
380 PRINT " ";
390 NEXT I
400 GOTO 470
410 PRINT " R";
420 FOR I=1 TO 310:NEXT I
430 IF Q=TRIP-10 THEN 460
440 FOR I=1 TO 38:PRINT " ";:NEXT I
450 GOTO 470
460 FOR I=1 TO 19:PRINT "◢◣";:NEXT I
470 PRINT
480 RETURN
500 REM DRAW SHIP, DETECT COLLISION
510 POKE 214,9:PRINT "▀"
520 PRINT TAB(COL);" "
530 PRINT TAB(COL-2);"     ";
540 COL=COL+D
550 POKE 211,COL:PRINT "▄"
560 S$=" ▒▒▒ ":PRINT TAB(COL-2);"R";
570 FOR I=1 TO 5
580 IF PEEK(1501+COL+I)=32 THEN 600
590 S=-1:PRINT "▒";:GOTO 610
600 PRINT MID$(S$,I,1);
610 NEXT I
620 PRINT "▀"
630 RETURN
700 REM PRINT LEVEL, TRIP, ETC.
```

```
710 PRINT "■■":POKE 214,10:PRINT
720 PRINT " LEVEL =";LEVEL;TAB(23);
730 PRINT "TRIP LENGTH =";TRIP:PRINT
740 FOR I=1 TO SHIPS
750 PRINT " ■■■■■ ";
760 NEXT I
770 FOR I=1 TO 3000:NEXT I
780 PRINT "■":GOSUB 800
790 RETURN
800 REM PRINT SCORE
810 PRINT "■■";TAB(10);PTS
820 RETURN
900 REM SUCCESS SOUND
910 POKE 54272,209:POKE 54273,18
920 POKE 54277,15:POKE 54278,251
930 POKE 54275,4:POKE 54276,65
940 FOR I=1 TO 300:NEXT I
950 POKE 54272,181:POKE 54273,23
960 FOR I=1 TO 50:NEXT I
970 POKE 54276,64
980 FOR I=1 TO 1000:NEXT I
990 RETURN
1000 REM LOSE NOISE
1010 POKE 54273,8
1020 POKE 54277,15:POKE 54278,251
1030 POKE 54276,129
1040 FOR I=1 TO 50:NEXT I
1050 POKE 54276,128
1060 FOR I=1 TO 1500:NEXT I
1070 RETURN
```

GLOSSARY OF VARIABLES

SHIPS	This represents the number of ships available to the player.
PTS	The number of points the player has accumulated.
LEVEL	The level on which the player is currently playing.
TRIP	A FOR/NEXT loop index indicating the length of the current trip.
COL	The number of the column occupied by the ship's center.

D Indicates the direction in which the ship is being moved. It can take on the values −2, 0, or 2. A direction of −2 indicates that the ship is moving to the left, 0 means that it is remaining in the same position, and 2 means that it is moving to the right. The reason for using a unit of 2 rather than 1 is that the ship moves only in units of 2.

Q A loop index indicating the distance or the number of rows that the ship has passed.

S This variable is set to −1 when any part of the spaceship hits a meteor.

LINES OF SPECIAL INTEREST

10: The POKE in this line turns on the sound chip.

130: Here we use a new PEEK. Location 197 turns out to be very useful. It indicates which key on the keyboard is currently being pressed. It differs from the GET statement in many ways. Rather than referring to the keyboard buffer, location 197 reflects which key is being actuated at the instant that the PEEK function is being performed. Also, as long as the key is depressed, its corresponding number is registered in location 197. Therefore, it is not necessary to keep releasing and pressing the key to have it register more than one time. Finally, the contents of location 197 reflect the physical key that is being pressed and not the character; that is to say, if a SHIFT key is used, it has no effect on the number in location 197. Table 2, on the following page, indicates which keys will cause values to appear in location 197.

150: This line keeps the ship from going off the edge of the screen. Notice that since the ship moves in steps of 2, it can actually go to the left no farther than column 4. Notice also that no meteors are produced in columns 0 or 39 and no part of the ship can enter these columns either.

170: This line assures that the player begins scoring points only when he reaches the first row of meteors. Note that the number of points scored is simply the level number.

230: This line calculates the bonus for a successful landing.

250–260: Once the level reaches 5 (good luck!) it is not incremented further, but it is still necessary to go through 5 trips before scoring an extra ship.

270–280: This is another new POKE. Location 214 is used to reposition the cursor to a new row. Unlike the repositioning method

TABLE 2

Key	PEEK(197)	Key	PEEK(197)
(RETURN)	1	E	14
(CRSR up/down)	7	F	21
(CLR/HOME)	51	G	26
(INST/DEL)	0	H	29
(CRSR left/right)	2	I	33
(space bar)	60	J	34
*	49	K	37
+	40	L	42
,	47	M	36
–	43	N	39
.	44	O	38
/	55	P	41
0	35	Q	62
1	56	R	17
2	59	S	13
3	8	T	22
4	11	U	30
5	16	V	31
6	19	W	9
7	24	X	23
8	27	Y	25
9	32	Z	12
:	45	£	48
;	50	↑	54
=	53	←	57
@	46	(RUN/STOP)	63
A	10	(F1/F2)	4
B	28	(F3/F4)	5
C	20	(F5/F6)	6
D	18	(F7/F8)	3

If no key is pressed, PEEK(197) returns 64

using locations 209 and 210, the cursor does not return to its original position after printing. However, there is one disadvantage to this POKE. One PRINT must be executed after the POKE before the cursor is actually positioned to the new line. Unfortunately, the result is that you wind up 1 line lower than the intended line. The POKE in line 270 merely repositions the cursor to row 12 after the first PRINT statement at the beginning of line 280 is executed.

290: Turn off the sound chip and clear out the keyboard buffer.

Subroutine

300-480: This subroutine draws a new row of meteors at the bottom of the screen. Here we take advantage of the automatic scrolling that occurs when information is displayed at the bottom of the screen. When the new row of meteors is drawn, the old meteors are automatically moved up one row. Of course, the spaceship also moves up one row, but this is taken care of in the subroutine in line 500. This subroutine also draws the surface of the planet when it is time for it to approach.

310: The cursor is repositioned to line 24.

320: As the trip nears its end, instead of printing meteors the program prints part of the surface of the planet.

330: This skips over column 0, which is not used.

340–390: Print meteors. If the INT function in line 350 returns a nonzero value, control passes to line 380, where a space is printed. If the resulting value is 0, a meteor is printed. Notice from line 350 that the lower the level, the greater the probability of randomly selecting a nonzero number and therefore the smaller the probability of printing a meteor.

410: The planet is printed in reverse graphics. (Notice that column 0 is again skipped over.)

420: Because less calculation is involved, the time taken to print the planet is shorter than the time taken to print meteors. This delay loop lengthens the time taken to print the planet, thereby maintaining the speed of the game.

430: The first row of the planet's surface is pointy and is therefore printed separately at line 460.

460: The graphics characters used here are shifted pound sign and shifted asterisk.

Subroutine

500–630: Draws the ship and checks for collisions with meteors. Recall that each time a row of meteors is drawn, the ship must be redrawn.

520–530: Erase the old ship. Line 530 prints five spaces; it ends in a semicolon because the top of the new ship goes on the line previously occupied by the bottom of the old ship.

550: Another new POKE; location 211 controls the column of the cursor. Its main advantage is that you can use it to accomplish backward tabbing, which is illegal in BASIC. Note also that the graphics character in the literal is Commodore key-I.

560: The pattern for the bottom of the ship is placed in variable S$. Only those characters which will not coincide with a meteor will be printed. Note that the graphics characters in the first literal are Commodore key-plus.

570–610: The bottom of the ship is the only part that can strike a meteor. Therefore, before printing any part of the bottom, the program must check to see if that space is occupied by a meteor. Line 580 peeks into the appropriate location of screen memory (using a formula based on the 1024 + row * 40 + column principle) to see if there is a space (screen display code 32) there. If so, line 600 prints the appropriate character from S$. If not, line 590 sets S to −1 (indicating a collision) and tabs forward one space to let the meteor show through.

620: Sets the character color to black in preparation for the printing of meteors.

Subroutine

700–790: Prints the player's status at the beginning of each trip.

740–760: For each ship the player has left, print a miniature spaceship symbol. Note that the graphics characters in line 750 are Commodore key-I and Commodore key-pound sign.

770: This delay loop keeps the information on the screen for a while.

Subroutine

800–820: This very simple subroutine prints the player's score. This must be done each time a row of meteors is printed, because otherwise the score would be scrolled off the top of the screen. Notice that the literal in the PRINT statement in line 810 positions the cursor at the top of the screen and changes the character color to cyan.

Subroutine

900–990: Plays a couple of musical notes when the spaceship successfully lands. Lines 910 and 950 play the notes D and F# respectively. A rectangular waveform which has a rather long duration is used.

Subroutine

1000–1070: Produces a crashing noise when the spaceship hits a meteor. This also has a rather long duration.

SUGGESTED ENHANCEMENTS

1. You may want to change the method of scoring. For example, you may decide to create a greater difference between the scores on the lower levels and those on the higher levels.
2. Once you have begun to master the game, you might want to allow for longer trips.
3. You may decide to place the beginning position of the spaceship closer to the bottom of the screen. That would mean that it reaches the meteor sooner and evasive action would have to be taken earlier. Be sure to modify lines 170, 320, 430, and 510.
4. The game may be rewritten to allow for two players to compete. You may want to give each player a different color spaceship and print both scores simultaneously on the screen.
5. You might want to arrange for the meteors to be printed in random colors. See the program "Phone-y Words" for a method of randomly changing character color.

CHAPTER 7

MAGIC SQUARES

Magic squares have been a source of delight for a good many centuries. It is recorded that a well-known German artist, Albrecht Dürer, included a magic square in his famous engraving called "Melencolia." He used a 4×4 square as shown:

16	3	2	13
5	10	11	8
9	6	7	12
4	15	14	1

If you examine this magic square carefully, you will notice that not only are consecutive numbers from 1 to 16 used, but also each of its rows, columns, and diagonals adds up to the same number, 34.

Through the years people have gone to considerable efforts to discover magic squares. Any magic square can always be rotated to give other magic squares. What this program does is to allow the player to input any odd number between 3 and 9. The size is limited to 9 because of the limited space on the screen. The reason that only odd numbers are used is that it allows for a rather

simple formula to generate magic squares whereas generating them for even dimensions is much more complicated.

Once the odd dimension has been typed in it is validated and, if it is found to be within the permitted range, a grid of appropriate size is drawn on the screen. The computer then fills in the boxes of the grid with the numbers 1 through N^2, where N is the odd-numbered dimension of the magic square. As the consecutive numbers are placed into their respective locations in the square, various tones are produced.

The method for placing the numbers in the grid is as follows: Starting in the center column of the top row, the number 1 is placed. The consecutive numbers are then placed moving up one row and to the right one column, before placing each subsequent number. Of course, the edges of the grid are sometimes going to be encountered while this process takes place. To allow for this, the program "wraps around"; that is to say, after placing a number in the topmost row it then places the next number in the bottom row and works its way up again. Similarly, when the rightmost column is reached, the program goes immediately back to the left column and continues moving to the right.

Another problem the program will encounter is reaching a space that has already been filled. In this case, instead of filling the box above and to the right of the previous box, the program fills the box directly below the previous one in the same column. Once the magic square is completed, the user is asked if another one is required.

```
10 REM MAGIC SQUARES
20 POKE 53281,7
30 PRINT "■"
40 PRINT:PRINT "INPUT AN ODD INTEGER 3-9."
50 INPUT "WHAT SIZE, PLEASE";N
60 IF N<>INT(N) OR N=INT(N/2)*2 THEN 40
70 IF N<3 OR N>9 THEN 40
80 GOSUB 300
90 ROW=1:COL=(N-3)/2
100 FOR I=1 TO N↑2
110 ROW=ROW-1:IF ROW<0 THEN ROW=N-1
```

```
120 COL=COL+1:IF COL=N THEN COL=0
130 IF PEEK(CRNER+ROW*80+COL*3+1)=32 THEN 160
140 ROW=ROW+2:IF ROW>=N THEN ROW=ROW-N
150 COL=COL-1:IF COL<0 THEN COL=N-1
160 GOSUB 500
170 NEXT I
180 PRINT:PRINT:PRINT "■ANOTHER (Y OR N)? ";
190 GET A$:IF A$="" THEN 190
200 IF A$<>"Y" AND A$<>"N" THEN 190
210 PRINT A$:PRINT
220 IF A$="Y" THEN 40
230 POKE 53281,6:PRINT "▼◆FINISHED"
240 END
300 REM PRINT GRID
310 R1=12-N:C1=(39-3*N)/2
320 CRNER=1065+40*R1+C1:PRINT "▼←"
330 FOR J=2 TO R1:PRINT:NEXT J
340 PRINT TAB(C1);"┌┬┬";
350 FOR J=2 TO N:PRINT "┬┬┬";:NEXT J
360 PRINT "┐":PRINT TAB(C1);
370 FOR J=1 TO N:PRINT "│  ";:NEXT J
380 PRINT "│"
390 FOR K=2 TO N
400 PRINT TAB(C1);"├┼┼";
410 FOR J=2 TO N:PRINT "┼┼┼";:NEXT J
420 PRINT "┤":PRINT TAB(C1);
430 FOR J=1 TO N:PRINT "│  ";:NEXT J
440 PRINT "│"
450 NEXT K
460 PRINT TAB(C1);"└┴┴";
470 FOR J=2 TO N:PRINT "┴┴┴";:NEXT J
480 PRINT "┘"
490 RETURN
500 REM PUT NUMBER IN GRID
510 B=CRNER+ROW*80+COL*3
520 D1=INT(I/10):D2=I-D1*10
530 D1=D1+48:D2=D2+48
540 IF D1=48 THEN D1=32
550 POKE B,D1:POKE B+1,D2
```

```
560 POKE B+54272,0:POKE B+54273,0
570 GOSUB 600
580 RETURN
600 REM TONE
610 POKE 54272,(B*10) AND 255
620 POKE 54273,INT(B*10/256)
630 POKE 54277,34
640 POKE 54278,34
650 POKE 54296,15
660 POKE 54276,33
670 FOR J=1 TO 10:NEXT J
680 POKE 54276,0:POKE 54296,0
690 RETURN
```

GLOSSARY OF VARIABLES

N The odd dimension of the square.

ROW The row of the grid currently being filled (numbered 0 through N − 1).

COL The column of the grid currently being filled, also numbered 0 through N − 1.

I This is a FOR/NEXT loop index whose values go from 1 to N^2. It represents the number that is currently being placed in the grid.

CRNER This variable marks the location of the upper lefthand corner of the grid in screen memory. Specifically, it is the address to be occupied by the 10s digit of the number in the upper lefthand box. Its value is calculated by the subroutine in line 300. As you may well have guessed, the reason this variable isn't named CORNER is that this contains the Commodore BASIC keyword OR.

B Generated in the subroutine in line 500, this variable points to the location in screen memory of the box currently being filled.

D1 The 10s digit of the number currently being placed in the grid.

D2 The 1s digit of the number currently being placed in the grid.

LINES OF SPECIAL INTEREST

20: Changes the screen color to yellow.

90: We set ROW and COL to point to the box below and to the left of the one we initially want to fill. That is because ROW will be decremented and COL incremented before they are used.

110–120: The values of ROW and COL are changed to point to the next space to be filled. If necessary, wraparound is performed.

130: Checks to see if this space is blank (screen display code of 32). In calculating the screen memory address, ROW is multiplied by 80 rather than 40 because rows in the grid are actually two screen rows apart. COL is multiplied by 3 because grid columns are actually three screen columns apart. The number 1 also is added because the program is checking the 1s column of the number (remember that CRNER points to the 10s digit).

140–150: In the event that the box is already filled, the values of ROW and COL are adjusted to point instead to the box below the last filled box.

230: Screen color is changed back to dark blue; character color is changed back to light blue.

Subroutine

300–490: Prints a grid of appropriate size on the screen; also calculates a value for CRNER.

310: R1 is the row at which the top line of the grid is drawn. C1 is the column at which the leftmost line of the grid is drawn.

320: In calculating the value for CRNER we use 1065 instead of 1024, an addition of 41, because the location indicated by CRNER is one row below and one column to the right of the corner of the grid itself.

330: This loop spaces down to the appropriate row. It starts at 2 instead of 1 to indicate that the PRINT at line 320 has already spaced down one row.

340: The graphics characters used here are Commodore key-A and shift-C (or shift-asterisk).

350: We could just as well have said "2 to N − 1," but since the actual value of J does not matter, this method saves us one subtraction operation. Although such efficiency is really not necessary here, it is good to keep such tricks in mind for those occasions in which speed is of the essence, such as in action games. Note that

the graphics characters used are Commodore key-R and shift-C (or shift-asterisk).

360: The graphics character used here is Commodore key-S.

370–380: The graphics character used here is shift-B (or shift-minus)

400: The graphics characters used here are Commodore key-Q and shift-C (or shift-asterisk).

410: The graphics characters used in this line are shift-plus and shift-C (or shift-asterisk).

420: The graphics character used here is Commodore key-W.

430–440: The graphics character used here is, once again, shift-B (or shift-minus).

460: The graphics characters here are Commodore key-Z and shift-C (or shift-asterisk).

470: The graphics characters used here are Commodore key-E and shift-C (or shift-asterisk).

480: The graphics character used here is Commodore key-X.

Subroutine

500–580: Places the value of I in the box of the grid indicated by ROW and COL.

510: The calculation here is similar to that in line 130 except that 1 is not added because we want to point to the 10s position.

540: If the digit in the 10s position is a 0, it is replaced by a space (blank).

550–560: This is the first time that we are poking to screen memory. When this is done it is also necessary to poke to color memory. The locations in color memory control the color of the characters in each location of the screen. There is therefore a one-to-one correspondence between the locations in color memory and screen memory. Color memory begins at location 55296, which is 54272 bytes from the beginning of screen memory. Therefore, if we know a location in screen memory, we can find the corresponding location in color memory by adding 54272 to the address. The values placed in color memory are the same as those stored in location 53281: 0 through 15. In line 560 we are coloring both digits black. (If the 10s position is occupied by a blank, it will be transparent regardless of the contents of color memory.)

Subroutine

600–690: Produces a tone for each number inserted in the grid. Note that in lines 610 and 620 that the frequency used is 10 times the value of B.

SUGGESTED ENHANCEMENTS

1. With a little clever programming it is possible to print the row, column, and diagonal totals to convince the user that they all agree and that the square is indeed a magic square.
2. You may be inclined to delve into the mysteries of an even-dimensioned magic square. We warn you, however, that this is not so easy.

CHAPTER 8

PENETRATE

The time has arrived for a little action. In this game you control a mighty cannon that shoots from the left side of the screen into a defensive gate that opens at its middle at random intervals. The cannon is directed at the opening all the time and your task is to fire the cannon at the appropriate times by hitting the space bar. The object of the game is to somehow manage to penetrate the opening of the gate. At the beginning of play you are given 100 points. Each time the cannonball makes it through the gate, you are awarded 10 points. However, if the gates close on your cannonball before or during penetration, you lose 10 points. At all times the score is posted in the upper lefthand side of the screen. If and when the score reaches 0, the game is automatically terminated.

```
10 REM PENETRATE
20 POKE 53281,7:POKE 54296,15
30 GOSUB 300
40 PTS=100:PRINT "£":GOSUB 500
50 SW=0
60 GET A$
```

```
70 IF A$=" " THEN 120
80 IF TIME<LIM THEN 60
90 IF SW=0 THEN 110
100 GOSUB 600:SW=0:GOTO 60
110 GOSUB 700:SW=1:GOTO 60
120 SPOT=1510:POKE SPOT,81:GOSUB 800
130 FOR I=1 TO 27
140 POKE SPOT,32:SPOT=SPOT+1
150 IF I=5 THEN POKE 54276,128
160 IF PEEK(SPOT)<>32 THEN 260
170 POKE SPOT,81:POKE SPOT+54272,0
180 IF TIME<LIM THEN 220
190 IF SW=0 THEN 210
200 GOSUB 600:SW=0:GOTO 220
210 GOSUB 700:SW=1:IF I>19 THEN 260
220 NEXT I
230 POKE SPOT,32
240 PTS=PTS+10:GOSUB 500
250 GOTO 60
260 PTS=PTS-10:GOSUB 500
270 IF PTS>0 THEN 60
280 POKE 53281,6:POKE 54296,0
290 PRINT "■■GAME OVER":END
300 REM SET UP
310 PRINT "■"
320 FOR ROW=8 TO 16
330 FOR COL=26 TO 33
340 SPOT=1024+ROW*40+COL
350 POKE SPOT,160:POKE SPOT+54272,0
360 NEXT COL
370 NEXT ROW
380 FOR I=55776 TO 55815
390 POKE I,0
400 NEXT I
410 FOR I=1504 TO 1509
420 POKE I,160
430 NEXT I
440 GOSUB 600
```

```
450 RETURN
500 REM DISPLAY POINTS #
510 PRINT "■";
520 IF PTS<1000 THEN PRINT "0";
530 IF PTS<100 THEN PRINT "0";
540 IF PTS<10  THEN PRINT "0";
550 PRINT MID$(STR$(PTS),2)
560 RETURN
600 REM OPEN GATES
610 FOR J=1530 TO 1537
620 POKE J,32
630 NEXT J
640 LIM=TIME+INT(RND(0)*4)*30+30
650 RETURN
700 REM SHUT GATES
710 FOR J=1530 TO 1537
720 POKE J,160
730 NEXT J
740 LIM=TIME+INT(RND(0)*4)*60+60
750 RETURN
800 REM NOISE
810 POKE 54273,16
820 POKE 54277,0:POKE 54278,248
830 POKE 54276,129
840 RETURN
```

GLOSSARY OF VARIABLES

PTS The current number of points the player has.

SW A variable whose value indicates whether the gate is open or closed. A value of 0 indicates "open" while 1 indicates "closed."

LIM This variable, whose value is set in the subroutines at lines 600 and 700, determines the time at which the gate either opens or closes.

SPOT In general, this variable indicates the location in screen memory of the cannonball or the location to which it is moving.

LINES OF SPECIAL INTEREST

40: Game starts out at 100 points.
50: Gates are initially opened.
60–80: These lines wait until the space bar is pressed or until the time indicated by LIM is reached, whichever comes first.
90–110: The time indicated by LIM was reached. Either open or close the gates, as appropriate, and continue waiting for the space bar to be pressed.
120: The space bar was pressed and the cannonball is fired. It is placed in its starting location in screen memory, as indicated by SPOT. Note that 81 is the screen display code for the solid circle.
130–220: Move the cannonball across the screen. The loop index I indicates how many spaces the cannonball has traveled from the cannon.
140: Remove the cannonball from the current location and increment SPOT to point to the next location.
150: After the cannonball has moved 5 spaces, the shooting sound is terminated.
160–170: If the next location is vacant, the cannonball is placed there. Notice that the second POKE in line 170 colors it black.
190–210: The time specified by LIM has been reached. Either open or close the gates. The conditional test in line 210 determines whether the gates are closing on the cannonball. If so, control passes on to line 260.
230–250: The gates have not closed on the cannonball. The player scores 10 points and control passes back to line 60 to again wait for the space bar to be pressed.
260–270: The cannonball has run into a closed gate or the gate closed upon it during penetration. The value of PTS is decremented and if the score has not yet reached 0, control goes back to line 60.
280–290: The score has reached 0 so the game is terminated.

Subroutine
300–450: Sets up the screen initially.

320–370: The loop indexes ROW and COL indicate the locations in screen memory that the gate is to occupy. In line 340 SPOT is used to calculate the actual memory location. In line 350 the value

160 is the screen display code for a solid box. Note that the gate is initially drawn closed.

380–400: The locations in color memory corresponding to the path that the cannonball takes are set to indicate black. This is done ahead of time so that when the cannonball is actually drawn, it is not necessary to poke into color memory. Note that before the cannonball is drawn, the black color is not seen because the corresponding locations in screen memory contain blanks.

410–430: Draw the cannon. Since it lies in the same row as that in which the cannonball travels, it is not necessary to poke into color memory.

440: Open the gates.

Subroutine

500–560: Displays the player's points. It is assumed that this value never goes above 4 digits.

520–540: Print leading 0s, depending upon the number of digits that the value of PTS occupies.

550: This use of the MID$ and STR$ functions removes the blank space that is normally printed to the left of positive numbers.

Subroutine

600–650: Opens the gates. The screen memory locations corresponding to the gate opening are set to blanks. The value of LIM is calculated such that the gate stays open for a random period between half a second and two seconds, in half-second increments.

Subroutine

700–750: Closes the gates. The screen memory locations corresponding to the gate opening are set to solid boxes. The value of LIM is calculated such that the gate stays closed for between one and four seconds, in one-second increments, the exact time being randomly decided.

Subroutine

800–840: Produces a bang sound using the noise generator in the sound chip. The sound is initiated here but is turned off in the main program at line 150.

SUGGESTED ENHANCEMENTS

1. The number of points awarded or deducted may be changed to suit the player.
2. The length of time that the gate stays open or closed may be modified for a more (or less) challenging game.
3. The length of the gate may be increased or decreased. A shorter gate is easier to penetrate while a longer one would increase the possibility of failure. This would involve changes not only in the main program but also in many of the subroutines.

CHAPTER 9

BAGELS

This is a takeoff on a popular game that has been around for a great number of years. In this Commodore 64 version the computer randomly selects a three-digit number, where the leading digits are permitted to be 0. The player has to make a guess at the number and the computer responds with clues as to how close he is to the right number. These clues take the form of LOX or BAGELS. For each digit in the player's guess, the machine displays the word LOX or BAGELS or nothing at all. The word BAGELS is displayed if the digit in question is the correct digit in the correct position of the number. In other words, if BAGELS is displayed three times, the number has been correctly guessed. The word LOX is displayed if a digit in the guess is one that is present in the randomly selected number but is in the wrong position. If the digit in the guess is absent from the selected number, nothing is displayed for that particular digit. Note that the order in which the words LOX and BAGELS appear has no bearing on the digits to which they refer. For example, if the computer responds to a guess of 123 with the message BAGELS LOX LOX, we know that all of the digits are present in the computer's number, although we do not know whether it is the 1 or the 2 or the 3 that is in the correct position. In the computer's response BAGELS always appears before LOX.

The program never displays more than one word for each digit in the player's guess or in the computer's number. For example, if the computer's number is 939 and the player guesses 694, then the word LOX is printed only once (for the player's 9), not twice. If the player had guessed 964, the program would have displayed only the word BAGELS. In addition, if the computer's number were 294, and the player guessed 909, the program would display the word LOX only once since the digit 9 occurs only once in the computer's number. If the player had guessed 099, the word BAGELS would be displayed only once. Perhaps a sample game would be in order for the purpose of explaining the kind of strategy that is used in playing this game.

Suppose that the computer has randomly selected its three-digit number. We don't know yet what it is.

We enter:	The computer displays:
123	LOX

At the beginning, it is not a good idea to repeat digits in the guess, since our initial object is to find out about as many different digits as possible in as few guesses as possible. In this case, what we have found out is that one of the digits 1, 2, or 3 is present in the computer's number, although none of the positions is correct. We guess that the 1 is correct (although not in its right place). So we put it in another position (the second), throw out 2 and 3 for the time being, and try two new digits.

We enter:	The computer displays:
415	BAGELS LOX

So far our assumption about the 1 seems to hold, namely that it is one of the digits in the computer's number and occurs in the second position. Since LOX was also displayed, we must therefore decide whether to keep the 4 or the 5. Let us arbitrarily stay with the 4, placing it in another position (though not the second!), and try a new digit in the first position.

We enter:	The computer displays:
614	BAGELS BAGELS

At this point it would appear that both the 1 and the 4 are correct and in their correct locations. We will therefore replace the 6 with another new digit.

We enter:	The computer displays:
714	BAGELS BAGELS

The 7 is not the one. We try another digit.

We enter:	The computer displays:
814	BAGELS BAGELS

So far, nothing is new. However we have two more digits—9 and 0—to try in the first position; if they don't work, we can try putting another 1 or a 4 in the first position.

We enter:	The computer displays:
914	BAGELS BAGELS BAGELS

YOU'VE EARNED YOUR LUNCH!!!
YOU TOOK 6 TRIES.

As you can see, when three BAGELS are displayed, the computer not only gives you a smart-aleck remark but tells you how many guesses it took you to arrive at the correct answer. The program will then ask you whether you want to continue, and you may reply by typing either Y for "yes" or N for "no" (hitting RETURN is not necessary).

One situation that we did not encounter in the above illustration was the case when the computer responds with no words at all. This would simply mean that none of the entered digits was correct and that the next try should not include any of them.

```
10 REM BAGELS
20 DIM N$(3),G$(3)
30 FOR I=1 TO 3
40 N$(I)=CHR$(INT(RND(0)*10)+48)
50 NEXT I
60 PRINT:PRINT "I AM THINKING OF A 3-DIGIT
   NUMBER"
70 T=0
80 T=T+1:I=0:PRINT:PRINT "? ";
90 PRINT "R ■■";
100 GET A$:IF A$="" THEN 90
110 IF A$=CHR$(13) THEN 200
120 IF A$="■" THEN 170
130 IF A$<"0" OR A$>"9" THEN 100
140 IF I=3 THEN 100
150 PRINT A$;:I=I+1:G$(I)=A$
160 GOTO 90
170 IF I=0 THEN 100
180 I=I-1:PRINT " ■■";
190 GOTO 90
200 IF I<3 THEN 100
210 PRINT "  ";
220 B=0
230 FOR I=1 TO 3
240 IF G$(I)<>N$(I) THEN 270
250 PRINT "BAGELS ";
260 B=B+1:G$(I)=""
270 NEXT I
280 FOR I=1 TO 3
290 FOR J=1 TO 3
300 IF G$(I)<>N$(J) OR G$(J)="" THEN 330
310 PRINT "LOX ";
320 GOTO 340
330 NEXT J
340 NEXT I
350 IF B<3 THEN 80
360 PRINT:PRINT
370 PRINT "YOU'VE EARNED YOUR LUNCH!!!"
```

```
380 PRINT "YOU TOOK";T;"TRIES."
390 PRINT:PRINT
400 PRINT "AGAIN (Y OR N)? ■ ■■";
410 GET A$:IF A$="" THEN 410
420 IF A$<>"Y" AND A$<>"N" THEN 410
430 PRINT A$:PRINT
440 IF A$="Y" THEN 30
450 END
```

GLOSSARY OF VARIABLES

N$	A string array each of whose elements is a digit in the computer's selected number.
G$	A string array each of whose elements is a digit in the player's guess.
T	Used to count the number of tries made by the player.
I	When the player's guess is being entered, this variable indicates the position of the digit last entered. It is used to index G$. However, it is also in other parts of the program for other purposes.
B	Counts how many times BAGELS has been printed for the current guess.

LINES OF SPECIAL INTEREST

30–50: The program selects a 3-digit number. Notice the method employed to convert this number into its corresponding character. This conversion is necessary because the comparisons are made with characters.

80–200: This block of statements inputs the player's guess using the GET command. The method is similar to that used in the program "Typing Test."

90: A cursor is printed to indicate where the next character is to be displayed.

140: A carriage return is necessary for the program to continue. If instead the player tries to strike a fourth character, it is ignored and control is passed to line 100 to wait for a RETURN.

170–190: The player is backspacing. If no characters have been entered yet (I = 0), control returns to line 100. Otherwise, the variable I is decremented and the cursor is moved back 1 position.

200: If the player hits the RETURN key before he has entered 3 digits, the RETURN is ignored.

230–270: Each digit in the guess (G$) is compared with the corresponding digit in the computer's number (N$). If any corresponding digits are found to be identical, the word BAGELS is printed, B is incremented, and the digit in G$ which matches the one in N$ is deleted (replaced with the null string). This is so that the digit will not be used when searching for LOX.

280–340: Compare each remaining digit in G$ with each digit in N$. If a match is found, LOX is printed. In line 300, the testing of G$(J) is to make sure that if a digit which occurs once in N$ occurs twice in G$, and BAGELS is printed for one of the occurrences in G$, nothing is printed for the other occurrence.

350: If BAGELS has not been printed 3 times this time around, the program goes back to get another guess.

SUGGESTED ENHANCEMENTS

1. The game could be made more challenging by increasing the number to more than three digits. This is not such a major programming change as it would appear.
2. The game could be made more difficult by using letters of the alphabet rather than numbers, or even both letters and numbers. This would require changes in line 40 and the lines that get the player's guess. Fortunately, the fact that the program already deals with characters rather than numbers makes this a relatively minor change.
3. The inclusion of sound at appropriate places could improve the aesthetic quality of the game.

CHAPTER 10

CRAPS

This is a Commodore 64 version of the casino game called craps which is customarily played with dice. There are different sets of rules for playing the game, and the version we present here is one of the more popular ones.

The game is played as follows. The player rolls two dice and adds up their face values. This number is referred to as his "point." He then repeatedly rolls the dice until he again obtains his point. If at any time in the game (including the first roll) he rolls a 2 ("snake eyes") or a 12 ("boxcars"), he loses immediately. Furthermore, if he rolls a 7 or 11 after the first roll, he also loses. If, however, the first roll is a 7 or an 11, he wins immediately. Otherwise, the player wins if he repeats his point value without obtaining a 7, 11, snake eyes, or boxcars.

The game of craps is usually accompanied with betting. A player wagers a certain amount before each round. If he wins, he receives the amount of the bet. If he loses, he must pay that amount.

When the program is run it will ask you how much you have on hand at the beginning of the game. This is the pool from which your subsequent bets will be paid and to which you will add your winnings (if any). The amount must be in whole dollars and must be greater than 0. If 0 is entered, or even if the RETURN key

is pressed, the program displays a message asking for positive dollar amounts and will again wait for input. The amount entered is displayed in the upper lefthand corner throughout the game. Next the program will draw the dice and ask you how much you wish to bet. This amount must be no greater than the amount that you have on hand. If the bet amount is 0 or if you just hit RETURN, the program terminates, displaying the amount of money left in the pool.

If the amount bet is greater than 0, the dice will roll; in this version of the game, this means that you will see the spots move around the dice and hear various tones being emitted. Just in case you are not aware that the dice are rolling, a message to this effect is also displayed. When this all stops, the dice display two values (one for each die). If this total value is not 7, 11, 2, or 12, it is recorded on the screen as your point and the dice roll again. When you finally win or lose, the program displays the appropriate message, emits an appropriate sound, and either adds your bet to, or deducts your bet from, the pool of cash.

```
10 REM CRAPS
20 DIM DICE(6),S(2)
30 FOR I=1 TO 6:READ DICE(I):NEXT I
40 DATA 16,257,273,325,341,365
50 POKE 53280,2:POKE 53281,14:KOLR=0
60 PRINT "■■STARTING AMOUNT? $";
70 GOSUB 1100:PRINT
80 IF X>0 THEN 110
90 PRINT "ONLY POSITIVE AMOUNTS, PLEASE";
100 FOR I=1 TO 500:NEXT I:GOTO 60
110 T=X:GOSUB 500
120 ROW=1:COL=0:GOSUB 1000
130 PRINT "$";STR$(T);"          ";
140 ROW=14:GOSUB 1000
150 PRINT "                ";:REM SPC(16)
160 ROW=18
170 GOSUB 1000
180 PRINT "AMOUNT OF BET? $    ■■■■■■■■";
```

```
190 GOSUB 1100
200 IF X=0 THEN 470
210 IF X<=T THEN 250
220 GOSUB 1000
230 PRINT "YOUR BET IS TOO HIGH";
240 FOR I=1 TO 500:NEXT I:GOTO 170
250 BET=X:POI=0
260 GOSUB 700:FOR I=1 TO 1000:NEXT I
270 IF SUM=2 OR SUM=12 THEN 400
280 IF SUM<>7 AND SUM<>11 THEN 310
290 IF POI=0 THEN 360
300 GOTO 400
310 IF POI>0 THEN 350
320 POI=SUM
330 ROW=14:COL=0:GOSUB 1000
340 PRINT "YOUR POINT IS";POI;:GOTO 260
350 IF SUM<>POI THEN 260
360 ROW=20:COL=0:GOSUB 1000
370 PRINT "YOU WIN!!";:GOSUB 1200
380 GOSUB 1000:PRINT "          ";
390 T=T+BET:GOTO 120
400 ROW=20:COL=0:GOSUB 1000
410 PRINT "YOU LOSE.";:GOSUB 1400
420 GOSUB 1000:PRINT "          ";
430 T=T-BET:IF T>0 THEN 120
440 ROW=21:GOSUB 1000
450 PRINT "YOU RAN OUT OF MONEY";
460 FOR I=1 TO 3000:NEXT I
470 POKE 53280,14:POKE 53281,6
480 PRINT "■■":ROW=1:GOSUB 1000
490 PRINT "$";T;TAB(15);"GAME OVER":END
500 REM ***** DRAW DICE FRAMES *****
510 PRINT "■"
520 KAR=160
530 ROW=3
540 FOR COL=5 TO 15:GOSUB 900:NEXT COL
550 FOR COL=24 TO 34:GOSUB 900:NEXT COL
560 COL=5
```

```
570 FOR ROW=4 TO 10:GOSUB 900:NEXT ROW
580 COL=15
590 FOR ROW=4 TO 10:GOSUB 900:NEXT ROW
600 COL=24
610 FOR ROW=4 TO 10:GOSUB 900:NEXT ROW
620 COL=34
630 FOR ROW=4 TO 10:GOSUB 900:NEXT ROW
640 ROW=11
650 FOR COL=5 TO 15:GOSUB 900:NEXT COL
660 FOR COL=24 TO 34:GOSUB 900:NEXT COL
670 RETURN
700 REM ******* ROLL DICE *********
710 ROW=12:COL=16:GOSUB 1000
720 PRINT "ROLLING...";
730 NUM=INT(RND(0)*7)+1
740 FOR I=1 TO NUM:FOR WHICH=1 TO 2
750 S(WHICH)=INT(RND(0)*6)+1:GOSUB 800
760 NEXT WHICH:NEXT I
770 ROW=12:COL=16:GOSUB 1000
780 PRINT "          ";:SUM=S(1)+S(2)
790 POKE 54276,0:RETURN
800 REM ******* DODICE  ******
810 Q=DICE(S(WHICH)):BOT=19*(WHICH)-12
820 FOR ROW=5 TO 9 STEP 2
830 FOR COL=BOT TO BOT+6 STEP 3
840 KAR=32:IF NOT-(Q AND 1) THEN 860
850 KAR=160:GOSUB 1600
860 GOSUB 900:Q=INT(Q/2)
870 NEXT COL
880 NEXT ROW
890 RETURN
900 REM ******* PUT A DOT *********
910 POKE 1024+ROW*40+COL,KAR
920 POKE 55296+ROW*40+COL,KOLR
930 RETURN
1000 REM **** PUT CURSOR AT ROW,COL ****
1010 SPOT=1024+40*ROW+COL
1020 POKE 209,SPOT AND 255
```

```
1030 POKE 210,INT(SPOT/255)
1040 POKE 211,0
1050 RETURN
1100 REM ********* INPUT X *********
1110 X=0
1120 PRINT "■ ■■";
1130 GET A$:IF A$="" THEN 1130
1140 IF A$=CHR$(13) THEN 1190
1150 IF A$<>"■■" THEN 1170
1160 X=INT(X/10):PRINT " ■■";:GOTO 1120
1170 IF A$<"0" OR A$>"9" THEN 1130
1180 PRINT A$;:X=X*10+ASC(A$)-48:GOTO 1120
1190 PRINT " ";:RETURN
1200 REM *** GOODBEEP ***
1210 POKE 54272,224:POKE 54279,68
1220 POKE 54273,37:POKE 54280,38
1230 POKE 54277,65:POKE 54284,17
1240 POKE 54278,244:POKE 54285,241
1250 POKE 54296,15
1260 POKE 54276,17:POKE 54283,17
1270 FOR I=1 TO 200:NEXT I
1280 POKE 54273,40:POKE 54280,41
1290 FOR I=1 TO 500:NEXT I
1300 POKE 54276,0:POKE 54283,0
1310 RETURN
1400 REM *** BADBEEP ***
1410 POKE 54272,224:POKE 54279,68
1420 POKE 54273,14:POKE 54280,15
1430 POKE 54277,65:POKE 54284,17
1440 POKE 54278,244:POKE 54285,241
1450 POKE 54286,224:POKE 54287,4
1460 POKE 54296,15
1470 POKE 54276,21:POKE 54283,17
1480 FOR I=1 TO 100:NEXT I
1490 POKE 54273,8:POKE 54280,9
1500 FOR I=1 TO 500:NEXT I
1510 POKE 54276,0:POKE 54283,0
1520 RETURN
```

```
1600 REM *** SPOT BEEP ***
1610 POKE 54276,0
1620 FR=COL*500+ROW*50
1630 POKE 54272,FR AND 255
1640 POKE 54273,INT(FR/256)
1650 POKE 54275,8
1660 POKE 54277,34:POKE 54278,242
1670 POKE 54296,15:POKE 54276,65
1680 RETURN
```

GLOSSARY OF VARIABLES

DICE This numeric array of 6 elements contains numbers which are encoded representations of the spot patterns on each face of the die.

S This is a numeric array with 2 elements. It represents the values on the faces of the 2 dice.

X The variable returned by the subroutine at line 1100. It represents a value input by the player.

T Holds the total amount of money that the player has.

ROW Passes to the subroutine at line 1000 the row at which the cursor is to be positioned.

COL Passes to the subroutine at line 1000 the column at which the cursor is to be positioned.

BET Holds the amount bet by the player.

POI Holds the player's point (if any); otherwise it is set to 0. We wanted to use the name POINT, but this contains the BASIC keyword INT and so had to be discarded.

SUM Returned by the subroutine at line 700; this is the sum of the face values of the 2 dice.

KAR Holds a screen display code which is passed to the subroutine at line 900. It actually takes on only one of two possible values: 32 for a blank or 160 for a spot.

NUM A randomly selected number between 1 and 7 which represents how many times the dice are rolled before stopping. This variable is used in the subroutine in line 700.

WHICH This variable holds the value 1 or 2 and indicates which die is currently being rolled.

LINES OF SPECIAL INTEREST

50: For the first time in this book we are poking to location 53280, to change the border color of the screen to red. This line also changes the background color to light blue. The variable KOLR is used only by the subroutine at line 900 to determine the color of the dice. (COLOR could not be used because it contains the keyword OR, and even COLR would have given us problems because it is treated the same as the variable COL, which is also being used in this program.) It is not really necessary to have a variable for the color of the dice, but this makes it possible to make three different color changes just by changing line 50.

100: This delay loop keeps the message in line 90 on the screen for a short amount of time to enable the message to be read.

120: This is the standard way of repositioning the cursor in this program. The values of ROW and COL are assigned and the subroutine at line 1000 is called.

130: The ten blanks printed on this line are used in case the value of T decreases such that the number of digits in it decreases. The blanks remove digits left over from the previous value of T.

140–150: Blanks are printed on row 14 to remove the message that may have been there previously; that row is used to record the player's point. The REM statement is left there to indicate how many spaces there are in the literal. Note that the SPC function does not actually print blanks but merely tabs over the specified number of columns. That is why it is not used in this case.

180: The four blanks and four backspace characters at the end of the literal are used to remove the characters that may have occupied that space previously (see lines 210–240 below).

200: A bet of 0 means that the program should be terminated.

210–240: If the bet is greater than the cash on hand, a message to this effect is displayed in the same row as the message printed in line 180. It is kept there for a while by a delay loop. Control then returns to line 170 to request another bet.

260: The delay loop in this line creates a pause between rolls of the dice.

290: This line is reached if SUM is 7 or 11. POI being equal to 0 means that this was the first roll and so the player has won.

300: This is not the first roll, so the 7 or 11 has caused the player to lose.

310: Only if the roll was not 7, 11, 2, or 12 is this line reached. If

this is not the first roll, control passes down to line 350 where it is compared to the point value which was the value of the first roll.

320–340: This is the first roll, so its value is saved and displayed as the point value. Control then resumes from line 260, where the dice are again rolled.

350: The roll is compared against the point. If they are not equal, control passes to line 260.

380: After a victorious fanfare is sounded, the winning message is blanked out.

440–460: The player has run out of money and a message to this effect is displayed. The delay in line 460 retains the message on the screen for a while.

470–490: These lines are reached if the player runs out of money or if he makes a bet of 0 (see line 200). The original border, screen, and character colors are restored. The screen is then cleared and the player's remaining balance is displayed.

Subroutine

500–670: This subroutine draws the outlines of the dice. The subroutine at line 900 is called repeatedly to draw each character in the outline.

Subroutine

700–790: This subroutine simulates the rolling of the dice. The faces of the dice are changed a random number of times between 1 and 7 as determined by the value of NUM (see line 730). In line 750 a random number between 1 and 6 becomes the face value for each die. In line 780 the message printed by line 720 is removed. The POKE in line 790 turns off the final beep that was initiated by the subroutine in line 1600, which was itself called by the subroutine in line 800.

Subroutine

800–890: This subroutine does the actual printing of the face of each die.

810: WHICH specifies which die is being rolled. S(WHICH) specifies the face value of this die, and DICE(S(WHICH)), the value assigned to Q, is the coded value for the dot pattern for this number. The code works as follows. The face of each die is divided into a 3×3 matrix of spots. For each face value, some of these spots are printed and some are not printed. For example, to represent the face value 5, the 1st, 3rd, 5th, 7th, and 9th spots are printed. (We are

numbering them left to right and top row to bottom row.) So each face value can be represented by a nine-bit number where each 1 represents a spot that is displayed and each 0 represents a spot that is not displayed. This subroutine extracts the bits from the coded value one at a time and determines whether the corresponding spot should be printed or blanked out. The role played by BOT is to determine the starting column for displaying the dots, depending on which die is being displayed.

840: This line determines whether the rightmost bit in Q is a 0 or a 1. The negation sign to the left of the parentheses is necessary because the NOT operator works correctly only on 0 and negative values. Notice that if a spot is not placed at the current location, then a blank must be placed there in case there was a spot there before.

850: This line is reached only if the rightmost bit in Q is a 1. Only in this case is the subroutine in line 1600 called.

860: Dividing Q by 2 has the effect of shifting its binary value one bit to the right, thereby moving the next bit into the rightmost position.

Subroutine

900–930: The character specified by the contents of KAR is placed in the correct location in screen memory, as specified by ROW and COL. The value of KOLR is placed in the corresponding location in color memory.

Subroutine

1000–1050: The cursor is repositioned using POKEs to locations 209 and 210. Location 211 is set to 0 so that it does not cause any extra positioning of the cursor.

Subroutine

1100–1190: The GET command is used to input the value of X from the keyboard.

1160: If the backspace cursor key is pressed, X is changed to eliminate the last digit entered and the cursor is moved back.

1180: X is multiplied by 10 and has added to it the numerical value of the digit entered.

Subroutine

1200–1310: Plays a melody indicating that the player has won. Two voices are used in this refrain.

Subroutine
1400–1520: Plays a loser's melody. Not only are two voices used here but ring modulation as well, which is an electronic music technique that creates unusual sound effects.

Subroutine
1600–1680: This subroutine produces a beep each time a spot is displayed on the screen.

1610: The previous beep is turned off. Doing this here allows the beep to remain on as long as possible.
1620: The frequency of the sound is based upon the spot's location on the screen.

SUGGESTED ENHANCEMENTS

1. The complete game can be rewritten to allow for two players to compete with each other.
2. If necessary, the rules can be changed to suit your own tastes.
3. Perhaps even more color can be introduced.

CHAPTER 11

HI-LO

This game is a Commodore 64 takeoff on a very entertaining and innovative toy calculator manufactured by Texas Instruments, Inc. The calculator is called "Dataman" and includes several instructive numeric games intended to enhance a youngster's interest in arithmetic. One of these games is a guessing game in which a number between 1 and 100 is randomly selected by the calculator. The child is supposed to guess the hidden number; when he or she succeeds, the display announces how many tries were used.

Of course, with such a powerful computer at our disposal, there is no reason to confine ourselves to such a limited range. In fact, the version that we now present allows for a range going from 1 to any number at all of your own choosing. The game is enhanced by color, sound, and some graphics. This is certainly not kid stuff!

When you run the program, you are asked for the upper limit of the range from which to select the random number. The program actually displays the number 1, the lower limit, and all you have to do is to supply the second number. A response of 0 causes the computer to ignore the value and the question is repeated. Neither is a negative or a fractional number accepted. Your all-knowing computer is far too smart for this.

After entering the upper limit of the range the computer displays the two ranges enclosed in boxes. It then requests your guess. The guesses are numbered sequentially so that you are always aware of how many attempts you have made. If you select a number outside of the prescribed range, it will be ignored and another request made. If the number you guessed is higher than the computer's number, the word LOWER is displayed, indicating that you must try for a lower number. Similarly, if you guess too low, the word HIGHER is displayed and you must look for a higher number. As the range narrows, so the numbers in the boxes change to reflect these new limits.

When the correct answer has been entered, the computer displays a message to this effect and plays a little melody by way of congratulations. The player is then asked if another game is desired. If so, another limit is requested and another round commences. If not, the computer displays a number specifying how well the player guessed. This number is based on the player's overall performance for all games played in a given session. A rating of 100 (like our IQs) is considered average. To be fair to those who select wide ranges of numbers, the size of the range is taken into account and the average expected number of guesses is correspondingly larger.

A close examination of the program will reveal a hidden strategy that ensures success in this game. Without revealing what this strategy is, let it suffice to say that "half a loaf is better than none."

```
10 REM HI-LO
20 POKE 54296,15:CUME=0:GAME=0
30 LO=1:PRINT:PRINT "RANGE? 1-";
40 GOSUB 600
50 IF S<1 THEN 30
60 HI=S:N=INT(RND(0)*HI)+1
70 GOSUB 300:TRY=0:GAME=GAME+1
80 GOSUB 400
90 GOSUB 600
100 IF S>=LO AND S<=HI THEN 160
110 PRINT TAB(INDENT);"□OUT OF RANGE";
```

```
120 PRINT "           □"
130 FOR I=1 TO 1000:NEXT I
140 PRINT TAB(INDENT);"GUESS #";
150 PRINT TRY+1;"■■:    ■■■■";:GOTO 90
160 TRY=TRY+1
170 IF S=N THEN 210
180 IF S>N THEN 200
190 GOSUB 800:LO=S:GOTO 80
200 GOSUB 900:HI=S:GOTO 80
210 GOSUB 1000
220 CUME=CUME+RAT/TRY
230 PRINT:PRINT "AGAIN (Y OR N)? ";
240 GET A$:IF A$<>"Y" AND A$<>"N" THEN 240
250 PRINT A$
260 IF A$="Y" THEN 30
270 POKE 54296,0:PRINT "▼GAME OVER"
280 PRINT:PRINT "YOUR RATING IS";
290 PRINT INT(CUME/GAME*100+.5):END
300 REM CALCULATE AVERAGE RATING
310 Q=HI:RAT=1
320 Q=INT(Q/2)
330 IF Q=0 THEN 360
340 RAT=RAT+1
350 GOTO 320
360 RETURN
400 REM SET UP
410 PRINT "▼■■■■■■■■■■■"
420 SLO=LEN(STR$(LO))+1
430 SHI=LEN(STR$(HI))+1
440 PRINT "◪";
450 FOR I=1 TO SLO:PRINT "⊟";:NEXT I
460 PRINT "◱";SPC(34-SHI-SLO);"◪";
470 FOR I=1 TO SHI:PRINT "⊟";:NEXT I
480 PRINT "◱":PRINT " ◫";LO;"◫";
490 PRINT SPC(34-SLO-SHI);
500 PRINT "◫";HI;"◫":PRINT " ◳";
510 FOR I=1 TO SLO:PRINT "⊟";:NEXT I
520 PRINT "◰";SPC(34-SHI-SLO);"◳";
```

```
530 FOR I=1 TO SHI:PRINT "⊟";:NEXT I
540 PRINT "◰":PRINT
550 INDENT=INT((26-SHI)/2)
560 PRINT TAB(INDENT);"GUESS #";
570 PRINT TRY+1;"■■: ";
580 RETURN
600 REM GET NUMBER
610 S=0
620 PRINT "■ ■■■";
630 GET A$:IF A$="" THEN 630
640 IF A$=CHR$(13) THEN 720
650 IF A$="■■" AND S>0 THEN 700
660 IF A$<"0" OR A$>"9" THEN 630
670 PRINT A$;
680 S=S*10+ASC(A$)-48
690 GOTO 620
700 S=INT(S/10):PRINT " ■■■■";
710 GOTO 620
720 PRINT " "
730 RETURN
800 REM HIGHER
810 PRINT "■■■■■■■■";TAB(17);"HIGHER"
820 POKE 53281,1
830 POKE 54272,135:POKE 54273,33
840 POKE 54277,34:POKE 54278,242
850 POKE 54276,17
860 FOR I=1 TO 500:NEXT I
870 POKE 54276,16
880 POKE 53281,6
890 RETURN
900 REM LOWER
910 PRINT "■■■■■■■■";TAB(17);"LOWER"
920 POKE 53281,0
930 POKE 54272,97:POKE 54273,8
940 POKE 54277,34:POKE 54278,242
950 POKE 54276,17
960 FOR I=1 TO 500:NEXT I
970 POKE 54276,16
```

```
980 POKE 53281,6
990 RETURN
1000 REM CORRECT
1010 PRINT:PRINT TAB(INDENT);"CORRECT!"
1020 POKE 54272,195:POKE 54273,16
1030 POKE 54275,8
1040 POKE 54277,7:POKE 54278,240
1050 POKE 54276,65
1060 FOR I=1 TO 100:NEXT I
1070 POKE 54272,210:POKE 54273,15
1080 FOR I=1 TO 100:NEXT I
1090 POKE 54272,195:POKE 54273,16
1100 FOR I=1 TO 100:NEXT I
1110 POKE 54272,96:POKE 54273,22
1120 FOR I=1 TO 300:NEXT I
1130 POKE 54272,49:POKE 54273,28
1140 FOR I=1 TO 250:NEXT I
1150 POKE 54276,64
1160 RETURN
```

GLOSSARY OF VARIABLES

CUME This is the player's cumulative rating for all the games played in a given session.

GAME This is the number of games played so far.

LO The low end of the current range.

HI The high end of the current range.

S The value returned by the subroutine at line 600. It is the number inputted by the player.

N The computer's randomly selected number which the player must guess.

TRY The counter for the number of guesses made for each individual game.

INDENT A value calculated in the subroutine at line 400 which specifies the starting column at which various messages are displayed.

RAT This is the value calculated in the subroutine at line 300 which represents the average number of guesses necessary to arrive at the computer's hidden random number.

SLO The size of the field in which the value of LO is displayed on the screen.

SHI The size of the field in which the value of HI is displayed on the screen.

LINES OF SPECIAL INTEREST

150: The first backspace character is to eliminate the blank inserted after numbers by the PRINT command. The last two blanks and two backspace characters are included to eliminate the end of the message printed by line 110.

190: This line is reached if the guess is too low.

200: This line is reached if the guess is too high.

210–220: If the player guesses correctly, the victory tune is played and the value of CUME is incremented by the ratio of the average number of guesses to the player's actual number of guesses.

290: The actual rating is computed by dividing CUME by the number of games. This in effect produces a rating that is averaged over all the games. Multiplying the result by 100 produces a rating that is standardized to 100. The addition of 0.5 causes the INT function to round off the number rather than simply truncate it.

Subroutine

300–360: This subroutine calculates the average number of guesses that should have to be made to arrive at N. The value for RAT is actually based upon the size of the range and is independent of the value N. This is important to note because some values of N may be easier to guess than others, but the value for RAT will be just as easy (or difficult) regardless. RAT is calculated as the number of bits in the binary representation of the variable HI. This is also known as the logarithm to base 2 of HI.

Subroutine

400–580: The role played by this subroutine is to redraw the screen with the current values of HI and LO placed in the boxes.

420–430: The boxes are to be "tailor made" to the size of the numbers when they are displayed. These lines therefore calculate the size of the numbers in display format, allowing for a blank on either side.

440: The graphics character used here is Commodore key-A.

450: The graphics character used in this case is either shift-C or shift-asterisk. This is also the character used in lines 470, 510, and 530.

460: In the first literal the graphics character used is Commodore key-S. In the second literal it is Commodore key-A.

480: The first graphics character used in this line is Commodore key-S. The second two are both either shift-B or shift-minus sign.

500: The first two graphics characters are either shift-B or shift-minus sign. The third one is Commodore key-Z.

520: In this line the first graphics character is Commodore key-X and the second is Commodore key-Z.

540: The graphics character found in this line is Commodore key-X.

550: The value for INDENT is selected to approximately center messages between the two numbers displayed on the screen.

Subroutine

600–730: This subroutine inputs a value for S using the GET command. As in previous programs there is a simulated cursor and only digits are accepted. The left-cursor function can be used to make corrections to mistyped characters. The PRINT statement in line 720 removes the final cursor symbol.

Subroutine

800–890: The player is instructed to guess higher and an appropriately high pitched tone is produced. Lines 820 and 880 also change the screen color.

Subroutine

900–990: The player is instructed to guess lower. Here a low-pitched tone is produced instead.

Subroutine

1000–1060: The player is informed that he has guessed correctly and a perhaps all-too-familiar melody is produced.

SUGGESTED ENHANCEMENTS

1. The range could start from a number other than 1.
2. A special sound can be played if the player guesses outside the range.

CHAPTER 12

BIORHYTHMS

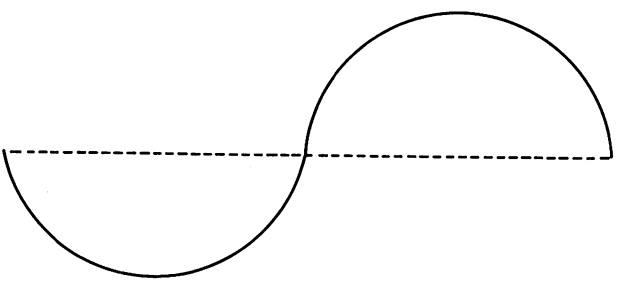
If you have been programming for any length of time, you will surely have come upon a period when no program you write seems to work correctly. In addition, your relations with your friends and relatives seem to be lacking, your zest for life is on the downgrade; in essence, nothing at all seems to go right. We customarily speak of such days as "down days." On the other hand, of course, there are those days (thankfully!) when everything we touch seems to turn to gold. Whatever plans we have made materialize to our satisfaction and enjoyment, meals taste superb, social engagements are rewarding, and so on. We tend to recall such periods as "up days." It is as if we human beings operate on some kind of a cycle which has its upward and downward turns. Why, even the stock market seems to follow such behavior.

According to adherents of the biorhythm theory, we all have cycles which begin at the moment of birth. These cycles influence our physical, emotional, and intellectual states. In certain coun-

tries of the world, in particular Japan and Switzerland, biorhythm theory is treated with considerable seriousness. Of course, there are many other people who reject any validity to biorhythm theory. They hold it in the same disrespect as they do astrology, extrasensory perception (ESP), unidentified flying objects (UFOs), and organic foods. The authors of this book disassociate themselves from either side of the argument, and merely present the theory for whatever interest it might hold for you.

According to biorhythm theory, the physical cycle lasts twenty-three days, the emotional cycle twenty-eight days, and the intellectual cycle thirty-three days. You will notice that there is a difference of five days between the lengths of each of the cycles. It is claimed that all three of these cycles begin on the day you are born and continue unaltered throughout your life.

These cycles of which we speak can be thought of as interminable sine curves that have the following shape:



The time it takes to get from one end of this curve to the other is, as we have said, twenty-three days for the physical, twenty-eight days for the emotional, and thirty-three days for the intellectual cycle.

In the first 11½ days of the physical cycle (that's when the curve is above the horizontal line as illustrated above) you are

considered to be in the positive half of your cycle. This implies that you should feel physically well, sturdy, robust, and ready to conquer the world. This should be most noticeable 5¾ days into the cycle, when the curve reaches its highest point. In the second 11½ days, however, you are going through the negative part of the cycle, and it is said that you will feel listless, anemic, and in a general condition of exhaustion.

The emotional cycle, which lasts for twenty-eight days, also has its two halves, the first during the first fourteen days and the second during the last fourteen days. During the positive half of the cycle, you will bear a cheerful countenance, be full of optimism and generally ebullient. Of course, during the negative half of the cycle you will be inclined to be pessimistic, surly, irritable, and unpleasant.

The longest cycle of the three, the intellectual cycle, which lasts thirty-three days, is also composed of two halves. During the "up" half, you will probably program at your finest level, you will come up with your greatest ideas, and perhaps dream up an exciting computer game. It is during this period that learning new material is a welcome challenge and your creative spirit seems to know no limits. It goes without saying that during the second half of the cycle, clever thoughts will seem to evade you, the spark of creativity seems utterly quenched, and you're probably better off avoiding intellectual confrontations.

With each of these cycles, there is always a change going from the plus or the up side to the minus or the down side, or vice versa. These days are indicated in the above diagram at those points where the curve hits the horizontal line. Such days are regarded as "critical days." According to the advocates of biorhythm theory, one is more accident prone on the critical days in both the physical and emotional cycles. If a critical day occurs in the intellectual cycle, it is not considered to be so noteworthy, except if it coincides with a critical day in one of the other two cycles. Should this be the case, particular caution is advised. Of course, there is always a possibility that on a given day, all three cycles will be at critical points. On such a day, it is recommended that you withdraw from normal daily activity. In fact, some even advocate spending the whole day in bed! Fortunately, such a day

will occur only once every seven years or so. In fact, any given set of the three cycle readings will recur only once every fifty-eight years!

We repeat once more that so far as we are concerned, biorhythm theory is just that—a theory. We have no idea whether or not it has any validity. We are merely presenting the theory to you for your edification, so that should you wish to take advantage of the program presented below, you would at least understand the implications involved.

The calculation of one's biorhythms requires the knowledge of the number of days the person has been alive up to the day in question. For each of the three cycles, this number is divided by 23, 28, or 33, depending upon the cycle of interest. The remainder from this division tells you how many days into the particular cycle you are. Having at your service a computer as powerful as the Commodore 64, it is easy to calculate each of these cycles for a period of seventeen days before and after any given date and display the whole period on the screen at one time.

To run the program, all you have to do is to type in your birthdate (in the format MM/DD/YYYY) and then the current date (in the same format). You may, of course, enter any two dates, but the first must precede the second, and they must both be no earlier than 1583, since this was when the current Gregorian calendar was introduced. In no time at all you will see a diagram of your biorhythm curves being drawn before your very eyes, to the accompaniment of some ethereal-sounding music. The vertical line down the center indicates the points of the curves representing the current day, and each column to the left of the vertical line represents a day before the current day, while each to the right represents a day after the current day. The physical cycle is represented by a succession of brown Ps in reversed graphics. In a similar manner, the emotional cycle is represented by the letter E in cyan, and the intellectual cycle by the letter I in dark blue.

After drawing your biorhythm, the program will wait for your response as to whether or not you wish to have another biorhythm displayed. After you have finished perusing the diagram on the screen, you may answer Y or N.

```
10 REM BIORHYTHMS
15 POKE 53281,1
20 DIM DAYS(12)
30 FOR I=1 TO 12:READ DAYS(I):NEXT I
40 DATA 31,28,31,30,31,30
50 DATA 31,31,30,31,30,31
60 PRINT:INPUT "BIRTHDATE (MM/DD/YYYY)";A$
70 S$=A$:GOSUB 300
80 IF S>0 THEN 100
90 PRINT "BIRTHDATE INVALID":GOTO 60
100 A=S
110 PRINT:INPUT "TODAY'S DATE (MM/DD/YYYY)";B$
120 S$=B$:GOSUB 300
130 IF S>0 THEN 150
140 PRINT "TODAY'S DATE INVALID":GOTO 110
150 B=S
160 IF A<=B THEN 190
170 PRINT "DATES IN WRONG ORDER"
180 A$="":B$="":GOTO 60
190 AGE=B-A
200 GOSUB 700
210 PRINT "DO YOU HAVE MORE DATES (Y OR N)?";
220 GET A$:IF A$="" THEN 220
230 IF A$<>"Y" AND A$<>"N" THEN 220
240 PRINT "■";
250 IF A$="Y" THEN 60
260 POKE 53281,6:PRINT "FINISHED"
270 END
300 REM CHANGE DATE TO ABSOLUTE DAYS
310 S=0:I=1
320 IF LEN(S$)>10 OR LEN(S$)<8 THEN 590
330 IF MID$(S$,I,1)<"0" OR MID$(S$,I,1)>"9"
    THEN 590
340 M=ASC(MID$(S$,I,1))-48
350 I=I+1
360 IF MID$(S$,I,1)="/" THEN 410
370 IF MID$(S$,I,1)<"0" OR MID$(S$,I,1)>"9"
    THEN 590
380 M=M*10+ASC(MID$(S$,I,1))-48
```

```
390 I=I+1
400 IF MID$(S$,I,1)<>"/" THEN 590
410 I=I+1
420 IF MID$(S$,I,1)<"0" OR MID$(S$,I,1)>"9"
    THEN 590
430 D=ASC(MID$(S$,I,1))-48
440 I=I+1
450 IF MID$(S$,I,1)="/" THEN 500
460 IF MID$(S$,I,1)<"0" OR MID$(S$,I,1)>"9"
    THEN 590
470 D=D*10+ASC(MID$(S$,I,1))-48
480 I=I+1
490 IF MID$(S$,I,1)<>"/" THEN 590
500 I=I+1
510 IF LEN(S$)-I<>3 THEN 590
520 Y=0
530 FOR J=I TO I+3
540 IF MID$(S$,J,1)<"0" OR MID$(S$,J,1)>"9"
    THEN 590
550 Y=Y*10+ASC(MID$(S$,J,1))-48
560 NEXT J
570 IF M<1 OR M>12 OR D<1 OR Y<1583 THEN 590
580 GOSUB 600
590 RETURN
600 REM CALCULATE DAYS (CHECK LEAP YR.)
610 LEAP=((INT(Y/4)*4=Y AND INT(Y/100)*100<>Y)
    OR INT(Y/400)*400=Y)
620 DAYS(2)=28:IF LEAP THEN DAYS(2)=29
630 IF D>DAYS(M) THEN 690
640 S=D+365*Y+INT(Y/4)-INT(Y/100)+INT(Y/400)
650 IF M=1 THEN 690
660 FOR I=1 TO M-1
670 S=S+DAYS(I)
680 NEXT I
690 RETURN
700 REM DRAW BIORHYTHM
710 PRINT "■";
720 P=AGE-INT(AGE/23)*23-17
```

```
730 E=AGE-INT(AGE/28)*28-17
740 I=AGE-INT(AGE/33)*33-17
750 FOR COL=0 TO 34
760 KOLR=0:KAR=67:ROW=10:GOSUB 1000
770 IF COL<>17 THEN 800
780 KAR=66
790 FOR ROW=0 TO 20:GOSUB 1000:NEXT ROW
800 PS=INT(10*SIN(2*π*(P+COL)/23)+.5)
810 ES=INT(10*SIN(2*π*(E+COL)/28)+.5)
820 IS=INT(10*SIN(2*π*(I+COL)/33)+.5)
830 ROW=10-PS:KAR=144:KOLR=9
840 VOICE=0:GOSUB 1000
850 ROW=10-ES:KAR=133:KOLR=3
860 VOICE=1:GOSUB 1000
870 ROW=10-IS:KAR=137:KOLR=6
880 VOICE=2:GOSUB 1000
890 NEXT COL
900 POKE 209,87:POKE 210,7
910 POKE 211,0:POKE 214,22
920 PRINT "■TODAY◆"
930 POKE 54276,0
940 POKE 54283,0
950 POKE 54290,0
960 RETURN
1000 REM DRAW KAR IN KOLR AT ROW,COL
1010 POKE 1024+40*ROW+COL,KAR
1020 POKE 55296+40*ROW+COL,KOLR
1030 REM MUSIC
1040 BASE=54272+VOICE*7
1050 FRQ=22000-ROW*1000
1060 POKE BASE+4,0
1070 POKE BASE,FRQ AND 255
1080 POKE BASE+1,INT(FRQ/256)
1090 POKE BASE+3,8
1100 POKE BASE+5,34:POKE BASE+6,34
1110 POKE 54296,15
1120 POKE BASE+4,65
1130 RETURN
```

GLOSSARY OF VARIABLES

DAYS	An array of 12 elements representing the number of days in each of the months.
A$	The birthdate in character form.
B$	The current date in character form.
S$	Used to pass a date to the subroutine in line 300.
A	The birthdate in numeric form.
B	The current date in numeric form.
S	The numeric form of a date as returned by the subroutine in line 300.
AGE	The person's age in days.
I	Used in the subroutine at line 300, this variable indicates the character in S$ that is currently being analyzed.
M	Holds the numeric value of the month, which is calculated in the subroutine at line 300.
D	Holds the numeric value of the day, which is calculated in the subroutine at line 300.
Y	Holds the numeric value of the year, which is calculated in the subroutine at line 300.
P	The biorhythm reading for the physical cycle on the day represented by column 0 on the screen (17 days before the current date).
E	The biorhythm reading for the emotional cycle on the day represented by column 0 on the screen (again, 17 days before the current date).
I	The biorhythm reading for the intellectual cycle on the day represented by column 0 on the screen (also 17 days before the current date).
ROW	The row at which the next symbol is being placed on the screen.
COL	The column at which the next symbol is being placed on the screen.
KAR	The screen display code for the next character to be displayed on the screen.
KOLR	The color of the next character to be displayed on the screen.
PS	The height of the sine curve of the physical cycle for the day being displayed. Its range can vary from −10 to +10.
ES	The height of the sine curve of the emotional cycle for the

day being displayed. Its range, too, can vary from −10 to +10.

IS The height of the sine curve of the intellectual cycle for the day being displayed. Its range also is between −10 and +10.

VOICE Indicates which voice in the sound chip is to be used for the symbol being displayed. The voices are numbered 0, 1, and 2.

LINES OF SPECIAL INTEREST

Subroutine
300–590: This subroutine extracts the month, day, and year from the date in S$ while checking for validity. The subroutine in line 600 is then invoked to convert these values into a single number. For more details of the technique employed, see the program "Elapsed Days" (Chapter 3).

Subroutine
600–690: Converts the date into the number of days since the year 1. Once again, further details can be found in "Elapsed Days."

Subroutine
700–960: This subroutine performs the entire task of displaying the biorhythm curves.

720–740: Calculate the biorhythm values for the day represented by column 0 on the screen. The subtraction of 17 may result in a negative value, but this is fine for our purposes.

750–890: Calculate the biorhythm for 35 days.

760: Draws part of the horizontal line.

770–790: When the current date is reached, these lines of code draw a vertical line down the center of the screen. Since this is done before plotting the portion of the curve for that day, it will be overwritten as necessary by the curves.

800–820: Calculates the distance of the curve from the horizontal line. The arguments to the sine functions convert the biorhythm values to corresponding radian values (note the use of the pi symbol.) Adding the value of COL to P, E, or I produces the biorhythm reading for the day that is being calculated.

830–840: The row at which a symbol is to be placed is calculated. KAR gets the screen display code for a reversed-graphics P. KOLR

is assigned the value 9 for brown. The subroutine at line 1000 is called to display the symbol and play an appropriate tone through voice 0.

850–880: These lines do for the emotional and the intellectual curves what lines 830 and 840 do for the physical curve. In line 850 KAR gets the value for a reversed-graphics E and KOLR gets the value for cyan; in 870 KAR gets the value for reversed-graphics I and KOLR for the color dark blue.

900–920: Notice how location 214 is being used; after the PRINT in line 920 is performed, the cursor will be repositioned at row 23. However, the message printed by line 920 will occur at the location determined by the POKEs in line 900.

930–950: Turn off all the voices.

Subroutine

1000–1130: This is a composite subroutine. By this we mean that we have combined two functions that could have been placed in two separate subroutines. Lines 1010 and 1020 place the specified character at the specified location on the screen. Lines 1040 onward produce an appropriate tone.

1040: The variable BASE is assigned the memory location corresponding to the first voice register of the voice specified by the variable (you guessed it!) VOICE.

1050: FRQ is assigned an appropriate frequency value based upon the row at which the last symbol is placed. The closer to the top of the screen it was placed, the higher the frequency. In this way, you can actually "hear" how bad or good your biorhythm is!

SUGGESTED ENHANCEMENTS

1. You might want to somehow indicate the date for each of the displayed readings. This will almost certainly restrict the number of days displayed.
2. You might want to include the option of displaying any one of the three curves separately.
3. You can arrange the program so that by pressing one of the function keys you can display the previous or the next thirty-five days of biorhythms.

CHAPTER 13

HANGMAN

There are probably few of us who have not enormously enjoyed playing the game of hangman, particularly in our younger years. For those of you who have lived on some outer planet all this time and do not know this wonderful game, here is how our Commodore 64 version of the game is played.

The computer draws a little scaffold and a noose. It then secretly selects a word from its pool of fifty words and displays a group of dashes, with the number of dashes corresponding to the number of letters making up the word. The player has to guess at the hidden word, one character at a time. This is done by pressing the appropriate letter on the keyboard (no RETURN is necessary). If the selected letter is contained in the secret word, that letter replaces all the dashes that correspond to that letter in the hidden word. If it appears in the word more than once, all occurrences are revealed. A high-pitched tone is sounded to acknowledge this successful move. If the chosen letter does not match any of the letters of the hidden word, the letter chosen is placed in a list below the dashes and a low-pitched tone is emitted to signal failure. In addition, part of a man is drawn hanging from the noose. If a letter that has already been chosen is selected, a message to this effect is displayed but the player is (mercifully) not penalized. The player wins if he guesses the secret word

before the hanging man is fully constructed. Once this occurs, a high-pitched tone is again produced. If, however, the hanging man is completed before the secret word is guessed, the player loses to the accompaniment of a well-known dirge. If at any time before the round is concluded the player wishes to surrender, he may type a question mark. This causes the computer to reveal the entire word to the accompaniment of the dirge. In any case, the player is asked if another round is required.

The hanging man is constructed in eight parts. First the head, then the torso, followed by the left arm, then the right arm, the left leg, the right leg, the hands, and finally the feet. Thus the player must make eight wrong guesses before he loses the game.

```
5 REM HANGMAN
10 DIM CHAR(25),F(10,3)
20 POKE 53281,1:PRINT"■";:GOSUB 1500
30 FOR I=0 TO 25:CHAR(I)=0:NEXT I
40 RESTORE:Q=INT(RND(0)*50)+1
50 FOR I=1 TO Q
60 READ WURD$
70 NEXT I
80 GOSUB 600:P=0:KOUNT=LEN(WURD$)
90 ROW=17:COL=15:GOSUB 1400
100 FOR I=1 TO KOUNT
110 PLACE=ASC(MID$(WURD$,I,1))-65
120 CHAR(PLACE)=CHAR(PLACE)+1
130 PRINT "-";
140 NEXT I
150 ROW=21:COL=0:GOSUB 1400
160 PRINT "LETTER? ";
170 GET A$:IF A$="" THEN 170
180 IF A$="?" THEN 400
190 IF A$<"A" OR A$>"Z" THEN 170
200 PRINT A$
210 PLACE=ASC(A$)-65
220 IF CHAR(PLACE)=0 THEN 320
230 IF CHAR(PLACE)>0 THEN 480
```

```
240 GOSUB 1400
250 PRINT "YOU GUESSED ";A$;" ALREADY"
260 IF CHAR(PLACE)=-1 THEN 280
270 POKE 56066+PLACE,8
280 FOR I=1 TO 1000:NEXT I
290 FOR I=1864 TO 1884:POKE I,32:NEXT I
300 POKE 56066+PLACE,0
310 GOTO 150
320 GOSUB 1200:P=P+1:GOSUB 800
330 ROW=19:COL=PLACE+10:GOSUB 1400
340 PRINT A$;
350 FOR I=1 TO 400:NEXT I:POKE 54276,0
360 POKE 1872,32
370 CHAR(PLACE)=-2:IF P<8 THEN 150
380 COL=0:ROW=23:GOSUB 1400
390 PRINT "YOU LOSE!"
400 ROW=17:COL=15:GOSUB 1400:PRINT WURD$
410 GOSUB 1600
420 ROW=24:COL=0:GOSUB 1400
430 PRINT "AGAIN (Y OR N)? ";
440 GET A$:IF A$="" THEN 440
450 IF A$="Y" THEN 30
460 IF A$<>"N" THEN 440
470 POKE 53281,6:PRINT"▼◆GAME OVER":END
480 GOSUB 1300
490 FOR I=1 TO LEN(WURD$)
500 IF MID$(WURD$,I,1)<>A$ THEN 520
510 POKE 1718+I,PLACE+1
520 NEXT I
530 FOR I=1 TO 400:NEXT I:POKE 54276,0
540 POKE 1872,32
550 KOUNT=KOUNT-CHAR(PLACE)
560 CHAR(PLACE)=-1:IF KOUNT>0 THEN 150
570 GOSUB 1300:ROW=23:GOSUB 1400
580 PRINT"YOU WIN!":FOR I=1 TO 1000:NEXT I
590 POKE 54276,0:GOTO 420
600 REM SCAFFOLD
610 PRINT "▼";SPC(8);"▄▄▄▄▄▄▄▄▄▄▄▄"
620 PRINT SPC(8);"▌ ▀";SPC(5);"▚    ▢"
```

```
630 PRINT SPC(8);"□";SPC(6);"▧   □"
640 PRINT SPC(8);"□";SPC(7);"▧ □"
650 PRINT SPC(9);"▧▬";SPC(6);"▧□"
660 PRINT SPC(18);"□"
670 PRINT SPC(18);"□"
680 PRINT SPC(18);"□"
690 PRINT SPC(18);"□"
700 PRINT SPC(18);"□"
710 PRINT SPC(18);"□"
720 PRINT SPC(18);"□"
730 PRINT SPC(18);"□"
740 PRINT SPC(4);"■                    ▄▄"
750 RETURN
800 REM DRAW PARTS
810 ON P GOTO 820,870,920,950,980,1020,1060,
    1090
820 REM HEAD
830 PRINT "■■■";TAB(9);"◪*◱"
840 PRINT TAB(9);"◫ ◫"
850 PRINT TAB(11);"◰"
860 RETURN
870 REM BODY
880 PRINT "■■■■■■";TAB(10);"■ "
890 PRINT TAB(10);"■ "
900 PRINT TAB(10);"■ "
910 RETURN
920 REM LEFT ARM
930 PRINT "■■■■■■";TAB(8);"▬▬"
940 RETURN
950 REM RIGHT ARM
960 PRINT "■■■■■■";TAB(11);"▬▬"
970 RETURN
980 REM LEFT LEG
990 PRINT "■■■■■■■■■";TAB(9);"◪"
1000 PRINT TAB(8);"◪"
1010 RETURN
1020 REM RIGHT LEG
1030 PRINT "■■■■■■■■■";TAB(11);"▧"
1040 PRINT TAB(12);"▧"
```

```
1050 RETURN
1060 REM HANDS
1070 PRINT "{S}{Q}{Q}{Q}{Q}{Q}{]}{]}{]}{]}{]}{]}{]}{]}
     □{]}{]}{]}□"
1080 RETURN
1090 REM FEET
1100 PRINT "{S}{Q}{Q}{Q}{Q}{Q}{Q}{Q}{Q}{Q}{]}{]}{]}{]}{]}{]}{]}
     ▄{]}{]}{]}{]}{]}▄"
1110 RETURN
1200 REM BADBEEP
1210 POKE 54272,132
1220 POKE 54273,3
1230 POKE 54277,160
1240 POKE 54278,250
1250 POKE 54296,15
1260 POKE 54276,33
1270 RETURN
1300 REM GOODBEEP
1310 POKE 54272,32
1320 POKE 54273,78
1330 POKE 54277,34
1340 POKE 54278,138
1350 POKE 54296,15
1360 POKE 54276,33
1370 RETURN
1400 REM POSITION CURSOR AT ROW,COL
1410 SPOT=1024+ROW*40+COL
1420 POKE 209,SPOT AND 255
1430 POKE 210,INT(SPOT/256)
1440 POKE 211,0
1450 RETURN
1500 REM LOAD DEATH MARCH
1510 FOR I=1 TO 50:READ A$:NEXT I
1520 FOR I=0 TO 10
1530 READ F(I,0),F(I,1),F(I,2),F(I,3)
1540 NEXT I
1550 RETURN
1600 REM PLAY DEATH MARCH
1610 FOR I=54277 TO 54291 STEP 7
```

```
1620 POKE I,24:POKE I+1,137
1630 NEXT I
1640 POKE 54296,15
1650 FOR I=0 TO 10
1660 FOR J=0 TO 2
1670 POKE J*7+54272,F(I,J) AND 255
1680 POKE J*7+54273,INT(F(I,J)/256)
1690 NEXT J
1700 FOR J=54276 TO 54290 STEP 7
1710 POKE J,33
1720 NEXT J
1730 FOR J=1 TO F(I,3):NEXT J
1740 FOR J=54276 TO 54290 STEP 7
1750 POKE J,32
1760 NEXT J
1770 NEXT I
1780 RETURN
1800 REM WORDS
1810 DATA"BENEFIT","INSIST","PLEASURE"
1820 DATA"STORM","ACRONYM","TRAPEZE"
1830 DATA"WORLDLY","FUNGUS","MARKET"
1840 DATA"YESTERDAY","GUIDANCE","YEARN"
1850 DATA"ENGAGE","RECLAIM","KNOWLEDGE"
1860 DATA"COLONY","UTILITY","JOKE"
1870 DATA"DRIVER","REMAINDER","HYDROGEN"
1880 DATA"OCTOPUS","TECHNICAL","LAUGH"
1890 DATA"QUIVER","VOLTAGE","NIMBLE"
1900 DATA"FOLIAGE","AMBITION","VALVE"
1910 DATA"GEOMETRY","LOCATION","GLACIER"
1920 DATA"OUTRAGEOUS","WARRANT","KICK"
1930 DATA"CIVILIZE","MATERNITY","HEART"
1940 DATA"DISARRAY","ZOMBIE","BLUEPRINT"
1950 DATA"MILDEW","IMPRESS","JINGLE"
1960 DATA"PERFECT","EXCESSIVE","QUOTA"
1970 DATA"NAVIGATE","SEPARATE"
2000 REM MUSIC
2010 DATA 3608,4291,5407,250
2020 DATA 3608,4291,5407,200
2030 DATA 3608,4291,5407,5
```

```
2040 DATA 3608,4291,5407,300
2050 DATA 4291,5407,0,170
2060 DATA 4050,4817,0,5
2070 DATA 4050,4817,0,170
2080 DATA 3608,4291,0,5
2090 DATA 3608,4291,0,170
2100 DATA 3406,4050,0,5
2110 DATA 3608,4291,0,300
```

GLOSSARY OF VARIABLES

CHAR An array of twenty-six elements numbered 0 through 25. Each element contains a value indicating how many of the corresponding letters occur in the secret word. Setting an element to −1 indicates that the letter has already been correctly guessed by the player. Setting the location to −2 indicates that the player has guessed the letter but the letter does not occur in the secret word.

F A matrix whose rows are numbered 0 through 10 and whose columns are numbered 0 through 3. This matrix is used to hold the death-march tune. Each row represents a note of the tune. The first three columns (0 through 2) represent the frequencies to be played by the three voices. The final column contains a value that determines how long each note is to be sustained.

WURD$ Holds the selected secret word.

P Records how many wrong guesses have been made and also indicates which part of the man is to be drawn next.

PLACE This variable is used in several different ways. In general, it converts a given letter to its corresponding sequence number between 0 and 25.

ROW Indicates the row at which the cursor is to be positioned.

COL Indicates the column at which the cursor is to be positioned.

KOUNT This indicates how many letters of the secret word have not been guessed by the player.

LINES OF SPECIAL INTEREST

40–70: Randomly selects a word from the pool of 50 words.

100–140: For each letter of WURD$, increment the corresponding element of CHAR and print a dash on the screen.

170–210: Get a letter from the player and assign to PLACE the corresponding sequence number (0–25). If the player types in a question mark (?), control passes to line 400, where the answer is revealed and the dirge is played.

220: If the corresponding location of CHAR is equal to 0, the player has made an incorrect guess.

230: If the corresponding location of CHAR is greater than 0, the player has made a correct guess.

240–310: The player has already selected the letter. If it is not present in the secret word (line 260 tests for this), then the copy of the letter in the list of wrong guesses is momentarily highlighted in orange (lines 270–280). Line 290 deletes the message printed in line 250.

320–340: The player has made an incorrect guess. The offending letter is placed in the list of incorrect guesses (below the dashes) in a location determined by the value of PLACE.

350: After a short time the low-pitched tone is shut off.

360: This POKE blanks out the selected letter that appears after the question mark.

370: Recall that setting an element of CHAR to −2 indicates that the corresponding letter was guessed by the player but does not occur in the word.

380–410: P is equal to 8 so the player has lost. The correct word is displayed in place of the dashes (see line 400).

470: Notice that the program ends here rather than at the end of the main routine.

490–520: Since the player has made a correct guess, the secret word is scanned to find all the occurrences of the guessed letter. In line 510 the expression 1718+I calculates the appropriate location in screen memory. The dash in that location is replaced with the guessed letter. Since the screen display codes for letters range from 1 to 26, the expression PLACE+1 produces the correct code.

560: Setting the appropriate element of CHAR to −1 indicates that the letter was guessed and occurs in the secret word.

570–590: The player has guessed all the letters and therefore wins.

Subroutine
600–750: Clears the screen and prints the scaffold.

610: The graphics character used is Commodore key-I. There are eleven of them present.
620: In the second literal the graphics characters used are shift-M and Commodore key-M. There are three spaces between them. These graphics characters are also the ones used in the second literals of the lines 630 and 640. However, in line 630 there are two spaces between these graphics characters and in 640 there is only one space between them.
630–640: The graphics character used in the first literal of both these lines is Commodore key-M.
650: The graphics characters in the first literal are shift-M and Commodore key-I. The characters in the second literal are shift-M and shift–at-sign (@).
660–730: In all these lines the graphics character used is Commodore key-M.
740: There are seventeen spaces within the literal.

Subroutine
800–1110: This subroutine draws the various parts of the body in the appropriate locations.

810: Transfers control to the appropriate line based upon the value of P.
830–1100: Graphics characters used in this subroutine, other than the control characters referenced in Table 1 in the Introduction, are as follows:

Line No.	Graphics Characters
830	Shift-U and shift-I
840	Shift-B (or shift-minus)
850	Shift-K
930	Commodore key-Y
960	Commodore key-Y
990	Shift-N
1000	Shift-N
1030	Shift-M
1040	Shift-M
1070	Shift-O and shift-P
1100	Commodore key-O

In lines 1070 and 1100 make sure to count the cursor characters carefully, and type them as one string with no spaces.

Subroutine
1200–1270: Plays a low tone to indicate a wrong letter. The tone is turned on here and turned off in the main routine.

Subroutine
1300–1370: Plays a high tone to indicate a correct letter (or a win). Again, the tone is only turned on here.

Subroutine
1400–1450: Repositions the cursor as specified by ROW and COL.

Subroutine
1500-1550: Loads the notes for the death march into matrix F.

1510: Since the notes are found in DATA statements following the word list, this line skips over the words to reach the notes.

Subroutine
1600–1780: Plays the death march.

1610–1630: Load the same ADSR values into all three voices. (See *Programmer's Reference Guide* for further details.)
1650–1770: Play all eleven chords.
1660–1690: For each voice, load the appropriate note of the chord into the frequency registers.
1700–1720: Turn on all the voices.
1730: The value in column 3 of matrix F is used to determine the length of this delay loop. The voices remain on for this length of time.
1740–1760: Terminate all notes of the chord.

1800–1970: The pool of fifty words.
2000–2110: The notes for the death march.

SUGGESTED ENHANCEMENTS

1. The vocabulary list can always be changed at will.
2. The length of the vocabulary list can be altered. Be sure to change lines 40 and 1510, however.
3. To make the game more difficult, have fewer body parts for the man. Among other changes, this would require changes to line 370.
4. Have the computer print out a performance rating showing the number of successful and unsuccessful games.
5. Change the vocabulary to that of a foreign language in order to help gain greater familiarity with the language.

CHAPTER 14

RUN HARRY!

In this game you assume the role of Harry, the formidable alien hunter. You are represented on the screen as a little white box. Shortly after the game commences aliens represented by black boxes start to appear and actually chase you, regardless of where you might be located, in much the same way that iron filings follow a magnet. However, you have several major advantages over the aliens. In the first place, you can run off the edge of the screen and reappear on the opposite edge. The aliens cannot pursue you off the edge of the screen but must turn around and run in the opposite direction. Also, at random intervals, some friends of yours will appear on the screen in the form of gray at-signs (@). If you succeed in engulfing one of these at-signs, your increased energy has the effect of changing the color of the aliens to purple. They will then run away from you; if you succeed in catching them, they will die and you will earn ten points for each alien eaten. (The score is constantly displayed at the top of the screen.) Be careful, though, because after a while the aliens will turn black again and you will have no power over them. Shortly before they change to black, they will turn light red, in order to alert you to their impending transformation. If one of the aliens catches you while it is colored black, you will turn green and die an ignominious death, accompanied by a plaintive tone.

Incidentally, to control Harry you need to use four keys. The up/down cursor key makes him go to the left, and the left/right cursor makes him move to the right. You can control these keys with your right hand. The A and Z keys makes him move up and down respectively; these keys can be operated with the left hand. These keys have been selected because we felt they would allow the greatest efficiency in controlling Harry's movements.

```
10 REM RUN HARRY!
20 DIM AR(19),AC(19)
30 ROW=10:COL=20
40 TTL=0:CNT=-1:S=0:KOLR=0
50 PRINT "■";0
60 FOR J=1 TO 40:PRINT "-";:NEXT J
70 POKE 1444,160:POKE 55716,1
80 CNT=CNT+1
90 AR(CNT)=INT(RND(0)*2)*22+2
100 AC(CNT)=INT(RND(0)*2)*39
110 SPOT=1024+40*AR(CNT)+AC(CNT)
120 IF PEEK(SPOT)<>32 THEN 90
130 POKE SPOT,160
140 POKE SPOT+54272,0
150 NXT=TIME+INT(RND(0)*6)*60+300
160 IF INT(RND(0)*(30-CNT))=7 THEN GOSUB 800
170 GOSUB 300:IF S<0 THEN 290
180 I=0
190 GOSUB 300:IF S<0 THEN 290
200 IF CNT<0 THEN 230
210 GOSUB 600:IF S<0 THEN 290
220 IF S=0 THEN 240
230 GOSUB 900
240 I=I+1:IF I<=CNT THEN 190
250 IF CNT=19 THEN 170
260 IF TIME<NXT THEN 160
270 IF S=1 THEN 170
280 GOTO 80
290 GOSUB 1600:PRINT "■GAME OVER":END
300 REM MOVE HARRY
```

```
310 A=PEEK(197)
320 IF A=2 THEN 390
330 IF A=10 THEN 420
340 IF A=12 THEN 450
350 IF A<>7 THEN RETURN
360 POKE 1024+40*ROW+COL,32
370 COL=COL-1:IF COL<0 THEN COL=39
380 GOTO 470
390 POKE 1024+40*ROW+COL,32
400 COL=COL+1:IF COL=40 THEN COL=0
410 GOTO 470
420 POKE 1024+40*ROW+COL,32
430 ROW=ROW-1:IF ROW<2 THEN ROW=24
440 GOTO 470
450 POKE 1024+40*ROW+COL,32
460 ROW=ROW+1:IF ROW=25 THEN ROW=2
470 SPOT=1024+40*ROW+COL
480 IF PEEK(SPOT)<>32 THEN 520
490 POKE SPOT,160
500 POKE SPOT+54272,1
510 RETURN
520 GOSUB 1000
530 RETURN
600 REM MOVE AN ALIEN
610 POKE 1024+40*AR(I)+AC(I),32
620 DR=SGN(ROW-AR(I)):IF S>0 THEN DR=-DR
630 DC=SGN(COL-AC(I)):IF S>0 THEN DC=-DC
640 IF DR=0 THEN 670
650 IF DC=0 THEN 710
660 ON INT(RND(0)*2) GOTO 710
670 AC(I)=AC(I)+DC
680 IF AC(I)>=0 AND AC(I)<=39 THEN 750
690 AC(I)=AC(I)-DC
700 DR=INT(RND(0)*2)*2-1
710 AR(I)=AR(I)+DR
720 IF AR(I)>=2 AND AR(I)<=24 THEN 750
730 AR(I)=AR(I)-DR
740 DC=INT(RND(0)*2)*2-1:GOTO 670
```

```
750 SPOT=1024+40*AR(I)+AC(I)
760 IF PEEK(SPOT)<>32 THEN 790
770 POKE SPOT,160:POKE SPOT+54272,KOLR
780 RETURN
790 GOSUB 1200:RETURN
800 REM PUT AN @
810 X=INT(RND(0)*40)
820 Y=INT(RND(0)*23)+2
830 SPOT=1024+40*Y+X
840 IF PEEK(SPOT)<>32 THEN 810
850 POKE SPOT,0
860 POKE SPOT+54272,15
870 RETURN
900 REM TURN THE TABLES
910 TURNS=TURNS-1:IF TURNS>8 THEN RETURN
920 IF TURNS=0 THEN 940
930 KOLR=10:GOTO 950
940 S=0:KOLR=0
950 IF CNT<0 THEN RETURN
960 FOR J=0 TO CNT
970 POKE 54272+40*AR(J)+AC(J),KOLR
980 NEXT J
990 RETURN
1000 REM HARRY COLLISION
1010 IF PEEK(SPOT)=0 THEN 1040
1020 IF S=1 THEN 1120
1030 S=-1:RETURN
1040 POKE SPOT,160:POKE SPOT+54272,1
1050 IF CNT<0 THEN RETURN
1060 TURNS=50:IF S=1 THEN RETURN
1070 S=1:KOLR=4
1080 FOR J=0 TO CNT
1090 POKE 54272+40*AR(J)+AC(J),4
1100 NEXT J
1110 RETURN
1120 GOSUB 1300
1130 RETURN
1200 REM ALIEN COLLISION
```

```
1210 IF PEEK(SPOT)=160 THEN 1250
1220 IF INT(RND(0)*2) THEN RETURN
1230 POKE SPOT,160:POKE SPOT+54272,KOLR
1240 RETURN
1250 IF (PEEK(SPOT+54272) AND 15)<>1 THEN RETURN
1260 S=-1
1270 RETURN
1300 REM KILL AN ALIEN
1310 POKE SPOT,102:POKE SPOT+54272,12
1320 GOSUB 1500
1330 PTS=0:J=0
1340 IF AR(J)<>ROW OR AC(J)<>COL THEN 1410
1350 PTS=PTS+1:CNT=CNT-1
1360 IF J>CNT THEN 1430
1370 FOR K=J TO CNT
1380 AR(K)=AR(K+1):AC(K)=AC(K+1)
1390 NEXT K
1400 GOTO 1340
1410 J=J+1
1420 IF J<=CNT THEN 1340
1430 POKE SPOT,160:POKE SPOT+54272,1
1440 POKE 54276,32:POKE 54296,0
1450 TTL=TTL+10*PTS
1460 PRINT "■";STR$(TTL)
1470 RETURN
1500 REM ALIEN DEATH NOISE
1510 POKE 54272,195:POKE 54273,16
1520 POKE 54277,0:POKE 54278,240
1530 POKE 54296,15:POKE 54276,33
1540 RETURN
1600 REM HARRY DEATH THEME
1610 POKE SPOT+54272,5
1620 POKE 54272,143:POKE 54273,10
1630 POKE 54277,34:POKE 54278,242
1640 POKE 54296,15:POKE 54276,33
1650 FOR J=1 TO 500:NEXT J
1660 POKE 54272,104:POKE 54273,9
1670 FOR J=1 TO 100:NEXT J
1680 POKE 54272,12:POKE 54273,7
```

```
1690 FOR J=1 TO 300:NEXT J
1700 POKE 54276,32:POKE 54296,0
1710 POKE 198,0
1720 RETURN
```

GLOSSARY OF VARIABLES

AR An array of twenty elements numbered 0 through 19 which represents the row locations of up to twenty aliens which can occupy the screen at any given time.

AC An array of twenty elements numbered 0 through 19 which represents the column locations of up to twenty aliens.

ROW Harry's row location.

COL Harry's column location.

TTL Player's total points.

CNT Count of the number of aliens (minus 1) on the screen at any particular time. It actually indicates the largest active subscript in AR and AC, since the aliens are numbered from 0 up to 19. If CNT is equal to −1, there are no aliens.

S This variable equals 0 under normal conditions. It is set to −1 when Harry is destroyed and is set to 1 after Harry has eaten a friendly at-sign.

KOLR This variable indicates the current color of the aliens.

NXT A randomly selected number indicating the time at which the next alien appears on the board. The time delay between the appearance of the aliens ranges from five to ten seconds, in increments of one second. The exception is when the aliens are colored purple, at which time no new aliens are introduced. (It is obvious why NEXT wasn't used for this variable.)

I Indicates which alien is currently being moved.

DR In the subroutine at line 600 this variable determines whether an alien is to move up or down. This is based upon the direction in which Harry is moving and who is chasing whom.

DC In the subroutine at line 600 this variable determines whether an alien is to move to the right or to the left.

TURNS This variable specifies how many moves Harry must make before the purple aliens turn black again.

PTS Used to count how many aliens Harry kills at one time. This variable exists basically for the reason that, unlike

us humans, more than one alien can occupy the same space at the same time. If they do occupy the same spot and one is engulfed, both are engulfed.

LINES OF SPECIAL INTEREST

70: Harry is placed on the screen at a fixed point.

80–140: A new alien is placed on the screen. He enters from one of the four corners, which accounts for the strange appearance of lines 90 and 100. Notice that the top row of the playing field is row 2. This was done so that the area displaying the points is not overrun.

150: Calculates the time at which the next alien is introduced.

160: Randomly determines whether a new at-sign should appear. Notice that the more aliens there are, the greater the chance of a new at-sign appearing. (The choice of the number 7 in this line was purely arbitrary. Any number less than 10 would have served just as well.)

170: Harry moves once, regardless of how many aliens there are.

180–240: For each alien on the screen, Harry is moved first and then the alien. Note that each time Harry moves, he may decrease the number of aliens, which is why a FOR/NEXT loop is not used here.

250: If there are already twenty aliens on the screen, don't try to add any more.

260: If the time delay between aliens has not expired, do not add another alien.

270: If the existing aliens are purple, don't add any more aliens.

280: Transfers control back to line 80 to add a new alien.

Subroutine

300–530: Moves Harry one space. Notice that if he moves past the edge of the screen, he is placed at the opposite edge. A test is also made in this subroutine to see whether Harry has collided with something. This could be either an at-sign or an alien, and the subroutine at line 1000 is invoked to perform further testing.

Subroutine

600–790: This routine moves the alien indicated by variable I, regardless of whether he is pursuing or fleeing Harry. A test is made to see whether the alien has collided with something.

620–630: The direction in which the alien moves is based upon

the direction from him to Harry. If Harry is pursuing him, the alien will go in a direction opposite to that which he would take if he were pursuing Harry.

640–650: The alien can move only one row or one column at a time, but not both. If he is in the same row as Harry, he moves one column; if he in the same column, he moves one row.

660: If the alien is in a different row and a different column from Harry, he must randomly choose whether he wants to move by one row or one column. This line uses a form of the ON...GOTO statement that works on the Commodore 64 but is actually "cheating." If the value returned by the INT function is 1, control transfers to line 710. If the resulting value is 0, the ON...GOTO statement is ignored and control passes down to the next line, which is 670.

690–700: The alien tried to move past the left or right edge of the screen, which it is not allowed to do. (Note that this happens only when the alien is fleeing Harry.) So it remains in the same column and moves either up or down one row.

730–740: The alien cannot move past the top or bottom of the screen so it moves left or right one column instead. Because of these lines and lines 690–700, an alien caught in a corner may have some trouble getting out.

Subroutine

800–870: This subroutine randomly places an at-sign on the screen. It is always placed in a vacant location on the screen.

Subroutine

900–990: This subroutine performs more functions than the accompanying REM statement would indicate. It is used to count down the variable TURNS; furthermore, when the value of TURNS reaches 8, this subroutine changes all the aliens to light red. When the value of TURNS reaches 0, this subroutine resets S to 0 and turns all the aliens black.

Subroutine

1000–1130: This is invoked when Harry bumps into something.

1030: If Harry has bumped into an alien and the alien is colored black (S = 0), S is set to −1.

1040–1050: Harry has bumped into an at-sign. Harry's image is then placed in that location. If there are no aliens on the screen, no further action is taken.

1060: The number of moves during which the aliens remain purple

is set to 50. This happens even if the aliens are already purple, so it is always beneficial to eat at-signs. If the aliens are already purple at this point, the subroutine performs no further action.

1070–1100: Set S to 1 and make all the aliens purple.

1120: This line is reached if Harry has collided with a purple (or light red) alien.

Subroutine

1200–1270: This subroutine is invoked if an alien crashes into something.

1220: If an alien hits an at-sign, half the time he passes just behind it. He reappears when he moves away from the at-sign.

1230: Half the time the alien obliterates the at-sign. The at-sign does not reappear.

1250: If the alien has hit another alien, nothing happens. This is why two aliens can occupy the same space. Since they are represented by separate locations in AR and AC, they still remain distinct.

1260: The alien has "caught" Harry.

Subroutine

1300–1470: The aliens at the location specified by ROW and COL are summarily liquidated.

1310: The value of SPOT comes from the subroutine at line 300, by way of the subroutine at line 1000. The value 102 is the screen display code for a checkerboard pattern that represents a vanishing alien.

1330–1420: These lines scan through the collection of all aliens and eliminate those which are at the current location. Since the number of aliens is decreased in these lines, a FOR/NEXT loop could not be used.

1350: The tally of aliens destroyed is incremented and the count of existing aliens is decremented.

1360–1390: The remaining aliens are shifted down one element in AR and AC. Thus, elements 0 through CNT in both arrays still all represent active aliens even though one has just been removed. This complicated way of doing things is dictated by the desire to keep the aliens physically arranged in the sequence in which they appeared on the screen.

1400: The same subscript of AR and AC is used again because a new alien now occupies this element.

1430: Places Harry in the current location.

1440: Turns off the alien death noise.
1450–1460: Add 10 points for each alien killed.

Subroutine
1500–1540: Alien death noise.

Subroutine
1600–1720: A sad farewell to Harry is sounded.

1610: Turns Harry green (not with envy!).
1710: Clears out the keyboard buffer. This is done so that stray As and Zs do not appear when the program ends.

SUGGESTED ENHANCEMENTS

1. You may change the keys that control the movement of Harry if you find that other keys are more convenient. To do this you will need to refer to Table 2 in Chapter 6.
2. In the current version of the game, the aliens stay purple for a fixed amount of time. You may instead wish to make this variable.
3. The number of points awarded for catching an alien may be adjusted as desires dictate; or, you may wish to award ten points for the first alien, twenty for the second, and so forth.

CHAPTER 15

WALLPAPER

The time has come to relax a little. The next program is a visual delight devoid of any brainteasing challenges. All you have to do when this program is run is to sit back and enjoy, once you have supplied some simple information. The program opens up with the question: How many patterns? This can be any positive integer at all. It will then ask: How many colorations? The response to this is also a positive integer. The program fills the screen with two alternating characters. These are not characters from the machine's normal character set but are randomly designed by the program and are not meant to represent anything in particular. The number of patterns requested determines how many times these characters are redesigned. Furthermore, the number of colorations determines how many times the background and foreground colors are changed for each pattern. Lastly, the program adds one more feature. For each coloration, the pattern on the screen is displayed in two different Commodore 64 graphics modes, to provide for greater variety.

This program makes considerable use of the Commodore 64's graphics capabilities. We shall not attempt to go into great detail

about these facilities. For more information we respectfully refer you to the *Programmer's Reference Guide*.

```
10 REM WALLPAPER
20 PRINT:INPUT "HOW MANY PATTERNS";P
30 IF P=INT(P) AND P>0 THEN 60
40 PRINT "POSITIVE INTEGERS ONLY, PLEASE"
50 GOTO 20
60 PRINT:INPUT "HOW MANY COLORATIONS";C
70 IF C=INT(C) AND C>0 THEN 100
80 PRINT "POSITIVE INTEGERS ONLY, PLEASE"
90 GOTO 60
100 POKE 53272,PEEK(53272) OR 14
110 FOR I=1024 TO 2023 STEP 2
120 POKE I,0:POKE I+1,1
130 NEXT I
140 FOR Q=1 TO P
150 FOR I=14336 TO 14351
160 POKE I,INT(RND(0)*256)
170 NEXT I
180 FOR R=1 TO C
190 FOR I=53281 TO 53283
200 POKE I,INT(RND(0)*16)
210 NEXT I
220 A=INT(RND(0)*8)+8
230 B=INT(RND(0)*8)+8
240 FOR I=55296 TO 56295 STEP 2
250 POKE I,A:POKE I+1,B
260 NEXT I
270 POKE 53270,PEEK(53270) OR 16
280 FOR I=1 TO 2000:NEXT I
290 POKE 53270,PEEK(53270) AND 239
300 NEXT R
310 NEXT Q
320 POKE 53272,(PEEK(53272) AND 240) OR 4
330 POKE 53281,6
340 PRINT "■"
350 END
```

GLOSSARY OF VARIABLES

P	The user's selected number of patterns.
C	The user's selected number of colorations.
Q	A loop index which counts through each of the patterns.
R	A loop index which counts through each of the colorations.
A	The randomly selected color of the first random character.
B	The randomly selected color of the second random character.

LINES OF SPECIAL INTEREST

100: The lower four bits of location 53272 determine where the character dot patterns are located. Setting these bits to 14 specifies location 14336. This is where we shall place our random character definitions.

110–130: Normally, this would fill the screen with alternating atsigns (@) and As. However, since we have changed the location of the character dot patterns, we can cause screen display code 0 and 1 to represent any randomly designed characters.

150–170: These statements randomly design two characters. Each character definition takes up 8 bytes of memory, so the program must poke into 16 bytes.

190–210: The program selects random colors for the background and for the characters themselves.

220–230: These are the random colors for the two characters. They must be between 8 and 15 in order for the characters to be displayed in multicolor mode. Multicolor mode means that the characters are displayed in three colors rather than one. One of these colors is determined by the values selected in these lines. The other two colors come from locations 53282 and 53283. (See lines 190–210.) Though the values for A and B range from 8 through 15, they are interpreted as the colors numbered 0 through 7.

240–260: Set the alternating characters to their appropriate colors.

270: Setting bit 4 of location 53270 to 1 turns on the multicolor mode.

290: Multicolor mode is turned off. Now the characters are displayed in a single color rather than in three. Their character colors, as defined by the variables A and B, are interpreted as the colors numbered 8 through 15.

320: Location 53272 is restored to its original value. The normal character set is now in effect.

SUGGESTED ENHANCEMENTS

1. You might be tempted to investigate further the Commodore 64's advanced graphics capabilities and try to figure out ways to make the patterns displayed by this program even more arresting.
2. At times you may wish to be able to repeat an especially appealing pattern. A routine can be written so that upon request the program informs you of all the colors and character patterns that are in effect at any particular moment. Only about twenty-one numbers between 0 and 255 would be involved.

CHAPTER 16

SIMON SAYS

A favorite group game played by both youngsters and adults alike is the game of Simon Says. As you will remember, the leader (generally referred to as Simon) carries out some conspicuous action and the players are supposed to repeat the action only if Simon gives the instruction after saying "Simon says." If the action is repeated by a player without it being preceded by "Simon says," the player leaves the game. A talented Simon (and there seem to be so many of them!) can very quickly eliminate most of the participating audience. The last person to remain is considered the winner.

The rules for the Commodore 64 version of the game are fewer although the game is not quite as simple. The computer flashes a random sequence of numbers 0 through 9 on the screen, each number accompanied by its own color and tone. As soon as the sequence is finished, the player must key in the numbers in the exact same order in which they were originally flashed by the computer. To make the life of the player easier, it is not necessary to press the RETURN key to enter a sequence. Neither is it necessary to do anything special when the entered sequence of numbers has been completed. The computer takes care of all this itself. However, the sequence must be entered within a certain

time interval. There are ten levels of play, with the easiest being level 0. As the player proceeds from this level to the higher levels, so the time period allowed for repeating the sequence decreases. Not only that, but the speed at which the computer displays the sequence is increased.

Within each level, the game proceeds as follows. The program flashes a single number which the player must repeat. The computer then flashes a sequence of two numbers, the first of which is the one flashed previously. If the player successfully repeats the sequence, the computer flashes three numbers, the first two of which were flashed earlier. This process continues up to twenty, each time adding one more number to the preceding sequence. If the player successfully repeats the sequence of all twenty numbers (good luck!), he passes on to the next higher level, starting once again with a sequence of one number, but this time at a faster speed.

When you first run the program, you are asked for the level at which you wish to start. This is a number between 0 and 9. It is not necessary to press the RETURN key, since the computer accepts the first digit that is pressed.

```
10 REM SIMON SAYS
20 DIM SEQ(19),COLR(9)
30 FOR I=0 TO 9:READ COLR(I):NEXT I
40 DATA 1,2,7,6,0,13,3,8,9,10
50 PRINT "WHAT'S YOUR DESIRED LEVEL (0-9)? ";
60 GET A$:IF A$<"0" OR A$>"9" THEN 60
70 PRINT A$
80 L=ASC(A$)-48
90 PRINT "♥ Q Q Q Q Q Q Q Q Q Q"
100 FOR LEVEL=L TO 9
110 PRINT TAB(6);"YOU ARE NOW ENTERING
    LEVEL";LEVEL
120 FOR Z=1 TO 1000:NEXT Z:PRINT "♥"
130 SPEED=(10-LEVEL)*50
140 FOR I=0 TO 19
150 SEQ(I)=INT(RND(0)*10)
```

```
160 FOR J=0 TO I
170 GOSUB 400
180 NEXT J
190 POKE 198,0
200 FOR J=0 TO I+1
210 DELAY=TIME+(11-LEVEL)*25
220 GET A$:IF A$>="0" AND A$<="9" THEN 260
230 IF TIME<=DELAY THEN 220
240 IF J>I THEN 280
250 GOSUB 700:GOTO 360
260 IF ASC(A$)-48<>SEQ(J) THEN 250
270 GOSUB 400
280 NEXT J
290 NEXT I
300 POKE 53281,7
310 PRINT "♥ Q Q Q Q Q Q Q Q Q Q Q";TAB(15);
    "VERY GOOD!"
320 TN=44:GOSUB 800
330 NEXT LEVEL
340 PRINT TAB(11);"YOU'RE AN EXPERT!"
350 TN=66:GOSUB 800
360 POKE 54296,0
370 POKE 53280,14:POKE 53281,6
380 END
400 REM DISPLAY A NUMBER
410 POKE 55795,SEQ(J)
420 POKE 53280,SEQ(J)
430 POKE 53281,COLR(SEQ(J))
440 POKE 1523,SEQ(J)+48
450 Q=SEQ(J):IF Q=0 THEN Q=10
460 T1=(Q/2+3)*2000
470 T2=T1*2:T3=T2*2
480 POKE 54273,INT(T1/256)
490 POKE 54280,INT(T2/256)
500 POKE 54287,INT(T3/256)
510 IF T2>32767 THEN T2=T2-65536
520 IF T3>32767 THEN T3=T3-65536
530 POKE 54272,T1 AND 255
```

```
540 POKE 54279,T2 AND 255
550 POKE 54286,T3 AND 255
560 POKE 54275,8:POKE 54282,8
570 POKE 54289,8
580 POKE 54277,28:POKE 54278,249
590 POKE 54284,28:POKE 54285,249
600 POKE 54291,28:POKE 54292,249
610 POKE 54276,65:POKE 54283,65
620 POKE 54290,65:POKE 54296,15
630 FOR Z=1 TO SPEED:NEXT Z
640 POKE 54276,64:POKE 54283,64
650 POKE 54290,64
660 FOR Z=1 TO SPEED*.65:NEXT Z
670 POKE 1523,32
680 FOR Z=1 TO SPEED*.35:NEXT Z
690 RETURN
700 REM BADBEEP
710 POKE 54273,2:POKE 54280,7
720 POKE 54275,8:POKE 54282,8
730 POKE 54277,31:POKE 54284,31
740 POKE 54278,244:POKE 54285,244
750 POKE 54296,15
760 POKE 54276,65:POKE 54283,65
770 FOR Z=1 TO 650:NEXT Z
780 POKE 54276,64:POKE 54283,64
790 RETURN
800 REM GOODBEEP
810 POKE 54273,TN:POKE 54280,TN*2
820 POKE 54275,8
830 POKE 54277,15:POKE 54284,15
840 POKE 54278,240:POKE 54285,240
850 POKE 54296,15
860 POKE 54276,65:POKE 54283,33
870 FOR Z=1 TO 500:NEXT Z
880 POKE 54273,TN*8/11
890 POKE 54280,TN*16/11
900 FOR Z=1 TO 100:NEXT Z
910 POKE 54273,TN*14/11
```

```
920 POKE 54280,TN*28/11
930 FOR Z=1 TO 1000:NEXT Z
940 POKE 54276,64:POKE 54283,32
950 RETURN
```

GLOSSARY OF VARIABLES

SEQ An array of twenty elements numbered 0 through 19 which stores the sequence being flashed by the computer.

COLR An array of ten elements numbered 0 through 9 that specifies the background colors to be used for each digit when it is flashed on the screen. (The reason why we could not use COLOR is that it contains the keyword OR.)

L The player's starting level.

LEVEL A FOR/NEXT index which specifies the level at which the game is currently being played.

SPEED Determines the amount of time between flashes. Since this variable is used in a delay loop, a higher value for SPEED actually causes a *longer* delay between flashes.

I A FOR/NEXT index which specifies the length of the current sequence.

DELAY Determines how much time the player has to enter each number of the sequence.

TN Specifies the frequency to be produced by the subroutine at line 800.

J This variable serves several different purposes. In lines 160–180 it specifies which number in the sequence is to be flashed. (Not the value of the number but its order in the sequence.) In lines 200–280, J specifies which number is being entered by the player.

LINES OF SPECIAL INTEREST

150: Adds one more random number to the sequence.

160–180: Flashes the sequence of numbers.

190: The keyboard buffer is cleared. This is done so that the player cannot begin keying in the sequence until the program has finished displaying it.

200: There are two good reasons for this loop going from I + 1 rather than I. The first is that this allows the player time to make the

mistake of entering one more number than he is supposed to; that is to say, the computer does not stop him as soon as all the numbers of the sequence have been entered. The second is that if the player does enter the sequence correctly, he is permitted time to breathe before the next sequence is flashed. (This is the same amount of time as is specified by the variable DELAY.)

220–230: The program waits for input as long as the time specified by DELAY is not exceeded.

240: If the player does not enter a number during the (I + 1)th iteration of the loop, then he has entered the sequence correctly. Control then passes to line 280 where the loop terminates immediately (since this is the last iteration) and the next sequence is initiated.

250: If this line is reached after line 240, then the player has entered one more number than is allowed. The game is therefore lost and is terminated. If control passes to this line from line 260, then the player loses because a wrong number was entered.

260–270: The number entered by the player is compared to the number that he should have entered. If the number is correct, it is flashed on the screen. Otherwise, control passes to line 250.

310–320: The player is congratulated for completing a level.

340–350: The player is congratulated for passing the final (ninth) level.

Subroutine

400–690: This subroutine flashes a number, also producing an appropriate tone, setting the background and border colors to those assigned to the specific number. Most of the lines in this subroutine are involved in producing the tone.

410: The number being flashed is also used as the character color of the digit which is displayed at the center of the screen.

420: The character color is also used as the screen's border color.

430: The contents of array COLR are used to determine the background (screen) color.

440: The number's screen display code is placed in screen memory at the location corresponding to the center of the screen.

450: The variable Q is used to determine the frequency of the tone. Higher numbers represent higher tones. However, to make the game easier for the player (about time!) the number 0 is given a higher frequency than 9 because it is located to the right of the 9 key on the keyboard.

460–470: The variables T1, T2, and T3 specify the frequencies for each of the three voices in the sound chip.

510–520: This is done because the AND operator does not work on numbers larger than 32767. If T2 or T3 exceeds this limit, it is converted to a negative number with the same bit pattern.

630: During this delay period, the number is displayed on the screen and the tone is sounded.

660: During this delay period the number remains on the screen but the tone is no longer heard.

680: During this delay period there is no number displayed on the screen. (In the 3 lines mentioned above, the total delay time is actually 2 * SPEED.)

Subroutine

700–790: Plays a low tone consisting of two voices when the player loses.

Subroutine

800–950: Plays a happy tune when the player completes a level or wins the game. The variable TN has the value 44 for the former case and 66 for the latter. Two voices are employed by this subroutine.

SUGGESTED ENHANCEMENTS

1. The maximum number of digits in a sequence can always be changed to suit a different audience.
2. As you become proficient at this game, more levels may be added.

CHAPTER 17

MATH WHIZ

Here's an opportunity to brush up on your arithmetic skills. What this program does is to provide you with an unlimited series of elementary arithmetic problems consisting of addition, subtraction, multiplication, and division. These are randomly selected and the numbers on which the operations take place are also randomly selected. As a result, no session is identical to any other.

The program prints all the problems in column form, positioning the cursor at the point where the answer should be entered. You may enter spaces before the number in order to line it up with the numbers above it. However, the program allows you to enter the answer just about anywhere on the line, with as many leading blanks or zeros as you wish. If the correct answer is entered, the computer beeps. If the wrong answer (or no answer at all) is entered, the computer hisses and replaces your incorrect answer with the correct one. The entire space of the screen is used for displaying problems, moving from left to right and from top to bottom. Once the screen is filled up, the program pauses a moment for you to see the last answer and then clears the screen, starting again from the top and resuming the problem set.

Whenever you feel that "enough is enough," you may press the F1 key. This terminates the session with a dispassionate display of your performance rating.

```
10 REM MATH WHIZ
20 RTN$=CHR$(13):BLK$=CHR$(32):POKE 53281,1
30 DIM FUN$(3)
40 FOR I=0 TO 3:READ FUN$(I):NEXT I
50 DATA "+","-","X","/"
60 PRINT "■";:ROW=0:COL=0:CNT=0:YES=0
70 T=INT(RND(0)*4)
80 ON T+1 GOTO 90,130,170,210
90 A=INT(RND(0)*100)
100 B=INT(RND(0)*100)
110 C=A+B
120 GOTO 240
130 A=INT(RND(0)*100)
140 B=INT(RND(0)*(A+1))
150 C=A-B
160 GOTO 240
170 A=INT(RND(0)*31)
180 B=INT(RND(0)*10)
190 C=A*B
200 GOTO 240
210 C=INT(RND(0)*31)
220 B=INT(RND(0)*30)+1
230 A=B*C
240 GOSUB 400
250 GOSUB 800:IF D=-9 THEN 380
260 CNT=CNT+1:IF D=C THEN 310
270 GOSUB 1000
280 GOSUB 600
290 L=3:S=C:GOSUB 700
300 GOTO 320
310 YES=YES+1:GOSUB 1100
320 ROW=ROW-3:COL=COL+5
330 IF COL<40 THEN 70
340 COL=0
350 ROW=ROW+5
360 IF ROW<25 THEN 70
370 FOR I=1 TO 1000:NEXT I:GOTO 60
380 GOSUB 1300:POKE 54296,0
```

```
390 END
400 REM PRINT PROBLEM
410 PRINT "■";
420 GOSUB 600
430 L=3:S=A:GOSUB 700
440 ROW=ROW+1
450 GOSUB 600
460 PRINT FUN$(T);
470 L=2:S=B:GOSUB 700
480 ROW=ROW+1
490 GOSUB 600
500 PRINT "----";
510 ROW=ROW+1
520 GOSUB 600
530 PRINT "■";
540 RETURN
600 REM POSITION CURSOR AT ROW,COL
610 POKE 211,0
620 SPOT=1024+ROW*40+COL
630 POKE 209,SPOT AND 255
640 POKE 210,INT(SPOT/256)
650 RETURN
700 REM PRINT S AT CURRENT SCREEN POS.
710 LL=3
720 IF S<100 THEN LL=2
730 IF S<10 THEN LL=1
740 IF LL=L THEN 760
750 FOR I=1 TO L-LL:PRINT " ";:NEXT I
760 PRINT STR$(S);
770 RETURN
800 REM INPUT D (USER'S ANSWER)
810 D=-1:M=0:ANS$=""
820 PRINT "■ ■■";
830 GET A$:IF A$="" THEN 830
840 IF A$=RTN$ THEN 970
850 IF A$<>"■" THEN 910
860 IF ANS$="" THEN 830
870 IF RIGHT$(ANS$,1)=BLK$ THEN 890
```

```
880 M=M-1
890 ANS$=LEFT$(ANS$,LEN(ANS$)-1)
900 PRINT " ■■■■";:GOTO 820
910 IF A$=BLK$ THEN 960
920 IF A$>="0" AND A$<="9" THEN 950
930 IF A$<>"■" THEN 830
940 D=-9:GOTO 990
950 M=M+1
960 PRINT A$;:ANS$=ANS$+A$:GOTO 820
970 PRINT " ";:IF M=0 THEN 990
980 D=VAL(ANS$)
990 RETURN
1000 REM WRONG NOISE
1010 POKE 54272,184
1020 POKE 54273,11
1030 POKE 54277,34
1040 POKE 54278,34
1050 POKE 54296,15
1060 POKE 54276,129:FOR Q=1 TO 100:NEXT Q
1070 POKE 54276,0
1080 RETURN
1100 REM RIGHT NOISE
1110 POKE 54272,168
1120 POKE 54273,97
1130 POKE 54274,0
1140 POKE 54275,8
1150 POKE 54277,34
1160 POKE 54278,34
1170 POKE 54296,15
1180 POKE 54276,65:FOR Q=1 TO 100:NEXT Q
1190 POKE 54276,0
1200 RETURN
1300 REM PERFORMANCE RATING
1310 POKE 53281,6
1320 PRINT "■■■■■■■";TAB(14);"■YOUR
     SCORES:"
1330 PRINT:PRINT
1340 PRINT TAB(9);"CORRECT      INCORRECT"
```

```
1350 PRINT TAB(9);"--------------------"
1360 PRINT:PRINT:PRINT TAB(10);YES;
1370 PRINT TAB(23);CNT-YES
1380 FINE=INT(YES/CNT*100)
1390 PRINT:PRINT TAB(9);"(";FINE;"■■% )";
1400 PRINT TAB(22);"(";100-FINE;"■■% )"
1410 RETURN
```

GLOSSARY OF VARIABLES

FUN$	A string array of four elements numbered 0 through 3 which contains the symbols for the four operations.
RTN$	Contains the RETURN character. (Is used to save space in the listing of the program.)
BLK$	Contains the space (or blank) character. This is used because of a possible bug in the Commodore 64 which sometimes substitutes the shifted space for the regular space character.
ROW	Specifies the row for cursor positioning.
COL	Specifies the column for cursor positioning.
CNT	Counts the number of problems answered, for use in the calculation of the performance rating.
YES	Specifies the number of problems answered correctly, for use in calculating the performance rating.
T	A randomly selected number between 0 and 3 which specifies which operation is to be used in the current problem.
A	The first of the two operands in a problem.
B	The second of the two operands in a problem.
C	The correct answer to the problem as found by the computer.
D	The answer entered by the player. This variable is set to −1 if the player makes no answer and to −9 if the player presses the F1 key.
L	The maximum length of a number to be printed for use in the subroutine at line 700.
S	Used to hold the value to be printed by the subroutine at line 700.
M	In the subroutine at line 800 this variable is used to count the number of digits entered by the user.
ANS$	Holds the answer entered by the player in character form.

LINES OF SPECIAL INTEREST

90–120: The operation is that of addition. Both A and B range between 0 and 99.

130–160: This is subtraction. Line 140 ensures that B is never greater than A, so that the result is never negative.

170–200: This is multiplication. For the sake of simplicity, B is kept to 1 digit. A is limited to the value 30 for compatibility with division (see below).

210–230: This is division. To ensure that the division comes out evenly, B and C are calculated and then A is derived from those values. B and C are limited to the value 30 so that A is never larger than three digits. Notice that B can never be 0; division by 0 is undefined in mathematics.

250: A value of −9 for D indicates that F1 is pressed and the program terminated.

260: CNT is incremented only if the user has not pressed F1 in the last problem.

270–300: The user gave the wrong answer or failed to answer. Line 290 displays the correct answer.

310: The user answered correctly.

320–360: Modifies ROW and COL to indicate the location at which the problem is to be displayed.

370: Pauses before clearing the screen.

Subroutine

400–540: Displays the problem at the current location. Notice that the values for L and S are set before calling the subroutine at line 700. For the second operand, L=2 because the second number is never larger than two digits. Notice also that every PRINT statement ends with a semicolon to retain the effectiveness of the cursor repositioning.

Subroutine

600–650: Repositions the cursor as specified by ROW and COL.

Subroutine

700–770: Prints the value of S at the current cursor location in a field whose width is specified by L.

710–730: The variable LL determines the actual number of digits in S.

740–750: If LL is less than L, print leading blanks before printing S. This causes all the numbers to be right justified.

Subroutine
800–990: Inputs the value for D.

810: If no digits are entered by the time the user presses RETURN, D will still contain −1.
860: The user cannot backspace if he has not entered any characters or if he has already backspaced over all of them.
870–880: The user is backspacing; if the last character entered is not a blank, then the digit count is decremented.
890: The last character entered is deleted from ANS$.
940: Setting D to −9 indicates that F1 has been pressed.
950: The user has pressed a digit, so the digit count is incremented.
960: Print the character and add it to the end of ANS$.
970: If no digits have been entered (the player may have entered blanks, though), the value of D is left at −1.

Subroutine
1000–1080: Produces a noise indicating that the player has entered either the wrong answer or no answer at all.

Subroutine
1100–1200: Produces a tone signaling that the user has entered a correct answer.

Subroutine
1300–1410: Displays the user's performance rating.

SUGGESTED ENHANCEMENTS

1. Larger numbers may be used in the problems to increase their difficulty. By the same token, smaller numbers will make the problems easier. Be aware, however, that this change would require extensive modifications since the width of the numbers will change. You may have to fit fewer problems on a screen. Also, take care to keep the selected numbers within workable ranges.
2. Modify the program to work with negative numbers as well.

CHAPTER 18

PHONE-Y WORDS

No matter what the telephone company public relations people may say about the ease with which seven-digit telephone numbers are remembered, the fact of the matter is that for many of us, having to remember a telephone number consisting of seven digits (rather than the two letters of a telephone exchange followed by five digits, such as in KL5–3891) requires superhuman powers of memory, which regrettably we don't all possess. Our trusty friend, the Commodore 64 computer, is able to lighten the burden somewhat.

If you take a look at the dial of the typical American phone, you will notice that the number 2 is associated with the letters A, B, and C. In the same way, the number 3 is associated with the letters D, E, and F, and so forth. (Unfortunately, the numbers 0 and 1 do not have any letters at all associated with them so any telephone number containing one or more of these digits is automatically rejected as a candidate for conversion.)

In this program the user is invited to type in any seven-digit telephone number. The computer then exhaustively displays all the possible words that can be formed from that number by using the letters that are associated with its digits. For example, the telephone number 776-4726 can produce such words as

PPMGPAM, RSOISAN, and SPOGPAN. When any one of these admittedly peculiar looking words is dialed, the correct telephone number is rung. However, there are no fewer than 2,187 different combinations that exist for the number. (This is 3^7—you can check it on your computer.) The speed and power of the computer being what it is, all of these possible combinations can be displayed on the screen before your very eyes. If this is done, you will find that among this extraordinarily large number of possibilities is the word PROGRAM. Now, it is our contention that it is much easier to associate the word PROGRAM with that telephone number than to have to remember the seven digits. It is with that in mind that we present our phone-y word program.

As soon as the program is run you are asked to enter the seven-digit phone number without the hyphen. The program does not accept any character other than the digits 2 through 9, since these are the only digits that have corresponding letters associated with them. In addition, a number with fewer than or more than seven digits is similarly rejected. Should you make a typing error when entering the number, you may backspace as much as needed. Once you are sure the number is correct, the RETURN key is pressed. The program responds by asking for your preference of background color. This must be an integer between 0 and 15 and corresponds to the standard codes for the Commodore 64. You are then asked for the color in which the words are to be displayed. Once again, an integer between 0 and 15 is required. The program, however, does not accept a character color that is identical with the background since this would result in a blank screen. The screen is then filled with the first group of words produced by the phone number entered. In view of the enormous number of words that are produced by this program, the screen will soon become full. In order to give the viewer an opportunity to scan the words in comfort for a possible candidate, the program pauses after displaying a screenful of words (there are 110 words per screenful). When the viewer is ready to see more words, the F1 key is pressed. The screen is then cleared and is immediately filled with another screenful of words. If at any time the viewer wishes to terminate the listing, the F7 key may be pressed. At the bottom of every screen is a message reminding the viewer of

the functions of the F1 and F7 keys. Once the listing has been completed, the program pauses, waiting for the viewer to press the F7 key. When this is done, the viewer is asked if another number is to be tried.

```
10 REM PHONE-Y WORDS
20 DIM LTTER$(9,2),N(6)
30 DATA A,B,C,D,E,F,G,H,I,J,K,L
40 DATA M,N,O,P,R,S,T,U,V,W,X,Y
50 FOR I=2 TO 9
60 FOR J=0 TO 2
70 READ LTTER$(I,J)
80 NEXT J
90 NEXT I
100 PRINT:PRINT "ENTER PHONE NO. - 7 DIGITS";
110 PRINT " (NO HYPHEN):"
120 GOSUB 200
130 GOSUB 400
140 PRINT:PRINT "ANOTHER (Y OR N)? ";
150 GET A$:IF A$<>"Y" AND A$<>"N" THEN 150
160 PRINT A$:PRINT
170 IF A$="Y" THEN 100
180 PRINT "▽FINISHED"
190 END
200 REM GET NUMBER
210 I=0
220 PRINT "■ ▬■■";
230 GET A$:IF A$="" THEN 230
240 IF A$>="2" AND A$<="9" THEN 310
250 IF A$=CHR$(13) THEN 360
260 IF A$<>"■" THEN 230
270 IF I=0 THEN 230
280 I=I-1
290 PRINT " ■■■";
300 GOTO 220
310 IF I>6 THEN 230
320 N(I)=ASC(A$)-48
330 I=I+1
340 PRINT A$;
```

```
350 GOTO 220
360 IF I<7 THEN 230
370 PRINT " "
380 RETURN
400 REM DISPLAY THE WORDS
410 PRINT:INPUT "BACKGROUND COLOR (0-15)";B
420 IF B<>INT(B) OR B<0 OR B>15 THEN 410
430 PRINT:INPUT "CHARACTER COLOR (0-15)";C
440 IF C<>INT(C) OR C<0 OR C>15 OR C=B THEN 430
450 POKE 53281,B
460 POKE 646,C:REM WELL, IT'S EASY
470 PRINT "■";:X=0
480 FOR C0=0 TO 2
490 FOR C1=0 TO 2
500 FOR C2=0 TO 2
510 FOR C3=0 TO 2
520 FOR C4=0 TO 2
530 FOR C5=0 TO 2
540 FOR C6=0 TO 2
550 PRINT LTTER$(N(0),C0);
560 PRINT LTTER$(N(1),C1);
570 PRINT LTTER$(N(2),C2);
580 PRINT LTTER$(N(3),C3);
590 PRINT LTTER$(N(4),C4);
600 PRINT LTTER$(N(5),C5);
610 PRINT LTTER$(N(6),C6);
620 PRINT " ";
630 X=X+1
640 IF X<110 THEN 660
650 GOSUB 800:IF X<0 THEN 760
660 NEXT C6
670 NEXT C5
680 NEXT C4
690 NEXT C3
700 NEXT C2
710 NEXT C1
720 NEXT C0
730 PRINT:PRINT:PRINT
740 PRINT "F7=END"
```

```
750 GET A$:IF A$<>"■" THEN 750
760 POKE 53281,6
770 PRINT "■■■ *** HOPE YOU FOUND A GOOD
    WORD! ***"
780 RETURN
800 REM WAIT FOR NEXT PAGE
810 X=0
820 PRINT
830 PRINT "F1=MORE";SPC(4);"F7=END"
840 GET A$:IF A$<>"■" AND A$<>"■" THEN 840
850 IF A$="■" THEN X=-1
860 PRINT "■";
870 RETURN
```

GLOSSARY OF VARIABLES

**LTTR$** This array holds the letters associated with each digit. There is admittedly some waste of memory space in this array, since the maximum row dimension is 9 (each row represents one digit) and therefore rows 0 and 1 are totally unused. To compensate for this, column 0 is put to work. The columns are numbered 0 to 2, providing room for the three letters associated with each digit. (Did you guess why we didn't use the variable name LETTER$? You're right, because it contains the keyword LET.)

N This is an array of seven elements numbered 0 through 6. It holds the seven digits in the phone number for processing.

I When this variable is used in the subroutine at line 200 it indicates how many digits of the phone number have been entered so far.

B Holds the inputted value of the background color.

C Holds the inputted value of the character color.

X Counts how many words have been printed on the current screen. Also, the subroutine at line 800 sets this variable to −1 when the viewer decides to prematurely terminate the listing.

C0–C6 These seven variables are the loop indices which indicate which letter is being printed for each digit.

LINES OF SPECIAL INTEREST

Subroutine

200–380: This subroutine inputs the phone number.

270: If there are no digits displayed on the screen, the user cannot backspace.

280: Move back to the previous digit.

310: If there are already more than 6 digits on the screen, this subroutine will not accept any more digits.

330: Notice that the variable I always indicates the next element of N to be filled.

360: A RETURN is not accepted until there are 7 digits on the screen.

370: Blanks out the final cursor symbol.

Subroutine

400–780: Gets the background and character colors, then prints all the words.

460: In a sense, this POKE is "cheating," since character color may be changed by printing one of the color-changing characters. However, location 646 controls the character color and this POKE is more concise than using a large ON...GOTO statement. This is the essence of the REM. Of course, cursor repositioning through the use of locations 209 through 214 could also be considered "cheating," since the cursor characters could perform the same function. Be that as it may...

650: If 110 words have been printed, wait for the viewer to press a function key. Then continue the FOR/NEXT loop or, if X = −1, terminate the listing.

730–750: The listing has finished normally and so the program just waits for the viewer to press F7.

Subroutine

800–870: Waits for the viewer to press a function key during the listing of the words.

810: Since a fresh screen is to be started, the count of the number of words on the screen is reset to 0.

850: If F7 was pressed, this case is indicated by setting X to −1.

860: No matter which key was pressed, the screen is cleared.

SUGGESTED ENHANCEMENTS

1. In this program advantage is taken of all the computer's features except sound. Perhaps you would like to make it a clean sweep.
2. Many hapless people suffer the terrible misfortune of having phone numbers which include 0s and 1s. An easy change which would ameliorate the situation is to allow the program to accept numbers of fewer than 7 digits. In that way, these people could, at least, form words with the useful part of their number, which is probably better than nothing. A slightly trickier change would be to have the program accept any seven-digit number and then display any 0s or 1s in their natural state, in place of any letter.

CHAPTER 19

TIC-TAC-TOE

There can't be many people in the world who haven't been delighted by this exceedingly popular game, one which has been around for centuries. Many programmers have written programs to simulate the playing of Tic-Tac-Toe in the short history of this technology; here is our Commodore 64 version, which we believe will please you.

In the unlikely event that you are not familiar with the game, two sets of parallel lines are drawn at right angles, resulting in nine areas into each of which an X or an O may be placed. Two competing players take turns entering their respective symbols. If one of the players succeeds in completing a horizontal, vertical, or diagonal line of his symbol, he is considered the winner.

Even though the game appears to be rather simple (as games go), a good player exercises considerable strategy when challenged. It is clear, for example, that given a choice, it is to the player's advantage to go first. The reason for this is simply that there are only nine boxes into which moves can be made. The player who has first crack can make five moves against the opponent's four. Not only that, but having the first move allows him to take the center box, which is common to both diagonals, the center horizontal, and the center vertical. At the same time,

good strategy demands that you always block your opponent's attempt to achieve a line of three adjacent (identical) symbols while trying to get a line of three yourself.

In this version of the game you are asked whether you opt for X or O. You make your selection by pressing the appropriate letter. Note, however, that whoever opts for the X symbol goes first. The playing board is displayed in the middle of the screen and below it are displayed the phrases MY TURN or YOUR TURN, whichever is appropriate. These phrases are presented from the computer's point of view. You make your move by positioning the cursor using the cursor keys. When the cursor is located in the square in which you wish to place your mark, the F1 key is pressed and the appropriate symbol is placed there. The cursor appears as a solid box. When it is in a space already occupied by either of the two symbols, this will be indicated by the symbol appearing in reversed graphics. At such time the F1 key is ineffective.

When either you or the computer scores "three in a row," the winning combination is displayed in reversed graphics and the message I WIN! or YOU WIN! appears below the playing board. If all the spaces are filled without either opponent achieving three in a row, the message IT'S A TIE is displayed. You are then asked if you wish to play again. If you do, you are again given an opportunity to choose your symbol.

Good luck. You may have a tough time beating the computer.

```
10 REM TIC-TAC-TOE
20 DIM T(8,2),SYM(2)
30 PRINT "DO YOU WANT TO BE X OR O? ";
40 GET A$:IF A$<>"X" AND A$<>"O" THEN 40
50 PRINT A$
60 SYM(2)=ASC(A$)-64:SYM(1)=39-SYM(2)
70 PL=1:IF SYM(2)=24 THEN PL=2
80 GOSUB 900:X=0:Y=0
90 FOR BOX=1 TO 9
100 IF PL=1 THEN 190
110 PRINT TAB(15);"YOUR TURN■"
```

```
120 ROW=1:COL=1:SPOT=1441
130 POKE 1441,PEEK(1441)+128:POKE 55713,14
140 GOSUB 1100
150 X=0:GOSUB 400
160 IF X=0 THEN 180
170 IF T(X,2)=3 THEN 310
180 PL=1:GOTO 280
190 PRINT TAB(15);" MY TURN □"
200 IF Y>0 AND T(Y,2)=0 THEN X=Y
210 GOSUB 600:FOR I=1 TO 1000:NEXT I
220 SPOT=1359+ROW*80+COL*2
230 POKE SPOT,SYM(1):POKE SPOT+54272,14
240 X=0:GOSUB 400
250 IF X=0 THEN 270
260 IF T(X,1)=3 THEN 310
270 Y=X:PL=2
280 NEXT BOX
290 PRINT TAB(15);"IT'S A TIE"
300 GOTO 360
310 GOSUB 1400
320 IF PL=1 THEN 350
330 PRINT TAB(15);" YOU WIN!"
340 GOTO 360
350 PRINT TAB(15);" I WIN!  "
360 PRINT:PRINT "AGAIN (Y OR N)? ";
370 GET A$:IF A$<>"Y" AND A$<>"N" THEN 370
380 PRINT A$:IF A$="Y" THEN 30
390 END
400 REM TEST FOR 3-IN-A-ROW
410 T(ROW,PL)=T(ROW,PL)+1
420 T(COL+3,PL)=T(COL+3,PL)+1
430 IF ROW<>COL THEN 450
440 T(7,PL)=T(7,PL)+1
450 IF ROW+COL<>4 THEN 470
460 T(8,PL)=T(8,PL)+1
470 IF T(ROW,PL)<2 THEN 500
480 IF T(ROW,3-PL)>0 THEN 500
490 X=ROW:RETURN
500 IF T(COL+3,PL)<2 THEN 530
```

```
510 IF T(COL+3,3-PL)>0 THEN 530
520 X=COL+3:RETURN
530 IF T(7,PL)<2 THEN 560
540 IF T(7,3-PL)>0 THEN 560
550 X=7:RETURN
560 IF T(8,PL)<2 THEN RETURN
570 IF T(8,3-PL)>0 THEN RETURN
580 X=8:RETURN
600 REM SEARCH FOR NEXT SPACE
610 IF X=0 THEN 760
620 IF X>3 THEN 670
630 ROW=X
640 FOR COL=1 TO 3
650 IF PEEK(1359+ROW*80+COL*2)=32 THEN RETURN
660 NEXT COL
670 IF X>6 THEN 720
680 COL=X-3
690 FOR ROW=1 TO 3
700 IF PEEK(1359+ROW*80+COL*2)=32 THEN RETURN
710 NEXT ROW
720 FOR ROW=1 TO 3
730 COL=ROW:IF X=8 THEN COL=4-ROW
740 IF PEEK(1359+ROW*80+COL*2)=32 THEN RETURN
750 NEXT ROW
760 ROW=2:COL=2:IF PEEK(1523)=32 THEN RETURN
770 FOR ROW=1 TO 3 STEP 2
780 FOR COL=1 TO 3 STEP 2
790 IF PEEK(1359+ROW*80+COL*2)=32 THEN RETURN
800 NEXT COL
810 NEXT ROW
820 FOR ROW=1 TO 3
830 FOR COL=1 TO 3
840 IF PEEK(1359+ROW*80+COL*2)=32 THEN RETURN
850 NEXT COL
860 NEXT ROW
870 RETURN
900 REM SET UP BOARD,CLEAR ARRAYS
910 PRINT "♥ ■ ■ ■ ■ ■ ■ ■ ■ ■"
```

```
920 PRINT TAB(17);"  "
930 PRINT TAB(17);"  "
940 PRINT TAB(17);"  "
950 PRINT TAB(17);"  "
960 PRINT TAB(17);"  "
970 PRINT:PRINT:PRINT
980 FOR I=1 TO 8
990 T(I,1)=0:T(I,2)=0
1000 NEXT I
1010 RETURN
1100 REM PLAYER PICKS A SPOT
1110 DR=0:DC=0
1120 GET A$:IF A$="" THEN 1120
1130 IF A$="" THEN 1220
1140 IF A$="" THEN 1240
1150 IF A$="" THEN 1260
1160 IF A$="" THEN 1280
1170 IF A$<>"" THEN 1120
1180 IF PEEK(SPOT)<>160 THEN 1120
1190 POKE SPOT,SYM(2)
1200 POKE SPOT+54272,14
1210 RETURN
1220 IF ROW=1 THEN 1120
1230 DR=-1:GOTO 1300
1240 IF ROW=3 THEN 1120
1250 DR=1:GOTO 1300
1260 IF COL=1 THEN 1120
1270 DC=-1:GOTO 1300
1280 IF COL=3 THEN 1120
1290 DC=1
1300 POKE SPOT,PEEK(SPOT)-128
1310 ROW=ROW+DR:COL=COL+DC
1320 SPOT=1359+ROW*80+COL*2
1330 POKE SPOT,PEEK(SPOT)+128
1340 GOTO 1110
1400 REM HIGHLIGHT 3-IN-A-ROW
1410 IF X>3 THEN 1480
1420 SPOT=1361+X*80
```

```
1430 FOR COL=1 TO 3
1440 POKE SPOT,PEEK(SPOT)+128
1450 SPOT=SPOT+2
1460 NEXT COL
1470 RETURN
1480 IF X>6 THEN 1550
1490 SPOT=1433+X*2
1500 FOR ROW=1 TO 3
1510 POKE SPOT,PEEK(SPOT)+128
1520 SPOT=SPOT+80
1530 NEXT ROW
1540 RETURN
1550 SPOT=1441:IF X=8 THEN SPOT=1445
1560 FOR ROW=1 TO 3
1570 POKE SPOT,PEEK(SPOT)+128
1580 SPOT=SPOT+78:IF X=7 THEN SPOT=SPOT+4
1590 NEXT ROW
1600 RETURN
```

GLOSSARY OF VARIABLES

T This matrix keeps track of how many of each symbol are in each row, column, and diagonal. T has 8 rows. The first three rows represent the three rows in the playing board. The second three rows of T represent the three columns on the board. The seventh row of T stands for the diagonal going from the top left corner of the board to the bottom right. The eighth row of T is for the other diagonal. T has 2 columns, one for each player. The computer is always player 1 and the human is always player 2.

SYM This array holds the screen display codes for each player's symbol. SYM(2) holds the symbol chosen by the human player. SYM(1) holds the other symbol to be used by the computer.

PL This variable indicates which player is currently taking its turn. The value 1 indicates the computer while 2 indicates the human.

X This variable is given a value by the subroutine at line 400. It is used to indicate a row in matrix T and therefore

represents one of the eight ways that exist to get three in a row. If X has a value between 1 and 8, this indicates that the player currently taking a turn has either two or three in a row in the row, column, or diagonal indicated by the value of the variable X. If the value of X is left at 0, this is a sign that no present or future wins are foreseen for the player at the present time. Incidentally, the value of X is also passed to the subroutine at line 600 for determining the computer's next move.

Y This variable is used to hold the value of X generated for the computer. It is needed so that when the computer goes the next time, it can see whether its previous turn left it with two in a row, which it can now try to convert to three in a row.

BOX This is the index of a FOR/NEXT loop that goes around 9 times. Each time around, one box of the playing board is filled by one of the two players.

ROW During the human's turn, this variable indicates the row of the playing board (1 through 3) where the cursor is located. When F1 is pressed, this indicates the row at which the human's symbol is to be placed. When it is the computer's turn, this variable indicates the row of the chosen move.

COL This serves a purpose similar to ROW except that it specifies a column.

SPOT Specifies a location in screen memory and is used to place symbols on the board.

DR In the subroutine at line 1100, this variable specifies the direction (up or down) in which the cursor is being moved.

DC This variable specifies the direction (left or right) in which the cursor is being moved in the subroutine at line 1100.

LINES OF SPECIAL INTEREST

60: Assigns the appropriate symbol to the corresponding player.

70: Decides which player goes first based upon who has the symbol X.

90–100: Each time through the loop only half of the lines in the loop are executed. Which half depends upon which player is currently taking a turn.

110–180: The human's turn.

120–130: The cursor is always initially positioned in the upper

lefthand box. Line 130 turns a space into a box or a symbol into its reversed-graphics equivalent.

160–170: If X is not equal to 0, and the row, column, or diagonal indicated by the value of X already contains three in a row, then the human has won.

180: Switch players and go to the next iteration of the loop.

190–270: The computer's turn.

200: If Y is greater than 0, it means that the computer had two in a row in its last turn. However, the human may have subsequently blocked this play by placing his symbol in the third box. T(Y,2) reflects how many symbols the human has in the row, column, or diagonal indicated by Y. If this value is 0, the computer can make the move that can give it three in a row. This is done by placing the value of Y into X. If, however, the computer will not be able to go for three in a row, the subroutine at line 600 gets the value of X that was generated during the human's move.

210: The delay loop in this line is used because the computer makes its moves incredibly fast. Were it not for this delay the human might be embarrassed.

220–230: The computer's symbol is placed at the appropriate location. Notice that in calculating the value for SPOT, ROW is multiplied by 80 and COL is doubled because there are two screen rows from one row of the playing board to the other. The same applies to the screen columns. Also, since the rows and columns on the playing board are numbered starting from 1 rather than 0, location 1359 is two screen rows and two screen columns away from the start of the playing board.

260: If the computer has three in a row, it wins.

270: The computer's value for X is saved in Y for its next turn. The players are switched.

290: If the FOR/NEXT terminates normally, this means that the game is a tie.

Subroutine

400–580: This subroutine increments the values in matrix T depending upon where the player's move was made. It then returns a value for X if the player has two in a row or three in a row anywhere on the board. This subroutine is used by both the computer and the human.

410–420: Every box is located in one row and one column. (Remember that columns 1 through 3 are represented by rows 4 through 6 of matrix T.)

430–440: The diagonal starting in the upper lefthand box consists of those boxes whose row numbers equal their column numbers. The diagonal is represented by row 7 of matrix T.

450–460: The other diagonal consists of boxes located such that their row value added to their column value yields 4. This diagonal is indicated by row 8 of matrix T.

470: If the player has less than 2 in a row (in the row in which its symbol was placed), no value will be assigned to X at this time.

480–490: The player has either two or three in a row; if the opponent has no symbols in this row, then either the player already has three in a row or he has two in a row with the third location being vacant. In either case, the value of this row is returned in X.

500–580: These lines are similar in format to lines 470–490, except that they examine the columns and the diagonals.

Subroutine

600–870: This subroutine determines the computer's next move. If the value of X is nonzero, then this subroutine places the computer's symbol in the vacant space of the row, column, or diagonal indicated by the value of X. This will be done either to give the computer three in a row or to block the human from getting three in a row. This depends upon whether the value of X is the one generated during the player's move or during the computer's previous move. If the value of X is 0, the computer selects a random location based upon a priority system to be described below.

630–660: X indicates a row. Search for the vacant column in this row and return from the subroutine when it is found. Upon return, ROW and COL indicate the vacant space. Line 650 is guaranteed to cause a return, so the FOR/NEXT loop never finishes and control never passes to line 670.

680–710: X indicates a column. Once again the vacant location in this column is found.

720–750: X indicates one of the two diagonals. Line 730 assigns the correct value to COL depending upon which diagonal is indicated. This loop is also guaranteed to return from the subroutine.

760–860: There are no two-in-a-rows on the board so the computer chooses the highest-priority space that is available. The center square has the highest priority. Next come the corner squares. The lowest priority is held by the remaining four squares. You will observe that lines 820–860, which only need to check the four low-priority squares, actually check every square on the board. Although this is somewhat redundant, it was simply easier to program this way.

Subroutine
900–1010: This subroutine clears the screen, draws the playing board, and zeros out matrix T.

Subroutine
1100–1340: This is the subroutine that moves the cursor (as controlled by the human player) and draws his symbol in the selected box when F1 is pressed.

1180: Since SPOT indicates the box where the cursor is located, a blank box actually appears as a solid square (screen display code 160).
1300: Changes the symbol in the current box from reversed graphics to its normal graphics equivalent. The reason for this is that the cursor is leaving that box.
1330: Indicates the new cursor location by changing the character (at the screen position indicated by SPOT) to reversed graphics.

Subroutine
1400–1600: Performs the task of highlighting three in a row by adding 128 to the screen display codes of the symbols in question, converting them to reversed graphics. The location of the three in a row is determined by the value of the variable X.

SUGGESTED ENHANCEMENTS

1. The musically inclined might be tempted to write some melodic accompaniment to the game.
2. The player may also be given a choice of color.
3. The ambitious programmer may be persuaded to amend the program so that the computer can play against itself.
4. There are variations on this game. One of them is a three-dimensional version, which you may consider implementing.

CHAPTER 20

ONE-ARMED BANDIT

If you are a fan of Las Vegas or Atlantic City, here is an opportunity to indulge yourself in the joys of the slot machine without having to move a foot out of your home—and you will probably be richer for the experience. This game is a Commodore 64 version of the one-armed bandit machine that lines the walls of the biggest casinos in the world.

The game opens up with the computer asking you how much money you have. The amount you enter can be an even number of dollars or an amount in dollars and cents. Nothing less than twenty-five cents is accepted, however. The program draws the windows of the slot machine. It also displays the amount of money you entered. The current balance is displayed at the top of the screen throughout the game.

Pressing the F1 key causes the machine to roll its three wheels after deducting twenty-five cents from your balance. The wheels contain four symbols: a bunch of grapes, a smiling face, a television set, and a bar sign. Each of the wheels contains six of these symbols; however, they are distributed differently on each

wheel. The first two wheels each contain two grapes, two faces, one TV, and one bar. The third wheel contains two grapes, one face, two TVs, and one bar. The result of this uneven distribution is that you are twice as likely to come up with three TVs as three bars; twice as likely to get three faces as three TVs, and twice as likely to get three grapes as three faces. The amount of money you win by coming up with three of a kind is based on the rarity of coming up with that combination. Thus the three bars give you $50, three TVs $25, three faces $12.50, and three grapes $6.25. Incidentally, for those of you interested in odds making, the awarding of money is biased slightly toward the house. (So what else is new?)

The wheels rotate a random number of times between one and twenty, with some delicate musical accompaniment. Whenever you wish to quit the game, you must press the F7 key. The program ends displaying the final balance. The game also ends if your balance falls below $ .25, since this means that you no longer have any quarters left to bet.

```
10 REM ONE-ARMED BANDIT
20 POKE 53281,1:POKE 54296,15
30 INPUT "■HOW MUCH MONEY DO YOU HAVE";CASH
40 IF VAL(STR$(CASH*100))<>INT(CASH*100+.1)
   THEN 60
50 IF CASH >= .25 THEN 80
60 PRINT:PRINT "DOLLARS AND CENTS >= .25,
   PLEASE"
70 PRINT:GOTO 30
80 GOSUB 500:GOSUB 300
90 GET A$:IF A$<>"■" AND A$<>"■" THEN 90
100 IF A$="■" THEN 230
110 CASH=CASH-.25
120 GOSUB 300
130 Q=INT(RND(0)*20)
140 FOR I=0 TO Q
150 GOSUB 800
160 NEXT I
170 IF M=0 THEN 210
```

```
180 CASH=CASH+50/2↑(4-M)
190 GOSUB 300
200 GOTO 90
210 IF CASH>=.25 THEN 90
220 FOR I=1 TO 2000:NEXT I
230 POKE 54296,0:POKE 53281,6
240 PRINT "■"
250 GOSUB 300
260 PRINT:PRINT "■GAME OVER"
270 END
300 REM PRINT CASH ON HAND
310 C$=MID$(STR$(CASH),2):L=LEN(C$):P=0
320 FOR J=1 TO L
330 IF MID$(C$,J,1)="." THEN P=J
340 NEXT J
350 IF P>0 THEN 370
360 C$=C$+".00":L=L+3:GOTO 390
370 IF P=L-2 THEN 390
380 C$=C$+"0":L=L+1
390 PRINT "■"
400 FOR J=0 TO (12-L):PRINT " ";:NEXT J
410 PRINT "■$";C$
420 RETURN
500 REM SET UP BOARD
510 PRINT "■■■■■■";TAB(3);
520 FOR J=1 TO 33:PRINT "■ ■";:NEXT J
530 PRINT
540 FOR J=1 TO 3
550 PRINT TAB(3);"■ ■";TAB(35);"■ ■"
560 NEXT J
570 PRINT TAB(3);"■ ■";TAB(6);
580 FOR J=1 TO 27:PRINT "■ ■";:NEXT J
590 PRINT TAB(35);"■ ■"
600 FOR J=1 TO 7
610 PRINT TAB(3);"■ ■";
620 FOR C=0 TO 2
630 PRINT TAB(C*9+6);"■ ■";SPC(7);"■ ■";
640 NEXT C
```

```
650 PRINT TAB(35);"■ ■"
660 NEXT J
670 PRINT TAB(3);"■ ■";TAB(6);
680 FOR J=1 TO 27:PRINT "■ ■";:NEXT J
690 PRINT TAB(35);"■ ■"
700 FOR J=1 TO 3
710 PRINT TAB(3);"■ ■";TAB(35);"■ ■"
720 NEXT J
730 PRINT TAB(3);
740 FOR J=1 TO 33:PRINT "■ ■";:NEXT J
750 PRINT:PRINT
760 PRINT TAB(3);"F1=ROLL";
770 PRINT TAB(30);"F7=END"
780 RETURN
800 REM ROLL THE SYMBOLS
810 FOR C=0 TO 2
820 S=INT(RND(0)*6)
830 IF S<2 THEN 910
840 IF C<2 AND S<4 THEN 900
850 IF S=2 THEN 900
860 IF C<2 AND S=4 THEN 890
870 IF S<5 THEN 890
880 T=4:GOTO 920
890 T=3:GOTO 920
900 T=2:GOTO 920
910 T=1
920 GOSUB 1000:GOSUB 1300
930 IF C=0 THEN M=T
940 IF C>0 AND M<>T THEN M=0
950 NEXT C
960 POKE 54276,32:POKE 54283,32
970 POKE 54290,32
980 RETURN
1000 REM PRINT A SYMBOL
1010 COL=C*9+8:PRINT "■■■■■■■■■■"
1020 ON T GOTO 1030,1090,1150,1210
1030 PRINT TAB(COL);"■  ◪  "
1040 PRINT TAB(COL);"●●●●●"
```

```
1050 PRINT TAB(COL);" ●●● "
1060 PRINT TAB(COL);"  ●  "
1070 PRINT TAB(COL);"     "
1080 RETURN
1090 PRINT TAB(COL);"▩◪▤▤▤◺"
1100 PRINT TAB(COL);"◫●  ●◫"
1110 PRINT TAB(COL);"◫  ♥  ◫"
1120 PRINT TAB(COL);"◫◹▤◸◫"
1130 PRINT TAB(COL);"◹▤▤▤◸"
1140 RETURN
1150 PRINT TAB(COL);"♣  ◹  ◸  "
1160 PRINT TAB(COL);"◪▤R▬▬▤◺"
1170 PRINT TAB(COL);"◫▩▩◎◫"
1180 PRINT TAB(COL);"◫▩▩◎◫"
1190 PRINT TAB(COL);"◹▤▤▤◸"
1200 RETURN
1210 PRINT TAB(COL);"77◪▤▤▤◺"
1220 PRINT TAB(COL);"◫▤▤▤◫"
1230 PRINT TAB(COL);"◫RBAR▬◫"
1240 PRINT TAB(COL);"◫▤▤▤◫"
1250 PRINT TAB(COL);"◹▤▤▤◸"
1260 RETURN
1300 REM NOISE
1310 BASE=54272+7*C
1320 POKE BASE+1,T*6
1330 POKE BASE+6,252
1340 POKE BASE+4,33
1350 RETURN
```

GLOSSARY OF VARIABLES

CASH Holds the player's current balance.

M This variable is returned by the subroutine at line 800. If it is equal to 0, this indicates that the player has not rolled three of a kind. If it has a value between 1 and 4, the player has succeeded in coming up with three of a kind, with the symbol indicated by the value of M: 1 stands for grapes, 2 is for faces, 3 is for TVs, and 4 is for bars.

C$ This variable holds the string equivalent of the player's balance.

L Holds the length of C$.

P In the subroutine at line 300 this variable is used to indicate the location of the decimal point in C$.

C This variable is used in several places, but in the subroutine at line 800 it indicates which wheel (0 through 2) is being rotated.

S Contains a random integer between 0 and 5 which indicates which of the symbols on each wheel is to be displayed. (Remember that each wheel has six symbols on it.)

T This variable is used in the subroutine at line 800. For the wheel currently being rotated, its value indicates the symbol on which the wheel comes to a halt. It takes a value between 1 and 4, which has the same meaning as for the variable M.

COL In the subroutine at line 1000 this variable determines at which screen column to print the current symbol.

LINES OF SPECIAL INTEREST

40: This line tests to see whether the number entered has more than two digits in its fractional portion. The expression to the left of the not-equal sign (<>) converts the result of CASH * 100 to a string and back to a number. This is done because the multiplication, as done by the Commodore 64, is sometimes too large by a small amount which is not visible when the result is printed but is detected by the machine. The STR$ function uses the value as displayed rather than the value as calculated. On the right side of the <>, in the INT function, 0.1 is added because sometimes the multiplication is too small by a small amount.

110–120: 25 cents is deducted before spinning the wheels.

140–160: Spin the wheels a random number of times. This is not really necessary, since in each spin the wheels are randomly repositioned. However, this makes for a more interesting visual display.

170–190: If M is nonzero, a three of a kind has come up. The value added to CASH is based on the odds of getting three of a kind in the symbol indicated by M.

220: The player has run out of money; this pause is so the player may view the result of his final roll before the screen is cleared.

Subroutine

300–420: Displays the player's balance at the top of the screen. This routine is somewhat tricky since it must always display the amount with two digits to the right of the decimal point.

310: The MID$ function is used to eliminate the leading blank that is generated by the STR$ function. P is initially set to 0 because if no decimal point is found in the lines that follow, it will be left with this value.

320–340: Search for a decimal point. If one is found, P is assigned its location in C$.

360: CASH is an integer so add a decimal point and two trailing 0s to C$.

380: CASH has only one digit to the right of its decimal point, so a trailing 0 is added.

400: Adds leading blanks to right-justify the value printed.

Subroutine

500–780: Prints the windows of the slot machine.

Subroutine

800–980: This subroutine rolls the wheels; that is, it selects the symbol to be displayed on each.

830–910: These lines convert the value of S to the appropriate symbol based upon which wheel is being rolled and how many of each symbol are on that wheel.

930: M is initially assigned the value of the symbol that appears on the first wheel.

940: If the second or third wheel displays a symbol different from that on the first wheel, then M is set to 0. If this occurs on the spin of the second wheel, then the value of T generated for the third wheel will never equal M since M is 0 and T is always nonzero.

960–970: Terminate the tones produced by the subroutine at line 1300. This is done for all three voices.

Subroutine

1000–1260: Displays the appropriate symbol at the proper column, based on the values of T and C.

1030–1250: Graphics characters used in this subroutine, other than the control characters referenced in Table 1 in the Introduction, are as follows:

Line Number	Graphics Symbol Used
1030	Shift-U
1040-1060	Shift-Q
1090	Shift-U, shift-C or shift-asterisk, shift-I
1100	Shift-B or shift-minus and shift-Q
1110	Shift-B or shift-minus and shift-S
1120	Shift-B or shift-minus, shift-J, shift-C or shift-asterisk, shift-K
1130	Shift-J, shift-C or shift-asterisk, shift-K
1150	Shift-M, shift-N
1160	Shift-U, shift-C or shift-asterisk, Commodore key-O, shift-I
1170-1180	Shift-B or shift-minus, Commodore key-plus, shift-W
1190	Same as 1130
1210	Same as 1090
1220, 1240	Shift-B or shift-minus, shift-C or shift-asterisk
1230	Shift-B or shift-minus
1250	Same as 1130

Take note that each PRINT statement displays five characters. Of course, when control characters (such as those that change character color or turn on reverse graphics) are used, the literals will actually contain more characters. And as stated in the Introduction, all control and graphics characters in these listings take up the space of two regular characters. So be careful in counting the number of blank spaces to be included in each literal. Line 1030 contains two spaces after both the control character and the

graphics character; line 1050 has one space at the beginning of the literal and one at the end; line 1060 has two spaces before the graphics character and two after; line 1070 contains five spaces; line 1100 has one space in the center of the literal; line 1110 has one space after the first graphics character and one after the second; line 1150 has one space after the control character and one space after both of the graphics characters.

Subroutine

1300–1350: Produces a tone whose frequency is based upon the value of T. The voice used is based on the value of C. Therefore, all three voices are sounded.

SUGGESTED ENHANCEMENTS

1. This might be a good opportunity for you to try to concoct your own symbols with the graphics characters available on the Commodore 64 to replace those symbols shown.
2. The number of symbols on each wheel can be changed along with their distribution. This way, you can change the odds of having three of a kind in any given symbol. You can also change line 180 to reflect this change in odds.

CHAPTER 21

PRINT-A-MONTH

Have you ever been in a situation where you are told a date which is in the distant future, and it is imperative for you to know in advance on what day of the week it falls? Not too many people have calendars on tap for all years! However, with the aid of your handy-dandy Commodore 64, this will never again present a problem to you—at least, once you have typed in the program that follows. Through the magic of the computer, it will now be possible for you to display on the screen the whole month for any year and month of your choice, be it in the distant future or the distant past. In view of the fact that the Gregorian calendar, by which we reckon dates today, was not consolidated until 1583, any year prior to that year is not accepted. For any date in or subsequent to 1583, you will have at your disposal what amounts to a perpetual calendar.

When the program is run, you are asked to enter the year in question. All the digits of the year are then typed in. It is then validated to ensure that it does not precede 1583, does not contain a decimal point, and is not a negative number. Once it has passed all these tests, the user is prompted to enter the month desired as an integer between 1 and 12. Once again, the number is validated and then the screen is cleared and in an instant, before your very eyes, a complete calendar for that month appears on

the screen. You will notice that Sundays are displayed in red. After the calendar of the month is displayed, the user is asked if there is a need for another run.

```
10 REM PRINT-A-MONTH
20 DIM DAYS(12),MNTH$(12)
30 DATA 1,31,28,31,30,31,30
40 DATA 31,31,30,31,30,31
50 DATA JANUARY,FEBRUARY,MARCH,APRIL
60 DATA MAY,JUNE,JULY,AUGUST
70 DATA SEPTEMBER,OCTOBER
80 DATA NOVEMBER,DECEMBER
90 FOR I=0 TO 12:READ DAYS(I):NEXT I
100 FOR I=1 TO 12:READ MNTH$(I):NEXT I
110 POKE 53281,5
120 INPUT "■WHAT YEAR";Y
130 IF Y=INT(Y) THEN 160
140 PRINT "HUH?"
150 GOTO 120
160 IF Y>=1583 THEN 190
170 PRINT "ONLY YEARS >= 1583, PLEASE"
180 GOTO 120
190 INPUT "WHAT MONTH (1-12)";M
200 IF M=INT(M) AND M>=1 AND M<=12 THEN 230
210 PRINT "AN INTEGER 1-12, PLEASE"
220 GOTO 190
230 PRINT "▼■■■■■■■";TAB(13);
    MNTH$(M);Y
240 PRINT:PRINT TAB(6);
250 PRINT "  ■S■    M   T   W   T   F   S"
260 PRINT TAB(6);
270 PRINT "--------------------------"
280 JDATE=Y+INT((Y-1)/4)-INT((Y-1)/100)
    +INT((Y-1)/400)
290 DAYS(2)=28
300 IF Y=INT(Y/4)*4 AND (Y<>INT(Y/100)*100
    OR Y=INT(Y/400)*400) THEN DAYS(2)=29
310 FOR I=0 TO M-1
```

```
320 JDATE=JDATE+DAYS(I)
330 NEXT I
340 DW=JDATE-INT(JDATE/7)*7
350 IF DW=0 THEN DW=7
360 FOR D=2-DW TO DAYS(M) STEP 7
370 PRINT TAB(6);"■";
380 FOR X=D TO D+6
390 IF X<1 OR X>DAYS(M) THEN 430
400 IF X<10 THEN PRINT " ";
410 PRINT X;"■";
420 GOTO 440
430 PRINT SPC(4);"■";
440 NEXT X
450 PRINT:PRINT
460 NEXT D
470 PRINT:PRINT:PRINT "■ANOTHER (Y OR N)? ";
480 GET A$:IF A$<>"Y" AND A$<>"N" THEN 480
490 PRINT A$
500 IF A$="Y" THEN 120
510 POKE 53281,6:PRINT "■FINISHED"
520 END
```

GLOSSARY OF VARIABLES

DAYS Elements 1 through 12 of this array hold the number of days contained by the corresponding month. DAYS(0) contains the value 1, which is always added into the computation.

**MNTH$** This string array holds the names of the months. The reason for not calling this variable by the more natural name MONTH$ is that the name contains the BASIC keyword ON.

Y Holds the inputted year.

M Holds the inputted month, an integer between 1 and 12.

JDATE This variable is used to compute the day of the week on which the first of the month falls. The computation involves finding what is referred to as the Julian date, the number between 1 and 365 (or 366, in the case of a leap year) representing the day of the year a particular date falls on. The variable JDATE is also used in other calculations.

DW A number between 1 and 7 which represents which day of the week the first of the month falls on. The number 1 stands for Sunday, 2 for Monday, and so on.

D A FOR/NEXT loop index whose values are the dates that begin each week.

X A FOR/NEXT index that takes the seven consecutive dates of the week, determined by D.

LINES OF SPECIAL INTEREST

50–80: Notice that this string data does not have quotation marks. On the Commodore 64 (but not on all computers) string data does not need quotation marks around it unless the data itself includes the comma character. We have used similar data without quotation marks in another program, namely "Phone-y Words" (Chapter 18).

280: The first value assigned to JDATE is the sum of the year number and the number of leap years preceding that year. This is similar to what was done in the program "Elapsed Days" (Chapter 3) to figure out how many days there are between the chosen date and the year 1. The reason Y is not multiplied by 365 is this: In line 340 below, the remainder of the division JDATE / 7 is taken. Since the remainder of 365 / 7 is 1, we need only add to JDATE the value 1 for each year preceding the year Y. Adding the number of leap years takes care of the fact that 366 / 7 is 2. (See "Elapsed Days" for the way to calculate leap years.)

290–300: If the year represented by Y is a leap year, set DAYS(2) to 29; otherwise, set it to 28.

310–330: Add to JDATE the number of days in the year up to and including the first day of the month represented by M. This added value is the actual Julian date. Notice how the value of DAYS(0) comes in handy to represent the first day of the month. If M were equal to 2, the Julian date would be DAYS(0)+DAYS(1) = 1+31 = 32, and the Julian date equivalent of February 1 is indeed 32.

340–350: DW is set to the integer remainder after dividing JDATE by 7, and then a value of 0 is replaced by 7.

360: The expression 2−DW determines the date of the first Sunday of the month, relative to the first of the month. In other words, if DW = 4, then the first of the month falls on a Wednesday. We therefore call the Tuesday preceding it the 0th of the month, the Monday before that the −1th day of the month, and the Sunday the −2th of the month. Indeed, 2−DW (in this case) would yield

−2. This strange method is used so that printing starts from the Sunday of the first week, even though blank fields might be involved.

370: Character color is set to red for Sunday.

390: Days before the first of the month and after the end of the month are filled with blanks.

400: Prints a leading blank for single-digit dates.

410: The character color is switched to blue after printing X, because X may have fallen on a Sunday.

SUGGESTED ENHANCEMENTS

1. You may want to indicate somehow that the year in question is or is not a leap year.
2. Since this program does not use sound, perhaps you might want to include some.
3. You may want to print each month with a different color.
4. A major amendment would be to arrange for the user to input a year and then be able to flip from month to month, forward ôr backward, using the function keys.

CHAPTER 22

FIND HARRY

There is no longer any need to journey to Hampton Court in London to become acquainted with a maze. The program you are about to see generates a random maze, one so large that it fills the whole of your screen.

When the program is run, Harry appears somewhere in the maze, at a random location. Harry is represented by a black box. Simultaneously with the appearance of Harry, his friend, the purple at-sign (@), is also placed on the screen, usually at a distant point from Harry. You control the movement of the at-sign, using the A key to move it up, the Z key to move it down, the up/down CRSR key to move left, and the left/right CRSR key to move right.

The object is to reach Harry. However, besides wending your way through the complex maze, there are two added difficulties. First, a number is displayed in the top left corner of the screen when play begins. This is the number of seconds you are allotted to reach Harry. It is initially set at 45. The number is counted down every second, and if you don't reach Harry by the time the clock reaches zero, the game ends. Second, the maze is constantly changing around the at-sign, so at one moment you may find yourself heading for an opening in the maze, only to find a barrier in your way in the next moment.

In order to reach Harry, you must reach a space adjacent to him (above or below, or to the left or to the right, but not diagonally adjacent) and then press the appropriate key that would move the at-sign into the space currently occupied by Harry. The at-sign does not actually move into the space, but a tone will be emitted indicating success.

Once you have successfully reached Harry, a new round begins with a completely different maze. However, the amount of time allocated is five seconds shorter than in the previous round. If you think this game sounds easy, you'll be a-mazed!

```
10 REM FIND HARRY
20 POKE 53281,7
30 LIMIT=45
40 S=0:GOSUB 200
50 PRINT "S ←";LIMIT
60 FINAL=TIME+LIMIT*60+59
70 GOSUB 500
80 IF S=0 THEN 70
90 IF S<0 THEN 140
100 GOSUB 1000
110 LIMIT=LIMIT-5:IF LIMIT>0 THEN 40
120 GOSUB 1000
130 GOTO 150
140 GOSUB 1100
150 PRINT "◆";:POKE 53281,6:POKE 198,0
160 END
200 REM SET UP MAZE
210 PRINT "▼"
220 FOR J=1 TO 40:PRINT "-";:NEXT J
230 FOR ROW=2 TO 24
240 B=ROW AND 1
250 FOR COL=1-B TO 38 STEP 2
260 SPOT=1024+ROW*40+COL
270 IF INT(RND(0)*2) THEN 300
280 POKE SPOT+54272,1
290 POKE SPOT,B+66
300 NEXT COL
```

```
310 NEXT ROW
320 HR=INT(RND(0)*23)+2
330 B=HR AND 1
340 HC=INT(RND(0)*20)*2+B
350 SPOT=1024+HR*40+HC
360 POKE SPOT+54272,0
370 POKE SPOT,160
380 DR=SGN(13.5-HR):DC=SGN(19.5-HC)
390 FR=INT(RND(0)*5)+2+(DR+1)*9
400 B=FR AND 1
410 FC=INT(RND(0)*9)+(DC+1)*15
420 SPOT=1024+FR*40+FC
430 POKE SPOT+54272,4
440 POKE SPOT,0
450 RETURN
500 REM CHANGE MAZE
510 FOR Y=-2 TO 2
520 FOR X=-2 TO 2 STEP 2
530 ROW=FR+Y
540 B=ROW AND 1:A=FC AND 1
550 COL=FC+X+ABS(1-(A+B))
560 IF ROW<2 OR ROW>24 THEN 670
570 IF COL<0 OR COL>39 THEN 660
580 IF ROW=FR AND COL=FC THEN 660
590 SPOT=1024+ROW*40+COL
600 IF INT(RND(0)*2) THEN 640
610 POKE SPOT+54272,1
620 POKE SPOT,B+66
630 GOTO 650
640 POKE SPOT,32
650 GOSUB 700:IF S<>0 THEN RETURN
660 NEXT X
670 NEXT Y
680 RETURN
700 REM MOVE FRIEND
710 DR=0:DC=0
720 A=PEEK(197):IF A=64 THEN 940
730 IF A=10 THEN 830
```

```
740 IF A=12 THEN 810
750 IF A=7 THEN 790
760 IF A<>2 THEN 940
770 IF FC=38 THEN 940
780 DC=1:GOTO 850
790 IF FC=0 THEN 940
800 DC=-1:GOTO 850
810 IF FR=24 THEN 940
820 DR=1:GOTO 850
830 IF FR=2 THEN 940
840 DR=-1
850 OLD=1024+FR*40+FC
860 SPOT=1024+(FR+DR)*40+FC+DC
870 IF PEEK(SPOT)<>32 THEN 930
880 POKE OLD,32
890 POKE SPOT+54272,4
900 POKE SPOT,0
910 FR=FR+DR:FC=FC+DC
920 GOTO 940
930 IF PEEK(SPOT)=160 THEN S=1
940 NOW=INT((FINAL-TIME)/60)
950 PRINT "S ";
960 IF NOW<10 THEN PRINT " ";
970 PRINT NOW:IF NOW>0 THEN RETURN
980 IF S=0 THEN S=-1
990 RETURN
1000 REM SUCCESS THEME
1010 POKE 54272,62:POKE 54273,42
1020 POKE 54277,34:POKE 54278,242
1030 POKE 54296,15:POKE 54276,33
1040 FOR J=1 TO 500:NEXT J
1050 POKE 54272,162:POKE 54273,37
1060 FOR J=1 TO 100:NEXT J
1070 POKE 54272,99:POKE 54273,56
1080 FOR J=1 TO 300:NEXT J
1090 POKE 54276,32:POKE 54296,0:RETURN
1100 REM FAILURE THEME
1110 POKE 54272,143:POKE 54273,10
```

```
1120 POKE 54277,34:POKE 54278,242
1130 POKE 54296,15:POKE 54276,33
1140 FOR J=1 TO 500:NEXT J
1150 POKE 54272,104:POKE 54273,9
1160 FOR J=1 TO 100:NEXT J
1170 POKE 54272,12:POKE 54273,7
1180 FOR J=1 TO 300:NEXT J
1190 POKE 54276,32:POKE 54296,0:RETURN
```

GLOSSARY OF VARIABLES

LIMIT	The amount of time allotted for reaching Harry.
FINAL	The actual time by which you must reach Harry. (This is a value that will be returned by the TIME function.)
S	This variable is initially set to 0. The subroutine at line 700 sets it to 1 if Harry is successfully reached and to −1 if the time limit runs out.
HR	The row at which Harry is located.
HC	The column at which Harry is located.
FR	The row at which the friendly at-sign is located.
FC	The column at which the friendly at-sign is located.
B	Determines whether the current screen row is even or odd.

LINES OF SPECIAL INTEREST

60: In this line the addition of 59 to the value of FINAL is an adjustment which is necessary because of the way the program tests for the time limit being exceeded.

100–110: The player has found Harry and another round is begun.

120–130: The player has miraculously found Harry in 5 seconds (the last round). A second success tone is played (the first one was played in line 100) and the program ends. We have no doubt at all that these lines will never be executed, and we don't mean because of a bug in the program!

140: The player has lost.

Subroutine

200–450: This subroutine draws the maze and places Harry and the at-sign on the screen.

240: The value of B is 0 if ROW is even and 1 if ROW is odd. There are two reasons for needing to know this information. First of all, even-numbered rows contain vertical bars (screen display code 66) while odd-numbered rows contain horizontal bars (screen display code 67). Also, in even-numbered rows the bars occur in odd-numbered columns while in odd-numbered rows the bars occur in even-numbered columns.

250: If ROW is even, COL ranges from 1 to 37 in steps of 2. If ROW is odd, COL ranges from 0 to 38.

290: The value of B is used to determine which graphics character is placed in screen memory.

330–340: Here B indicates whether HR is even or odd. This information is needed because in order to avoid placing Harry on a bar, he must be located either in an even row and an even column or an odd row and an odd column.

380: In order to place the at-sign as far away from Harry as possible, the screen is divided into quadrants. The at-sign is placed in the quadrant diagonally opposite the one that Harry is in. Furthermore, it is placed in a random location which is in close proximity to the edges of the screen. The variables DR and DC each receive the value -1 or 1, which are to determine in which quadrant the at-sign is to be placed.

390: This rather complicated expression places the at-sign randomly in one of the five rows closest to the top or bottom of the screen, depending upon which quadrant the at-sign is in.

400–410: Again to avoid collision with a bar, the at-sign must be placed in an even column if the row is even or an odd column if the row is odd. The manner in which the column is calculated is rather arcane, but if you wish to take the time to analyze it, you will see that it places the at-sign in the half of its quadrant that is closest to the edge of the screen.

Subroutine

500–680: This subroutine changes the maze in the five rows and five columns surrounding the at-sign.

510–520: The variables Y and X represent the row distance and column distance from the at-sign, rather than specific locations on the screen. This is done because the at-sign might move during the execution of the subroutine.

530: ROW indicates the row currently being changed.

540–550: COL indicates the column to be changed. This calculation is based on whether ROW is odd or even and whether the at-sign is at an odd or even column.

580: Don't change the maze where the at-sign is standing!

650: After making each change, gives the at-sign an opportunity to move. Control is returned to the main routine if either Harry is found or the time has expired.

Subroutine

700–990: In this subroutine the keyboard is checked and the at-sign is moved if an appropriate key is pressed. A test is also made to determine if Harry has been found. Whether or not the at-sign has been moved, the amount of time left (as displayed in the upper left corner of the screen) is updated and a test is made to see whether this amount has been reduced to 0.

870: The at-sign may not move onto a space occupied by a bar. If the location indicated by SPOT is actually occupied by Harry, this will be detected at line 930.

930: Harry has been reached.

940–970: Display the amount of time left. The variable NOW contains this value. In line 960 a blank is printed if NOW has only one digit.

980: S is set to -1 if the time has expired (in other words, NOW = 0). This is suppressed, however, if Harry is found simultaneously.

Subroutine

1000–1090: Plays a tune indicating that Harry has been found.

Subroutine

1100–1190: Plays a tune indicating that the time has expired.

SUGGESTED ENHANCEMENTS

1. The maze can be made either more dense or less dense, depending on whether you want to make the game more difficult or less difficult. This can be accomplished by modifying lines 270 and 600.
2. To add a little more diversity to the game, objects may be placed in the maze. Depending upon your preference, you may either require these to be avoided or award points if they are touched by the at-sign while trying to make contact with Harry.

CHAPTER 23

SCRAMBLER

Here, for a change, is a brainteaser. The computer displays a randomly selected word on the screen in jumbled order, and your job is to unscramble the word. Just so that you don't have too easy a time, you have to unscramble the word within a certain time limit, based on the level of difficulty selected and the length of the word. A low level number means that the time limit allocated is longer.

The game continues indefinitely, so it's the ideal game to give to a child from whom you need some reprieve! However, to terminate the program, all you have to do is to press the F1 key and the game ends with a performance rating displayed on the screen.

```
10 REM SCRAMBLER
20 POKE 53281,7:CNT=0:YES=0
30 INPUT"■WHAT SPEED DO YOU WANT (0-10)";N
40 IF N<>INT(N) OR N<0 OR N>10 THEN 30
50 RESTORE:Q=INT(RND(0)*50)+1
60 FOR I=1 TO Q
70 READ WURD$
80 NEXT I
```

```
90 GOSUB 600:L=0:ANSWER$="":CNT=CNT+1
100 PRINT "♥ Q Q Q Q Q Q Q Q Q";TAB(15);S$
110 POKE 209,159:POKE 210,5
120 LIM=TIME+((10-N)*24+80)*LEN(S$)
130 PRINT "R ▬▮▮";
140 GET A$:IF A$<>"" THEN 230
150 IF TIME<LIM THEN 140
160 IF ANSWER$=WURD$ THEN 320
170 GOSUB 400
180 PRINT:PRINT:PRINT "SORRY, TIME'S UP!"
190 FOR I=1 TO 200:NEXT I:POKE 54276,0
200 PRINT:PRINT "THE WORD IS:   ";WURD$
210 FOR I=1 TO 2000:NEXT I
220 POKE 198,0:GOTO 50
230 IF A$=CHR$(13) THEN 320
240 IF A$="▬" THEN 390
250 IF A$<>"▮▮" THEN 290
260 IF L=0 THEN 140
270 L=L-1:ANSWER$=LEFT$(ANSWER$,L)
280 PRINT " ▮▮▮▮";:GOTO 130
290 IF A$<"A" OR A$>"Z" THEN 140
300 PRINT A$;:L=L+1:ANSWER$=ANSWER$+A$
310 GOTO 130
320 PRINT " ":IF ANSWER$<>WURD$ THEN 360
330 GOSUB 500:PRINT "CORRECT!":YES=YES+1
340 FOR I=1 TO 200:NEXT I:POKE 54276,0
350 GOTO 50
360 GOSUB 400
370 PRINT:PRINT "SORRY, THAT'S NOT THE WORD"
380 GOTO 190
390 GOSUB 700:END
400 REM BADBEEP
410 POKE 54272,132
420 POKE 54273,3
430 POKE 54277,160
440 POKE 54278,250
450 POKE 54296,15
460 POKE 54276,33
```

```
470 RETURN
500 REM GOODBEEP
510 POKE 54272,32
520 POKE 54273,78
530 POKE 54277,34
540 POKE 54278,138
550 POKE 54296,15
560 POKE 54276,33
570 RETURN
600 REM SCRAMBLE WURD$ INTO S$
610 S$="":TEMP$=WURD$
620 FOR SIZE=LEN(WURD$) TO 1 STEP -1
630 LOC=INT(RND(0)*SIZE)+1
640 S$=S$+MID$(TEMP$,LOC,1)
650 TEMP$=LEFT$(TEMP$,LOC-1)
    +RIGHT$(TEMP$,SIZE-LOC)
660 NEXT SIZE
670 RETURN
700 REM PERFORMANCE RATING
710 POKE 53281,6
720 PRINT "♥ Q Q Q Q Q";TAB(14);"♦YOUR
    SCORES:"
730 PRINT:PRINT
740 PRINT TAB(9);"CORRECT      INCORRECT"
750 PRINT TAB(9);"--------------------"
760 PRINT:PRINT:PRINT TAB(10);YES;
770 PRINT TAB(23);(CNT-1)-YES
780 FINE=INT(YES/(CNT-1)*100)
790 PRINT:PRINT TAB(9);"(";FINE;"■■% )";
800 PRINT TAB(22);"(";100-FINE;"■■% )"
810 RETURN
900 REM WORDS
910 DATA"BENEFIT","INSIST","PLEASURE"
920 DATA"STORM","ACRONYM","TRAPEZE"
930 DATA"WORLDLY","FUNGUS","MARKET"
940 DATA"YESTERDAY","GUIDANCE","YEARN"
950 DATA"ENGAGE","RECLAIM","KNOWLEDGE"
960 DATA"COLONY","UTILITY","JOKE"
970 DATA"DRIVER","REMAINDER","HYDROGEN"
```

```
980 DATA"OCTOPUS","TECHNICAL","LAUGH"
990 DATA"QUIVER","VOLTAGE","NIMBLE"
1000 DATA"FOLIAGE","AMBITION","VALVE"
1010 DATA"GEOMETRY","LOCATION","GLACIER"
1020 DATA"OUTRAGEOUS","WARRANT","KICK"
1030 DATA"CIVILIZE","MATERNITY","HEART"
1040 DATA"DISARRAY","ZOMBIE","BLUEPRINT"
1050 DATA"MILDEW","IMPRESS","JINGLE"
1060 DATA"PERFECT","EXCESSIVE","QUOTA"
1070 DATA"NAVIGATE","SEPARATE"
```

GLOSSARY OF VARIABLES

CNT	Keeps a count of the number of words the player attempted. This is used to calculate the performance rating at the finish of the game.
YES	The number of correct guesses the player made.
N	The player's selected speed or level.
WURD$	Holds the randomly selected word.
L	While the player types in the unscrambled word, this variable keeps a count of how many letters he has input so far.
ANSWER$	Stores the player's unscrambled word.
S$	Holds the scrambled word as returned by the subroutine at line 600.
LIM	The time before which the player must enter the answer.
TEMP$	In the subroutine at line 600 this variable is initially assigned the word in WURD$. During the scramble, characters are removed one at a time from TEMP$.
SIZE	In the subroutine at line 600 this variable indicates how many characters are currently stored in TEMP$.
LOC	A randomly selected number pointing to one of the characters in TEMP$.

LINES OF SPECIAL INTEREST

130–310: This is the entire code for getting the player's response. It is a repeat of similar routines used in some of our previous programs.

150–180: Test to see whether the time limit has expired. Notice

that if it has indeed expired before the player presses the RETURN key but after he has entered all the correct letters, his answer will still be accepted.

190: After a short delay the tone caused by the subroutine at line 400 is terminated.

210: The correct answer is left on the screen for a short time.

220: The keyboard buffer is cleared so that characters entered after the expiration of the time limit are not read by the GET statement at the commencement of the next round.

240: The F1 key may be pressed anytime before pressing the RETURN key, even after some of the letters have already been entered.

330–350: The player has answered correctly.

360–380: The player has answered incorrectly.

390: The player has pressed the F1 key to terminate.

Subroutine

400–470: Produces a tone indicating that either the time has expired or the player has guessed incorrectly.

Subroutine

500–570: Produces a tone indicating that the player has answered correctly.

Subroutine

600–670: This subroutine places in S$ a scrambled version of the word in WURD$.

620–660: Each time through this loop, one character is placed in S$.

630: Chooses one of the characters in TEMP$ by selecting a number between 1 and SIZE.

640: The selected character is concatenated to S$.

650: This line effectively removes the selected character from TEMP$. TEMP$ is now shorter by one character.

Subroutine

700–810: The performance rating is displayed. The reason that CNT-1 is used rather than CNT is that the round during which F1 was pressed has also been included in CNT.

900–1070: The pool of fifty words from which the random words are selected.

SUGGESTED ENHANCEMENTS

1. As soon as the pool of words has been learned by rote, they may be changed at will.
2. Modify the program such that in any given run no word is selected more than once. Once all fifty words have been exhausted the program should terminate.
3. The number of words in the pool can be increased or decreased at any time. Line 50 would have to be modified.
4. You may substitute key foreign words to encourage a student of the language to have some fun while learning.

CHAPTER 24

LOTTERY

Across New York State, few activities seem to generate as much excitement as the state lottery. It is not uncommon to find long lines of people outside kiosks and stores that sell lottery tickets. This is hardly surprising, since it is possible to win millions of dollars, if you are lucky.

To enter the lottery in New York State, you pay a minimum bet of one dollar for two chances. For each chance you select six numbers between 1 and 44, with no repeats. When the lottery is drawn, six numbers are "pulled out of a hat," so to speak, from a pool of forty-four. If the six numbers selected by a purchaser match these six, that person becomes a winner. The New York State lottery also allows for a so-called bonus number.

The success of the New York State lottery encouraged some neighboring states to conduct their own state lotteries. New Jersey and Pennsylvania have chosen to follow the lottery route, though their respective rules differ somewhat from those of New York. In New Jersey, for example, the numbers drawn range from 1 to 36 rather than 1 to 44.

The highest prize ever offered in a lottery in the United States was $21.1 million in a New York drawing in May 1984. The record for an individual win in the Unites States was the $8.8 million awarded by the Pennsylvania lottery in July of 1983.

Early in 1984 intense public excitement was created in Canada, where a lottery is conducted by the country's Interprovincial Lottery Corporation. This lottery is run nationwide and has become a national weekly mania. In Canada, six numbers are drawn from a pool of forty-nine. The odds of picking the six lucky numbers in the contest, called the 6-49, are approximately fourteen million to one. In January of 1984 the prize amounted to 14 million Canadian dollars, equivalent to about 11 million U.S. dollars.

In the United States all lottery winnings are taxed and are paid to the recipients over a period of twenty years, in equal annual amounts. In Canada, however, the money is not taxable and the winnings are paid in one lump sum. It is no wonder, then, that more and more Americans journey northward to try their luck, despite the fact that a United States law, passed in 1930, prohibits the importation of lottery tickets. Nevertheless, it is not illegal for an American to win a Canadian lottery or to bring home the prize. However, the American must pay about 50 percent of the winnings as income tax.

Under the 6-49 rules (and indeed most other lottery rules as well), for any week that the six numbers drawn have not been chosen by any entrant, the prize for that week is thrown into the next week's offering. This is the reason why the prizes are sometimes so astronomical. (Incidentally, the biggest lottery in the world is Spain's El Gordo, which in its draw in December 1983 offered a prize of $73 million!)

In the program which we present to simulate a lottery, we have opted for the Canadian version, which allows for six numbers between 1 and 49. You are given the opportunity to "mark" a scorecard, which is displayed on the screen and is numbered from 1 to 49. Once your six numbers have been chosen, the computer selects its six random numbers, plus an extra, and you can see how well you fared. This program could of course be used to select your entries for the Canadian lottery. Maybe you could be one of the lucky ones. At any rate, we wish you the best of luck.

When the program is run, the screen is cleared, and a 7×7 board is displayed with the numbers 1 through 49 in sequential order, starting from the top lefthand corner. The numbers are selected through the positioning of a cursor. When the cursor is

over a number, the number is seen in reversed graphics, colored blue. The cursor is moved around with the CRSR keys. To select a number, you must position the cursor over it and press the F1 key. The cursor then turns purple. When the cursor is removed, the selected number appears in reverse graphics, colored red. (Whenever the cursor is positioned over a selected number, the number appears in purple instead of red.) If you change your mind, you can position the cursor over a selected number and press the F3 key. This deselects the number, returning its appearance to normal.

Once you have chosen your six numbers, you must press the F5 key to submit your entry to the computer. (If this key is pressed before the six numbers have been selected, a message is printed advising you of how many more numbers must be entered.) The computer then displays its six randomly selected numbers at the bottom of the screen, along with an "extra." If your six numbers match those selected by the computer, a message is displayed (along with musical accompaniment) announcing that you have won "the big one." If five of your numbers are among the six chosen by the computer and your sixth number matches the "extra," you win second prize. No points are awarded in this game, but you may of course keep your own score. Be warned, however, that just as in a real lottery, the chances of winning a prize are very small. The computer continues to allow you to play until the F7 key is pressed. The program then terminates.

```
5 REM LOTTERY
10 DIM P(48),Q(48),N(6)
20 POKE 53281,15:GOSUB 300
30 GOSUB 500:CNT=0
40 FOR I=0 TO 48:P(I)=0:NEXT I
50 S=0:GOSUB 700:IF S=0 THEN 50
60 IF S<0 THEN 190
70 IF CNT=6 THEN 100
80 PRINT "■■SELECT";6-CNT;"MORE NUMBERS."
90 PRINT "□□□";:GOTO 50
100 GOSUB 1200:PRINT:GOSUB 1300
110 IF MATCH<6 THEN 170
120 IF XTRA>0 THEN 150
```

```
130 PRINT "YOU WON THE BIG ONE!!!"
140 I=1:GOSUB 1400:GOTO 30
150 PRINT "YOU WON SECOND PRIZE!!"
160 I=2:GOSUB 1400:GOTO 30
170 PRINT "SORRY, YOU DIDN'T WIN."
180 GOSUB 1600:GOTO 30
190 POKE 53281,6:PRINT "■■GAME OVER":END
200 REM CLEAR A LINE
210 PRINT "■";
220 FOR I=1 TO 38:PRINT " ";:NEXT I
230 PRINT:PRINT "■";
240 RETURN
300 REM PRINT GRID
310 PRINT "■■":PRINT
320 PRINT TAB(9);"■■■";
330 FOR I=1 TO 6:PRINT "■■■";:NEXT I
340 PRINT "■":PRINT TAB(9);
350 FOR I=1 TO 7:PRINT "■  ";:NEXT I
360 PRINT "■"
370 FOR J=1 TO 6
380 PRINT TAB(9);"■■■";
390 FOR I=1 TO 6:PRINT "■■■";:NEXT I
400 PRINT "■":PRINT TAB(9);
410 FOR I=1 TO 7:PRINT "■  ";:NEXT I
420 PRINT "■"
430 NEXT J
440 PRINT TAB(9);"■■■";
450 FOR I=1 TO 6:PRINT "■■■";:NEXT I
460 PRINT "■"
470 RETURN
500 REM PUT UP NUMBERS
510 PRINT "■":PRINT:PRINT
520 FOR I=0 TO 42 STEP 7
530 FOR J=1 TO 7
540 PRINT TAB(7+J*3);
550 IF I+J<10 THEN PRINT " ";
560 PRINT MID$(STR$(I+J),2);
570 NEXT J
580 PRINT:PRINT
```

```
590 NEXT I
600 PRINT
610 PRINT TAB(9);"F1=SELECT  F3=DESELECT"
620 PRINT TAB(9);"F5=SUBMIT  F7=END"
630 PRINT:PRINT:GOSUB 200:PRINT
640 PRINT:PRINT:GOSUB 200:PRINT "□□";
650 ROW=0:COL=0:TYPE=2:GOSUB 1000
660 RETURN
700 REM MOVE CURSOR
710 GET A$:IF A$="" THEN RETURN
720 IF A$="□" THEN 810
730 IF A$="▒" THEN 830
740 IF A$="▌" THEN 850
750 IF A$="▐" THEN 870
760 IF A$="▄" THEN 890
770 IF A$="▀" THEN 920
780 IF A$="▐" THEN 950
790 IF A$<>"▐" THEN RETURN
800 S=-1:RETURN
810 IF ROW=0 THEN RETURN
820 DR=-1:DC=0:GOTO 960
830 IF ROW=6 THEN RETURN
840 DR=1:DC=0:GOTO 960
850 IF COL=0 THEN RETURN
860 DR=0:DC=-1:GOTO 960
870 IF COL=6 THEN RETURN
880 DR=0:DC=1:GOTO 960
890 IF CNT=6 OR P(ROW*7+COL)=1 THEN RETURN
900 CNT=CNT+1:P(ROW*7+COL)=1
910 TYPE=3:GOSUB 1000:RETURN
920 IF CNT=0 OR P(ROW*7+COL)=0 THEN RETURN
930 CNT=CNT-1:P(ROW*7+COL)=0
940 TYPE=2:GOSUB 1000:RETURN
950 S=1:RETURN
960 TYPE=P(ROW*7+COL):GOSUB 1000
970 ROW=ROW+DR:COL=COL+DC
980 TYPE=P(ROW*7+COL)+2:GOSUB 1000
990 RETURN
1000 REM COLOR A SPACE
```

```
1010 RR=ROW*2+3:CC=COL*3+10
1020 SPOT=1024+RR*40+CC
1030 IF PEEK(SPOT)>128 THEN 1080
1040 IF TYPE=0 THEN 1110
1050 POKE SPOT,PEEK(SPOT)+128
1060 POKE SPOT+1,PEEK(SPOT+1)+128
1070 GOTO 1110
1080 IF TYPE>0 THEN 1110
1090 POKE SPOT,PEEK(SPOT)-128
1100 POKE SPOT+1,PEEK(SPOT+1)-128
1110 ON TYPE+1 GOTO 1120,1140,1160,1180
1120 POKE SPOT+54272,6
1130 POKE SPOT+54273,6:RETURN
1140 POKE SPOT+54272,2
1150 POKE SPOT+54273,2:RETURN
1160 POKE SPOT+54272,6
1170 POKE SPOT+54273,6:RETURN
1180 POKE SPOT+54272,4
1190 POKE SPOT+54273,4:RETURN
1200 REM CHOOSE NUMBERS
1210 FOR I=0 TO 48:Q(I)=0:NEXT I
1220 FOR I=0 TO 5
1230 N(I)=INT(RND(0)*49)
1240 IF Q(N(I))>0 THEN 1230
1250 PRINT "■";MID$(STR$(N(I)+1),2);
     "■ ";:Q(N(I))=1
1260 NEXT I
1270 N(6)=INT(RND(0)*49)
1280 IF Q(N(6))>0 THEN 1270
1290 PRINT "■EXTRA: ■";
     MID$(STR$(N(6)+1),2);"■":RETURN
1300 REM CHECK FOR MATCH
1310 MATCH=0:XTRA=0
1320 FOR I=0 TO 6
1330 IF P(N(I))=1 THEN MATCH=MATCH+1
1340 NEXT I
1350 IF P(N(6))=1 THEN XTRA=1
1360 RETURN
1400 REM WIN SONG
```

```
1410 POKE 54273,16:POKE 54277,68
1420 POKE 54278,246:POKE 54296,15
1430 POKE 54276,33
1440 FOR J=1 TO 300:NEXT J
1450 POKE 54273,14
1460 FOR J=1 TO 100:NEXT J
1470 POKE 54273,20
1480 FOR J=1 TO 700:NEXT J
1490 IF I=2 THEN 1540
1500 POKE 54273,14
1510 FOR J=1 TO 400:NEXT J
1520 POKE 54273,24
1530 FOR J=1 TO 800:NEXT J
1540 POKE 54276,32
1550 FOR J=1 TO 3000:NEXT J
1560 POKE 54296,0
1570 RETURN
1600 REM LOSE TUNE
1610 POKE 54273,8:POKE 54277,68
1620 POKE 54278,246:POKE 54296,15
1630 POKE 54276,33
1640 FOR J=1 TO 300:NEXT J
1650 POKE 54273,4
1660 FOR J=1 TO 1000:NEXT J
1670 POKE 54276,32
1680 FOR J=1 TO 3000:NEXT J
1690 POKE 54296,0
1700 RETURN
```

GLOSSARY OF VARIABLES

P This is an array of forty-nine elements numbered 0 through 48. When the *player* selects a number, the corresponding element is set to 1. It may be helpful to keep in mind that even though the numbers displayed on the screen range from 1 to 49, they are represented internally as 0 through 48. This has no direct bearing on the game as seen by the player.

Q This is an array of forty-nine elements numbered 0 through

48. When the *machine* randomly selects a number, it sets the corresponding element of Q to 1.

N This array of seven elements, numbered 0 through 6, holds the seven numbers chosen by the computer. N(6) contains the extra (bonus) number.

CNT This variable keeps track of how many numbers the player has selected so far.

S The subroutine at line 700 sets this variable to 1 when the F5 key is pressed or to −1 when the F7 key is pressed. Otherwise, it retains its initial value of 0.

MATCH Specifies how many of the player's numbers match the computer's numbers. This count includes the extra number.

XTRA If one of the player's numbers matches the extra number, this variable is set to 1.

I This passes the value 1 or 2 to the subroutine at line 400, indicating which prize has been won. Elsewhere it merely plays the role of an ordinary index to a FOR/NEXT loop.

ROW Indicates the row of the playing board (0 through 6) occupied by the cursor.

COL Indicates the column of the playing board (also 0 through 6) occupied by the cursor.

TYPE This variable is used to indicate to the subroutine at line 1000 how to display a particular number. A value of 0 indicates that the cursor is moving away from an unselected number. The number is therefore converted from reversed graphics to normal graphics and is colored blue. A value of 1 indicates that the cursor is moving off a selected number. The number remains in reversed graphics but its color is changed to red. A value of 2 indicates that the cursor is moving onto an unselected number or a number is being deselected. The number is converted to reversed graphics, if necessary, and is colored blue. A value of 3 specifies that the cursor is moving onto a selected number or a number is being selected; it is colored purple.

LINES OF SPECIAL INTEREST

50: Waits for the player to press F5 or F7; the subroutine at line 700 assumes control until then.

70–90: If the player pressed F5 before entering 6 numbers, inform him of this and go back to the subroutine at line 700.

Subroutine
200–240: This subroutine moves the PRINT cursor (the invisible marker of the current PRINT position) up one line, blanks out the line, and remains there. Such a subroutine is useful because the program prints several messages on the same line at different times.

Subroutine
300–470: Prints the grid into which the numbers are placed.

Subroutine
500–660: Fills the grid with sequential numbers from 1 to 49.

630–640: Clears rows 21 and 23 on the screen.
650: Positions the cursor over the first number.

Subroutine
700–990: Moves the cursor as controlled by the player and performs selection and deselection, as requested. This subroutine returns after performing each operation, or if no key is pressed.

760: The character in quotation marks indicates the F1 key.
770: The character in quotation marks indicates the F3 key.
780: The character in quotation marks indicates the F5 key.
790: The character in quotation marks indicates the F7 key.
890: If the F1 key is pressed and either 6 numbers have been selected or the number under the cursor has already been selected, then the F1 key is ignored. Note that array P is subscripted by use of the value of ROW and COL. This can be done because the number in any given location of the grid is always the same.
920: If the F3 key is pressed and either no numbers have been selected or the number under the cursor has not been selected, the F3 key is ignored.
960: Moves the cursor off the current number. TYPE gets the value 0 or 1, depending upon whether or not the number has been selected.
980: Moves the cursor into the new location. TYPE gets the value 2 or 3, depending upon whether or not the number has been selected.

Subroutine
1000–1190: Changes the appearance of a number depending upon the value of TYPE.

1010–1020: Convert the values of ROW and COL into an actual location in screen memory.

1030–1070: If TYPE is not 0 and the current number is in normal graphics, convert it to reverse graphics.
1080–1100: If TYPE = 0 and the number is in reverse graphics, convert to normal graphics.
1110–1190: Color the number depending upon the value of TYPE.

Subroutine
1200–1290: The computer selects and displays its seven numbers.

1250: The functions MID$ and STR$ are used to eliminate the leading blank that is usually attached to the number by the PRINT statement. This blank would be especially unattractive since it would be printed in reversed graphics.
1270: The extra number is contained in N(6).

Subroutine
1300–1360: Determines how many of the player's numbers match the computer's numbers and also whether the player has chosen the extra number as one of his six.

Subroutine
1400–1570: Plays triumphant music if the player has won a prize. Note that lines 1500–1530 are skipped if the player has won only the second prize.

Subroutine
1600–1700: Plays a doleful tune if the player failed to win a prize.

SUGGESTED ENHANCEMENTS

1. The computer's selections could be displayed in ascending order to make for ease of reading.
2. The player could be allowed to decide for himself both the range of numbers in the pool and the number of selections to be made from the pool.

CHAPTER 25

CONCENTRATION

This is another one of those extremely popular games that can be played with a deck of ordinary playing cards. Most of the time, the game is played between two people. As you may recall, the deck of fifty-two cards is shuffled and the cards are laid out on a table facedown. One of the two players turns any two of the cards faceup. If these two cards do not have the same face value (the suit is ignored in this game), the cards are turned facedown again and the other player takes his turn. If, however, the two cards do have the same face value, the player removes them from the table and stacks the cards in his pile. He may then immediately go again, and this continues until he turns up two cards that do not match. At this point play continues with the other player. The strategy involved in this game is to concentrate on the location of the cards when a nonmatching pair is turned faceup. Then when a card shows up on the first turn, the location of a matching card might be remembered.

This process of alternating play continues until all the cards have been removed from the table. At this point the players count up how many matching pairs each has. The player with the greater number of matches is the winner. Since a deck of fifty-two cards yields twenty-six matches (assuming that the deck is not phony), it is possible for the game to terminate in a tie.

In this Commodore 64 version of Concentration, the facedown cards are represented by black rectangles which are laid out in four rows of thirteen cards each. The top of the screen displays a message in yellow indicating whose turn it is. The number of successful matches scored by each player is displayed just beneath this. We use the letters A for ace, J for jack, Q for queen, and K for king. Also, the letter T is used to represent the 10 card.

A cursor is used to select those cards that the player wishes to turn faceup. The location of the cursor is indicated by a white rectangle appearing in place of a black one. The cursor is moved using the two CRSR keys. To select a particular card, the player positions the cursor over the appropriate card and presses the F1 key. When the cursor is located directly over a faceup card, the card appears in reversed graphics. As soon as two cards are turned over, the computer determines if they are indeed a match. If they are, a high tone is emitted, the cards are removed from the screen, the player's score is automatically incremented in the display, and the same player goes again. When the cursor is moved around the playing area, it skips over those locations from which cards were removed. If the cards do not match, a low tone is produced, the cards are turned facedown again, and play is passed over to the other player. Every time a new play is begun the cursor is repositioned over the card closest to the top left of the playing area. This is done so that the cursor cannot be used as a pointer to a card's location.

Once all the cards have been removed from the screen, the computer automatically decides who the winner is and displays this information. It also highlights the number of the winning player in the score area for extra effect. If it is a tie, a message to this effect is displayed.

```
10 REM CONCENTRATION
20 DIM P(51),BOARD(3,12,1),WHICH(1,1),PTS(2)
30 POKE 53281,12:GOSUB 300
40 OVER=0:CNT=52
50 PTS(1)=0:PTS(2)=0:PL=1
60 PRINT "■■■";TAB(14);"■PLAYER # ■";PL
70 ROW=3:COL=12:DIR=4:GOSUB 700
```

```
80 POKE 198,0
90 GOSUB 500
100 IF OVER<2 THEN 80
110 IF MATCH>0 THEN 130
120 PL=3-PL:GOSUB 1200:GOTO 60
130 PTS(PL)=PTS(PL)+1:GOSUB 1400
140 IF CNT>0 THEN 70
150 PRINT "[S][Q][Q]";TAB(14);"                "
160 IF PTS(1)=PTS(2) THEN 230
170 W=1:IF PTS(2)>PTS(1) THEN W=2
180 PRINT "[Q][Q][Q][R]";TAB(W*20-6);W
190 GOSUB 1500
200 PRINT:PRINT:PRINT
210 PRINT TAB(7);"PLAYER # [R]";W;"[■]IS THE
    WINNER!"
220 GOTO 240
230 PRINT "[Q][Q][Q][Q][Q][Q][Q]";TAB(14);"[◥]IT WAS
    A TIE"
240 POKE 53281,6:PRINT "[◆]":END
300 REM INITIAL SETUP, DEAL CARDS
310 PRINT "[♥][Q][Q][Q][Q][Q][■]"
320 PRINT TAB(5);"PLAYER #  1";
330 PRINT TAB(25);"PLAYER #  2"
340 PRINT TAB(5);"----------";
350 PRINT TAB(25);"----------"
360 GOSUB 1600:PRINT "[Q][Q][Q][■]"
370 FOR R=0 TO 3:FOR C=0 TO 12
380 P(R*13+C)=R*100+C
390 NEXT C:NEXT R
400 T=52
410 FOR SUIT=1 TO 4:FOR CARD=1 TO 13
420 N=INT(RND(0)*T):T=T-1
430 R=INT(P(N)/100+.1):C=P(N)-R*100
440 BOARD(R,C,0)=SUIT:BOARD(R,C,1)=CARD
450 PRINT " [R]  [■]";
460 P(N)=P(T)
470 NEXT CARD:PRINT:PRINT:NEXT SUIT
480 RETURN
500 REM MOVE CURSOR
```

```
510 GET A$:IF A$="" THEN 510
520 IF A$="■" THEN 580
530 IF A$="■" THEN 590
540 IF A$="■" THEN 600
550 IF A$="■" THEN 610
560 IF A$<>"■" THEN 510
570 GOSUB 900:RETURN
580 DIR=1:GOTO 620
590 DIR=2:GOTO 620
600 DIR=3:GOTO 620
610 DIR=4
620 SPOT=1024+(2*ROW+13)*40+(3*COL+1)
630 IF PEEK(SPOT)=160 THEN 670
640 POKE SPOT,PEEK(SPOT)-128
650 POKE SPOT+1,PEEK(SPOT+1)-128
660 GOTO 680
670 POKE SPOT+54272,0:POKE SPOT+54273,0
680 GOSUB 700
690 RETURN
700 REM MOVE CURSOR TO NEXT CARD
710 ON DIR GOTO 720,740,760,780
720 ROW=ROW-1:IF ROW<0 THEN ROW=3
730 GOTO 800
740 ROW=ROW+1:IF ROW>3 THEN ROW=0
750 GOTO 800
760 COL=COL-1:IF COL>=0 THEN 800
770 COL=12:GOTO 720
780 COL=COL+1:IF COL<=12 THEN 800
790 COL=0:GOTO 740
800 SPOT=1024+(2*ROW+13)*40+(3*COL+1)
810 IF PEEK(SPOT)=32 THEN 710
820 IF PEEK(SPOT)=160 THEN 860
830 POKE SPOT,PEEK(SPOT)+128
840 POKE SPOT+1,PEEK(SPOT+1)+128
850 RETURN
860 POKE SPOT+54272,1
870 POKE SPOT+54273,1
880 RETURN
900 REM TURN UP A CARD
```

```
910 IF PEEK(SPOT)<>160 THEN RETURN
920 WHICH(OVER,0)=ROW:WHICH(OVER,1)=COL
930 OVER=OVER+1
940 ON BOARD(ROW,COL,0) GOTO 950,970,990,1010
950 POKE SPOT+54272,2:POKE SPOT+54273,2
960 POKE SPOT+1,211:GOTO 1030
970 POKE SPOT+54272,2:POKE SPOT+54273,2
980 POKE SPOT+1,218:GOTO 1030
990 POKE SPOT+54272,0:POKE SPOT+54273,0
1000 POKE SPOT+1,193:GOTO 1030
1010 POKE SPOT+54272,0:POKE SPOT+54273,0
1020 POKE SPOT+1,216
1030 CARD=BOARD(ROW,COL,1)
1040 IF CARD>1 THEN 1060
1050 POKE SPOT,129:GOTO 1130
1060 IF CARD>9 THEN 1080
1070 POKE SPOT,CARD+176:GOTO 1130
1080 ON CARD-9 GOTO 1090,1100,1110,1120
1090 POKE SPOT,148:GOTO 1130
1100 POKE SPOT,138:GOTO 1130
1110 POKE SPOT,145:GOTO 1130
1120 POKE SPOT,139
1130 ON OVER GOTO 1140,1150
1140 MATCH=CARD:RETURN
1150 IF CARD<>MATCH THEN MATCH=0
1160 RETURN
1200 REM TURN DOWN CARDS
1210 POKE 54273,6:POKE 54275,8
1220 POKE 54277,34:POKE 54278,244
1230 POKE 54296,15:POKE 54276,65
1240 FOR I=1 TO 500:NEXT I
1250 POKE 54276,64
1260 FOR I=1 TO 1000:NEXT I
1270 POKE 54296,0
1280 FOR OVER=0 TO 1
1290 R=WHICH(OVER,0):C=WHICH(OVER,1)
1300 SPOT=1024+(2*R+13)*40+(3*C+1)
1310 POKE SPOT,160:POKE SPOT+1,160
1320 POKE SPOT+54272,0
```

```
1330 POKE SPOT+54273,0
1340 NEXT OVER
1350 OVER=0
1360 RETURN
1400 REM REMOVE CARDS
1410 GOSUB 1600:GOSUB 1500
1420 FOR I=1 TO 1200:NEXT I
1430 FOR OVER=0 TO 1
1440 R=WHICH(OVER,0):C=WHICH(OVER,1)
1450 SPOT=1024+(2*R+13)*40+(3*C+1)
1460 POKE SPOT,32:POKE SPOT+1,32
1470 CNT=CNT-1
1480 NEXT OVER
1490 OVER=0:RETURN
1500 REM SUCCESS TONE
1510 POKE 54273,32:POKE 54275,8
1520 POKE 54277,15:POKE 54278,240
1530 POKE 54296,15:POKE 54276,65
1540 FOR I=1 TO 150:NEXT I
1550 POKE 54276,64
1560 FOR I=1 TO 100:NEXT I
1570 POKE 54296,0
1580 RETURN
1600 REM PRINT POINTS
1610 PRINT "S Q Q Q Q Q Q Q "
1620 PRINT TAB(5);PTS(1);"  MATCHES";
1630 PRINT TAB(25);PTS(2);"  MATCHES"
1640 RETURN
```

GLOSSARY OF VARIABLES

P This array contains fifty-two elements numbered 0 through 51. It is used in the initial setup to hold values representing locations where cards may be placed.

BOARD This is a three-dimensional matrix that represents the layout of the cards. The first dimension indicates the rows, numbered 0 through 3. The second dimension indicates the columns, numbered 0 through 12. The third dimension

is used to select special information about the card, located at the specified row and column. Setting the third dimension to 0 selects the card's suit, numbered 1 through 4. Setting the dimension to 1 selects the card's face value, numbered 1 through 13.

WHICH This matrix has two dimensions, both numbered 0 through 1. It holds information about the two faceup cards. The first subscript specifies one of the two cards. The second subscript specifies the card's row or column location.

PTS This array of two elements, numbered 1 and 2, keeps track of the number of matches made by each player.

OVER Keeps count of how many cards are currently faceup on on the screen.

CNT Keeps count of how many cards are currently on the screen, whether faceup or facedown. When cards are removed, this variable is decremented.

PL A value of 1 or 2 specifies which player is currently taking a turn.

ROW Specifies at which row the cursor is located.

COL Specifies at which column the cursor is located.

DIR This variable tells the subroutine at line 700 in which direction to move the cursor. A value of 1 means up, 2 means down, 3 means to the left, and 4 means to the right.

MATCH After two cards have been turned faceup, this variable is nonzero if the cards match.

LINES OF SPECIAL INTEREST

70: The intention of this line is to place the cursor on the screen. This is also done by the subroutine at line 700. However, the subroutine is designed so that it must move the cursor in some specified direction before displaying the cursor. Therefore, this line sets the values of ROW and COL to indicate that the cursor is over the lower righthand card and sets DIR to indicate movement to the right. The subroutine at line 700 wraps the cursor around to the upper lefthand corner and displays it there.

80–100: Wait until 2 cards have been turned over.

120: No match—switch players.

150–240: Indicate the winner and end the game.

150: Eleven spaces erase the area indicating whose turn it is.

170–180: Decide which player has the higher score and highlight that player's number.

230: The first character in the second literal changes the character color to cyan.

Subroutine

300–480: This subroutine sets up the screen and displays the facedown cards. It also places the values of the cards in random locations in matrix BOARD.

370–390: Places values in array P representing the fifty-two locations where cards may be placed. The values placed in P are three-digit numbers where the leftmost digit represents a row (0 through 3) and the remaining digits represent a column (0 through 12).

400: The variable T represents how many unselected values are still in P.

410–470: Each time through this loop a value is selected from P and the card indicated by SUIT and CARD is placed in the location specified by the value selected from P.

420: The variable N selects a location in array P. The value of T is decremented to indicate that a selection has been made.

430: The selected value is converted into values for R and C, which specify a location in matrix BOARD.

440: Place the card in BOARD.

450: Display a facedown card.

460: Replaces the value just selected from P with the topmost value in P. This ensures that the unselected values are all located in the first T elements of P.

Subroutine

500–690: Waits for the player to press a key. If it is a CRSR key, the cursor is moved. If it is the F1 key, this situation is also handled. This subroutine returns after a valid key is pressed and the information processed.

570: The F1 key was pressed.

580–610: Determine in which direction the cursor is to be moved.

640–660: If the cursor was over a faceup card, change the card from reversed graphics to normal graphics.

670: The cursor was over a facedown card, so this line merely changes the color of the rectangle to black.

Subroutine

700–880: Moves the cursor and displays it.

720–750: If the cursor is on the top row and is moved up, it wraps

around to the bottom row, in the same column. The reverse applies to moving down from the bottom row.

760–770: If the cursor is in the leftmost column and is moved to the left, not only does it wrap around to the rightmost column but it is also moved up 1 row. Control is passed to line 720 in the event that moving up 1 row also requires wraparound.

780–790: If the cursor is in the rightmost column and is moved to the right, not only does it wrap around to the leftmost column but it is also moved down 1 row. Control is then passed to line 740 in the event that moving down 1 row also requires wraparound.

810: This skips over empty locations.

830–840: The new cursor location is over a faceup card. The card is converted to reversed graphics.

860–870: The new cursor location is over a facedown card. The rectangle representing the card is colored white.

Subroutine

900–1160: This subroutine turns a card faceup by displaying its face value and suit symbol. It also checks for matches.

910: Nothing is done if the card is already faceup.

920–930: Place the card's location in WHICH and increment OVER.

940–1020: Determine the card's suit, color the card's location red or black as appropriate, and display the correct suit symbol in reversed graphics. The number 1 represents hearts, 2 diamonds, 3 spades, and 4 clubs.

1040–1050: The card's face value is 1, so display an A in reversed graphics.

1080–1120: The card is a 10, Jack, Queen, or King, so display the appropriate letter in reversed graphics.

1140: This is the first card to be turned faceup, so its face value is saved in MATCH.

1150: This is the second card to be turned faceup. If its face value matches the number in MATCH, this value is left in MATCH. Otherwise, MATCH is set to 0.

Subroutine

1200–1360: This subroutine is called when the two cards do not match. It plays a low tone and turns the cards facedown again.

1210–1270: Play the low tone. The loop in line 1260 serves the additional purpose of leaving the cards faceup long enough for the players to memorize them.

1290–1300: Convert the row and column location as stored in matrix WHICH to an actual screen location.
1310: Replaces the card image with a rectangle.
1320–1330: The rectangle is colored black. Since the cursor was over one of these cards, the cursor momentarily disappears but is restored by line 70.
1350: OVER is reset to 0 to indicate that all cards are facedown.

Subroutine
1400–1490: Removes the matching cards from the screen and calls the routine to play a tone and update the points.

1410: The player scoring the match has already had one point added to his score before this subroutine is called. Therefore, the subroutine at line 1600 displays the updated scores.
1420: This loop lets the players see the cards before they are removed.
1470: Since the loop iterates twice, this statement is performed twice.

Subroutine
1500–1580: Produces a tone to indicate a match. This tone is also used to announce the winner of the game.

Subroutine
1600–1640: Displays the number of points or matches currently held by each player.

1610: The last character in the literal sets the character color to cyan.
1620–1630: The backspacing is done to guarantee that a blank space is placed between the number and the word "MATCHES."

SUGGESTED ENHANCEMENTS

1. The game could be modified to allow for more than two players.
2. Greater excitement could be added to the game by including two decks rather than one.
3. The difficulty level could be increased by requiring that the color values match as well as the face values. This means that any given card has only one match in the deck.

CHAPTER 26

ETCHER

Many of us are closet artists. If it weren't for the fact that we are so sloppy and would get paint all over our clothes we probably would be tempted to try our hand at some artwork. With the next program you can indulge yourself to your heart's content without the slightest fear of any paint spoiling your clothes. This is because this program takes advantage of the Commodore 64's ability to perform what is called bit-mapped graphics. In bit-mapped graphics you obtain much finer resolution on the screen than is possible in the regular graphics mode.

When the program is run you are asked to enter a number between 0 and 15. This number represents the background color of the screen on which the lines are drawn. You are then asked for another number between 0 and 15, representing the color of the lines themselves. This color must be different from the background color. If it is not, you are asked to start from the beginning by entering a number for the background color. Once the color codes have been validated, a message is printed asking you to wait a few moments while the screen is being set up. The screen then clears and you see it being filled with the background color. There is then a delay of about forty seconds. This delay is unavoidable, because 8,000 bytes of memory have to be set to 0. When a tone is emitted you may commence drawing.

You draw on the screen by controlling a dot-sized cursor. This cursor may be difficult to see on small-screen televisions. Its movements are directed by the two CRSR keys. As you draw a line, the cursor is on the leading edge of the line. It doesn't look like anything special—just the last dot on the line. You may notice that your vertical lines are somewhat thick and off color. This effect is unavoidable; it is due to the operation of the television tube. Doubling the vertical lines might normalize their color and the difference in thickness can be compensated for by doubling the horizontal lines.

You may at some point wish to erase a line that is already drawn. To do this you must press the F3 key. The cursor disappears—or rather the last drawn line becomes a tiny bit shorter at the end. The cursor is now invisible and any point at which it is positioned is erased. Therefore, to erase a line you must backtrack over it. In this process you may lose track of the cursor—after all, it is invisible. Should this happen, you can press the F1 key. This makes the cursor visible again. Once it is located, you can press F3 to make it vanish and move it in the required direction. This process may have to be repeated several times before the cursor is back to the desired position. Once erasing is finished, the F1 key is again pressed and the cursor draws lines instead of erasing them. The invisible cursor may also be used to draw discontinuous lines.

To erase everything on the screen, press the F5 key. This process, like the setting-up stage, takes about forty seconds. A tone is emitted when the program is ready to accept your input again. Finally, the F7 key terminates the program.

```
10 REM ETCHER
20 S=0:PRINT "■"
30 INPUT "BACKGROUND COLOR (0-15)";B
40 IF B<>INT(B) OR B<0 OR B>15 THEN 30
50 INPUT "LINE COLOR (0-15)";F
60 IF F<>INT(F) OR F<0 OR F>15 THEN 50
70 IF B<>F THEN 110
80 PRINT:PRINT "DIFFERENT COLORS, PLEASE."
```

```
90 PRINT "START FROM BEGINNING":PRINT
100 GOTO 30
110 FB=F*16+B
120 PRINT:PRINT "SETTING UP - PLEASE WAIT"
130 PRINT "FOR TONE BEFORE COMMENCING"
140 FOR I=1 TO 2000:NEXT I
150 GOSUB 200
160 GOSUB 400
170 IF S=0 THEN 160
180 PRINT "■FINISHED"
190 POKE 198,0:END
200 REM SET UP FOR BIT GRAPHICS
210 POKE 53272,(PEEK(53272) AND 240) OR 8
220 POKE 53265,PEEK(53265) OR 32
230 FOR I=1024 TO 2023
240 POKE I,FB
250 NEXT I
260 GOSUB 300
270 RETURN
300 REM CLEAR SCREEN
310 FOR I=8192 TO 16191
320 POKE I,0
330 NEXT I
340 Y=199:X=0:TYPE=1:POKE 15879,128
350 POKE 54273,16
360 POKE 54277,34:POKE 54278,240
370 POKE 54296,15:POKE 54276,17
380 FOR I=1 TO 200:NEXT I
390 POKE 54276,0:POKE 54296,0:RETURN
400 REM MOVE CURSOR/READ F-KEYS
410 GET A$:IF A$="" THEN RETURN
420 IF A$="■" THEN 510
430 IF A$="■" THEN 530
440 IF A$="■" THEN 550
450 IF A$="■" THEN 570
460 IF A$="■" THEN 590
470 IF A$="■" THEN 600
480 IF A$="■" THEN 610
490 IF A$="■" THEN 620
```

```
500 RETURN
510 IF Y=0 THEN RETURN
520 Y=Y-1:GOTO 650
530 IF Y=199 THEN RETURN
540 Y=Y+1:GOTO 650
550 IF X=0 THEN RETURN
560 X=X-1:GOTO 650
570 IF X=319 THEN RETURN
580 X=X+1:GOTO 650
590 TYPE=1:GOTO 650
600 TYPE=0:GOTO 650
610 GOSUB 300:RETURN
620 POKE 53272,(PEEK(53272) AND 241) OR 4
630 POKE 53265,PEEK(53265) AND 223
640 S=-1:RETURN
650 ROW=Y AND 248:COL=X AND 248
660 LOC=8192+ROW*40+COL+(Y AND 7)
670 BIT=2 ↑ (7-(X AND 7))
680 IF TYPE THEN 710
690 POKE LOC,PEEK(LOC) AND (255-BIT)
700 RETURN
710 POKE LOC,PEEK(LOC) OR BIT
720 RETURN
```

GLOSSARY OF VARIABLES

B The user's selected background color.

F The user's chosen line color.

FB An 8-bit value in which the upper 4 bits are equal to F and the lower 4 bits are equal to B.

S This variable is set to −1 when F7 is pressed.

Y Represents the row occupied by the cursor. In bit-mapped graphics the screen is divided into 200 rows, so Y may have a value from 0 to 199.

X Represents the column occupied by the cursor. In bit-mapped graphics the screen is divided into 320 columns, so X may have a value from 0 to 319.

TYPE This variable has the value 0 when the cursor is invisible (erasing) and the value 1 when it is visible (drawing).

LINES OF SPECIAL INTEREST

Subroutine

200–270: Performs some necessary household chores to initiate bit-mapped graphics.

210: The lower 4 bits of location 53272 are used to specify either the location of the patterns used to display normal characters or the location of bit-mapped screen memory. Setting these bits to a value of 8 indicates that the bit map begins at location 8192.

220: Sets the appropriate bit of location 53265 to 1. This switches on bit-mapped mode.

230–250: In bit-mapped mode, the area normally used as screen memory is instead used as color memory. The value of FB is placed in each of these locations so that the background color and line color are everywhere the same.

Subroutine

300–390: Clears bit-mapped screen memory. This is accomplished by setting every location to 0.

340: Positions the cursor in the lower lefthand corner of the screen. The POKE causes the cursor to be immediately displayed on the screen.

350–390: Emit a tone indicating that screen clearing has been completed.

Subroutine

400–720: This routine moves the cursor and performs the appropriate actions when the function keys are pressed. This subroutine returns immediately if no key is pressed. If one is pressed, it returns after performing the appropriate operation.

470: The character in quotation marks indicates the F3 key.

500: No valid key was pressed.

510–580: Increment or decrement X or Y depending upon how the cursor is being moved.

590: The F1 key was pressed. Control is passed to line 650 in order to redraw the cursor in its new form.

600: The F3 key was pressed. Control is passed to line 650 in order to redraw the cursor in its new form.

610: The F5 key was pressed.

620–640: The F7 key was pressed. Before returning, the location of character memory is restored to normal and bit-mapped mode is turned off.

650–670: This is the standard method of calculating which bit in which memory location must be set on or off to draw or erase a dot. For further details see the *Programmer's Reference Guide*.

690: TYPE = 0, so set the appropriate bit to 0.

710: TYPE = 1, so set the appropriate bit to 1.

SUGGESTED ENHANCEMENTS

1. Instead of giving the player the option of choosing the colors, have the computer select them randomly, care being taken that both colors are not the same.
2. By storing different values in locations 1024 through 2023, different background and line colors can appear in different areas of the screen.

CHAPTER 27

MOSAIC

Before the days of the hand-held calculators, one of the most popular of the hand-held games was the small, palm-sized mosaic board consisting of movable tiles which had to be reorganized into a preset order by moving the tiles around, one tile at a time.

The next program is a simulation of this tile game. You are asked to enter your "goal"—an image of how the group of tiles is to look when the puzzle is completed. You must enter fifteen characters, which the program places in a 4×4 frame, starting from the upper lefthand corner. Any of the printable characters (except the quotation mark) may be used. To form a picture, any of the graphics characters may be used. However, no characters may be repeated. The program inserts a space in place of the sixteenth tile. If you so choose, you may enter the space yourself at any point, in which case you will have to enter all sixteen characters. As the characters are entered they are displayed in reversed graphics. The frame is also drawn by the computer automatically during the goal entry process.

Once you have entered all the required symbols, the computer asks you to wait, as the internal copy of the puzzle is undergoing a shuffle. After a few moments the program displays the shuffled puzzle at the top of the screen. It is this copy with which you have to work. Tiles are moved through the use of a cursor, which

is initially placed in the puzzle frame on some character adjacent to the space. When the cursor is positioned on a tile, the character is displayed in normal (nonreversed) graphics in white. The cursor can then be moved to any tile using the two cursor keys. However, it may not be moved onto or across the empty space in the puzzle. To move a tile into the space the cursor is positioned on a tile adjacent to the space and the F1 key is pressed. The computer moves the tile along with the cursor into the location formerly occupied by the space. The location from which the tile was moved now becomes the space. It is also possible to move several tiles at a time. However, all the tiles must be in the same row or the same column as the space. The cursor is positioned on the tile farthest from the space, among the tiles which are to be moved. When the F1 key is pressed, the tile on which the cursor is positioned as well as all the tiles between it and the space (in the same row or column) are pushed one position in the direction of the space.

The program constantly compares the puzzle you are working on with the goal, which is constantly displayed for you on the screen. When the puzzle exactly matches the goal, the computer beeps immediately and prints a congratulatory message. You are then asked if you wish to construct another puzzle.

If at some point during the course of the game you weary of the puzzle, pressing the F7 key terminates the round, and the program asks if you care for another. The computer does not solve the puzzle for you, however.

```
10 REM MOSAIC
20 BLK$=CHR$(32):QUO$=CHR$(34)
30 DIM TAKEN(126),GRID$(3,3),DIRG$(3,3)
40 FOR I=0 TO 63:TAKEN(I)=0:NEXT I
50 PRINT "■":POKE 214,11:PRINT
60 PRINT TAB(15);"ENTER GOAL:":GOSUB 300
70 POKE 214,11:PRINT
80 PRINT TAB(15);"PLEASE WAIT"
90 FOR I=0 TO 3:FOR J=0 TO 3
100 DIRG$(I,J)=GRID$(I,J)
110 NEXT J:NEXT I
```

```
120 GOSUB 500
130 PRINT TAB(15);"■   GOAL:   "
140 GOSUB 700
150 POKE 214,3:PRINT
160 R=ROW-1:IF R>=0 THEN 180
170 R=ROW:C=COL-1:IF C<0 THEN C=COL+1
180 FOR I=0 TO R:PRINT:NEXT I
190 PRINT TAB(18+C);
200 S=0:GOSUB 900:IF S<0 THEN 250
210 FOR I=0 TO 3:FOR J=0 TO 3
220 IF DIRG$(I,J)<>GRID$(I,J) THEN 200
230 NEXT J:NEXT I
240 GOSUB 1400
250 POKE 214,21:PRINT
260 PRINT TAB(11);"ANOTHER (Y OR N)? ";
270 GET A$:IF A$<>"Y" AND A$<>"N" THEN 270
280 IF A$="Y" THEN 40
290 PRINT "■GAME OVER":END
300 REM INPUT GOAL
310 CNT=0:PRINT
320 PRINT TAB(17);"■■■■■■■"
330 COL=1:PRINT TAB(17);"■R";
340 PRINT "●■";
350 GET A$:IF A$<BLK$ OR A$=QUO$ THEN 350
360 A=ASC(A$):IF A>127 AND A<161 THEN 350
370 A=A-32:IF A>128 THEN A=A-65
380 IF TAKEN(A)=1 THEN 350
390 TAKEN(A)=1:IF A$=BLK$ THEN 410
400 PRINT A$;:GOTO 420
410 PRINT "■ R";
420 GRID$(INT(CNT/4),CNT AND 3)=A$
430 CNT=CNT+1
440 A$=BLK$:IF CNT=15 AND TAKEN(0)=0 THEN 410
450 IF CNT=16 THEN 480
460 COL=COL+1:IF COL<5 THEN 340
470 PRINT "■■":GOTO 330
480 PRINT "■■":PRINT TAB(17);"■■■■■■■"
490 RETURN
```

```
500 REM SCRAMBLE
510 FOR R=0 TO 3:FOR C=0 TO 3
520 IF DIRG$(R,C)=BLK$ THEN 540
530 NEXT C:NEXT R
540 TMES=INT(RND(0)*51)+50
550 FOR Q=1 TO TMES
560 DC=0
570 N=3:IF R=0 OR R=3 THEN N=2
580 DR=INT(RND(0)*N)-SGN(R)
590 IF DR<>0 THEN 640
600 IF C=1 OR C=2 THEN 630
610 DC=SGN(1-C)
620 GOTO 640
630 DC=INT(RND(0)*2)*2-1
640 ROW=R+DR:COL=C+DC
650 DIRG$(R,C)=DIRG$(ROW,COL)
660 DIRG$(ROW,COL)=BLK$
670 R=ROW:C=COL
680 NEXT Q
690 RETURN
700 REM PRINT MOSAIC
710 PRINT "■■■";TAB(17);"PUZZLE:"
720 PRINT:PRINT TAB(17);"◪▭▭▭▭◪"
730 FOR I=0 TO 3
740 PRINT TAB(17);"▯■";
750 FOR J=0 TO 3
760 IF DIRG$(I,J)=BLK$ THEN 790
770 PRINT DIRG$(I,J);
780 GOTO 800
790 PRINT "■ ■";
800 NEXT J
810 PRINT "■▯"
820 NEXT I
830 PRINT TAB(17);"◪▭▭▭▭◪"
840 RETURN
900 REM INPUT A MOVE
910 PRINT "■";DIRG$(R,C);"◆■";
920 GET A$:IF A$="" THEN 920
```

```
930 IF A$="■" THEN 1000
940 IF A$="■" THEN 1030
950 IF A$="■■" THEN 1060
960 IF A$="■" THEN 1090
970 IF A$="■" THEN 1120
980 IF A$<>"■" THEN 920
990 S=-1:RETURN
1000 IF R=0 OR (R=ROW+1 AND C=COL) THEN 920
1010 PRINT "■";DIRG$(R,C);"■■■■";
1020 R=R-1:GOTO 910
1030 IF R=3 OR (R=ROW-1 AND C=COL) THEN 920
1040 PRINT "■";DIRG$(R,C);"■■■■";
1050 R=R+1:GOTO 910
1060 IF C=0 OR (C=COL+1 AND R=ROW) THEN 920
1070 PRINT "■";DIRG$(R,C);"■■■■";
1080 C=C-1:GOTO 910
1090 IF C=3 OR (C=COL-1 AND R=ROW) THEN 920
1100 PRINT "■";DIRG$(R,C);"■";
1110 C=C+1:GOTO 910
1120 IF R<>ROW AND C<>COL THEN 920
1130 DR=SGN(R-ROW):DC=SGN(C-COL)
1140 LOC=1242+ROW*40+COL
1150 POKE LOC+54272,14
1160 FOR I=ROW TO R-DR STEP DR
1170 FOR J=COL TO C-DC STEP DC
1180 DIRG$(I,J)=DIRG$(I+DR,J+DC)
1190 POKE LOC,PEEK(LOC+DR*40+DC)
1200 LOC=LOC+DR*40+DC
1210 NEXT J
1220 NEXT I
1230 DIRG$(R,C)=BLK$:PRINT BLK$;
1240 IF DR=0 THEN 1280
1250 IF DR<0 THEN 1270
1260 PRINT "■■■";:GOTO 1300
1270 PRINT "■■■";:GOTO 1300
1280 IF DC<0 THEN 1300
1290 PRINT "■■■";
1300 ROW=R:COL=C:R=R-DR:C=C-DC
```

```
1310 PRINT "■";DIRG$(R,C);"◆■■";
1320 RETURN
1400 REM SUCCESS
1410 PRINT "■■■";TAB(15);"YOU GOT IT!"
1420 POKE 54272,30:POKE 54273,134
1430 POKE 54286,15:POKE 54287,67
1440 POKE 54277,0:POKE 54278,240
1450 POKE 54296,15:POKE 54276,21
1460 FOR I=1 TO 200:NEXT I
1470 POKE 54276,20:POKE 54296,0
1480 RETURN
```

GLOSSARY OF VARIABLES

BLK$ Used to hold the space character.

QUO$ This is used to store the quotation-mark character.

TAKEN An array of 127 elements numbered 0 through 126. Each element represents one of the 127 printable characters. Setting an element to 1 means that the corresponding character has already been entered as part of the goal.

GRID$ A matrix that represents the player's goal pattern. Its rows and columns are numbered 0 through 3.

DIRG$ A matrix that holds the scrambled copy of the puzzle. This is the matrix on which the player operates.

ROW After the subroutine at line 500 is executed this variable indicates the row location of the space in the matrix.

COL This variable indicates the column location of the space after the subroutine at line 500 is executed.

R After line 170 in the program this variable indicates the row location of the cursor.

C After line 170 this variable indicates the column location of the cursor.

S The subroutine at line 900 sets this variable to −1 when the F7 key is pressed.

CNT In the subroutine at line 300 this variable keeps count of how many characters the player has entered so far.

TMES This variable is used in the subroutine at line 500 to determine how many shuffling moves should be made to scramble the puzzle.

LINES OF SPECIAL INTEREST

90–110: The goal matrix is copied into the working matrix before the latter is scrambled.

160–170: The cursor is positioned on some tile adjacent to the space. The subroutine at line 900 (which is called later) performs the function of actually displaying the cursor.

180–190: Moves the computer's PRINT cursor (as opposed to the program's cursor) to the location in the puzzle where the program's cursor is to be displayed.

200–230: The subroutine at line 900 returns only when a function key has been pressed. If it was the F1 key, the puzzle matrix is compared with the goal matrix. If they are not identical, the subroutine is called again.

240: The matrices match so the puzzle was solved.

Subroutine

300–490: Inputs the player's goal while drawing the frame.

320: The graphics characters used in this line are Commodore key-D, shift-R, and Commodore key-F.

330: The graphics character used in this line is shift-H. At the beginning of each row, reversed graphics mode is turned on.

340: A reversed graphics dot (shift-Q) is used as a cursor to indicate where the next character keyed in is displayed.

350: Nonprinting characters with ASCII values below 32, as well as the quotation mark, are ignored.

360: The remaining nonprinting characters are ignored.

370: Since the variable A is used to subscript the array TAKEN, it is adjusted so that its lowest possible value is 0. The second statement in this line is required because the graphics characters accessible through the Commodore key are actually translated into ASCII codes 193–223 rather than 161–191 as they are classified in the *Programmer's Guide*. These statements assure than no elements of array TAKEN are wasted.

380: A character that has already been entered is ignored.

410: The space is displayed in normal graphics.

420: The value of CNT is used to form the appropriate subscripts to matrix GRID$.

440: Handles the situation in which the computer must enter the space as the 16th character. Notice that the space is represented by element 0 of TAKEN.

470: A new row is begun. The graphics character used is shift-G.
480: The frame is finished. The graphics characters used are shift-G, Commodore key-C, shift-E, and Commodore key-V.

Subroutine
500–690: Scrambles the puzzle.

510–530: Set R and C to indicate the location of the space.
540: Between 50 and 100 tiles are moved in a scramble.
570–580: The value of DR specifies the left or right direction in which the computer searches for a tile. It can take on the values −1, 0, or 1. However, these lines randomly select a direction from among only those directions that are actually possible. For example, if the space is in the top row, DR may take on only the values 0 or 1.
600–630: If DR = 0, a nonzero value for DC must be chosen. DC indicates a left or right direction. If DR were nonzero, DC would have the value 0 from line 560. However, in the case where DR is 0, DC must be assigned a value of 1 or −1, again depending on where the space is located.
640: ROW and COL are set to indicate the location of the selected tile.
650: The selected tile is moved into the location formerly occupied by the space.
660: The location formerly occupied by the tile is now filled with a blank.
670: R and C are again set to point to the space. Notice that when this routine ends, ROW and COL also indicate the location of the space.

Subroutine
700–840: Prints the scrambled puzzle at the top of the screen.

Subroutine
900–1320: This lengthy routine moves the cursor, as directed by the player, and performs the operations requested by the function keys.

910: Displays the cursor at the location specified by R and C.
990: S is set to −1 when F7 is pressed.
1000–1110: Move the cursor. Lines 1000, 1030, 1060, and 1090 prevent the cursor from moving onto a space.
1120: To move one or more tiles, the cursor must be in the same row or the same column as the space.

1130–1220: All the tiles from the cursor to the space are moved. In this process PEEKs and POKEs are easier to use than PRINT statements.

1230: The location formerly occupied by the cursor is set to blank. Notice that since POKEs and PEEKs were used in the lines above, the machine's PRINT cursor is still at this location.

1260–1310: Reposition the cursor onto the tile it covered previously. The values of R, C, ROW, and COL are also adjusted.

Subroutine

1400–1480: Displays a message and emits a beep indicating success.

SUGGESTED ENHANCEMENTS

1. In the event that the player is overcome with frustration, the program might be modified in such a way that when the F7 key is pressed the computer actually solves the puzzle step by step. This modification would require considerable analysis of the strategy used.
2. The shuffling subroutine might be improved by enabling it to move more than one tile at a time and ensuring that it does not keep moving the same tiles back and forth. Keep in mind, however, that the shuffling must still be done as a human would physically do it, because merely placing the tiles in random locations may result in an unsolvable puzzle.

CHAPTER 28

FEED HARRY

Our friend Harry returns for a final appearance. This time he is fighting for his life. He must eat those dratted at-signs (@) in order to gain energy. Each time he eats one a crunching sound is emitted. However, as he is moving around the playing field, foraging for food, he loses some energy. On the other hand, when he just stands still he loses twice as much energy. It is for these reasons that he must constantly eat at-signs. Fortunately, the less energy he has, the more at-signs appear on the field. Should the area become too cluttered with them (or should Harry accumulate a large surplus of energy), their production rate is automatically reduced.

Harry is represented by a white box. He begins the game at the center of the playing field. The A and Z keys are used to control his up and down movements, respectively, while the up/down CRSR key moves him to the left and the left/right CRSR key moves him to the right. An orange bar at the top left of the screen indicates by its length how much energy Harry currently has. It shrinks as he loses energy and increases in length by a fixed amount every time Harry gobbles an at-sign. When this line dissolves to nothing, the game is over.

Note that the game starts immediately after the RUN command

is issued, and Harry will lose all of his energy very quickly if he doesn't start eating at-signs right away. Therefore, prepare to start moving Harry as soon as the RUN command is entered.

```
10 REM FEED HARRY
20 DIM SYM(7)
30 FOR I=0 TO 7:READ SYM(I):NEXT I
40 DATA 32,101,116,117,97,246,234,231
50 POKE 53281,13:POKE 54296,15
60 PRINT "■■■  ■"
70 FOR I=1 TO 40:PRINT "⊟";:NEXT I
80 ROW=12:COL=20
90 POKE 1524,160:POKE 55796,1
100 F=0:PTS=16
110 MOVE=0:S=0
120 FOR I=1 TO 5
130 IF INT(RND(0)*(H2/2+F))=0 THEN GOSUB 500
140 GOSUB 600
150 IF A=64 THEN MOVE=MOVE+1
160 NEXT I
170 D=-2:IF MOVE<3 THEN D=-1
180 IF S>0 THEN D=4*S
190 IF PTS+D<0 THEN D=-PTS
200 GOSUB 300
210 IF PTS>0 THEN 110
220 POKE 54296,0:POKE 53281,6
230 POKE 198,0
240 PRINT "■GAME OVER"
250 END
300 REM FOOD MARKER
310 H1=INT(PTS/8)
320 PTS=PTS+D
330 H2=INT(PTS/8):IF H2>40 THEN H2=40
340 L=PTS AND 7
350 IF H1=H2 THEN 440
360 IF H1>H2 THEN 410
370 FOR I=H1+1 TO H2
380 POKE 1023+I,160:POKE 55295+I,8
390 NEXT I
```

```
400 GOTO 440
410 FOR I=H2+1 TO H1+1
420 POKE 1023+I,32
430 NEXT I
440 IF H2=40 THEN 470
450 POKE 1024+H2,SYM(L)
460 POKE 55296+H2,8
470 RETURN
500 REM PLACE FOOD
510 F=F+1
520 SPOT=INT(RND(0)*920)+1104
530 IF PEEK(SPOT)<>32 THEN 510
540 POKE SPOT,0:POKE SPOT+54272,12
550 RETURN
600 REM MOVE HARRY
610 SPOT=1024+ROW*40+COL
620 A=PEEK(197):IF A=64 THEN RETURN
630 IF A=10 THEN 730
640 IF A=12 THEN 710
650 IF A=7 THEN 690
660 IF A<>2 THEN RETURN
670 IF COL=39 THEN RETURN
680 COL=COL+1:GOTO 750
690 IF COL=0 THEN RETURN
700 COL=COL-1:GOTO 750
710 IF ROW=24 THEN RETURN
720 ROW=ROW+1:GOTO 750
730 IF ROW=2 THEN RETURN
740 ROW=ROW-1
750 POKE SPOT,32
760 SPOT=1024+ROW*40+COL
770 IF PEEK(SPOT)=0 THEN GOSUB 800
780 POKE SPOT,160:POKE SPOT+54272,1
790 RETURN
800 REM CRUNCH
810 S=S+1:F=F-1
820 POKE 54273,22:POKE 54280,4
830 POKE 54277,34:POKE 54278,244
840 POKE 54284,34:POKE 54285,242
```

```
850 POKE 54282,8
860 POKE 54276,129:POKE 54283,65
870 FOR J=1 TO 20:NEXT J
880 POKE 54276,128:POKE 54283,64
890 RETURN
```

GLOSSARY OF VARIABLES

SYM This array of eight elements numbered 0 through 7 contains the display codes for graphics characters representing bars of varying thickness. The orange bar at the top of the screen is incremented by eighths of a character space. It is displayed as a series of orange boxes (each representing eight increments) plus one of the characters from array SYM, which extends it to the correct length. SYM(0) is a space representing 0 increments. SYM(1) is a graphics character representing 1 increment, and so forth up to SYM(7), which represents 7 increments.

ROW Indicates Harry's row location on the screen.

COL Indicates Harry's column location on the screen.

F Represents the number of at-signs currently on the screen. (F stands for "food.")

PTS The number of energy units Harry currently possesses.

D Specifies, for use by the subroutine at line 500, how many units of energy Harry is to gain or lose.

MOVE This variable records how many moves Harry *didn't* make.

S Counts how many at-signs Harry eats.

A The result of PEEK(197), as performed in the subroutine in line 600.

H1 In the subroutine at line 300 this variable indicates how many complete orange boxes are contained in the orange bar before the value of PTS is updated.

H2 Same as for H1 except it is after PTS is updated.

L Used to subscript array SYM for choosing the appropriate graphics character to end the orange bar.

LINES OF SPECIAL INTEREST

60: Since the player starts out with 16 units of energy (see line 100), an orange bar consisting of two complete boxes is displayed to represent this.

80–90: Start Harry at the center of the screen.
100: Starts the game with 16 energy units.
110–160: Harry gets 5 chances to move before his energy score is incremented or decremented. The counts of moves not taken and at-signs not eaten are reset to 0 before each set of 5 chances.
130: The decision as to whether a new at-sign should be added is based upon the amount of energy units Harry has (indirectly represented by H2) and the number of at-signs that are already on the playing field.
150: MOVE is incremented if no key is pressed.
170: If Harry stands still for 3 or more of the last 5 chances to move, he loses 2 energy units. Otherwise, he loses only 1.
180: If Harry ate 1 or more at-signs during the past 5 chances to move, no energy units are deducted and, on the contrary, he gains 4 energy units per at-sign eaten.
190: Ensures that PTS is never reduced below 0.

Subroutine
300–470: Increases or decreases the length of the orange bar.

330: In the unlikely event that the orange bar extends the entire row, it is not extended further.
340: L is, in effect, assigned the integer remainder after PTS is divided by 8.
370–390: Increases the number of complete squares in the orange bar.
410–430: Decreases the number of complete squares in the orange bar.
450–460: Place the appropriate graphics character at the end of the orange bar to extend it to the correct length.

Subroutine
500–550: Places an at-sign on the screen.

510: Increments the at-sign count.
520–530: Selects a vacant location in screen memory.

Subroutine
600–790: Moves Harry if this is requested by the player.

770: If Harry moves onto an at-sign, performs the appropriate actions.

Subroutine

800–890: Produces a crunching noise when Harry eats an at-sign. Both the noise generator and the square wave generator are used for maximum effect.

810: Increments the count of eaten at-signs and decrements the count of at-signs on the screen.

SUGGESTED ENHANCEMENTS

1. For those interested in music, a solemn dirge could be sounded when Harry runs out of energy.
2. This game could be made more arcadelike by scoring points each time an at-sign is eaten.
3. Aliens could be introduced to compete with Harry for the at-signs. May the fittest survive!

CHAPTER 29

MORSE CODE

If you are a ham radio operator, you will know that in order to qualify for a license you must be able to receive and send Morse code. For the various levels of license required, there are different levels of proficiency in Morse that are demanded by the federal authorities. With this program you will be able to sharpen your skills in Morse without the aid of a teacher. All you need is your faithful Commodore 64. Morse code is an internationally recognized communications code in which each letter, number, and punctuation mark is given a code represented by a unique series of dots and dashes.

When this program is run, a menu is displayed offering the user two options. The first permits him to enter any message and the computer translates it into Morse code. The second option allows the user to enter a message in Morse code and the computer decodes it. In order to select one of these options, the user must press the number corresponding to the option desired (1 or 2) without pressing the RETURN key.

For option 1 the user enters a message which may consist of letters of the alphabet, spaces, and commas, periods, or question marks. The computer allows only one space between words. Since a coded message takes up much room on the screen, no more than one line may be entered for encoding. Should a mistake be

made when entering a message, the backspace key (the CRSR key) may be used. When the computer displays the encoded form of the message, it places each word on a separate line and inserts blank spaces between the codes for each letter of the message. In addition, the computer also beeps the message in Morse, placing short pauses between letters and longer pauses between words. The player is then asked whether he wishes to run more messages. If so, the menu of options is again displayed.

The second option allows the player to enter a message encoded in Morse. The period is used to represent the dot and the minus sign is used for the dash. When either of these keys is pressed, the computer displays the character on the screen and produces the tone corresponding to either a dot or a dash, depending upon which character was typed. The player must insert one blank space between the codes representing each letter. In addition, each word must be entered on a separate line, meaning that the RETURN key must always be pressed between words. When the last letter of the last word of the message is typed in, the RETURN key must be pressed an extra time. The computer then displays the decoded message. If the user enters an invalid code, the computer displays the message "ERROR AFTER:" and follows this with the message decoded up to the point where the error occurred. At the end of each round the option is given to go around again.

```
10 REM MORSE CODE
20 RTN$=CHR$(13):BLK$=CHR$(32)
30 DIM MO$(28)
40 FOR I=0 TO 28:READ MO$(I):NEXT I
50 DATA ".- ","-... ","-.-. ","-.. ",". "
60 DATA "..-. ","--. ","..... ",".. "
70 DATA ".--- ","-.- ",".-.. ","-- ","-. "
80 DATA "--- ",".--. ","--.- ",".-. "
90 DATA "... ","- ","..- ","...- ",".-- "
100 DATA "-..- ","-.-- ","--.. "
110 DATA "--..-- ",".-.-.- ","..-- "
120 PRINT "▽I CAN:■"
130 PRINT "1) ENCODE YOUR MESSAGE"
```

```
140 PRINT "2) DECODE YOUR MESSAGE"
150 PRINT "■■WHAT SHOULD WE DO (1-2)?";
160 GET A$:IF A$="" THEN 160
170 IF A$<"1" OR A$>"2" THEN 160
180 PRINT A$
190 ON ASC(A$)-48 GOTO 200,300
200 REM ENCODE YOUR MESSAGE
210 PRINT "■PLEASE ENTER MESSAGE:■"
220 M=0:GOSUB 900:GOSUB 500
230 PRINT "■YOUR ENCODED MESSAGE IS:■";RTN$;Y$
240 S$=Y$:GOSUB 1400
250 PRINT "■■■MORE (Y OR N)? ";
260 GET A$:IF A$="" THEN 260
270 IF A$<>"Y" AND A$<>"N" THEN 260
280 PRINT A$:IF A$="Y" THEN 120
290 POKE 54296,0:PRINT "■FINISHED":END
300 REM DECODE YOUR MESSAGE
310 PRINT "■PLEASE ENTER MESSAGE:■"
320 M=1:GOSUB 900:GOSUB 700
330 IF NOT ERR THEN 360
340 PRINT "■ERROR AFTER:■";RTN$;Y$
350 GOTO 250
360 PRINT "■YOUR DECODED MESSAGE IS:■";RTN$;Y$
370 GOTO 250
500 REM ENCODE X$ INTO Y$
510 Y$=""
520 FOR II=1 TO LEN(X$)
530 A=ASC(MID$(X$,II,1))
540 IF A>64 THEN Y$=Y$+MO$(A-65)
550 IF A=32 THEN Y$=Y$+RTN$
560 IF A=44 THEN Y$=Y$+MO$(26)
570 IF A=46 THEN Y$=Y$+MO$(27)
580 IF A=63 THEN Y$=Y$+MO$(28)
590 NEXT II
600 RETURN
700 REM DECODE X$ INTO Y$
710 Y$="":SV$="":ERR=0
720 FOR II=1 TO LEN(X$)
```

```
730 A$=MID$(X$,II,1)
740 IF A$=RTN$ OR A$=BLK$ THEN 770
750 SV$=SV$+A$
760 GOTO 880
770 SV$=SV$+BLK$
780 FOR JJ=0 TO 28
790 IF SV$=MO$(JJ) THEN 820
800 NEXT JJ
810 ERR=-1:GOTO 890
820 IF JJ<26 THEN Y$=Y$+CHR$(JJ+65)
830 IF JJ=26 THEN Y$=Y$+","
840 IF JJ=27 THEN Y$=Y$+"."
850 IF JJ=28 THEN Y$=Y$+"?"
860 SV$=""
870 IF A$=RTN$ THEN Y$=Y$+BLK$
880 NEXT II
890 RETURN
900 REM INPUT X$
910 X$="":P=1
920 PRINT "■ ■■■";
930 GET A$
940 IF A$="" THEN 920
950 PRINT " ■■";
960 IF M=0 THEN 1050
970 IF A$="." OR A$="-" THEN 1140
980 IF A$<>BLK$ OR P=1 THEN 1000
990 IF RIGHT$(X$,1)<>BLK$ THEN 1140
1000 IF A$="■■" THEN 1070
1010 IF A$<>RTN$ THEN 920
1020 IF P=1 THEN 1250
1030 IF RIGHT$(X$,1)=BLK$ THEN
     X$=LEFT$(X$,LEN(X$)-1)
1040 P=0:GOTO 1140
1050 IF A$=RTN$ THEN 1140
1060 IF A$<>"■■" THEN 1100
1070 X$=LEFT$(X$,LEN(X$)-1):P=P-1
1080 IF P=0 THEN P=40
1090 PRINT A$;:GOTO 920
1100 IF A$<>BLK$ OR P=1 THEN 1120
```

```
1110 IF RIGHT$(X$,1)<>BLK$ THEN 1140
1120 IF A$="," OR A$="." OR A$="?" THEN 1140
1130 IF A$<"A" OR A$>"Z" THEN 920
1140 PRINT A$;
1150 IF M=0 THEN 1180
1160 IF A$=BLK$ OR A$=RTN$ THEN 1190
1170 S$=A$:GOSUB 1400
1180 IF A$=RTN$ THEN 1250
1190 X$=X$+A$
1200 P=P+1
1210 IF P<41 THEN 920
1220 IF M=0 THEN 1250
1230 P=1
1240 GOTO 920
1250 RETURN
1300 REM SIGNAL SOUND
1310 POKE 54272,120
1320 POKE 54273,40
1330 POKE 54275,8
1340 POKE 54277,34
1350 POKE 54278,34
1360 POKE 54296,15
1370 POKE 54276,65
1380 RETURN
1400 REM PRODUCE MESSAGE
1410 FOR I=1 TO LEN(S$)
1420 W$=MID$(S$,I,1)
1430 IF W$=BLK$ THEN 1510
1440 IF W$=RTN$ THEN 1520
1450 DELAY=40:IF W$="-" THEN DELAY=100
1460 GOSUB 1300
1470 FOR J=1 TO DELAY:NEXT J
1480 POKE 54276,64
1490 FOR J=1 TO 120-DELAY:NEXT J
1500 GOTO 1530
1510 FOR J=1 TO 120:NEXT J:GOTO 1530
1520 FOR J=1 TO 360:NEXT J
1530 NEXT I
1540 RETURN
```

GLOSSARY OF VARIABLES

RTN$ Contains the RETURN character.

BLK$ Contains the blank character.

MO$ An array of twenty-nine strings numbered 0 through 28. It holds the Morse for all the letters and punctuation marks.

M Indicates what sort of message the subroutine at line 900 is inputting. A value of 0 indicates an ordinary message while a value of 1 specifies a coded message.

X$ Contains the message inputted by the user. It is returned by the subroutine at line 900 and is used by the subroutines at lines 500 and 700.

**Y$** Contains either the encoded version of X$ as returned by the subroutine at line 500 or the decoded version of X$ as returned by the subroutine at line 700.

S$ Used to pass a Morse message or just a dot or a dash to the subroutine at line 1400.

ERR The value of this variable is set in the subroutine at line 700. If it has a value "true" (or −1), it indicates that there was an error in the Morse message entered by the user.

SV$ Used by the subroutine at line 700 to hold the code for a single character.

P This variable, used by the subroutine at line 900, keeps track of which column on the screen (numbered 1–40, for a change) the cursor is located at. P is set to 0 right after the RETURN key is pressed.

LINES OF SPECIAL INTEREST

50–110: Notice that each code is followed by a blank space. This is done to simplify the comparisons that are made later in the program.

Subroutine

500–600: Converts a normal message into Morse. No error checking is necessary in this routine because the subroutine at line 900 made sure that the content of X$ is valid.

540–580: Concatenate the appropriate character code to Y$.

550: This places each word on a separate line. It is not necessary

to insert blanks between the codes for each letter, because, as you will recall, the strings in MO$ already contain these blanks.

Subroutine
700–890: Converts the coded message into standard form.

750: Adds the current dot or dash to SV$.
770: Adds the blank indicating the end of a character.
780–810: Search MO$ for the character code. If it is not there, ERR is set to −1.
820–850: Concatenate the appropriate character to Y$.
870: A RETURN indicates the end of a word, and this is indicated in Y$ by a blank.

Subroutine
900–1250: Inputs the user's message into X$.

950: Makes the cursor disappear.
970–1040: Input a Morse message.
980: A blank may not be entered as the first character on a line.
990: A blank may not be entered right after another blank. Notice that if the IF test fails, line 1010 ensures that the blank is ignored.
1020: A RETURN at the beginning of a line indicates the end of a message.
1030: Removes a blank that was entered at the end of a line.
1050–1130: Input a normal message.
1070–1090: Handles backspacing for both normal and Morse messages. Notice that line 1080 tests for wraparound.
1100: A blank may not be the first character on a line.
1110: A blank may not be entered right after another blank. Line 1130 ensures that the second blank is ignored.
1140: Prints the entered character for both a normal and a Morse message.
1160–1170: If it is a Morse message and the current character is a dot or a dash, produce the appropriate tone.
1180: For a normal message, if the current character is a RETURN, then the message is over.
1210–1220: When a normal message reaches forty characters in length (P equals 41), the program accepts no more input and immediately begins processing the message.
1230: For a Morse message the variable P is set to recognize the wraparound.

Subroutine
1300–1380: Initiates a tone for use by the subroutine at line 1400.

Subroutine
1400–1540: Sounds out the Morse message contained in S$. Recall from line 1170 that S$ may also contain a single dot or dash.

1450: Assigns a value to the variable DELAY according to whether the current character is a dot or a dash.
1470: Specifies the duration of the tone.
1490: Specifies the duration of the silence between tones.
1510: A short silence between letters.
1520: A longer silence between words.

SUGGESTED ENHANCEMENTS

1. You may wish to add more options to the options menu. For example, the program could be modified to either display a Morse message and ask you to decode it or display a normal message that you must encode.
2. We did not allow for the full Morse code, just a reasonable part of it, in order to keep the program within manageable limits. You may feel that you want the whole range to be included. This wouldn't warrant a major modification.
3. When the computer encodes your message under option 2, it might be desirable to have the program repeat the sounding out of the message if you press one of the function keys.

CHAPTER 30

PI IN THE SKY

So finally we come to the end of this book. We have decided to conclude it on a frivolous note. You are no doubt aware that the first TV video games were simple tennis or Ping Pong–style games. This last program of ours is also such a game; with it, we have succeeded in turning the clock back. However, instead of knocking around a simple ball, we have chosen to use the Commodore 64's pi symbol. Since in our version of the game a series of pi's falls down from the sky, we have named the game "Pi in the Sky."

The player controls a white paddle that is located at the bottom of the screen. Pressing the up/down CRSR key moves the paddle left, while the left/right CRSR key moves it to the right. The pi is dropped from a random column at the top of the screen and bounces off the top and sides of the screen when it hits them. The paddle must be positioned so that it keeps the pi from reaching the bottom of the screen, which it otherwise falls through. If the pi is successfully intercepted by the paddle, the paddle turns red momentarily, a beep is emitted, the player is awarded a point, and the pi bounces off the paddle. On the other hand, if the player misses, the pi vanishes through the bottom of the screen and a raspberry is sounded. Another pi is then dropped from above. The player has a supply of only three pi's from the sky, and once

the third is lost, the game ends. The number of points scored by the player, as well as the number of pi's he has left (including the current one), are constantly displayed at the top of the screen.

To make the game slightly more difficult, the pi sometimes stops momentarily (in an attempt to fool you) and then continues along its merry path.

```
10 REM PI IN THE SKY
20 NUM=3:HITS=0:POKE 54296,15
30 PRINT "■":POKE 53281,15
40 FOR I=2002 TO 2005
50 POKE I,239:POKE I+54272,1
60 NEXT I
70 PADD=2002
80 COL=INT(RND(0)*40)
90 DR=1:DC=INT(RND(0)*2)*2-1
100 IF COL=0 THEN DC=1
110 IF COL=39 THEN DC=-1
120 PI=1024+COL
130 POKE PI,94:POKE PI+54272,0
140 S=0
150 PRINT "■■ POINTS:";HITS;
160 PRINT TAB(18);"PIECES OF PI LEFT:";NUM
170 IF INT(RND(0)*4) THEN GOSUB 300
180 IF S<0 THEN 230
190 PRINT "■■ POINTS:";HITS;
200 PRINT TAB(18);"PIECES OF PI LEFT:";NUM
210 GOSUB 500
220 GOTO 150
230 GOSUB 900
240 NUM=NUM-1:IF NUM>0 THEN 80
250 PRINT "■■ POINTS:";HITS;
260 PRINT TAB(18);"PIECES OF PI LEFT: 0"
270 PRINT:PRINT TAB(15);"GAME OVER"
280 POKE 53281,6:POKE 198,0
290 END
300 REM MOVE PI
310 IF DR=1 AND PI>=PADD-40 AND PI<=PADD-37
    THEN 390
```

```
320 NXT=PI+DR*40+DC
330 IF NXT>=1024 THEN 350
340 DR=1:GOTO 400
350 IF NXT=PADD OR NXT=PADD+3 THEN 380
360 IF NXT<2024 THEN 400
370 S=-1:POKE PI,32:RETURN
380 DC=-DC
390 DR=-1:GOSUB 700
400 IF COL+DC>=0 AND COL+DC<=39 THEN 420
410 DC=-DC
420 POKE PI,32
430 COL=COL+DC:PI=PI+DR*40+DC
440 POKE PI+54272,0:POKE PI,94
450 RETURN
500 REM MOVE PADDLE
510 FOR I=1 TO 4
520 A=PEEK(197):IF A=2 OR A=7 THEN 550
530 NEXT I
540 RETURN
550 IF A=2 THEN 630
560 IF PADD=1984 THEN RETURN
570 IF PI=PADD-1 THEN 620
580 PADD=PADD-1
590 POKE PADD+54272,1:POKE PADD,239
600 POKE PADD+4,32
610 RETURN
620 DR=-1:DC=-1:GOSUB 700:RETURN
630 IF PADD=2020 THEN RETURN
640 IF PI=PADD+4 THEN 690
650 PADD=PADD+1
660 POKE PADD+54275,1:POKE PADD+3,239
670 POKE PADD-1,32
680 RETURN
690 DR=-1:DC=1:GOSUB 700:RETURN
700 REM HIT
710 HITS=HITS+1
720 FOR I=PADD+54272 TO PADD+54275
730 POKE I,2
740 NEXT I
```

```
750 POKE 54272,195:POKE 54273,16
760 POKE 54275,4
770 POKE 54277,0:POKE 54278,240
780 POKE 54276,65
790 FOR I=1 TO 50:NEXT I
800 FOR I=PADD+54272 TO PADD+54275
810 POKE I,1
820 NEXT I
830 POKE 54276,64
840 RETURN
900 REM MISS
910 POKE 54272,48:POKE 54273,4
920 POKE 54275,1
930 POKE 54277,68:POKE 54278,244
940 POKE 54276,65
950 FOR I=1 TO 500:NEXT I
960 POKE 54276,64
970 RETURN
```

GLOSSARY OF VARIABLES

NUM	Specifies the number of pi's the player has left.
HITS	The number of points or hits the player has recorded.
PADD	Specifies the location in screen memory at which the left side of the paddle is located.
COL	This variable indicates the column from which the pi is dropped.
DR	The up or down direction in which the pi is currently traveling. It always has the value 1 or −1.
DC	Specifies the left or right direction in which the pi is traveling. This variable, like DR, always has the value 1 or −1; therefore, the pi is always traveling in a diagonal direction.
PI	Indicates the screen memory location at which the pi is currently situated.
S	Set to −1 when a pi is lost.
NXT	This variable is used in the subroutine at line 300 to specify the next location in screen memory that the pi enters if it continues in its current direction.

LINES OF SPECIAL INTEREST

40–70: Draw the paddle at the center of the bottom line of the screen.
100–110: Adjust the pi's direction if it is starting from a corner.
130: Places the pi on the screen.
170: This line ensures that the pi pauses about one out of four times.
190–200: Notice that points can be scored both in the subroutine in line 300 and in the subroutine at line 500.
250–260: Display the final score.

Subroutine
300–450: Moves the pi one step.

310: Detects a collision of the pi with the paddle.
330–340: If the pi hits the top of the screen, its vertical direction is changed from upward to downward.
350: This line detects a case not caught by line 310: the pi hits a corner of the paddle.
360–370: If the pi hits the bottom of the screen, the player loses it.
380: The pi has hit the corner of the paddle, so its horizontal direction is switched (as well as its vertical direction in the following line).
390: The pi has hit the paddle, so its vertical direction is changed from downward to upward, and the subroutine at line 700 is called to perform all the necessary ceremony connected with the event.
400–410: If the pi hits the left or right side of the screen, its horizontal direction is switched.

Subroutine
500–690: Moves the paddle as directed by the player.

510–540: Location 197 is checked 4 times to detect the pressing of a key before the subroutine returns. This is done as a sort of delay because without it the pi would be observed to move faster when the paddle is not moving than when it is. Once the pressing of a key is detected the subroutine moves the paddle and does not check location 197 again.
560–620: Move the paddle left.
560: The player is trying to move the paddle past the left side of the screen.
570: Detects the case when the side of the paddle hits the pi.

Normally, if the pi passes the side of the paddle in the normal course of its movements, no collision is detected. However, if the paddle makes a thrusting movement toward the pi, a collision is detected.
590–600: The movement is accomplished by adding a box to the left side of the paddle and removing a box from the right side.
620: The left side of the paddle has hit the pi. The direction of the pi's movement is changed and the hit is recorded. (Notice that in this situation the paddle is not moved on the screen.)
630–690: The paddle is moved to the right. These lines correspond on a one-to-one basis with lines 560–620.

Subroutine
700–840: A hit is recorded.

710: The player's points are incremented.
720–740: The paddle is colored red.
750–790: A beep is sounded.
800–820: The paddle's color is restored to white.
830: The beep is terminated.

Subroutine
900–970: A raspberry is sounded when a pi is lost.

SUGGESTED ENHANCEMENTS

1. The program may be amended so that when the pi hits the top or the sides of the screen, a beep is sounded.
2. The number of pi's that the player starts out with can be increased, or an extra pi can be awarded every time a certain number of points is scored.
3. Some people might feel that the action in this game is too slow. To speed it up, advantage can be taken of certain BASIC techniques which include the following: Change all integer numbers to real numbers by adding a decimal point; remove all the blanks from all the statements (except literals), regardless of the fact that this renders the program difficult to read; place frequently used numbers such as 54272 into variables and refer to the variables rather than listing the number each time.

ABOUT THE AUTHORS

HENRY MULLISH is Senior Research Scientist and Lecturer in Computer Science at the Courant Institute of Mathematical Sciences of New York University. He is the author of over a dozen books on computer programming.

HERBERT COOPER is a Ph.D. student and graduate research assistant in Computer Science at Columbia University. He has been programming computers since he was twelve.

$9.95

How would you like to pit yourself against 30 great new games and mind-bending puzzles *and* develop your BASIC programming skills on your Commodore 64—all at the same time? Well, here's your chance. Best-selling computer authors Henry Mullish and Herbert Cooper have focused their expertise on making the most of the Commodore's great game-playing, color-graphics and sound-generating capabilities. Each chapter of *Zappers for the Commodore 64* consists not only of a brief description of an exciting game and its program listing, but also a line-by-line description of the program, a detailed discussion of how the program is structured and actually works, and a helpful glossary of the variables the program uses. There's even a section of possible modifications and enhancements for you to make, so you can customize each game as you want it. In this way you'll learn about all the important elements of good programming, and you'll have a great time doing it, because all of the programs look great, sound great and are challenging to play. And unlike a lot of other books, the programs here are quick and easy to type in. So whether you're trying to solve the secret of the Magic Squares, navigate a tricky Lunar Landing, or outstump the computer in a Math Whiz contest, you'll be learning the secrets of your Commodore 64 — and having fun as well.

Computer Book Division
Simon & Schuster, Inc.
New York

Cover design by Art Bonanno
Cover illustration by
Art Bonanno/Omar Valdes

1084-995

0-671-50714-1